Sustav internet stvari za prikupljanje dijagnostičkih podataka o osobnom vozilu

Mostovac, Tony

Undergraduate thesis / Završni rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Bjelovar University of Applied Sciences / Veleučilište u Bjelovaru**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:144:475279

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-02-05**

Repository / Repozitorij:

Digital Repository of Bjelovar University of Applied Sciences

VELEUČILIŠTE U BJELOVARU
PREDDIPLOMSKI STRUČNI STUDIJ RAČUNARSTVO

SUSTAV INTERNET STVARI ZA PRIKUPLJANJE DIJAGNOSTIČKIH PODATAKA O OSOBNOM VOZILU

Završni rad br. 10/RAČ/2021

Tony Mostovac

Bjelovar, Rujan 2021.

obrazac ZR - 001

Veleučilište u Bjelovaru

Trg E. Kvaternika 4, Bjelovar

1. DEFINIRANJE TEME ZAVRŠNOG RADA I POVJERENSTVA

Kandidat: **Mostovac Tony** Datum: 30.08.2021. Matični broj: 001980

JMBAG: 0314019499

Kolegij: **SIGURNOST RAČUNALA I PODATAKA**

Naslov rada (tema): **Sustav internet stvari za prikupljanje dijagnostičkih podataka o osobnom vozilu**

Područje: **Tehničke znanosti** Polje: **Računarstvo**

Grana: **Programsko inženjerstvo**

Mentor: **Dario Vidić, mag.ing.el.techn.inf.** zvanje: **predavač**

Članovi Povjerenstva za ocjenjivanje i obranu završnog rada:

1. **Ivan Sekovanić, mag.ing.inf.et comm.techn., predsjednik**
2. **Dario Vidić, mag.ing.el.techn.inf., mentor**
3. **Krunoslav Husak, dipl.ing.rač., član**

2. ZADATAK ZAVRŠNOG RADA BROJ: 10/RAČ/2021

U radu je potrebno izraditi sustav internet stvari koji koristeći pripadne elektroničke komponente prkuplja dijagnostičke podatke s osobnog vozila. Podatke je moguće pregledavati koristeći web preglednik za koji je zadužena web aplikacija koja se brine za skladištanje i prikaz svih dijagnostičkih podataka.

Zadatak uručen: 30.08.2021. Mentor: **Dario Vidić, mag.ing.el.techn.inf.**

Zahvala

Zahvaljujem se svim profesorima i profesoricama Veleučilišta u Bjelovaru, a posebice veliku zahvalu dugujem mentoru mag.ing.el.techn.inf. Dariju Vidiću te dipl.ing.rač. Krunoslavu Husaku, profesoru koji mi je pružao veliku moralnu potporu tijekom studiranja.

Želim se zahvaliti svojim prijateljima, obitelji i najviše djevojci koja me je uvijek podupirala u svemu.

SADRŽAJ

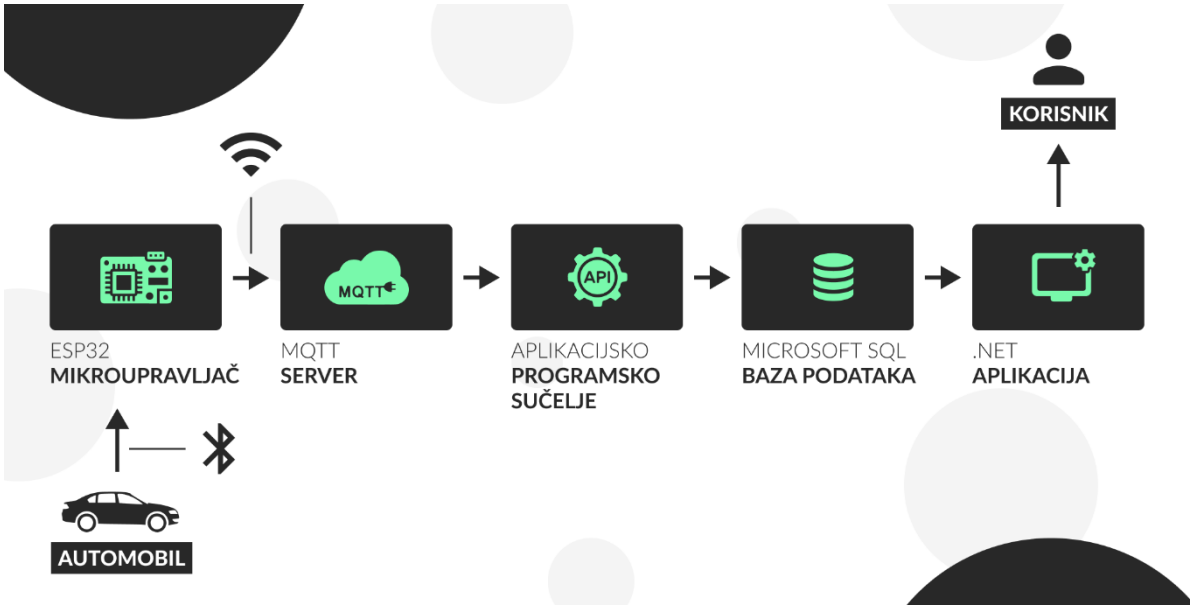
1. UVOD

Godišnje se proizvede gotovo 80 milijuna osobnih automobila, što čini automobilski sektor jednim od najvećih gospodarskih grana sekundarnog sektora. Napredak računalne tehnologije nije doveo velike promjene isključivo u računalnom svijetu već i u potencijalno povezanim granama. Pojava tzv. industrije 4.0, odnosno razvoj pametnog sustava zvanog internet stvari (engl. *Internet of Things*) značajno je dovelo do transformacije postojeće industrije. Ideja da komponente nekog sustava budu povezane, da međusobno komuniciraju ili da jednostavno korisnik ima uvid u stanje sustava zahvatila je i automobilsku industriju. Uz pomoć dijagnostičkog OBD sustava u automobilu, koji će kasnije biti detaljnije opisan, navedeni rad kombinacijom klijentskog i poslužiteljskog dijela uspostavit će sustav koji korisniku omogućuje uvid u podatke osobnog automobila

Poglavlje dva ukratko opisuje pojedine komponente arhitekture IoT sustava rada. U poglavlju tri opisan je OBD sustav koji je usko povezan s mikroupravljačem (poglavlje 4), a poglavlje pet sadrži sve bitne informacije o MQTT protokolu te i njegovoj primjeni u radu. Šesto poglavlje opisuje implementaciju aplikacijskog programskog sučelja, a sedmo Microsoft SQL bazu podataka te strukturu tablica. U osmom poglavlju opisana je .NET aplikacija te u devetom zaključak.

2. ARHITEKTURA IOT SUSTAVA

IoT sustav za prikupljanje dijagnostičkih podataka iz osobnog automobila je sustav razvijen u svrhu završnog rada preddiplomskog studija računarstva, a čine ga sljedeće komponente: .NET web aplikacija, baza podataka, MQTT broker, aplikacijsko programsko sučelje te potreban hardver. Svi prethodno navedeni dijelovi sustava bit će detaljnije opisani kroz rad. Slika 2.1 prikazuje cjelokupnu arhitekturu IoT sustava rada te njihovu međusobnu povezanost.



Slika 2.1: Arhitektura IoT sustava za prikupljanje dijagnostičkih podataka iz vozila

2.1 OBD2 sustav

OBD2 je složeni dijagnostički sustav, ugrađen u automobilu, zadužen za konstantno praćenje stanja i zdravlja vozila. Korištenjem odgovarajućeg hardvera može se pristupiti dijagnostičkim podacima vozila putem OBD2 priključka.

2.2 Hardver

Ključnu ulogu u prikupljanju podataka iz osobnog vozila izvršavaju fizičke komponente tj. hardver. Mikroupravljač ESP32 bluetooth bežičnom vezom spojen je na ELM327 OBD2 adapter. Naravno, komunikacija ne bi bila moguća bez automobila i njegovog OBD2 dijagnostičkog priključka.

2.3 MQTT

MQTT posrednik (engl. *broker*) zadužen je za razmjenu poruka tj. podataka između dijelova sustava prema MQTT principu objave/pretplate (engl. *publish/subscribe*). Za potrebe projekta koristi se konfigurirani i objavljeni posrednik na serveru Veleučilišta u Bjelovaru. Strukturirani uvid u podatke koji se izmjenjuju omogućuje MQTT Explorer programski alat (engl. *software*).

2.4 Aplikacijsko programsko sučelje

Aplikacijsko programsko sučelje, odnosno skraćeno API, pisano je koristeći programski jezik GO, a zaduženo je za prikupljanje podataka s MQTT-a te zapisivanje tih istih podataka u bazu podataka.

2.5 Baza podataka

Baza podataka ostvarena je pomoću Microsoft SQL Servera, a obavlja svrhu pohranjivanja podataka dohvaćenih s MQTT posrednika. Spomenuto dohvaćanje podataka vrši se putem aplikacijskog programskog sučelja.

2.6 .NET aplikacija

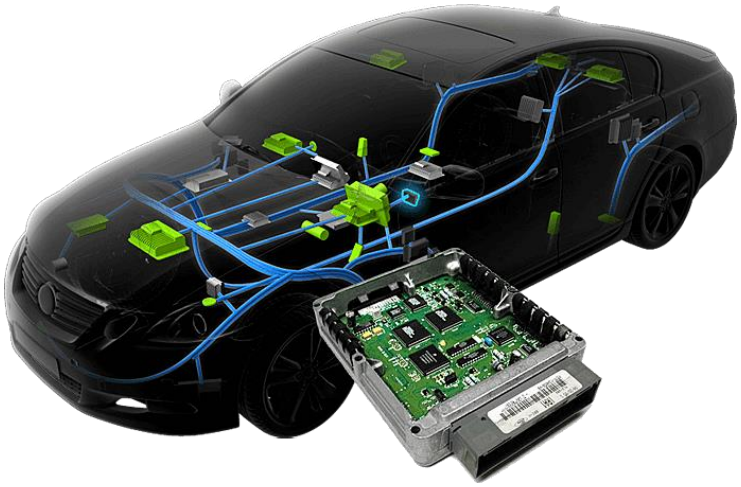
.NET aplikacija razvijena je pomoću .NET softvera otvorenog koda (engl. *open source*). Aplikacija dohvaća podatke iz baze podataka te ih prikazuje korisniku.

3. OBD SUSTAV

3.1 Uvod

Vozači osobnih automobila svjesni su različitih indikatora koji svijetle na njihovim nadzornim pločama (engl. *Dashboard*). Međutim, većina vlasnika automobila ne zna da je riječ o složenom dijagnostičkom sustavu OBD koji konstantno prati stanje i zdravlje vozila. Prvi OBD sustav tj. OBD-1, pojavio se 80-ih godina prošlog stoljeća, ali je zamišljeni sustav urodio plodom tek pojavom OBD-2 standarda. 2001 godine Europska unija propisuje obavezno korištenje OBD dijagnostike za sve aute s benzinskim motorima, a od 2004 godine i za aute s dizelskim motorima.

