IZRADA SUSTAVA ZA NADZOR KUĆNIH LJUBIMACA

Kolenda, Andreas

Master's thesis / Specijalistički diplomski stručni

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Split / Sveučilište u Splitu**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:228:903002

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-02**

Repository / Repozitorij:

Repository of University Department of Professional Studies

S V E U Č I L I Š T E U S P L I T U

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Specijalistički diplomski stručni studij Primijenjeno računarstvo

ANDREAS KOLENDA

ZAVRŠNI RAD

IZRADA SUSTAVA ZA NADZOR KUĆNIH LJUBIMACA

Split, rujan 2022.

S V E U Č I L I Š T E U S P L I T U

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Specijalistički diplomski stručni studij Primijenjeno računarstvo

Predmet: Kriptovalute

ZAVRŠNI RAD

Kandidat: Andreas Kolenda

Naslov rada: Izrada sustava za nadzor kućnih ljubimaca

Mentor: Nikola Grgić, viši predavač

Split, rujan 2022.

Sadržaj

Sažetak 1

Summary 1

1. Uvod 2

2. Korištene tehnologije 4

2.1 Ekosustav .NET 4

2.1.1 .NET 6 5

2.1.2 Blazor 6

2.1.2.1 Modeli poslužitelja 8

2.1.3 .NET Standard 11

2.1.3.1 Verzioniranje 12

3. Mikrokontroler ESP32 i senzori 14

3.1 Posluživanje podataka 18

4. Pozadinski (*backend*) dio aplikacije 23

4.1 Entity Framework Core 23

4.2 ER dijagram 24

4.3 .NET Core kôd 26

4.3.1 Kontroleri 27

4.3.2 Repozitoriji 31

4.3.3 Periodička provjera lokacije i detekcije plamena 37

5. Klijentski dio aplikacije (frontend) 40

5.1 Naslovnica 40

5.2 Moji ljubimci 44

5.3 Veterinarske stanice 47

5.4 Statistika 48

5.5 Administracijska ploča 50

6. Prikaz rada sustava 52

7. Zaključak 60

Literatura 61

Sažetak

U ovom radu izrađen je i opisan sustav za praćenje kućnih ljubimaca koji korisniku pruža mogućnost praćenja ljubimca, odnosno prikaza njegove lokacije u stvarnom vremenu. Kako bi se dodatno olakšalo praćenje i briga za ljubimca, podržane su i dodatne funkcionalnosti poput povezivanja senzora temperature i vlažnosti zraka pomoću kojega će korisnik u stvarnom vremenu moći dobiti uvid u uvjete u kojima se njegov ljubimac nalazi. Omogućeno je i povezivanje senzora za detekciju plamena, kako bi se mogla poslati obavijesti da se ljubimac nalazi u neposrednoj opasnosti od požara. Korisnik može definirati sigurnosnu zonu, kako bi u slučaju da ljubimac iziđe iz nje, dobio email obavijest kao upozorenje. Nadalje, omogućeno je i dohvaćanje popisa svih najbližih veterinarskih stanica u željenome krugu, kako bi se korisniku pružile sve potrebne informacije na jednom mjestu.

Ključne riječi: API, Blazor, dotNET 6, izvršno okruženje, mikrokontroler

Summary

Title: Developing a pet tracking system

The main topic of this thesis is the development of a pet tracking system. The system implements features aimed at improving and simplifying pet tracking and pet care in general. One such feature is a connection to air temperature and humidity sensors which provides real-time information about air conditions in the room where the pet is located. Another example is a built-in support for flame detection sensor, so that a user can be promptly notified in case their pet is in danger of a fire. Furthermore, chances of pet escape are reduced with the use of a pet safety zone with a modifiable radius and a notification system for alerting the user via e-mail when the pet leaves the zone. Lastly, the system implements pet clinic support and retrieves data for clinics in user's proximity to save time in case of emergencies.

Keywords: API, Blazor, dotNET 6, runtime, microcontroller

1. Uvod

Vlasnici jednog ili više ljubimaca, jako često se nađu u situaciji da se ljubimac mora ostaviti u kući ili stanu bez nadzora, bilo to zbog posla ili nečega drugog. Uvijek se javlja određena doza nesigurnosti hoće li nešto loše poći po zlu, odnosno hoće li se nešto loše dogoditi dok je ljubimac sam. Web aplikacija koja nudi sve nadzorne informacije je rješenje koje je u takvim situacijama vrlo poželjno. Imati u rukama sigurnost svog ljubimca, čak i kada je riječ o većim fizičkim udaljenostima, je nešto što bi velika većina ljudi odmah objeručke prihvatila.

U ovom završnom radu će biti predstavljena i pobliže objašnjena web aplikacija za nadzor kućnih ljubimaca, odnosno korištenje razvojnog okvira .NET 6 i Blazora u kombinaciji s radnim okvirima CSS, Bulma i Bootstrap. Također, biti će opisan i rad s radnim okvirom Entity Framework Core koji služi za rad s bazom podataka, koja je u ovoj aplikaciji MSSQL baza podataka. Uz to, isprogramiran je i poslužitelj podataka (GPS lokacije i informacije o zraku), u ovome slučaju ESP32 mikrokontroler koji pomoću GPS senzora i senzora za kvalitetu zraka pruža sve potrebne informacije aplikaciji. Aplikacija predstavlja sustav za nadzor kućnih ljubimaca s uključenim prikazom trenutne lokacije ljubimca na Google karti i informacija o kvaliteti zraka prostorije u kojoj se ljubimac nalazi. Naravno, aplikacija podržava i razne dodatne mogućnosti poput slanja obavijesti ukoliko ljubimac izađe iz radijusa postavljene sigurne zone, koje će biti detaljno prikazane i opisane.

Izrađen je prototip hardvera potrebnog za uspješan rad cijelog sustava koji se sastoji od: mikrokontrolera, GPS modula, senzora temperature i vlažnosti zraka te senzora za detekciju plamena. GPS modul i senzori su povezani na mikrokontroler. Prototip je namijenjen ljubimcima za nošenje, npr. oko vrata zakačen na ogrlicu.

Pozadinski dio aplikacije je razvijen korištenjem razvojnog okvira .NET 6 i Entity Framework Core, točnije aplikacijsko programsko sučelje (engl. *Application Programming Interface - API*) preko kojega se odvijaju svi zadaci vezani uz podatke, kao što su npr.: dohvaćanje korisničkih podataka, dohvaćanje svih korisničkih ljubimaca, te njihovo ažuriranje i brisanje, dohvaćanje trenutne lokacije i informacija o kvaliteti zraka, i slično.

Klijentski dio aplikacije je odrađen pomoću razvojnog okvira Blazor koji je također od Microsofta, te odlično djeluje u kombinaciji s .NET-om.

Ovaj rad se sastoji od sedam poglavlja. Nakon sažetka i uvoda, u drugom poglavlju opisan je ekosustav .NET i razvojna okruženja koja su korištena pri izradi ovog završnog rada. U trećem poglavlju je opisan ESP32 mikrokontroler te njegova konfiguracija kako bi postao poslužitelj koji će aplikaciji posluživati potrebne podatke očitane sa senzora. U četvrtom poglavlju je opisana baza podataka aplikacije, model podataka u toj bazi i pozadinski dio aplikacije, odnosno API s njegovim krajnjim točkama (eng. *endpoints*) uz opisanu komunikaciju s navedenom bazom podataka te dohvat podataka iz iste. Zatim u petom poglavlju je opisan klijentski dio aplikacije i njegova komunikacija s pozadinskim dijelom. Šesto poglavlje prikazuje korištenje aplikacije te njene osnovne i dodatne funkcionalnosti, kako samog korisnika, tako i administratora aplikacije. Naposljetku, sedmo poglavlje ukratko zaključuje cijeli rad.

2. Korištene tehnologije

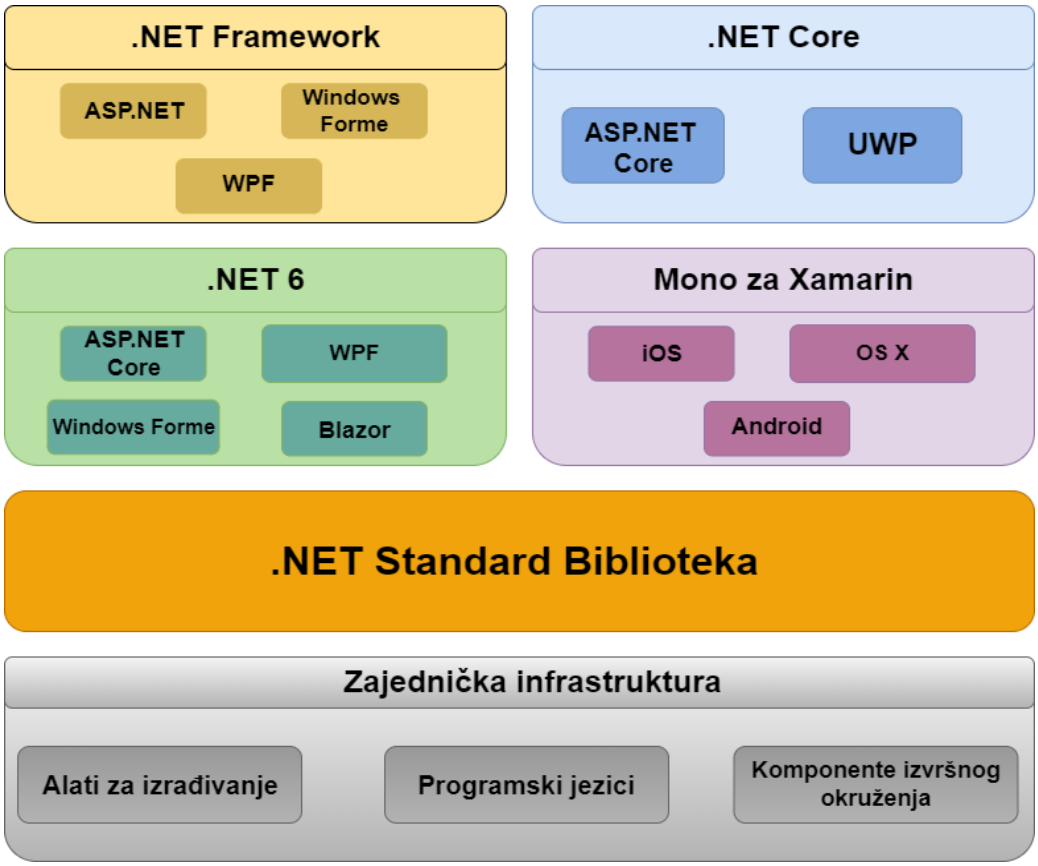
U ovome poglavlju će biti ukratko objašnjene sve korištene tehnologije za izradu ovoga projekta. Ovisno o vremenu izrade web aplikacije, te najviše korištenih tehnologija u to vrijeme, web aplikacije mogu biti izrađene pomoću više različitih programskih jezika i tehnologija. Razvojna okruženja Visual Studio i Visual Studio Code i njihovi alati su najčešće korišteni, dok trenutno jedna od najpopularnijih tehnologija za izradu pozadinskih (eng. *backend*) dijelova web aplikacije je ekosustav .NET koji uključuje izvršna okruženja poput .NET (Core) i ASP.NET. Iako, svoje mjesto pri vrhu tehnologija korištenih za izradu sučelja, odnosno klijentskih dijelova (eng. *frontend*) web aplikacije tvrtka Microsoft pokušava zauzeti sa svojom Blazor tehnologijom.

2.1 Ekosustav .NET

.NET je besplatna, platforma otvorenog kôda za programere, odnosno kolekcija programskih jezika i biblioteka koji zajedno omogućuju izgradnju svih vrsta aplikacija na svim platformama. .NET je razvijen od strane Microsofta i većina ljudi pri spomenu .NET-a pomisli na .NET Framework i Visual Studio, iako je zapravo .NET Framework samo dio ekosustava .NET. Trenutno, u 2022. godini, ekosustav .NET sadrži različita izvršna okruženja (engl. *runtimes*) kao što su:

- već spomenuti .NET Framework – samo za Windows operativne sustave.
- .NET Core – više-platformsko, kompatibilno sa starijim verzijama.
- .NET 6 - spoj najboljih stvari iz .NET Frameworka i .NET Corea u jedno.
- Mono za Xamarin – više-platformsko orijentirano mobilnoj tehnologiji. [1]

Sva navedena izvršna okruženja implementiraju .NET Standard biblioteku. .NET Standard biblioteka sadrži specifikacije .NET API-ja koji imaju implementaciju za svako izvršno okruženje. Osmišljeno je na ovaj način kako bi se kôd izrađen za jedno izvršno okruženje mogao izvršiti na svakom drugom izvršnom okruženju. Sva izvršna okruženja koriste alate za izrađivanje i infrastrukturu kako bi sastavili (eng. *compile*) i pokrenuli kôd, što uključuje programske jezike poput C#, F#, Visual Basic, sastavljače (eng. *compilers*) poput Rosyln, te alate za izrađivanje poput MS Build ili (Core) CLR. Međusobna povezanost svih komponenti je prikazana na slici 1.

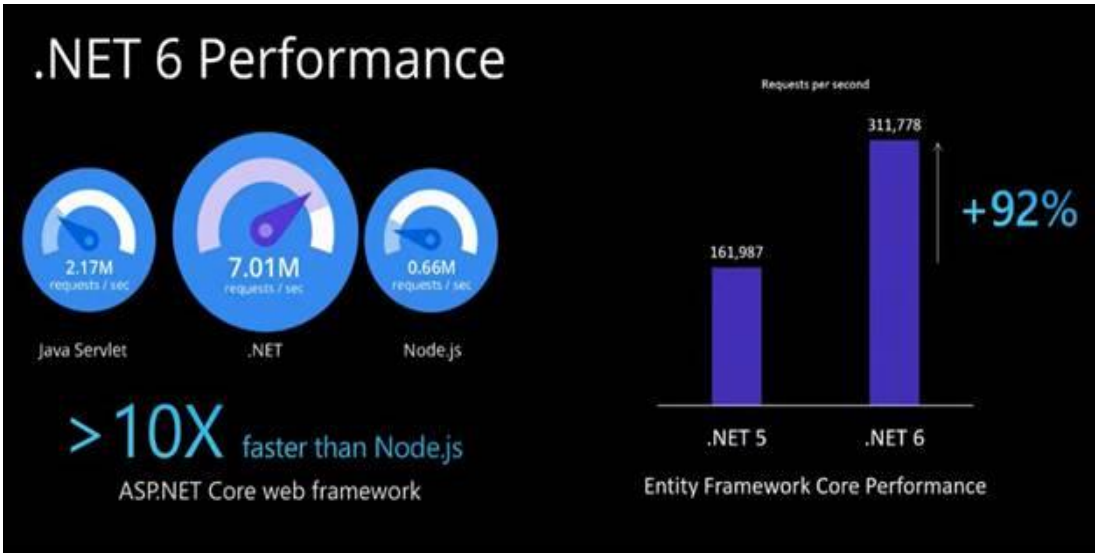


Slika 1 – Ekosustav .NET

2.1.1 .NET 6

.NET 6 je najnovije izvršno okruženje u ekosustavu .NET. Objavljen je u studenom 2021. godine. .NET 6, za razliku od svog prethodnika, je znatno poboljšana verzija, kako u smislu performansa tako i u smislu novih značajki. Najveća značajka .NET 6 je ujedinjavanje platformi. Prije dolaska .NET 6, .NET Framework i .NET Core su bila dva odvojena proizvoda. .NET Framework je bio originalni .NET koji nije bio otvorenog kôda (eng. *open-source*), dok je .NET Core verzija otvorenog kôda .NET-a napisana iz početka i radi na svim platformama: Windowsu, Linuxu i Macu. Od .NET 6 nadalje, postoji će samo jedna verzija .NET-a te se i zove samo .NET. Objedinjeno je izvršno okruženje koje razvojnim programerima omogućava izradu aplikacija za oblak (eng. *cloud*), web, desktop, mobilne uređaje, igre, Internet stvari (eng. *Internet of Things – IoT*) i umjetnu inteligenciju. Sve navedene podplatforme dijele zajedničke biblioteke, API-je i temeljenu infrastrukturu uključujući i programske jezike i *compiler*e.

.NET 6 je također najbrže izvršno okruženje do sada, čak među najbržima od svih, ne samo Microsoft-ovih okruženja [2]. Usporedba brzine .NET 6 s drugim izvršnim okruženjima je prikazana na slici 2.



Slika 2 - .NET 6 performanse [2]

Kao što je vidljivo, .NET 6 je čak 10 puta brži od konkurentnog izvršnog okruženja Node.jsa.

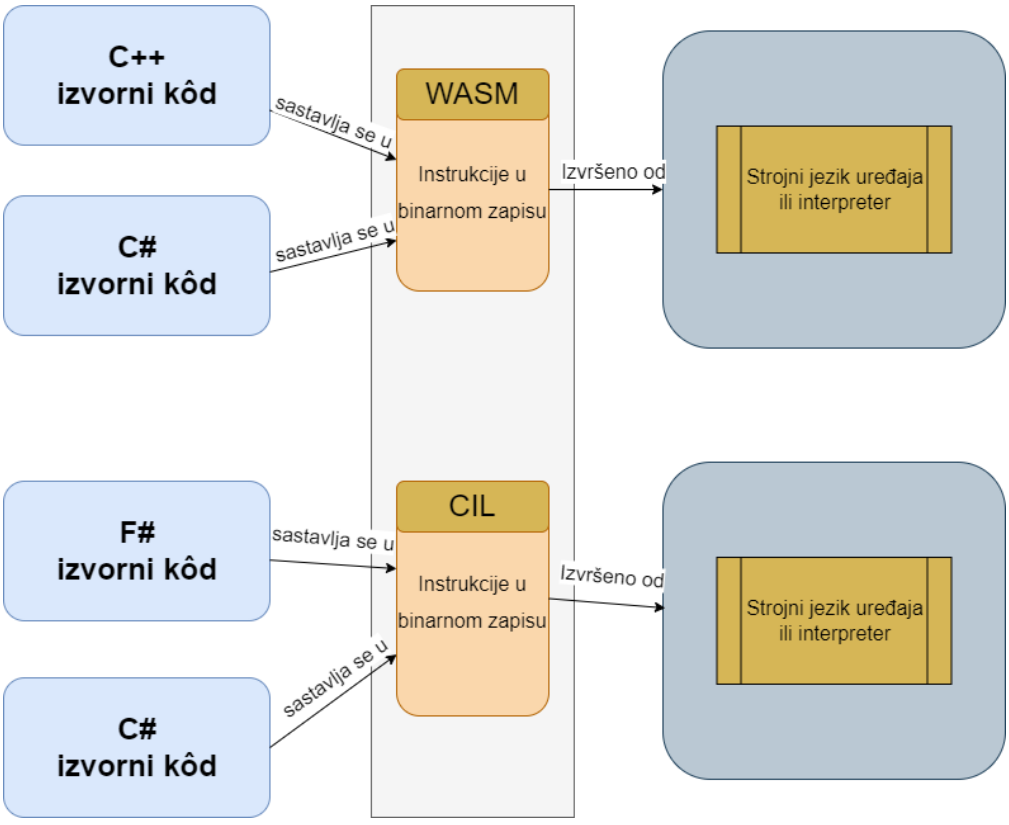
C# 10 verzija programskog jezika i F# 6 su najnovije verzije C#-a i F#-a, te dolaze zajedno s .NET 6. C# 10 nudi razna poboljšanja, ali je najveći naglasak bio na jednostavnost korištenja. Neke od glavnih značajki ove verzije su: korištenje globalnih *using* direktive, *record structs*, imenski prostor (eng. *namespace*) na razini datoteke, konstantni interpolirani *string*-ovi, poboljšanja vezana za *lambda* funkcije i provjeravanje *null* vrijednosti parametara. Velika novost koja je također uvedena je tzv. „Hot Reload“ koji nudi programeru mogućnost mijenjana izvornog kôda dok je pokrenut, te na taj način može vidjeti vizualne promjene u stvarnom vremenu bez potrebe za ponovnim učitavanjem aplikacije.

