3 MQTT

3.1 Pub/sub model

„*Publisher-subscriber*“ model komunikacije, takođe poznat kao *pub/sub* model, je arhitekturni obrazac koji se koristi za razmenu poruka između različitih komponenata u sistemu. Ovaj model se često koristi u distribuiranim sistemima i aplikacijama gde je potrebno asinhrono slanje i primanje poruka. U ovom modelu, postoje tri osnovne komponente: *publisher* (izdavač, ili oglašivač), *subscriber* (pretplatnik) i *broker* (posrednik).

Publisher (oglašivač) je entitet koji generiše i šalje poruke. On nije svestan ko će tačno primiti poruke koje šalje. Umesto toga, oglašivač jednostavno objavljuje (publikuje) poruke na određene teme ili kanale. Na primer, oglašivač može biti senzor koji šalje podatke o izmerenoj temperaturi.

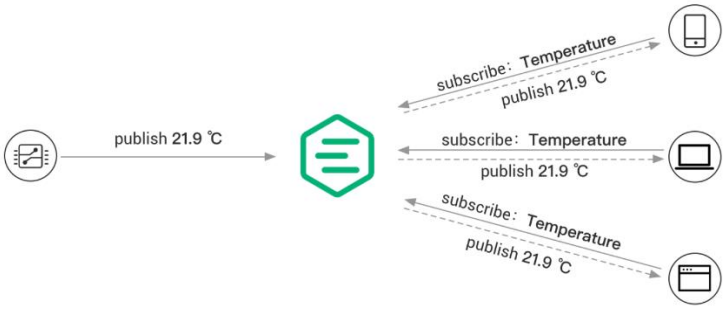
Subscriber (pretplatnik) je entitet koji prima poruke koje su objavljene od strane oglašivača. Pretplatnik se pretplaćuje (*subscribe*) na određene teme ili kanale kako bi primio poruke koje su relevantne za njega. Na primer, aplikacija koja prikazuje trenutnu temperaturu može biti pretplatnik na temu koja sadrži informacije o temperaturi.

Broker (posrednik) je komponenta koja olakšava komunikaciju između oglašivača i pretplatnika. Broker prihvata poruke od oglašivača i distribuira ih pretplatnicima na osnovu njihovih pretplata. Broker može imati različite funkcionalnosti, kao što su filtriranje poruka, usmeravanje poruka, ili osiguravanje pouzdanosti isporuke.

Pub/sub model se može razumeti i kako vrsta klijent-server modela sa podeljenim ulogama klijenata. Broker je server, poznat svim klijentima; neki klijenti igraju ulogu oglašivača, a neki ulogu pretplatnika. Na osnovu prethodnog je jasno da *pub/sub* model omogućava fleksibilnu i skalabilnu komunikaciju između različitih komponenata sistema, omogućavajući im da rade nezavisno jedne od drugih. U ovom modelu, oglašivači ne moraju da znaju identitete pretplatnika; oni, prosto šalju svoje podatke (na određene teme) brokeru, a ovaj ih distribuira svim trenutno zainteresovanim pretplatnicima. Takođe, pretplatnici ne moraju da znaju identitet oglašivača jer ono što povezuje oglašivače i pretplatnike jesu teme i broker. Ukoliko se komunikacija obavlja preko Interneta, fizičke lokacije oglašivača, pretplatnika i brokera nisu bitne. U tom slučaju, da bi se uređaj pridružio *pub/sub* sistemu potrebno je da poznaje IP adresu brokera i da ima pravo pristupa brokeru.

3.2 MQTT protokol

MQTT (*Message Queue Telemetry Transport*) je upravo protokol za *pub/sub* komunikaciju. U MQTT terminologiji, objedinjeni pojam za oglašivače i pretplatnike je MQTT klijenti, dok se za broker nekada koristi pojam MQTT server.



Sl. 1 MQTT sistem

Princip MQTT protokola je ilustrovan na Sl. 1. Entitet u sredini je broker, koji je okružen MQTT klijentima. Entitet levo na slici je senzorski uređaj, npr. tipa pametni termometar, koji u ovoj postavci igra ulogu oglašivača. Termometar periodično meri temperaturu okoline i publikuje rezultate merenja tako što brokeru šalje poruku o trenutnoj temperaturi. Ova poruka ne sadrži samo brojčanu vrednost trenutne temperature, već i indikaciju o „temi“, odnosno o tome šta je sadržaj poruke, npr. „*Temperatura u kabinetu br. 601 na Elektronskom fakultetu u Nišu*“. Na desnoj strani slike su pretplatnici, odnosno MQTT klijenti koji igraju ulogu pretplatnika. To mogu biti uređaji tipa mobilni telefon, tablet, PC računar i sl. koji su zainteresovani za informaciju o temperaturi. Da bi postao pretplatnik, MQTT klijent mora da se prijavi brokeru navodeći temu na koju se pretplaćuje, npr. „*Pretplaćujem se na temu Temperatura u kabinetu br. 601 na Elektronskom fakultetu u Nišu*“. Uvek kada primi poruku oglašivača, broker prosleđuje njen sadržaj svim MQTT klijentima koji su se pretplatili na konkretnu temu. MQTT klijent može da se odjavi sa teme na koju se prethodno prijavo. Takođe, MQTT klijent može da se prijavi za prijem poruka o više različitih tema. Uz to, MQTT klijent može istovremeno igrati ulogu oglašivača i pretplatnika.

3.2.1 Teme

MQTT distribuira poruke između oglašivača i pretplatnika na osnovu tema. U MQTT sistemu, teme su tipično organizovane na hijerarhijski način, u više nivoa. Tema se zapisuje kao string u kome su tematski nivoi razdvojeni znakom „/“ (slično kao URL, tj. veb adresa). Na primer, pretpostavimo da smo u stanu instalirali nekoliko senzora za merenje temperature, vlažnosti i kvaliteta vazduha i to po jedan od svake vrste u kuhinji, spavaćoj i dnevnoj sobi. Spisak svih tema u ovom sistemu bi mogao da glasi:

myhome/bedroom/temperature
myhome/bedroom/humidity
myhome/bedroom/airquality
myhome/livingroom/temperature
myhome/livingroom/humidity
myhome/livingroom/airquality
myhome/kitchen/temperature
myhome/kitchen/humidity
myhome/kitchen/airquality

Dakle, u poruci koju senzor temperature u spavaćoj sobi šalje brokeru kao tema biće naveden string „*myhome/bedroom/temperature*“. MQTT klijent koji želi da prati temperaturu u spavaćoj sobi, prijavljuje se brokeru na temu „*myhome/bedroom/temperature*“.

MQTT dopušta upotrebu dva specijalna zamenska znaka (*wildcards*) u stringu teme: # i +, koji se mogu koristiti samo u stringovima za pretplatu. Znak „#“ se može pisati samo na kraju stringa i zamenjuje bilo koji pojam u poslednjem nivou. Na primer, pretpostavimo da MQTT klijent želi da prati sva tri parametra u spavaćoj sobi. Umesto da se pojedinačno pretplaćuje na svaku temu, klijent može, u jednom koraku, da se pretplatiti na sve teme koje su vezane za spavaću sobu i to slanjem brokeru stringa za pretplatu oblika „*myhome/bedroom/#*“.

Znak „+“ zamenjuje jedan unutrašnji nivo u tematskoj hijerarhiji. Na primer, pretplatom na „*myhome/+/temperature*“, MQTT klijent se zapravo pretplaćuje na sve teme koje se tiču temperature, bez obzira iz koje sobe potiču. Efekat je isti kao da se klijent pojedinačno pretplatio na teme:

myhome/bedroom/temperature
myhome/livingroom/temperature
myhome/kitchen/temperature

Znak „+“ nije dopušteno pisati na kraju stringa (zamenski znak za kraj stringa je „#“)

Više pretplatnika se može pretplatiti na istu temu, i broker će proslediti sve poruke na toj temi ovim pretplatnicima. Više oglašivača takođe može slati poruke na istu temu, i broker će prosleđivati ove poruke pretplaćenim klijentima u redosledu kako su primljene.

