Programsko inženjerstvo

Ak. god. 2023./2024.

Prijava oštećenja

Dokumentacija, Rev. *2*

Grupa: *SourceresOfTheNorth*

Voditelj: *Petar Belošević*

Datum predaje: *19.1.2024.*

Nastavnik: *Vlado Sruk*

Sadržaj

1 Dnevnik promjena dokumentacije 3

2 Opis projektnog zadatka 5
2.1 Postojeća rješenja za slični problem 9
2.2 Mogućnosti nadogradnje i skaliranja rješenja 10

3 Specifikacija programske potpore 11
3.1 Funkcionalni zahtjevi 11
3.1.1 Obrasci uporabe 14
3.1.2 Sekvencijski dijagrami 24
3.2 Ostali zahtjevi 30

4 Arhitektura i dizajn sustava 31
4.1 Baza podataka 33
4.1.1 Opis tablica 33
4.1.2 Dijagram baze podataka 37
4.2 Dijagram razreda 38
4.3 Dijagram stanja 42
4.4 Dijagram aktivnosti 44
4.5 Dijagram komponenti 46

5 Implementacija i korisničko sučelje 48
5.1 Korištene tehnologije i alati 48
5.2 Ispitivanje programskog rješenja 53
5.2.1 Ispitivanje komponenti 53
5.2.2 Ispitivanje sustava 57
5.3 Dijagram razmještaja 64
5.4 Upute za puštanje u pogon 66
5.4.1 Konfiguracija poslužitelja baze podataka 66
5.4.2 Konfiguracija poslužitelja web aplikacije 68

6 Zaključak i budući rad **73**

Popis literature **75**

Indeks slika i dijagrama **77**

Dodatak: Prikaz aktivnosti grupe **78**

1. Dnevnik promjena dokumentacije

Rev.	Opis promjene/dodatka	Autori	Datum
0.1	Napravljen predložak. Započet opis zadatka.	Petar Belošević	23.10.2023.
0.2	Dovršen opis projektnog zadatka.	Petar Belošević	28.10.2023.
0.3	Dodani dionici i aktori.	Petar Belošević	29.10.2023.
0.4	Napravljeni obrasci uporabe i dijagrami obrazaca uporabe.	Petar Belošević	4.11.2023.
0.5	Dodani sekvencijski dijagrami s opisima, napisani ostali zahtjevi.	Petar Belošević	7.11.2023.
0.6	Dodan opis arhitekture sustava.	Petar Belošević	8.11.2023.
0.7	Opis baze podataka	Petar Belošević	9.11.2023.
0.8	Dodan dijagram razreda s opisom, izmjenjen opis arhitekture sustava.	Petar Belošević	16.11.2023.
1.0	Završna verzija za prvu kontrolnu točku.	Petar Belošević	17.11.2023.
1.1	Popravci grešaka iz prve kontrolne točke.	Petar Belošević	10.12.2023.
1.2	Dodan dijagram stanja s pripadnim opisom.	Petar Belošević	16.12.2023.

Nastavljeno na idućoj stranici

Nastavljeno od prethodne stranice

Rev.	Opis promjene/dodatka	Autori	Datum
1.3	Dodan dijagram aktivnosti s pripadnim opisom.	Petar Belošević	18.12.2023.
1.4	Dodan dijagram komponenti s pripadnim opisom.	Petar Belošević	27.12.2023.
1.5	Napisano poglavlje o korištenim tehnologijama i alatima.	Petar Belošević	28.12.2023.
1.6	Dodan dijagram razmještaja s pripadnim opisom.	Petar Belošević	29.12.2023.
1.7	Napravljena prva verzija zaključka.	Petar Belošević	29.12.2023.
1.8	Napisane upute za puštanje u pogon.	Petar Belošević	30.12.2023.
1.9	Dokumentirani testovi komponenti sustava.	Petar Belošević	13.1.2024.
1.10	Dokumentirano ispitivanje sustava.	Petar Belošević	14.1.2024.
1.11	Popravljen opis baze podataka i dijagram aktivnosti, mali popravci kroz cijelu dokumentaciju.	Petar Belošević	16.1.2024.

2. Opis projektnog zadatka

Cilj ovog projekta je razviti programsku podršku za stvaranje web aplikacije *"StreetPatrol"*. Ideja aplikacije je ponuditi običnim građanima jednostavan i efikasan način prijave oštećenja javnih površina i cesta u gradu te dobivanje povratnih informacija o konkretnom oštećenju i njihovoj prijavi.

Građani će svojim djelovanjem putem aplikacije unaprjeđivati kvalitetu života u svojoj zajednici. Svojim prijavama, koje će biti javno dostupne, građani će poticati gradske urede da saniraju prijavljena oštećenja. Aplikacija će također pomoći gradskim uredima u obavljanju svojih zadaća. Naime, preko aplikacije uredi će imati puno bolji uvid u probleme koji postoje u zajednici. Zahvaljujući aplikaciji uredi će moći brže i kvalitetnije reagirati na probleme u zajednici.

Građani aplikaciju mogu koristiti kao registrirani i kao neregistrirani korisnici. Građani imaju mogućnost podnošenja nove prijave, pregleda postojećih prijava i pregled statistike prijava. Registrirani građani također imaju opciju pregleda svojeg profila dok neregistrirani građani imaju mogućnost registracije, tj. izrade profila odnosno prijave, ako već imaju izrađeni profil.

Neregistrirani korisnik se može registrirati te prilikom registracije mora navesti sljedeće informacije:

- ime
- prezime
- email adresa
- lozinka

Prijavljeni korisnici prilikom pregleda svojeg profila mogu vidjeti prije navedene podatke o sebi koje mogu i izmijeniti. Također imaju pregled nad svim prijavama koje su podnesli kroz aplikaciju. Prikazana im je i kratka statistika prijava (broj riješenih prijava, broj prijava na čekanju i u procesu rješavanja).

Građanima se prilikom podnošenja nove prijave otvara obrazac koji je potrebno ispuniti. Prijava sadrži naslov, kratak opis, lokaciju i kategoriju oštećenja. Opcionalno se u prijavi može priložiti fotografija oštećenja. Lokacija oštećenja se prilaže odabirom položaja na karti ili upisivanjem adrese na kojoj se nalazi oštećenje. Ako korisnik priloži fotografiju, aplikacija će pokušati iz nje izvući podatke o lokaciji

te ih ponuditi korisniku kao lokaciju oštećenja. Odabirom kategorije oštećenja korisnik indirektno odabire gradski ured nadležan za konkretan problem. Prilikom opisivanja oštećenja aplikacija će pokušati sama zaključiti kojoj kategoriji oštećenje pripada te će istu predložiti korisniku. Također, ako u sustavu već postoji slična nedavna prijava na istoj lokaciji aplikacija će predložiti korisniku postojeću prijavu i ponuditi mu da svoju prijavu nadoveže na postojeću. Nakon podnošenja prijave aplikacija korisniku daje jedinstveni kod podnesene prijave koja služi za praćenje te prijave preko same aplikacije. Prijavljeni korisnici će također primati obavijesti o svojoj prijavi preko emaila.

Za pregled postojećih prijava korisnik koristi interaktivnu kartu s označenim prijavama na njoj. Prijave se mogu dodatno filtrirati po vremenu podnošenja prijave i kategoriji oštećenja. Korisnik također može pretraživati prijave po jedinstvenom kodu prijave. Na taj način svaki korisnik može pregledavati svoje prijave na aplikaciji. Odabirom neke prijave prikazuju se podaci iz podnesenog obrasca o odabranoj prijavi. Također se prikazuje status prijave (na čekanju, u procesu rješavanja, riješena). Prijavljeni korisnici također vide svoje prijave na pregledu svojeg profila. Prijave koje su riješene mogu se pregledati jedino preko njihovog koda ili preko pregleda korisničkog profila kod registriranih korisnika. U drugim slučajevima takve prijave nisu više dostupne za pregled.

Prikaz statistike prijava nudi filtriranje podataka po vremenu podnošenja prijave, prostoru i kategoriji oštećenja. Za odabrane parametre aplikacija prikazuje:

- ukupan broj podnesenih prijava
- broj prijava sa statusom *na čekanju* i njihov udio
- broj prijava sa statusom *u procesu rješavanja* i njihov udio
- broj prijava sa statusom *riješena* i njihov udio
- prosječan broj podnesenih prijava u danu
- prosječan broj dana koji prijava provede na čekanju
- prosječan broj dana za vrijeme kojih je prijava u procesu rješavanja

Djelatnici gradskog ureda se obavezno u sustav prijavljuju preko profila gradskog ureda. Djelatnici pregledavaju prijave kroz 3 liste, jedna za svaku vrstu statusa: novopristigle, one u procesu rješavanja i riješene prijave. Djelatnici ureda imaju isključivo pristup prijavama za oštećenja za koja je nadležan njihov ured. Ostale prijave djelatnicima nisu vidljive.

Sustav nudi mogućnost registracije novih gradskih ureda. U tom slučaju je na glavnoj stranici potrebno odabrati opciju registracije gradskog ureda. Aplikacija

tada prikazuje obrazac za koji je potrebno navesti:

- ime gradskog ureda
- email adresa gradskog ureda
- lozinka za račun gradskog ureda

Djelatnici pregledavaju pristigle prijave i stavljaju ih u proces rješavanja. Time prijave prelaze u listu prijava u procesu rješavanja i mijenjaju svoj status. Aplikacija u tom slučaju automatski šalje obavijest prijavitelju ako se radi o registriranom korisniku.

Kada neka prijava bude riješena, djelatnik prijavi mijenja status u *riješena*. Prijava prelazi u za to predviđenu listu te aplikacija šalje obavijest prijavitelju ako se radi o registriranom korisniku.

Prethodne dvije radnje bi trebale sadržavati komunikaciju s nadležnim gradskim službama. Dakle, po primitku prijave djelatnik promjenom statusa prijave delegira prijavu nadležnoj gradskoj službi. Kada nadležna služba riješi problem šalje informaciju uredu koji onda može prijavu označiti kao riješenu. Ova funkcionalnost nije implementirana jer zahtjeva komunikaciju s vanjskim sustavima što nije u opsegu ovog projekta.

Djelatnik ima mogućnost objedinjavanja prijava. Naime, ako se pristigla prijava referira na problem za koji već postoji prijava, djelatnik može grupirati te prijave. Aplikacija u tom slučaju šalje prikladnu obavijest registriranim korisnicima čije su prijave zahvaćene ovom radnjom.

Ako procijeni da je pristigla prijava poslana na krivi ured, djelatnik ima mogućnost proslijediti prijavu uredu koji je zapravo nadležan za pristiglu prijavu. Djelatnik to čini tako što mijenja kategoriju prijave čime će prijava automatski biti dodijeljena pripadajućem uredu. Korisniku koji je podnio prijavu se u tom slučaju šalje prikladna obavijest putem maila ako je taj korisnik registriran.

Slično, ako procijeni da pristigla prijava uopće nije valjana, djelatnik ima mogućnost odbaciti prijavu. U tom slučaju se prijava briše iz baze podataka. Korisnik koji je podnio prijavu dobiva prikladnu obavijest putem maila ako je registriran.

Djelatnici gradskih ureda mogu pregledavati statistiku prijava kao i obični korisnici.

Aplikacija je napravljena na primjeru Grada Zagreba te će karte u aplikaciji prikazivati područje Grada Zagreba. Aplikacija nema fiksirani broj gradskih ureda, već je omogućeno dodavanje novih ureda, Kao što je već i navedeno. Time je aplikaciju lakše prilagoditi promjenama u organizaciji gradskih ureda. Također, na

ovaj način je lakše konfigurirati aplikaciju za implementaciju u nekom drugom području osim Grada Zagreba.

2.1 Postojeća rješenja za slični problem

Slične aplikacije već postoje u Hrvatskoj. Tako primjerice Grad Rijeka na svojim stranicama nudi mogućnost prijave raznih prekršaja i oštećenja (`https://gov.rijeka.hr/zahtjevi-i-obrasci/komunalna-djelatnost-i-prijava-komunalnog-nereda/prijave-ostecenja-nereda-ili-nepropisnog-parkiranja/383`). Njihova stranica nudi mogućnost prijave u 4 kategorije. Neke prijave se podnose putem obrazaca, neke samo telefonskim putem, a za neke su ponuđene i specijalne stranice. Tako primjerice za prijavu većine problema, ali i za podnošenje pohvala, Grad Rijeka nudi web aplikaciju *Gradsko oko* (`http://rijeka.oko.hr/`) dostupnu i u mobilnoj verziji. Aplikacija nudi mogućnost podnošenja i pregleda prijava putem interaktivne karte. Prijave su kategorizirane po grupama i klasifikacijama. Prilikom pregleda se mogu filtrirati po vremenu i kategorijama. Prijave i pregled prijava su dostupne samo prijavljenim korisnicima te svaka prijava mora biti odobrena od strane administratora.

Slika 2.1: Aplikacija *Gradsko oko* Grada Rijeke

2.2 Mogućnosti nadogradnje i skaliranja rješenja

Jedna od mogućih nadogradnji sustava bila bi dodavanje izbornika jezika na kojem se prikazuje sučelje. Time bi se olakšalo korištenje aplikacije korisnicima koji govore drugim jezicima. Također bi se time olakšao prijenos i implementacija sustava van Hrvatske.

