SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5417

Implementacija sustava za nadziranje i upravljanje bežičnim razvojnim modulima ESP8266

Ivan Trubić

Zagreb, lipanj 2018.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

SADRŽAJ

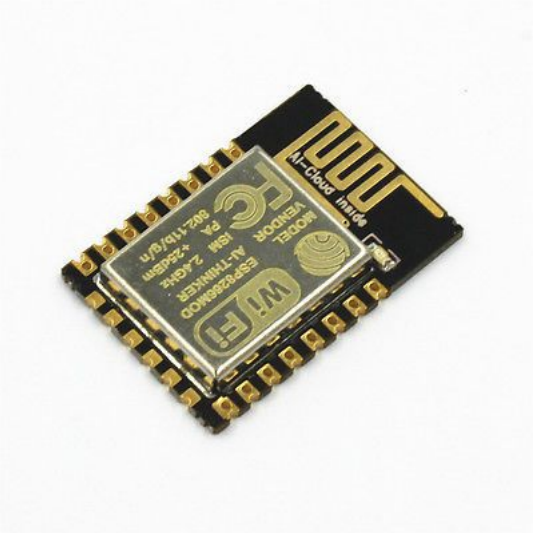
1. Uvod

ESP8266 je System on Chip (SoC) rješenje tvrtke Espresiff Systems vrlo malene površine mnogih mogučnosti. Dizajniran je kao vrlo jeftin modul sa mogučnošću spajanja na WiFi mrežu te niskom potrošnjom energije što ga čini vrlo povoljnim za ugrađivanje u IoT (*Internet Of Things*) uređaje. U ovome će radu ESP8266 *SoC* biti korišten kao krajnji čvor u sustavu bežičnog prikupljanja podataka, nadziranja te upravljanja koji komunicira s ostalim sustavom koristeći *Message Queueing Telemetry Transport* protokol (MQTT). Topologija takvog sustava se sastoji od krajnjeg čvora na kojem je spojen neki uređaj koji želimo kontrolirati ili nadzirati, pristupne točke WiFi mreži, poslužitelja na kojemu se nalaze MQTT broker za upravljanje porukama, baza podataka koja sadrži trenutne podatke čvorova te API kao sučelje između baze podataka i krajnjeg klijenta. Ovakav sustav je vrlo jednostavan i jeftin za implementaciju te je izrazito jako skalibilan što znači da pisanjem vrlo jednostavnog programskog koda za pojedini čvor možemo u svoju mrežu postaviti virtualno beskonačno krajnjih čvorova kojima se vrlo jednostavno može upravljati. Cilj ovoga rada je napraviti gotov sustav otvorenoga koda koji će prateći *Homie* konvenciju za MQTT protokol olakšati amaterima brzo rađenje prototipa bilo iz znatiželje, potrebe ili kreativnosti.

2. Razrada

2.1. ESP8266

ESP8266 je *SoC* koji u sebi ima mikrokontroler RISC arhitekture te WiFi modul za spajanje na mrežu. Ciljano tržište za ovaj čip su hobisti koji žele neku svoju ideju pretvoriti u stvarnost na vrlo jednostavan način. Dizajniran je vrlo jednostavno i kompaktno kako bi cijena čipa bila što niža te zato što je namjenjen za jednostavne i lokalizirane poslove koji uključuju upravljanje vrlo malim brojem uređaja odjednom. Točno u tome leži prava snaga čipa jer možemo imati više čipova koji komuniciraju međusobno umjesto da imamo jedan snažan mikrokontroler koji će upravljati uređajima centralizirano.



Slika 2.1: ESP8266 *SoC*

2.2. Značajke

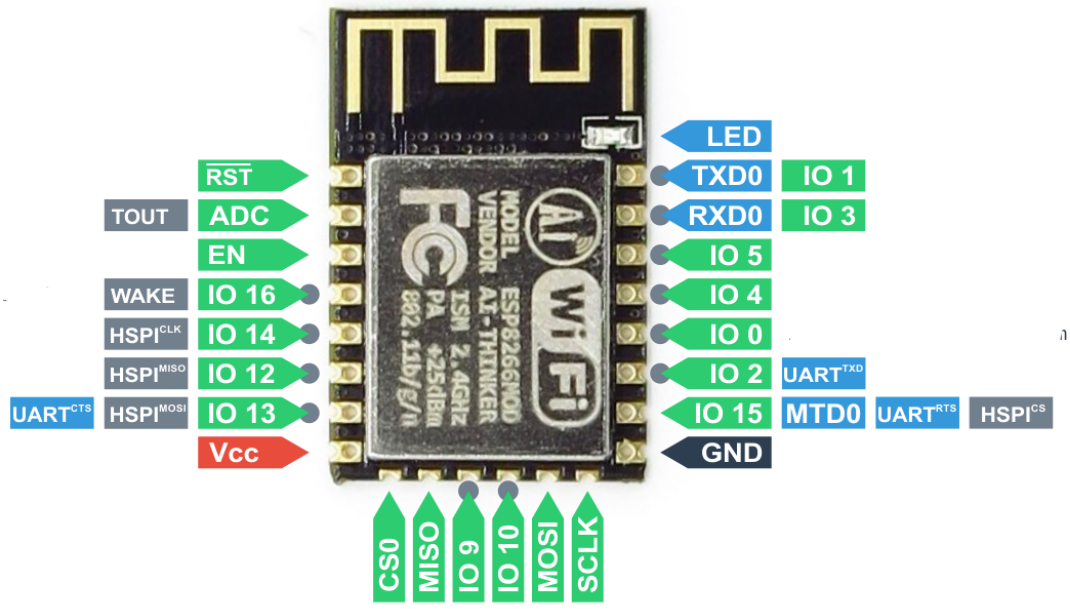
ESP8266 u sebi ima 32-bitni RISC mikrokontroler koji radi na frekvenciji od 80MHz. Memorija se sastoji od:

- 32 KiB instrukcijskog RAM-a
- 32 KiB priručne memorije instrukcijskog RAM-a
- 80 KiB korisničkog RAM-a

Vanjske QSPI *flash* memorije do 16MiB ovisno o modelu, IEEE 802.11 b/g/n WiFi, 16 GPIO pina, SPI sučelje, softverska implementacija I2C protokola, I2S sučelje, UART te 10-bitni analogno-digitalni pretvornik.

2.3. Povezivost

Najbitnije sučelje prema vanjskome svijetu na ESP8266 su bežična mreža i 16 GPIO pinova. Mreža podržava IEEE 802.11 b/g/n protokol, P2P konfiguraciju te TCP/IP protokolni stog. GPIO pinovi se mogu zasebno programirati kao ulazi/izlazi, postaviti im *pull-up/down* otpornike, te svaki ima ugrađen 10 bitni digitalno-analogni pretvornik tako da se svaki pin ponaša kao analogni izlaz. Ima također jedan analogan ulaz sa 10 bitnim analogno-digitalnim pretvornikom za spajanje raznih analognih senzora bez posebnih sučelja.



Slika 2.2: Izvodi ESP8266

2.4. Programiranje

ESP8266 se može programirati na više različitih načina. Originalno se programirao uz pomoć drugoga mikrokontrolera sve dok firma Espressif Systems nije izdala poseban *SDK*(eng. *Software Development Kit*) nakon čega to više nije bilo potrebno. Nakon toga su se počeli javljat mnogi SDK-ovi ponajviše otvorenog koda zajednice programera od čega su najpoznatiji:

- NodeMCU - baziran na skriptnom programskom jeziku Lua
- Arduino - baziran na jeziku C++ iste sintakse kao i bilokoji arduino mikrokontroler
- PlatformIO - nova razina razvoja ugradbenih sustava sa posebnim IDE-om
- MicroPython - baziran na programskom jeziku python

Slika 2.3: Arduino razvojno okruženje

Zbog velike raširenosti Arduino mikrokontrolera je zajednica programera razvila SDK za Arduino razvojno sučelje. Cilj je bio približavanje ESP8266 modula ama-

terima te olakšanje razvoja firmware-a kroz poznatu sintaksu, poznato jednostavno razvojno okruženje te već postoječe biblioteke.

2.5. Usporedba drugih razvojnih modula koji koriste WiFi komunikacijski protokol sa ESP8266 modulom

Pronalazak alternativnih modula se pokazalo izazovno zbog tolike raširenosti ESP modula. U tablici 2.1 je vidljivo i zašto je to tako. Moduli CC3000 i RN-131 su u zajednici i od strane proizvođača označeni kao NRND (engl. *Not Recommended for New Designs*) jer više nisu podržani. Cijenovno niti funkcionalno nisu više adekvatni za korištenje u *IoT* svrhe.

Kategorija	ESP8266	ESP32	CC3000	RN-131
WiFi Standard	802.11 b/g/n	802.11 b/g/n	802.11 b/g	802.11 b/g
Tip paketa	TCP i UDP	TCP i UDP	TCP i UDP	TCP i UDP
Režim rada	Klijent i Server	Klijent i Server	Klijent i Server	Klijent i Server
AP režim	P2P i Soft-AP	Soft-AP	Ne	Soft-AP
Dimenzije [mm]	26x16x3	26x18x3	16.3x13.5x3	20x37x3.5
Sučelje	TTL-Serial	TTL-Serial	SPI	SPI
Enkripcija	WPA2-PSK	WPA2 i WAPI	WPA-PSK	WPA2-PSK
Struja u sleep režimu rada	<10 μA	5 μA	?	4μA
Struja napajanja	80mA	500mA	?	40mA
Napon napajanja	3.0-3.6V	2.3-3.6V	2.7-3.6V	3.0-3.7
Digitalni pinovi	9	34	0	10
Analogni pinovi	1	18	0	8
Sadrži mikrokontroler	Da	Da	NE	Ne
Cijena [$]	6.95	8.95	34.95	35.38

Tablica 2.1: usporedba značajki WiFi modula i cjena

C3000 i RN-131 moduli nemaju programljivi mikrokontroler, ne koriste suvremeni 802.11n standard koji je znatno brži te su cjenovno i po nekoliko puta skuplji od vrlo močnih ESP modula. Iz tog su se razloga ESP moduli probili na tržištu i u tolikoj se mjeri proširili.

