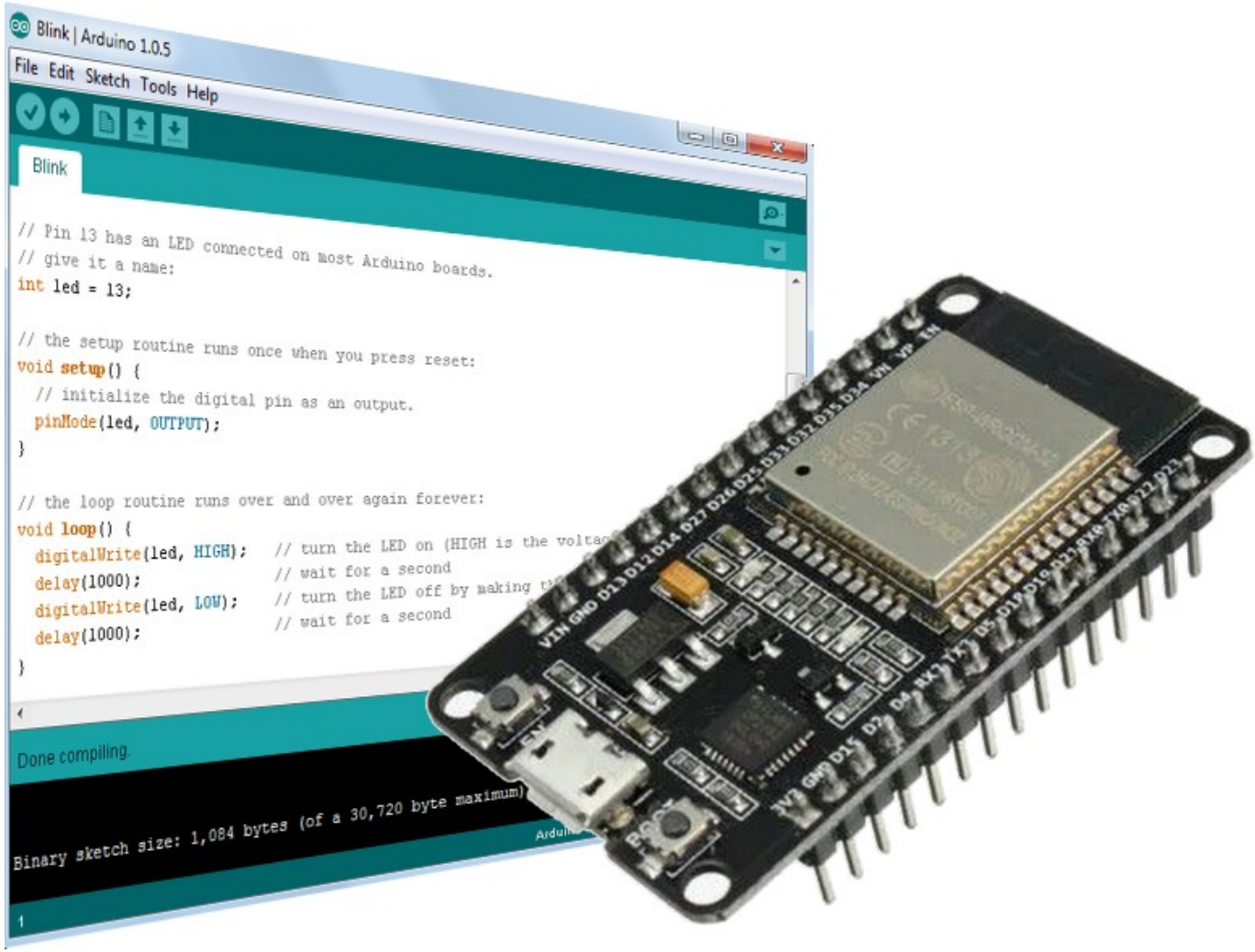
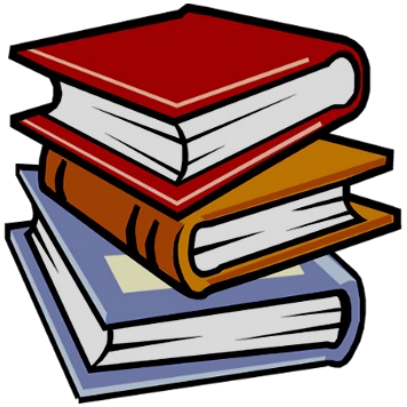
ESP32 mikrovezérlők programozása Arduino környezetben



8. ESP32 MQTT kliens/szerver

Felhasznált és ajánlott irodalom

- Leírások, dokumentáció:
  - HiveMQ : MQTT Essentials
  - OASIS: MQTT Version 3.1.1 specification
  - ESPRESSIF: ESP32 Arduino Core Documentation

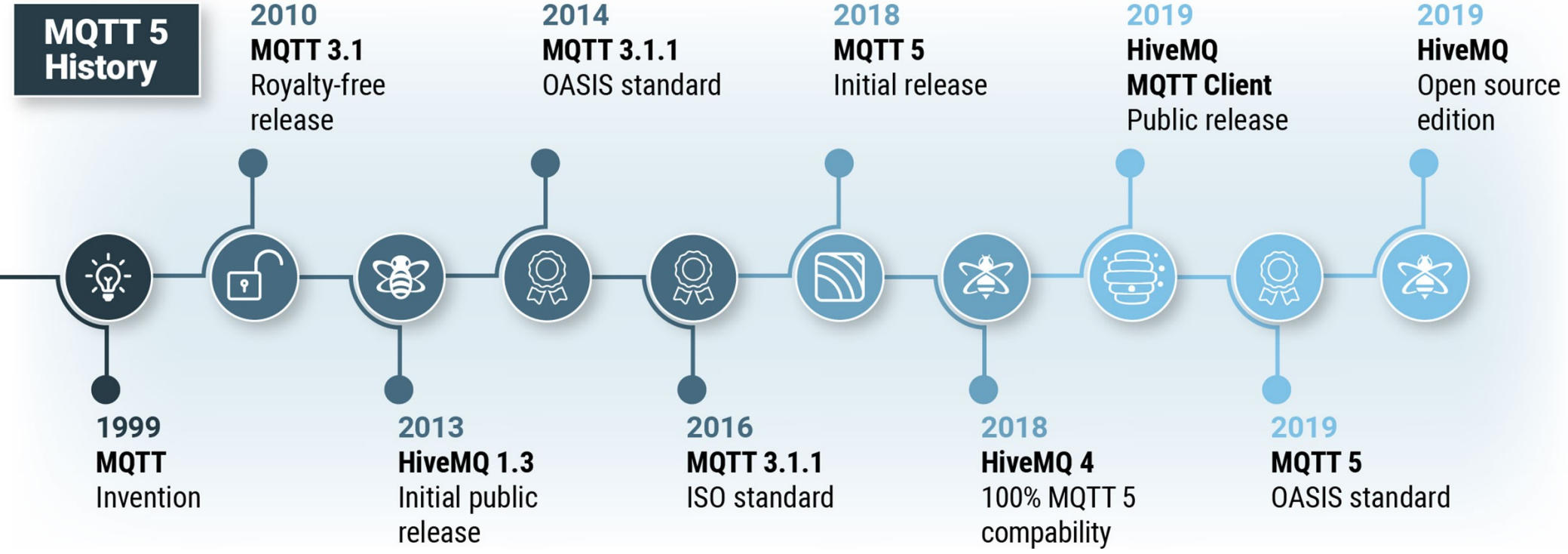


- Arduino programkönyvtárak:
  - Vyacheslav Shiryaev: sMQTTBroker library for ESP32
  - Nick O'Leary: PubSubClient - Arduino Client for MQTT
  - Benoit Blanchon: ArduinoJson library

- PC és mobil segédprogramok:
  - Thomas Nordquist: MQTT Explorer
  - Routix software: MQTT Dash

