1.1. Uvod

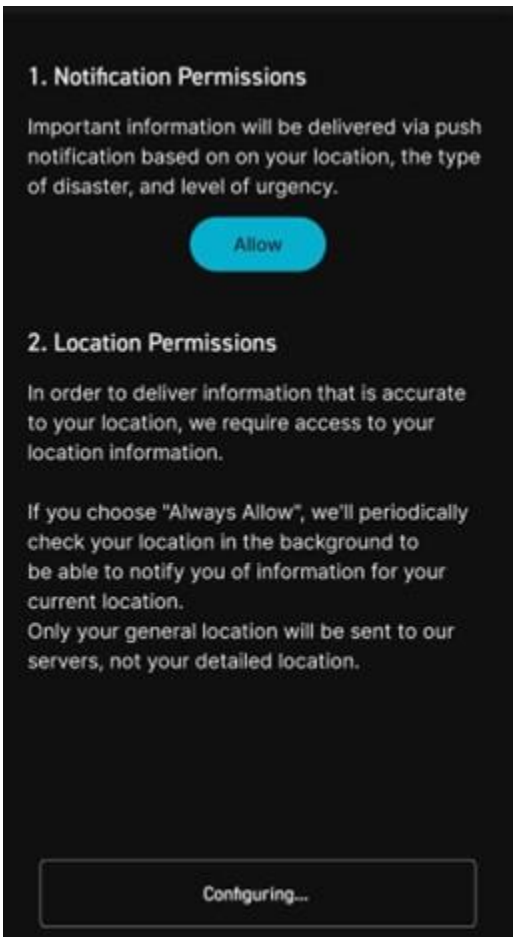
Cilj ovo projekta je razvoj web aplikacije koja će korisnicima omogućiti pojednostavljeni pregled nesreća i nepogoda kao i poboljšanu koordinaciju i komunikaciju između građana, vlasti i humanitarnih organizacija u kriznim situacijama.

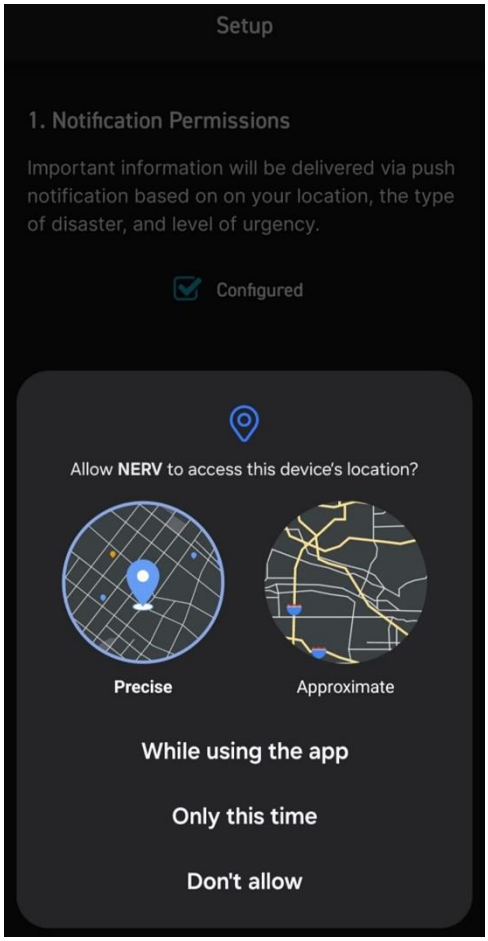
1.2. Motivacija i korisnički zahtjevi

Glavna motivacija bila nam je optimizirati i olakšati reakciju hitnih službi u slučaju nesreća i vremenskih nepogoda kojih je danas zbog klimatskih promjena sve više. Aplikacija bi korisnicima dala pregled opasnosti i omogućila pravovremeno planiranje kao i mogućnost prijave nesreće. S druge strane, ustanovama koje su zadužene za pružanje pomoći u tim situacijama bio bi omogućen brzi pristup velikom broju korisnika u svrhe pružanja ključnih informacija kao što su lokacije skloništa i usluge kod tih skloništa u slučaju požara, poplava, potresa, itd. Bilo je bitno omogućiti prijavu nesreća i anonimnim korisnicima kako bi svi mogli brzo reagirati i prijaviti opasnost. Korisnici se potiču napraviti profil kako bi dobili pristup dodatnim funkcionalnostima kao SMS/E-mail obavijesti.

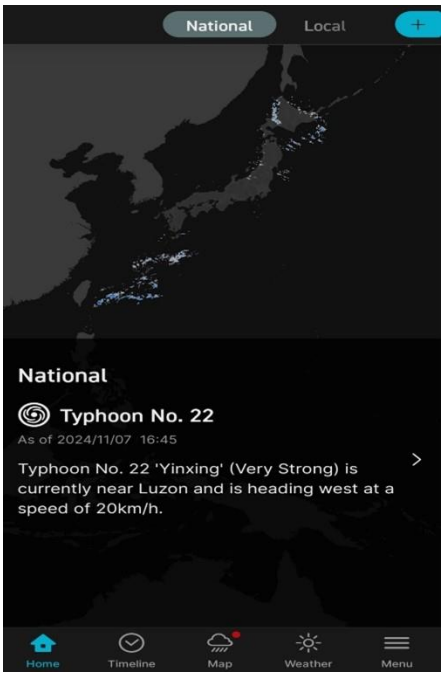
1.3. Slična rješenja

Odličan primjer sličnih rješenja je aplikacija „NERV Disaster Prevention“ koja čovjeku omogućava pregled prirodnih katastrofa u svijetu.

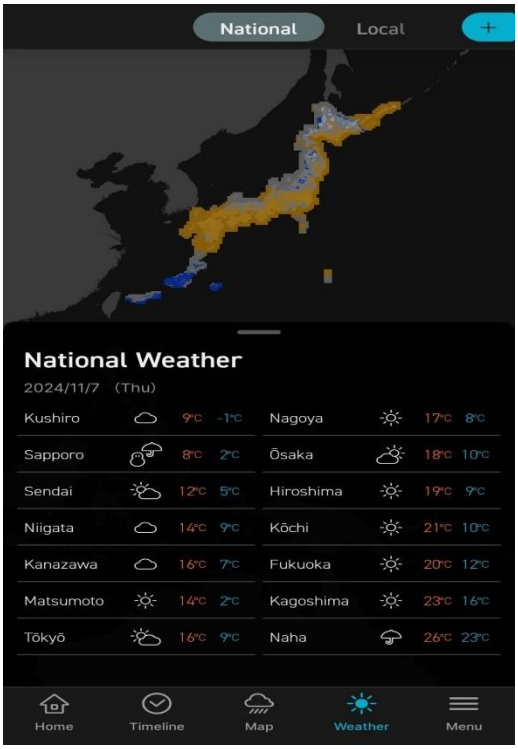




NERV Disaster Prevention ne omogućava stvaranje korisničkog računa, ali za samo korištenje aplikacije nužno je omogućiti notifikacije i praćenje lokacije kako bi aplikacija mogla obavijestiti korisnika.



Pri otvaranju aplikacije otvara se mapa koja omogućava pregled na nacionalnoj i lokalnoj razini. Na dnu se nalaze obavijesti o nadolazećim nepogodama. Moguće je pregledati i zapisnik kojim se mogu lakše tumačiti podaci sa mape.



Omogućen je i jednostavan pregled vremenske prognoze.

Razlika između ovog i našeg rješenja je da se naša aplikacija više fokusira na lokalne nesreće kao i širi skup nesreća od prirodnih katastrofa. Također, jedna od glavnih

funkcionalnosti naše aplikacije je suradnja sa hitnim službama i dobrotvornim organizacijama kako bi se korisnicima omogućio dolazak do prave pomoći, umjesto isključivo slanja notifikacije kao što je slučaj kod „NERV Disaster Prevention“ aplikacije. Nadalje, naša aplikacija omogućava samim korisnicima da prijave nesreću dok se NERV oslanja isključivo na službene ustanove.

1.4. Opseg projekta

Za rad na projektu zaduženo je 7 studenata FER-a. Projekt se radi u edukativne svrhe provjere znanja u sklopu predmeta „Programsko inženjerstvo“. Vremensko ograničenje potpune izrade projekta je 14 tjedana, tj. 7 tjedana za izradu prve verzije web aplikacije koja uključuje funkcionalan login i interaktivnu mapu nesreća te još 7 tjedana za finalni proizvod koji će sadržavati sve funkcionalne i nefunkcionalne zahtjeve zadane u zadatku te potpunu dokumentaciju projekta.

2.1. Funkcionalni zahtjevi

ID zahtjeva	Opis	Prioritet	Izvor	Kriterij Prihvaćenja
FZ-01	Sustav omogućava korisniku da stvori novi račun, unosom valjanog korisničkog imena, lozinke i email-a.	Visok	Zahtjev dionika	Korisnik se može registrirati u sustav unošenjem valjanih podataka.
FZ-02	Sustav omogućava prijavu na sustav pomoću korisničkog imena i lozinke ili preko Google računa.	Visok	Zahtjev dionika	Korisnik može unijeti valjano korisničko ime i lozinku te se uspješno prijaviti u sustav.
FZ-03	Sustav omogućava korištenje sustava anonimno bez registracije za pregled nesreća i sigurnosnih smjernica.	Srednji	Zahtjev dionika	Anonimni korisnici mogu pristupiti funkcionalnostima kao što su pregled nesreća i sigurnosnih smjernica.
FZ-04	Sustav omogućava da korisniku da vidi sve potvrđene nesreće u svojoj blizini i pregled detaljne informacije o nesreći.	Visok	Povratne informacije korisnika	Korisnik može pregledati listu nesreća koje su potvrđene od strane povlaštenih korisnika u radijusu od 5 km od svoje lokacije s detaljima o vremenu, mjestu i opisu nesreće.
FZ-05	Sustav omogućava pristup uputama i	Srednji	Dokument zahtjeva	Korisnik može vidjeti sigurnosne upute i smjernice

ID zahtjeva	Opis	Prioritet	Izvor	Kriterij Prihvaćenja
	sigurnosnim smjernicama unutar aplikacije.			unutar aplikacije, sortirane prema vrsti nesreće.
FZ-06	Sustav omogućava prijavu nesreće putem aplikacije te dodatne informacije.	Visok	Dokument zahtjeva	Korisnik može podnijeti prijavu s detaljima nesreće uključujući lokaciju, slike, vrstu i opis događaja.
FZ-07	Sustav šalje notifikacije o novonastalim nesrećama u blizini putem e-maila.	Visok	Povratne informacije korisnika	Korisnik prima notifikaciju u roku od 1 minute nakon potvrde nesreće u radijusu od 5 km.
FZ-08	Sustav šalje notifikacije o završenim nesrećama u blizini putem e-maila	Visok	Povratne informacije korisnika	Korisnik prima notifikaciju nakon što je nesreća završila.
FZ-09	Hitna Služba ima povlašten pristup i mogućnosti.	Visok	Zahtjev dionika	Hitna Služba može pregledati sve prijavljene nesreće, postavljati upute, pregledavati statističke podatke.
FZ-10	Sustav administratoru omogućava uklanjanje korisnika, dodjeljivanje uloga i odobravanje prijedloga za unaprjeđenje sustava.	Visok	Zahtjev dionika	Administrator može obavljati sve administrativne radnje unutar korisničkog panela.
FZ-11	Sustav omogućava korisniku da stvori organizaciju ili da postane dio postojeće organizacije uz dopuštenje od strane organizacije.	Srednji	Zahtjev dionika	Korisnik može obavljati radnje u ime organizacije. Korisnik mora biti verificiran da bi mogao kreirati vlastitu organizaciju.
FZ-12	Korisnik ima pregled povijesti svojih prijava.	Nizak	Dokument zahtjeva	Korisnik može pregledati sve nesreće koje je on prijavio.
FZ-13	Povlašteni korisnici imaju mogućnost potvrde nesreće.	Visok	Povratne informacije korisnika	Povlašteni korisnik je u mogućnosti potvrditi autentičnost prijave.

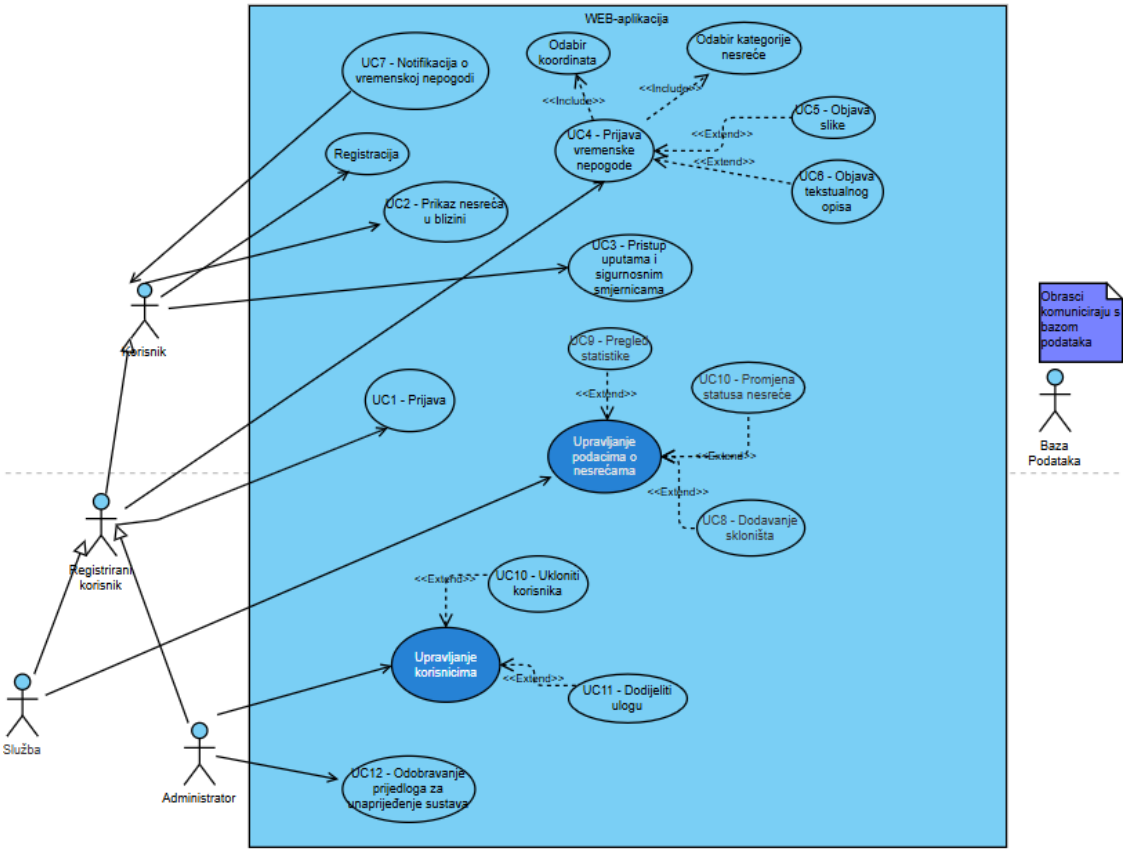
ID zahtjeva	Opis	Prioritet	Izvor	Kriterij Prihvaćenja
FZ-14	Korisnici mogu verificirati svoj identitet putem maila.	Nizak	Povratne informacije korisnika	Korisnik prijavljen u sustav na svoju email adresu dobije verifikacijski token. U slučaju da je korišten OAUTH2 korisnik je automatski verificiran.