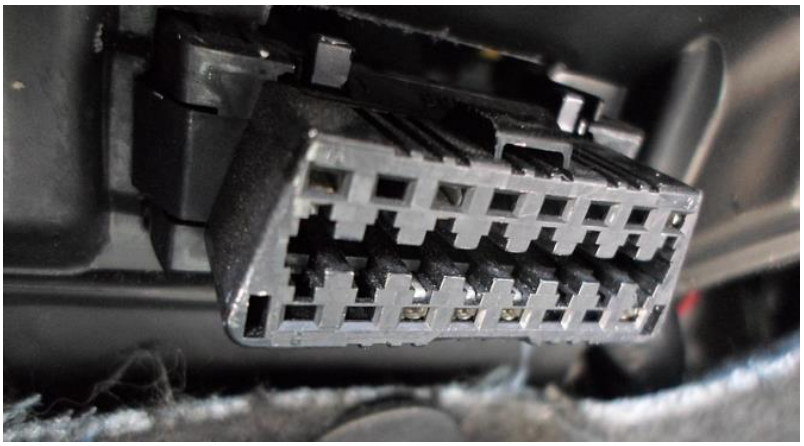
Današnji automobili posjeduju čak do 100 računala tzv. ECU-a (slika 3.1). Njihov zadatak je analiziranje podataka dobivenih s različitih senzora pomoću kojih se izvršavaju određene funkcije npr. podešavanje rada motora te ako se dogodi neka nepravilnost, zapisivanje koda greške u memoriju [1].



Slika 3.1: Primjer ECU-a u automobilu

3.2 Svrha

Svrha OBD-a je usklađivanje ECU-ova u jednu funkcionalnu cjelinu zbog lakšeg uvida u stanje automobila. Svi moderni automobili posjeduju OBD priključak, a primjer priključka prikazan je na slici 3.2. Pozicija priključka je uvijek u kabini vozila lako dostupna vozaču, a najčešće mjesto je ispod volana zaštićena plastičnim poklopcem [1].



Slika 3.2: Primjer OBD priključka

3.3 OBD2 PID

OBD2 PID su kodovi koji služe za zahtjeve dohvata podataka iz vozila zbog dijagnostičke ili analitičke potrebe. Prilikom dohvata informacija, korisnik odgovarajućim alatom šalje zahtjev koji sadrži PID u kombinaciji s načinom rada (engl. *mode*). Oba prethodna dijela zahtjeva šalju se u heksadekadskom brojevnom sustavu. Tablica 3.1 prikazuje popis podržanih načina rada, a tablica 3.2 neke od podržanih PID-a kao primjer što se može dohvatiti [2]. Konkretan primjer zahtjeva bit će prikazan u poglavlju 3.4.2 implementacije OBD2 adaptera u sustavu.

Tablica 3.1: Podržani načini rada

Mode (hex)	Opis
01	Prikaži trenutne podatke
02	Prikaži „zamrznute“ podatke (engl. *freeze frame data*)
03	Prikaži spremljene dijagnostičke kodove pogrešaka
04	Očisti dijagnostičke kodove pogrešaka
05	Rezultati ispitivanja, nadzor senzora kisika
06	Rezultati ispitivanja, ostale komponente
07	Prikaži dijagnostičke kodove pogrešaka na čekanju
08	Operacije upravljanja ugrađenih komponenata
09	Zatraži informacije o vozilu
0A	Trajni dijagnostički kodovi pogrešaka

Tablica 3.2: Primjer PID-ova

Mode (hex)	PID (hex)	Vraćeni bajtovi podataka	Opis	Minimalna vrijednost	Maksimalna vrijednost	Mjerna jedinica
01	05	1	Temperatura rashladne tekućine motora	-40	215	°C
01	0A	1	Pritisak goriva	0	765	kPa
01	0C	2	Broj okretaja motora	0	16 383	rpm
04	02	2	Očisti jednostavne kodove pogrešaka	/	/	/
09	02	5x5	Identifikacijski broj vozila	/	/	5 ASCII kodiranih linija

3.4 ELM327 adapter

ELM327 (slika 3.3) je tvornički programirani mikroupravljač od strane tvrtke ELM Electronics koji služi za prikupljanje informacija o osobnom vozilu putem OBD priključka. Dohvat podataka može se odvijati putem UART-a spojenog na ručni dijagnostički alat ili računalnog programa spojenog putem USB-a, bluetootha ili WiFi-ja. Google Play mrežna trgovina aplikacija trenutno nudi velik broj aplikacija za povezivanje s ovakvim adapterom.

Slika 3.3: ELM327 bluetooth adapter

3.4.1 Specifikacije izabranog adaptera

Klasični ELM327 OBD2 adapter, prikazan na prethodnoj slici 3.3, izabran je za ovaj rad zbog svoje niske cijene te mogućnosti spajanja na mikroupravljač bluetooth bežičnom mrežom.

Podržane funkcije adaptera [3]:

- Čitanje dijagnostičkih kodova grešaka,
- Ispravljanje nekih jednostavnijih grešaka, npr. poznata *check engine* lampica,
- Čitanje podataka u stvarnom vremenu (engl. *real-time data*), npr. trenutna brzina, količina goriva, radna temperatura motor i sl.,

Podržani protokoli adaptera [3]:

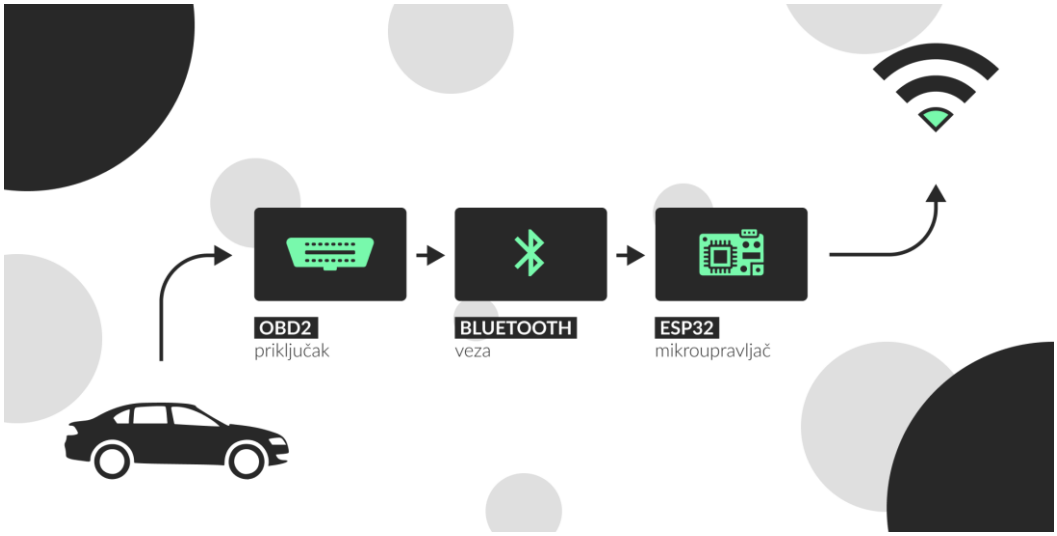
- Fordov SAE-J1850 PWM,
- GM-ov SAE-J1850 VPW,
- ISO 9141-2 protokol za Europska, Azijska i Chrysler vozila
- ISO 14230-4 poznat kao KWP2000 protokol za Azijska vozila,
- ISO 15765-4 (CAN) moderni protokol

3.4.2 Implementacija u sustavu

ELM327 kao zasebna komponenta u sustavu nema posebnu svrhu zbog svoje nemogućnosti prikazivanja podataka, zato se u radu koristi ESP32 mikroupravljač koji određenim funkcijama dohvaća određene podatke te ih prosljeđuje dalje. Dohvaćanje podataka vrši se PID zahtjevima koji su predefinirani u *ELMduino* programskoj biblioteci (engl. *library*) za Arduino IDE (poglavlje 4.1). Primjer zahtjeva i nešto više o programskoj biblioteci detaljnije je opisano u poglavlju 4.3.1.

4. MIKROUPRAVLJAČ

Rad je zahtijevao svestran mikroupravljač s WiFi i bluetooth bežičnim mogućnostima zbog interakcije s ELM327 adapterom i spajanjem na internetsku mrežu. ESP32 je mikroupravljač koji ispunjava zahtjeve sustava, a programiran je u Arduino IDE razvojnom okruženju. Slika 4.1. prikazuje povezanost mikroupravljača s ostatkom hardvera..



Slika 4.1: Povezanost hardvera

4.1 Arduino IDE

Arduino IDE je razvojno okruženje otvorenog koda (engl. *open source*) namijenjeno za programiranje Arduino ploča (engl. *board*) koristeći programske jezike C i C++, a dostupno je na više platformi [4]. Funkcionalnost i opsežnost programiranja mikroupravljača koristeći Arduino IDE se bazira na programskim bibliotekama koje su često pisane od strane Arduino korisnika i/ili zajednice [5]. Za potrebe rada korištene su četiri programske biblioteke čija će svrha i funkcionalnost biti opisana u 4.3 poglavlju.

4.2 ESP32

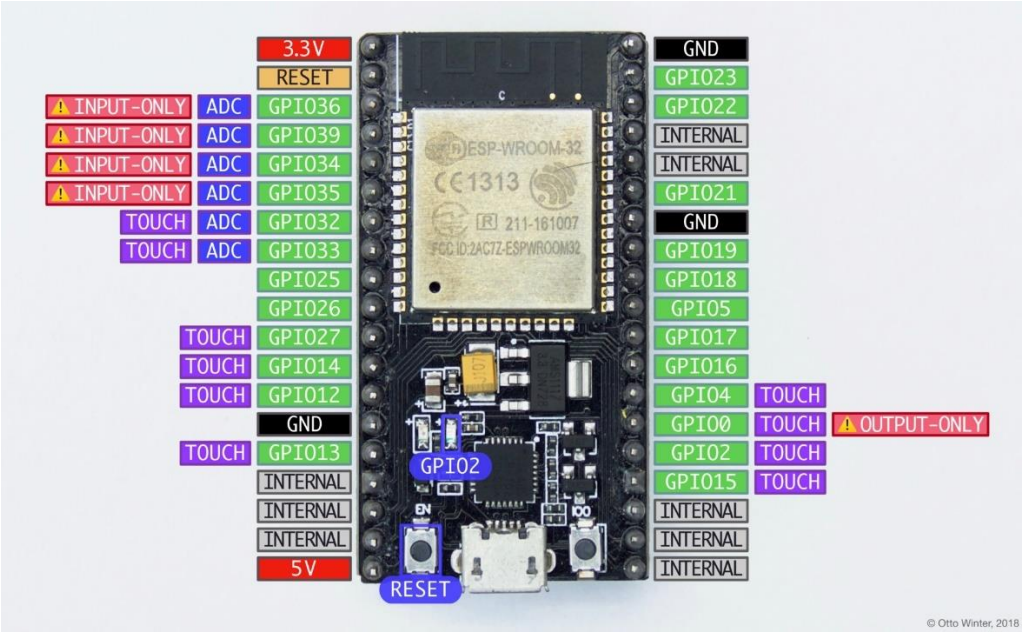
ESP32 je serija jeftinih integriranih krugova malih snaga s ugrađenim WiFi i Bluetooth modulima. Spomenuti mikroupravljač najčešće koristi Tensilica Xtensa LX6 32-bitni mikroprocesor u dvojezgrenoj ili jednojezgrenoj varijaciji. Izašao je 2016 godine kao nasljednik ESP8266 mikroupravljača [6].

