Praćenje lokacije objekta u realnom vremenu preko iOS aplikacije uz korištenje arduino platforme

Horvat, Robert

MEĐIMURSKO VELEUČILIŠTE U ČAKOVCU

STRUČNI STUDIJ RAČUNARSTVA

ROBERT HORVAT

PRAĆENJE LOKACIJE OBJEKTA U REALNOM VREMENU PREKO IOS APLIKACIJE UZ KORIŠTENJE ARDUINO PLATFORME

ZAVRŠNI RAD

ČAKOVEC, rujan 2022.

MEĐIMURSKO VELEUČILIŠTE U ČAKOVCU

STRUČNI STUDIJ RAČUNARSTVA

ROBERT HORVAT

PRAĆENJE LOKACIJE OBJEKTA U REALNOM VREMENU PREKO IOS APLIKACIJE UZ KORIŠTENJE ARDUINO PLATFORME

REALTIME OBJECT TRACKING VIA IOS APPLICATION USING ARDUINO PLATFORM

ZAVRŠNI RAD

Mentor:
dipl. ing. Jurica Trstenjak

ČAKOVEC, rujan 2022.

MEĐIMURSKO VELEUČILIŠTE U ČAKOVCU
ODBOR ZA ZAVRŠNI RAD

Čakovec, 22. ožujka 2021.

ZAVRŠNI ZADATAK br. 2020-RAČ-R-126

Pristupnik: **Robert Horvat (0313020181)**
Studij: redovni preddiplomski stručni studij Računarstvo
Smjer: Programsko inženjerstvo

Zadatak: **Praćenje lokacije objekta u realnom vremenu preko iOS aplikacije uz korištenje Arduino platforme**

Opis zadatka:

Cilj Završnog rada je da se prati tracking autobusnih linija na 2 načina: preko službenog mobitela koji je spojeni na internet i s njega se čitaju podaci o lokaciji u realnom vremenu da bi se krajnji korisnik obavjestio kada se autobus približi njihovoj lokaciji i kompletni Raspberry Pi sustav s aplikacijom u kiosk mode-u s prijavom vozača i pozadinskim procesima za praćenje lokacije ovisno o odabranoj ruti na ekranu prije početka vožnje. U tu svrhu će se koristiti Raspberry Pi, Arduino microkontroler, GPS modul za čitanje geografske širine i dužine, modul za spajanje na internet (SIM kartice ili preko WiFi mreže) radi zapisivanja podataka u bazu i praćenje kretanja u realnom vremenu. U iOS aplikaciji se prikazuje mapa s aktualnom lokacijom odabranog uredaja(autobusa).

Rok za predaju rada: 20. rujna 2021.

Mentor:

Jurica Trstenjak, v. pred.

Predsjednik povjerenstva za završni ispit:

ZAHVALA

Zahvaljujem svom mentoru, profesoru i uzoru na stručnom vodstvu i motivaciji za konstantni napredak te željom za znanjem.

Zahvaljujem kolegama koji su doprinjeli mojem osobnom razvoju kroz međusobno suparništvo.

Zahvaljujem svojoj obitelji i boljoj polovici na strpljenju i podršci tijekom studiranja.

SAŽETAK

Tema završnog rada je mobilna aplikacija za praćenje objekta u realnom vremenu uz mikrokontroler koji obrađuje gps signale svoje trenutne lokacije. Ideja aplikacije je bazirana na primjeru prijevoznik-putnik. GPS odašiljač stavlja se u vozilo prijevoznika i odašilje njegove koordinate u realnom vremenu uz najmanje moguće kašnjenje u svrhu što točnijeg prikaza podataka. Mobilna aplikacija instalira se na iOS uređaj i korisnik prati točne koordinate vozila kako bi imao uvid u vrijeme dolaska i lakše planirao polazak prema stanici. Ovaj završni rad trebao bi dokazati da se taj problem može riješiti s malim elektroničkim sklopom i jednostavnom mobilnom aplikacijom.

Za hardverski dio rada, korišteni su nodeMCU D1 mini mikrokontroler s integriranim bežičnim prijemnikom i velleman U-BLOX NEO 7m gps prijemnik koji imaju glavnu ulogu u cijelom sklopu odašiljača. Za spajanje komponenata korišteni su popratni materijali poput premosnih žica i USB kabela. GPS prijemnik dohvaća podatke o lokaciji i šalje ih serijskom komunikacijom na D1 mini mikrokontroler koji pretvara podatke u čitljiv format i šalje preko internet veze na Firebase bazu podataka koja automatski ažurira korisničko sučelje mobilne aplikacije. Aplikacija prikazuje sučelje s listom uređaja. Klikom na neki od uređaja može se doći do sučelja koje sadrži interaktivnu mapu sa satelitskim pregledom i pribadačom koja prikazuje trenutnu lokaciju objekta.

iOS aplikacija rađena je u macOS sustavu korištenjem xCode razvojnog okruženja i pisana je u swift programskom jeziku. Za pohranu podataka i manipulaciju istima, koristi se noSQL cloud-hosted baza podataka Firebase koja ažurira podatke u realnom vremenu pretplatom na promjene uzrokovane na samoj bazi. Sve tehnologije korištene za izradu aplikacije su besplatne osim Firebase baze podataka, koja je besplatna do određene količine procesiranih zahtjeva, a nakon toga se plaća ovisno o količini obrađenih zahtjeva. Firebase servisi su svestrani. Oni omogućuju da se iz jedne baze

podataka mogu čitati i pisati podaci s različitih platformi, stoga je Firebase baza podataka bila idealan izbor za izradu ovog završnog rada.

Ključne riječi: *GPS, iOS, Arduino, Firebase, Swift, nodeMCU*

SADRŽAJ

1. UVOD

Tema ovog rada je izrada IoT (*engl. Internet of Things*) rješenja za praćenje objekta u realnom vremenu kojeg korisnik može pratiti preko aplikacije na Apple mobilnim uređajima. Svrha aplikacije je olakšati korisnicima praćenje autobusnih linija. Aplikacija je jednostavna za korištenje i osvježava podatke svaku sekundu što daje vrlo preciznu lokaciju.

IoT rješenje sastavljeno je od NodeMCU D1 mini mikrokontrolera na koji je vezan U-Blox NEO 7m GPS modul koji igra glavnu ulogu u navedenom sklopu. Aplikacija za praćenje koristi Swift kao programski jezik i rađena je po uzoru na arhitekturu MVC (*engl. Model-View-Controller*). Oba dijela rada, zajedno su vezani Firebase *real-time* bazom podataka koja se nalazi u oblaku (*engl. cloud-hosted*) i ažurira podatke u realnom vremenu na svim uređajima koji su povezani na nju.

