Implementacija IoT rješenja u postojeće sustave upravljanja mikrokontrolerima

Bajec, Dominik

Sveučilište
Sjever

Završni rad br. 463/EL/2020

Implementacija IoT rješenja u postojeće sustave upravljanja mikrokontrolerima

Dominik Bajec, 2070/336

Varaždin, lipanj 2020. godine

Sveučilište
Sjever

Odjel za Elektrotehinuku

Završni rad br. 463/EL/2020

Implementacija IoT rješenja u postojeće sustave upravljanja mikrokontrolerima

Student

Dominik Bajec, 2070/336

Mentor

Josip Srpak, dipl. ing el., predavač

Varaždin, lipanj 2020. godine

Sveučilište Sjever
Sveučilišni centar Varaždin
104. brigade 3, HR–42000 Varaždin

Prijava završnog rada

Definiranje teme završnog rada i povjerenstva

ODJEL Odjel za elektrotehniku

STUDIJ preddiplomski stručni studij Elektrotehnika

PRISTUPNIK Dominik Bajec | MATIČNI BROJ 2070/336

DATUM 15.06.2020 | KOLEGIJ PLC sustavi upravljanja

NASLOV RADA Implementacija IoT rješenja u postojeće sustave upravljanja mikrokontrolerima

NASLOV RADA NA ENGL. JEZIKU Implementation of IoT solutions in existing microcontrol systems

MENTOR Josip Srpak dipl.ing.el. | ZVANJE Predavač

ČLANOVI POVJERENSTVA

1. mr.sc. Ivan Šumiga dipl.ing.el., viši predavač
2. doc.dr.sc. Dunja Srpak dipl.ing.el.
3. Josip Srpak dipl.ing.el., predavač
4. Miroslav Horvatić dipl.ing.el., predavač, rezervni član
5.

Zadatak završnog rada

BROJ 463/EL/2020

OPIS

U završnom radu je potrebno opisati implementaciju IoT rješenja u postojeće sustave upravljanja sa mikrokontrolerima. Zatim realizirati primjer sustava bez mogućnosti IoT povezivanja korištenjem Arduino razvojne platforme, te sustav za povezivanje postojećeg sustava na IoT korištenjem esp8266 modula. Prikupljanje podataka senzora i upravljanje izlazima vrši postojeći Arduino sustav. Pomoću esp8266 izvršiti svu obradu podataka i primati/slati podatke prema postojećem Arduino sustavu koristeći serijsku komunikaciju. Kako bi korisnik imao pristup podacima i upravljanju, potrebno je implementirati MQTT protokol na esp8266 modul, te izraditi Android MQTT mobilnu aplikaciju za nadgledanje i upravljanje sustavom.
U radu je potrebno:
- opisati IoT rješenja i MQTT protokol,
- opisati problematiku i implementaciju sustava, te osnovne elemente i sklopovlje sustava,
- izraditi sustav na Arduino razvojnoj platformi i opisati programska rješenja,
- izraditi sustav za povezivanje IoT implementacijom MQTT protokola koristeći esp8266 modul,
- izraditi Android mobilnu aplikaciju za nadgledanje i upravljanje sustavom,
- komentirati i analizirati dobivene rezultate.

ZADATAK URUČEN 17.06.2020. | POTPIS MENTORA Josip Srpak

SVEUČILIŠTE SJEVER

Predgovor

„Internet stvari“ (eng. *Internet of Things*) u današnje vrijeme implementira se u sve više sustava i implementacija je poprimila eksponencijalni rast. Kako bi današnji sustavi komunicirali na daljinu, međusobno ili sa korisnicima, trebaju postati „pametni“. Da bi sustavi postali „pametni“ i dio „Internet stvari“ moraju imati pristup internetu i neku vrstu međusobne komunikacije. U današnje vrijeme takvi sustavi ostvaruju se korištenjem mikrokontrolera koji imaju pristup internetu i mogućnost upravljanja ulazima i izlazima. Većinom sustavi bazirani na mikrokontrolerima nemaju na raspolaganju veliku količinu resursa ako se usporede primjerice s osobnim računalima današnjeg doba. Iz navedenog razloga takvi sustavi, da bi postali dio „Internet stvari“, moraju imati „laganu“ komunikaciju putem interneta. Komunikacija koja je izvorno bila zamišljena da bude „lagana“ kako ne bi zahtijevala mnogo resursa zasniva se upravo na MQTT (eng. Message Queue Telemetry *Transport*) protokolu te je danas u sve većoj primjeni. MQTT je „lagan“ protokol osmišljen za komunikaciju između strojeva (engl. *Machine-to-machine, M2M*).

Sažetak

U završnom radu opisano je kako se IoT (eng. *Internet of Things*) razvio kroz povijest, zašto je važna njegova primjena te koliku količinu podataka generira na internetu. Opisane su osnovne informacije zašto je je IoT potreban u industriji ali i kod osobne upotrebe korisnika te problemi koji nastaju zbog same sigurnosti. Kratko se osvrće i komentira utjecaj nove pete generacije mobilne mreže (5G) na IoT sustave. Na kraju IoT poglavlja opisane su prednosti i nedostaci primjene. U nastavku se govori o MQTT protokolu (eng. *Message Queue Telemetry Transport*) koji se kasnije praktično implementira, te su objašnjene njegove funkcije i princip rada. Detaljnije se osvrće na značenje klijent, broker, način na koji se povezuje s brokerom i održava veza, kvalitete usluge (QoS) te na ostale bitnije parametre MQTT protokola.

Uz praktični dio rada opisan je Arduino sustav koji predstavlja postojeći stroj/uređaj na kojeg se nadograđuje Esp8266 modul kako bi postao dio IoT svijeta. U nastavku rada detaljno su opisani korišteni senzori te njihove specifikacije kao i dio programskog koda za njihovo upravljanje. Korišteni su senzori temperature, vlage, tlaka kao i ventilator za demonstraciju upravljanja i nadzor. Opisana je implementacija MQTT protokola na Esp8266 modul i primjena UART komunikacije koja je temelj razmjene podataka između Arduino razvojne platforme i samog Esp8266 mikrokontrolera. Uz praktičan dio završnog rada korištenjem *Paho* biblioteke izrađena je i Android MQTT mobilna aplikacija za nadzor sustava i njegovo upravljanje te su dijelovi aplikacije dodatno objašnjeni.

Ključne riječi: IoT – Internet stvari, MQTT protokol, M2M- komunikacija između strojeva, MQTT klijent i broker, QoS – kvaliteta usluge, UART – serijska komunikacija, Arduino, Esp8266, Android

Summary

The final paper describes how the IoT (Internet of Things) has evolved throughout history, why its application is important and how much data it generates on the Internet. Basic information on why IoT is needed in industry, but also in the personal use of users, and the problems that arise due to security itself are described. The impact of the new fifth generation mobile network (5G) on IoT systems is briefly reviewed and commented on. The advantages and disadvantages of the application are described at the end of the IoT chapter. The MQTT protocol (Message Queue Telemetry Transport), which is later practically implemented, is discussed below, and its functions and working principle are explained. It looks in more detail at the meaning of the client, the broker, the way it connects with the broker and maintains the connection, the quality of service (QoS) and other important parameters of the MQTT protocol.

In addition to the practical part of the paper, the Arduino system is described, which is an existing machine / device to which the Esp8266 module is upgraded to become part of the IoT world. The following describes in detail the sensors used and their specifications, as well as part of the program code for their control. Temperature, humidity, pressure sensors as well as a fan were used to demonstrate control and monitoring. The implementation of the MQTT protocol on the Esp8266 module and the application of UART communication, which is the basis of data exchange between the Arduino development platform and the Esp8266 microcontroller itself, are described. In addition to the practical part of the final work, using the Paho library, an Android MQTT mobile application for system monitoring and management was created, and parts of the application were further explained.

Keywords: IoT - Internet of Things, MQTT protocol, M2M- communication between machines, MQTT client and broker, QoS - quality of service, UART - serial communication, Arduino, Esp8266, Android

Popis korištenih kratica

IoT	Internet of Things - Internet stvari
SCADA	Supervisory Control And Data Acquisition - Računalni sustav za nadzor, mjerenje i upravljanje industrijskim sustavima
M2M	Machine to machine – komunikacija između strojeva
IIOT	Industrial Internet of Things - Industrijski internet stvari
RFID	Radio-frequency identification – tehnologija koja koristi radio frekvenciju za razmjenu podataka
MQTT	Message Queue Telemetry Transport - komunikacijski protokol
TLS/SSL	Transport Layer Security/Secure Sockets Layer - skupina protokola za sigurnu komunikaciju
TCP/IP	Transmission Control Protocol/Internet Protocol - standardni protokoli internet veza
HTTP	Hyper Text Transfer Protocol – protokol za komunikaciju klijent-server
QoS	Quality of Service - kvaliteta usluge
UART	Universal Asynchronous Receiver/Transmitter - asinkrona serijska komunikacija

Sadržaj

Uvod

U završnom radu bilo je potrebno osmisliti ideju povezivanja postojećeg sustava upravljanog mikrokontrolerom sa IoT svijetom koji je danas sve popularniji. U svijetu i oko nas još uvijek postoji mnogo sustava koji nisu u početku nastajanja bili zamišljeni kao IoT ili su izrađeni još u vrijeme kada IoT nije postojao. Mikrokontroleri koji su korišteni vjerojatno nemaju mogućnost pristupa internetu jer za to nije bilo potrebe ili su imali premalo resursa za dodatno procesiranje internet protokola.

Na primjeru tvrtke koja se bavi proizvodnjom kompleksnog stroja za kojeg žele da postane dio IoT svijeta ili je prikupljanje informacija sa stroja ili njegovo upravljanje jako bitno. Jedna od mogućnosti je izrada potpuno nove elektronike korištenjem mikrokontrolera koji ima mogućnost povezivanja s internetom ili nadogradnja takve elektronike vanjskim modulom. Implementacija vanjskog modula na postojeću elektroniku zvuči jednostavnije i možda financijski isplativije ali svakako i ta opcija nosi svoje probleme. Vanjski modul, odnosno elektroniku, potrebno je povezati sa postojećom te ostvariti komunikaciju među mikrokontrolerima. Potrebno je prilagoditi program mikrokontrolera na postojećoj elektronici stroja kako bi slao i primao podatke. Kako bi se ostvarila komunikacija između postojeće elektronike i nadograđenog modula, mikrokontroleri trebaju podržavati neku vrstu komunikacije (primjerice serijska komunikacija UART), a elektronika zahtjeva mogućnost hardverskog spajanja.

Stroj koji tvrtka proizvodi možda već ima ostvareno softversko i hardversko rješenje za serijsku komunikaciju no koristi se primjerice za povezivanje sa računalom koje prati parametre ili šalje parametre prema stroju. Takvo računalo je vrlo vjerojatno resursima presnažno i financijski neisplativo, a povezano je kablom na serijski port za komunikaciju. Računalo se može zamijeniti vanjskim modulom koji ima mogućnost povezivanja na internet, a korisnik u tom slučaju može pristupiti stroju iz bilo kojeg dijela svijeta putem aplikacije. Ideja vanjskog modula za povezivanje stroja na IoT može proširiti ponudu na tržištu i tvrtki stvoriti nove prihode. Modul može postati dodatna opcija koja se kupcu nudi uz stroj, a samim time prate se industrijski trendovi što stroj i dalje čini konkurentnim na tržištu.

Kako bi se prikazalo koliko takva proširenja mogu biti efikasna, u završnom radu izrađen je primjer postojećeg sustava/stroja koji nema mogućnost povezivanja na internet kako bi postao dio IoT svijeta. Postojeći sustav realiziran je na razvojnoj platformi Arduino, a sastoji se od senzora za prikupljanje podataka o temperaturi i vlazi (HTU21D), tlaku zraka (BMP180) te ventilatoru koji služi za simuliranje ulazno/izlaznog modula kojim se može upravljati. Ujedno se mogu nadgledati i promjene koje se dešavaju uzrokovane upravljanjem.

Za vanjsko proširenje, odnosno modul, korišten je esp8266 koji sve podatke prima i šalje na Arduino pomoću serijskog porta (UART). Kako esp8266 raspolaže s dovoljnom količinom resursa, obavlja sve dodatne obrade podataka koje prima kako bi se Arduino (postojeći sustav) što manje opteretio, prikazuje ih na ekranu i izvršava sve zadatke vezane uz održavanje internetske konekcije. Za upravljanje putem interneta, implementiran je MQTT protokol na sam esp8266 modul te je razvijena Android aplikacija za nadzor i upravljanje samim sustavom.

1. IoT – „Internet stvar" (Internet of Things)

IoT (eng. *Internet of Things, IoT*) naziv je za milijarde fizičkih uređaja u čitavom svijetu koji su povezani s internetom, a svi prikupljaju i dijele podatke. Zahvaljujući dolasku jeftinih mikrokontrolera i sveprisutnosti bežičnih mreža, moguće je pretvoriti bilo što u IoT.

Povezivanje različitih objekata i dodavanjem senzora dolazi do digitalne inteligencije uređaja, omogućujući im komuniciranje i razmjenu podataka u stvarnom vremenu bez interakcije čovjeka, iako ljudi mogu komunicirati i upravljati uređajem. IoT čini svijet oko nas pametnijim i odgovornijim, spajajući digitalne i fizičke objekte, također može koristiti umjetnu inteligenciju i strojno učenje kako bi prikupljanje podataka bilo lakše i dinamičnije.

IoT uređaj može biti i obična žarulja u kući ako se može uključiti pomoću aplikacije za pametni telefon ili termostat kojim se pomoću neke aplikacije upravlja svakim danom i podešava željena temperaturu stana. Ovakav sustav može se primijeniti na različitim stvarima, primjerice od dječjih igračaka do autonomnih automobila. Neki veći objekti mogu biti prepuni malih IoT pametnih sustava, kao što je mlazni motor aviona koji se sastoji od tisuća senzora za prikupljanje i slanje podataka kako bi se rad motora mogao analizirati. U još većem obimu, projekti pametnih gradova preplavljuju čitave regije senzorima kako bi pomogli kontrolirati i nadgledati okolinu.

Izraz IoT uglavnom se koristi za uređaje za koje se obično ne očekuje da imaju internetsku vezu i koji imaju mogućnost komunikacije s mrežom neovisno o ljudskom djelovanju. Iz tog razloga osobno računalo se ne smatra IoT uređajem kao ni pametni telefon iako je prepun različitih senzora.

IoT je i proširenje za SCADA (eng. *supervisory control and data acquisition*) sustave, kategorija softverskih aplikativnih programa za kontrolu procesa, prikupljanje podataka u stvarnom vremenu s udaljenih lokacija za kontrolu opreme i nadzor. SCADA sustavi uključuju hardverske i softverske komponente. Hardver prikuplja i unosi podatke u računalo s instaliranim SCADA softverom, gdje se zatim pravodobno obrađuje i prezentira. Razvoj SCADA-e je takav da su se kasne generacije SCADA sustava razvile u prve generacije IoT sustava. [1] [2]

1.1. Povijest

Kevin Ashton izrekao je frazu "Internet stvari" 1999. godine, iako je bilo potrebno najmanje još jedno desetljeće kako bi tehnologija dostigla viziju.

Ideja o implementiranju senzora i inteligentnog sustava u osnovne predmete raspravljala se tijekom 1980-ih i 1990-ih (postoje i neke mnogo ranije informacije o pokušajima implementacije). Usprkos nekim ranijim projektima, za napredak je ipak bilo potrebno mnogo vremena jer tehnologija nije bila u korak s idejama. Elektroničke komponente bile su prevelike i glomazne te nije bilo načina da uređaji učinkovito komuniciraju.

Bili su potrebni procesori koji su jeftini i dovoljno štedljivi da bi zadovoljili uvjete za upotrebu i povezivanje milijardi uređaja. Pojavom RFID oznaka, radi se zapravo o čipovima male snage koji imaju mogućnost bežične komunikacije, riješio se jedan mali dio ovog problema. Razvoju je doprinijela sve veća dostupnost širokopojasnog interneta i lokalno bežično umrežavanje. Implementacija IPv6 protokola koji bi, između ostalog, trebao osigurati dovoljno IP adresa za svaki uređaj koji se povezuje na internet, bio je također nužan korak za široku upotrebu IoT-a.

Prvi IoT uređaj bio je Coca-Cola automat na Sveučilištu Carnegie Mellon u ranim 80-ima. Pomoću interneta moglo se nadgledati stanje automata, odnosno ima li u njemu pića i koja je trenutna temperatura hlađenja. [1] [3]

1.2. Zašto je važan

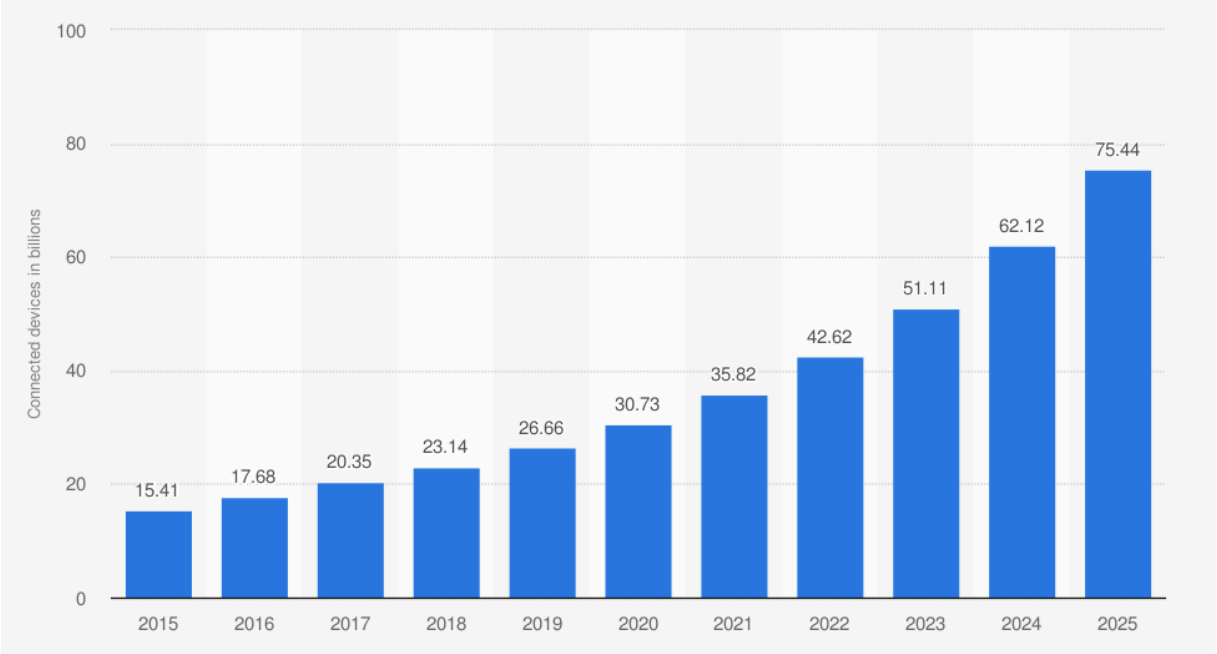
IoT omogućava ljudima pametniji život i rad, a ujedno je cilj i stvoriti u potpunosti kontrolu nad životom. S obzirom što nudi pametne sustave i uređaje za automatiziranje domova, IoT ima veliku ulogu za posao. IoT ostvaruje mogućnost tvrtkama i privatnim osobama da u realnom vremenu nadgledaju kako njihovi sustavi rade, pružajući uvid u sve podatke koje IoT uređaji skupljaju, od performansi uređaja do opskrbnih lanaca i logističkih operacija. Omogućuje tvrtkama automatiziranje procesa i smanjivanje troškova rada. Ujedno smanjuje stvaranje otpada i poboljšava usluge, a na taj način proizvod i isporuka robe pojeftinjuju.