ESP32 je izabran za ulogu mikroupravljača zbog svoje relativno niske cijene i dobre internetske podrške, a za potrebe ovog sustava korišten je NodeMCU ESP32s razvojni modul.

Tehničke karakteristike čipa [7]:

- Mikroprocesor: Tensilica Xtensa LX6 32-bit sa frekvencijom takta do 240 MHz,
- Nazivni napon: 2.2 – 3.6 V,
- Dva 64-bitna tajmera (engl. *Timer*),
- Tajmer za nadzor ispravnog rada (engl. *Watchdog timer*)
- Trideset četiri opće namjenskih ulaza/izlaza (I/O),
- Osamnaest ADC pretvornika rezolucije 12 bitova,
- Dva DAC pretvornika rezolucije 8 bitova,
- Deset kapacitivnih senzora na dodir,
- Šesnaest PWM pinova,
- Flash memorija: 4 MB,
- SRAM memorija: 520 KB,
- ROM memorija: 448 KB,
- Ugrađeni WiFi modul s brzinom do 150 Mbps-a,
- Ugrađeni v4.2 Bluetooth modul, SPI, I2C i UART komunikacija

Rad nije zahtijevao korištenje dostupnih pinova, ali su zato iskorišteni bluetooth i WiFi modul. Prethodno spomenuti moduli su neophodni u uspostavi komunikacije s ELM327 dijagnostičkim uređajem te s MQTT brokerom. Raspored pinova te primjer NodeMCU ESP32s razvojnog modula prikazan je na slici 4.2.



Slika 4.2: Raspored pinova NodeMCU ESP32s razvojnog modula [7]

4.3 Funkcionalnost u IoT sustavu

ESP32 mikroupravljač ima dvije vrlo važne uloge u sustavu. Prva je dohvaćanje podataka od ELM327 adaptera bluetooth vezom, a druga je prosljeđivanje tih istih podataka MQTT posredniku (poglavlje 5) putem WiFi mreže.

Prethodne navedene uloge ne bi bile moguće bez korištenja dodatnih programskih biblioteka za Arduino IDE razvojno okruženje. Svrha rada zahtijevala je upotrebu četiri različitih programskih biblioteka:

- *BluetoothSerial*
- *ELMduino*
- *WiFi*
- *PubSubClient*

4.3.1 Implementacija programskih biblioteka

BluetoothSerial programska biblioteka korištena je za uspostavu bluetooth komunikacije s ELM327 adapterom. Programski kod (programski kod 4.1) za provođenje komunikacije je jednostavan, potrebno je navesti pin „1234“ nužan za uparivanje s OBD2 adapterom (59. linija), uključiti serijski bluetooth na mikroupravljaču te učiniti ESP32

dostupnim pod imenom „ArduHUD“ (60. linija) i uspostaviti vezu s adapterom na način gdje se kao parametar *connect()* funkcije (62. linija) navede naziv uređaja tj. „OBDII“.

Programski kod 4.1: Implementacija BluetoothSerial programske biblioteke

```
1   #include "BluetoothSerial.h"
    …
59  SerialBT.setPin("1234");
60  SerialBT.begin("ArduHUD", true);
61
62  if (!SerialBT.connect("OBDII"))
    …
```

ELMduino programska biblioteka ima svrhu lakše razmjene podataka između mikroupravljača s OBD2 bluetooth adapterom. Komunikacija se ostvaruje uz pomoć *BluetoothSerial* biblioteke, ali slanje PID zahtjeva koristeći jednostavne funkcije omogućuje *ELMduino*. U programskom kodu 4.2 prikazan je primjer pozivanja funkcija za dohvaćanje brzine vozila (programska linija 91) i temperature rashladne tekućine motora (programska linija 92).

Programski kod 4.2: Pozivanje ELMduino funkcija za prikupljanje željenih podataka

```
2   #include "ELMduino.h"
    …
91  float veh_speed = myELM327.kph():
92  float temp = myELM327.engineCoolantTemp();
    …
```

Jedan od prikupljenih podataka je broj okretaja motora u minuti. Prethodno navedeni podatak dohvaća se pomoću funkcije *rpm()* koja kao parametre (linija 3) sadrži način rada i PID. Dakle, „01“ način rada u kombinaciji s „0C“ PID-om dohvaća brzinu okretaja motora. Programski kod 4.3 prikazuje definiciju predefinirane *rpm()* funkcije u samoj biblioteci sa spomenutim parametrima.

Programski kod 4.3: Definicija ELMduino rpm() predefinirane funkcije

```
1  float ELM327::rpm()
2  {
3    if (queryPID(1, 0x0C)
4      return conditionResponse(findResponse(), 2, 1.0 / 4.0);
5
6    return ELM_GENERAL_ERROR;
7  }
```

Slika 4.3 prikazuje rezultat *rpm()* funkcije za dohvat broja okretaja motora (u minuti) u konzoli Arduino IDE razvojnog okruženja.

```
Query string: 010C
Clearing input serial buffer
Sending the following command/query: 010C
        Received char: 4
        Received char: 1
        Received char: 0
        Received char: C
        Received char: 0
        Received char: C
        Received char: 3
        Received char: 1
        Received char: \r
        Received char: \r
        Received char: >
Delimiter found
All chars received: 410C0C31
Expected response header: 410C
Single response detected
64-bit response:
        responseByte_0: 49
        responseByte_1: 12
        responseByte_2: 0
        responseByte_3: 0
        responseByte_4: 0
        responseByte_5: 0
        responseByte_6: 0
        responseByte_7: 0
Broj Okretaja Motora: 780.25
```

Slika 4.3: Ispis broja okretaja motora

Na slici je prikazan „010C“ upit/zahtjev (engl. *Query*) te odgovor u 64-bitnom obliku, prvi bajt (A) iznosi 16 kao i drugi (B) te se prema relaciji 4.1 izračuna broj okretaja.

$$\text{broj okretaja motora} = \frac{(256 * \text{A}) + \text{B}}{4} \tag{4.1}$$

Svrha *WiFi* programske biblioteke jest spajanje mikroupravljača na internetsku/WiFi mrežu zbog prosljeđivanja podataka MQTT posredniku. Uspostava veze je vidljiva na programskom kodu 4.4 gdje programske linije 12 i 13 sadrže deklaraciju i inicijalizaciju globalnih varijabli za naziv i lozinku WiFi mreže, a linija 38 prikazuje korištenje funkcije *begin()* koja kao parametre sadrži prethodno navedenih varijable.

Programski kod 4.4:Uspostava veze s WiFi mrežom

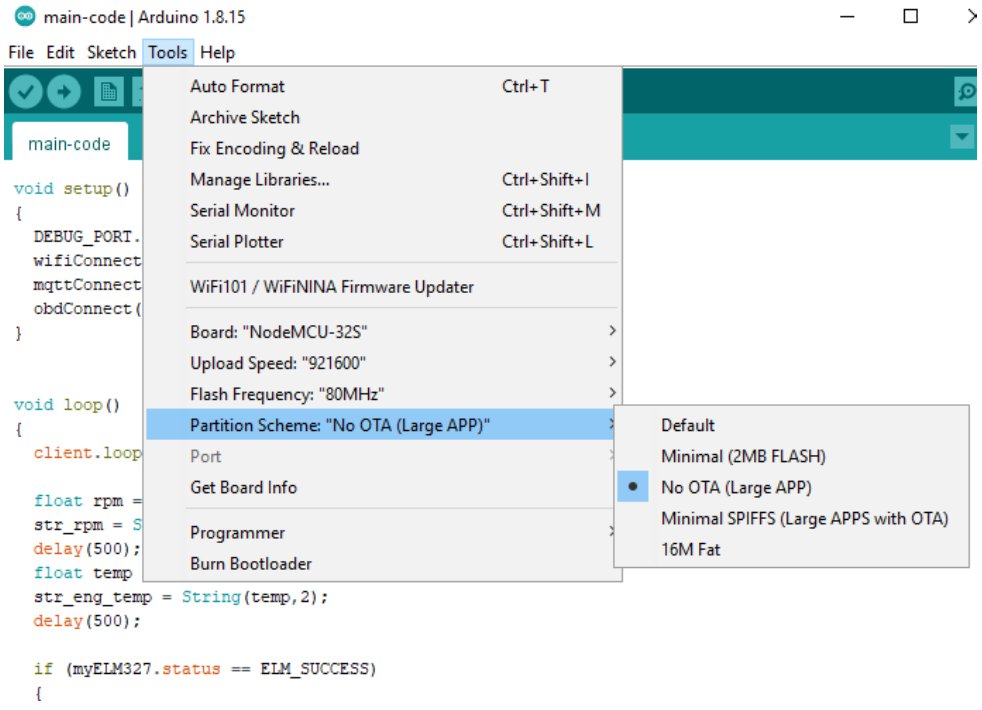
```
3   #include "WiFi.h"
    …
12  const char* ssid = "naziv_mreze";
13  const char* password = "lozinka_mreze";
    …
38  WiFi.begin(ssid,password);
    …
```

PubSubClient je programska biblioteka zadužena za spajanje na MQTT posrednik te se uz pomoć nje na jednostavan način prosljeđuju željeni podaci metodom objave.
Zbog svoje specifičnosti i usko povezanosti s MQTT protokolima implementacija navedene programske biblioteke bit će prikazana u poglavlju 5.3.

4.3.2 Problemi u implementaciji

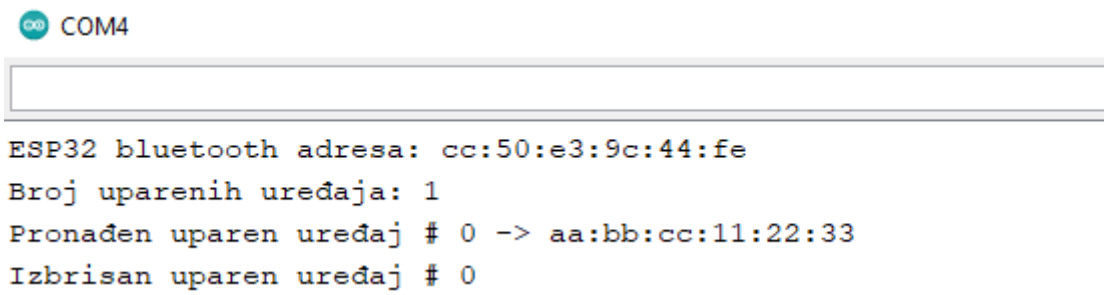
Složeni sustavi poput IoT-a temelje se na mnogo različitih tehnologija, protokola, razmjena informacija i sl. te je začuđujuće ako ne dođe do nekih problema. Neki od takvih problema (na razini ESP32) su manjak prostora/memorija na mikroupravljaču za pohranu programskog koda i problemi u uspostavi bluetooth komunikacije s OBD2 adapterom. Godina proizvodnje automobila je također važna, službena preporuka od strane autora *ELMduino* programske biblioteke je korištenje automobila proizvedenih nakon 2008 godine.

