Progresivna web aplikacija za upravljanje vatrogasnim društvom

Domić, Filip

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Pula / Sveučilište Jurja Dobrile u Puli**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:137:798680

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

FILIP DOMIĆ

Progresivna web aplikacija za upravljanje vatrogasnim društvom

Diplomski rad

Pula, 2022.

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

FILIP DOMIĆ

Progresivna web aplikacija za upravljanje vatrogasnim društvom

Diplomski rad

JMBAG: 0303063538, redovni studij
Studijski smjer: Informatika

Predmet: Napredne algoritmi i strukture podataka
Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske i komunikacijske znanosti
Znanstvena grana: Informacijski sustavi i informatologija
Mentor: izv. prof. dr. sc. Tihomir Orehovački

Pula, rujan 2022.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani FILIP DOMIĆ, kandidat za magistra INFORMATIKE ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

U Puli, rujan, 2022 godine

IZJAVA

o korištenju autorskog

djela

Ja, FILIP DOMIĆ dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom Progresivna web aplikacija za upravljanje vatrogasnim društvom koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu diplomskih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama. Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

Potpis

U Puli, rujan 2022 godine

SADRŽAJ

1. Uvod

U današnje vrijeme je sve veći naglasak na digitalizaciji poslovanja, ne u svrhu same digitalizacije, nego u svrhu pojednostavljivanja i ubrzavanja samih radnih procesa. Dobrovoljna vatrogasna društva (DVD) danas imaju sve veću ulogu u javnim djelatnostima, te se iskazuje potreba za lakšim praćenjem događanja unutar samog DVD-a. Trenutno stanje društva gdje su članovi DVD-a, ponajviše predsjednik samog društva, imali aplikaciju koja bilježi samo određene stvari, ali je na razini države pa je jako nepregledna. Pokazala se potreba za izradom aplikacije koja će pomoći u boljem kontroliranju i bilježenju njima potrebnih stvari. Motivacija za pisanje ovog rada je nastala iz „žalbi“ predsjednika DVD-a, koji je zaista naglasio da im je potrebno osmisliti nešto bolje, kako bi i oni kao društvo bolje doprinosili svojoj zajednici.

Ideja koja se zasigurno u budućnosti može implementirati u svaki DVD zasebno, kako bi organizacija svakog društva bila na boljoj razini. Predsjednik društva prikazao je sve dosadašnje probleme koje imaju s bilježenjem podataka. Kako bi mu se pomoglo napravljena je progresivna web aplikacija (PWA) koja će upravljati sa svim događanjima unutar društva. Ova aplikacija biti će prilagođena za web preglednike na računalu i na mobilnim uređajima, ali također će se moći instalirati kao aplikacija na samom računalu i mobilnom uređaju. Unutar aplikacije, vodstvo DVD-a će imati uvid u sve korisnike društva, te će postojati uvid u opremu koju korisnici posjeduju. Do sada je bio veliki problem, radi velikog broja korisnika, imati uvid kojem vatrogascu je potreba nova oprema ili mu nešto od opreme nedostaje, iz razloga što postojeća aplikacija ne bilježi ništa navedeno. Sljedeća funkcionalnost koja će biti u aplikaciji je upravljanje skladištem. Unutar skladišta će biti zabilježena sva oprema i tako će društvo moći lakše raspoređivati rezervnu opremu vatrogascima i vozilima. Svaki artikl u društvu će dobiti svoj QR kod kojem će se moći pomoću aplikacije vrlo lako utvrditi svi detalji o pojedinom artiklu unutar vatrogasnog društva. Također će postojati bilježenje svih vozila, unutar kojeg će se imati uvid kada je vozilo potrebno odvesti na redovni servis ili kada je potrebno napraviti registraciju. Biti će dodan gumb za uzbunjivanje pomoću kojeg će svim korisnicima postrojbe biti poslana poruka o nekom požaru, te će nakon završetka

požara biti zabilježene sve aktivnosti. Detaljnije objašnjen postupak rada s aplikacijom i njene mogućnosti nalaze se u nastavku rada.

2. PWA koncept

Progresivna web aplikacija (PWA) je koncept razvoja aplikacija koji želi postići da se aplikacije koje su napisane za web preglednik mogu koristiti na uređajima (mobilnim i stolnim) na isti način na koji su korisnici navikli koristiti native aplikacije. Naziv PWA je osmislio Google 2015. godine [16], iako je sam koncept kao takav postojao puno prije. Na svojim stranicama Google se hvali da su to moderne aplikacije koje rade offline i koje su okrenute mobilnim uređajima te se mogu uspoređivati s najboljim nativnim aplikacijama. PWA možemo protumačiti kao normalnu web aplikaciju koju možemo instalirati na uređaj, ali i ne moramo, te je možemo koristiti bez interneta.

PWA koncept, dolaskom na tržište, donio je nešto novo, iz razloga što jako dobro nadopunjuje dosadašnji vrlo popularni koncept responzivnog web dizajna. Kada je responzivni koncept došao na tržište jako je digao očekivanja korisnika mobilnih uređaja. Razlog tome je što su se korisnici navikli da je sve prilagođeno uređaju i da nema više znatne razlike između pretrage na mobitelu ili računalu. No svima je jasno da responzivna web stranica, nije ni blizu mobilnoj aplikaciji, a razvoj nativne mobilne aplikacije zna biti izrazito skupo. U toj cijeloj priči Google se dosjetio da izbacili na tržište neku hibrid kombinaciju. Tako je stvoren savršeni hibrid nativne aplikacije i mobilne web aplikacije.

PWA aplikaciju možemo prepoznati vrlo jednostavno. Kada na uređaju otvorimo stranicu koja je PWA, preglednik će nam sugerirati ikonicom (koja ovisi o implementaciji samog preglednika) da je ova aplikacija dostupna za korištenje izvan preglednika. Klikom na ikonu za instalaciju otvara nam se poruka *“Želite li instalirati web aplikaciju na svoj početni zaslon?”*. Od tog trenutka aplikacije se može koristiti samostalno bez otvaranja preglednika.

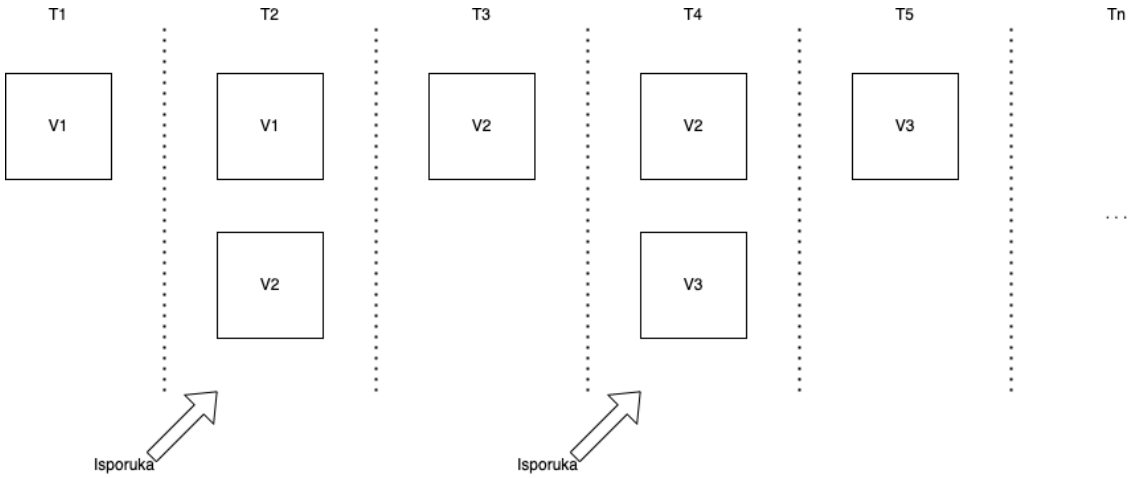
Iako se PWA na prvu čini kao odličan koncept kojeg bi svi trebali implementirati, ima tu i nekih negativnih strana, a to je da PWA nije podržan od strane svih preglednika. Svega 74% modernijih preglednika sposobno je koristiti ovaj koncept [16].

Osim samog mobile native koncepta, potrebno je naglasiti i ostale funkcionalnosti koje nam PWA donosi, a to su: rad u offline modu, slanje push-notifikacija, otvaranja aplikacije u full screen modu, 60 fps animacije i mnoge druge.

Svakako valja naglasiti možda jednu od najjačih činjenica o PWA, a to da je framework agnostik, te kao takav se lagano može iskoristiti u gotovo svakom razvojnom okuženju.

2.1. PWA koncept učitavanja

Budući da se PWA aplikacija instalira na korisnički uređaj, bitno je sagledati njezin ciklus korištenja i isporuka. Kako bi to bolje predočili napravljen je sljedeći dijagram koji je prikazan na slici 1.



Slika 1. Dijagram isporuka i instalacija PWA aplikacije

Na dijagramu se vidi da kada korisnik prvi puta otvori aplikaciju (*T1*) uslijed čega se dohvaća prva verzija aplikacije sa servera, te se ona instalira na korisnički uređaj i odmah je spremna za korištenje, prikazano na slici 1 u polju T1.

➔ *U nekom trenutku isporučena je nova verzija aplikacije*

Korisnik otvara aplikaciju i odmah je može koristiti (ne čeka na učitavanja). U pozadini sustav provjerava da li je dostupna nova verzija aplikacije, te ako

postoji, preuzima ju na uređaj, instalira, te je priprema da bude spremna za pokretanje, prikazano na slici 1 u polju T2.

➔ *Korisnik koristi staru verziju aplikacija, nova je instalirana u pozadini i bit će pokrenuta kod sljedećeg pokretanja*

Po učitavanju aplikacije korisniku se dohvaća zadnja instalirana verzija, prikazano na slici 1 u polju T3.

3. Analiza sustava

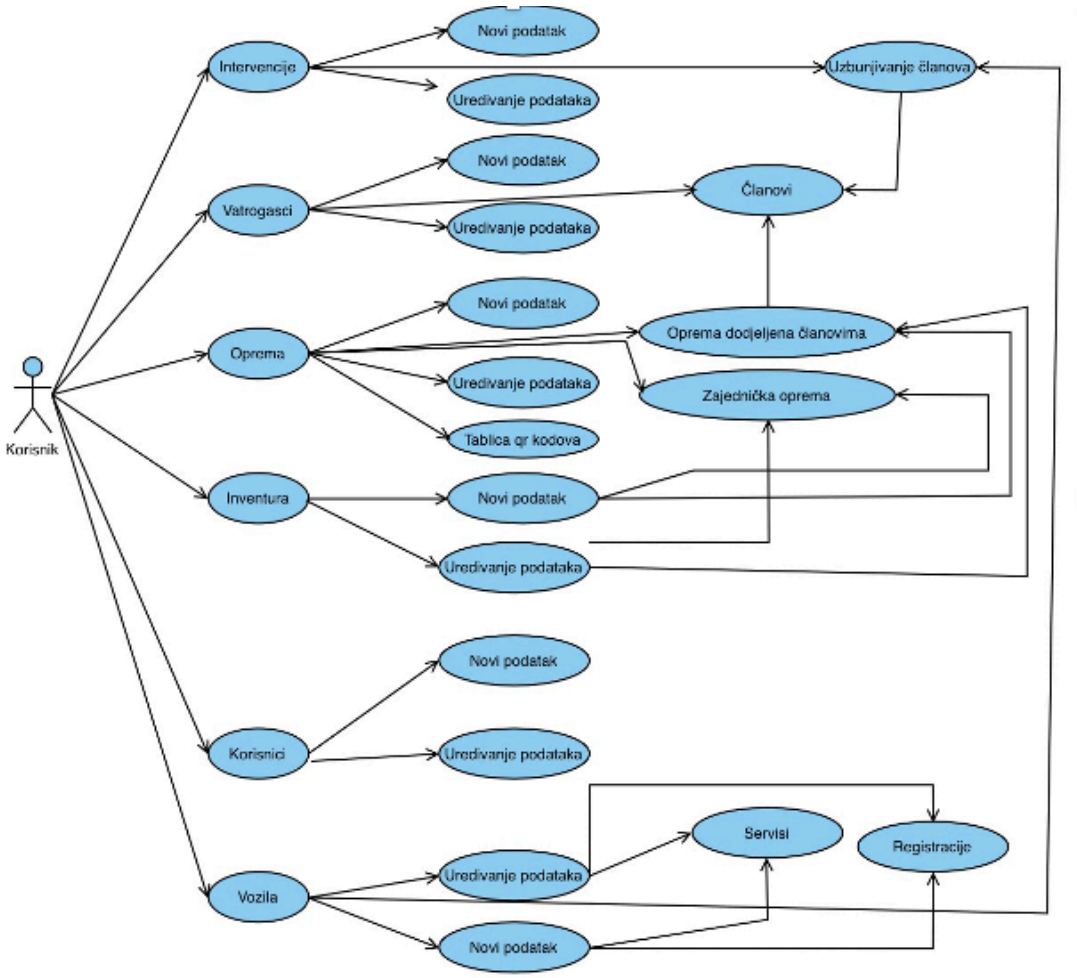
Kako bi bolje shvatili potrebe sustava napravljena je analiza zahtjeva korisnika. Analiza je obavljena korištenjem sljedećih tehnika:

- Analiza korisničkih zahtjeva (use-case)
- Analiza komponenti sustava
- Analiza uloga u sustavu
- Analiza relacijskog modela
- Analiza interakcija u sustavu

3.1. Analiza korisničkih zahtjeva (use-case)

Prva analiza koja je napravljena je prikupljanje korisničkih zahtjeva. Korisnik identificira da su mu u sustavu potrebne sljedeće stvari: evidencija vatrogasaca i opreme, inventura, upravljanje vozilima, te vođenje intervencija. Kako bi lakše predočili korisničke zahtjeve, napravljen je dijagram korisničkih zahtijeva koji je prikazan na slici 2.

Dijagram prikazuje podsustave koje će sustav imati, te koje operacije će biti dostupne. Iz ove analize smo dobili predodžbu što sve sustav treba raditi.
Na dijagramu se vidi da korisnik u sustavu treba imati mogućnost da kreira intervencije i da preko intervencija napravi obavještavanje vatrogasaca. Kako bi mogao dodavati vatrogasce na intervencije, korisnik treba imati mogućnost da kreira i uređuje šifarnike vatrogasaca. Drugi dio sustava je kontrola resursa koji će se raditi kroz inventure. Šifarnik, koji je korisnicima bitan za inventure je šifarnik opreme. Korisnik također treba imati mogućnost upravljanja svojim voznim parkom koji se sastoji do upravljanja vozilima, servisima i registracijama tih vozila. Kako bi sustavi mogao ispravno funkcionirati, korisnik će trebati imati mogućnost upravljanjem korisnika u sustavu.

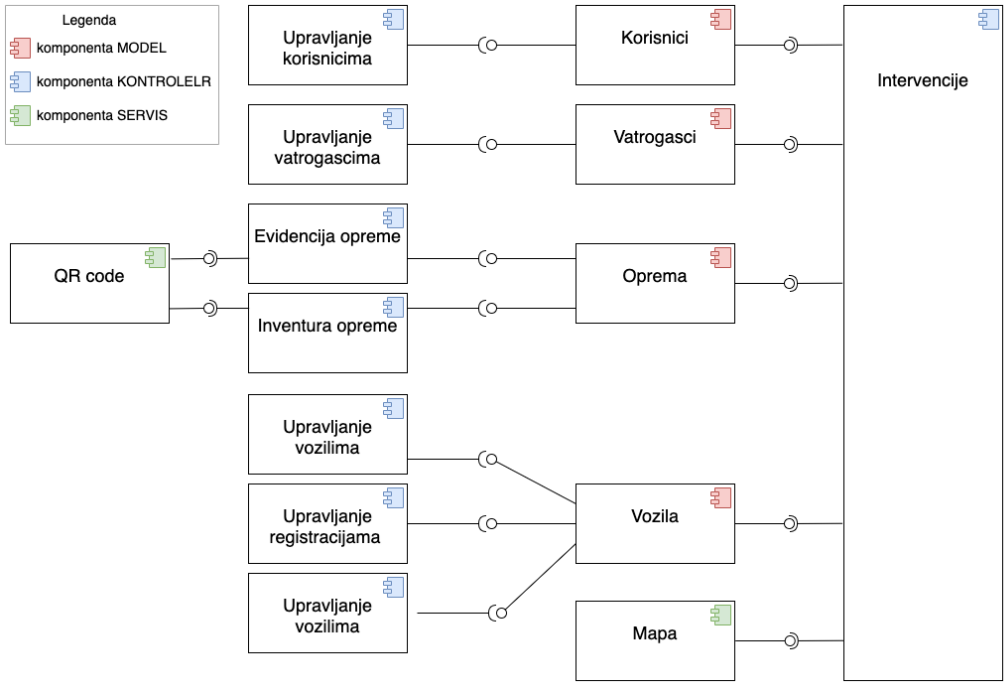


Slika 2. Use-case dijagram aplikacije

3.2. Analiza komponenti sustava

Nakon što su poznati korisnički zahtjevi može se izraditi plan komponenti u sustavu. Na dijagramu ispod možemo vidjeti da se on sastoji od 3 vrste komponenti: crveno – komponente modeli predstavljaju opisnik jednog entiteta, plavo – komponente kontroleri koji predstavljaju koordinatore podataka (prikaz podataka na ekranu, obrada i spremanje), te zeleno – komponente servisi koji su zaslužni za obradu podataka.

Na dijagramu možemo vidjeti da je najveća komponenta u sustavu Intervencije, što je i očekivano iz razloga što je to komponenta koja odrađuje najkritičnije zadatke. Ostale komponente poput korisnika, vatrogasaca i opreme predstavljaju neke oblike šifrarnika, dok za opremu osim evidencija imamo i opciju revizija putem inventura. Dijagram komponenti sustava prikazan je na slici 3.



Slika 3. Dijagram komponenti sustava

3.3. Analiza uloga u sustavu

Aplikacija je namijenjena za više različitih korisničkih uloga. Kako bi lakše izradili aplikaciju prilagođenu svakoj ulozi, napravljena je analiza korisničkih uloga. Ovlasti u aplikaciji prikazani su u tablici 1.

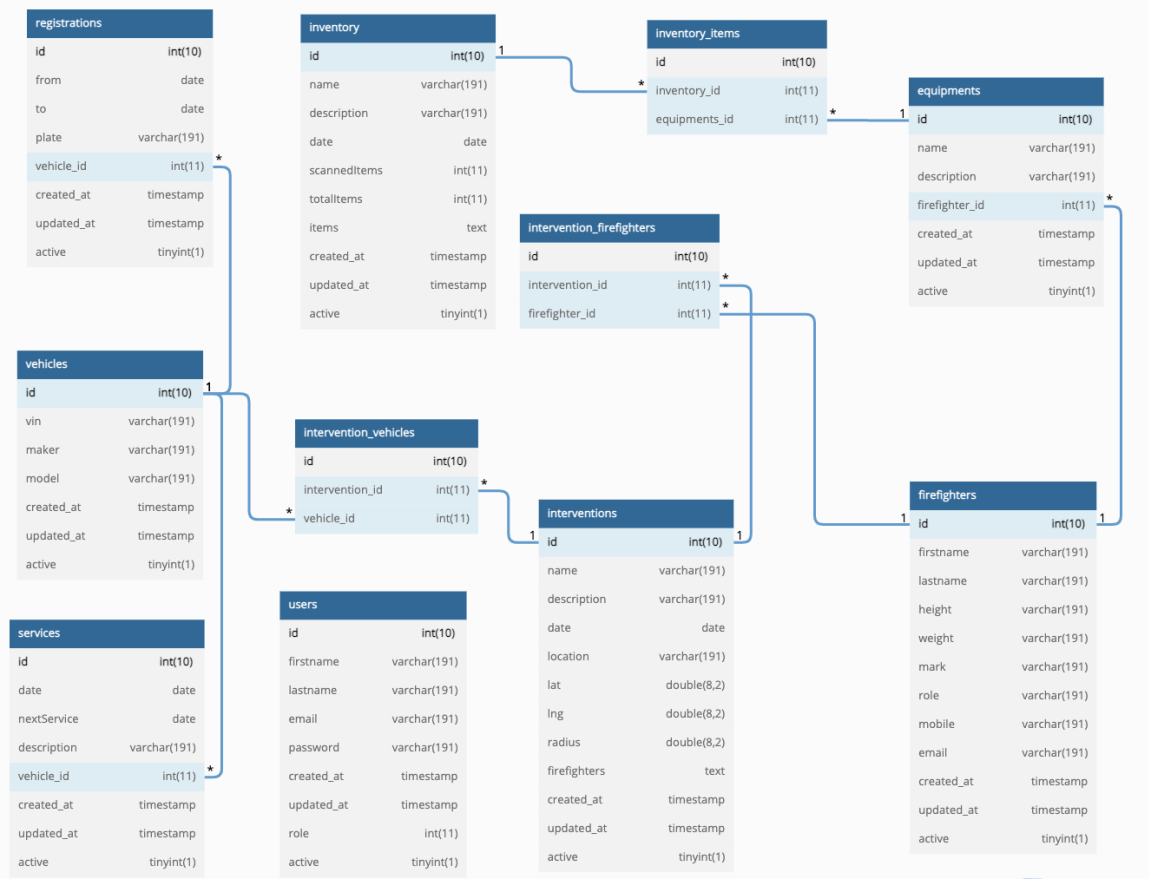
	Intervencije	Vatrogasci	Vozila	Oprema	Inventura	Korisnici
Administrator						
Predsjednik						
Skladištar						
Vatrogasac						

Tablica 1. Tablica ovlasti u aplikaciji

Kao što se može vidjeti u tablici, administrator je uloga koja može raditi apsolutno sve. Predsjednik je osoba s visokim ovlastima koja ima pregled nad skoro svim komponentama sustava, dok su skladištari i vatrogasci usko specijalizirane uloge koje vide samo one stvari koje su njima potrebne za svakodnevni rad.