Kao takav, IoT je jedna od najvažnijih tehnologija svakodnevnog života i njegova upotreba nastavit će rasti. Sve više tvrtki shvaća potencijal povezanih uređaja što ih održava konkurentnima. [1]

1.3. Količina uređaja i količina podataka

Iako je jasno da je IoT vrlo popularan, nema puno valjanih informacija koje bi točno prikazale kako se današnje tržište ponaša u ovom trenutku. U svakom slučaju sigurno se može reći da na svijetu postoji više IoT uređaja nego ljudi.

Statista koja se bavi prikupljanjem nacionalne i međunarodne statistike i podataka, navodi da će do 2025. godine biti ukupno 75,44 milijardi IoT uređaja. Nagađa se da industrijska i automobilska oprema predstavljaju najveću količinu povezanih uređaja, ali se pretpostavlja i snažan rast popularnosti pametnih kuća u skoroj budućnosti. [4]

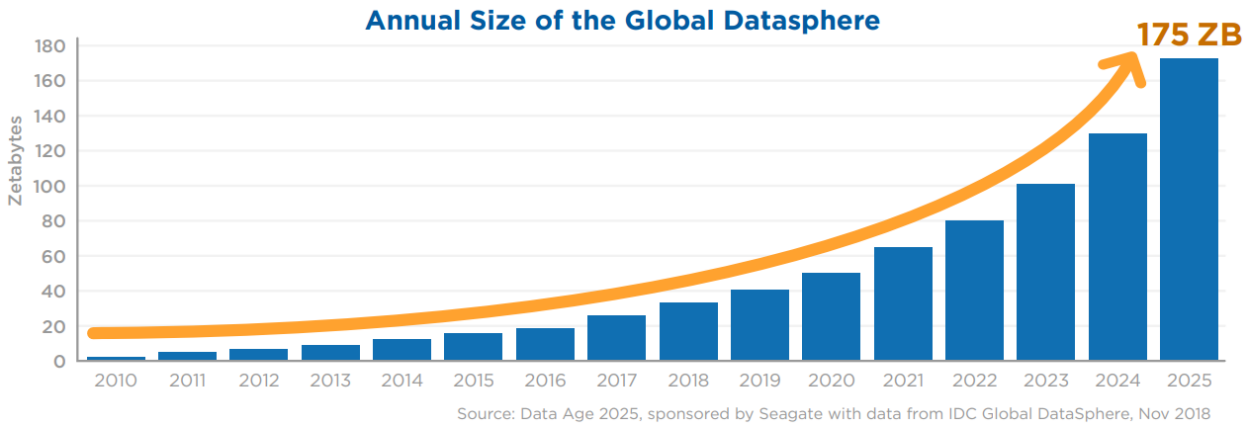


Slika 1.1: Porast količine IoT uređaja u milijardama do 2025. godine [4]

Statistički podaci:

- Na Internet se svake sekunde spaja 127 novih IoT uređaja
- U 2019. godini bilo je 26,66 milijardi aktivnih IoT uređaja
- Broj IoT uređaja za 2020. godinu iznosi 31 milijardu
- Predviđa se da će do 2025. godine biti ukupno 75,44 milijardi IoT uređaja

IoT uređaj vjerojatno će sadržavati jedan ili više senzora koje će koristiti za prikupljanje podataka. Što će ti senzori prikupiti ovisit će o pojedinačnom uređaju i njegovoj zadaći. Senzori unutar industrijskih strojeva mogu mjeriti temperaturu ili tlak. Sigurnosna kamera može imati senzor blizine zajedno sa zvukom i video zapisom, dok će kućna meteorološka stanica vjerojatno sadržavati senzor vlage. Sve ove podatke senzora, i još puno više, potrebno je negdje poslati. To znači da će IoT uređaji trebati prenositi podatke i to će činiti putem Wi-Fi-a, 4G, 5G mreže. Tehnička analitičarska tvrtka IDC (International Data Corporation) izračunava da će do 2025. godine IoT uređaji stvoriti 175 zetabajta podataka. [5]



Slika 1.2: Porast količine podataka do 2025. godine *[5]*

Neki od tih podataka stvarat će brzo ažuriranje poput očitanja temperature sa senzora. Drugi bi uređaji mogli stvoriti ogromne količine podatkovnog prometa, poput nadzorne kamere. Također navodi da će količina podataka stvorenih putem IoT uređaja brzo rasti u sljedećih nekoliko godina. Većina podataka generirat će se i dalje video nadzorom, ali drugi industrijski i medicinski uređaji s vremenom će generirati više podataka. [5]

1.4. Velika količina podataka (Big data)

„Velika količina podataka“ (eng. *Big data*) označava veliki skup (veličine gigabajta, petabajta ili zetabajta) strukturiranih, nestrukturiranih ili polustrukturiranih podataka nad kojima se vrši analiza kako bi se dobio uvid u korisne informacije.

Predviđanja govore da će do 2025. godine biti generirano 175 zetabajta podataka putem IoT sustava, a barem jedan dio tih podataka prikuplja se upravo iz razloga da se obrade i kasnije analiziraju. Dakle, kao što je već zaključeno, IoT uređaji ne bi imali smisla i stekli ovoliku globalnu popularnost da ne prikupljaju razne podatke čime olakšavaju i pojeftinjuju način života.

IoT u raznim organizacijama i tvrtkama predstavlja glavni alat za prikupljanje podataka za analizu u realnom vremenu. Velika količina podataka koje je generirao IoT predstavlja glavni ključ za analiziranje sustava što pomaže u poboljšavanju proizvoda i donošenju preventivnih radnji i odluka. [5]

Velika količina podataka prikupljena zahvaljujući primjenom IoT sustava temelji se na četiri uzastopna koraka: [6]

- Prikupljanje velike količine podataka gdje glavnu ulogu ima IoT uređaj
- U velikom sustavu podataka koji je u osnovi zajednička distribuirana baza podataka, ogromna količina podataka pohranjuje se u velike datoteke
- Analiziranje pohranjenih velikih podataka pomoću analitičkih alata poput Hadoop MapReduce ili Spark
- Izrada izvještaja o analiziranim podacima

1.5. Industrija

„Industrijske Internet stvari“ (eng. *Industrial Internet of Things, IIoT*) ili, kako se točnije naziva, četvrta industrijska revolucija ili skraćenog naziva industrija 4.0 sva su imena koja se koriste za IoT tehnologije u poslovnom okruženju. Koncept je isti kao kod potrošačkih IoT uređaja u kući, ali u ovom je slučaju cilj koristiti kombinaciju senzora, bežične mreže, velikih podataka, umjetne inteligencije i analitike za mjerenje i optimizaciju industrijskih procesa.

Ako se uvede u čitav lanac opskrbe, a ne samo na pojedine dijelove, učinak bi mogao biti još veći prateći proizvodnju od početka do kraja. Povećavanje produktivnosti rada ili financijska ušteda dva su potencijalna cilja, ali IoT također može potaknuti nove izvore prihoda za tvrtku. Umjesto da prodaju samostalan proizvod, na primjer motor, proizvođači mogu prodavati i sustav za predviđanje kvarova i održavanja motora.

Proizvođači danas sve češće dodaju senzore komponentama svojih proizvoda kako bi mogli pratiti podatke o njihovim performansama. To može pomoći tvrtkama da uoče kada neku komponentu treba zamijeniti prije nego što nanese štetu. Tvrtke također mogu koristiti podatke koje generiraju senzori kako bi učinili svoje sustave i lance opskrbe učinkovitijima, jer će imati puno točnije podatke o onome što se stvarno događa. [1] [2]

1.6. Osobna upotreba

IoT čini naše okruženje, naše domove, urede i vozila, pametnijim. Bolje razumijevanje načina funkcioniranja naših domova i mogućnost podešavanja postavki mogu pomoći uštedi energije, na primjer, smanjujući troškove grijanja ili računa za struju. Kućni sigurnosni sustavi olakšavaju nadgledanje onoga što se događa unutar i izvan kuće, na dvorištu ili u garaži. U međuvremenu, pametni sustavi grijanja mogu pomoći podesiti temperaturu domova prije nego što se korisnik vrati, a pametna rasvjeta se može sama isključiti u slučaju kad korisnik nije kod kuće kako bi se uštedjela električna energija i smanjili troškovi. Za potrošače, pametni dom je vjerojatno mjesto gdje će najprije doći u kontakt s IoT sustavima, odnosno uređajima. To je zapravo jedno od najvećeg IoT područja u kojem se velike konkurentne kompanije (posebno Amazon i Google) teško natječu.

Gledajući izvan uređaja pametne kuće, senzori mogu pomoći u praćenju buke ili koliko je zagađena naša okolina. Autonomni automobili i pametni gradovi u potpunosti bi mogli promijeniti način na koji se gradi i upravlja okolinom koja nas okružuje. [1]

1.7. Sigurnost korisnika

IoT spaja digitalni i fizički svijet, što potencijalno znači da hakiranje može imati razne opasne posljedice na korisnike IoT uređaja. Hakiranje sustava koji kontroliraju proces u industriji moglo bi dovesti do katastrofe, preuzimanje kontrole nad zrakoplovima ili autonomnim automobilima moglo bi također završiti katastrofom. Privatni korisnici mogu biti meta hakerskih napada, povezujući razne IoT uređaje koji zahtijevaju ili prikupljaju osobne podatke. Takvi podaci su atraktivna meta za ljude koji žele profitirati od osobnih podataka drugih korisnika. Svaki dodatni uređaj povezan na internet stvara prijetnju za zaštitu osobne sigurnosti, pogotovo uređaji koji se povezuju na pametne telefone ili računala u kojima su većinom pohranjeni svi naši podaci.

Nedostaci u softverima IoT sustava otkrivaju se svakodnevno, ali mnogi IoT uređaji nemaju mogućnost ažuriranja softvera što znači da su u stalnoj opasnosti. Najveća meta hakerskih napada su nadzorne kamere povezane na internet iz razloga što takvi sustavi pružaju napadaču kontrolu i pregled situacije u okolini. Sigurnost korisnika toliko je važna stvar da su pokrenuta istraživanja vezana za IoT te se otkriva da je i do 100 000 vrsta kamera povezanih na internet laka meta hakerskih napada. Rizici koji se također javljaju su napadi na industrije koje raspolažu IoT sustavima integriranih u proizvodnju ili vođenje statističkih podataka financija, a glavni im je cilj industrijska špijunaža. Takve industrije moraju osigurati sigurnost svojih mreža, a potrebno je i kriptiranje podataka kako bi se onemogućila krađa ili eventualna manipulacija podacima.

Današnji IoT uređaji bore se sa pružanjem potpune sigurnosti sustava zbog ubrzanog rasta istih na tržištu. Priča o sigurnosti korisnika, industrije ili preuzimanjem kontrole nad autonomnim automobilima vrlo je zabrinjavajuća ako su takvi sustavi ostali nedovoljno zaštićeni s obzirom na količinu hakerskih napada koji se događaju svugdje u svijetu. [1]

Neke minimalne radnje za pružanje sigurnosti korisniku i sprječavanje hakerskih napada su:

- Kontinuirano obavještavanje korisnika o uređajima koji imaju zastarjele verzije softvera
- Poboljšanje pametnog upravljanja lozinkom (npr. obavezne promjene zadane lozinke).
- Onemogućavanje udaljenog pristupa uređaju, osim ako nije potrebno za osnovne funkcije.
- Implementacija zaštite podataka njihovim kriptiranjem

1.8. Utjecaj pojave 5G mreže na IoT

Prva generacija mobilnih mreža (1G) bila je fokusirana na prijenosu telefonskih poziva, druga generacija (2G) bila je fokusirana na prijenosu telefonskih poziva i sms poruka, treća generacija (3G) nadovezala se na drugu generaciju (2G) uz dodatak mogućnosti prijenosa podataka. U današnje doba dobro nam je poznata četvrta generacija mobilnih mreža (4G) koja je fokusirana na sve što sadržava treća generacija (3G) samo na puno većim brzinama prijenosa. Danas se sve više govori o novoj, odnosno petoj generaciji mobilnih mreža (5G). Peta generacija mobilnih mreža fokusira se na povećanje brzina prijenosa podataka, smanjenje kašnjenja prijenosa pa čak i na IoT.

Spomenuto je da peta generacija mobilne mreže (5G) stavlja fokus na povećanje brzine što u teoriji znači da će brzina 5G mreže iznositi 10-50Gbps. Brzina 4G mreže iznosila je u teoriji 300Mbps, dok 3G mreža 42Mbps. [7]

Generacija mreže	Prosječna brzina preuzimanja	Maksimalna brzina preuzimanja	Teorijska brzina preuzimanja
3G	7.9Mbps	20Mbps	42Mbps
4G	35.9Mbps	90Mbps	300Mbps
5G	130-240Mbps	600Mbps	10-50Gbps

Tablica 1.1: Prikaz brzina mreža po generacijama

Peta generacija mobilnih mreža (5G) zapravo je nešto više od velikih brzina preuzimanja. Kombinacija velike brzine povezivanja i razmjene podataka, vrlo malo kašnjenje i široka pokrivenost omogućava autonomnost prometne infrastrukture. Takve specifikacije mreže pružaju mogućnost povezivanja automobila, autobusa i kamiona pri čemu bi kašnjenje komunikacije bilo svedeno na minimum. Nova generacija mreže omogućit će upravljanje na daljinu uređajima čije je upravljanje kritično vezano uz performanse mreže u stvarnom vremenu, poput upravljanja strojevima u okolini koja može naštetiti zdravlju čovjeka. U IoT svijetu, peta generacija mobilne mreže (5G), omogućuje primjerice kontrolu dronova sa prijenosom slike 4K rezolucije i upravljanje u realnom vremenu s minimalnim kašnjenjem komunikacije (oko 1ms). Također, kompanija Ericsson navodi kako će pojavom pete generacije mreže doći do porasta IoT uređaja za medicinske potrebe. [8]

1.9. Prednosti

Ušteda troškova

- IoT čini naše sustave efikasnijim. Omogućuje da elektronički uređaji međusobno učinkovito komuniciraju čime štede troškove energije. Omogućuje razmjenu i prikupljanje podataka za daljnju analizu i obradu te se tako smanjuju troškovi proizvodnje u industrijama.

Informacije

- Znanje je skupo, ali ne znanje je još skuplje. Uz više informacija mogu se donositi bolje odluke, bilo da se radi o općim odlukama ili vođenju tvrtke. IoT može prikupljati bitne informacije o sustavu unutar industrije te je moguće odraditi preventivne servise koji će nas kasnije financijski koštati jeftinije od zastoja cijelog postrojenja.

Komunikacija

- IoT potiče komunikaciju između strojeva (engl. *Machine to machine, M2M*). Na ovaj način sustavi se mogu međusobno povezati te postaju inteligentniji i autonomniji čime se povećava i učinkovitost samih uređaja.

Automatizacija i upravljanje

- Svi strojevi mogu međusobno komunicirati čime se postiže poboljšavanje u sustavima automatizacije i samog upravljanja. S obzirom na prisutnost bežične komunikacije koja je svakim danom kvalitetnija, sustavi automatizacije također napreduju. Veliku ulogu u automatizaciji upravo pridonose IoT sustavi.

Povećavanje produktivnosti

- Produktivnost ima ključnu ulogu u profitabilnosti bilo kojeg posla. IoT poboljšava efikasnost rada i također smanjuje neusklađenost sustava te time smanjuje i same troškove.

1.10. Nedostaci

Život ovisan o tehnologiji

- Sve više mlađih naraštaja ovisno je o tehnologiji, a uz pomoć IoT-a, ta ovisnost postaje sve veća u svakodnevnoj rutini. Potpun oslonac na IoT sustave i uređaje može stvoriti probleme u slučaju zatajenja ili pada infrastrukture.

Privatnost i sigurnost

- Hakerski napadi današnja su svakodnevnica u IoT svijetu što otežava čuvanje podataka od neovlaštenih korisnika. Spajanjem sve više uređaja i strojeva s internetom, puno je više informacija dostupno ukoliko oni nisu zaštićeni nekom vrstom sigurnosne zaštite.

Manje radnih mjesta

- U svakodnevnici sve se više sustava automatizira gdje IoT također ima veliki doprinos. Takvim pristupom prirodno je da se zahtjevi za ljudskim resursima smanjuju, a kvalificirane radne snage sve je manje.

Složenost

- U svim sustavima postoje mogućnosti kvarova, pa tako i kod IoT-a. Pojava kvarova, bili oni softverskog ili hardverskog tipa, na IoT sustavima kojih je sve više, mogla bi biti sve češća.

Kompatibilnost

- Međunarodni standard za kompatibilnost IoT-a trenutno ne postoji. Implementacija različitih rješenja proizvođača mogu otežati međusobnu komunikaciju uređaja.

2. MQTT - Message Queue Telemetry Transport

Ako postoji mnogo različitih uređaja koji moraju međusobno komunicirati, potrebno je koristiti neku vrstu komunikacijskog protokola. Pomoću komunikacijskog protokola uređaji mogu zatražiti podatke od drugih uređaja, a uređaji koji takve zahtjeve zaprime moraju odgovoriti sa traženim podacima. Uređaj ne mora nužno zatražiti podatke već ih bez upita može kontinuirano primati, a sve te podatke uređaj mora obraditi. Komunikacija s uređajima, ili uspostava komunikacije između uređaja u realnom vremenu putem interneta, može uzrokovati probleme. Neki uređaji napajaju se iz baterija te njihova potrošnja energije mora biti niska, a neki uređaji jednostavno nemaju dovoljno resursa te nisu u mogućnosti izvršavati zahtjevne protokole. Uz sve do sada navedeno potrebno je imati na umu da uređaj koji se spaja na internet mora biti zaštićen od neovlaštenog pristupanja odnosno hakiranja. [9] [10]

Za komunikaciju putem interneta mogao bi se iskoristiti dobro poznati HTTP protokol koji će povezati uređaje kako bi međusobno komunicirali, ali HTTP protokol u pogledu na resurse i mrežni promet nije toliko „lagan“. Kod svake razmjene poruka HTTP protokol raskida vezu, a kako bi ponovo komunicirao mora uspostaviti vezu po principu *„3 way handshake“* što bi značilo da će uređaji većinu vremena i resursa trošiti na uspostavu i raskid veze. Kod malo detaljnijeg proučavanja HTTP protokola, vidljivo je da klijent može pristupiti poslužitelju no poslužitelj ne može slati podatke klijentu ako on to nije prethodno zatražio. Ovakav protokol stvara problem kod dvosmjerne komunikacije. [11]

Sada je već jasno da za komunikaciju između uređaja limitiranih resursa ili niske potrošnje energije mora postojati manje zahtjevan protokol od HTTP protokola. Postoji protokol posebno osmišljen da bude „laganiji“ od HTTP-a i koji može funkcionirati kada su u pitanju nepouzdane i nestabilne mreže kod kojih može doći do nekontroliranog raskida veze između uređaja. MQTT (eng. *Message Queue Telemetry Transport*) prikladniji je za komunikaciju između uređaja, pogotovo kada je njihov broj veći, a potrebna je razmjena podataka u realnom vremenu putem interneta uz malu potrošnju energije. MQTT je protokol osmišljen za komunikaciju između strojeva (engl. *Machine-to-machine, M2M*) i radi na principu pretplati-objavi, a zasniva se na TCP/IP sloju. [9] [10]

2.1. Povijest

Krajem devedesetih godina sustavi nadzorne kontrole i prikupljanja podataka korišteni su za upravljanje velikom industrijskom infrastrukturom poput naftovoda ili elektroenergetskih mreža (u stvari, SCADA se i danas vrlo često koristi). U osnovi, SCADA sustavi omogućavali su operaterima daljinsku kontrolu raznih aspekata infrastrukture, poput uključivanja ili isključivanja generatora u električnoj mreži. Usput, budući da SCADA sustavi mogu upravljati strojevima i uređajima daljinskim spajanjem na njih, oni se u velikoj mjeri smatraju pretečama današnjih IoT sustava.