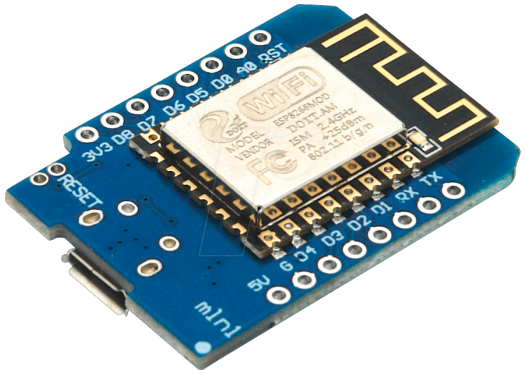
2. KORIŠTENE KOMPONENTE

Za hardverski dio završnog rada koristi se NodeMCU D1 mini mikrokontroler koji predstavlja mozak cijele operacije. Za primanje podataka o lokaciji koristi se U-Blox NEO 7m GPS serijski modul koji pretvara set podataka o lokaciji u korisniku čitljive podatke i serijskom komunikacijom šalje iste podatke prema mikrokontroleru na daljnju obradu.

2.1. NodeMCU D1 mini

NodeMCU D1 Mini s mikro-USB priključkom opremljen je snažnim ESP8266 procesorom i integriranom bežičnom mrežom (*engl. Wireless Local Area Network - WLAN*). Taj mikrokontroler je posebno popularan u krugovima programera i elektroničara zbog svojih malih dimenzija i svestranosti.[1]

NodeMCU D1 mini opremljen je čipom (*engl. chip)* CH340 koji omogućuje lakše i pristupačnije programiranje. Odlikuje ga njegova kompaktnost koja omogućava korištenje za različite svrhe. Koristi se za razne aplikacije i projekte zbog njegovih dimenzija i dizajna. Uz korištenje raznih senzora može mjeriti temperaturu, vlažnost i tlak zraka, obavještavati korisnika o prisutnosti nekog tijela u blizini preko PIR (*engl. Passive Infrared*) senzora, mjeriti udaljenost sa ultrazvučnim senzorima itd.



Slika 1. NodeMCU D1 mini mikrokontroler

Izvor: https://reichelt.com *(10.09.2022)*

Neke od tehničkih specifikacija NodeMCU D1 mini mikrokontrolera prikazane su u tablici. [1]

Radni napon	3.3V DC
Procesor	ESP8266-12F
Taktna frekvencija	80 MHz / 160MHz
Flash memorija	4 MB
Broj digitalnih/analognih pinova	9 / 1
Dimenzije	25 x 35 x 6 mm (V x Š x D)

Tablica 1. Tehničke specifikacije NodeMCU D1 mini mikrokontrolera

2.2. U-Blox NEO 7m

U-Blox NEO 7m je GPS (*engl. Global Positioning System*) modul koji pruža izrazito visoku osjetljivost i minimalno vrijeme kašnjenja u dohvaćanju GPS podataka. Dobiveni parametri se mogu slati preko serijskog porta ili spremati u EEPROM (*engl. Electrically-Erasable Programmable Read-Only Memory)*. Modul sadrži SMA (*engl. SubMiniature version A*) koaksijalni priključak na koji se mogu spojiti mnogobrojne vanjske antene. Radni napon mu nije ograničen na 3.3V DC, već podržava i 5V DC radi lakšeg spajanja na različite vrste mikrokontrolera (uključujući i mikroUSB ulaz). Na njega je dodana i keramička antena da bi se lakše dobio signal s nekih od glavnih i većih satelita iako se za stabilniji i precizniji signal preporuča korištenje vanjske antene. Zadana procjena bauda (*engl. Baud rate*) je postavljena na 9600 bps. Procjena bauda je brzina kojom podaci putuju kroz modem izražena u bps (*engl*. *bits per second*). [2]

Slika 2. U-Blox NEO 7m

Izvor: https://www.velleman.eu (10.09.2022)

Neke od tehničkih specifikacija U-Blox NEO 7m GPS modula prikazane su u tablici.[2]

Radni napon	3.3V DC / 5V DC
Pinovi	VCC (+5V), GND, TX, RX, PPS (vremenski puls)
Baud rate	9600 baud
Dimenzije	40 x 25 x 15 mm (V x Š x D)

Tablica 2. Tehničke specifikacije U-Blox NEO 7m GPS modula

2.3. Popratni materijali

Od popratnih materijala koristile su se komponente tipične za elektronički sklop. Nekoliko premosnih žica (*engl. Jumper wires)* te kabel za prijenos podataka (mikro USB na USB A).

3. FIREBASE

3.1. Firebase

Firebase je platforma koja se koristi za razvoj web i mobilnih aplikacija, a osnovana je 2011. godine. Kasnije, 2014. godine, ju je otkupio Google. Firebase nudi brojne usluge uz jako malo truda unutar samog projekta. Može se integrirati u web stranice, Android iOS ili čak preko cross-platform razvoja. Često se koristi na raznim aplikacijama ili igrama. Firebase je platforma koja nudi kompletno *back-end* rješenje s integriranom bazom podataka, autentifikacijom i čak praćenjem analitike i performansi.[3]

Svi njezini servisi su smješteni u oblaku (*engl. Cloud-hosted*) i skaliraju se s jako malo ili čak bez truda od strane programera. To znači da su svi njezini servisi i komponente održavani od strane *Google*-a. Programer se spaja na njihove usluge preko kompleta za razvoj softvera (*engl. Software Development Kit - SDK)* specifičnog za platformu za koju se dio programa razvija i tako pristupa podacima direktno, bez posrednika.

Slika 3. Firebase logo

Izvor: https://firebase.google.com (10.09.2022)

Neki od servisa koje Firebase platforma nudi su prikazane u tablici.[3]

Autentifikacija	Registracija i prijava korisnika
Realtime database	Baza podataka sinkronizirana u realnom vremenu
Cloud Firestore	Baza podataka sinkronizirana u realnom vremenu
Cloud Storage	Spremanje datoteka
Firebase Hosting	Mjesto na kojem se nalazi web stranica
ML Kit	Strojno učenje (*engl. Machine Learning*)
Analitika	Analiza korisnika i način korištenja aplikacije
Predviđanja	Primjenjivanje strojnog učenja u svrhu predviđanja ponašanja korisnika
Cloud Messaging	Slanje poruka i obavijesti korisnicima
Daljinske konfiguracije	Uključivanje i isključivanje dijelova aplikacije po određenim parametrima bez potrebe za ažuriranjem verzije aplikacije
A / B testiranje	Testiranje aplikacija u svrhu poboljšanja marketinga i same aplikacije
In-App Messaging	Postizanje komunikacije s aktivnim korisnicima preko ciljanih poruka