2.2. Ostali zahtjevi

ID Zahtjeva	Opis	Priortiet
NF-01	Aplikacija mora biti dostupna većinu vremena kroz godinu.	Visok
NF-02	Aplikacija mora biti dostupna svim korisnicima, primarno u Hrvatskoj pa i u cijelom svijetu.	Visok
NF-03	Aplikacija mora brzo ažurirati prijave nesreća, unutar 10 sekundi od prijave.	Visok
NF-04	Aplikacija mora podržati istovremeno velik broj korisnika.	Visok
NF-05	Aplikacija mora grupirati dojave po mjestu nesreće i kategoriji radi preglednosti.	Srednji
NF-06	Web sučelje mora biti pregledno i jednostavno za korištenje.	Visok
NF-07	Web sučelje mora biti responzivno na svim uređajima, uključujući stolna računala, tablete i mobilne telefone.	Visok
NF-08	Sustav treba biti oblikovan tako da omogućuje jednostavno održavanje.	Visok
NF-09	Sustav treba imati dovoljnu dokumentaciju.	Visok

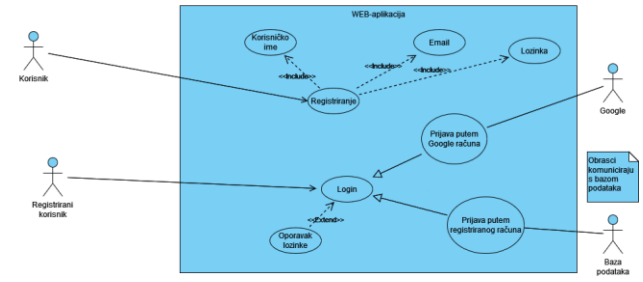
3.1. Dijagrami obrazaca uporabe

Globalni dijagram obrazaca uporabe

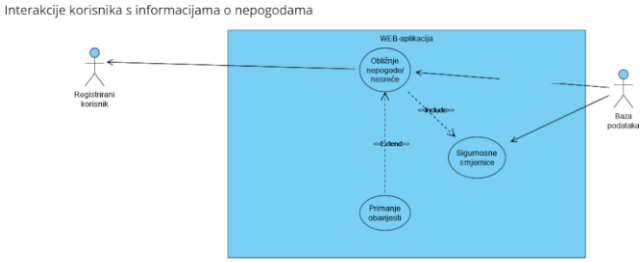


Dijagram obrazaca uporabe - Prijava i registracija korisnika

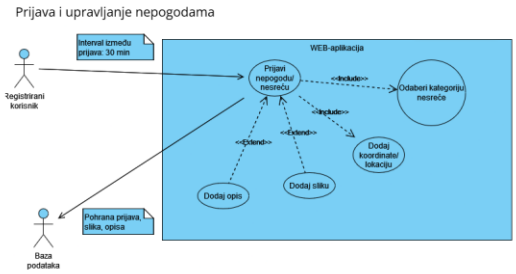
Prijava i registracija korisnika



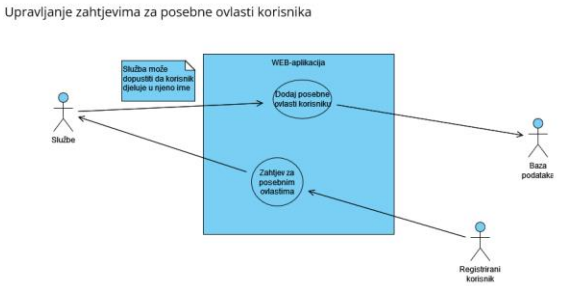
Dijagram obrazaca uporabe - Interakcije korisnika s informacijama o nepogodama



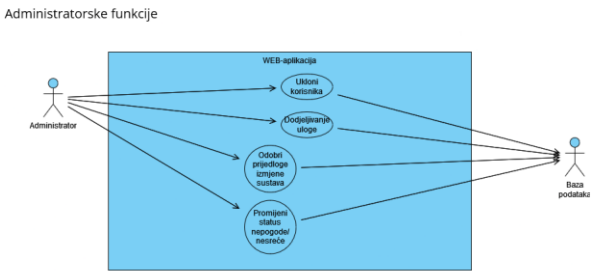
Dijagram obrazaca uporabe - Prijava i upravljanje nepogodama



Dijagram obrazaca uporabe - Upravljanje zahtjevima za posebne ovlasti korisnika



Dijagram obrazaca uporabe - Administratorske funkcije



3.2. Opis obrazaca uporabe

UC1 - Prijava

- Glavni sudionik: Registrirani korisnik
- Cilj: Omogućiti pristup aplikaciji o vremenskim nepogodama registriranim korisnicima.
- Sudionici: Baza podataka
- Preduvjet: Korisnik mora biti prethodno registriran u sustavu.
- Opis osnovnog tijeka:

1. Registrirani korisnik odabire opciju za prijavu.
2. Unosi korisničko ime i lozinku.
3. Klikne na dugme za potvrdu prijave.
4. Sustav provjerava podatke u bazi podataka.
5. Ako su podaci točni, korisnik dobija pristup aplikaciji i preusmjerava se na početnu stranicu.

• Moguća odstupanja:

1. Neispravni podaci za prijavu: Ako su podaci netočni, sustav obavještava korisnika o pogrešci. Korisnik može ponovno unijeti podatke ili koristiti opciju za oporavak lozinke.
2. Zaboravljena lozinka: Sustav preusmjerava korisnika na postupak oporavka lozinke putem e-maila.
3. Privremena nedostupnost baze: Ako baza nije dostupna, korisnik je obaviješten o privremenoj grešci i može pokušati kasnije.

UC2 - Prikaz nesreća u blizini

- Glavni sudionik: Korisnik
- Cilj: Pružiti korisniku informacije o nesrećama i vremenskim nepogodama u njegovoj blizini.
- Sudionici: Baza podataka, vanjski sustavi za praćenje nepogoda

- Preduvjet: Korisnik je prijavljen i omogućio pristup svojoj lokaciji.
- Opis osnovnog tijeka:

1. Korisnik odabire opciju za prikaz nesreća.
2. Sustav dohvaća korisnikovu lokaciju.
3. Vanjski servisi šalju podatke o nesrećama i nepogodama u blizini.
4. Sustav prikazuje informacije o nesrećama unutar postavljenog radijusa.
5. Korisnik vidi detalje o nesreći uključujući vrstu nepogode, točnu lokaciju i sigurnosne mjere.

• Moguća odstupanja:

1. Nema informacija o nesrećama: Ako nema podataka, sustav obavještava korisnika da nema prijavljenih nesreća u blizini.
2. Neuspješno dohvaćanje lokacije: Ako korisnik nije omogućio pristup lokaciji, sustav daje obavijest s uputama za aktivaciju.

UC3 - Pristup uputama i sigurnosnim smjernicama

- Glavni sudionik: Korisnik
- Cilj: Omogućiti pristup sigurnosnim smjernicama i uputama u slučaju nepogoda.
- Sudionici: Baza podataka
- Preduvjet: Korisnik je prijavljen.
- Opis osnovnog tijeka:

1. Korisnik odabire opciju "Upute i sigurnosne smjernice".
2. Sustav prikazuje stranice s informacijama o sigurnosnim mjerama za različite nepogode.

• Moguća odstupanja:

1. Nedostupnost informacija: Ako podaci nisu dostupni, korisnik je obaviješten o grešci.
2. Neadekvatne informacije: Korisnik može prijaviti problem, a administratori mogu ažurirati informacije.

UC4 - Prijava vremenske nepogode

- Glavni sudionik: Registrirani korisnik
- Cilj: Omogućiti korisniku prijavu vremenskih nepogoda u stvarnom vremenu.
- Sudionici: Baza podataka, Administratori sustava
- Preduvjet: Korisnik je prijavljen.
- Opis osnovnog tijeka:

1. Korisnik odabire opciju za prijavu nepogode.
2. Sustav prikazuje obrazac za unos podataka.
3. Korisnik unosi podatke, uključujući vrstu nepogode, lokaciju, intenzitet i opcionalno fotografiju.
4. Korisnik potvrđuje prijavu.

5. Sustav pohranjuje prijavu i obavještava korisnika o uspjehu.

• Moguća odstupanja:

1. Neispravan unos: Ako podaci nisu ispravni, sustav traži ispravke.
2. Neuspješan prijenos fotografije: Ako prijenos ne uspije, korisnik može nastaviti bez fotografije.

UC5 - Objava slike

- Glavni sudionik: Registrirani korisnik
- Cilj: Omogućiti dodavanje slike vezane uz vremensku nepogodu.
- Sudionici: Baza podataka, Administratori sustava
- Preduvjet: Korisnik je prijavljen, prijava nepogode je uspješno unesena.
- Opis osnovnog tijeka:

1. Korisnik odabire prijavu vremenske nepogode.
2. Sustav prikazuje detalje prijave.
3. Korisnik odabire opciju za dodavanje slike/video sadržaja.
4. Korisnik odabire datoteku i potvrđuje prijenos.
5. Sustav provodi provjeru datoteke i pohranjuje je.
6. Slika postaje vidljiva svim korisnicima.

• Moguća odstupanja:

1. Neispravan format slike: Ako format nije podržan, sustav traži ispravnu sliku.
2. Neispravna veličina datoteke: Ako je slika prevelika, korisnik je obaviješten da je potrebno smanjiti je.
3. Neuspješan prijenos slike: Ako prijenos ne uspije, korisnik može pokušati ponovno.

UC6 - Objava tekstualnog opisa

- Glavni sudionik: Registrirani korisnik
- Cilj: Omogućiti korisniku dodavanje tekstualnog opisa uz prijavljenu nepogodu.
- Sudionici: Baza podataka, Administratori sustava
- Preduvjet: Korisnik je prijavljen, prijava nepogode je uspješno unesena.
- Opis osnovnog tijeka:

1. Korisnik odabire prijavu nepogode.
2. Sustav prikazuje detalje prijave.
3. Korisnik unosi dodatni tekstualni opis.
4. Sustav pohranjuje opis i povezuje ga s prijavom.
5. Tekstualni opis postaje vidljiv svim korisnicima.

• Moguća odstupanja:

1. Neispravan unos teksta: Ako tekst nije relevantan ili je prekratak, korisnik mora unijeti smisleniji opis.

2. Neuspješno spremanje opisa: Ako opis nije pohranjen, korisnik može pokušati ponovno.

UC7 - Notifikacija o vremenskoj nepogodi

- Glavni sudionik: Korisnik
- Cilj: Obavijestiti korisnika o prijavljenim vremenskim nepogodama u njegovoj okolini.
- Sudionici: Baza podataka, Administratori sustava
- Preduvjet: Aplikacija je instalirana (nije nužno prijavljen).
- Opis osnovnog tijeka:

1. Korisnik instalira aplikaciju i pokreće je.
2. Sustav nudi opciju za primanje notifikacija o nepogodama.
3. Korisnik omogućava obavijesti putem push notifikacija, SMS-a ili e-maila.
4. Sustav prepoznaje novu nepogodu (npr. poplava, oluja).
5. Korisniku se šalje obavijest s osnovnim informacijama (vrsta nepogode, lokacija, procjena intenziteta).
6. Korisnik može otvoriti aplikaciju za više informacija.

• Moguća odstupanja:

1. Neispravna dostava: Pokušaj ponovnog slanja obavijesti ili kontaktiranje podrške.
2. Prekomjerno slanje obavijesti: Korisnik prilagođava vrste nepogoda ili regije za obavijesti.

UC8 - Dodavanje skloništa

- Glavni sudionik: Služba
- Cilj: Dodati skloništa za pomoć građanima.
- Sudionici: Baza podataka
- Preduvjet: Služba je prijavljena.
- Opis osnovnog tijeka:

1. Služba se prijavljuje.
2. Odabire opciju za dodavanje skloništa.
3. Unosi podatke o skloništu.
4. Sustav sprema podatke i obavještava službu.
5. Skloništе je dostupno korisnicima.

• Moguća odstupanja:

1. Neispravan unos: Ponovno unos ili kontaktiranje podrške.
2. Pogrešna lokacija: Ispravak putem mapa ili geolokacije.

UC9 - Pregled statistike

- Glavni sudionik: Služba
- Cilj: Pregledati statistiku vremenskih nepogoda.

- Sudionici: Baza podataka
- Preduvjet: Služba je prijavljena.
- Opis osnovnog tijeka:

1. Služba se prijavljuje.
2. Odabire opciju za pregled statistike.
3. Sustav prikazuje podatke o nepogodama.
4. Služba filtrira i analizira podatke.

• Moguća odstupanja:

1. Neispravan prikaz: Ponovno učitavanje podataka.
2. Pogrešno filtriranje: Ponovno postavljanje filtara.

UC10 - Promjena statusa nesreće

- Glavni sudionik: Služba
- Cilj: Ažurirati status prijavljenih nepogoda.
- Sudionici: Baza podataka
- Preduvjet: Služba je prijavljena.
- Opis osnovnog tijeka:

1.Služba se prijavljuje. 2.Odabire nepogodu za promjenu statusa. 3.Biranje novog statusa. 4.Ažuriranje baze i obavještavanje korisnika.

• Moguća odstupanja:

1. Pogrešna promjena statusa: Poništavanje ili ponovni odabir statusa.

UC11 - Ukloniti korisnika

- Glavni sudionik: Administrator
- Cilj: Ukloniti korisnika iz sustava.
- Sudionici: Baza podataka
- Preduvjet: Administrator je prijavljen.
- Opis osnovnog tijeka:

1.Administrator se prijavljuje. 2.Odabire korisnika za uklanjanje. 3.Potvrda uklanjanja.

• Moguća odstupanja:

1. Greška pri uklanjanju: Ponovno pokušati ili kontaktirati podršku.

UC12 - Dodijeliti uloge

- Glavni sudionik: Administrator
- Cilj: Dodijeliti uloge korisnicima.
- Sudionici: Baza podataka
- Preduvjet: Administrator je prijavljen.

- Opis osnovnog tijeka:

1. Administrator se prijavljuje.
2. Odabire korisnika i novu ulogu.
3. Potvrda promjene.

• Moguća odstupanja:

1. Pogrešna dodjela uloge: Poništavanje ili ponovni odabir uloge.

UC13 - Odobravanje prijedloga za unaprjeđenje sustava

- Glavni sudionik: Administrator
- Cilj: Omogućiti administratoru pregled i odobravanje prijedloga korisnika za unaprjeđenje sustava.
- Sudionici: Registrirani korisnik, Baza podataka, Korisnički sustav
- Preduvjet: Administrator je prijavljen, korisnici su podnijeli prijedloge.
- Opis osnovnog tijeka:

1. Administrator se prijavljuje u sustav.
2. Sustav prikazuje prijedloge korisnika.
3. Administrator pregledava prijedloge (naziv, opis, prioritet).
4. Administrator odabire prijedlog za pregled.
5. Administrator odlučuje:

- Odobriti prijedlog za implementaciju.
- Odbiti prijedlog uz objašnjenje.