3.4. Analiza relacijskog modela

Prije implementacije aplikacije je potrebno definirati podatkovni model. Čitanjem dijagrama sustava jednostavno dolazimo do podatkovnog modela sustava. U samom modelu možemo vidjeti koji će se sve podatci obrađivati unutar aplikacije. Relacijski model prikazan je na slici 4.

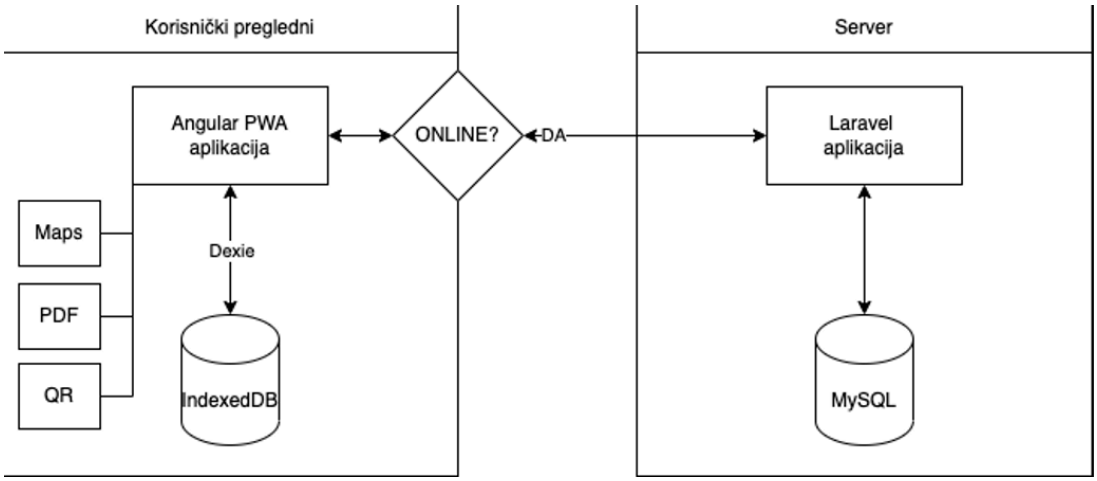


Slika 4. Relacijski model

3.5. Analiza interakcija u sustavu

Budući da aplikacija mora raditi u offline načinu rada, što omogućava PWA, potrebno je voditi brigu o načinu spremanja podataka u takvom načinu rada. PWA nam ne pruža rješenje ovog problema, ali postoje druge tehnologije koje mogu pomoći u tome.

Prije nego uključimo te tehnologije, potrebno je vidjeti kakva će biti interakcija u sustavu. Dijagram interakcije prikazan je na slici 5.



Slika 5. Dijagram interakcije komponenti u sustav

Na dijagramu možemo vidjeti da je rješenje za spremanje podataka u offline načinu rada korištenje baze na strani klijenta. Klijentska web tehnologija koja nam omogućava to rješenje je IndexedDB baza. Kako bi se olakšao rad s tom bazom, korišten je plugin Dexie. Dakle, korisnik će tijekom korištenja aplikacije sve operacije izvoditi nad klijentskom bazom, a kada postoji veza prema internetu ta baza će se sinkronizirati s onom na serveru.

4. Tehnologije razvoja

Radi lakšeg razvijanja samog programa podijeljen je na dva dijela. Odnosno, projekt je podijeljen na dva sloja: klijentski sloj i serverski sloj.

4.1. Klijentski razvoj

Klijentski razvoj pisan je u klijentskim tehnologijama za web programiranje, a to su:

1. HTML
2. CSS
3. ANGULAR

4.1.1. HTML (HyperText Markup Language)

HTML [4] je kratica za **HyperText Markup Language**, što bi u doslovnom prijevodu značilo prezentacijski jezik za izradu web stranica. Dokumenti za hipertekst stvaraju se uz pomoć HTML jezika. Uz pomoć HTML jezika oblikujemo razne sadržaje i stvaramo hipertekstualni dokument.

HTML [5] jezik je vrlo jednostavno za naučiti i vrlo lako se upotrijebi te je to jedan od razloga velike prihvaćenosti od strane korisnika. Najveći uspjeh je postigao iz razloga što je u samim početcima bio besplatan i ostao je besplatan te je na taj način dostupan svim korisnicima. Prikaz hipertekst dokumenata omogućen nam je preko web preglednika. Zapravo prava zadaća HTML jezika je da uputi preglednik na koji način će prikazivati hipertext dokument krajnjim korisnicima. Svaki ovaj dokument nastoji izgledati jednako na svim web preglednicima i računalima. HTML se ne smatra pravim programskim jezikom i tako ljude koji koriste ovaj jezik ne smatramo programerima. Ovim programskim jezikom ne možemo izvršavati nikakve zadaće zbrajanja, oduzimanja ili slično, on nam isključivo služi za opis naših hipertekstualnih dokumenata. HTML [6] dokumenti su zapravo obične tekstualne datoteke, ekstenzija im je .html ili .htm. Osnovna građa svakog HTML dokumenta su

znakovi (eng. Tags), koji nam opisuju kako će naša stranica zapravo izgledati na web pregledniku. Poveznice unutar HTML dokumenata povezuju dokumente u uređenu hijerarhijsku strukturu i time određuju način na koji korisnik vidi sadržaj stranica. Slika 6 prikazuje izgled HTML koda za stranicu Korisnici.

```
<nz-page-header [nzGhost]="false">
  <nz-page-header-title>Korisnici</nz-page-header-title>
  <nz-page-header-extra>
    <button (click)="this.goToEdit()" nz-button nzType="primary">Novi</button>
  </nz-page-header-extra>
</nz-page-header>

<nz-table #basicTable [nzData]="this.data | async" [nzLoading]="this.db.isLoading" [nzScroll]="{ x: '600px' }"
>
  <thead>
    <tr>
      <th>Ime</th>
      <th>Prezime</th>
      <th>Email</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let data of basicTable.data" (click)="this.goToEdit(data.id)">
      <td>{{ data.firstname }}</td>
      <td>{{ data.lastname }}</td>
      <td>{{ data.email }}</td>
    </tr>
  </tbody>
</nz-table>

```

Slika 6. Html kod Vatrogasci Frontend

4.1.2. CSS (Cascading Style Sheets)

CSS [7] je kratica od Cascading Style Sheets ili na hrvatskom kaskadna tablica stilova. CSS je zapravo stilski jezik, koji nam služi za uređenje HTML dokumenata koji je napisan uz pomoć HTML jezika. Kako se web razvoj počeo razvijati, pojavila se potreba za ljepši prikaz elemenata na stranici. U samim početcima HTML nije imao neki stilski jezik i sve se ubacivalo u definiciju te je dolazilo do pretjerano velikog i teško čitljivog koda. Zbog toga se ubrzo vidjela potreba za stilskim jezikom koji će rasteretiti HTML, a koji je namijenjen prikazu sadržaja, a ne pozicioniranju i uređenju. Odnosno, CSS[8] nam omogućava dodavanja stila elementima na HTML stanici. U konačnici CSS nam služi za uređenje samog izgleda stanice i rasporeda elemenata na istoj. Tablica stilova u CSS-u također sadrži neka pravila koja moramo znati. Pravila se sastoje od dvije stvari: selektori i blokovi. Blokove odvajamo od selektora korištenjem vitičastih zagrada.

4.1.2.1. Selektori

U tablici 2 ispisani su neki od važnijih selektora. Selektor označava dio elementa na koji se primjenjuje neki stil.

Selektor možemo podijeliti na:

- svi elementi istog tipa, npr. svi elementi koji su naslov (<h1>)
- elementi određenog id ili class atributa:
    - *id*: jedinstven element
    - *class*: može biti više elemenata

Selektor	Primjer	Opis
.class	.prClass	Odabire sve elemente s class="prClass"
#id	#prId	Odabire element s id="prId"
element	p	Odabire sve elemente <p>
element.class	p.prClass	Odabire sve elemente <p> s class="prClass"
element,element	div,p	Odabire sve elemente i sve elemente <p>
[attribute]	[atribut]	Odabire sve elemente s ciljnim atributom

Tablica 2. CSS selektori

4.1.2.2. Deklaracijski blok

Nakon selektora slijedi deklaracijski blok unutar kojega dodajemo naredbe kojima uređujemo pojedini element. Deklaracijski blok je prepoznatljiv po vitičastim zagradama koje nam daju do znanja da će se za taj selektor napraviti neke izmjene. Unutar deklaracijskog bloka postoje razna svojstva. Svako svojstvo se nadopunjuje dvotočkom (**:**), te se poslije dvotočke dodaje vrijednost koja će se primijeniti na selektor. Zatvaranje naredbe vrši se s točka zarezom (**;**).

Na slici 7. prikazan je jedan isječak CSS koda. Može se vidjeti da se ovdje nalaze ukupno četiri CSS bloka. Svaki blok započinje selektorom. Prva sekcija koristi posebni selektor Angular frameworka (:host),. Nakon njega slijedi element selektor (img), te nakon njega dva klasa selektora (.container i .login-form). Unutar svakog deklaracijskog bloka nalaze se određeni stilovi koji se primjenjuju na taj blok.

```css
:host {
  width: 100%;
  height: 100%;
}

img {
  width: 256px;
  margin-bottom: 32px;
}

.container {
  width: 100%;
  height: 100%;

  display: flex;
  justify-content: center;
  align-items: center;
  flex-direction: column;
}

.login-form {
  min-width: 300px;
}
```

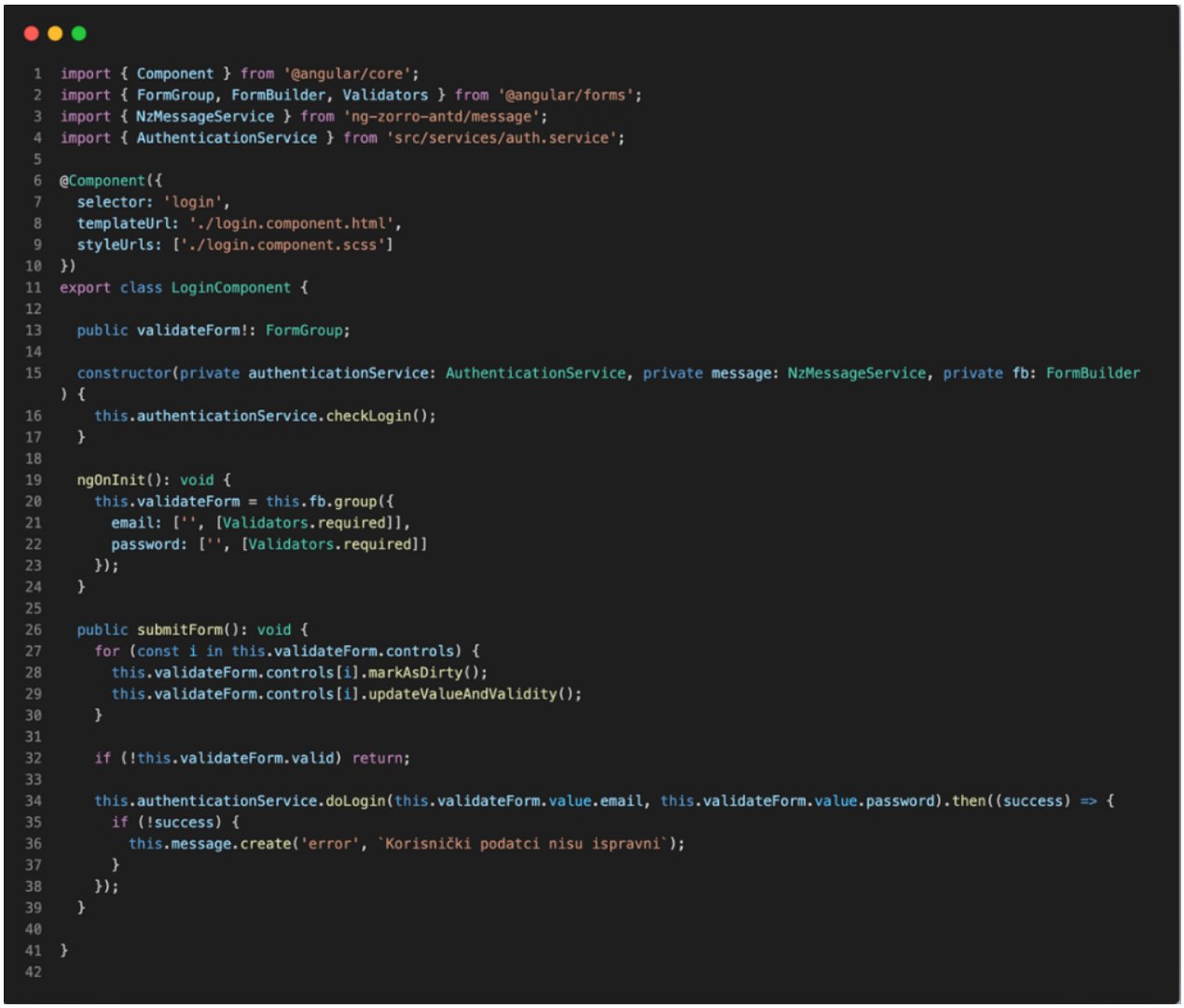
Slika 7. CSS kod Frontend vatrogasci

5. Angular (biblioteka NG-Zorro)

Angular [9] je frontend framework koji je 2016. godine razvio Google. Taj framework je strukturiranog oblika i napravljen je na temelju JavaScripta te služi kako bi se mogla napraviti dinamična web aplikacija. Angular je omogućio korištenje standardnog HTML-a i CSS-a, ali je znatno proširio mogućnosti HTML-a. Angular [10] je vrlo jednostavan za korištenje i njegove komponente su vrlo korisne. Angularov data binding i dependency injection su omogućili da se pisanje koda svede na minimum kako bi mogli brže i kvalitetnije napisati dio programske logike. Omogućio je da se dio klijentske aplikacije “ono što korisnik vidi u aplikaciji” razvije relativno brzo. No da bi aplikacija bila potpuna, mora se odraditi backend dio u kojem se odrađuje trajna pohrana podataka spajanjem na bazu podataka. Angular taj dio ne pokriva i zato se uz njega moraju koristiti još neke tehnologije kao na primjer Ruby, C#, Java, Php Laravel i slično.

Ukratko, Angular je sve ono što bi se trebalo nalaziti u HTML-u da su ga razvijali u smjeru da bude framework za razvoj aplikacija. HTML je savršen deklarativni jezik za statičke dokumente. Problem je što se s njim ne može razvijati logika i dinamična aplikacija pa se iz tog razloga uz HTML koristiti još neki framework.

Može se reći da je Angular svojim dolaskom pojednostavio razvoj web aplikacija, iz samog razloga što je dao prostora programerima odnosno možemo reći da im je dao dozu apstrakcije. No tu dolazi do problema jer se može dovesti u pitanje fleksibilnost aplikacije. Angular nije rješenje za sve vrste web aplikacija, ali za većinu CRUD (CRUD aplikacije su aplikacije u kojima se koristi create, read, update i delete) aplikacija je vrlo dobar i koristan. Angular nikako nije dobar za razvoj web igara i GUI editora koji puno koriste DOM elemente. One su u potpunosti različite od CRUD aplikacije pa koriste neka sasvim drugačija rješenja. Većina poslovnih aplikacija spada u definiciju CRUD aplikacije i u tom području je Angular među najboljim frameworkovima. Slika 8. prikazuje jednu klasu napisanu u Angular framework-u.



```
import { Component } from '@angular/core';
import { FormGroup, FormBuilder, Validators } from '@angular/forms';
import { NzMessageService } from 'ng-zorro-antd/message';
import { AuthenticationService } from 'src/services/auth.service';

@Component({
  selector: 'login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.scss']
})
export class LoginComponent {

  public validateForm!: FormGroup;

  constructor(private authenticationService: AuthenticationService, private message: NzMessageService, private fb: FormBuilder
) {
    this.authenticationService.checkLogin();
  }

  ngOnInit(): void {
    this.validateForm = this.fb.group({
      email: ['', [Validators.required]],
      password: ['', [Validators.required]]
    });
  }

  public submitForm(): void {
    for (const i in this.validateForm.controls) {
      this.validateForm.controls[i].markAsDirty();
      this.validateForm.controls[i].updateValueAndValidity();
    }

    if (!this.validateForm.valid) return;

    this.authenticationService.doLogin(this.validateForm.value.email, this.validateForm.value.password).then((success) => {
      if (!success) {
        this.message.create('error', `Korisnički podatci nisu ispravni`);
      }
    });
  }

}

```

Slika 8. Angular Controller

6. Serverski razvoj

6.1. MySQL baza podataka

MySQL [15] je besplatni sustav pomoću kojeg se upravlja bazom podataka nekog otvorenog koda. MySQL je često izbor baze podataka iz razloga što je vrlo jednostavna za korištenje i vrlo je pregledna . Često se koristi za projekte koji su otvorenog koda, kao na primjer Linuxovi serveri, ali postoje inačice za ostale sustave kao što su Windows , macOs i ostali. MySql je baza koja je slobodna za većinu upotrebe.

MySQL nije bila prihvaćena na samim početcima iz razloga zato što su joj nedostajale neke od osnovnih funkcionalnosti koji su definiranne SQL standardom. Naknadno je optimizirana i dobila je mnoštvo funkcionalnosti te je postala je popularna. Ova baza je vrlo stabilna i ima jako dobro dokumentiranje svojih modula i ekstenzija. Jedna jako bitna stvar je što ima podršku brojnih programskih jezika.

MySQL baza je relacijskog tipa koji se pokazao kao najbolji način za pretraživanje i skladištenje velikog broj podataka. U suštini to predstavlja osnovu svakog pravog informacijskog sustava. Kao i ostali sustavi za upravljanje bazama, i MySQL poštuje ACID načela pri izvođenju transakcija nad podatcima. MySQL i PHP osvojili su veliki dio tržišta jer su otvorenog pristupa i besplatni za korištenje.

6.2. PHP Laravel

Laravel [11] je baziran na PHP-u, odnosno Simfony [12] framework-u te je open-source web framework. Laravel je razvio Taylor Otwell. Definirao je da će Laravel biti MVC (model – view controller) arhitekture koja će se primjenjivati u web aplikacijama. Laravel se smatra jednim od najrasprostranjenih PHP [13] frameworka-a trenutno u svijetu, te se također nalazi na GitHub-u pod licencom MIT- a.

Taylor Otwell-ov cilj pri razvoju Laravela je bio da osigura napredniju arhitekturu. Prvu beta inačicu je izbacio u listopadu 2011. godine, a cijela verzija je izašla pred kraj listopada iste godine pod nazivom Laravel 1. Nakon tri sljedeće inačice, za inačicu Laravel 4 odradili su potpuni rewrite, odnosno ponovno napisali kompletno novi kod. Izlaskom Laravel 5 objavljeno je da će on sadržavati podršku koja će trajati najmanje dvije godine, sve dok se popravljaju programske greške. Od izlaska kompletne sigurne verzije će najmanje tri godine sadržavati podršku te poslije toga je plan izlaska nove nadogradnje svake dvije godine. Slika 9. prikazuje jedan primjer modela koji je napisan u PHP Laravel framework-u.

