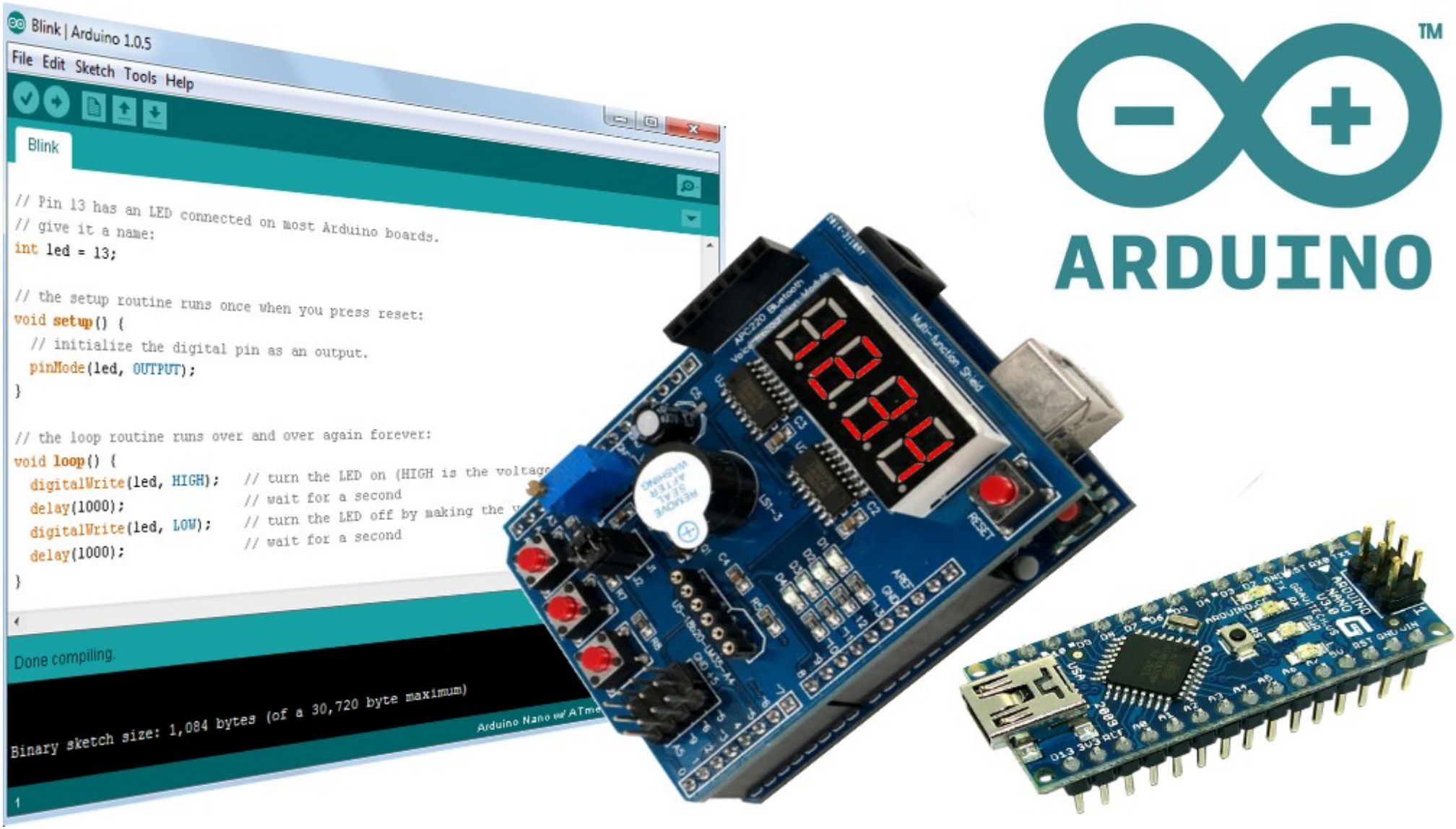
Arduino tanfolyam kezdőknek és haladóknak



15. ESP8266 alapú MQTT szerver

Felhasznált és ajánlott irodalom

- Leírások, dokumentáció:
  - HiveMQ : MQTT Essentials
  - OASIS: MQTT Version 3.1.1 specification
  - ESP8266 Community: ESP8266 Arduino Core's documentation
- Arduino programkönyvtárak:
  - Martin Ger: uMQTTBroker library for ESP8266 Arduino
  - Nick O'Leary: PubSubClient - Arduino Client for MQTT
  - Benoit Blanchon: Mastering ArduinoJson 6
- PC és mobil segédprogramok:
  - Thomas Nordquist: MQTT Explorer
  - Wireshark community: Wireshark protocol analyzer
  - Routix software: MQTT Dash

A csatlakozások titkos adatai

- Az **ESP8266** kliensként történő csatlakozásokhoz a személyes adatokat kiszerveztük egy **secrets.h** nevű fejléc állományba, amelyet a **Vázlatfüzet** (Sketchbook) mappa **libraries/secrets** almappájában helyeztünk el

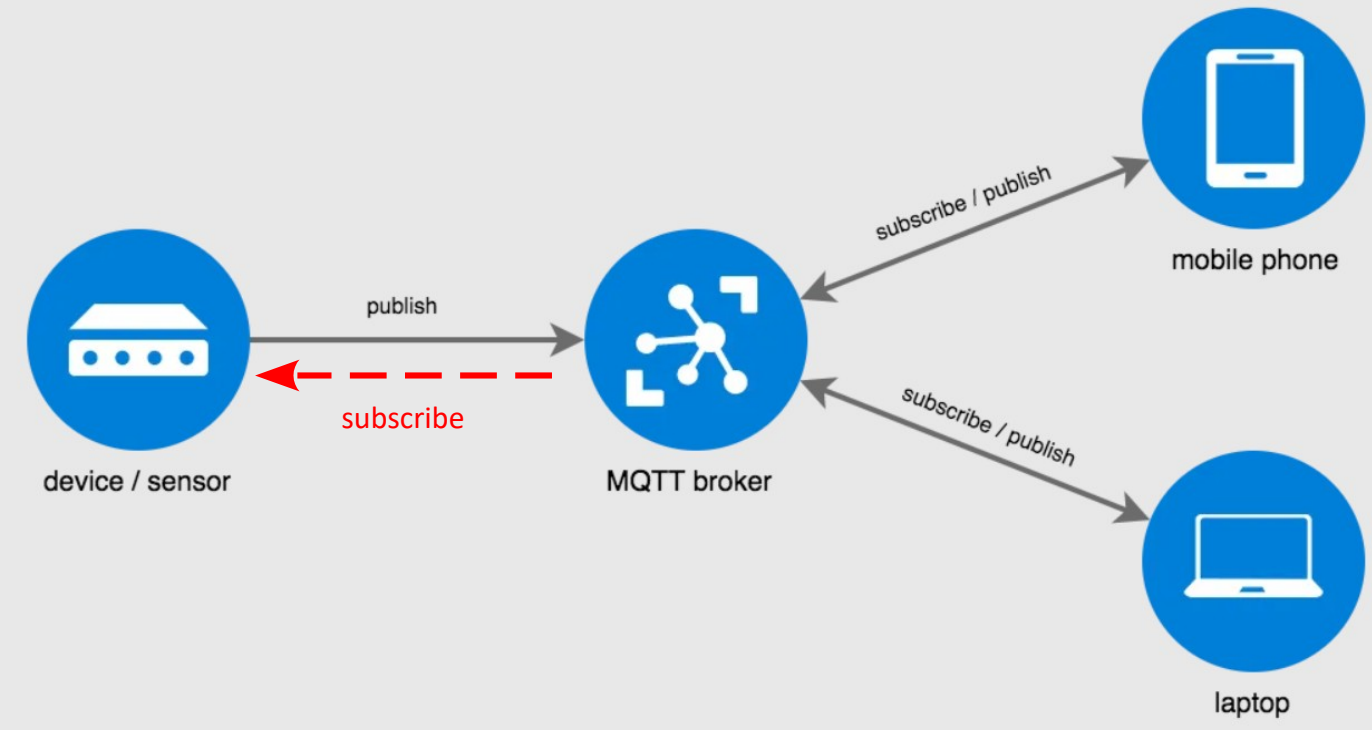
```
#include <ESP8266WiFi.h>
#include "secrets.h"

void setup_wifi() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  Serial.print("Connecting to ");
  Serial.print(WIFI_SSID);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.print("Connected! IP address: ");
  Serial.println(WiFi.localIP());
}
```

```
#define WIFI_SSID  "MY_SSID"
#define WIFI_PASS  "MY_PASSWORD"

String OPENWEATHERMAP_APPID =
"**************************";

String THINGSPEAK_WRITE_APIKEY =
"xxxxxxxxxxxxx";

#define FLESPI_TOKEN
"Ws44Ev13*********rrQkOlbQGElNuMoDnPNk1g"

#define HIVEMQ_USER "********"
#define HIVEMQ_PASS "***********"

#define uMQTT_USER  "hobbi"
#define uMQTT_PASS  "megtestesules"
```

MQTT alapok

- A **publish/subscribe** séma elkülöníti az adatküldő klienst (**publisher**) az üzeneteket fogadó kliens(ek)től (**subscriber**).
- A kapcsolatot egy harmadik komponens, az ún. bróker biztosítja
- Az **MQTT** kommunikáció **TCP/IP** alapú
- A **bróker** többnyire külső szolgáltató (pl. **HiveMQ**), vagy helyi, nagyobb teljesítményű eszköz (**PC**, esetleg **Raspberry Pi**)

- Most egy **ESP8266 NodeMCU** kártya felhasználásával készítünk brókert, (*olcsó és nem ráz...*), nyilván szerényebb igényeket célozva...



Az uMQTTBroker programkönyvtár

- **Martin Ger** uMQTTBroker Arduino programkönyvtára lehetővé teszi, hogy egy **ESP8266** felhasználásával alakítsunk ki egy szerény képességű **MQTT brókert** a helyi hálózaton
- A bróker az alábbiakat támogatja:
  - MQTT v3.1 és v3.1.1 protokoll
  - több (~ 8) kliens egyidejű kapcsolódása
  - Megőrzött üzenetek
  - QoS level 0
  - username/password autentikáció
- A bróker az alábbiakat jelenleg nem támogatja:
  - QoS 1 és 2 szint
  - sok TCP(=MQTT) kliens
  - non-clear sessions
  - TLS

uMQTTBroker C++ style API

```
class uMQTTBroker {
public: uMQTTBroker(uint16_t portno=1883, uint16_t max_subscriptions=30,
                    uint16_t max_retained_topics=30);
    void init();

//--- Callbacks on client actions -------------
    virtual bool onConnect(IPAddress addr, uint16_t client_count);
    virtual void onDisconnect(IPAddress addr, String client_id);
    virtual bool onAuth(String username, String password, String client_id);
    virtual void onData(String topic, const char *data, uint32_t length);

//--- Infos on currently connected clients ----
    virtual uint16_t getClientCount();
    virtual bool getClientId(uint16_t index, String &client_id);
    virtual bool getClientAddr(uint16_t index, IPAddress& addr);

//--- Interaction with the local broker -------
    virtual bool publish(String topic, uint8_t* data, uint16_t data_length,
                         uint8_t qos=0, uint8_t retain=0);
    virtual bool publish(String topic, String data, uint8_t qos=0, uint8_t retain=0);
    virtual bool subscribe(String topic, uint8_t qos=0);
    virtual bool unsubscribe(String topic);

//--- Cleanup all clients on Wifi connection loss ---
    void cleanupClientConnections();
};
```

Jelszó ellenőrzése

Adat beérkezése

ESP8266_uMQTT_Broker.ino

- Az első kísérlethez a uMQTTBrokerSampleOOFull.ino mintapéldából indulunk ki, amelyet kicsit átalakítottunk:
- A WiFi (STA) belépő kódot kiszerveztük a ***secrets.h*** állományba
- Az ***onAuth()*** visszahívási függvényben megfogalmaztuk a belépés érvényességének feltételét (sikeres belépéskor a függvénynek ***true*** értékkel kell visszatérnie), a belépés ellenőrzésére használt `uMQTT_USER` és `uMQTT_PASS` szintén a a ***secrets.h*** állományban vannak definiálva
- Kísérleteinkben a routeren keresztül a helyi hálózatra kapcsolódtunk, ezért a STATION módot választottuk
- Az első próbák után a ***loop()*** függvényben célszerű kommentbe tenni a számláló publikálását
- Csak a kipróbálásnál célszerű használni a ***printClients()*** függvényt

ESP8266_uMQTT_Broker.ino

```
#include <ESP8266WiFi.h>
#include "uMQTTBroker.h"
#include "secrets.h"
char ssid[] = WIFI_SSID;       //****** your network SSID (name)
char pass[] = WIFI_PASS;       //****** your network password
bool WiFiAP = false;           // Do yo want the ESP as AP?

void startWiFiClient() {
  Serial.println("Connecting to "+(String)ssid);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: " + WiFi.localIP().toString());
}

void startWiFiAP() {
  WiFi.mode(WIFI_AP);
  WiFi.softAP(ssid, pass);
  Serial.println("AP started");
  Serial.println("IP address: " + WiFi.softAPIP().toString());
}
```