Uz male preinake moguće je prilagoditi aplikaciju da pruža istu potporu građanima nekog drugog grada ili općine, bilo u Hrvatskoj ili nekoj drugoj zemlji. Aplikacija bi se mogla skalirati i na veće jedinice lokalne samouprave kao što su županije, s obzirom na to da obično imaju sličan ustroj ureda i slične nadležnosti kao i gradovi. Ako se ideja skalira još više, aplikacija bi mogla pružati uslugu korisnicima cijele regije (npr. Slavonija, Dalmacija, Središnja Hrvatska) ili čak čitave države. U tom slučaju u aplikaciji bi bilo potrebno dodati mogućnost prepoznavanja nadležne jedinice lokalne samouprave prije predlaganja nadležnog ureda za konkretan problem. Takvo ponašanje bi se moglo implementirati uz malo prerađivanja i korištenje postojećih funkcionalnosti aplikacije. Generalno gledano, aplikacija bi se mogla skalirati tako da pruža uslugu korisnicima nadnacionalnoj razini, primjerice na području cijele Europske Unije. Ponašanje aplikacije bi zapravo bilo isto uz potrebe skaliranja funkcionalnosti prepoznavanja nadležnih administrativnih jedinica na nadnacionalnoj razini. U tom slučaju aplikacija svakako mora pružati mogućnost prikaza na jeziku svake države u kojoj pruža uslugu. Glavna prepreka u ovakvoj implementaciji bi bila potreba za značajnijom infrastrukturnom podrškom, prvenstveno u obliku velikih poslužitelja koji mogu obrađivati puno zahtjeva i pohranjivati velike količine podataka. Također, aplikacija bi na toj razini zahtijevala dobru suradnju i koordinaciju velikog broja institucija u više različitih država što može biti zahtjevno.

3. Specifikacija programske potpore

3.1 Funkcionalni zahtjevi

Nabrojeni funkcionalni zahtjevi:

- F01: registracija novog korisnika
- F02: prijava korisnika
- F03: podnošenje prijave oštećenja (naslov, opis, lokacija, slika - opcionalno, kategorija)
- F04: automatsko prepoznavanje kategorije prijave
- F05: mogućnost automatskog predlaganja nadovezivanja prijave na postojeću
- F06: pregledavanje podataka o korisničkom računu prijavljenog korisnika
- F07: uređivanje podataka o korisničkom računu prijavljenog korisnika
- F08: pregledavanje prijava prijavljenog korisnika
- F09: pregledavanje svih aktivnih prijava u sustavu preko interaktivne karte
- F10: mogućnost filtriranja prijava u sustavu prilikom pregledavanja
- F11: pregled statistike prijava
- F12: filtriranje statistike prijava
- F13: djelatnik gradskog ureda može odbaciti nevažeću prijavu
- F14: djelatnik gradskog ureda može proslijediti prijavu drugom uredu
- F15: djelatnik gradskog ureda može mijenjati status prijave
- F16: djelatnik gradskog ureda može objediniti prijave istog oštećenja
- F17: povratna informacija korisniku (ako je registriran) za svaku promjenu nad prijavom
- F18: pregled prijava ureda za djelatnike gradskog ureda
- F19: dodatno filtriranje prijava za djelatnike gradskog ureda
- F20: odjava iz sustava
- F21: brisanje korisničkog računa

Dionici:

1. Gradska uprava
2. Djelatnici gradskih ureda
3. Gradske službe
4. Građani
   - Registrirani korisnici
   - Neregistrirani korisnici
5. Razvojni tim

Aktori i njihovi funkcionalni zahtjevi:

1. Neregistrirani korisnik (inicijator) može:
   (a) napraviti svoj profil za koji mu je potrebno ime, prezime, email adresa i lozinka
   (b) podnositi anonimnu prijavu koja sadrži naslov, opis, lokaciju i kategoriju oštećenja, a opcionalno i fotografiju i prijavu na koju se nadovezuje
   (c) pregledavati postojeće prijave preko karte te ih filtrirati po vremenu i kategoriji oštećenja
   (d) pregledavati statistiku postojećih prijava za odabran period, kategoriju oštećenja i prostor
2. Registrirani korisnik (inicijator) može:
   (a) pregledavati i mijenjati osobne podatke
   (b) obrisati svoj profil
   (c) pregledavati svoje prijave
   (d) podnositi prijavu koja sadrži naslov, opis, lokaciju i kategoriju oštećenja, a opcionalno i fotografiju i prijavu na koju se nadovezuje
   (e) pregledavati postojeće prijave preko karte te ih filtrirati po vremenu i kategoriji oštećenja
   (f) pregledavati statistiku postojećih prijava za odabran period, kategoriju oštećenja i prostor
3. Djelatnik gradskog ureda (inicijator) može:
   (a) pregledavati prijave pristigle u njihov ured
   (b) dodatno filtrirati prijave prilikom pregleda prema njihovom statusu
   (c) prijavama sa statusom na čekanju mijenjati status u u procesu rješavanja
   (d) prijavama sa statusom u procesu rješavanja mijenjati status u riješena

(e) objediniti prijave ako se referiraju na isti problem
(f) proslijediti prijave drugom uredu promjenom kategorije prijave
(g) označiti prijavu kao nevažeću
(h) pregledavati statistiku prijava
(i) dodatno filtrirati za koje prijave želi pregledati statistiku

4. Baza podataka (sudionik) može:
   (a) spremati sve podatke o korisnicima
   (b) spremati sve podatke o gradskim uredima
   (c) spremati sve podatke o prijavama

3.1.1 Obrasci uporabe

UC1 - Prijava oštećenja

- **Glavni sudionik:** Korisnik
- **Cilj:** Prijaviti oštećenje nadležnom gradskom uredu
- **Sudionici:** Gradski ured, Baza podataka
- **Preduvjet:** Korisnik je pronašao oštećenje koje želi prijaviti
- **Opis osnovnog tijeka:**
  1. Korisnik odabire opciju za prijavljivanje oštećenja
  2. Korisnik popunjava podatke za prijavu oštećenja i podnosi prijavu
  3. Prijava se sprema u bazu podataka
  4. Korisnik dobiva jedinstveni kod svoje prijave.
- **Opis mogućih odstupanja:**

  2.a Korisnik nije unio sve obavezne podatke u prijavu.
    1. Korisnik dobiva obavijest.
    2. Korisnik popunjava preostala obavezna polja ili odustaje od podnošenja prijave

UC2 - Pregled svih prijava

- **Glavni sudionik:** Korisnik
- **Cilj:** Pregled aktivnih prijava u sustavu
- **Sudionici:** Baza podataka
- **Preduvjet:** -
- **Opis osnovnog tijeka:**
  1. Korisnik odabire glavnu stranicu
  2. Aplikacija prikazuje prijave na karti
  3. Korisnik odabire prijavu
  4. Prikazuju se podaci o prijavi

UC3 - Filtriranje prijava

- **Glavni sudionik:** Korisnik
- **Cilj:** Filtriranje prijava prilikom pregleda
- **Sudionici:** Baza podataka
- **Preduvjet:** Korisnik pregledava prijave na glavnoj stranici
- **Opis osnovnog tijeka:**
  1. Korisnik odabire opciju filtriranja

2. Korisnik odabire opcije kod filtriranja
3. Aplikacija prikazuje filtrirane prijave

UC4 - Pregled statistike prijava

- **Glavni sudionik:** Korisnik ili Djelatnik gradskog ureda
- **Cilj:** Pregled statistike prijava u sustavu
- **Sudionici:** Baza podataka
- **Preduvjet:** -
- **Opis osnovnog tijeka:**
  1. Korisnik odabire opciju za pregledavanje statistike prijava
  2. Aplikacija prikazuje razne podatke za prijave u sustavu

UC5 - Filtriranje statistike prijava

- **Glavni sudionik:** Korisnik ili Djelatnik Gradskog ureda
- **Cilj:** Filtriranje prijava prilikom pregleda statistike
- **Sudionici:** Baza podataka
- **Preduvjet:** Korisnik je odabrao opciju pregleda statistike prijava
- **Opis osnovnog tijeka:**
  1. Korisnik odabire opcije za filtriranje
  2. Aplikacija prikazuje statistiku za filtrirane prijave

UC6 - Registracija

- **Glavni sudionik:** Neregistrirani korisnik ili Djelatnik gradskog ureda
- **Cilj:** Izrada korisničkog računa za korisnika ili novi gradski ured
- **Sudionici:** Baza podataka
- **Preduvjet:** -
- **Opis osnovnog tijeka:**
  1. Korisnik odabire opciju registracije
  2. Korisnik popunjava podatke za izradu korisničkog računa i potvrđuje registraciju
  3. Podaci o novom računu se spremaju u bazu podataka
- **Opis mogućih odstupanja:**

  2.a Unesena email adresa je već zauzeta
    1. Aplikacija prikazuje prikladnu obavijest i vraća korisnika na stranicu za registraciju
    2. Korisnik unosi drugu email adresu ili odustaje od registracije

2.b Unesena lozinka je preduga ili prekratka
   1. Aplikacija prikazuje prikladnu obavijest i vraća korisnika na stranicu za registraciju
   2. Korisnik unosi drugu lozinku ili odustaje od registracije

UC7 - Pregled vlastitih prijava

- **Glavni sudionik:** Registrirani korisnik
- **Cilj:** Pregled prijava koje je korisnik do sada podnio
- **Sudionici:** Baza podataka
- **Preduvjet:** Korisnik pregledava svoj korisnički račun
- **Opis osnovnog tijeka:**
  1. Registrirani korisnik prilikom pregleda korisničkog računa odlazi na sekciju pregleda vlastitih prijava
  2. Aplikacija prikazuje listu dosadašnjih prijava Registriranog korisnika
  3. Registrirani korisnik odabire prijavu
  4. Prikazuju se podaci o prijavi
- **Opis mogućih odstupanja:**

  3.a Registrirani korisnik ne želi pregledati konkretnu prijavu
     1. Korisnik nakon pregleda liste prijava izlazi iz opcije pregleda vlastitih prijava

UC8 - Pregled korisničkog računa

- **Glavni sudionik:** Registrirani korisnik
- **Cilj:** Pregled vlastitog korisničkog računa
- **Sudionici:** Baza podataka
- **Preduvjet:** -
- **Opis osnovnog tijeka:**
  1. Registrirani korisnik odabire prikaz stranice svojeg profila
  2. Aplikacija prikazuje podatke o korisničkom računu
  3. Registrirani korisnik pregledava podatke o svojem korisničkom računu

UC9 - Promjena podataka o računu

- **Glavni sudionik:** Registrirani korisnik
- **Cilj:** Promjena podataka o korisničkom računu
- **Sudionici:** Baza podataka
- **Preduvjet:** Registrirani korisnik pregledava svoj korisnički račun

- **Opis osnovnog tijeka:**
  1. Registrirani korisnik mijenja željene podatke o računu
  2. Registrirani korisnik potvrđuje izmjene i vraća se u pregled korisničkog računa
  3. Izmijenjeni podaci se spremaju u bazu podataka
- **Opis mogućih odstupanja:**

  4.a Registrirani korisnik je promijenio email adresu u već zauzetu email adresu
    1. Aplikacija javlja pogrešku s odgovarajućom porukom i vraća korisnika na stranicu za registraciju
    2. Registrirani korisnik unosi drugu email adresu ili odustaje od promjene email adrese ili uređivanja podataka o korisničkom računu

UC10 - Brisanje korisničkog računa

- **Glavni sudionik:** Registrirani korisnik
- **Cilj:** Brisanje korisničkog računa
- **Sudionici:** Baza podataka
- **Preduvjet:** Registrirani korisnik pregledava svoj korisnički račun
- **Opis osnovnog tijeka:**
  1. Registrirani korisnik odlazi na stranicu svojeg profila
  2. Registrirani korisnik prilikom pregledavanja svojeg korisničkog računa odabire opciju brisanja korisničkog računa
  3. Korisnički račun se briše, korisnik postaje Neregistrirani korisnik i aplikacija ga vraća na početnu stranicu

UC11 - Prijava u sustav

- **Glavni sudionik:** Registrirani korisnik ili Djelatnik gradskog ureda
- **Cilj:** Prijava korisnika u sustav
- **Sudionici:** Baza podataka
- **Preduvjet:** -
- **Opis osnovnog tijeka:**
  1. Korisnik odabire opciju prijave
  2. Korisnik unosi svoju email adresu i lozinku
  3. Aplikacija otvara početnu stranicu
- **Opis mogućih odstupanja:**

  2.a Registrirani korisnik unosi nevažeću kombinaciju email adrese i lozinke

1. Aplikacija javlja pogrešku s odgovarajućom porukom i traži ponovnu prijavu
2. Korisnik unosi ispravnu kombinaciju ili odustaje od prijave i koristi aplikaciju u anonimnom načinu

UC12 - Odjava iz sustava

- **Glavni sudionik:** Registrirani korisnik ili Djelatnik gradskog ureda
- **Cilj:** Odjava korisnika iz sustav
- **Sudionici:** -
- **Preduvjet:** Korisnik je prijavljen u sustav
- **Opis osnovnog tijeka:**
  1. korisnik odabire ikonu svojeg profila
  2. Korisnik na ponuđenom izborniku odabire opciju odjave iz sustava
  3. Aplikacija otvara početnu stranicu u anonimnom načinu

UC13 - Obrada prijava

- **Glavni sudionik:** Djelatnik gradskog ureda
- **Cilj:** Obrada prijave u gradskom uredu
- **Sudionici:** Baza podataka, Gradska služba
- **Preduvjet:** Djelatnik pregledava prijave ureda
- **Opis osnovnog tijeka:**
  1. Djelatnik gradskog ureda odabire prijavu
  2. Djelatnik mijenja status prijave
  3. Prijava se delegira nadležnoj gradskoj službi ako joj je status promijenjen u *u procesu rješavanja*