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Equipment extends Model
{
    protected $table = 'equipments';

    protected $fillable = [
        'name', 'description', 'firefighterId'
    ];

    public function replaceOldPks($map) {
        if (isset($this->firefighterId) && $this->firefighterId < 0) {
            $this->firefighterId = $map[$this->firefighterId];
        }
    }

}

```

Slika 9. Laravel model

7. Produkcijska okolina

Kako bi se omogućilo korištenje sustava nakon faze programiranja potrebno je uspostaviti produkcijsku okolinu kojoj će moći pristupiti svaki korisnik. Za potrebe produkcije zakupljen je server u oblaku kod davatelje usluge DigitalOcean te je na njemu podignuta okolina LAMP. LAMP [14] označava akronim programa koje su potrebni za pokretanje same aplikacije, a to su: LINUX kao operativni sustav, APACHE kao aplikativni server, MySQL [15] kao baza podataka, te PHP kao programski jezik.

Za pristup do produkcijske okoline koristi se sljedeća adresa:

https://vatrogasac-filip.onlyoneif.com

7.1. Pristupni podatci

Predsjednik

Email: predsjednik@dvd.hr

Lozinka: predsjednik

Skladištar

Email: skladistar@dvd.hr

Lozinka: skladistar

Vatrogasac

Email: vatrogasac@dvd.hr

Lozinka: vatrogasac

8. Ovlasti u aplikaciji

Ovlasti u svakoj aplikaciji su vrlo bitna stvar jer se tako želi zaštiti bitne stvari u aplikaciji i omogućiti manipulaciju s podatcima. Ne može se dopustiti da svi korisnici aplikacije mogu imati sve informacije i pristup svim sadržajima koje aplikacija nudi. Svaka aplikacija je podijeljena tako da svaki korisnik ima ovlasti prema hijerarhiji u samoj aplikaciji U slučaju ove aplikacije, ona je podijeljena na dvije skupine korisnika (ovlasti): Predsjednik društva i vatrogasce.

8.1. Predsjednik društva i Zapovjednik postrojbe

Predsjednik društva i Zapovjednik postrojbe imaju najveće ovlasti u aplikaciji. Oba korisnika imaju skoro sve ovlasti kao administrator. Imaju mogućnost dodavati nove vatrogasce, dodati novo vozilo, bilježiti intervencije i dodavati inventar.

8.2. Skladištar

Skladištar ima ovlasti u aplikaciji za pristupanje vozilima, opremi i izvođenju inventure.

8.3. Vatrogasci

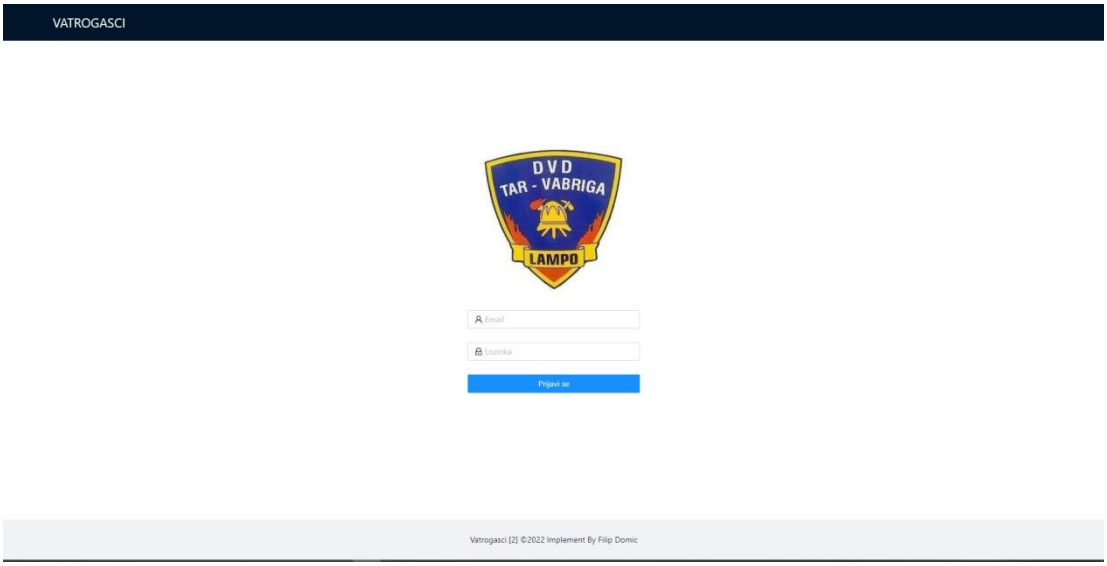
Vatrogasci imaju vrlo ograničene ovlasti, oni mogu pristupiti samo intervencijama.

9. Aplikacija

9.1. Početna stranica aplikacije

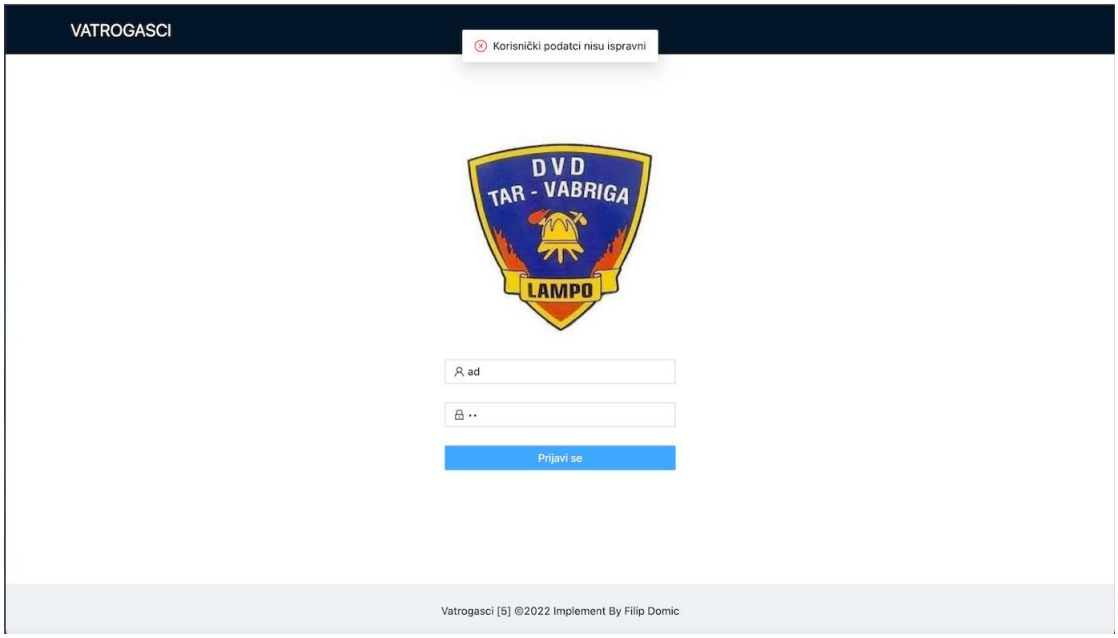
9.1.1. Prijava na stranicu

Prvi pogled na aplikaciju prikazuje stranicu za prijavu. Dizajn stranice je osmišljen da bude jednostavan i što pristupačniji svim korisnicima aplikacije. Razlog tome je što su neki od korisnika dobrovoljnog vatrogasnog društva korisnici starije populacije. Prvi korak na stranici za prijavu je provjera e-mail adrese i zaporke. Aplikacija provjerava da li su uneseni podatci valjani. U slučaju da su valjani podatci, sustav kreira web JWT [10] (autorizacijski token). Web token prilikom kreiranja dodjeljuje neko vrijeme valjanosti tokena, odnosno postoji neki period u aplikaciji unutar kojega se aplikacija može normalno koristiti, a nakon isteka vremena automatski se izlazi iz aplikacije. Slika 10 prikazuje izgled stranice za prijavu.



Slika 10. Stranica za prijavu

U slučaju da je korisnik poslao krive podatke, sustav javlja grešku da zaporka ili email nisu pravilno uneseni i da se provjeri točnost podataka kako bi ponovio proces prijave. Grešku javlja pomoću skočnog prozorčića u kojem piše „Netočni podatci. Probajte ponovo“. Korisniku se iz sigurnosnih razloga ne daje točan opis koji podatak je krivi (da li je kriva lozinka, korisničko ime ili oboje), nego im se daje generička poruka da podatci nisu ispravni. Stranica za prijavu prikazana je na slici 11.



Slika 11. Stranica za prijavu krivi unos podataka

9.1.2. Prijava na stranicu (programski kod)

Definiran je kontejner s klasom „container“. Unutar samog kontejnera nalazi se slika koja predstavlja logo na sredini stranice, te se u nastavku prikazuje obrazac unutar koje se nalaze dva podatka: email i lozinka, te se na kraju obrasca nalazi gumb za prijavu. Nakon pritiska na gumb za prijavu, u pozadini će se pozvati metoda koja će provjeriti da li su sva polja upisana, a ako nisu ispisat će se poruka u polju da je potrebno upisati podatak. Login HTML kod prikazan je na slici 12.

```
<div class="container">

  <img alt="" src="./assets/images/dvd.jpeg" />

  <form nz-form [formGroup]="this.validateForm" class="login-form" (ngSubmit)="submitForm()">

    <nz-form-item>
      <nz-form-control nzErrorTip="Upiši email">
        <nz-input-group nzPrefixIcon="user">
          <input type="text" nz-input formControlName="email" placeholder="Email" />
        </nz-input-group>
      </nz-form-control>
    </nz-form-item>

    <nz-form-item>
      <nz-form-control nzErrorTip="Upiši lozinku">
        <nz-input-group nzPrefixIcon="lock">
          <input type="password" nz-input formControlName="password" placeholder="Lozinka" />
        </nz-input-group>
      </nz-form-control>
    </nz-form-item>

    <button nz-button class="login-form-button login-form-margin" [nzType]="'primary'">Prijavi se</button>
  </form>

</div>

```

Slika 12. Prijava na stranicu (Login.html)

Kao što je prije navedeno, prilikom pritiska na gumb šalju se podatci, ali kada se na početku učitava stranica, pozvat će se konstruktor koji će provjeriti da li je korisnik ulogiran i postoji li jwt token. Ako token postoji, korisnik će biti preusmjeren na početnu stranicu aplikacije. Ukoliko jwt token ne postoji, poslije konstruktora pozvat će se metoda ngOnInit() koja će inicijalizirati polja. Sve se to odvija u pozadini. Nakon pritiska na gumb i upisanih podataka pozvat će se metoda submitForm() koja na početku provjerava da li su svi podatci upisani. Ako svi podatci nisu upisani, dobit će se poruka da je podatak obavezan. Ako

je korisnik zadovoljio uvjete validacije, podatci se šalju na backend i provjerava se da li su uneseni korisnički podatci valjani. Ako podatci nisi valjani, prikazuje se obavijest kroz skočni prozor objašnjeno u prethodnom poglavlju. Ako su svi uvjeti zadovoljeni dobiva se jwt token i korisnik je preusmjeren na početnu stranicu aplikacije. Login kontroler prikazan je na slici 13.

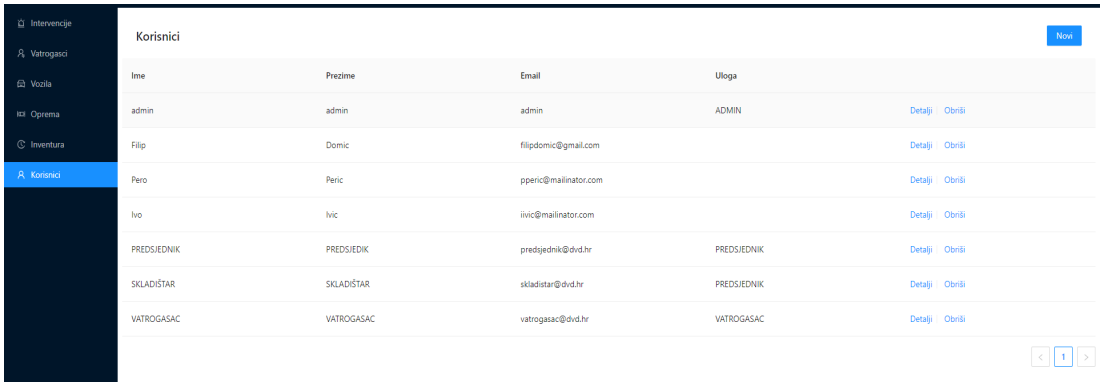
```
export class LoginComponent {

  public validateForm!: FormGroup;

  constructor(private authenticationService: AuthenticationService, private message: NzMessageService, private
 fb: FormBuilder) {
    this.checkLogin();
  }

  ngOnInit(): void {
    this.validateForm = this.fb.group({
      email: ['', [Validators.required]],
      password: ['', [Validators.required]]
    });
  }

  async checkLogin() {
    await this.authenticationService.checkLogin();
  }

  public submitForm(): void {
    for (const i in this.validateForm.controls) {
      this.validateForm.controls[i].markAsDirty();
      this.validateForm.controls[i].updateValueAndValidity();
    }

    if (!this.validateForm.valid) return;

    this.authenticationService.doLogin(this.validateForm.value.email, this
.validateForm.value.password).then((success) => {
      if (!success) {
        this.message.create('error', `Korisnički podatci nisu ispravni`);
      }
    });
  }