Sredstva za povezivanje s udaljenim uređajima i kontrolu nad njima nazivana su "telemetrijom". Problemi s telemetrijom u povijesti bili su u tome što su postojali razni protokoli obično razvijani od strane proizvođača uređaja ili strojeva. To je rezultiralo raznim vrstama problema s kompatibilnošću. Dva IBM-ova inženjera odlučila su da će to pokušati riješiti razvijanjem jedinstvenog protokola otvorenog koda. Ti su inženjeri bili Andy Stanford-Clark i Arlen Nipper i zajedno su započeli rad na nečemu što su nazvali MQTT. Andy Stanford-Clark i Arlen Nipper imali su ideju da protokol koji razvijaju mora biti kompaktan, lako razumljiv i jednostavan za implementaciju, a takve ideje se održavaju i danas te se pokušava osigurati da bilo koje ažuriranje protokola poštuje početne ideje. Prvu verziju MQTT protokola objavili su 1998. godine. Pojavom IoT-a proizvođači su morali pronaći protokol za povezivanje svojih uređaja. Jednostavnost i mala potrošnja energije MQTT-a odgovarala je njihovim potrebama i polako njegova primjena postaje sve češća. Tada je oko 2008. godine došlo do prekretnice kada se pojavljuje MQTT broker otvorenog koda - Mosquitto (koji se danas naziva Eclipse Mosquitto).

Do 2014. godine MQTT je bio OASIS standard, a 2016. postao je ISO standard. Do ovog trenutka je već povezao milijune uređaja širom svijeta u svim vrstama aplikacija i industrija. IBM, Amazon i Microsoft samo su neki primjeri velikih kompanija koje su prihvatile MQTT kao svoj osnovni protokol. [12] [13]

2.2. Princip rada

MQTT moguće je podijeliti u četiri faze: uspostavljanje veze, provjera autentičnosti, komunikacija, raskid veze. Svaki MQTT klijent uspostavlja vezu s brokerom pomoću protokola kontrole prijenosa / internetskog protokola (TCP/IP) uz definiran standardni ili prilagođeni port. Standardni portovi su 1883 za komunikaciju bez šifriranja i 8883 za šifriranu komunikaciju kod koje se koristi sloj sigurnosnih portova (eng. *Secure Scokets Layer*), odnosno SSL i sigurnost transportnog sloja (eng. *Transport Layer Security*), odnosno TLS. Kod SSL/TLS rukovanja, klijent potvrđuje certifikat poslužitelja i ovjerava poslužitelja. Klijent također može pružiti klijentsku potvrdu brokeru tijekom rukovanja. Broker može to koristiti za autentifikaciju klijenta. Iako nije izričito dio MQTT specifikacije, brokeri su postali uobičajeni za podršku autentifikacije klijenta SSL/TLS certifikatom na strani klijenta. Budući da MQTT protokol ima cilj biti kompaktan i lako razumljiv SSL/TLS ne može uvijek biti dobra opcija, a u nekim slučajevima je nepoželjna. U takvim se slučajevima provjera autentičnosti provodi korištenjem korisničkog imena i lozinke jasnog teksta, koje klijent šalje poslužitelju u sklopu CONNECT / CONNACK paketa. Neki brokeri, posebno javno dostupni brokeri, prihvatit će anonimne klijente. U takvim slučajevima, korisničko ime i lozinka nisu potrebni.

MQTT poruka se sastoji od fiksnog zaglavlja veličine 2 bajta, izbornog varijabilnog zaglavlja, a veličina poruke je ograničena na 256 megabajta (MB) u što se ubraja informacija poruke i razina usluge (QoS). Tijekom komunikacije, klijent može obavljati operacije objavljivanja, pretplate, odjave i ping-a. Kod objavljivanja šalje se binarni blok podataka odnosno poruka. Izvršavanjem ping naredbe vrši se PINGREQ / PINGRESP paketni niz, a ova operacija nema drugu funkciju osim održavanja aktivne veze i osiguravanja da TCP veza nije prekinuta od strane pristupnika ili usmjerivača (eng. *router*).

Kad objavitelj (eng. *publisher*) ili pretplatitelj (eng. *subscriber*) želi prekinuti vezu s MQTT brokerom, brokeru šalje DISCONNECT poruku, a zatim prekida vezu. To se naziva gracioznim isključivanjem (eng. *graceful shutdown*) jer se klijentu pruža mogućnost da se lako ponovno poveže slanjem identiteta i nastavlja tamo gdje je stao. Ako se prekid veze dogodi iznenada, bez vremena da objavitelj pošalje DISCONNECT poruku, broker može poslati pretplatnicima automatsku poruku da je objavitelj izgubio vezu sa brokerom. Takva poruka se naziva oporuka (eng. *Will*). [9] [10]

2.3. MQTT klijent

Kad se spominje MQTT klijent tada se misli na uređaje koji koriste MQTT protokol za spajanje na MQTT broker putem interneta. Objavitelj (eng. *publisher*) i pretplatitelj (eng. *subscriber*) su zapravo MQTT klijenti, a ovisno o radnji koju trenutno izvršavaju (šalju ili primaju podatke) poprimaju navedene nazive. Kod objavitelja i pretplatitelja bitno je spomenuti kako jedan klijent može izvršavati obje zadaće. MQTT klijent može biti računalo koje nadzire i grafički prikazuje podatke te pruža korisniku interakciju s IoT svijetom, a može biti i mikrokontroler baziran na MQTT biblioteci koja mu je namijenjena. Zaključno o klijentu se može reći da je to svaki uređaj koji koristi MQTT protokol unutar TCP/IP mrežne strukture. Danas postoji mnogo MQTT biblioteka pa tako klijentom može postati doslovce svaki uređaj koji ima pristup internetu. Biblioteke većinom pokrivaju širok spektar programskih jezika, a neki od jezika i platforma s dostupnim bibliotekama su: C, C++, Arduino, Android, Java, JavaScript, .NET, C#. [14]

2.4. MQTT broker

Broker ima glavnu ulogu u svakom objavi/pretplati protokolu, pa tako i kod MQTT-a. Broker ima zadaću primati i obraditi sve primljene poruke te voditi računa koji klijent prima, a koji objavljuje podatke. Svaki broker ujedno mora moći održavati konekciju s velikim brojem priključenih uređaja na MQTT (primjerice Emqtt broker može održati vezu između 3.1 milijuna uređaja istovremeno). Bitna uloga brokera je ujedno sačuvati sesije i propuštene poruke svojih klijenata (ako klijent to zatraži). Provjeru autentičnosti i autorizaciju klijenta također može provesti broker, a postoji mogućnost i njegovog proširenja pa se mogu primijeniti i dodatne mjere internetske sigurnosti. Broker se može instalirati na bilo koje računalo ili server s dovoljnom količinom raspoloživih resursa, a važno ga je integrirati u stabilne sustave iz razloga što prima, obrađuje i preusmjerava velike količine podataka. Danas postoje javno dostupni brokeri (odnosno web servisi) namijenjeni za testiranje implementiranih rješenja MQTT komunikacije. [14] [15]

Naziv	Web adresa	Podržani portovi
Flespi	mqtt.flespi.io	1883 (TCP), 80 (WebSocket), 8883 (SSL), 443 (Secure WebSockets)
Mosquitto	test.mosquitto.org	1883, 8883(SSL), 8884 (SSL+Clientcertificate), 8080 (WebSockets), 8081(WebSockets+SSL)
HiveMQ	broker.hivemq.com	1883, 8000 (WebSockets)
Adafruit IO	adafruit.io	1883 (TCP), 8883 (SSL), 443 (Secure WebSockets)

Tablica 2.1: Neki od dostupnih MQTT brokera[15]

2.5. Povezivanje

Veze se uvijek uspostavljaju između klijenta i brokera. Klijenti se ne mogu izravno povezati međusobno. Veza se uspostavlja tako što klijent šalje CONNECT poruku, na što broker odgovara s porukom CONNACK. Potrebno je uspostaviti vezu kako bi se omogućila objava razine kvalitete komunikacije (QoS) ili izvršila pretplata na temu. Klijenti se obično povezuju s brokerom pomoću ID-a klijenta (ClientID), korisničkog imena i zaporke. Klijent može imati samo jednu vezu s istim brokerom. Bitno je napomenuti kako broker koristi ID klijenta za njegovu identifikaciju i on mora biti jedinstven. [14]

2.6. Održavanje veze

Parametar održavanja na životu (eng. *Keep alive*) definira najduže razdoblje u kojem veza može ostati uspostavljena bez da klijenti razmjenjuju poruke. Time se omogućuje prelazak u stanje mirovanja tijekom kojeg se sve poruke blokiraju na brokeru i isporučuju klijentima kasnije kad se probude. Da bi veza ostala uspostavljena, klijent šalje PINGREQ prije isteka parametra održavanja na životu, na što broker odgovara sa PINGRESP. Kad uređaj nije vidljiv unutar definiranog vremena održavanja na životu, broker može odlučiti prekinuti sesiju i zaustavit će slanje poruka namijenjenih uređaju do sljedeće sesije. [14]

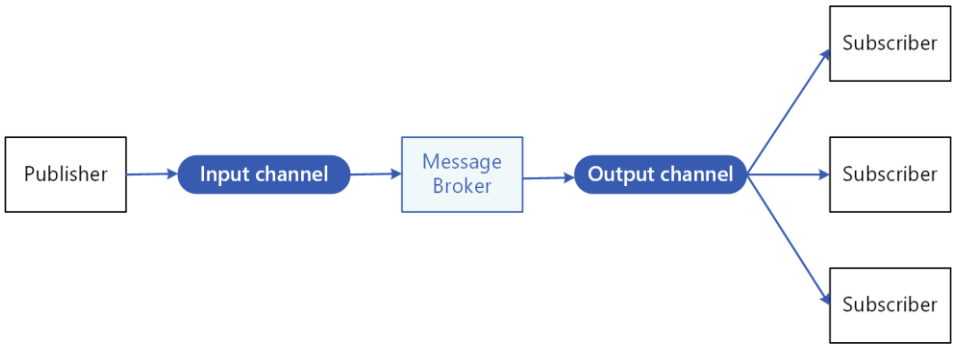
2.7. Čista sesija

Parametar čista sesija (eng. *Clean Session*) omogućuje uspostavu nove svježe veze bez poruka koje su čekale na brokeru ponovno spajanje klijenta. Ako zastavica čiste sesije nije postavljena, broker mora upotrebljavati već spremljenu sesiju vezanu s ID-om klijenta i na temelju te sesije uspostaviti vezu. U slučaju da sesija ne postoji (ako se klijent spaja prvi put) tada broker mora stvoriti novu sesiju, a ona mora nakon raskida veze biti upamćena od strane brokera i klijenta. Broker će ujedno, ako klijent u trenutku dospijeća poruke nije povezan na broker, pohranjivati sve poruke za klijenta ali pod uvjetom da je klijent prethodno bio pretplaćen na kvaliteti usluge (QoS) razine 1 ili 2.

Ako je zastavica čiste sesije postavljena, broker i klijent moraju obrisati staru sesiju i stvoriti novu. Životni ciklus sesije bit će u skladu s mrežnom vezom i poruke koje klijent nije primio će biti izgubljene, a stara sesija ne smije se ponovo koristiti prilikom sljedećeg povezivanja. [14]

2.8. Objavi/pretplati

U komunikaciji klijent-server koja nam je poznata primjerice kod HTTP protokola, komunicira se izravno s krajnjom točkom. Kod modela objavi/pretplati (eng. *publish/subscribe*) klijenti se ne spajaju direktno međusobno već se pretplaćuju na temu kako bi slali i primali poruke. Kako bi se izvršila pretplata na temu, klijent prvo mora poslati brokeru SUBSCRIBE zahtjev. SUBSCRIBE zahtjev može sadržavati više tema. Broker tada odgovara na SUBSCRIBE zahtjev s potvrdom pretplate odnosno SUBACK porukom. Takav model razdvaja objavitelja i pretplatitelja što je važno za komuniciranje. Objavitelj i pretplatitelj ne moraju međusobno razmjenjivati IP adrese te ne moraju biti istovremeno aktivni. [16] [17]



Slika 2.1: Princip modela objavi/pretplati [17]

2.9. Teme i zamjenski znakovi

Sve poruke koje se izmjenjuju između klijenata prolaze kroz broker, a sve te poruke nalaze se unutar definiranih tema. Svaka tema može se sastojati od nekoliko razina korištenjem separatora odnosno znaka „ / “. Bitno je napomenuti da kod otvaranja nove teme klijent ne mora slati dodatne inicijalizacije već samo niz znakova (eng. *String*) kodiranih u UTF-8 formatu kako bi se formirao naziv teme. Primjer definirane teme bio bi: kuća/soba/temperatura ili kuća/garaža/vlaga

Kod pretplata mogu se koristiti puni nazivi tema na koje se klijent želi pretplatiti ili zamjenski znakovi (eng. *Wildcard*) koji omogućuju pretplatu na više tema odjednom. Zamjenske znakove moguće je koristiti samo za pretplate na temu dok ih za objavljivanje nije moguće koristiti.

Jedan od zamjenskih znakova je „ # “ koji ima ulogu izvršiti pretplatu na više tema, dok drugi zamjenski znak „ + “ ima ulogu izvršiti pretplatu samo na jednu razinu hijerarhije. [18]

Primjer upotrebe zamjenskog znaka „ # “: kuća/prviKat/#

-navedenom naredbom izvršava se pretplata na sve razine tema unutar glavne teme „kuća/prviKat“

kuća/prviKat/soba1/temperatura

kuća/prviKat/soba1/vlaga

kuća/prviKat/soba2/temperatura

kuća/prviKat/soba2/vlaga

-pretplata se neće izvršiti na sljedećim temama:

kuća/prizemlje/soba1/temperatura

kuća/prizemlje/soba1/vlaga

Primjer upotrebe zamjenskog znaka „ + “: kuća/+/soba1/temperatura

- navedenom naredbom izvršava se pretplata na jednu hijerarhiju unutar glavne teme

kuća/prviKat/soba1/temperatura

kuća/prizemlje/soba1/temperatura

-pretplata se neće izvršiti na sljedećim temama:

kuća/prviKat/soba1/vlaga

kuća/prviKat/soba2/temperatura

kuća/prviKat/soba2/vlaga

kuća/prizemlje/soba1/vlaga

2.10. Kvaliteta usluge (QoS)

Kvaliteta usluge (eng. *Quality of Service, QoS*) je zapravo jedna vrsta dogovora između brokera i klijenta kojim definiraju razinu potvrde o zaprimljenim porukama. Kad klijent objavljuje brokeru, klijent određuje razinu kvalitete usluge (QoS) za tu poruku. Kad broker pošalje poruku klijentu koji se pretplaćuje, kvalitetu usluge (QoS) koji je prvi klijent postavio za tu poruku, koristi se ponovo od brokera do pretplatnika. Na taj način klijent koji objavljuje poruke određuje kvalitetu usluge (QoS) sve do krajnjih primatelja. [19]

Postoje tri razine kvalitete usluge (QoS):

- 0 – „najviše odjednom“ (eng. *At most once*)
- 1 –„barem jednom“ (eng. *At least once*)
- 2 –„Točno jednom“ (eng. *Exactly once*)

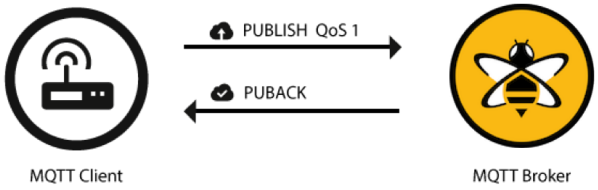
QoS razina 0 – „najviše odjednom“ (eng. *At most once*)

- Radi se o najjednostavnijem načinu slanja poruke. Klijent u ovom slučaju jednostavno objavi poruku ali nema potvrdu od brokera o primitku iste. Takva razina kvalitete usluge prikladna je kada postoji pouzdana veza između klijenta i brokera ili kod kontinuiranog slanja podataka. Primjerice kod očitavanja senzora temperature jedan izgubljeni podatak ne stvara problem već samo dolazi do zakašnjenja. [19]

Slika 2.2: Vizualna predodžba razmjene poruka za QoS 0 [19]

QoS razina 1 – „barem jednom“ (eng. *At least once*)

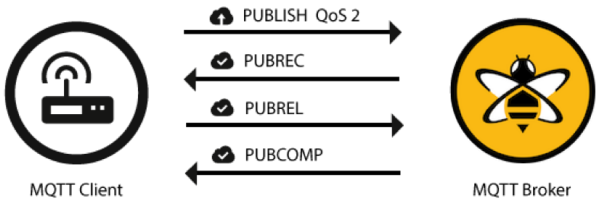
- Metoda koja jamči da će se poruka prenijeti uspješno do brokera. Broker će tada poslati potvrdu o primitku, ali u slučaju da potvrda ne dođe do klijenta klijent neće shvatiti da je poruka koju je poslao uspješno došla do brokera te će je ponovo poslati. Klijent će tako nastavljati slati poruke sve dok ne dobije potvrdu, a time je zajamčeno da će poruka stići do brokera, ali postoji mogućnost slanja iste poruke više puta. QoS razine 1 je dobar odabir ako postoji potreba da broker primi poruku ali i može tolerirati primanje iste poruke više puta. Kod podataka koji su promjenjive prirode pojava iste poruke možda neće narušiti sustav, ali kod podatka kojim se vrši radnja, primjerice brojanje pritiska tipke, može se stvoriti dojam da je tipka pritisnuta više puta. [19]



Slika 2.3: Vizualna predodžba razmjene poruka za QoS 1 [19]

QoS razina 2 – „točno jednom“ (eng. *Exactly once*)

- Radi se o najvišoj razini slanja poruke. U drugoj razini kvalitete usluge postoji niz od četiri razmijenjenih poruka, poznatije kao četverostruko rukovanje (eng. *four step handshake*), između klijenta i brokera. Na taj način su klijent i broker sigurni da su uspješno razmijenili poruke. QoS razine 2 garantira sigurnu isporuku poruke točno jednom, ali to ima svoju cijenu u pogledu na količinu prijenosa podataka. [19]



Slika 2.4: Vizualna predodžba razmjene poruka za QoS 2 [19]

3. Implementacija IoT rješenja

U završnom radu izrađen je primjer postojećeg sustava/stroja koji nema mogućnost povezivanja na internet kako bi postao dio IoT svijeta. Postojeći sustav realiziran je na razvojnoj platformi Arduino, a sastoji se od senzora za prikupljanje podataka o temperaturi i vlazi (HTU21D), tlaka zraka (BMP180) te ventilatora koji služi za simuliranje ulazno/izlaznog modula kojim se može upravljati. Ujedno se mogu nadgledati promjene koje se dešavaju uzrokovane upravljanjem. Za vanjsko proširenje, odnosno modul, korišten je esp8266 koji sve podatke prima i šalje na Arduino pomoću serijskog porta (UART). Kako esp8266 raspolaže s dovoljnom količinom resursa, obavlja sve dodatne obrade podataka koje prima kako bi se Arduino (postojeći sustav) što manje opteretio, prikazuje ih na ekranu i izvršava sve zadatke vezane uz održavanje internetske konekcije. Za upravljanje putem interneta, implementiran je MQTT protokol na sam esp8266 te je razvijena Android aplikacija za nadzor i upravljanje samim sustavom.

