Program za nadgledanje umreženih senzora s pomoću protokola MQTT

Matija, Kralj

Undergraduate thesis / Završni rad

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:925670

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Matija Kralj

Program za nadgledanje umreženih senzora s pomoću protokola MQTT

ZAVRŠNI RAD

Varaždin, 2018.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Matija Kralj

Matični broj: 44073/15–R

Studij: Informacijski sustavi

Program za nadgledanje umreženih senzora s pomoću protokola MQTT

ZAVRŠNI RAD

Mentor:

Doc. dr. sc. Nikola Ivković

Varaždin, srpanj 2018

Matija Kralj

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Aplikacija izrađena u C# jeziku pomoću alata Visual Studio, koja prati rad dva virtualna senzora i jednog stvarnog digitalnog senzora koji je spojen i napajan uređajem Raspberry Pi. Uređaj Raspberry Pi je povezan na Internet, s njega se mogu čitati mjerni podaci senzora te ih slati na server pomoću protokola MQTT, stvarajući tzv. Internet stvari (engl. *Internet of things*, IoT). Klijentska aplikacija (aplikacija na osobnom računalu) će biti u mogućnosti slati zahtjev serverskoj aplikaciji, koja se nalazi na uređaju Raspberry Pi, za mjerenjem jednog od spomenuta tri senzora ili pak za prikazivanjem povijesti mjerenja. Serverska aplikacija zapisuje podatke mjerenja senzora u tekstualne datoteke i to svakih pet minuta kreirajući povijest mjerenja senzora. Klijentska aplikacija prikazuje trenutne podatke mjerenja, ali također i iscrtava grafove ovisno o vrijednostima mjerenja iz datoteke (povijesti mjerenja) koja se nalazi na uređaju. Uređaj Raspberry Pi i klijent komuniciraju na način da su oba uređaja spojena na MQTT broker (server) koji se zove „iot.eclipse.org“, a komunikacija se odvija pomoću protokola MQTT.

Ključne riječi: MQTT, protokol, IoT, Eclipse, Raspberry Pi, umreženi senzori, nadgledanje, C#, aplikacija.

Sadržaj

1. Uvod

Kako sam već na kolegiju „Mreže računala 2“ radio seminar na temu „Chat aplikacija pomoću MQTT protokola“ imam neka saznanja o samom protokolu, kao što su način rada protokola i znanja o njegovoj praktičnoj primjeni. Zbog svega ovoga što sam naveo, te zbog zainteresiranosti za radom na praktičnim stvarima i na stvarima koje se sve više koriste i razvijaju, kao i zbog same mogućnosti povezivanja elektrotehnike (uređaj Raspberry Pi, senzori, LED), programiranja koje me sve više interesira (izrada aplikacija i njihovih funkcionalnosti) i mreža računala (MQTT protokol i snimanje Internet prometa) odlučio sam se za ovu vrlo zanimljivu temu završnog rada. Iz naziva teme završnog rada može se zaključiti kako ću izraditi neku aplikaciju, a ta aplikacija će biti izrađena u alatu Visual Studio, pomoću C# jezika. Točnije biti će potrebno izraditi dvije aplikacije, jedna aplikacija (u daljnjem tekstu: serverska aplikacija) će služiti uređaju Raspberry Pi (u daljnjem tekstu: uređaj ili Raspberry Pi) kako bih imao funkcionalnosti čitanja mjernih vrijednosti digitalnih senzora i slati iste pomoću protokola MQTT drugoj aplikaciji (u daljnjem tekstu: klijentska aplikacija). Klijentska aplikacija će imati sučelje koje korisnik vidi i pomoću kojeg ostvaruje interakciju sa serverskom aplikacijom. Spomenuo sam digitalni senzor (senzor temperature i vlage, Dht11), ovaj senzor se spajaju na pinove Raspberry Pi-a, a sve funkcionira na način da mu uređaj daje ulazni napon, a senzor njemu za uzvrat vraća mjerne podatke. Ostala dva senzora biti će implementirani virtualno u samoj serverskoj aplikaciji, te će biti opisano kasnije kako rade. Senzori postaju umreženi tek onda kada uređaj ima pristup Internetu, te se ti podaci mogu poslati i pohraniti u neko spremište podataka ili prikazati korisniku. Svi podaci razmijenjeni između dvije aplikacije biti će poslani, odnosno primljeni pomoću protokola MQTT preko javnog MQTT Eclipse brokera. U naslovu rada stoji i riječ „nadgledanje“, što znači da će korisnik biti u mogućnosti čitati vrijednosti senzora, pratiti vrijednosti prema traženog senzora i prikazivati ih na grafovima. U radu ću opisati sam protokol MQTT, te u sklopu toga opisati što znači riječ „broker“, opisati ću što je to Internet stvari, što su senzori, te će također biti opisan uređaj Raspberry Pi i njegova uloga u ovom završnom radu.

2. Protokol MQTT

MQTT (engl. *Message Queuing Telemetry Transport*) je klijent-server objavi/pretplati (engl. *publish/subscribe*) protokol za prijenos poruka, koji je izrazito lagan što se tiče količine podataka koju šalje i koristi (engl. *light weight*), otvoren, jednostavan i dizajniran da se jednostavno implementira („International Organization for Standardization/ International Electrotechnical Commission [ISO/IEC]“, 06.2016). Odgovor na pitanje tko je osmislio protokol MQTT može se pronaći na MQTT web stranici, gdje se navodi da su to bili Dr. Andy Stanford-Clark iz IBM-a i Arlen Nipper iz tvrtke Arcom (današnji Eurotech) („MQTT Frequently Asked Questions [MQTT FAQ], bez dat.).

2.1. Kako MQTT radi?

U ISO/IEC (06.2016) navode da protokol radi preko TCP/IP (engl. *Transmission Control Protocol* / *Internet Protocol*) ili preko drugih protokola koji osiguravaju prijenos podataka u stvarnom, odnosno organiziranom redoslijedu (engl. *ordered data*), te vezu u oba smjera i slanje bez gubitka podataka.

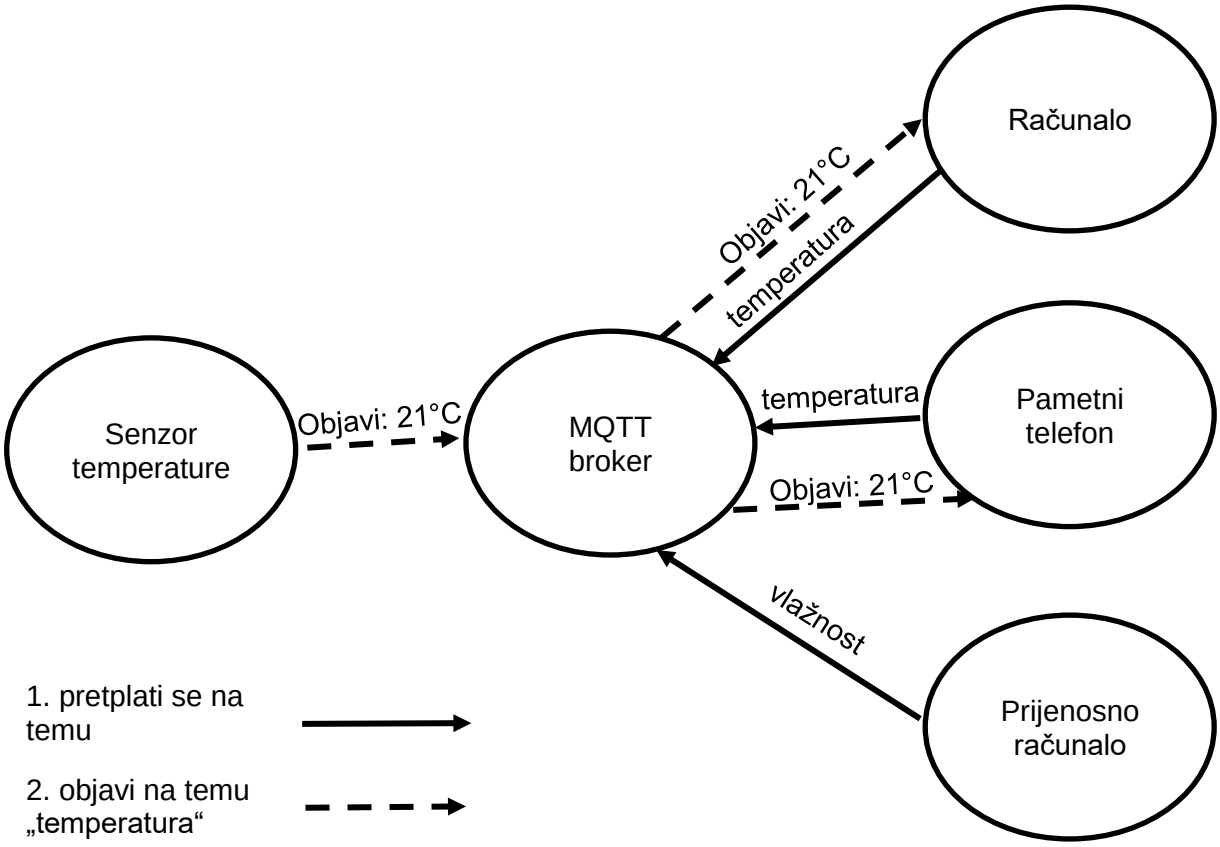
Dakle, ovakav prijenos podataka je siguran i pouzdan, za slanje više poruka dovoljno je jednom uspostaviti vezu između uređaja, te je po završetku rada zatvoriti. Naravno, ovaj protokol nije najbrži mogući, ali to nije toliko važno zbog samog područja primjene, odnosno mnogo su važniji točnost podataka i mehanizmi koji osiguravaju da se podaci gube u što manjoj količini. Protokol MQTT se najviše koristi kod uređaj-uređaj (engl. *Machine to Machine*, M2M) komunikacije i kod Interneta stvari (engl. *Internet of Things*, IoT), primjerice kod IoT pametne kuće nikako ne bismo željeli da nam se u kući umjesto hlađenja uključi grijanje zbog gubitka poslanih podataka prema brokeru ili od brokera prema uređaju koji upravlja termostatom.

Kako sam u uvodu spomenuo da je ovo objavi/pretplati prijenos poruka, kako bismo mogli objavljivati poruke ili ih primati, prvo se je potrebno pretplatiti na neku temu (engl. *topic*). Svi uređajiu toj temi primaju poslane poruke i mogu ih slati svima u toj temi. Ovakvo slanje i primanje podataka može se usporediti jednim pojmom koji se koristi kod radio i televizijske komunikacije, a to je emitiranje (engl. *broadcast*).

Kod emitiranja nekog filma (poruka) na televizijskom programa, svi koji imaju taj program (svi koji su pretplaćeni na tu temu) mogu gledati taj film (čitati tu poruku). Postoje dvije ključne razlike između protokola MQTT i televizijskog emitiranja. Prva od njih je ta da sam uređaj (TV) ne može emitirati svoj sadržaj drugim uređajima, odnosno nije zadovoljen uvjet veze u oba smjera. Druga razlika leži u činjenici da se pomoću protokola MQTT ne mogu slati slika ni zvuk.

2.2. MQTT broker

Kako se u uvodnom dijelu ovog poglavlja spominje da je MQTT klijent-server protokol, može se zaključiti kako nam je za rad protokolom potreban server, koji se u MQTT žargonu naziva broker. Postoji puno MQTT brokera, a samo neki od njih su: „HiveMQ“, „Mosquitto“, „Eclipse IoT“, kojeg i sam koristim u ovom radu, te će biti ukratko opisan kasnije. Na sljedećoj slici je, prema slici na HiveMQ-ovoj stranici („dc-square GmbH“, bez dat.), ugrubo prikazan način na koji funkcionira MQTT protokol uz pomoć brokera (u ovom radu je radi kvalitetnijeg opisivanja teme slika malo izmijenjena). Prikazan je konkretan slučaj mjerenja temperature i pretplate na temu (engl. *topic*) „temperatura“, te objavljivanje na istu temu.



Slika 1: Grubi prikaz načina rada MQTT s brokerom („dc-square GmbH“,bez dat.)

Na slici se može primijetiti kako na MQTT brokeru (u daljnjem tekstu, broker) postoji tema „temperatura“ koju je netko kreirao. Uređaji „Računalo“ i „Pametni telefon“ pretplaćeni su na temu „temperatura“, dok je uređaj „Prijenosno računalo“ pretplaćeno na temu „vlažnost“. Isprekidane strelice prikazuju objavljivanje na temu temperature, tako su svi uređaji osim prijenosnog računala dobili informaciju o trenutnoj temperaturi 21°C. Dakle, možemo zaključiti kako MQTT nije protokol poput SMTP-a (engl. *Simple Mail Transfer Protocol*) ili sličnih, jer ne

trebamo znati e-mail ili IP adresu, već broker prosljeđuje podatke od objavljivača na temu (engl. *publisher*) svim pretplaćenim uređajima na toj temi. Sve to samo potvrđuje već prije navedene činjenice i da je ovo *light weight* protokol, te da mu za rad ne treba jako stabilna i brza Internet veza. Potvrdu o tome da je ovo *light weight* protokol nalazimo u tome da poruke ne moraju nositi teret podataka o primatelju. Potvrdu druge činjenice pronalazimo u tome da objavljivač šalje samo jednu poruku koju šalje na broker koji se dalje brine o svemu drugome, te ima brzu i stabilnu Internet vezu i nikako ne može biti „usko grlo“ ove mreže uređaja.

2.3. Sigurnost

Dc-square GmbH (MQTT Security Fundamentals, bez dat.) na svojoj web stranici o MQTT sigurnosti navode kako je sigurnost ovakvog protokola od značajne važnosti kako bi svi podaci i detalji, osobito oni osjetljivi, ostali sačuvani i skriveni od javnosti, i to zato jer protokol kreira Internet stvari (povezuje uređaje na Internet; automobile, pametne kuće i sve uređaje u njoj, nadzorne kamere, i sl.). Također navode i kako ni oni, ni bilo koje druge kompanije koje imaju doticaj ili utjecaj na MQTT ne žele propuste u sigurnosti protokola, jer bi to bila i velika šteta na njihov račun.

Postoji nekoliko načina ili stupnjeva osiguravanja protokola, a to su: autentifikacija korisničkim imenom i lozinkom, napredna provjera autentičnosti, autorizacija, TLS / SSL (engl. *Transport Layer Security / Secure Sockets Layer*) sigurnosni slojevi („dc-square GmbH“, MQTT Security Fundamentals, bez dat.). Navode i kako su autentifikacija i autorizacija dva „najbolja prijatelja“ i da se osiguravanje protokola MQTT odvija na sljedećim slojevima: infrastruktura, operacijski sustavi, MQTT broker. U infrastrukturu spadaju vatroštit (engl. *firewall*), balansiranje opterećenja (engl. *Load balancer*) i DMZ (engl. *demilitarized zone*), operacijski sustavi pridonose sigurnosti ažuriranjima, korištenjem SSH ključeva (engl. *Secure Shell*). Na MQTT brokeru se osiguravaju autentikacija i autorizacija, te TLS i veličina poruka koje, kako navode, maksimalno mogu dosegnuti 256MB, ali su u praksi i puno manje, čak manje i od 1kB.

2.4. Razina kvalitete pružanja usluge kod MQTT-a

Kvaliteta usluge (engl. *Quality of Service*, QoS) je dogovor (sporazum) između pošiljatelja i primatelja poruke koji definira jamstvo isporuke određene poruke, ovisno o razini dogovora („dc-square GmbH“, MQTT Essentials Part 6, bez dat.). Kod protokola MQTT postoje tri razine sporazuma. Razina 0: najviše jednom (engl. *At most once*), razina 1: barem jedanput (engl. *At least once*), razina 2: točno jedanput (engl. *Exactly once*). U dc-squeare GmbH spominju kako

je QoS ključna značajka protokola jer omogućava korisniku da sam izabere razinu sporazuma, što je jako važno jer korisnik najbolje zna kakvom kvalitetom Internet usluge i kakvim podacima raspolaže.

2.4.1 QoS 0

Ukoliko je korisnikova veza na Internet ili kvaliteta usluge Interneta mala, korisnik može odabrati da li mu je važnije da se poruka sigurno isporuči (razina 1 ili 2) ili mu je važnija brzina obrade poruke koja se šalje (razina 0). Naime, ukoliko korisnik ima uređaj koji je po moći svojih performansi na niskoj razini i ne želi da mu se poruka dugo zadržava u spremniku, te je se želi brzo riješiti da ne usporava sustav, odabrati će QoS 0. Ova razina još se zove i „pucaj i zaboravi“ (engl. „*fire and forget*“), a funkcionira na način da klijent pošalje poruku na broker i dalje ga nije „briga“ za poruku - odmah ju briše iz svog spremnika (engl. *Buffer*). Ova razina usluge još se može koristiti i kod poruka koje nisu od neke velike važnosti i često se šalju iterativno, primjerice kod poruka provjere veze, tzv. „ping“ poruke.