}
```

Slika 13. Prijava na stranicu (Login.controller)

9.2. Korisnici

9.2.1. Tablica korisnika

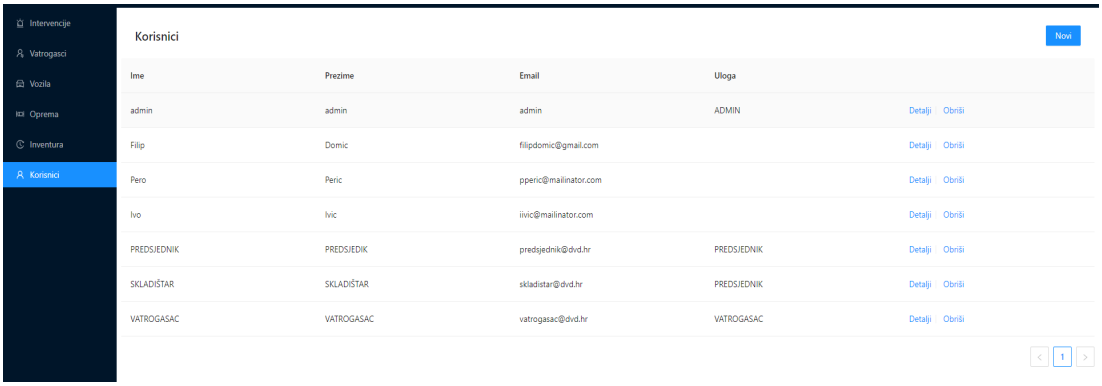
Prva mogućnost u aplikaciji je dodavanje novih korisnika vatrogasnog društva. Otvaranjem stranice korisnika prikazuje se tablica svih korisnika dobrovoljnog vatrogasnog društva. Unutar tablice se nalaze neki osnovni podatci o korisnicima dobrovoljnog vatrogasnog društva. U gornjem desnom kutu se nalazi gumb „Novi“ pomoću kojega se dodaju novi korisnici. Pritiskom na red mogu se promijeniti neke vrijednosti za korisnika dobrovoljnog vatrogasnog društva, a povratak je moguć pritiskom na gumb u gornjem lijevom kutu. Tablica korisnika prikazana je na slici 14.



Slika 14. Tablica korisnici

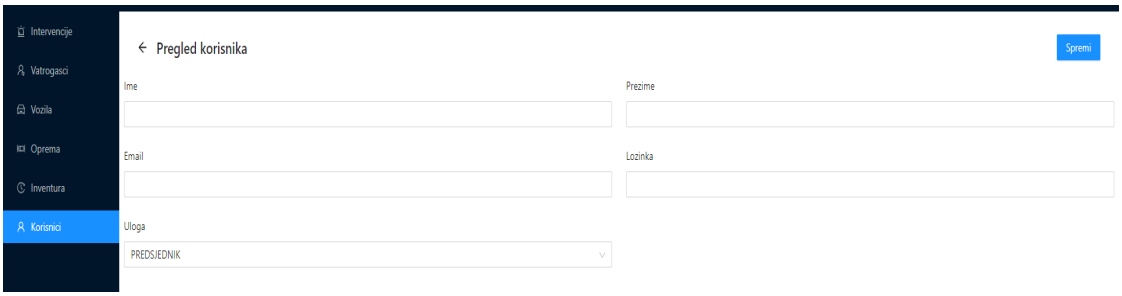
9.2.2. Dodavanje novog korisnika: Otvaranje forme

Za početak će se stisnuti gumb za dodavanje novog korisnika u gornjem desnom kutu. Nakon pritiska na gumb, automatski će se preusmjeriti na prozor unutar kojega će se moći upisati novog korisnika tima. Tablica korisnika prikazana je na slici 15.



Slika 15. Tablica korisnici

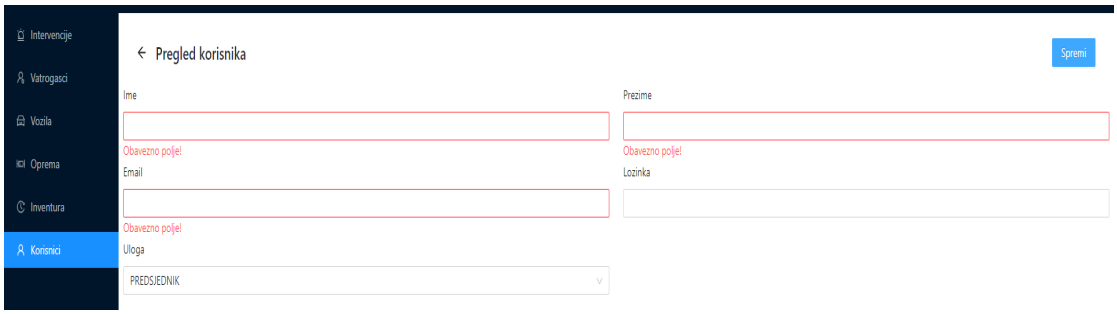
U prozoru se javlja jednostavni obrazac za popunjavanje korisničkih podataka. U gornjem desnom kutu se nalazi gumb za spremanje podataka. Ako korisnik ne želi spremiti novi podatak klikom u gornjem lijevom kutu na strjelicu se vraća u tablicu s podatcima. Prozor za dodavanje novog korisnika prikazan je na slici 16.



Slika 16. Prozor za dodavanje novog korisnika

9.2.3. Dodavanje novog korisnika: pogrešan unos podataka

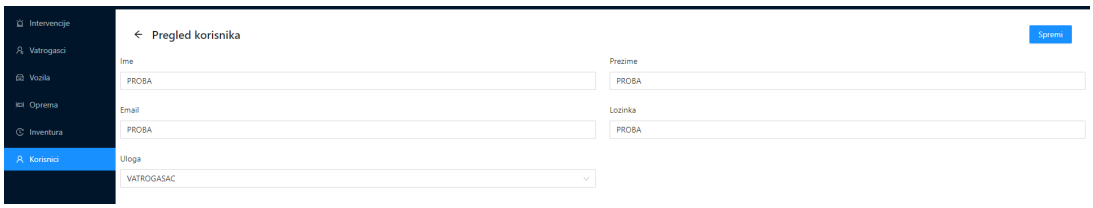
U sljedećem koraku pokušat će se stisnuti gumb za spremanje i spremiti nepotpuni obrazac. Podatci se ne mogu spremiti ako nisu popunjena sva obavezna polja te će javiti poruku s tom informacijom. Nakon popunjavanja svih polja sustav će dopustiti spremanje novog korisnika vatrogasne postrojbe i automatski će preusmjeriti u tablicu korisnika. Pokušaj dodavanja novog korisnika prikazan je na slici 17.



Slika 17. Pokušaj dodavanja novog korisnika

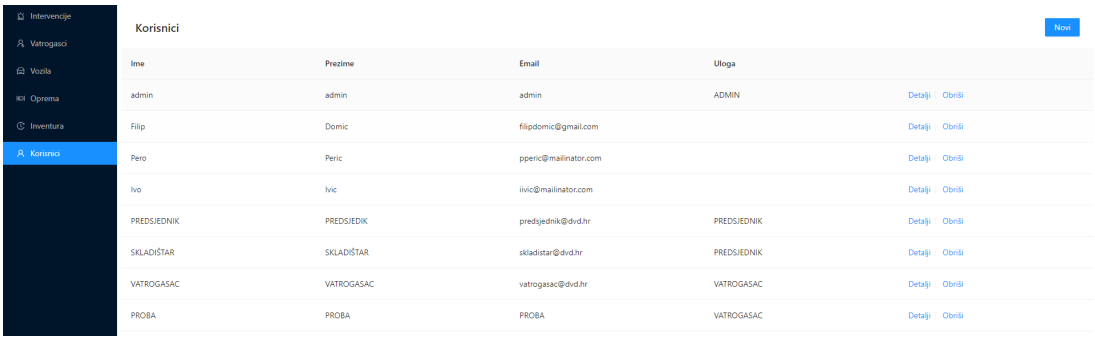
9.2.4. Dodavanje novog korisnika: Točan unos podataka

Nakon popunjavanja obrasca s točnim podatcima pritiskom na gumb se šalju podatci na provjeru kroz validator. Ako su svi uvjeti zadovoljeni može se spremiti novi korisnik. Popunjen obrazac s podatcima prikazan je na slici 18.



Slika 18. Popunjen obrazac s podatcima

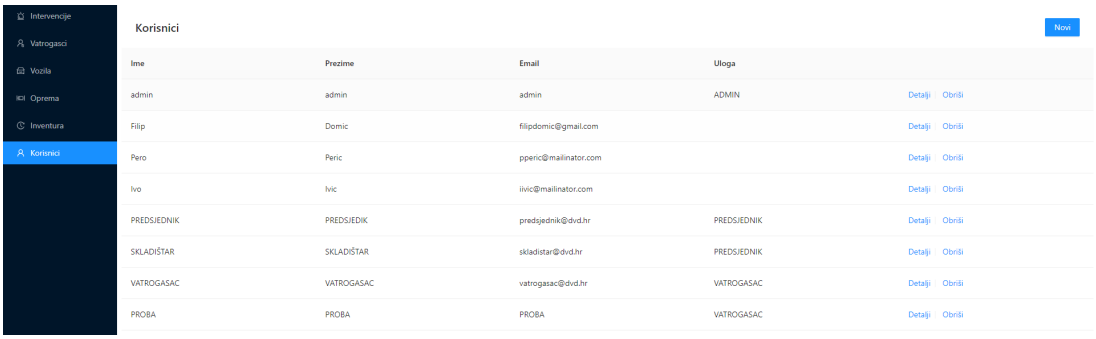
Nakon pritiska na gumb, sustav preusmjerava korisnika na prozor s tablica svih korisnika. Novi korisnik u tablici prikazan je na slici 19.



Slika 19. Novi korisnika u tablici

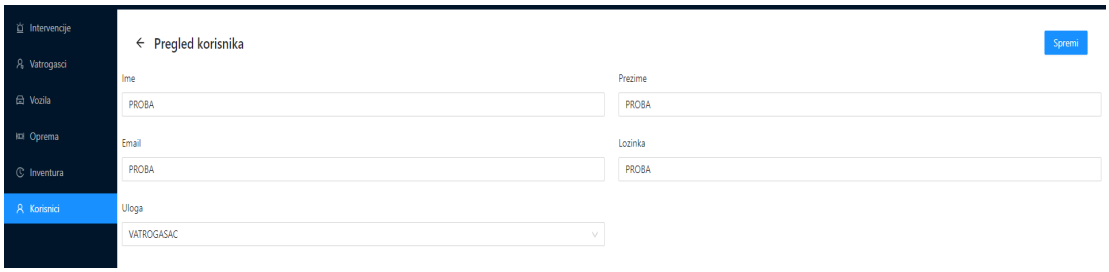
9.2.5. Uređivanje korisnika

Nakon što je dodan novi korisnik, moguće ga je uređivati. Za uređivanje korisnika će se prilikom pritiska na željenog korisnika otvoriti forma za uređivanje. Tablica korisnika s podatkom proba prikazan je na slici 20.



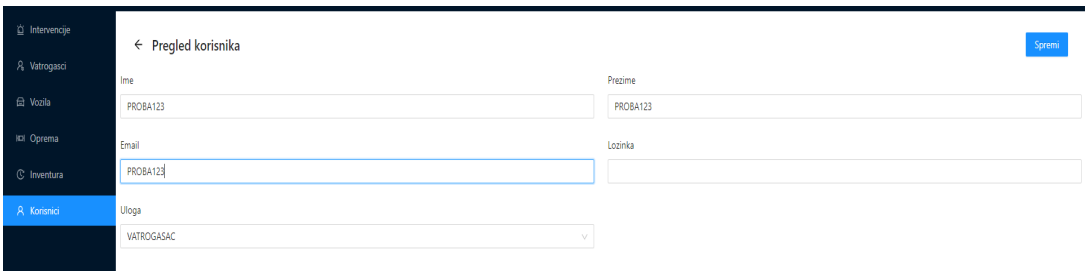
Slika 20. Tablica korisnika s podatkom proba

Popunjeni obrazac s korisnikom „proba“ prikazan je na slici 21. Isti taj korisnik će u sljedećem koraku biti uređen.



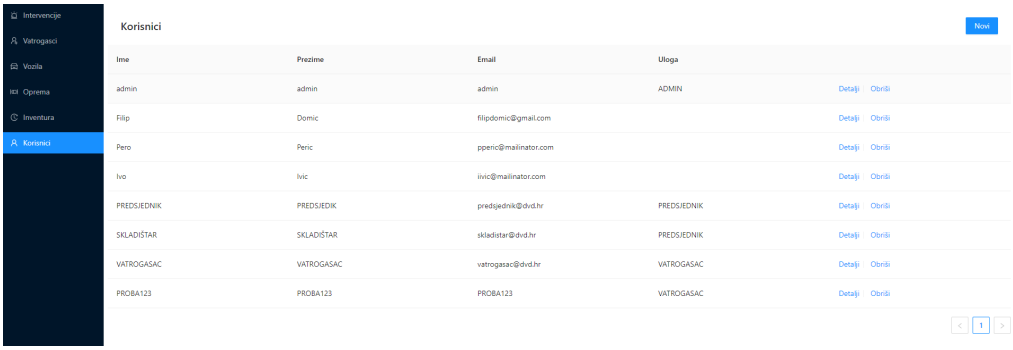
Slika 21. Obrazac popunjen s podatkom „proba“

Na isti način kao kod spremanje novog korisnika popunjava se obrazac sa željenim podatcima. Prilikom pritiska na gumb „Spremi“, sadržaj obrasca se sprema u tablicu. Obrazac popunjen s uređenim podatkom prikazan je na slici 22.



Slika 22. Obrazac popunjen s uređenim podatkom

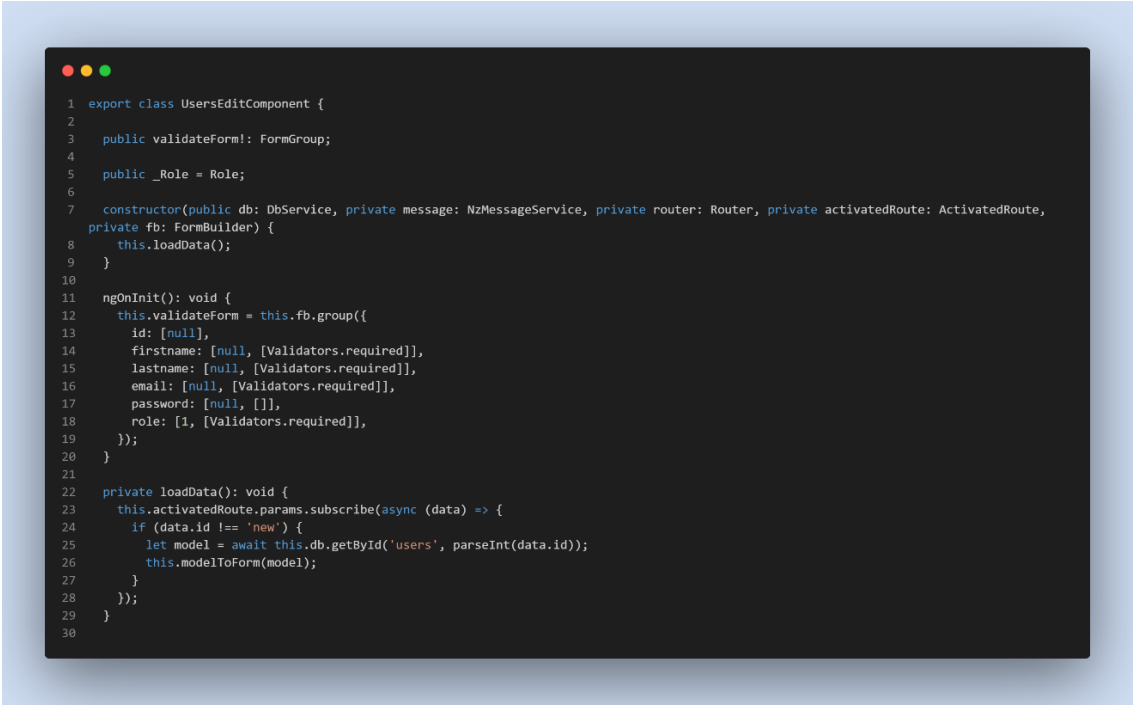
Uređeni podatak u tablici prikazan je na slici 23. Potrebno je naglasiti da se ne može izbrisati sadržaj obaveznog polja i pokušati spremiti jer sustav odmah javiti grešku da je to polje s obaveznim unosom.



Slika 23. Uređeni korisnik u tablici

9.2.6. Dodavanje ili uređivanje novog korisnika (Programski kod)

Kada se pristupi stranici za dodavanje novog korisnika ili uređivanje, automatski će se pozvati konstruktor. U samom konstruktoru postoji poziv metode loadData() ako se radi o uređivanju korisnika. Metoda ngOnInit() inicijalizirat će postojeći obrazac na stranici, nakon toga metoda loadData() napunit će obrazac s podatcima iz baze. Programski isječak prvog dijela kontrolera Korisnik prikazan je na slici 24.



```
export class UsersEditComponent {

  public validateForm!: FormGroup;

  public _Role = Role;

  constructor(public db: DbService, private message: NzMessageService, private router: Router, private activatedRoute: ActivatedRoute,
private fb: FormBuilder) {
    this.loadData();
  }

  ngOnInit(): void {
    this.validateForm = this.fb.group({
      id: [null],
      firstname: [null, [Validators.required]],
      lastname: [null, [Validators.required]],
      email: [null, [Validators.required]],
      password: [null, []],
      role: [1, [Validators.required]],
    });
  }