3.1. Arduino

Arduino je razvojno okruženje otvorenog hardvera (eng. *open hardware*) i otvorenog koda (eng. *open software*), koje pojednostavljuje programiranje Atmega328p mikrokontrolera. Za programiranje koristi vlastiti Arduino jezik, baziran na programskom jeziku Processing.

Razvojna pločica ima 14 pinova koji mogu biti digitalni ulazi ili izlazi i 6 pinova koji mogu biti analogni (10-bita) ili digitalni ulazi i digitalni izlazi. Mogućnost za pulsno-širinsku modulaciju (eng. *Pulse-Width Modulation, PWM*) ima na 6 pinova, odnosno izlaza, a njihova frekvencija je 976.56Hz i 490.20Hz. Svaki izlaz može podnijeti najviše 40mA, a sveukupno izlazi ne smiju prijeći 200mA. [20]

Slika 3.2: Arduino Uno [20]

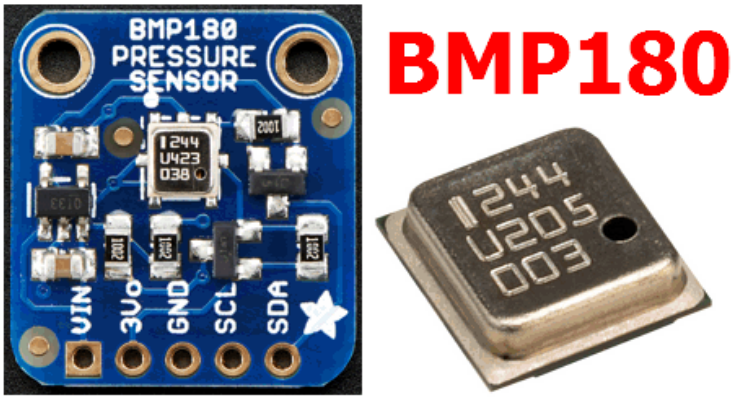
Slika 3.1: Arduino Nano [20]

3.1.1. BMP180

BMP180 je jeftin i precizan senzor za mjerenje barometarskog tlaka zraka kojeg proizvodi tvrtka Bosch. Radi se zapravo o piezorezistivnom senzoru tlaka, a takvi senzori sastavljeni su od poluvodičkog materijala (obično silicija) koji mijenja otpor kad se primjeni sila poput atmosferskog tlaka. Senzor mjeri pritisak i temperaturu jer temperatura mijenja gustoću plinova poput zraka. Pri višim temperaturama zrak nije tako gust i težak tako da vrši manji pritisak na senzor, a kod nižih temperatura zrak je gušći i teži pa je sam pritisak veći. BMP180 koristi mjerenja temperature u stvarnom vremenu kako bi kompenzirao očitanja tlaka kod promjene gustoće zraka.

BMP180 ima 176-bitni EEPROM koji sadrži 11 različitih koeficijenata kalibracije koji su jedinstveni za svaki senzor. Oni se koriste za izračunavanje pravog barometarskog tlaka i temperature. Pravi tlak i temperatura izračunavaju se korištenjem prilično složenih algoritama. [21] [22]

U našem slučaju koristi se senzor koji se već nalazi na vlastitoj tiskanoj elektronskoj pločici, a na njoj se još nalazi 3.3V regulator napona, I2C sklop za promjenu razine napona (eng. *logic level shifter*), otpornici za pull-up I2C komunikacijskih linija.



Slika 3.3: BMP180 sa regulatorom napona i komponentama za promjenu razine napona [23]

Napon napajanja	3V – 5V
Napon logike (I2C)	kompatibilno od 3V do 5V
Raspon osjeta pritiska	300 – 1100 hPa
Rezolucija	0.03 hPa
Temperaturna točnost	± 2°C
Radni raspon temperature	-40 to +85°C

Tablica 3.1: Osnovne karakteristike BMP180 senzora [24]

Kod programiranja korištena je *SFE_BMP180* biblioteka kako bi se olakšalo programiranje i izbjeglo pisanje složenih proračuna i algoritama za proračun tlaka.

Korištene naredbe iz biblioteke:

-prvo se koristi metoda za pokretanje mjerenja temperature:

`status = bmp180.startTemperature();`

-tada je potrebno čekati najmanje 4,5 milisekundi i koristi se *getTemperature (T)* za primanje vrijednosti i pohranjuje se u varijablu: `status = bmp180.getTemperature(T);`

- Metoda *startPressure()* šalje naredbu za pokretanje mjerenja tlaka. Kao dodatni parametar zadaje se broj između 0 i 3. Vrijednost 3 daje visoku razlučivost, ali i duže odgađanje između mjerenja. Vrijednost 0 pruža nižu razlučivost, ali je brža. Funkcija vraća broj milisekundi koje Arduino treba pričekati prije nego što očita vrijednost tlaka s senzora:

`status = bmp180.startPressure(3);`

- Zatim se koristi metoda *getPressure()* kako bi se preračunala vrijednost tlaka uz poznatu temperaturu prethodno izmjerenu:

`status = bmp180.getPressure(P, T);`

```
#include <Wire.h>
#include <SFE_BMP180.h>
SFE_BMP180 bmp180;
void setup()
{
  bmp180.begin();
}
void loop()
{
    char status;
    double T, P;
    status = bmp180.startTemperature();
    if (status != 0)
    {
      delay(status);
      status = bmp180.getTemperature(T);
      if (status != 0)
      {
        status = bmp180.startPressure(3);
        if (status != 0)
        {
          delay(status);
          status = bmp180.getPressure(P, T);
          if (status != 0)
          {
            pressure = bmp180.sealevel(P, Altitude);
          }
        }
      }
   }
}
```

Programski blok 3.1: Arduino kod za senzor tlaka – BMP180

3.1.2. HTU21D

HTU21D je jeftin, vrlo precizan, digitalni senzor vlage i temperature, a sve što treba su dvije linije za I2C komunikaciju. Ovaj je senzor idealan za mjerenje vlage i evidentiranje podataka, savršen za meteorološku stanicu ili sustav za kontrolu vlage. Ovaj I2C digitalni senzor vlažnosti zraka točna je i inteligentna jeftina alternativa. Ima tipičnu točnost od ± 2% s radnim rasponom koji je optimiziran od 5% do 95% relativne vlažnosti. Rad izvan ovog raspona je i dalje mogući, samo točnost može malo odstupati. Mjerenje temperature ima raspon od -40 do 125 °C. [25]

U našem slučaju koristi se senzor koji se već nalazi na vlastitoj tiskanoj elektronskoj pločici, a na njoj se još nalazi 3.3V regulator napona, I2C sklop za promjenu razine napona (eng. *logic level shifter*), otpornici za pull-up I2C komunikacijskih linija.

Slika 3.4: HTU21D sa regulatorom napona i komponentama za promjenu razine napona [26]

Napon napajanja	3V – 5V
Napon logike (I2C)	kompatibilno od 3V do 5V
Raspon osjeta vlage	0 – 100 %RH
Rezolucija osjeta vlage	0.04 %RH
Raspon mjerenja temperature	od -40 do +125°C
Temperaturna točnost	± 0.3°C

Tablica 3.2: Osnovne karakteristike HTU21D senzora [27]

Kod programiranja korištena je *SparkFunHTU21D* biblioteka kako bi se olakšalo programiranje.

Korištene naredbe iz biblioteke:

-funkcija za čitanje temperature: `myHumidity.readHumidity()`

-funkcija za čitanje vlage: `myHumidity.readTemperature()`

Arduino program:

```
#include <Wire.h>
#include "SparkFunHTU21D.h"

HTU21D myHumidity;

float hum = 0.0;
float temp = 0.0;

void setup()
{
  myHumidity.begin();
}

void loop()
{
    hum = myHumidity.readHumidity();
    temp = myHumidity.readTemperature();
}
```

Programski blok 3.2: Arduino kod za senzor temperature i vlage – HTU21D

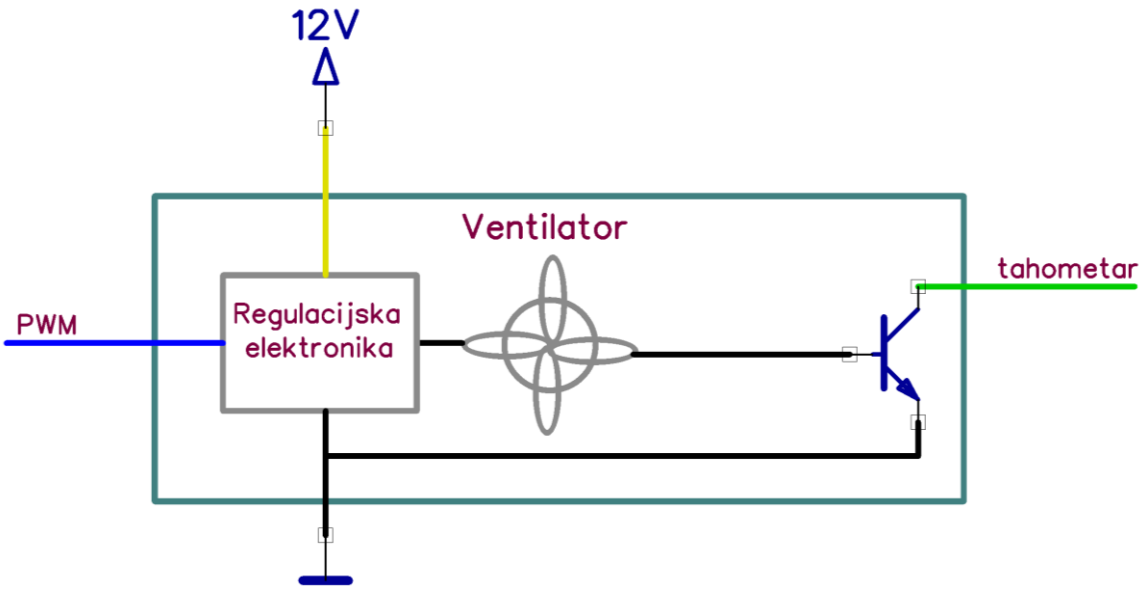
3.1.3. Ventilator

Kao jedan ulazno-izlazni modul koristi se ventilator koji se inače može pronaći na hladnjaku procesora osobnog računala. Model ventilatora koji se koristi (Delta Electronic-afb0612vhc- p52z) ima četiri žice od kojih dvije služe za napajanje, jedna za upravljanje brzinom te jedna za kontrolu broja okretaja. Jedan od zadataka bio je omogućiti upravljanje ventilatorom te nadziranje njegove brzine vrtnje, a kako bi to odradili prvo je potrebno istražiti kako taj ventilator funkcionira.

Već je spomenuto da ventilator ima četiri žice, a zbog nedostatka tehničke dokumentacije za navedeni model, potrebno je otkriti njihovu funkciju. Svaka žica je različite boje te se prema tome raspoznaje njihova funkcija. Prema naljepnici na samom ventilatoru za početak se zaključuje da se napaja iz istosmjernog izvora 12V (DC12V) dok su daljnjim testiranjima utvrđene ostale funkcije:

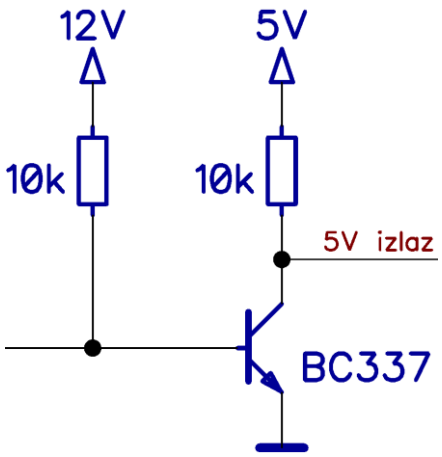
Boja žice	Funkcija
žuta	pozitivan polaritet napajanja
crna	negativan polaritet napajanja
zelena	tahometar
plava	upravljanje brzinom (PWM)

Tablica 3.3: Funkcije ventilatora prema bojama žica

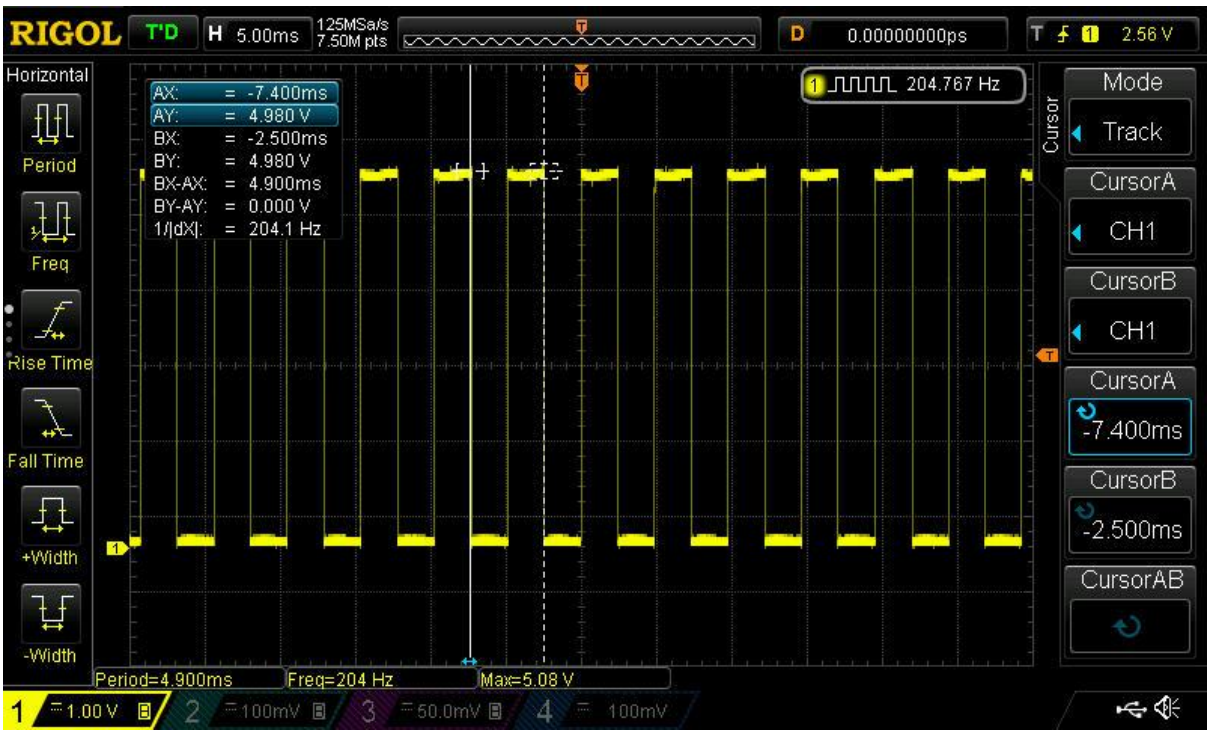


Slika 3.5: Shema funkcioniranja ventilatora

Ventilator ima ugrađen tahometar za mjerenje brzine koji funkcionira na način da svaku polovicu kruga na svom izlazu daje impuls. U samom početku testiranja tahometar na svojem izlazu ne daje upotrebljiv signal te se dolazi do zaključka kako se radi o izlazu tipa otvoreni kolektor (eng. *opent collector*). Kako bi izlaz tipa otvoreni kolektor funkcionirao dodaje se pull-up otpornik, u ovom slučaju na 12V. Ovdje se nailazi na prepreku iz razloga što Arduino ne može primiti 12V signal na svom ulazu te uz pomoć NPN tranzistora (BC337) signal se konvertira na upotrebljivih 5V. Potrebno je napomenuti kako je iskorišten invertirajući spoj (slika 3.6) ali u ovom slučaju to ne stvara nikakav problem.



Slika 3.6: Shema konvertiranja signala tahometra s 12V na 5V



Slika 3.7: Izlazni signal tahometra - osciloskop

Preostala žica je PWM signal, a testiranje je pokazalo kako spajanjem na negativan polaritet napajanja ventilator se vrti minimalnom brzinom, dok spajanjem na 5V ventilator se vrti maksimalnom brzinom.

Pošto se Arduino koristi kao postojeći sustav i cilj ga je što manje opteretiti, programiranje se svodi samo na osnovne logičke funkcije kao što su: paljenje i gašenje ventilatora, brzina vrtnje (PWM 8-bitni broj) te mjerenje trajanja vremena između signala tahometra.

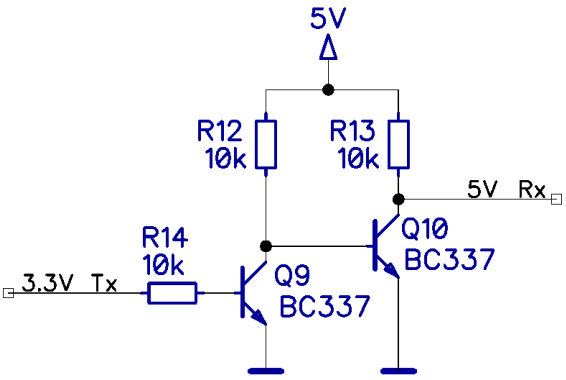
```
#define fanPulse 7
#define fanVcc 2
#define pwm 3

int Auto = 1;
int AutoSpeed = 150;
int onoff = 1;

void setup()
{
  pinMode(fanPulse, INPUT);
  pinMode(pwm, OUTPUT);
  pinMode(fanVcc, OUTPUT);
  digitalWrite(fanVcc, HIGH);
  analogWrite(pwm, AutoSpeed);
}
void loop()
{
  //pozivanje ostalih funkcija i ventilator funkcije
}
void ventilator()
{
  if (strcmp(SerialString, "ventilator") == 0)
  {
    if (Auto == 1)
    {
      AutoSpeed = 150;
      digitalWrite(fanVcc, HIGH);
      analogWrite(pwm, AutoSpeed);
    }
    else
    {
      if (onoff == 1)
      {
        digitalWrite(fanVcc, HIGH);
        analogWrite(pwm, AutoSpeed);
      }
      else
      {
        digitalWrite(fanVcc, LOW);
      }
    }
  }
}
```

Programski blok 3.3: Arduino kod za upravljanje ventilatorom

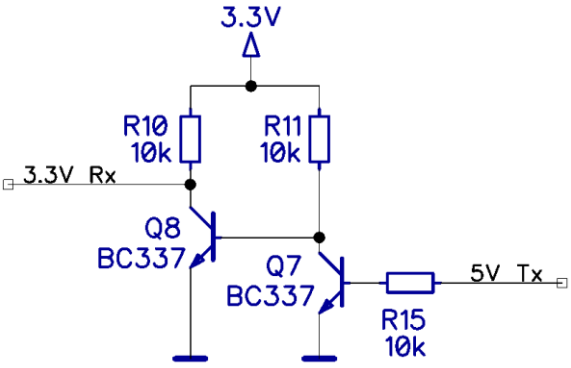
3.1.4. Sklop za promjenu razine napona (logic level shifter)

Kako bi Arduino i Esp8266 mogli komunicirati pomoću serijskog porta, prvo je potrebno osigurati da se to odvija na istoj naponskoj razini. Arduino na svojim digitalnim izlazima kao logičku jedinicu postavlja 5V, dok Esp8266 za logičku jedinicu postavlja 3.3V. Kako ne bi došlo do razlike potencijala i oštećivanja samog hardvera koristi se sklop za promjenu razine napona (eng. *logic level shifter*). Sklop za promjenu razine napona služi za pretvaranje jedne logičke naponske razine u drugu naponsku razinu i na taj način ostvaruje kompatibilnost između komponenti. Kako bi to bilo ostvareno koriste se BC337 NPN tranzistori.