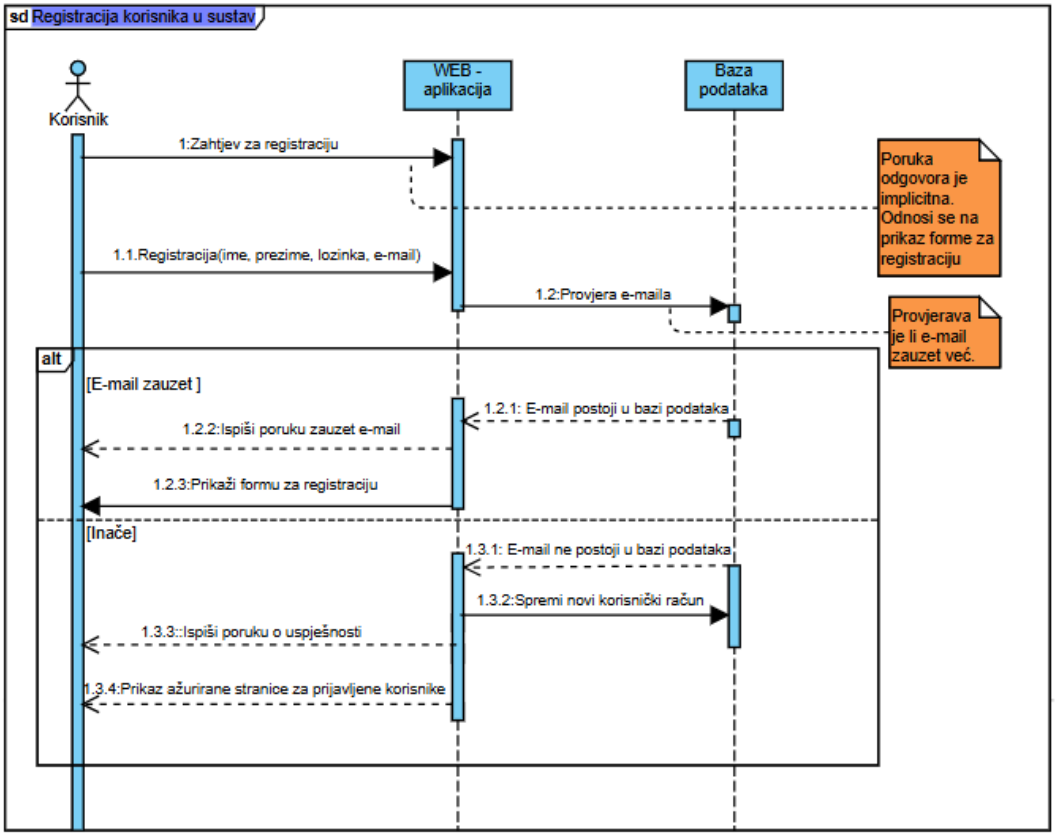
6. Administrator potvrđuje odluku.
7. Ako je prijedlog odobren, prelazi u fazu implementacije.
8. Korisnik prima obavijest o statusu prijedloga.

- Moguća odstupanja:

1. Administrator ne može pregledati prijedloge: Greške u sustavu mogu zahtijevati ponovni pokušaj ili kontaktiranje podrške.
2. Prijedlog nije dostupan: Ako su prijedlozi nedostupni zbog greške u bazi, administrator može čekati njihovu dostupnost.
3. Neodgovarajući prijedlog: Administrator može odbiti prijedlog i tražiti dodatne informacije od korisnika.

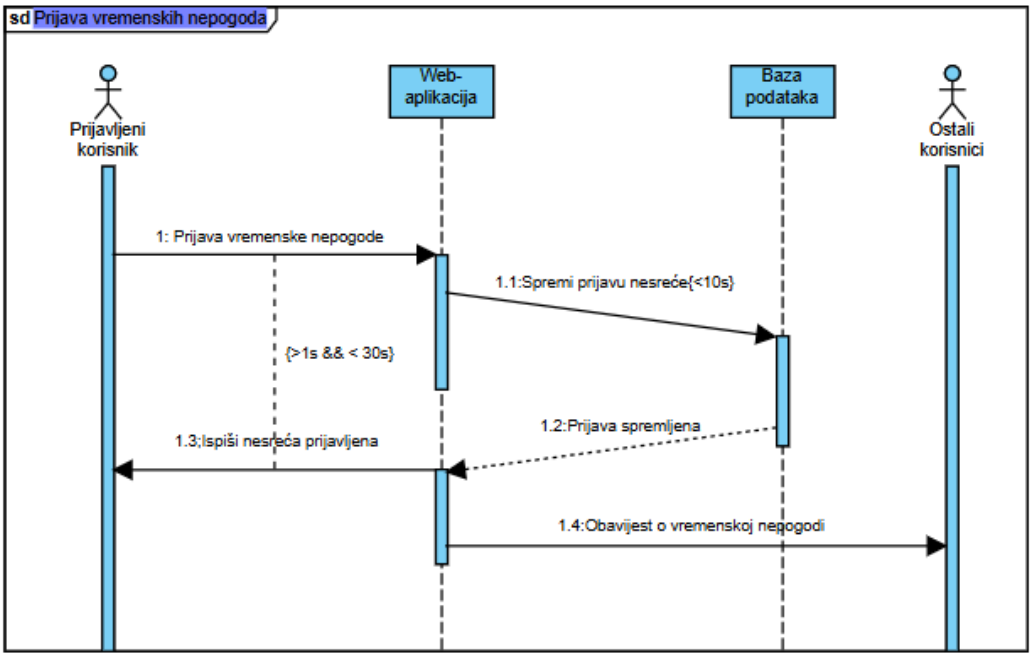
3.3. Sekvencijski dijagrami

Registracija korisnika



1. Korisnik se registrira, sustav provjerava podatke.
2. Prikazuje se poruka o uspjehu ili grešci ovisno o ispravnosti podataka.

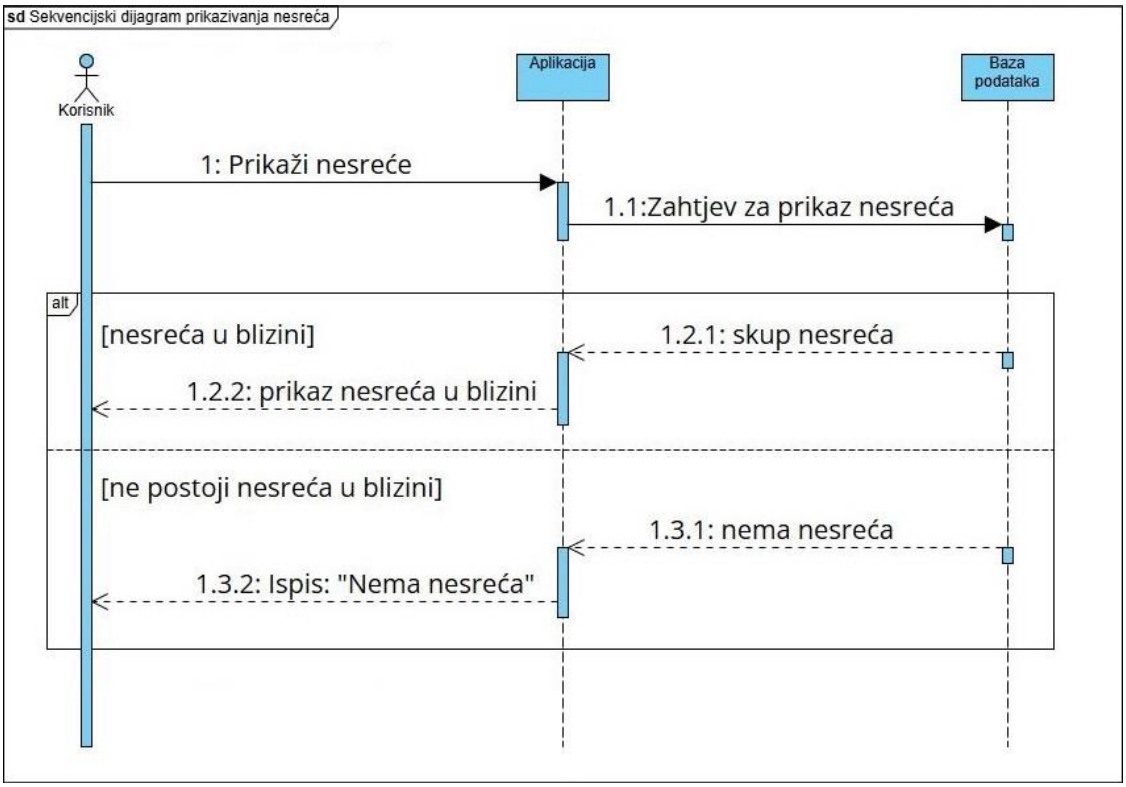
Prijava vremenske nepogode



1. Korisnik prijavljuje nepogodu, sustav sprema prijavu i potvrđuje.
2. Obavijest se šalje ostalim korisnicima.

- Obuhvaća UC4 (Prijava vremenske nepogode) i UC7 (Notifikacija o vremenskoj nepogodi).

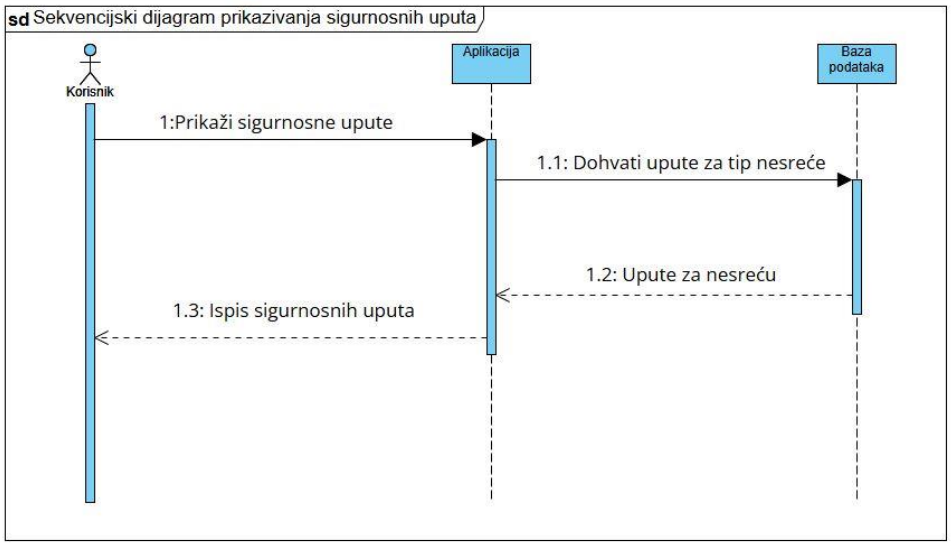
Prikazivanje nesreće



1. Korisnik traži prikaz nesreća u aplikaciji.
2. Aplikacija provjerava bazu podataka za nesreće u blizini.
3. Ako postoje nesreće, prikazuju se; inače se ispisuje poruka "Nema nesreća".

- Obuhvaća UC2 (Prikaz nesreća u blizini).

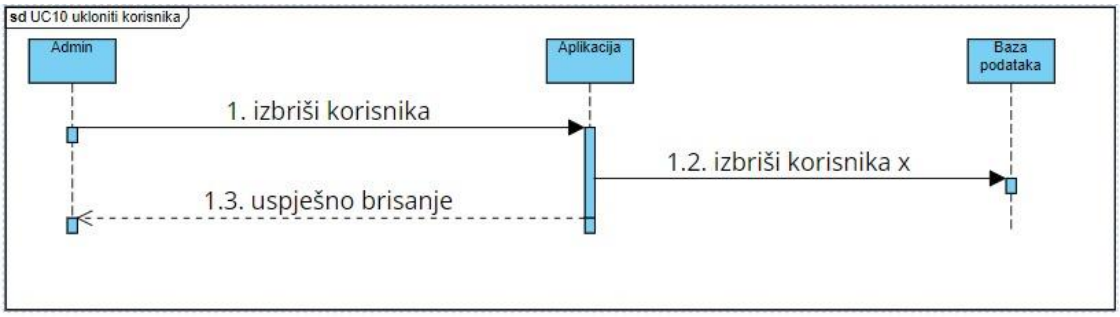
Prikazivanje sigurnosnih uputa



1. Korisnik traži sigurnosne upute za određeni tip nesreće.
2. Aplikacija dohvaća upute iz baze podataka i prikazuje ih korisniku.

- Obuhvaća UC3 (Pristup uputama i sigurnosnim smjernicama).

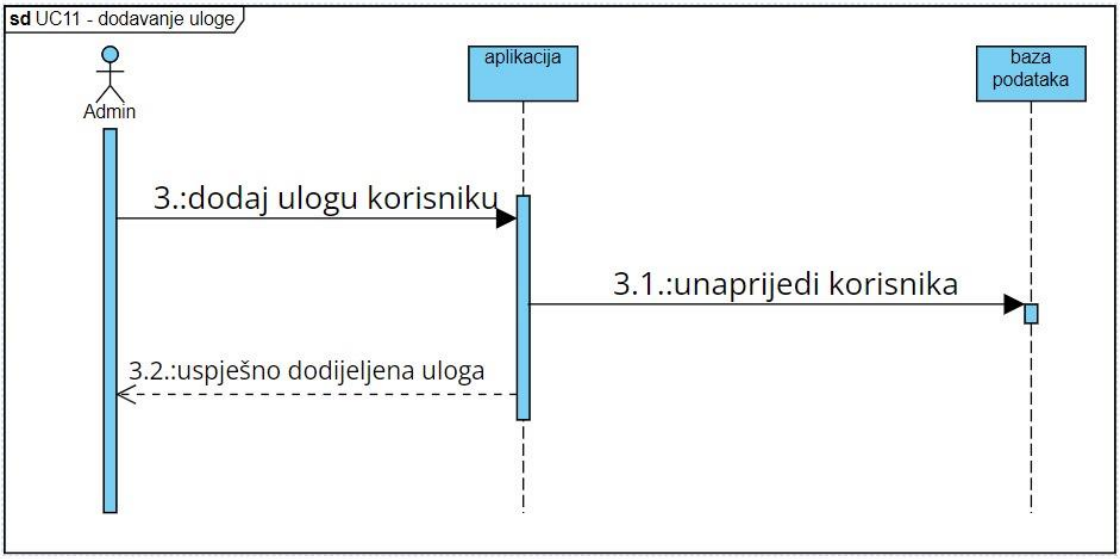
Brisanje korisnika



1. Administrator odabire korisnika za brisanje
2. Sustav potvrđuje uspješno brisanje.

- Obuhvaća UC11 (Ukloniti korisnika).

Dodjela uloge korisniku



1. Administrator dodaje ulogu korisniku.
2. Aplikacija ažurira korisnički status u bazi i potvrđuje dodjelu uloge.

- Obuhvaća UC12 (Dodijeliti uloge).

4.1. Arhitektura sustava

Ovaj dokument opisuje arhitekturu sustava koji omogućuje korisnicima prijavu incidenata, praćenje statusa prijava i primanje obavijesti. Arhitektura koristi klijent-poslužitelj pristup, s Reactom na frontend strani i Java Spring Boot na backend strani. Za sigurnost i autentifikaciju koristi OAuth2 i JSON Web Token (JWT) protokole, čime se osigurava sigurna i autorizirana interakcija unutar sustava.

Opis arhitekture

- Stil arhitekture: Sustav koristi klijent-poslužitelj arhitekturu s višeslojnim modelom, gdje je klijent razvijen u Reactu za dinamičko korisničko sučelje, dok je backend u Spring Bootu odgovoran za poslovnu logiku i sigurnost. Klijent-poslužitelj arhitektura omogućava jasnu podjelu odgovornosti između prezentacijskog sloja i poslovne logike, čime se olakšava održavanje i skalabilnost. Korištenje OAuth2 i JWT protokola dodaje sloj sigurnosti, omogućujući kontrolirani pristup resursima.