  private loadData(): void {
    this.activatedRoute.params.subscribe(async (data) => {
      if (data.id !== 'new') {
        let model = await this.db.getById('users', parseInt(data.id));
        this.modelToForm(model);
      }
    });
  }

```

Slika 24. Korisnik prvi dio (UserEditComponent)

Nakon prvog dijela, ako su upisani podatci u formu ili ih se želi urediti, pozvat će se jedna od dvije metode formToModel() ili modelToForm. Ove dvije metode služe za komuniciranje s interface-om koji omogućava da se unutar programskog jezika dohvaća i manipulira sadržajem nekog strukturiranog dokumenta. Konkretno u ovom slučaju, komunikacija Angular s HTML kodom. Kada je obrazac popunjen novim podacima ili su podaci uređeni, šalju se s gumbom i poziva se metoda submitForm(). Ova metoda na samom početku provjerava valjanost podataka, ako su podatci valjani dolazi se do if-a u kojem se provjerava da li je za novog korisnika upisana lozinka, ako nije javlja grešku. Ako je zadovoljen uvjet ide se do sljedećeg if-a koji provjerava da li je s tim podatcima poslan id. Ako je id poslan radi se o uređivanju. Kada su svi dijelovi metoda odrađeni, na kraju metode nalazi se

funkcija za preusmjeravanje na stranicu users gdje se nalazi tablica s podatcima. Programski isječak drugog dijela kontrolera Korisnik prikazan je na slici 25.

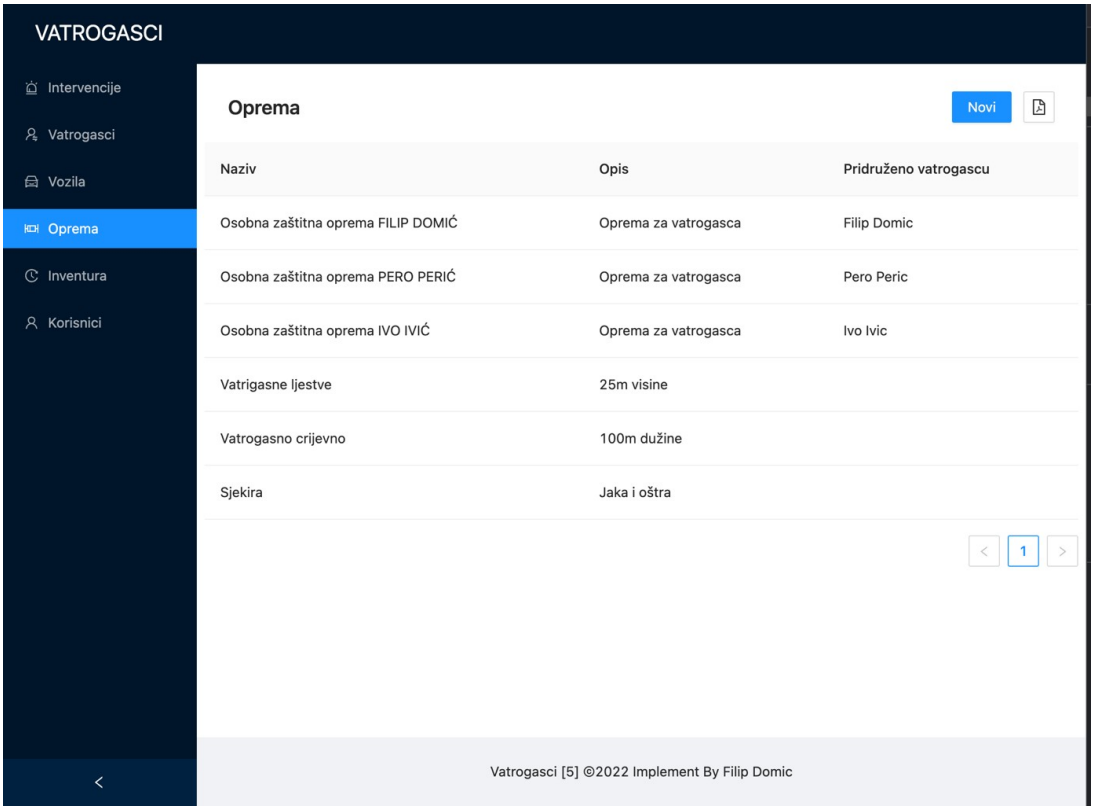
```
1
2   private formToModel(): UserInterface {
3     let model: UserInterface = {} as any;
4     for (const i in this.validateForm.controls) {
5       let value = this.validateForm.controls[i].value;
6       if (i === 'password') {
7         if(!value) continue;
8         value = btoa(value);
9       }
10      model[i] = value;
11    }
12    return model;
13  }
14
15  private modelToForm(model: UserInterface): void {
16    let data = {};
17    for (const i in this.validateForm.controls) {
18      if (i === 'password') continue;
19      data[i] = model[i];
20    }
21    this.validateForm.patchValue(data);
22  }
23
24
25  // === DOM LISTENERS ===
26
27  public async submitForm() {
28    for (const i in this.validateForm.controls) {
29      this.validateForm.controls[i].markAsDirty();
30      this.validateForm.controls[i].updateValueAndValidity();
31    }
32
33    if (!this.validateForm.valid) return;
34
35    let model = this.formToModel();
36
37    if (!model.id && !model.password) {
38      this.message.create('error', `Za nove korisnike potrebno je definirati lozinku!`);
39      return;
40    }
41
42    if (model.id) await this.db.update('users', model.id, model);
43    else await this.db.add('users', model);
44
45    this.router.navigate(['users']);
46  }
```

Slika 25. Korisnik drugi dio (UserEditComponent)

9.3. Oprema

9.3.1. Tablica opreme

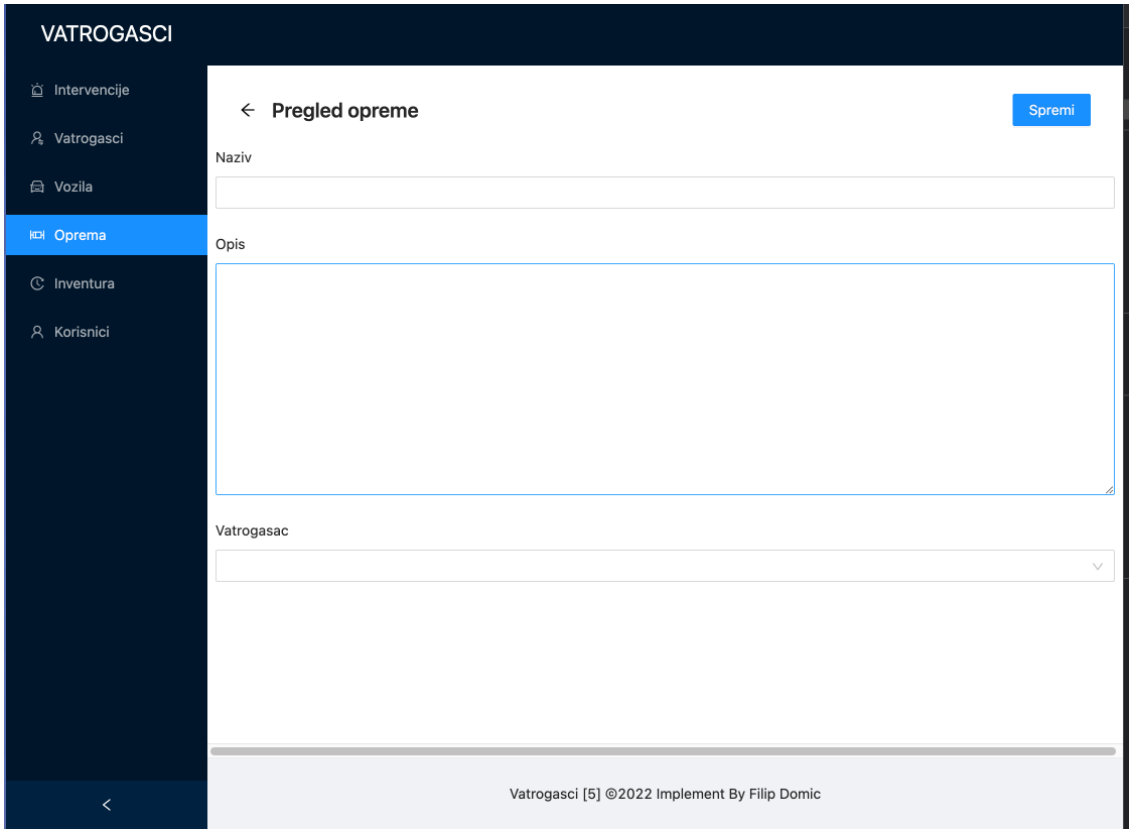
Skladište je prozor sa svom opremom koju ima vatrogasno društvo. Oprema se može podijeliti u dvije kategorije: osobnu vatrogasnu opremu, odnosno opremu koja će se dodijeliti nekom vatrogascu i zajednička oprema koja nije nikome dodana i zajednička je za sve vatrogasce.
U osobnu vatrogasnu opremu se ubrajaju odijelo, kaciga, rukavice. Zajednička oprema su vatrogasne ljestve, vatrogasno crijevo i slično. Otvaranjem opreme prikazuje se tablica sa svom opremom u društvu. Na ovaj način će dobrovoljno vatrogasno društvo imati uvid u to tko je zadužio koji dio opreme, te je lakše napraviti inventuru. O inventuri će biti detaljnije opisano u sljedećem poglavlju. U gornjem desnom kutu postoje dva polja: novi artikl i generator QR koda. Tablica opreme prikazana je na slici 26.



Slika 26. Tablica opreme

9.3.2. Dodavanje nove opreme

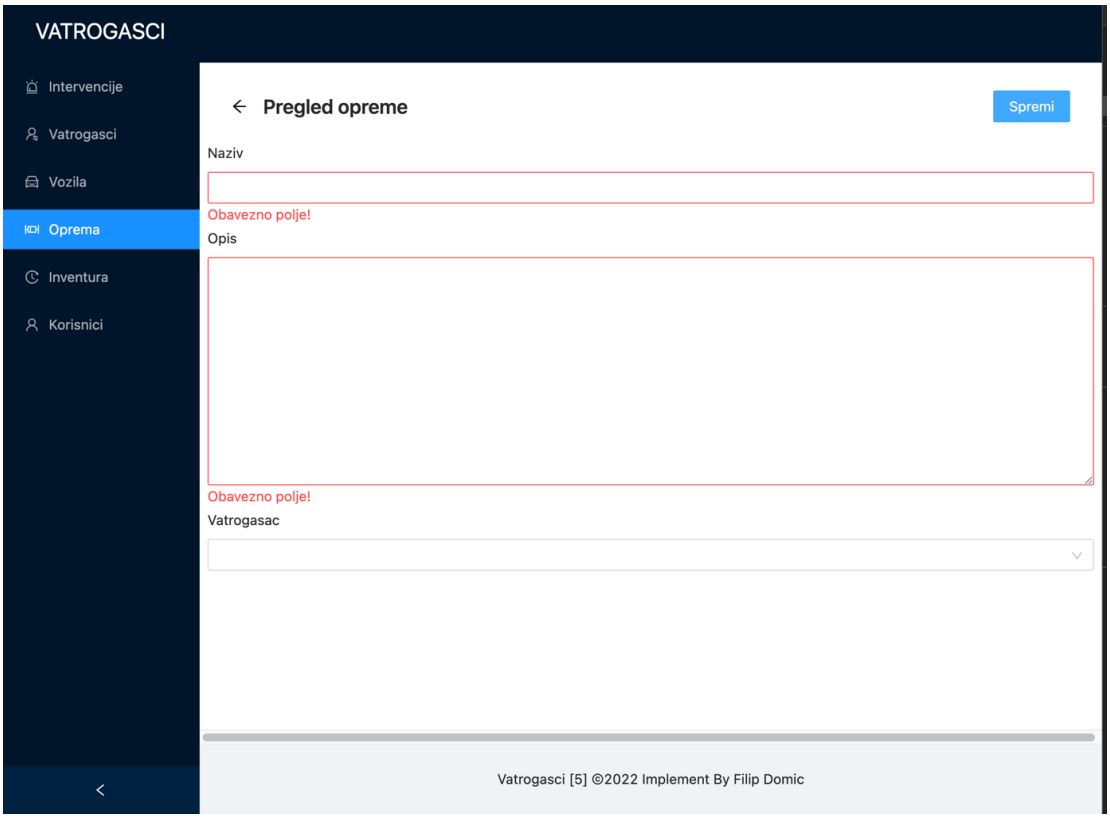
Stiskom na gumb za novu opremu, preusmjerit će se na prozor s obrascem za dodavanje nove opreme. U prozoru su naziv i opis obavezni unos, a vatrogasca se dodjeljuje u slučaju ako je vatrogasna oprema osobna. Prozor za dodavanje nove opreme prikazan je na slici 27.



Slika 27. Prozor za dodavanje nove opreme

9.3.3. Dodavanje nove opreme: netočan unos

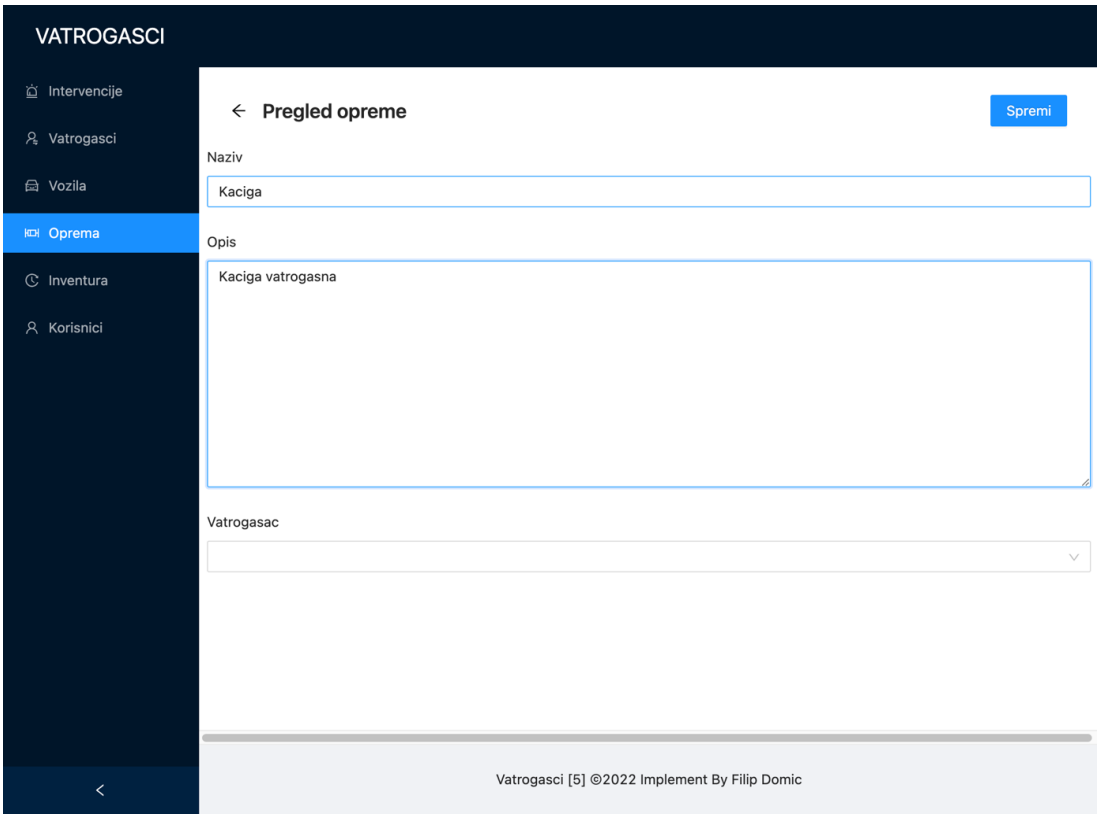
U sljedećem koraku pokušat će se stisnuti gumb za spremanje i pokušat će se spremiti nepopunjeni obrazac. Podatak se ne može spremiti ako nisu popunjena sva obavezna polja te se prikazuje poruka. Nakon ispravnog popunjavanja svih polja, sustav će dopustiti spremanje nove opreme te će automatski preusmjeriti na tablicu opreme. Pokušaj dodavanje nove opreme s nepopunjenim obrascem i prikazom poruka prikazano je na slici 28.



Slika 28. Pokušaj dodavanja nove opreme

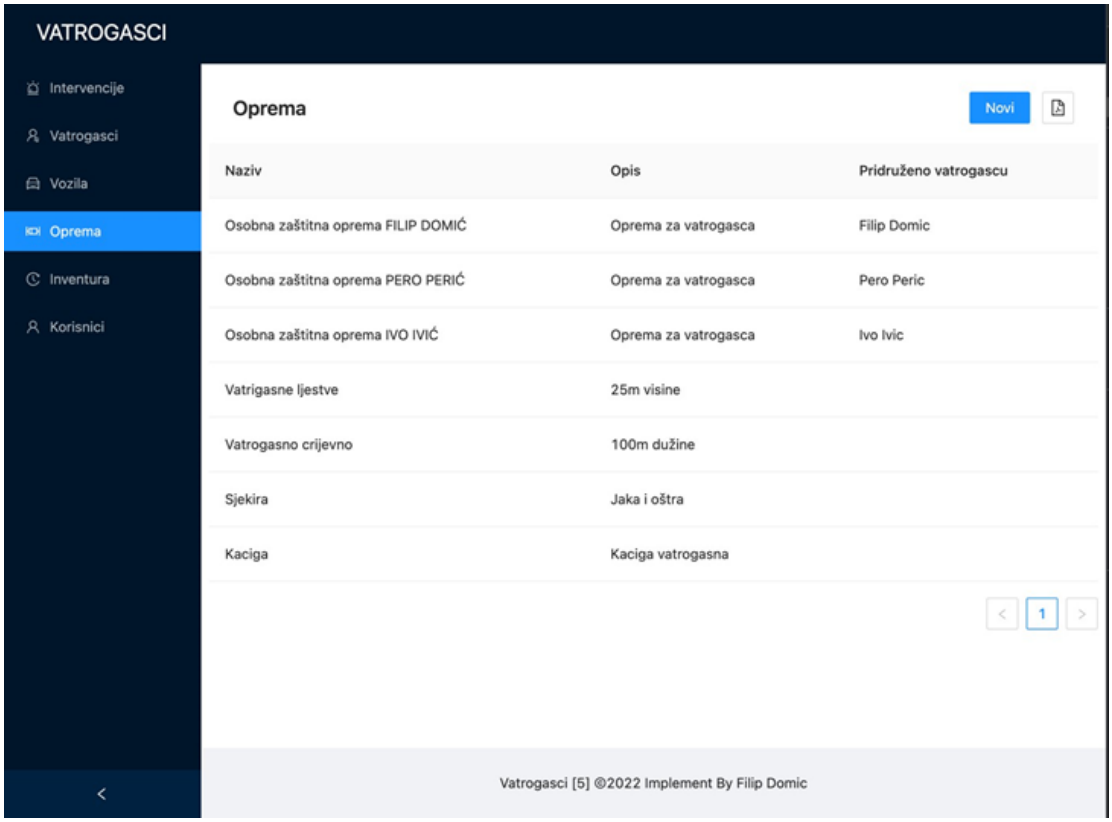
9.3.4. Dodavanje nove opreme: Točan unos podataka i spremanje

Nakon popunjavanja obrasca s točnim podatcima pomoću gumba šalju se podatci na provjeru kroz validator .Ako su svi uvjeti zadovoljeni može se spremiti nova oprema. Nakon provjere podataka, sustav automatski preusmjeri na tablicu sa opremom. Popunjeni obrazac s novom opremom prikazani je na slici 29.



Slika 29. Popunjeni obrazac sa podatcima oprema

Slika 30. prikazuje tablicu s novom vatrogasnom opremom koja je uspješno spremljena u prethodnom koraku. U ovom slučaju nije dodijeljena osobna oprema, taj korak će se napraviti kada bude prikazano dodavanje Vatrogasaca u postrojbu. Uređivanje je moguće na isti način kao i uređivanje korisnika.



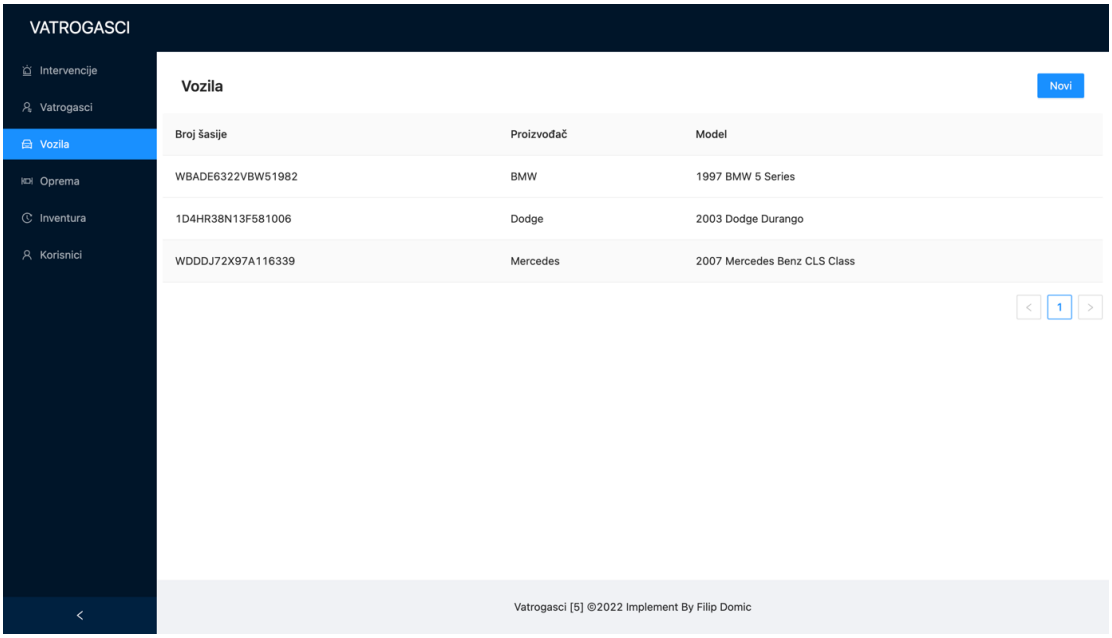
Slika 30. Tablica opreme s novim podatkom

9.4. Vozila

9.4.1. Tablica vozila

U ovom poglavlju predstavljen je prikaz tablice vozila. Neće se prikazivati dodavanje vozila ili promjenu postojećih podataka, zato što su isti postupci kao i za dodavanje nove opreme ili korisnika.

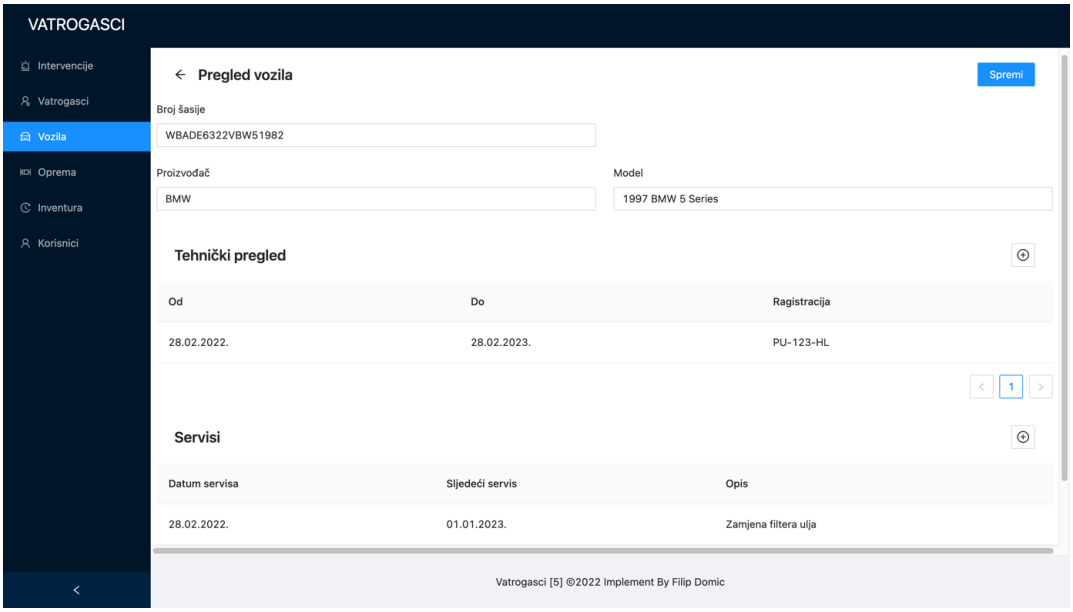
Prikazati će se dodavanje servisa i tehničkog pregleda za pojedino vozilo. Pritiskom na željeni red se automatski preusmjeri na odabrano vozilo sa svim detaljima. Tablica svih vozila prikazana je na slici 31.



Slika 31. Tablica svih vozila

9.4.2. Detalji za vozilo

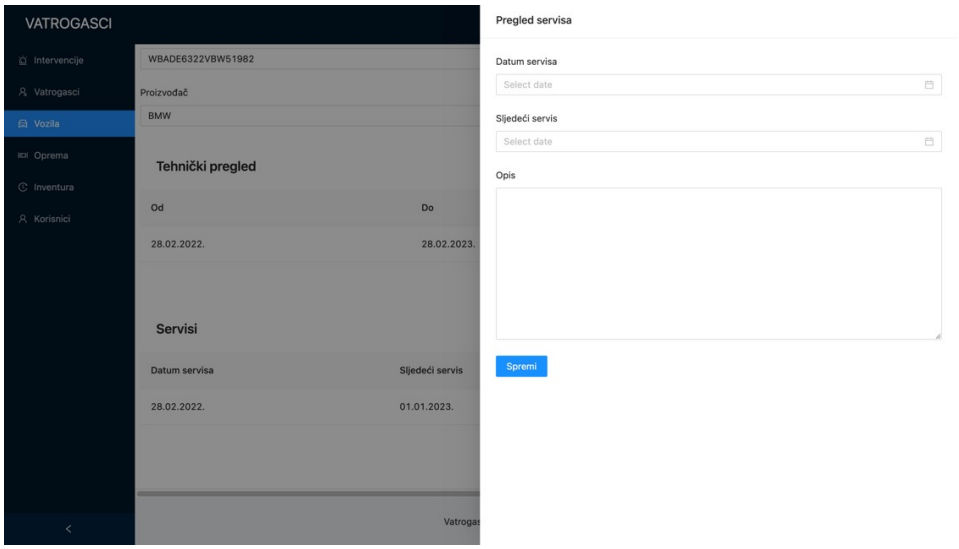
Na vrhu su svi detalji o modelu vozila i broju šasije, a pri dnu ekrana su u dvije različite tablice podijeljeni podatci o tehničkom pregledu i svim servisima vozila radi lakšeg praćenja stanja vozila. Dodat će se jedan servis za ovo vozilo. Pritiskom plusa u desnom kutu, otvorit će se obrazac za unos podataka. Detalji vozila prikazani su na slici 32.



Slika 32. Detalji vozila

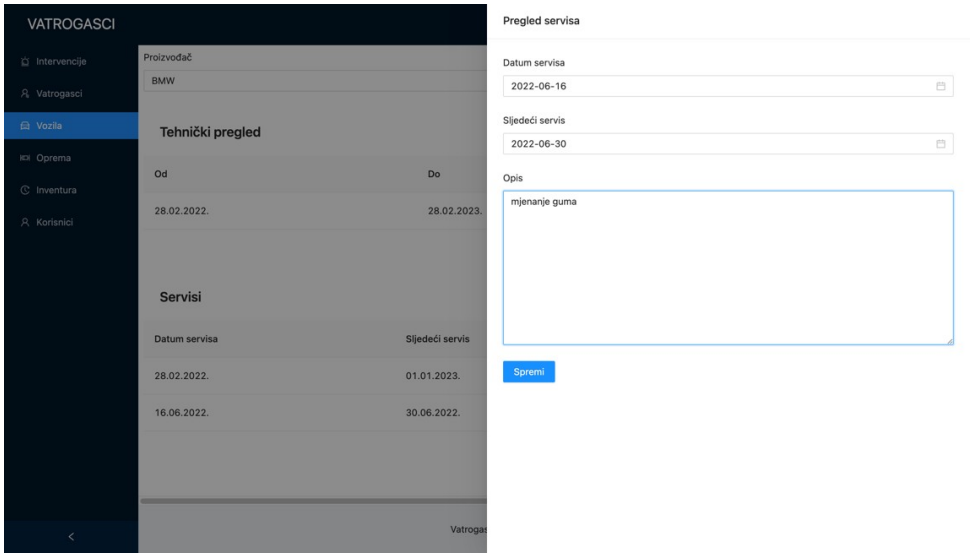
9.4.3. Dodavanje servisa za vozilo

Otvoren je obrazac za dodavanje podataka za servis za željeno vozilo. U obrascu postoji datum servisa, sljedeći servis i opis servisa. Obrazac za servise prikazan je na slici 33.



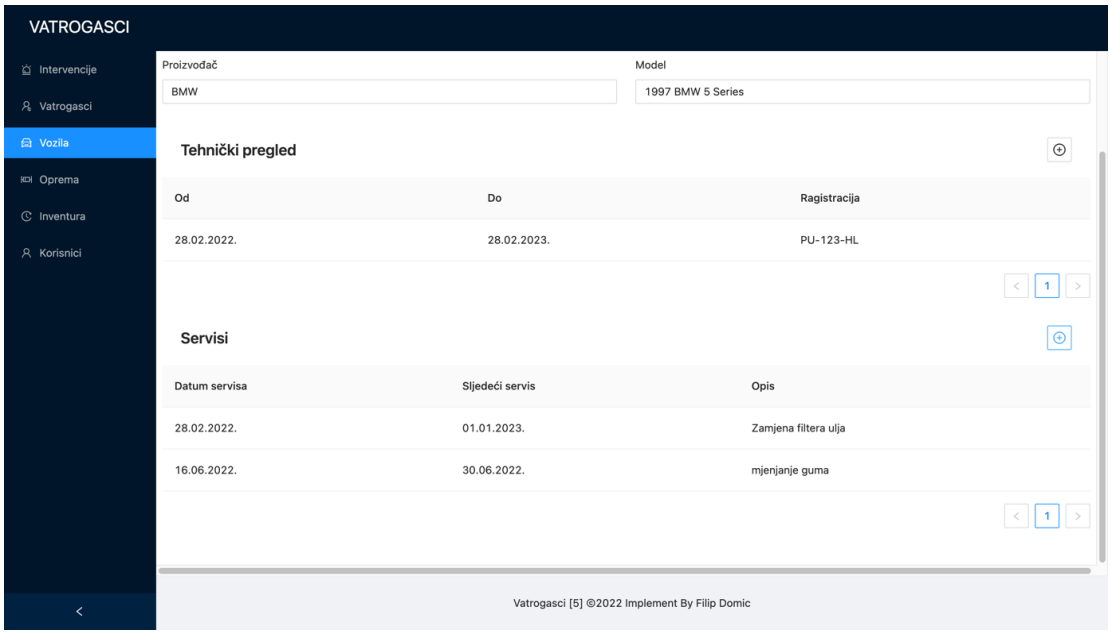
Slika 33. Obrazac za servis

Nakon otvaranja obrasca popunjavaju se željena polja s podatcima. Dodavanje novog servisa prikazano je na slici 34.



Slika 34. Dodavanje novog servisa

Na ovaj način je olakšano praćenje sljedećih servisa i registracije. Dodavanje tehničkog pregleda je odrađuje na isti način te je potrebno popuniti sva polja. Spremljen je novi podatak koji je vidljiv u tablicu i prikazan je na slici 35.

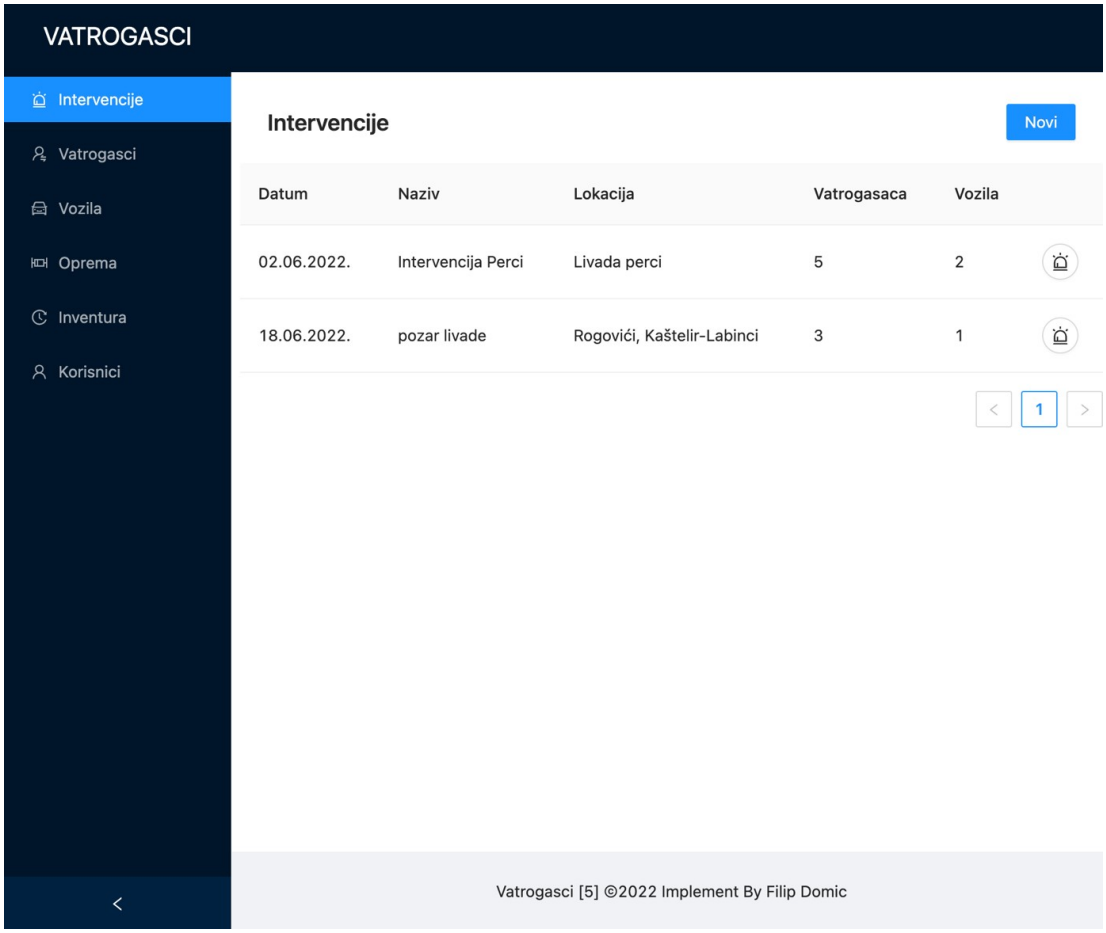


Slika 35. Dodano novo vozilo

9.5. Intervencije

9.5.1. Tablica Intervencije

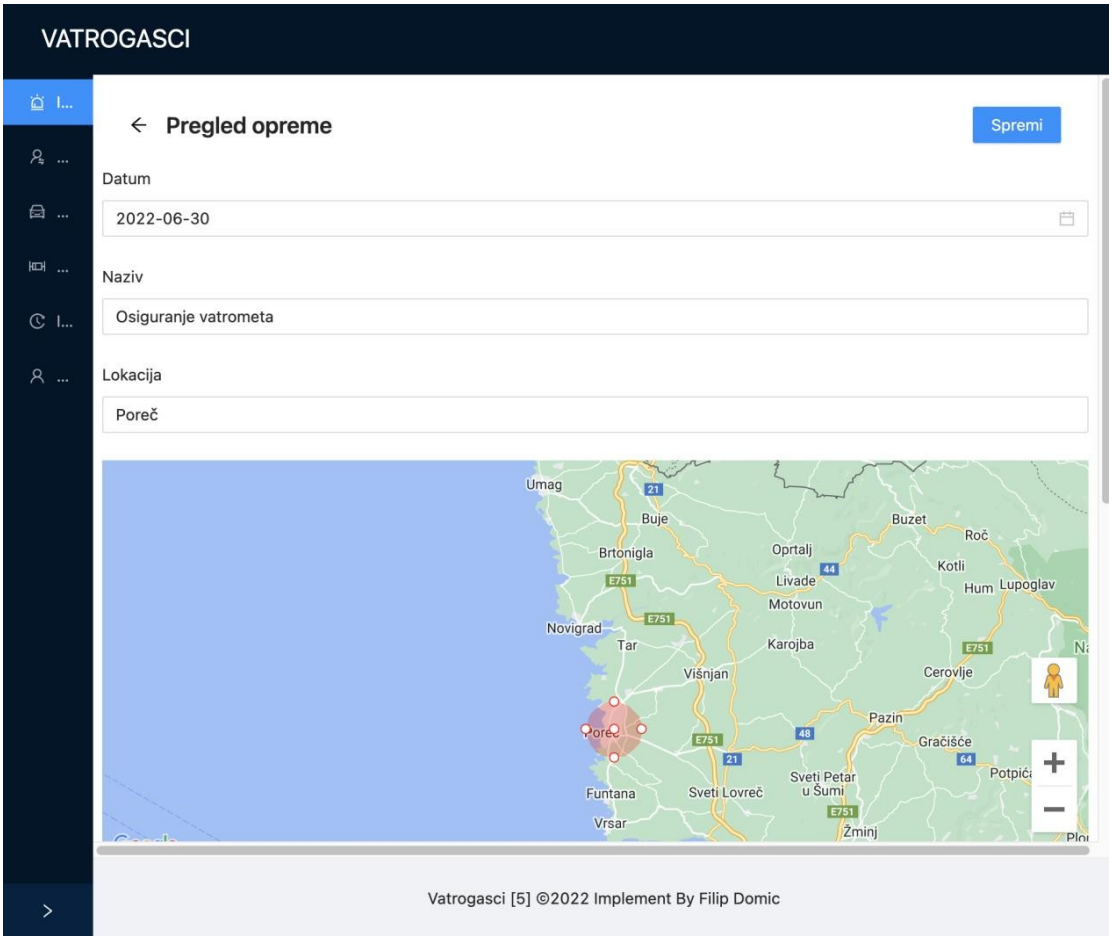
U ovom poglavlju je prikazana tablica za intervencije. Tablica ima neke od osnovnih podataka koliki broj vozila i broj ljudi je izašlo na intervenciju. U gornjem desnom kutu se nalazi gumb za dodavanje nove intervencije, te u samim redovima postoji i gumb za uzbunjivanje. Prvo će se dodati nova intervencija te će se potom uzbuniti neki od korisnika i poslati im e-mail. Tablica intervencija prikazana je na slici 36.



Slika 36. Tablica intervencije

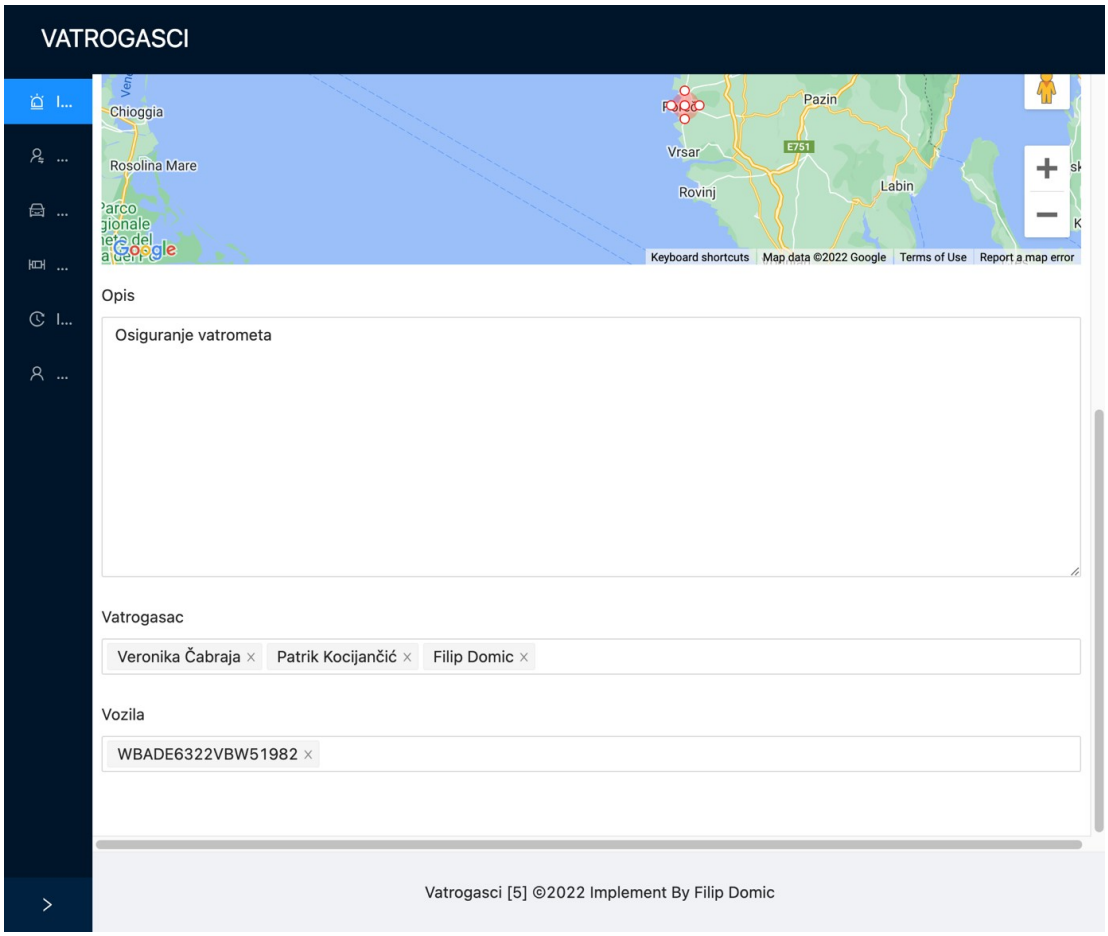
9.5.2. Obrazac za Intervencije

Nakon pritiska na gumb otvara se obrazac unutar kojeg se upisuju podatci. Prvo je polje datum. S obzirom na to da je ova aplikacija namijenjena dobrovoljnom vatrogasnom društvu, dosta „intervencija“ je dogovoreno unaprijed kao npr. osiguranje vatrometa, nekih manifestacija i slično. U slučaju da se radi o požaru moguće je dodati današnji datum i uzbuniti članove postrojbe. U ovaj obrazac upisuje se neki budući događaj, a radi se o osiguranju vatrometa. Prvi dio dodavanja nove intervencije je prikazan na slici 37.



Slika 37. Dodavanje nove intervencije prvi dio

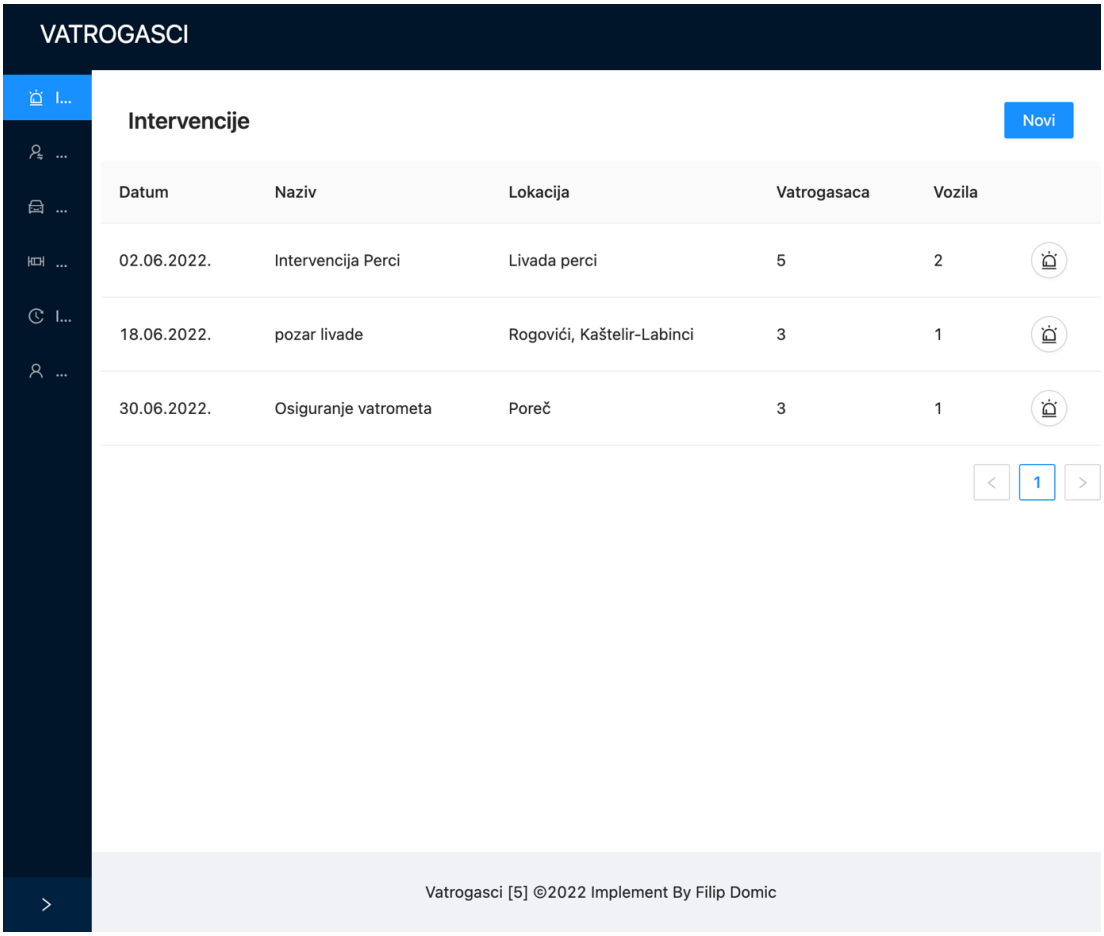
U drugom djelu obrasca će se napisati opis kao na primjer: „trajanje same intervencije“ ako se radi o nekim osiguranjima. Potom će se dodati svi članovi za koje se smatra da moraju biti obaviješteni te vozilo koje će se koristiti. Drugi dio dodavanja nove intervencije je prikazan na slici 38.



Slika 38. Dodavanje nove intervencije drugi dio

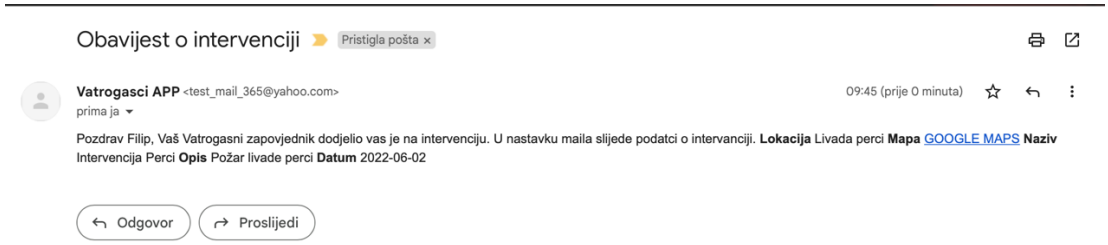
9.5.3. Uzbunjivanje vatrogasaca

Kada je sama intervencija spremljena, dolazi na red uzbunjivanje članova tima. To će se odraditi tako da se u redu željene intervencije pritisne gumb za uzbunjivanje članova postrojbe. Nakon pritiska gumba, automatski će se poslati e-mailovi svim članovima koji su obilježeni u obrascu. Uzbunjivanje vatrogasaca prikazano je na slici 39.



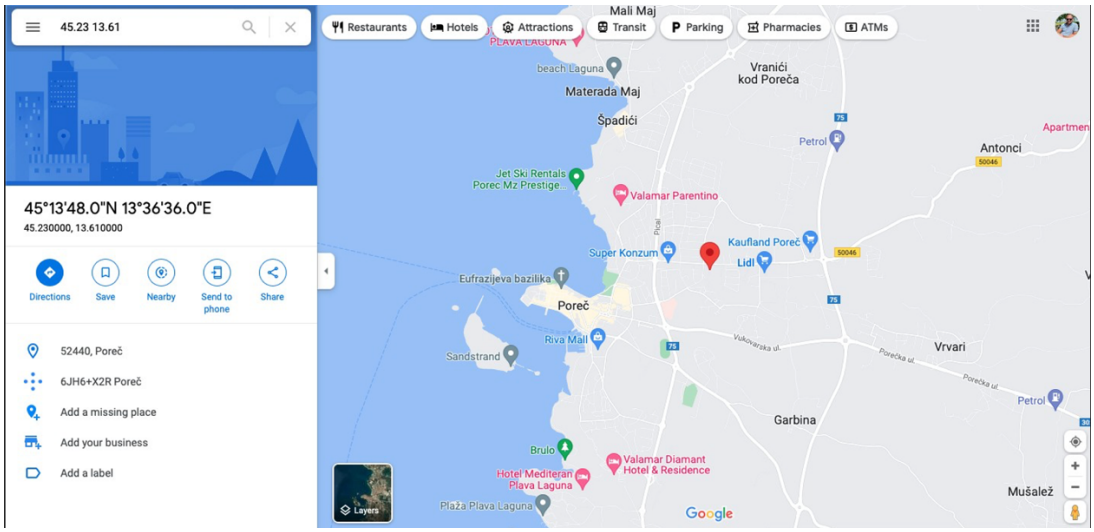
Slika 39. Uzbunjivanje vatrogasaca

Nakon pritiska gumba, mail je poslan članovima koji su označeni. Na slici 40. prikazan je mail u kojem su svi detalji intervencije.



Slika 40. Obavijest o intervenciji

Poslana je lokacija intervencije koju je moguće učitati na karti i tako lakše dobiti rutu do intervencije, prikazano na slici 41.



Slika 41. Lokacija intervencije

9.5.4. Intervencije (Programski kod)

U isječku programskog koda za intervencije vidljivo je da na samom početku su definirani atributi koji će biti korišteni za popunjavanje padajućih izbornika (vatrogasci i vozila). Također, u ovom modulu se koriste Google Mape, te kako bi ih prikazali potrebno je definirati neke podatke za njih.

U ostatku koda vidi se inicijalizacija samog obrasca, te procedure za dohvat podataka za padajuće izbornike. InterventionsEditComponent prikazan je na slici 42.