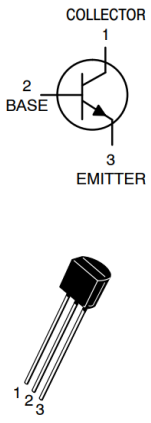
Za svaku liniju (Rx i Tx) korištena su dva tranzistora iz razloga što se signal invertira dva puta, a to znači da se signal vrača u početno stanje ali promijenjenog napona. Korištenjem samo jednog tranzistora dobili bi se invertirajući signali što u ovom slučaju nije prihvatljivo (svaka logička jedinica u komunikaciji postala bi logička nula).



Slika 3.8: Promjena logičkog napona 3.3V na 5V



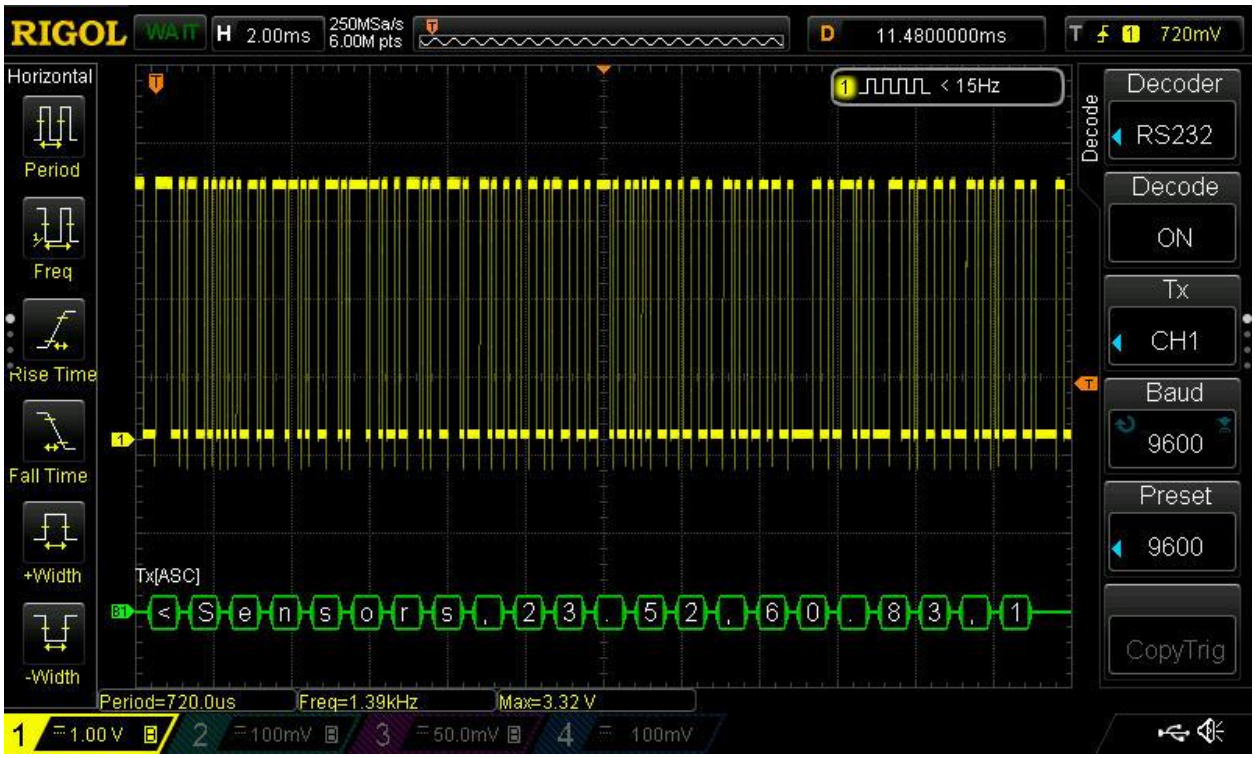
Slika 3.9 Promjena logičkog napona 5V na 3.3V



Slika 3.10: BC337 elektrode [28]

Tip tranzistora	NPN
Kolektor-Emiter napon	45V
Emiter-Baza napon	5V
Struja kolektora	800mA
Hfe	min. 100, max 630

Tablica 3.4: Osnovne karakteristike BC337 tranzistora [28]



Slika 3.11: Testiranje sklopa za promjenu razine napona 5V na 3.3V - osciloskop

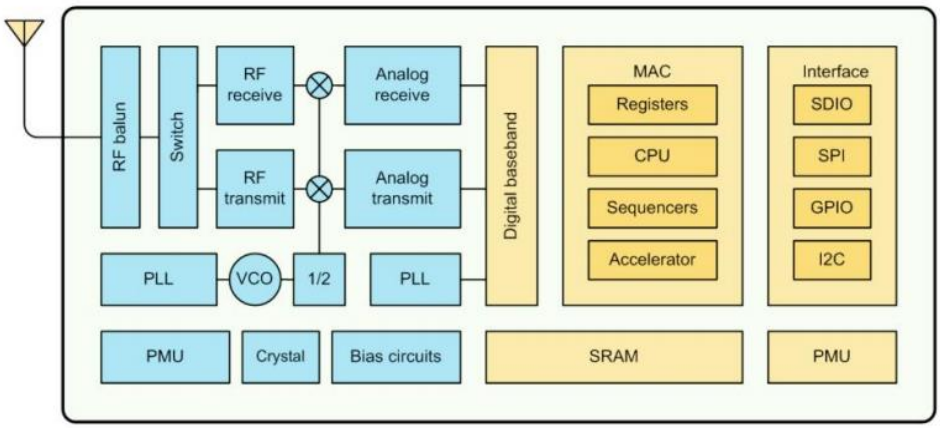
3.2. Esp8266 (Esp-12F)

Eap-12F wifi modul razvila je kompanija Ai-thinker. Esp-12F modul bazira se na Esp8266 čipu, odnosno na Tensilica L106 32-bitnom mikrokontroleru. Modul podržava takt od 80MHz, 160MHz, RTOS, integrirani WiFi te posjeduje integriranu antenu. Esp8266 u potpunosti podržava standardni IEEE802.11 b/g/n model i kompletan TCP/ IP snop protokola. Nudi cjelovito i samostalno rješenje za Wi-Fi umrežavanje, može se koristiti kao samostalni mikrokontroler, a alternativno može služiti kao Wi-Fi adapter. Bežični pristup internetu može se dodati bilo kojem dizajnu temeljenom na mikrokontroleru koristeći SPI, I2C ili UART sučelje.

Ovaj modul je proširen s 4 MB vanjskom SPI memorijom (flash memorija) za pohranu korisničkih programa. Ako je veći prostor za pohranu potreban, preferira se SPI memorija veće veličine. Teoretski gledano, može podržati kapacitet do 16 MB memorije. [29]

Slika 3.12: Esp-12F modul [29]

Slika 3.13: Esp-12F razvojna pločica [29]



Slika 3.14: Blok dijagram Esp-12F modula [29]

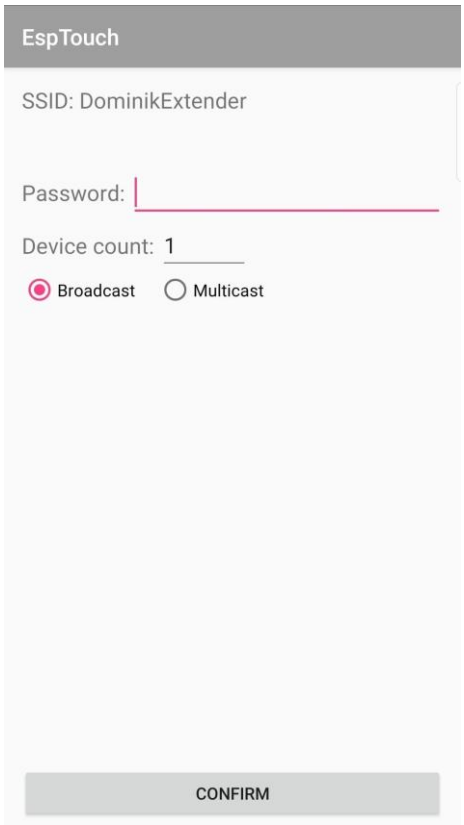
Neke od osnovnih specifikacija: [29]

- Integrirani 32-bitni procesor male potrošnje: Tensilica L106, 80 MHz
- 802.11 b/g/n
- 64 KB RAM za instrukcije i 96 KB RAM za podatke
- Interna 4MB SPI memorija
- 1 analogni ulaz (10-bita)
- Kompletan TCP/IP snop protokola
- Wi-Fi 2.4 GHz, WPA/WPA2
- Podržani protokoli: IPv4, TCP/UDP/HTTP/FTP
- 16 GPIO ulazna/izlaza
- podržava AP/STA/STA+AP načine rada

3.2.1. SmartConfig / ESP-TOUCH protokol

Vanjski modul (esp8266) potrebno je povezati na internet, a kako bi to korisniku olakšali koristi se navedeni protokol. Kako to Espressif naziva (tvrtka koja proizvodi esp8266 mikrokonrolere), ESP-TOUCH protokol. Objašnjavaju da je to protokol implementiran u SmartConfig tehnologiju kako bi se pomoglo korisnicima povezati uređaje (esp8266 i esp32) na internet uz pomoć pametnog telefona. Kako uređaj (esp8266) u početku nije spojen na mrežu, ne može poslati nikakav podatak na ostale uređaje. Sa ESP-TOUCH komunikacijskim protokolom, uređaj koji ima pristup WiFi tehnologiji, kao primjerice pametni telefon, može poslati nekoliko UDP paketa. Kad se esp8266 nalazi u Smart Config-u, on prima UDP pakete (u svakom paketu nalaze se informacije za SSID i Password). Esp8266 obavi importiranje tih podataka i pohranjuje ih u trajnu memoriju i na taj način upamti nove postavke mreže.

U završnom radu implementiran je ESP-TOUCH protokol koji se aktivira na način da korisnik mora držat pritisnuto tipkalo tri sekunde, a nakon pokretanja, unutar dvije minute potrebno je poslati podatke mreže korištenjem Android aplikacije. Zbog kompleksnosti samog programa, ovdje nije izdvojen dio koda koji pokreće ESP-TOUCH protokol. [30]



Slika 3.15: Sučelje Android aplikacije za unošenje mrežnih postavki

3.2.2. ST7735-TFT ekran

Kako bi se omogućio nadzor nad sustavom u slučaju kada internet nije dostupan implementiran je ST7735 1.8″ TFT ekran. Na ekranu se ispisuju sve bitne informacije koje se inače šalju na MQTT broker, stanje WiFi povezanosti, stanje povezanosti s MQTT brokerom. Ujedno se prikazuju i informacije kod pokretanja SmartConfig-a koji je ranije opisan.

ST7735 TFT je 1,8″ zaslon rezolucije 128×160 piksela i može prikazivati širok raspon boja (18-bitna boja, 262.144 nijanse). Zaslon koristi SPI protokol za komunikaciju i ima vlastiti međuspremnik za adresiranje piksela te se može koristiti sa svim vrstama mikrokontrolera. Za nadopunu zaslona dolazi i utor za SD karticu na koji se mogu spremiti bitmape i lako prikazati na zaslonu. [31]

Neke od osnovnih specifikacija: [31]

- Dijagonala 1,8″
- Rezolucija 128×160, 18-bitna (262.144) boja
- SPI komunikacija
- Ugrađeni microSD utor - koristi još dvije digitalne linije
- 5V kompatibilno! Koristite s 3,3 V ili 5 V logikom
- Ugrađeni 3.3V, 150mA
- 2 bijela LED pozadinska svjetla, spojena na tranzistor kako bi se moglo prigušiti pozadinsko svjetlo (PWM)
- Potrošnja struje najviše ovisi o pozadinskom osvjetljenju: kod maksimalnog pozadinskog osvjetljenja potrošnja je oko 50mA

Slika 3.16: Izgled sučelja – TFT ekran

Slika 3.17: SmartConfig informacija – TFT ekran

Kod programiranja korištene su *Adafruit_GFX* i *Adafruit_ST7735* biblioteke kako bi se olakšalo programiranje.

Korištene naredbe iz biblioteke:

-crtanje zaobljenog pravokutnika:

```
tft.drawRoundRect(x, y, širina, visina, polumjer, boja);
```

- crtanje ispunjenog zaobljenog pravokutnika:

```
tft.fillRoundRect(x, y, širina, visina, polumjer, boja);
```

-postavljanje kursora na željenu poziciju:

```
tft.setCursor(x, y);
```

-postavljanje boje teksta - prvo boja teksta zatim boja pozadine:

```
tft.setTextColor(boja teksta, boja pozadine);
```

-postavljanje veličine teksta – najmanji tekst 0, najveći tekst 20:

```
tft.setTextSize(od 0 do 20);
```

-ispis teksta ili vrijednosti na ekran:

```
tft.println(tekst ili varijabla za ispis);
```

Kako bi se olakšalo iscrtavanje sučelja na ekranu kreirana je funkciju koja prima parametre o tekstu koji se želi ispisati, bojama i mjernim jedinicama.

```
Display_Mode("Brzina ventilatora", 0xC817, "rpm",
             "Temp", ST7735_RED, 'C', " Vlaga", ST7735_BLUE, '%',
             "Tlak", 0x0E44, "hPa", "Osnovne informacije",
             ST7735_WHITE);
```

Programski blok 3.4: Pozivanje funkcije za iscrtavanje sučelja na ekranu

```
void Display_Mode(String name1, int mode_color, String mode_unit,
                  String box1_name, int box1_color, char box1_unit,
                  String box2_name, int box2_color, char box2_unit,
                  String box3_name, int box3_color, String box3_unit,
                  String box4_name, int box4_color)
{
  tft.drawRoundRect(0, 0, 160, 50, 5, mode_color);
  tft.fillRoundRect(0, 0, 160, 12, 5, mode_color);
  tft.setCursor(28, 2);
  tft.setTextColor(ST7735_WHITE);
  tft.setTextSize(1);
  tft.println(name1);
  tft.setCursor(115, 25)
  tft.setTextColor(mode_color, ST7735_BLACK);
  tft.setTextSize(2);    tft.println(mode_unit);
  tft.drawRoundRect(0, 51, 53, 30, 5, box1_color);
  tft.fillRoundRect(0, 51, 53, 12, 5, box1_color);
  tft.setCursor(15, 53);
  tft.setTextColor(ST7735_WHITE);
  tft.setTextSize(1);
  tft.println(box1_name);
  tft.setTextColor(box1_color, ST7735_BLACK);
  tft.setCursor(39, 66);
  tft.print(char(223));
  tft.setCursor(43, 68);
  tft.setTextSize(1);
  tft.println(box1_unit);
  tft.drawRoundRect(54, 51, 53, 30, 5, box2_color);
  tft.fillRoundRect(54, 51, 53, 12, 5, box2_color);
  tft.setCursor(60, 53);
  tft.setTextColor(ST7735_WHITE);
  tft.setTextSize(1);
  tft.println(box2_name);
  tft.setCursor(97, 68);
  tft.setTextColor(box2_color, ST7735_BLACK);
  tft.setTextSize(1);
  tft.println(box2_unit);
  tft.drawRoundRect(108, 51, 52, 30, 5, box3_color);
  tft.fillRoundRect(108, 51, 52, 12, 5, box3_color);
  tft.setCursor(124, 53);
  tft.setTextColor(ST7735_WHITE);
  tft.setTextSize(1);    tft.println(box3_name);
  tft.setCursor(140, 68);
  tft.setTextColor(box3_color, ST7735_BLACK);
  tft.setTextSize(1);    tft.println(box3_unit);
  tft.drawRoundRect(0, 82, 160, 45, 5, box4_color);
  tft.fillRoundRect(0, 82, 160, 12, 5, box4_color);
  tft.setCursor(24, 85);
  tft.setTextColor(ST7735_BLACK);
  tft.setTextSize(1);
  tft.println(box4_name);

}
```

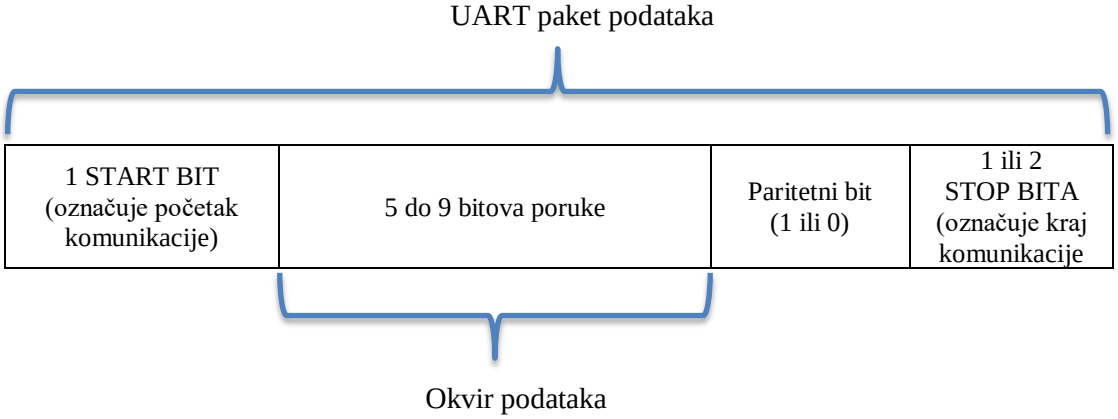
Programski blok 3.5: Funkcija za iscrtavanje sučelja

3.2.3. UART komunikacija

Kako bi se ostvarila komunikacija između dva mikrokontrolera, u našem slučaju između Arduina (atmega328P) i esp8266, korištena je serijska komunikacija UART. Prethodno je već spomenuto rješenje vezano uz naponske razine, a sad preostaje omogućiti komunikaciju.

UART (eng. *Universal Asynchronous Receiver/Transmitter*) je univerzalan asinkroni prijemnik/odašiljač, a radi se o integriranom elektroničkom sklopu koji se može nalaziti unutar mikrokontrolera ili u zasebnom čipu. Glavna uloga UART-a je prijenos i prijem serijskih podataka. Za UART komunikaciju potrebne su samo dvije linije za prijenos podataka: Tx za odašiljanje podataka i Rx za primanje podataka. Mikrokontroler koji odašilje podatke pretvara paralelni skup podataka u serijski i šalje ih kroz Tx liniju na Rx ulaz drugog mikrokontrolera, a kod zaprimanja podataka ponovo se pretvaraju u paralelan skup podataka. Spomenuto je da se radi o asinkronom prijenosu podataka što znači da u komunikaciji nema taktnog signala (eng. c*lcok signal*) kako bi se prijenos podataka između mikrokontrolera sinkronizirao. Umjesto taktnog signala, mikrokontroler koji odašilje podatke postavlja početni bit (eng. *start bit*) i završni bit (eng. *stop bit*) unutar paketa poruke kojeg šalje te na taj način primatelj zna kada komunikacija započinje i završava. Kad primatelj detektira početni bit, započinje primanje podataka unaprijed određenom frekvencijom odnosno brzinom prijenosa (eng. *baud rate*). Brzina prijenosa je izražena u broju prenesenih bitova u sekundi (bps). Kako bi komunikacija bila moguća oba mikrokontrolera moraju imati definiranu jednaku brzinu prijenosa, a ona može odstupati do 10%.