3.2.2 QoS

Kvalitet usluge (*QoS - Quality of Service*) se odnosi na nivo pouzdanosti koji pruža MQTT protokol prilikom isporuke poruka između MQTT klijenata i brokera. MQTT podržava tri QoS nivoa: QoS 0, QoS 1 i QoS 2.

QoS 0 (Najviše jednom). Ovo je osnovni (najniži) nivo kvaliteta servisa kojim se poruka isporučuje principom „pošalji i zaboravi“. Naime, poruke se razmenjuju između klijenta i brokera bez bilo kakvog potvrđivanja ili garancije isporuke. Poruka se isporučuje najviše jednom, i nema načina za ponovni pokušaj isporuke.

QoS 1 (Najmanje jednom). Ovaj QoS nivo garantuje da će poruka biti isporučena primaocu najmanje jedanput. Pošiljalac čuva poslatu poruku sve dok ne dobije potvrdu prijema od primaoca. Ukoliko potvrda ne stigne u određenom vremenskom periodu, pošiljalac ponovo šalje istu poruku. Može se desiti da primalac primi duplikate iste poruke ukoliko postoji problem u dostavi potvrde – otuda „najmanje jedanput“.

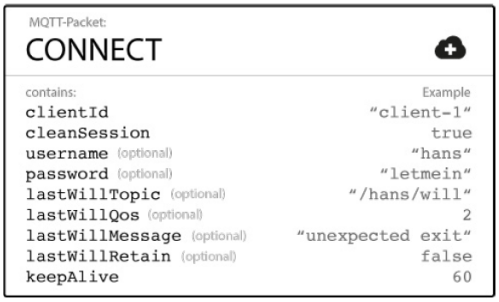
QoS 2 (Tačno jednom). Ovo je najviši nivo pouzdanosti u isporuci poruka u MQTT protokolu, koji garantuje da će svaka poruka biti tačno jednom isporučena primaocu. U odnosu na QoS 1, QoS 2 garantuje da se u prenosu neće pojaviti duplikati poruka. Nedostatak ovog nivoa kvaliteta servisa je potreba da se između pošiljaoca i primaoca razmene čak četiri poruke – po dve u svakom smeru. (Kod QoS 1 prenosi se samo jedna poruka, kod QoS 2 se prenose dve, a kod QoS 2 čak četiri poruke.)

QoS nivo specificira MQTT klijent. Indikacija o QoS nivou je prisutna u svakoj poruci koju oglašivač šalje brokeru. Takođe, kada se pretplaćuje na određenu temu, pretplatnik navodi i QoS nivo kojim bi broker trebalo da mu isporučuje poruke.

3.2.3 CONNECT

Uspostavljanje konekcije između MQTT klijenta i brokera uvek inicira klijent. Klijent prvo otvara TCP konekciju sa brokerom, a zatim, kroz otvorenu konekciju šalje brokeru poruku CONNECT. Sadržaj ove poruke je prikazan na Sl. 2. Poruka CONNECT nosi nekoliko parametara, od kojih su obavezni: *ClientID*, *Clean Session* i *Keep Alive*. Broker odgovara klijentu porukom potvrde CONNACK. Nakon toga, klijent može

da publikuje svoje podatke, da se pretplaćuje na različite teme i prima poruke publikovane na tim temama. MQTT može da okonča svoju vezu sa brokerom slanjem poruke DISCONNECT.



Sl. 2 Format poruke CONNECT

ClientID

ClientID je identifikator, tj. ime u obliku stringa karaktera, pod kojim se MQTT klijent prijavljuje brokeru. Svi klijenti konektovani sa istim brokerom moraju imati jedinstvene *ClientID*. MQTT klijent se može prijati i bez navođenja *ClientID*. U tom slučaju, broker dodeljuje klijentu jedinstveni *ClientID*.

Clean Session

Clean Session je fleg u poruci CONNECT koji predstavlja indikaciju o tome da li klijent sa brokerom želi da uspostavi *clean* ili *persistent* sesiju. Kod MQTT-a, sesija se odnosi na logičku vezu između MQTT klijenta i MQTT brokera (za razliku od TCP konekcije koja se može razumeti kao „fizička“ veza za prenos podataka). MQTT sesija može biti *čista* (*clean*) ili *trajna*, tj. *perzistentna* (*persistent*). Razlika između ova dva tipa sesije je u tome kako prekid TCP konekcije utiče na vezu između klijenta i brokera. U slučaju čiste sesije, nakon prekida TCP konekcije broker zaboravlja na klijenta i briše sve njegove pretplate. Novo konektovanje istog klijenta pokreće novu sesiju. U slučaju perzistentne sesije, broker čuva podatke o klijentu i nakon prekida TCP konekcije. Smatra se da je ovaj prekid privremen, nastao usled pojave nestabilnosti u mreži i da će se klijent, koji je trenutno *offline*, ponovo javiti. Za vreme dok je konekcija u prekidu, broker čuva sve podatke koji su se u međuvremenu publikovali na temama za koje je klijent zainteresovan i isporučiće ove podatke klijentu odmah nakon što on obnovi svoju perzistentnu sesiju. Prilikom obnavljanja perzistentne sesije, klijent ne mora ponovo da registruje svoje pretplate.

Keep Alive

Keep Alive je mehanizam za održavanje aktivne veze između MQTT klijenta i brokera. Ovaj mehanizam podržan je istoimenim parametrom u poruci CONNECT koji definiše maksimalno vreme neaktivnosti veze (u sekundama). Problem koji se rešava *keep alive* mehanizmom je u vezi sa dugim periodima neaktivnosti komunikacije klijent-broker. Naime, ako klijent i broker nisu razmenjivali poruke neki duži vremenski period, postavlja se pitanje da li je to zbog toga što nije bilo potrebe za komunikacijom ili zato što je, u međuvremenu, došlo do prekida TCP konekcije. Da bi se veza održavala „živom“, klijent ima obavezu da po isteku vremena „*Keep Alive*“ nakon poslednje komunikacije sa brokerom, pošalje brokeru poruku PINGREQ. Nakon što primi poruku PINGREQ, broker šalje poruku PINGRESP kao odgovor nazad klijentu. Izostanak PINGREQ u očekivanom vremenskom *keep-alive* intervalu predstavlja indikaciju brokeru da je veza sa klijentom u prekidu. Slično, izostanak PINGRESP odgovora klijent tumači kao prekid konekcije.

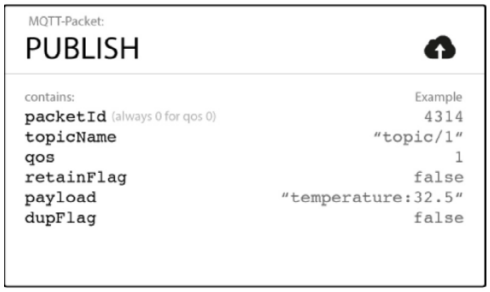
Last Will

„*Last Will*“ je još jedan MQTT mehanizam koji se aktivira u slučaju neočekivanog prekida veze između klijenta i brokera i njegova svrha je da ostalim zainteresovanim klijentima u sistemu prenese „poslednju želju“, ili „testament“ neočekivano diskonektovanog klijenta. Prilikom uspostavljanja veze sa brokerom,

klijent ima mogućnost da u poruci CONNECT navede svoju „poslednju želju“ (polje *lastWillMessage*) zajedno sa temom poslednje želje (polje *lastWillTopic*). „Poslednja želja“ je tipično kratka tekstualna poruka, poput „Veza sa senzorom temperature je privremeno u prekidu“, ili neki kôd istog značenja. U slučaju neočekivanog prekida veze, broker publikuje poslednju želju svim klijentima koji su pretplaćeni na temu poslednje želje.