Slika 2.4: Modul ESP32

Slika 2.5: Modul CC3000

Slika 2.6: Modul RN-131

2.6. Message Queuing Telemetry Transport - MQTT

MQTT je protokol široko primjenjen u *IoT* sustavima temeljen na prijenosu poruka u režimu pretplate i objave. Prijenos poruka se vrši preko TCP/IP protokola te se za to brine program na poslužitelju koji ima ulogu brokera. Pretplata i objava se vrše na temelju teme (eng. *topic*) gdje uređaji MQTT brokeru šalju poruke koje taj broker ovisno o temi šalje drugim uređajima koji su na tu temu pretplačeni. Prava moć MQTT protokola su teme koje zamjenjuju MAC i IP adrese uređaja kako bi postavljanje infrastrukture postalo vrlo trivijalno i lagano za implementirati. Tema je oblika *"ovo/je/primjer/teme"* gdje je svaka razina teme odvojena separatorom *"/"*. Tema može imati bilokoliko razina te se one mogu adresirati pomoću zamjenskih znakova:

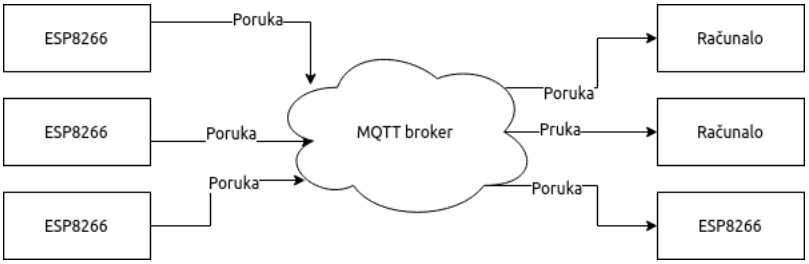
- '+' - predstavlja jednu razinu u temi. Npr. tema *"mojdom/prizemlje/+/temperatura"* će prikazivati sve poruke na temama *"mojdom/prizemlje/kuhinja/temperatura"* te *"mojdom/prizemlje/dnevnasoba/temperatura"*.
- '#' - predstavlja više razina u temi. Npr. tema *"mojdom/prizemlje/#"* će prikazivati sve teme koje su podskup ove teme.

Ovakva struktura tema je vrlo pregledna i čitka te se na taj način može vrlo dobro logički odjeliti svaki čvor unutar takve mreže što čini MQTT protokol vrlo pogodnim za *IoT* primjenu. Svaki uređaj se nalazi na svojoj temi koja pobliže opisuje gdje se taj uređaj nalazi te pomoću teme možemo iščitati semantiku poslane poruke. Poseban početni znak koji se može naći u temi je znak dolara *"$"* koji predstavlja unutarnje statistike samog MQTT brokera. Na takve teme ne možemo objavljivati poruke a nisu ni podložne zamjenskim znakovima tako da ako se uređaj pretplati na temu *"#"* dobivat će sve poruke osim tih koje se nalaze na temama unutarnjih statistika.

Poruke mogu biti i očuvane (eng. *retained*) tako da se prilikom objavljivanja poruke postavi zastavica očuvanja što MQTT brokrer interpretira tako da na toj temi sprema zadnju objavljenu poruku. Tako očuvana poruka je objavljena svim novim uređajima koji se pretplačuju na tu temu u nekom kasnijem trenutku tako da ti uređaji dobivaju trenutne vrijednosti na takvim temama. Ta je opcija vrlo korisna kod objavljivanja podataka primjerice nekih senzora zato što takvi uređaji odmah imaju trenutne vrijednosti s kojima odmah mogu početi baratati umjesto da čekaju tek sljedeću objavu.

MQTT klijent je bilokakav uređaj spojen na internet koji izvršava kod MQTT biblioteke te je spojeno na MQTT broker. Klijenti su implementirani već za gotovo sve platforme (npr. Arduino, iOS, Android) u raznim programskim jezicima (npr. Python, C, C++, C#, Java). Također i drugi ESP modul može biti klijent tako da izvršava kod ovisno o poruci nekoga drugoga uređaja.