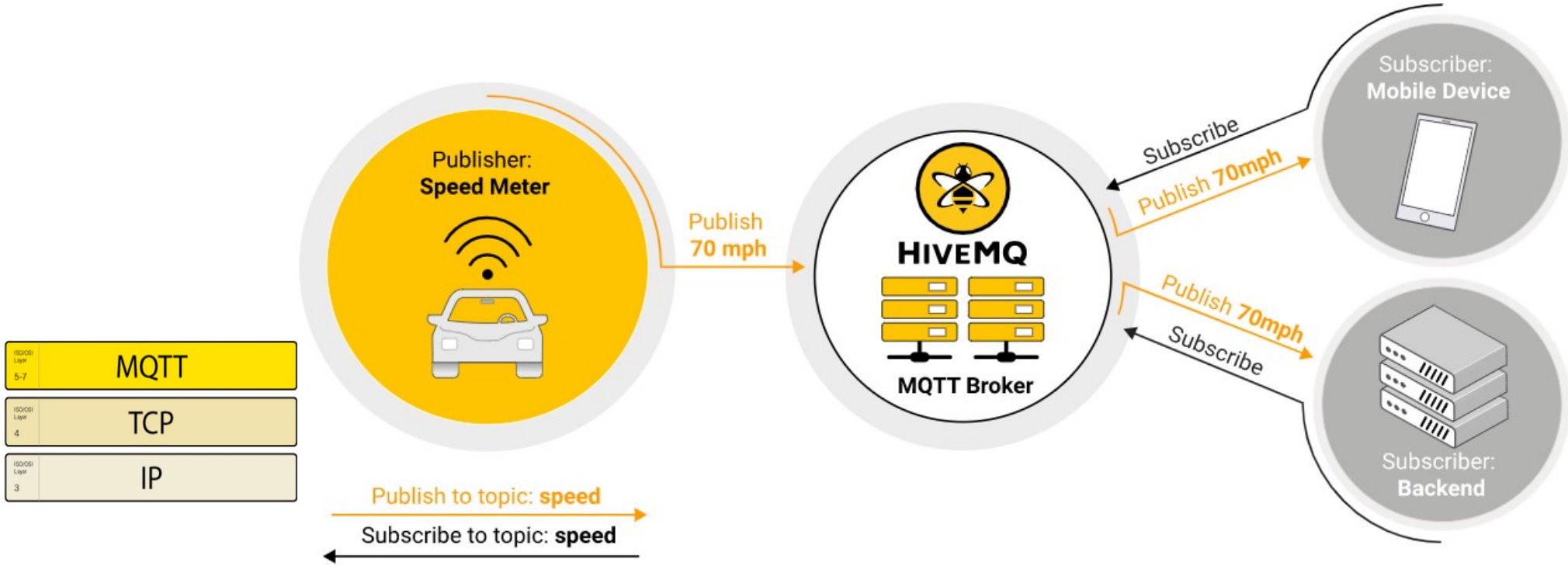
MQTT – MQ Telemetry Transport protokoll

- MQTT: kliens/szerver publish/subscribe üzenettovábbító protokoll
- Eredetileg az **IBM** szakemberei dolgozták ki ezt a protokollt telemetriai adatok továbbítására, amit 2010-ben nyilvánossá tettek
- **OASIS** - Organization for the Advancement of Structured Information Standards – egy globális nonprofit konzorcium, amely oroszlánrészt vállalt az **MQTT** protokoll szabványosításában és továbbfejlesztésében



MQTT alapok

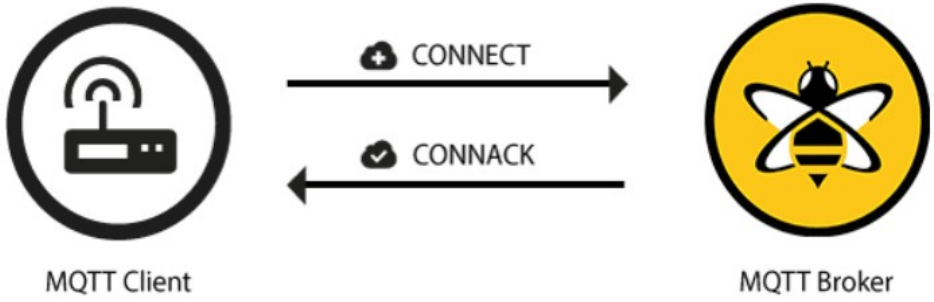
- A **publish/subscribe** séma a hagyományos kliens/szerver sémától eltérően elkülöníti az adatküldő klienst (**publisher**) azoktól a kliensektől amelyek fogadják az üzeneteket (**subscribers**).
- A küldő és fogadó kliensek sohasem kerülnek közvetlen kapcsolatba. A kapcsolatot egy harmadik komponens, az ún. bróker kezeli
- Az **MQTT** kommunikáció **TCP/IP** alapú



Forrás: https://www.hivemq.com/mqtt-essentials/

MQTT kliens – bróker/szerver kapcsolat

- A **kliens**, amely adatot küld és/vagy fogad, lehet PC, mobiltelefon vagy egy IOT eszköz (mikrovezérlő)
- A **bróker** (szerver) dolga:
  - az összes **beérkező üzenet** fogadása és szűrése
  - eldönti, hogy **ki iratkozott fel** az adott témakörű üzenetek fogadására
  - az **üzenetek továbbítása** az adott témakörre feliratkozott klienseknek
  - a bróker tárolja az **állandó kapcsolatú** (persistent sessions) klienseknek a kapcsolati adatait és a feliratkozások és az elmulasztott üzenetek adatait is
- A kliens `CONNECT` üzenetet küld, a bróker `CONNACK` üzenettel és egy `STATUS` kóddal válaszol

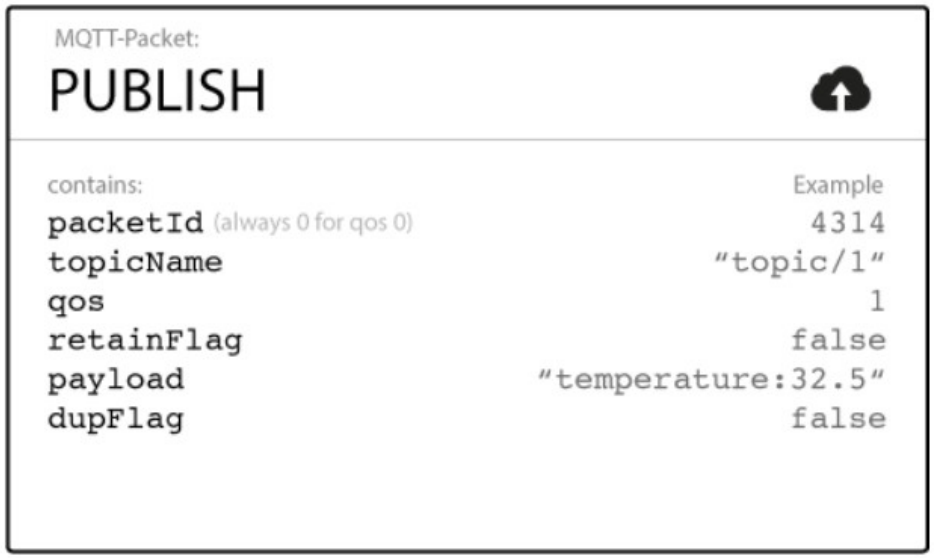


Return Code	Return Code Response
0	Connection accepted
1	Connection refused, unacceptable protocol version
2	Connection refused, identifier rejected
3	Connection refused, server unavailable
4	Connection refused, bad user name or password
5	Connection refused, not authorized