2.1.2 Blazor

Blazor je otvorenog kôda i više-platformski razvojni okvir za izradu interaktivnih klijentskih web sučelja koji to postiže koristeći .NET. Ime mu je nastalo kombinacijom dvije riječi: browser (preglednik) i Razor (.NET-ov mehanizam za generiranje HTML prikaza). Blazor omogućuje pisanje web sučelja koristeći C# programski jezik kao da je riječ o pozadinskom dijelu aplikacije, poput API-ja, što je vrlo bitan i veliki napredak u

tehnološkom smislu s obzirom da sada programer ne mora nužno imati znanje programskog jezika JavaScript kako bi izradio web sučelje. Umjesto pisanja JavaScript kôda, koristi se C# pomoću Razor sintakse, s čime je programeru omogućeno korištenje i poznavanje samo jednog jezika i za pozadinski i za klijentski dio aplikacije. Kôd napisan u Blazoru može biti izvršen kako na serveru, tako i na klijentskom pregledniku.

Blazor C# kôd izvršava na pregledniku pomoću WebAssemblya (WASM). WASM je set instrukcija dizajniran tako da bi se mogao izvršavati na bilo kojem domaćinu (eng. *host*) koji je sposoban protumačiti (eng. *interpret*) navedene instrukcije. *Host* to može odraditi čitajući i izvršavajući binarne datoteke interpretirane ili izravnim prevođenjem u strojni jezik specifičan za uređaj, kao i interpretiranje kôda u CIL - *Common Intermediate Language* [3] (slika 3).



Slika 3 – Interpretiranje kôda

Blazor ne zahtjeva instaliran .NET na klijentskom računalu kako bi radio kroz WebAssembly. Podržanost WebAssemblya na različitim web preglednicima je prikazana na tablici 1.

Tablica 1 - WASM podržani web preglednici

Web preglednik	Od verzije
Android preglednik	67
Chrome	57
Chrome za Android	57
Edge	16
Firefox	52
Firefox za Android	52
Safari	11
iOS Safari	11
Opera	44
Opera za Android	43

2.1.2.1 Modeli poslužitelja

Web aplikacija Blazor može biti poslužena (eng. *hosted*) na dva načina: Blazor WebAssembly i Blazor Server.

U Blazor Server način, kao što se od imena da naslutiti, aplikacija je izvršena i vrti se na serveru unutar ASP.NET Core aplikacije. Ažuriranja korisničkog sučelja, upravljanje događajima (eng. *event-handling*) i JavaScript pozivi su odrađeni preko SignalR veze koristeći WebSockets protokol. Stanje na poslužitelju povezano sa svakim povezanim klijentom naziva se krug. Krugovi nisu vezani uz određenu mrežnu vezu i mogu tolerirati privremene prekide mreže i pokušaje klijenta da se ponovno poveže s poslužiteljem kada se veza izgubi. Na klijentskoj strani, Blazor skripta (blazor.server.js) uspostavlja SignalR vezu s poslužiteljem. Skripta se poslužuje aplikaciji na strani klijenta iz ugrađenog resursa u zajedničkom razvojnom okviru ASP.NET Core. Aplikacija na strani klijenta odgovorna je za održavanje i vraćanje stanja aplikacije prema potrebi [4]. Neke prednosti Blazor Server načina posluživanja su:

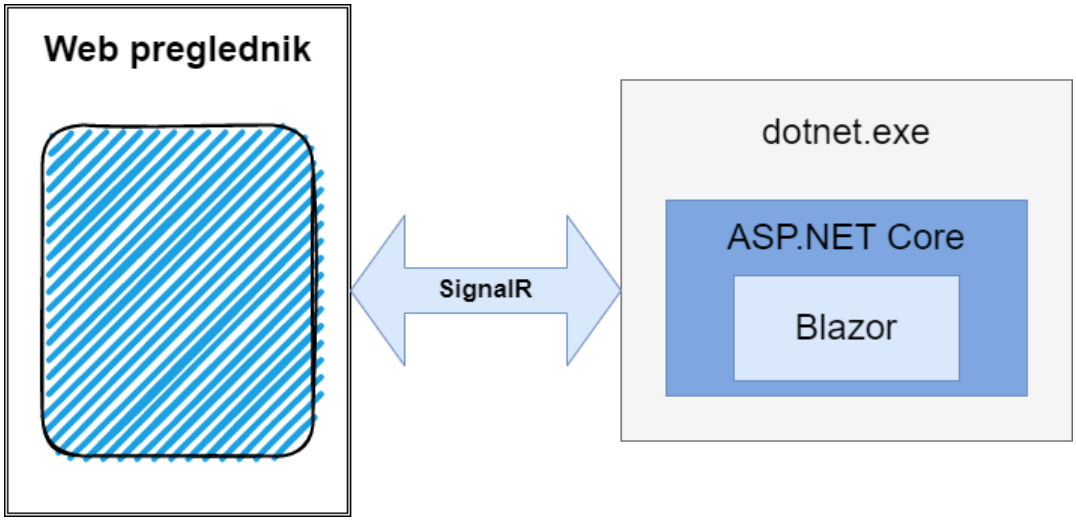
- Veličina preuzimanja znatno je manja od aplikacije Blazor WebAssembly, te se aplikacija se učitava puno brže.

- Aplikacija u potpunosti iskorištava mogućnosti poslužitelja, uključujući i upotrebu .NET Core API-ja.
- Podržani su tanki (slabiji) klijenti. Na primjer, aplikacije Blazor Server rade s preglednicima koji ne podržavaju WebAssembly i na uređajima s ograničenim resursima.
- Kôd aplikacije, uključujući i kôd komponenata aplikacije, ne poslužuje se klijentima.

Dok su neke mane:

- Obično postoji veća latencija. Svaka interakcija korisnika uključuje mrežni skok.
- Nema izvanmrežne podrške. Ako veza klijenta ne uspije, aplikacija prestaje raditi.
- Skaliranje serverskih resursa je potrebno u slučaju velikog broja klijenata i rasta aplikacije.

Način rada Blazor Server posluživanja je prikazan na slici 4.



Slika 4 - Arhitektura Blazor Server

Blazor WebAssembly aplikacije se pokreću na klijentskoj strani u web pregledniku na izvršnom okruženju .NET temeljenom na WebAssemblyu. Aplikacija Blazor, njene ovisnosti (eng. *dependencies*) i izvršno okruženje .NET preuzimaju se u web preglednik.

Aplikacija se izvršava izravno na niti korisničkog sučelja web preglednika, a ažuriranja korisničkog sučelja i upravljanje događajima odvijaju se unutar istog procesa. Sredstva aplikacije raspoređuju se (eng. *deploy*) kao statične datoteke na web poslužitelj ili uslugu koja može posluživati statički sadržaj klijentima. Prikaz načina rada Blazor WASM-a je prikazan na slici 5.

Slika 5 - Arhitektura Blazor WebAssembly

Aplikacija Blazor WASM izrađena bez pozadinskog dijela aplikacije (ASP.NET Core aplikacije) koja bi posluživala njene datoteke se naziva samostalna (eng. *standalone*) aplikacija Blazor WASM, dok u slučaju da aplikacija ima pozadinski dio koji će posluživati datoteke naziva se *hosted* aplikacija Blazor WASM. U projektu za izradu sustava za praćenje kućnih ljubimaca, izabrana je *hosted* aplikacija Blazor WASM. Korištenjem *hosted* Blazor WASM dobije se puno *full-stack* razvojno iskustvo s .NET-om, uključujući i mogućnost dijeljenja kôda između klijentskih i poslužiteljskih aplikacija, podršku za renderiranje unaprijed i integraciju s MVC (*Model-View-Controller*) i Razor Pages. Za komunikaciju s pozadinskom poslužiteljskom aplikacijom, klijentska aplikacija može koristiti razne razvojne okvire za razmjenu poruka i protokola, poput web API-ja, gRPC-web i SignalRa. Skripta `blazor.webassembly.js` koju pruža *hosted* aplikacija Blazor WASM, te ona upravlja preuzimanjem izvršnog okvira .NET, aplikacije i datoteka o kojima ovisi. Također upravlja i inicijalizacijom izvršnog okvira kako bi se aplikacija mogla pokrenuti.

Prednosti Blazor WASM modela poslužitelja uključuju:

- Ne postoji ovisnost .NET poslužitelja nakon što se aplikacija preuzme s njega, tako da aplikacija ostaje funkcionalna i u slučaju da poslužitelj ode s mreže.
- Klijentski resursi su u potpunosti iskorišteni.
- Rad je prebačen s poslužitelja na klijenta, pa skaliranje poslužitelja nije potrebno kao npr. u Blazor Server modelu.
- ASP.NET Core web poslužitelj nije potreban za posluživanje aplikacije.

No, Blazor WASM model dolazi i s nekakvim ograničenjima, od kojih su:

- Aplikacija je ograničena na mogućnosti web preglednika
- Potreban je sposoban klijentski hardver i softver (npr. podrška za WebAssembly je obavezna)
- Veličina preuzimanja je veća, a učitavanje aplikacija traje duže.

2.1.3 .NET Standard

Različita izvršna okruženja koriste različite biblioteke klasa. Na primjer, .NET Framework koristi biblioteku klasa .NET Framework, dok .NET Core sadrži vlastitu biblioteku klasa, kao i Xamarin sa svojom bibliotekom klasa. Na ovaj način je jako teško dijeliti kôd između različitih izvršnih okruženja jer ne koriste iste API-je. Kako bi riješio taj problem, Microsoft je 2016. godine predstavio .NET Standard biblioteku koja predstavlja skup službenih specifikacija koje određuju koje API-je je moguće koristiti te je implementiran od strane svih izvršnih okruženja. .NET Standard je nasljednik Biblioteka prenosivih klasa (engl. *Portable Class Libraries* - PCL) [5]. Određena izvršna okruženja implementiraju određenu verziju .NET Standarda (implementirajući određeni skup API-ja), pa tako npr. .NET Framework 4.6.2 implementira .NET Standard 2.0, dok .NET 6 implementira .NET Standard 2.1 koji je zadnja verzija .NET Standarda.

Kao što je već spomenuto, biblioteka .NET Standard je nastala kao nasljednik Biblioteka prenosivih klasa s glavnim postignućem, a to je neovisnost o drugoj platformi. Osnovne i najbitnije razlike između .NET Standarda te njegovog prethodnika, Biblioteka prenosivih klasa su prikazane u tablici 2.

Tablica 2 - Usporedba .NET Standarda i PCL-a

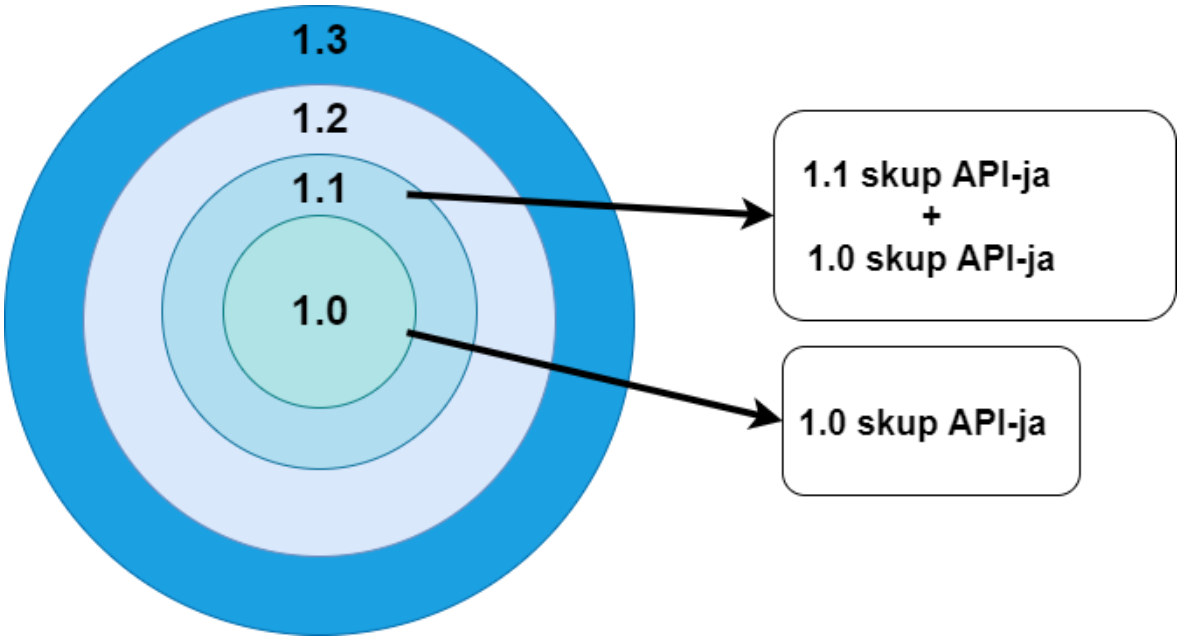
.NET Standard	Biblioteke prenosivih klasa
Pomno odabrani set aplikacijskih programskih sučelja (API-ja) od strane Windowsa	Aplikacijska programska sučelja su definirana od strane platforme za koju se biblioteka izrađuje
Neovisan o drugoj platformi	Ograničena lista platformi

Microsoft je unutar izvršnog okruženja .NET 5 spojio .NET Core i .NET Standard, što znači da sada postoji zajednička baza kôda za sva .NET radna opterećenja (eng. *workloads*), nebitno radi li se o aplikacijama za stolna računala (desktop aplikacije), uslugama u oblaku ili mobilnim aplikacijama. Naime, bitno je naglasiti da .NET 5 ne zamjenjuje .NET Standard. Općenito, moguće je koristiti:

- .NET Standard 2.0 za dijeljenje kôda između .NET Frameworka i svih ostalih izvršnih okruženja
- .NET Standard 2.1 za dijeljenje kôda između izvršnih okruženja Xamarin i .NET Core 3.x
- .NET 6 (i buduće verzije) za svo dijeljenje kôda novijih izvršnih okruženja.

2.1.3.1 Verzioniranje

Svaka verzija .NET Standarda ima skup API-ja (npr. `System.String` i `System.IO`) i uključuje sve API-je iz prethodnih verzija tako da je kompatibilan sa starijim verzijama, odnosno verzije .NET Standarda se grade jedna na drugu kao što je prikazano na slici 6. Kao što je već spomenuto, određene verzije izvršnih okruženja .NET implementiraju određene verzije .NET Standarda.



Slika 6 - NET Standard verzioniranje

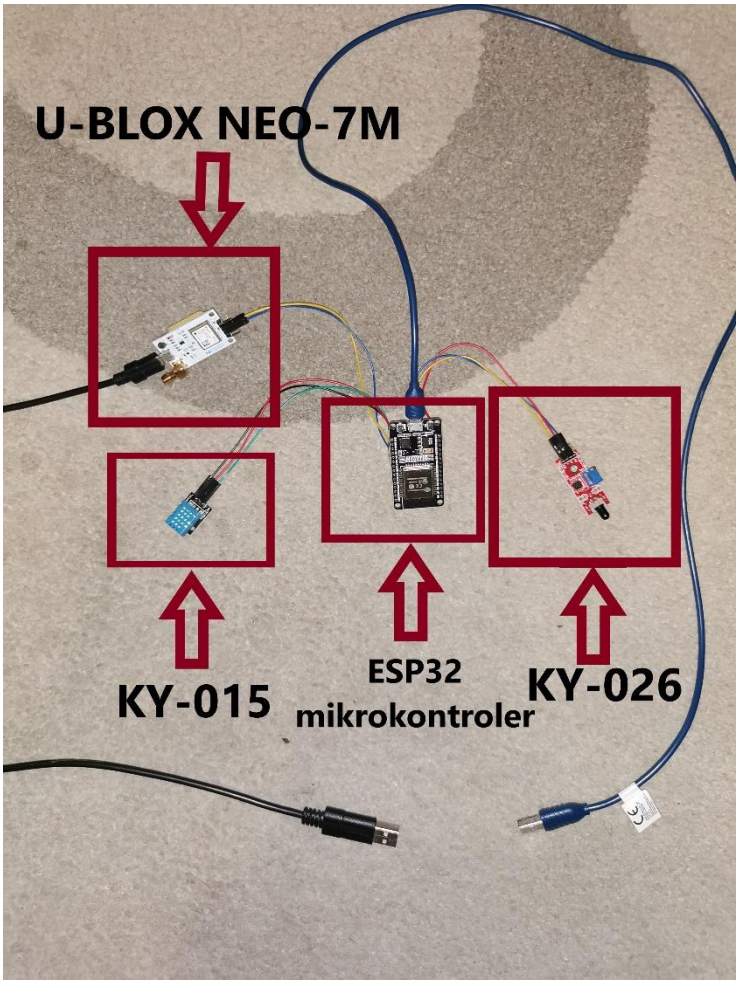
Niže verzije .NET Standarda podržavaju veći broj platformi, što znači da ukoliko je cilj što veća podržanost programa, potrebno je ciljati na najnižu moguću verziju. Suprotno tome, više verzije .NET Standarda omogućavaju korištenje većeg broja API-ja.