MQTT broker je program koji se izvršava na nekom poslužitelju te upravlja porukama i uređajima koji su na njega spojeni. Brine se o pravim MAC i IP adresama uređaja na mreži koje su mu potrebne za samu realizaciju izmjena poruka kako se korisnici ne bi time trebali zamarati. Također se brine i o autorizaciji uređaja na mreži tako da prava na pretplačivanje mogu biti implementirana pomoću korisničkih imena i lozinki kako bi se očuvala privatnost podataka i spriječile razne maliciozne aktivnosti.



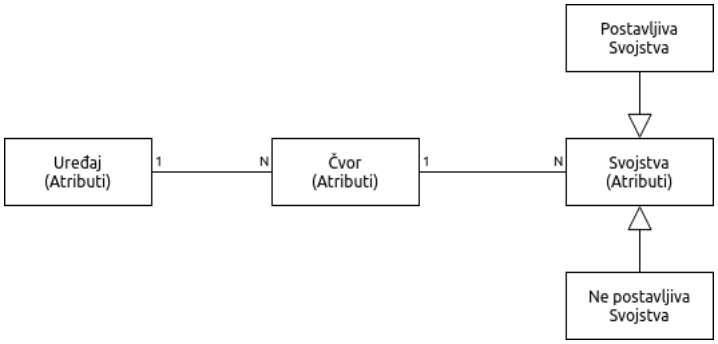
Slika 2.7: Uloga MQTT brokera

2.7. Homie konvencija za MQTT protokol

Homie konvencija je konvencija nastala iz potrebe zajednice za standardiziranim načinom komunikacije između različitih *IoT* uređaja koji koriste MQTT protokol. Njome se propisuje i implementira struktura tema i poruka kako bi se na temelju njih mogle implementirati pogodnosti kao što su praćenje verzija firmware-a, OTA (eng. *Over The Air*) osvježavanja tog istog firmware-a, omogućuje automatsko otkrivanje uređaja na mreži te osigurava lakše poimanje mreže uređaja i struktura podataka. Nalaže da je uređaj fizička instanca sklopovlja koji može logički prestavljati više čvorova. Čvor je logički odvojen dio uređaja kao na primjer neki senzor temperature ili relej koji upravlja svjetlom.

Jedan čvor može imati više svojstava koja ga pobliže označuju kao na primjer RGB Led dioda koja može imati svojstva *intenzitet* i *boja*. Svojstva mogu biti postavljiva ili ne postavljiva što čuva integritet tih svojstava tako da možemo objavljivanjem poruka mjenjati samo vrijednosti koje su promjenjive kao što su paljenje odnosno gašenje svjetla dok ne možemo mjenjati trenutnu temperaturu koju temperaturni senzor trenutno mjeri. Atributi pobliže označavaju uređaje, čvorove i svojstva. Oni se u temi označavaju sa početnim znakom "$" i vrlo su bitni kod već spomenute implementacije automatskog otkrivanja uređaja i čvorova.

Svaka tema u ovoj konvenciji ima korijen *"Homie"* nakon čega ide identifikator uređaja što može biti neki tekst definiran od strane programera ili uobičajeno MAC



Slika 2.8: Logička podjela Homie konvencije

adresa mrežnog adaptera. Svi atributi vezani uz uređaj slijede identifikatoru te počinju znakom "$". To su razni podaci od kojih je bitno za napomenuti:

- *$mac* - MAC adresa
- *$localip* - lokalna IP adresa
- *$nodes* - čvorovi
- *$state* - stanje uređaja
- *$stats* - statistika uređaja
- *$name* - ime uređaja

Nakon identifikatora uređaja slijedi identifikator čvora. Identifikator čvora je definiran od strane programera te mora biti jedinstven za taj uređaj. Atributi čvora su:

- *$name* - ime
- *$type* - tip
- *$propertyes* - svojstva

koji slijede identifikator čvora.

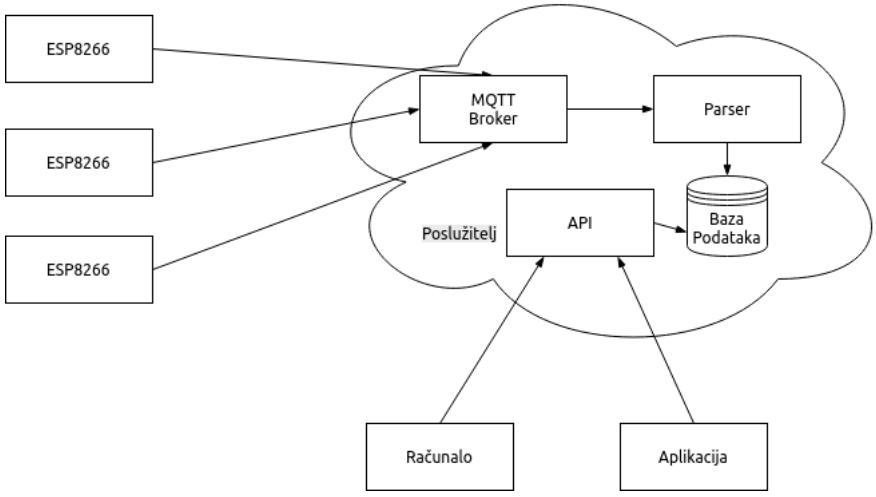
Slijedi svojstvo čvora koje također mora biti jedinstveno unutar jednoga čvora. Atributi svojstva su ime (*$name*), postavljivost (*$settable*), mjerna jedinica (*$unit*), tip podataka (*$datatype*) te format (*$format*) koji opisuje opseg vrijednosti.