Forrás: https://www.hivemq.com/mqtt-essentials/

MQTT Publish - adatküldés

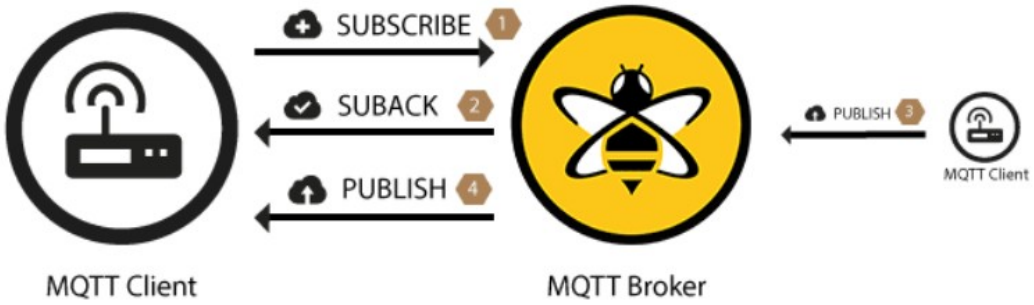
- Sikeres kapcsolódás után a kliens **adatokat küldhet** (publish) a brókernek
- Minden üzenetnek tartalmaznia kell egy **témakör** (topic) megnevezését, amely alapján a bróker szétküldi az adatot mindazon klienseknek, amelyek feliratkoztak (subscribe) az adott témakörre
- **topicName** - egyszerű karakterfüzér, amely `/` jellel tagolva hierarchikusan strukturálható, például: *home/kitchen/temperature*, vagy *home/room/temperature*
- **payload** – a témakörhöz tartozó adat, ami lehet szöveges vagy numerikus is
- **qos** – quality of services
  0 – legfeljebb egyszer
  1 – legalább egyszer
  2 – pontosan egyszer



Forrás: https://www.hivemq.com/mqtt-essentials/

MQTT Subscribe - feliratkozás

- A kliens **feliratkozhat** (subscribe) témakörök adainak fogadására
- Minden üzenetnek tartalmaznia kell egy, vagy több **témakör** (topic) megnevezését, amelyekre feliratkozunk



MQTT-Packet:
SUBSCRIBE

contains:		Example
packetId		4312
qos1	(list of topic + qos)	1
topic1		"topic/1"
qos2		0
topic2		"topic/2"
...		...

MQTT-Packet:
SUBACK

contains:		Example
packetId		4313
returnCode 1	(one returnCode for each topic from SUBSCRIBE, in the same order)	2
returnCode 2		0
...		...

SUBACK válaszkódok

Return Code	Return Code Response
0	Success - Maximum QoS 0
1	Success - Maximum QoS 1
2	Success - Maximum QoS 2
128	Failure

Forrás: https://www.hivemq.com/mqtt-essentials/

MQTT témakörök

- Az **MQTT** témakör egy UTF-8 karakterfüzér amit a bróker az üzenetek szűrésére használ

- Helyettesítő karakterek:
  - Egyszintű: +

  - Többszintű: #

PubSubClient programkönyvtár

- **PubSubClient** – kliens programkönyvtár MQTT kommunikációra
- **PubSubClient** (*server, port, [callback], **client,** [stream]*) – constructor
  server – pl. broker.hivemq.com 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

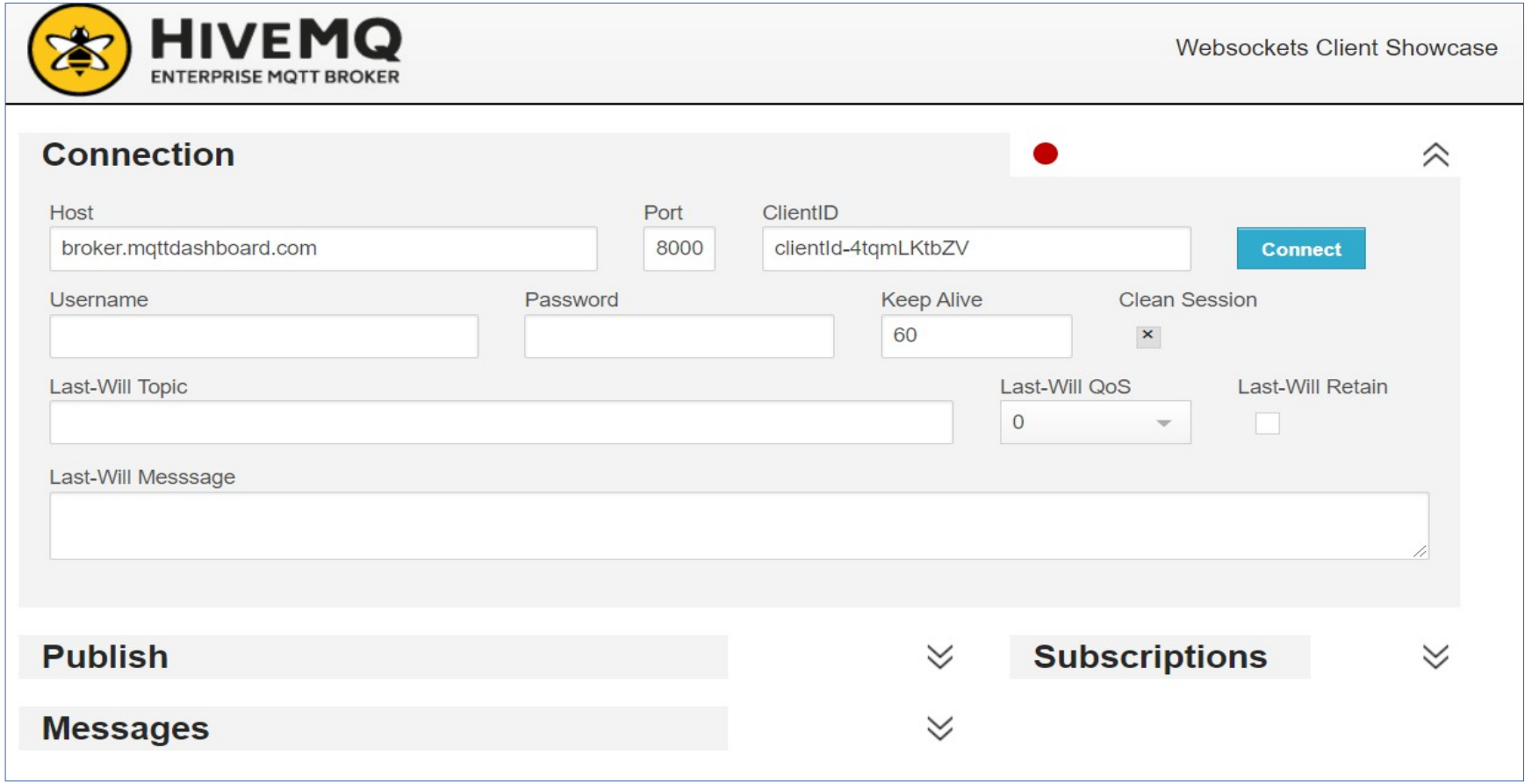
```
void callback(char* topic, byte* payload, unsigned int length);
```

- **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

HiveMQ nyilvános MQTT bróker és kliens

- Regisztrálni a https://www.hivemq.com/mqtt-cloud-broker/ címen lehet, az itt megadott **belépési név** és **jelszó** kell majd a csatlakozáshoz
- Nyilvános **MQTT** bróker: `broker.hivemq.com`, port: `1883`
- Online **MQTT** kliens: `http://www.hivemq.com/demos/websocket-client/`



Példaprogramok

ESP32_HiveMQ_DHT22 – kliens HiveMQ brókerhez

ESP32_sMQTT_DHT22 – kliens helyi ESP brókerhez

ESP32_sMQTT_DHT22_neopixel– kibővített funkciójú MQTT kliens ESP32 brókerhez

ESP32_sMQTTBroker – jelszóvédett MQTT bróker

ESP32_sMQTTBroker_BMP280 – ESP32 bróker és JSON publikáló kliens

A csatlakozások titkos adatai

- Az **ESP32** kliensként történő csatlakozásokhoz a személyes adatokat kiszerveztük egy **secrets.h** nevű fejléc állományba, amelyet a **Vázlatfüzet** (Sketchbook) mappa **libraries/secrets** almappájában helyeztünk el