Tablica 3. Popis servisa platforme Firebase

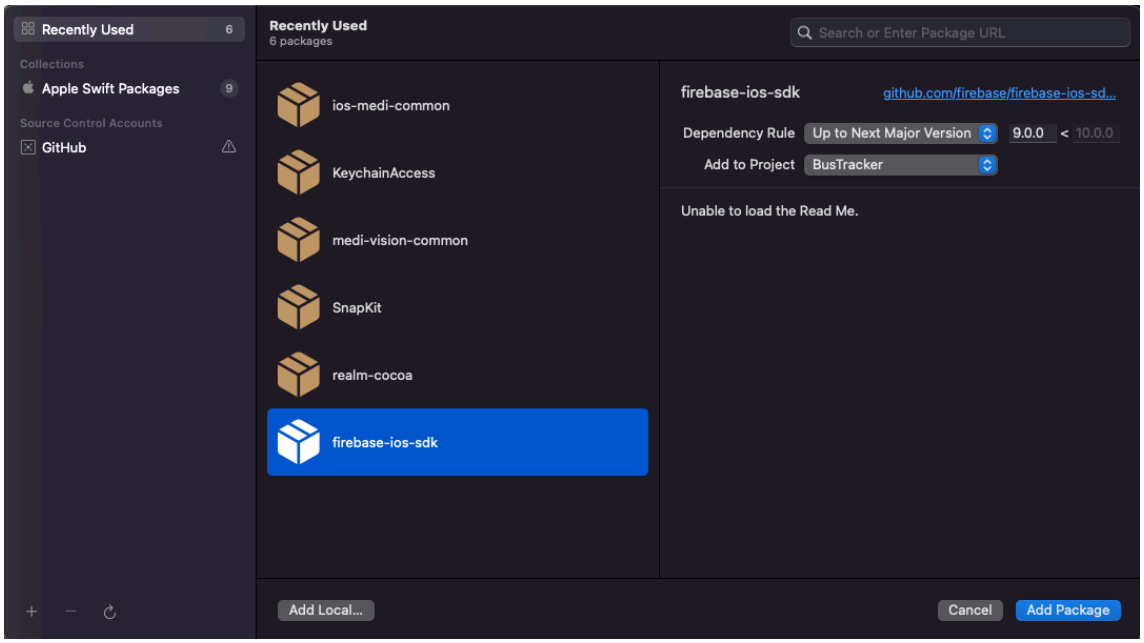
3.2. Firebase real-time baza podataka

Firebase real-time baza podataka je baza podataka smještena u oblaku. Podaci se spremaju u JSON (*engl. JavaScript Object Notation*) formatu i sinkroniziraju se u realnom vremenu za svakog korisnika koji je trenutno vezan na nju neovisno o platformi. Firebase real-time baza podataka nije relacijska baza podataka već je skalirana horizontalno i bazira se na parovima ključ-podatak. Za svaki ključ postoji neki podatak spremljen pod tim indeksom. Takav tip baze podataka naziva se NoSQL (*engl. Not Only Structured Query Language*). Upravo zbog toga što njezina struktura nije ograničena na fiksne veličine tablica i podataka, odlikuje se brzinom, lakšim održavanjem i podržavanjem velikog broja istovremeno spojenih korisnika.[3]

Umjesto tipičnih HTTP zahtjeva (*engl. Hypertext Transfer Protocol requests*), Firebase real-time baza podataka koristi sinkronizaciju podataka - za svaku promjenu podataka, svaki spojeni uređaj dobiva novi set podataka unutar nekoliko milisekundi. Aplikacije povezane na Firebase real-time bazu podataka ostaju u toku čak i kada ostanu bez pristupa internetu. Podaci ostaju na disku te se sinkroniziraju sa serverom u trenutku ponovnog spajanja na mrežu.[3]

3.3. Primjer implementacije (iOS swift)

Da bi se korisnik mogao spojiti na Firebase servise preko iOS aplikacije, mora se koristiti njihova biblioteka, odnosno njihov dio kôda, pisan specifično za platformu koja ga koristi. iOS aplikacije dodaju svoje biblioteke preko sustava zvanog Swift upravitelj paketa (*engl. Swift Package Manager - SPM*). Biblioteka se dodaje preko web adrese na kojoj je ona smještena. U 90 % slučajeva je to GitHub.



Slika 4. Swift upravitelj paketa

Izvor: autor

3.4. Primjer implementacije (Arduino C++)

Za spajanje mikrokontrolera na Firebase servise, koristi se C++ biblioteka nazvana firebase-arduino (https://github.com/FirebaseExtended/firebase-arduino). Konkretna biblioteka oslobađa programera pisanja metoda za manipulaciju podacima i skraćuje vrijeme programiranja. Biblioteka se uključuje preko Arduino IDE razvojnog okruženja koji cijeli program sprema na mikrokontroler da bi odrađivao posao koji mu je zadan.

4. RAZVOJNO OKRUŽENJE

Razvojno okruženje (*engl. Integrated Development Environment - IDE*) je aplikacija koja olakšava programeru posao rezimirajući sve bitne stavke koje su potrebne za obavljanje posla. Razvojno okruženje dizajnirano je za povećanje produktivnosti. U razvojnim okruženjima najčešće se kombinira sustav za izradu, modificiranje, kompajliranje, implementaciju i otklanjanje pogrešaka.

4.1. Arduino IDE

Arduino razvojno okruženje (*engl. Arduino Integrated Development Environment*) razvojno je okruženje otvorenog kôda (*engl. Open Source*) koje se koristi pri razvoju programa za mikrokontrolere. Sastoji se od tekstualnog polja u kojem se piše programski kôd, polje s porukama, tekstualna konzola, alatna traka s gumbima za neke od važnijih akcija i nekoliko izbornika. [4]