UC14 - Odbacivanje pristigle prijave

- **Glavni sudionik:** Djelatnik gradskog ureda
- **Cilj:** Odbacivanje nevaljale prijave
- **Sudionici:** Baza podataka
- **Preduvjet:** Djelatnik gradskog ureda obrađuje prijavu, prijava ima status *na čekanju*
- **Opis osnovnog tijeka:**
  1. Djelatnik procjenjuje da prijava nije utemeljena
  2. Djelatnik odabire opciju za odbacivanje prijave

UC15 - Prebacivanje prijave na drugi ured

- **Glavni sudionik:** Djelatnik gradskog ureda
- **Cilj:** Prosljeđivanje prijave nadležnom gradskom uredu
- **Sudionici:** Baza podataka
- **Preduvjet:** Djelatnik gradskog ureda obrađuje prijavu, prijava ima status *na čekanju*
- **Opis osnovnog tijeka:**
  1. Djelatnik procjenjuje da je neki drugi ured nadležan za ovu prijavu
  2. Djelatnik odabire opciju prebacivanja prijave na drugi ured
  3. Djelatnik odabire novu kategoriju te prijave (čime indirektno mijenja ured) i potvrđuje unos

UC16 - Objedinjenje nepovezanih prijava

- **Glavni sudionik:** Djelatnik gradskog ureda
- **Cilj:** Objedinjavanje prijava istog oštećenja
- **Sudionici:** Baza podataka
- **Preduvjet:** Djelatnik gradskog ureda obrađuje prijavu, prijava ima status *na čekanju*
- **Opis osnovnog tijeka:**
  1. Djelatnik procjenjuje da postoji više prijava istog oštećenja
  2. Djelatnik odabire opciju objedinjavanja prijava
  3. Djelatnik odabire prijave koje želi objediniti i potvrđuje odabir

UC17 - Slanje povratnih informacija

- **Glavni sudionik:** Djelatnik gradskog ureda
- **Cilj:** Informiranje prijavitelja o promjenama u njegovoj prijavi
- **Sudionici:** Baza podataka
- **Preduvjet:** Djelatnik gradskog ureda je prilikom obrade prijave izvršio neku radnju/promjenu nad prijavom
- **Opis osnovnog tijeka:**
  1. Aplikacija šalje obavijest prijavitelju o promjenama vezanim uz prijavu
- **Opis mogućih odstupanja:**

  1.a Prijava je anonimna
    1. Aplikacija ne radi ništa

UC18 - Pregled prijava gradskog ureda

- **Glavni sudionik:** Djelatnik gradskog ureda

- **Cilj:** Pregledavanje prijava koje gradski ured obrađuje/treba obraditi/je obradio
- **Sudionici:** Baza podataka
- **Preduvjet:** -
- **Opis osnovnog tijeka:**
  1. Djelatnik gradskog ureda odlazi na glavnu stranicu za Djelatnike gradskog ureda
  2. Aplikacija prikazuje prijave iz liste novopristiglih prijava

UC19 - Filtriranje prijava gradskog ureda

- **Glavni sudionik:** Djelatnik gradskog ureda
- **Cilj:** Filtriranje prijava koje gradski ured mora obraditi/obrađuje/je obradio
- **Sudionici:** Baza podataka
- **Preduvjet:** Djelatnik gradskog ureda pregledava prijave
- **Opis osnovnog tijeka:**
  1. Djelatnik gradskog ureda odabire listu prijava prema statusu prijava
  2. Aplikacija prikazuje odabranu listu prijava

UC20 - Prepoznavanje kategorije

- **Glavni sudionik:** Korisnik
- **Cilj:** Automatsko prepoznavanje kategorije oštećenja
- **Sudionici:** Baza podataka
- **Preduvjet:** Korisnik podnosi prijavu oštećenja
- **Opis osnovnog tijeka:**
  1. Aplikacija dohvaća popis ključnih riječi i njihovih kategorija
  2. Korisnik unosi opis oštećenja
  3. Aplikacija na temelju prepoznanih ključnih riječi predlaže kategoriju
  4. Korisnik ostavlja predloženu kategoriju ili ručno odabire ispravnu kategoriju

UC21 - Nadovezivanje prijave

- **Glavni sudionik:** Korisnik
- **Cilj:** Nadovezivanje prijave na već postojeću prijavu istog oštećenja
- **Sudionici:** Baza podataka
- **Preduvjet:** Korisnik podnosi prijavu oštećenja
- **Opis osnovnog tijeka:**

1. Korisnik potvrđuje i podnosi prijavu
2. Aplikacija provjerava ako u bazi podataka već postoji slična prijava
3. Ako takva postoji, aplikacija nudi korisniku da se nadoveže na već postojeću prijavu
4. Korisnik može nadovezati prijavu ili ju podnijeti kao samostalnu

Dijagrami obrazaca uporabe

Slika 3.1: Dijagram obrasca uporabe, mogućnosti korisnika

Slika 3.2: Dijagram obrasca uporabe, obrada prijava

3.1.2 Sekvencijski dijagrami

Obrasci uporabe - UC06, UC11 - Registracija u sustav, Prijava u sustav

Korisnik(može biti i djelatnik) na naslovnoj stranici odabire opciju prijave ili registracije te mu se zatim otvara prikladna stranica. Ako se radi o stranici za prijavu, korisnik unosi svoj email i zaporku i potvrđuje unos. Ako su uneseni podaci validni korisniku se prikazuje naslovna stranica za prijavljenog korisnika, odnosno za djelatnika gradskog ureda. U suprotnom se korisnika obavještava o grešci i ponovno traži unos podataka za prijavu. Ako se radi o stranici za registraciju, korisnik mora unijeti svoje ime, prezime, datum rođenja, email adresu i lozinku. Ako se registrira novi gradski ured, potrebno je označiti tu opciju. U tom slučaju se traži unos imena ureda, email adresa, lozinka i barem jedna kategorija za koju će novi gradski ured biti nadležan. Sustav provjerava u bazi ako za danu email adresu već postoji korisnički račun. Ako račun već postoji, sustav korisniku javlja grešku i ponovno traži unos podataka za registraciju. Ako je unos dobar, korisniku se prikazuje naslovna stranica za prijavljenog korisnika, odnosno za djelatnika gradskog ureda. Korisnik ne mora odabrati opciju prijave ili registracije. U tom slučaju koristi stranicu kao anoniman neregistrirani korisnik. Djelatnik se mora prijaviti ili registrirati ured da bi obavljao svoju funkcionalnost.

Slika 3.3: Sekvencijski dijagram, prijava/registracija

Obrasci uporabe - UC1 - Prijava oštećenja

Korisnik odabire opciju podnošenja nove prijave oštećenja. U prijavi korisnik upisuje kratak naslov, opis oštećenja, lokaciju i kategoriju. Korisnik također može priložiti fotografiju oštećenja. Lokaciju korisnik može priložiti preko interaktivne karte, upisom adrese ili se lokacija može izvući iz meta podataka fotografija, ako je fotografija priložena. Aplikacija će sama pokušati prepoznati kategoriju oštećenja i na temelju toga i ponuditi nadležan ured. Za to su joj potrebne ključne riječi za svaku kategoriju koje je aplikacija dohvatila prilikom odabira podnošenja nove prijave. Aplikacija nakon unosa podataka dohvaća postojeće prijave koje imaju sličnu lokaciju, istu kategoriju i koje još nisu obrađene. Ako takve prijave postoje, aplikacija će korisniku ponuditi da svoju prijavu nadoveže na jednu od ponuđenih prijava. Korisnik može, ali i ne mora nadovezati svoju prijavu kada mu je ta opcija ponuđena. Nakon podnošenja prijave korisnik dobiva kod prijave preko koje može pratiti njezino stanje.

Slika 3.4: Sekvencijski dijagram, podnošenje prijave

Obrasci uporabe - UC13, UC14, UC15, UC16, UC17, UC18, UC19 - Obrada prijava, Odbacivanje pristigle prijave, Prebacivanje prijave na drugi ured, Objedinjenje nepovezanih prijava, Slanje povratnih informacija, Pregled prijava gradskog ureda, filtriranje prijava gradskog ureda

Djelatnik gradskog ureda odabire jednu od 3 lista prijava, jedna za svaki status. Za izlistane prijave djelatniku se prikazuju podaci o svakoj prijavi. Djelatnik dalje može promijeniti status prijave. Ako vidi da je oštećenje u toj prijavu već prijavljeno, djelatnik može odabrati sve takve prijave i objediniti ih. Ako utvrdi da njegov ured nije nadležan za ovu prijavu, djelatnik odabire novu kategoriju i time prosljeđuje prijavu drugom uredu. Ako je prijava u potpunosti nevaljana djelatnik ju može odbaciti. Bilo koja od ovih radnji nad prijavom rezultirat će slanjem povratne informacije korisniku koji je podnio prijavu, ako je taj korisnik registriran.

Slika 3.5: Sekvencijski dijagram, obrada prijava

3.2 Ostali zahtjevi

- Sustav treba podržati rad više korisnika u stvarnom vremenu
- Sustav treba biti implementiran kao web aplikacija s responzivnim dizajnom
- Aplikacija mora koristiti HTTPS protokol
- Aplikacija mora biti jednostavna za korištenje
- Neispravno korištenje korisničkog sučelja ne smije narušiti cjelokupan rad sustava
- Izvršavanje bilo koje akcije na aplikaciji ne smije trajati dulje od 10 sekundi
- Osjetljivi podaci, kao što su lozinke korisnika, se u bazu podataka moraju spremati u kriptiranom obliku koristeći *bcrypt* algoritam

4. Arhitektura i dizajn sustava

Arhitektura ovog sustava je bazirana na arhitekturi klijent-poslužitelj. Sustav se sastoji od 3 sloja: sloj korisničkog sučelja, sloj aplikacijske logike, sloj podataka.

Web preglednik je program preko kojeg korisnik koristi aplikaciju. Preglednik omogućuje klijentu komunikaciju s web poslužiteljem aplikacije. Preglednik nam omogućava prikaz sloja korisničkog sučelja sustava. Korisnik preko web preglednika komunicira s web poslužiteljem slanjem i primanjem HTTP zahtjeva. Primljene podatke i datoteke web preglednik zna interpretirati i prikazati korisniku tako da sama interakcija s aplikacijom bude jednostavna. Neki od popularnih web preglednika su Chrome, Safari i Edge.

Web poslužitelj je računalo na kojem se aplikacija pokreće. Na njemu se dakle nalazi sloj aplikacijske logike. Web poslužitelj aplikaciji prosljeđuje zahtjeve na obradu i njezine odgovore prosljeđuje natrag klijentima. Web aplikacija na poslužitelju obrađuje zaprimljene zahtjeve. Ako obrada zahtjeva to zahtjeva, web aplikacija dodatno komunicira sa slojem podataka.

Baza Podataka predstavlja sloj podataka. U bazi podataka se na siguran način spremaju svi podaci koje je potrebno trajno čuvati. To podrazumijeva podatke o prijavama, korisničkim računima i slično. Takvi podaci moraju ostati očuvani i ako je rad aplikacije prekinut iz bilo kojeg razloga. Baza podataka je detaljnije opisana u poglavlju 4.1.

Web aplikacija je podijeljena na frontend i backend.

Frontend je zapravo prezentacijski dio aplikacije i zadužen je za interakciju s korisnikom. On oblikuje korisničko sučelje i samim time definira što će korisnik vidjeti na web pregledniku kada koristi aplikaciju.

Backend je dio aplikacije koji obrađuje zahtjeve i kontrolira rad ostalih dijelova sustava. Baceknd je usko povezan s web poslužiteljem.

Za izradu backend dijela aplikacije odabrali smo programski jezik Java s radnim okvirom Spring Boot. Spring Boot radni okvir koristi višeslojnu arhitekturu u kojoj svaki sloj komunicira sa slojem koji se u hijerarhiji nalazi iznad ili ispod njega. Arhitektura Spring Boota se sastoji od četiri sloja:

- **Prezentacijski sloj** (*Presentation layer*) - Ovo je najviši sloj arhitekture i pred-

stavlja frontend dio aplikacije. Ovaj sloj prima HTTP zahtjeve i vrši autentifikaciju. Također vrši pretvorbu JSON objekata u objekte u Javi i obrnuto. Na kraju svoje obrade Prezentacijski sloj prosljeđuje HTTP zahtjev sljedećem sloju.

- **Poslovni sloj** (*Business layer*) - Ovaj sloj sadrži svu poslovnu logiku te je zadužen za validaciju i autorizaciju.
- **Sloj postojanosti** (*Persistence layer*) - Ovaj sloj sadrži logiku vezanu uz pohranu u bazu podataka. Sloj pretvara zapise iz baze podataka u objekte iz poslovnog sloja i obrnuto.
- **Sloj baze podataka** (*Database layer*) - Ovaj sloj sadrži bazu podataka ili više njih. Sloj je zadužen za obavljanje CRUD (copy, read, update, delete) operacija.

Slika 4.1: Prikaz Spring Boot arhitekture

Za razvoj frontend dijela aplikacije odlučili smo koristiti programski jezik JavaScript uz radni okvir React. React omogućuje jednostavnu izradu korisničkog sučelja preko JavaScripta, bez direktnog korištenja HTML-a.

Za razvojne okoline smo odlučili koristiti IntelliJ IDEA za rad u Spring Boot radnom okviru i Visual Studio Code za rad u React radnom okviru.