Problem s nedostatkom prostora za programiranje mikroupravljača javio se zbog prevelikog programskog koda. Naime, programski kod zauzima cca 1.5 MB dok mikroupravljač podržava do 1.2 MB prema standardnim postavkama. ESP32 sadrži 4MB *flash* memorije, međutim zbog rasporeda particija samo je 1.2MB dostupno za programski kod. Arduino IDE razvojno okruženje nudi mogućnost promjene postavki particija te na taj način povećava particiju za programski kod [9]. Na slici 4.4. je prikazano kako promijeniti postavke particija na spomenutom okruženju



Slika 4.4: Prikaz promjena postavki particija

Poteškoće u uspostavi bluetooth veze između uređaja javljaju se ako se mikroupravljač ponovno želi spojiti s adapterom, dakle, kada je mikroupravljač uparen s adapterom i uspješno spojen, ako dođe do isključenja adaptera ili ESP32-a ponovno povezivanje neće biti moguće. Problem ponovnog povezivanja rješava se na način gdje se na mikroupravljač prenese programski kod, dostupan na GitHUB-u [10], zadužen za brisanje svih uparenih bluetooth uređaja. Slika 4.5. prikazuje rezultat brisanja uparenog „OBDII“ uređaja MAC adrese „aa:bb:cc:11:22:33“ na serijskom monitoru Arduino IDE razvojnog okruženja.



Slika 4.5: Rezultat brisanja uparenih uređaja

IoT sustav rada testiran je na osobnim vozilima proizvedenim prije i poslije 2008 godine. Isproban je na automobilu Alfa Romeo 147 1.9 JTD (2002.), Renault Scenic 1.5 dCi (2004.), VW Golf VI 1.9 TDI (2009.) te Renault Megane (2011.) 1.5 dCi. Poslije testiranja, sa sigurnošću se može zaključiti da bežični bluetooth prijenos podataka putem spomenutog hardvera je nepouzdan za starije automobile, odnosno, za automobile proizvedene prije 2008 godine. Alfa Romeo dao je neispravne/čudne podataka, s Renault Scenicom komunikacija nije uopće mogla biti uspostavljena, a s Golf-om VI i Renault Meganeom nije bilo nikakvih problema.

5. MQTT

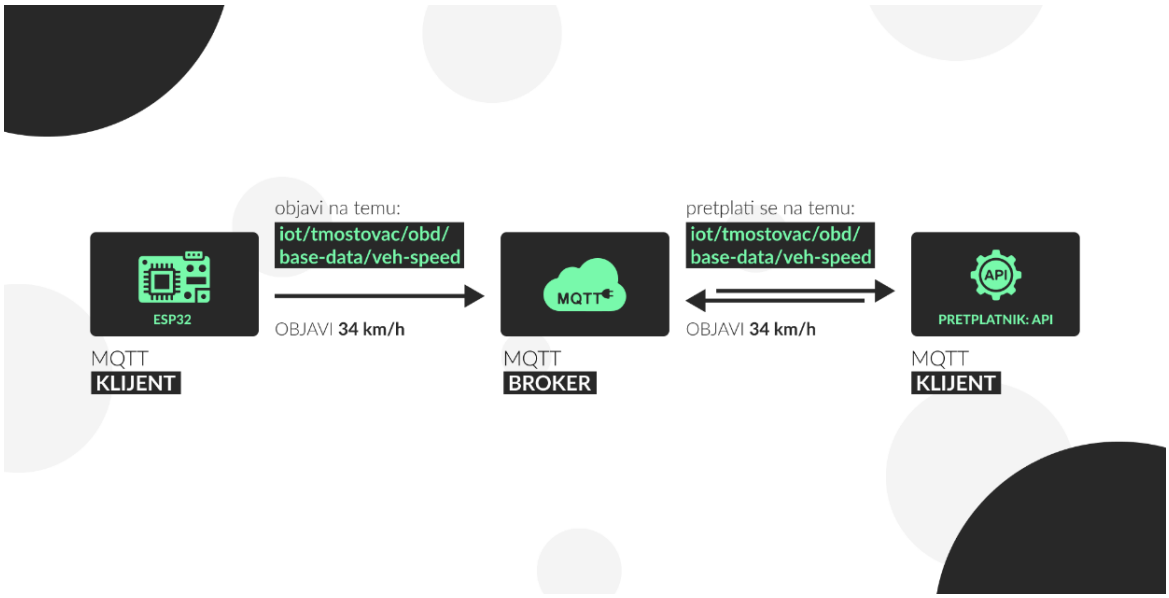
5.1 Općenito

MQTT je mrežni protokol koji se temelji na publish-subscribe metodi razmjene poruka između međusobno poveznih uređaja. MQTT protokol je OASIS standard za IoT sustave , a primjenjuje se u logističkim, automobilskim, transportnim, prerađivačkim i sl. industrijama. MQTT klijentima potrebni su minimalni resursi te kao takvi mogu biti korišteni na malim mikroupravljačima poput ESP32, a zaglavlje MQTT poruka je malo kako bi se optimizirala propusnost mreže. MQTT nudi mogućnost komunikacije u oba smjera, dakle, klijent-poslužitelj i poslužitelj-klijent, što olakšava emitiranje (engl. *Broadcasting*) poruka grupama. IoT sustavu vrlo je važna pouzdanost isporuke poruka, zato MQTT nudi 3 definirane razine kvalitete usluga slanja poruka: 0 - najviše jednom, 1 - barem jednom i 2 - točno jednom. Olakšani pristup šifriranja podataka vrši se pomoću TLS-a, a provjere autentičnosti putem OAuth standarda za delegiranje pristupa [11].

MQTT protokol stvoren je zbog potrebe stvaranja protokola koji ima vrlo visoku učinkovitost u smislu propusnosti (engl. *bandwidth-efficient*) te koji troši malo električne energije jer je dio uređaja povezan satelitskom vezom, a prvi takav primjer korišten je za nadzor naftovoda u pustinji 1999 godine [12].

5.2 Arhitektura MQTT-a

MQTT sustav sastoji se dva tipa mrežnih entiteta, posrednik poruka i klijenata. MQTT posrednik je server koji pohranjuje primljene poruke od klijenta te ih prosljeđuje drugim klijentima. Publish-subscribe arhitektura onemogućuje slanje poruka pošiljatelja (engl. *publisher*) direktno primatelju/pretplatniku (engl. *subscriber*) već se podaci kategoriziraju u obliku tema na serveru (engl. *topic*). Tim objavljenim podacima klijent, odnosno pretplatnik, pristupa na način gdje se pretplaćuje na temu koju želi kako bi dohvatio poruke [13]. Slika 5.1. prikazuje primjer objavi-pretplati arhitekture IoT sustava za prikupljanje dijagnostičkih podataka iz osobnog vozila.



Slika 5.1: MQTT Publish-Subscribe arhitektura

5.3 Implementacija u IoT sustavu rada

MQTT je neophodni posrednik između hardverskog i softverskog dijela zadužen za prikupljanje podataka od strane mikroupravljača te da omogući te iste podatke dostupnim ostalim dijelovima sustava npr. API-ju. Na razini mikroupravljača korištena je već spomenuta programska biblioteka PubSubClient koja daje ESP32 funkcionalnost MQTT klijenta, a programski alat, odnosno posrednik, MQTT Explorer nudi strukturirani uvid u poruke. Tablica 5.1. prikazuje primjer nekih korištenih podataka u svrhu rada, njihove dodijeljene teme, varijabla koja se šalje na tu temu, u obliku *stringa*, opis podatka te mjerna jedinica.

Tablica 5.1: Primjer korištenih poruka i pripadajućih informacija

Tema	Varijabla	Opis	Mjerna jedinica
iot/tmostovac/obd/basic-data/veh-speed	str_veh_speed	Trenutna brzina osobnog vozila	km/h
iot/tmostovac/obd/basic-data/eng-rpm	str_eng-_rpm	Trenutni broj okretaja motora	r/min
iot/tmostovac/obd/aditional-data/maff-flow-rate	str_maff_flow_rate	Masa protoka zraka koji ulazi u motor	g/s

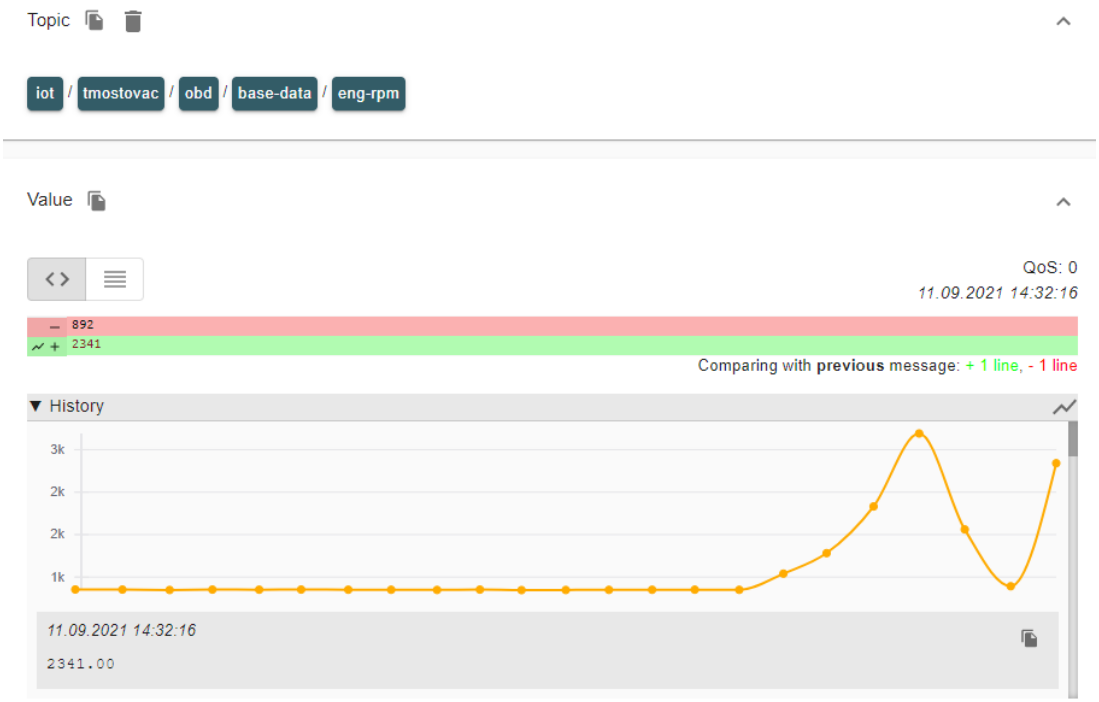
iot/tmostovac/obd/diagnostic-dana/dist-traveled-with-mil	str_ dist-traveled-with-mil	Prijeđeni put s indikatorskom lampicom kvara	km

5.3.1 MQTT Explorer

IoT sustav za prikupljanje dijagnostičkih podataka koristi MQTT protokol kroz MQTT Explorer posrednik čija je svrha kategoriziranje poruka u teme i strukturirani uvid u poruke. Neke od funkcija su:

- Vizualizacija tema te njihove aktivnosti
- Traženje/filtriranje tema
- Rekurzivno brisanje tema
- Objava tema
- Različiti prikazi trenutnih i prethodnih primljenih poruka

MQTT Explorer dostupan je na više operacijskih sustava te je optimiziran za obradu tisuća tema i nekoliko stotina poruka po minuti [14]. Slika 5.2. prikazuje vremenski graf broja okretaja motora kao primjer strukturiranog uvida u poruke.