Slika 5. Arduino IDE

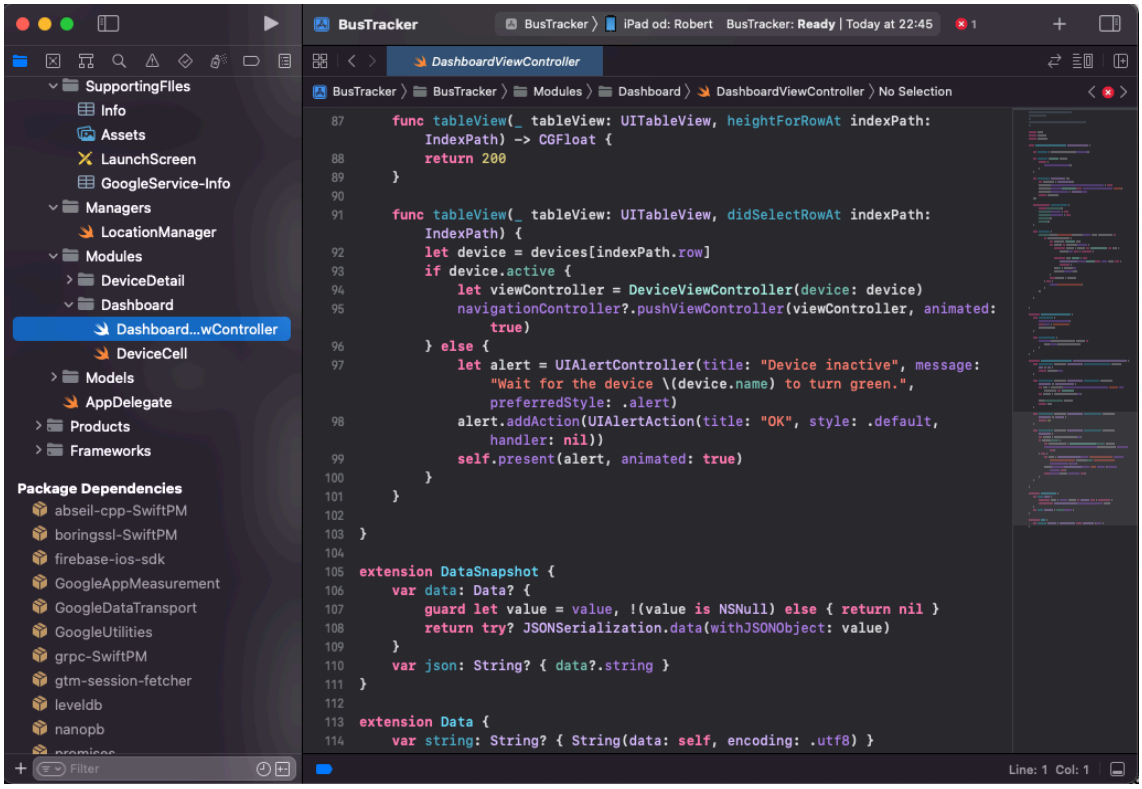
Izvor: autor

Programi pisani korištenjem Arduino IDE razvojnog okruženja nazivaju se skicama (*engl. Sketches*). Skice su pisane u tekstualnom polju za izradu programa i spremaju se s datotečnim nastavkom .ino. Nakon što je program dovršen, klikom na gumb "provjeri", programski kôd se kompajlira i provjerava se njegova ispravnost. Nakon provjere programskog kôda, on se klikom na gumb "prenesi" ponovno kompajlira i šalje na mikrokontroler koji je prethodno bio konfiguriran.[4]

4.2. Xcode

Xcode je Apple-ovo razvojno okruženje za macOS operativni sustav. Xcode se koristi za razvoj aplikacija za macOS, iOS, iPadOS, watchOS i tvOS operativne sisteme. Xcode podržava izvorni kôd za nekoliko programskih jezika poput C, C++, Java, Objective-C, Objective-C++, Python, AppleScript, Ruby, Swift itd.[5]

Xcode je prvi izbor programera koji razvijaju aplikacije za Apple ekosisteme iz razloga što je jedino razvojno okruženje podržano od strane Apple-a. Xcode ima integrirano nekoliko standardnih alata poput alata za ispravljanje pogrešaka (*engl. Debugging*), automatsko nadopunjavanje programskog kôda, umetanja isječaka, upozorenja na greške pa čak i ugrađeni sustav verzioniranja kôda (Git).



Slika 6. Xcode IDE

Izvor: autor

5. MIKROKONTROLER

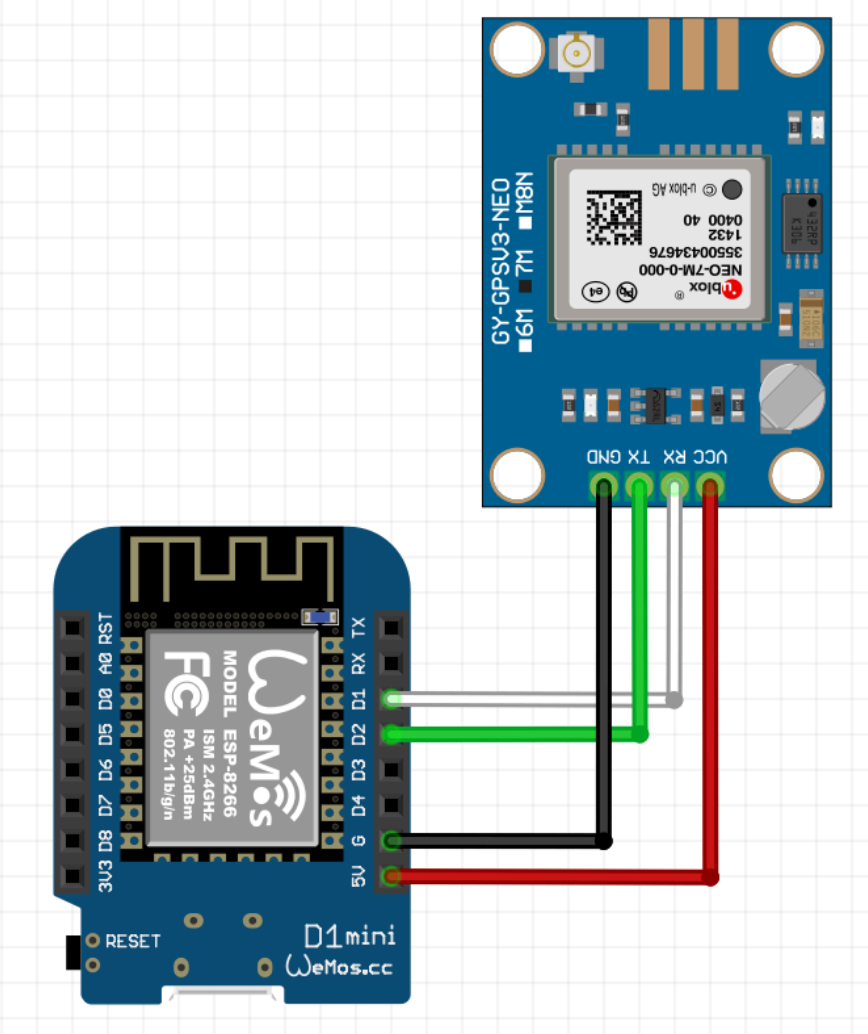
Fizički dio završnog rada izveden je u obliku elektroničkog sklopa koji se sastoji od mikrokontrolera i GPS modula kao glavnih aktera i popratnih materijala. Mikrokontroler je programabilni uređaj koji se sastoji od jezgre procesora, memorije i I/O modula. Najčešća mjesta na kojima se mikrokontroleri koriste su u industriji u sklopu kontrole strojeva, industriji automobila i automatizaciji općenito. Mikrokontroler vrti jedan program koji se zapisuje u njegovu Flash memoriju (memorija koja se može brisati i reprogramirati električnim putem), a u njegovoj ROM (*engl. Read Only Memory*) memoriji pohranjuju se podaci nakon gašenja.

5.1. Fritzing shema

Shema je prikaz strujnog kruga na jasan i standardiziran način. Shema ožičenja (*engl. Wiring diagram*) često je korisna kako bi se ilustriralo i prikazalo spajanje samih komponenti na jednostavniji i čitkiji način. Takve sheme su tipično namijenjene krajnjim korisnicima i/ili instalaterima.

Fritzing je program otvorenog kôda, vrlo popularan u krugovima elektroničara i programera, osmišljen kako bi u što većoj mjeri asistirao stvaranju prototipova. Fritzing vizualizira fotografiju sklopa na uredniji i pristupačniji način da bi se korisnik sklopa lakše snašao. Time se uvelike smanjuje mogućnost pogreške kod spajanja ili mijenjanja elektroničkih komponenti.[6]

Fritzing shema elektroničkog sklopa za praćenje objekta u realnom vremenu uz korištenje arduino platforme prikazana je na slici 7.