Svaka poruka mora biti zapisana u string formatu radi konzistentnosti između uređaja.

Primjer teme svjetla spojenog na relej na nekom uređaju po Homie konvenciji je: "*Homie/ime-uređaja/ime-čvora/upaljeno-svjetlo*".

3. Programska podrška za praćenje komunikacije više modula ESP8266 na WiFi mreži

Kod implementacije programske podrške treba prvo napraviti topologiju sustava. Sustav ima krajnje uređaje koji svoje poruke šalju poslužitelju te poslužitelj te poruke prosljeđuje dalje pretplačenim klijentima. Pretplačeni klijent je u ovoj implementaciji program *"parser.py"* koji, kako samo ime kaže, parsira poruke te ih sprema na odgovarajuč način u bazu podataka. Bazi podataka treba pristupiti te se to vrši uz pomoć API-a koji vrši odgovarajući SQL upit te vraća sadržaj baze u JSON formatu. Ideja topologije sustava je prikazana na slici 3.1.



Slika 3.1: Topologija sustava

3.1. Implementacija MQTT klijenta na ESP8266 modul pomoću Homie konvencije

Arduino IDE je vrlo popularan u zajednici te je jako dobro podržan što od strane zajednice što od strane proizvođača Arduino mikrokontrolera. Zajednica je iz tog razloga napravila biblioteku funkcija za Homie konvenciju za ESP8266 modul. Biblioteka se lagano instalira pomoću ugrađenog upravljača bibliotekama u Arduino IDE-u upisom web sjedišta na kojoj se ta biblioteka nalazi te pritiskom ta tipku *Install*. U kodu se uključivanjem te biblioteke mogu koristiti sve funkcije upisom

Listing 3.1: Uključivanje Homie biblioteke

```
#include <Homie.h>
```

Arduino mikrokontroleri zahtjevaju posebne funkcije *setup()* i *loop()*. Uloga *setup* funkcije ulazna funkcija u koji se pišu programski odsječci za postavljanje samog mikrokontrolera nakon čega se beskonačno puta ponavlja funkcija *loop* u kojoj se nalazi kod koji se konstantno izvodi. U duhu te konvencije se u *setup* funkciji postavlja ime i verzija firmware-a za OTA postavljanje novog firmwera i poziva se setup metode instance Homie klase.

```
setup(){
    Homie_setFirmware("prvi-firmware", "1.0.0");
    Homie.setup();
}
```

U *loop* funkciji se poziva metoda *loop()* Homie objekta čija je funkcija kontrola veze te pozivanje funkcije za upravljanje porukama kada su one objavljene na temu na koju je pretplaćen uređaj.

Za stavaranje novog čvora (u ovome slučaju LED diode) trebamo napraviti novi objekt klase *HomieNode*

```
HomieNode ledNode("led", "switch");
```

kojem se postavlja ime i tip.

Čvor *ledNode* može imati svojstvo *on* koje označava da LED dioda može biti upaljena ili ugašena. To svojstvo je postavljivo jer se u ovome slučaju želi dioda paliti ili gasiti i definira se tako da se kao argument navede funkcija koja prilikom objave poruke pali ili gasi LED diodu.

```
setup(){
```

```
    Homie_setFirmware("prvi-firmware", "1.0.0");
    ledNode.advertise("on").settable(ledHandler);
    Homie.setup();
}
```

Taj se programski kod treba nalaziti u *setup* funkciji nakon postavljanja verzije firmware-a i prije funkcije za postavljanje Homie implementacije jer se prilikom tog postavljanja moraju navesti svi čvorovi koji se nalaze na tom uređaju.