```
export class InterventionsEditComponent {

  public validateForm!: FormGroup;

  public firefighters: Array<FirefighterInterface> = [];

  public vehicles: Array<VehicleInterface> = [];

  public map = {
    zoom: 13,
    lat: 45.3033261,
    lng: 13.617756,
    circle: {
      lat: 45.3033261,
      lng: 13.617756,
      radius: 2000 // m
    }
  };


  constructor(public db: DbService, private message: NzMessageService, private router: Router,
private activatedRoute: ActivatedRoute, private fb: FormBuilder) {
    this.loadData();
  }

  ngOnInit(): void {
    this.validateForm = this.fb.group({
      id: [null],
      date: [null, [Validators.required]],
      name: [null, [Validators.required]],
      description: [null, [Validators.required]],
      location: [null, [Validators.required]],
      vehicles: [null, [Validators.required]],
      firefighters: [null, [Validators.required]],
    });
  }

  private loadData(): void {
    this.activatedRoute.params.subscribe(async (data) => {
      if (data.id !== 'new') {
        let model = await this.db.getById('interventions', parseInt(data.id));
        this.modelToForm(model);
      }
    });

    this.db.getAll('firefighters').then((data) => {
      this.firefighters = data.filter(x => x.active !== 0);
    });

    this.db.getAll('vehicles').then((data) => {
      this.vehicles = data.filter(x => x.active !== 0);
    });
  }
```

Slika 42. Intervencije prvi dio(InterventionsEditComponent)

U prvom dijelu programskog koda vidljive su stvari koje su vezane za inicijalni prikaz samog obrasca, dok su u drugom dijelu vidljive metode koje služe za manipulaciju i transformaciju podataka (formToModel i modelToForm), te metode koje služe za interakciju korisnika s aplikacijom (submitForm). Drugi dio InterventionsEditComponent prikazan je na slici 43.