Kívánság szerint WiFi kliens (STA), vagy WiFi AP mód Inicializálása

ESP8266_uMQTT_Broker.ino

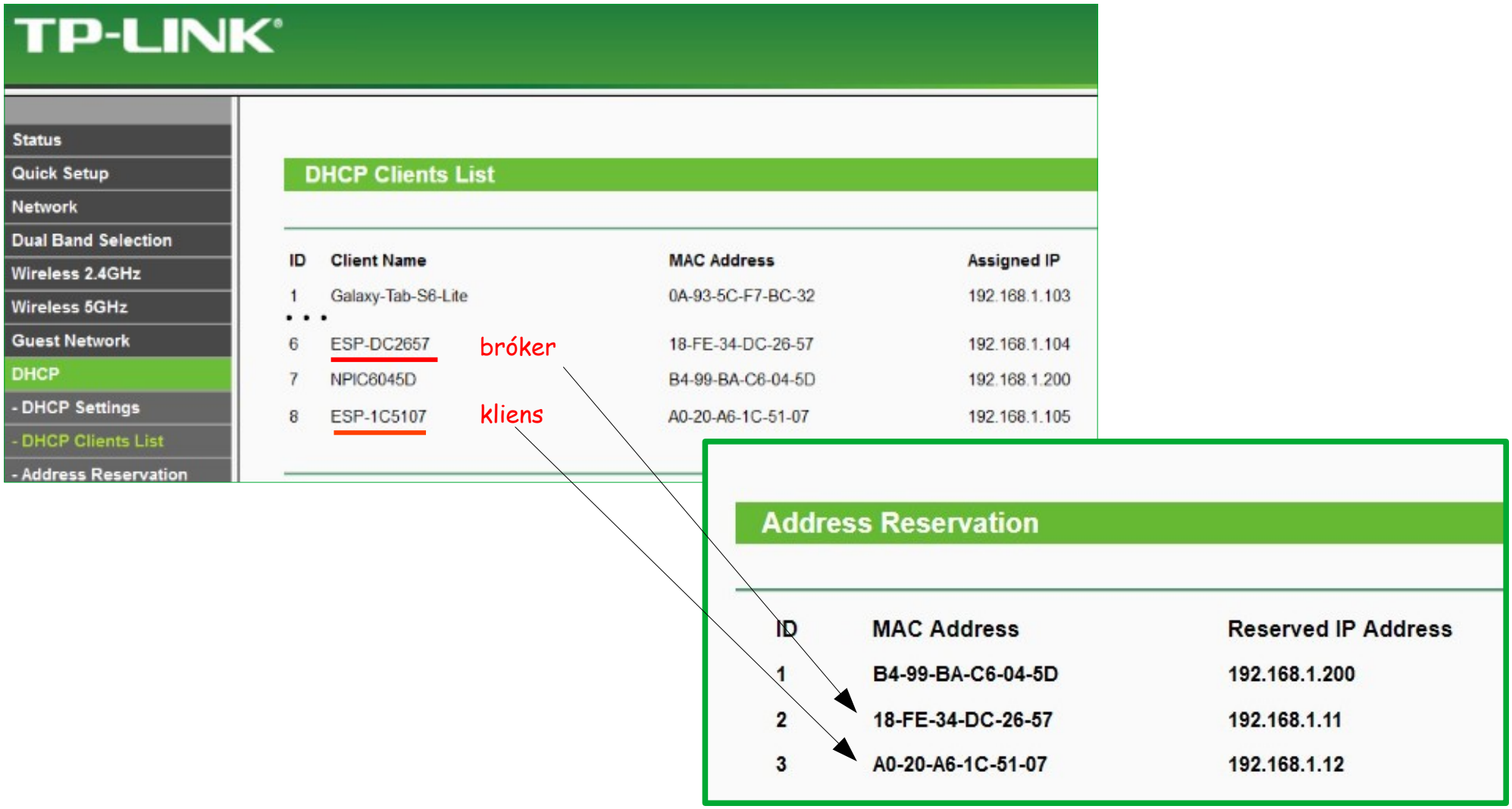
```
class myMQTTBroker: public uMQTTBroker {
public:
  virtual bool onConnect(IPAddress addr, uint16_t client_count) {
    Serial.println(addr.toString()+" connected");
    return true;
  }

  virtual void onDisconnect(IPAddress addr, String client_id) {
    Serial.println(addr.toString()+" ("+client_id+") disconnected");
  }

  virtual bool onAuth(String username, String password, String client_id) {
    Serial.println("Username/Password/ClientId: "+username+"/"+password+"/"+client_id);
    return (username.equals(String(uMQTT_USER)) && password.equals(String(uMQTT_PASS)));
  }

  virtual void onData(String topic, const char *data, uint32_t length) {
    char data_str[length+1];
    os_memcpy(data_str, data, length);
    data_str[length] = '\0';
    Serial.println("received topic '"+topic+"' with data '"+(String)data_str+"'");
    //  printClients();
  }

 // Sample for the usage of the client info methods
 virtual void printClients() { . . . }
};
```

ESP8266_uMQTT_Broker.ino

```
myMQTTBroker myBroker;

void setup() {
  Serial.begin(115200);
  Serial.println();

  if (WiFiAP) startWiFiAP();                    // Start WiFi
  else startWiFiClient();
  Serial.println("Starting MQTT broker");      // Start the broker

  myBroker.init();
  myBroker.subscribe("#");                      // Subscribe to everything
}

int counter = 0;

void loop() {
  //--- Publish the counter value as String
  myBroker.publish("broker/counter", (String)counter++);
  delay(1000);                                  // wait a second
}
```

A bróker megérdemel egy fix IP címet!

- A bróker könnyű eléréséhez rendeljük fix IP címeket a helyi routeren a NodeMCU modulokhoz!

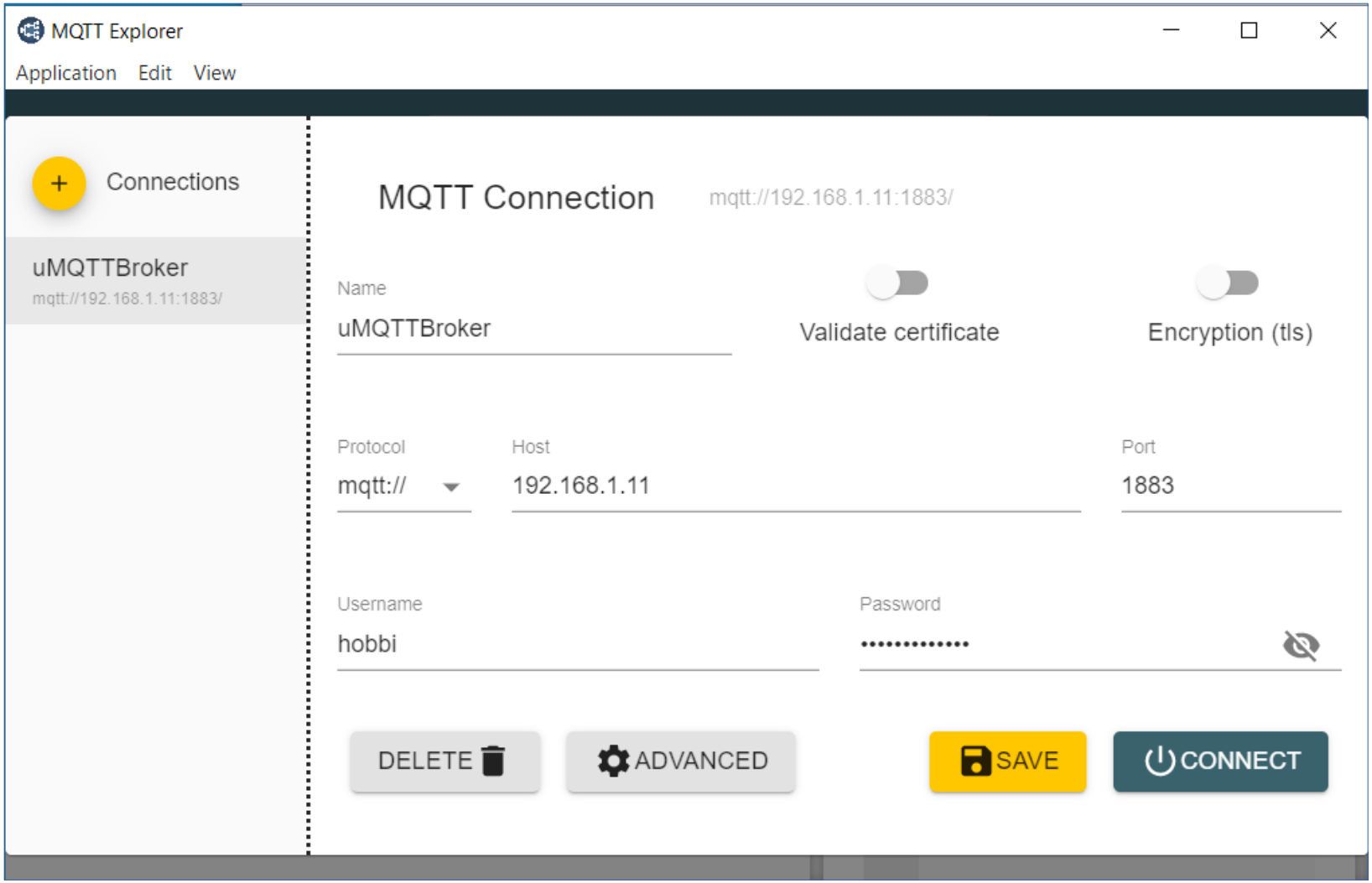


Az MQTT üzenetek közelnézetből

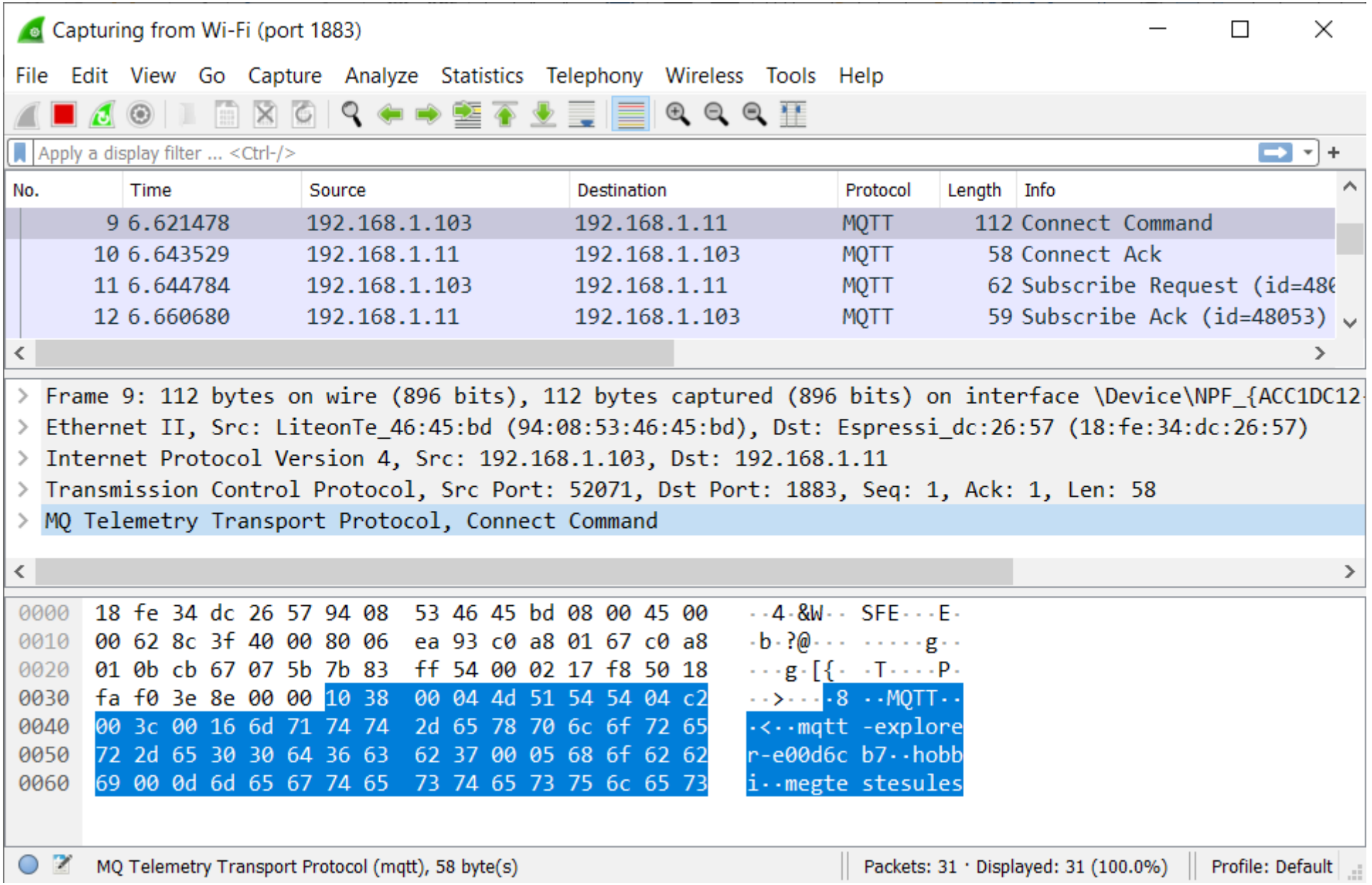
- Az **ESP8266_uMQTTBroker** működését az **MQTT Explorer** nevű számítógépes klienssel próbáljuk ki, s az üzenetcsomagokat a **Wireshark** program segítségével vizsgáljuk meg közelebbről

MQTT Explorer (Windows alkalmazás)

- Kapcsolódjunk egy klienssel a helyi brókerhez!
- Az **Advanced** gombra kattintva iratkozzunk fel minden topikra (#)!