Slika 5.2: Vremenski graf broja okretaja motora na MQTT Explorer-u

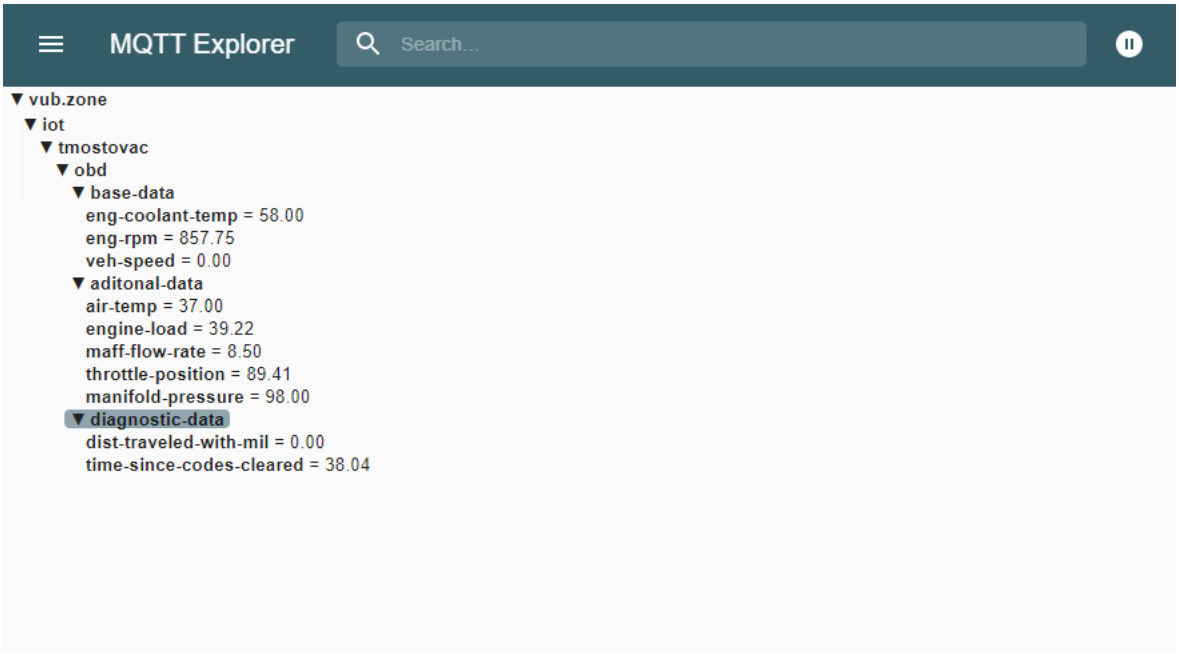
5.3.2 *PubSubClient* programska biblioteka

PubSubClient programska biblioteka korištena je za objavljivanje podataka na MQTT server. Kako bih se podaci uopće mogli objaviti, potrebno je prvo uspostaviti vezu s MQTT-om. Programski kod 5.1. prikazuje sve potrebno za uspostavu veze, a programske linije 18 do 21 čine potrebne autorizacijske i konekcijske parametre, kao npr. naziv posrednika, lozinka, korisničko ime i sl. Funkcija *mqttConnect()* zadužena je za spajanje na server uz pomoć navedenih parametra, a funkcija *getBaseData()* dohvaća podatke (programska linija 89) te ih objavljuje na MQTT (programska linija 91 i 92). Važno je napomenuti da funkcija *publish()* kao parametar prima naziv teme te dohvaćeni podatak iz vozila u obliku *stringa.*

Programski kod 5.1: Primjena PubSubClient programske biblioteke

```
4   #include <PubSubClient.h>…
    …
18  const char *mqtt_broker = "naziv-servera";
19  const int mqtt_port = port;
20  const char *mqtt_username = "korisnicko-ime";
21  const char *mqtt_password = "lozinka";
    …
45  void mqttConnect(){
46    client.setServer(mqtt_broker, mqtt_port);
47    Serial.println("Connecting to public emqx mqtt broker.....");
48    if(client.connect("tmostovac", mqtt_username, mqtt_password)){…}
49    else {…}
50  }
    …
88  void getBaseData(){
89    float eng_coolant_temp = myELM327.engineCoolantTemp();
90    …
91    client.publish("iot/tmostovac/obd/base-data/eng-coolant
92  temp",String(eng_coolant_temp).c_str());
93    …
98  }
    …
```

Nakon što su podaci uspješno objavljeni na MQTT, putem opisanog programskog alata MQTT Explorer dostupan je uvid u objavljene teme. Slika 5.3. prikazuje zadnje objavljene podatke i njihove pripadajuće teme.



Slika 5.3: Prikaz zadnjih objavljenih podatka

6. APLIKACIJSKO PROGRAMSKO SUČELJE

Aplikacijsko programsko sučelje IoT sustava za prikupljanje dijagnostičkih podataka iz automobila služi za dohvaćanje prethodno navedenih podataka s MQTT-a te prosljeđivanje tih istih u bazu podataka. API je napravljen od strane voditelja kolegija IoT te je pisan u programskom jeziku GO, a prilagođen je sustavu i rekonfiguriran od strane autora rada.

6.1 Općenito

Ideja svakog API-ja kao softverskog posrednika jest povezivanje i uspostava komunikacije između dviju aplikacija. U usporedbi s korisničkim sučeljem, koje povezuje aplikaciju s korisnikom, aplikacijsko programsko sučelje povezuje dijelove softvera i nije namijenjeno za korištenje izravno od strane krajnjeg korisnika [15].

API omogućuje slanje ili primanje informacija/podataka putem određenih zahtjeva, odnosno pozivima. Takva vrsta komunikacije može se izvršiti uz pomoć JSON-a. JSON je format datoteke i format razmjene podataka strukturiran u obliku teksta koji je čovjeku razumljiv [16].

Osnovne pozive čine:

- GET – dohvaćanje,
- PUT – ažuriranje,
- POST – stvaranje,
- DELETE - brisanje

6.2 Programski jezik GO

GO je programski jezik otvorenog koda razvijen od strane Google tvrtke, iako je relativno nov svoju primjenu pokazuje u brojnim aplikacijama kao što su Netflix, Google, PayPal, Twitter i itd. [17]. Eksperimentalni programski jezik GO razvijen je kako bi spojio performanse i sigurnosne prednosti C++-a s brzinom dinamičkog jezika kao što je Python [18].

Originalna svrha GO-a je bila razvoj programa povezanih s mrežnom infrastrukturom, međutim, GO je ubrzo pronašao svoju funkcionalnost u širokom rasponu različitih tipova aplikacija, kao npr:

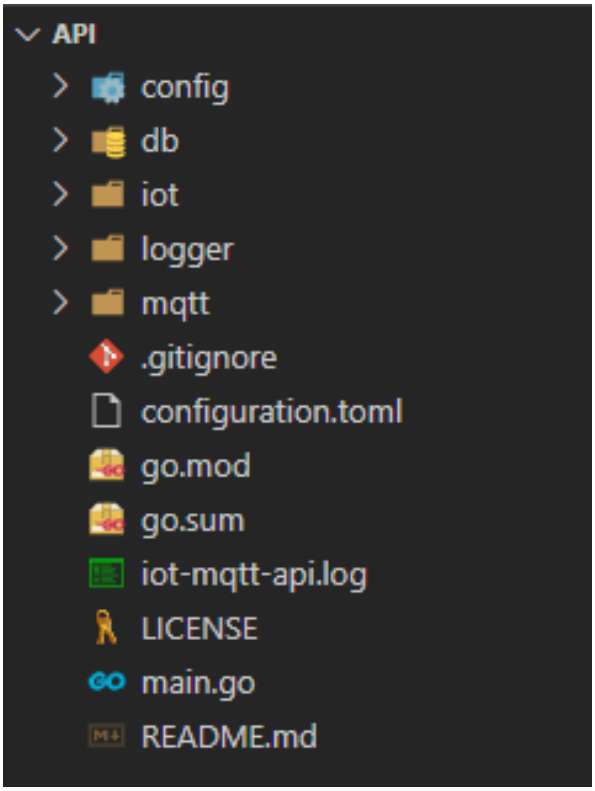
- Aplikacije temeljene na oblaku (engl. *cloud-based*),

- DevOps automatizacija,
- Alati za naredbeni redak (engl. *command line*),
- Primjena u svijetu umjetne inteligencije i znanosti o podacima (engl. *data science*),
- Programiranje mikroupravljača, robotika i videoigre

Naglu popularnost programski jezik GO nije stekao samo zbog široke primjene već i zato što ga je, prema riječima programerima, jednostavnije i brže naučiti nego ostale programske jezike koji su trenutno u trendu [19].

6.3 Dijelovi API-ja u radu

API je pisan pomoću besplatnog programskog alata za uređivanje koda Visual Studio Code koristeći programski jezik GO. Na slici 6.1. prikazan je popis mapa i datoteka koje API koristi za implementaciju funkcionalnosti u radu



Slika 6.1: Stablo mapa i datoteke

Sve prikazane mape predstavljaju GO paket koji se izvršava neku funkcionalnost API-ja. Spomenute pakete čine:

- Konfiguracija (*config*),
- Baza podataka (*db*),
- MQTT (*mqtt*),
- IoT (*iot*),
- Zapisivanje grešaka (*logger*)

Konfiguracijski dio, *config* mapa te *configuration.toml* datoteka, bavi se povezivanjem na bazu podataka i MQTT server. Datoteka *configuration.toml* sadrži potrebne podatke kao što su lozinka, korisničko ime i sl. (slika 6.2.).

Slika 6.2: Konekcijski podaci

Paket za bazu podataka, *db* mapa, sadrži funkcije i tipove podataka koje omogućavaju komunikaciju s bazom podataka, a povezivanjem na MQTT bavi se *mqtt* paket, tj. *mqtt* mapa. Funkcionalnost zapisivanja grešaka obavlja *logger* paket, a glavni dio API-ja vrši *iot* paket koji je detaljnije opisan u poglavlju 6.4.