Slika 7. Fritzing shema sklopa za praćenje lokacije objekta u realnom vremenu

Izvor: autor

NodeMCU D1 mini	U-Blox NEO 7m
D1 - digitalni pin 1	RX pin
D2 - digitalni pin 2	TX pin
G - pin uzemljenja (-)	GND - pin uzemljenja (-)
5V - pin napona 5V DC	VCC - pin ulaza napona

Tablica 4. Popis spojeva

5.2. Implementacija kôda

Da bi elektronički sklop služio namijenjenoj svrhi, potrebno je implementirati programski kôd koji će odraditi zadani zadatak. Mikrokontroleri u amaterskoj i hobi elektronici najčešće se programiraju u programskom jeziku C. Program korišten kao razvojno okruženje za programiranje sklopa za praćenje objekta u realnom vremenu je Arduino IDE. U nastavku se može vidjeti primjer arduino programskog kôda. Dio programskog kôda u kojem se uključuju korištene biblioteke, definiraju konstante i varijable.

```
// Uključene biblioteke
#include <FirebaseArduino.h> // Biblioteka za interakciju s
firebase servisima
#include <ESP8266WiFi.h> // Biblioteka za korištenje Wi-Fi
mreže
#include <TinyGPS++.h> // Biblioteka za interakciju s GPS
modulom
#include <SoftwareSerial.h> // Biblioteka za čitanje serijskih
podataka s GPS modula

// Definirane konstante
#define FIREBASE_HOST "realtimetracker-e9a8d-default-
rtdb.firebaseio.com" // url na kojem je smještena baza podataka
#define FIREBASE_AUTH
"tkzdsxfoEkAfN7v8cfywgSAL3Y1FcuyLX0r9WyjV" // token za pristup
bazi podataka
#define WIFI_SSID "iPhone od: Robert" // SSID Wi-Fi mreže na
koju se spaja radi pristupa internetu
#define WIFI_PASSWORD "12345678" // zaporka Wi-Fi mreže na koju
se spaja

const int RX = 4; // broj RX digitalnog pina
const int TX = 5; // broj TX digitalnog pina

// Definirane varijable
String deviceName; // ime uređaja
TinyGPSPlus gps; // gps objekt
SoftwareSerial ss(RX, TX); // objekt za serijsku manipulaciju
podacima
```

Primjer setup() funkcije koja se odvrti prilikom pokretanja uređaja:

```
void setup() {
  Serial.begin(115200); // Započni serijsku komunikaciju na
baudu 115200 za ispis logova
  ss.begin(9600); // započni serijsku komunikaciju s GPS
modulom na baudu 9600

  // spajanje na Wi-Fi mrežu
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
    Serial.print("Spajanje....");
    while (WiFi.status() != WL_CONNECTED) {
      Serial.print(".");
      delay(500);
    }

  Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);

  // provjera veze uređaja prema bazi
  Serial.print("Spajanje na bazu u tijeku... ");
  deviceName = Firebase.getString("devices/device1/name");
  if (Firebase.failed()) {
    Serial.println("Neuspješno spajanje na bazu!");
    return;
  }
  else
  {
    Serial.println("Uređaj " + deviceName + " se uspješno
spojio na bazu.");
  }
}
```

Funkcija loop() koja se odrađuje sve dok je uređaj uključen:

```
// loop funkcija koja se vrti u krug sve dok je uređaj uključen
void loop() {
  checkWifi();
  checkGPS(1000);
}
```

Funkcija checkWifi() koja provjerava status uređaja na mreži:

```
// funkcija za provjeru veze na Wi-Fi mrežu
void checkWifi() {
  if (WiFi.status() == WL_CONNECTED) {
    return;
  }
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
    Serial.print("Spajanje");
    while (WiFi.status() != WL_CONNECTED) {
      Serial.print(".");
      delay(500);
    }
}
```

Funkcija checkGPS() koja obrađuje dobiveni signal s GPS modula i šalje u bazu podataka:

```
// funkcija za provjeru podataka s GPS modula s ulaznim
parametrom ms
// koji označava milisekunde, odnosno svakih koliko milisekundi
se provjerava signal
void checkGPS(unsigned long ms) {
  unsigned long start = millis();
  while (millis() - start < ms) {
    while (ss.available()) {
      gps.encode(ss.read());
      if (gps.location.isValid()) {
        updateFirebase();
      }
    }
  }
}
```

Funkcija updateFirebase() koja ažurira podatke u bazi podataka:

```
// funkcija koja azurira podatke dobivene s GPS modula u bazi
void updateFirebase() {
  float latitude = gps.location.lat();
  float longitude = gps.location.lng();
  Firebase.setFloat("devices/device1/latitude", latitude); //
spremanje podatka

  // ispis greške u spremanju podatka ukoliko dođe do iste
  if (Firebase.failed()) {
      Serial.print("Dogodila se greška u spremanju podatka
'latitude': ");
      Serial.println(Firebase.error());
      return;
  }

  Firebase.setFloat("devices/device1/longitude", longitude); //
spremanje podatka

  // ispis greške u spremanju podatka ukoliko dođe do iste
  if (Firebase.failed()) {
      Serial.print("Dogodila se greška u spremanju podatka
'longitude': ");
      Serial.println(Firebase.error());
      return;
  }
}
```

6. iOS APLIKACIJA

iOS (ili ranije iPhone OS) je mobilni operativni sistem koji je razvio Apple za svoj uređaj iPhone. iOS je po zastupljenosti drugi mobilni operativni sustav trenutno. Aplikacije za iOS operativni sustav razvijaju se u razvojnom okruženju XCode koje je dostupno na macOS operativnim sistemima i besplatno je za preuzimanje preko *App Store* trgovine. Starije aplikacije bile su bazirane na jeziku Objective-C no s predstavljanjem Swift-a 2014. godine, Objective-C polako pada u zaborav.[7]

6.1. Swift programski jezik

Swift programski jezik je nastao kao produkt istraživanja programskih jezika u kombinaciji s desetljećima iskustva u izradi Apple platformi. Swift je jako moćan i intuitivan objektno orijentirani programski jezik za iOS, iPadOS, macOS, tvOS i watchOS operativne sisteme. Pisanje kôda korištenjem Swift programskog jezika čini se interaktivno, sintaksa je sažeta, ali precizna i uključuje moderne značajke koje programeri vole. Swift je optimiziran na razini sakupljača smeća (*engl. Garbage collector*). Naime Swift automatski upravlja memorijom korištenjem metode brojanja referenci, čime se korištenje memorije svodi na minimum.[7]

Primjer Swift modela s nasljeđivanjem:

```
import Foundation

struct Device: Decodable {
    var id: String = ""
    var name: String = ""
    var longitude: Double?
    var latitude: Double?
    var active: Bool = false

    private enum CodingKeys: String, CodingKey {
        case name, longitude, latitude, active
    }
}
```

6.2. Korištene biblioteke

Biblioteke ili popularnijeg naziva okviri (*engl. Frameworks*) su dijelovi kôda koji su napisani da bi se mogli koristiti na više mjesta da bi se postigao DRY (*engl. Don't Repeat Yourself*) način programiranja u što većem smislu. Okviri su samostalni i višekratno upotrebljivi dijelovi kôda koje se može uvesti u aplikaciju. U kombinaciji sa Swift kontrolom pristupa (*engl. Swift Access Control*), okviri uvelike pomažu definirati izvrsno razrađena sučelja koja se mogu puno lakše testirati između raznih modula aplikacije. Okviri se u iOS aplikaciju mogu dodati na razne načine. Najčešći i preporučljivi način je preko Swift upravitelja paketa (*engl. Swift Package Manager - SPM*). U ovom primjeru aplikacije koristi se SPM iz razloga što je puno pregledniji i preporučuje se od strane Apple-a. Neke biblioteke su integrirane i unesene prilikom same kreacije novog projekta, no da bi si olakšali život, programeri često koriste dodatne biblioteke i okvire.