Čvor ledNode je trenutno pretplačen na temu *Homie/DeviceID/ledNode/on*. Kad je na toj temi objavljena poruka funkcija ledHandler je pozvana i ona ovisno o sadržaju poruke upravlja LED diodom.

```
bool ledHandler(const HomieRange& range, const String& value){
  if(value == "ON"){
    digitalWrite(led, HIGH);
    ledNode.setProperty("on").send("true");
    Serial.println("Led is on");
  }

  else if(value == "OFF"){
    digitalWrite(led, LOW);
    ledNode.setProperty("on").send("false");
    Serial.println("Led is off");
  }

  else {
    Serial.println("Wrong message!");
    return false;
  }
  return true;
}
```

Programiranje samog uređaja je pomoću Arduino SDK veoma trivijalno. U padajućem izborniku se odabire odgovarajuć uređaj te se prema tome postavljaju sve ostale postavke te pritiskom na gumb *Upload* se kod prevede i prebacuje u memoriju uređaja. Pri tome na ESP8266 treba pin GPIO0 spojiti na logičku nulu kako bi prebacivanje programa bilo uspješno.

Uređaj se kod prvog priključivanja nalazi u konfiguracijskom režimu rada. U tak-

vom režimu se uređaj ponaša kao bežična pristupna točka lokalnoj mreži čije ime glasi *Homie-xxxxxxxxxxx* pri čemu je dio sa znakovima *x* MAC adresa uređaja. Računalom se spaja na uređaj pri čemu je lozinka točno ta MAC adresa i uređaj tako može primiti konfiguracijsku datoteku u JSON formatu. Datoteka izgleda ovako:

```
{
    "name": "Ledica",
    "device-id": "ledica",
    "wifi": {
        "ssid": "<ime_mreze>",
        "password": "<lozinka>"
    },
    "mqtt": {
        "host": "<ip_adresa_MQTT_brokera>",
        "port": 1883,
        "base_topic": "devices/",
        "auth": false
    },
    "ota": {
        "enabled": true
    }

}
```

Funkcija postavki je:

- name - Ime uređaja
- device-id - Jedinstveno ime uređaja - MAC adresa ako ne iskoristimo ovu postavku
- wifi - Ime i lozinka mreže na koju se uređaj spaja
- mqtt - Postavke MQTT brokera, IP adresa, port, korijenska tema (umjesto homie/), te autorizacija
- ota - Bežično reprogramiranje uređaja

U konfiguracijskom režimu rada uređaj izlaže funkcionalnosti JSON API-a preko kojeg možemo pozivati metode navedene u tablici 3.1 Prema tome se JSON datoteka s postavkama se šalje na uređaj pomoću programa *curl* koristeći /config funkciju naredbom:

Metoda	Funkcija	Opis	Odgovori
GET	/heart	Provjera povezanosti sa uređajem	204 No content
GET	/device-info	Informacije o uređaju	200 OK + JSON datoteka
GET	/networks	Ispis svih vidljivih mreža	200 OK, 503 Service Unavaliable
PUT	/config	Slanje konfuguracijske datoteke	200 OK, 400 Bad Request, 403 Forbidden
GET	/wifi/connect	Povezivanje sa mrežom	202 Accepted, 400 Bad Request
GET	/wifi/status	Ispis stanja veze	200 OK + JSON datoteka
PUT	/proxy/control	Postavljanje funkcije posrednika	200 OK, 400 Bad Request

Tablica 3.1: API pozivi

```
curl –X PUT http://homie.config/config ––header 'Content–Type: applic
```

nakon ćega uređaj vrača kod 200 OK i prelazi u normalan režim rada u kojemu izvršava kod iz memorije. LED indikator svojom brzinom paljenja i gašenja naznačuje stanje veze s mrežom i MQTT brokerom. Sporo paljenje i gašenje označava spajanje na mrežu dok brzo označava spajanje sa MQTT brokerom nakon čega se LED indikator gasi i uređaj nastavlja izvršavati svoj program.

3.2. Postavljanje poslužitelja

Uloga poslužitelja je pokretanje MQTT brokera i popratnih programa kao što su baza podataka, parser MQTT poruka i API. Poslužitelju se pristupa pomoću ssh protokola i njime se upravlja preko naredbene linije.

SQLite baza podataka je vrlo lagana baza podataka koja koristi SQL sintaksu, ne zahtjeva posebne procese za rad i zapisuje promjene direktno u datoteku na disku. Uloga baze podataka je ta da zapisuje u sebi stanja svih čvorova svih uređaja u sustavu za nadziranje kako bismo pomoću API-a mogli pristupiti tim podacima.

Platforma poslužitelja je raspian distribucija GNU/Linux operacijskog sustava jer je poslužitelj popularno malo računalo Raspberry PI. Baza podata se instalira pomoću naredbe:

```
#sudo apt-get install sqlite
```

koja uz pomoć upravljača paketima *apt* instalira sustav za upravljanje bazama podataka zajedno sa svim programima o kojima taj sustav ovisi.

Pokretanjem sustava za upravljanje bazom podataka se prikazuje naredbeni redak u koji upisom teksta SQL sintakse radimo novu bazu podataka. Struktura baze podataka je navedena u tablicama 3.2 i 3.3. Dvije relacijske tablice su povezane zajedničkim ključem *mac* te se u MQTT tablici čuvaju podaci vezani za sam uređaj dok se u *node* tablici čuvaju podaci vezani za pojedine čvorove.