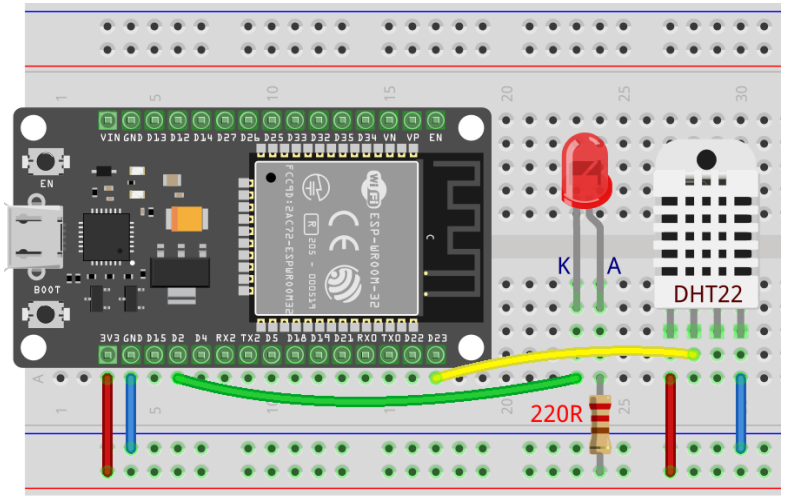
```
#include <WiFi.h>
#include "secrets.h"

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"
```

Hőmérséklet és páratartalom mérése

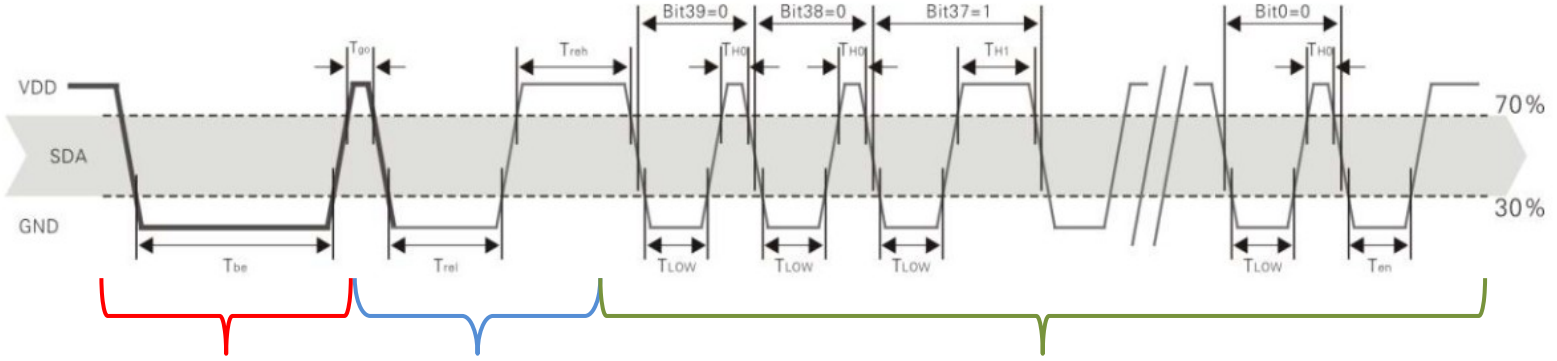
Az Am2302 (DHT22) szenzor jellemzői:

- Hőmérséklet és rel. páratartalom mérésére
- Felbontás: 0.1 °C, illetve 0.1 %
- 1-wire, nem szabványos protokol
- Mintavételezési gyakoriság: 2 s
- Tápfeszültség: 3,3 – 6 V



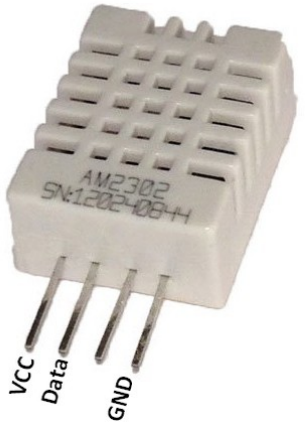
GPIO2: LED-, GPIO23: DHT22

- 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 …



ESP32_HiveMQ_DHT22.ino

- **MQTT** kliens program a **DHT22** szenzor adatok továbbítására, a **PubSubClient** programkönyvtár felhasználásával
- A bróker neve: `broker.hivemq.com` a belépéshez szükséges `HIVEMQ_USER` és `HIVEMQ_PASS` a *secrets.h* állományban definiált
- A publikáláshoz választott témakörök neve: `cspista/home/temp`, valamint `cspista/home/humidity`
- Kapcsolódáskor vagy újrakapcsolódáskor a `cspista/home` témakörbe küldünk egy `"DHT22 connected"` üzenetet
- A feliratkozásra használt témakör neve: `cspista/home/led`
- A csatlakozáshoz használt **clientID** tetszőleges, de egyedi név kell, hogy legyen, itt most egy E-mail cím lesz: `hobbi@cspista.hu`
- A `cspista/home/led` témakörre feliratkozva, a beérkező üzenetek első karakterét figyeljük, s ha az '1', akkor bekapcsoljuk, egyébként kikapcsoljuk a beépített LED-et (GPIO2, katódvezérléssel)

ESP32_HiveMQ_DHT22.ino 3/1.

```
#include <WiFi.h>
#include <PubSubClient.h>
#include "DHT.h"
#include "secrets.h"
const char* mqtt_server = "broker.hivemq.com";
#define DHTPIN          23              // GPIO23 pin for DHT sensor
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;

void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived ["); Serial.print(topic); Serial.print("] : ");
  for (int i = 0; i < length; i++) { Serial.print((char)payload[i]); }
  Serial.println();
  if ((char)payload[0] == '1') {
    digitalWrite(BUILTIN_LED, LOW);   // Turn the LED on (pulling down cathode)
  } else {
    digitalWrite(BUILTIN_LED, HIGH);  // Turn the LED off (pulling up cathode)
  }
}
```

ESP32_HiveMQ_DHT22.ino 3/2.

```
void reconnect() {
  while (!client.connected()) {     // Loop until we're reconnected
    Serial.print("Attempting MQTT connection...");
    char clientId[] = "hobbi@cspista.hu";
    if (client.connect(clientId, HIVEMQ_USER, HIVEMQ_PASS)) { // Attempt to connect
      Serial.println("connected");
      // Once connected, publish an announcement...
      client.publish("cspista/home", "DHT22 connected");
      client.subscribe("cspista/home/led");  // ... and resubscribe
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      Delay(5000);                  // Wait 5 seconds before retrying
    }
  }
}

void setup() {
  pinMode(BUILTIN_LED, OUTPUT);     // LED
  Serial.begin(115200);
  dht.begin();
  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
}
```