6.2.1. SnapKit

SnapKit je lagani DSL (*engl. Domain Specific Language*) koji rad s automatskim rasporedom vizualnih komponenata na ekranu (*engl. auto layout*) čini lakšim. Automatski raspored je moćan alat za opisivanje odnosa i ograničenja između različitih prikaza i složenijih hijerarhija korisničkog sučelja u aplikacijama, ali pisanje kôda za tu vrstu ograničenja često može biti prilično neintuitivno i teško. [8]

Cilj ovog DSL-a je stvoriti sintaksu koja je intuitivnijia i lakša za korištenje, posebno za ograničenja automatskog rasporeda. SnapKit se može zamisliti i kroz usporedbu alata koji nije naoštren. Na primjer: jabuka se može oguliti i tupim nožem, međutim puno lakše i brže će se taj posao obaviti oštrijim nožem. Dakle isti rezultat se može postići i korištenjem resursa koje Swift nudi sam po sebi, no sa SnapKit-om je to puno lakše i brže.[8]

Usporedba programskog kôda s korištenjem SnapKit-a i bez:

```
// SnapKit
tableView.snp.makeConstraints { (make) in
    make.edges.equalToSuperview()
}

// Nativno
tableView.translatesAutoresizingMaskIntoConstraints = false
NSLayoutConstraint.activate([
    tableView.leadingAnchor.constraint(equalTo:
view.leadingAnchor),
    tableView.topAnchor.constraint(equalTo: view.topAnchor),
    tableView.trailingAnchor.constraint(equalTo:
view.trailingAnchor),
    tableView.bottomAnchor.constraint(equalTo:
view.bottomAnchor)
])
```

6.2.2. Firebase SDK[9]

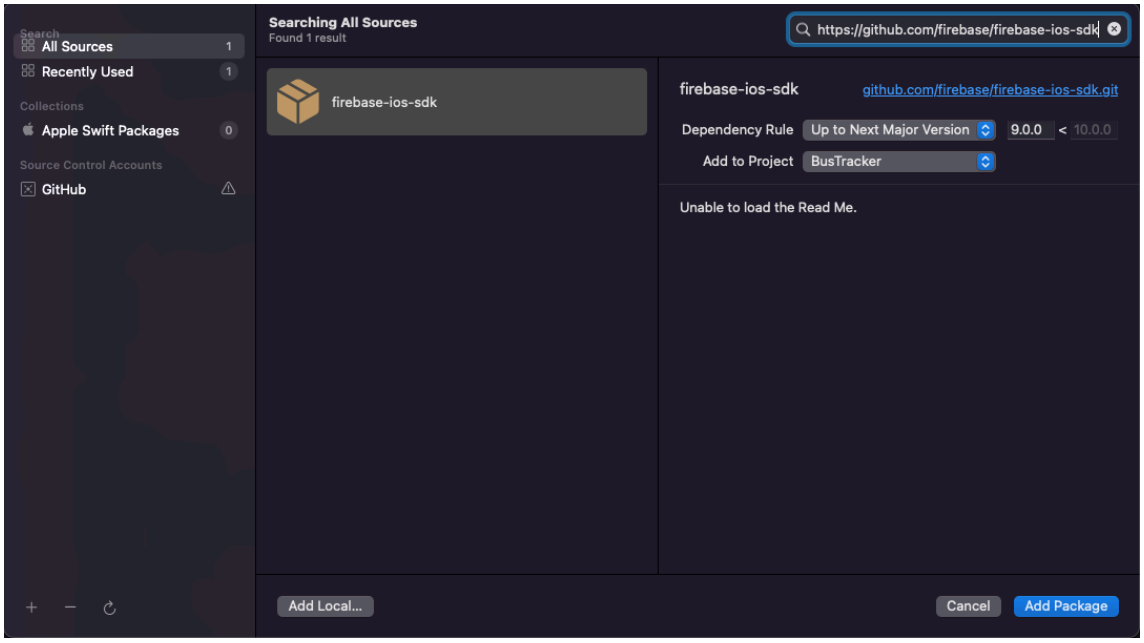
SDK (*engl. Software Development Kit*) je skup programskih alata koji se koriste u programiranju da bi se kreirala aplikacija za specifičnu platformu, u ovom slučaju iOS. SDK uključuje mnoštvo stvari poput biblioteka, dokumentacije, primjera programskog kôda te uputa programerima kako bi ih što lakše integrirali u svoju aplikaciju. Firebase SDK je jedan takav alat. Da bi se moglo čitati podatke s iste baze podataka na koju mikrokontroler zapisuje, mora se integrirati Firebase SDK koji je kreiran specifično za iOS platformu i programski jezik swift. Kako bi se Firebase SDK mogao integrirati u postojeću aplikaciju, mora se zadovoljiti nekoliko uvjeta:

1. Verzija razvojnog okruženja XCode mora biti 13.3.1 ili novija
2. Najmanja verzija operativnog sistema na kojima je aplikacija dostupna mora biti iOS 10, macOS 10.12, tvOS 12 i watchOS 6 ili viša
3. Mora se osposobiti ili fizički uređaj poput iPhone-a ili simulator koji XCode razvojno okruženje nudi samo po sebi
4. Nakon toga kreće se u implementaciju SDK-a u aplikaciju

Prvi korak u integraciji Firebase SDK-a u iOS aplikaciju je kreiranje Firebase projekta na njihovoj online konzoli. (https://console.firebase.google.com/) Nakon što je projekt kreiran, mora se registrirati iOS aplikacija. Aplikacija se registrira na istoj konzoli klikom na gumb "*iOS+"* nakon čega se unosi specifični identifikator aplikacije (*engl. Bundle ID*) koji mora biti formatiran po Apple standardima. Taj identifikator se može pronaći u XCode projektu. Jedan primjer takvog identifikatora je *com.yourcompany.yourproject.* Nakon što je obavljen unos potrebnih podataka klikom na gumb "Register app", aplikacija je uspješno registrirana.

Drugi korak integracije odnosi se na dodavanje konfiguracijske datoteke u XCode projekt koju Firebase generira nakon registracije aplikacije. Konfiguracijska datoteka obično ima naziv "*GoogleService-Info.plist*".