Ime	Opis
name	Ime uređaja
online	Status uređaja
localip	Lokalna IP adresa uređaja
mac	MAC adresa uređaja
homie	Verzija implementacije Homie konvencije
signal	Jačina bežičnog signala
uptime	Vrijeme rada uređaja
fwname	Ime trenutnog firmware-a
fwvesion	Verzija trenutnog firmware-a
implementation	??
config	JSON konfiguracijska datoteka
ota_enabled	Omogučenost OTA
ota_status	Status OTA

Tablica 3.2: Shema tablice mqtt

Ime	Opis
mac	MAC adresa uređaja
node	Ime čvora
type	Tip čvora
propertyes	Atributi čvora
node_on	??

Tablica 3.3: Shema tablice node

3.3. Parser

Parser je program napisan u programskom jeziku Python te mu je zadaća primati MQTT poruke sa uređaja, parsirati te poruke i sadržaj spremati u bazu podataka.

Za povezivanje na MQTT broker koristi se *phao.mqtt.client* biblioteka. Instalacija te biblioteke se provodi uz pomoć programa *pip* koji služi za instalaciju python paketa naredbom

```
#pip -g install phao-mqtt
```

Opcija -g nalaže to da se taj paket instalira globalno na računalo a ne samo lokalno za taj projekt. Biblioteka se u kod uključuje linijom

```
import phao.mqtt.client as mqtt
```

čime smo postavili alias *mqtt* te možemo koristiti njene funkcije samo koristeći taj alias. Povezivanje sa bazom podataka se vrši pomoću sqlite3 biblioteke koja se također instalira pomoću programa pip. U glavnome djelu programa se povezuje na bazu podataka koja se izrađuje ako ona ne postoji te se provjerava postojanje tablica. Zatim se spaja na MQTT broker i pokreće se beskonačna petlja koja čita poruke.

Listing 3.2: Main funkcija programskog koda

```
def main():
    global database
    database = sqlite3.connect("mqtt.db")
    check_database(database)

    client = mqtt.Client()
    client.on_connect = on_connect
    client.on_message = on_message
```

```
client.connect(host, port, keepalive)

client.loop_forever()
```

Client metode *on_connect* i *on_message* u funkcije implementirane od strane programera i uključuju se pomoću referenciranja. Te funkcije su implementacije ponašanja parsera prilikom dotičnih događanja. Pri uspješnom spajanju na MQTT broker se ispisuje poruka o uspjehu te se klijent pretplačuje na odgovarajuću temu što je u ovome slučaju korijenska tema željenih uređaja.

Listing 3.3: Akcije prilikom spajanja na MQTT broker

```
def on_connect(client, userdata, flags, rc):
    print("Connected!")
    client.subscribe(base_topic)
```

Prilikom primanja poruke se pokreče *on_message* funkcija koja čita poruku, parsira ju te sprema podatke u bazu podataka.

Listing 3.4: Parsiranje poruka i spremanje u bazu podataka

```
def on_message(client, userdata, msg):
    global database
    cursor = database.cursor()
    atribute = msg.topic.split('/')[-1]
    atribute_value = msg.payload.decode('UTF-8')
    messages[atribute] = atribute_value
    if atribute == "\$type":
        node = msg.topic.split('/')[2]
        messages["node"] = node

    if (atribute == "\$online" and
        list(cursor.execute("select␣count(*)␣from␣mqtt;"))[0][0] == 0

        cursor.execute(
                        """
                        insert into mqtt values (
                        '{}',
                        '{}',
                        '{}',
```

```
                    '{}',
                    '{}',
                    '{}',
                    '{}',
                    '{}',
                    '{}',
                    '{}',
                    '{}',
                    '{}',
                    '{}'
                    );
        """.format(
            messages["\$name"],
            messages["\$online"],
            messages["\$localip"],
            messages["\$mac"],
            messages["\$homie"],
            messages["signal"],
            messages["uptime"],
            messages["name"],
            messages["version"],
            messages["\$implementation"],
            messages["config"],
            messages["enabled"],
            messages["status"]
        )
                    )

cursor.execute(
                    """
                    insert into node values (
                    '{}',
                    '{}',
                    '{}',
                    '{}',
                    '{}'
```

```
                    );
            """.format(
                messages["\$mac"],
                messages["node"],
                messages["\$type"],
                messages["\$properties"],
                messages["on"]
            )
                    )

    if(atribute == "uptime" and
        list(cursor.execute("select count(*) from mqtt;"))[0][0] ==
        '\$online' in messages.keys()):

        update_mqtt = """
                        update mqtt
                        set
                        name = '{0}',
                        online = '{1}',
                        localip = '{2}',
                        mac = '{3}',
                        homie = '{4}',
                        signal = '{5}',
                        uptime = '{6}',
                        fwname = '{7}',
                        fwversion = '{8}',
                        implementation = '{9}',
                        config = '{10}',
                        ota_enabled = '{11}',
                        ota_status = '{12}'
                        where mac = '{3}';
                        """.format(messages["\$name"],
                                messages["\$online"],
                                messages["\$localip"],
                                messages["\$mac"],
                                messages["\$homie"],
```

```
                                        messages["signal"],
                                        messages["uptime"],
                                        messages["name"],
                                        messages["version"],
                                        messages["\$implementation"],
                                        messages["config"],
                                        messages["enabled"],
                                        messages["status"]
                                        )
        update_node = """update node
                                set
                                mac = '{0}',
                                node = '{1}',
                                type = '{2}',
                                properties = '{3}',
                                node_on = '{4}'
                                where mac = '{0}';
                                """.format(
                                    messages["\$mac"],
                                    messages["node"],
                                    messages["\$type"],
                                    messages["\$properties"],
                                    messages["on"]
                                )

        cursor.execute(update_mqtt)
        cursor.execute(update_node)

    database.commit()
```