- Podsustavi:
  -Klijent (React): Klijent omogućuje korisnicima prijavu incidenata, praćenje statusa i primanje obavijesti. React pruža responzivno i dinamično korisničko sučelje koje komunicira s backendom putem REST API-ja.
  -Backend API (Spring Boot): Backend upravlja poslovnom logikom, autentifikacijom, autorizacijom i komunikacijom s bazom podataka. Spring Boot pruža stabilnu i skalabilnu platformu za razvoj RESTful servisa i sigurnosnih

protokola.
-Autentifikacijski servis (OAuth2 i JWT): Upravlja sigurnom prijavom korisnika i kontrolom pristupa resursima. OAuth2 služi za autorizaciju, dok JWT tokeni osiguravaju siguran način autentifikacije za klijentske zahtjeve.
-Notifikacijski podsustav: Omogućuje slanje push obavijesti ili e-mailova korisnicima. Integracija s vanjskim servisima poput Firebase Cloud Messaging (FCM) ili SMTP servera omogućava pravovremeno obavještavanje korisnika.

- Mrežni protokoli: Komunikacija između klijenta i backend servera odvija se putem HTTPS protokola radi zaštite podataka. REST API omogućuje komunikaciju između frontenda i backenda, dok OAuth2 i JWT osiguravaju da samo autorizirani korisnici imaju pristup resursima. OAuth2 upravlja inicijalnom autorizacijom, dok se JWT koristi za autentifikaciju prilikom svakog zahtjeva, čime se smanjuje potreba za višestrukom prijavom.

- Globalni upravljački tok

1. Prijava korisnika (OAuth2): Korisnici se prijavljuju putem React sučelja, a autentifikacija se odvija putem OAuth2 protokola. Backend provjerava vjerodajnice i generira JWT token koji se vraća klijentu.
2. Zahtjevi s JWT autentifikacijom: Klijent uključuje JWT token u zaglavlje svakog zahtjeva prema backendu. Backend validira token kako bi potvrdio autentifikaciju i autorizaciju korisnika.
3. Backend obrada: Spring Boot backend obrađuje zahtjeve, komunicira s bazom podataka i pohranjuje podatke.
4. Notifikacijski podsustav: Backend šalje obavijesti korisnicima putem integriranog notifikacijskog servisa (npr. FCM za push ili SMTP za e-mail).
- Sklopovskoprogramski zahtjevi: Web aplikacija je optimizirana za moderne preglednike (Chrome, Firefox, Safari).

Obrazloženje odabira arhitekture

Izbor arhitekture temeljen na principima oblikovanja

Arhitektura je odabrana na temelju visoke kohezije, niske povezanosti, skalabilnosti i sigurnosti:

- Modularnost i sigurnost: React frontend i Spring Boot backend omogućuju odvojeni razvoj i održavanje korisničkog sučelja i poslovne logike. Kombinacija OAuth2 i JWT osigurava da samo autorizirani korisnici imaju pristup osjetljivim podacima.
- Skalabilnost: Cloud infrastruktura omogućava lako skaliranje resursa, dok klijent-poslužitelj arhitektura osigurava brzi odziv i odvojenost klijenta i servera.
- Sigurnost podataka: HTTPS osigurava sigurnu komunikaciju, dok OAuth2 omogućuje siguran pristup resursima, a JWT osigurava autentifikaciju bez potrebe za višestrukom prijavom.

Organizacija sustava na visokoj razini

- Klijent-poslužitelj Klijent-poslužitelj arhitektura omogućuje podjelu funkcionalnosti između React frontenda koji korisnicima pruža interaktivno sučelje i Spring Boot backenda koji upravlja poslovnom logikom i sigurnosnim protokolima.
- Baza podataka Koristi se PostgreSQL relacijska baza podataka za pohranu podataka o korisnicima, incidentima i povijesti obavijesti. PostgreSQL omogućuje sigurnu pohranu i napredne SQL funkcionalnosti za analizu podataka.
- Grafičko sučelje React frontend pruža interaktivno korisničko sučelje, omogućavajući korisnicima prijavu, praćenje statusa i primanje obavijesti. React koristi REST API pozive za komunikaciju s backendom, omogućujući brze i dinamične promjene na sučelju.

Organizacija aplikacije

Frontend i Backend slojevi

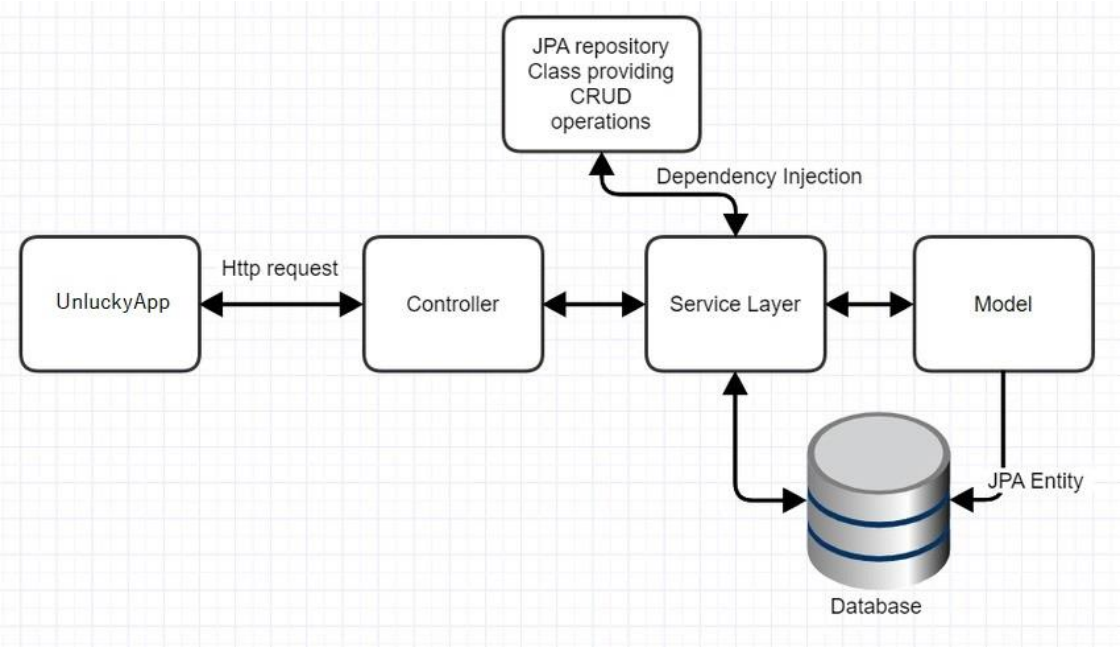
• Frontend (React): Klijent u Reactu omogućuje korisnicima rad s aplikacijom i uključuje JWT token u sve zahtjeve prema backendu. Frontend služi kao interaktivno sučelje koje je sigurno povezano s backendom.

• Backend (Spring Boot): Spring Boot backend obrađuje poslovnu logiku, autentifikaciju, autorizaciju, pohranu podataka i komunikaciju s bazom podataka. Backend koristi JWT za provjeru svakog zahtjeva koji stiže s frontenda.

MVC arhitektura

Backend koristi Model-View-Controller (MVC) arhitekturu koja odvaja podatkovni model (PostgreSQL baza podataka), poslovnu logiku (Controller u Spring Bootu) i prikaz koji klijent koristi za prikaz podataka korisnicima.

Blok dijagram arhitekure backenda



- Ova slika prikazuje arhitekturu Java Spring Boot aplikacije koja koristi slojeve za komunikaciju s bazom podataka pomoću JPA (Java Persistence API) za CRUD (Create, Read, Update, Delete) operacije. Ova arhitektura omogućava odvajanje korisničkog sučelja, poslovne logike i pristupa podacima, čineći aplikaciju jednostavnom za održavanje i proširivanje.

4.2. OPIS BAZE

4.2.1. Opis tablica

Glavni zadatak baze podataka je da osigura pohranu i pregled svih podataka vezanih za prijave nesreća u aplikaciji kao i podatke o svim prijavljenim korisnicima. Kako bi baza podataka ispravno funkcionirala bitno je da je normalizirana na razinu 3NF te da sadržava sve potrebne entitete.

1. **AppUser**

Entitet AppUser je iznimno bitna tablica u našoj bazi jer sadržava sve podatke o svim prijavljenim korisnicima. Za svakog korisnika se bilježi korisničko ime (username), password (lozinka) kao i email adresa. Svaki korisnik jedinstveno je određen email adresom te username-om kojima se registrirao u aplikaciji.

Column	Data Type	Description
id (PK)	`bigint`	Unique identifier for the user
organisation_id (FK)	`bigint`	Reference to `organisation.id` (if the user is in an org)
verified	`boolean`	Indicates if the user is verified (e.g., email

Column	Data Type	Description
		confirmed)
email	`varchar(255)`	User’s email address
org_rank	`varchar(255)`	Role/rank within the organization (if any)
password	`varchar(255)`	User’s password
username	`varchar(255)`	User’s username

2. **Role**

Entitet Role definira sve uloge koje prijavljeni korisnici mogu imati (od običnih korisnika do admin pozicija).

Column	Data Type	Description
id (PK)	`bigint`	Unique identifier for the role
name	`varchar(255)`	Name of the role (e.g., “ADMIN”)

3. **City**

Entitet City opisuje gradove(naselja) koja su definirana svojim jedinstvenim poštanskim brojem i nazivom. Služi kao sadržajna jedinica za sva skloništa i lokacije u bazi podataka.

Column	Data Type	Description
id (PK)	`bigint`	Unique identifier for the city
name	`varchar(255)`	Name of the city
zip_code	`varchar(255)`	Postal (ZIP) code
latitude_city_center	`double precision`	Latitude coordinate of the city center
longitude_city_center	`double precision`	Longitude coordinate of the city center

4. **Instruction**

Sadrži upute ponašanja koje sustav prikazuje ovisno o tipu prijavljene nesreće. Svaka uputa jedinstveno je određena vlastitim identifikacijskim brojem.

Column	Data Type	Description
id (PK)	`bigint`	Unique identifier for the instruction
disaster_type	`smallint`	Type of disaster (could be a code or enum)
contact_id (FK)	`bigint`	Reference to `contact.id`
description	`varchar(255)`	Short description of the instruction
title	`varchar(255)`	Title of the instruction

5. **Contact**

Sadržava kontakt informaciju prikladnih službi ovisno o vrsti prijavljene nesreće. | **Column** | **Data Type** | **Description** | |--------------|----------------|---| | **id** (PK) | `bigint` | Unique identifier for the contact | | name | `varchar(255)` | Name of the contact (e.g., “Police”) | | number | `varchar(255)` | Phone number |

---

6. **Organisation**

Ovaj entitet definira podatke organizacija koje vode aplikaciju.

Column	Data Type	Description
id (PK)	`bigint`	Unique identifier for the organization
name	`varchar(255)`	Name of the organization
email	`varchar(255)`	Organization’s email address
description	`varchar(255)`	Short description of the organization

---

7. **Location**

Kako bi aplikacija ispravno funkcionirala neophodno je da svaka prijava ima lokaciju. Stoga se u ovom entitetu spremaju jedinstvene koordinate svake lokacije, poštanski broj unutar kojeg se nalaze kao i adresa u slučaju da postoji na toj lokaciji.

Column	Data Type	Description
id (PK)	`bigint`	Unique identifier for the location
latitude	`double precision`	Latitude coordinate
longitude	`double precision`	Longitude coordinate
adress	`varchar(255)`	Address (if applicable)
city_id (FK)	`bigint`	Reference to `city.id`

---

8. **Shelter**

U slučaju nepogoda, organizacija može slati ljude na lokacije skloništa gdje će im biti dana pomoć. Svako od ovih skloništa ima jedinstveni id unutar iste općine (poštanski broj, tj. zip_code). Stoga je svako sklonište jedinstveno određeno svojim poštanskim brojem kao i svojim id brojem. Ovaj entitet uz id sadrži informacije o maksimalnom dozvoljenom broju ljudi kao i o imenu skloništa.

Column	Data Type	Description
id (PK)	`bigint`	Unique identifier for the shelter
city_id (FK)	`bigint`	Reference to `city.id`

Column	Data Type	Description
location_id (FK)	`bigint`	Reference to `location.id`
name	`varchar(255)`	Name of the shelter
max_no_people	`integer`	Maximum capacity (number of people allowed)

---

9. **Report**

Entitet Report je iznimno bitan za bilježenje podataka o svakoj prijavi nesreće u aplikaciji. Sadrži atribute koji opisuju vrijeme prijave, opcionalni komentar i jedinstveni id prijave kojim je svaka prijava određena.

Column	Data Type	Description
id (PK)	`bigint`	Unique identifier for the report
location_id (FK)	`bigint`	Reference to `location.id`
user_id (FK)	`bigint`	Reference to `users.id`
report_date_time	`timestamp(6)`	Timestamp of when the report was created
status	`smallint`	Report status (e.g., 0=open, 1=in progress, etc.)
description	`varchar(500)`	Description of the incident
disaster_type	`varchar(255)`	Type of disaster (text/category)

---

10. **Photo**

Sadrži podatke o opcionalno podnesenim slikama uz prijavu nesreće. Svaka slika ima jedinstveni id, tj. URL.

Column	Data Type	Description
id (PK)	`bigint`	Unique identifier for the photo
report_id (FK)	`bigint`	Reference to `report.id`
uploaded_time	`timestamp(6)`	Timestamp indicating when the photo was uploaded
name	`varchar(255)`	Name of the photo/file
type	`varchar(255)`	Type (e.g., MIME type)
data	`oid`	Binary data of the photo (BLOB)

---

11. **Email_token**

Tablica za spremanje i upravljanje email tokenima (za npr. verifikaciju korisničkih računa).

Column	Data Type	Description
id (PK)	`bigint`	Unique identifier for the token

Column	Data Type	Description
user_id (FK)	`bigint`	Reference to the user (`users.id`)
token	`varchar(255)`	Verification token
confirmed	`boolean`	Indicates whether the email (token) is confirmed
time_stamp	`timestamp(6)`	Timestamp for creation or confirmation of the token

---

12. **Organisation_members**

Povezna (many-to-many) tablica koja spaja organizacije i korisnike koji su članovi tih organizacija.

Column	Data Type	Description
organisation_id (FK)	`bigint`	Reference to `organisation.id`
members_id (FK)	`bigint`	Reference to `users.id`

Note: Typically, the primary key is a composite of both columns.

---

13. **Organisation_pending_members**

Tablica za one korisnike koji su poslali zahtjev ili čekaju potvrdu za ulazak u organizaciju (također many-to-many veza).

Column	Data Type	Description
organisation_id (FK)	`bigint`	Reference to `organisation.id`
pending_members_id (FK)	`bigint`	Reference to `users.id`

Note: Typically, the primary key is a composite of both columns.

---

14. **Users_cities**

Povezna tablica (many-to-many) za korisnike i gradove. Primjerice, korisnik može biti povezan s više gradova.

Column	Data Type	Description
app_user_id (FK)	`bigint`	Reference to `users.id`
cities_id (FK)	`bigint`	Reference to `city.id`

Note: Typically, the primary key is a composite of both columns.