```
1
2   private formToModel(): InterventionInterface {
3     let model: InterventionInterface = {} as any;
4     for (const i in this.validateForm.controls) {
5       let value = this.validateForm.controls[i].value;
6       if (i === "vehicles") value = value.join(";");
7       if (i === "firefighters") value = value.join(";");
8       model[i] = value;
9     }
10    model.lat = this.map.circle.lat;
11    model.lng = this.map.circle.lng;
12    model.radius = this.map.circle.radius;
13    return model;
14  }
15
16  private modelToForm(model: InterventionInterface): void {
17    let data = {};
18    for (const i in this.validateForm.controls) {
19      let value = model[i];
20      if (i === "vehicles") value = value ? value.split(";").map(x => parseInt(x)) : [];
21      if (i === "firefighters") value = value ? value.split(";").map(x => parseInt(x)) : [];
22      data[i] = value;
23    }
24    this.validateForm.patchValue(data);
25
26    this.map = {
27      ...this.map,
28      lat: model.lat,
29      lng: model.lng,
30      circle: {
31        lat: model.lat,
32        lng: model.lng,
33        radius: model.radius
34      }
35    };
36  }
37
38  // === DOM LISTENERS ===
39
40  public setRadius(event): void {
41    this.map.circle.radius = parseInt(event);
42  }
43
44  public setCenter(event): void {
45    this.map.circle.lat = event.lat;
46    this.map.circle.lng = event.lng;
47  }
48
49  public async submitForm() {
50    for (const i in this.validateForm.controls) {
51      this.validateForm.controls[i].markAsDirty();
52      this.validateForm.controls[i].updateValueAndValidity();
53    }
54
55    if (!this.validateForm.valid) return;
56
57    let model = this.formToModel();
58
59    if (model.id) await this.db.update('interventions', model.id, model);
60    else model.id = await this.db.add('interventions', model);
61
62    this.router.navigate(['interventions']);
63  }
64
```

Slika 43. Intervencije drugi dio(InterventionsEditComponent)

Komponenta za tablični prikaz podataka sastoji se od dohvata podataka iz IndexedDb baze preko Dexie plugina (liveQuery), te definiranja akcija koje se mogu izvesti nad određenim zapisom. Te akcije su brisanje zapisa i slanje obavještajnog e-maila. Može se primijetiti da slanje je e-maila moguće samo kada je aplikacija u mrežnom načinu rada (online). InterventionsTableComponent prikazano je na slici 44.

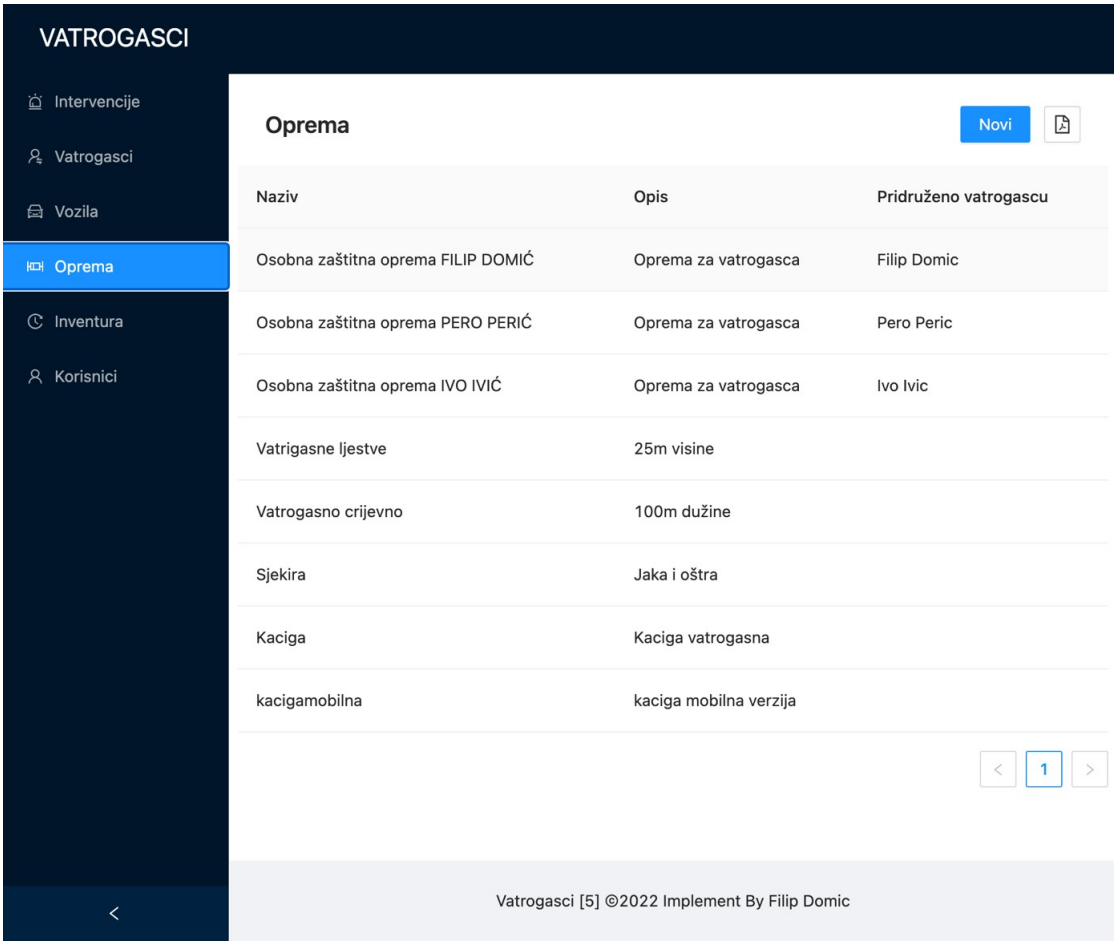
```
export class InterventionsTableComponent {

  public data: any = liveQuery (() => this.db.getInstance().interventions.toArray());

  constructor(public db: DbService, private pdf: PdfService, private router: Router, private
 message: NzMessageService) {
  }

  public arrayToNumber(items: string): number {
    return items.split(';').length;
  }

  public goToEdit(id?: number): void {
    if (!navigator.onLine) {
      this.message.create('warning', `Intervencije su moguće samo kada je sustav ONLINE!`);
      return;
    }
    let sufix = 'new';
    if (id) sufix = id.toString();
    this.router.navigate(['interventions', sufix]);
  }

  public sendAlert(id: number): void {
    if (!navigator.onLine) {
      this.message.create('warning', `Intervencije su moguće samo kada je sustav ONLINE!`);
      return;
    }
    this.db.sendAlert(id);
  }

  // === DOM METHODS ===

  public deleteItem(model: any): void {
    this.db.update('interventions', model.id, { ...model, active: false });
  }

}
```

Slika 44. Intervencije (InterventionsTableComponent)

9.6. QR kod generator i inventura

9.6.1. Generiranje QR kodova

Aplikacija sadrži u gornjem desnom kutu gumb za generiranje QR kodova za sve artikle koji se nalaze u tablici opreme. Pritiskom gumba dobije se PDF file s nazivima svih artikala i njihovi QR kodovi. Tablica sa svom opremom prikazana je na slici 45.



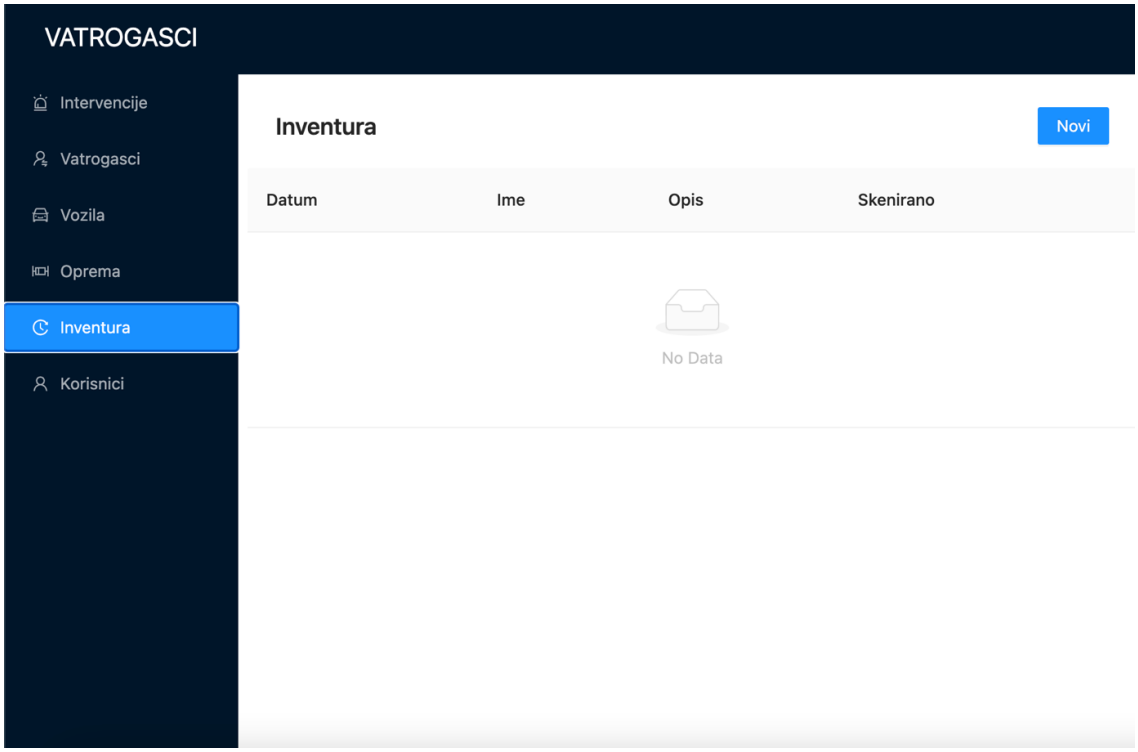
Slika 45. QR kod generiranje

Izgenerirani su QR kodovi za sve artikle, te se sada može napraviti inventura artikala. U pdf-u su svi nazivi i svi QR kodovi za pojedini artikl. Popis QR kodova prikazan je na slici 46.

Slika 46. Popis QR kodova

9.6.2.Tablica inventura

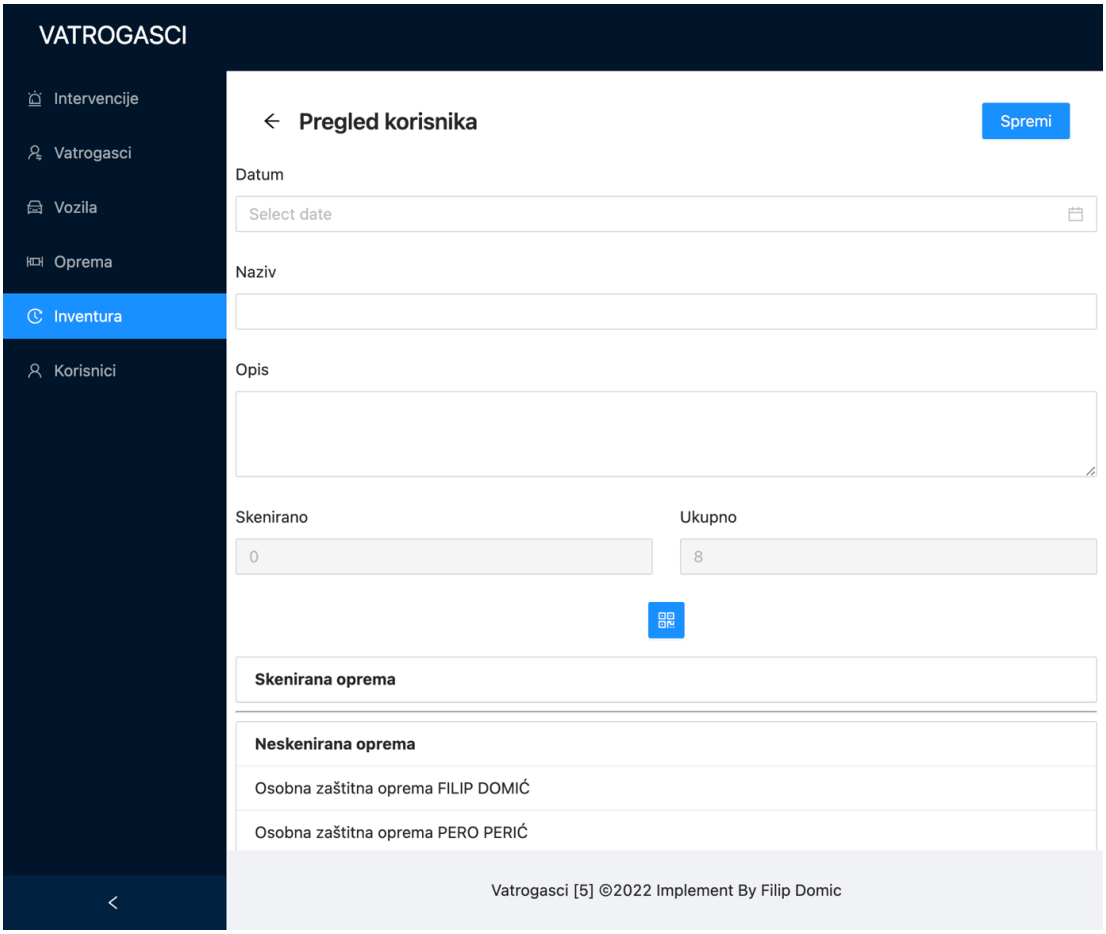
U prozoru je prikazana tablica inventura. U gornjem desnom kutu se može napraviti nova inventuru i generator QR kodova koji znatno olakšava posao. Pritiskom gumba u gornjem desnom kutu će se otvoriti obrazac za kreiranje inventure. Tablica inventure prikazana je na slici 47.



Slika 47. Tablica za inventure

9.6.3. Prozor za kreiranje inventure

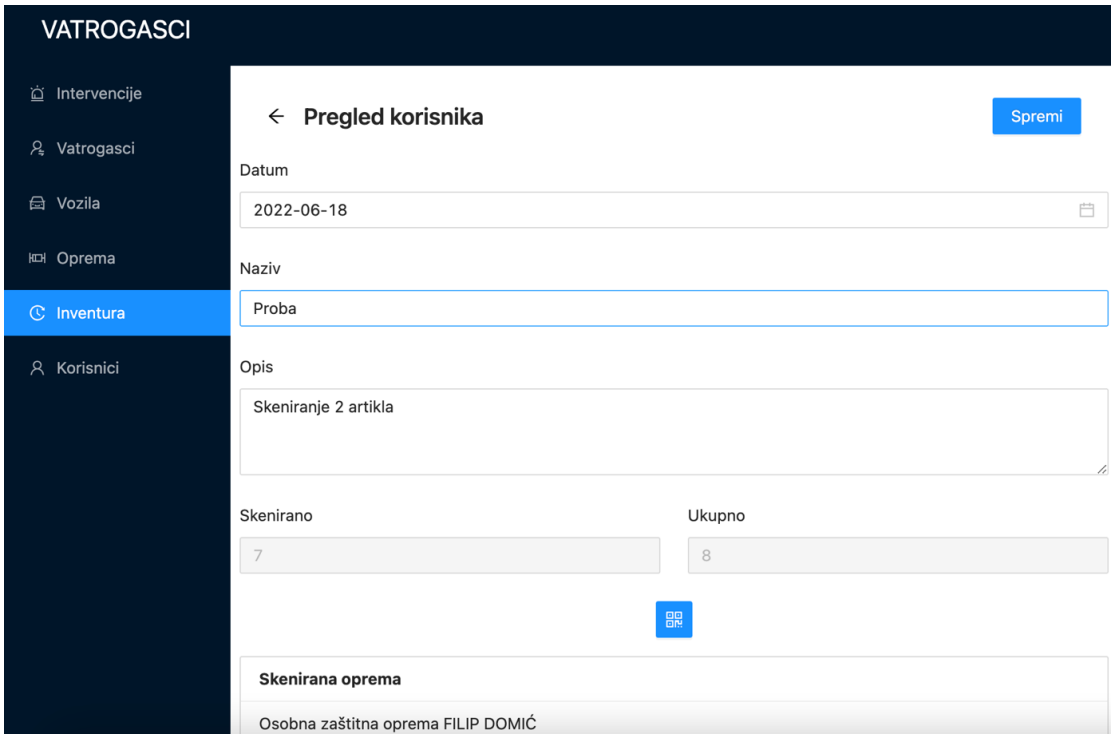
Otvoren je prozor unutar kojega se radi inventura. Unutar forme će se upisati osnovni podatci kao šta su datum, naziv i opis. Postoji opcija za skeniranje koja se nalazi na sredini ekrana. Ova opcija radi tako da se prilikom pritiska na gumb otvara kamera pomoću koje će se skenirati artikli. Obrazac za inventure prikazan je na slici 48.



Slika 48. Obrazac za inventure

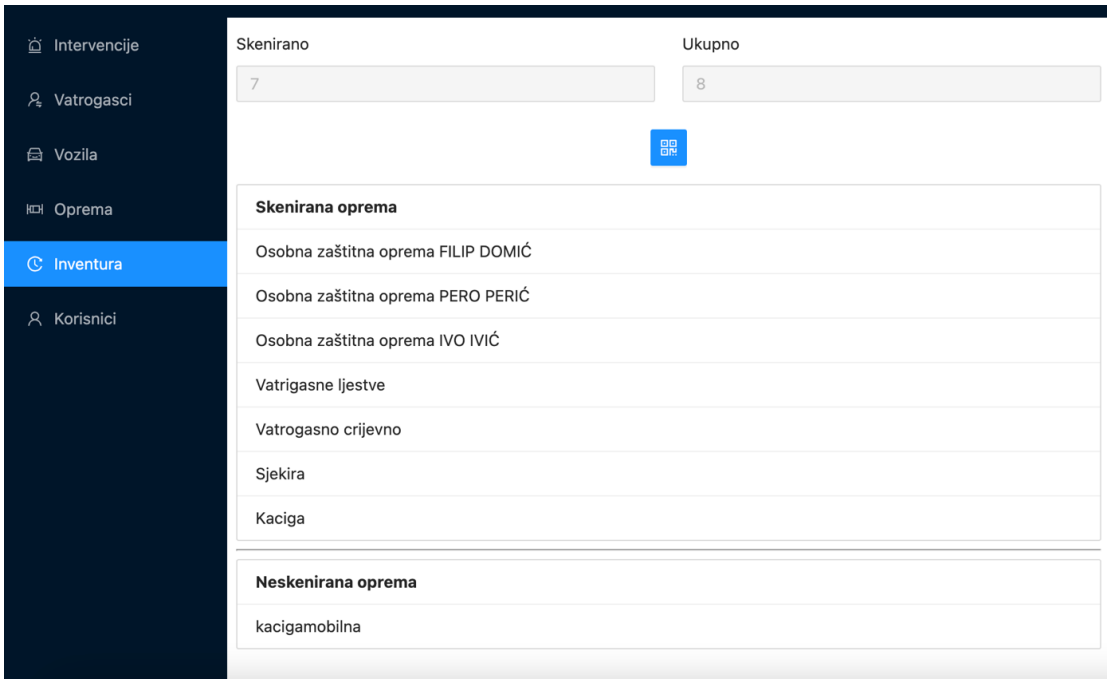
9.6.4. Skeniranje opreme

Popunjen je prvi dio obrasca s datumom, nazivom i opisom. Nakon popunjavanja polja s nazivima i opisom, može se koristiti kamera za skeniranje artikala. Skenirani su neki artikli, te postoje polja koja prikazuju sve artikle koji su skenirani i artikli koji nisu skenirani. Obrazac za inventure prikazan je na slici 49.



Slika 49. Obrazac za inventure prvi dio

Prikazano je skenirano 7 od 8 artikala kako bi se u samom prozoru prikazalo što se dogodi ako svi artikli nisu skenirani. Skenirani su samo neki artikli i oni su popunili polje skenirana oprema, a u polju ne skenirana oprema se nalaze artikli koji nisu skenirani ili nedostaju. Drugi dio obrasca za inventure prikazan je na slici 50.



Slika 50. Obrazac za inventure drugi dio

9.6.5. Skeniranje opreme (Programski kod)

Programski kod prikazan na slici ispod namijenjen je za prikaz i skeniranje opreme prilikom inventure. Na početku se mogu vidjeti dvije metode koje provjeravaju koje su stavke skenirane, odnosno nisu skenirane. U nastavku postoji metoda koja se poziva u trenutku kada korisnik pozove akciju spremanja. Na kraju se nalazi metoda koja je najzanimljivija, a radi se o callback metodi QR skenera. Tu metodu će pozvati servis za QR skener kada korisnik kamere skenira QR kod. U metodi se vrši obrada da se pronađe skenirani objekt, te da se postavi u stanje da je skeniran. InventoryEditComponent prikazan je na slici 51.