3.2.4 PUBLISH

MQTT klijent publikuje svoje podatke slanjem brokeru poruke *PUBLISH* (Sl. 3). Ova poruka sadrži naziv teme (*topicName*) i podatke koje klijent publikuje na tu temu (polje *payload*). Osim toga, poruka PUBLISH sadrži još nekoliko polja. Polje *QoS* sadrži QoS nivo. Polja *packetId* i *dupFlag* su relevantna samo ukoliko se koristi QoS nivoa 1 ili 2. Polje *packetId* nosi jedinstveni identifikator paketa i koristi se za otkrivanje duplikata. Polje *dupFlag* sadrži indikaciju da je klijent ponovo poslao isti paket.



Sl. 3 Format poruke PUBLISH

Retained messages

Retained messages ili zadržane poruke su mehanizam MQTT-a koji omogućava brokeru da sačuva poslednju poruku objavljenu na određenoj temi i dostavi je svim novim pretplatnicima odmah po njihovoj pretplati. Ovaj mehanizam osigurava da pretplatnici prime najnovije informacije objavljene na temi, čak i ako nisu bili povezani u trenutku kada je poruka prvobitno objavljena. Na primer, u aplikacijama za kućnu automatizaciju, zadržana poruka može se koristiti za čuvanje trenutnog stanja nekog uređaja (npr. "svetlo je upaljeno") tako da svaki novi pretplatnik prima to stanje odmah po povezivanju. Da li će publikovana poruka biti zadržana reguliše MQTT klijent putem bita *retainFlag* u poruci PUBLISH.

3.2.5 SUBSCRIBE

MQTT klijent se pretplaćuje na teme slanjem brokeru poruke *SUBSCRIBE*. Ova poruka ima jednostavnu strukturu i sadrži nazive jedne ili više tema na koje se klijent pretplaćuje. Pri tom, teme mogu da sadrže zametske znake (kao što je prethodno objašnjeno). U poruci SUBSCRIBE, za svaku temu navodi se i QoS nivo kojim broker treba da isporučuje klijentu odgovarajuće podatke. Broker potvrđuje prijem poruke SUBSCRIBE slanjem klijentu poruku potvrde SUBACK. U poruci SUBACK, eksplicitno je potvrđene pretplata na svaku temu iz poruke SUBSCRIBE; odnosno indikacija da je pretplata, iz bilo kog razloga, obijena. MQTT klijent može da se odjavi sa teme na koju se prethodno prijavio slanjem brokeru poruke UNSUBSCRIBE.

3.3 MQTT broker

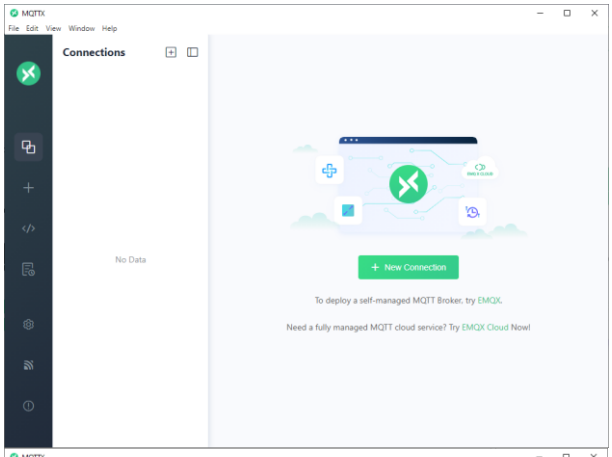
MQTT sistem se sastoji iz brokera i MQTT klijenta. Broker je aplikacija koja je pokrenuta na serverskom računaru, koji je, preko mreže, dostupan klijentima za konektovanje. Broker može biti privatni ili javni. Privatni MQTT broker je pod punom kontrolom projektanta koji realizuje MQTT sistem. Projektant instalira broker na svom serveru i u mogućnosti je da administrira sve parametre rada brokera, uključujući prava

pristupa klijenata, sigurnosne mere i sl. S druge strane, javni broker je pokrenut na klaudu, globalno je dostupan preko Interneta i pruža svoje brokerske usluge registrovanim korisnicima. Javni broker nije u vlasništvu projektanta MQTT sistema, ali projektant može da iznajmi pravo korišćenje brokera. Pojedini javni brokeri omogućavaju besplatno korišćenje svojih usluga, tipično, sve dok se ne prekorači raspoloživi broj razmenjenih MQTT poruka.

Za ilustraciju koncipiranja MQTT sistema, koristićemo javni i besplatni MQTT broker EMQX. Veb stranica ovog brokera je dostupna na linku https://www.emqx.com/en/mqtt/public-mqtt5-broker. Na toj stranici se mogu naći podaci za pristup brokeru. Broker je dostupan za TCP konektovanje preko URL-a broker.emqx.io na portu **1883**. Prilikom konektovanja, MQTT klijent treba da dostavi brokeru korisničko ime: **emqx** i lozinku: **public**.

3.4 MQTT Client Toolbox

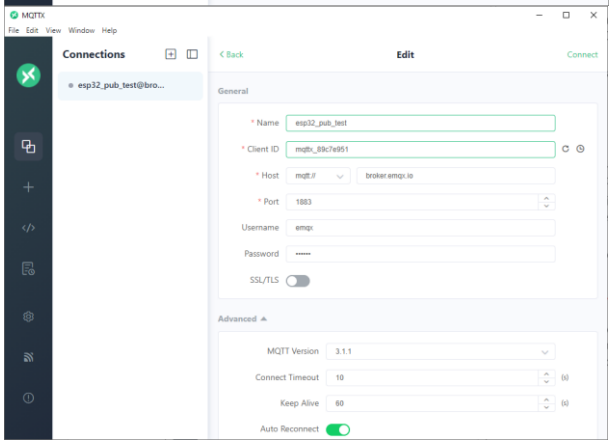
Koristan alat za razvoj i testiranje MQTT sistema je „MQTT Client Toolbox“. To su Windows programi koji realizuje funkciju MQTT klijenta. Postoji više ovakvih programa, a mi ćemo koristiti program MQTTX, koji je dostupan za besplatno preuzimanje na linku https://mqttx.app/. MQTTX omogućava konektovanje na željeni MQTT broker, ručno slanje, tj. publikovanje poruka, prijavljivanje na teme i prijem i pregled primljenih poruka. Sledi kratak tutorijal za korišćenje programa MQTTX.



Ovako izgleda prozora programa MQTTX nakon prvog pokretanja (←).

U traci s leve strane dostupan je grafički meni u kome je trenutno izabrana prva opcija ispod zelenog logoa. Ova opcija prikazuje sve kreirane konekcije (za sada nema ni jedne).

Radi uspostavljanje konekcije sa brokerom potrebnoj je izabrati opciju „+“ u grafičkom meniju.



U dijalogu koji se otvara, podešavaju se parametri konekcije, odnosno parametri koje će MQTT klijent (program MQTTX) poslati brokeru u poruci CONNECT.

U dijalogu levo uneti su parametri za konektovanje sa brokerom EMQX.