Ipak, u dc-squeare GmbH-u preporučaju da se ova razina koristi samo kad je konekcija između servera i klijenta jako stabilna.

2.4.2 QoS 1

Pitanje je, što je s razinom 1? Razina 1, odnosno razina „barem jedanput“, osigurava da će poruka stići na broker barem jednom. U dc-squeare GmbH-u navode kako klijent sprema poruku koju je poslao prema brokeru tako dugo dok od brokera ne dobije povratnu potvrdnu poruku „PUBACK“. Klijent brokeru šalje jednu te istu poruku tako dugo dok ne dobije potvrdu o primljenoj poruci. Naravno i „PUBACK“ poruke se mogu izgubiti. Ukoliko se izgube, klijent će ponovno poslati istu poruku, zbog toga je moguće da broker više puta dobije istu poruku.

QoS1 se koristi kada je važno da se poruka isporuči, a duplikati su prihvatljivi, te se može reći da je najčešće korištena razina (dc-squeare GmbH, bez dat.).

2.4.3 QoS 2

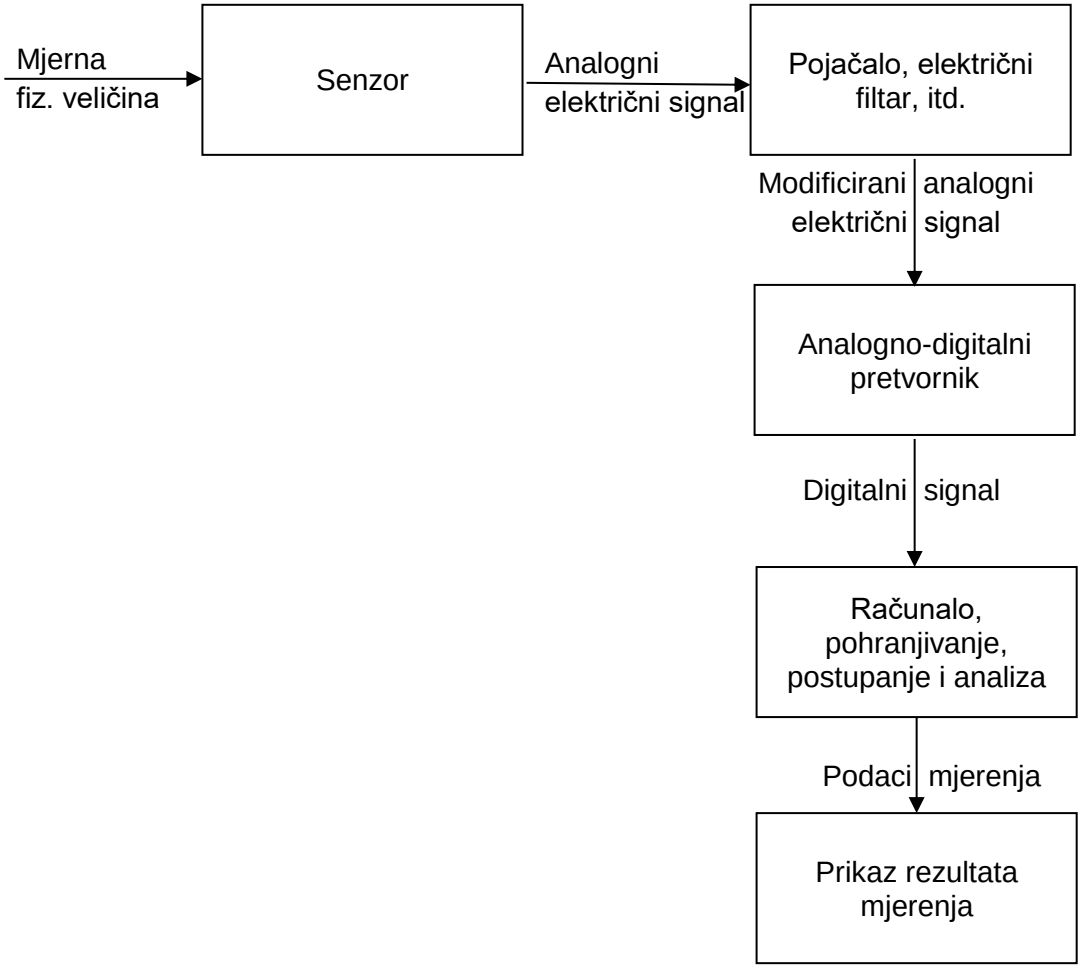
Glavna razlika između razine 1 i 2 je ta da razina 2 osigurava da poruka stigne na broker točno jedanput. U dc-squeare GmbH-u navode da je to najsigurnija, ali i najsporija metoda slanja MQTT poruka. Spominju i to da ukoliko je korisnik odabrao ovu razinu sporazuma, klijent i broker šalju pakete umjesto samih poruka. Svaki paket ima svoju jedinstvenu oznaku, koja nije redni broj paketa, jer svaki klijent ima jedinstvenu oznaku paketa. Klijent i broker na ovoj razini odrađuju „četverostruko rukovanje“ (engl. *four-part handshake*) koje radi na način da klijent pošalje MQTT poruku (koja je zapravo paket s jedinstvenim identifikatorom) brokeru, kada broker dobije taj paket od pošiljatelja, vraća potvrdni paket „PUBREC“ klijentu. Ukoliko

klijent (pošiljatelj) nije dobio „PUBREC“ potvrdu, ponovno šalje početni paket brokeru, ali ovaj put sa zastavicom koja ukazuje na to da je ovaj paket duplikat, i to radi tako dugo dok ne dobije prije spomenutu potvrdu od brokera. U trenutku kada klijent dobije potvrdu brokera, sprema taj paket potvrde, a paket koji je slao prema brokeru može izbrisati. Pošiljatelj (klijent) odgovara na spomenuti „PUBREC“ paket jednim drugim paketom koji se zove „PUBREL“. Ovaj paket koristi brokeru na način da kada ga dobije, sigurno može izbrisati sve spremljene procedure i pakete ovog klijenta i odgovoriti klijentu s paketom „PUBCOMP“ što je znak da klijent može nastaviti sa slanjem drugih paketa (dc-squeare GmbH, bez dat.). Ukoliko se završni paketi („PUBREL“, „PUBCOMP“) u nekom trenutku ne izruče, ponavljaju se slični postupci kao i s ostalim paketima navedeni prije.

QoS 2 razina se koristi u situacijama u kojima je od kritične važnosti da se poruka isporuči i da se isporuči samo jednom, dakle bez duplikata, sporija je od ostalih razina (dc-squeare GmbH, bez dat.).

3. Senzori

Senzori prema izlaznim vrijednostima mogu biti digitalni ili analogni i služe za mjerenje ili prepoznavanje postojanja neke fizikalne veličine. Piljac (2010, str. 2) navodi kako su senzori jednostavni elektronički elementi ili složene elektroničke komponente pomoću kojih neku fizikalnu veličinu pretvaramo u električni signal. Piljac (2010, str. 1) također navodi i sljedeću definiciju fizikalne veličine: „Svojstva materije i energije koja možemo mjeriti nazivamo mjerljivim svojstvima ili fizikalnim veličinama“. Neka fizikalna mjerljiva svojstva su npr. temperatura, vlažnost zraka, količina plina u zraku, itd. Naravno, svi senzori su zapravo analogni i mjere analogne veličine jer je u prirodi sve analogno. Digitalni senzori se od analognih razlikuju po tome jer imaju ugrađene dodatne module koji pretvaraju analogne veličine u digitalne. U svrhe mjerenja neke fizikalne veličine (elektroničkim putem, automatski) najlakše je koristiti senzore s digitalnim izlaznim signalom jer računala „znaju“ raditi s digitalnim signalima. Ukoliko bi se koristili senzori s analognim izlaznim signalom, analogni signal bi se trebao na neki način pretvoriti prvo u digitalni, te nakon toga pretvoriti digitalni signal u vrijednost. Sljedeća slika prikazuje načelo rada senzora s digitalnim izlaznim signalom.



Slika 2. Načelo mjerenja fizilakne veličine pretvorbom u električni signal (Piljac, 2010, str. 2)

4. Internet stvari

Internet stvari (engl. *Internet of Things*, IoT) je prema Vermesanu i Friessu (2013, str. 7, 8) koncept i paradigma koja uzima u obzir razne stvari (predmete, engl. *Things*) u nekom okruženju koje su bežično ili žično povezane, a imaju jedinstveno adresiranje i mogu međudjelovati i surađivati jedni s drugima stvarajući nove servise (engl. *Services*). Također Vermesan i Friess (2013, str. 2) spominju i kako ideja IoT-a nije nova, no tek u ovo novije vrijeme počinje rasti i široko se primjenjivati, ali i da je još uvijek u fazi razvoja i proučavanja.

Internet stvari može se kreirati i na način da se uređaji, stvari, integriraju na neku veću komponentu koja ima pristup Internetu i može upravljati i koristiti navedene uređaje, naredbe za upravljanje pristižu putem Interneta. Na taj način sam ja zapravo i kreirao Internet stvari, senzore, stvari, integrirao sam na neku veću komponentu, na uređaj Raspberry Pi, koja ima pristup Internetu.

4.1. Problemi s konceptom Internet stvari

Pronašao sam jednu zanimljivu rečenicu koju ću pokušati objasniti onako kako ja to vidim i smatram zašto je ta rečenica ispravna, a ona glasi: „Every poorly secured device that is connected online potentially affects the security and resilience of the Internet globally, not just locally“ (Rose, Eldridge, Chapin, [ISOC] listopad 2015, str. 32). Dakle, iz ISOC-a tvrde da postoje problemi, i to oni sigurnosni. Prema tome, Ova prije citirana rečenica ima svoja uporišta, jedan primjer na kojem se ovo može objasniti je „hakiranje“ osobnog računala i postavljanje potencijalno opasnog koda (maliciozni kod), tada je narušena samo lokalna sigurnost. No kako su računala povezana pomoću Interneta, taj maliciozni kod može se proširiti na sve datoteke. Svaka datoteka koja se proslijedi putem Interneta (Internet aplikacije, Internet servisi, i dr.) može potencijalno biti zaražena, te se problem može proširiti globalno, tako je i s IoT-om i sa svim uređajima u mreži. Jedno od rješenja ovog i ujedno najvažnijeg problema je da uređaji rade kao crne kutije, odnosno da vanjski svijet nema gotovo nikakav pogled na unutarnji rad uređaja.

Jesu li ljudi voljni platiti za sigurnost i veću cijenu ili bi radije riskirali i platili manje? Ili pitanje poput, jesu li programeri i inženjeri spremni raditi više za istu cijenu proizvoda, samo da bi proizvod bio sigurniji? Rješenja za standarde i metrike IoT uređaja, kako im mjerimo učinkovitost? To su samo neka od pitanja na koja još treba dati odgovore i rješenja, navode Rose i suradnici iz ISOC-a (listopad 2015, str 35-37).

Bez obzira na sva, još uvijek neodgovorena pitanja, teškoće, probleme ili loše strane Interneta stvari, dobre strane su one što su podigle popularnost ovog koncepta. Jedna od

najvažnijih komponenti današnjice je vrijeme i ušteda vremena, tvrdi Prateek Saxena (The advantages and disadvantages of Internet Of Things, 15.6.2016), direktor u AppInventivu.

4.2. Raspberry Pi

Što je to Raspberry Pi? Odgovor na to, često postavljano pitanje, može se pronaći na web službenoj web stranici „The Raspberry Pi Foundation“ kompanije u odjeljku FAQ. Raspberry Pi je računalo veličine kreditne kartice koje može na sebe priključiti ekran (TV, monitor) i tipkovnicu. Malo, sposobno računalo koje se koristi u raznim projektima i sposobno je za puno stvari kao i obično stolno ili prijenosno računalo (The Raspberry Pi Foundation, bez dat.). Raspberry Pi može imati operacijski sustav koji je pohranjen na microSD (*micro Secure Digital*) kartici, a u svojem završnom radu koristim Windows 10 IoT Core operacijski sustav i Raspberry Pi 2 Model B. U tablici ispod slijede specifikacija korištenog uređaja.

Procesor	QUAD Core Broadcom BCM2836, koji radi na frekvenciji 900MHz
Radna memorija (RAM)	1GB LPDDR2
Pohrana	Slot za microSD karticu
Sučelja za komunikaciju (*portovi*)	4 USB 2.0 (engl. *Universal Serial Bus*) *porta* HDMI (engl. *High-Definition Multimedia Interface*) priključak CSI (engl. *Camera Serial Interface*) kamera *port* DSI (engl. *Display Serial Interface*) *port* za prikaz na ekran Kompozitni video priključak 3.5mm priključak za zvučni izlazs
Izlazno ulazni pinovi za opću upotrebu	40 GPIO pinova (engl. *General-purpose input/output*)
Napajanje	Micro USB napajanje (+5V, 2A istosmjerno)
Pristup Internetu	10/100 BaseT Ethernet socket WiFi – nema

Tablica 1: Specifikacija uređaja Raspberry Pi 2 Model B (autorski rad, prema tehničkoj dokumentaciji uređaja)

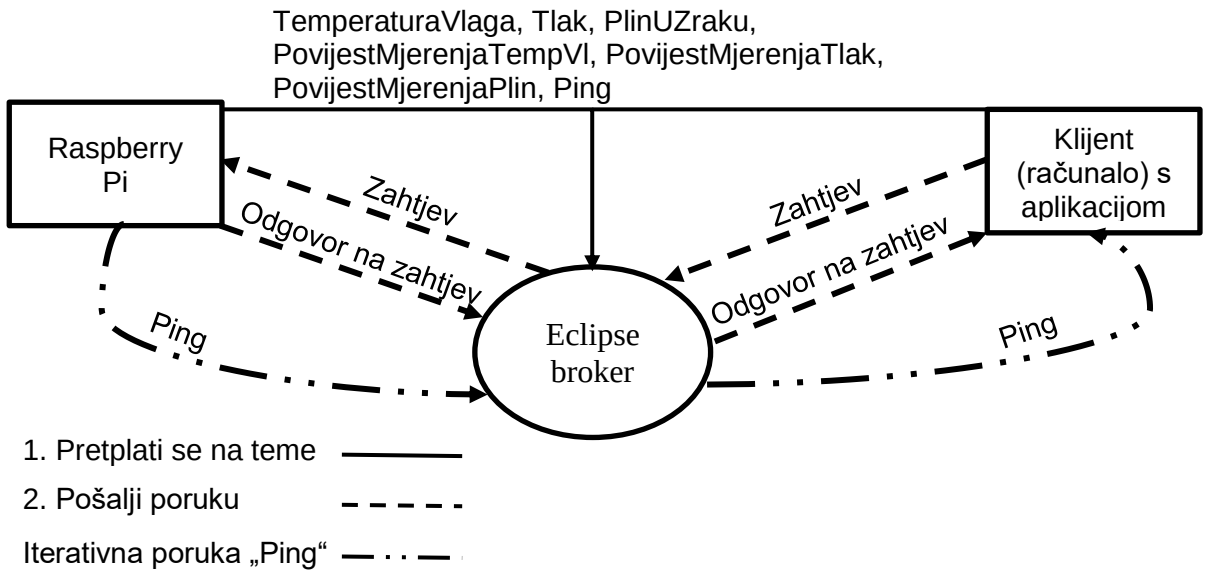
Za kraj ovog poglavlja, još ću samo staviti sliku pinova kojima Raspberry Pi 2 model B raspolaže, sami brojevi pinova i njihova imena. Imena pinova su zapravo ono što je važno, zbog toga jer se pinovima iz programskih jezika pristupa prema imenu, npr. „GPIO21“.

Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I2C)		DC Power 5v	04
05	GPIO03 (SCL1 , I2C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I2C ID EEPROM)		(I2C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

Slika 3: Raspored pinova Raspberry Pi 2 model B uređaja (Raspberry Pi Geek pinout, bez dat.)

5. Program za nadgledanje senzora

Program za nadgledanje umreženih senzora s pomoću MQTT protokola su zapravo dvije aplikacije izrađene u programskom jeziku C#. Jedna aplikacija izrađena je i implementirana na uređaj Raspberry Pi (u daljnjem tekstu serverska ili Raspberry aplikacija), odnosno na njegovu microSD karticu na kojoj se nalazi i operacijski sustav Windows 10 IoT Core. Druga aplikacija može se nalaziti na bilo kojem stolnom računalu koje ima operacijski sustav Windows (u daljnjem tekstu klijentska ili Windows aplikacija), a služi za slanje zahtjeva serverskoj aplikaciji preko „iot.eclipse.org“ brokera uz pomoć protokola MQTT. Također služi i za primanje informacija od serverske aplikacije, te za prikaz tih informacija korisniku u grafičkom obliku, odnosno ova aplikacija je korisničko sučelje za rad (upravljanje) sa serverskom aplikacijom. Slijedi slika koju sam osmislio da lakše prikažem rad mojih aplikacija.