4.1 Baza podataka

Ovaj sustav će za svoje potrebe koristiti relacijsku bazu podataka implementirana u PostgreSQL-u. Ovakav tip baze podataka svojom strukturom omogućuje jednostavno modeliranje elemenata iz stvarnog svijeta i zbog toga je široko primjenjiv. Odabrali smo implementaciju u PostgreSQL-u jer smo s ovom implementacijom već dobro upoznati. Relacijske baze podataka se sastoje od relacija. Relacije su predstavljene tablicama koje su definirane svojim nazivom i skupom atributima. Bazu podataka koristimo za jednostavnu i sigurnu pohranu, izmjenu, umetanje i dohvat podataka potrebnih za rad sustava. Naša baza podataka se sastoji od sljedećih relacija, odnosno tablica:

- Users
- Report
- Image
- Category
- CategoryKeywords
- ReportGroup
- Feedback
- CityOffice

4.1.1 Opis tablica

Users tablica čuva podatke o korisničkim računima koje korisnici izrađuju tijekom registracije. Tablica je povezana vezom *One-to-Many* s tablicom *Report* preko atributa *userID*.

Tablica 4.1:

Users		
userID	INT	jedinstveni identifikator korisnika
email	VARCHAR	email adresa korisnika
firstName	VARCHAR	ime korisnika
lastName	VARCHAR	prezime korisnika
password	VARCHAR	kriptirana lozinka za korisnički račun

Report tablica čuva podatke o pojedinim prijavama oštećenja koje korisnici

podnose. Tablica je povezana vezom *Many-to-One* s tablicom *Users* preko atributa *userID*, *Many-to-One* vezom s tablicom *Category* preko atributa *categoryID*. Također postoji auto-refleksivna *Many-to-One* veza kojom se modelira nadovezivanje prijava.

Tablica 4.2:

Report		
reportID	INT	jedinstveni identifikator prijave
description	VARCHAR	opis oštećenja
reportTS	TIMESTAMP	vrijeme prijave
reportHeadline	VARCHAR	naslov prijave
lng	REAL	geografska dužina lokacije oštećenja
lat	REAL	geografska širina lokacije oštećenja
userID	INT	jedinstveni identifikator korisnika koji je podnio prijavu
categoryID	INT	jedinstveni identifikator kategorije oštećenja
isConnectedToID	INT	jedinstveni identifikator prijave na koju je ova prijava nadovezana

Image tablica čuva podatke o slikama koje se prilažu u prijavama oštećenja. Tablica je povezana *Many-to-One* vezom s tablicom *Report* preko atributa *reportID*.

Tablica 4.3:

Image		
imageID	INT	jedinstveni identifikator slike
reportID	INT	jedinstveni identifikator prijave kojoj slika pripada
URL	VARCHAR	URL slike

Category tablica čuva podatke o kategorijama kojima oštećenje može pripadati. Tablica je povezana *One-to-Many* vezom s tablicom *Report* preko atributa *categoryID*, *Many-to-One* vezom s tablicom *CityOffice* preko atributa *cityOfficeID* i

One-to-Many vezom s tablicom *CategoryKeywords* preko atributa *categoryID*.

Tablica 4.4:

Category		
categoryID	INT	jedinstveni identifikator kategorije
categoryName	VARCHAR	naziv kategorije
cityOfficeID	INT	jedinstveni identifikator gradskog ureda koji obrađuje prijave oštećenja iz ove kategorije

CategoryKeywords tablica čuva podatke o ključnim riječima po kojima se može identificirati kategorija oštećenja. Tablica je povezana *Many-to-One* vezom s tablicom *Category* preko atributa *categoryID*.

Tablica 4.5:

CategoryKeywords		
keywordID	INT	jedinstveni identifikator ključne riječi
keyword	VARCHAR	ključna riječ
categoryID	INT	jedinstveni identifikator kategorije kojoj ključna riječ pripada.

Feedback tablica čuva podatke o statusu prijava koje su povezane. Također čuva podatke potrebne za slanje povratnih informacija korisnicima i računanje nekih statistika. Tablica je povezana *Many-to-One* vezom s tablicom *Report* preko atributa *reportID*.

Tablica 4.6:

Feedback		
reportID	INT	jedinstveni identifikator grupe prijava na koju se podaci referiraju
status	VARCHAR	status prijava u referiranoj grupi
changeTS	TIMESTAMP	vrijeme kada se postavio status prijava iz referirane grupe

CityOffice tablica čuva podatke o računima gradskih ureda. Tablica je pove-

zana *One-to Many* vezom s tablicom *Category* preko atributa *cityOfficeID*.

Tablica 4.7:

CityOffice		
cityOfficeID	INT	jedinstveni identifikator gradskog ureda
cityOfficeName	VARCHAR	naziv gradskog ureda
cityOfficeEmail	VARCHAR	email adresa gradskog ureda
cityOfficePassword	VARCHAR	kriptirana lozinka za račun gradskog ureda

4.1.2 Dijagram baze podataka

Slika 4.2: Relacijski dijagram baze podataka

4.2 Dijagram razreda

Na slikama 4.3, 4.4, 4.5, 4.6 su prikazani razredi koji pripadaju backend dijelu aplikacije. Razredi su arhitekturom podijeljeni na kontrolere, repozitorije i servise. Također postoje Domain i DTO (Data Transfer Object) modeli. Svi dijagrami su podložni promjenama, s obzirom na to da je aplikacija još uvijek u fazi razvoja.

Slika 4.3 prikazuje klase koje imaju ulogu kontrolera, tj. u kodu imaju anotaciju @RestController. Te klase definiraju endpointove i koriste se za dohvat i manipuliranje podacima iz baze podataka. ClientForwardController je kontroler koji povezuje Spring server React aplikacijom. Klase sa @RestController anotacijom koriste instance servisa. Servisi su klase koje imaju @Service anotaciju. Servisi implementiraju metode za upis (npr. createUser(Users);, createCitiyOffice(cityOffice), itd.), brisanje, dohvat i izmjenu podataka u bazi. Klasa ServerApplication je glavna klasa koja pokreće aplikaciju, a ServerApplicationTest služi za testiranje funkcionalnosti aplikacije, ali još nema implementiranu nikakvu funkcionalnost.

Slika 4.3: Kontroleri na backendu

Slika 4.4 prikazuje sučelja koja nasljeđuju sučelje JpaRepository. Sučelje JpaRepository definira razne CRUD (create, read, update, delete) metode za rad nad entitetima u bazi. Definirane metode izvršavaju razne vrste upite nad bazom podataka.

Slika 4.4: Data Access Objects

Slika 4.5 prikazuje klase u paketu repo koje reprezentiraju entitete u bazi podataka te njihove međusobne veze. Ove klase definiraju tipove atributa pojedinih entiteta, ograničenja nad atributima i primarne i strane ključeve. Svaka klasa ima implementirane metode get() i set() za svaki od atributa, kao i odgovarajuće konstruktore. Klasa FeedbackID služi kao pomoćna klasa za definiranje primarnog ključa feedback. Korisnici (Users) stvaraju prijave (Report), a gradski uredi (CityOffice) upravljaju prijavama i prilikom promjena automatski zapisuje nove povratne informacije (Feedback) u bazu podataka za tu grupu prijava.

Slika 4.5: Modeli

Slika 4.6 prikazuje klase servisa koje definiraju metode za CRUD operacije nad bazom podataka. EntityMissingException i RequestDeniedException su pomoćne klase za baratanje pogreškama. Klase servisa koriste instance Repository klasa iz slike 2. za poziv metoda definiranih u sklopu sučelja Repository odnosno JpaRepository.

Slika 4.6: Servisi

4.3 Dijagram stanja

Dijagram stanja je vrsta UML dijagrama koja prikazuje stanja objekta i prijelaze između stanja. Prijelazi se aktiviraju određenim događajima i opcionalno dodatnim uvjetima.

Slika 4.7 prikazuje dijagram stanja za korisnika koji može, ali i ne mora biti prijavljen u sustav.

Korisniku je na početku prikazana glavna stranica na kojoj je karta sa prikazanim aktivnim prijavama u sustavu. Zaglavlje stranice sadrži gumbove za podnošenje nove prijave oštećenja (”Prijava štete”), prikaz statistike prijava (”Statistika”). Također, zaglavlje sadrži gumb sa e-mailom korisnika koji služi za upravljanje korisničkim računom ako je korisnik prijavljen u sustav. Ako korisnik nije prijavljen, zaglavlje sadrži gumbove za prijavu (”Log in”) i registraciju (”Register”) u sustav. S obzirom da je zaglavlje isto na svim dijelovima aplikacije, korisnik ima pristup ovim opcijama bilo gdje u aplikaciji.

Na glavnoj stranici korisnik može odabrati filtar prijava. Pomoću tog filtra korisnik može filtrirati aktivne prijave u sustavu koje će se prikazati na karti na početnoj stranici. Odabirom neke od prikazanih prijava na karti korisniku se otvara stranica odabrane prijave.

Kod pregleda statistike prijava korisnik mora ispuniti obrazac filtra. Nakon potvrde obrasca korisniku se prikazuje statistika prijava koje su odabrane pomoću filtra.

Prilikom prijave novog oštećenja korisnik ispunjava obrazac prijave. Kada je u obrazac upisan opis oštećenja, sustav korisniku predlaže kategoriju oštećenja. Kada korisnik podnese prijavu, sustav provjerava ako već postoji slična prijava. Ako postoji, sustav nudi korisniku nadovezivanje njegove prijave na postojeću. Korisnik to može prihvatiti, ali može i poslati nepovezanu prijavu. Nakon konačnog podnošenja prijave sustav korisniku prikazuje jedinstveni kod prijave. Korisnik se vraća na početnu stranicu kada potvrdi zaprimljeni kod prijave.

Ako je korisnik prijavljen može pregledati podatke o svojem korisničkom računu. Prilikom pregleda korisničkog računa može dodatno pregledavati vlastite prijave. Kada želi pregledati neku od prijava, korisniku se prikazuje stranica te prijave, isto kao i kod pregleda svih prijava na početnoj stranici. Korisnik također može mijenjati podatke o svojem korisničkom računu. Kod potvrde se izmjene spremaju, ako je do njih došlo. Također, korisnik kod pregleda računa može izbrisati svoj račun

čime on postaje neprijavljen i neregistriran.

Slika 4.7: Dijagram stanja

4.4 Dijagram aktivnosti

Dijagram aktivnosti je ponašajni UML dijagram koji modelira ponašanje sustava nizom akcija koje zajedno čine aktivnost. Dijagram aktivnosti se primjenjuje za modeliranje upravljačkog i podatkovnog toka. Ovakav tip dijagrama nije preporučljiv za modeliranje sustava sa događajima poticanim ponašanjem. Kod dijagrama aktivnosti svaka akcija se izvršava nakon završavanja prethodne akcije. Općenito je u dijagramu aktivnosti naglasak na jednostavnosti prikaza.

Na dijagramu 4.8 prikazan je proces podnošenja prijave oštećenja i njezine obrade u gradskom uredu.

Korisnik odabire opciju "Prijava štete" te ispunjava podatke o oštećenju na dobivenom obrascu. Kada je unesen opis prijave, aplikacija pomoću ključnih riječi pokušava prepoznati kategoriju oštećenja koju predlaže korisniku. Korisnik može prihvatiti predloženu kategoriju ili ručno odabrati drugu. Nakon što korisnik podnese prijavu, sustav provjerava ako već postoji slična prijava spremljena u bazi podataka. Ako postoji, sustav predlaže korisniku nadovezivanje njegove prijave na postojeću. Korisnik može, ali ne mora prihvatiti prijedlog i nakon toga potvrditi slanje prijave.

Nakon što je prijava podnesena i spremljena u bazu podataka, djelatnik gradskog ureda može obraditi tu prijavu. Ako smatra da je prijava nevaljala, djelatnik ju odbacuje. Ako je prijava podnesena krivom uredu, korisnik prijavu može proslijediti drugom uredu. Ako je prijava valjana, djelatnik tu prijavu šalje gradskoj službi za sanaciju oštećenja. U svakom od navedenih slučaja promjena u prijavi se sprema u bazu podataka i obavještava se korisnika koji je podnio prijavu, ako je korisnik registriran u sustavu. Na kraju, kada gradska služba sanira oštećenje iz valjane prijave, ta se promjena također evidentira u bazi podataka i obavještava se korisnik kao i u ostalim slučajevima.

Slika 4.8: Dijagram aktivnosti

4.5 Dijagram komponenti

Dijagram komponenti je strukturni statički UML dijagram koji je dio specifikacije arhitekture programske potpore. Njime se vizualizira organizacija i međuovisnost između implementacijskih komponenata i odnos same programske okoline prema okolini. Ovakav tip dijagrama je pogodan kod komponentno-usmjerenog modela razvoja programske potpore i uslužno-orijentirane arhitekture. Osnovni elementi dijagrama komponenti su komponente, sučelja i poveznice.

Na dijagramu 4.9 je prikazana organizacija i međuovisnost komponenata i sučelja u sustavu za prijavu oštećenja na javnim površinama. Dijagram se sastoji od komponente *Web preglednik*, *PostgreSQL baza podataka* i *Web aplikacija*. *Web aplikacija* ima dvije pod-komponente: *React frontend* i *Spring Boot backend*.

Pomoću *Web preglednik* komponente se šalju zahtjevi *Wev aplikacija* komponenti za pojedinim stranicama preko sučelja za dohvat vizualnog sadržaja. *React router* komponenta unutar *React frontend* komponente na temelju URL-a dobivenog od komponente *Web preglednik* donosi odluku koja će se datoteka dohvatiti, to jest koja će se stranica prikazati pomoću komponente *React view*. Komponenta *index.js* je početna komponenta koja služi kao korijen za sve ostale elemente za prikaz. *ReactJS* je biblioteka za sami React. *React view* komponenta po potrebi korisniku osvježava prikaz i preko REST API-ja razmjenjuje podatke sa *Spring Boot backend* komponentom aplikacije u JSON formatu.