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

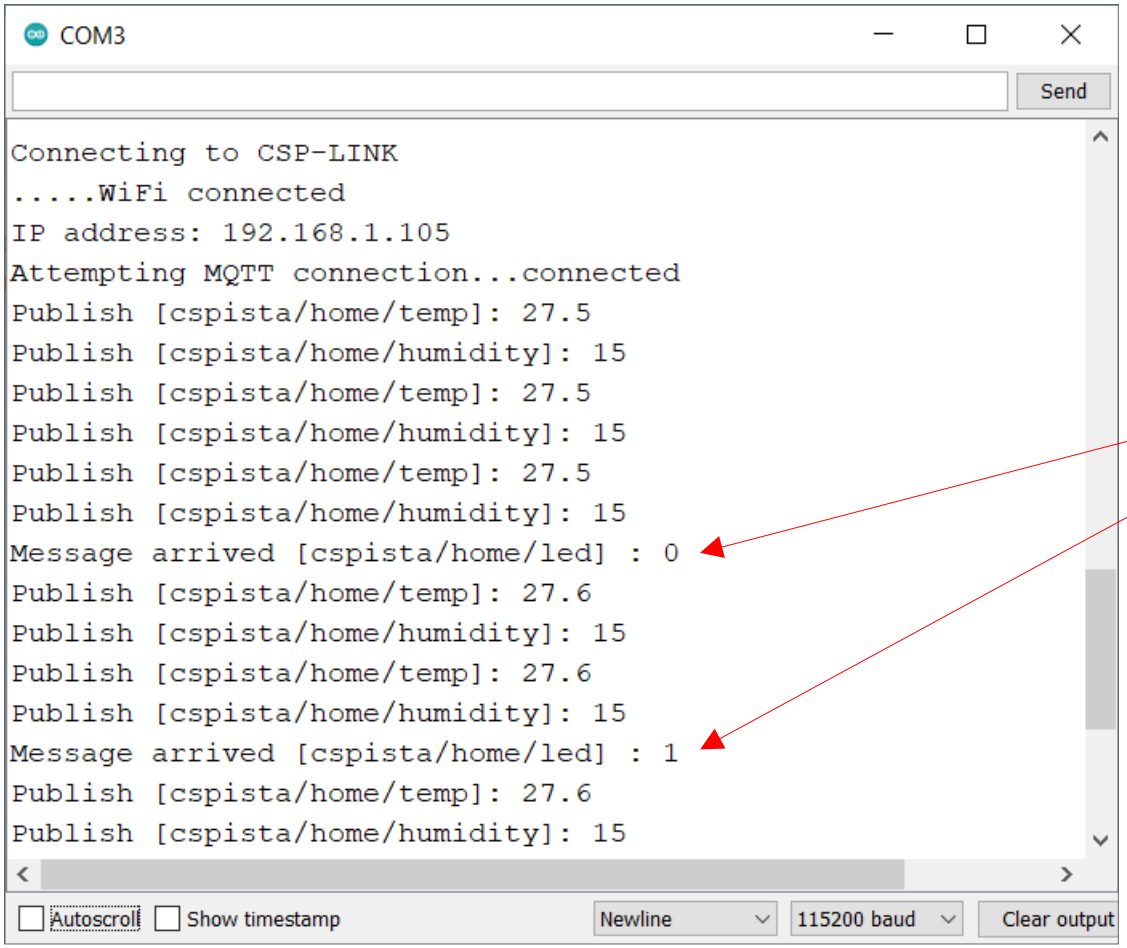
ESP32_HiveMQ_DHT22.ino 3/3.

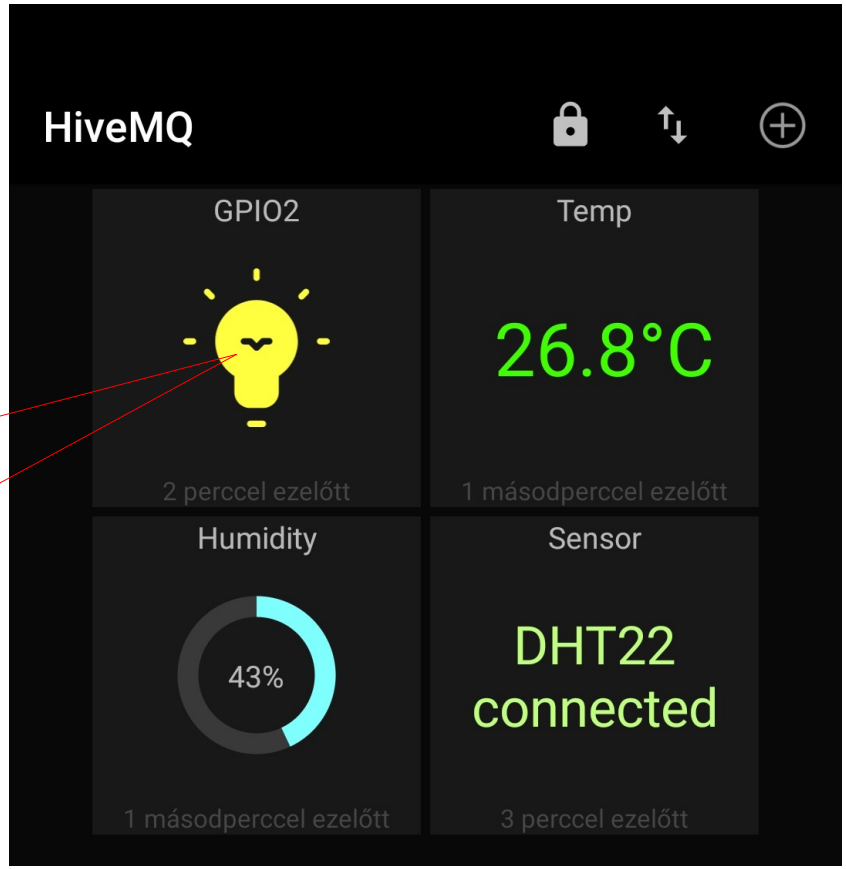
```
void loop() {

  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  unsigned long now = millis();
  if (now - lastMsg > 5000) {
    lastMsg += 5000;
    float t = dht.readTemperature();
    if (!isnan(t)) {
      sprintf(msg, "%5.1f", t);
      client.publish("cspista/home/temp", msg);
      Serial.print("Publish [cspista/home/temp]:");
      Serial.println(msg);
    }
    float h = dht.readHumidity();
    if (!isnan(h)) {
      sprintf(msg, "%3.0f", h);
      client.publish("cspista/home/humidity", msg);
      Serial.print("Publish [cspista/home/humidity]:");
      Serial.println(msg);
    }
  }
}
```

Rendszeresen meg kell hívni!

ESP32_HiveMQ_DHT22.ino

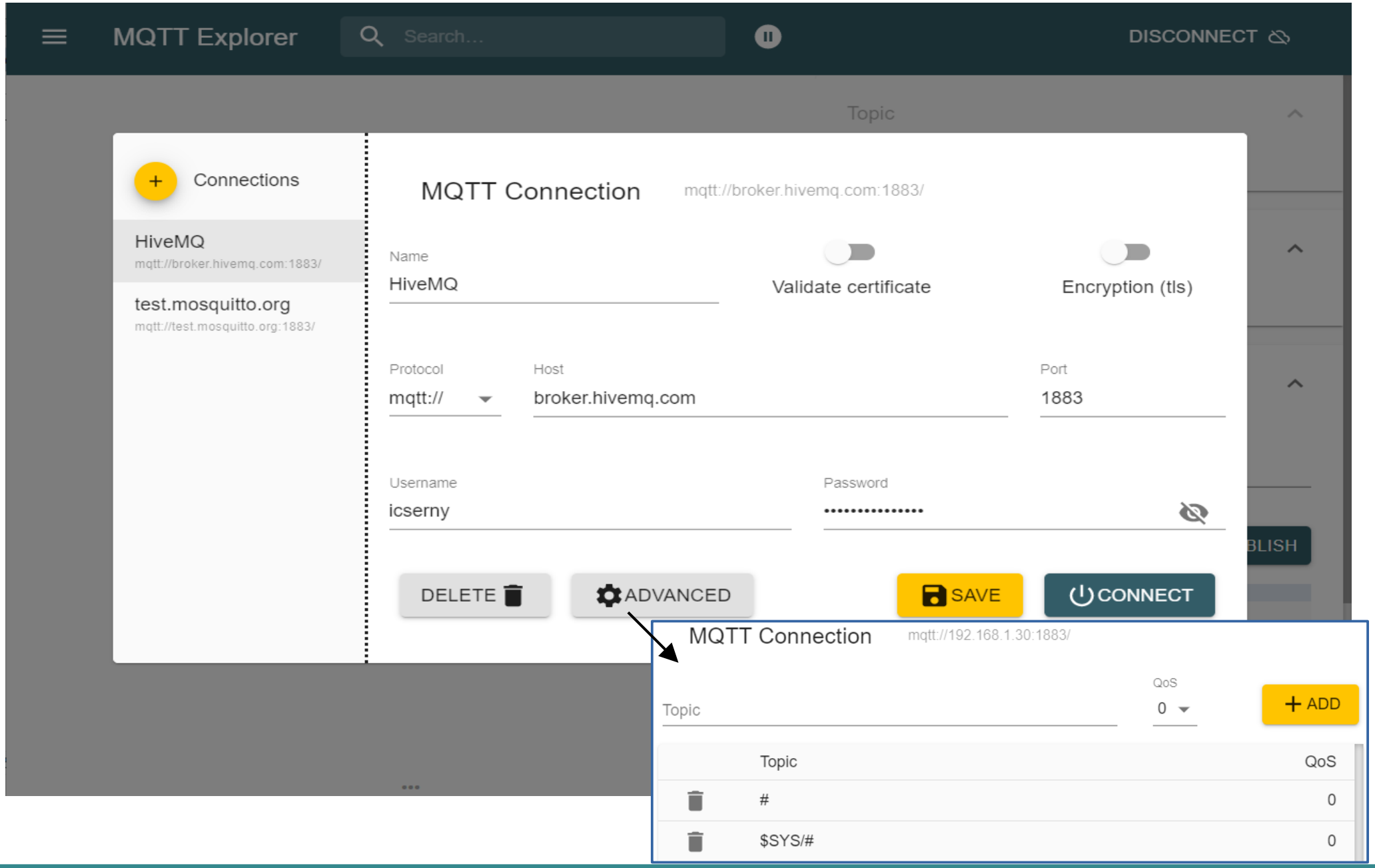
- A program futási eredménye (a LED-et itt az **MQTT DASH** Android alkalmazással kapcsolgattuk)