3. Mikrokontroler ESP32 i senzori

Mikrokontroler ESP32 u ovome radu služi kao poslužitelj podataka aplikaciji. Čita, odnosno prikuplja podatke sa senzora koji su spojeni na njega, te ih poslužuje na *endpoint*u poput API-ja. Mikrokontroler ESP32 već ima ugrađenu podršku za Bluetooth i Wi-Fi povezivanje, što omogućuje mikrokontroler da poslužuje podatke na IP adresi koju je dobio na lokalnoj mreži. Senzori i moduli koji su korišteni te spojeni na mikrokontroler su:

- KY-015 – senzor koji očitava temperaturu i vlažnost zraka
- KY-026 - senzor za detekciju plamena, šalje signal u slučaju da detektira otvoreni plamen
- U-BLOX NEO-7M – GPS modul za očitavanje lokacije

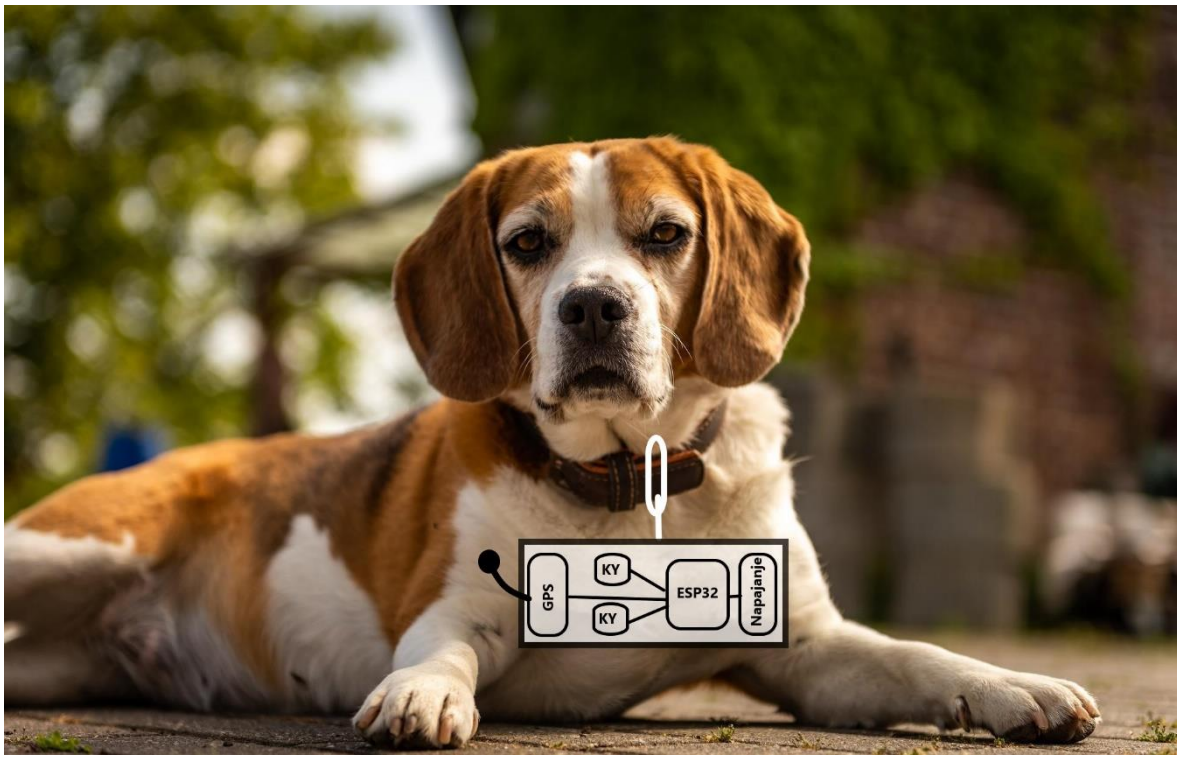
Slika 7 prikazuje prototip hardvera potrebnog za rad sustava, odnosno mikrokontroler povezan sa svim navedenim dodacima.



Slika 7 - Mikrokontroler i dodaci - prototip

ESP32 je serija jeftinih sustav na čipu (eng. *system on a chip – SoC*) mikrokontrolera male snage s integriranim Wi-Fi i Bluetooth modulom. ESP32 obitelj mikrokontrolera uključuje čipove ESP32-D0WDQ6, ESP32-D2WD, ESP32-S0WD, i sustava u paketu (eng. *system in package - SiP*) ESP32-PICO-D4. ESP32 je izrađen i razvijen od strane kineske tvrtke Espressif Systems. ESP32 mikrokontroleri su nasljednici ESP8266 mikrokontrolera. ESP32 mikrokontroleri su jako dobro integrirani s ugradbenim antenskim prekidačima, RF balun transformatorima, pojačalom snage, niskošumnim prijemnim pojačalom, filtrima i modulima za upravljanje napajanjem. [6] Kao što je vidljivo na slici 7, dva su USB-A priključka za napajanje. Jedan za mikrokontroler i drugi za GPS modul. GPS modul se može napajati i preko mikrokontrolera, ali s obzirom da mu je potrebno maksimalno što mikrokontroler može pružiti, puno je bolja i sigurnija opcija imati posebno zasebno za GPS modul. Ostali senzori se napajaju preko mikrokontrolera.

Prototip je namijenjen za ljubimce, odnosno da budu nošeni konstantno. Jedan primjer nošenja je zakačiti prototip za ogrlicu ljubimcu, kao što je prikazano na slici 8.



Slika 8 - Primjer prototipa hardvera na ljubimcu

Za programiranje mikrokontrolera u ovome projektu koristilo se razvojno okruženje Arduino IDE verzije 1.8.19. Prvo se dodaju potrebne biblioteke za rad te se postavljaju potrebne globalne varijable kao što je prikazano u ispisu 1.

```
#include <Arduino.h>
#include <WiFi.h>
#include <ESPAsyncWebServer.h>
#include <HardwareSerial.h>
#include <TinyGPSPlus.h>
#include "AsyncJson.h"
#include "ArduinoJson.h"
#include "DHT.h"

#define DHTPIN 4
#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);
AsyncWebServer server(80);
HardwareSerial mySerial(2);
TinyGPSPlus gps;

int Digital_Pin = 14;

float flat, flon;
String gpsUuidStr = " 77853f27-7ca1-42b3-968d-73b88c36568a";
String airSensorUuidStr = "21a7f0a0-98d2-4cf9-9b2c-5d611bbc5263";
String flameDetectionUuidStr = "b9819153-26ce-4ad8-bbcf-
4faaf09c5e22";

const char* ssid = "Andy";
const char* password = "abcd1234";
IPAddress local_IP(192, 168, 43, 100);
IPAddress gateway(192, 168, 43, 1);
IPAddress subnet(255, 255, 255, 0);
```

Ispis 1 - Početni kôd za postavljanje ESP32 mikrokontrolera

Bitne stavke prikazane na slici iznad su: postavljanje ulaz/izlaz-a opće namjene (eng. *General-purpose input/output* – GPIO), odnosno PIN-a na kojemu je spojen senzor s mikrokontrolerom. U ovome slučaju senzor KY-015 (koji spada u verziju 11 DHT serije senzora) je spojen na GPIO 4 na ESP32 mikrokontroleru. Mikrokontroler se također postavlja da bude poslužitelj te poslužuje na portu 80. Na GPIO 14 je spojen senzor za detekciju plamena (KY-026). Jedinstvene identifikacijske brojeve je moguće odraditi i na način da ih program nasumično generira, no radi jednostavnosti ovdje su postavljeno statički. Nadalje slijede postavke za spajanje mikrokontrolera na lokalnu bežičnu mrežu sa statičkom IP adresom, te na kraju je metoda koja vraća HTTP status kôd 404 u slučaju da se pošalje HTTP zahtjev na krajnju točka (*endpoint*) koja nema implementaciju. Pri početku rada mikrokontrolera, prvi dio kôda započinje spajanje na bežičnu mrežu s prethodno postavljenim podacima (ispis 2).

```
void initWiFi() {
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  Serial.print("Connecting to WiFi ..");

  while (WiFi.status() != WL_CONNECTED) {
    Serial.print('.');
    delay(1000);
  }

  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());
}

void setup() {
  pinMode(Digital_Pin, INPUT);

  Serial.begin(9600);
  mySerial.begin(9600, SERIAL_8N1, 17, 16);

  if (!WiFi.config(local_IP, gateway, subnet)) {
    Serial.println("STA Failed to configure");
  }
  initWiFi();
```

Ispis 2 - Spajanje za bežičnu mrežu - mikrokontroler

3.1 Posluživanje podataka

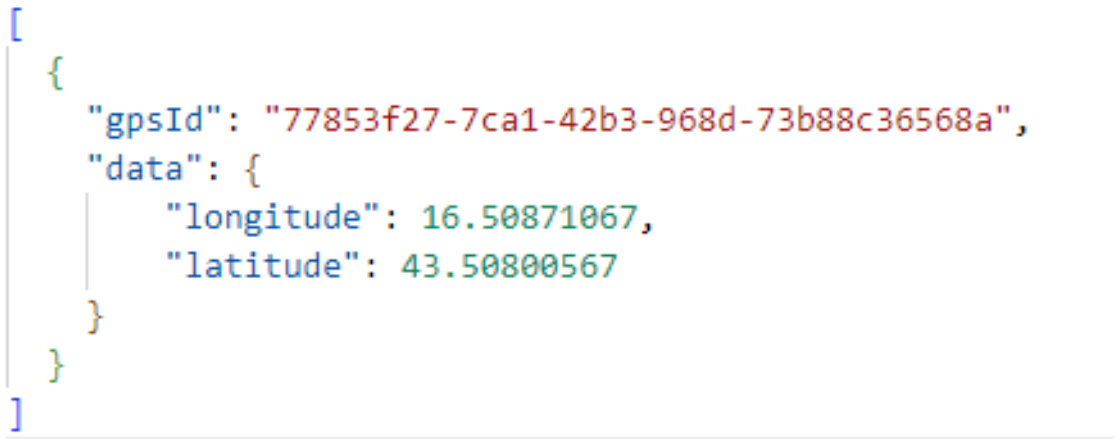
Kao što je već navedeno, mikrokontroler služi kao poslužitelj podataka sa senzora za aplikaciju. Izrađene su tri krajnje točke s kojih se pomoću HTTP zahtjeva mogu dobiti traženi podaci. Kôd koji postavlja krajnju točku, te primjer rezultata dohvaćanja GPS podataka s krajnje točke `~/gps-data` su prikazani u ispisu 3 i slici 9.

```
server.on("/gps-data", HTTP_GET, [](AsyncWebServerRequest * request)
{
    StaticJsonDocument<256> gpsDoc;
    JsonArray rootArray = gpsDoc.to<JsonArray>();
    JsonObject rootObject = rootArray.createNestedObject();

    rootObject["gpsId"] = gpsUuidStr;
    JsonObject gpsData = rootObject.createNestedObject("data");
    gpsData["longitude"] = gps.location.lng();
    gpsData["latitude"] = gps.location.lat();

    String response;
    serializeJson(gpsDoc, response);
    request->send(200, "application/json", response);
  });
```

Ispis 3 - Kôd za postavljanje krajnje točke za GPS podatke



Slika 9 - GPS podaci s mikrokontrolera

Iz podataka je moguće pročitati zemljopisnu dužinu i širinu, odnosno koordinate koje su prikupljene s GPS modula spojenog na Arduino ESP32 mikrokontroler, te jedinstveni identifikacijski broj koji označava GPS modul s kojega su ti podaci prikupljeni. Na isti način je moguće dohvatiti podatke o temperaturi sa senzora KY-015 kao što je prikazano u ispisu 4 i slici 10.

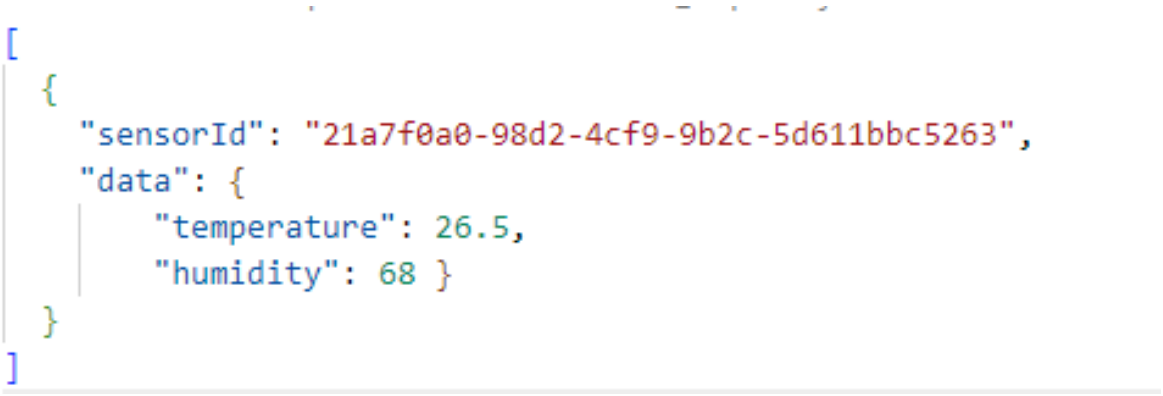
```
server.on("/room-data", HTTP_GET, [](AsyncWebServerRequest * request)
{
    StaticJsonDocument<256> roomDoc;
    JsonArray rootArray = roomDoc.to<JsonArray>();
    JsonObject rootObject = rootArray.createNestedObject();

    rootObject["sensorId"] = airSensorUuidStr;
    JsonObject airData = rootObject.createNestedObject("data");

    airData["temperature"] = dht.readTemperature();
    airData["humidity"] = dht.readHumidity();

    String response;
    serializeJson(roomDoc, response);
    request->send(200, "application/json", response);
  });
```

Ispis 4 - Kôd za postavljanje krajnje točke za podatke o zraku



Slika 10 - Podaci o zraku s mikrokontrolera

Završno, na vrlo sličan način je odrađeno pružanje i dohvaćanje podataka sa senzora za detekciju plamena, kao što se vidi u ispisu 5 i slici 11.

```
server.on("/flame-detection", HTTP_GET, [](AsyncWebServerRequest *
request) {
    StaticJsonDocument<256> flameDoc;
    JsonArray rootArray = flameDoc.to<JsonArray>();
    JsonObject rootObject = rootArray.createNestedObject();

    rootObject["sensorId"] = flameDetectionUuidStr;
    JsonObject                  flameDetectionData                  =
rootObject.createNestedObject("data");

    int flameDetected;
    flameDetected = digitalRead(Digital_Pin);

    if(flameDetected == 1)
    {
      flameDetectionData["flameDetected"] = true;
    }
    else
    {
      flameDetectionData["flameDetected"] = false;
    }

    String response;
    serializeJson(flameDoc, response);
    request->send(200, "application/json", response);
  });
```

Ispis 5 - Kôd za postavljanje krajnje točke za podatke o detekciji plamena

```
[
  {
    "sensorId": "b9819153-26ce-4ad8-bbcf-4faaf09c5e22",
    "data": {
        "flameDetected": false
        }
    }
]
```


Slika 11 - Podaci o detekciji plamena

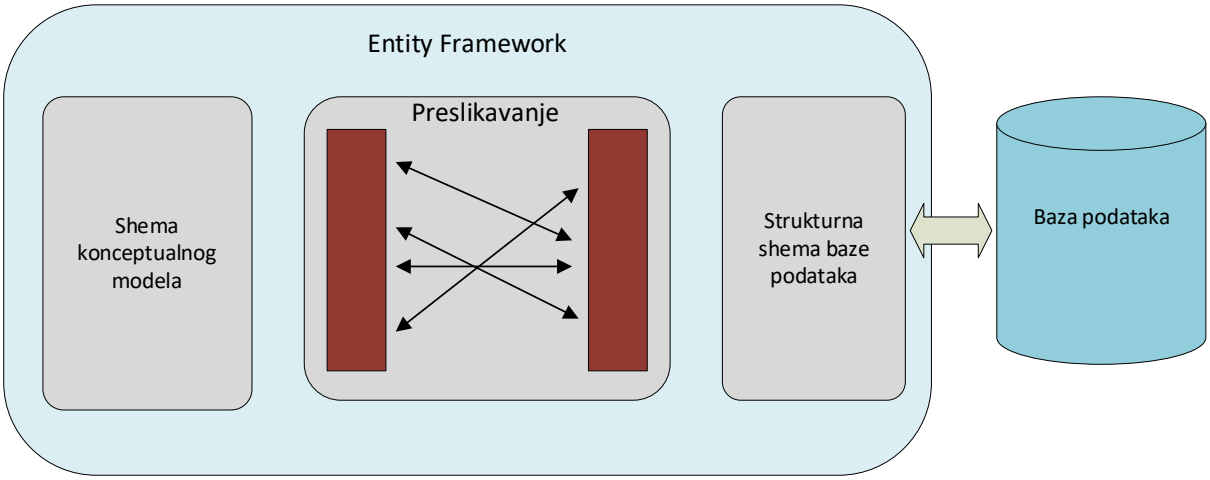
Vidljivo je da se podaci poslužuju kao niz objekata, što omogućuje ispis podataka svih senzora s jednim HTTP zahtjevom u slučaju da ih ima više, što u ovom projektu nije slučaj, pa je vraćeni rezultat uvijek jedan objekt.

4. Pozadinski (*backend*) dio aplikacije

Backend dio aplikacije je dio u kojem se obavljaju sve funkcionalnosti i logika aplikacije. To naravno uključuje i rad s bazom podataka kao i samu bazu podataka. Naglasak je na podatke te rad s njima, što naravno dovodi i do toga da se radi najviše na logici i osmišljavaju te optimiziranju cijelog rada sustava i svih njegovih zadataka. Iz tog razloga se često promijene ne vide grafički toliko koliko npr. u klijentskom dijelu aplikacije gdje je skoro svaka promjena vezana za nekakvo grafičko sučelje. Jako je bitno dobro osmisliti bazu podataka i temelje aplikacije, jer mijenjanje tih stvari kasnije može koštati značajno vremena.

4.1 Entity Framework Core

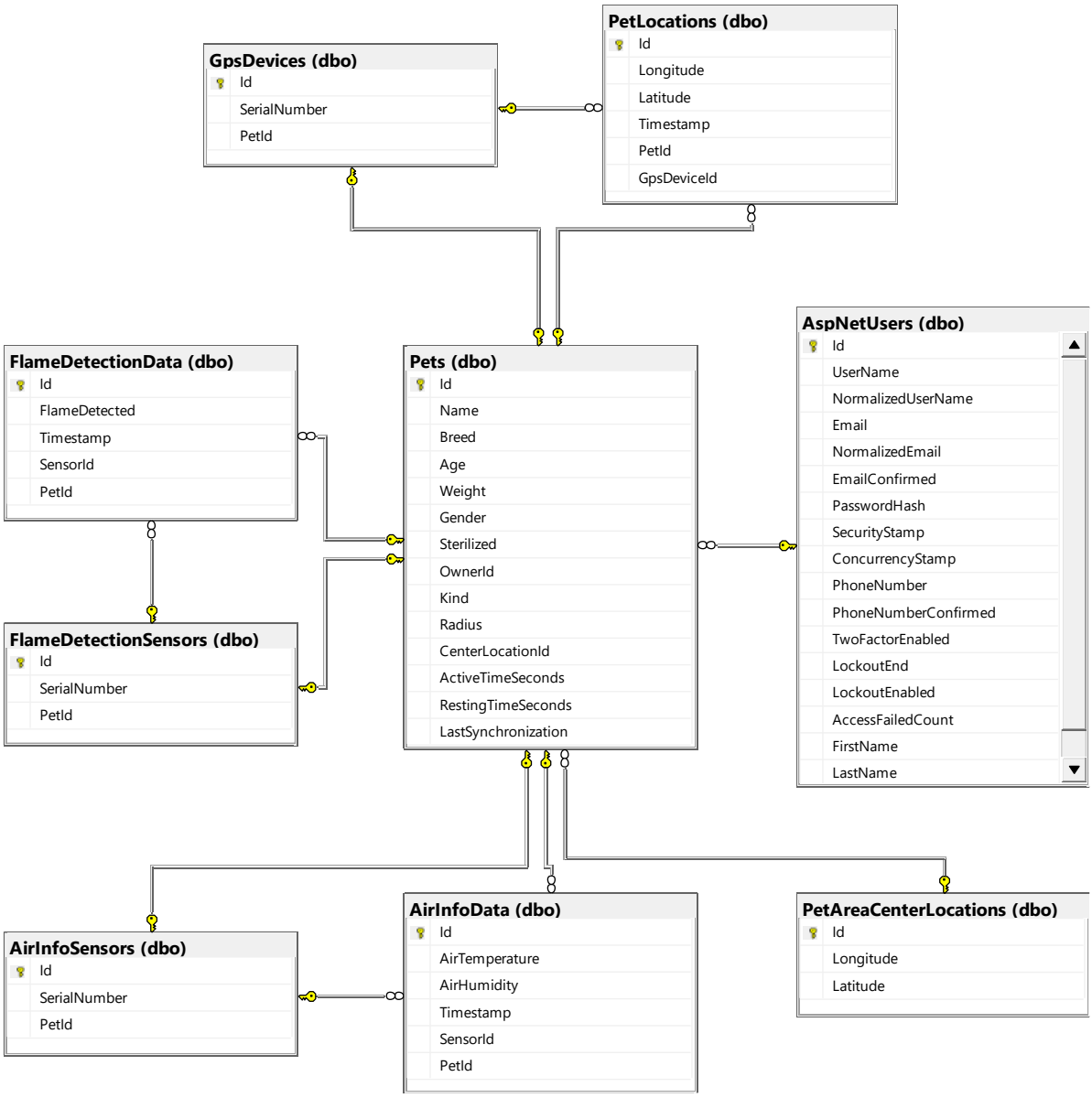
Za komunikaciju i rad s bazom podataka pomoću kôda u ovome projektu je korišten objektno-relacijski okvir mapiranja Entity Framework Core (EF Core) namijenjen za .NET, što je najnovija verzija Microsoft-ovog Entity Frameworka. EF Core podržava upite LINQ, praćenje promjena, ažuriranja i migracije shema. Microsoft ga je predstavio 2016. godine zajedno s web-okvirom ASP.NET Core 1.0 i izvršnim okruženjem .NET Core 1.0. Originalno je bio nazvan Entity Framework 7, ali je preimenovan kako bi ukazao na to da je riječ o kompletno novom projektu koji je pisan iz početka, a ne nova samo nova verzija, niti da će zamijeniti već postojeći okvir objektno-relacijskog mapiranja Entity Framework 6. Namijenjen je da bude lagana, proširiva, otvorenog kôda i više-platformska verzija popularne tehnologije pristupa podacima Entity Framework. Radi s većim brojem različitih baza podataka, kao što su: SQL Server/SQL Azure, SQLite, Azure Cosmos DB, MySQL, PostgreSQL i mnogi drugi. Okvir objektno-relacijskog mapiranja EF Core pruža nove mogućnosti koje neće biti implementirane u EF6, dok nisu ni sve mogućnosti EF6 implementirane u EF Coreu. Trenutno najnovija verzija je EF Core 6.0 koja je ujedno i korištena u izradi ovoga projekta. Jednostavan primjer preslikavanja, odnosno mapiranja koje obavlja objektno-relacijski okvir mapiranja Entity Framework je prikazan na slici 12.