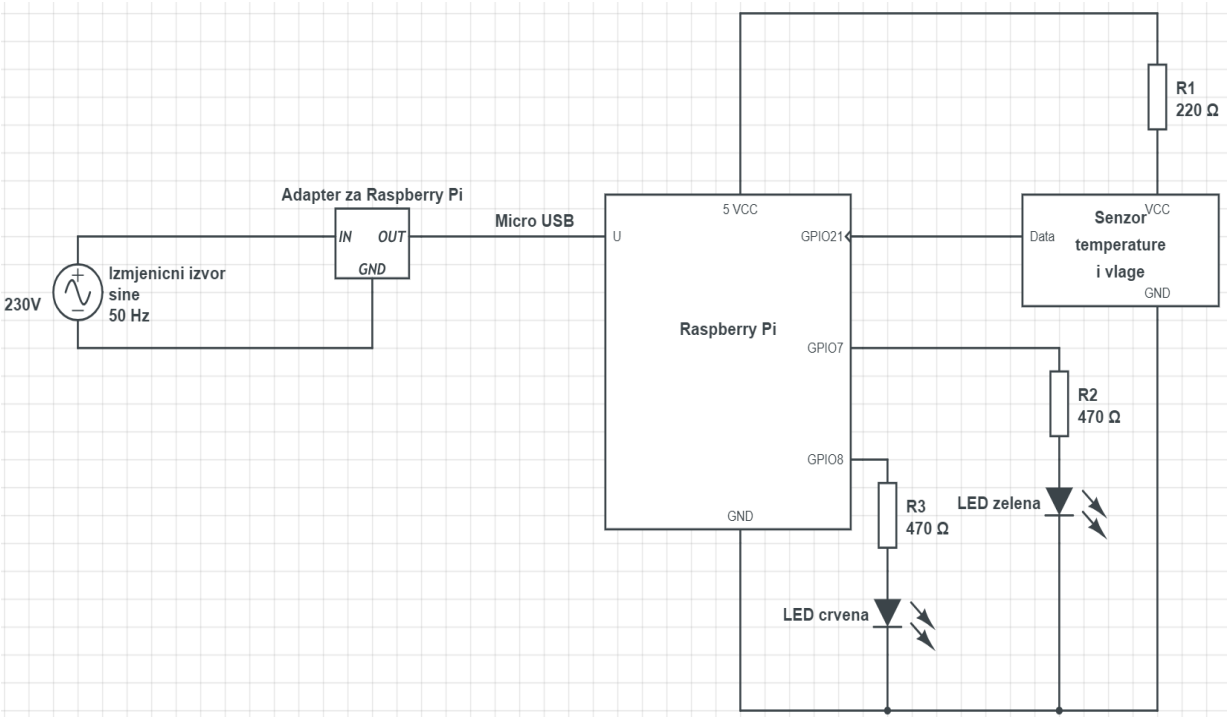


Slika 4: Blok dijagram MQTT poruka (autorski rad)

Kao što vidimo iz slike iznad, klijent (računalo s aplikacijom) i Raspberry Pi, odnosno aplikacija koja je implementirana na uređaj, razmjenjuju poruke u kojima klijent šalje zahtjev za obavljanjem neke radnje, Raspberry to obavi i vrati rezultat. Također, Raspberry Pi šalje poruku „Ping“ svaku sekundu i to beskonačno dugo (while – true petlja), odnosno tako dugo dok ima Interneta ili je Raspberry uključen. Spomenuta poruka „Ping“ služi samo da klijent vidi je li Raspberry Pi povezan na MQTT broker. Naravno, da bi sve ovo funkcioniralo, komunikacija mora ići preko „iot.eclipse.org“ brokera na koji se i klijent i Raspberry Pi povezuju i pretplate na ovih sedam tema koje vidimo na slici (puna crta). Detaljnije o svim porukama i radu obje aplikacije vidjeti ćemo kod dijagrama slijeda.

5.1. Električna shema rada

U ovom potpoglavlju ćemo vidjeti električnu shemu ovog rada, odnosno kako su senzor i Led diode povezani na pinove Raspberry Pi uređaja i ukratko ću ju opisati, što će nam kasnije pomoći za što detaljnije i preciznije opisivanje rada aplikacija.



Slika 5: Shema završnog rada (autorski rad)

Cijeli strujni krug započinje spajanjem adaptera na izmjenični izvor napajanja od 230V i 50Hz. Adapter u sebi sadrži transformator koji taj napon pretvara u 5V i struju od 2A istosmjerno. Izlaz iz adaptera je micro USB koji se spaja na za to predviđeno mjesto na Raspberry Pi uređaju. Nakon dolaska napajanja na uređaj, odmah na izlaznom pinu uređaja (5 VCC) postoji napon, ostali GPIO pinovi nemaju nikakav izlazni ni ulazni napon tako dugo dok se tako ne postavi u aplikaciji.

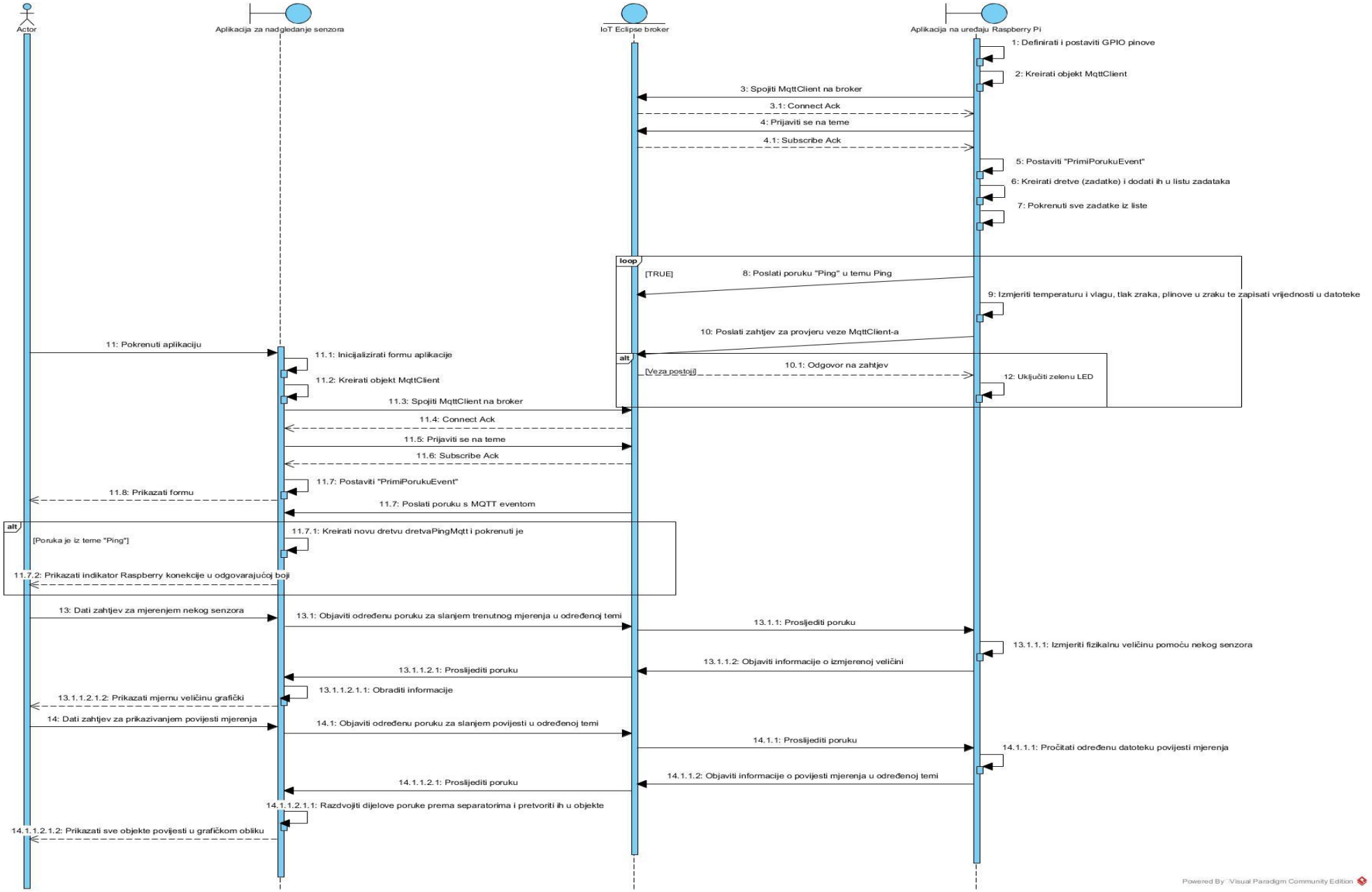
Dakle, napon od 5V prolazi kroz otpornik od 220Ω koji stvara mali pad napona i pruža otpor struji koja ulazi u senzor. Otpornik ne utječe na performanse ni na mjerenje, ali služi kao zaštita od prevelike ulazne struje. Na uzemljenje (GND, engl. *Ground*), GND pin senzora, spaja se GND pin Raspberry Pi uređaja. Data pin senzora je izlazni digitalni pin koji, ovisno o trenutnoj temperaturi i vlažnosti zraka, daje odgovarajuće binarne kombinacije koje određenim pretvorbama tvore decimalne vrijednosti koje ljudi znaju čitati. Data pin je, kako sam spomenuo, digitalni izlazni pin koji je direktno spojen na pin GPIO21 Raspberry Pi uređaja, s toga GPIO21 će biti definiran kao ulazni pin. Senzor radi na način kao što je to nacrtano na

slici 2:„Načelo mjerenja fizilakne veličine pretvorbom u električni signal“, izlazni signal računalo zna lako čitati ako se zna kako izgleda izlazni signal i koja je frekvencija slanja bitova.

GPIO7 i GPIO8 će biti definirani kao izlazni pinovi uređaja na koje su spojeni otpornici od 470Ω smanjuju preveliku ulaznu struju u LED diode (engl. *Light Emitting Diode*). LED diode će biti aktivirane ovisno o tome kako će aplikacija njima upravljati, odnosno svijetlit će ovisno o tome da li je izlazni signal na pinovima visoki ili niski. Zelena LED dioda indicira može li Raspberry Pi slati poruke na MQTT broker, a crvena indicira da li je aplikacija na uređaju pokrenuta.

5.2. Dijagram slijeda i opis rada aplikacija

U ovom poglavlju postaviti ću sliku dijagrama slijeda aplikacija i prikazati kako aplikacije komuniciraju, te kako MQTT broker sudjeluje u svemu tome. Biti će prikazano što se događa u aplikacijama u trenutku kada aktor pritisne neki gumb, no više ću se fokusirati na samu razmjenu poruka nego na detaljnom crtanju rada samih aplikacija. Pošto je slika vrlo velika, postaviti ću je na jedan poseban papir radi lakše čitljivosti.



Slika 6: Dijagram slijeda rada aplikacija (autorski rad)

Na slici „Dijagram slijeda rada aplikacija“ nisu stavljeni svi mogući scenariji, npr. nije detaljno nacrtano da neki aktor (korisnik) prvo mora uključiti uređaj Raspberry Pi i dati mu pristup Internetu. Također nije detaljno nacrtano što se događa točno između brokera i pošiljatelja (signalne poruke), ali je ovo objašnjeno kod poglavlja „QoS“, točnije QoS 1, jer tu razinu koristim u svojim aplikacijama. Sve ovo ću sada detaljnije pojasniti i opisati sliku, te nadodati neke dijelove koje nisam nacrtao u dijagramu slijeda, jer ih nije imalo smisla crtati.

5.2.1. Inicijalizacija aplikacije na uređaju Raspberry Pi

U trenutku uključivanja uređaja Raspberry Pi, konfigurirano je da se automatski pokreće i izrađena aplikacija koja služi za mjerenje temperature i vlage. Pokretanjem aplikacije prvo se definiraju i postavljaju GPIO pinovi za rad sa senzorom i LED diodama (pinovi prema slici 4: Shema završnog rada). Dakle pinovi GPIO7 i GPIO8 postavljaju se kao izlazni pinovi uređaja, odnosno kao ulazni pinovi za LED diode. Pin GPIO8 (LED crvena) odmah se postavlja na napon visoke vrijednosti (3.3 V) i dioda odmah počinje svijetliti što indicira da je aplikacija pokrenuta.

Nakon toga aplikacija kreira objekt tipa MqttClient koji dolazi s C# bibliotekom „M2Mqtt“. Novonastalom objektu se u konstruktor prosljeđuju parametri redom: broker tipa tekst (engl. *string*, „iot.eclipse.org“), brojčana vrijednost *porta* na kojem je broker (engl. *integer*, int, MqttSettings.MQTT_BROKER_DEFAULT_SSL_PORT), logička vrijednost koja se koristi da se objektu da na znanje kako se kreira sigurna komunikacija (engl. *bool*, true) koja može biti *true* ili *false*. Zadnji argument konstruktora je verzija SSL (eng. Secure Sockets Layer), te predstavlja enumerator kojem se tekstualna vrijednost pretvara u brojčanu. Brojčana vrijednost enumeratora koji ja koristim je 4 (MqttSslProtocols, MqttSslProtocols.TLSv1_2). Konstruktor naravno može biti i kraći, pa bi u tom slučaju primao samo jednu vrijednost i to ime brokera, no ovako je sigurnije. Sljedeće što se izvodi u aplikaciji je spajanje kreiranog objekta na broker koje sam izveo kao što je na slici koja slijedi.

```
private void KreirajKlijenta()
{
    if (this.Lozinka != "" && this.Username != "")
    {
        MqttKlijent.Connect(Guid.NewGuid().ToString(), this.Username, this.Lozinka);
    }
    else if (this.Username != "")
    {
        MqttKlijent.Connect(this.Username);
    }
    else
    {
        MqttKlijent.Connect(Guid.NewGuid().ToString());
    }
}
```

Slika 7: Metoda „KreirajKlijenta“, klase „MqttVeza“ (autorski rad)

Kao što možemo vidjeti na prošloj slici, postoje tri vrste spajanja objekta. Ukoliko je u konstruktor objekta klase „MqttVeza“ proslijeđena lozinka i korisničko ime, klijent se povezuje tako da dobije „guid“ kod koji mu služi kao identifikator, prema Paolu Patiernu (Paolo Patierno, MQTT Client Library Encyclopedia – M2Mqtt, bez dat.), te mu se pripišu proslijeđeno korisničko ime i lozinka. Ukoliko je u konstruktor spomenute klase proslijeđeno samo korisničko ime i ništa više, jedinstveni identifikator korisnika postaje baš to ime. Ukoliko se nije proslijedilo ni korisničko ime ni lozinka, identifikator korisnika postaje „guid“ kod, bez pridružene lozinke i imena. Nakon uspješnog povezivanja na broker, broker vraća potvrdu o spajanju (engl. *Connect Ack*) natrag aplikaciji.

Nakon što je klijent kreiran spomenutom C# metodom, slijedi pretplata na teme ugrađenom metodom objekta MqttClient, koja se naziva jednostavno „Subscribe“. Metoda kao argumente prima polje tipa tekst (*string*) kao popis imena tema i polje tipa bajt (engl. *Byte*) koje reprezentira razine kvalitete usluge (QoS) za svaku navedenu temu iz prvog argumenta (temi u polju s indeksom 0 pridružuje se QoS u polju s indeksom 0, itd., **QoS ne mora biti jednak za sve teme**). Nakon pretplate, broker vraća potvrdu o pretplaćivanju (engl. *Subscribe Ack*) natrag aplikaciji.

Nakon svih spomenutih radnji i akcija, još su preostale neke važne stvari koje treba napomenuti i izvršiti, a jedna od njih je kreirati osluškivač događaja (engl. *Event handler*) koji sam nazvao „PrimiPorukuEvent“, te ga je potrebno „zakačiti“, tj. pričvrstiti (engl. *Attach*) za događaj mqttKlijent.MqttMsgPublishReceived. Metoda koja se naziva „PrimiPorukuEvent“, odnosno osluškivač događaja, pokreće se svaki puta kada broker pošalje poruku u bilo koju temu na koju je klijent (u ovom slučaju Raspberry Pi) pretplaćen. U ovom osluškivaču je kreiran kod koji, po primitku određene poruke u određenoj temi, radi stvari koje je korisnik zatražio. U daljnjem djelu rada će sve biti detaljnije objašnjeno.