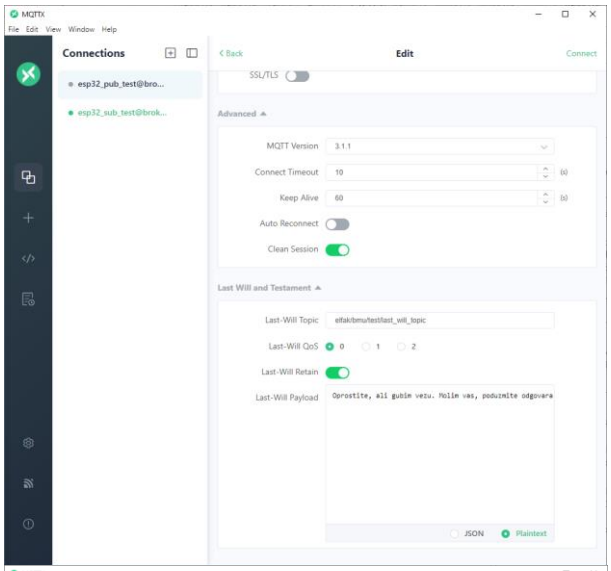
Name: „esp32_pub_test“ - lokalno ime konekcije. Nije relevantno za broker, već je to samo naziv konekcije u okviru MQTTX programa.

ClientID: identifikator klijenta. Mora biti jedinstven među svim klijentima koji se konektuju na dati broker. Tekst koji vidimo u ovom polju predložio je sam program MQTTX. „Heksadecimalni“ deo ovog zapisa je slučajno generisan, što bi trebalo da garantuje jedinstvenost ovog identifikatora.

Host i *Port*: unosi se URL i port brokera.

Username i *Password*: korisničko ime i lozinka (emqx i public).

SSL/TLS: ostaje isključeno (uključivanjem se bira opcija šifrovane konekcije).



Nastavak podešavanja pratimo na sledećoj slici (←).

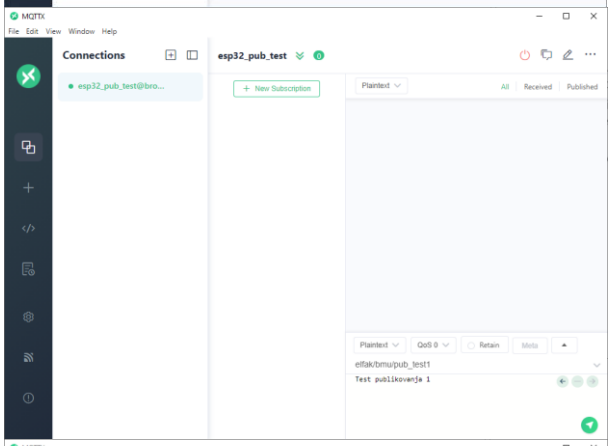
MQTT Version: Postoji nekoliko verzija MQTT protokola. U ovom polju je izabrano 3.1.1, zato što je upravo ta verzija predstavljena u pripremi ove vežbe.

Za ostale parametre u delu *Advanced* ostaviti podrazumevane vrednosti. Među ovim parametrima uočavamo *Keep Alive* i *Clean Session*.

Poslednja sekcija za podešavanje odnosi se na „poslednju želju“ MQTT klijenta.

U polju *Last-Will Topic* navedena je tema poslednje želje (elfak/bmu/test/last_will_topic). Ukoliko polje *Last Will Topic* ostane prazno, mehanizam poslednje želje neće biti aktiviran.

Sadržaj poslednje želje napisan je u polju *Last Will Payload*.



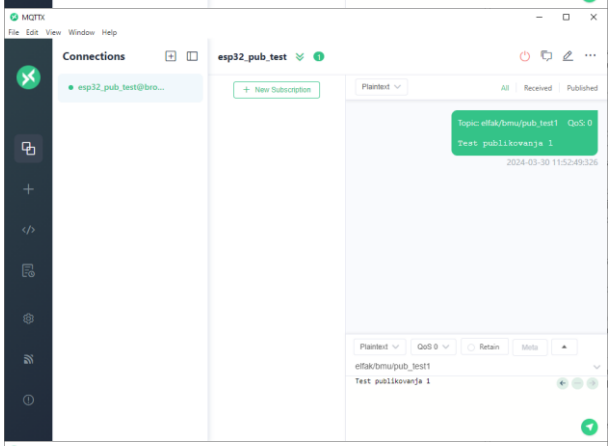
Takođe, za poslednju želju, izabran je QoS nivoa 0 i uključena opcija zadržavanja (broker prosleđuje poslednju želju i klijentima koji će se prijaviti nakon raskida konekcije „esp32_pub_test“).

Nakon što su uneti svi potrebni podaci, konekcija se pokreće klikom na link *Connect* (gore desno).

Ako je konekcija uspešno ostvarena, pokazaće se dijalog (←).

U panelu *Connections* vidimo ime upravo kreirane konekcije.

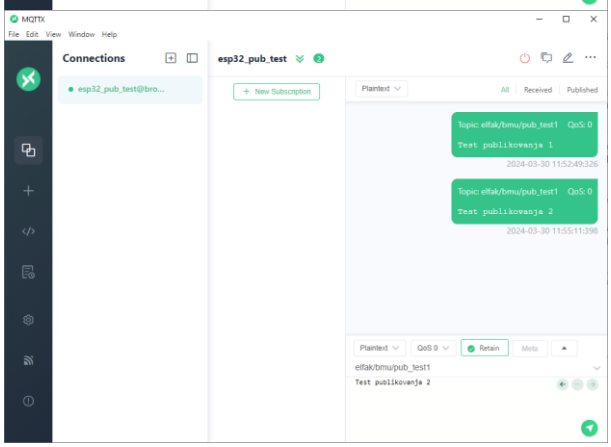
Desni panel je predviđen za publikovanje poruka na željene teme. Tema i sadržaj poruke se upisuju u tekst polja na dnu ovog panela.



Kao što vidimo, sve je pripremljeno za slanje poruke „Test publikovanja 1“ na temu „elfak/bmu/pub_test1“. Takođe, uočimo da je izabran QoS nivo 0 i isključena opcija zadržavanja poruke.

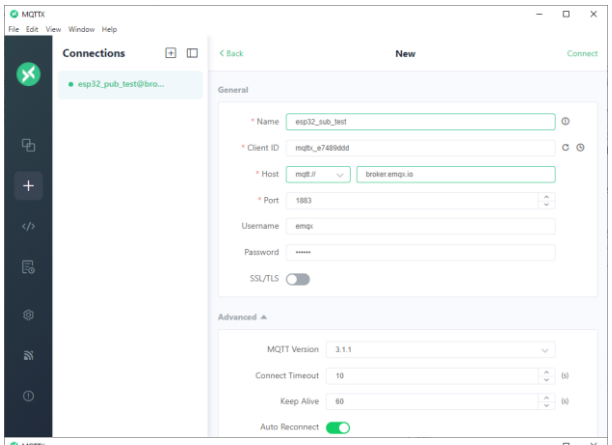
Poruka se šalje (publikuje) klikom na zeleno dugme dole desno.

Nakon publikovanja, poslata poruka, zajedno sa temom i vremenom slanja je ispisana u gornjem panelu (←).



Pošaljimo još jednu poruku „Test publikovanja 2“ na istu temu, ali sada sa uključenom opcijom za zadržavanje poruke (←).

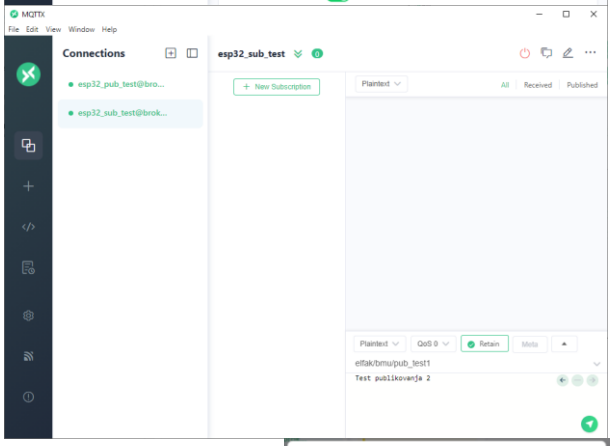
Međutim, dve poslate poruke nema ko da primi, jer ne postoje klijenti koji su prijavljeni na temu „elfak/bmu/pub_test1“.