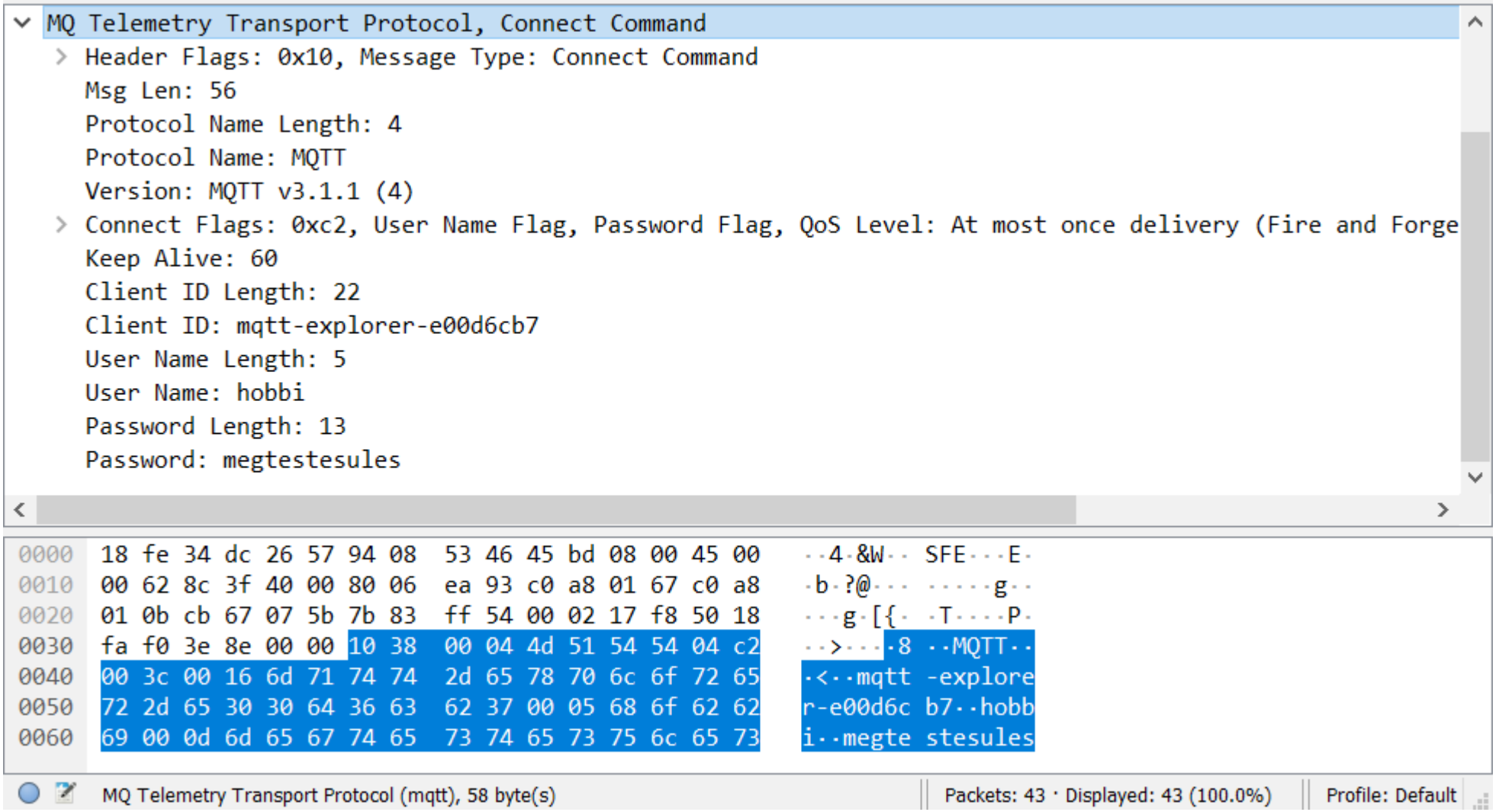


A Connect esemény



A Connect esemény

- A Wireshark hajlandó számunkra dekódolni az üzenetet, így bájtokra lebontva és szövegesen is láthatjuk az üzenetet



Subscribe kérés

- Az alábbiakban a beérkezett Subscribe Request látható – melyben feliratkozunk mindenre IS…

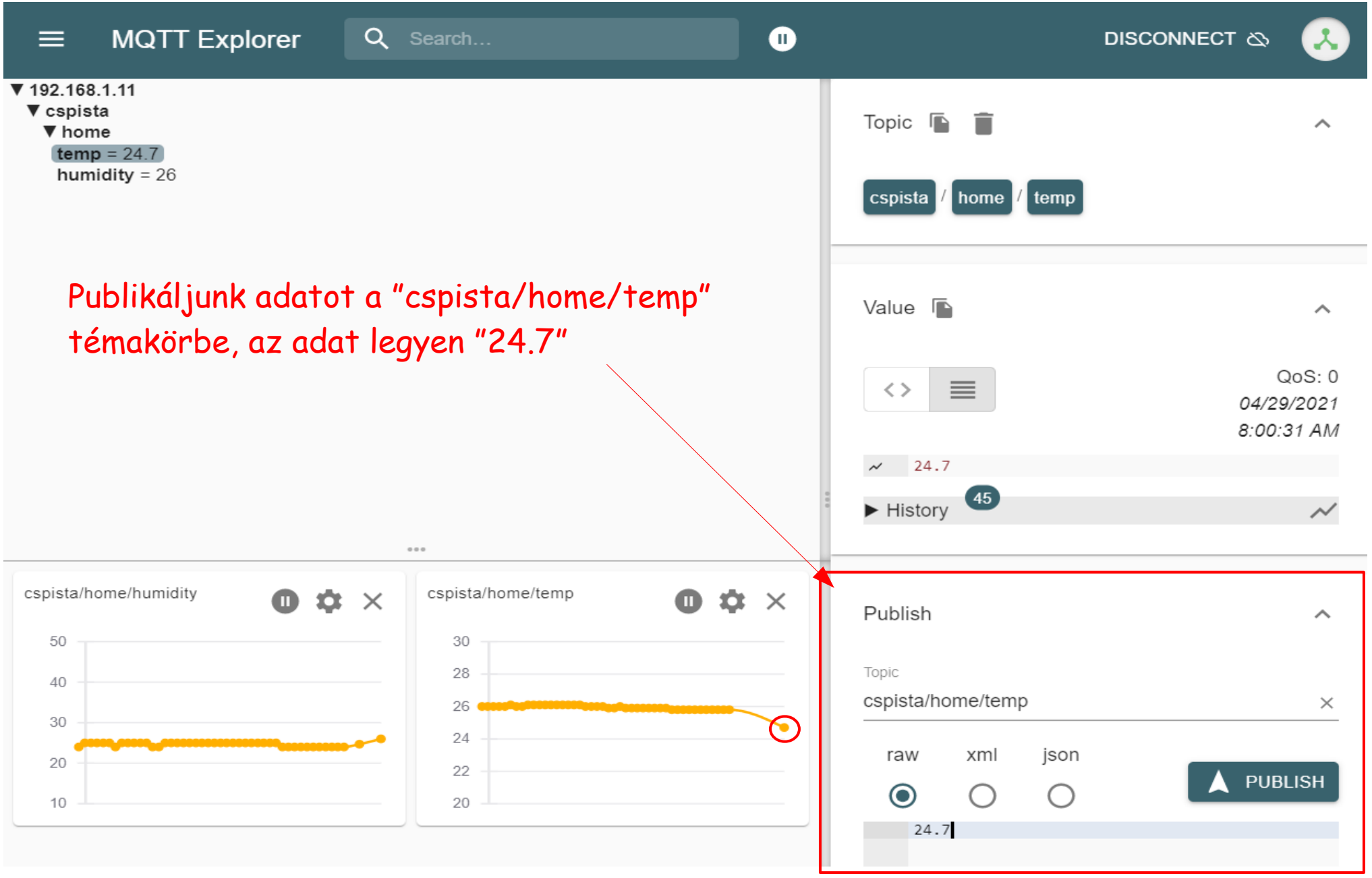
No.	Time	Source	Destination	Protocol	Length	Info
9	6.621478	192.168.1.103	192.168.1.11	MQTT	112	Connect Command
10	6.643529	192.168.1.11	192.168.1.103	MQTT	58	Connect Ack
11	6.644784	192.168.1.103	192.168.1.11	MQTT	62	Subscribe Request (id=48053) [#]
12	6.660680	192.168.1.11	192.168.1.103	MQTT	59	Subscribe Ack (id=48053)

```
> Frame 11: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface \Device\NPF_{ACC1DC12-A348-44E4-A
> Ethernet II, Src: LiteonTe_46:45:bd (94:08:53:46:45:bd), Dst: Espressi_dc:26:57 (18:fe:34:dc:26:57)
> Internet Protocol Version 4, Src: 192.168.1.103, Dst: 192.168.1.11
> Transmission Control Protocol, Src Port: 52071, Dst Port: 1883, Seq: 59, Ack: 5, Len: 8
v MQ Telemetry Transport Protocol, Subscribe Request
  > Header Flags: 0x82, Message Type: Subscribe Request
    Msg Len: 6
    Message Identifier: 48053
    Topic Length: 1
    Topic: #
    Requested QoS: At most once delivery (Fire and Forget) (0)

MQ Telemetry Transport Protocol, Subscribe Ack
> Header Flags: 0x90, Message Type: Subscribe Ack
  Msg Len: 3
  Message Identifier: 48053
  Granted QoS: At most once delivery (Fire and Forget) (0)
                90 03  bb b5 00
```

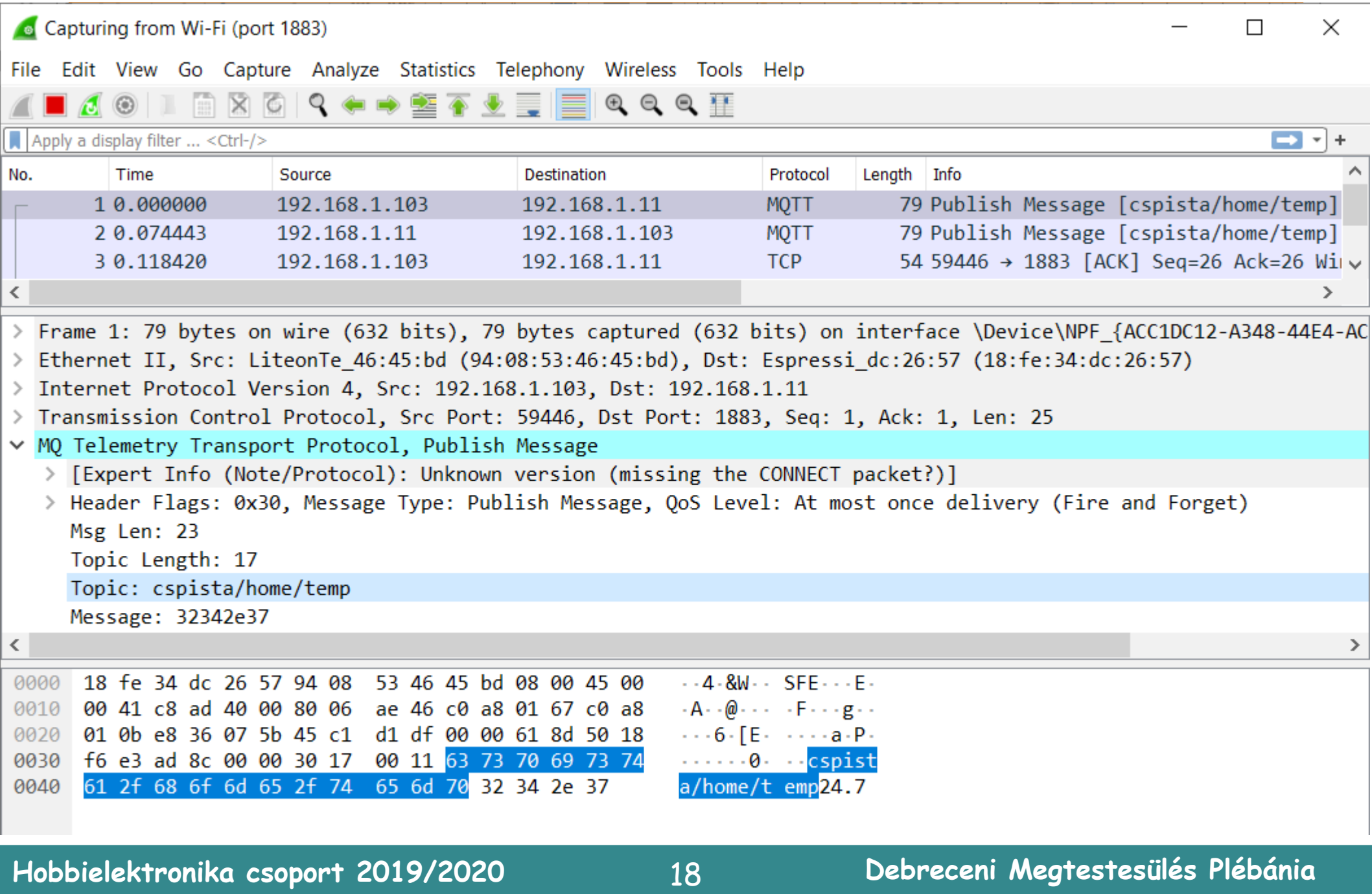
```
0000  18 fe 34 dc 26 57 94 08  53 46 45 bd 08 00 45 00   ··4·&W·· SFE···E·
0010  00 30 8c 40 40 00 80 06  ea c4 c0 a8 01 67 c0 a8   ·0·@@··· ·····g··
0020  01 0b cb 67 07 5b 7b 83  ff 8e 00 02 17 fc 50 18   ···g·[{· ······P·
0030  fa ec 6a 84 00 00 82 06  bb b5 00 01 23 00        ··j····· ····#·
```

A nyugtázás

Adatküldés (Publish esemény)



Publish esemény



A publish üzenet formátuma (QoS=0)