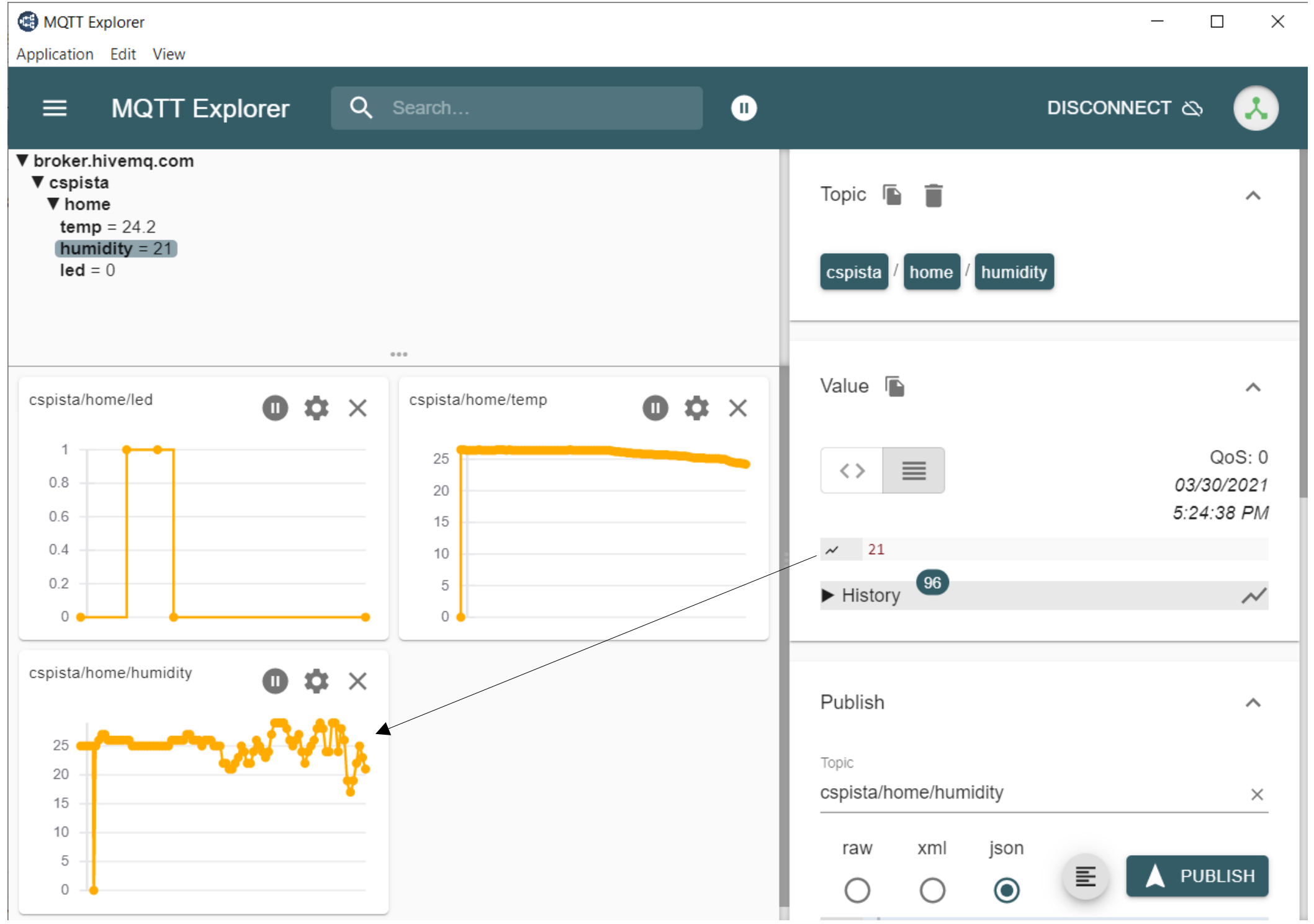




Switch/button	Text + postfix (°C)
progress+posfix (%)	Text

MQTT Explorer (Windows alkalmazás)




MQTT Explorer
Application Edit View
MQTT Explorer
Search...
DISCONNECT
broker.hivemq.com
cspista
home
temp = 24.2
humidity = 21
led = 0
Topic
cspista / home / humidity
Value
QoS: 0
03/30/2021
5:24:38 PM
21
History 96
cspista/home/led
cspista/home/temp
cspista/home/humidity
Publish
Topic
cspista/home/humidity
raw xml json
PUBLISH

ESP32 MQTT bróker

- Készítsünk helyi brókert egy ESP32 felhasználásával!
- **Előzmények:** a 2021. április 29-i előadásban **ESP8266 Nodemcu** kártya és az uMQTTBroker programkönyvtár segítségével alakítottunk ki helyi MQTT brókert. Ez a programkönyvtár azonban **ESP32**-re nincs adaptálva
- **ESP32 Lehetőségek:**
  - **TinyMQTT:** MQTT client/broker (Mqtt 3.1.1 / Qos 0 támogatás)
  - **sMQTTBroker:** simple MQTT broker (Mqtt 3.1.1 / Qos 0 támogatás)
- Ebben az előadásban az utóbbit fogjuk használni

```
#include"sMQTTBroker.h"
sMQTTBroker broker;

void setup() {
    WiFi.begin(WIFI_SSID, WIFI_PASS);
    while (WiFi.status() != WL_CONNECTED) delay(1000);
    broker.init(1883);
}

void loop() {
    broker.update();
}
```

A minimalista MQTT bróker pl. így néz ki

A minimalista megoldás hátránya, hogy nincs jelszavas ellenőrzés, nincs naplózás és a beérkező üzeneteket sem látjuk (szűrés vagy feldolgozás céljából)

Az sMQTTBroker osztály publikus tagfüggvényei

- Az **init()** és **update()** fügvények használatát már bemutattuk
- A további lehetőségek kihasználására az **sMQTTBroker** osztályból saját broker osztályt származtatunk és a virtuális függvényeket felüldefiniáljuk

```
bool init(unsigned short port);
void update();
void publish(std::string &topic, std::string &payload,char qos=0,bool retain=false);
void restart();

// inner functions
void publish(sMQTTClient *client, sMQTTTopic *topic, sMQTTMessage *msg);
bool subscribe(sMQTTClient *client, const char *topic);
void unsubscribe(sMQTTClient *client, const char *topic);
bool isTopicValidName(const char *filter);
void updateRetainedTopic(sMQTTTopic *topic);
bool isClientConnected(sMQTTClient *client);
```

```
// virtual functions
virtual bool onConnect(sMQTTClient *client, std::string &user, std::string &pass) {
  return true; };
virtual void onRemove(sMQTTClient*) {};
virtual void onPublish(sMQTTClient *client,std::string &topic,std::string &payload) {};
virtual bool onEvent(sMQTTEvent *event) { return true; }
```

ESP32_sMQTTBroker.ino 2/1.