Podaci koji se šalju kroz UART smješteni su unutar paketa. Svaki paket sastoji se od bita koji označuje početak poruke, pet do devet bitova poruke (ovisno o tipu UART-a), opcionalni paritetni bit (eng. *parity bit*), jedan ili dva bita koji označuju kraj komunikacije. [32]



Tablica 3.5: Struktura UART paketa [32]

START BIT (bit koji označuje početak komunikacije) – u stanju kada nema komunikacije, Tx linija za odašiljanje podataka nalazi se na naponskoj razini logičke jedinice. Kako bi se započela komunikacija, linija za odašiljanje prelazi u naponsku razinu logičke nule u trajanju jednog ciklusa trajanja brzine prijenosa podataka. Mikrokontroler koji prima podatke na svom Rx ulazu detektira logičku nulu i počinje čitanje podataka koji se šalju. [32]

OKVIR PODATAKA – okvir podataka sadrži poruku koja se šalje. Može sadržavati pet do osam bitova ako se koristi paritetni bit, a u slučaju kad se paritetni bit ne koristi okvir podataka može sadržavati devet bitova. [32]

PARITETNI BIT – Paritetnim bitom osigurava se provjera točnosti prenesene poruke. Ako je paritetni bit postavljen na logičku nulu, to znači da se u okviru podataka nalazi paran broj logičkih jedinica. Ako je paritetni bit postavljen na logičku jedinici, to znači da se u okviru podataka nalazi neparan broj logičkih jedinica. Kad se paritetni bit podudara s podacima, UART može zaključiti da je zaprimljeni podatak točan, odnosno nije došlo do greške u komunikaciji. [32]

STOP BIT (bit koji označuje kraj komunikacije) – kako bi se označio kraj komunikacije, Tx linija za odašiljanje podataka prelazi u stanje logičke jedinice u trajanju dva ciklusa trajanja brzine prijenosa podataka. [32]

Da bi se lakše raspoznali podaci koji se razmjenjuju između Arduino platforme i esp8266 razvijen je algoritam. Poruke se šalju točno definiranim oznakama za raspoznavanje te znakovima za odvajanje podataka kako bi se mogli spremati u različite varijable.
Primjer poruke koju esp8266 prima na serijski port:
<sensors,25.21,59.45,1011,25541>
-znak „ < “ označava početak poruke koja nas zanima, a sve ostale poruke koje ne sadrže navedeni početni znak se ignoriraju
-podnaslov „ sensors “ označava tip parametara koji se šalju, a u nastavku su izmjerene veličine dobivene od senzora te se svaki podatak odvaja znakom zareza „ , “
-znak „ > “ označava kraj poruke, a sve ostale nakon navedenog znaka ne spremaju se u varijablu zaprimljene poruke
-poruka zapravo ima sljedeći sadržaj: <sensors,temperatura,vlaga,tlak,impulsTahometra>

```
const byte numChars = 42;
char receivedChars[numChars];
char tempChars[numChars];
char SerialString[numChars] = {0};
float SerialHum = 0.0;
float SerialTemp = 0.0;
float SerialPress = 0.0;
float SerialPulse = 0;
bool newDataAvailable = false;

void setup()
{
  Serial.begin(115200);
}
void loop()
{
    recvStartEndMarks();
    if (newDataAvailable == true) {
         strcpy(tempChars, receivedChars);  // Ova je privremena kopija
        potrebna za zaštitu izvornih podataka
        // jer strtok () koji se koristi u parseData () zamjenjuje
        zareze s \0
         parseData();
         showParsedData();
         newDataAvailable = false;
    }
}
void recvStartEndMarks() {
    static boolean receiveInProgress = false;
    static byte index = 0;
    char start_Marker = '<';
    char end_Marker = '>';
    char mess;

    while (Serial.available() > 0 && newDataAvailable == false)
{
        mess = Serial.read();

        if (receiveInProgress == true) {
            if (mess!= end_Marker) {
                receivedChars[index] = mess;
                index ++;
                if (index >= numChars) {
                    index = numChars - 1;
                }
            }
            else {
                receivedChars[index] = '\0'; // završavanje string-a
                receiveInProgress = false;
                index = 0;
                newDataAvailable = true;
            }
        }

        else if (mess == start_Marker) {
            receiveInProgress = true;}
    }}
```

```
void parseData()// raščlanjivanje podataka na dijelove
{
  char * strtokIndx;

  strtokIndx = strtok(tempChars, ",");// prvi dio poruke - string
  strcpy(SerialString, strtokIndx);  // kopiranje poruke u varijablu

  strtokIndx = strtok(NULL, ",");//nastavlja se gdje se u poruci stalo
  SerialTemp = atof(strtokIndx);

  strtokIndx = strtok(NULL, ",");//nastavlja se gdje se u poruci
  SerialHum = atof(strtokIndx); //konverzija podatka u float


  strtokIndx = strtok(NULL, ",");//nastavlja se gdje se u poruci
  SerialPress = atof(strtokIndx); //konverzija podatka u float

  strtokIndx = strtok(NULL, ",");//nastavlja se gdje se u poruci
  SerialPulse = atof(strtokIndx); //konverzija podatka u float

}
```

Programski blok 3.6: Algoritam za raščlanjivanje poruka

3.2.4. Obrada podataka

Kako esp8266 raspolaže s dovoljnom količinom resursa, on obavlja sve dodatne obrade podataka koje prima kako bi se Arduino (postojeći sustav) što manje opteretio. Neke od obrada podataka su: računanje brzine i frekvencije ventilatora, računanje tlaka u različitim mjernim jedinicama, zaokruživanje decimalnog broja na dvije decimale, formiranje JSON objekta.

Brzina i frekvencija ventilatora:

-podatak o ventilatoru koji dolazi na serijski port je zapravo podatak koliko je potrebno mikrosekundi da bi se rotor ventilatora okrenuo za pola kruga.

-kako bi se dobio podatak o frekvenciji prvo je mikrosekunde potrebno pretvoriti u sekunde te zatim izračunati frekvenciju koja se dijeli s dva jer ventilator daje dva impulsa za puni krug:

$$Hz = \frac{1000000}{vrijeme\ plovice\ kruga} \div 2$$

-za vrijeme punog kruga rotora potrebno je podatak pomnožiti sa dva:

$$Krug = vrijeme\ polovice\ kruga \times 2$$

-podatak o brzini vrtnje ventilatora u broju okretaja po minuti dobije se sljedećom formulom:

$$rpm = \frac{1000000}{vrijeme\ plovice\ kruga} \times 60$$

```
double rawdata = 0.0;
float freq = 0.0;
float rev = 0.0;
float rpm = 0.0;

void calcFanSpeed(float pulseDuration)
{

  if (pulseDuration != 0)
  {
    rawdata = 1000000 / pulseDuration;

    rev = pulseDuration * 2;
    rev = rounding(rev);

    freq = rawdata / 2;
    freq = rounding(freq);/

    rpm = rawdata / 2 ;
    rpm = rpm * 60;
    rpm = rounding(rpm);
  }
  if (pulseDuration == 0)
  {
    rev = 0;
    freq = 0;
    rpm = 0;
  }
}
```

Programski blok 3.7: Proračun brzine vrtnje ventilatora

Računanje tlaka u različitim mjernim jedinicama:

Kako bi se korisniku pružilo uvid u različite mjerne jedinice tlaka zraka, esp8266 odrađuje konverziju.

hPa→bar: $bar = \frac{hPa}{1000}$

hPa→psi: $psi = hPa \times 0.0145037738$

hPa→mmHg: $mmHg = hPa \times 0.7500615781$

```
float bar = 0.0;
float psi = 0.0;
float mmhg = 0.0;

void calcPressure(float x)
{
  bar = x / 1000;
  psi = x * 0.0145037738;
  psi = rounding(psi);
  mmhg = x * 0.7500615781;
  mmhg = rounding(mmhg);
}
```

Programski blok 3.8: Konverzija mjernih jedinica

Zaokruživanje decimalnog broja na dvije decimale:

Kako bi se maknuo nepotreban broj decimalnih mjesta u float tipu varijabli koriste se trikovi programiranja.

-decimalni broj pomnoži se sa 100: $37{,}66666 \times 100 = 3766{,}66$

-dobivenom decimalnom broju pribroji se 0.5: $3766{,}66 + 0.5 = 3767{,}16$

-varijabla float pretvori se u varijablu tipa integer: 3767,16 postaje 3767

-dobiven integer podijeli se sa 100 i vrati u float: $3767 \div 100 = 37{,}67$

-dobije se broj s dva decimalna mjesta

```
float rounding(float var)
{
  float value = (int)(var * 100 + .5);
  return (float)value / 100;
}
```

Programski blok 3.9: Funkcija za zaokruživanje decimala

Formiranje JSON objekta:

Svi podaci koji se šalju na MQTT broker šalju se i primaju u JSON formatu zbog lakšeg raščlanjivanja unutar Android aplikacije ali i kod zaprimanja poruka na esp8266.

```
#include <ArduinoJson.h>
String jsonData;


void jsonSend()
{
  if (strcmp(SerialString, "sensors") == 0)
  {
    const size_t capacity = JSON_OBJECT_SIZE(1) + JSON_OBJECT_SIZE(9);
    DynamicJsonDocument doc(capacity);

    JsonObject sensors = doc.createNestedObject("sensors");
    sensors["temp"] = SerialTemp;
    sensors["hum"] = SerialHum;
    sensors["pressure"] = SerialPress;
    sensors["bar"] = bar;
    sensors["psi"] = psi;
    sensors["mmhg"] = mmhg;
    sensors["rpm"] = rpm;
    sensors["freq"] = freq;
    sensors["rev"] = rev;

    serializeJson(doc, jsonData);
  }
}
```

Programski blok 3.10: Kreiranje JSON objekta

```
{
  sensors :
  {
    temp : 23.61
    hum : 37.88
    pressure : 1020.16
    bar : 1.02016
    psi : 14.8
    mmhg : 765.18
    rpm : 6334.46
    freq : 105.57
    rev : 9472
  }
}
```

Programski blok 3.11: Izgled kreiranog JSON objekta

3.2.5. Esp8266 – MQTT klijent

U ovom završnom radu prikazano je kako funkcionira MQTT klijent koji je prethodno opisan. Esp8266 pretplaćuje se na temu gdje prima i šalje podatke na MQTT broker te je time ujedno objavitelj (eng. *publisher*) i pretplatitelj (eng. *subscriber*). Za povezivanje s MQTT brokerom korištena je *PubSubClient.h* biblioteka koja omogućava slanje i primanje poruka. Prije pozivanja funkcija iz biblioteke potrebno je definirati varijable koje sadrže podatke o MQTT klijentu na koji se korisnik želi spojiti. Varijable koje se pružaju biblioteci su: informacije o serveru, port servera, korisničko ime za autentifikaciju, ID klijenta, šifra MQTT klienta, tema na koju se korisnik pretplaćuje.

Klijent može imati samo jednu vezu s istim brokerom. Bitno je napomenuti kako broker koristi ID klijenta za njegovu identifikaciju i on mora biti jedinstven. Kako bi se postiglo da svaki Esp8266 ima jedinstven ID, u dio ID-a dodaje se MAC adresa Esp8266 uređaja.

Korištene naredbe iz biblioteke:

-povezivanje klijenta na MQTT broker:

```
client.connect (ID klijenta, [username, šifra], [will Tema,
will QoS, will Retain, will poruka], [čista sesija])
```

-provjeravanje povezanosti klijenta sa MQTT brokerom (vraća *true* ili *false*):

```
client.connected()
```

-pretplaćivanje klijenta na željenu temu:

```
client.subscribe(tema,QoS)
```

-slanje poruke na MQTT broker:

```
client.publish(Tema, poruka, [duljina poruke], [retained])
```

-funkcija za primanje poruke:

```
void callback(char* tema, byte* poruka, unsigned int duljina)
```

Bitno je napomenuti da funkcija za primanje i slanje poruke koristi istu memorijsku lokaciju, a samim time primljene poruke potrebno je spremati u dodatnu varijablu kako bi se zadržale prilikom slanja drugih poruka na MQTT broker.

```
if (wifiConnected == true)
    {

      if (!client.connected())
      {
        if (client.connect(MACadr.c_str(), mqttUser, mqttPassword ))
        {
          client.subscribe(topic);
        }
      }
```

Programski blok 3.12: Povezivanje na MQTT broker

```
//primanje poruke

void callback(char* topic, byte* payload, unsigned int length)
{
  poruka = "";
  for (int i = 0; i < length; i++)
 {
    poruka += ((char)payload[i]);
  }
  newMQTTmess = true;

}

//slanje poruke

client.publish(topic, (char*) jsonData.c_str(), 0);
```

Programski blok 3.13: Primanje i slanje poruke sa MQTT brokera

4. Android aplikacija

Kao dio praktičnog dijela završnog rada izrađena je i Android MQTT mobilna aplikacija korištenjem *Paho* biblioteke za nadzor sustava i njegovo upravljanje. Android aplikacija pruža korisniku nadgledanje i upravljanje sustavom bez obzira gdje se trenutno nalazi, a sve što je potrebno je Android operacijski sustav koji se može nalaziti na mobitelu ili tabletu te internet konekcija. Svi podaci koje aplikacija prima s MQTT brokera prikazuju se grafički korisniku radi lakšeg nadgledanja te je omogućeno i upravljanje jednim dijelom sustava.

Aplikacija je razvijana u razvojnom okruženju Android Studio koje se sastoji od nekoliko bitnih dijelova za razvoj aplikacije poput Gradle, AndroidManifest, Activity, Resursi.

Gradle – Gradle je zadužen za „građenje“ same aplikacije. Odnosno, koristi se za kreiranje aplikacije (spajanje kodova dizajna i Java koda logike, kompajliranje i opcija za otklanjanje pogrešaka u kodu) kako bi je mogli pokretati na Android sustavu.

AndroidManifest - Svaka Android aplikacija, odnosno projekt, sadrži AndroidManifest. To je jedan od najvažnijih dijelova projekta gdje se nalaze bitne stavke o samoj aplikaciji, odnosno unutar AndroidManifest-a se definira struktura, zahtjevi, dozvole koje su potrebne za rad aplikacije.

Activity – Activity se može usporediti sa Windows sustavom gdje se pokreću programi unutar samog operativnog sistema, a ti programi stvaraju vizualne prozore s grafičkim sučeljem. Activity je zapravo program unutar Android sustava te omogućava prikaz različitih korisničkih interfejsa. MainActivity je naziv za glavni dio programa te se unutar njega piše program aplikacije u Java ili Kotlin programskim jezikom.

Resursi - Unutar Android projekta postoji mjesto gdje se spremaju sve korištene slike, ikone i dizajn aplikacije. Resursi je zapravo naziv mape koja sadrži sve dodatne grafičke prikaze koji se dodaju u aplikaciju.

4.1. Paho biblioteka

Paho biblioteka podržana na Android platformi omogućuje povezivanja Android mobilne aplikacije na MQTT broker. Konekcija se ostvaruje s MQTT brokerom te se izvršava i održava u pozadini same aplikacije i tako omogućuje održavanje veze u trenutku kad Android aplikacija odrađuje različite dodatne aktivnosti. Takva biblioteka potrebna je kako bi se moglo kontinuirano primati i slati MQTT poruke na broker te kako bi se ostvarila stabilnost same aplikacije.

4.1.1. Inicijalizacija

Kako bi Paho biblioteka postala dio mobilne aplikacije prvo je potrebno dodati samu biblioteku u *Gradle* koji je dio Androida.

```
dependencies

{
Implementation 'org.eclipse.paho:org.eclipse.paho.client.mqttv3:1.1.0'
implementation 'org.eclipse.paho:org.eclipse.paho.android.service:1.1.1'
}
```

Programski blok 4.1: Paho Gradle

4.1.2. Povezivanje na broker

Spomenuto je da *Paho* biblioteka odrađuje povezivanje i održavanje veze u pozadini, a kako bi to bilo moguće Android sustav mora pokrenuti pozadinski servis. Unutar AndroidManifest.xml dodaje se sljedeći kod:

```
<!-- Mqtt Service -->
 <service android:name="org.eclipse.paho.android.service.MqttService">
 </service>
```

Programski blok 4.2: Omogućavanje pokretanja MQTT servisa

Kako bi se MQTT servis pokrenuo potrebne su još i dodatne dozvole:

```
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"
/>
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

Programski blok 4.3: Dozvole za pokretanje MQTT servisa

Nakon pokretanja servisa s dodatnim dozvolama unutar Java programa koji je odgovoran za cijelu aplikaciju potrebno je pokrenuti postupak ostvarivanja konekcije na MQTT broker. Prije pozivanja metode iz biblioteke potrebno je definirati varijable koje sadrže podatke o MQTT klijentu na koji se korisnik želi spojiti. Varijable koje se pružaju biblioteci su: informacije o serveru, port servera, korisničko ime za autentifikaciju, ID klijenta, šifra MQTT klienta, tema na koju se pretplaćuje (zapravo vrlo slično kako to zahtjeva i Esp8266 *PubSubClient.h* biblioteka)

```
public void connect() {
      mqttAndroidClient = new
      MqttAndroidClient(getApplicationContext(), MQTTHOST, clientId);

        MqttConnectOptions mqttConnectOptions = new
        MqttConnectOptions();
        mqttConnectOptions.setKeepAliveInterval(15);

        mqttConnectOptions.setCleanSession(false);
        mqttConnectOptions.setUserName(USERNAME);
        mqttConnectOptions.setPassword(PASSWORD.toCharArray());


        try {

        mqttAndroidClient.connect(mqttConnectOptions, null, new
        IMqttActionListener() {

                @Override
                public void onSuccess(IMqttToken asyncActionToken) {

                    Log.w("Mqtt", "connect succeed");

                    subscribeToTopic();
                }

                @Override
                   public void onFailure(IMqttToken asyncActionToken,
                   Throwable exception) {
                   Log.w("Mqtt", "Failed to connect to: " + MQTTHOST +
                   exception.toString());
                }
            });


        } catch (MqttException ex) {
            ex.printStackTrace();
        }
```

Programski blok 4.4: Metoda za povezivanje na MQTT broker

4.1.3. Pretplata na temu

Pretplaćivanje na temu može se izvršiti pozivanjem *mqttAndroidClient.subscribe* metode koja prima parametre o nazivu teme na koju se korisnik želi pretplatiti te kvalitetu usluge (QoS). Ujedno metoda provjerava uspješnost pretplate, a u slučaju da dođe do pogreške javlja povratnu informaciju.

```
public void subscribeToTopic() {
      try {
            mqttAndroidClient.subscribe(topicStr, 0, null, new
            IMqttActionListener() {
                  @Override
                  public void onSuccess(IMqttToken asyncActionToken) {
                      Log.w("Mqtt", "Subscribed!");

                  }

                  @Override
                   public void onFailure(IMqttToken asyncActionToken,
                   Throwable exception) {
                      Log.w("Mqtt", "Subscribed fail!");
                  }
             });

        } catch (MqttException ex) {
            System.err.println("Exceptionst subscribing");
            ex.printStackTrace();
        }
    }
```

Programski blok 4.5: Metoda za pretplatu na željenu temu

4.1.4. Objava poruke

MqttAndroidClient dozvoljava da se poruke šalju na MQTT broker pozivanje publishMessage metode. *MqttAndroidClient* ne dozvoljava slanje poruka ako klijent nije spojen i pojavit će se greška prilikom pokušaja slanja. Iz navedenog razloga dodatno se provjerava povezanost klijenta i brokera prije slanja poruke. U slučaju da klijent nije spojen prvo se uspostavlja nova konekcija.