Controller komponenta unutar *Spring Boot backend* komponente preko REST API-ja prima zahtjeve, šalje ih dalje na obradu i nakon obrade šalje odgovore na njih. *Service* komponenta prima zahtjeve od *Controller* komponente, obrađuje ih i po potrebi komunicira sa *Repository* komponentom. *Repository* komponenta komunicira sa *JpaRepository* komponentom preko JPA sučelja, a *JpaRepository* komponenta preko PSQL sučelja komunicira sa *PostgreSQL baza podataka* komponentom koja služi za trajnu pohranu podataka. *Controller*, *Service* i *Repository* komponente koriste *DTO* komponentu koja služi za enkapsulaciju, organizaciju i razmjenu podataka između komponenti.

Slika 4.9: Dijagram komponenti

5. Implementacija i korisničko sučelje

5.1 Korištene tehnologije i alati

Backend dio aplikacije pisan je u objektno orijentiranom programskom jeziku **Java**[1]. Jedna od glavnih značajki ovog jezika je takozvano načelo WORA (eng. *write once, run anywhere*) koje omogućava da se jednom prevedeni kod pomoću Javinog virtualnog stroja može izvršavati na bilo kojem računalu. Jedini preduvjet je da to računalo na sebi ima instaliranu Javinu platformu.

Kako bi se olakšao razvoj backend dijela aplikacije, korišten je radni okvir **Spring Boot**[2] koji je specijalizacija radnog okvira Spring. Spring Boot je popularan radni okvir za izgradnju samostojećih web aplikacija. Glavna prednost ovog radnog okvira je automatska konfiguracija dijelova koda koji se standardno pojavljuju kod web aplikacija. Tako Spring Boot automatizira komunikaciju s bazom podataka, rad s datotekama u JSON formatu, podjela programskog koda u unaprijed definirane slojeve itd.

Backend dio aplikacije je razvijan u razvojnom okruženju **IntelliJ IDEA**[3] tvrtke JetBrains. Samo razvojno okruženje je namjenjenu razvoju u programskim jezicima Java, Kotlin, Groovy, Scala te je jedno od vodećih okruženja za te programske jezike. IntelliJ IDEA programeru nudi razne funkcionalnosti koje ubrzavaju i olakšavaju programiranje kao što su samozavršavanjne koda analizom konteksta, navigacija u kodu, refaktoriranje koda, debuggiranje, rad sa distribuiranim sustavima za upravljanje verzijama podataka, kao što je Git, rad s bazom podataka itd. Također, IntelliJ IDEA programer može dodatno nadograditi instaliranjem proširenja za dodatne funkcionalnosti koja su dostupna na web repozitoriju JetBrainsa.

Frontend dio aplikacije je pisan u programskom jeziku **JavaScript**[4] koji efektivno postao nezaobilazan jezik za razvoj klijentske strane web aplikacija. JavaScript je dinamičan skriptni jezik sa mekim tipovima podataka (eng. *soft-typed*)

[1] https://www.java.com/en/
[2] https://spring.io/projects/spring-boot/
[3] https://www.jetbrains.com/idea/
[4] https://www.javascript.com/

kojeg podržava svaki moderan web preglednik. Za sam jezik JavaScript postoji napisano puno javno dostupnih biblioteka koje olakšavaju ostvarivanje raznih funkcionalnosti u kodu i potiču ponovnu uporabu postojećeg koda. Iako je Javascript de facto standard za razvoj frontend dijela aplikacije, također se može koristiti i za razvoj backend dijela aplikacije.

Za brži i lakši razvoj frontend dijela aplikacije korištena je vrlo popularna biblioteka **React**[5] razvijena od tvrtke Meta (nekada Facebook). Aplikacije napravljene u Reactu koriste komponente koje se mogu koristiti na više mjesta. React koristi virtualni DOM (Document Object Model) te se pomoću Reacta grade jednostranične web aplikacije koje osvježavaju samo komponente koje je potrebno mijenjati. Time se značajno poboljšavaju performanse web aplikacije.

Kako bi dodatno olakšali vizualno oblikovanje korisničkog sučelja korišten je radni okvir **Bootstrap**[6]. Bootstrap je popularan radni okvir za stilski jezik CSS koji olakšava vizualno oblikovanje korisničkog sučelja nuđenjem gotovih dizajnova osnovnih elemenata web stranice kao što su na primjer gumbovi, forme i sl.

Za razvoj frontend dijela aplikacije korišteno je razvojno okruženje **Visual Studio Code**[7] tvrtke Microsoft. Visual Studio Code je moguće nadograđivati raznim proširenjima pomoću kojih se može stvoriti razvojno okruženje za gotovo svaki programski jezik kao što je C, C++, Java, JavaScript, Python, Go, itd. Zbog te fleksibilnosti se Visual Studio Code može koristiti u gotovo svakoj domeni koja uključuje neku vrstu programiranja.

Baza podataka je napravljena u sustavu **PostgreSQL**[8]. PostgreSQL je jedan od najraširenijih sustava za upravljanje relacijskim bazama podataka temeljen na SQL-u (Structured Query Language). PostgreSQL omogućuje izradu baza podataka sa raznim funkcionalnostima kao što su okidači, transakcije, procedure, mogućnost paralelnog pristupa više korisnika istovremeno, konzistencija i sigurnost podataka, itd.

Za spremanje slika oštećenja koristili smo **Firebase**[9]. Firebase je platforma za razvoj web i mobilnih aplikacija. Također nudi uslugu pohrane datoteka u oblaku što je i nama trebalo u ovom projektu.

Svi dijagrami za opis sustava napravljeni su u **UML-u**[10] (Unified Modeling Lan-

[5] https://react.dev/
[6] https://getbootstrap.com/
[7] https://code.visualstudio.com/
[8] https://www.postgresql.org/
[9] https://firebase.google.com/
[10] https://www.uml.org/

guage). UML je generički jezik za vizualno modeliranje. Napravljen je kako bi pružao standard za dizajniranje sustava. Taj standard omogućava bolju vizualizaciju, specifikaciju, oblikovanje i dokumentiranje artefakata programske potpore. U svrhu vizualizacije pojedinog aspekta sustava definiran je veliki broj vrsta dijagrama koji omogućuju različite poglede na pojedini dio sustava.

Za izradu UML dijagrama korišten je grafički alat **Visual Paradigm**[11]. On podržava izradu mnogih vrsta UML dijagrama kao što su dijagram obrazaca uporabe, sekvencijski dijagram, dijagram stanja, dijagram komponenti, dijagram aktivnosti, dijagram razmještaja, dijagram razreda, i mnogi drugi.

Za potrebe izrade dijagrama razreda korištena je funkcionalnost razvojnog okruženja IntelliJ IDEA za automatsko generiranje dijagrama razreda iz postojećeg programskog koda. Dobiveni dijagram razreda je dodatno uređen i dorađen u grafičkom alatu za izradu dijagrama **draw.io**[12].

Izrada relacijskog dijagrama baze podataka i modeliranje same baze podataka ostvareno je uz grafički alat **ERDPlus**[13]. ERDPlus omogućuje izradu ER dijagrama baze podataka koji se sastoji od entiteta i veza između njih. Iz ER dijagrama je moguće automatsko generiranje relacijskih dijagrama baze podataka. Iz dobivenih relacijskih dijagrama je moguće automatsko generiranje SQL naredbi za stvaranje odgovarajućih tablica u bazi podataka.

Dokumentacija je pisana u jeziku **LaTeX**[14]. LaTeX je široko korišten jezik za pisanje dokumenata koji pruža široke mogućnosti i omogućuje jednostavniju izradu složenih dokumenata. S obzirom da se radi o markup jeziku, LaTeX omogućuje odvajanje stila od sadržaja što kod složenih dokumenata daje veliku prednost u odnosu na standardne aplikacije za pisanje dokumenata kao što su MS Word.

U svrhu pisanja dokumentacije u LaTeX-u korišten je **TeXstudio**[15]. TeXstudio je uređivač teksta posebno napravljen upravo za pisanje dokumenata u jeziku LaTeX.

Za upravljanje programskim kodom, dokumentacijom i ostalim datotekama koje su korištene u ovom projektu korišten je sustav **Git**[16]. Git je popularan distribuirani sustav za upravljanje verzijama podataka. Izuzetno je koristan kada više ljudi u timu radi na nekom projektu jer omogućuje koordinaciju istovremenog rada više ljudi na projektu. Sastoji se od udaljenog repozitorija, obično na nekoj

[11] https://www.visual-paradigm.com/
[12] https://www.drawio.com/
[13] https://erdplus.com/
[14] https://www.latex-project.org/
[15] https://www.texstudio.org/
[16] https://git-scm.com/

Git platformi u oblaku, te lokalnih kopija tog repozitorija na računalima korisnika koji sudjeluju u projektu. Git prati kompletnu povijest promjena u repozitoriju i nudi razne mogućnosti koje olakšavaju distribuirani rad.

Kao Git platforma u oblaku je za ovaj projekt korišten **GitHub**[17]. GitHub je danas najraširenija platforma za spremanje programskih projekata i distribuirani sustav za upravljanje verzijama.

Za puštanje aplikacije i popratne baze podataka u pogon korištena je usluga oblaka **Render**[18]. Render je sustav u oblaku koji služi za izgradnju i pokretanje raznih programskih sustava, pa tako i web aplikacija i baza podataka. Prednosti Rendera su njegovo lagano integriranje s GitHubom, automatsko deployanje prilikom promjena na grani u Git repozitoriju te postojanje besplatne usluge uz određena ograničenja koja za ovaj projekt nisu pretjerano bitna.

Prilikom puštanja web aplikaacije u pogon korištena je tehnologija **Docker**[19]. Docker omogućuje jednostavnu izradu kontejnera koji u sebi sadrže neku programsku podršku. Kontejner je oblik virtualizacije na razini operacijskog sustava. Ovim oblikom virtualizacije se stvara okruženje jednostavnije od onog dobivenog virtualnim strojem, ali još uvijek omogućuje da programska podrška u kontejneru bude nezavisna od okruženja na kojem je kontejner pokrenut. Docker kontejner se opisuje posebnom skriptom koja se zove Dockerfile.

Članovi tima su međusobno komunicirali preko društvenih mreža **Discord**[20] i **Messenger**[21]. Messenger je jednostavna društvena mreža koja kao glavni način komunikacije koristi izmjenu poruka između dva korisnika ili unutar grupe korisnika. Discord je društvena mreža u kojoj korisnici mogu komunicirati porukama, glasovnim pozivima i video pozivima. Korisnici mogu komunicirati unutar širih zajednica koje se nazivaju serveri. Unutar servera postoje tekstualni kanali kao i glasovni kanali za glasovne pozive sa više korisnika.

Za ispitivanje sustava u cjelini korišten je radni okvir **Selenium**[22]. Selenium je alat za automatizaciju web preglednika. Stoga dobro služi za testiranje web aplikacija kao sustava preko web preglednika. Skripte za testiranje se mogu pisati u raznim programskim jezicima.

[17] https://github.com/
[18] https://render.com/
[19] https://www.docker.com/
[20] https://discord.com/
[21] https://www.messenger.com/
[22] https://www.selenium.dev/

Za testiranje pojedinih komponenti sustava korišten je radni okvir **JUnit**[23]. JUnit je radni okvir koji služi za testiranje funkcionalnosti pojedinih dijelova programskog koda pisanog u programskom jeziku Java.

[23] https://junit.org/junit5/

5.2 Ispitivanje programskog rješenja

5.2.1 Ispitivanje komponenti

Ispitivanje komponenti vrši se radi provjeravanja ispravnosti pojedinih klasa i metoda sustava. Testovi se izvode tako što se ispitivanoj metodi šalje neki ulaz i provjerava se ponašanje metode na dobivenu ulaz sa očekivanim.

Za ispitivanje komponenti sustava koristili smo JUnit testove. Svi testovi su bili uspješni, kao što je prikazano na slici 5.1.