- Amint látjuk, maga a program váza semmivel sem bonyolultabb, mint a korábban bemutatott „*simple broker*”, a varázslat tehát a **MyBroker** osztály definíciójában rejlik:
- A kliensek kapcsolódásakor az **onConnect()** függvényben kapott felhasználónevet és jelszót a **secrets.h** állományban definiált `uMQTT_USER` és `uMQTT_PASS` értékével hasonlítjuk össze és egyezés esetén *true* értékkel térünk vissza (egyébként *false* értékkel)
- A visszahívási függvényekben naplózást végzünk a soros portra történő kiírásokkal

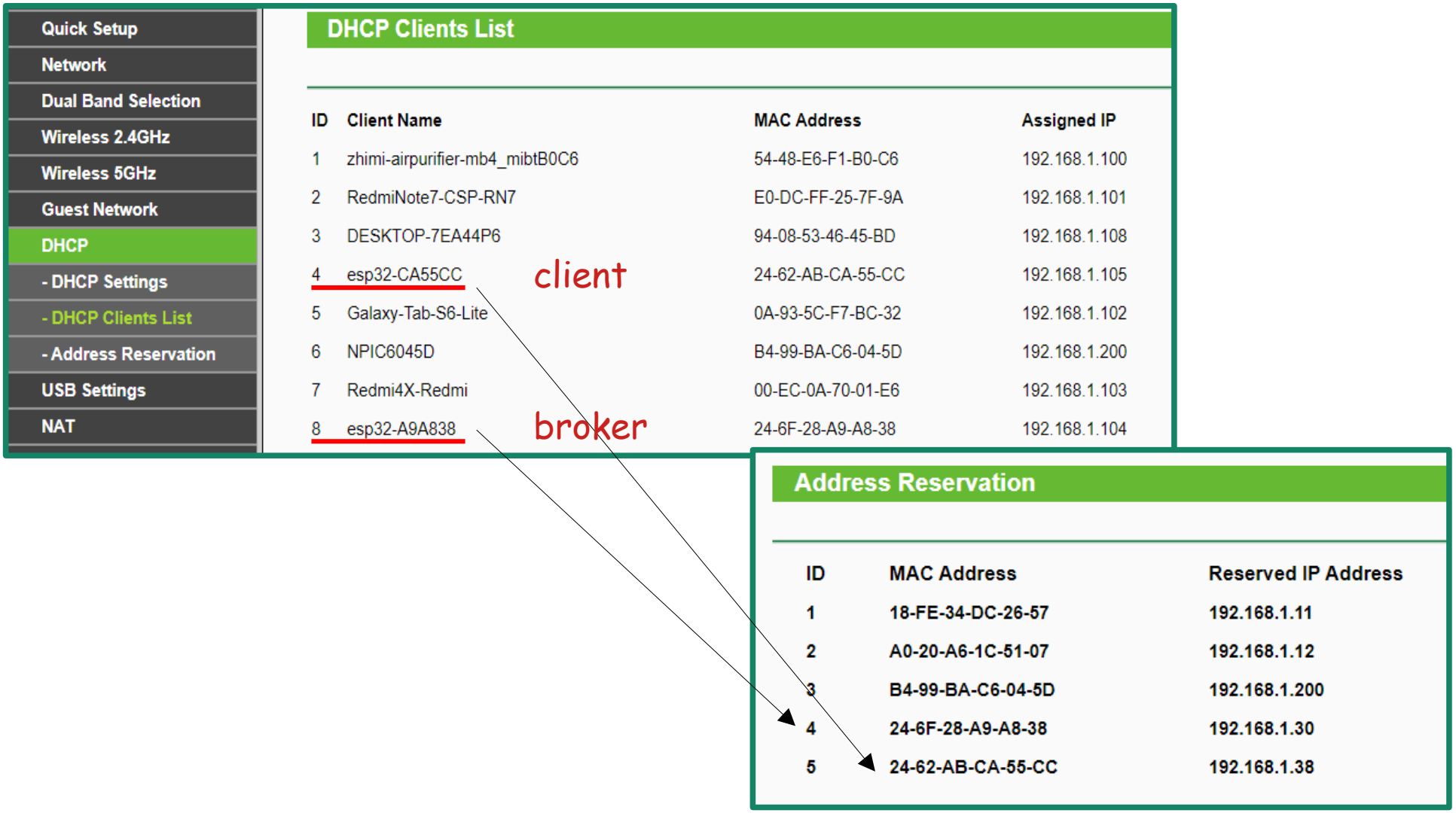
```
#include <WiFi.h>
#include <sMQTTBroker.h>
#include "secrets.h"

class MyBroker: public sMQTTBroker {

  . . .

}


MyBroker broker;

void setup() {
  Serial.begin(115200);
  setup_wifi();
  broker.init(1883);
};

void loop() {
  broker.update();
};
```

ESP32_sMQTTBroker.ino 2/2.

```
class MyBroker: public sMQTTBroker {
public:
  bool onConnect(sMQTTClient *client, const std::string &user, const std::string &pass)
    { if( (0 == user.compare(uMQTT_USER)) && (0 == pass.compare(uMQTT_PASS))) {
        std::printf("Client %s connected\r\n", client->getClientId().c_str());
        return true;
      } else {
        std::printf("Client %s failed to connect\r\n", client->getClientId().c_str());
        return false;
      }
    };
  void onRemove(sMQTTClient* client) {
      std::printf("Client %s disconnected\r\n", client->getClientId().c_str());
    };
  void onPublish(sMQTTClient *client, const std::string &topic, const std::string &payload){
      std::printf("Client %s published %s into %s\r\n", client→getClientId().c_str(),
         payload.c_str(), topic.c_str());
    }
  bool onEvent(sMQTTEvent *event) {
    switch (event→Type())  {
      case NewClient_sMQTTEventType:
        { sMQTTNewClientEvent *e = (sMQTTNewClientEvent*)event;
          e→Login();  e→Password();  }  break;
      case LostConnect_sMQTTEventType: WiFi.reconnect();  break;
    }
    return true;
  }
};
```