5.2.2. Kreiranje i opis dretvi aplikacije na uređaju

Kasnije, nakon navedene inicijalizacije, kreira se pet zadataka (engl. *Task*), svaki od njih se obavlja svojom dretvom (engl. *Thread*). Aplikacija kreira dretvu za Internet provjeru, koja radi beskonačno dugo, a zadatak joj je da svakih 400 ms provjerava ima li prije spomenuti objekt tipa MqttClient pristup Internetu (MqttKlijent.IsConnected). Ukoliko ima pristup Internetu aplikacija šalje visoki napon na GPIO7. Kao što možemo vidjeti na slici „Shema završnog rada“ to je pin na koji se spaja zelena LED dioda. Dakle, zelena LED dioda indicira ima li aplikacija vezu s brokerom. Ukoliko veze nema, odnosno uvjet MqttKlijent.IsConnected je neistinit, na GPIO7 se šalje signal niskog napona, te se LED dioda gasi. Još jedna dretva koja se odmah pokreće i radi paralelno sa svim ostalim zadacima je ona koja izvršava zadatak slanja poruke

„Ping“ u MQTT temu „Ping“. Dretva radi na način da pretvori tekst „Ping“ u polje bajtova (byte[]), te onda beskonačno dugo, svakih jednu sekundu u temu „Ping“ šalje bajtove koji, kada se primateljskoj strani pretvaraju u tekst, kreiraju riječ „Ping“. Preostale tri dretve koje se kreiraju imaju zadatak da ponovo, kao i prve dvije dretve, traju beskonačno dugo, i da svakih pet minuta mjere fizikalnu veličinu za koju su odgovorne, te taj podatak, odnosno podatke, zapišu u odgovarajuće tekstualne datoteke. Kada se dretve spremanja povijesti mjerenja pokrenu, čekaju jednu minutu, te nakon toga uzmu podatke sa senzora, te ako je npr. kod senzora temperature i vlage, vlaga različita od nule, zapiše trenutno vrijeme, temperaturu i vlagu u jednu varijablu tipa *string* u formatu npr.: 07/29/2018 1:04 PM;26T83V. Kao što vidimo, u varijabli imamo zapisane separatore ';', 'T', 'V', zbog toga jer se preko MQTT-a šalje tekst (ne objekti), pa su separatori potrebni da bi kasnije klijentska aplikacija (aplikacija na Windows računalu) mogla znati koji dio teksta je što. Slično se izvodi i za senzor plinova u zraku i za senzor tlaka zraka. No kako su ti senzori virtualni, mjerenje je malo drugačije. Naime to su jednostavni generatori slučajnih brojeva napravljeni tako da prikazuju donekle realno moguće podatke. Nakon toga aplikacija zapisuje sadržaj varijable u određenu tekstualnu datoteku (ovisno o kojem se senzoru radi) uz pomoć C# klase „StreamWriter“ iz System.IO biblioteke, naravno ako je prije navedeni uvjet ispunjen. Ako uvjet nije ispunjen, podaci se nisu zapisali, a dretva čeka preostalih četiri minute nakon kojih kreće opet ispočetka.

Nisam stavio čekanje od pet minuta odjednom, nego sam ga „rascjepkao“ na komade od jedne minute i četiri minute zbog toga da se senzor (stvarni senzor) može pripremiti za rad, ukoliko se je aplikacija tek pokrenula. Spomenuo sam prije uvjet da se u datoteku zapišu samo podaci ako je vlaga različita od nule. Razlog tome je taj da se ponekad zna dogoditi da imamo nevaljale podatke, vlaga i temperatura su nula, pa sam stavio uvjet da se podaci zapišu samo ako je vlaga različita od nule, jer temperatura realno može biti nula. Na dijagramu slijeda se ove dretve vide u dijelu gdje je „loop“ okvir između aplikacije na Raspberry Pi uređaju i IoT Eclipse brokera, a poruke sam namjerno prikazao kosom crtom (poruke s čekanjem, engl. *Delay message*), samo kako bih naglasio da postoji namjerno čekanje u dretvama, iako te poruke u stvarnosti nemaju velikog kašnjenja što se tiče slanja.

5.2.3. Inicijalizacija klijentske aplikacije

U trenutku kada korisnik (aktor) pokrene aplikaciju (klijentska aplikacija, aplikacija koja se nalazi na Windows računalu), prvo što se događa je inicijalizacija svih komponenti aplikacije i glavne forme. Kreira se objekt tipa MqttClient, koji se spaja na broker i prijavljuje na teme, a sve je isto kao i kod aplikacije na uređaju Raspberry Pi. Iako je sve gotovo isto, nisam mogao napraviti jednu „dll“ klasu (engl. *Dynamic Link Library*) koju bih koristio kod obje aplikacije, jer ne postoji *class library* koji bi mogao raditi i s UWP (engl. *Universal Windows Platform*, koristi

se za kreiranje aplikacija koje će biti implementirane na udaljenim Windows uređajima) i s .NET okvirom (engl. .NET *framework*). Jedina razlika kod ove aplikacije u odnosu na aplikaciju koja je na uređaju Raspberry Pi (u pogledu maloprije navedenih koraka) je način kreiranja objekta tipa MqttClient, odnosno razlika je u argumentima. Kod ove aplikacije je potrebno, ukoliko koristimo prije spomenuti SSL / TLS, unijeti X509 certifikate. S toga, konstruktor objekta izgleda ovako: prvi argument je ime brokera tipa *string*, drugi je *port* na kojem je broker i on je tipa *integer* (cijeli broj), sljedeće je atribut koji reprezentira sigurnost veze, poprima vrijednost istine ili laži, odnosno *true* ili *false* (do sada je sve isto kao i kod aplikacije na uređaju). Sljedeći argument je tzv. caCert koji je tipa X509Certificate i reprezentira javni ključ (CAcert wiki, 2017.), argument nakon toga je clientCert koji je isto tipa X509Certificate i on je u realnim slučajevima zapravo datoteka pomoću koje server prepoznaje klijenta (SSL2BUY global ssl provider, bez dat.). Ovi certifikati služe kod enkripcije i prepoznavanja klijenta, te jamče veliku sigurnost podataka koji se šalju Internetom. Nakon navedenih koraka, potrebno je kreirati, isto kao kod aplikacije na uređaju, jedan osluškivač događaja i povezati ga s događajem primanja poruke. Naravno kod ove aplikacije programska logika unutar ovog osluškivača je puno drugačija nego kod aplikacije na uređaju Raspberry Pi. Po postavljanju osluškivača, glavna forma može se prikazati korisniku.

5.2.4. Međudjelovanje dviju aplikacija

Daljnji koraci i radnje aplikacije ovise o tome što korisnik želi (na što klikne), no sa sigurnošću znamo radnju koja će sada biti opisana. Spomenuo sam da aplikacija na Raspberry Pi uređaju svaku sekundu šalje poruku „Ping“ u temu „Ping“. U trenutku kada klijentska aplikacija primi spomenutu poruku pokreće se dretva kojoj je zadatak da postavi indikator Raspberry Pi konekcije na MQTT broker (panel kontrolu u gornjem desnom kutu aplikacije) u zelenu boju i čeka 5 sekundi, nakon kojih promijeni boju spomenutog panelu u crvenu. Ukoliko više ne dođe poruka „Ping“, panel ostaje crvene boje, a u suprotnome boja se ponovno postavi u zelenu boju. Poruke dolaze svaku sekundu, a dretva traje minimalno 5 sekundi, tu nastaje problem da se svaku sekundu kreira nova dretva dok stara još nije završila. Ovaj problem sam riješio tako da se nova dretva može kreirati tek kad stara postavi zastavicu da je završila, odnosno varijablu tipa bool postavi na istinitu tvrdnju (engl. *true*). U trenutku dolaska poruke ispituje se stanje ove varijable. Ukoliko je uvjet istinit dretva se kreira, ukoliko nije ne izvodi se ništa.

Ako korisnik želi informacije o trenutnoj temperaturi i vlazi, iz padajućeg izbornika odabire „Temperatura i vlaga“, te se klikom na gumb „Izmjeri trenutne vrijednosti“, šalje poruka „MjeriTemperaturaVlaga“ (u obliku polja bajtova) u temu „TemperaturaVlaga“. Broker tu poruku prihvaća te ju propagira dalje do svih uređaja koji su pretplaćeni na spomenutu temu, a između

tih uređaja nalazi se i Raspberry Pi. Raspberry Pi primitkom ove poruke očitava temperaturu i vlagu sa senzora Dht11. Mjerenje je obavljeno uz pomoć C# biblioteke koju je kreirao Daniel Porrey, a zove se „Dht“ i služi za rad s Dht11 i Dht12 senzorima. Nakon mjerenja temperature i vlage, dobiveni podaci se šalju na broker u obliku istom kako se i zapisuju u datoteku, dakle s tri separatora ';', 'T' i 'V'. Broker prosljeđuje poruku klijentskoj aplikaciji, odnosno svima koji su na tu temu pretplaćeni. Aplikacija tada mora pretvoriti polje bajtova u tekst (*string*) te tada pretvoriti primljen tekst u objekt već prije spomenutom metodom uz pomoć separatora. Aplikacija nakon pretvaranja podataka u objekt zapisuje taj isti objekt u listu sučelja (engl. *interface*) IPovijestMjerenja, kako bi aplikacija mogla raditi sa svim senzorima, ne samo s jednim. Nakon toga aplikacija prolazi kroz petlju gdje se za svaki podatak u listi provjerava kojeg je tipa. Ukoliko je tipa temperatura i vlaga (PovijestTempVlaga), onda se taj podatak korisniku prikazuje tako da se na kružnoj traci napretka (engl. *Circular progress bar*) iscrtava postotak vlažnosti i tako da se temperatura prikazuje na mjerilu (engl. *Gauge*) pomakom kazaljke na određenu brojčanu vrijednost. I mjerilo i progress bar su paketi preuzeti s NuGet paketa („AGauge_V2“, „CircularProgressBar“). Ukoliko korisnik želi da mu se prikaže mjerenje plinova u zraku ili tlaka zraka, odabere drugi element iz padajuće liste u aplikaciji, te se klikom na spomenuti gumb „Izmjeri trenutne vrijednosti“, šalje poruka u temu ovisno o tome koji je senzor izabran, a poruka izgleda ovako :“Mjeri{Senzor}“, umjesto vitičastih zagrada, upisuje se traženi senzor. Aplikacija na uređaju Raspberry Pi, nakon mjerenja podataka (uzimanja slučajnih brojeva), radi gotovo isto kao i kod mjerenja temperature i vlage, jedino što separatori i tema na koju se mjerenje šalje nisu jednaki.

Gotovo isto se događa kada korisnik zatraži da mu se prikaže povijest mjerenja senzora Dht11 (temperatura i vlaga). Dakle korisnik odabire iz padajuće liste element, te klikom na gumb „Grafički prikaži povijest“ šalje se poruka kao skup bajtova u temu "PovijestTemperaturaVlaga". U trenutku kada poruka stigne do aplikacije na uređaju Raspberry Pi, aplikacija pomoću klase File i njezine metode ReadAllLines, iz biblioteke System.IO, čita sve retke datoteke koji se zapisuju u listu. Kako se lista ne može slati pomoću MQTT protokola, sve podatke iz liste aplikacija zapisuje u varijablu tipa *string* koja se pretvara u skup bajtova i šalje na broker. Broker poruku objavljuje svima na toj temi, pa tako i našoj klijentskoj aplikacij. Primitkom ove poruke, aplikacija pretvara bajtove u *string* i „cjepka“ tekst po već spomenutom principu na temelju spomenutih separatora. Kreira se instanca grafa iz biblioteke OxyPlot preuzete s NuGet paketa, potom se primljena poruka nakon cjepkanja sprema u novu instancu već prije navedene liste sučelja IPovijestMjerenja. Nakon toga izvršava se kreirana metoda koja crta graf na temelju listePovijesti (implementirano na ovaj način da lako radi s ostalim senzorima). Svakim pozivom metode kreiraju se dvije funkcijske serije (engl. *Function series*) koje se mogu shvatiti kao linije grafa, od kojih je jedna za vlagu,

a druga za temperaturu. Za svaki objekt liste na grafu kreira se točka za svaku seriju. Prolaskom kroz sve podatke, modelu grafa se dodaju linije, a kontroli plotView se dodaje model koji prikazuje podatke. Svaki puta kada pristigne povijest mjerenja kreira se nova instanca grafa zbog toga da se dodaju ispočetka novi model i linije postojećem plotView-u. Ukoliko to ne bi bilo tako, u aplikaciji bismo imali preklapanja grafova, trošili bismo svaki puta sve više i više memorije, a ovako se to može izbjeći. Isto vrijedi i za druge senzore, osim što su različite teme i separatori. Također, kod crtanja grafa povijesti plinova u zraku, crtaju se četiri, a kod tlaka zraka jedna linija, a ne kao kod temperature i vlage dvije. Poruka koju klijentska aplikacija šalje, da bi dobila povijest mjerenja od serverske aplikacije izgleda ovako „Povijest{Senzor}“, umjesto vitičastih zagrada dolazi ime senzora koje može biti „TemperaturaVlaga“, „PlinUZraku“, „Tlak“. Ukoliko poruka glasi drugačije, serverska aplikacija neće vratiti nikakve podatke, te ovo vrijedi i za mjerenje trenutnih vrijednosti.

U aplikaciji postoji i jedna dretva koja provjerava da li je uređaj s određenom IP adresom spojen na Internet, odnosno odgovara li taj uređaj na provjere aktivnosti (*ping*). Korisnik unosi IP adresu, te ukoliko je ona ispravna (ako je u formatu ###.###.###.###) dretva može započeti svoju radnju, ukoliko nije ispravna panel (indikator Internetske veze uređaja Raspberry Pi na formi) je žute boje. Radnja dretve započinje tako da varijablu koja se zove „ZaustaviDretvu“ tipa *bool* postavi na neistinitu vrijednost (engl. *False*), a nakon toga kreira objekt koji je tipa Ping iz C# biblioteke System.Net.NetworkInformation. Dretva radi beskonačno dugo, a uvjet zaustavljanja joj je provjera spomenute varijable „ZaustaviDretvu“. Varijabla se postavi na istinitu vrijednost (engl. *True*) onda kada korisnik unese drugu IP adresu u za to predviđen prostor. Dakle dretva radi beskonačno dugo na sljedeći način: čeka jednu sekundu te onda kreira objekt tipa PingReply, iz iste biblioteke kao i Ping, koji dobiva vrijednost koju vraća metoda Ping.Send. Nakon ovog slanja signala, provjerava se da li je status objekta koji je tipa PingReply uspješan ili neuspješan. Ukoliko je uspješan, boja indikatora postavlja se u zeleno, u suprotnome postavlja se u crvenu boju. Ukoliko se je dogodila iznimka tijekom izvođenja ovog zadatka, indikator se također postavlja u crvenu boju, a dretva kreće ispočetka.

5.3. Pregled važnijih klasa i metoda

U ovom odjeljku ću ukratko prikazati neke važnije metode na način da ću postaviti njihove slike na početku svakog potpoglavlja, te opisati metode i argumente koje primaju i gdje ih koristim. U produžetku naslova potpoglavlja, pošto se radi o metodi, stajati će naziv klase u kojoj je kreirana, a u zagradi aplikacija u kojoj se koristi (klijent ili Raspberry). Na početku svakog potpoglavlja biti će slika koda iz aplikacije, a ispod nje kratki opis rada.

5.3.1. Metoda „PosaljiPoruku“ klase PomocnaKlasa (klijent / Raspberry)

```
public static void PosaljiPoruku(string poruka, Enumeracije.Teme tema)
{
    MqttClient mqttKlijent = MqttVeza.DohvatiMqttClient();
    byte[] porukaZaSlanje = Encoding.UTF8.GetBytes(poruka);
    try
    {
        mqttKlijent.Publish(tema.ToString(), porukaZaSlanje);
    }
    catch (Exception)
    {
        MessageBox.Show("Nema pristupa internetu, ili je server isključen");
    }
}
```

Slika 8: Metoda "PosaljiPoruku" (autorski rad)

Metoda je statična i vraća *void*, odnosno može se pozivati bez da se kreira objekt tipa PomocnaKlasa i metoda ne vraća nikakav rezultat. Prvi parametar koji ova metoda prima je poruka koja se šalje, u tekstualnom formatu, a drugi argument je sama tema u koju će poruka biti poslana. Napravio sam klasu u kojoj imam enumeracije svih tema i senzora da olakšam korištenje samog koda aplikacije. Objekt tipa MqttClient nam je potreban zapravo za samo slanje poruke metodom MqttClient.Publish, a do tog objekta dolazimo statičnom metodom iz klase MqttVeza, kao što je vidljivo na slici. Također, vidljivo je da se poruka ne šalje kao tekst (*string*) nego kako skup (polje) bajtova koji se dobiva C# metodom Encoding.UTF8.GetBytes. Spomenuta metoda MqttClient.Publish, koja je zapravo srž opisivane metode, prima navedene parametre, dakle prima parametar tipa *string* koji predstavlja temu i parametar tipa polje bajtova (byte[]) koji je poruka koja se šalje.