Slika 5.1: Rezultati testova komponenti

Prije svakog testa se u sustavu stvaraju dvije prijave u svrhu testiranja. Nakon svakog testa se iste prijave uklanjaju iz baze podataka.

```
Petar Belošević *
@BeforeEach
void setup() {
    saved1 = reportController.createReport(
            reportID: null,  reportHeadline: "problem", lat: 45.8000646,  lng: 15.978519, description: "nastao je problem",
            reportTS: null, userID: null, groupID: null,  categoryID: 1L, new StandardSession( manager: null), mockImagesInit(),  address: "Lisinski"
    );
    saved2 = reportController.createReport(
            reportID: null,  reportHeadline: "problem", lat: 45.8000646,  lng: 15.978519, description: "nastao je problem",
            reportTS: null, userID: null, groupID: null,  categoryID: 1L, new StandardSession( manager: null), mockImagesInit(),  address: "Lisinski"
    );
}

Filip +1
@AfterEach
void clear() {
    reportController.deleteReport(saved1.getReportID());
    reportController.deleteReport(saved2.getReportID());
}
```

Slika 5.2: Pomoćne metode u testovima komponenti

Test 1:

U ovom testu se prvo provjerava ako se prilikom stvaranja nove prijave njezin status automatski postavlja na ”neobrađen”. Zatim se provjerava ako se status prijave ispravno mijenja u ”uProcesu”.

```
Filip *
@Test
public void updateStatusTest(){
    assertEquals(reportController.getStatus(saved1.getReportID()).getKey().getStatus(), "neobrađen");
    reportController.changeStatus(saved1.getReportID(),  status: "uProcesu");
    assertEquals(reportController.getStatus(saved1.getReportID()).getKey().getStatus(), "uProcesu");
}
```

Slika 5.3: Test komponenti 1

Test 2:

U ovom testu se provjerava ako komponenta ispravno reagira kada se više puta pokušava promijeniti status prijave na isti status. U tom slučaju pripadajuća metoda treba baciti odgovarajuću iznimku.

```
Filip
@Test
public void updateStatusSameUpdate(){
    reportController.changeStatus(saved1.getReportID(),  status: "uProcesu");
    assertThrows(IllegalArgumentException.class,()->{ reportController.changeStatus(saved1.getReportID(),  status: "uProcesu");});
}
```

Slika 5.4: Test komponenti 2

Test 3:

U ovom testu se ispitiva reagiranje komponente na pokušaj promjene statusa nepostojećoj prijavi, to jest prijavi sa nepostojećim ID-om. Od pripadne metode se u tom slučaju očekuje bacanje iznimke.

```
Filip
@Test
public void updateStatusNonExistingReport(){
    assertThrows(IllegalArgumentException.class,()->{reportController.changeStatus( reportID: -1L,  status: "uProcesu");});
}
```

Slika 5.5: Test komponenti 3

Test 4:

U ovom testu se ispitiva reagiranje komponente na pokušaj promjene statusa prijavi bez davanja ID-a prijave. Metodi se dakle kao ID prijave predaje null referenca. Metoda u tom slučaju treba baciti iznimku.

```
Filip
@Test
public void updateStatusNullReportID(){
    assertThrows(IllegalArgumentException.class,()->{reportController.changeStatus( reportID: null,  status: "uProcesu");});
}
```

Slika 5.6: Test komponenti 4

Test 5:

U ovom testu se prvo provjerava ako inicijalno prijava nije nadovezana niti na jednu prijavu. Zatim se druga prijava nadovezuje na prvu prijavu predajom ID-eva prijava u metodu za nadovezivanje. na kraju se provjerava ispravnost provedenog nadovezivanja.

```
Filip *
@Test
public void groupReportsTest() {
    assertNull(reportController.getReport(saved2.getReportID()).getReport().getGroup());
    reportController.groupReports(saved1.getReportID(), List.of(new Long[]{saved2.getReportID()}));
    assertEquals(reportController.getReport(saved2.getReportID()).getReport().getGroup().getReportID(),saved1.getReportID());
}
```

Slika 5.7: Test komponenti 5

Test 6:

U ovom testu se provjerava ispravno ponašanje prilikom pokušaja nadovezivanja prijave uz davanje nevažećeg ID-a prijave na koju se nadovezuje. U tom slučaju pripadna metoda treba baciti iznimku.

```
Filip *
@Test
public void groupReportsLeaderDoesNotExists() {
    assertThrows(IllegalArgumentException.class, ()->{ reportController.groupReports( groupLeaderId: -1L, List.of(new Long[]{saved2.getReportID()}));});
}
```

Slika 5.8: Test komponenti 6

Test 7:

U ovom testu se provjerava ispravno ponašanje prilikom pokušaja nadovezivanja prijave bez davanja prijave na koju se nadovezuje. U tom slučaju pripadna metoda treba baciti iznimku.

```
Filip
@Test
public void groupReportsLeaderNullTest(){
    assertThrows(IllegalArgumentException.class, ()->{ reportController.groupReports( groupLeader: null, List.of(new Long[]{saved2.getReportID()}));});
}
```

Slika 5.9: Test komponenti 7

Test 8:

U ovom testu se provjerava ispravno ponašanje prilikom pokušaja nadovezivanja prijava bez davanja prijava koje se nadovezuju. Konkretnije, umjesto prijava koje se trebaju nadovezati predaje se null referenca. U tom slučaju pripadna metoda treba baciti iznimku.

```
Filip *
@Test
public void groupReportsGroupMembersNullTest() {
    assertThrows(IllegalArgumentException.class, ()->{ reportController.groupReports(saved1.getReportID(),  groupMembers: null);});
}
```

Slika 5.10: Test komponenti 8

Test 9:

U ovom testu se provjerava ispravno ponašanje prilikom pokušaja nadovezivanja prijava uz davanje praznog polja prijava koje se nadovezuju. U tom slučaju pripadna metoda treba baciti iznimku.

```
Filip *
@Test
public void groupReportsGroupMembersArrayEmptyTest() {
    assertThrows(IllegalArgumentException.class, ()->{ reportController.groupReports(saved1.getReportID(), List.of(new Long[]{}));});
}
```

Slika 5.11: Test komponenti 9

Test 10:

U ovom testu se pokušava napraviti nadovezivanje prijave na samu sebe. Takva radnja očekuje izazivanje iznimke.

```
Filip
@Test
public void groupReportGroupSameReportTest(){
    assertThrows(IllegalArgumentException.class, ()->{reportController.groupReports(saved1.getReportID(), List.of(new Long[]{saved1.getReportID()}));});
}
```

Slika 5.12: Test komponenti 10

Test 11:

U ovom testu se provjerava ispravnost promjene statusa povezanih prijava. Prvo se provjerava ako prijave imaju status "neobrađen". Zatim se druga prijava nadovezuje na prvu te se prvoj prijavi mijenja status u "uProcesu". Očekuje se da promjena nad glavnom prijavom rezultira istom promjenom i u svim nadovezanim prijavama. Stoga se na kraju provjerava ako obje prijave imaju promijenjen status u "uProcesu".

```
Filip *
@Test
public void updateGroupStatusTest(){
    assertEquals(reportController.getStatus(saved1.getReportID()).getKey().getStatus(),"neobrađen");
    assertEquals(reportController.getStatus(saved2.getReportID()).getKey().getStatus(),"neobrađen");
    reportController.groupReports(saved1.getReportID(), List.of(new Long[]{saved2.getReportID()}));
    reportController.changeStatus(saved1.getReportID(), status: "uProcesu");
    assertEquals(reportController.getStatus(saved1.getReportID()).getKey().getStatus(),"uProcesu");
    assertEquals(reportController.getStatus(saved2.getReportID()).getKey().getStatus(),"uProcesu");
}
```

Slika 5.13: Test komponenti 11

5.2.2 Ispitivanje sustava

Ispitivanje sustava kao cjeline odrađeno je pomoću Selenium WebDrivera za Chrome koji se koristio unutar JUnit testova. Svi testovi su uspješno prošli.

Slika 5.14: Rezultati ispitivanja sustava

Prije svakog testa se inicijalizira ChromeDriver u posebnoj metodi te se na kraju svakog testa inicijalizirani ChromeDriver gasi u posebnoj metodi.

```
Petar Belošević
@BeforeEach
public void initializeDriver() {
    System.setProperty("webdriver.chrome.driver", "C:\\Program Files\\ChromeDriver\\chromedriver.exe");
    driver = new ChromeDriver();
    driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));
    // driver.manage().timeouts().implicitlyWait(15, TimeUnit.SECONDS);
    driver.get("https://progi-projekt-test.onrender.com/");
}

Petar Belošević
@AfterEach
public void quitDriver() {
    driver.quit();
}
```

Slika 5.15: Pomoćne metode za ispitivanje sustava

Test 1:

U ovom testu se ispituje rad sustava tijekom pokušaja prijave korisnika koji postoji u bazi podataka. Simulira se odabir opcije prijave korisnika na korisničkom sučelju. U obrazac za prijavu se upisuju podaci ”abc@gmail.com” za e-mail i ”pass1234”

za lozinku te se klikne na gumb za prijavu. Očekuje se da prijava bude uspješna. To treba rezultirati preusmjeravanjem na glavnu stranicu te prikazom gumba sa korisnikovom e-mail adresom u zaglavlju stranice.

```
// Test 1
Petar Belošević *
@Test
public void testLoginUser() {
    String eMail = "abc@gmail.com", password = "pass1234";

    // log-in button
    driver.findElement(By.id("loginDropdown")).click();
    // login user
    driver.findElement(By.cssSelector("#root > div > header > div > div > div > div > div:nth-child(1) > ul > li:nth-child(1) > a")).click();
    // e-mail input
    driver.findElement(By.id("UserLogin")).sendKeys(eMail);
    // password input
    driver.findElement(By.id("passLogin")).sendKeys(password);
    // log-in button
    driver.findElement(By.id("loginButton")).click();

    Wait<WebDriver> wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    wait.until(d -> driver.findElement(By.cssSelector("#root > div > header > div > div > div > div > a > button")).isDisplayed());

    String redirectURL = driver.getCurrentUrl();
    assertEquals( expected: "https://progi-testing-v2.onrender.com/", redirectURL);
    // e-mail u buttonu
    assertEquals(eMail, driver.findElement(By.cssSelector("#root > div > header > div > div > div > div > a > button")) .getText());
}
```

Slika 5.16: Ispitivanje sustava: Test 1

Test 2:

Ovaj test je ispitivanje pokušaja prijave korisnika s neispravnom lozinkom. Radnje ovog testa su iste kao i u prethodnom testu. Razlika je što se na kraju ne očekuje preusmjeravanje na glavnu stranicu već prikaz prozora sa porukom ”Greška kod prijave: neispravan mail ili lozinka!”.

```
// Test 2
Petar Belošević *
@Test
public void testLoginUserInvalidPassword() {
    String eMail = "abc@gmail.com", password = "pass123456";

    // log-in button
    driver.findElement(By.id("loginDropdown")).click();
    // login user
    driver.findElement(By.cssSelector("#root > div > header > div > div > div > div > div:nth-child(1) > ul > li:nth-child(1) > a")).click();
    // e-mail input
    driver.findElement(By.id("UserLogin")).sendKeys(eMail);
    // password input
    driver.findElement(By.id("passLogin")).sendKeys(password);
    // log-in button
    driver.findElement(By.id("loginButton")).click();

    Wait<WebDriver> wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    wait.until(d -> driver.findElement(By.cssSelector("body > div:nth-child(3) > p")).isDisplayed());

    assertEquals( expected: "Greška kod prijave: neispravan mail ili lozinka!", driver.findElement(By.cssSelector("body > div:nth-child(3) > p")).getText());
}
```

Slika 5.17: Ispitivanje sustava: Test 2

Test 3:

U ovom se testu ispituje pokušaj registracije novog korisnika. Simulira se odabir opcije registracije novog korisnika. Na novoj se stranici u obrazac upisuju podaci "Ime" za ime, "Prezime" za prezime, "novi.user.1@gmail.com" za e-mail i "lozinka123" za lozinku. Na kraju se registracija dovršava pritiskom gumba za registraciju. Očekuje se uspješna registracija što rezultira preusmjeravanjem na početnu stranicu te prikazom gumba sa korisnikovom e-mail adresom u zaglavlju stranice.

```
// Test 3
Petar Beloševič *
@Test
public void registerUser() {
    String name = "Ime", surname = "Prezime", eMail = "novi.user.1@gmail.com", password = "lozinka123";

    // register button
    driver.findElement(By.cssSelector("#registerDropdown")).click();
    // register user
    driver.findElement(By.cssSelector("#root > div > header > div > div > div > div > div:nth-child(2) > ul > li:nth-child(1) > a")).click();
    // name input
    driver.findElement(By.cssSelector("#imeReg")).sendKeys(name);
    // surname input
    driver.findElement(By.cssSelector("#prezReg")).sendKeys(surname);
    // e-mail button
    driver.findElement(By.cssSelector("#mailReg")).sendKeys(eMail);
    // password input
    driver.findElement(By.cssSelector("#passReg")).sendKeys(password);
    // register button
    driver.findElement(By.cssSelector("#root > div > form > button")).click();

    Wait<WebDriver> wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    wait.until(d -> driver.findElement(By.cssSelector("#root > div > header > div > div > div > div > a > button")).isDisplayed());

    String redirectURL = driver.getCurrentUrl();
    assertEquals( expected: "https://progi-testing-v2.onrender.com/", redirectURL);
    // e-mail u buttonu
    assertEquals(eMail, driver.findElement(By.cssSelector("#root > div > header > div > div > div > div > a > button")) .getText());
}
```

Slika 5.18: Ispitivanje sustava: Test 3

Test 4:

U ovom testu se ispituje sustav prilikom pokušaja registracije korisnika koji već postoji u sustavu, to jest sa već iskorištenom e-mail adresom. Ponavljaju se radnje iz prošlog testa, ali se u obrazac za registraciju unose drugi podaci. Za ovaj test se unose podaci "xx" za ime, "yy" za prezime, "abc@gmail.com" za e-mail i "pass1234" za lozinku. S obzirom da korisnik s danom e-mail adresom već postoji u sustavu. Aplikacija treba prikazati pripadnu poruku te nakon klika na gumb uz poruku treba ostati na stranici za registraciju.

```
// Test 4
Petar Belošević *
@Test
public void registerExistingUser() {
    String name = "xx", surname = "yy", eMail = "abc@gmail.com", password = "pass1234";

    // register button
    driver.findElement(By.cssSelector("#registerDropdown")).click();
    // register user
    driver.findElement(By.cssSelector("#root > div > header > div > div > div > div > div:nth-child(2) > ul > li:nth-child(1) > a")).click();
    // name input
    driver.findElement(By.cssSelector("#imeReg")).sendKeys(name);
    // surname input
    driver.findElement(By.cssSelector("#prezReg")).sendKeys(surname);
    // e-mail button
    driver.findElement(By.cssSelector("#mailReg")).sendKeys(eMail);
    // password input
    driver.findElement(By.cssSelector("#passReg")).sendKeys(password);
    // register button
    driver.findElement(By.cssSelector("#root > div > form > button")).click();

    Wait<WebDriver> wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    wait.until(d -> driver.findElement(By.cssSelector("body > div:nth-child(3) > button")).isDisplayed());

    driver.findElement(By.cssSelector("body > div:nth-child(3) > button")).click();
    assertEquals( expected: "https://progi-testing-v2.onrender.com/user-register", driver.getCurrentUrl());
}
```