```
public void publishMessage(String payload) {
    try {
        if (!mqttAndroidClient.isConnected()) {
            mqttAndroidClient.connect();
        }

        MqttMessage message = new MqttMessage();
        message.setPayload(payload.getBytes());
        message.setQos(0);
        mqttAndroidClient.publish(topicStr, message,null, new
        IMqttActionListener() {
            @Override
            public void onSuccess(IMqttToken asyncActionToken) {
                Log.w("Mqtt", "publish succeed!");
            }

            @Override
            public void onFailure(IMqttToken asyncActionToken, Throwable
            exception) {
                Log.w("Mqtt", "publish failed!");
            }
        });
    } catch (MqttException e) {
        Log.w("Mqtt", e.toString());
        e.printStackTrace();
    }
}
```

Programski blok 4.6: Metoda za objavu poruke na MQTT broker

4.1.5. Primanje poruke sa brokera

Primanje poruke vrši se, kako se to naziva, *callback* metodom. Radi se o metodi koju korisnik nema potrebe pozivati već se podaci popunjavaju u varijable čim je podatak dostupan, a tu radnju izvršava pozadinski proces. Takva metoda se upotrebljava kod MQTT protokola iz razloga što klijent ne može znati kad će nova poruka stići, a neprestano pozivanje klasične metode može zakasniti i propustiti novu poruku.

```
mqttAndroidClient.setCallback(new MqttCallback() {
    @Override
    public void connectionLost(Throwable cause) {Log.w("Mqtt", "connection
    lost");
            }
    @Override
    public void messageArrived(String topic, MqttMessage message) throws
    Exception {
                try {
                    JSONObject obj=new JSONObject(new
                   String(message.getPayload()));

                    JSONObject sensors = obj.getJSONObject("sensors");

                    temp = sensors.getString("temp");
                    hum = sensors.getString("hum");
                    pressure = sensors.getString("pressure");
                    rpm = sensors.getString("rpm");
                    barData = sensors.getString("bar");
                    psiData= sensors.getString("psi");
                    mmhgData = sensors.getString("mmhg");
                    freqData= sensors.getString("freq");
                    revData = sensors.getString("rev");
                 } catch (JSONException e) {
                    e.printStackTrace();
                }

            }

            @Override
            public void deliveryComplete(IMqttDeliveryToken token) {
                Log.w("Mqtt", "msg delivered");
            }
        });
```

Programski blok 4.7: Callback metoda za zaprimanje poruke

4.2. Realizacija aplikacije

Nakon uspješne realizacije povezivanja s MQTT brokerom, korištenjem *Paho* biblioteke, preostaje izrada samog dizajna aplikacije kao i programske logike koja će omogućiti korisniku vizualno nadgledanje te samu interakciju sa procesom.

4.2.1. Grafičko sučelje

Kako bi se razvilo vizualno korisničko sučelje koristi se XML jezik. XML (eng. *Extensible Markup Language*) je opisni jezik za stvaranje grafičkog sučelja Android aplikacije. Za lakše razumijevanje može ga se usporediti s HTML-om koji ima istu ulogu kod izrade web stranica. XML kodom definiraju se različiti vizualni objekti kao i njihova hijerarhija te pozicije na kojima se nalaze kako bi se aplikacija mogla pokretati na uređajima različitih rezolucija.

Svaki vizualni objekt koji će biti u direktnoj interakciji s korisnikom (primjerice gumb) mora sadržavati svoj ID kako bi ga mogli povezati s Java programskim kodom i izvršiti radnju kad Android sustav prepozna da je došlo do interakcije korisnika. [33]

Primjeri XML koda:

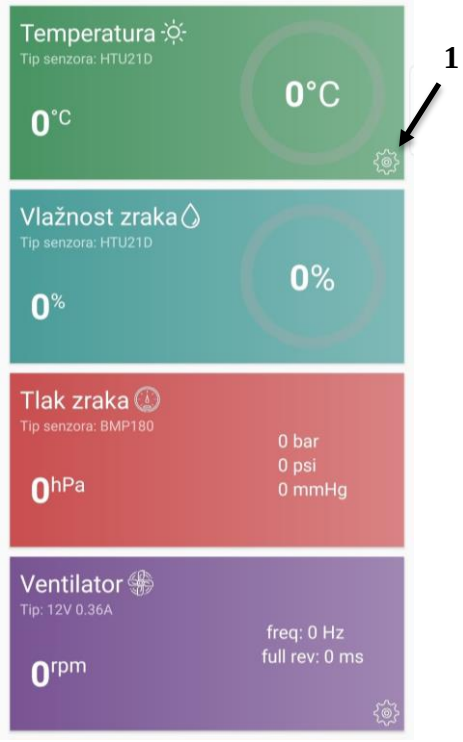
-XML kod za kreiranje gumba s ID oznakom: settingsTemp, definiranom visinom i širinom, marginama

```
<Button
    android:id="@+id/settingsTemp"
    android:layout_width="25dp"
    android:layout_height="25dp"
    android:layout_marginBottom="5dp"
    android:layout_marginEnd="5dp"
    android:layout_gravity="bottom|right"
    android:background="@drawable/settings"/>
```

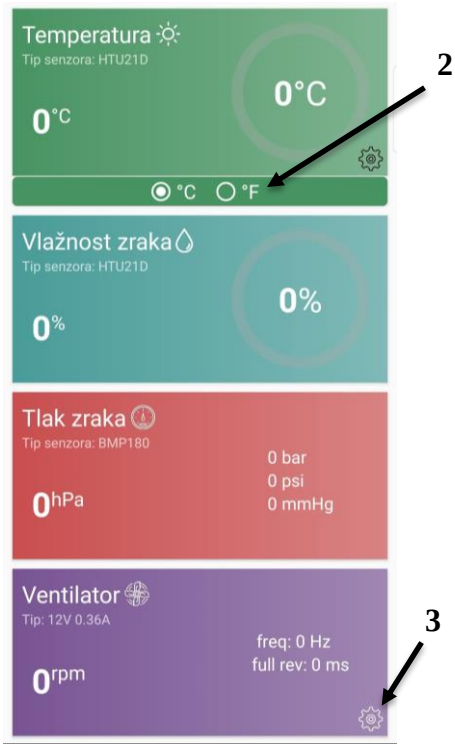
-XML kod za kreiranje tekstualnog prikaza s ID oznakom progresstxtTemp koji će na ekranu ispisati „PROBA“, definiranom visinom i širinom, veličinom samog teksta, bojom pozadine (u ovom slučaju prozirna pozadina) te bojom teksta kao i stilom (u ovom slučaju **bold**).

```
<TextView
    android:id="@+id/progresstxtTemp"
    android:background="#00000000"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="35dp"
    android:textColor="#FFFFFF"
    android:text="PROBA"
    android:textStyle="bold"/>
```

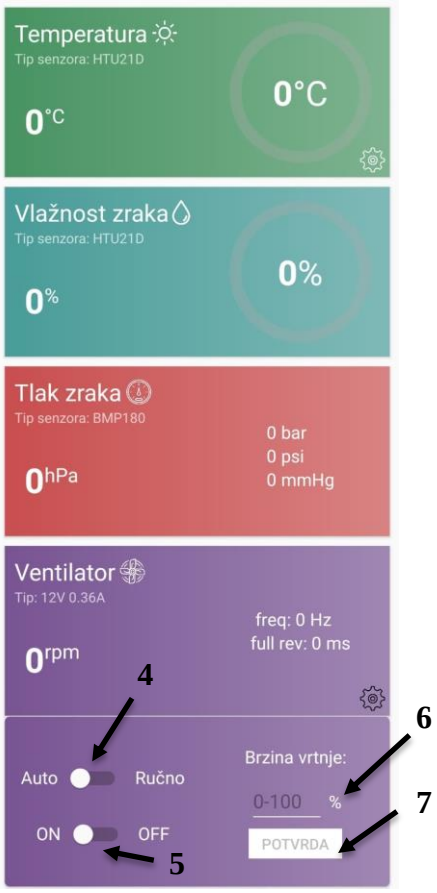
Na ovakav način kreirano je cijelo grafičko sučelje aplikacije, a svaki element posjeduje vlastiti ID kako bi se mogli mijenjati njegovi parametri unutar Java koda ovisno o podacima koji se primaju s MQTT brokera.



Slika 4.2: Grafičko sučelje



Slika 4.1:Grafičko sučelje– postavke temperature



Slika 4.3: Grafičko sučelje – postavke ventilatora

Broj	Funkcija
1	Pritiskom na tipku korisniku se otvara/zatvara dodatni meni za postavke temperature
2	Korisnik može odabrati prikaz temperature u stupnjevima Celzija ili Fahrenheita
3	Pritiskom na tipku korisniku se otvara/zatvara dodatni meni za postavke ventilatora
4	Pritiskom na tipku korisnik bira automatski ili ručni način rada ventilatora
5	Pritiskom na tipku korisnik može ugasiti ili upaliti ventilator (pod uvjetom da je ručni način rada prethodno odabran)
6	Korisnik može zadati brzinu vrtnje ventilatora od 0 do 100 % (pod uvjetom da je ručni način rada prethodno odabran)
7	Pritiskom na tipku korisnik potvrđuje željenu brzinu ventilatora (pod uvjetom da je ručni način rada prethodno odabran)

Tablica 4.1: Funkcije aplikacije

4.2.2. Programska logika

Povezivanje grafičkih objekata s programom:

Programska logika koja omogućuje vizualnu promjenu grafičkog sučelja (ovisno o dolaznim MQTT podacima) i interakciju s korisnikom programirana je Java programskim jezikom. Bez programske logike samo sučelje neće prikazivati podatke jer aplikacija ne zna gdje koji podatak pripada. Iz tog razloga potrebno je povezati ID grafičkog objekta s programom. U samom početku programa unutar metode *onCreate()* potrebno je pozivati metode *findViewById(R.id."ID objkta")* kako bi se programski dio povezao s grafičkim sučeljem.

```
TextView temperatura, vlaga, tlak, rpmfan, bar, psi, mmhg, freq, rev,
progresstxtTemp, progresstxtHum, progressUnit, tempUnit;
    ProgressBar progressHum, progressTemp;
    Button settingsTemp, settingsVent, potvrda;
    RadioButton celsiusCheck, farenhCheck;
    Switch switch1, switch2;
    EditText brzinaText;

    CardView cardTemp, cardVent;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);


        temperatura=findViewById(R.id.temperatura);
        vlaga= findViewById(R.id.vlaga);
        tlak= findViewById(R.id.tlak);
        rpmfan=  findViewById(R.id.rpmfan);
        bar= findViewById(R.id.bar);
        psi=  findViewById(R.id.psi);
        mmhg=  findViewById(R.id.mmhg);
        freq=  findViewById(R.id.freq);
        rev=  findViewById(R.id.rev);
        progressUnit=findViewById(R.id.progressUnit);
        tempUnit=findViewById(R.id.tempUnit);
        progresstxtTemp= findViewById(R.id.progresstxtTemp);
        progresstxtHum= findViewById(R.id.progresstxtHum);

        progressTemp=findViewById(R.id.progressTemp);
        progressHum=findViewById(R.id.progressHum);

        settingsTemp=findViewById(R.id.settingsTemp);
        settingsTemp.setOnClickListener(this);
        settingsVent=findViewById(R.id.settingsVent);
        settingsVent.setOnClickListener(this);

        potvrda=findViewById(R.id.potvrda);
        potvrda.setOnClickListener(this);
        potvrda.setEnabled(false);
```

```
        celsiusCheck=findViewById(R.id.celsiusCheck);
        celsiusCheck.setOnClickListener(this);

        farenhCheck=findViewById(R.id.farenhCheck);
        farenhCheck.setOnClickListener(this);

        brzinaText=findViewById(R.id.brzinaText);
        brzinaText.setFocusable(false);

        cardTemp = findViewById(R.id.cardTemp);
        cardTemp.setVisibility(View.GONE);
        cardVent = findViewById(R.id.cardVent);
        cardVent.setVisibility(View.GONE);

        switch1=findViewById(R.id.switch1);
        switch1.setClickable(true);
}
```

Programski blok 4.8: Povezivanje grafičkih objekata sa programom

Interakcija korisnika:

Korisniku su putem grafičkog sučelja ponuđene tipke koje može pritisnuti i izvršiti neku zadaću (tablica 4.1). Aplikacija mora prepoznati da je korisnik pritisnuo određenu tipku i prema tome izvršiti dio koda koji je vezan uz tu tipku, a to se ostvaruje pomoću metode *onClick()* i grananja *switch-case*.

```
    public void onClick(View v)
    {
        switch (v.getId())
        {
            case R.id.settingsTemp:
                colapseExpandTemp();
                break;

            case R.id.settingsVent:
                colapseExpandVent();
                break;

            case R.id.celsiusCheck:
                farenhCheck.setChecked(false);
                celsiusToF(temp);
                break;

            case R.id.farenhCheck:
                celsiusCheck.setChecked(false);
                celsiusToF(temp);

                break;

            case R.id.switch1:
                checkAutoRuc();
                break;
```

```
            case R.id.potvrda:

                if(TextUtils.isEmpty(brzinaText.getText().toString()))
                {
                Toast.makeText(this, "Upišite vrijednost 1-100 ",
                Toast.LENGTH_SHORT).show();
                    return;
                }

                speed=brzinaText.getText().toString();
                 int provjera
                 =Integer.parseInt(brzinaText.getText().toString());

                if(provjera<0)
                {
                    speed="0";
                    brzinaText.setText("0");
                }
                if(provjera>100)
                {
                    speed="100";
                    brzinaText.setText("100");
                }


                Auto="0";
                ventilatorSEND();
                break;

        }
    }
```

Programski blok 4.9: Zadaća koja se izvršava ovisno o pritisnutoj tipki

Metoda `colapseExpandTemp()`:

Metoda se poziva pritiskom na tipku *settingsTemp* (tablica 4.1, funkcija 1). Izvršava se dio koda koji prikazuje ili sakriva dodatnu karticu u kojoj se nalaze mogućnosti promjene mjerne jedinice temperature.

```
void colapseExpandTemp()
 {
   TransitionManager.beginDelayedTransition(cardTemp,new
   AutoTransition());
      if(cardTemp.getVisibility()==View.GONE)
      {
       settingsTemp.setBackgroundResource(R.drawable.settingspres);
        cardTemp.setVisibility(View.VISIBLE);
      }
      else
      {
          settingsTemp.setBackgroundResource(R.drawable.settings);
          cardTemp.setVisibility(View.GONE);
      }
 }
```

Programski blok 4.10: Metoda za otvaranje/zatvaranje postavki temperature

Metoda `colapseExpandVent()`:

Metoda se poziva pritiskom na tipku *settingsVent* (tablica 4.1, funkcija 3). Izvršava se dio koda koji prikazuje ili sakriva dodatnu karticu u kojoj se nalaze mogućnosti za upravljanje ventilatorom.

```
void colapseExpandVent()
{
   TransitionManager.beginDelayedTransition(cardVent,new
   AutoTransition());
      if(cardVent.getVisibility()==View.GONE)
      {
         settingsVent.setBackgroundResource(R.drawable.settingspres);
         cardVent.setVisibility(View.VISIBLE);
      }
      else
      {
          settingsVent.setBackgroundResource(R.drawable.settings);
          cardVent.setVisibility(View.GONE);
      }
}
```

Programski blok 4.11:Metoda za otvaranje/zatvaranje postavki ventilatora

Metoda `celsiusToF()`:

Metoda provjerava opciju koju je korisnik odabrao (Celzijus ili Fahrenheit) te se vrši konverzija temperature. Ujedno odrađuje se postavljanje novih mjernih jedinica na grafičkom sučelju kao i animacija „punjenja“ grafa za prikaz temperature.

```
void celsiusToF(String temp)
    {
        if(celsiusCheck.isChecked())
        {
            progressUnit.setText("°C");
            tempUnit.setText("°C");

            float floatTemp = Float.parseFloat(temp);
            int tempProgress = Math.round(floatTemp);

            progressTemp.setMax(50);
            ObjectAnimator animationprogressTemp =
ObjectAnimator.ofInt(progressTemp, "progress", lastTemp, tempProgress);
            animationprogressTemp.setDuration(1000);
            animationprogressTemp.setInterpolator(new
DecelerateInterpolator());
            animationprogressTemp.start();
            progresstxtTemp.setText(String.valueOf(tempProgress));
            temperatura.setText(temp);
            lastTemp=tempProgress;
        }

        if(farenhCheck.isChecked())
        {
            progressUnit.setText("°F");
            tempUnit.setText("°F");

            float floatTemp = Float.parseFloat(temp);
            float ffloattemp = floatTemp*9/5+32;
            int tempProgress = Math.round(ffloattemp);

            progressTemp.setMax(122);
            ObjectAnimator animationprogressTemp =
ObjectAnimator.ofInt(progressTemp, "progress", lastTemp, tempProgress);
            animationprogressTemp.setDuration(1000);
            animationprogressTemp.setInterpolator(new
DecelerateInterpolator());
            animationprogressTemp.start();

            progresstxtTemp.setText(String.valueOf(tempProgress));
            String s = Float.toString(ffloattemp);
            temperatura.setText(s);

            lastTemp=tempProgress;
        }
    }
```

Programski blok 4.12: Metoda za pretvorbu mjernih jedinica temperature

Metoda `checkAutoRuc()`:

Metoda koja se izvršava pritiskom tipke *switch1* (tablica 4.1, funkcija 4). Izvršava se dio koda koji omogućuje unašanje parametara, isključivanje i uključivanje ventilatora u slučaju kada je odabran ručni način rada. U slučaju kada je odabran automatski način rada, sve postavke vračaju se u početno stanje na grafičkom sučelju.

```
void checkAutoRuc()
{
    if(switch1.isChecked())
    {
        switch2.setClickable(true);
        potvrda.setEnabled(true);
        brzinaText.setFocusableInTouchMode(true);
    }

    else
    {
        switch2.setClickable(false);
        switch2.setChecked(false);
        potvrda.setEnabled(false);
        brzinaText.setFocusable(false);

        Auto="1";
        speed="50";
        ventilatorSEND();
    }
}
```

Programski blok 4.13: Metoda ručni/automatski način rada ventilatora

Tipka *POTVRDA:*

Pritiskom na tipku *POTVRDA* (tablica 4.1, funkcija 7) provjerava se tekstualno polje (tablica 4.1, funkcija 6). U slučaju da je tekstualno polje prazno, javlja se obavijest kako polje mora sadržavati upisan podatak od 0 do 100. U slučaju da je u polje upisan broj manji od 0, u polje se automatski upisuje minimalni broj, odnosno 0. U slučaju da je u polje upisan broj veći od 100, u polje se automatski upisuje maksimalan broj, odnosno 100. Pritiskom na tipku ne poziva se metoda već se unutar smog *switch-case* grananja izvršava dio koda.

```
case R.id.potvrda:

     if(TextUtils.isEmpty(brzinaText.getText().toString()))
     {
         Toast.makeText(this, "Upišite vrijednost 1-100 ",
Toast.LENGTH_SHORT).show();
          return;
     }

      speed=brzinaText.getText().toString();
      int provjera =Integer.parseInt(brzinaText.getText().toString());

      if(provjera<0)
      {
        speed="0";
        brzinaText.setText("0");
      }
      if(provjera>100)
        {
          speed="100";
          brzinaText.setText("100");
        }
          Auto="0";
          ventilatorSEND();
          break;
```

Programski blok 4.14: Metoda tipke potvrda

Metoda `ventilatorSEND()`:

Svi podaci koji se šalju na MQTT broker šalju se i primaju u JSON formatu zbog lakšeg raščlanjivanja unutar Esp8266 programa. Navedena metoda stvara JSON objekt, s postavljenim parametrima ventilatora, koji se zatim šalje na MQTT broker.