5.3.2. Metoda „Spoji“ klase MqttVeza (klijent / Raspberry)

```
public MqttClient Spoji(string[] teme = null)
{
    if (teme != null) this.Teme = teme;

    else
        Teme = Enumeracije.DohvatiPoljeTema();

    for (int i = 0; i < Teme.Count<string>(); i++)
    {
        Qos.Add(MqttMsgBase.QOS_LEVEL_AT_LEAST_ONCE);
    }
    try
    {
        MqttKlijent = new MqttClient(this.Broker, MqttSettings.MQTT_BROKER_DEFAULT_SSL_PORT, true,
            new X509Certificate(), new X509Certificate(), MqttSslProtocols.TLSv1_2);

        KreirajKlijenta();

        MqttKlijent.Subscribe(Teme, Qos.ToArray());

        return MqttKlijent;
    }
    catch (Exception)
    {
        return null;
    }
}
```

Slika 9: Metoda "Spoji" (autorski rad)

Metoda nije statična i tip koji vraća na kraju izvođenja je MqttClient, a trenutno ne koristim ovaj povratni tip kao neki objekt u svojoj aplikaciji, jer imam metodu kojom dohvaćam taj isti objekt, no ukoliko bi se aplikacija mijenjala u budućnosti, može ostati tako zbog fleksibilnosti. Ova metoda je jedna od onih bez koje cijela aplikacija uopće ne bi mogla raditi, odnosno ne bi bilo povezivanja na broker. Kao što možemo vidjeti, argument metode nije obavezan zato jer sam kao globalnu privatnu varijablu tipa polje *stringova* koja se zove „Teme“ već postavio na onih sedam početnih tema (TemperaturaVlaga, Tlak, PlinUZraku, PovijestMjerenjaTempVl, PovijestMjerenjaTlak, PovijestMjerenjaPlin, Ping). Uz ovo navedeno ide i prva linija koda ove metode, a to je da ukoliko argument nije unesen, odnosno ako je ostao na *null* vrijednosti (nije inicijaliziran), tada se ništa ne događa. No ukoliko bismo postavili neki argument u pozivu metode, prije spomenuta varijabla „Teme“ bi postala jednaka tom argumentu. Sljedeća stvar koja je obavezna, vrlo važna i mora se odraditi uvijek kada se želimo pretplatiti na teme je da svaku temu, koju smo unijeli u polje tema, trebamo povezati s već prije spomenutom i objašnjenom kvalitetom usluge (QoS). Trenutno imamo sedam tema u polju, s toga su indeksi polja 0, 1, 2, 3, 4, 5, 6 pa za svaki ovaj broj moramo dodati QoS. Koji se može dodavati naravno ručno ili ovako kako sam to ja učinio, pomoću jedne petlje koja se ponavlja toliko puta koliko ima tema (u ovom slučaju sedam puta). Za svaku iteraciju petlje, u globalnu, privatnu listu (List<byte>) QoS dodaje se razina kvalitete pružanja usluge (QoS). Ja za svaku temu koristim istu razinu usluge i to QoS1. Naravno svaka tema može imati različit QoS. Slijedi kreiranje klijenta koje je već detaljno bilo objašnjeno kod podnaslova inicijalizacija klijentske aplikacije, također metoda „KreirajKlijenta“ je objašnjena kod potpoglavlja „Inicijalizacija

aplikacije na uređaju Raspberry Pi“. Na kraju predstoji još da se objekt tipa MqttClient pretplati na teme uz odgovarajuće QoS, oba parametra metode za pretplatu na teme su polja. Ukoliko se je dogodila iznimka kod bilo kojeg navedenog koraka, metoda vraća *null* umjesto MqttCleint objekta, a iznimka se najčešće događa ukoliko računalo nema pristupa Internetu.

5.3.3. Metoda ZapisiPodatkeUListu klase „Povijest“ (klijent)

```
private static List<double> listaVrijednostiVarijabli = null;

private static List<int> listaRednihBrojevaVarijable = null;
private static List<IPovijestMjerenja> returnList = null;

2 references
public static List<IPovijestMjerenja> ZapisiPodatkeUListu(string povijestMjerenja, Enumeracije.Senzori senzor)
{
    returnList = new List<IPovijestMjerenja>();

    DateTime vrijemeIzStringa = new DateTime();

    int brojSeparatora = DohvatiBrojSeparatora(senzor);

    listaRednihBrojevaVarijable = DodajRedneBrojeveUListu(brojSeparatora);

    var splitString = povijestMjerenja.Split(DohvatiSeparatore(senzor));

    for (int brojac = 0; brojac < splitString.Length - 1; brojac += brojSeparatora)
    {
        vrijemeIzStringa = DohvatiVrijemeIzStringa(splitString[brojac]);

        listaVrijednostiVarijabli = KreirajVarijable(splitString);

        DodajObjektUListu(senzor, vrijemeIzStringa);

        PovecajRedniBrojVarijable(brojSeparatora);
    }
    return returnList;
}
```

Slika 10: Metoda "ZapisiPodatkeUListu" (autorski rad)

Ovu metodu sam već puno spominjalo, ali joj nisam navodio ime. Naime ova metoda je algoritam koji mi služi za ono spomenuto „cjepkanje“ dolazne poruke mjerenja i povijesti. Algoritam je radi lakše čitljivosti razdijeljen tako da u sebi sadrži još nekoliko drugih metoda koje ću također opisat. Prvo što primjećujemo je da ova metoda za povratni tip ima listu koja je tipa IPovijestMjerenja, što je zapravo sučelje (engl. *interface*), a to je zbog toga da se u nju mogu spremiti podaci svih senzora koji to sučelje nasljeđuju. Prvi argument metode je naravno podatak koji želimo zapisati u listu kao objekt. Dakle, taj prvi argument je ili cijela povijest mjerenja ili trenutno mjerenje senzora i to u tekstualnom obliku (engl. *string*). Drugi argument je enumeracija senzora koja govori o tome kakve podatke, kojeg senzora, metoda može očekivati. Ovaj argument zapravo samo služi za dohvaćanje separatora. Važno je napomenuti da nemaju svi senzori isti broj separatora i separatori ne moraju biti nužno jednaki znakovi. Opisati ću metodu na primjeru mjerenja temperature i vlage, pa su zbog toga u pitanju tri separatora (senzor plinova u zraku ih ima čak pet). Prvi važniji korak je dohvaćanje broja

separatora, što je vrlo jednostavna privatna metoda koja samo uzima iz klase (u ovom slučaju) povijesti temperature i vlage broj separatora i vraća ga. Ukoliko bi bio neki drugi senzor postojala bi druga klasa u kojoj se dohvaća broj separatora. Nakon ovog koraka primjećujemo da u već prije kreiranu listu rednih brojeva dodajemo redne brojeve na temelju broja separatora, a metoda izgleda ovako:

```
private static List<int> DodajRedneBrojeveUListu(int brojSeparatora)
{
    List<int> returnLista = new List<int>();
    for (int i = 1; i < brojSeparatora; i++)
    {
        returnLista.Add(i);
    }
    return returnLista;
}
```

Slika 11: Metoda "DodajRedneBrojeveUListu" (autorski rad)

Dakle, metoda sa slike je privatna metoda koja vraća listu u kojoj se nalaze cijeli brojevi (*integer*) i na temelju broja separatora jednostavno dodaje brojeve od 1 pa do prvog manjeg broja od ukupnog broja separatora (u slučaju senzora temperature i vlage broji od 1 pa do 2, oba broja su uključena). Ovo radim kako bih imao veću kontrolu nad daljnjim povećanjem brojeva, a uskoro ću malo više to pojasniti na primjeru.

Sljedeći korak glavne metode ovog podnaslova je razdvojiti (engl. Split) povijest mjerenja na temelju separatora u jedno polje *stringova* koje sam nazvao „splitString“. Tekst se razdvaja C# metodom koja se zove Split, a argument koji prima je polje separatora (char[]). Ja u svojem primjeru prosljeđujem metodu koja se zove „DohvatiSeparatore“, a ona tada na temelju danog joj senzora vraća separatore. Metoda je vrlo jednostavna, pa je ne treba detaljnije objašnjavati. Nakon ovih koraka kreće glavni dio metode, a to je petlja koja broji od nule pa sve do prvog manjeg broja ukupnog broja polja *stringova* koje sadrži „rascjepkan“ tekst umanjen za 1. Vrijeme iz *stringa* nalazi se na prvoj poziciji trenutnog *stringa*, odnosno ako imamo 3 separatora, nalazi se na nultoj poziciji, pa na trećoj, šestoj, itd. Zbog toga za dohvaćanje vremena koristim metodu „DohvatiVrijemeIzStringa“ koja za argument prima tekst koji je zapravo datum, ali u tekstualnom obliku. U polju *stringova* varijable „splitString“ se na poziciji 0 nalazi prvo vrijeme, pa mogu koristiti brojač petlje za traženje datuma u polju. Brojač se na kraju svake petlje povećava za broj separatora. Metoda „DohvatiVrijemeIzStringa“ jednostavno samo pretvara tekst datuma koji je u en-US formatu u datum. Nakon toga slijedi dodavanje vrijednosti u prije kreiranu, globalnu, privatnu listu brojeva (double), a dodaju se u listu zapravo vrijednosti koje su u istom skupu kao i datum koji smo trenutno obrađivali (dakle ako smo obrađivali datum koji je na poziciji 0, sljedeće vrijednosti u ovom primjeru sa senzorom temperature i vlage, biti će one koje su na pozicijama 1 i 2. I to bi radilo za bilo koji senzor, sa

bilo koliko separatora). Metoda koja dodaje varijable u listu izgleda ovako:

```
private static List<double> KreirajVarijable(string[] splitString)
{
    List<double> returnLista = new List<double>();
    foreach (var redniBroj in listaRednihBrojevaVarijable)
    {
        if (double.TryParse(splitString[redniBroj], NumberStyles.Number, CultureInfo.InvariantCulture, out double rezultat))
        {
            returnLista.Add(rezultat);
        }
    }
    return returnLista;
}
```

Slika 12: Metoda „KreirajVarijable" (autorski rad)

Metoda na slici iznad, vraća listu brojeva (double) i u novonastalu listu zapisuje vrijednosti iz argumenta. S gledišta glavne metode koja stoji u ovom podnaslovu, globalna lista „listaVrijednostiVarijabli“ se svaki puta ispočetka postavlja na neku vrijednost (nakon iteracije petlje u glavnoj metodi, u listu se ne dodaju nove vrijednosti nego se stare brišu, a nakon toga zapišu nove). Metoda „KreirajVarijable“ radi na način da uzme sve redne brojeve i liste rednih brojeva koju sam prije spominjao, a za svaki taj redni broj u polju „rascjepkanih“ *stringova* postoji odgovarajuća pozicija (0, 1, 2). Ukoliko je pretvorba uspjela, broj se dodaje u listu, a ukoliko nije uspjela, odnosno tekst je datum ili nešto drugo, vrijednost se ne dodaje u listu.

Nakon što je gore iznad opisana metoda završila, ove vrijednosti iz liste moraju se dodati kao objekti u neku drugu listu koja će obuhvaćati i datum i vrijeme, a i vrijednosti (bez obzira koliko ih ima). Upravo ovdje nastupa sljedeća metoda koja se zove „DodajObjektUListu“. Spomenuta metoda prima dva parametra, jedan je senzor (kako bi metoda mogla znati s čime ima „posla“), a drugi je vrijeme iz rascjepkanog *stringa*. Ovisno o tome što je prvi argument, metoda u listu zapisuje objekt klase koja nasljeđuje sučelje „IPovijestMjerenja“ jer je i lista tog tipa.

Po završetku ovog koraka, odnosno metode „DodajObjektUListu“ još je samo potrebno povećati redne brojeve varijabli kako bi uvijek pokazivali na točno ona mjesta u *stringu* na koja i mora pokazivati kako bi sve funkcioniralo. Tako se svaki broj koji je u listim poveća za onoliko koliko ima separatora, to isto učini i brojač petlje.

Ovdje dolazimo do onog primjera za koji sam napisao da će još malo približiti sliku svega - dakle ukoliko imamo N mjerenja u originalnom *stringu* i imamo 3 separatora (jedan za datum, drugi za temperaturu i treći za vlagu). Cjepkanjem ovog *stringa* pomoću separatora dobije se polje *stringova* koje ima 3N zapisa kojima je sačuvan redoslijed, odnosno podaci su redom u polju zapisani tako da je na nultoj poziciji datum, na prvoj temperatura, a na drugoj vlaga, na trećoj bi onda bio opet datum, itd. U petlji imamo brojač koji ide od 0, pa se na kraju petlje poveća za 3 (broj separatora) tako da preskočimo obradu ova dva podatka koja se nalaze unutar raspona od 0 do 3. Brojač uvijek pokazuje na element u polju koji predstavlja

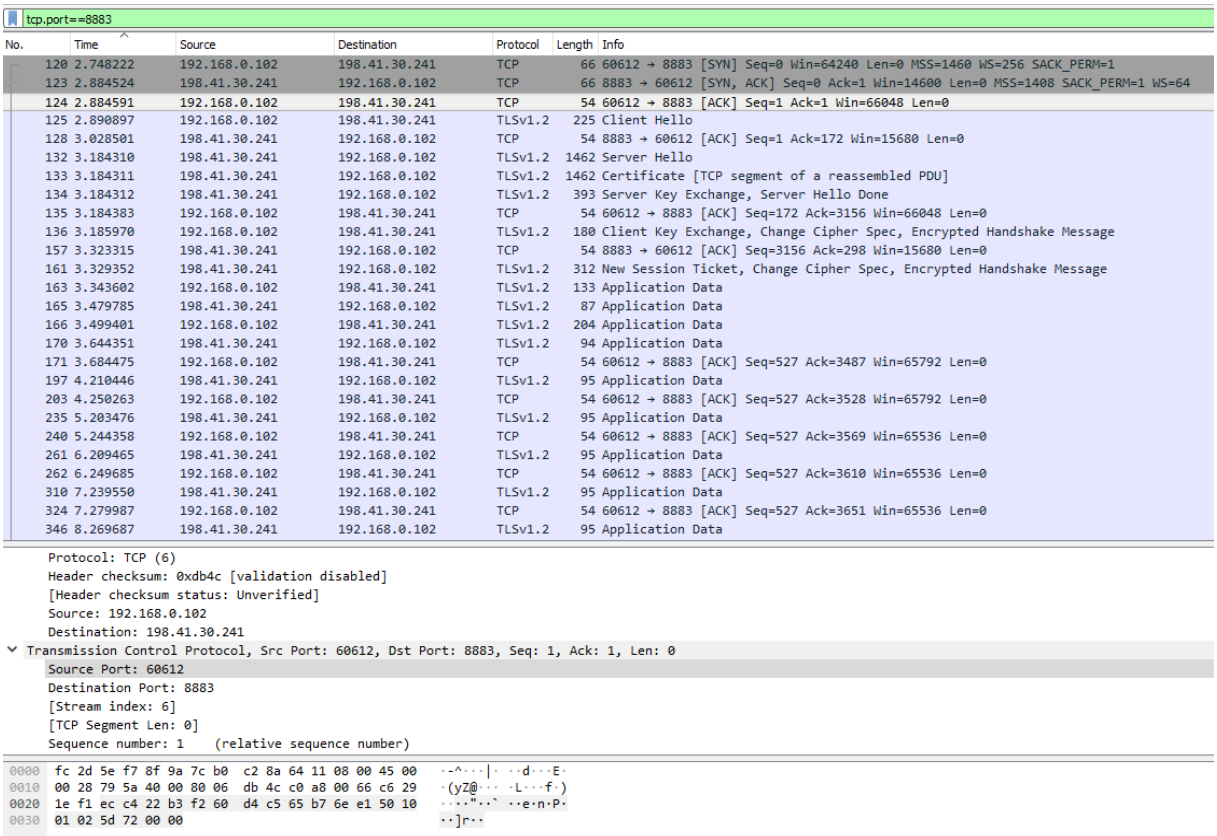
datum, a po završetku iteracije povećavaju se redni brojevi varijabli iz liste. U listi se tada nalaze brojevi 1 i 2, element na poziciji 1 označava temperaturu, a na poziciji 2 vlagu. Dakle, da bi se sve održavalo u redoslijedu, ti brojevi u listi se moraju povećati za broj separatora također kao i brojač iz petlje. Kada je brojač jednak 0 (datum), tada su redni brojevi 1 (temperatura) i 2 (vlaga), a kada je brojač jednak 3 (datum), redni brojevi su 4 (temperatura) i 5 (vlaga). Iz toga slijedi da ukoliko je brojač jednak N-3 (datum), redni brojevi su N-2 (temperatura) i N-1 (vlaga). Nikada ne dolazimo do broja N zbog toga jer je to ukupni broj elemenata u polju, a polje počinje od indeksa 0, a ne 1.