Treći korak je dodavanje Firebase SDK-a u aplikaciju. Da bi se dodala bilo kakva biblioteka ili okvir u aplikaciju, koristi se Swift upravitelj podataka (SPM). Da bi se dodala nova biblioteka, mora se otvoriti XCode projekt na koji dodajemo nove biblioteke, klik na alatnu traku ***File/Add Packages*** otvara prozor sa slike 13.



Slika 8. Prozor s formom za dodavanje Swift paketa i dodatnih biblioteka

Izvor: autor

Unosom adrese web mjesta na SDK (*https://github.com/firebase/firebase-ios-sdk*) u tekstualno polje u desnom gornjem uglu, otvara se ***firebase-ios-sdk*** paket koji se zatim klikom na gumb "Add Package" dodaje u projekt.

Zadnji korak je inicijalizacija Firebase servisa u samoj aplikaciji. Firebase se inicijalizira kod pokretanja aplikacije. iOS aplikacije imaju AppDelegate.swift datoteku u kojoj su pozicionirane metode zaslužne za konfiguraciju životnog ciklusa same aplikacije. U metodi *application(_:didFinishLauchingWithOptions:)* dodaje se linija programskog kôda koja konfigurira Firebase biblioteku.

Primjer programskog kôda inicijalizacije Firebase servise u iOS aplikaciji:

```
import UIKit
import Firebase

@main
class AppDelegate: UIResponder, UIApplicationDelegate {

  var window: UIWindow?

  func application(_ application: UIApplication,
didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {
    self.window = UIWindow(frame: UIScreen.main.bounds)

    // configure firebase
    FirebaseApp.configure()

    let vc = UINavigationController(rootViewController:
DashboardViewController())
    self.window?.rootViewController = vc
    self.window?.makeKeyAndVisible()

    return true
  }
}
```

6.3. Model

Model je komponenta aplikacije koja se odnosi na svu podatkovnu logiku s kojom korisnik ima interakciju. Model može predstavljati ili podatke koji se prenose s jednog korisničkog sučelja ili bilo koje druge podatke povezane s poslovnom logikom. Najčešći primjer je objekt studenta koji će dohvatiti informacije o studentu iz baze podataka, manipulirati njima i po potrebi ih ažurirati i spremiti nazad u bazu podataka.

U ovom radu, koristi se jedan model Device. On predstavlja mikrokontroler koji trenutno šalje podatke. Device model sastoji se od nekoliko obilježja koji su prikazani u tablici 5.

id	Jedinstveni identifikator uređaja
name	Ime uređaja
longitude	GPS koordinata
latitude	GPS koordinata
active	Oznaka ako je uređaj aktivan

Tablica 5. Obilježja modela Device

Primjer programskog kôda modela Device:

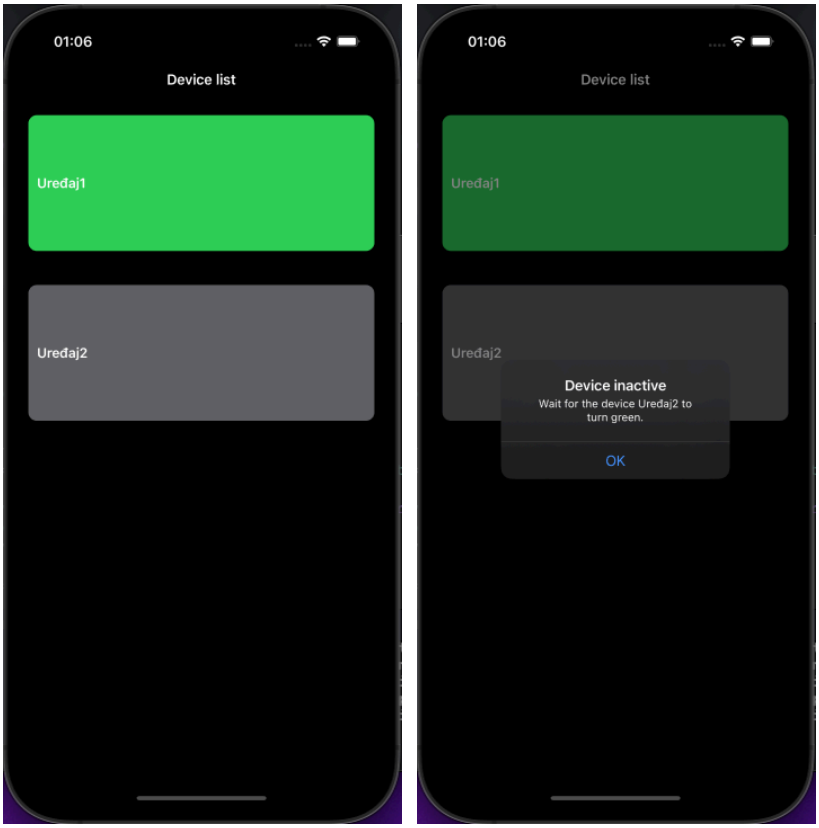
```
import Foundation

struct Device: Decodable {
    var id: String = ""
    var name: String = ""
    var longitude: Double?
    var latitude: Double?
    var active: Bool = false

    private enum CodingKeys: String, CodingKey {
        case name, longitude, latitude, active
    }
}
```

6.4. Moduli

Moduli su organizacijske strukture koje sadrže organizirane datoteke korištene za taj dio aplikacije. U ovom radu korištena su dva modula.

6.4.1. Lista uređaja

Jedan modul je Dashboard modul, odnosno lista uređaja. On je zamišljen kao lista svih uređaja s nazivom i bojom pozadine biranom na temelju njegove aktivnosti. Dashboard modul sastoji se od ViewControllera u kojem se nalazi programski kôd za svu poslovnu logiku vezanu uz dohvat ili manipulaciju podacima i jedna datoteka s posebno kreiranom komponentom za korisničko sučelje vezana uz sam uređaj. Struktura Dashboard modula, konkretno ViewControllera vidljiva je na slici 17.