6.4 Primjena u radu

Temeljnu funkcionalnost API-ja obavlja *iot* paket, naime, u *iot.go* datoteci nalazi se sve potrebno za deklariranje varijabli, pretplatu na teme te unos podataka u bazu podataka. Slika 6.3. prikazuje funkciju *ReceiveData()* koja na temelju pretplaćene teme parsira dohvaćeni *string* u željeni tip podataka. Za potrebe rada podaci su pretvoreni u oblik decimalnog broja s pomičnom točkom (engl. *float*), jer se radi o podacima kao što su temperatura, udaljenost i sl.

```
func (iot *IOT) ReceiveData() {
    for {
        message := <-iot.mqtt.Messages

        switch message.Topic() {
        case "iot/tmostovac/obd/base-data/veh-speed":
            iot.data.VehicleSpeed, _ = strconv.ParseFloat(string(message.Payload()), 64)
        case "iot/tmostovac/obd/base-data/eng-rpm":
            iot.data.EngineRPM, _ = strconv.ParseFloat(string(message.Payload()), 64)
        case "iot/tmostovac/obd/base-data/eng-coolant-temp":
            iot.data.EngineCoolantTemperature, _ = strconv.ParseFloat(string(message.Payload()), 64)
        iot.insertBaseDataInDB(iot.data.VehicleSpeed, iot.data.EngineRPM, iot.data.EngineCoolantTemperature)
        }
    }
}
```

Slika 6.3: Funkcija ReceiveData()

Dobivene varijable nastale ranije spomenutom pretvorbom koriste se kao parametri funkcijama zaduženim za unos tih istih podataka u bazu podataka. Jedna od takvih funkcija je *insertBaseDataInDB()* (slika 6.4.) koja unosi informacije u tablicu *BaseData* baze podataka. Struktura tablica detaljnije je opisana u poglavlju 7.2.

```
func (iot *IOT) insertBaseDataInDB(engineCoolantTemperature float64, engineRPM float64, vehicleSpeed float64) {
    _, err := iot.database.Conn.Exec("INSERT INTO BaseData (eng_coolant_temp, eng_rpm, veh_speed, inserted_at) VALUES ($1, $2, $3, CURRENT_TIMESTAMP)",
    engineCoolantTemperature, engineRPM, vehicleSpeed)
    if err != nil {
        log.Println(err)
    }
}
```

Slika 6.4: Funkcija insertBaseDataInDB()

7. BAZA PODATAKA

Baza podataka je jedan od neophodnih dijelova svakog ozbiljnog IoT sustava pa tako i sustava ovog rada. Korištena je Microsoft SQL baza podataka koja je u vlasništvu Veleučilišta u Bjelovaru, a služi za skladištenje dohvaćenih podataka s MQTT posrednika.

7.1 Microsoft SQL

Microsoft SQL je relacijska baza podataka koja koristi Transact SQL za SQL upite te je razvijena od strane Microsofta. Osim što korisniku nudi niz osnovnih SQL upita, Microsoft SQL također daje mogućnost korištena složenijih stvari kao što su mijenjanje programskog toka (npr. IF naredba). Spomenuti T-SQL je implementacija SQL-91-a s nekim ekstenzijama, a nastao je suradnjom Microsofta i Sysbasea [20].

Jedan od zahtjeva ovoga sustava rada je imati što ispravnije upisane podatke, zbog toga je bilo potrebno korištenje SQL Server Management Studija.

SQL Server Management Studio je Microsoftovo okruženje za upravljanje SQL infrastrukturom koji pruža alate za konfiguriranje, praćenje i upravljanje instancama baza podataka [21]. SSMS je korišten radi potrebe nadziranja upisanih podataka, a primjer istih takvih prikazan je na slici 7.1.

Results | Messages

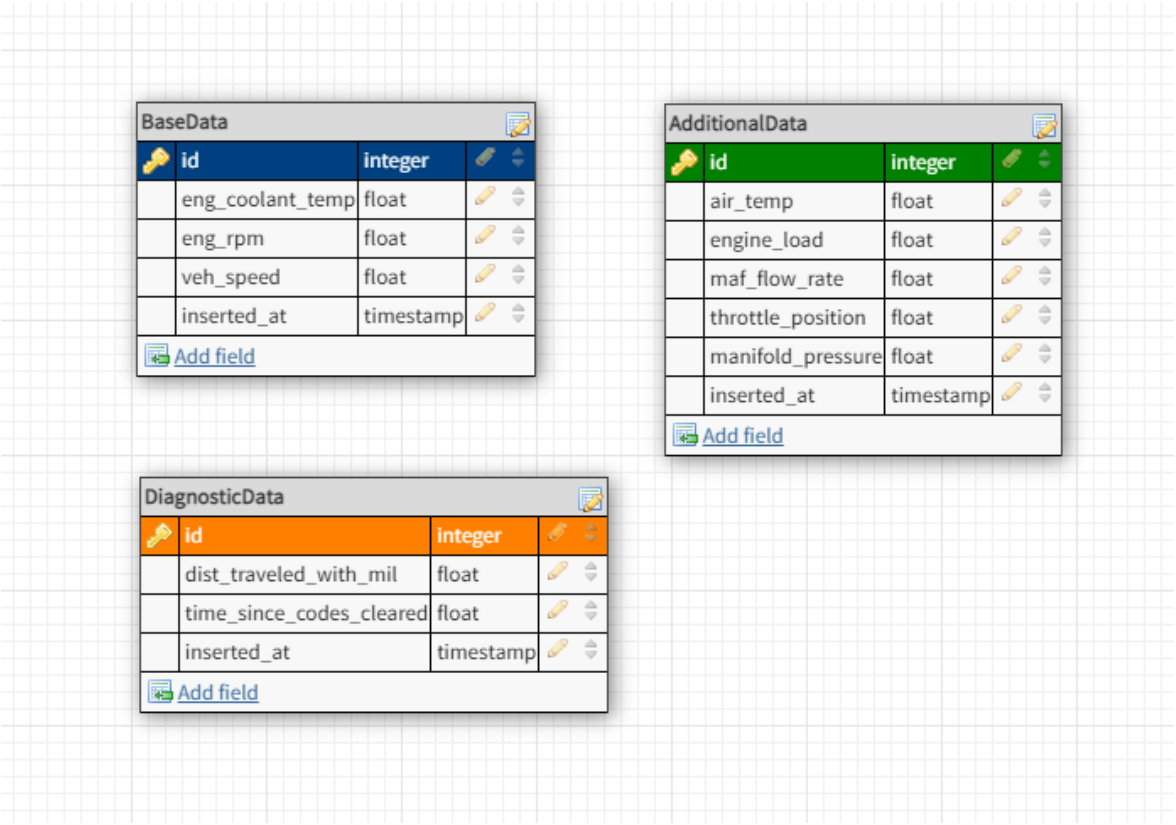
	Id	air_temp	engine_load	maf_flow_rate	throttle_position	manifold_pressure	inserted_at
1	878	29	30,2	32,16	5,81	97	2021-09-20 17:29:17.8700000
2	879	29	30,2	32,16	5,68	97	2021-09-20 17:29:21.5333333
3	880	29	30,2	32,16	5,93	97	2021-09-20 17:29:25.0133333
4	881	29	30,2	32,16	5,87	97	2021-09-20 17:29:28.6000000
5	882	29	30,98	32,16	5,81	97	2021-09-20 17:29:32.2000000
6	883	29	30,2	32,16	8,68	98	2021-09-20 17:29:35.8700000
7	884	29	29,41	32,16	9	98	2021-09-20 17:29:39.7966667
8	885	29	29,8	38,43	9,62	98	2021-09-20 17:29:43.2366667
9	886	29	30,2	38,43	9,75	98	2021-09-20 17:29:47.2500000
10	887	29	30,2	38,43	8,81	98	2021-09-20 17:29:50.9300000
11	888	29	29,8	38,43	9,5	98	2021-09-20 17:29:54.3000000
12	889	29	29,41	38,43	8,75	98	2021-09-20 17:29:58.0066667
13	890	29	29,41	38,43	9,18	98	2021-09-20 17:30:01.5666667
14	891	29	29,02	44,71	9,12	98	2021-09-20 17:30:04.8466667
15	892	29	30,2	44,71	9,81	98	2021-09-20 17:30:08.1400000
16	893	29	29,41	44,71	9,25	98	2021-09-20 17:30:11.7166667
17	894	29	29,41	44,71	9,62	98	2021-09-20 17:30:15.4100000
18	895	29	29,41	44,71	9,37	98	2021-09-20 17:30:18.9933333

Query executed successfully.

Slika 7.1:Prikaz upisanih podataka

7.2 Model baze podataka

Model baze podataka IoT sustava za prikupljanje dijagnostičkih podataka iz automobila izrađen je koristeći web aplikaciju DB Designer [22]. DB Designer omogućava vizualno strukturiranje, odnosno, kreiranje tablica i svih pripadajućih atributa. Za potrebe sustava napravljene su tri tablice podataka *BaseData*, *AdditionalData* i *DiagnosticData*. *BaseData* sadrži osnovne podatke, kao što su npr. brzina vozila i sl., *AdditionalData* proširene, tj. dodatne podatke o automobilu, a *DiagnosticData* sadrži dijagnostičke informacije. Svaka tablica je neovisna jedna o drugoj te sve sadrže jedinstveni broj (stupac *id*) i vrijeme kada su podaci upisani (stupac *inserted_at*). Na temelju tih tablica izrađeni su modeli prikaza na .NET aplikaciji (poglavlje 8.2). Na slici 7.2. prikazani su atributi, stupci i relacije izrađenih tablica.



Slika 7.2: Model baze podataka

8. .NET APLIKACIJA

Aplikacija za prikaz podataka korisniku IoT sustava za prikupljanje dijagnostičkih podataka iz automobila razvijena je putem Microsoftovog .NET-a. .NET aplikacija dohvaća podatke iz baze podataka te ih prikazuje korisniku u obliku grafičkog korisničkog sučelja. Rad aplikacije se temelji na programskim klasama modela podataka i klasama za dohvat podataka iz baze podataka. Te iste klase pisane su koristeći C# programski jezik te su primjenjive u bilo kojoj .NET aplikaciji kao npr. desktop aplikaciji, web aplikaciji i sl..

8.1 .NET

.NET je razvojna platforma (engl. *developer platform*) otvorenog koda, koju je osmislio Microsoft za razvoj različitih aplikacija. Dakle, drugim riječima, .NET je alat za izradu i razvoj C# programskih aplikacija. .NET Framework je izvorna verzija namijenjena za Windows računala, a nova .NET Core verzija za više platformi .NET-a koja se može primijeniti na i na MacOS i Linux operativnom sustavu [23]. .NET aplikacija ovog rada razvijena je uz pomoć razvojnog okruženja Microsoft Visual Studio

8.2 Dohvat podataka

Logika dohvata podataka podijeljena je na dva dijela: klase modela podataka i klase za dohvat podataka. Klase modela podataka temeljene su na strukturi tablica sustava ovog rada u Microsoft SQL bazi podataka, a klase za dohvat podataka zadužene su za spajanje na bazu i dohvat podataka prema definiranim kriterijima. Svaka klasa modela podataka moram imati i pripadajuću klasu dohvata podataka, npr. model podataka *BaseDataModel.cs* strukturiran prema tablici *BaseData* mora imati i klasu koja će prikupljati podatke iz te tablice te ih spremiti u obliku objekta tog modela.