---

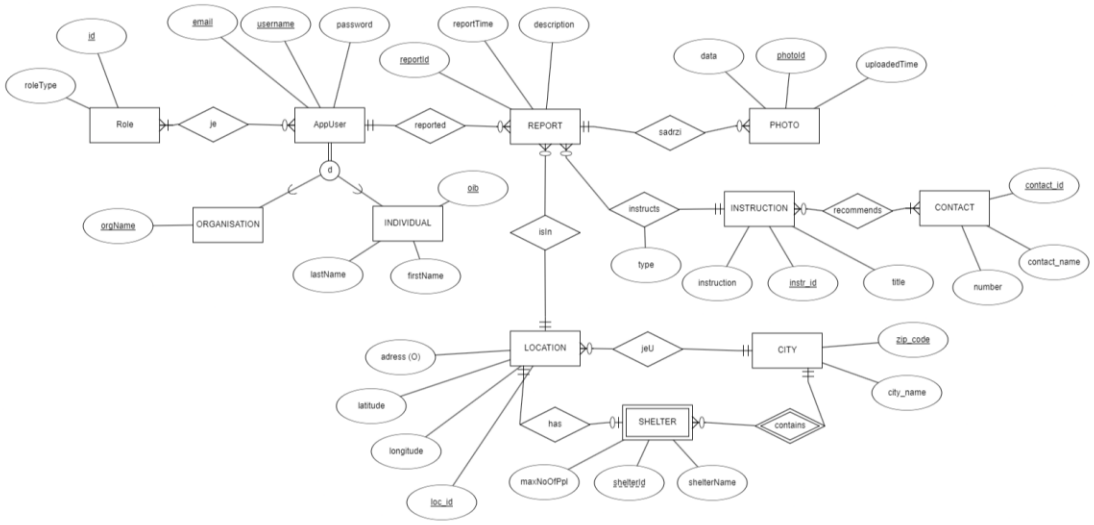
15. **Users_roles**

Povezna tablica (many-to-many) za korisnike i njihove uloge.

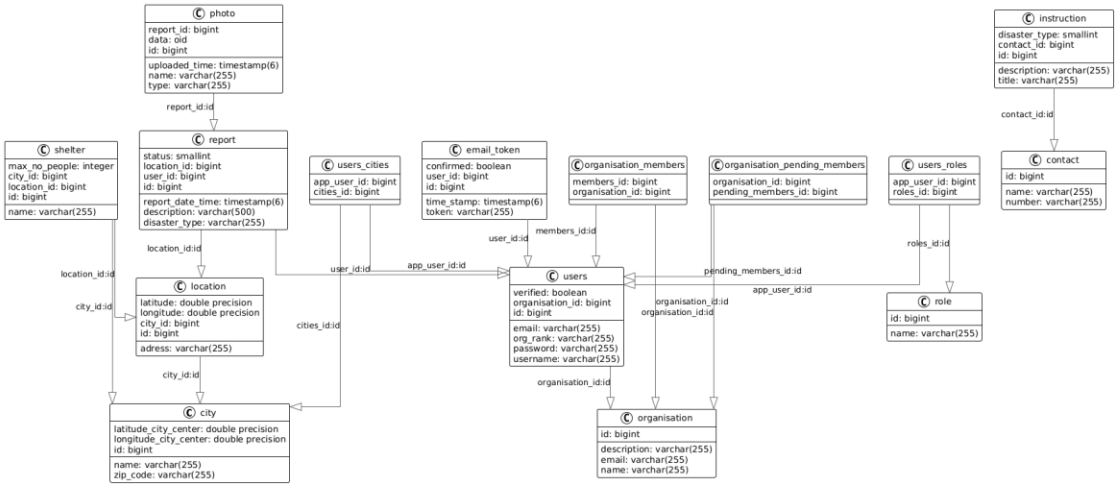
Column	Data Type	Description
app_user_id (FK)	`bigint`	Reference to `users.id`
roles_id (FK)	`bigint`	Reference to `role.id`

Note: Typically, the primary key is a composite of both columns.

4.2.2. Dijagram baze podataka

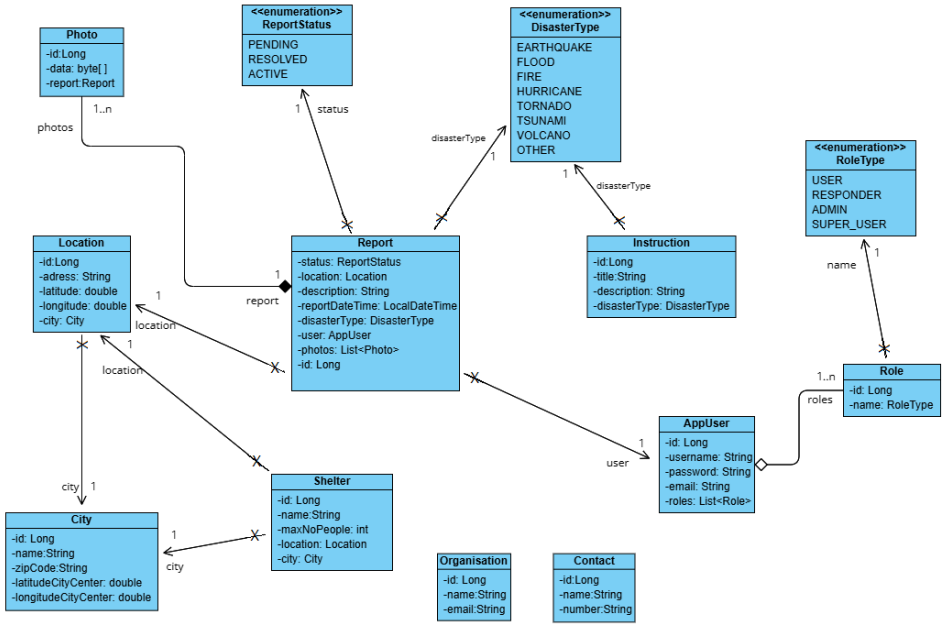


4.2.3. Relacijski dijagram

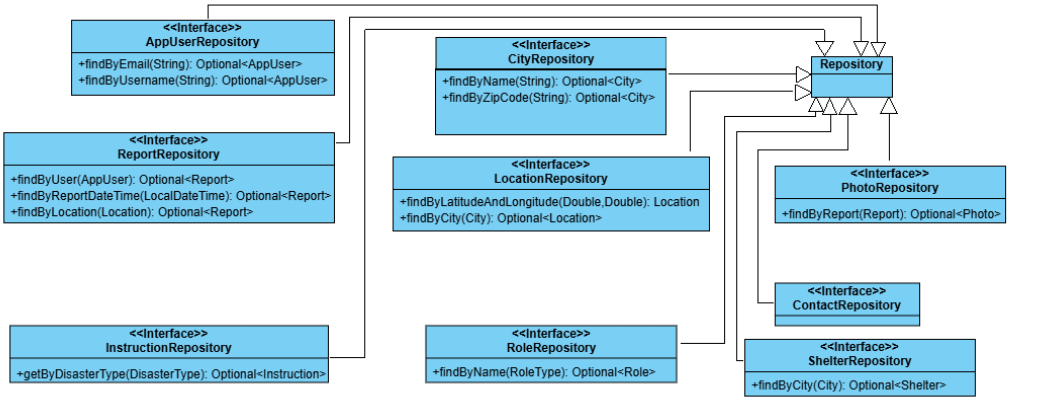


4.3. Dijagrami razreda

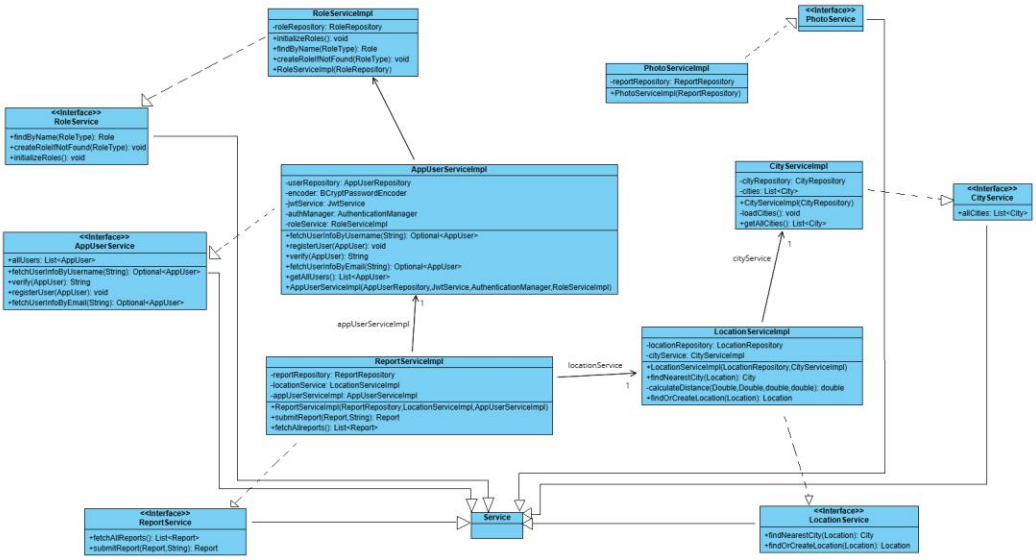
- Ovaj globalni dijagram razreda prikazuje sve odnose među razredima.



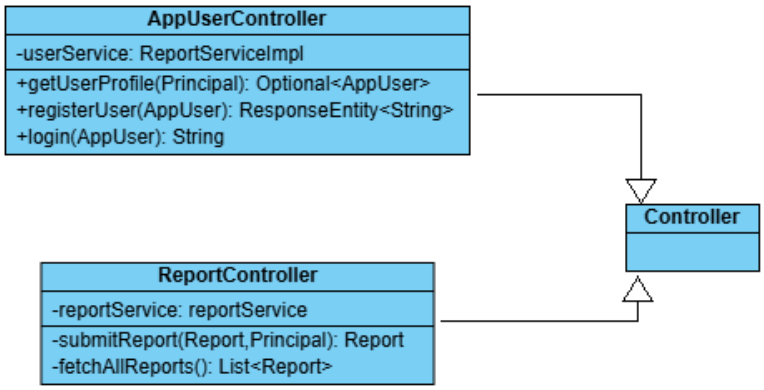
- Ovaj dijagram razreda prikazuje modele, odnosno razrede entiteta te enumeracije i njihove odnose.



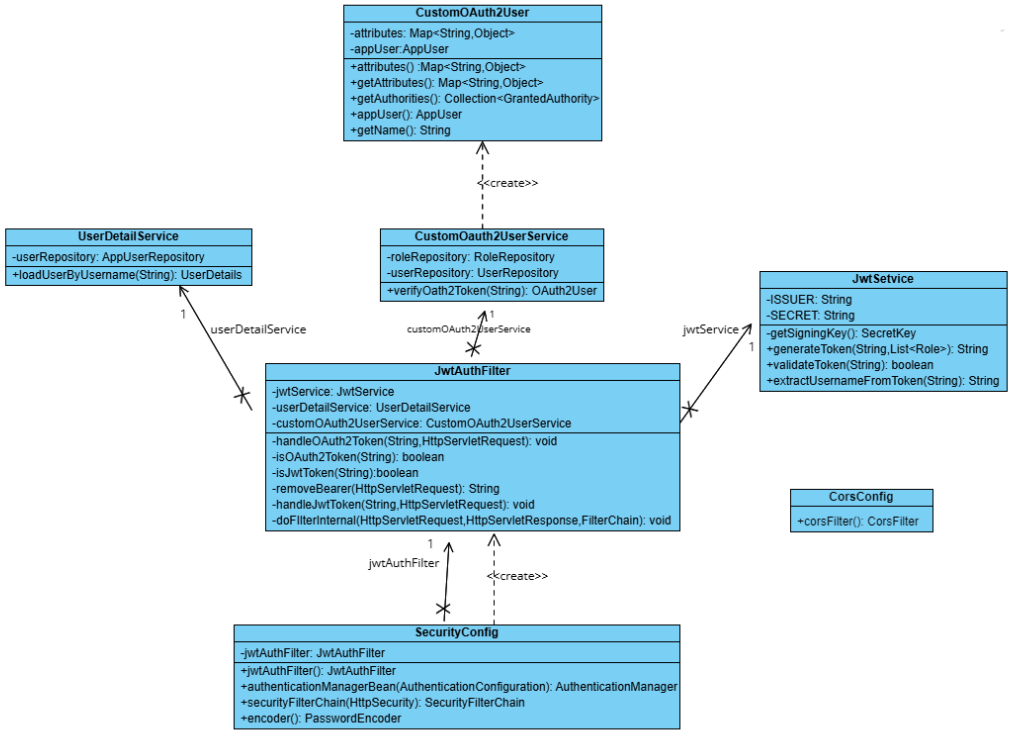
- Ovaj dijagram razreda prikazuje repozitorije za entitete.



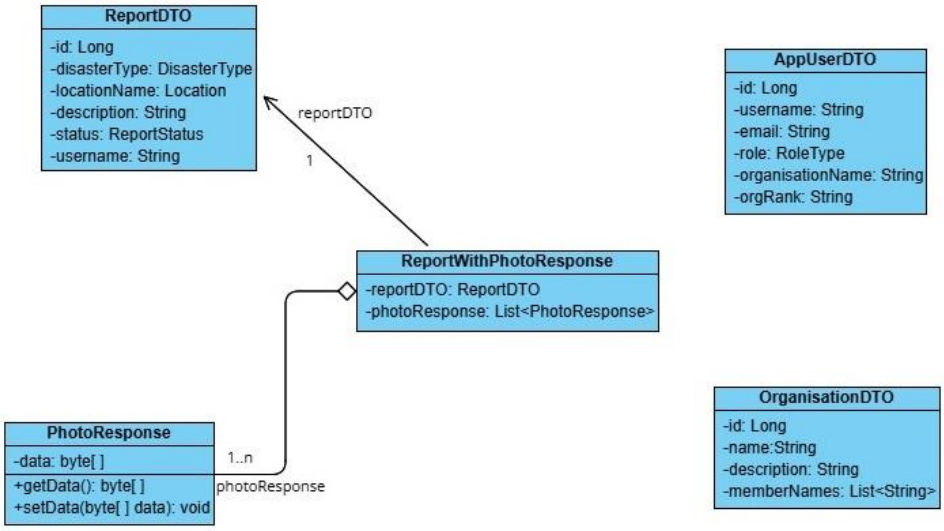
- Ovaj dijagram razreda prikazuje sučelja odnosno servise entiteta te njihove implementacije.



- Ovaj dijagram razreda prikazuje kontrolere za korisnike aplikacije te za prijave nesreća.