5.4. Testiranje sustava

U ovome dijelu biti će riječi o tome koliko aplikacija koristi računalnih resursa za svoje izvođenje, te će biti snimljena razmjena internet paketa između osobnog računala i brokera.

5.4.1. Pregled razmjene poruka između klijenta i brokera

Razmjena poruka, odnosno promet između klijenta i brokera snimljen je alatom Wireshark. Prikaz snimanja razmjene prometa ću prikazati sljedećom slikom i ukratko opisati što se događa, te ću pokušati povući paralelu između prije navedene teorije i stvarnog prometa.



Slika 13:Promet između klijenta i brokera (autorski rad)

Prvo što možemo primijetiti na slici iznad je to da je filter (tekst u traci zelene boje) postavljen na način da prikazuje TCP poruke i to samo one koje se odvijaju na priključku, „vratima“ (engl. *Port*) 8883. *Port* 8883 je naime već spomenuti *port*, no samo sam mu naveo ime, no ne i vrijednost. Kod potpoglavlja „Inicijalizaciija aplikacije na uređaju Raspberry Pi“ naveo sam da konstruktor objekta MqttClient prima parametar koji je brojčana vrijednost *porta* i naveo sam da sam umjesto tog parametra napisao sljedeći izraz, koji dolazi s c# M2Mqtt bibliotekom: „MqttSettings.MQTT_BROKER_DEFAULT_SSL_PORT“. Baš ovaj izraz, jer se koristi TLS/SSL sigurnosni sloj, ima vrijednost 8883, tako da se sva komunikacija odvija preko tog *porta*. Također, kod poglavlja „Kako MQTT radi?“ spomenuo sam da se MQTT komunikacija odvija preko TCP/IP protokola. Dakle, slika iznad dokazuje teoriju koju sam spomenuo i dokazuje da moja aplikacija razmjenjuje poruke preko sigurnog sloja.

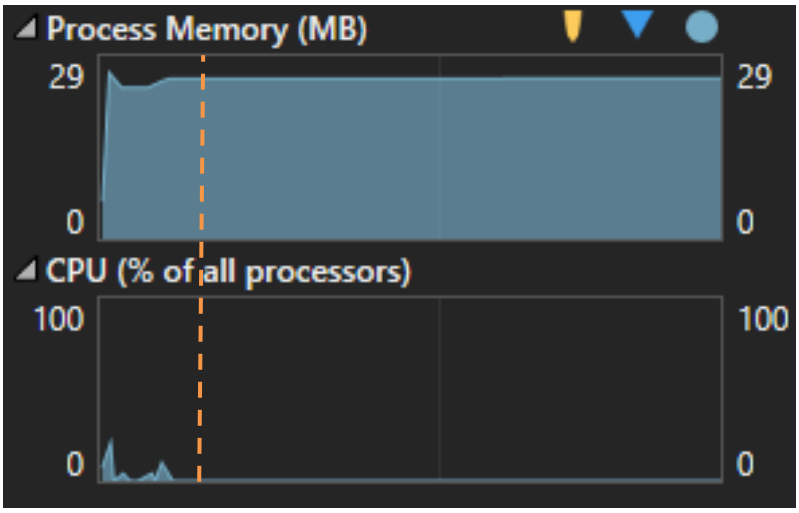
Sljedeće što je na slici vidljivo je u stupcu odredište (engl. *Destination*) i u stupcu izvor (engl. *Source*) u kojem postoje dvije različite IPv4 adrese. Adresa 192.168.0.102 je adresa koju je moje računalo posjedovalo u trenutku snimanja prometa, a adresa 198.41.39.241 je adresa koju u trenutku mjerenja posjeduje broker na koji šaljem poruke („iot.eclipse.org“). Poruka koja je na slici označena (treća poruka od vrha) ima adresu izvorišta moje osobno računalo, a to znamo zato jer je pri dnu slike sivo označen izvorišni *port,* kojem je vrijednost 60612 i to je *port* preko kojeg moja aplikacija komunicira s ostatkom svijeta. Ovo je dokaz da uistinu koristim protokol MQTT za komunikaciju jer nigdje ne vidimo izravnu komunikaciju između uređaja Raspberry Pi, koji bi imao vrlo sličnu adresu kao i moje osobno računalo jer su povezani na isti usmjernik.

Prema slici, vidi se da je dužina poruka od 54 pa sve do 1462 bajta. Ova najveća poruka koja iznosi 1462 bajta nije zapravo poruka koju jedna aplikacija šalje drugoj, već je to rukovanje između brokera i uređaja koji se spaja na broker. Možemo vidjeti da nema onog četverostrukog rukovanja kao kod QoS 2, već postoji samo jedan paket koji se šalje kao odgovor na pristiglu poruku, a to je paket ACK (engl. *Acknowledgment*) koji se šalje u trenutku kada je poruka pristigla na odredište, to je zbog toga što koristim QoS 1. Svaka poruka koja se šalje, u paketu sadrži i količinu bajtova koji se trenutno šalju (engl. *Bytes in flight*) i informaciju o tome koliko je bajtova poslano od zadnje PSH (engl. *Push*) zastavice. Ove informacije pomažu kod određivanja da li je poruka izgubljena ili ne, a ovisno o tome šalje se (ili ne šalje) ACK signalna poruka. Ukoliko je poruka izgubljena, umjesto PSH zastavice, postavlja se RST (engl. *Reset*) zastavica, a takva poruka je u alatu Wireshark označena crvenom bojom.

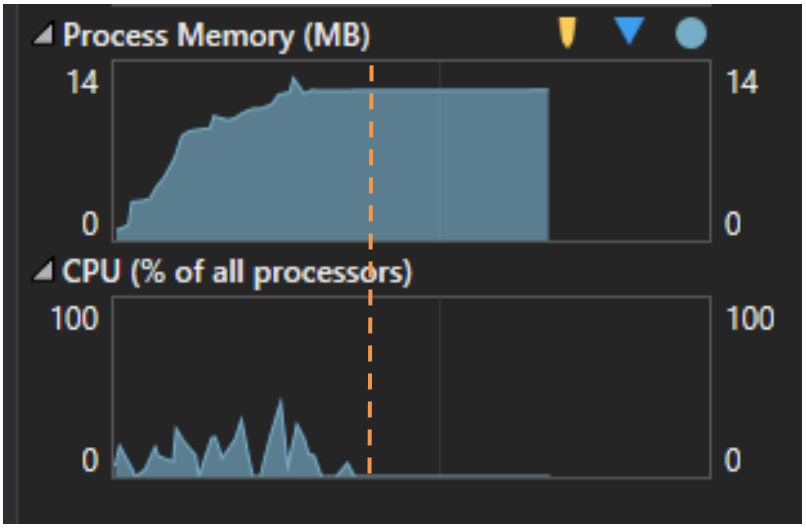
Svaka poruka, pošto je učahurena u TCP protokol, sadrži zaglavlje koje je veličine 20 bajta, također iz istog razloga. S porukom dolazi i već prije spomenuti redni broj poruke (engl. *Sequence number*), podatak o rednom broju sljedeće poruke (engl. *Next sequence number*) i ACK broj (engl. *Acknowledgment number*). Svaka poruka ima svoj jedinstveni redni broj poruke.

5.4.2. Pregled korištenja računalnih resursa

U ovom potpoglavlju govoriti ću o tome koliko MB (megabajt) memorije i koliki postotak svih procesora koriste osobno računalo i računalo Raspberry Pi za inicijaliziranje aplikacija, a koliko koriste nakon inicijalizacije, odnosno u periodu stabilnog rada. Ovdje ću postaviti slike korištenja resursa tijekom pokretanja aplikacije u tzv. „Debug“ modu alata Visual Studio 2017. Slijede slike koja prikazuje količinu korištenja resursa pri pokretanju i stabilnom radu klijentske aplikacije i aplikacije na uređaju Raspberry Pi, te će nakon toga biti ukratko opisane.



Slika 14: Pregled korištenja resursa klijentske aplikacije (autorski rad)



Slika 15: Pregled korištenja resursa serverske aplikacije (autorski rad)

Kod obje slike, jasno je vidljiva granica između stabilnog rada aplikacije i inicijalizacije. Naime, sve s lijeve strane grafa je inicijalizacija, a desno stabilni rad aplikacije. Kada govorim o „stabilnom radu“ aplikacije to znači da je u tom trenutku aplikacija postavljena i spremna za rad, čeka naredbe, poruke ili pak izvršava zadatke već pokrenutih dretvi iz inicijalizacije.

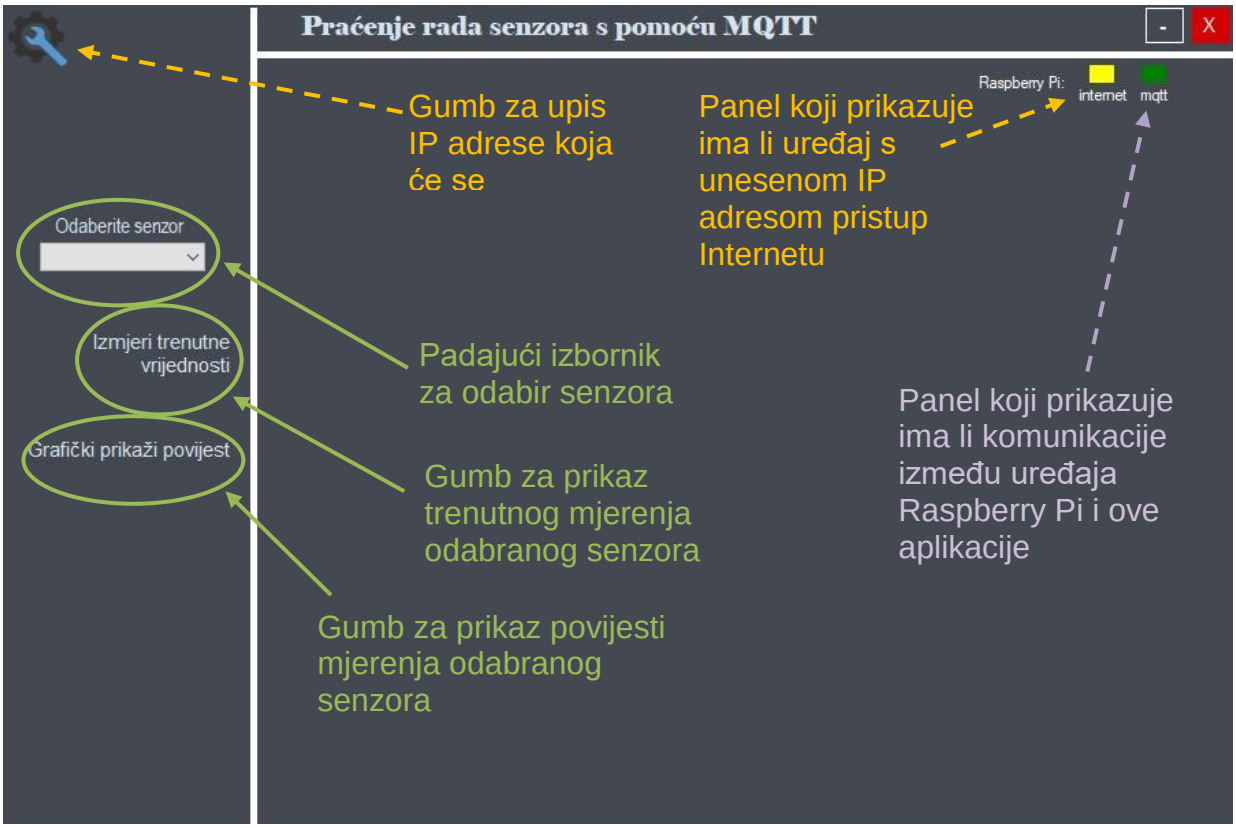
Vidljivo je da kod serverske aplikacije inicijalizacija traje nešto duže, a razlog tome je razlika u procesorskoj moći između dva računala, no to nije jedina razlika. Govoreći o postotku korištenja svih procesora, vidimo kako serverska aplikacija troši veći postotak procesora i graf iscrtava više maksimalnih ekstrema, od kojih jedan čak seže do približno 45%. S druge strane, kod klijentske aplikacije imamo vrlo malo maksimalnih ekstrema, od kojih najveći iznosi nešto više od 20%. Najveći razlog tome je količina posla kojeg serverska aplikacije kod pokretanja mora izvršiti. Naime, potrebno je kreirati pet dretvi, postaviti ih u listu, te nakon toga izvršiti, potrebno je postaviti pinove Raspberry Pi uređaja onako kako je opisano u prethodnim poglavljima. Također, važno je napomenuti da je potrebno kreirati neke objekte koji su nužni za rad i kod jedne i kod druge aplikacije, to su objekt tipa MqttClient i objekt tipa MqttVeza.

Nakon inicijalizacije, obje aplikacije rade stabilno i rade uz minimalno korištenje svih procesora koje računalo sadrži. Klikom na gumb kojim se korisniku prikaže trenutno mjerenje, koristi približno 4% procesora kod serverske aplikacije, a obrada primljenih podataka i prikaz istih kod klijentske aplikacije troši maksimalno 9% procesora. Dretva koja šalje poruku „Ping“ koristi maksimalno 4% procesora ukoliko se dogodi još neki događaj u tom periodu. Dretve koje pišu podatke u tekstualne datoteke svakih pet minuta, troše maksimalno 13% procesora. Ukoliko korisnik prikazuje povijest mjerenja grafički, serverska aplikacija ne troši ništa više od 3% procesora za čitanje datoteke (potencijalno se može povećati zbog toga jer se datoteka neprestano puni podacima) i 3% za slanje poruke. Klijentska aplikacija za obradu i prikaz tih podataka na grafu ne troši više od 10% procesora.

Što se tiče potrošnje memorije, vidljivo je da je potrošnja izrazito stabilna i kod Raspberry Pi aplikacije (serverska aplikacija) iznosi 14 MB, što je više nego duplo manje u odnosu na klijentsku aplikaciju (aplikacija na osobnom računalu) koja koristi 29 MB memorije. Razlog tome je uglavnom taj što klijentska aplikacija ima grafičko sučelje koje se mora iscrtavati i prikazivati korisniku, a to je moguće jedino na način da se čuva u memoriji.

5.5. Sučelje klijentske aplikacije

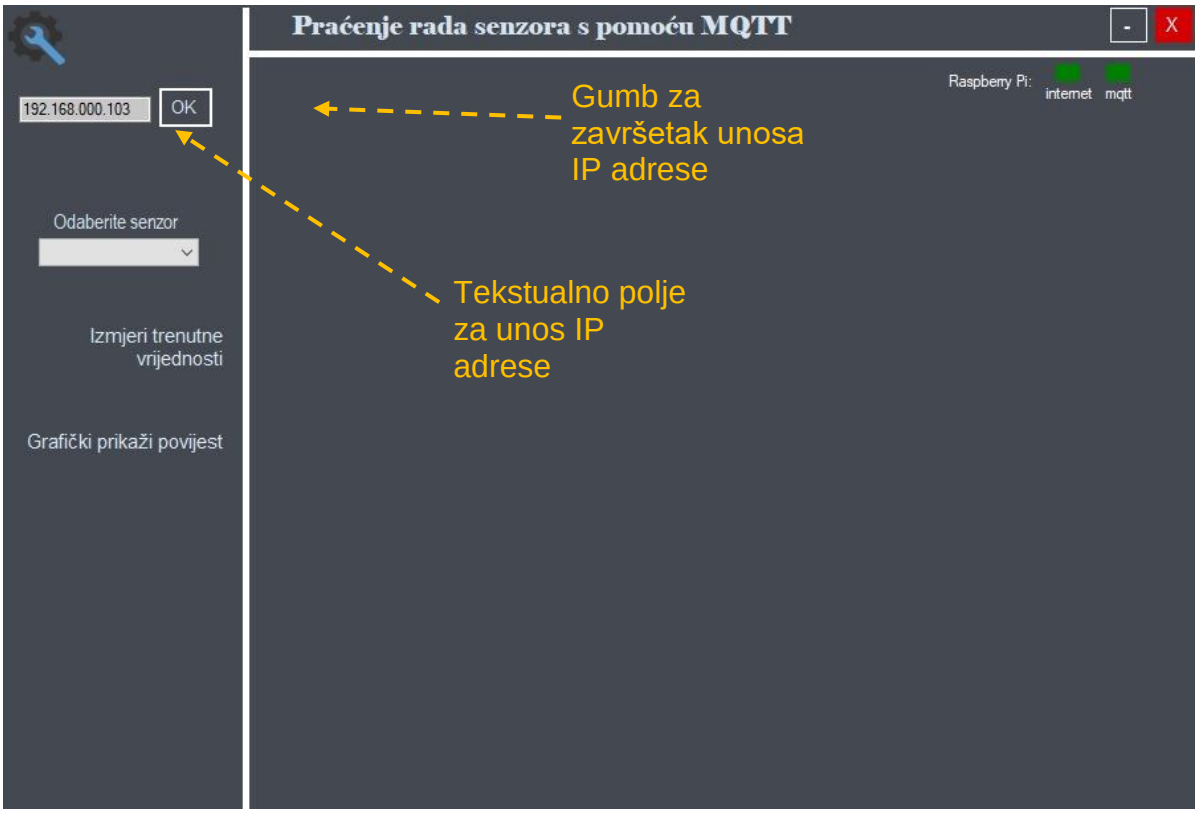
U ovome potpoglavlju ću postaviti slike sučelja aplikacije i opisati ga, te označiti na njemu neke važnije dijelove koji su prije bili spomenuti, no možda nije bilo jasno gdje se na sučelju nalaze. Postaviti ću osam slika sučelja i ukratko opisati ukoliko će biti potrebno.