Slika 5.19: Ispitivanje sustava: Test 4

Test 5:

U ovom testu se pokušava izbrisati korisnika iz sustava. Simulira se prijava korisnika u sustav sa podacima "novi.user.1@gmail.com" za e-mail i "lozinka123" (korisnik koji se registrirao u ranijem testu) kao što je opisano u testu 1. Zatim se klikom na gumb sa korisnikovim e-mailom i odabirom profilne stranice odlazi na stranicu korisnikovog profila. Tamo se simulira klik na gumb za brisanje računa nakon čega se prikazuje prozor sa porukom "Korisnički račun uspješno izbrisan!" i pripadnim gumbom. Pritiskom na gumb se korisnika treba preusmjeriti na glavnu stranicu.

```
// Test 5
Petar Belošević *
@Test
public void deleteUser() {
    String eMail = "novi.user.1@gmail.com", password = "lozinka123";
    // log-in button
    driver.findElement(By.id("loginDropdown")).click();
    // login user
    driver.findElement(By.cssSelector("#root > div > header > div > div > div > div > div:nth-child(1) > ul > li:nth-child(1) > a")).click();
    // e-mail input
    driver.findElement(By.id("UserLogin")).sendKeys(eMail);
    // password input
    driver.findElement(By.id("passLogin")).sendKeys(password);
    // log-in button
    driver.findElement(By.id("loginButton")).click();

    Wait<WebDriver> wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    wait.until(d -> driver.findElement(By.cssSelector("#root > div > header > div > div > div > div > a > button")).isDisplayed());
    // go to profile page
    driver.findElement(By.cssSelector("#root > div > header > div > div > div > div > a > button")).click();
    driver.findElement(By.cssSelector("#root > div > header > div > div > div > div > ul > li:nth-child(1) > a")).click();
    // delete account
    driver.findElement(By.cssSelector("#root > div > form > button:nth-child(4)")).click();

    wait.until(d -> driver.findElement(By.cssSelector("body > div:nth-child(3) > p")).isDisplayed());
    assertEquals( expected: "Korisnički račun uspješno izbrisan!", driver.findElement(By.cssSelector("body > div:nth-child(3) > p")).getText());
    driver.findElement(By.cssSelector("body > div:nth-child(3) > button")).click();

    wait.until(d -> driver.findElement(By.cssSelector("#root > div > div:nth-child(2) > h1")).isDisplayed());
    assertEquals( expected: "https://progi-testing-v2.onrender.com/", driver.getCurrentUrl());
}
```

Slika 5.20: Ispitivanje sustava: Test 5

Test 6:

U ovom testu se simulira pregledavanje statistike prijava. Klikom na gumb ”Statistika” u zaglavlju stranice korisnika se preusmjerava na stranicu za statistiku. Na prikazanoj stranici se djelomično popunjava obrazac za prikaz statistike. Za kategoriju se odabire prva kategorija koja počinje na ”r”, za datum početka se odabire 1.1.2024., a za datum završetka intervala se odabire 2.2.2024. Tako popunjen obrazac se šalje na poslužitelja. Nakon toga se ispod filtra očekuje prikaz statističkih podataka o prijavama odabranima poslanim filtrom.

```
// Test 6
Petar Belošević
@Test
public void statisticsReview() {
    driver.findElement(By.cssSelector("#root > div > header > div > div > ul > li:nth-child(3) > a > button")).click();

    Wait<WebDriver> wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    wait.until(d -> driver.findElement(By.cssSelector("#root > div > h1")).isDisplayed());

    assertEquals( expected: "https://progi-testing-v2.onrender.com/statistika", driver.getCurrentUrl());

    driver.findElement(By.cssSelector("#root > div > form > label:nth-child(1) > select")).sendKeys( ...keysToSend: "r");
    driver.findElement(By.cssSelector("#root > div > form > label:nth-child(2) > input[type=date]")).sendKeys( ...keysToSend: "01012024");
    driver.findElement(By.cssSelector("#root > div > form > label:nth-child(3) > input[type=date]")).sendKeys( ...keysToSend: "02022024");
    driver.findElement(By.cssSelector("#root > div > form > button")).click();

    assertTrue(driver.findElement(By.cssSelector("#root > div > ul > li > p:nth-child(1)")).isDisplayed());
}
```

Slika 5.21: Ispitivanje sustava: Test 6

5.3 Dijagram razmještaja

Dijagram razmještaja je statički strukturni dijagram koji opisuje topologiju sustava i usredotočen je na odnos sklopovskih i programskih dijelova. Osnovni elementi dijagrama su čvorovi, artefakti i spojevi.

Dijagram 5.22 prikazuje topologiju sustava za prijavljivanje oštećenja na javnim površinama. Sustav je napravljen na temelju arhitekture klijent-poslužitelj u kojoj klijent i poslužitelj komuniciraju HTTPS protokolom. Sustav je stavljen u pogon pomoću dva odvojena servisa na usluzi Render, jedan za aplikaciju i jedan za bazu podataka. Na Render servisu za bazu podataka postoji instanca PostgreSQL baze podataka te taj servis prima zahtjeve za bazu podataka na standardnom portu 5432. Na Render servisu za web aplikaciju je pokrenut Docker kontejner unutar kojeg su upogonjeni React frontend i Spring Boot backend dijelovi aplikacije. Spring Boot backend po potrebi komunicira sa bazom podataka slanjem SQL querryja na port 5432 Render servisa za bazu podataka. Render servis za web aplikaciju sluša HTTPS zahtjeve na portu 443. Zaprimljene HTTPS zahtjeve Render servis prosljeđuje na port 8080 Docker kontejnera kao HTTP zahtjeve. Klijentski uređaj na sebi ima pokrenuti web preglednik pomoću kojeg šalje HTTPS zahtjeve na port 443 Render servisa za web aplikaciju.

Slika 5.22: Dijagram razmještaja

5.4 Upute za puštanje u pogon

Ovaj sustav je pušten u pogon pomoću online servisa Render pa se u uputama za puštanje u pogon pretpostavlja korištenje istog. Za tu svrhu potrebno je napraviti korisnički račun na Renderu.

5.4.1 Konfiguracija poslužitelja baze podataka

Potrebno je odabrati opciju stvaranja nove PostgreSQL baze podataka.

Slika 5.23: Odabir stvaranja nove baze podataka

Zatim je potrebno konfigurirati bazu.

Slika 5.24: Konfiguriranje baze podataka

Za ime instance baze podataka, ime same baze i ime korisnika se mogu unijeti proizvoljna imena. Za regiju je poželjno odabrati geografski najbližu regiju, s obzirom da će to rezultirati najmanjom latencijom. U našem slučaju najbliža regija bila je *Frankfurt (EU Central)*. Ostale parametre nije potrebno mijenjati.

Slika 5.25: Odabir vrste instance baze podataka

Sljedeće je potrebno odabrati vrstu instance baze podataka, odnosno cjenovni rang baze. Ovo je isključivo pitanje preferencije osobe koja pušta sustav u pogon, odnosno koliko je spremna platiti za puštanje baze podataka u pogon. Za naše potrebe bilo je dovoljna besplatna opcija te smo istu i odabrali.

Nakon toga treba kliknuti na *Create Database* i time će baza biti puštena u pogon.

5.4.2 Konfiguracija poslužitelja web aplikacije

Prije puštanja aplikacije u pogon potrebno je u Git repozitoriju ažurirati podatke za spajanje s bazom podataka. Za to je potrebno izmijeniti vrijednosti varijabli *spring.datasource.url*, *spring.datasource.username* i *spring.datasource.password* u datoteci *IzvorniKod/server/src/main/resources/application.properties* u *main* grani repozitorija.

```
1
2   server.tomcat.accesslog.enabled=true
3
4   spring.jpa.hibernate.ddl-auto=update
5   spring.jpa.show-sql=true
6
7   spring.datasource.url=jdbc:postgresql://dpg-cm1dhamn7f5s73e7gmv0-a.frankfurt-postgres.render.com/progi_ostecenja
8   spring.datasource.username=progi
9   spring.datasource.password=[redacted]
10
11  spring.session.store-type=jdbc
12  spring.session.timeout=30m
13  spring.mvc.pathmatch.matching-strategy=ant_path_matcher
```

Slika 5.26: Datoteke s podacima za spajanje s bazom podataka

Na web stranici Rendera potrebno je odabrati karticu *Dashboard* i zatim odabrati ranije upogonjenu bazu podataka. Na novootvorenoj stranici potrebno je pronaći odjeljak *Connections* gdje se nalaze potrebni parametri. vrijednosti *Username* i *Password* potrebno je kopirati u pripadajuće varijable u datoteci na Git repozitoriju. U varijabli *spring.datasource.url* potrebno je zamijeniti tekst koji slijedi nakon *jdbc:postgresql://*. Taj tekst potrebno je zamijeniti dijelom varijable *External Database URL* na Renderu koji započinje sa "dpg-", a prethodi mu znak '@'. Dakle, vrijednost varijable *spring.datasource.url* mora započeti sa "jdbc:postgresql://dpg-...".

Slika 5.27: Datoteke s podacima za spajanje s bazom podataka

Zatim je na Renderu potrebno odabrati opciju stvaranja novog web servisa.

Slika 5.28: Odabir stvaranja novog web servisa

Sljedeće je potrebno odabrati opciju izgradnje i upogonjavanja iz Git repozito-

rija.

Slika 5.29: Odabir načina izgradnje

Nakon toga potrebno je odabrati pripadni repozitorij ili unijeti URL repozitorija. Zatim slijedi konfiguracija web servisa.

Slika 5.30: Konfiguriranje web servisa

Za *Name* je potrebno unijeti proizvoljan naziv web servisa. Uneseno ime će biti dio URL-a samog web servisa preko kojeg će mu se moći pristupiti. Za regiju je ponovno poželjno odabrati geografski najbližu regiju što je u našem slučaju *Frankfurt (EU Central)*. *Branch* koji se koristi za stvaranje web servisa je potrebno ostaviti na *main*. *Root Directory* je potrebno ostaviti prazno. Za *Runtime* je potrebno odabrati *Docker*.

Slika 5.31: Odabir vrste instance web servisa

Sljedeće je potrebno odabrati vrstu instance web servisa, odnosno njegov cjenovni rang. Ovo je isključivo pitanje preferencije osobe koja pušta sustav u pogon, odnosno koliko je spremna platiti za puštanje web servisa u pogon. Za naše potrebe bilo je dovoljna besplatna opcija te smo istu i odabrali.

Nakon treba kliknuti na *Create Web Service* i time će web servis biti puštena u pogon. Prilikom prvog pokretanja web servisa baza podataka će se napuniti inicijalnim podacima.

U bazi će biti sljedeći podaci:

- Korisnici:
  - e-mail: abc@gmail.com, ime: xx, prezime: yy, lozinka: pass1234
  - e-mail: abc2@gmail.com, ime: xx, prezime: yy, lozinka: pass1234
  - e-mail: abc3@gmail.com, ime: xx, prezime: yy, lozinka: pass1234
  - e-mail: abc4@gmail.com, ime: xx, prezime: yy, lozinka: pass1234
- Gradski uredi:
  - ime: Rupasti, e-mail: rupasti@gmail.com, lozinka: password

  - ime: Rasvjeta, e-mail: rasvjeta@gmail.com, lozinka: password
  - ime: Smece, e-mail: smece@gmail.com, lozinka: password
  - ime: Korupcija, e-mail: korupcija@gmail.com, lozinka: password
- Kategorije:
  - kategorija: rupa na cesti, ured: Rupasti
  - kategorija: rupa na pjesackome, ured: Rupasti
  - kategorija: ulicna rasvjeta, ured: Rasvjeta
  - kategorija: smece na ulici, ured: Smece
  - kategorija: smece u parku, ured: Smece
  - kategorija: institucionalna korupcija, ured: Korupcija
- Ključne riječi za kategorije:
  - riječ: Rupa, kategorija: rupa na cesti
  - riječ: Cesta, kategorija: rupa na cesti
  - riječ: Pjesacki, kategorija: rupa na pjesackome
  - riječ: Pješački, kategorija: rupa na pjesackome
  - riječ: Rasvjeta, kategorija: ulicna rasvjeta
  - riječ: Lampa, kategorija: ulicna rasvjeta
  - riječ: Smeće, kategorija: smece na ulici
  - riječ: Smrdi, kategorija: smece na ulici
  - riječ: Park, kategorija: smece u parku
  - riječ: Otpad, kategorija: smece u parku
  - riječ: Mito, kategorija: institucionalna korupcija
  - riječ: Korupcija, kategorija: institucionalna korupcija
- jedna prijava rupe na cesti na lokaciju kod koncertne dvorane Vatroslava Lisinskog sa priloženom slikom

6. Zaključak i budući rad

Projektni zadatak našeg tima bio je razvoj web aplikacije koja omogućuje građanima da prijavljuju razna oštećenja gradskim uredima. S druge strane aplikacija omogućuje djelatnicima gradskih ureda da upravljaju s pristiglim prijavama.