Slika 12 - Entity Framework preslikavanje/mapiranje

4.2 ER dijagram

ER dijagram prikazuje relacije između entiteta u bazi podataka. Jednostavan je za korištenje, odnosno razumijevanje i zbog toga ga mogu razumjeti ljudi raznovrsnih zanimanja. Iz tog razloga je odličan izbor kao alat za komunikaciju između dizajnera baze podataka, korisnika i programera u ranim fazama razvoja baze podataka. Na slici 13 je prikazan ER dijagram sustava za praćenje ljubimaca kako bi se prikazale sve korištene tablice u projektu i njihova međusobna povezanost.



Slika 13 - ER dijagram

Kao što je vidljivo, glavni poslovni entitet predstavlja ljubimac koji je povezan sa svim ostalim entitetima poput senzora, lokacija i korisnika. Pomoću ovakvog dizajna baze podataka, pomoću ljubimca je moguće doći do svih podataka o tome ljubimcu. Također, u zapisima drugih entiteta je moguće vidjeti s kojim ljubimcem je povezan određeni zapis, što također može biti vrlo korisno. Entiteti vezani za korisnike (`AspNetUsers`, `AspNetUserRoles`, …) su dodane pomoću izvršnog okvira Duende Identity Server verzije 5.

4.3 .NET Core kôd

S obzirom da ima previše detalja i kôda unutar sustava da bi se sve prikazalo i objasnilo u ovome radu, fokus će biti samo na bitnijim dijelovima sustava. Prvi kôd koji se izvršava prilikom pokretanja aplikacije odrađuje registriranje i postavljanje svih potrebnih servisa koje će aplikacija koristiti, poput postavljanja konteksta baze podataka za mogućnost rada s bazom podataka iz aplikacije. Navedeni početni kôd se nalazi u datoteci *Program.cs*. Također, potrebno je i postaviti servis za slanje maila korisnicima ukoliko je to potrebno. Nadalje, potrebno je registrirati i sve repozitorije koji će se koristiti unutar kontrolera, što će biti kasnije detaljnije objašnjeno. Kôd potreban za postavljanje ovih servisa je dan u ispisu 6.

```
builder.Services.AddDbContext<PetTrackerDbContext>(options =>
    options.UseSqlServer(
        builder.Configuration.GetConnectionString("PetTrackerDbConnectionString")));

builder.Services.AddDatabaseDeveloperPageExceptionFilter();

builder.Services.AddScoped<ApplicationUserClaimsPrincipalFactory>();

var emailConfig = builder.Configuration.GetSection(EmailConfiguration.EmailConfig).Get<EmailConfiguration>();

builder.Services.AddMailKit(optionsBuilder =>
{
    optionsBuilder.UseMailKit(new MailKitOptions
    {
        Server = emailConfig.SmtpServer,
        Port = emailConfig.Port,
        SenderName = emailConfig.SenderName,
        SenderEmail = emailConfig.From,
        Account = emailConfig.UserName,
        Password = emailConfig.Password,
        Security = true
    });
});

builder.Services.AddScoped<IPetsRepository, PetsRepository>(pr => new PetsRepository(
    pr.GetRequiredService<ILogger<PetsRepository>>(),
    pr.GetRequiredService<IHttpClientFactory>(),
    pr.GetRequiredService<PetTrackerDbContext>(),
    pr.GetRequiredService<IMapper>(),
    builder.Configuration.GetValue<string>("LocationDataRelativeUrl"),
    builder.Configuration.GetValue<string>("AirDataRelativeUrl"),
    builder.Configuration.GetValue<string>("FlameDetectionDataRelativeUrl")
));
```

Ispis 6 - Početni kôd prilikom pokretanja aplikacije - 1. dio

Postaviti se moraju i HTTP klijenti pomoću kojih se šalju HTTP zahtjevi izvan aplikacije. Iz tog razloga, odrađeni su HTTP klijenti Imenovani (eng. *Named clients*) radi jednostavnosti slanja zahtjeva pomoću relativnih URL putanja. Također, kako bi se koristio API Google karata, potrebno je prvo postaviti račun i sve popratne stvari na Google platformi što rezultira privatnim podacima potrebnim za spajanje i korištene spomenutog API-ja. U ovome projektu, ti privatni podaci su spremljeni unutar Azure Key Vaulta te se programski

dohvaćaju prilikom pokretanja aplikacije, kao što je prikazano u ispisu 7 uz postavljanje navedenih HTTP klijenata.

```
builder.Services.AddHttpClient("PetTracker", client =>
{
    client.BaseAddress = new Uri(builder.Configuration["PetTrackerBaseUrl"]);
});

builder.Services.AddHttpClient("GMPlaces", client =>
{
    client.BaseAddress = new Uri(builder.Configuration["Google:MapsPlacesBaseUrl"]);
});

var keyVaultSecretClient = new SecretClient(new Uri(builder.Configuration.GetValue<string>("KeyVault:KeyVaultUrl")), new DefaultAzureCredential());
var googleClientId = keyVaultSecretClient.GetSecret(builder.Configuration.GetValue<string>("Google:KeyVaultClientIdSecretName")).Value.Value;
var googleClientSecret = keyVaultSecretClient.GetSecret(builder.Configuration.GetValue<string>("Google:KeyVaultClientSecretSecretName")).Value.Value;

builder.Services.AddAuthentication()
    .AddIdentityServerJwt()
    .AddGoogle(googleOptions =>
    {
        googleOptions.ClientId = googleClientId;
        googleOptions.ClientSecret = googleClientSecret;
    });
```

Ispis 7 - Početni kôd prilikom pokretanja aplikacije - 2. dio

4.3.1 Kontroleri

Za izradu API-ja i krajnjih točaka (eng. *endpoints*) unutar izvršnog okruženja .NET Core potrebno je izraditi kontrolere u kojima će biti navedene krajnje točke i odrađena njihova implementacija, odnosno ponašanje krajnje točke prilikom zaprimanja HTTP zahtjeva. Kontroleri su praktički mozak cijele aplikacije, odnosno API-ja s obzirom da preusmjeravaju sve HTTP zahtjeve na potrebne krajnje točke. Svi kontroleri, odnosno krajnje točke, u ovome projektu su ograničeni na „interne“ HTTP zahtjeve. Točnije, samo potvrđeni HTTP zahtjevi koji dolaze od stvarnog korisnika aplikacije će biti pušteni na obradu. Ta zaštita/funkcionalnost je omogućena pomoću uloga (eng. *roles*) Identity Servera koje se postavljaju prilikom pokretanja aplikacije kako bi korisnička uloga bila vidljiva, odnosno poslana uz HTTP zahtjev. Kontroleru se dodaje atribut s imenom role koja je potrebna za korištenje istog, kao što je prikazano u ispisu 8.

```
namespace PetTracker.Server.Controllers
{
    [Authorize(Roles = "User")]
    [ApiController]
    [Route("api/[controller]")]
    1 reference
    public class PetsController : ControllerBase
    {
        private readonly IPetsRepository _petsRepository;
        private readonly UserManager<ApplicationUser> _userManager;


        0 references
        public PetsController(IPetsRepository petsRepository, UserManager<ApplicationUser> userManager)
        {
            _petsRepository = petsRepository;
            _userManager = userManager;
        }

        [HttpGet("location")]
        0 references
        public ActionResult<PetLocationDto> GetPetLocation(int petId)
        {
            var (petLocation, errorMsg) = _petsRepository.GetLatestPetLocation(petId);

            if (petLocation is null)
                return BadRequest(errorMsg);

            return Ok(petLocation);
        }
```

Ispis 8 – Kontroler Pets

Kao što je vidljivo u ispisu iznad, za korištenje kontrolera Pets (svih njegovih krajnjih točaka) potrebna je korisnička uloga „User“, odnosno običnog (registriranog) korisnika aplikacije. Također je moguće i postaviti različita pravila pristupa za pojedine krajnje točke. U ispisu se također vidi i jedna krajnja točka koja služi za dohvaćanje lokacije određenog ljubimca. Za komunikaciju s ovom krajnjom točkom potrebno je poslati HTTP GET zahtjev na URL adresu: *~/api/pets/location?petId=[petId]* što je moguće razlučiti iz postavljenih atributa Route. U kontroler se također dodaju potrebni servisi za njegov rad (repozitorij Pets i servis UserManager) koji su, kao što je već objašnjeno, postavljeni prilikom pokretanja aplikacije te se ubacuju u kontroler pomoću uzorka dizajna Dependency Injection.

```
[HttpGet]
0 references
public async Task<ActionResult<ICollection<PetDto>>> GetUserPets()
{
    var user = await GetLoggedInUser();

    var userPets = _petsRepository.GetUserPets(user.Id);

    return Ok(userPets);
}

[HttpPost]
0 references
public async Task<IActionResult> CreateUserPet([FromBody] PetDto pet)
{
    if (pet is null)
        return BadRequest();

    var user = await GetLoggedInUser();

    pet.OwnerId = user.Id;

    try
    {
        var createdPet = _petsRepository.AddUserPet(pet);
        return Created("pets", createdPet);
    }
    catch
    {
        return StatusCode(StatusCodes.Status500InternalServerError, "An error has occurred while adding a new pet.");
    }
}
```

Ispis 9 - Krajnje točke kontrolera Pets - 1. dio

Kao što je vidljivo u ispisu 9, kontroler Pets sadrži i 2 krajnje točke vezane za dohvaćanje svih ljubimaca od prijavljenog korisnika te za dodavanje novog ljubimca. Prva krajnja točka odgovara na HTTP GET zahtjeve, dok je za dodavanje novog ljubimca naravno potrebno poslati HTTP POST zahtjev koji sadrži informacije o novom ljubimca unutar svog tijela. Odrađene su i zaštite od slanja praznih HTTP POST zahtjeva kao i vraćanje ispravnog HTTP statusnog kôda u slučaju nekakve greške prilikom dodavanja novog ljubimca. Nadalje, u ispisu 10 je prikazan implementacijski kôd dvije krajnje točke koje služe za dohvaćanje informacije o kvaliteti zraka sa senzora vezanih za pojedinog ljubimca i krajnja točka za izračun i dohvat statistike o ljubimcu. Detaljniji koraci, odnosno implementacija svih zadataka koje je potrebno izvršiti će biti objašnjeni u kasnijem poglavlju specifičnom za repozitorije. To je implementirano na ovaj način jer je dobra praksa i sustav ima bolje performanse ukoliko kontroler sadrži što manje logike u sebi, već zadatke prosljeđuje drugom dijelu sustava, u ovom slučaju repozitoriju.

```
    [HttpGet("air-info")]
    0 references
    public async Task<ActionResult> GetPetRoomAirInfo(int petId)
    {
        var (petRoomAirInfo, errorMsg) = await _petsRepository.UpdateAndGetPetAirInfo(petId);

        if (petRoomAirInfo is null)
            return BadRequest(errorMsg);

        return Ok(petRoomAirInfo);
    }

    [HttpGet("statistics")]

    public ActionResult<PetStatistics> GetPetStatistics(DateTime dateFrom, DateTime dateTo, int petId)
    {
        var pet = _petsRepository.GetPetById(petId);

        if (pet is null)
            return BadRequest($"Pet with ID {petId} is not found.");

        var petStatistics = _petsRepository.GetPetStatistics(dateFrom, dateTo, pet);

        return Ok(petStatistics);
    }
}
```

Ispis 10 - Krajnje točke kontrolera Pets - 2. dio

Izrađen je i kontroler Admin koji prihvaća samo HTTP zahtjeve poslane od strane korisnika koji ima korisničku ulogu Administratora. Sadrži krajnje točke poput dohvaća svih registriranih korisnika, ažuriranje drugih korisnika (njegovih informacija), te brisanje korisnika kao što je prikazano u ispisu 11.

```
[Authorize(Roles = "Administrator")]
[ApiController]
[Route("api/[controller]")]
1 reference
public class AdminController : ControllerBase
{
    private IUsersRepository _usersRepository;

    0 references
    public AdminController(IUsersRepository usersRepository)
    {
        _usersRepository = usersRepository;
    }

    [HttpGet("users")]
    0 references
    public IActionResult GetAllUsers()
    {
        var allUsers = _usersRepository.GetAllUsers();

        if (allUsers.Count == 0)
            return NotFound("No users found.");

        return Ok(allUsers);
    }

    [HttpPut("users")]
    0 references
    public async Task<IActionResult> UpdateUser([FromBody] UserDto user)
    {
        if (user is null)
            return BadRequest();

        var updateUserResult = await _usersRepository.UpdateUser(user);

        if (updateUserResult.Succeeded)
            return NoContent();
        else
            return StatusCode(StatusCodes.Status500InternalServerError, "An error has occurred while updating the user.");
    }
```

Ispis 11 - Kontroler Admin

Također je izrađen i kontroler za validaciju GPS uređaja i dohvaćanje svih veterinarskih stanica u krugu 3 kilometra od ljubimca kao što je prikazano u ispisu 12.

```
namespace PetTracker.Server.Controllers
{
    [Authorize(Roles = "User")]
    [ApiController]
    [Route("api/[controller]")]
    1 reference
    public class GpsController : ControllerBase
    {
        private IGpsRepository _gpsRepository;

        0 references
        public GpsController(IGpsRepository gpsRepository)
        {
            _gpsRepository = gpsRepository;
        }

        [HttpGet("validate")]
        0 references
        public IActionResult ValidateNewGpsDevice(string serialNumber)
        {
            if (serialNumber.Length != 8)
                return BadRequest("Serial number is not valid!");

            var isValid = !_gpsRepository.IsDeviceInUse(serialNumber);

            return isValid ? Ok() : BadRequest("Serial number is not valid!");
        }

        [HttpGet("clinics")]
        0 references
        public async Task<ActionResult<List<GoogleMapsPlaceInfo>>> GetVetClinics(double longitude, double latitude, int radius)
        {
            if (radius > 50000)
                return BadRequest("Radius cannot exceed 50 kilometers.");

            var closestVetClinics = await _gpsRepository.GetClosestVetClinics(latitude, longitude, radius);

            return Ok(closestVetClinics);
        }
    }
}
```

Ispis 12 - Kontroler Gps

Svaki kontroler vezan za senzore i module/uređaje sadrži krajnju točku za provjeru valjanosti unesenog serijskog broja od strane korisnika aplikacije. Krajnja točka provjerava nalazi li se taj senzor ili modul već u bazi podataka dodanih uređaja i javlja grešku ukoliko se nalazi s obzirom da samo jedan senzor ili GPS modul može biti vezan s jednim ljubimcem. Iz toga razloga ostali kontroleri (za kvalitetu zraka i detekciju plamena) neće biti prikazani posebno u ispisu. Krajnja točka za dohvaćanje veterinarskih stanica poziva metodu iz repozitorija koja koristi i šalje HTTP zahtjev na Google Maps Places API za dohvat svih potrebnih podataka.

4.3.2 Repozitoriji

U ovome projektu, repozitoriji sadrže metode koje krajnje točke pozivaju. Odnosno, u repozitoriju se nalazi kôd za odrađivanje potrebnih zadataka/logike. Iz razloga čitljivosti, snalažljivosti i jednostavnosti, svaki kontroler ima repozitorij iz kojega poziva sve potrebne metode, pa tako kontroler Pets koristi metode iz repozitorija Pets. Repozitorij Pets sadrži sve potrebne metode za uspješan rad cijele aplikacije. Tako na primjer metoda

`UpdateAndGetPetLocation` služi za dohvaćanje lokacijskih podataka s GPS modula, točnije mikrokontrolera koji služi njegove podatke kao što je ranije objašnjeno. Metoda dohvati lokacijske podatke za određenog ljubimca te ih ažurira (ubaci) u bazi podataka kao što je prikazano u ispisu 13. U slučaju da ne uspije pronaći odgovarajuće podatke za GPS modul ili nešto drugo ne bude uspješno, metoda zapisuje greške kako bi programeri mogli lakše doći do razloga greške.

```
public async Task<PetLocationDto?> UpdateAndGetPetLocation(Pet pet)
{
    _logger.LogInformation($"{nameof(UpdateAndGetPetLocation)} started...");

    PetLocationDto? petLocation = null;
    try
    {
        var gpsDataResponse = await _httpClient.GetAsync(TemperatureDataRelativeUrl);
        gpsDataResponse.EnsureSuccessStatusCode();

        var gpsDataJson = await gpsDataResponse.Content.ReadAsStringAsync();
        var allGpsData = JsonConvert.DeserializeObject<List<GpsSensorData>>(gpsDataJson);

        if (allGpsData is null)
            throw new DeserializationFailedException();

        if (allGpsData.Count == 0)
            _logger.LogError("No GPS data found.");

        var petGpsData = allGpsData.Find(d => d.GpsId == pet.GpsDevice?.Id.ToString());
        petLocation = petGpsData?.PetLocation;

        if (petLocation is null)
            _logger.LogWarning("Couldn't match pet's GPS device ID with device ID from data response. Pet ID: {Id}", pet.Id);
        else
            UpdatePetLocation(pet, petLocation.Longitude, petLocation.Latitude);

        _logger.LogInformation($"{nameof(UpdateAndGetPetLocation)} completed successfully.");
    }
    catch (DeserializationFailedException deserializationFailedEx)
    {
        _logger.LogError(deserializationFailedEx, "Failed to deserialize pet location data.");
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Failed to retrieve pet location data.");
    }

    return petLocation;
}
```

Ispis 13 - Repozitorij Pets – dohvaćanje i zapisivanje lokacijskih podataka

Ukoliko je dohvaćanje i upisivanje u bazu podataka bilo uspješno, dodaje se novi zapis u bazi podataka u entitet PetLocations kao što je prikazano na slici 14.