```
public getScanned(): Array<EquipmentInterface> {
    if (!this.validateForm.value.items) return [];
    return this.allEquipment.filter((x) => {
      return this.validateForm.value.items.find(y => y == x.id);
    });
  }

  public getUnScanned(): Array<EquipmentInterface> {
    if (!this.validateForm.value.items) return this.allEquipment;
    return this.allEquipment.filter((x) => {
      return !this.validateForm.value.items.find(y => y == x.id);
    });
  }

  // === DOM LISTENERS ===

  public async submitForm() {
    for (const i in this.validateForm.controls) {
      this.validateForm.controls[i].markAsDirty();
      this.validateForm.controls[i].updateValueAndValidity();
    }

    if (!this.validateForm.valid) return;

    let model = this.formToModel();

    if (model.id) await this.db.update('inventory', model.id, model);
    else await this.db.add('inventory', model);

    this.router.navigate(['inventory']);
  }

  public qrModalStete(state: boolean): void {
    this.modal = {
      isVisible: state
    };
    this.cdr.detectChanges();
  }

  public onQrDetect(value: string): void {
    console.log("QR: " + value);
    this.qrModalStete(false);
    if (value === '') return;
    let found = this.validateForm.value.items.find(x => x == value);

    if (found) return;
    let items = [...this.validateForm.value.items];
    items.push(value);
    this.validateForm.patchValue({ scannedItems: items.length, items: items });
    this.cdr.detectChanges();
  }

```

Slika 51. Skeniranje opreme (InventoryEditComponent)

10. Tehnologije razvoja

Za izradu ovog projekta korištene su standardne WEB tehnologije (HTML, CSS i JS) ali uz upotrebu frontend razvojnog framework Angular. Za razvoj web servisa korišten je PHP u kombinaciju s razvojnim framework-om Laravel, te MySQL kao skladište za podatke.
Kako su zahtjevi projekta da aplikacija mora raditi u offline načinu rada, za izradu aplikacije korišten je PWA koncept. PWA omogućava da se aplikacija na uređaju ponaša kao nativna aplikacija, a kako bi mogli pohranjivati podatke lokalno na uređaju iskorišten je IndexedDB.

10.1. Preduvjeti

Kako bi mogli započeti programirati u gore navedenim tehnologijama potrebno je instalirati sljedeće programe:

- Node.JS – potreban je za razvoj Angular aplikacija
- PHP i composer – potreban je za razvoj Laravel aplikacija

10.2. Uspostavljanje Angular projekta

Za uspostavljanje Angular projekta sve što je potrebno napraviti je pokrenuti sljedeću naredbu:

```
$ npm install -g @angular/cli
$ ng new vatrogasci-frontend
```

Pri pokretanju, angular-cli će generirati sve potrebne stvari za izradu aplikacije. Sve što je preostalo nakon toga je pokrenut app:

```
$ npm run start
```

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));

```

Slika 52. Ulazna točka (entry point) Angular aplikacije

Svaka web aplikacija mora imati neku ulaznu točku (entry point). Obično se ta datoteke naziva index.js, osnosno index.ts. Na slici iznad može se vidjeti kako ta datoteka izgleda u slučaju Angular aplikacije. Iz primjera je vidljivo da aplikacija kada se učita, prva komponenta koju će pokrenuti je AppModule.

Nakon što je utvrđeno da je AppModule ulazna datoteka aplikacije, u njoj se može utvrditi koja je prva komponenta koja će se učitati. Pod definicijom bootstrap je vidljivo da je prva komponenta koja će se prikazati AppComponent. Osnovni dio datoteke AppModule prikazan je na slici 53.

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
....

@NgModule({
  declarations: [
    AppComponent,
    RouterComponent,
    LoginComponent,
    ...
  ],
  imports: [
    BrowserModule,
    AppRouting,
    ...

    ServiceWorkerModule.register('ngsw-worker.js', {
      enabled: environment.production,
      registrationStrategy: 'registerImmediately'
    }),

    AgmCoreModule.forRoot({
      apiKey: 'AIzaSyA3hwQsHjG8rmzaeKSXNL5aBZIG8ag-CA0'
    }),

  ],
  providers: [
    { provide: NZ_I18N, useValue: en_US },
    AuthenticationService,
    ...
  ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }

```

Slika 53. Prikaz osnovnog dijela datoteke AppModule

10.3. Upostavljanje Laraval projekta

Za uspostavljanje Laravel projekta je potrebno pokrenuti sljedeću naredbu:

```
$ composer create-project laravel/laravel vatrogasci-backend
```

Pri pokretanju, composer će generirati sve potrebne stvari za izradu aplikacije. Sve što je preostalo nakon toga je pokrenut app:

```
$ php artisan serve
```

Kako bi izložili pristup aplikacije prema van, potrebno je ispravno definirati rute u aplikaciji. Sve rute u aplikaciji definiraju se u datoteci api.php. Datoteka api.php prikazana je na slici 54.

```php
<?php

use Illuminate\Http\Request;

/*
|--------------------------------------------------------------------------
| API Routes
|--------------------------------------------------------------------------
|
| Here is where you can register API routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| is assigned the "api" middleware group. Enjoy building your API!
|
*/

Route::get('fetch', 'SyncController@fetch');
Route::post('push', 'SyncController@push');
Route::get('send/{id}', 'SyncController@send');

```

Slika 54. Datoteka api.php

Na gornjoj slici je vidljivo da korisnik u aplikaciji ima definiran jedan kontroler (SyncController) i 3 rute (fetch, push, send).

10.4. Uspostavljanje PWA

Angular-cli jako olakšava stvar i kod dodavanja nekih vanjskih datoteka poput podrške za PWA. Tako da za dodavanje PWA podrške je potrebno pokrenuti sljedeću operaciju:

```
$ ng add @angular/pwa
```

PWA je dostupan u aplikaciji nakon pokretanja gore navedene operacije.

Ovdje treba napomenuti da PWA radi samo kada je aplikacija stvorena za produkcijski rad. Tijekom rada u programerskom modu PWA je isključen.

Kod uspostavljanja PWA aplikacije najbitnije je imati dobro definiranu manifest datoteku. Definicija manifest datoteke prikazana je na slici 55.

Slika 55. Definicija manifest datoteke

Na slici iznad je prikazano kako je definirana manifest datoteka, te koji su njeni dijelovi. Većina modernih browsera zna interpretirati ovu datoteku, te omogućiti da web aplikaciju koristimo kao nativnu aplikaciju.

10.5. Uspostavljanje ngZorro

Kako ne bi morali kreirati iz nule svaki gumbi i element u aplikacije, praksa je da se kod izrade aplikacija koristi neki od UI frameworka. Za izradu ove aplikacije odabran je Ant framework i njegova implementacija ngZorro.

Za dodavanje ngZorra u Angular projekt potrebno je pozvati:

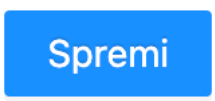
```
$ ng add ng-zorro-antd
```

Korištenje komponenti iz ngZorra je jednostavno. Može se demonstrirati na primjeru. Definiranje gumba prikazano je na slici 56.

```
1     <button (click)="this.submitForm()" nz-button nzType="primary">Spremi</button>
2
```

Slika 56. Definiranje gumba s ngZorro komponentom

Na gornjem primjeru se može vidjeti da se sa samo jednom linijom koda može definirati gumb koji stilski lijepo izgleda. Izgled gumba prikazan je na slici 57.



Slika 57. Izgled gumba u aplikaciji

NgZorro je izuzetno bogata kolekcija komponenti, te se za korištenje komponenti može konzultirati s odličnom dokumentacijom:

https://ng.ant.design/components

10.6. Uspostavljanje IndexedDB baze

Kako bi aplikacija mogla raditi i u izvan mrežnom načinu rada (offline) potrebno je osigurati da na uređaju korisnika budu pohranjeni svi podati. Za upravljanje podatcima na korisničkom uređaju izabrana je tehnologija IndexedDB. IndexedDB je baza podataka koja je ugrađena unutar korisničkog preglednika, te je njezino korištenje specificirano sukladno dogovorenim konvencijama. Kako bi bilo lakše upravljati IndexedDB-om korišten je plugin Dexie koji omogućava jednostavnije i efikasnije zapisivanje i dohvaćanje podataka iz lokalne baze. Isječak servisa prikazan na slici 58.

```
@Injectable()
export class DbService {

  public isLoading: boolean = false;

  private db;

  private tables = [
    'users',
  ];

  public async getAll(tableName: string) {
    return await this.db[tableName].toArray();
  }

  public async getById(tableName: string, id: number) {
    return await this.db[tableName].get({ id: id });
  }

  public async add(tableName: string, data: any, sync?: boolean) {
    if (!data.id) {
      let newId = await this.getNewId();
      data.id = newId * -1;
    }
    const ret = await this.db[tableName].add(data);
    if (!sync) this.sync();
    return ret;
  }

  public async update(tableName: string, id: number, data: any) {
    data['_updated_'] = true;
    const ret = await this.db[tableName].update(id, data);
    this.sync();
    return ret;
  }

}

```

Slika 58. Isječak servisa za upravljanje bazom

Na slici gore se može vidjeti servis koji je kreiran za komunikaciju s internom bazom. U isječku se vidi na koji način je definirana tablica za korisnike (*users*),

te kako se mogu dohvatiti podatci iz same baze (*getAll* i *getById*), odnosno kako se mogu dodati novi podatci (*add*) i ažurirati se (*update*).

10.7. Prikaz mape

Za prikaz mapa korišten je Google Maps. Integracija između Angulara i Google Maps-a napravljena je korištenjem plugina AGM (Angular Google Maps - https://angular-maps.com/). Kako bi se mogle koristiti Google Mape potrebno je u Google Consola aplikaciji zatražiti izdavanje API ključa.

Nakon što je ključ izdan, u modulu aplikacije se može definirati servis koji je prikazan na slici 59.

```
import { AgmCoreModule } from '@agm/core';

@NgModule({

  imports: [
    AgmCoreModule.forRoot({
      apiKey: 'AIzaSyA3hwQsHjG8rmzaeKSXNL5aBZIG8ag-CA0'
    }),

})
export class AppModule { }

```

Slika 59. Definiranje servisa za prikaz mapa

Korištenje ovog plugina uvelike olakšava korištenje mapa u Angularu, što je i prikazano na slici 60

```
<agm-map [latitude]="this.map.lat" [longitude]="this.map.lng" [zoom]="this.map.zoom">
  <agm-circle [latitude]="this.map.circle.lat" [longitude]="this.map.circle.lng" [radius]="this.map.circle.radius"
    [fillColor]="'red'" [circleDraggable]="true" [editable]="true" (radiusChange)="this.setRadius($event)"
    (centerChange)="this.setCenter($event)">
  </agm-circle>
</agm-map>
```

Slika 60. Definicija mape

10.8. Ispis u PDF

Funkcionalnost ispisa potrebna je kako bi određene zapisnike iz sustava mogli generirati u PDF obliku. Za generiranje PDF-ova korišten je JavaScript plugin pdfmake (http://pdfmake.org/playground.html). Generiranje Pdf dokumenta prikazano je na slici 61.

```
private _printEquipmentList(items: Array<{ id: string; name: string; img: string }>) {

  let docDefinition = {
    content: [],
    styles: {
      header: {
        fontSize: 18,
        bold: true
      }
    }
  }

  docDefinition.content.push({
    text: 'Popis vatrogasne opreme',
    style: 'header'
  });

  items.forEach(item => {
    docDefinition.content.push({
      alignment: 'justify',
      margin: [0, 32, 0, 0],
      columns: [
        {
          text: item.id + ' | ' + item.name
        },
        {
          image: item.img,
          width: 128
        }
      ]
    });
  })

  pdfMake.createPdf(docDefinition).download('oprema.pdf');
}
```

Slika 61. Isječak koda za generiranje PDF dokumenata

Na slici iznad može se vidjeti da se ispis PDF-a svodi na kreiranje PDF definicije sukladno standardu koji propisuje pdfmake. Nakon što je definicija kreirana, ona se predaje funkciji *createPdf* koja nakon toga kreira PDF te ga korisnik može preuzeti.

10.9. QR kodovi – generiranje i skeniranje

Za označavanje fizičkih artefakata (poput vatrogasne opreme) koriste se QR kodovi. Za generiranje QR kodova korišten je JavaScript plugin qrcodejs (https://davidshimjs.github.io/qrcodejs/), dok je za skeniranje korišten plugin nimiq/qr- scanner (https://github.com/nimiq/qr-scannera). Kreiranje QR koda prikazano je na slici 62.

```
const qrcode = new QRCode(document.createElement('div'), {
  text: item.id.toString(),
  width: 128,
  height: 128,
  colorDark: "#000000",
  colorLight: "#ffffff",
  correctLevel: QRCode.CorrectLevel.H
});
qrcode._oDrawing._elImage.onload = ev => {
  data.push({
    id: item.id.toString(),
    name: item.name,
    img: ev.target.src
  });
};
```

Slika 62. Isječak koda za kreiranje QR kod-a

Iz gornjeg isječka koda se vidi da je sve što je potrebno za kreiranje QR koda je napraviti funkciju *QRCode* te predati definiciju unutar koje se nalazi vrijednost koja će biti kodirana unutar QR kod, prikazano na slici 62.

11. Zaključak

Danas u 21. stoljeću bez računala i računalnih aplikacija se ne može apsolutno ništa. Sve je veći naglasak na digitalizaciji poslovanja i poslovnih procesa. Ono što digitalizacija u svojoj osnovi želi postići je da svim korisnicima unutar sustava omogući jednostavnije i preciznije poslovanje.

To je upravo ono što se je htjelo ovim radom napraviti i postići – jednostavnost i preciznost. Ovom aplikacijom postignuto je da svi njeni korisnici (voditelji, skladištari, vatrogasci) na jednostavan način u svakom trenutku mogu imati točnu sliku kakvo je stanje u društvu.
Ova aplikacija omogućava korisnicima da na brži način komuniciraju, te da kada je potrebno za neku intervenciju se može precizno utvrditi tko/kada/gdje treba biti.

Osim samih poslovnih izazova koji su se rješavali ovim radom, bilo je i određenih tehnoloških izazova koje je trebalo riješiti. Najveći tehnološki izazov kojeg je trebalo riješiti je korištenje aplikacije u izvan mrežnom načinu rada. Da bi se zadovoljio taj zahtjev korišten je PWA koncept, pomoću kojeg se na relativno jednostavan način uspjelo dobiti da aplikacija koja je pisana u web tehnologiji namijenjenoj web preglednicima, može raditi i na uređajima u načinu rada kao klasična aplikacija. Postoji srodna aplikacija, koja je potaknula da se napravi PWA aplikaciju, pošto postojeća aplikacija za vatrogasce nije dostupna bez interneta, te također nije dostupna za sve članove društva. Postojeća aplikacija je namijenjena samo za računalo i za profesionalne vatrogasne jedinice. Ova aplikacija cilja širu populaciju poput dobrovoljnih vatrogasnih društava.

Kako tehnologija i digitalizacija mogu u puno toga pomoći, tako i dalje postoje neke stvari koje tehnologija ne može odraditi. Primjerice, slanje intervencijskog zahtjeva kada se aplikacija nalazi u izvan mrežnom radu. Danas je taj zahtjev tehnološki i logični nerješiv, ali sigurno će jednog dana i za to postojati rješenje.

U samoj aplikaciji ima mjesta za poboljšanja, a u nastavku su opisani neki prijedlozi. Jedno od poboljšanja za intervencije mogla bi biti da svaka pojedina osoba koja je obaviještena o intervenciji može odgovoriti na poruku hoće li doći na intervenciju. Također, za intervencije bi bilo korisno dodati da se slanje intervencije odvija tako da dok osoba ne odgovori hoće li prisustvovati intervenciji, da toj osobi nakon 5 min zvoni mobitel, te da ga govorna sekretarica podsjeća na intervenciju. Unutar aplikacije se može dodati nova opcija za bilješke sa sastanaka i razne akcije. Unutar nove opcije bilo bi moguće izvući grafove sa statistikama intervencija gdje bi se mogli pratiti tko je bio na kojim intervencija, ili koliko se na primjer intervencija dogodilo tijekom godine. Također bi se moglo napraviti poboljšanje za vozila da se spoji sa gps uređajem koji prati stanje kilometara vozila i lokacije vozila.

12. Popis literature

[1]. Management Institute, (2008): Vodič kroz znanje o upravljanju projektima, Mate d.o.o., Zagreb, dostupno: https://www.scribd.com/doc/207257138/Vodic-Kroz-Znanje-o-Upravljanju-Projektima (10.07.2022)

[2]. Rajduković, M. i sur. (2012): Planiranje i kontrola projekata, Sveučilište u Zagrebu, Građevinski fakultet, Zagreb, dostupno: knjiga (10.07.2022)

[3]. Krajina T, Uvod u GIT , Zagreb, dostupno: https://tkrajina.github.io/uvod-u-git/git.pdf (12.07.2022)

[4]. Crowther R. (2013.) *Hello! HTML5 & CSS3*, Manning Publications, Shelter Island, dostupno: knjiga (12.07.2022)

[5]. Duckett J. (2011.) *HTML & CSS - Design and Build Websites*, John Wiley & Sons, Inc., Indianapolis, dostupno: knjiga (15.07.2022)

[6]. Goldstein A., Lazaris L., Weyl E. (2015.) *HTML5 & CSS3 For the Real World (Second Edition)*, SitePoint Pty. Ltd., Melbourne, dostupno: knjiga(15.07.2022)

[7]. Shenoy A., Guarini G. (2013.) *HTML5 and CSS3 Transition, Transformation and Animation,* Packt Publishing Ltd., Birmingham, dostupno: knjiga (16.07.2022)

[8]. Imagine Publishing Ltd. (2014.) *HTML5 & CSS3 – The Complete Manual*, Imagine Publishing Ltd., Bournemouth, dostupno: knjiga (16.07.2022)

[9]. Hussain A., (2016) *Angular 2: From Theory To Practice*, Daolrevo Ltd,

[10]. Fain Y., Moiseev A.,(2016) Angular Development with TypeScript, Manning, dostupno: knjiga (20.07.2022)

[11]. Nicholas Cerminara,(2008) *Getting Started with Laravel Homestead*, dostupno: https: //scotch.io/tutorials/getting-started-with-laravel-homestead# toc-clone-the-repo, dostupnono: knjiga (20.07.2022)

[12]. Bacer W., Bower T., (2008) Symfony 1.3 Web Application Development,Packt publishing, dostupno: knjiga (22.07.2022)

[13]. Zeekat Software Development,(2005) *The Model View Controller pattern in web ap- plications*, dostupno: https://zeekat.nl/articles/mvc-for-the-web.html, knjiga (22.07.2022)

[14]. Elmasri R., Navathe S., (2010) Fundamentals of Database Systems, 6th Edition. Addison- Wesley, Reading MA, knjiga (22.07.2022)

[15]. Silberschatz A., Korth H.F., Sudarshan S., (2011) Database system concepts, 6th Edition, New York: The McGraw-Hill Companies, knjiga (22.07.2022)

[16]. Hume D.A., Progressive web apps Izvor,(2018) knjiga (22.07.2022)

13. Popis slika, tablica, grafova

14. Sažetak

Koristeći PHP Laravel za backend i Angular za frontend osmišljena je progresivna web aplikacija koja je razvijena u suradnji s lokalnim dobrovoljnim vatrogasnim društvom (DVD). Prilagođena je i za rad na web preglednicima na računalu i za mobilne uređaje. Aplikacija je osmišljena zbog postojećih raznih problema u društvu, koje bi ova aplikacija mogla riješiti. Korištenje PWA Aplikacije, hibrid nativne aplikacije i mobilne web aplikacije, omogućeno je i offline. Cilj ovoga rada je bio prikazati da u današnje vrijeme, digitalizacija poslovanja zaista znači sve. Postići jednostavnost korištenja uz razne mogućnosti je zaista idealno rješenje problema. Postojeći problemi ovog društva, koji su potaknuli pisanje ovog rada su bili: uvid kojem vatrogascu je potreba nova oprema ili mu nešto od opreme nedostaje, samo skladištenje opreme, slanje masovnih mailova vezanih uz intervencije i sl.. Ovom aplikacijom postiglo se da svi njezini korisnici (voditelji, skladištari, vatrogasci) na jednostavan način u svakom trenutku mogu imati točnu sliku stanja u dobrovoljnom vatrogasnom društvu .

Ključne riječi: PWA koncept, Dijagram slučaja uporabe (Use case dijagram), HTML, CSS, Angular, MySql, PHP laravel

15. Abstract

Using PHP Laravel for the backend and Angular for the frontend, a progressive web application was designed, which was developed in cooperation with local Volunteer Fire Brigade (LFB). It is also suitable for working on web browsers on computers and mobile devices. The application was designed due to various existing problems in society, which this application could solve. Using the PWA application, that is a hybrid of the native app and mobile web app, is enabled also in offline mode. The aim of this paper was to show that nowadays the digitization of business really means everything. Achieving ease of use with a variety of options is really the ideal solution to the problem. The existing problems of this company, which motivated the writing of this paper were: insight into which firefighter needs new equipment or is missing some of the equipment, equipment storage itself, sending mass emails related to interventions, etc.. With this application, it was achieved that all its users (managers, storekeepers, firefighters) can have an accurate picture of the situation in the society at any time.

Keywords: PWA concept, Use case diagram, HTML, CSS, Angular, MySql, PHP laravel