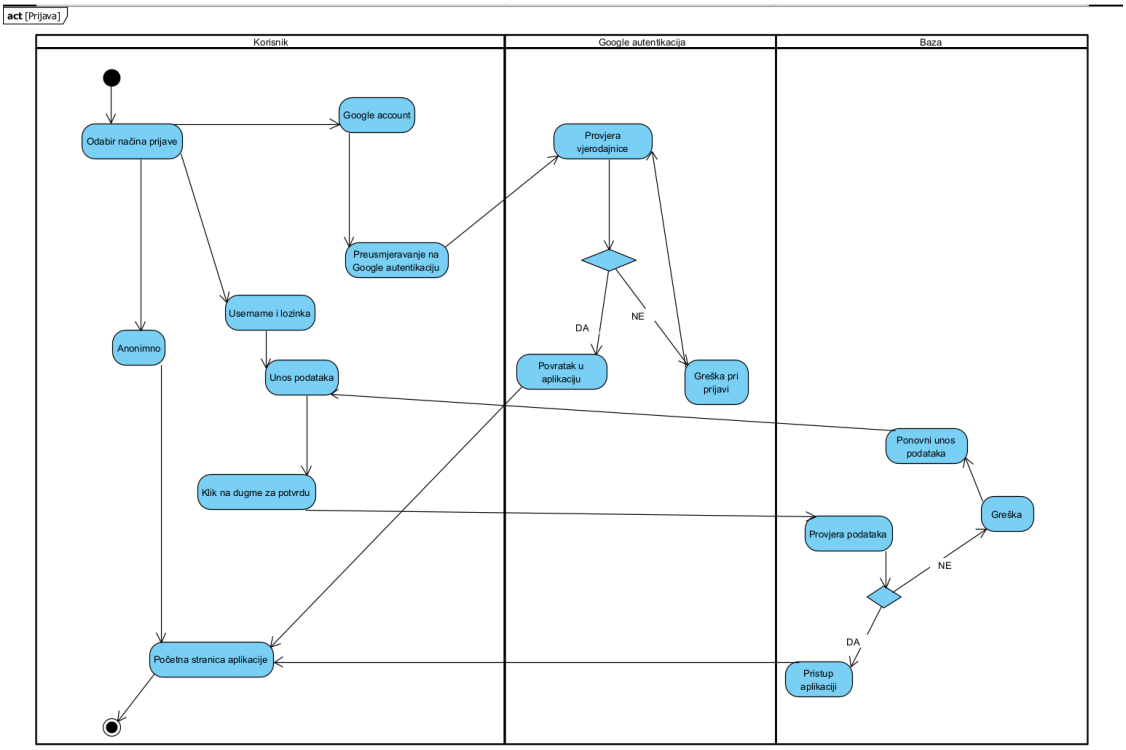
- Ovaj dijagram razreda prikazuje odnose razreda vezane uz sigurnost za prijavu u aplikaciju.



- Ovaj dijagram razreda prikazuje odnose razreda koji se koriste kao DTO (data transfer object), odnosno kao objekti prijenosa podataka.

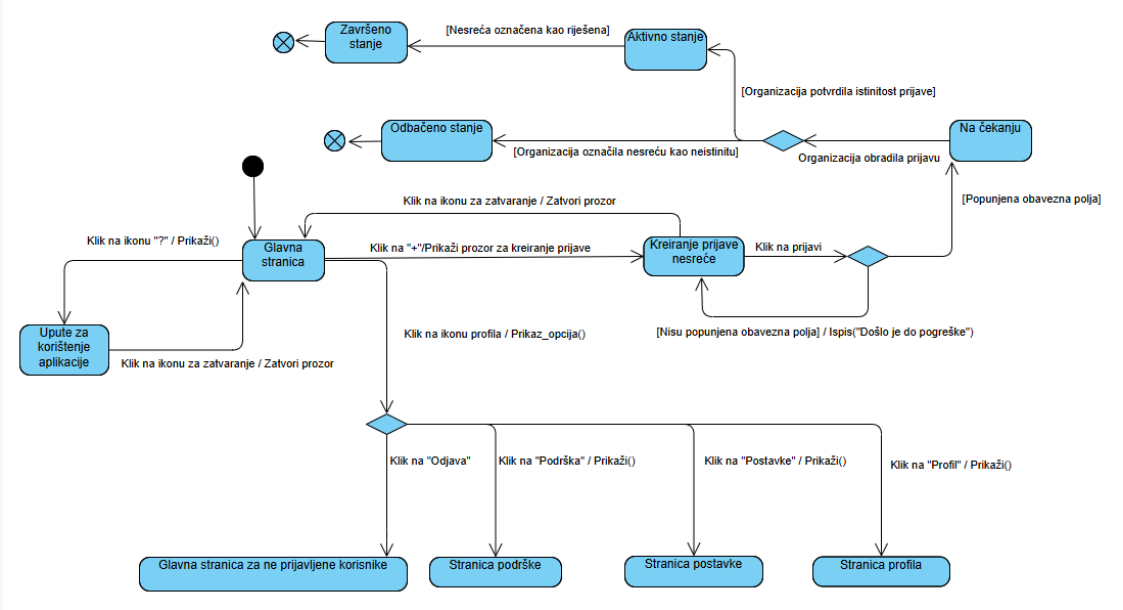
4.4. Dinamičko ponašanje aplikacije

UML Dijagram Aktivnosti



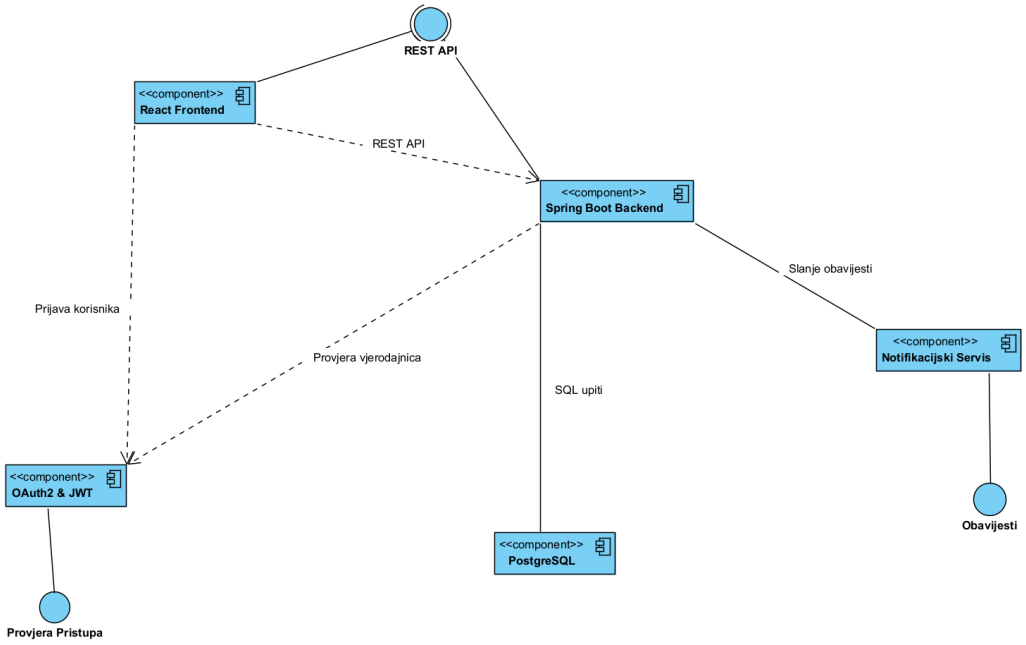
- Dijagram aktivnosti za prijavu korisnika web-aplikacije. Omogućava se pregled web-aplijacije anonimno bez prijave, uz manje ovlasti. Korisnici se mogu prijaviti preko Google računa nakon čega prolaze Google autentifikaciju i nakon što ju prođu vraćeni su u aplikaciju. Također postoji opcija u slučaju već postojećeg računa, unijeti username i lozinku. Uneseni podatci provjeravaju se u bazi podataka i korisniku se omogućava pristup aplikaciji.

UML Dijagram Stanja



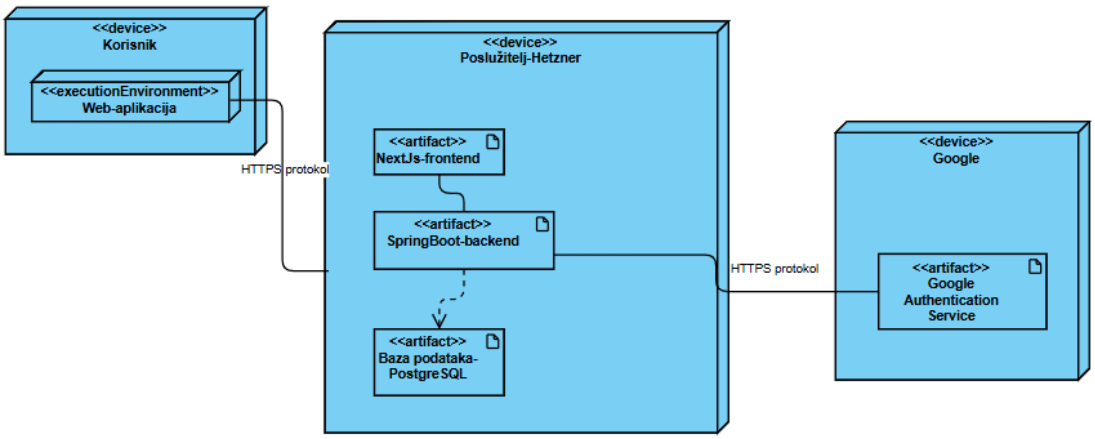
- Dijagram stanja za prijavljene korisnike web-aplikacije. Korisnici mogu prijavljivati nesreće koje kasnije mogu biti potvrđene ili odbijene od strane organizacije. Ako su potvrđene od strane organizacije biti će vidljive na početnoj stranici te će se klikom na njih moći vidjeti njihov opis. Korisnici se isto tako mogu prebaciti na stranice postavka ili odjaviti.

Dijagram komponenata



- React Frontend – Korisničko sučelje koje komunicira s backendom putem REST API-ja.
- Spring Boot Backend – okvir za poslužiteljsku stranu aplikacije
- OAuth2 & JWT – Koristi se za autentifikaciju i autorizaciju korisnika.
- PostgreSQL – Baza podataka koja pohranjuje podatke.
- Notifikacijski Servis – Zadužen za slanje obavijesti korisnicima.

Dijagram razmještaja



- Korisnici pristupaju aplikaciji putem web aplikacije, dok su sve ključne komponente sustava raspoređene na platformi Hetzner. Unutar infrastrukture nalaze se poslužitelji za frontend, backend i bazu podataka. Komunikacija između korisničkih uređaja i poslužitelja odvija se putem HTTPS protokola, čime je osigurana sigurna razmjena podataka. Frontend aplikacija obrađuje zahtjeve korisnika i prosljeđuje ih backend poslužitelju, koji dalje komunicira s bazom podataka radi pohrane ili dohvaćanja informacija. Backend je dodatno povezan s Google poslužiteljem putem HTTPS protokola za autentifikacijske usluge koje pruža Google Authentication Service.

Ispitivanje komponenti

AppUserTest

1. Test : Kreiranje organizacije od strane nepotvrđenog korisnika

1. Funkcionalnost koju testiramo:
- Testira se može li korisnik, koji se nije verificirao, kreirati organizaciju
2. Ispitni slučaj:

- Ulazni podaci: User - kojemu je dodan id = 1L, username = "Test User", i atribut verified = false Organisation - organizacija kojoj je id = 1L, name = "Test org", description = "Description"
- Očekivani rezultati: Očekujemo da će se dogoditi iznimka "403 FORBIDDEN "Owner not verified"".
- Dobiveni rezultati: Uspješno ispitivanje, dogodila se očekivana iznimka.

3. Postupak provođenja ispitivanja: Ispitni slučaj testira mogućnost, odnosno nemogućnost kreiranja organizacije, od strane nepotvrđenog korisnika. Testiranje se obavlja simuliranjem metode principal.getName() i metode appUserRepository.findByUsername(), tako da smo definirali što trebaju vraćati. Nakon toga smo pozvali metodu organisationService.createOrganisation() te smo uhvatili iznimku koju ona treba baciti. Na kraju smo usporedili dobivenu poruku nakon greške i očekivanu poruku u metodi assertEquals(). Metodom verify() smo provjerili da se u organisationRepository nije spremilo ništa, što smo i htjeli.

4. Kod ispitnog slučaja:

```java
@Test
public void testCreateOrg_UnverifiedUser(){

    AppUser user = new AppUser();

    user.setId(1L);
    user.setUsername("Test User");
    user.setVerified(false);

    Organisation organisation = new Organisation();

    organisation.setId(1L);
    organisation.setName("Test org");
    organisation.setDescription("Description");

    Principal principal = mock(Principal.class);
    when(principal.getName()).thenReturn(user.getUsername());
    when(appUserRepository.findByUsername(user.getUsername())).thenReturn(java.util.Optional.of(user));

    Exception exception = assertThrows(org.springframework.web.server.ResponseStatusException.class, () -> {
        organisationService.createOrganisation(organisation, principal.getName());
    });

    assertEquals( expected: "403 FORBIDDEN \"Owner not verified\"", exception.getMessage());

    verify(organisationRepository, never()).save(any(Organisation.class));

}
```

---

2. Test : Zabrana pristupa korisniku (ban) na aplikaciji

1. Funkcionalnost koju testiramo:

- Testira se može li se nekom korisniku zabraniti pristup aplikaciji.

2. Ispitni slučaj:

- Ulazni podaci: User1 - kojemu je dodan id = 1L, username = "User" User1 - kojemu je dodan id = 2L, username = "User getting banned"
- Očekivani rezultati: Očekujemo da će se dogoditi iznimka "501 NOT_IMPLEMENTED "Ban user function not implemented"".
- Dobiveni rezultati: Uspješno ispitivanje, dogodila se očekivana iznimka.

3. Postupak provođenja ispitivanja: Ispitni slučaj testira mogućnost prvog usera, da zabrani pristup aplikaciji drugom useru. Testiranje se obavlja pozivanjem metode appUserService.banUser(), te smo uhvatili iznimku koju ona treba baciti. Ta mogućnost nije implementirana stoga očekujemo poruku 501 NOT IMPLEMENTED. Na kraju uspoređujemo dobivenu poruku nakon greške i očekivanu poruku u metodi assertEquals(). Metodom verify() smo provjerili da se u appUserRepository nije spremilo ništa, kao što smo i htjeli.

4. Kod ispitnog slučaja:

```
@Test
public void testBanUser_unimplementedMethod(){

    AppUser user = new AppUser();

    user.setId(1L);
    user.setUsername("User");

    AppUser userBanned = new AppUser();

    userBanned.setId(2L);
    userBanned.setUsername("User getting banned");

    Principal principal = mock(Principal.class);
    when(principal.getName()).thenReturn(user.getUsername());

    Exception exception = assertThrows(org.springframework.web.server.ResponseStatusException.class, () -> {
        appUserService.banUser(userBanned.getUsername(), principal.getName());
    });

    assertEquals( expected: "501 NOT_IMPLEMENTED \"Ban user function not implemented\"", exception.getMessage());

    verify(appUserRepository, never()).save(any(AppUser.class));
}
```

---

3. Test : Registracija korisnika u aplikaciju

1. Funkcionalnost koju testiramo:

- Testira se može li se korisnik registrirati u aplikaciju

2. Ispitni slučaj:

- Ulazni podaci: User - kojemu pripada username = "New user", password = "Password" i email = "NewUser@mail.com"
- Očekivani rezultati: Očekujemo da će se kod uspješno izvršiti bez ikakvih pogrešaka.
- Dobiveni rezultati: Uspješno ispitivanje, bez ikakve greške.

3. Postupak provođenja ispitivanja: Ispitni slučaj testira mogućnost korisnika da se registrira u aplikaciju unošenjem korisničkog imena, lozinke i email-a. Nakon toga se poziva metoda appUserService.registerUser(user) koja registrira korisnika u sustav. Nakon toga u metodi verify() provjeravamo jesmo li nešto spremili u appUserRepository samo jednom.