Results | Messages

	Id	Longitude	Latitude	Timestamp	PetId	GpsDeviceId
1	12919	16,44019	43,60813	2022-07-21 19:34:13.7668116	3	842747DA-55C5-4D8F-853C-A71F56987E3B
2	12920	16,54019	43,50813	2022-07-21 19:34:14.8868901	4	8DD197A6-831C-4990-AC06-5AEE7D55EAF5
3	12921	16,44019	43,40813	2022-07-21 19:34:15.2472948	8	7636DE27-A3FD-4AB9-8680-41CB8C2A09DC
4	12922	16,44019	43,50813	2022-07-21 19:34:15.4684655	11	7DFC0EF0-FAA6-4583-AAC1-6804D7E5DE4B
5	12923	16,34019	43,50813	2022-07-21 19:34:15.7397180	19	393B6C5D-3FCD-4847-A34C-699D4507446E
6	12924	16,44019	43,60813	2022-07-21 19:34:41.4878105	3	842747DA-55C5-4D8F-853C-A71F56987E3B

Slika 14 - Zapis lokacije ljubimca u bazi podataka

Zasebna metoda (`GetLatestPetLocation`) odrađuje dohvaćanje najnovije zapisane lokacije iz baze podataka (ispis 14). Također automatski provjerava je li ljubimac izašao iz

postavljene sigurnosne zone (radijusa), i u slučaju da je postavlja svojstvo `LeftSafetyRadius` na *'true'* kako bi upozorenje moglo prikazati korisniku na početnoj stranici.

```
2 references
public (PetLocationDto? location, string? errMsg) GetLatestPetLocation(int petId)
{
    var pet = GetPetById(petId);

    if (pet is null)
    {
        _logger.LogError("No pet found in DB with ID: {PetId}", petId);
        return (null, $"Pet with ID: {petId} not found.");
    }

    var latestPetLoc = pet.PetLocations?.OrderByDescending(l => l.Timestamp).FirstOrDefault();
    if (latestPetLoc is null)
    {
        return (null, "Failed to retrieve latest pet location.");
    }

    var latestPetLocDto = _mapper.Map<PetLocationDto>(latestPetLoc);
    var distanceFromCenter = DistanceCalculatorService.HaversineDistanceInMeters(
          latestPetLoc.Longitude, latestPetLoc.Latitude, pet.CenterLocation!.Longitude, pet.CenterLocation.Latitude);
    if (distanceFromCenter > 5 && distanceFromCenter > pet.Radius)
    {
        latestPetLocDto.LeftSafetyRadius = true;
    }

    return (latestPetLocDto, null);
}
```

Ispis 14 - Repozitorij Pets - čitanje najnovijih lokacijskih podataka iz baze podataka

Dohvaćanje i ažuriranje informacija o kvaliteti zraka i detekciji plamena se odrađuje na vrlo sličan način, stoga bi bilo redundantno pokazivati ispise toga kôda. Odrađene su naravno i metode potrebne za dohvaćanje svih korisničkih ljubimaca te ažuriranje njihovih informacije kao što je prikazano na ispisima 15 i 16.

```
4 references
public ICollection<PetDto> GetUserPets(string userId)
{
    _logger.LogInformation($"{nameof(GetUserPets)} started...");

    var userPets = _context.Pets
        .Include(p => p.GpsDevice)
        .Include(p => p.PetLocations)
        .Include(p => p.CenterLocation)
        .Include(p => p.AirInfoSensor)
        .ThenInclude(s => s!.Data)
        .Include(p => p.FlameDetectionSensor)
        .ThenInclude(s => s!.Data)
        .Where(u => u.OwnerId == userId)
        .ToList();

    if (userPets is null)
    {
        _logger.LogError("An error has happened while getting the user's pets. User ID: {UserId}", userId);
        throw new UserNotFoundException();
    }

    var userPetsDto = _mapper.Map<ICollection<Pet>, ICollection<PetDto>>(userPets);

    _logger.LogInformation($"{nameof(GetUserPets)} completed successfully.");

    return userPetsDto;
}
```

Ispis 15 - Repozitorij Pets – dohvaćanje korisničkih ljubimaca

```
5 references
public Pet? UpdatePet(PetDto pet)
{
    _logger.LogInformation($"{nameof(UpdatePet)} started...");

    var petToUpdate = GetPetById(pet.Id);

    if (petToUpdate is not null)
    {
        petToUpdate.Name = pet.Name ?? petToUpdate.Name;
        petToUpdate.Kind = pet.Kind.ToString();
        petToUpdate.Gender = pet.Gender.ToString();
        petToUpdate.Age = pet.Age;
        petToUpdate.Breed = pet.Breed;
        petToUpdate.Sterilized = pet.Sterilized;
        petToUpdate.Weight = pet.Weight;
        if (petToUpdate.AirInfoSensor?.Id != pet.AirInfoSensor.Id)
        {
            petToUpdate.AirInfoSensor = _mapper.Map<AirInfoSensor>(pet.AirInfoSensor);
        }
        if (petToUpdate.FlameDetectionSensor?.Id != pet.FlameDetectionSensor.Id)
        {
            petToUpdate.FlameDetectionSensor = _mapper.Map<FlameDetectionSensor>(pet.FlameDetectionSensor);
        }
        petToUpdate.CenterLocation = _mapper.Map<PetAreaCenterLocation>(pet.CenterLocation);
        petToUpdate.Radius = pet.Radius;

        _context.SaveChanges();

        _logger.LogInformation($"{nameof(UpdatePet)} completed successfully.");

        return petToUpdate;
    }

    _logger.LogError($"{nameof(UpdatePet)} failed.");
    return null;
}
```

Ispis 16 - Repozitorij Pets - ažuriranje informacije ljubimca

Naravno, u obje metode je potrebno mapiranje u klase koje sadrže sve informacije potrebne za prikazivanje na klijentskom dijelu aplikacije iz entiteta baze podataka. Navedeno mapiranje je odrađeno pomoću NuGet paketa AutoMapper.

Jedna od većih metoda unutar repozitorija Pets je metoda `GetPetStatistics` koja odrađuje izračun statistike ljubimca. Kao što je vidljivo u ispisu 17, prvo se dohvaćaju sve lokacije na kojima je ljubimac bio u danom vremenskom intervalu kojega je postavio korisnik. U slučaju da za ljubimca nema niti jedan zapis lokacije, zapisuje se greška u aplikaciji i izračun statistike nije moguć. Nadalje, izračunava se ukupna udaljenost (u metrima) koju je ljubimac prešao u navedenom razdoblju.

```
4 references
public PetStatistics? GetPetStatistics(DateTime dateFrom, DateTime dateTo, Pet pet)
{
    _logger.LogInformation($"{nameof(GetPetStatistics)} started...");

    var petLocations = _context.PetLocations
        .Where(loc => loc.PetId == pet.Id)
        .Where(loc => loc.Timestamp >= dateFrom)
        .Where(loc => loc.Timestamp <= dateTo)
        .ToArray();

    if (!petLocations.Any())
    {
        _logger.LogWarning("No pet locations for pet {Name}. ID: {Id}", pet.Name, pet.Id);
        return null;
    }

    var distanceSum = 0d;
    for (var i = 0; i < petLocations.Length - 1; i++)
    {
        var loc1 = petLocations[i];
        var loc2 = petLocations[i + 1];

        distanceSum += DistanceCalculatorService.HaversineDistanceInMeters(loc1.Longitude, loc1.Latitude, loc2.Longitude, loc2.Latitude);
    }

    var petStatistics = new PetStatistics
    {
        DistanceTravelled = distanceSum,
        Pet = _mapper.Map<PetDto>(pet)
    };
```

Ispis 17 - Repozitorij Pets - statistika ljubimca - 1. dio

Nastavno, u drugom dijelu metode (ispis 18) se nastavlja izračun prosječne temperature i vlažnosti zraka prostorije. Nakon toga, računa se vrijeme koje je ljubimac proveo odmarajući, odnosno aktivno na način da se uzmu dvije lokacije ljubimca, od kojih jedna prethodi drugoj te se izračuna postoji li ikakva udaljenost između njih. Ukoliko postoji, ljubimac je u tome periodu bio aktivan, dok u protivnome je odmarao. Upit se radi direktno nad bazom podataka radi performansi, jer ovim načinom nije potrebno prebacivati potencijalno ogroman set podataka u memoriju računala.

```
    if (pet.AirInfoSensor is not null)
    {
        var petRoomAirInfoData = _context.AirInfoData
            .Where(d => d.SensorId == pet.AirInfoSensor.Id)
            .Where(d => d.Timestamp >= dateFrom)
            .Where(d => d.Timestamp <= dateTo);

        var avgAirTemp = petRoomAirInfoData.Average(d => d.AirTemperature);
        var avgAirHumidity = petRoomAirInfoData.Average(d => d.AirHumidity);

        petStatistics.AvgRoomAirTemperature = avgAirTemp;
        petStatistics.AvgRoomAirHumidity = avgAirHumidity;
    }

    var petIdParam = new SqlParameter("@PetId", pet.Id);
    var datetimeFromParam = new SqlParameter("@DateTimeFrom", dateFrom);
    var datetimeToParam = new SqlParameter("@DateTimeTo", dateTo);
    var movementTypeParamActive = new SqlParameter("@MovementType", "Active");
    var movementTypeParamRest = new SqlParameter("@MovementType", "Rest");

    var activeMovementInfo = _context.PetMovement
        .FromSqlRaw("EXECUTE GetPetsMovementTime @PetId, @MovementType, @DateTimeFrom, @DateTimeTo", petIdParam, movementTypeParamActive, datetimeFromParam,
        .AsEnumerable()
        .FirstOrDefault();
    var restingMovementInfo = _context.PetMovement
        .FromSqlRaw("EXECUTE GetPetsMovementTime @PetId, @MovementType, @DateTimeFrom, @DateTimeTo", petIdParam, movementTypeParamRest, datetimeFromParam, d
        .AsEnumerable()
        .FirstOrDefault();

    if (activeMovementInfo?.MovementSeconds is not null && restingMovementInfo?.MovementSeconds is not null)
    {
        var activeTime = TimeSpan.FromSeconds(activeMovementInfo.MovementSeconds);
        var restingTime = TimeSpan.FromSeconds(restingMovementInfo.MovementSeconds);

        petStatistics.ActiveTime = activeTime;
        petStatistics.RestingTime = restingTime;
    }

    _logger.LogInformation($"{nameof(GetPetStatistics)} completed successfully.");

    return petStatistics;
}
```

Ispis 18 - Repozitorij Pets - statistika ljubimca - 2. dio

Osim najopsežnijeg repozitorija Pets, u projektu se nalaze i drugi repozitoriji, poput repozitorija Users, koji sadrži metode namijenjene za administraciju korisnika u aplikaciji. Repozitorij Users sadrži osnovne metode za administraciju korisnika, poput dohvaćanja svih registriranih korisnika unutar aplikacije, ažuriranje njihovih podataka te brisanje korisnika. Navedene se metode iz repozitorija Users pozivaju iz kontrolera Users koji, kao što je navedeno ranije, prihvaća samo zahtjeve od korisnika koji imaju korisnički ulogu administratora. Kôd za dohvaćanje svih korisnika i ažuriranje korisnika je prikazan u ispisu 19.

```
public ICollection<UserDto> GetAllUsers()
{
    var allUsers = _context.Users.
        Include(u => u.Pets)
        .ThenInclude(p => p.GpsDevice)
        .ToList();

    if (!allUsers.Any())
    {
        _logger.LogError("No users found in the database.");
        throw new UserNotFoundException();
    }

    var allUsersDto = _mapper.Map<ICollection<ApplicationUser>, ICollection<UserDto>>(allUsers);

    return allUsersDto;
}

2 references
public async Task<IdentityResult> UpdateUser(UserDto user)
{
    var userToUpdate = await _userManager.FindByIdAsync(user.Id);

    if (userToUpdate is null)
    {
        _logger.LogError("No user found in DB with ID: {Id}", user.Id);
        return IdentityResult.Failed();
    }

    userToUpdate.FirstName = user.FirstName!;
    userToUpdate.LastName = user.LastName;
    userToUpdate.PhoneNumber = user.PhoneNumber;
    userToUpdate.Email = user.Email;

    var result = await _userManager.UpdateAsync(userToUpdate);

    if (!result.Succeeded)
    {
        _logger.LogError("Failed to update user: {UserName}", userToUpdate.UserName);
        foreach (var error in result.Errors)
        {
            _logger.LogError("Error code: {Code}. Description: {Description}", error.Code, error.Description);
        }
    }

    return result;
```

Ispis 19 - Repozitorij Users - dohvaćanje i ažuriranje korisnika

Za ažuriranje korisničkih podataka, iskorišten je servis `UserManager` kojega nudi izvršni okvir Duende Identity Server s obzirom da se on koristi u aplikaciji. Prvo se dohvaćaju već postojeći korisnički podaci te se iz objekta iz HTTP zahtjeva koji je proslijeđen iz kontrolera, odnosno podaci zapisuju umjesto već postojećih/dohvaćenih.

Ostali repozitoriji (Gps, AirSensors, FlameDetectionSensors) su vrlo slični. Svi sadrže metodu `IsDeviceInUse` koja provjerava ispravnost senzora koji korisnik želi povezati. Jedino repozitorij Gps ima dodatnu metodu `GetClosestVetClinics` koja služi za dohvaćanje svih veterinaskih stanica u krugu 3 kilometra od ljubimca kao što je prikazano u ispisu 20.

```
public async Task<List<GoogleMapsPlaceInfo>?> GetClosestVetClinics(double latitude, double longitude, int radius)
{
    _logger.LogInformation($"{nameof(GetClosestVetClinics)} started...");

    var query = HttpUtility.ParseQueryString(string.Empty);
    query.Add("location", latitude.ToString(CultureInfo.InvariantCulture) + "," + longitude.ToString(CultureInfo.InvariantCulture));
    query.Add("radius", radius.ToString());
    query.Add("type", "veterinary_care");
    query.Add("key", GoogleMapsAPIKey);

    var uriBuilder = new UriBuilder(_httpClient.BaseAddress + "nearbysearch/json")
    {
        Query = query.ToString()
    };

    try
    {
        var vetClinicsResponse = await _httpClient.GetAsync(uriBuilder.Uri);
        vetClinicsResponse.EnsureSuccessStatusCode();

        var vetClinicsJson = await vetClinicsResponse.Content.ReadAsStringAsync();
        var closestVetClinics = JsonConvert.DeserializeObject<GoogleMapsPlaces>(vetClinicsJson);

        if (closestVetClinics is null || closestVetClinics.VetClinics is null)
            throw new DeserializationFailedException("Deserialization failed for closest veterinary clinics.");

        _logger.LogInformation($"{nameof(GetClosestVetClinics)} completed successfully.");
        return closestVetClinics.VetClinics;
    }
    catch (HttpRequestException httpEx)
    {
        _logger.LogError(httpEx, "Failed to retrieve closest vetinary clinics Google Maps Places API.");
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, $"{nameof(GetClosestVetClinics)} failed.");
    }

    return null;
}
```

Ispis 20 - Repozitorij Gps - dohvaćanje najbližih veterinarskih stanica

Za dohvaćanje najbližih stanica se šalje zahtjev HTTP GET na Google Maps Places API. Vraćeni rezultati se obrađuju, te ovisno o vraćenom rezultatu metoda vraća odgovor. U slučaju greške prilikom izvođenja kôda, metoda zapisuje sve greške radi lakšeg detektiranja i ispravljanja.

4.3.3 Periodička provjera lokacije i detekcije plamena

Servis koji cijelo vrijeme od pokretanja aplikacije radi u pozadini je servis za dohvaćanje i provjeru svih lokacija ljubimaca, odnosno njihove lokacije i sigurne zone te provjera detekcije plamena. Pozadinski servis je implementiran pomoću ASP.NET Core-ovog servisa `BackgroundService` kao što je prikazano u ispisu 21.

```
public class PeriodicLocationCheckerHostedService : BackgroundService
{
    private readonly TimeSpan _timerPeriod = TimeSpan.FromMinutes(2);
    private readonly ILogger<PeriodicLocationCheckerHostedService> _logger;
    private readonly IServiceScopeFactory _serviceScopeFactory;

    0 references
    public PeriodicLocationCheckerHostedService(ILogger<PeriodicLocationCheckerHostedService> logger, IServiceScopeFactory serviceScopeFactory)
    {
        _logger = logger;
        _serviceScopeFactory = serviceScopeFactory;
    }

    0 references
    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        using PeriodicTimer timer = new(_timerPeriod);

        while (
            !stoppingToken.IsCancellationRequested &&
            await timer.WaitForNextTickAsync(stoppingToken))
        {
            try
            {
                await using AsyncServiceScope asyncScope = _serviceScopeFactory.CreateAsyncScope();
                var sampleService = asyncScope.ServiceProvider.GetRequiredService<GetAndCheckPetsLocationService>();

                await sampleService.CheckSensorsAndSendAlert();

                _logger.LogInformation($"Executed {nameof(PeriodicLocationCheckerHostedService)}.");
            }
            catch (Exception ex)
            {
                _logger.LogInformation("Failed to execute {PeriodicLocationCheckerHostedService} with exception message {Message}.",
                    nameof(PeriodicLocationCheckerHostedService), ex.Message);
            }
        }
    }
}
```

Ispis 21 - Periodički servis – registriranje servisa

Postavljeno vrijeme periodičke provjere je 2 minute, što znači da se ovaj servis pokreće svake 2 minute i vrti dok se ne odrade svi zadaci ili se pošalje token za otkazivanje. Unutar periodičkog servisa se poziva servis `GetAndCheckPetsLocationService` koji sadrži logiku za dohvat i provjeru svih podataka. Metoda koja odrađuje dohvat i provjeru je prikazana u ispisu 22.

```
3 references
public async Task CheckSensorsAndSendAlert()
{
    _logger.LogInformation($"{nameof(CheckSensorsAndSendAlert)} has started...");

    var allPets = _petsRepository.GetAllPets();

    foreach (var pet in allPets)
    {
        var latestPetLocation = await _petsRepository.UpdateAndGetPetLocation(pet);
        var (flameDetectionInfo, _) = await _petsRepository.UpdateAndGetFlameDetectionInfo(pet.Id);

        if (latestPetLocation is null)
        {
            _logger.LogError("Latest pet location is null. Pet ID: {Id}", pet.Id);
            continue;
        }

        if (pet.CenterLocation is null)
        {
            _logger.LogError("Pet center location is null. Pet ID: {Id}", pet.Id);
            continue;
        }

        if (latestPetLocation.LeftSafetyRadius)
        {
            var message = $"<h1> Ljubimac {pet.Name} je napustio radius sigurnosne zone (radius {pet.Radius} metara) koji ste postavili. <h1/> <br>" +
                "<h2 style='color:red;'> Molimo da provjerite sigurnost svog ljubimca! </h2>";

            await _emailService.SendAsync(pet.Owner!.Email, "Pet Tracker Location Alert", message, true);

            _logger.LogInformation("Security zone email alert sent for pet {Name} to owner {UserName}", pet.Name, pet.Owner.UserName);
        }

        if (flameDetectionInfo?.Data is null)
        {
            _logger.LogError("Latest flame detection reading is null. Pet ID: {Id}", pet.Id);
            continue;
        }

        if (flameDetectionInfo.Data.FlameDetected)
        {
            var message = $"<h1> Plamn je detektiran pored Vašeg ljubimca {pet.Name}! <h1/> <br>" +
                "<h2 style='color:red;'> Molimo Vas provjerite sigurnost svog ljubimca. </h2>";

            await _emailService.SendAsync(pet.Owner!.Email, "Pet Tracker Flame Alert", message, true);
```

Ispis 22 - Periodički servis - dohvat i provjera

Kao što je vidljivo, u slučaju da je ljubimac izišao iz sigurnosne zone ili je detektiran plamen, određeni mail upozorenja se šalje korisniku na email adresu s kojom je registrirao račun unutar aplikacije. Naravno, kako je ovo periodički servis nezavisan za pojedinog korisnika, provjeravaju se svi registrirani ljubimci u aplikaciji. Za pojedinačno ažuriranje informacija šalju se odvojeni zahtjevi s klijentskog dijela aplikacije, kao što će biti detaljnije objašnjeno u zasebnom poglavlju.