Slika 9. Prikaz liste uređaja

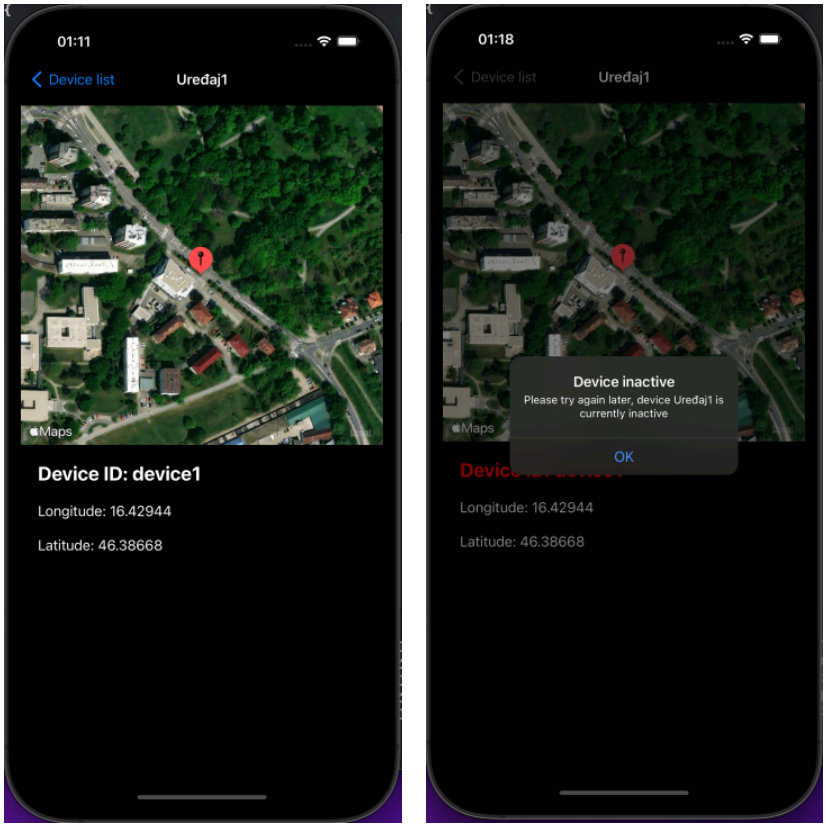
Izvor: autor

Primjer programskog kôda za dohvaćanje podataka s DashboardViewController.swift datoteke:

```
func observe() {
        database.child("devices").observe(.value, with:
{ (snapshot) in
            if snapshot.exists() {
                var devices: [Device] = []
                for device in snapshot.children {
                    guard let device = device as? DataSnapshot,
let data = device.data else { continue }

                    guard var dev: Device = try?
JSONDecoder().decode(Device.self, from: data) else { continue }
                    dev.id = device.key
                    devices.append(dev)
                }
                self.devices = devices
            } else {
                print("No data recieved.")
            }
        })

    }
```

6.4.2. Detalji uređaja

Drugi modul je DeviceDetail modul, odnosno detalji uređaja. Taj modul je zamišljen kao prikaz lokacije uređaja na mapi te podacima o uređaju poput njegovog naziva, koordinata i statusa aktivnosti. Mapa na tom ekranu je interaktivna te podržava približavanje, udaljavanje i kretanje po osima X i Y. Kako se mijenja lokacija uređaja, tako se ikona na mapi kreće i korisnik ima pravovremeni podatak o lokaciji odabranog uređaja. Ukoliko se uređaj isključi u tijeku praćenja, korisniku dolazi obavijest da je uređaj trenutno ugašen i da pokuša ponovno kasnije.



Slika 10. Prikaz detalja uređaja

Izvor: autor

Primjer programskog kôda za dohvat podataka u DeviceViewController.swift datoteci:

```
func observe() {

database.child("devices").child(device.id).observe(.value,
with: { (snapshot) in
            if snapshot.exists() {
                guard let data = snapshot.data else { return }
                guard var dev: Device = try?
JSONDecoder().decode(Device.self, from: data) else { return }
                dev.id = snapshot.key
                self.device = dev
            } else {
                print("No data recieved.")
            }
        })
    }
```

Primjer programskog kôda za ponovno postavljanje lokacije uređaja u

DeviceViewController.swift datoteci:

```
func resetLocation() {
        guard let latitude = device.latitude, let longitude =
device.longitude else {
            let alert = UIAlertController(title: "Invalid
coordinates", message: "Please try again later, device \
(device.name) needs to recalibrate.", preferredStyle: .alert)
            alert.addAction(UIAlertAction(title: "OK",
style: .default, handler: { [weak self] _ in

self?.navigationController?.popViewController(animated: true)
            }))
            self.present(alert, animated: true)
            return

        }
        guard device.active else {
            idLabel.textColor = .red
            let alert = UIAlertController(title: "Device
inactive", message: "Please try again later, device \
(device.name) is currently inactive", preferredStyle: .alert)
            alert.addAction(UIAlertAction(title: "OK",
style: .default, handler: { [weak self] _ in

self?.navigationController?.popViewController(animated: true)
            }))
            self.present(alert, animated: true)
            return
        }

        let center = CLLocationCoordinate2D(latitude: latitude,
longitude: longitude)
        mapRegion.center = center
        mapView.setRegion(mapRegion, animated: true)
        mapView.removeAnnotations(mapView.annotations)
        // Drop a pin at current tracking device location
        let annotation: MKPointAnnotation = MKPointAnnotation()
        annotation.coordinate =
CLLocationCoordinate2DMake(latitude, longitude)
        mapView.addAnnotation(annotation)
        idLabel.text = "Device ID: \(device.id)"
        latitudeLabel.text = "Latitude: \(latitude)"
        longitudeLabel.text = "Longitude: \(longitude)"
        title = device.name
    }
```

7. ZAKLJUČAK

Živimo u ubrzanom vremenu gdje svaka sekunda puno znači. Ideja završnog rada potekla je iz konstantnog kašnjenja autobusa. Kada ljudi žele stići na svoje odredište na vrijeme i imati isplanirani put, žele što točnije informacije o dolascima i polascima javnih prijevoza, međutim nikad se, sa sigurnošću, ne mogu pouzdati u službene podatke dolazaka i polazaka. Ova aplikacija je prototip GPS tragača koji se može integrirati u vozila javnog prijevoza kako bi se u svakom trenutku znalo gdje se nalaze i koliko vremena im treba da bi stigli do zadane stanice te bi tako korisnici mogli organizirati svoje vrijeme i biti sigurni u svakom trenutku.

U radu su opisana razvojna okruženja i tehnologije korištene da bi se ovaj rad realizirao. Opisana je i Firebase realtime baza podataka te način na koji ona upravlja podacima i ažurira ih na svim spojenim uređajima u isto vrijeme. Tema završnog rada je ostvarena korištenjem navedenih tehnologija, a taj prototip se može vrlo lako nadograđivati u budućnosti. Primjerice dodavanjem GSM (*engl. Global System for Mobile communication*) modula koji će se pomoću SIM (*engl. Subscriber Identity Module*) kartice spajati na internetsku mrežu te doprinijeti mobilnosti i neovisnosti o drugim uređajima. Mobilna aplikacija pak može integrirati algoritam koji će prepoznavati korisnikovu lokaciju i na temelju dobivene lokacije uređaja izračunati trajanje dolaska.

8. POPIS LITERATURE

[1] NodeMCU D1 mini, https://diyi0t.com/esp8266-wemos-d1-mini-tutorial/ (10.09.2022.)

[2] Velleman U-Blox NEO-7m, https://www.velleman.eu/products/view/?id=439218 (10.09.2022.)

[3] Firebase, https://firebase.google.com (10.09.2022)

[4] Arduino IDE, https://docs.arduino.cc/software/ide-v1/tutorials/Environment (11.09.2022.)

[5] Xcode, https://www.softwaretestinghelp.com/xcode-tutorial/ (11.09.2022)

[6] Fritzing, https://fritzing.org (11.09.2022)

[7] Swift, https://developer.apple.com/swift/ (12.09.2022)

[8] SnapKit, https://www.raywenderlich.com/3225401-snapkit-for-ios-constraints-in-a-snap (12.09.2022)

[9] Firebase SDK, https://firebase.google.com/docs/ios/setup (12.09.2022)