Za potrebe primjera, u sljedećim potpoglavljima opisana je klasa modela *AdditionalDataModel.cs* u potpoglavlju 8.2.1 i njegova usko povezana klasa za dohvat podataka *AdditonalDatas.cs* u potpoglavlju 8.2.2.

8.2.1 Klasa modela podataka

Microsoft SQL baza podataka ovog IoT sustava sadrži tri tablice prema kojima su izrađene klase modela podataka, pojednostavljeno rečeno, .NET aplikacija posjeduje tri klase modela podataka. Na slici 8.1. prikazan je model podataka *AdditionalDataModel.cs*

prema SQL tablici *AdditonalData*.. Prethodno navedeni model sadrži definirane varijable prema stupcima u tablici i potrebne konstruktore kako bi objekt na temelju te klase modela mogao biti instanciran.

```
namespace Dashboard
{
    10 references
    class AdditionalDataModel
    {
        1 reference
        public AdditionalDataModel() { }

        1 reference
        public AdditionalDataModel(object id, object air_temp, object engine_load, object maf_flow_rate,
            object throttle_position, object manifold_pressure, object inserted_at)
        {
            Id = (int)id;
            airTemp = (float)air_temp;
            engineLoad = (float)engine_load;
            mafFlowRate = (float)maf_flow_rate;
            throttlePosition = (float)throttle_position;
            manifoldPressure = (float)manifold_pressure;
            insertedAt = (DateTime)inserted_at;
        }

        2 references
        public int Id { get; set; }
        3 references
        public float airTemp { get; set; }
        3 references
        public float engineLoad { get; set; }
        3 references
        public float mafFlowRate { get; set; }
        3 references
        public float throttlePosition { get; set; }
        3 references
        public float manifoldPressure { get; set; }
        3 references
        public DateTime insertedAt { get; set; }
    }
}
```

Slika 8.1: AdditionalDataModel klasa modela podataka

8.2.2 Klasa za dohvat podataka

Budući da .NET aplikacija sadrži tri klase modela podataka, isto tako mora sadržavati i tri klase za dohvat podataka. Klasa *AdditonalDatas.cs* posjeduje funkcije zadužene za dohvat podataka iz SQL baze. Primjer takve funkcije je *GetLastAdditionalData()* (slika 8.2.) koja dohvaća zadnje upisane podatke iz SQL tablice *AdditionalData*. Funkcija šalje SQL upit te na temelju dohvaćenih podataka vraća objekt tipa *AdditonalDataModel*. Zahvaljujući tom instanciranom objektu ostvaruje se prikaz podataka korisniku.

```
1 reference
public static AdditionalDataModel GetLastAdditionalData()
{
    AdditionalDataModel data = new AdditionalDataModel();
    string query = "SELECT TOP(1) * FROM [AdditonalData] ORDER BY inserted_at DESC";

    using (SqlConnection sqlConnection = new SqlConnection(ConnectDB.ConnectionString()))
    {
        SqlDataAdapter adapter = new SqlDataAdapter(query, sqlConnection);
        DataSet ds = new DataSet();
        adapter.Fill(ds);
        foreach (DataRow row in ds.Tables[0].Rows)
        {
            data.Id = (int)row["Id"];
            data.airTemp = (float)row["air_temp"];
            data.engineLoad = (float)row["engine_load"];
            data.mafFlowRate = (float)row["maf_flow_rate"];
            data.throttlePosition = (float)row["throttle_position"];
            data.manifoldPressure = (float)row["manifold_pressure"];
            data.insertedAt = (DateTime)row["inserted_at"];
        }
    }
    return data;

}
```

Slika 8.2: Prikaz GetLastAdditionalData() funkcije

8.3 Prikaz podataka

8.3.1 Kriteriji za uspješan prikaz

Kako bi prikaz podataka bio uspješan tj. kako bi u konačnici IoT sustav ovog rada ispravno radio, potrebno je osposobiti sve komponente sustava. Dakle, sustav za uspješan rad mora zadovoljiti sljedeće kriterije:

1. ELM327 adapter mora biti spojen na OBD2 priključak vozila
2. ESP32 mikroupravljač treba biti u blizini OBD2 adaptera te imati dostupnu bežičnu internetsku mrežu
3. Vozilo mora biti upaljeno
4. API mora biti pokrenut

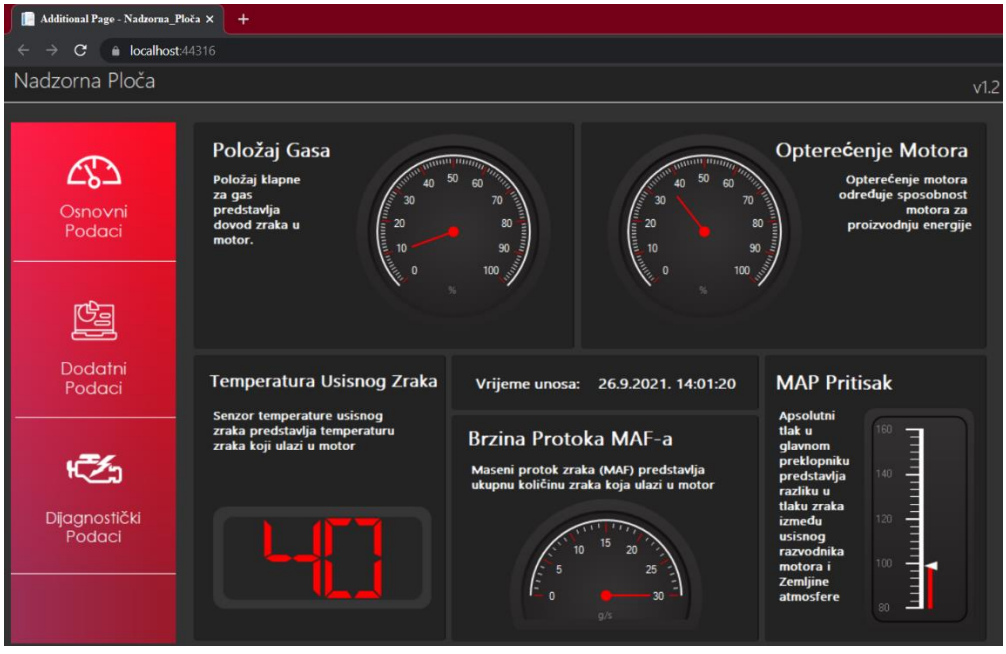
Kada su prethodno navedeni uvjeti zadovoljeni, onda korisnik ovog sustava može vidjeti podatke o svom osobnom vozilu kako se uspješno prikazuju i mijenjaju.

8.3.2 Dizajn aplikacije

Dizajn, odnosno izgled aplikacije prilagođen je prema klasama modela podataka koje su izrađene na temelju strukture SQL tablica. Na slici 8.3. prikazani su osnovni podaci o trenutnom stanju vozila prema *BasicDataModel.cs* klasi modela.

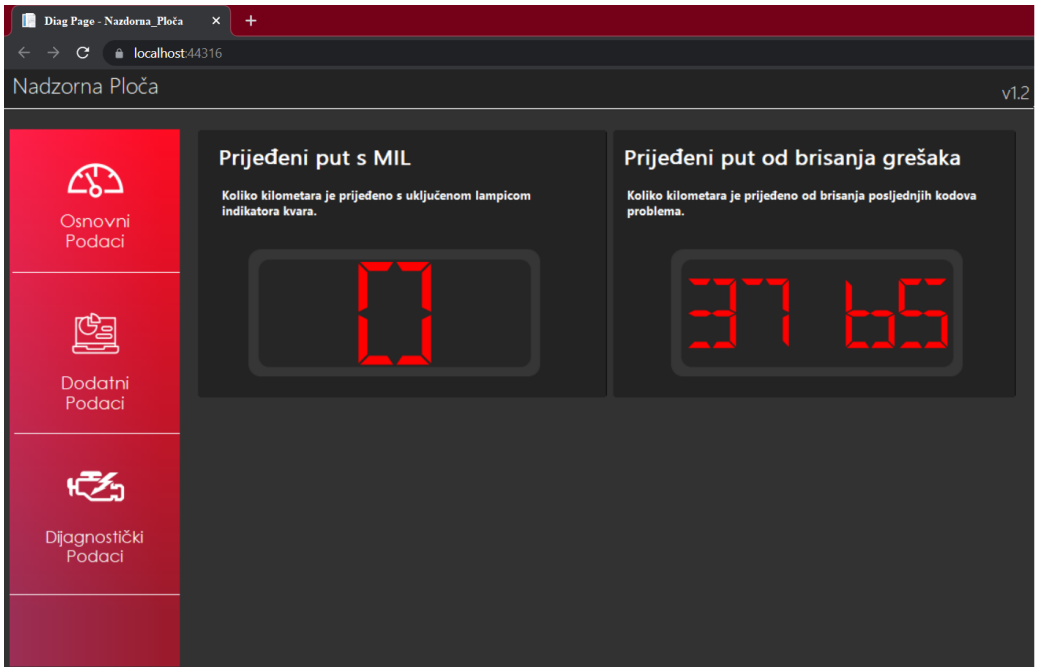
Slika 8.3: Prikaz osnovnih podataka

Slika 8.4. prikazuje dodatne podatke o automobilu, kao npr. položaj gasa, temperatura usisnog zraka i sl., prema *AdditionalDataModel.cs* klasi modela podataka.



Slika 8.4: Prikaz dodatnih podataka

Dijagnostički podaci (slika 8.5.) korisniku daju informaciju o tome koliko je kilometara vozilo prešlo s indikatorom kvara te koliko je kilometara prijeđeno od brisanja posljednjih kodova grešaka. Dijagnostički podaci dizajnirani su prema *DiagnosticDataModel.cs* klasi modela podataka.



Slika 8.5: Prikaz dijagnostičkih podataka

9. ZAKLJUČAK

Tema ovog završnog rada je razvoj sustava interneta stvari koji prikuplja dijagnostičke podatke iz automobila. Zadaci završnog rada su sljedeći: opis i implementacija hardvera zaduženog za dohvaćanje podataka iz osobnog vozila, opis MQTT protokola i njegove pripadajuće arhitekture, opis i izrada baze podataka koja je potrebna kako bi se dohvaćeni podaci pohranili, opis i implementacija aplikacijskog programskog sučelja zaduženog za prikupljanje podataka s MQTT posrednika i upisivanje tih istih podataka u bazu podataka i opis i izrada aplikacije koja obnaša ulogu grafičkog korisničkog sučelja te prikazuje prikupljene podatke korisniku. Za potrebu izrade sustava, korišteni su: ELM327 OBD2 bluetooth adapter, ESP32 mikroupravljač, Microsoft SQL baza podataka, aplikacijsko programsko sučelje, .NET aplikacija i istraživanja vezana uz kolegije Sigurnost Računala i Podataka, Internet Stvari, Baze podataka i .NET.