5. Klijentski dio aplikacije (frontend)

Klijentski dio aplikacije je ono što korisnik vidi i ono što je njemu kao korisniku aplikacije bitno. Cijeli pozadinski dio nije bitan ukoliko nije pravilno povezan s klijentskim dijelom aplikacije. U ovome će poglavlju biti opisan upravo klijentski dio i njegova povezanost s pozadinskim dijelom aplikacije. Svaka stranica je podijeljena u dvije datoteke. Jedna koja sadrži HTML kôd, odnosno Razor stranica (*`ImeStranice.razor`*) i druga koja sadrži programski kôd vezan za tu stranicu (`ImeStranice.razor.cs`). Tako i naslovnica ima datoteku u kojoj se nalazi programski kôd potreban za njen ispravan rad.

5.1 Naslovnica

Prva stranica koja se korisniku prikazuje je početna stranica, odnosno naslovnica. Ukoliko korisnik nije prijavljen u aplikaciju, prikazuje mu se tekst koji ukratko opisuje što je aplikacija PetTracker, te za daljnje korištenje je potrebna prijava, odnosno registracija. To ponašanje je implementirano pomoću ASP.NET-ove komponente AuthorizeView. Nakon prijave, dohvaćaju se svi korisnički ljubimci i prikazuju se na Google karti kao oznake lokacije. U slučaju greške prilikom dohvaćanja ljubimaca, ili korisnik nema niti jednog dodanog ljubimca, prikazuje mu se određena poruka. Dio HTML kôda naslovnice je prikazan u ispisu 23.

```
        ,
        <div class="has-text-centered mb-5">
            <h1>Moji ljubimci</h1>
        </div>

        <GoogleMap @ref="@petMap" Id="petMap" Options="@petMapOptions" Height="70%" OnAfterInit="CreatePetMarkers" />

        <div>
            <div class="mt-2 has-text-weight-semibold">
                <span>Lociraj ljubimca:</span>
            </div>
            <div class="select is-link mt-1 is-select-mobile mb-4">
                <select @onchange="SetMapCenter">
                    @foreach (var pet in UserPets)
                    {
                        if (pet.Location is not null)
                        {
                            <option value="@pet.Id">@pet.Name</option>
                        }
                    }
                </select>
            </div>
        </div>
    </div>
</div>
<div class="w-25 is-fullwidth-mobile is-inline-block pt6-mobile-0 pet-info-overview-list">
    @foreach (var pet in UserPets)
    {
        <div class="pet-info-overview-block">
            <span><b>@pet.Name</b></span>
            <br />
            <span><b>Zadnja sinkronizacija: </b>@(pet.LastSynchronization.HasValue ? pet.LastSynchronization.Value.ToLocalTime().ToString(SharedConstants.LocalDa
            <br />
            <span><b>Senzori aktivni:</b> @((pet.LastSynchronization.HasValue && pet.LastSynchronization.Value < DateTime.UtcNow.AddMinutes(-2)) ? "Da" : "Ne")<,

            @if (pet.FlameDetectionSensorData?.Data?.FlameDetected is true)
            {
                <br />
                <span class="has-text-danger"><b>Detektiran plamen pored ljubimca @pet.Name!</b></span>
                <br />
            }
            @if (pet.Location?.LeftSafetyRadius is true)
```

Ispis 23 - Naslovnica HTML kôd

```
protected override async Task OnInitializedAsync()
{
    var authState = await AuthenticationStateProvider.GetAuthenticationStateAsync();
    var currentUser = authState.User.Identity;

    if (currentUser?.IsAuthenticated == true)
    {
        MarkersLoaded = false;
        await GetUserPetsAndLocations();
        await GetLatestPetRoomAirInfo();
        RunRetrieveLocationsTimer();
    }
}

1 reference
private async Task GetUserPetsAndLocations()
{
    PetsLoaded = false;

    var userPetsResponse = await HttpClient.GetAsync("api/pets");

    if (!userPetsResponse.IsSuccessStatusCode)
    {
        PetsLoaded = true;
        return;
    }

    var userPetsJson = await userPetsResponse.Content.ReadAsStringAsync();
    var userPets = JsonConvert.DeserializeObject<List<PetDto>>(userPetsJson);

    if (userPets is not null)
    {
        UserPets = userPets;
        await CreateMapForPets();
    }

    PetsLoaded = true;
    StateHasChanged();
}
```

Ispis 24 - Naslovnica - programski kôd

U ispisu 24 je vidljivo da nakon uspješne inicijalizacije stranice, prvo se provjerava je li korisnik prijavljen kako se ne bi bespotrebno slali zahtjevi i trošili resursi ukoliko korisnik niti nema račun unutar aplikacije. Ukoliko je korisnik prijavljen, poziva se metode za dohvaćanje korisnikovih ljubimaca i svih ostalih informacije vezanih za njih u slučaju da je korisnik dodao bar jednog ljubimca. Unutar metode `GetUserPetsAndLocations` se šalje zahtjev GET HTTP na interni API koji je izrađen u pozadinskom dijelu aplikacije. Ovo je primjer povezivanja prednjeg i pozadinskog dijela aplikacije. Nakon inicijalizacije stranice, dosta metoda se lančano pokreće kako bi se sve uspješno postavilo i prikazalo korisniku. To su metode za postavljanje Google karte i dohvaćanja svih dostupnih bitnih informacija o ljubimcima. Metoda za izradu i postavljanje Google karte je prikazana u ispisu 25.

```
1 reference
private async Task CreateMapForPets()
{
    var pet = UserPets!.FirstOrDefault();
    if (pet is null)
        return;

    if (pet.Location is null)
        await GetPetsLatestLocation();

    var defaultPetLocation = UserPets?.FirstOrDefault()?.Location;
    if (defaultPetLocation is null)
        return;

    petMapOptions = new MapOptions()
    {
        Zoom = 22,
        Center = new LatLngLiteral()
        {
            Lat = defaultPetLocation.Latitude,
            Lng = defaultPetLocation.Longitude
        },
        ClickableIcons = false,
        StreetViewControl = false,
        MapTypeId = MapTypeId.Satellite
    };
}
```

Ispis 25 - Naslovnica - izrada Google karte

Naravno, nije dovoljno samo izraditi mapu, već i prikazati ljubimce na istoj. Metoda `CreatePetMarkers` prikazana u ispisu 26 odrađuje upravo to.

```
1 reference
private async Task CreatePetMarkers()
{
    foreach (var pet in UserPets!)
    {
        if (pet.Location is null)
            continue;

        var marker = await Marker.CreateAsync(petMap!.JsRuntime, new MarkerOptions
        {
            Position = new LatLngLiteral(pet.Location.Latitude, pet.Location.Longitude),
            Map = petMap.InteropObject,
            Clickable = true,
            Title = pet.Id.ToString(),
            Label = new MarkerLabel
            {
                Text = pet.Name ?? string.Empty,
                FontWeight = "bold",
                FontSize = "24"
            },
        });

        var infoWindow = await CreateMapInfoWindow(pet);

        await marker.AddListener("click", async () =>
        {
            await infoWindow.Open(petMap.InteropObject);
        });

        PetMarkers ??= new Dictionary<string, Marker>();

        PetMarkers.Add(pet.GpsDevice!.Id.ToString(), marker);
    }

    MarkersLoaded = true;
}
```

Ispis 26 - Naslovnica - izrada oznaka ljubimaca na Google karti

Kao što je navedeno u prošlom poglavlju, osvježavanje podataka na naslovnici je odvojeni proces od periodičkog servisa, tako da je na naslovnici implementiram vremenski brojač koji šalje novi zahtjev HTTP i ažurira podatke korisniku za prikaz (ispis 27).

```
#pragma warning disable S3168 // Periodic timer must be "fire and forget"
1 reference
private async void RunRetrieveLocationsTimer()
#pragma warning restore S3168
{
    while (await _periodicTimer.WaitForNextTickAsync())
    {
        await GetPetsLatestLocation();
        await GetLatestPetRoomAirInfo();
        await GetLatestFlameDetectionInfo();
        UpdatePetMarkers();
        StateHasChanged();
    }
}
```

Ispis 27 - Naslovnica - osvježavanje podataka

Prilikom okidanja vremenskog brojača, pozivaju se metode za dohvaćanje najnovijih lokacija ljubimaca i svih dostupnih informacije koji se mogu povezati s ljubimcem. Zatim se osvježavaju lokacije na Google karti te se stranici javlja kako je se stanje podataka promijenilo i potrebno je stranicu, odnosno njene komponente, osvježiti.

5.2 Moji ljubimci

Iduća izrađena stranica je stranica „Moji ljubimci“ na kojoj korisnik može pronaći listu svih svojih ljubimaca s detaljnijim informacijama o pojedinom ljubimcu. Također, na ovoj stranici korisnik ima mogućnost dodavanja novih ljubimaca te ažuriranja i brisanja već postojećih. Za dodavanje i ažuriranja ljubimaca korisniku se otvara novi prozor sa formom u koju se upisuju, odnosno mijenjanju podaci o ljubimcu. U ispisu 28 je prikazan HTML kôd od polja za odabir spola ljubimca.

```
<div class="field is-horizontal ml-6">
    <div class="field-label">
        <label class="label">Spol</label>
    </div>
    <div class="field-body">
        <div class="field is-narrow">
            <div class="control">
                <InputRadioGroup @bind-Value="Pet.Gender" Name="Gender">
                    <label class="radio-label">
                        <InputRadio Value="@PetGender.Male" Name="Gender" />
                        <span>@PetGender.Male.GetDisplayName()</span>
                    </label>
                    <label class="radio-label">
                        <InputRadio Value="@PetGender.Female" Name="Gender" />
                        <span>@PetGender.Female.GetDisplayName()</span>
                    </label>
                </InputRadioGroup>
            </div>
        </div>
    </div>
</div>
```

Ispis 28 - Moji ljubimci - odabir spola

Za povezivanje ljubimca sa senzorima potrebno je upisati serijski broj senzora. Za sada je odrađena provjera serijskog broja za samo na način da se provjerava sadrži li serijski broj GPS modula 8 znakova i serijski broj ostalih senzora 6 znakova. Naravno, ovu provjeru je moguće urediti, odnosno unaprijediti po potrebi. GPS modul je obavezno unijeti prilikom dodavanja ljubimca, dok ostale senzore je moguće povezivati i naknadno s obzirom da nisu

obavezni. GPS modul je obavezan jer predstavlja cijelu bit sustava. HTML kôd koji prikazuje polje za unos serijskog broja GPS modula unutar web forme je dan u ispisu 29.

```
<div class="field is-horizontal ml-6">
    <div class="field-label is-normal">
        <label class="label">GPS S/N<span class="has-text-danger">*</span></label>
    </div>
    <div class="field-body">
        <div class="field is-narrow has-addons">
            <div class="control has-icons-right">
                <InputText @bind-Value="Pet.GpsDevice.SerialNumber" class="input" placeholder="Serijski broj GPS-a" disabled="@(IsPetEdit || IsGpsSNValid is true)" />
                <ValidationMessage For="@(() => Pet.GpsDevice.SerialNumber)"></ValidationMessage>
                <span class="has-text-danger" hidden="@(IsGpsSNValid is null || IsGpsSNValid is true)">Serijski broj GPS-a nije ispravan.</span>
                <span class="@(IsGpsSNValid is true ? "icon is-right check-icon" : "is-hidden")">
                    <i class="fa-solid fa-check"></i>
                </span>
            </div>
            <div class="control">
                <button type="button" class="button is-info has-text-weight-semibold" @onclick="ValidateGpsSN"
                        disabled="@(IsPetEdit || !context.IsModified() || Pet.GpsDevice?.SerialNumber?.Length != 8 || IsGpsSNValid is true)">
                    Provjeri S/N
                </button>
            </div>
        </div>
    </div>
</div>
```

Ispis 29 - Moji ljubimci - povezivanje GPS modula

U ispisu se može primijetiti da se nakon unosa serijskog broja poziva metode `ValidateGpsSN` koji šalje zahtjev na već objašnjenu krajnju točku za validacijsku provjeru senzora (koristi li se već u aplikaciji). No, da se ne bi slali bespotrebni zahtjevi i tako trošili resurse, botun za provjeru serijskog broja je dostupan za kliknuti tek ukoliko serijski broj zadovoljava postavljene uvjete (u ovome slučaju minimalno 8 znamenki) i ponovno prestaje biti interaktivna kada validacija uspješno prođe s obzirom da nema smisla više provjeravati. Na sličan način radi i provjera i povezivanje ostalih senzora, a metode za provjeru, odnosno koje se pozivaju su prikazane u ispisu 30.

```
1 reference
private async Task ValidateGpsSN()
{
    var snValidationResponse = await HttpClient.GetAsync($"api/pets/gps/validate?serialNumber={Pet.GpsDevice.SerialNumber}");

    IsGpsSNValid = snValidationResponse.IsSuccessStatusCode;
}

1 reference
private async Task ValidateAirSensorSN()
{
    var snValidationResponse = await HttpClient.GetAsync($"api/airsensors/validate?serialNumber={Pet.AirInfoSensor.SerialNumber}");

    IsAirSensorSNValid = snValidationResponse.IsSuccessStatusCode;
}

1 reference
private async Task ValidateFlameDetectionSensorSn()
{
    var snValidationResponse = await HttpClient.GetAsync($"api/flamedetectionsensors/validate?serialNumber={Pet.FlameDetectionSensor.SerialNumber}");

    IsFlameDetectionSensorSNValid = snValidationResponse.IsSuccessStatusCode;
}
```

Ispis 30 - Moji ljubimci - metode za pozivanje validacije

Nakon što je forma popunjena, i klijentska (*frontend*) validacija uspješno prošla, dolazi na red kôd prikazan u ispisu 31.

```
private async Task HandleValidSubmit()
{
    var petJson = JsonConvert.SerializeObject(Pet);
    var petContent = new StringContent(petJson, Encoding.UTF8, "application/json");

    if (IsPetEdit)
    {
        try
        {
            var updatePetResp = await HttpClient.PutAsync("api/pets", petContent);
            updatePetResp.EnsureSuccessStatusCode();

            ToastService.ShowSuccess("Ljubimac uspješno ažuriran.", "Uspjeh");
        }
        catch
        {
            ToastService.ShowError("Neuspješno ažuriranje ljubimca.", "Greška");
            TogglePetModal();
            return;
        }
    }
    else
    {
        try
        {
            var addPetResp = await HttpClient.PostAsync("api/pets", petContent);
            addPetResp.EnsureSuccessStatusCode();

            ToastService.ShowSuccess("Ljubimac uspješno dodan.", "Uspjeh");
        }
        catch
        {
            ToastService.ShowError("Neuspješno dodavanje ljubimca.", "Greška");
            TogglePetModal();
            return;
        }
    }

    TogglePetModal();

    await GetUserPets();

    StateHasChanged();
}
```

Ispis 31 - Moji ljubimci - procesiranje ispravne forme

Prvo se provjerava je li riječ o ažuriranju podataka ljubimca ili dodavanje novog s obzirom da se za ažuriranje šalje zahtjev HTTP PUT, a za dodavanje POST. Provjera se rezultat zahtjeva i zatim se korisniku prikazuje obavijest o uspješnosti, odnosno neuspješnosti. Nakon toga, osvježava se stranica kako bi se prikazao novi ili uređeni ljubimac. Također postoji i metoda za brisanje ljubimaca, no vrlo je jednostavna i bilo bi redundantno objašnjavati ju.

5.3 Veterinarske stanice

Iduća stranica dostupna prijavljenom korisniku je stranica za ispis svih veterinarskih stanica u krugu 3 kilometra. Moguće je naravno navedeni radijus odraditi dinamički, no u ovome projektu je radi jednostavno odrađeno fiksno. Nakon učitavanja ljubimaca, što je odrađena na isti način na svim stranici gdje je potrebno, korisniku se nudi na izbor ljubimac kojeg želi uzeti kao središte. Korisniku su prikazani samo ljubimci kojima je uspješno dohvaćena i zapisana lokacija, kako bi se spriječile moguće greške. HTML kôd navedenoga je dan u ispisu 32

```
<div class="w-50 mx-auto is-fullwidth-mobile pt-5">
    <div class="mt-3 has-text-weight-semibold">
        <span>Pronađi najbliže veterinarske stanice od ljubimca:</span>
    </div>
    <div class="select is-link mb-2 w-50">
        <select @onchange="GetClosestVetClinicsForPet" class="w-100 is-block">
            <option value="" selected disabled hidden>--Odaberite ljubimca--</option>
            @foreach (var pet in UserPets)
            {
                if (pet.Location is not null)
                {
                    <option value="@pet.Id">@pet.Name</option>
                }
            }
        </select>
    </div>
    @if (ClosestVetClinics is null)
    {
        if (VetClinicsLoaded == true)
        {
            <div class="notification is-danger has-text-centered is-size-4 has-text-weight-semibold has-text-black">
                <i class="fa-solid fa-face-frown"></i>
                <span>Neuspjelo učitavanje lokacija veterinaskih stanica.</span>
            </div>
        }
        else if (VetClinicsLoaded == false)
        {
            <h1 class="data-loading">
                <i class="fas fa-spinner fa-spin"></i>
                <span>Učitavanje lokacija veterinaskih stanica...</span>
            </h1>
        }
    }
    else if (ClosestVetClinics.Count == 0)
    {
        <div class="notification is-warning is-light has-text-centered is-size-4 has-text-weight-semibold has-text-black">
            <i class="fa-solid fa-triangle-exclamation"></i>
            <span>Nije pronađena niti jedna veterinarska stanica u krugu 3 km.</span>
        </div>
```

Ispis 32 - Veterinarske stanice - HTML kôd

Naravno, u slučaju greške prilikom dohvaćanje liste veterinarskih stanica ili da nije pronađena niti jedna, korisniku se ispisuje određena poruka. Metoda koja se poziva (`GetClosestVetClinicsForPet`) služi za slanje zahtjeva GET HTTP na pozadinski API i provjeru rezultata, odnosno procesiranje zahtjeva prikazana je u ispisu 33.

```
private async Task GetClosestVetClinicsForPet(ChangeEventArgs e)
{
    if (string.IsNullOrEmpty(e.ToString()))
        return;

    VetClinicsLoaded = false;

    var petId = Convert.ToInt32(e.Value);
    var selectedPet = UserPets!.Find(p => p.Id == petId);

    var closestVetClinicsResponse = await HttpClient.GetAsync($"api/gps/clinics?" +
        $"latitude={selectedPet!.Location!.Latitude}" +
        $"&longitude={selectedPet.Location.Longitude}" +
        $"&radius=3000");

    var closestVetClinicsContent = await closestVetClinicsResponse.Content.ReadAsStringAsync();

    ClosestVetClinics = JsonConvert.DeserializeObject<List<GoogleMapsPlaceInfo>>(closestVetClinicsContent);

    VetClinicsLoaded = true;
}
```

Ispis 33 - Veterinarske stanice - dohvat stanica

U zahtjevu se kao parametri upita šalju koordinate odabranog ljubimca jer će se ta lokacija uzeti za središte od kojega će se u krugu 3 km tražiti veterinarske stanice.