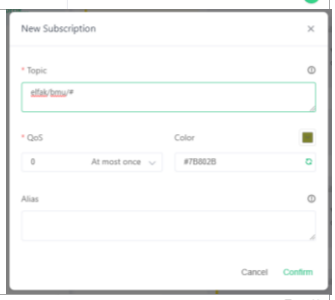
Sada ćemo kreirate jednog takvog klijenta (←).

Ponavljamo postupak za kreiranje nove konekcije.

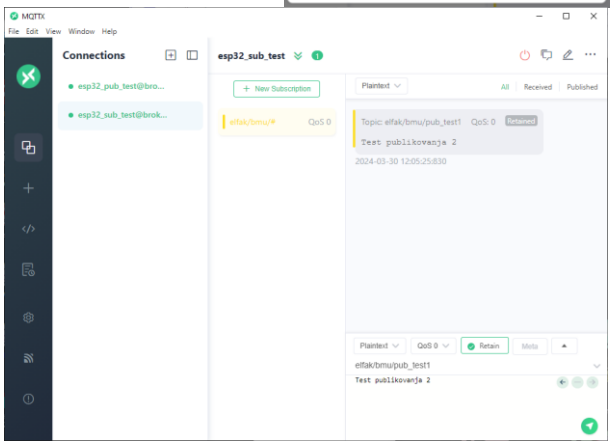
Nova konekcija je nazvana „esp32_sub_test“. Ostali parametri su isti kao kod konekcije „esp32_pub_test“, s tim što sada nije definisana poslednja želja.



Nakon klika na *Connect*, u panelu *Connections* sada vidimo dve konekcije. Izabrana je upravo kreirana nova konekcija (←).



Novu konekciju možemo razumeti kao da je to novi MQTT klijent. Tog novog klijenta pretplatićemo na temu „elfak/bmu/#“. Potrebno je kliknuti na dugme „+ New Subscription“, zatim upisati naziv teme na koju se pretplaćujemo i klikunti na *Confirm* (←).

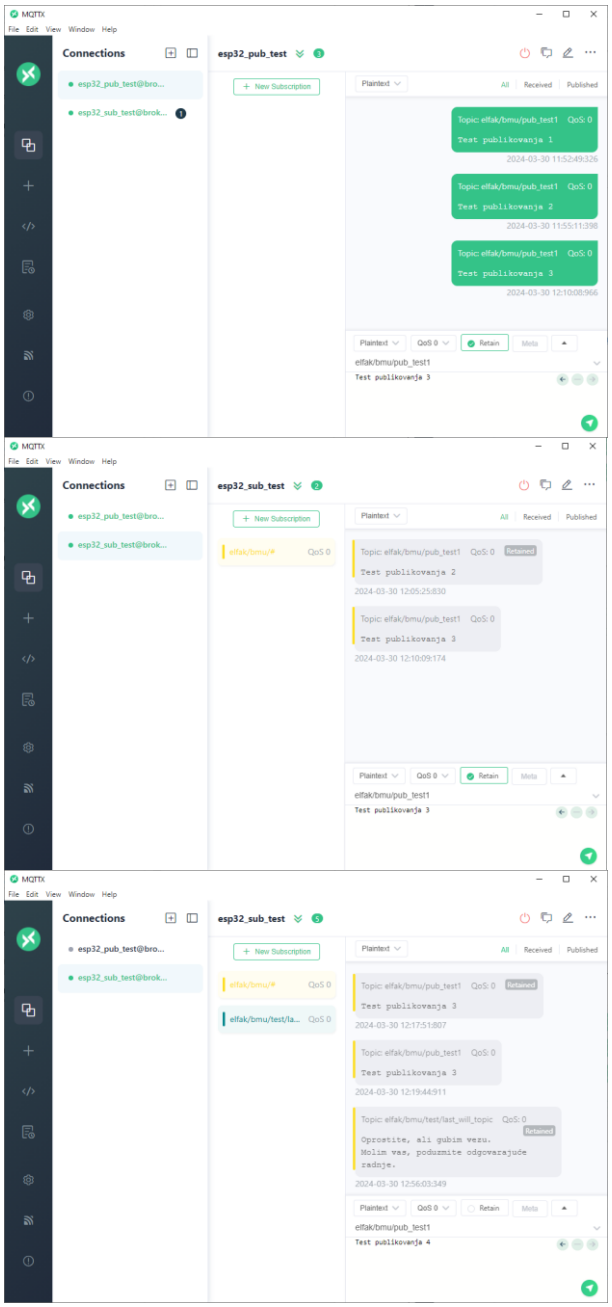


Izgled dijaloga programa MQTTX nakon pretplate vidimo na slici (←). U panelu za pretplate navedena je teme na koju smo se upravo pretplatili.

U desnom panelu ispisana je poruka koju je ovaj klijent primio odmah nakon pretplate. Klijent „esp32_pub_test“ je poslao dve poruke: prva sa isključenom i druga sa uključenom opcijom zadržavanja poruke. Klijent „esp32_sub_test“, primi je, naravno, samo drugu poruku.

Neka sada klijent „esp32_pub_test“ pošalje još jednu poruku „Test publikovanja 3“ na istu temu (←).

U panelu za konekcije vidimo da klijenta „esp32_sub_test“ čeka jedna nepročitana poruka. Ako u ovom panelu izaberemo klijenta „esp32_sub_test“ videćemo da mu je broker isporučio poruku koju je prethodno publikovao klijent „esp32_pub_test“ (sledeća slika).



Na kraju, da proverimo šta će se desiti kada raskinemo konekciju klijenta „esp32_pub_test“. U panelu *Connections* treba izabrati „esp32_pub_test“, a zatim klik na dugme za *Disconnect* (gore desno, oblika simbola za isključenje napajanja). Konekcija je sada raskinuta, ali klijent „esp32_sub_test“ nije primio nikakvu poruku o poslednjoj želji klijenta „esp32_pub_test“. To je zato što je *Disconnect* regularno isključene klijenta (klijent šalje brokeru poruku DISCONNECT), a poslednja želja se šalje samo u slučaju neregularnog prekida konekcije.

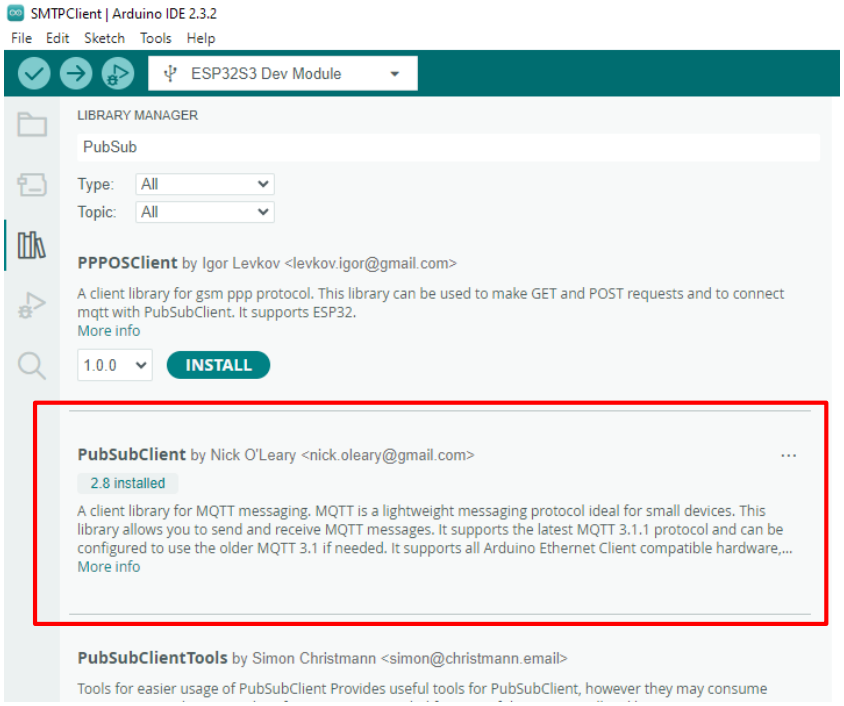
Neregularni prekid konekcije možemo simulirati tako što ćemo na laptopu isključiti WiFi mrežu ili izvući mrežni kabl. Nakon toga, potrebno je sačekati da istekne vreme *Keep Alive* (nešto više od 1 minuta) i ponovo uključiti mrežu.