Slika 16: Početni izgled sučelja aplikacije s oznakama (autorski rad)

Ova prva slika prikazuje sučelje koje se prikazuje korisniku u trenutku pokretanja aplikacije, a jedina razlika može biti eventualno boja panela koji prikazuje ima li komunikacije između uređaja Raspberry Pi i aplikacije. Ukoliko postoji komunikacija, odnosno uređaj Raspberry Pi (serverska aplikacija implementirana na uređaju) ima pristup Internetu, povezan je na isti broker kao i klijentska aplikacija. Također, ako su obje aplikacije povezane na temu „Ping“, panel će biti zelene boje kao što je vidljivo na ovoj slici iznad, ali će svakih pet sekundi postati crveni, te će opet promijeniti boju u zelenu. Ukoliko jedna od aplikacija nema pristup Internetu ili brokeru, navedeni panel će cijelo vrijeme biti crvene boje.

Oznake, odnosno strelice i tekst, obojene narančastom bojom su povezane, odnosno između komponenata na koje pokazuju postoji ovisnost. Više o tome biti će riječi u nastavku, nakon sljedeće slike.

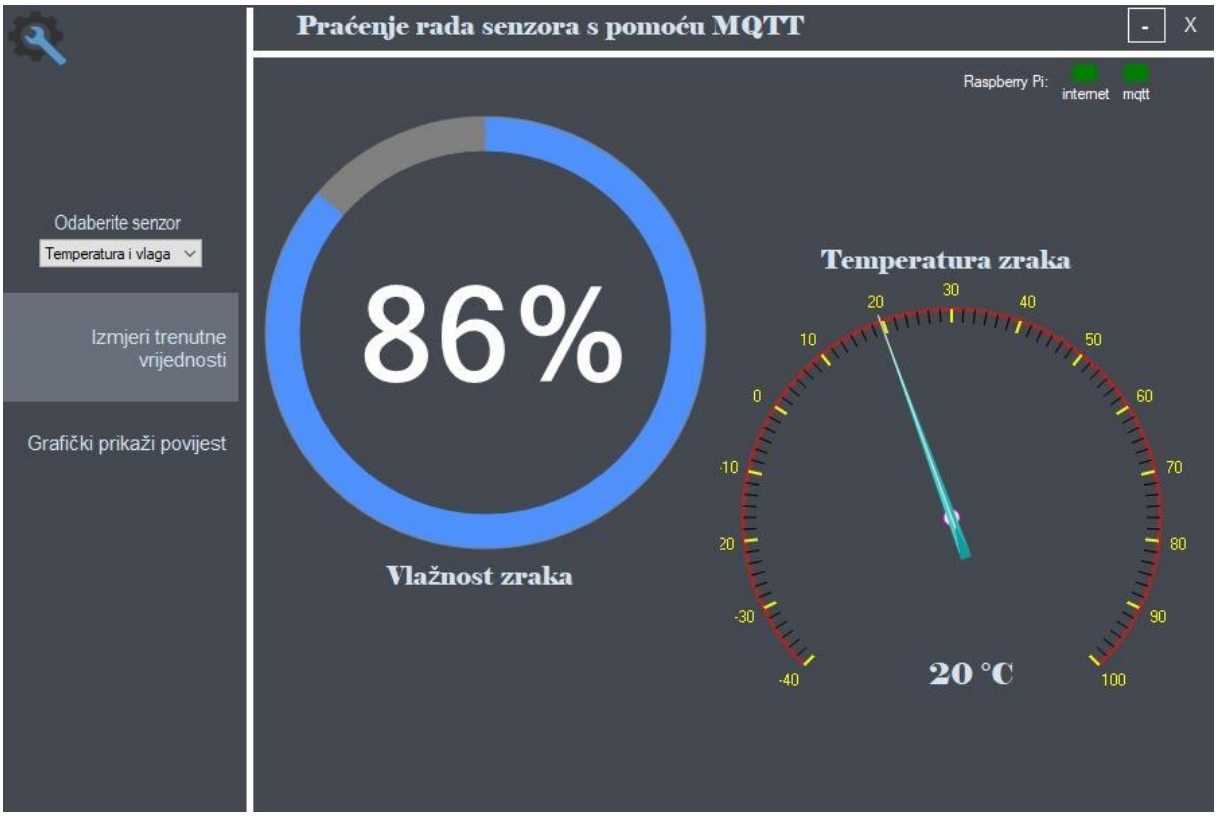


Slika 17: Unos IP adrese za provjere Internet veze (autorski rad)

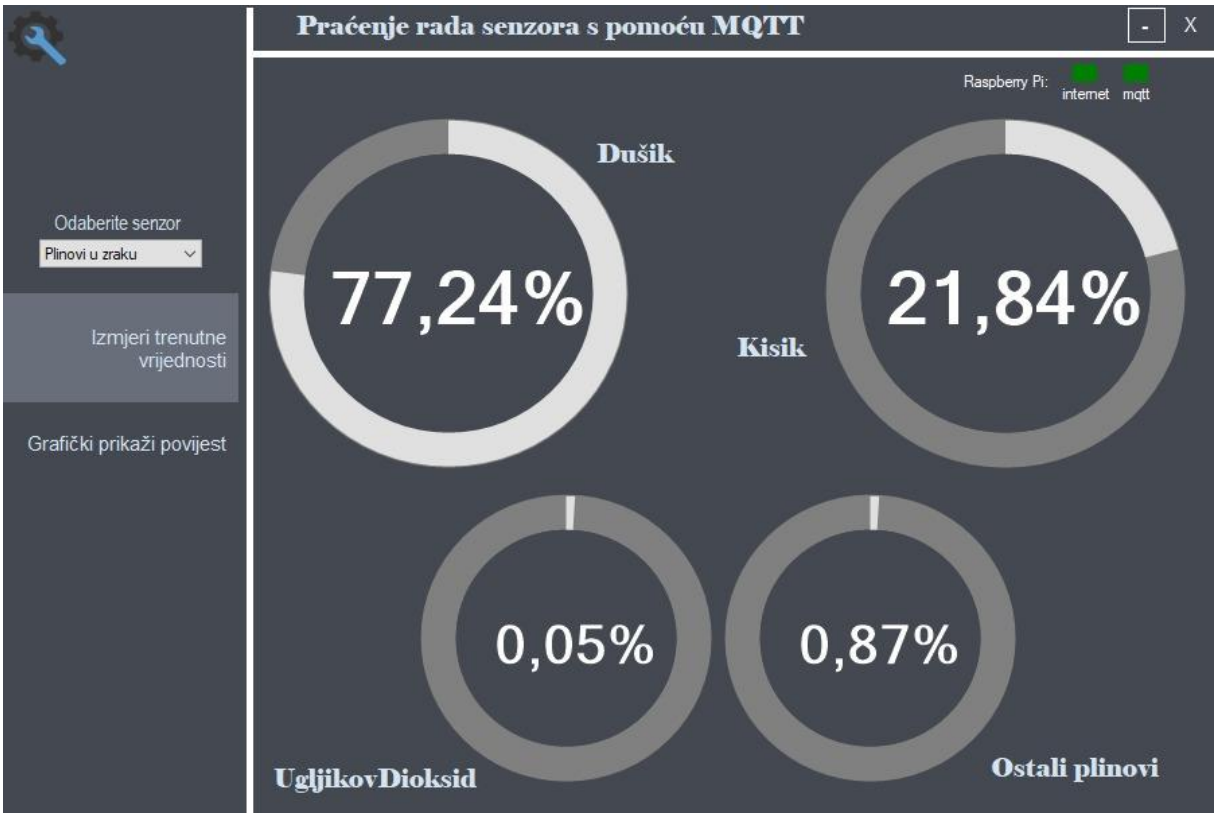
Klikom na gumb za upis IP adrese koja će se provjeravati, prikazuje se polje za tekstualni unos adrese, točnije *Masked TextBox* i gumb „OK“ kojeg korisnik klikne po završetku unosa adrese. Ukoliko uređaj s tom IP adresom postoji, odnosno ima pristup Internetu, panel koji prikazuje ima li uređaj s unesenom IP adresom pristup Internetu postaje zelene boje, u suprotnome je crvene. Ukoliko ništa nije uneseno u polje za unos IP adrese, panel je žute boje, kao na slici 16. Da bi aplikacija radila, nije potrebno unositi IP adresu uređaja, već služi isključivo za svrhe uočavanja pogrešaka, isto kao i panel desno od ovoga.

Ukoliko su oba panela crvena, potrebno je provjeriti vezu prema Internetu osobnog računala, pa potom ponovno pokrenuti aplikaciju. Ukoliko problem i dalje postoji, a osobno računalo ima pristup Internetu onda je problem kod uređaja Raspberry Pi. Ukoliko je panel koji prikazuje MQTT vezu crvene boje, a panel koji prikazuje Internet vezu uređaja s određenom IP adresom zelene boje, tada postoji mogućnost da je greška kod brokera ili u nekoj od aplikacija.

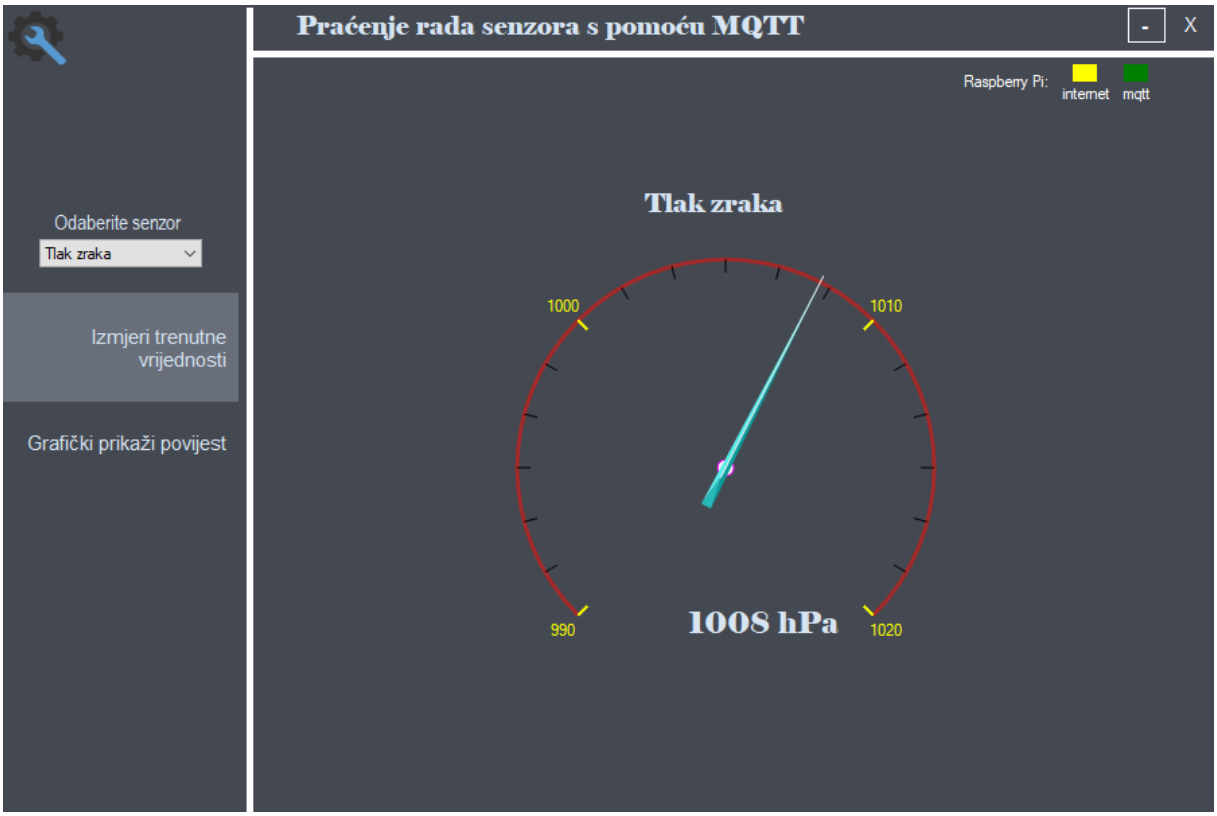
Slijedi nekoliko slika koje prikazuju rad aplikacije, odnosno izgled mjerenja trenutnih vrijednosti svakog senzora i izgled sučelja kod prikazivanja povijesti mjerenja svakog senzora.



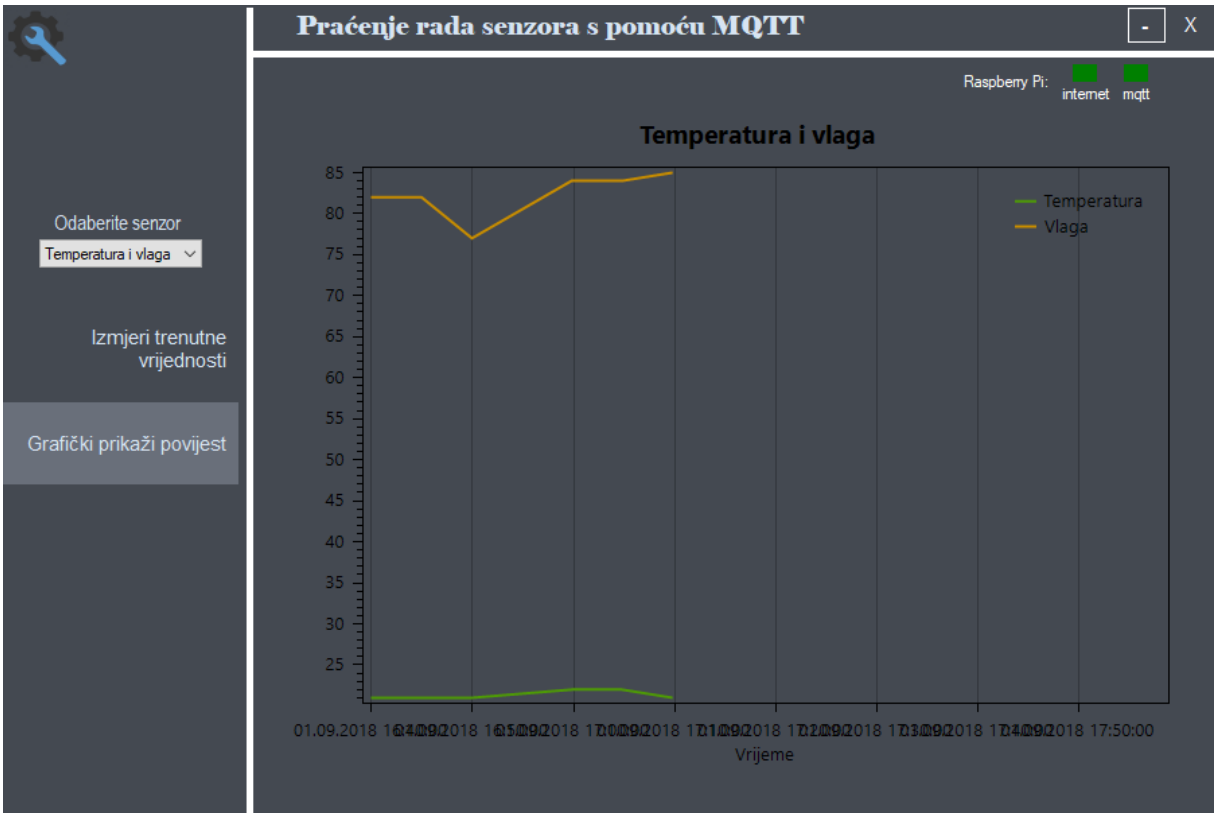
Slika 18: Izgled sučelja kod mjerenja trenutne temperature i vlage u zraku (autorski rad)