5.4 Statistika

Nadalje, korisnik može koristiti i stranicu za statistiku pojedinog ljubimca kako bi dobio informacije poput ukupno prijeđenih metara u odabranom razdoblju, njegovom aktivnom i statusu odmaranja itd.

Nakon učitavanja korisnikovih ljubimaca, nudi mu se forma za popuniti te korisnik treba odabrati: ljubimca kojemu će se izračunati statistika, datum i vrijeme otkad i dokad želi statistiku. Kôd za prikaz forme je dan u ispisu 34.

```
<EditForm Model="@PetStatisticsRequest" class="box w-75 mx-auto is-fullwidth-mobile" OnValidSubmit="@HandleValidSubmit">
    <DataAnnotationsValidator />

    <div class="field">
        <label class="label is-inline-block">Statistika od:</label>
        <div class="control is-inline-block ml-1">
            <InputDate Type="InputDateType.DateTimeLocal" @bind-Value="PetStatisticsRequest.From" />
            <ValidationMessage For="@(() => PetStatisticsRequest.From)"></ValidationMessage>
        </div>
    </div>

    <div hidden="@(!DateFromLaterThanTo)" class="mb-4">
        <span class="has-text-danger">Datum od ne može biti noviji od datuma do.</span>
    </div>

    <div class="field is-inline-block is-block-mobile">
        <label class="label is-inline-block is-block-mobile">Statistika do:</label>
        <div class="control ml-2 is-inline-block is-block-mobile">
            <InputDate Type="InputDateType.DateTimeLocal" @bind-Value="PetStatisticsRequest.To" />
            <ValidationMessage For="@(() => PetStatisticsRequest.To)"></ValidationMessage>
        </div>
    </div>
    <div class="is-inline-block is-block-mobile has-text-centered-mobile is-fullwidth-mobile">
        <button type="button" class="button is-info has-text-weight-semibold is-small" @onclick="(() => PetStatisticsRequest.To = DateTime.Now.AddMinutes(-2).TrimSeconds())">
            Do najnovijeg vremena
        </button>
    </div>

    <div class="w-75 is-fullwidth-mobile">
        <div class="mt-2 has-text-weight-semibold">
            <label class="label">Statistika za ljubimca:</label>
        </div>
        <div class="select is-link mt-1 is-select-mobile mb-4">
            <InputSelect @bind-Value="PetStatisticsRequest.PetId">
                <option value="" selected disabled hidden>--Odaberite ljubimca--</option>
                @foreach (var pet in UserPets)
                {
                <option value="@pet.Id">@pet.Name</option>
                }
            </InputSelect>
    </div>
</div>
```

Ispis 34 - Statistika - HTML kôd

Radi boljeg korisničkog iskustva, korisniku je također ponuđen botun za odabir najnovijeg vremena za polje „Do“ kako bi morao izabrati samo donju granicu za statistiku. Nakon popunjavanja svega potrebnog, potrebno je kliknuti na botun „Pretraži“ nakon čega se poziva `HandleValidSubmit` metoda (ispis 35) i šalje zahtjev na pozadinski API koji će odraditi sve potrebne kalkulacije, kao što je objašnjeno ranije.

```
private async Task HandleValidSubmit()
{
    DateFromLaterThanTo = false;

    if (DateTime.Compare((DateTime)PetStatisticsRequest.From!, (DateTime)PetStatisticsRequest.To!) > 0)
    {
        DateFromLaterThanTo = true;
        return;
    }

    PetStatisticsLoaded = false;

    var petStatisticsResponse = await HttpClient.GetAsync($"api/pets/statistics?" +
        $"dateFrom={PetStatisticsRequest.From.Value.ToUniversalTime()}" +
        $"&dateTo={PetStatisticsRequest.To.Value.ToUniversalTime()}" +
        $"&petId={PetStatisticsRequest.PetId}");

    if (petStatisticsResponse.IsSuccessStatusCode)
    {
        var petStatisticsResponseContent = await petStatisticsResponse.Content.ReadAsStringAsync();
        PetStatistics = JsonConvert.DeserializeObject<PetStatistics>(petStatisticsResponseContent);
    }

    PetStatisticsLoaded = true;
}
```

Ispis 35 - Statistika - slanje zahtjeva na interni API

Kao parametri upita zahtjeva, šalju se odabrani datumi od strane korisnika i ID od odabranog ljubimca kako bi pozadinski servis mogao znati za kojeg ljubimca treba izračunati statistiku.

5.5 Administracijska ploča

Krajnje, napravljena je i stranica za upotrebnu od strane administratora aplikacije na kojoj ima prikaz svih registriranih korisnika unutar aplikacije te ih po potrebi ima mogućnosti uređivati i brisati. Iz razloga preglednosti i jednostavni korištenja, implementirana je mogućnost filtriranja korisnika te paginacija (10 korisnika po stranici). Dio HTML kôda potreban za prikaz svih korisnika s navedenim mogućnostima je dan u ispisu 36.

```
<table class="table is-hoverable has-background-warning-light">
    <thead>
        <tr class="v-middle-align">
            <th class="v-middle-align">Id</th>
            <th class="v-middle-align">Korisničko ime</th>
            <th class="v-middle-align">Ime</th>
            <th class="v-middle-align">Prezime</th>
            <th class="v-middle-align">Email</th>
            <th class="v-middle-align">Ljubimci (ID)</th>
            <th class="v-middle-align">Akcije</th>
        </tr>
        <tr>
            <td>
                <input class="input" @oninput="(args => FilterUsers(args.Value?.ToString(), nameof(User.Id)))" placeholder="Filter..." />
            </td>
            <td>
                <input class="input" @oninput="(args => FilterUsers(args.Value?.ToString(), nameof(User.UserName)))" placeholder="Filter..." />
            </td>
            <td>
                <input class="input" @oninput="(args => FilterUsers(args.Value?.ToString(), nameof(User.FirstName)))" placeholder="Filter..." /
            </td>
            <td>
                <input class="input" @oninput="(args => FilterUsers(args.Value?.ToString(), nameof(User.LastName)))" placeholder="Filter..." />
            </td>
            <td>
                <input class="input" @oninput="(args => FilterUsers(args.Value?.ToString(), nameof(User.Email)))" placeholder="Filter..." />
            </td>
            <th></th>
            <th></th>
        </tr>
    </thead>
    <tbody>
        @foreach (var user in Users)
        {
            <tr class="v-middle-align">
                <th class="v-middle-align">@user.Id</th>
                <td class="v-middle-align">@user.UserName</td>
                <td class="v-middle-align">@user.FirstName</td>
                <td class="v-middle-align">@user.LastName</td>
                <td class="v-middle-align">@user.Email</td>
                @if (user.Pets is not null)
                {
                    <td class="v-middle-align">
```

Ispis 36 - Administracijska ploča - HTML kôd

Filtriranje je odrađeno na način da je dodano prazno polje u svakom stupcu koji označava neki korisnički podatak poput ID-a, imena, prezimena… Nakon upisivanja nečega u navedeno prazno polje, poziva se metoda koja filtrira korisnike po tome što je korisnik unio u polje. Metoda `FilterUsers` je dana u ispisu 37.

```
protected async Task FilterUsers(string? value, string columnName)
{
    if (!string.IsNullOrEmpty(value))
    {
        await GetAllUsers();

        Users = columnName switch
        {
            "Id" => Users?.Where(u => u.Id.Contains(value, StringComparison.InvariantCultureIgnoreCase)).ToList(),
            "UserName" => Users?.Where(u => u.UserName.Contains(value, StringComparison.InvariantCultureIgnoreCase)).ToList(),
            "FirstName" => Users?.Where(u => u.FirstName?.Contains(value, StringComparison.InvariantCultureIgnoreCase) == true).ToList(),
            "LastName" => Users?.Where(u => u.LastName?.Contains(value, StringComparison.InvariantCultureIgnoreCase) == true).ToList(),
            "Email" => Users?.Where(u => u.Email?.Contains(value, StringComparison.InvariantCultureIgnoreCase) == true).ToList(),
            _ => Users
        };
    }
    else
    {
        await GetAllUsers();
    }
}
```

Ispis 37 - Administracijska ploča - filtriranje korisnika

Paginacija je u ovome projektu odrađena samo na prednjoj strani aplikacije (ispis 38), pa nema veliki značaj za performanse sustava koliko ima za preglednost.

```
3 references
private async Task GetAllUsers()
{
    UsersLoaded = false;

    var allUsersResponse = await HttpClient.GetAsync("api/admin/users");
    if (allUsersResponse.IsSuccessStatusCode)
    {
        var allUsersJson = await allUsersResponse.Content.ReadAsStringAsync();
        var allUsers = JsonConvert.DeserializeObject<List<UserDto>>(allUsersJson);
        if (allUsers is not null)
        {
            TotalPages = (int)Math.Ceiling(decimal.Divide(allUsers.Count, PageSize));
            var skipCount = (CurrentPage - 1) * PageSize;
            Users = allUsers
                .Skip(skipCount)
                .Take(PageSize)
                .ToList();
        }
    }

    UsersLoaded = true;
}
```

Ispis 38 - Administracijska ploča – paginacija

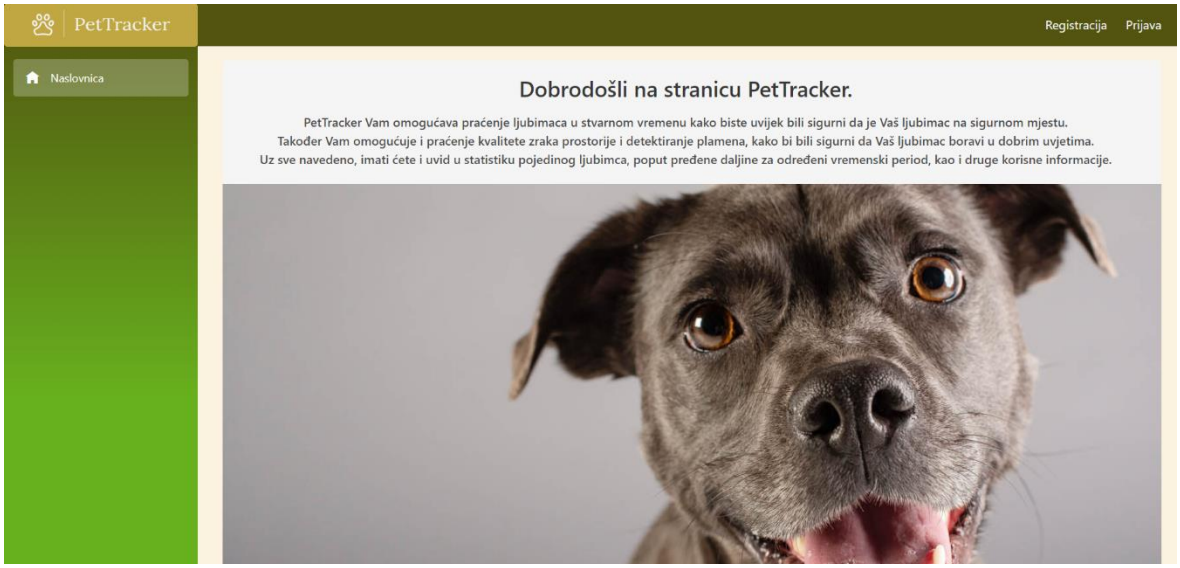
U slučaju velikog broja korisnika i osjetnog utjecaja na performanse sustava, moguće je odraditi paginacija i na pozadinskom dijelu aplikacije, odnosno da krajnja točka vraća određeni broj traženih rezultata, umjesto svih. Trenutna paginacija prikazuje maksimalno 10 korisnika na stranici, te u slučaju više korisnika, izrađuju se dodatne potrebne stranice. Radi boljeg korisničkog iskustva, u lijevom dnu je prikazan broj ukupnih korisnika i koliko ih se trenutno prikazuje od ukupnog broja.

6. Prikaz rada sustava

U ovome poglavlju će biti opisan tok aplikacije i njene mogućnosti koju su dane na raspolaganje korisniku, odnosno administratoru sustava.

Početna stranica koja opisuje sustav ukratko je prva stranica na koja će se prikazati korisniku nakon otvaranja stranice. Ovisno o tome je li korisnik prijavljen ili ne, prikazat će mu se određena stranica. Neprijavljenom korisniku, odnosno gostu stranice se prikazuje početna stranica koja je prikazana na slici 15.

a

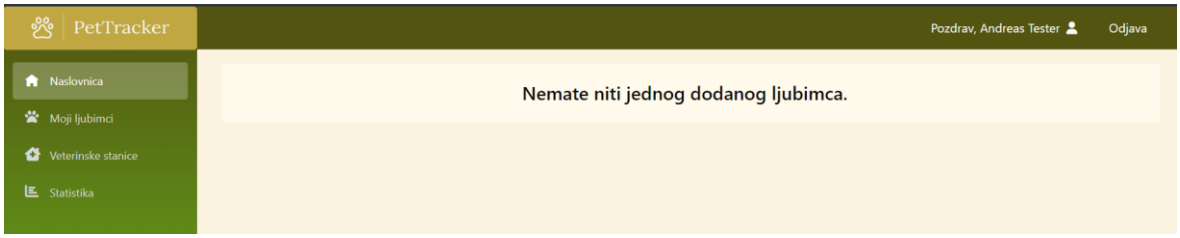


Slika 15 - Početna stranica - neprijavljeni korisnik

Osim ove naslovnice, neprijavljeni korisnik nema nikakvu mogućnost korištenja aplikacije osim prijave ili registracije. Registracija korisnika je prikazana na slici 16.

Slika 16 - Registracija korisnika

Radi jednostavnosti, također je omogućena i registracija pomoću Google računa klikom na botun „Google prijava“. Nakon uspješne registracije ili prijave, korisnika se preusmjerava nazad na Naslovnicu, gdje mu se prikazuje Google karta s lokacijama ljubimaca ili u slučaju da nema niti jednog dodanog ljubimca, obavijest o tome (slika 17).

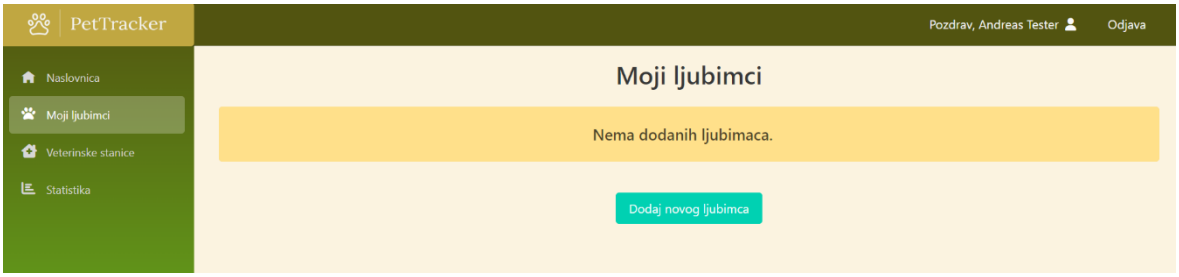


Slika 17 - Naslovnica - prijavljeni korisnik

U protivnom slučaju, korisniku se prikazuje Google karta s pozicijama ljubimaca, na koju je moguće kliknuti za najbitnije informacije vezane za ljubimca u obliku malog prozorčića iznad lokacije. Također, desno od Google karte prikazana je lista najbitnijih informacija o svim ljubimcima, poput aktivnosti senzora i eventualnih opasnosti poput izlaska iz sigurnosne zone i požara. Početna stranica s već dodanim ljubimcima je prikazan na slici 18.

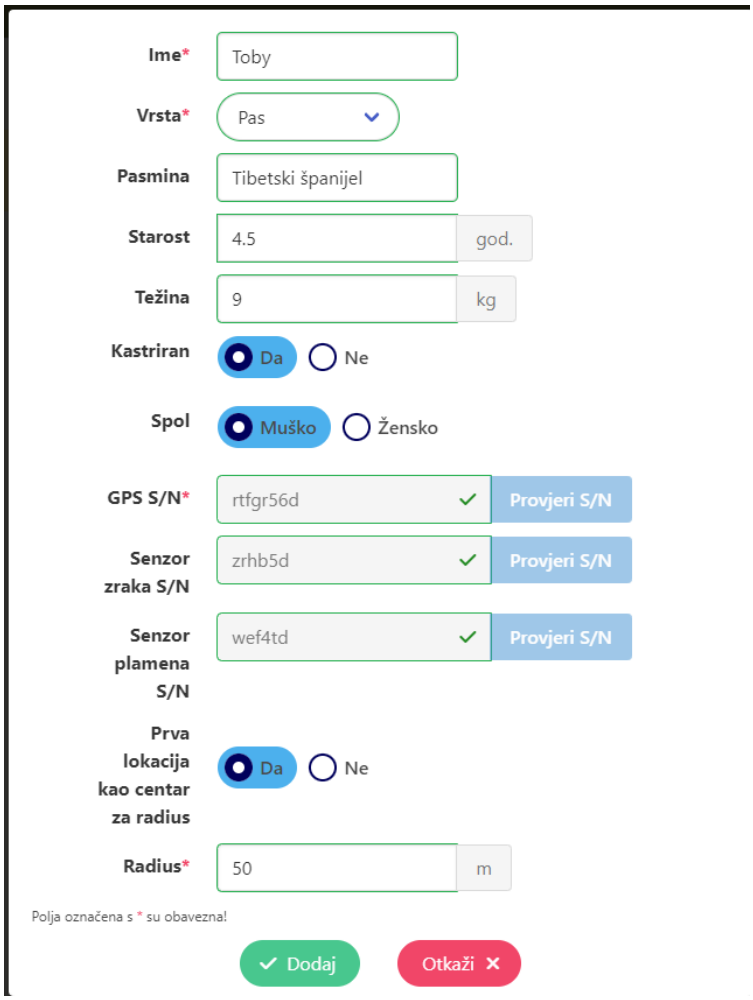
Slika 18 - Naslovnica - prijavljeni korisnik s dodanim ljubimcima

Dodavanje i uređivanje ljubimaca se odrađuje na stranici Moji ljubimci koja je prikazana na slici 19. S obzirom da još niti jedan ljubimac nije dodan, javlja se obavijest i mogućnost dodavanja ljubimca.