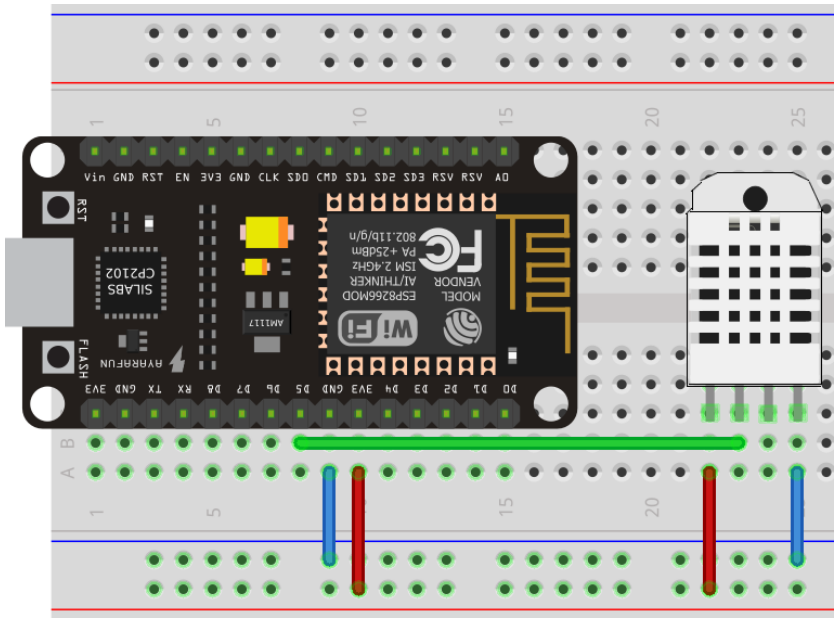
Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (3)				DUP flag	QoS level		RETAIN
	0	0	1	1	X	X	X	X
byte 2	Remaining Length							

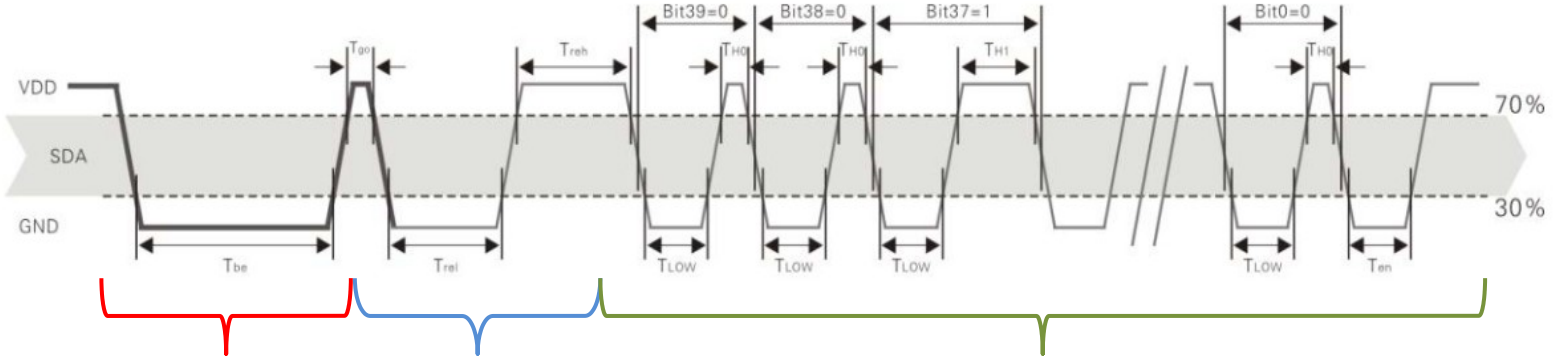
	Description	7	6	5	4	3	2	1	0
Topic Name									
byte 1	Length MSB (0)	0	0	0	0	0	0	0	0
byte 2	Length LSB (3)	0	0	0	0	0	0	1	1
byte 3	‘a’ (0x61)	0	1	1	0	0	0	0	1
byte 4	‘/’ (0x2F)	0	0	1	0	1	1	1	1
byte 5	‘b’ (0x62)	0	1	1	0	0	0	1	0

The Payload contains the Application Message that is being published. The content and format of the data is application specific. The length of the payload can be calculated by subtracting the length of the variable header from the Remaining Length field that is in the Fixed Header. It is valid for a PUBLISH Packet to contain a zero length payload.

ESP8266_uMQTT_DHT22.ino

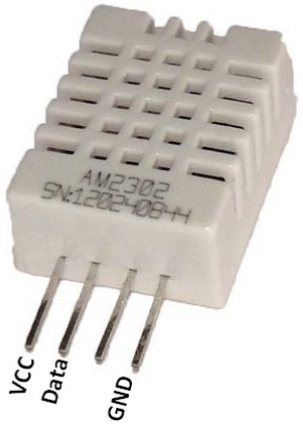
- Az előző előadásban bemutatott **ESP8266_HiveMQ_DHT22** kliens alkalmazást módosítsuk úgy, hogy a helyi brókerhez csatlakozzon!
- **Emlékeztető: DHT22** szenzorral hőmérsékletet és rel. páratartalmat mérünk
- Programkönyvtár: Adafruit_DHT
- Adatlap: sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf





Host indítójel | Szenzor nyugtázó jel | 40 bitnyi adat: 32 bit információ + 8 bit ellenőrző összeg

Összesen tehát 85 időzítést tartalmaz egy-egy tranzakció …



Emlékeztető: PubSubClient programkönyvtár

- **PubSubClient** – kliens programkönyvtár **MQTT** kommunikációra
- **PubSubClient** (*server, port, [callback], **client,** [stream]*) – constructor
  server – pl. mqtt.flespi.io vagy más bróker, *port* – általában 1883
  client – egy WiFiClient objektum
  Megjegyzés: *server* és *port* a **setServer()** függvénnyel is megadhatók!
- **connect** (*clientID, [username, password], [willTopic, willQoS, willRetain, willMessage], [cleanSession]*) – kapcsolódása bróker szerverhez
  clientID - egy egyedi azonosító
- **setCallback** (*callback*) – visszahívási függvény regisztrálása
```
void callback(const char[] topic, byte* payload, unsigned int length);
```
- **publish**(*topic, payload, [length], [retained]*) – adatküldés a brókernek
  topic – az adott témakör, *payload* – a témakörben küldött új adat
- **subscribe** (topic, [qos]) – feliratkozás a megadott témakörre
  topic – az adott témakör, *qos* – csak 0 vagy 1 lehet (Qos2 nem támogatott)
- **loop()** – rendszeresen meg kell hívni a beérkező üzenetek fogadásához

PubSubClient API dokumentáció: https://pubsubclient.knolleary.net/api

ESP8266_uMQTT_DHT22.ino

- **A mért adatok publikálásához** két témakör használunk:
  - `cspista/home/temp`
  - `cspista/home/humidity`
- Az adatoknak csak a mérőszámát írjuk ki, hogy numerikus adatként lehessen kezelni (egyébként ℃ és % lenne a „mértékegység”)
- Kapcsolódáskor vagy újrakapcsolódáskor a `cspista/home` témakörbe küldünk egy `"DHT22 connected"` üzenetet
- **A feliratkozásra** (és a **GPIO2**-re kötött LED vezérlésére) használt témakör neve:
  - `cspista/home/led`

ESP8266_uMQTT_DHT22.ino

Lecseréljük erre: 192.168.1.11

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include "DHT.h"
#include "secrets.h"
const char* mqtt_server = "broker.hivemq.com";
#define DHTPIN          14          // GPIO14 (D5) pin for DHT sensor
DHT dht(DHTPIN, DHT22);
WiFiClient espClient;
PubSubClient client(espClient);
unsigned long lastMsg = 0;
char msg[50];
int value = 0;
int nClient = 0;

void setup() {
  pinMode(BUILTIN_LED, OUTPUT);     // Initialize the BUILTIN_LED pin as an output
  dht.begin();
  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
}

void callback(char* topic, byte* payload, unsigned int length) {
  if ((char)payload[0] == '1') {
    digitalWrite(BUILTIN_LED, LOW);   // Turn the LED on
  } else digitalWrite(BUILTIN_LED, HIGH);  // Turn the LED off
}
```

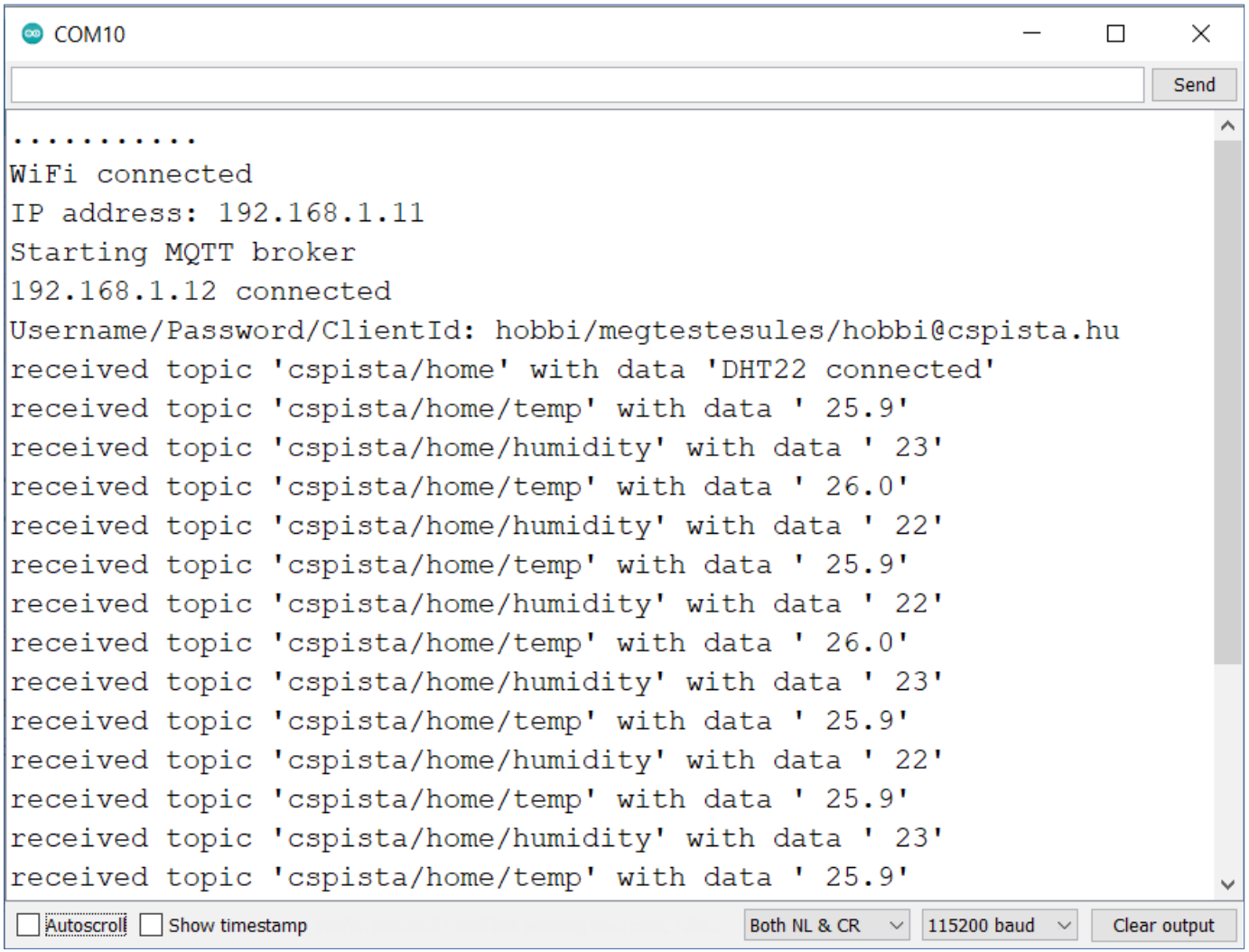
```
void setup_wifi() {
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }
}
```

ESP8266_uMQTT_DHT22.ino