Svrha rada je izrada sustava interneta stvari uz pomoć kojeg korisnik svog osobnog vozila može imati uvid u podatke o svome automobilu, a to su npr.: brzina vozila, temperatura rashladne tekućine motora, položaj gasa, pojedini dijagnostički podaci i sl.

Tijekom procesa rada na cjelokupnom sustavu završnog rada pojavilo se nekoliko problema. OBD2 bluetooth adapteri poput ELM327 dostupni na hrvatskom tržištu nisu najbolje kvalitete, bluetooth veza često se prekidala te je tehnička dokumentacija vezana uz ELM327 oskudna. Drugi se problem pojavio zbog godine proizvodnje automobila, iako većina europskih automobila od ranih dvijetisućitih posjeduje OBD2 sustav, samo posjedovanje OBD2 priključka nije jamčilo i uspješan dohvat podataka. Preporuke su korištenje kvalitetnijih hardverskih komponenti te primjenu na osobnim vozilima proizvedenim nakon 2008 godine. Uz prethodno navedene uvjete, sustav za prikupljanje dijagnostičkih podataka o osobnom vozilu može postati primjenjiv sustav u području interneta stvari i ideji pametnih automobila, osim toga ovaj sustav se može primijeniti u poslovnom okruženju gdje bi se za veći vozni park mogli automatski prikupljati podaci s dijagnostičkog sučelja i na vrijeme reagirati ako se pojave greške na automobilima kojih vozač nije svjestan.

10. LITERATURA

[1] G. Neuman. “More auto computers means more comlicated, costly and longer repairs“ [Online]. 2016. Dostupno na: https://www.ceinetwork.com/cei-blog/auto-computers-means-complicated-costly-longer-repairs/. (15.08.2021)

[2] OBD-II Resource. “OBD-II PIDs“ [Online]. 2010. Dostupno na: http://obdcon.sourceforge.net/2010/06/obd-ii-pids/. (15.08.2021)

[3] T. Miller. “ELM327: All you need to know before buying one“ [Online]. 2021. Dostupno na: https://obdsolaris.com/elm327-definite-guide/. (15.08.2021)

[4] Wikipedia. Arduino IDE [Online]. 2021. Dostupno na: https://en.wikipedia.org/wiki/Arduino_IDE. (17.08.2021)

[5] Arduino. “ Libraries“ [Online]. 2021. Dostupno na: https://www.arduino.cc/reference/en/libraries/. (17.08.2021)

[6]Wikipedia. ESP32 [Online]. 2021. Dostupno na: https://en.wikipedia.org/wiki/ESP32. (17.08.2021)

[7] ESP32. “ The Internet of things with ESP32“ [Online]. 2021. Dostupno na: http://esp32.net/. (18.08.2021)

[8] ESPHome. “NodeMCU ESP32“ [Online]. 2021. Dostupno na: https://esphome.io/devices/nodemcu_esp32.html. (18.08.2021)

[9] IOTESPRESSO. “How to set partitions in ESP32“ [Online]. 2021. Dostupno na: https://iotespresso.com/how-to-set-partitions-in-esp32/. (23.08.2021)

[10] GitHub [Online]. 2019. Dostupno na: https://github.com/espressif/arduino-esp32/blob/master/libraries/BluetoothSerial/examples/bt_remove_paired_devices/bt_remove_paired_devices.ino. (23.08.2021)

[11] MQTT [Online]. 2020. Dostupno na: https://mqtt.org/. (27.08.2021)

[12] Wikipedia. MQTT [Online]. 2021. Dostupno na: https://en.wikipedia.org/wiki/MQTT. (27.08.2021)

[13] EMQ. “Introduction to MQTT publish-subscribe model“ [Online]. 2020. Dostupno na: https://www.emqx.com/en/blog/mqtt-5-introduction-to-publish-subscribe-model. (27.08.2021)

[14] T. Nordquist. MQTT Explorer [Online]. 2019. Dostupno na: http://mqtt-explorer.com/. (27.08.2021)

[15] Wikipedia. API [Online]. 2021. Dostupno na: https://en.wikipedia.org/wiki/API. (06.09.2021)

[16] M. Wyatt. “What is an API? A digestible definiton with API examples for ecommerce owners“ [Onlin]. 2021. Dostupno na: https://www.bigcommerce.com/blog/what-is-an-api/#so-what-is-json-and-why-is-it-used. (06.09.2021)

[17] GO [Online]. 2021. Dostupno na: https://go.dev/solutions/#case-studies. (09.09.2021)

[18] J. Kincaid: “Google's Go: a new programming language that's python meets c++“ [Online]. 2009. Dostupno na: https://techcrunch.com/2009/11/10/google-go-language/. (09.09.2021)

[19] W. Boyd: “What is Go? An intro to Google's Go programming language (aka Golang)“ [Online]. 2021. Dostupno na: https://acloudguru.com/blog/engineering/what-is-go-an-intro-to-googles-go-programming-language-aka-golang. (09.09.2021)

[20] Wikipedia. Microsoft SQL Server [Online]. 2021. Dostupno na: https://en.wikipedia.org/wiki/Microsoft_SQL_Server. (15.09.2021)

[21] Microsoft. SQL Server Management Studio [Online]. 2021. Dostupno na: https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15. (15.09.2021.)

[22] DB Designer [Online]. 2021. Dostupno na: https://www.dbdesigner.net/. (15.09.2021)

[23] Microsoft. .NET [Online]. 2021. Dostupno na: https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet. (23.09.2021)

11. OZNAKE I KRATICE

ADC – engl. *Analog-digital converter* (analogni digitalni pretvarač)

ASCII – engl. *American Standard Code for Information Interchange* (američki standardni kod za razmjenu informacija)

CAN – engl. *Controller Area Network*

DAC – engl. *Digital-analog converter* (digitalni analogni pretvarač)

GM – engl. *General Motors*

IDE – engl. *Integrated Development Environment* (integrirano razvojno okruženje)

JSON – engl. *JavaScript Object Notation*

KB – engl. *Kilobyte* (kilobajt)

KWP – engl. *Keyword Protocol 2000*

MAC – engl. *Media Access Control* (adresa za kontrolu pristupa medijima)

MB – engl. *Megabyte* (megabajt)

MQTT – engl. *MQ Telemetry Transport* (protokol za prijenos informacija između uređaja)

NET – engl. *Internet/Network* (Internet/internetska mreža)

OASIS – engl. *Organization for the Advancement of Structured Information Standards*

OBD – engl. *On-Board Diagnostics* (ugrađena dijagnostika, dijagnostika na ploči)

PID – engl. *Parameter ID* (parametar ID)

PWM i VPW – engl. *Pulse Width Modulation* (modulacija širine impulsa)

RAM – engl. *Random Access Memory* (memorija s nasumičnim pristupom, izravna memorija)

ROM – engl. *Read Only Memory* (memorija iz koje se podaci mogu samo čitati, unutarnja memorija)

RS-232 – engl. *Recommended Standard 232*

SAE – engl. *Society of Automotive Engineers*

SSMS – engl. *SQL Server Management Studio*

SPI – engl. *Serial Peripheral Interface* (sinkrona serijska komunikacija)

SRAM – engl. *Static Random Access Memory* (statički RAM)

I2C – engl. *Serial Interface* (sinkrona serijska komunikacija)

UART – engl. *Universal Asynchronous Receiver/Transmitter* (asinkrona serijska komunikacija)

12. SAŽETAK

Sustav interneta stvari za prikupljanje dijagnostičkih podataka o osobnom vozilu

U ovom radu opisan je razvoj sustava interneta stvari za prikupljanje dijagnostičkih podataka o osobnom automobila. Prethodno spomenuti sustav zadužen je za prikupljanje, pohranu, dohvat i prikaz podataka što približnije stvarnom vremenu. IoT sustav čini: hardver realiziran pomoću ELM327 OBD2 adaptera i ESP32 mikroupravljač, MQTT posrednik, aplikacijsko programsko sučelje, razvijeno pomoću GO programskog jezika, koje dohvaća podatke s MQTT servera i te iste podatke pohranjuje u Microsoft SQL bazu podataka te .NET aplikacija koja prikazuje prikupljene podatke korisniku sustava rada. Rad se temelji na dijagnostičkom OBD2 sustavu osobnih vozila u kombinaciji s tehnologijom Interneta stvari, skladištenjem podataka i predstavljanjem dohvaćenih podataka korisniku.

Ključne riječi: OBD2, sustav, IoT, MQTT, podaci, baza podataka, GO, osobno vozilo, .NET.

13. ABSTRACT

Internet of Things system for collecting diagnostic dana on a personal vehicle

This paper describes the development of an Internet of Things system for collecting diagnostic data on a personal car. The aforementioned system i sin charge of collecting, storing, retrieving and displaying dana as close as possible to real time. The IoT system consists of: hardware implemented using ELM327 OBD2 adapter and ESP32 microcontroller, MQTT broker, application programming interface, developed using GO programming language, which retrieves data from MQTT server and stores the same data in Microsoft SQL database and .NET application that displays collected data to the user of the operating system. The work is based on the OBD2 diagnostic system of personal vehicles in combination with Internet of Things technology, data storage and presentation of retrieved data to the user.

Keywords: OBD2, system, IoT, MQTT, data, database, GO, personal vehicle, .NET.

IZJAVA O AUTORSTVU ZAVRŠNOG RADA

Pod punom odgovornošću izjavljujem da sam ovaj rad izradio/la samostalno, poštujući načela akademske čestitosti, pravila struke te pravila i norme standardnog hrvatskog jezika. Rad je moje autorsko djelo i svi su preuzeti citati i parafraze u njemu primjereno označeni.

Mjesto i datum	Ime i prezime studenta/ice	Potpis studenta/ice
U Bjelovaru, 4. listopad 2021	Tony Mostovac	Tony Mostovac

Prema Odluci Veleučilišta u Bjelovaru, a u skladu sa Zakonom o znanstvenoj djelatnosti i visokom obrazovanju, elektroničke inačice završnih radova studenata Veleučilišta u Bjelovaru bit će pohranjene i javno dostupne u internetskoj bazi Nacionalne i sveučilišne knjižnice u Zagrebu. Ukoliko ste suglasni da tekst Vašeg završnog rada u cijelosti bude javno objavljen, molimo Vas da to potvrdite potpisom.

Suglasnost za objavljivanje elektroničke inačice završnog rada u javno dostupnom nacionalnom repozitoriju

Tony Mostovac

ime i prezime studenta/ice

Dajem suglasnost da se radi promicanja otvorenog i slobodnog pristupa znanju i informacijama cjeloviti tekst mojeg završnog rada pohrani u repozitorij Nacionalne i sveučilišne knjižnice u Zagrebu i time učini javno dostupnim.

Svojim potpisom potvrđujem istovjetnost tiskane i elektroničke inačice završnog rada.

U Bjelovaru, 4. Listopad 2021

Tony Mostovac

potpis studenta/ice