Pošto se u međuvremenu diskonektovao i klijent „esp32_sub_test“, potrebno je ponovo ga konektovati. U trenutku konektovanja, ovaj klijent dobija poruku: „Oprostite, ali gubim vezu. Molim vas, poduzmite odgovarajuće radnje.“, što je zapravo poslednja želja klijenta „esp32_pub_test“ (←)

Na osnovu prethodnog, MQTT možemo razumeti i kao protokol za ćaskanje (*chat*) između pametnih uređaja. Kao što ljudi imaju *Viber* ili *WhatsApp* tako pametni uređaji imaju MQTT. Zapravo, od svih tih „ljudskih“ aplikacija, najsličniji MQTT-u, po konceptu, je *Twitter* (tj. X), gde korisnici modu da publikuju svoje poruke i da se pretplaćuju na različite teme.

3.5 ESP32-S3 MQTT klijent

U ovom primeru, realizovaćemo jednostavan MQTT sistem koji se sastoji od modula ESP32-S3 u ulozi MQTT oglašivača i pretplatnika, programa MQTTX u ulozi MQTT pretplatnika i javnog MQTT brokera EMQX. Za razvoj MQTT aplikacije na modulu ESP32-S3 koristićemo Arduino biblioteku *PuSubClient*, koju je potrebno, uz pomoć *Library Manager*-a instalirati u Arduino okruženje (slika). Dokumentacija za biblioteku *PubSubClient* je dostupna na linku https://pubsubclient.knolleary.net/api.



Sl. 4 Instalacija *PutSubClient* biblioteke

Program u listingu realizuje ESP32-S3 MQTT klijent sledeće funkcionalnosti. Na svaku promenu stanja tastera, ESP32-S3 publikuje poruku na temu „*elfak/bmu/pbstate*“. U zavisnosti od novog stanja tastera, sadržaj publikovane poruke je „HIGH“ ili „LOW“. ESP32-S3 je takođe pretplaćen na temu „*eflak/bmu/ledcmd*“. Na ovoj temi se objavljuju poruke za upravljanje LED: „*turn on*“ pali, dok „*turn off*“ gasi LED.

U liniji 8 je instanciran objekat *client* klase *PubSubClient*. Preko ovog objekta dostupna je celokupna funkcionalnost MQTT klijenta. Prilikom instanciranja, objektu *client* je prosleđena referenca na objekat *wifiClient*. Objekat *client* će (interno) koristiti *wifiClient* za WiFi i TCP/IP komunikaciju. U linijama 11-14, deklarisani su parametri MQTT brokera (URL, port, korisničko ime i lozinka). U linijama 16-17 su deklarisane MQTT teme. *Pub_topic* je tema na koju će ESP32-S3 da publikuje svoje poruke, dok je *sub_topic* tema na koju će ESP32-S3 da se pretplati.

L. 1. Program *MQTTPubSub*

```
1  #include <WiFi.h>
2  #include <PubSubClient.h>
3
4  const char ssid[] = "***";       //SSID WiFi mreze
5  const char password[] = "***"; //lozinka WiFi mreze
6
7  WiFiClient wifiClient;
8  PubSubClient client(wifiClient);
9
10 //MQTT Broker
11 const char mqtt_broker[]   = "broker.emqx.io";
12 const char mqtt_username[] = "emqx";
13 const char mqtt_password[] = "public";
14 const int  mqtt_port = 1883;
15 //topics
16 const char pub_topic[] = "elfak/bmu/pbstate";
17 const char sub_topic[] = "elfak/bmu/ledcmd";
18 //button & LED
19 const int bttnPin = 2;  // broj pina tastera
20 const int ledPin = 42;  // broj pina LED
21 int bttnState;      //tekuce stanje tastera
22 int lastBttnState;  //prethodno stanje tastera
```

```
23
24  void setup() {
25    pinMode(bttnPin, INPUT_PULLDOWN);
26    pinMode(ledPin, OUTPUT);
27    digitalWrite(ledPin, LOW);
28    Serial.begin(115200);
29    //povezivanje na WiFi mrezu
30    WiFi.begin(ssid, password);
31    while(WiFi.status() != WL_CONNECTED){
32      Serial.print(".");
33      delay(300);
34    }
35    while (WiFi.localIP() == INADDR_NONE) {
36      Serial.print(".");
37      delay(300);
38    }
39    Serial.println("\nConnected to the WiF network");
40    IPAddress ip = WiFi.localIP();
41    Serial.print("IP adresa   : "); Serial.println(ip);
42
43    //konektovanje sa MQTT brokerom
44    client.setServer(mqtt_broker, mqtt_port);
45    client.setCallback(mqtt_callback);
46    while (!client.connected()){
47      String client_id = "esp32-client-";
48      client_id += String(WiFi.macAddress());
49      Serial.printf("The client %s connects to MQTT broker\n",client_id.c_str());
50      if(client.connect(client_id.c_str(),mqtt_username,mqtt_password)) {
51        Serial.println("Public EMQX MQTT broker connected");
52      } else {
53        Serial.print("failed with state ");
54        Serial.print(client.state());
55        delay(2000);
56      }
57    }
58    //publikovanje trenutnog stanja tastera
59    bttnState = digitalRead(bttnPin);
60    lastBttnState = bttnState;
61    if(bttnState == HIGH)
62      client.publish(pub_topic, "HIGH");
63    else
64      client.publish(pub_topic, "LOW");
65    //pretplata na temu "LED komanda"
66    client.subscribe(sub_topic);
67  }
68
69  //funkcija koja se poziva po prijemu poruke od MQTT brokera
70  void mqtt_callback(char *topic, byte *payload, unsigned int length) {
71      //konverzija sadrzaja poruke u String
72      String msg = "";
73      for (int i = 0; i < length; i++) {
74          msg += (char)payload[i];
75      }
76      //stampanje teme i sadrzaja poruke
77      Serial.print("Message arrived in topic: "); Serial.println(topic);
78      Serial.print("Message: "); Serial.println(msg);
79      Serial.println();
80      //izvrsenje komande
81      if(msg == "turn on")
82        digitalWrite(ledPin, HIGH);
83      else if(msg == "turn off")
84        digitalWrite(ledPin, LOW);
85  }
86
87  void loop() {
88    //promena stanja tastera inicira publikovanje poruke na temu "stanje tastera"
89    bttnState = digitalRead(bttnPin);
90    if(bttnState != lastBttnState){
91      if(bttnState == HIGH)
92        client.publish(pub_topic, "HIGH");
```

```
93      else
94        client.publish(pub_topic, "LOW");
95    }
96    lastBttnState = bttnState;
97    client.loop();   //neophodno za rad MQTT klijenta!
98    delay(100);
99  }
```