```
void reconnect() {
  while (!client.connected()) {                    // Loop until we're reconnected
    char clientId[] = "hobbi@cspista.hu";
    if (client.connect(clientId, uMQTT_USER, uMQTT_PASS )) { // Attempt to connect
      client.publish("cspista/home", "DHT22 connected"); // Once connected, publish...
      client.subscribe("cspista/home/led");          // ... and resubscribe
    } else delay(5000);                              // Wait 5 seconds before retrying
  }
}
void loop() {
  if (!client.connected()) reconnect();
  client.loop();
  unsigned long now = millis();
  if (now - lastMsg > 5000) {
    lastMsg = now;
    float t = dht.readTemperature();
    if (!isnan(t)) {
      sprintf(msg, "%5.1f", t);
      client.publish("cspista/home/temp", msg);
    }
    float h = dht.readHumidity();
    if (!isnan(h)) {
      sprintf(msg, "%3.0f", h);
      client.publish("cspista/home/humidity", msg);
    }
  }
}
```

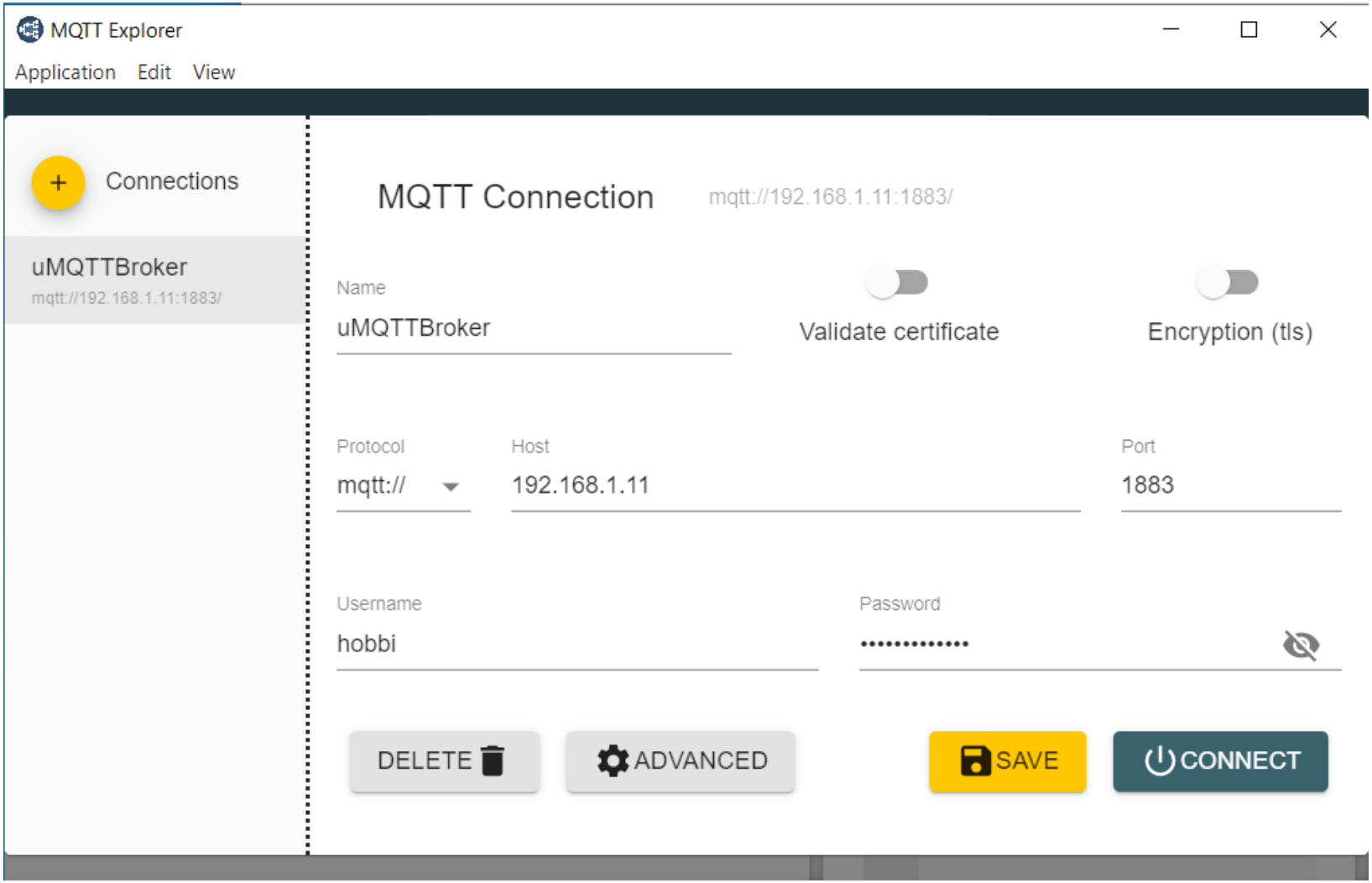
ESP8266_uMQTT_Broker.ino

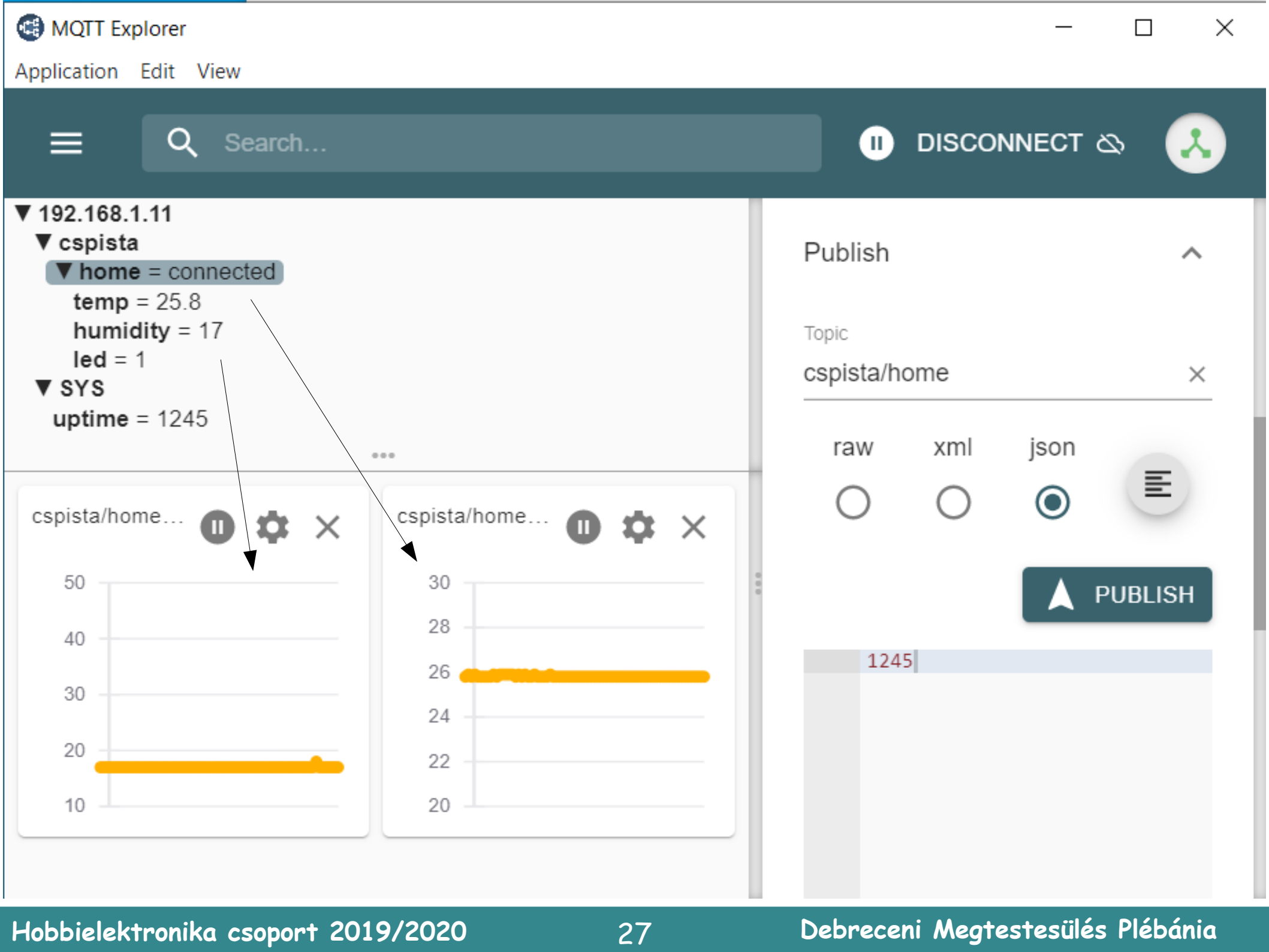
- A bróker és a kliens együttes futtatásakor ezt látjuk a bróker terminál kimenetén



MQTT Explorer (Windows alkalmazás)

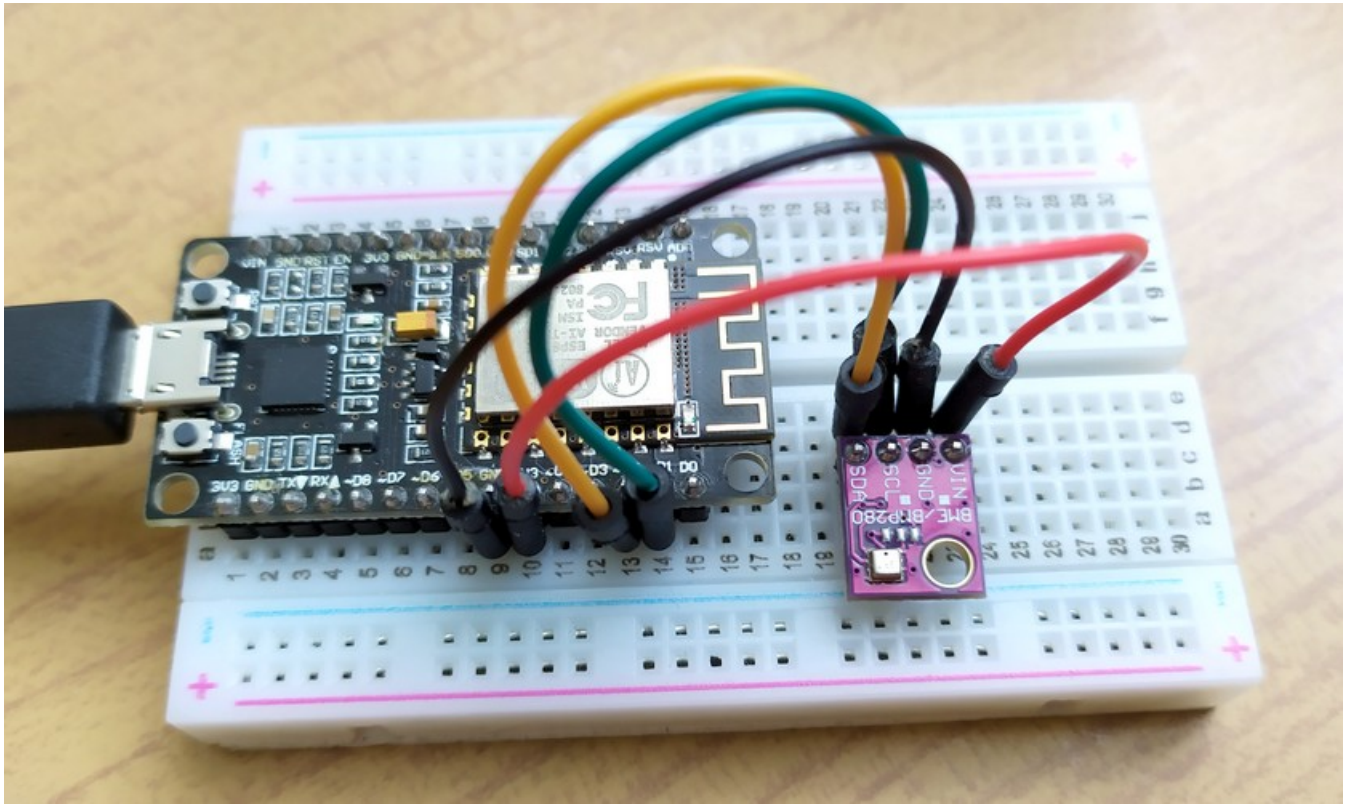
- Kapcsolódjunk egy újabb klienssel a helyi brókerhez!
- Az **Advanced** gombra kattintva iratkozzunk fel minden topikra (#)!




MQTT Explorer
Application Edit View
Search...
DISCONNECT
192.168.1.11
cspista
home = connected
temp = 25.8
humidity = 17
led = 1
SYS
uptime = 1245
Publish
Topic
cspista/home
raw
xml
json
PUBLISH
1245
cspista/home...
cspista/home...
50
40
30
20
10
30
28
26
24
22
20

Adatküldés JSON formátumban

- Kibővítjük a rendszert egy **BME280** szenzorral (hőmérséklet, nyomás és relatív páratartalom mérése), s a mérési adatokat egyetlen topikban, JSON formátumba csomagolva publikáljuk
- Technikai okokból nem új klienst vetünk be, hanem a bróker szoftverét bővítjük ki **mérés és publish** funkciókkal
- Bekötés:
  - SDA – GPIO4 (D2)
  - SCL – GPIO5 (D1)
  - GND – GND
  - VIN – 3V3



ESP8266_uMQTT_Broker_BMP280.ino

```
#include <ESP8266WiFi.h>
#include "uMQTTBroker.h"
#include "secrets.h"

#include <ArduinoJson.h>
StaticJsonDocument<200> doc;

#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#define SEALEVELPRESSURE_HPA (1013.25)
Adafruit_BME280 bme;

bool WiFiAP = false;                        // Do yo want the ESP as AP?
myMQTTBroker myBroker;                      // myMQTTBroker definiója a 9. oldalon!

void setup() {
  Serial.begin(115200);
  Serial.println();
  bme.begin(0x76);
  if (WiFiAP) startWiFiAP();                // Start WiFi
  else startWiFiClient();
  Serial.println("Starting MQTT broker");   // Start the broker
  myBroker.init();
  myBroker.subscribe("#");                  // Subscribe to everything
}
```

ESP8266_uMQTT_Broker.ino

```
void loop() {
  //---  Publish the counter value as String ------
  // myBroker.publish("broker / counter", (String)counter++);
  //--- wait a second
  delay(1000);
  //--- BME280 sensor data ---------
  doc["temperature"] = String(bme.readTemperature(),1);
  doc["humidity"] = (int)bme.readHumidity();
  doc["pressure"] = String(((bme.readPressure() + 1440) / 100.0F),1);
  String payload = doc.as<String>();              // Serialization
  myBroker.publish("cspista/home/bme280", payload);
  delay(4000);
}
```

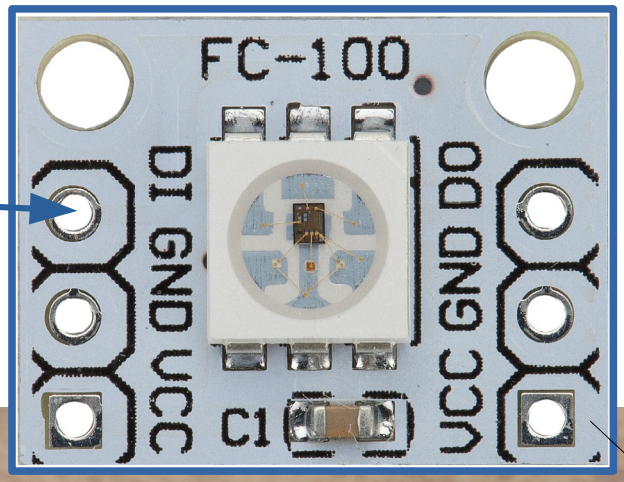
- Arra ügyeljünk, hogy a **serializeJson()** adatfolyamként kezeli a ***String*** objektumokat, azaz hozzáfűz... Például:

```
String output = "JSON = ";
serializeJson(doc, output);
```

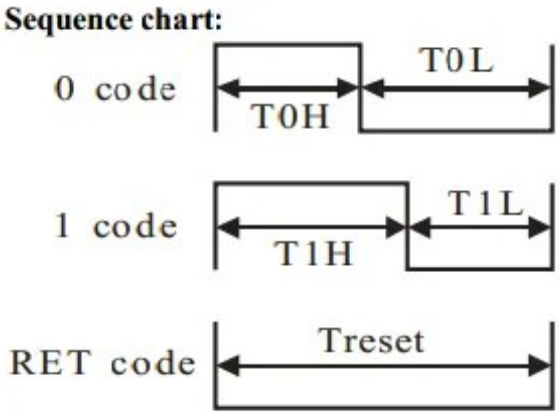
- Esetünkben most egyszerűbbnek tűnt a típuskonverzió, de nem ez az egyetlen út

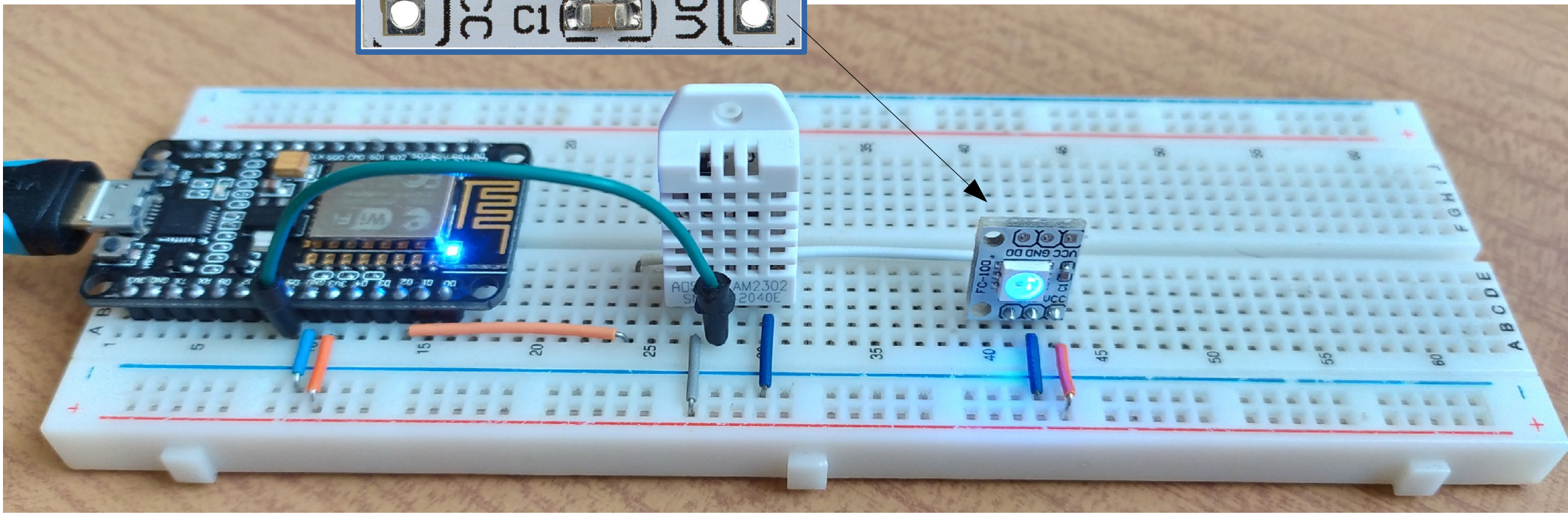
Bővítsük a kliensünket is: Neopixel

- A DHT22-vel ellátott MQTT klienst bővítsük egy **WS2812** (Neopixel) digitálisan vezérelhető RGB LED-del!



GPIO4 (D2)





ESP8266_uMQTT_neopixel.ino

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <Adafruit_NeoPixel.h>
#include "DHT.h"
#include "secrets.h"
const char* mqtt_server = "192.168.1.11";
#define DHTPIN          14     // GPIO14 (D5) pin for DHT sensor
#define NEOPIXEL_PIN     4     // GPIO4 (D2) pin for Neopixel
#define NUMPIXELS        1     // NeoPixel string size (csak 1 db-ot használunk!)
Adafruit_NeoPixel pixels(NUMPIXELS, NEOPIXEL_PIN, NEO_GRB + NEO_KHZ800);

DHT dht(DHTPIN, DHT22);
WiFiClient espClient;
PubSubClient client(espClient);

unsigned long lastMsg = 0;
#define MSG_BUFFER_SIZE  (50)
char msg[MSG_BUFFER_SIZE];
int value = 0;
int nClient = 0;

String led_topic = "cspista/home/led";        // Így könnyebb lesz összehasonlítani!
String rgb_topic = "cspista/home/rgb";
```

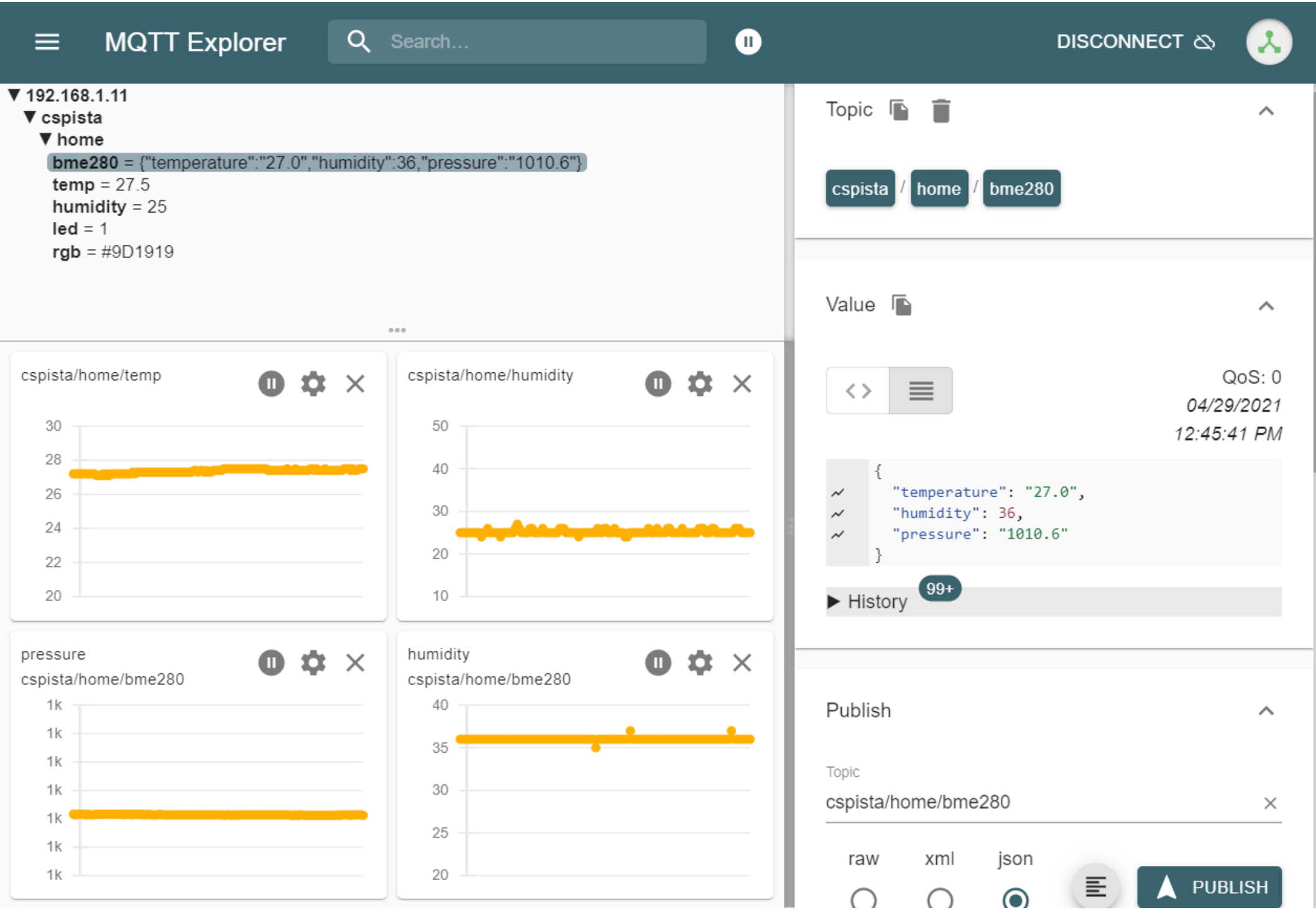
ESP8266_uMQTT_neopixel.ino

```
void setup() {
  pinMode(BUILTIN_LED, OUTPUT);     // Initialize the BUILTIN_LED pin as an output
  Serial.begin(115200);
  pixels.begin();                   // INITIALIZE NeoPixel strip object
  pixels.clear();                   // Set all pixel colors to 'off'
  pixels.setPixelColor(0, pixels.Color(0, 150, 0));
  pixels.show();                    // Send the updated pixel colors to the hardware.
  dht.begin();                      // Initialize the DHT22 sensor
  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
}

void reconnect() {
  while (!client.connected()) {                     // Loop until we're reconnected
    char clientId[] = "hobbi@cspista.hu";
    if (client.connect(clientId, uMQTT_USER, uMQTT_PASS )) { // Attempt to connect
      client.publish("cspista/home", "DHT22 connected"); // Once connected, publish...
      client.subscribe(led_topic.c_str());          // ... and resubscribe
      client.subscribe(rgb_topic.c_str());
    } else delay(5000);                             // Wait 5 seconds before retrying
  }
}
```

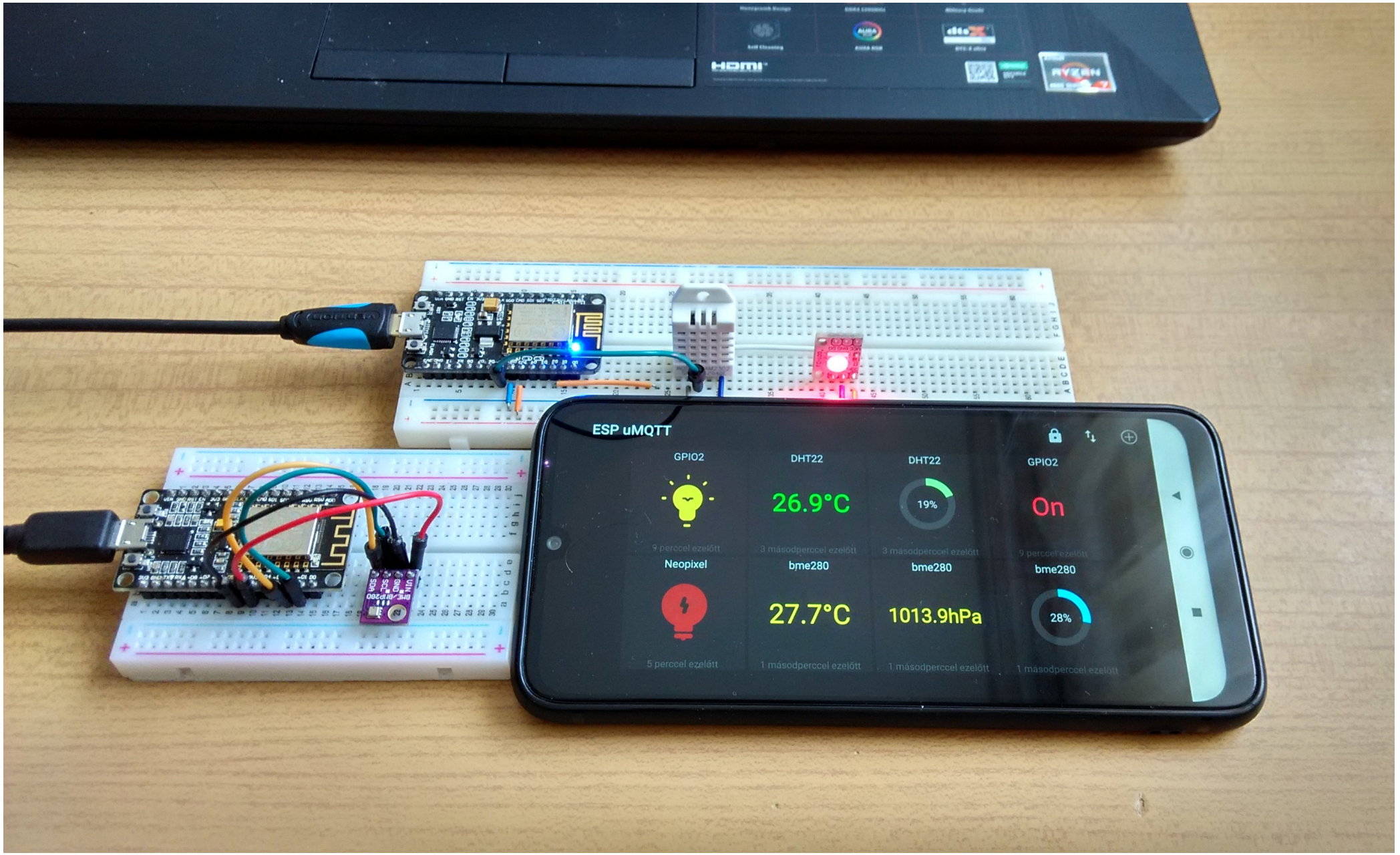
ESP8266_uMQTT_neopixel.ino

```
void callback(char* topic, byte* payload, unsigned int length) {
  if (led_topic.equals(String(topic))) {
    // Switch on the LED if an 1 was received as first character
    if ((char)payload[0] == '1') {
      digitalWrite(BUILTIN_LED, LOW);   // Turn the LED on

    } else {
      digitalWrite(BUILTIN_LED, HIGH);  // Turn the LED off
    }
  }
  else if (rgb_topic.equals(String(topic))) {
    long color = strtol((char*)(payload+1), NULL, 16);
    pixels.setPixelColor(0, color);
    pixels.show();
  }
}


// Az itt nem részletezett loop() és a setup_wifi() függvények ugyanazok,
// mint korábban...
```