```
    void ventilatorSEND()
    {
        JSONObject ventilator = new JSONObject();
        try {
            ventilator.put("Auto", Auto);
            ventilator.put("ONOFF", ONOFF);
            ventilator.put("speed", speed);

        } catch (JSONException e) {

            e.printStackTrace();
        }
        JSONObject studentsObj = new JSONObject();
        try {
            studentsObj.put("ventilator", ventilator);

        } catch (Exception e) {
            e.printStackTrace();
        }
        String jsonStr = studentsObj.toString();
        publishMessage(jsonStr);
    }
```

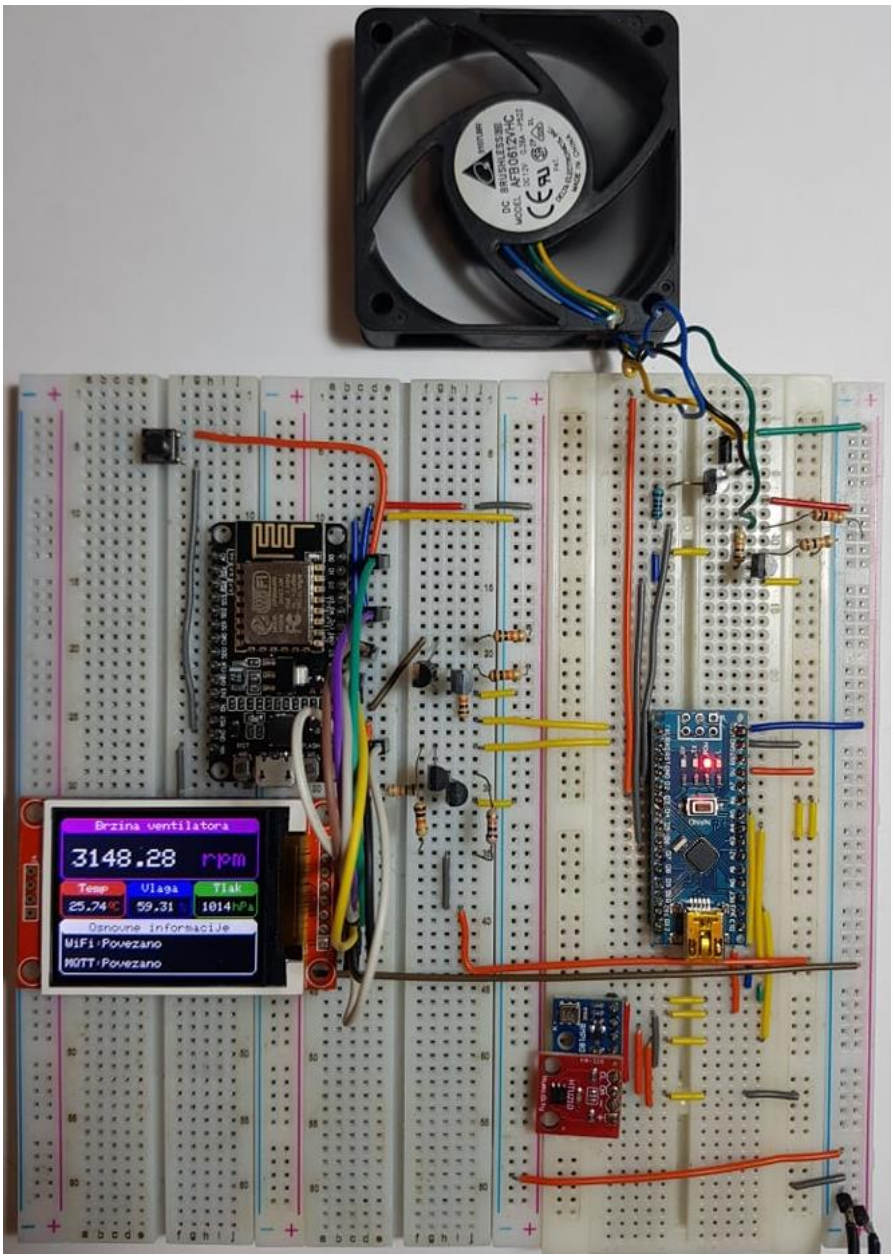
Programski blok 4.15: Metoda za kreiranje JSON objekta

```
{
  ventilator :
  {
    Auto   : 0
    ONOFF : 1
    speed : 94
  }
}
```

Programski blok 4.16: Izgled kreiranog JSON objekta

5. Testiranje sustava

Na kraju izrade praktičnog dijela rada testirane su sve funkcije. Aplikacija i sustav rade stabilno i bez zagušenja u komunikaciji. U mobilnoj aplikaciji mogu se pratiti parametri koji izvorno potječu iz senzora do Arduino razvojne platforme, a zatim se šalju putem UART komunikacije na Esp8266 mikrokontroler. Esp8266 mikrokontroler šalje podatke na MQTT broker u JSON formatu, a Android aplikacija podatke uspješno raščlanjuje u različite varijable i prikazuje korisniku. Upravljanje sustavom također je uspješno testirano, a podaci se šalju iz mobilne aplikacije na MQTT broker prema Esp8266 mikrokontroleru zatim putem UART komunikacije na Arduino razvojnu platformu koja upravlja ventilatorom.



Slika 5.1: Izgled i testiranje sustava

Slika 5.2: Mobilna aplikacija, temperatura - Celzijus

Slika 5.3: Mobilna aplikacija, temperatura - Fahrenheit

Slika 5.5: Mobilna aplikacija, ručni način - ventilator ugašen

Slika 5.4: Mobilna aplikacija, ručni način - ventilator na 100% brzine

6. Zaključak

IoT sustavi olakšavaju korisniku život te industrijama rad s manje gubitaka. Potrebno je razviti i svijest kako se korisnik ne bi trebao u potpunosti oslanjati na takve sustave. Ponekad mogu biti nesigurni ili može doći do kvara što je izvan kontrole čovjeka. Vrlo je bitno primijeniti sve današnje standarde zaštite podataka na internetu kako bi se korisnicima i industrijama osigurala sigurnost. Sigurnost je možda i najveći problem IoT sustava iz razloga što ih je na tržištu sve više, a nitko ne može garantirati da je sustav u potpunosti siguran i zaštićen. IoT je mnogo promijenio način života i funkcioniranje industrija što je dovelo do velikog tehnološkog napretka.

MQTT protokol jedan je od „najlakših“ protokola na internetu. Sve više IoT kompanija odlučuje se upravo za MQTT protokol. Takva vrsta protokola omogućila je brojnim uređajima „laganu“ i jeftinu komunikaciju. Korisniku se pruža nadgledanje i interakcija sa sustavom što je jako bitno za daljnji razvoj automatiziranih sustava. „Lagana“ i sigurna komunikacija temelj je popularnosti MQTT protokola. Instalacijom besplatnog brokera svatko može implementirati protokol u svoj sustav te postati dio IoT svijeta.

U završnom radu prikazana je direktna primjena IoT sustava i MQTT protokola. Izradom primjera postojećeg sustava na Arduino razvojnoj platformi mogu se vidjeti nedostaci. Takav sustav nema mogućnosti spajanja na internet te korisniku nije omogućeno nadgledanje i upravljanje na daljinu. Podacima koji se skupljaju senzorima nije moguće pristupiti, a upravljanje ventilatora oslanja se na automatski način rada. Proširivanje takvog sustava s jeftinim mikrokontrolerom koji ima pristup internetu omogućuje korisniku nadzor i upravljanje. Kako bi sustav postao dio IoT svijeta implementiran je vanjski modul koji se bazira na Esp8266 mikrokontroleru. Ostvarivanjem komunikacije i pružanjem podataka korisniku putem Android mobilne aplikacije može se shvatiti koliko su takvi sustavi važni u današnjici. Nije potrebno trošiti dodatno vrijeme i novac kako bi se provjerio rad sustava na udaljenoj lokaciji, a prate se i industrijski trendovi.

U Varaždinu,_____ _____

Sveučilište
Sjever

IZJAVA O AUTORSTVU
I
SUGLASNOST ZA JAVNU OBJAVU

Završni/diplomski rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, Dominik Bajec *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/ica završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom *Implementacija IoT rješenja u postojeće sustave upravljanja mikrokontrolerima* *(upisati naslov)* te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)

Dominik Bajec
(vlastoručni potpis)

Sukladno Zakonu o znanstvenoj djelatnost i visokom obrazovanju završne/diplomske radove sveučilišta su dužna trajno objaviti na javnoj internetskoj bazi sveučilišne knjižnice u sastavu sveučilišta te kopirati u javnu internetsku bazu završnih/diplomskih radova Nacionalne i sveučilišne knjižnice. Završni radovi istovrsnih umjetničkih studija koji se realiziraju kroz umjetnička ostvarenja objavljuju se na odgovarajući način.

Ja, Dominik Bajec *(ime i prezime)* neopozivo izjavljujem da sam suglasan/na s javnom objavom završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom *Implementacija IoT rješenja u postojeće sustave upravljanja mikrokontrolerima (upisati naslov)* čiji sam autor/ica.

Student/ica:
(upisati ime i prezime)

Dominik Bajec
(vlastoručni potpis)

7. Literatura

[1] R.buyya, A. V. Dastjerdi, Internet of Things – Principles and Paradigms, 2016.
[2] https://www.networkworld.com/article/3207535/what-is-iot-the-internet-of-things-explained.html, dostupno 2020.
[3] https://www.itransition.com/blog/iot-history, dostupno 2020.
[4] https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/, dostupno 2020.
[5] D.Reinsel , J. Gantz, J. Rydning, IDC- The Digitization of the World –Frin Edge to Core, 2018.
[6] https://blogs.oracle.com/bigdata/how-big-data-powers-the-internet-of-things, dostupno 2020.
[7] https://www.ericsson.com/en/about-us/company-facts/ericsson-worldwide/india/authored-articles/5g-and-iot-ushering-in-a-new-era, dostupno 2020.
[8] GSMA, 5G, the Internet of Things (IoT) and Wearable Devices, 2019.
[9] Gastón C. Hillar, MQTT Essentials - A Lightweight IoT Protocol, 2017.
[10] V. Lampkin, W. Tat Leong, L. Olivera, S. Rawat, N. Subrahmanyam, R. Xiang, Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry, 2012.
[11] https://medium.com/mqtt-buddy/mqtt-vs-http-which-one-is-the-best-for-iot-c868169b3105, dostupno 2020.
[12] https://en.wikipedia.org/wiki/MQTT, dostupno 2020.
[13] https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/, dostupno 2020.
[14] https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/, dostupno 2020.
[15] https://github.com/mqtt/mqtt.github.io/wiki/public_brokers, dostupno 2020.
[16] https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe/, dostupno 2020.
[17] https://docs.microsoft.com/en-us/azure/architecture/patterns/publisher-subscriber, dostupno 2020.
[18] https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/, dostupno 2020.
[19] https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/, dostupno 2020.
[20] https://www.arduino.cc/en/Guide/Introduction, dostupno 2020.
[21] https://www.adafruit.com/product/1603, dostupno 2020.
[22] https://learn.sparkfun.com/tutorials/bmp180-barometric-pressure-sensor-hookup-/all, dostupno 2020.
[23] http://www.datasheetcafe.com/bmp180-datasheet-pressure-sensor/, dostupno 2020.
[24] https://cdn-shop.adafruit.com/datasheets/BST-BMP180-DS000-09.pdf, dostupno 2020.
[25] https://www.adafruit.com/product/1899, dostupno 2020.
[26] https://iotdk.intel.com/docs/upm/v0.4.1/java/classupm__htu21d_1_1_h_t_u21_d.html, dostupno 2020.
[27] https://cdn-shop.adafruit.com/datasheets/1899_HTU21D.pdf, dostupno 2020.
[28] https://www.onsemi.com/pub/Collateral/BC337-D.PDF, dostupno 2020.
[29] https://www.elecrow.com/download/ESP-12F.pdf, dostupno 2020.
[30] https://www.espressif.com/sites/default/files/documentation/esp-touch_user_guide_en.pdf, dostupno 2020.
[31] https://www.electronics-lab.com/project/using-st7735-1-8-color-tft-display-arduino/, dostupno 2020.
[32] https://www.circuitbasics.com/basics-uart-communication/, dostupno 2020.
[33] https://developer.android.com/guide/topics/ui/declaring-layout, dostupno 2020.

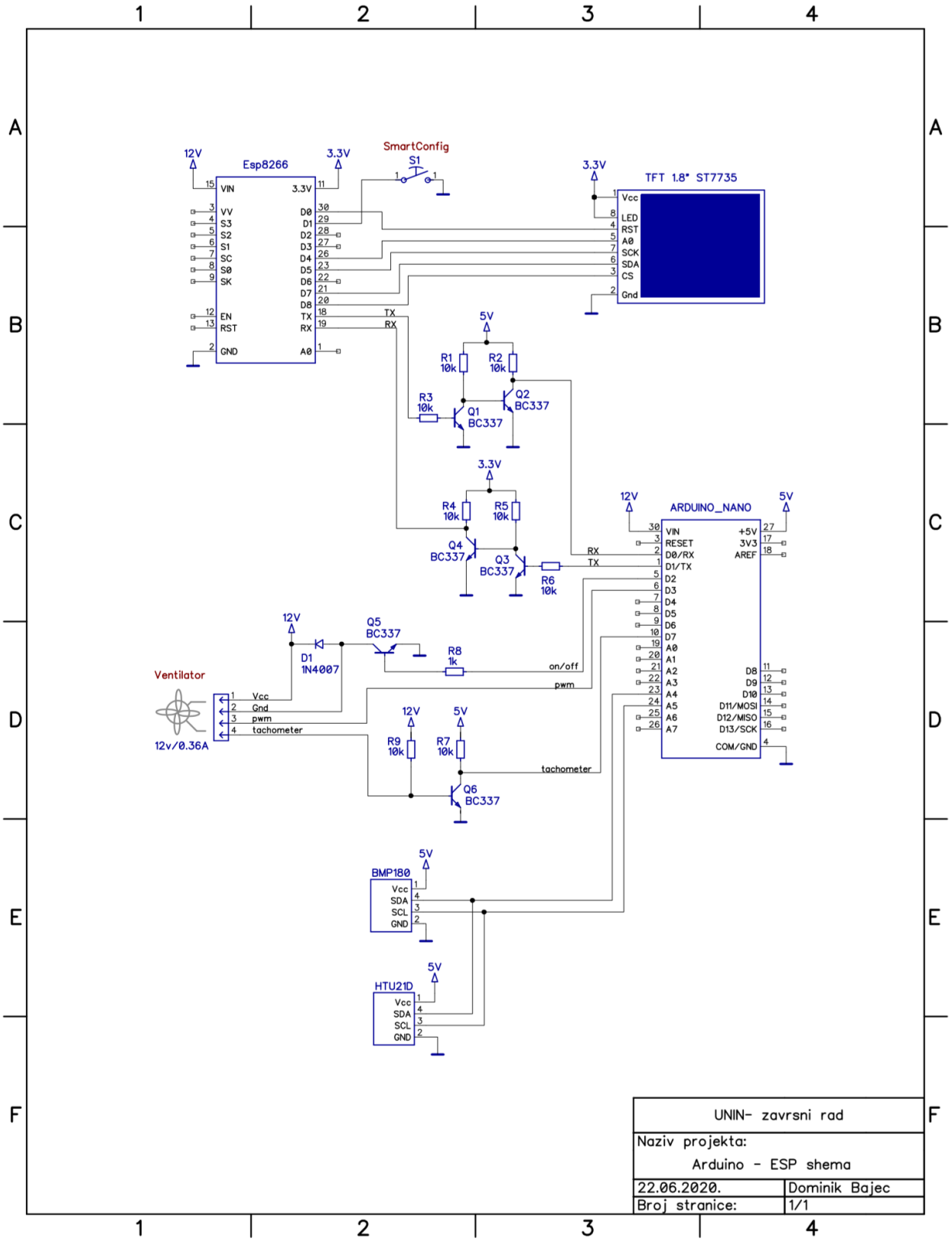
Popis slika

Popis tablica

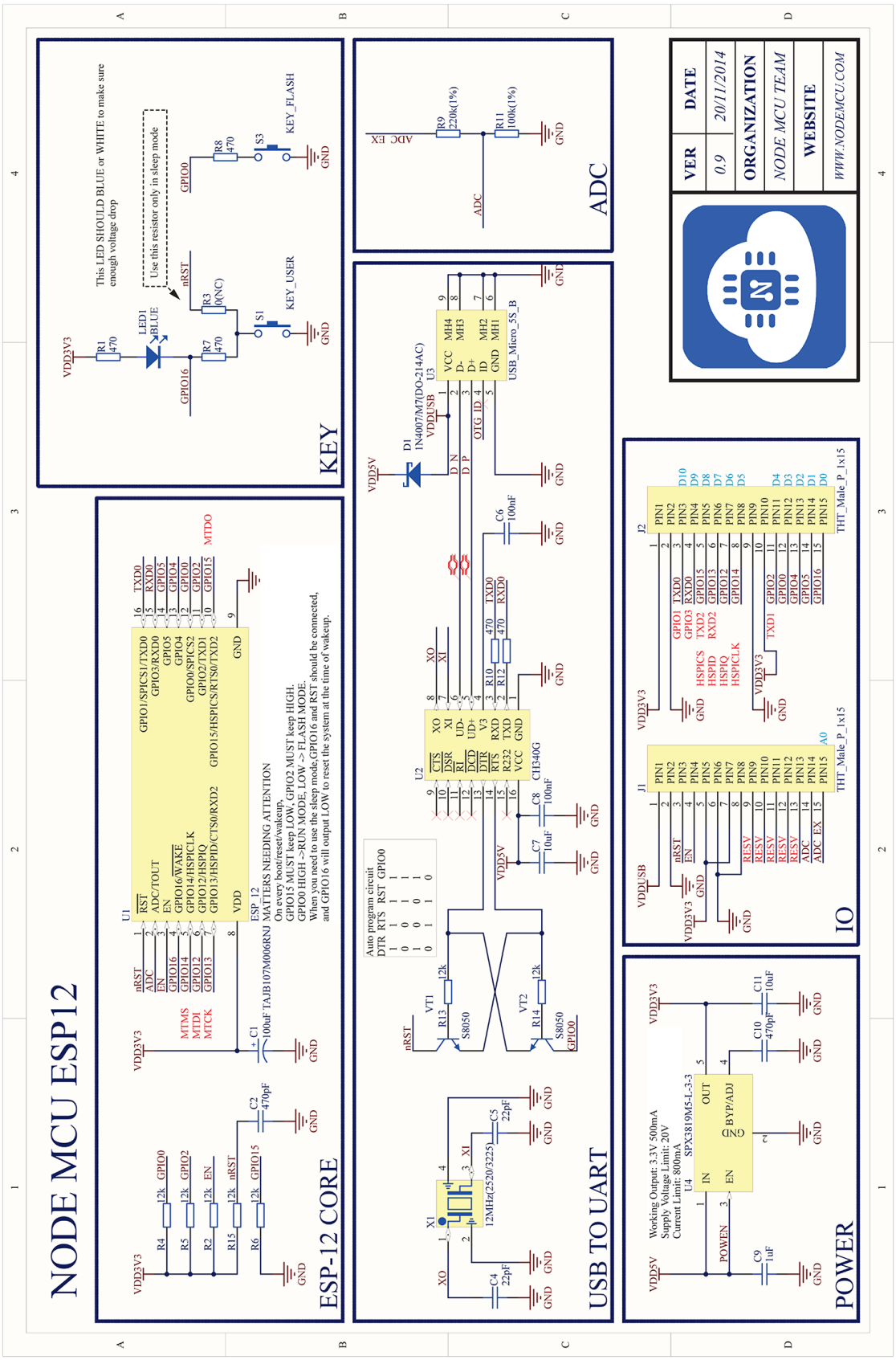
Popis programskih blokova

Prilozi

Shema projekta:



Eesp-12F razvojna pločica – Shema



Arduino Nano – Shema