Nakon gotovo 4 mjeseca predanog i kontinuiranog rada možemo reći da smo ostvarili ciljeve projekta te putem stekli vrijedna iskustva i usvojili brojne vještine. Proces razvoja projekta je bio logički podijeljen u dvije faze.

Prva faza je započela formiranjem tima i upoznavanjem sa projektnim zadatkom. Nakon toga je slijedio period u kojem smo raspravljali o zadatku, razmjenjivali ideje o oblikovanju programske potpore i izrađivali skice za osnovni dizajn korisničkog sučelja. Iz svega toga smo izlučili, popisali i dokumentirali funkcijske i nefunkcijske zahtjeve. Na temelju funkcijskih zahtjeva izrađeni su obrasci uporabe te dijagrami obrasca uporabe i sekvencijski dijagrami. Također je raspravljen i razrađen dizajn baze podataka sa popratnim relacijskim dijagramom. Paralelno s tim članovi tima su podijeljeni na tim za razvoj frontend dijela aplikacije i tim za razvoj backend dijela aplikacije. Za oba dijela aplikacije su odabrane tehnologije pomoću kojih će se ti dijelovi razvijati. S obzirom da su odabrane tehnologije bile nove za članove tima, u ovoj fazi je dio vremena trebao biti odvojen za upoznavanje i usvajanje tih tehnologija. Krajem prve faze definirana je arhitektura sustava na temelju odabranih tehnologija te je na temelju toga napravljen dijagram razreda. Članovi backend i frontend tima su započeli implementirati sustav na temelju dogovorenih zahtjeva. Na kraju prve faze realizirane su osnovne funkcionalnosti sustava, kao što su prijava, registracija i osnovni obrazac za slanje prijava, i napravljen osnovni dizajn korisničkog sučelja.

U drugoj fazi je glavni fokus bio na ostvarivanju svih funkcionalnosti koje smo definirali i dokumentirali u prvoj fazi projekta. Paralelno s time u dokumentaciji je bilo potrebno izraditi nekoliko UML dijagrama koji opisuju sustav iz više perspektiva, popisati sve korištene tehnologije i alate tijekom trajanja projekta te napisati upute za puštanje aplikacije u pogon. Na kraju je odrađeno i dokumentirano ispitivanje komponenti i sustava u cjelini.

Prva faza projekta je sadržavala više aktivnosti te je tekla dosta sporije. Glavni

razlog tome je bilo potrebno vrijeme da se članovi tima uhodaju i upoznaju s projektom. Također je značajan faktor bio što nitko od članova tima nije imao iskustva sa timskim radom na projektu ovakvog tipa niti iskustva sa glavnim tehnologijama koje su se koristile tijekom projekta. Stoga je bilo potrebno vrijeme da se te prepreke savladaju.

Druga faza projekta nije sadržavala toliko aktivnosti, ali je bila izuzetno izazovna jer je bilo potrebno ostvariti veći dio funkcionalnosti u relativno kratkom vremenu.

Tim je imao redovite sastanke gotovo svaki tjedan na kojima smo raspravljali o onome što smo napravili u proteklom tjednu te raspravljali na čemu bi trebali raditi dalje. Ovime se maksimalno poticalo da svi članovi tima doprinose projektu na tjednoj bazi, nastojala se poboljšati komunikacija između članova tima te ažurnost svakog člana. Kroz projekt se brzo moglo vidjeti da su jedni od bitnih elemenata dobro odrađenog projekta dobra organizacija i koordinacija tima, redovit i kontinuiran rad, otvorena komunikacija te fleksibilnost članova tima da reagiraju na probleme koji su se javljali tijekom projekta. Jedna od glavnih prepreka je bila nedostatak prijašnjeg iskustva svih članova tima na ovakvim projektima. Također, nepoznavanje tehnologija koje smo kao tim koristili kroz projekt se također pokazalo kao jedna od većih prepreka. To je dosta usporavalo razvoj projekta zbog duljeg vremena potrebnog za uhodavanje i upoznavanje tehnologija i timskog načina rada, pogotovo u prvoj fazi projekta.

Na kraju projekta možemo reći da smo zadovoljni postignutim i naučenim. Sama aplikacija ima puno prostora za unaprjeđenje, ali kroz ovaj projekt smo dobili prijeko potrebno iskustvo timskog rada na projektnim zadacima. Također smo usvojili nova znanja i savladali korištenje novih tehnologija što će nam biti izuzetno korisno u budućem radu.

Popis literature

1. GeeksForGeeks, Spring Boot Architecture, https://www.geeksforgeeks.org/spring-boot-architecture/
2. Programsko inženjerstvo, FER ZEMRIS, `http://www.fer.hr/predmet/proinz`
3. Visual Paradigm, https://www.visual-paradigm.com/
4. draw.io, https://app.diagrams.net/
5. The Unified Modeling Language, `https://www.uml-diagrams.org/`
6. Baeldung, Spring Boot H2 database, https://www.baeldung.com/spring-boot-h2-database
7. Guide to Spring Email, https://www.baeldung.com/spring-email
8. File uploading with Spring Boot & Firebase Cloud Storage., https://medium.com/@poojithairo uploading-with-spring-boot-firebase-cloud-storage-e5ef2fbf942d
9. Efficient Data Transfer in REST APIs: A Deep Dive into the DTO Pattern with Spring Boot and MySQL, https://blog.stackademic.com/efficient-data-transfer-in-rest-apis-a-deep-dive-into-the-dto-pattern-with-spring-boot-and-mysql-df2bdf1ece74

Indeks slika i dijagrama

2.1 Aplikacija *Gradsko oko* Grada Rijeke 9

3.1 Dijagram obrasca uporabe, mogućnosti korisnika 22
3.2 Dijagram obrasca uporabe, obrada prijava 23
3.3 Sekvencijski dijagram, prijava/registracija 25
3.4 Sekvencijski dijagram, podnošenje prijave 27
3.5 Sekvencijski dijagram, obrada prijava 29

4.1 Prikaz Spring Boot arhitekture 32
4.2 Relacijski dijagram baze podataka 37
4.3 Kontroleri na backendu 38
4.4 Data Access Objects 39
4.5 Modeli 40
4.6 Servisi 41
4.7 Dijagram stanja 43
4.8 Dijagram aktivnosti 45
4.9 Dijagram komponenti 47

5.1 Rezultati testova komponenti 53
5.2 Pomoćne metode u testovima komponenti 54
5.3 Test komponenti 1 54
5.4 Test komponenti 2 54
5.5 Test komponenti 3 55
5.6 Test komponenti 4 55
5.7 Test komponenti 5 55
5.8 Test komponenti 6 56
5.9 Test komponenti 7 56
5.10 Test komponenti 8 56
5.11 Test komponenti 9 56
5.12 Test komponenti 10 57
5.13 Test komponenti 11 57

5.14 Rezultati ispitivanja sustava 58
5.15 Pomoćne metode za ispitivanje sustava 58
5.16 Ispitivanje sustava: Test 1 59
5.17 Ispitivanje sustava: Test 2 59
5.18 Ispitivanje sustava: Test 3 60
5.19 Ispitivanje sustava: Test 4 61
5.20 Ispitivanje sustava: Test 5 62
5.21 Ispitivanje sustava: Test 6 63
5.22 Dijagram razmještaja 65
5.23 Odabir stvaranja nove baze podataka 66
5.24 Konfiguriranje baze podataka 67
5.25 Odabir vrste instance baze podataka 67
5.26 Datoteke s podacima za spajanje s bazom podataka 68
5.27 Datoteke s podacima za spajanje s bazom podataka 69
5.28 Odabir stvaranja novog web servisa 69
5.29 Odabir načina izgradnje 70
5.30 Konfiguriranje web servisa 70
5.31 Odabir vrste instance web servisa 71

Dodatak: Prikaz aktivnosti grupe

Dnevnik sastajanja

1. sastanak
   - Datum: 23. listopada 2023.
   - Prisustvovali: Petar Belošević, Vinko Brkić, Tomislav Grudić, Fran Meglić, Eno Peršić, Bruno Mikulan, Filip Vučenik
   - Teme sastanka:
     - napravljen git repozitorij
     - rasprava o zadatku - definirani ključni funkcijski i nefunkcijski zahtjevi, razmjena ideja oko aplikacije
     - podjela poslova (pisanje opisa zadatka, izrada UML dijagrama, zapisivanje zahtjeva, raspodjela timova)
2. sastanak
   - Datum: 30. listopada 2023.
   - Prisustvovali: Petar Belošević, Vinko Brkić, Tomislav Grudić, Fran Meglić, Eno Peršić, Bruno Mikulan, Filip Vučenik
   - Teme sastanka:
     - dizajn baze podataka
     - dizajn korisničkog sučelja aplikacije
     - detaljnija rasprava o svojstvima aplikacije
3. sastanak
   - Datum: 6. studenoga 2023.
   - Prisustvovali: Petar Belošević, Vinko Brkić, Tomislav Grudić, Fran Meglić, Eno Peršić, Bruno Mikulan, Filip Vučenik
   - Teme sastanka:
     - izvještaj svakog dijela tima o svojem napretku
     - rad na dizajnu aplikacije
4. sastanak
   - Datum: 13. studenoga 2023.

- Prisustvovali: Petar Belošević, Vinko Brkić, Tomislav Grudić, Fran Meglić, Eno Peršić, Bruno Mikulan, Filip Vučenik
- Teme sastanka:
  - izvještaj svakog dijela tima o svojem napretku
  - korekcije na pojedinim dijelovima aplikacije
  - rad na spajanju frontend i backend dijelova aplikacije

5. sastanak
   - Datum: 15. studenoga 2023.
   - Prisustvovali: Petar Belošević, Vinko Brkić, Tomislav Grudić, Fran Meglić, Eno Peršić, Bruno Mikulan, Filip Vučenik
   - Teme sastanka:
     - rad na spajanju frontend i backend dijelova aplikacije
6. sastanak
   - Datum: 16. studenoga 2023.
   - Prisustvovali: Petar Belošević, Vinko Brkić, Tomislav Grudić, Fran Meglić, Eno Peršić, Bruno Mikulan, Filip Vučenik
   - Teme sastanka:
     - rad na deploymentu aplikacije
     - definiranje završnih zadataka prije prve predaje
7. sastanak
   - Datum: 11. prosinca 2023.
   - Prisustvovali: Petar Belošević, Vinko Brkić, Tomislav Grudić, Fran Meglić, Eno Peršić, Bruno Mikulan, Filip Vučenik
   - Teme sastanka:
     - kratka evaulacija dosadašnjeg rada, dogovor o raspodijeli bodova
     - pregled nedovršenog posla, daljnjih obaveza i zadataka kod dokumentacije i same aplikacije
8. sastanak
   - Datum: 18. prosinca 2023.
   - Prisustvovali: Petar Belošević, Vinko Brkić, Tomislav Grudić, Fran Meglić, Eno Peršić, Bruno Mikulan, Filip Vučenik
   - Teme sastanka:
     - redovna kontrola napravljenog posla
     - pregled nedovršenog posla
     - rasprava o implementaciji višejezičnosti

9. sastanak
    - Datum: 8. siječnja 2024.
    - Prisustvovali: Petar Belošević, Vinko Brkić, Tomislav Grudić, Fran Meglić, Eno Peršić, Bruno Mikulan, Filip Vučenik
    - Teme sastanka:
        - redovna kontrola napravljenog posla
        - pregled nedovršenog posla
        - rasprava oko nekoliko endpointova i povezivanja frontend i backend dijelova aplikacije
        - dogovor oko promjene načina spremanja lokacije oštećenja u bazi podataka
10. sastanak
    - Datum: 11. siječnja 2024.
    - Prisustvovali: Petar Belošević, Vinko Brkić, Tomislav Grudić, Fran Meglić, Eno Peršić, Bruno Mikulan, Filip Vučenik
    - Teme sastanka:
        - kratka rasprava oko promjene načina nadovezivanja prijava
        - razrješavanje problema vezanih uz povezivanje frontend i backend dijelova aplikacije

Tablica aktivnosti

	Petar Belošević	Vinko Brkić	Tomislav Grudić	Fran Meglić	Bruno Mikulan	Eno Peršić	Filip Vučenik
Upravljanje projektom	8h						
Opis projektnog zadatka	4h						1h
Funkcionalni zahtjevi	2h		1h				
Opis pojedinih obrazaca	3h						
Dijagram obrazaca	3h				2h		
Sekvencijski dijagrami	4h	2h		1h			
Opis ostalih zahtjeva	1h					1h	
Arhitektura i dizajn sustava	3h						
Opis baze podataka	2.5h		2h				
Dijagram razreda	1h		2h				
Dijagram stanja	3h						
Dijagram aktivnosti	3.5h						
Dijagram komponenti	3h						
Korištene tehnologije i alati	5h	2h			2h		
Ispitivanje programskog rješenja	4.5h						
Dijagram razmještaja	1h						
Upute za puštanje u pogon	2.5h						0.5h
Dnevnik sastajanja	1h						
Zaključak i budući rad	3h						
Popis literature	0.5h						

Nastavljeno na idućoj stranici

Nastavljeno od prethodne stranice

	Petar Beloševič	**Vinko Brkić**	**Tomislav Grudić**	**Fran Meglić**	**Bruno Mikulan**	**Eno Peršić**	**Filip Vučenik**
Povezivanje frontenda i backenda					3h	2h	2h
Puštanje u pogon							3h
rad na frontendu	2h	14h			10h	14h	
rad na backendu	1h		4h	10h			10h
izrada baze podataka			0.5h	1h			0.5h
spajanje s bazom podataka							
usvajanje korištenih tehnologija	4h	6h	5h	8h	3h	6h	10h

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.