4. Kod ispitnog slučaja:

```
@Test
public void testRegistration(){

    AppUser user = new AppUser();
    user.setUsername("New user");
    user.setPassword("Password");
    user.setEmail("NewUser@mail.com");

    appUserService.registerUser(user);
    verify(appUserRepository, times( wantedNumberOfInvocations: 1)).save(any(AppUser.class));
}
```

---

4. Test : Pridruživanje korisnika organizaciji u slučaju da je već pripadnik druge organizacije

1. Funkcionalnost koju testiramo:

- Testira se može li se korisnik pridružiti organizaciji ukoliko je već član druge organizacije.

2. Ispitni slučaj:

- Ulazni podaci: Organization1 - stara organizacija kojoj je name = "OldOrganisation", id = 1L, description = "Old organisation" i lista članova gdje je user Organization2 - nova organizacija kojoj je name = "NewOrganisation", id = 2L, i description = "New organisation" User - kojemu pripada username = "Leaving user", email = "LeavingUser@mail.com", pripadnost organizaciji oldOrganisation te rank volontera
- Očekivani rezultati: Očekujemo da će se dogoditi iznimka "403 FORBIDDEN "User already in an organisation"".
- Dobiveni rezultati: Uspješno ispitivanje, dogodila se očekivana greška.

3. Postupak provođenja ispitivanja: Ispitni slučaj kreira 2 testne organizacije, te daje pripadnost useru "Leaving user" jednoj od njih. Kod simulira ponašanja metoda organisationRepository.findByName() i appUserRepository.findByUsername(). Nakon toga zove metodu appUserService.joinOrg() koja baca grešku. Poruku greške i očekivanu poruku uspoređujemo u metodi assertEquals().

4. Kod ispitnog slučaja:

```java
@Test
public void testJoiningOrganisation_WithBeingInOrganisation() {
    Organisation oldOrganisation = new Organisation();
    oldOrganisation.setName("OldOrganisation");
    oldOrganisation.setId(1L);
    oldOrganisation.setDescription("Old organisation");

    AppUser user = new AppUser();
    user.setUsername("LeavingUser");
    user.setEmail("LeavingUser@mail.com");
    user.setOrganisation(oldOrganisation);
    user.setOrgRank(OrgRank.VOLUNTEER);

    oldOrganisation.setMembers(List.of(user));

    Organisation newOrganisation = new Organisation();
    newOrganisation.setName("NewOrganisation");
    newOrganisation.setId(2L);
    newOrganisation.setDescription("New organisation");


    when(organisationRepository.findByName(anyString())).thenReturn(Optional.of(newOrganisation));
    when(appUserRepository.findByUsername(user.getUsername())).thenReturn(Optional.of(user));

    Exception exception = assertThrows(ResponseStatusException.class, () -> {
        appUserService.joinOrg(newOrganisation.getName(), user.getUsername());
    });

    assertEquals( expected: "403 FORBIDDEN \"User already in an organisation\"", exception.getMessage());

}
```

Izlaz nakon pokretanja klase AppUserTest

- Slika zaslona prikazuje da su sva 4 testa uspješno izvršena. Proces je završio sa "exit code 0", što potvrđuje uspješno izvršavanje testova.

MoreAppUserTest

5. Test : Objavljivanje nesreće unutar granica Hrvatske

1. Funkcionalnost koju testiramo:

- Testira se može li korisnik prijaviti nesreću unutar Hrvatske.

2. Ispitni slučaj:

- Ulazni podaci: User - kojemu pripada username = "ValidUser" Lokacija - koordinate latitude = 45.791780 i longitude = 15.954962 (predstavljaju lokaciju u Zagrebu) Report - kojemu je pridodan user koji objavljuje nesreću, description =

"Earthquake!", tip nesreće EARTHQUAKE (potres), te lokacija na kojoj je nesreća

- Očekivani rezultati: Očekujemo da će se kod normalno izvršiti bez ikakve pogreške.
- Dobiveni rezultati: Uspješno ispitivanje, uspješno je prijavljena nesreća.

3. Postupak provođenja ispitivanja: Ispitni slučaj testira prijavljivanje nesreće. U kodu se simulira metoda locationRepository.findByLatitudeAndLongitude() koja vraća lokaciju. Pozvana je metoda reportService.submitReoport(). Lokacija na kojoj je prijavljena nesreća se nalazi unutar Hrvatske te je uspješno prijavljena. Metoda verify() provjerava da je report spremljen u reportRepository samo jednom, što je očekivano i ispravno ponašanje.
4. Kod ispitnog slučaja:

```
@Test
public void testSubmitReport_InsideCroatia() {

    AppUser user = new AppUser();
    user.setUsername("ValidUser");

    Location location = new Location();
    location.setLatitude(45.791780);  //Zagreb
    location.setLongitude(15.954962);

    Report report = new Report();
    report.setUser(user);
    report.setDescription("Earthquake!");
    report.setDisasterType(DisasterType.EARTHQUAKE);
    report.setLocation(location);

    when(locationRepository.findByLatitudeAndLongitude(anyDouble(), anyDouble())).thenReturn(location);

    reportService.submitReport(report, user.getUsername());

    verify(reportRepository, times( wantedNumberOfInvocations: 1)).save(any(Report.class));
}
```

---

6. Test : Objavljivanje nesreće izvan granica Hrvatske

1. Funkcionalnost koju testiramo:

- Testira se može li korisnik prijaviti nesreću izvan Hrvatske.

2. Ispitni slučaj:

- Ulazni podaci: User - kojemu pripada username = "ValidUser" Lokacija - koordinate latitude = 48.806130 i longitude = 2.210209 (predstavljaju lokaciju u Parizu) Report - kojemu je pridodan user koji objavljuje nesreću, description = "Fire!", tip nesreće Fire (požar), te lokacija na kojoj je nesreća
- Očekivani rezultati: Očekujemo da će se dogoditi pogreška 400 BAD REQUEST s porukom "Location is outside of Croatia".
- Dobiveni rezultati: Uspješno ispitivanje, neuspješno je prijavljena nesreća jer je ona izvan Hrvatske.

3. Postupak provođenja ispitivanja: Ispitni slučaj testira prijavljivanje nesreće. U kodu se simulira metoda locationRepository.findByLatitudeAndLongitude() koja

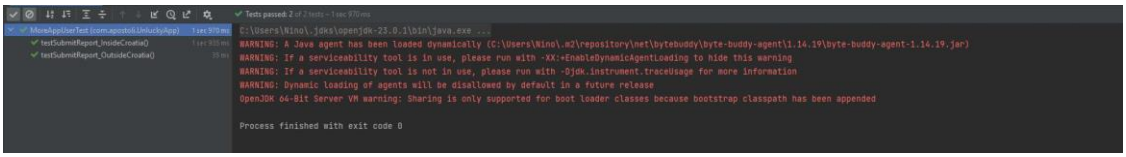
vraća null. Pozvana je metoda reportService.submitReoport(). Lokacija na kojoj je prijavljena nesreća je izvan Hrvatske te je neuspješno prijavljena. Aplikacija prepoznaje lokacije unutar Hrvatske, te u ovom slučaju, lokacija je bila u Parizu pa nije bilo moguće prijaviti nesreću. Dvije metode assertEquals() provjeravaju podudara li se http status greške i očekivani http status, te podudara li se poruka greške. Metoda verify() provjerava da u reportRepository nije ništa spremljeno, što je očekivano i ispravno ponašanje.

4. Kod ispitnog slučaja:

```
@Test
public void testSubmitReport_InsideCroatia() {

    AppUser user = new AppUser();
    user.setUsername("ValidUser");

    Location location = new Location();
    location.setLatitude(45.791780);  //Zagreb
    location.setLongitude(15.954962);

    Report report = new Report();
    report.setUser(user);
    report.setDescription("Earthquake!");
    report.setDisasterType(DisasterType.EARTHQUAKE);
    report.setLocation(location);

    when(locationRepository.findByLatitudeAndLongitude(anyDouble(), anyDouble())).thenReturn(location);

    reportService.submitReport(report, user.getUsername());

    verify(reportRepository, times( wantedNumberOfInvocations: 1)).save(any(Report.class));
}
```

Izlaz nakon pokretanja klase MoreAppUserTest

- Slika zaslona prikazuje da su oba testa uspješno izvršena. Proces je završio sa "exit code 0", što potvrđuje uspješno izvršavanje testova.



Ispitivanje sustava

Testiranje komponente Login

1. **Testirana funkcionalnost:**
    - Prikaz gumba "Prijava".
2. **Detalji ispitnog slučaja:**
    - **Ulazni podaci:** Nema.
    - **Očekivani rezultat:** Gumb s tekstom "Prijava" je prikazan na ekranu.
    - **Postupak ispitivanja:**
        1. Renderirajte komponentu `<Login />`.

2. Koristite `screen.getByText` za provjeru postojanja gumba.
- **Dobiveni rezultat:** Test uspješno prošao.

Testiranje komponente Home

1. **Testirane funkcionalnosti:**
   - Prikaz osnovnih elemenata (npr. naslov, Google karta, footer).
   - Uvjetni prikaz gumba na temelju postojanja tokena u `localStorage`.
   - Ispravna obrada tokena iz `localStorage`.
2. **Primjeri ispitnih slučajeva:**
   - **Prikaz naslova:**
     - **Ulazni podaci:** Nema.
     - **Očekivani rezultat:** Naslov "Nesreće HR" je prikazan.
     - **Postupak ispitivanja:**
       1. Renderirajte komponentu `<Home />`.
       2. Provjerite prikaz naslova pomoću `screen.getByRole`.
   - **Uvjetni prikaz:**
     - **Ulazni podaci:** Token je postavljen u `localStorage`.
     - **Očekivani rezultat:** Prikazuje se gumb "Prijavi nesreću".
     - **Postupak ispitivanja:**
       1. Postavite token u `localStorage`.
       2. Renderirajte komponentu `<Home />`.
       3. Provjerite postojanje gumba pomoću `screen.getByRole`.
   - **Dobiveni rezultati:** Svi testovi su uspješno prošli.

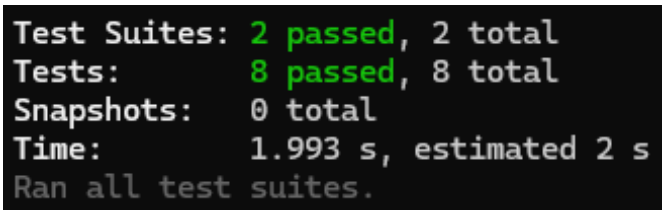
Testiranje prijave putem obrasca

1. **Testirana funkcionalnost:**
   - Slanje obrasca za prijavu.
2. **Detalji ispitnog slučaja:**
   - **Ulazni podaci:**
     - Korisničko ime: "randomUser".
     - Lozinka: "randomPassword".
   - **Očekivani rezultat:**
     - Mock API poziv s točnim parametrima.
     - Token je pohranjen u `localStorage`.
   - **Postupak ispitivanja:**
     1. Renderirajte komponentu `<Login />`.
     2. Simulirajte unos korisničkog imena i lozinke koristeći `fireEvent.change`.
     3. Simulirajte slanje obrasca pomoću `fireEvent.click`.
     4. Provjerite parametre API poziva i pohranu tokena.

- **Dobiveni rezultati:** Test uspješno prošao.

Izlaz nakon pokretanja testova

- Na priloženoj slici je vidljivo da su svi provedeni testovi uspješno prošli provjere.



Korištene tehnologije i alati

1. Programski jezici

- **TypeScript (verzija 5.x)**: odabran za razvoj frontend aplikacije jer omogućava dodavanje tipova u JavaScript, što rezultira stabilnijim i skalabilnijim kodom. Time se smanjuju greške tijekom razvoja i olakšava suradnja unutar tima.
- **Java (verzija 21)**: Java je korištena za backend dio aplikacije zbog svoje pouzdanosti, performansi i bogatog ekosustava alata i biblioteka.

2. Radni okviri i biblioteke

Backend

- **Spring Boot (verzija 3.3.5)**: Spring Boot (verzija 3.3.5) omogućuje brzi razvoj i implementaciju REST API-ja, pružajući snažan okvir za upravljanje poslovnim pravilima i logikom unutar aplikacije.
- **Spring Data JPA**: Omogućava upravljanje podacima i ORM (objektno-relacijsko mapiranje) s bazom podataka.
- **Spring Security**: Koristi se za autentifikaciju i autorizaciju korisnika.
- **JWT (io.jsonwebtoken 0.12.5)**: Koristi se za generiranje i validaciju JSON Web Tokena.
- **Google API Client (verzija 2.7.0)**: Koristi se za integraciju s Google servisima (poput Google OAuth2).
- **Spring Starter Mail (verzija 3.4.1)**: Koristi se za slanje e-mailova iz aplikacije.
- **Swagger (springdoc-openapi-starter-webmvc-ui 2.3.0): Koristi se za automatsko generiranje i testiranje REST API endpoint-a.
- **Lombok (verzija 1.18.34)**: Alat za automatsko generiranje boilerplate koda poput `gettera`, `settera` i `toString()` metoda.

Frontend

- **Next.js (verzija 14.2.15)**: React-based framework koji podržava server-side rendering i static site generation.

- **Tailwind CSS (verzija 3.4.1)**: Koristi se za stiliziranje korisničkog sučelja zbog brzine i jednostavnosti.
- **ShadCN UI**: Koristi se za izradu prilagodljivih i responzivnih UI komponenti.
- **Radix UI**: Biblioteka koja omogućava izradu modernih i pristupačnih elemenata korisničkog sučelja.

3. Baza podataka

- **PostgreSQL**: Relacijska baza podataka korištena za pohranu i upravljanje podacima aplikacije.

4. Razvojni alati

- **IntelliJ IDEA**: Preporučeni IDE za razvoj backend aplikacija.
- **Visual Studio Code**: Koristi se za razvoj i uređivanje frontend dijela aplikacije.
- **Postman**: Alat za testiranje REST API zahtjeva.
- **Maven**: Upravljanje ovisnostima i proces građenja backend projekta.

5. Alati za ispitivanje

- **Git (verzija 2.45.1.windows.1)**: Alat za lokalno upravljanje verzijama koda koji omogućava jednostavno praćenje promjena u projektu. Podržava grananje (branching) i spajanje (merging), čime se olakšava paralelan razvoj različitih komponenti.
- **GitHub**: Online platforma za pohranu projekata koji koriste Git. Omogućava suradnju timova kroz pushanje (slanje) i pullanje (dohvaćanje) koda između lokalnih i udaljenih repozitorija te osigurava strukturiran pristup projektu.

6. Alati za razmještaj

- **Coolify** je jednostavan alat za razmještaj aplikacija na vlastitim serverima, idealan za one koji žele self-hosted rješenje s podrškom za Docker i moderne tehnologije. Korisnici ga biraju zbog automatiziranog razmještaja, CI/CD funkcionalnosti, lakog upravljanja bazama podataka i otvorenog koda, što pruža fleksibilnost i kontrolu uz smanjenje troškova.

7. Cloud platforma

- **Hetzner** je njemačka hosting platforma poznata po povoljnim cijenama, pouzdanim serverima i cloud uslugama. Idealna je za skalabilne projekte i nudi data centre u Europi s odličnim performansama.