Povezivanje na WiFi mrežu u funkciji *setup*() je realizoavano na od ranije poznat način. Nakon WiFi povezivanja sledi konektovanje sa MQTT brokerom. U naredbi *setServer*() u liniji 44, MQTT klijentu se dostavlja URL i port brokera. U sledećoj liniji, u naredbi *setCollback*(), klijentu se takođe dostavlja i tzv. *callback* funkcija, odnosno funkcija koju će klijent pozivati uvek nakon što primi poruku od brokera. Ova funkcija je neophodna ukoliko klijent namerava da se kod brokera pretplati na temu (jednu ili više). U *while* petlji koja sledi (linije 46 – 57) čeka se na uspostavljanje veze između klijenta i brokera. Status povezanosti se ispituje funkcijom *connected*(), koja vraća *true* ukoliko je MQTT klijent konektovan sa brokerom. Pri prvom ispitivanju uslova *while* petlje, *connected*() svakako vraća *false*, jer procedura konektovanja sa brokerom nije još uvek inicirana. Ključna naredba u telu *while* petlje je u liniji 50 gde se, radi povezivanja sa brokerom, poziva funkcija *connect*(). Ova funkcija prvo otvara TCP konekciju sa brokerom, a zatim šalje brokeru MQTT poruku CONNECT. Argumenti funkcije *connect*() su: korisničko ime i lozinka, što je definisano konstantnim stringovima na početku programa i ID klijenta, koje se formira programskim kôdom u linijama 47 i 48. Kao što je ranije rečeno, ID klijenta, tj. *ClientID* mora biti jedinstven u okviru brokera. Da bi se obezbedila jedinstvenost, *ClientID* se formira nadovezivanjem stringa „**esp32-client-**“ i MAC adrese ESP32-S3 modula. Funkcija *connect*() vraća *true* u slučaju uspešnog konektovanja, odnosno *false* ako postoje problemi u povezivanju. U slučaju neuspešnog povezivanja, cela ova procedura se ponavlja posle $2s$.

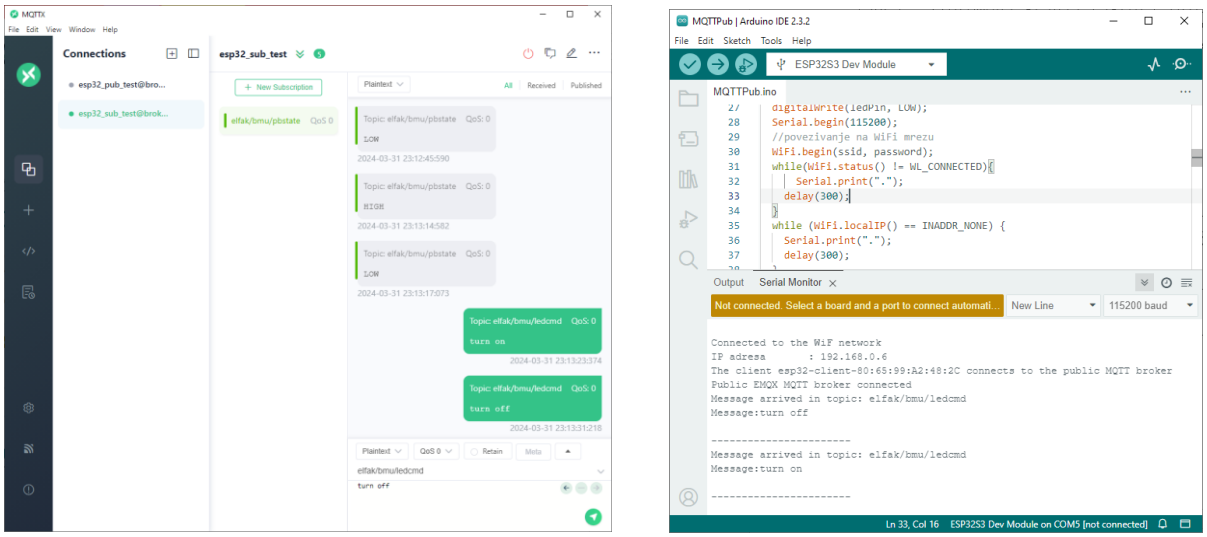
Nakon što je uspešno povezan sa brokerom, ESP32-S3 može da publikuje svoje podatke i da se pretplaćuje na teme. U linijama 59 – 67, ESP32-S3 publikuje, na temu *pbstate*, trenutni status tastera. Ako je taster pritisnut, sadržaj publikovane poruke biće „HIGH“; u suprotnom, u poruci se šalje „LOW“. Konačno, na kraju funkcije *setup*(), u liniji 66, ESP32-S3 se pretplaćuje na temu "*elfak/bmu/ledcmd*".

Programski kôd u funkciji *loop*() je jednostavan. Ispituje se stanje tastera i u slučaju promene, publikuje se poruka odgovarajućeg sadržaja na temu "*elfak/bmu/ledcmd*". Potrebno je uočiti poziv funkcije *client.loop*() u liniji 97. Ovu funkciju je neophodno pozivati iz *loop*() kako bi MQTT klijent, *client*, mogao da obavlja svoje aktivnosti.

Razmotrimo sada funkciju *mqtt_callback*(),linije 70 – 85. Kao što je prethodno rečeno, ova funkcija će biti izvršena uvek kada MQTT klijent primi poruku od brokera. Pri tom, MQTT klijent dostavlja ovoj funkciji, preko njenih argumenata, temu i sadržaj primljene poruke. Tačnije, funkciji se dostavlja pointer na string koji sadrži temu poruke (*char *topic*), i pointer na niz bajtova koji sadrži samu poruku (*byte *payload*). String je niz karaktera sa 0 na kraju. Niz bajtova ne sadrži oznaku za kraj niza. Zbog toga je potreban i treći argument, *unsigned int length*, koji sadrži dužinu poruke, odnosno niza bajtova, izraženu brojem bajtova. Na početku funkcije *mqtt_callback*(), niz bajtova je prepisan, bajt po bajt, u string *msg*. To učinjeno zato što se stringom može lakše manipulisati nego nizom bajtova. Na primer, string *msg* se može direktno odštampati u naredbi *Serial.println*(*msg*), linija 78, što sa nizom bajtova ne bi bilo moguće. Takođe, dva stringa se mogu upoređivati operatorom „==“, što se koristi u delu programskog kôda u linijama 81-84, gde se na osnovu sadržaja primljene poruke, LED pali ili gasi.

Programa *MQTTPubSub* je testiran pomoću aplikacije MQTTX, koja je takođe konektovana za broker EMQX i pretplaćena na temu "*elfak/bmu/pbstate*" (SI. 5). Prve tri poruke u panelu za poruke potiču od ESP32-S3. Nakon startovanja, ESP32-S3 publikuje poruku o trenutnom stanju tastera (prva poruka „LOW“). Posle

toga, taster je pritisnut, a onda i otpušten, što je izazvalo publikovanje još dve poruke na temu "*elfak/bmu/pbstate*" (prvo „HIGH“, onda „LOW“). U nastavku testiranja, iz aplikacije MQTTX je poslata poruka „*turn on*“ na temu "*elfak/bmu/ledcmd*", a onda i poruka „*turn off*“, a istu temu. Kao što se može videti u serijskom monitoru Arduino IDE, ESP32-S3 je primio ove dve poruke, jer je pretplaćen na temu "*elfak/bmu/ledcmd*". Ono što se ne vidi je to da je, kao odgovor na poruke „*turn on*“ i „*turn off*“, ESP32-S3 prvo upalio, a onda ugasio LED.