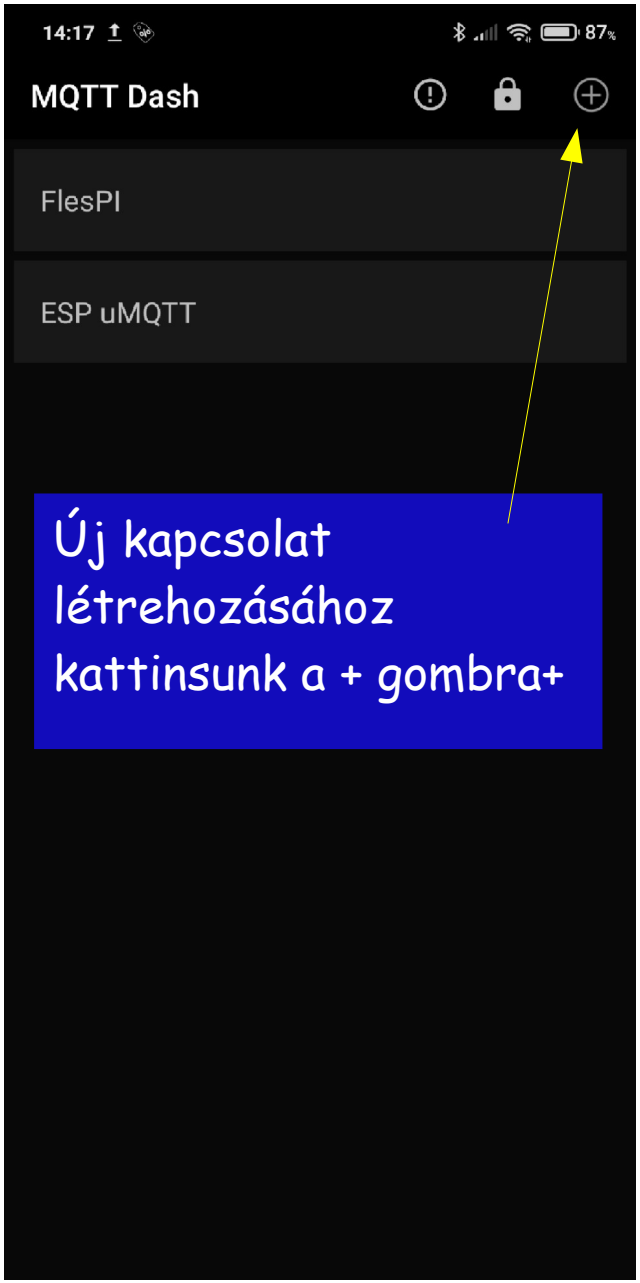
- Az ***rgb_topic***-ba érkező adat **#RRGGBB** formátumú hexadecimális szám (pl. **#203A4C**), amit a # karakter elhagyásával ***long*** típusúvá konvertálunk

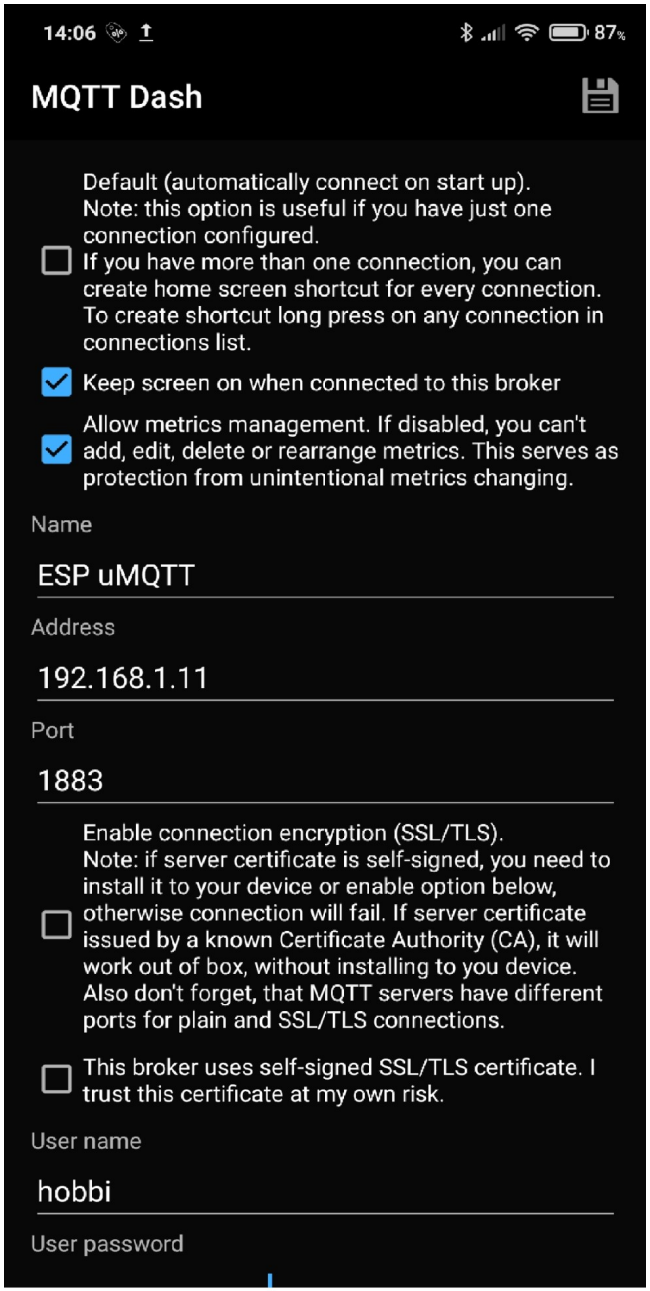

MQTT Explorer
Search...
DISCONNECT
192.168.1.11
cspista
home
bme280 = {"temperature":"27.0","humidity":36,"pressure":"1010.6"}
temp = 27.5
humidity = 25
led = 1
rgb = #9D1919
cspista/home/temp
30
28
26
24
22
20
cspista/home/humidity
50
40
30
20
10
pressure
cspista/home/bme280
1k
1k
1k
1k
1k
1k
1k
humidity
cspista/home/bme280
40
35
30
25
20
Topic
cspista / home / bme280
Value
QoS: 0
04/29/2021
12:45:41 PM
{
"temperature": "27.0",
"humidity": 36,
"pressure": "1010.6"
}
History
99+
Publish
Topic
cspista/home/bme280
raw
xml
json
PUBLISH