Preduvjeti

- Node.js 16 ili noviji (za React i klijentski dio)
- Java Development Kit (JDK) 11 ili noviji (za Spring Boot backend)
- PostgreSQL 14 ili noviji (za bazu podataka)
- Git (za upravljanje kodom)
- Docker 20.10+ (opcionalno)

Preuzimanje izvornog koda

Izvorni kod aplikacije moguće je preuzeti kloniranjem Git repozitorija:

```
git clone git@github.com:MarkoPekasFER/apostoli.git
cd apostoli
```

Pokretanje aplikacije

1. Postavljanje baze podataka (PostgreSQL)

1. Pokreni PostgreSQL servis
2. Kreiraj bazu i korisnika:

```
CREATE DATABASE apostoli;
CREATE USER apostoli_user WITH PASSWORD 'tvoja_lozinka';
GRANT ALL PRIVILEGES ON DATABASE apostoli TO apostoli_user;
```

2. Pokretanje backenda

```
cd backend
# Za Maven projekt:
./mvnw spring-boot:run
```

Aplikacija će biti dostupna na `http://localhost:8080`

4. Pokretanje Next.js frontenda

```
cd frontend
npm install       # Instaliraj dependencies (prvi put)
npm run dev
```

Frontend će biti dostupan na `http://localhost:3000`

Troubleshooting

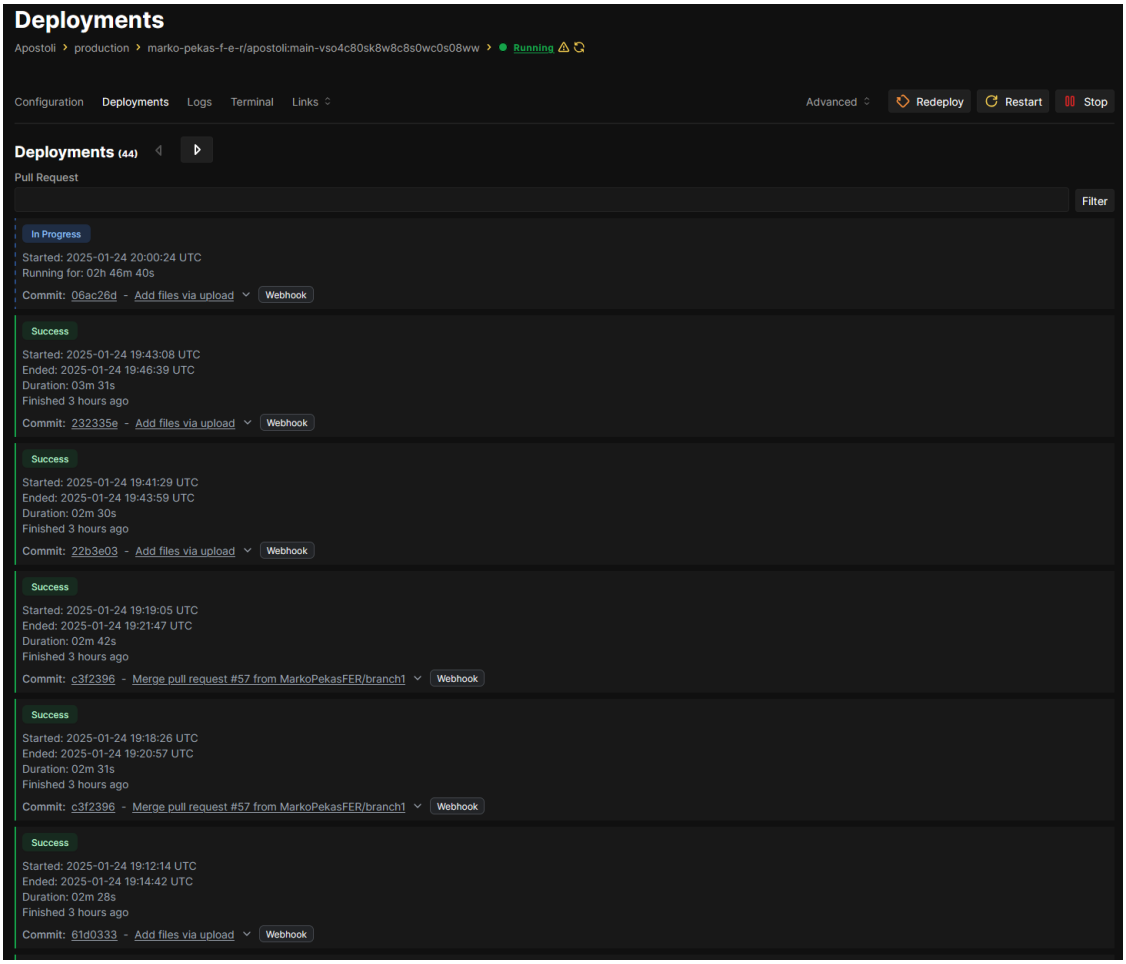
- **"Port already in use"**: Promjeni port u `application.properties` (npr. `server.port=8081`)
- **Connection refused to PostgreSQL**: Provjeri da li je PostgreSQL servis aktivan
- **NPM dependency greške**: Obriši `node_modules` i ponovi `npm install`
- **Java verzija**: Provjeri `java -version` i osiguraj JDK 11+

Za dodatnu pomoć kontaktiraj developera na marko.pekas@fer.hr

---

Deployment u produkciji

Kako bi smo ustvarili continuous integration postavili smo coolify na Hetzner server i preko njega povezali github.



ZAKLJUČAK

- zadatak ove grupe bio je napraviti aplikaciju za prijavljivanje nesreća i nepogoda koja bi omogućila brzo i jednostavno korištenje korisnicima i adminima. Vjerujemo da smo u tome i uspjeli

Prvi ciklus

- u prvom ciklusu jedan od glavnih problema bio je odlučiti kako pristupiti izradi aplikacije. Drugim riječima, odrediti bitne funkcionalne i nefunkcionalne zahtjeve te postaviti realne deadline-e u kojima bi komponente projekta trebale biti gotove. Naš se tim podijelio u 3 dijela. Backend dio tima pobrinuo se za dizanje stranice, autentifikaciju te ispravnu povezanost s bazom podataka. Frontend se pobrinuo za dizajn stranice (koji je morao biti jednostavan i intuitivan za korištenje), povezivanje s backendom kao i dio funkcionalnosti prijavljivanja nesreći.

Dokumentacija se pobrinula da su zapisani svi dijagrami, funkcionalni i nefunkcionalni zahtjevi, ali i pristup i smjer razvijanja projekta.

- u ovom dijelu projekta najveći problem predstavilo je uspješno implementiranje autentifikacije, no uz dodatni angažman i dogovor tima, obavljen je uspješno i na vrijeme. Glavni razlog tome bio je kontinuirani rad i pravovremeno rješavanje drugih zadataka zbog kojih je bilo moguće imati dovoljno vremena viška za rješavanje problema i zaostataka.

Drugi ciklus

- u drugom ciklusu su postali očiti neki bugovi i nedostatci u pristupu koje je bilo potrebno ispraviti, ali i dodati neke nove funkcionalnosti poput omogućavanja anonimnog pristupa aplikaciji. Uz to je bilo potrebno i dovršiti dokumentaciju te se pobrinuti da su svi planirani dijelovi aplikacije gotovi na vrijeme. Uz redovne sastanke tima i dobru komunikaciju članova uspjeli smo napraviti zadano.

- svi članovi tima ovim projektom upoznali su se s raznim alatima i načinima rada s kojima se do sada nisu susreli, ali najvažnije je bilo steći iskustvo i vještine rada na projektu kao dio većeg tima. Prije svega radi kasnijeg rada u većim timovima, na puno većim projektima u struci gdje će te vještine biti još bitnije za kvalitetan proizvod nego na manjem projektu poput ovog.

- za unaprjeđenje budućeg rada tima i brži razvoj projekta najvažniji dio je uspješna komunikacija između svih članova tima, kako bi se smanjila konfuzija i povećala učinkovitost i brzina kod implementacije promjena.

Reference i literatura koja nam je pomogla pri ostvarivanju projekta:

1. Programsko inženjerstvo, FER ZEMRIS: http://www.fer.hr/predmet/proinz

Koristili smo koncepte i principe naučene na predmetu za planiranje i implementaciju projekta.

2. SpringBoot: https://spring.io/projects/spring-boot

Framework za ubrzani razvoj backend aplikacija s integracijom sigurnosti, JPA i web servisa.

3. Postgre: https://www.postgresql.org

Relacijska baza podataka korištena za pohranu podataka o prirodnim nepogodama.

4. Postgre hosting: https://neon.tech/docs/reference/api-reference

Cloud hosting baza podataka koji omogućuje jednostavan pristup i skalabilnost.

5. React: https://react.dev

Frontend JavaScript biblioteka korištena za izradu korisničkog sučelja.

6. Tailwind: https://tailwindcss.com

CSS framework koji omogućuje brzo i prilagodljivo stiliziranje korisničkog sučelja.

7. Google API: https://developers.google.com/identity/protocols/oauth2

Korišten za OAuth2 autentifikaciju korisnika putem Google računa.

8. Visual Paradigm: https://www.visual-paradigm.com

Alat za dizajn UML dijagrama koji smo koristili za planiranje arhitekture aplikacije.

9. Spring MVC: https://www.javatpoint.com/spring-mvc-tutorial

Model-View-Controller arhitektura za organizaciju backend aplikacije.

10. Notion: https://www.notion.so

Alat za upravljanje projektima i dokumentaciju tijekom razvoja.

11. World Cities Database: https://simplemaps.com/data/world-cities

Korištena za informacije o gradovima, poput koordinata, pri izradi sustava obavijesti.

12. Postman: https://www.postman.com

Alat za testiranje REST API-ja kako bismo osigurali ispravnu funkcionalnost backend servisa.

13. Maven: https://maven.apache.org/

Upravljanje ovisnostima i gradnja projekta putem konfiguracije u pom.xml.

14. IntelliJ IDEA: https://www.jetbrains.com/idea/

IDE korišten za razvoj backend aplikacije zbog podrške za SpringBoot i Java.

15. OpenAPI Documentation (Swagger): https://swagger.io/

Omogućuje automatsku dokumentaciju i testiranje API endpointova.

16. Springdoc OpenAPI: https://springdoc.org/

Korišten za integraciju Swagger sučelja s aplikacijom baziranom na Spring Bootu.

17. JWT Documentation: https://github.com/jwtk/jjwt

Dokumentacija korištena za implementaciju sigurnosti putem JSON Web Tokena.

18. Lombok: https://projectlombok.org/

Alat za smanjenje boilerplate koda (npr. getter/setter metode) u Java klasama.

19. Hibernate Documentation: https://hibernate.org/

Referenca za korištenje JPA implementacije za rad s bazom podataka.

Dnevnik sastajanja

1. sastanak
- Datum: 14. listopada 2024.
- Prisustvovali: Pekas Marko, Kresović David, Gretić Vanja, Karuc Dominik, Rajčević Fran, Salai Nino, Zorić Vito
- Teme sastanka:
    - upoznavanje članova
    - podjela na backend i frontend
    - odabir tehnologija i alata
2. sastanak
- Datum: 19. listopada 2024.
- Prisustvovali: Pekas Marko, Kresović David, Gretić Vanja, Karuc Dominik, Rajčević Fran, Salai Nino, Zorić Vito
- Teme sastanka:
    - uspostavljena Discord grupa članova tima
    - definiranje funkcionalnih i nefukcionalnih zahtjeva
3. sastanak
- Datum: 26. listopada 2024.
- Prisustvovali: Pekas Marko, Kresović David, Gretić Vanja, Karuc Dominik, Rajčević Fran, Salai Nino, Zorić Vito
- Teme sastanka:
    - razrada i kreiranje UML dijagrama
    - dizajniranje baze podataka
4. sastanak
- Datum: 3. studenog 2024.
- Prisustvovali: Pekas Marko, Kresović David, Gretić Vanja, Karuc Dominik, Rajčević Fran, Salai Nino, Zorić Vito
- Teme sastanka:
    - raspodjela uloga za implementaciju karte i logina
    - raspodjela uloga za kreiranje dokumentacije
5. sastanak
- Datum: 9. studenog 2024.
- Prisustvovali: Pekas Marko, Kresović David, Gretić Vanja, Karuc Dominik, Rajčević Fran, Salai Nino, Zorić Vito
- Teme sastanka:
    - pregled dosadašnjeg napretka
    - definiranje nedostataka
6. sastanak
- Datum: 12. studenog 2024.
- Prisustvovali: Pekas Marko, Kresović David, Gretić Vanja, Karuc Dominik, Rajčević Fran, Salai Nino, Zorić Vito

- Teme sastanka:
    - dovršavanje prve faze projekta

7. sastanak

- Datum: 8. siječnja 2025.
- Prisustvovali: Pekas Marko, Kresović David, Gretić Vanja, Karuc Dominik, Rajčević Fran, Salai Nino, Zorić Vito
- Teme sastanka:
    - pregled nedostataka projekta

8. sastanak

- Datum: 13. siječnja 2025.
- Prisustvovali: Pekas Marko, Kresović David, Gretić Vanja, Karuc Dominik, Rajčević Fran, Salai Nino, Zorić Vito
- Teme sastanka:
    - raspodjela zadataka po članovima

9. sastanak

- Datum: 16. siječnja 2025.
- Prisustvovali: Pekas Marko, Kresović David, Gretić Vanja, Karuc Dominik, Rajčević Fran, Salai Nino, Zorić Vito
- Teme sastanka:
    - pregled dovršenog posla

10. sastanak

- Datum: 20. siječnja 2025.
- Prisustvovali: Pekas Marko, Kresović David, Gretić Vanja, Karuc Dominik, Rajčević Fran, Salai Nino, Zorić Vito
- Teme sastanka:
    - dovršavanje pozadinskog koda i dokumentacije

11. sastanak

- Datum: 24. siječnja 2025.
- Prisustvovali: Pekas Marko, Kresović David, Gretić Vanja, Karuc Dominik, Rajčević Fran, Salai Nino, Zorić Vito
- Teme sastanka:
    - testiranje
    - završavanje projekta

Tablica aktivnosti

Aktivnost	Pekas Marko	Kresović David	Gretić Vanja	Karuc Dominik	Rajčević Fran	Salai Nino	Zorić Vito
Upravljanje projektom	4h			2h		1h	
Opis projektnog zadatka	1h		3h				

Aktivnost	Pekas Marko	Kresović David	Gretić Vanja	Karuc Dominik	Rajčević Fran	Salai Nino	Zorić Vito
Funkcionalni zahtjevi	1h		2h	2h	2h	3h	
Opis pojedinih obrazaca		4h	30min				
Dijagram obrazaca					3h	2h	6h
Sekvencijski dijagrami			3h		4h	3h	
Opis ostalih zahtjeva			30min		2h	2h	
Arhitektura i dizajn sustava	3h	3h		3h		2h	
Baza podataka	1h		3h	4h			
Dijagram razreda				3h		6h	
Dijagram stanja							
Dijagram aktivnosti		1h					
Dijagram komponenti		1h					
Korištene tehnologije i alati	1h	3h		8h		3h	3h
Ispitivanje i testiranje programskog rješenja				15h		7h	
Upute za puštanje u pogon	3h			30min			
Dnevnik sastajanja						30min	
izrada aplikacije	5h	2h		60h	5h	10h	
izrada početne stranice	3h				1h		
izrada baze podataka			3h	2h			
spajanje s bazom podataka				1h			
izrada prezentacije	1h						8h