Sl. 5 Testiranje programa MQTTPubSub

Napomena 1: Sintaksa funkcije za konektovanje sa MQTT brokerom je oblika:

boolean **connect** (clientID, [username, password], [willTopic, willQoS, willRetain, willMessage], [cleanSession])

Argumenti funkcije *connect* koji su navedeni u srednjim zagradama, [], su opcioni. U našem programu smo naveli samo prvi tri argumenta, *clientID*, korisničko ime i lozinku, dok smo ostale, prosto, izostavili. Bez navođenja argumenata koji se tiču poslednje želje, ova opcija će biti isključena. Ako se izostavi poslednji argument, *cleanSession*, otvoriće se, podrazumevano, čista sesija.

Pretpostavimo da želimo da uvedemo opciju poslednje želje tastera. Dovoljno je samo modifikovati poziv funkcije *connect*:

client.connect(client_id.c_str(), mqtt_username, mqtt_password, „pb_last_will_topic“, 0, 0, „pb is offline“);

Na ovaj način, kreirana je poslednja želja sa temom „*pb_last_will_topic*“ i sadržajem „*pb is offline*“. QoS nivo poslednje želje je 0 i opcija zadržavanja ove poruke je isključena.

Napomena 2: Puna sintaksa funkcije *publish* je oblika:

boolean **publish** (topic, payload, [length], [retained])

I u ovoj funkciji uočavamo dva opciona argumenta, koja nismo krostili našem programu. Argument *length* je dužina sadržaja poruke, *payload*, izražena brojem bajtova. Funkcija *publish* dozvoljana da argument *payload* bude tipa pointer na niz karaketera (*char*[]), ili pointer na niz bajtova (*byte*[]). U prvom slučaju, funkcija podruazumeva da se niz karaktera završava 0 i zato nije potran podatak o dužini poruke. (Ovo je opcija koja smo koristili u našem programa). Ako se sadržaj poruke prenosi kao niz bajtova, tada je obavezno preneti funkciji dužinu ovog niza preko argumenta *length*. Drugi opcioni parametar, *retained*, koji je tipa *boolean*, određuje da li broker treba da zadrži publikovanu poruku (*retained=true*) ili ne (*retained=false*). Ako se ovaj argument ne navede u pozivu funkciju *publish*, podrazumeva se *retained=false*.

Napomena 3: Puna sintaksa funkcije *subscribe* je oblika:

boolean subscribe (topic, [qos])

Opcioni argument *qos* je ceo broj 0, 1 ili 2 koji definiše QoS nivo. Ako je argument *qos* izostavljen, podrazmevano važi QoS nivo 0.

Napomena 4: U programu MQTTPubSub, u dva navrata, koristi se tip podataka *String*: u funkciji *setup*() prilikom formiranja ClientID i u funkciji *mqtt_callback*() za smeštanje sadržaja primljene poruke. U Arduino C jeziku, postoje dva načina za predstavljanje stringova. Klasičan način je da se string defniše kao niz karaktera završen bajtom 0. Npr. u naredbi:

```
char mqtt_broker[] = "broker.emqx.io";
```

Promenljiva *mqtt_brocker* je string čiji je sadržaj "broker.emqx.io". Iako nije eksplicitno navedeno, kompajler automatski proširuje niz karaktera bajtom 0 da bi označio kraj stringa. To onda, na primer, omogućava da se promenljiva *mqtt_brocker* koristi kao argument u funkciji *print*:

```
Serial.print(mqtt_broker);
```

Funkcija *print*() štampa (šalje na serijski kanal) karakter po karaketer stringa *mqtt_brocker* sve dok ne naiđe na oznaku za kraj stringa (0). Bez 0 na kraju, štampanje bi se nastavilo i nakon kraja stringa.

Za rad sa „klasičnim“ stringovima dostupne su standardne C funkcije, npr.

sizeof(str)	Vraća dužinu niza karaktera
strlen(str)	Vraća dužini stringa koji je smešten u nizu karaktera
strcat(str, str1)	Na string str nadovezuje string str1
strcpy(str1, str)	Kopira string str u string str1
strcmp(str, str1)	Poredi stringove str i str1

Na primer, ako je string definisan kao: `char str[] = „primer stringa“;` tada će *sizeof*(*str*) vratiti 15 (jer se računa i pridodata 0), dok će *strlen*(*str*) vratiti 14.

Druga vrsta stringova u Ardino C-u je dostupna kroz tip podataka *String* (sa velikim S) koji podržav bogatiji skup operacija i omogućava komtniji rad sa strigovima. Na primer, ako želimo da na string *str* nadovežemo string *str1*, dovoljno je napisati: *str* += *str1*; (isto što i *str* = *str* + *str1*). Ako želimo da uporedimo dva stringa, umesto funkcije *strcmp*(), možemo koristiti operatore „==“ i „!=“. Na primer, u kôd iz linija 47 i 48:

```
String client_id = "esp32-client-";
client_id += String(WiFi.macAddress());
```

U prvo liniji je deklarisan String *client_id* sadržaja "*esp32-client-*". U drugoj liniji, string *client_id* je proširen stringom koji sadrži MAC adresu ESP32-S3 modula. Funkcija *WiFi.macAddress*() vraća MAC adresu u obliku klasičnog stringa (npr. 01:40:3F:29:C8:A1). *String*(*WiFi.macAddress*()) konvertuje klasičan string u tip *String*, koji se onda nadovezuje na string *client_id*. Konačno, sadržaj stringa *client_id* je: "*esp32-client-*01:40:3F:29:C8:A1". Uočimo da je string *client_id* nakon prve linije imao dužinu 14, a da je nakon druge linije njegova dužina porasla na 32. Ovo dinamičko proširenje niza kraktera je obavljeno automatski. Ako bismo isti kôd želeli da napišemo korišćenjem klasičnih stringova onda bismo u startu morali da predvidimo kolika će biti konačna dužina stringa client_id:

```
char client_id[32];
strcpy(client_id,"esp32-client-");
strcat(client_id, WiFi.macAddress());
```

Za više detalja o tipu podataka String pogledajte na linku:

https://www.arduino.cc/reference/en/language/variables/data-types/stringobject/

Na primer, korisne su funkcije za konverziju celog broja u String i obrnuto. Primera radi:

```
int n = 13;
String str = String(n);
```

Sadržaj stringa *str* je „13“ (vrednost celobrojne promenljive *n* predstaljena u obliku ASCII stringa, tj. teksta).

```
String str = String("13");
int n = str.toInt();
```

String *str* iz prve naredbe sadrži tekst „13“. U drugoj naredbi celobrojna promenljiva n dobija vrednost 13.

3.6 Zadatak

Zadatak 3.1 (20): Projektovati i realizovati MQTT sistem koji se sastoji iz dva ESP32-S3 modula, aplikacije MQTTX i koristi broker EMQX. Na jedan ESP32-S3 modul, M1, su povezani senzor temperature i LED modul, dok su na drugi, M2, povezani: dispjel, LED modul i tasterski modul. Modul M1 periodično meri temperaturu, a modul M2 prikazuje na displeju izmerenu temperaturu. Pritisak (klik) na jedan od dva taster modula M2 menja stanje zelene, a pritisak na drugi taster menja stanje crvenu LED modula M1.

Zadatak 3.2 (25): Program iz zadatka 3.1 proširiti tako da modul M1, uvek kada promeni stanje neke LED, oglašava novo stanje svojih LED, kako bi modul M2 postavio svoje LED u isto stanje.

Zadatak 3.3 (30): Program iz zadatka 3.2 proširiti poslednjom željom modula M1. Reakcija modula M2 na prijem poslednje želje modula M1 bi mogla da bude određeni ispis na displeju, ili blinkanje jedne ili obe LED.

Napomene:

- Deo zadataka je i definisanje MQTT tema
- Poruke bi trebalo da se publikuju sa zadržavanjem (*retained*).
- Rad sistem bi tebalo da se prati preko aplikacije MQTTX koja bi bila pretplaćena na sve teme koje postoje u sistemu.

- ♦ ♦ ♦ -