Slika 19 - Moji ljubimci - prikaz bez dodanih ljubimaca

Nakon klika na botun „Dodaj novog ljubimca“, prikazuje se forma koju je potrebno popuniti kako bi se dodao novi ljubimac. Neka polja su obavezna za popuniti, dok neka nisu. Obavezna polja su označena crvenom zvjezdicom kao što je i prikazano na slici 20.

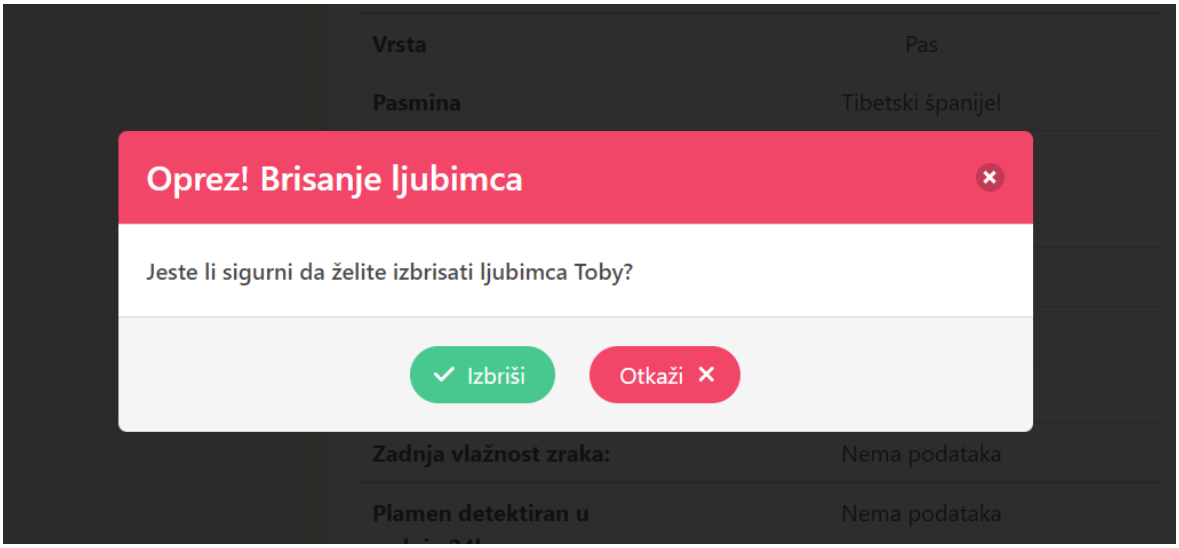


Slika 20 - Dodavanje novog ljubimca

Korisniku se zatim prikazuje obavijest o uspješnosti i ukoliko je dodavanje ljubimca uspješno, prikazuje mu se novo dodani ljubimac na stranici Moji ljubimci kao što je prikazano na slici 21.

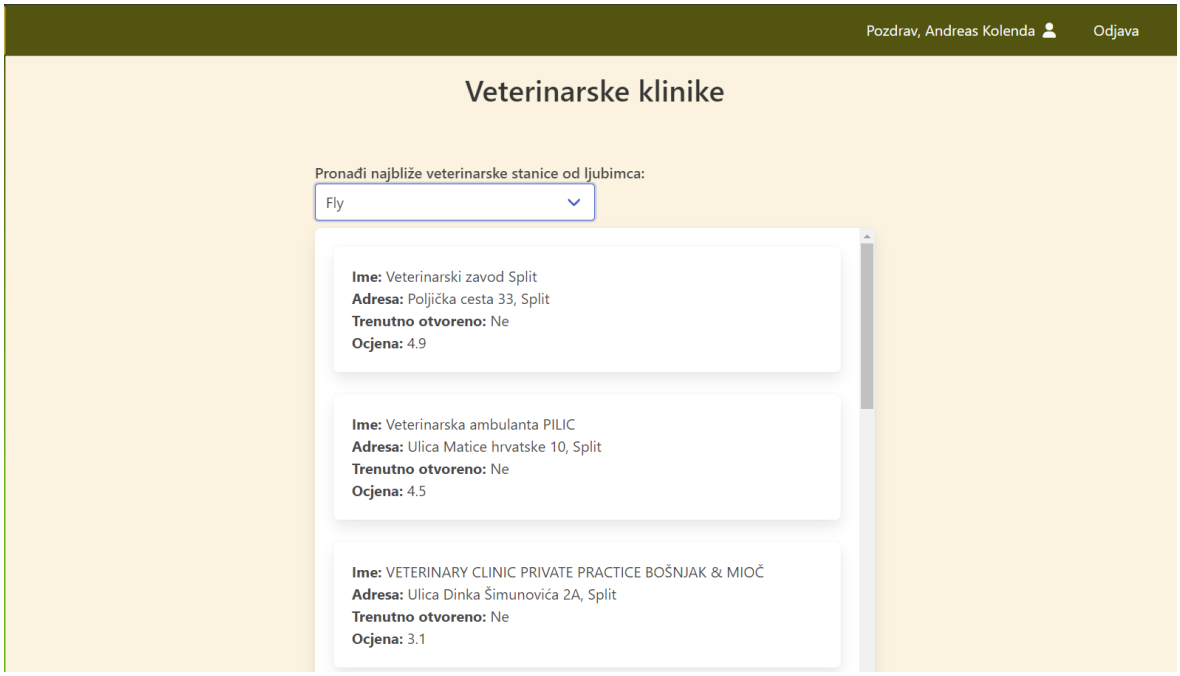
Slika 21 - Uspješno dodan ljubimac

Ažuriranje, odnosno mijenjanje podataka o ljubimcu radi na principu iste forme, stoga to neće biti prikazano. U slučaju brisanja ljubimca, korisniku se prvo otvara potvrdni prozor s kojim korisnik potvrđuje da želi izbrisati ljubimca (slika 22).



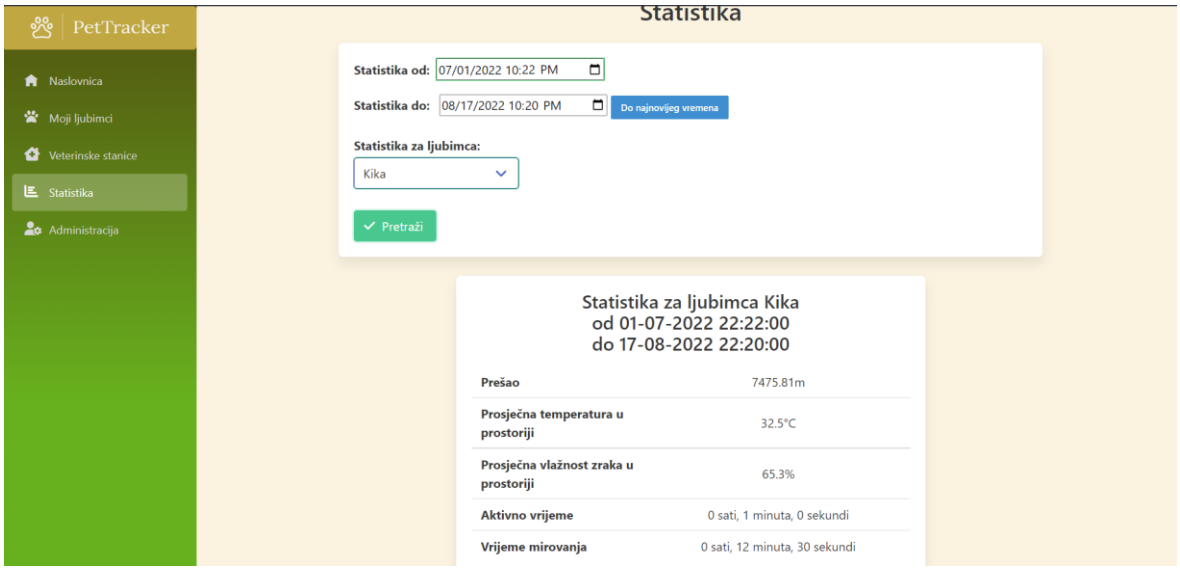
Slika 22 - Potvrda brisanja ljubimca

Zatim korisnik može iskoristiti mogućnosti stranice Veterinarske stanice kako bi dobio listu svih veterinarskih stanica u krugu 3 km od nekog ljubimca. Potrebno je samo odabrati željenog ljubimca (slika 23).



Slika 23 - Prikaz veterinarskih stanica

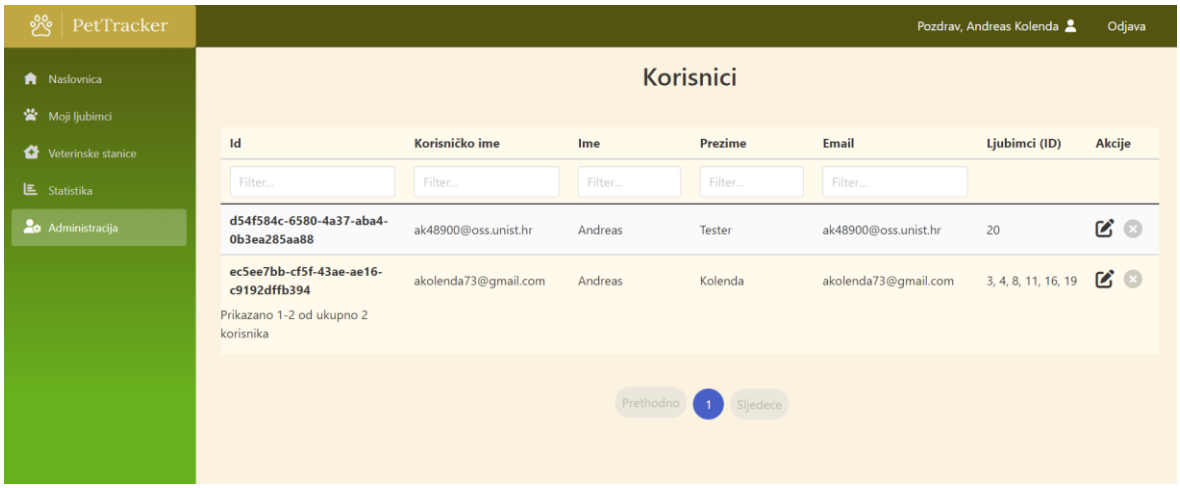
Iduća vrlo korisna stranica za korisnika je Statistika na kojoj korisnik ima mogućnost dobivanja izračunate statistike za pojedinog ljubimca u zadanom vremenskom razdoblju. Primjer izračunate statistike za ljubimca Kika je dan na slici 24.



Slika 24 - Statistika ljubimca

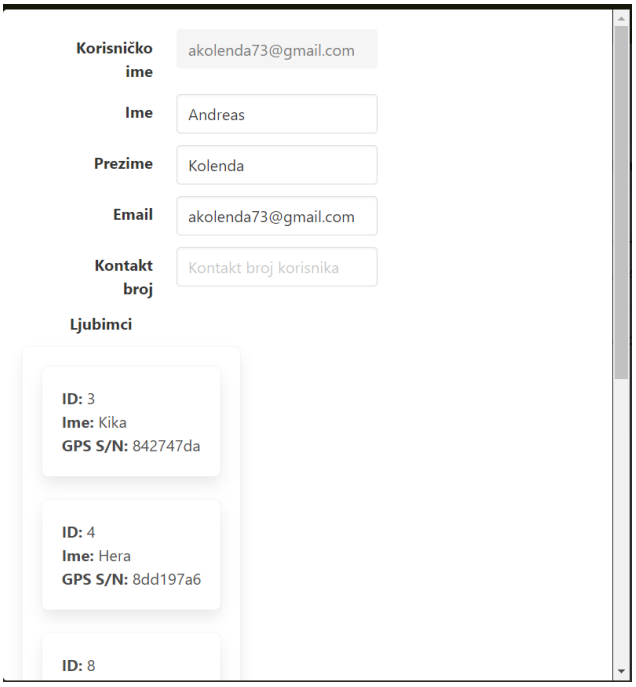
Na dobivenim podacima je moguće vidjeti ukupno metara prijeđenih kretanjem ljubimca, prosječnu temperaturu prostorije u kojoj boravi kao i prosječnu vlažnost zraka. Također, jedna od zanimljivih stvari korisniku je ukupno aktivno i vrijeme mirovanja u zadanom periodu gdje korisnik može vidjeti koliko mu je vremenski aktivan ljubimac.

Administratori sustava na raspolaganju imaju i stranicu Administracija pomoću koju imaju pregled svih registriranih korisnika sustava i mogućnost uređivanja istih (slika 25).



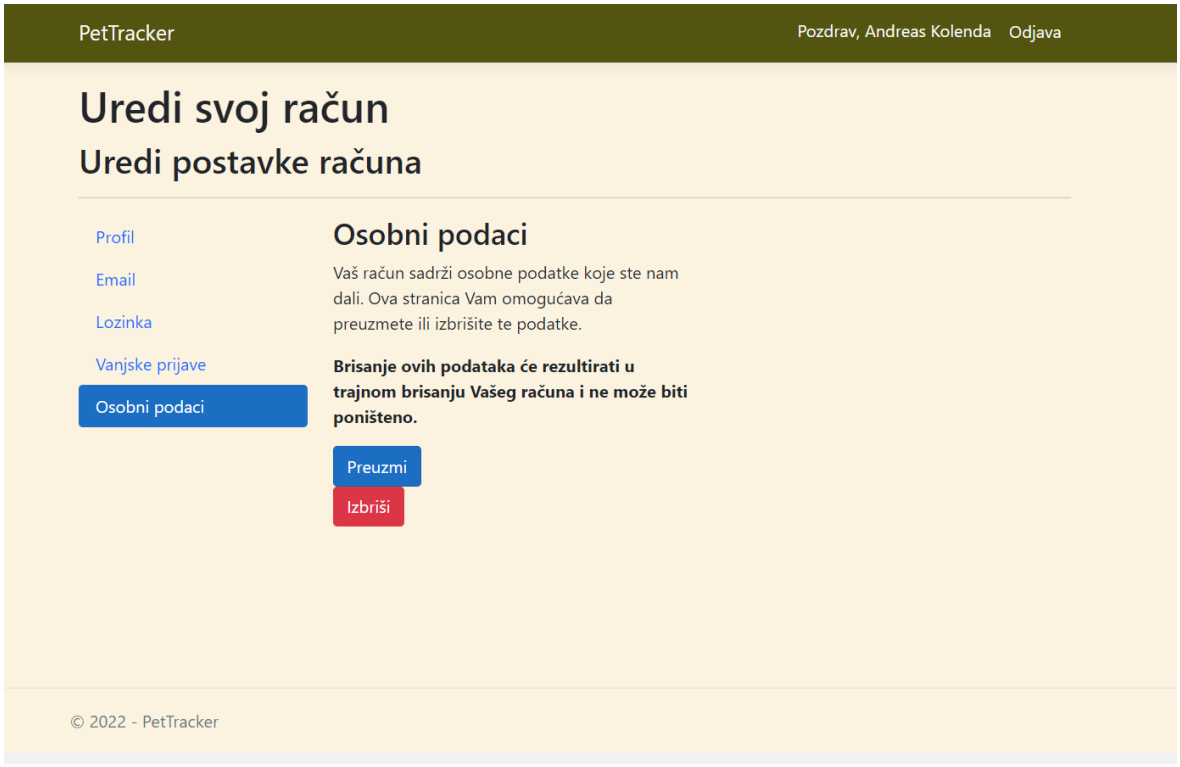
Slika 25 - Administracija korisnika

Uređivanje korisnika, odnosno forma uređivanja korisnika je prikazana na slici 26.



Slika 26 - Uređivanje korisničkih podataka

Korisnik naravno može uređivati svoje podatke te ih također i izbrisati kao i cijeli korisnički račun. Osim toga, omogućena mu je promjena email adrese kao i lozinke, kao i povezivanje već postojećeg računa s Google računom, kao što je prikazano na slici 27.



Slika 27 - Korisnički podaci

7. Zaključak

U ovome radu je izrađen sustav, odnosno aplikacija za praćenje ljubimaca. Osim lokacije ljubimaca, moguće je pratiti i kvalitetu njihovog života što uključuje praćenje temperature i vlažnosti zraka prostorije u kojoj je senzor postavljen. Korisnik sustava time ima pregled svih bitnih stvari na jednom mjestu. Također korisniku dolazi email obavijest u slučaju da sustav registrira da je ljubimac u potencijalnoj opasnosti, ili zbog napuštanja sigurne zone ili detekcije otvorenog plamena. Iz tog pogleda, korisnik ima sve što mu može biti potrebno kako bi mogao mirno ostaviti svog ljubimca samoga i bez nadzora, uz mogućnost provjere njegove lokacije bilo gdje se nalazi.

Iako aplikacija ima sve potrebne stvari kako bi uspješno radila, aplikacija je skalabilna i vrlo ju je lagano unaprijediti u različitim pogledima. Na primjer, moguće je unaprijediti mikrokontroler s GSM modulom kako serviranje podataka ne bi ovisilo o povezanosti na bežičnu mrežu, već bi podaci uvijek bili dostupni na javno dostupnoj IP adresi preko SIM kartice. Također, mogao bi se unaprijediti cijeli prototip hardvera kako bi stalo sve potrebno u kutijicu manjih dimenzija koja bi bila manje primjetna od trenutnog prototipa. Softversko unaprjeđenje je moguće promjenom pravila validacije uređaja/senzora i naravno dodavanja potpuno novih značajki u aplikaciji. Aplikacija također nudi korisniku mogućnost pretraživanja svih najbližih veterinarskih stanica od svakog ljubimca kako bi mogao na istome mjestu i pronaći potrebno liječenje za ljubimca, ukoliko mu bude potrebno. Osim toga, aplikacija sadrži i statistiku pomoću koje korisnik može pratiti aktivnost svog ljubimca, odnosno njegovu količini kretanja i odmaranja, uz informaciju o prosječnoj kvaliteti zraka kako bi mogao ustanoviti boravi li ljubimac u prihvatljivim uvjetima ili ga je možda pametnije preseliti u drugu prostoriju. Aplikacija je vrlo jednostavno i intuitivna za korištenje, stoga korisnik ne bi trebao imati problema razumjeti aplikaciju i njeno korištenje bez pomoći ili posebnog tehničkog predznanja.

Literatura

[1] Milanović, M: „A Brief Walk-Through of the .NET Ecosystem“, https://dzone.com/articles/a-brief-walk-through-net-ecosystem (posjećeno 11.07.2022.)

[2] Chand, M: „What Is New In .NET 6.0“, https://www.c-sharpcorner.com/article/what-is-new-in-net-6-0 (posjećeno 13.07.2022.)

[3] Blazor University: „What is WebAssembly“, https://blazor-university.com/overview/what-is-webassembly (posjećeno 15.07.2022.)

[4] Microsoft: „ASP.NET Core Blazor hosting models“, https://docs.microsoft.com/en-us/aspnet/core/blazor/hosting-models (posjećeno 20.07.2022.)

[5] Microsoft: „.NET Standard“, https://docs.microsoft.com/en-us/dotnet/standard/net-standard (posjećeno 22.07.2022.)

[6] ESP32: „the Internet of THINGS with ESP32“, http://esp32.net (posjećeno 01.09.2022.)