Funkcija za provjeru baze podataka služi tome da pri prvom pokretanju bude provjereno postojanje samih tablica te kreiranje ako one ne postoje. Ako pokušaj čitanja daje iznimku onda se stvaraju ispravne tablice čiji je kod zapisan u varijablama *create_mqtt_sql* te *create_node_sql*.

Listing 3.5: Funkcija za provjeru baze podataka

```
create_mqtt_sql = """
```

```
                        create table mqtt(
                        name text,
                        online text,
                        localip text,
                        mac text primary key,
                        homie text,
                        signal text,
                        uptime text,
                        fwname text,
                        fwversion text,
                        implementation text,
                        config text,
                        ota_enabled text,
                        ota_status text
                        ); """
create_node_sql = """
                        create table node(
                        mac text,
                        node text,
                        type text,
                        properties text,
                        node_on text
                        );"""

def check_database(base):
    try:
        cursor = base.cursor()
        cursor.execute("select␣*␣from␣mqtt;")
    except sqlite3.OperationalError:
        cursor.execute(create_mqtt_sql)
        cursor.execute(create_node_sql)
        base.commit()
```

Svi se podaci periodički objavljuju.

3.4. API

REST API je aplikacijsko sučelje koje putem HTTP protokola primitkom zahtjeva izlaže ili mjenja izložene podatke. REST je akronim za stil arhitekture *Representational State Transfer*. Korištenje ovakve arhitekture se preferira na internetu zbog količine prenesenih podataka za razliku od SOAP (eng. *Simple Object Access Protokol*) arhitekture.

Programiranje REST API-a je olakšano uz pomoć Flask programskog okruženja za programski jezik Python. Nudi već gotova rješenja za razvoj relativno jednostavnih aplikacija te se uključuje u program instalacijom pomoću alata *pip* te uključivanjem u programski kod.

U Flask okruženju se resursi prikazuju kao klase kojima se pridjeljuje kraj putanje URL-a. To se postiže tako da se doda resurs navodeći odgovarajuću klasu te definicijom krajnje točke:

```
api.add_resource(Devices, "/devices")
```

Unutar klase se navode metode imena *get, put, post* i *delete* koje odgovaraju HTTP zahtjevima koji mogu biti poslani. Svaka metoda implementira odgovarajuče akcije za tu klasu.

Aplikacija se pokreće u *main* programskom odsječku pozivom *run()* metode.

```
app.run(port=5000)
```

Kompletan programski kod izgleda ovako:

Listing 3.6: REST API programski kod

```
from flask import Flask
from flask_restful import Resource, Api
from sqlalchemy import create_engine
from flask_jsonpify import jsonify

db = create_engine("sqlite:///mqtt.db")
app = Flask(__name__)
api = Api(app)


class Devices(Resource):
    def get(self):
```

```
        connection = db.connect()
        query = connection.execute("""
                                   select * from mqtt, node
                                   where mqtt.mac = node.mac;
                                   """)
        result = [dict(zip(tuple(query.keys()), i)) for i in query.cu
        return jsonify(result)


api.add_resource(Devices, "/devices")

if __name__ == "__main__":
    app.run(port=5000)
```

Get metoda implementira spajanje na bazu podataka, izvršavanje SQL upita te vraća JSON prikaz cijele baze podataka.

4. Zaključak

Zaključak.

Implementacija sustava za nadziranje i upravljanje bežičnim razvojnim modulima ESP8266

Sažetak

Sažetak na hrvatskom jeziku.

Ključne riječi: Ključne riječi, odvojene zarezima.

Management Framework for ESP8266 WiFi Development Modules

Abstract

Abstract.

Keywords: Keywords.