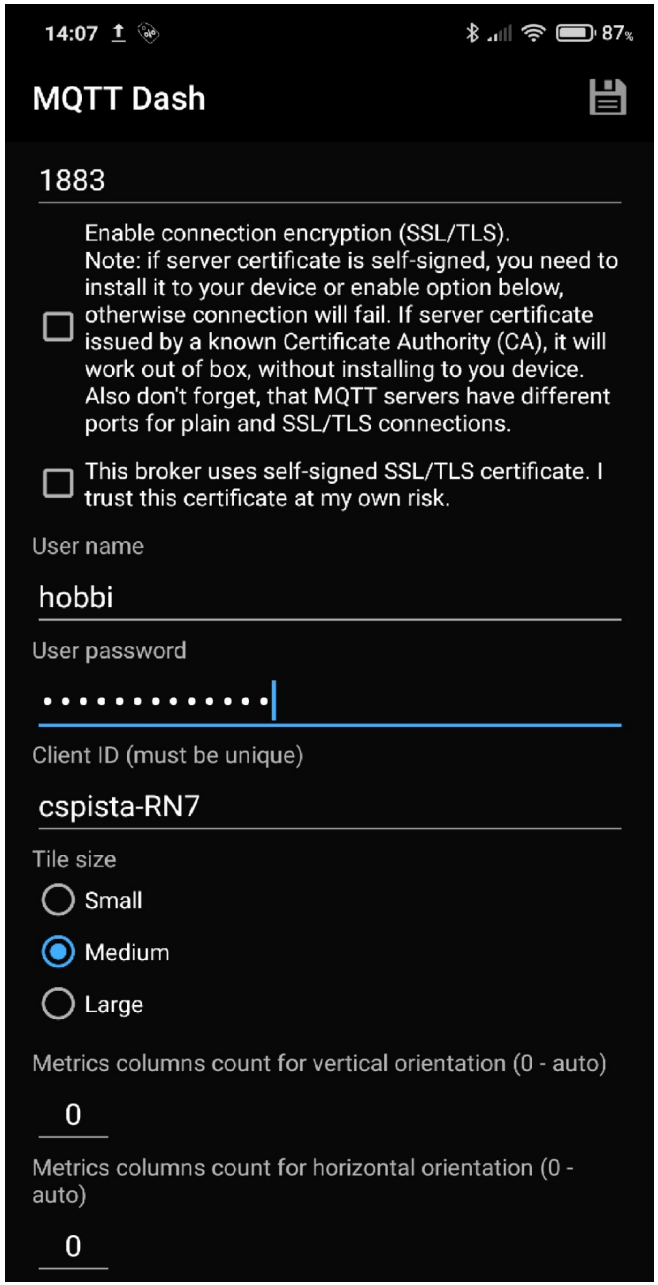
Még egy kliens: MQTT Dash



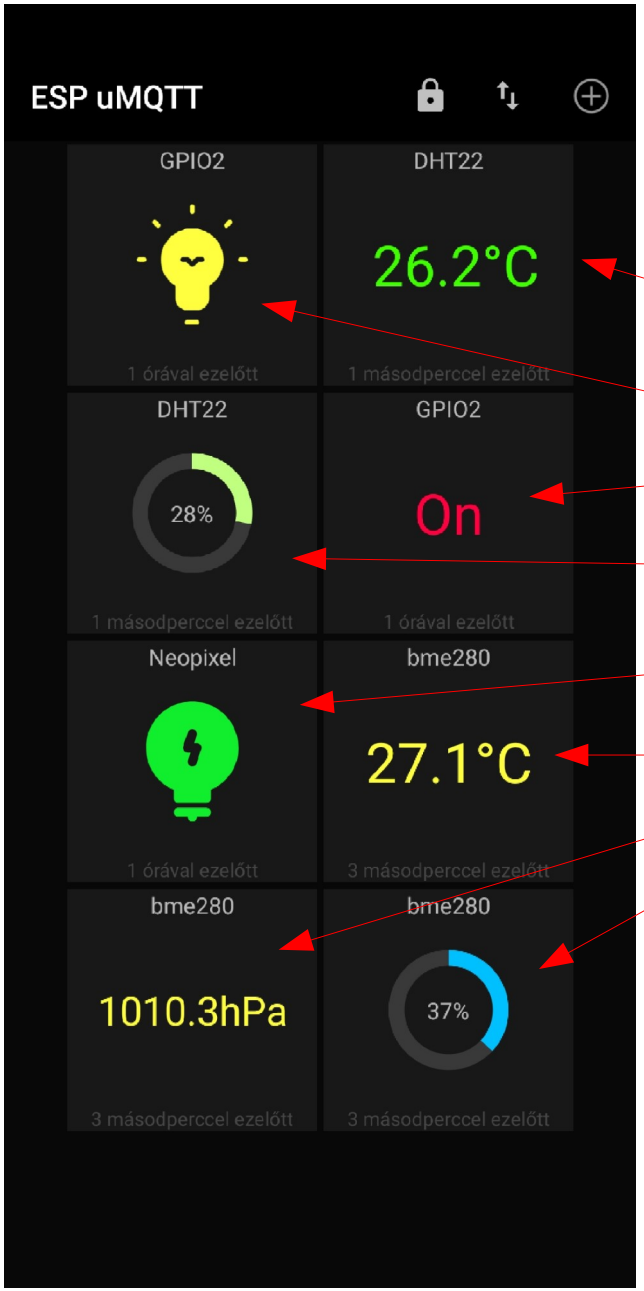
MQTT Dash konfigurálás







MQTT Dash konfigurálás



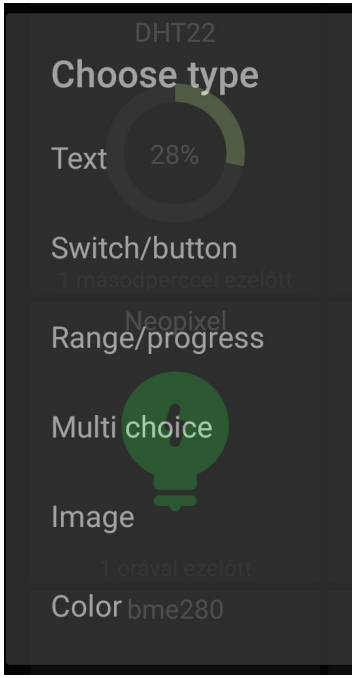
cspista/home/temp

cspista/home/led

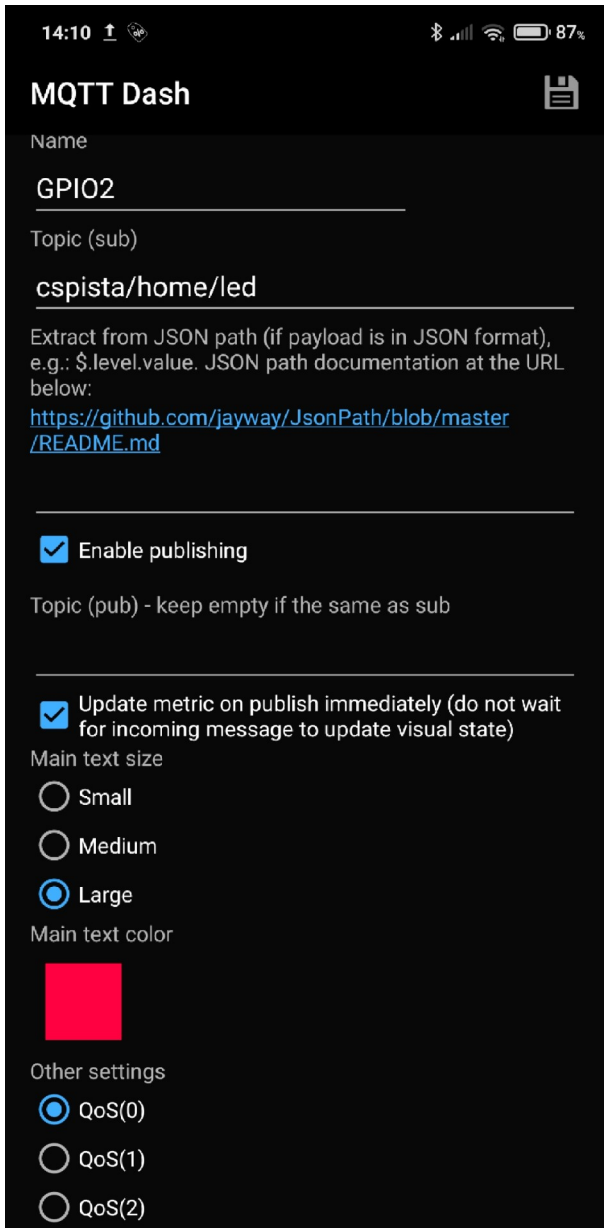
cspista/home/humidity

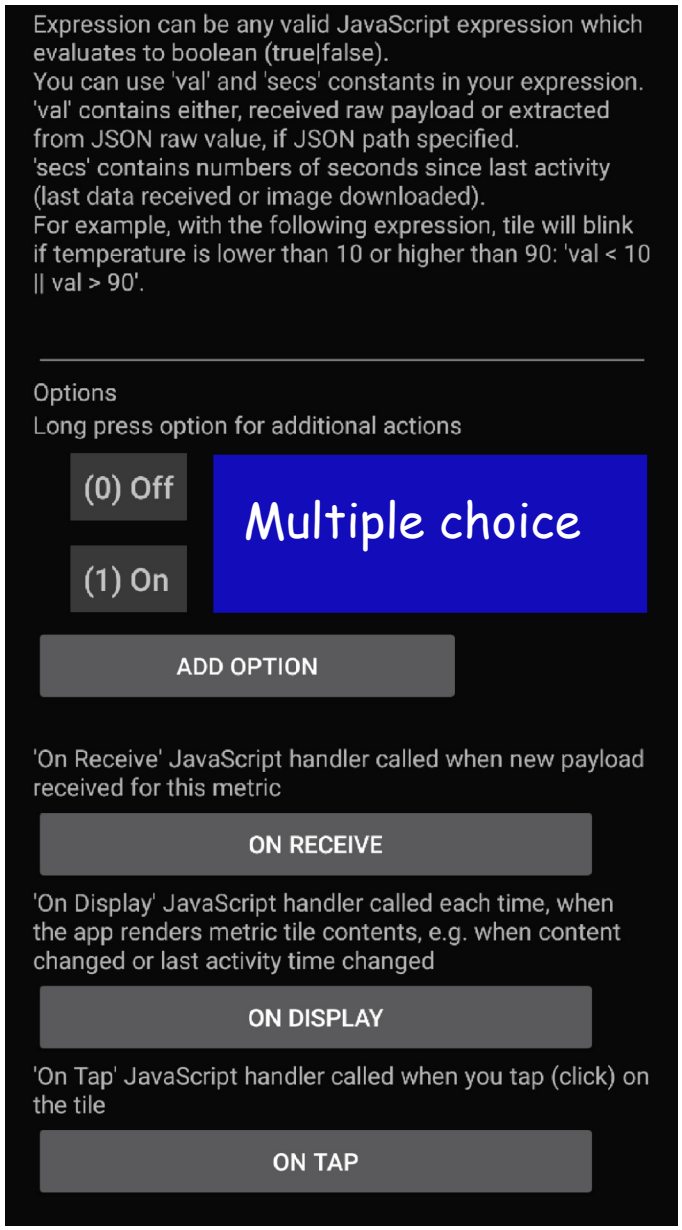
cspista/home/rgb

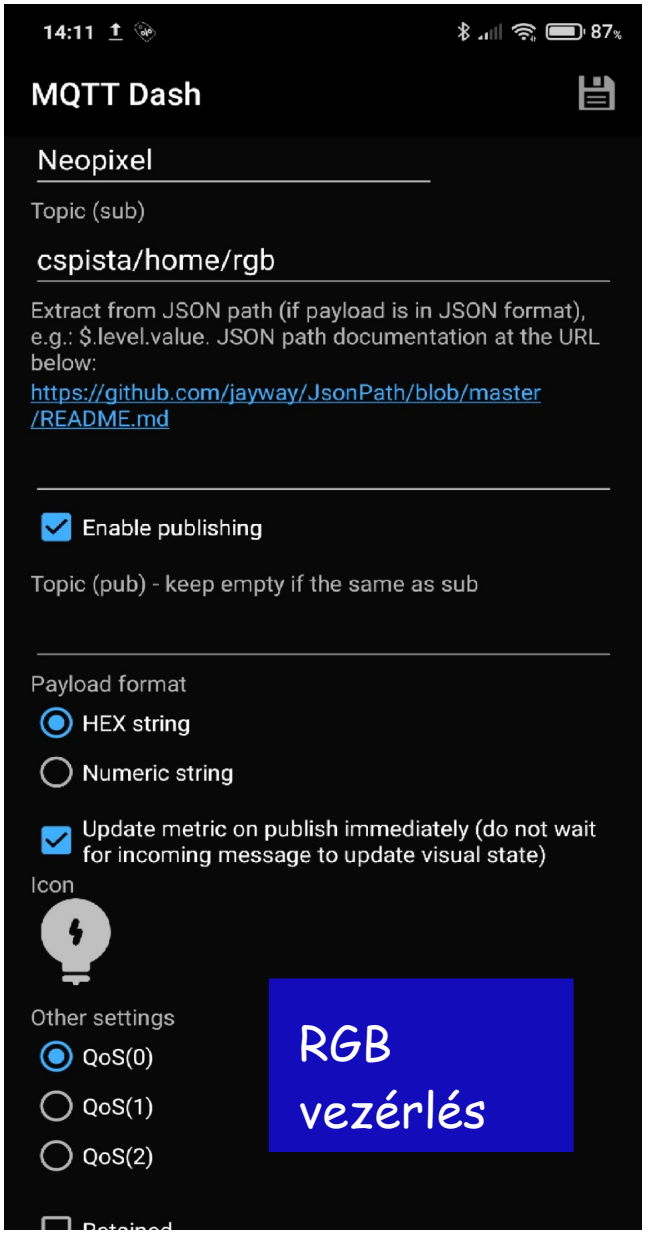
cspista/home/bme280



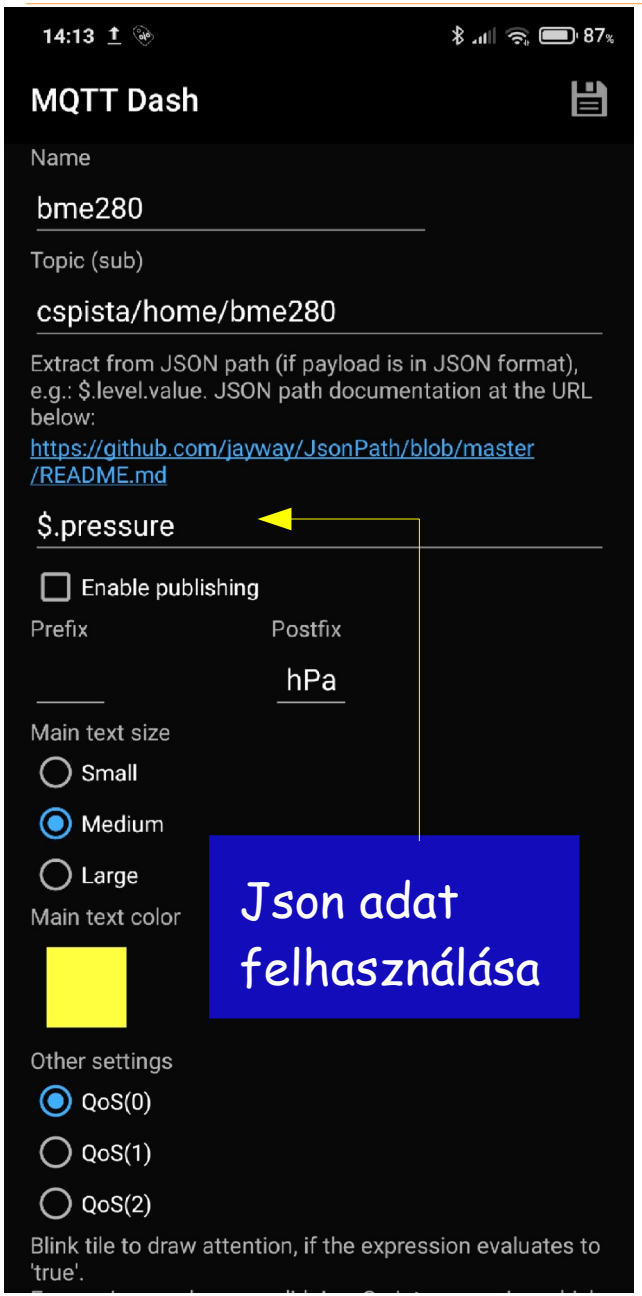
MQTT Dash konfigurálás







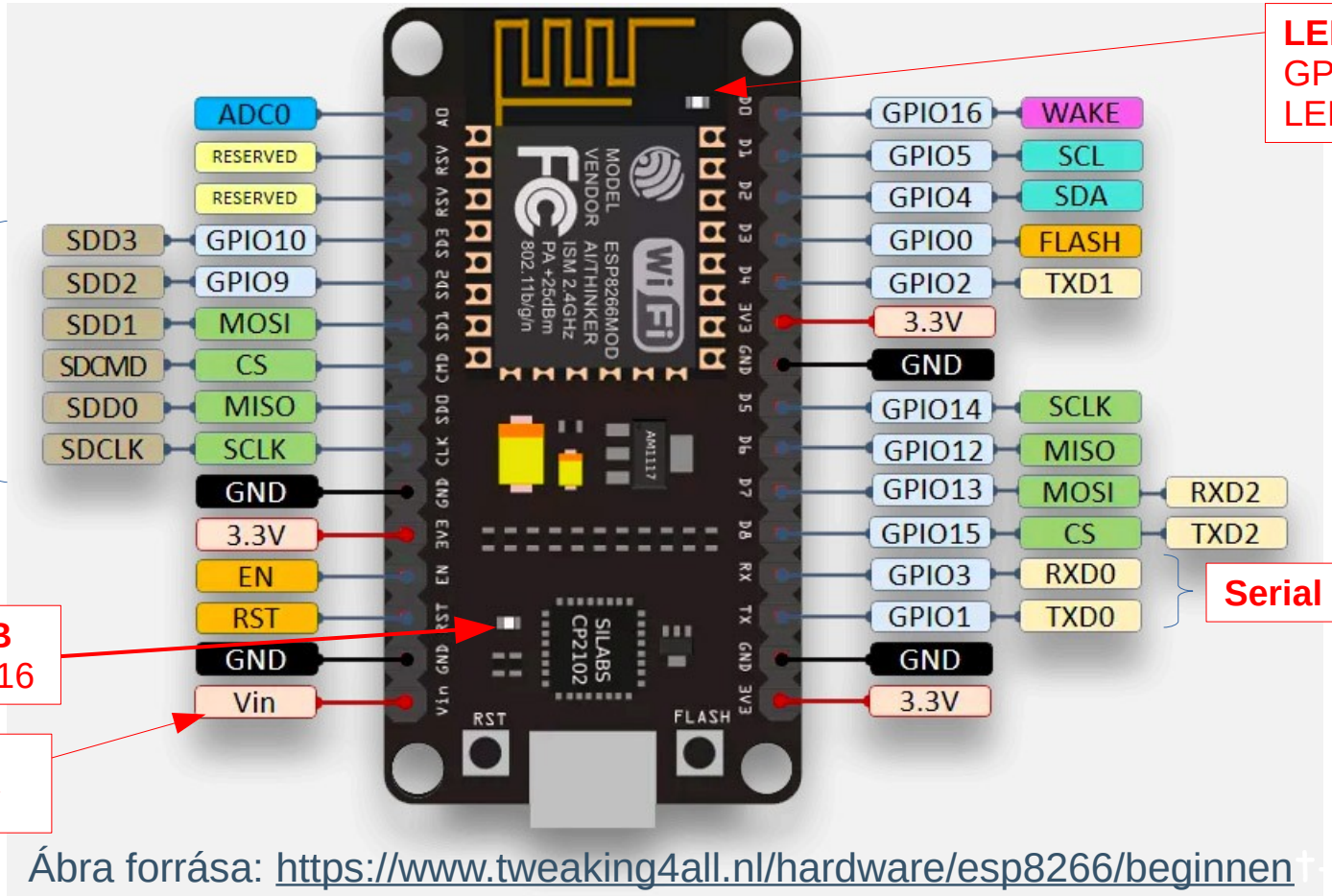
MQTT Dash konfigurálás



- Többszintű adatstruktúra esetén a neveket ponttal elválasztva adhatjuk meg
- A **$.** a kiindulási szintet jelenti (gyökér)

NodeMCU GPIO kivezetések

- A ki- és bemenetek jelszintje 3,3 V (ADC0 esetén 1 V lenne, de a **NodeMCU** kártyán van egy 200 k + 100 k előosztó)
- FLASH a jobboldali nyomógombhoz csatlakozik
- Az Arduino számozás a **GPIO*n*** jelölésből leválasztott ***n*** szám



Ábra forrása: https://www.tweaking4all.nl/hardware/esp8266/beginnen

Név	GPIO
D0	16
D1	5
D2	4
D3	0
D4	2
D5	14
D6	12
D7	13
D8	15
A0	19