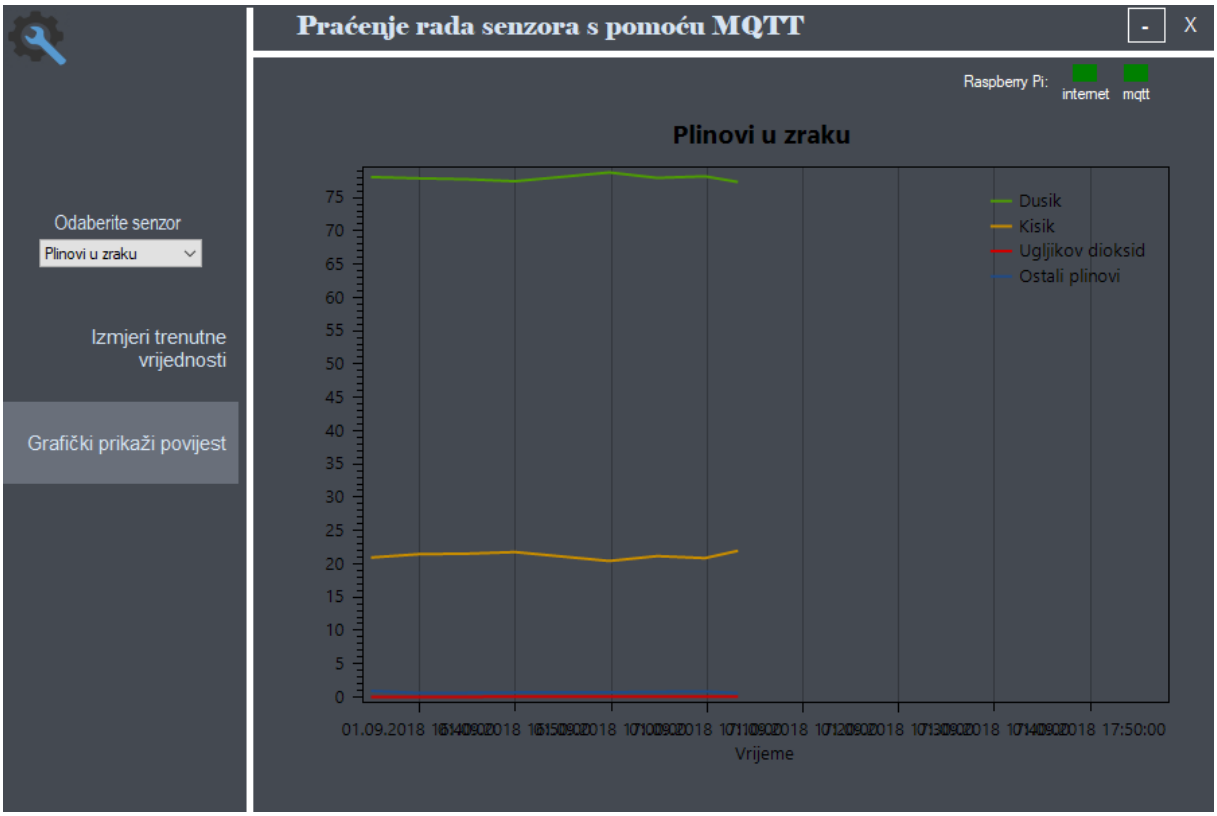
Slika 19: Izgled sučelja kod mjerenja trenutnih količina plinova u zraku (autorski rad)



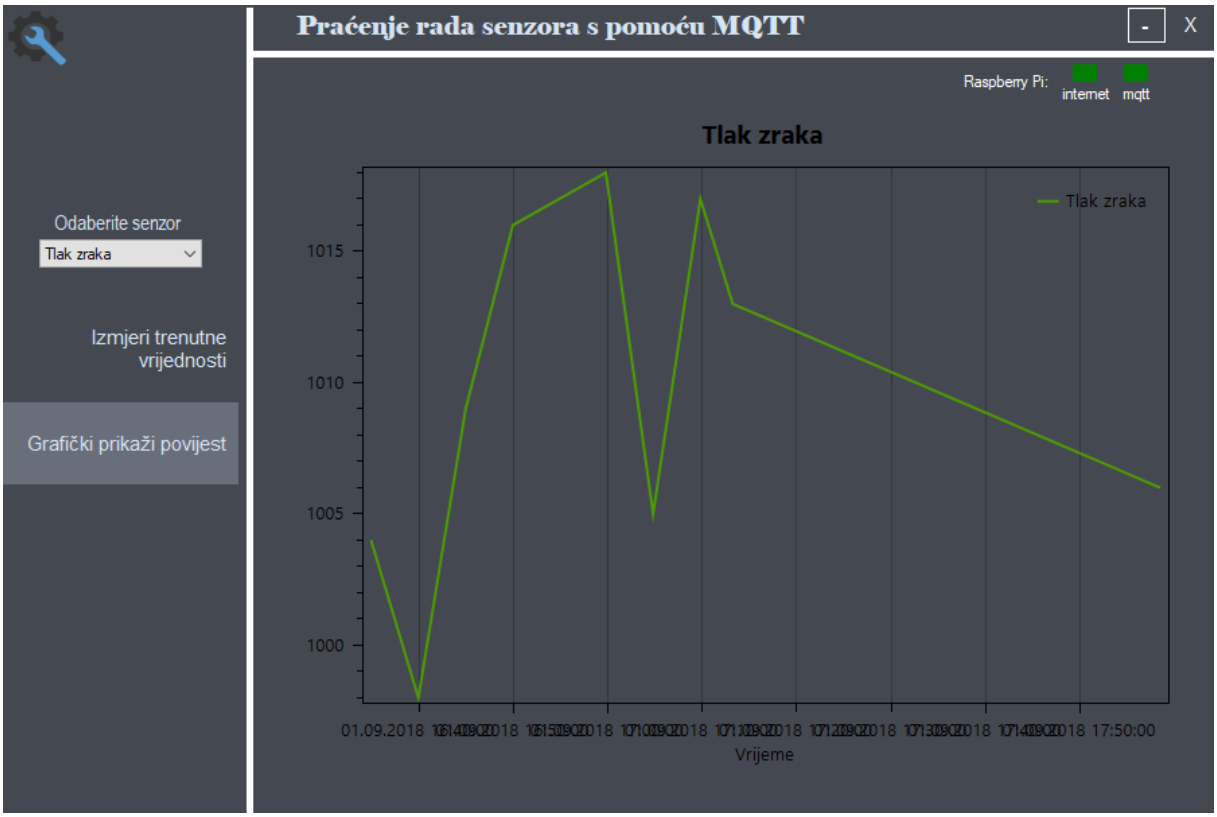
Slika 20: Izgled sučelja kod mjerenja trenutnog tlaka zraka (autorski rad)



Slika 21: Izgled sučelja kod prikazivanja povijesti mjerenja temperature i vlage u zraku (autorski rad)



Slika 22: Izgled sučelja kod prikazivanja povijesti mjerenja plinova u zraku (autorski rad)



Slika 23: Izgled sučelja kod prikazivanja povijesti mjerenja tlaka zraka (autorski rad)

Kod slike prikazivanja povijesti tlaka zraka imamo ogromne skokove, tj. velike razlike između tlaka zraka između dva mjerenja, a razlog tome je što sam u serverskoj aplikaciji ovaj senzor (i senzor plinova u zraku) izveo kao generator slučajnih brojeva koji ima raspon od 995 hPa do 1020 hPa. Također, na istom grafu je vidljivo da u samoj završnici grafa imamo jednu liniju koja cijelo vrijeme pada, a to je zbog toga jer je predzadnje mjerenje bilo npr. 1.9.2018. i iznosilo je 1013 hPa, a zadnje mjerenje sljedeći dan i iznosilo je 1007 hPa. Dakle, ne mora značiti da je u tom razdoblju tlak zraka bio u konstantnom padu, nego su jednostavno točke grafa tako posložene jer između njih nije bilo drugog mjerenja (uređaj Raspberry Pi je bio isključen). U realnom sustavu to naravno ne bi bilo tako jer su uređaji za mjerenje uključeni konstantno (24/7) i konstantno vrše mjerenja.

6. Zaključak

Po završetku ovog završnog rada, ove naročito zanimljive teme, nakon kreiranja aplikacija, snimanja razmjene Internet podataka dolazim do sljedećih zaključaka. Kako sam za izradu aplikacija koristio programski jezik C# u alatu Visual Studio 2017, mogu sa sigurnošću reći kako je C# biblioteka M2Mqtt, biblioteka za rad s protokolom MQTT jako dobro implementirana i odrađena, vrlo ju je jednostavno koristiti, te na službenim stranicama MQTT-a postoje kvalitetni primjeri koji opisuju korištenje biblioteke. Što se tiče biblioteka za senzore, uspio sam pronaći samo jednu biblioteku koja bi odgovarala senzoru koji imam, a to je biblioteka „Dht“, koja je također iznimno dobro implementirana i jednostavna za koristiti, te mi nisu trebali nikakvi primjeri da je počnem koristiti. Kako nisam uspio pronaći druge biblioteke za mjerenje fizikalnih veličina senzorima, implementirao sam dva senzora virtualno, te oni ne daju stvarne vrijednosti nego generiraju slučajne brojeve u određenom rasponu. Smatram da je to zbog toga jer se C# uz Visual Studio ne koristi na uređajima kao što je Raspberry Pi, pogotovo za ovako jednostavne stvari kao što je mjerenje fizikalnih veličina pomoću senzora, zbog toga nema baš puno biblioteka koje omogućuju mjerenje stvarnim senzorima. Naime, da bi program napravljen u C# jeziku i u alatu Visual Studio radio, na uređaju mora biti instaliran operacijski sustav koji to podržava, a to je Windows 10 IoT Core, koji je za ovakve svrhe „pretežak“. S nekim drugim operacijskim sustavima, kao što je Raspbian koji je puno „lakši“ za sam uređaj, mogu se i puno lakše mjeriti ulazni podaci u uređaj.

Slažem se sa tvrdnjom da Internet stvari još uvijek nije siguran, naveo sam da lokalni problem sigurnosti (sigurnosni propust) nikad nije lokalni problem ako je uređaj povezan na Internet. Ukoliko se Internet stvari kreira pomoću uređaja Raspberry Pi, te uređaj nije dobro sakriven, postoji velika opasnost da se netko direktno poveže na njega, pogotovo zato jer je Raspberry Pi danas „bijela kutija“, te tako stvara prijetnju ostalim uređajima. No kako bilo, Raspberry Pi se po mojem mišljenju koristi najviše u kućama, gdje je napadaču teško pristupiti. Kod npr. ulične rasvjete ne treba se koristiti Raspberry Pi ili sl., nego nešto jeftinije i jednostavnije i takav uređaj bi morao radi sigurnosti biti nepoznat široj javnosti, no to nije rješenje za sve sigurnosne probleme.

Snimanjem prometa Internet poruka između brokera i klijenta, alatom Wireshark, potvrdio sam da su poruke koje se šalju kriptirane i sigurne što se tiče neželjenog prisluškivanja. Također mogu potvrditi tezu da se podaci šalju preko protokola TCP, da su poruke složene u stvarnom i organiziranom redoslijedu, jedna iza druge i da svaka ima svoj jedinstveni redni broj. Iz snimanja prometa također sam zaključio i da su poruke koje se šalju uistinu „lagane“, odnosno veličina im je dosezala svega nešto manje od 400 bajtova. Najveća

poruka je bila „Pozdrav“, koji klijent i server šalju kod rukovanja i veličina im je bila svega 1462 bajta, što potvrđuje tezu da su poruke u praksi često i manje od 1000 bajtova (1kB).

U radu sam za crtanje dijagrama slijeda koristio besplatni alat Visual Paradigm 15.0, a za crtanje električne sheme koristio sam besplatni *online* alat CircuitLab koji je besplatan, vrlo jednostavan za korištenje, a završni izvoz projekta kao slike je visoke kvalitete.

Sav kod u obje aplikacije pisao sam sam i to na način da sam se trudio pisati kod koji je što čitljiviji, što čišći i što optimiziraniji za procesor i memoriju računala. Glavne klase aplikacija (a to su kod klijentske aplikacije klasa forme (uiGlavnaForma), a kod serverske aplikacije klasa „MainPage“) trudio sam održavati sa što manje koda kako bi čitaocu bilo što jasnije što koja linija koda radi, pa zbog toga u aplikacijama imam puno klasa, neke s više, a neke s manje koda.

Da rezimiram, protokol MQTT dolazi od engleske skraćenice *Message Queuing Telemetry Transport*, protokol je baziran na klijent-server, objavi/pretplati principu. Server preko kojeg uređaji komuniciraju, u MQTT žargonu, zove se broker, a postoje javni, već kreirani MQTT brokeri koje javnost može koristiti (jedan od njih je IoT Eclipse koji ja koristim u radu). Velika prednost protokola je to što daje korisniku mogućnost odabira kvalitete pružanja usluga (engl. *Quality of Service*, QoS). Spomenuta mogućnost je vrlo važna, iako se možda ne čini tako. Korisniku koji zna što radi vrlo je važno da se razmjena poruka izvodi onako kako on želi, da se čim prije „riješi“ poslane poruke (QoS 0). Da se poruke šalju tako dugo dok ne stigne potvrda od brokera da je barem jedna poruka stigla (QoS 1, mogu se javiti duplikati) ili pak da se poruka pošalje točno jedanput, uz rizik smanjenja performansi protokola. Vrlo važno je napomenuti da su poruke ovog protokola čisti tekst i ne može biti ništa drugo osim teksta, te da su poruke vrlo male što se tiče veličine.

Smatram da će protokol zbog svega navedenog sve više i više biti u upotrebi u skoroj budućnosti. Čim se riješe problemi sigurnosti, protokol će se jako brzo proširiti i vjerojatno staviti u upotrebu kod gotovo svih javnih i privatnih uređaja kojima se često trebaju podešavati vrijednosti. Pod pojmom „podešavati vrijednosti“ može spadati bilo što, primjerice upravljanje gradskom rasvjetom, upravljanje grijanjem i rasvjetom u kući, upravljanje nadzornim kamerama, baždarenje javnih satova (ura) i drugih uređaja za mjerenje. Vjerujem da će protokolu uloga biti široko rasprostranjena i da će Internet stvari biti uvelike korišten pojam i princip kod uštede vremena (već postoje pametni hladnjaci koji javljaju vlasniku čega u hladnjaku nedostaje i tako štedi vlasniku vrijeme).

Popis literature

Piljac, I. (2010). *Senzori fizikalnih veličina i elektroanalitičke metode*. Zagreb: MEDIAPRINT – TISKARA HRASTIĆ d.o.o.

Vermesan, O, i Friess, P. (2013). *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems*. Aalborg, Denmark: River Publishers

International Organization for Standardization / International Electrotechnical Commission [ISO/IEC] (06.2016)

Preuzeto 14.07.2018. s https://www.iso.org/standard/69466.html

MQTT Frequently Asked Questions [MQTT FAQ] (bez dat.)

Preuzeto 27.07.2018. s http://mqtt.org/faq

dc-square GmbH (bez dat.), *MQTT 101 – How to Get Started with the lightweight IoT Protocol*

Preuzeto 28.07.2018. s https://www.hivemq.com/blog/how-to-get-started-with-mqtt

dc-square GmbH (bez dat.), *MQTT Security Fundamentals*

Preuzeto 28.07.2018. s https://www.hivemq.com/mqtt-security-fundamentals/

dc-square GmbH (bez dat.), *MQTT Essentials Part 6: Quality of Service*

Preuzeto 30.07.2018. s https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels

Rose, K., Eldridge, S., Chapin L. The Internet Society [ISOC] (listopad 2015), *The Internet of things: an overwiev*

Preuzeto 30.07.2018. s https://www.Internetsociety.org/wp-content/uploads/2017/08/ISOC-IoT-Overview-20151221-en.pdf

Saxena, P. (15.6.2016), *The advantages and disadvantages of Internet Of Things*

Preuzeto 30.07.2018. s https://e27.co/advantages-disadvantages-Internet-things-20160615/

The Raspberry Pi Foundation (bez dat.), *FAQS*

Preuzeto 31.07.2018. s https://www.raspberrypi.org/help/faqs/#introWhatIs

Patierno, P. (bez dat.), *MQTT Client Library Encyclopedia – M2Mqtt*

Preuzeto 02.08.2018. s https://www.hivemq.com/blog/mqtt-client-library-encyclopedia-m2mqtt

Tehnička dokumentacija uređaja Raspberry Pi 2 Model B, bez dat.

CAcert wiki (2017.), *FAQ/AboutUs*

Preuzeto 05.08.2018. s http://wiki.cacert.org/FAQ/AboutUs

SSL2BUY global ssl provide (bez dat.), *What is the Difference between Client and Server Certificates?*

Preuzeto 05.08.2018. s https://www.ssl2buy.com/wiki/what-is-the-difference-between-client-and-server-certificates

Raspberry Pi Geek [Slika] (bez dat.)

Preuzeto 09.09.2018. sa http://www.raspberry-pi-geek.com/howto/GPIO-Pinout-Rasp-Pi-1-Model-B-Rasp-Pi-2-Model-B

Popis slika

Popis tablica