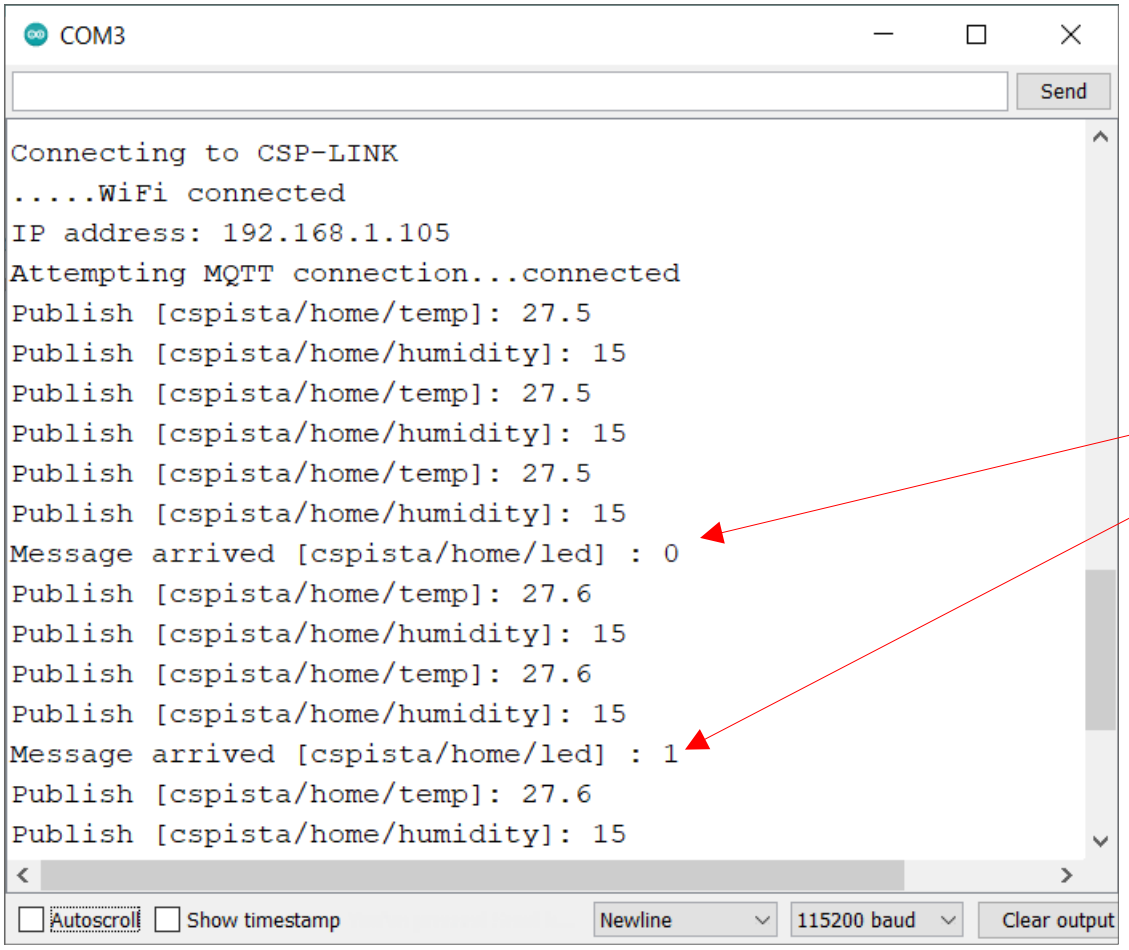
A bróker megérdemel egy fix IP címet

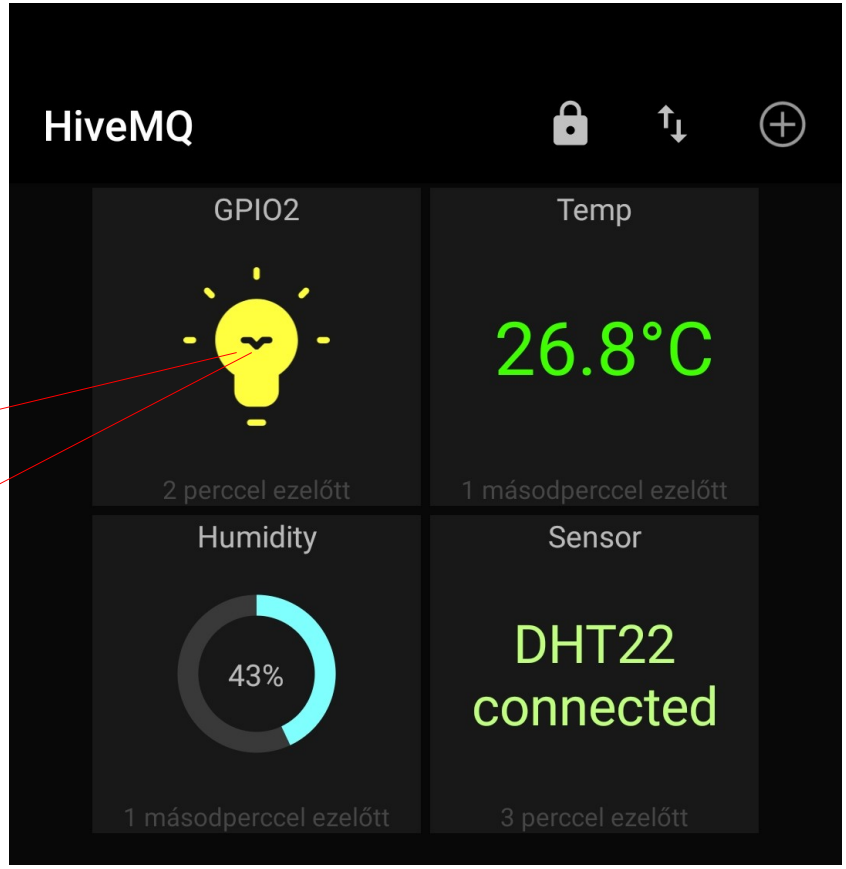
- A bróker könnyű eléréséhez rendeljük fix IP címeket a helyi routeren az ESP32 modulokhoz!



ESP32_sMQTT_DHT22.ino – az új kliens

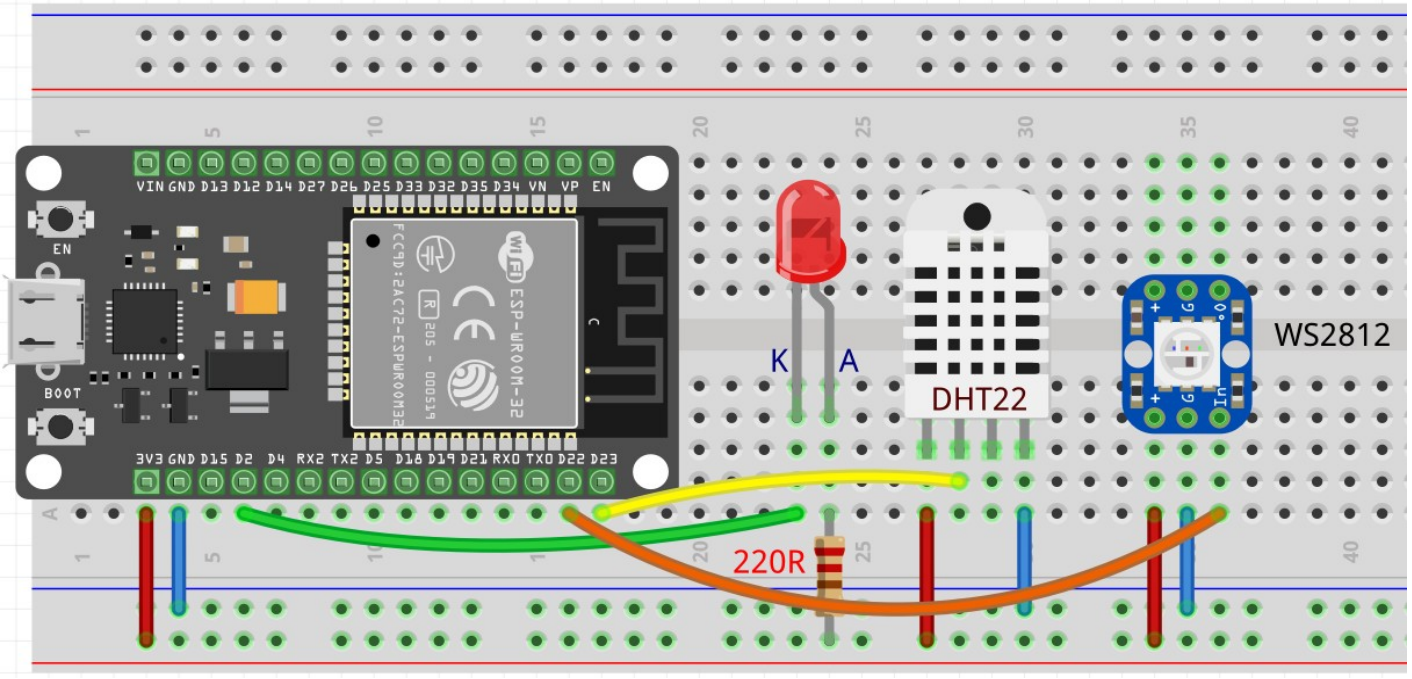
- Ez a kliens csak abban különbözik az első mintapéldában bemutatottól, hogy a bróker neve `broker.hivemq.com` helyett most `192.168.1.30`, a belépéshez pedig a ***secrets.h*** állományban definiált `uMQTT_USER`, illetve `uMQTT_PASS` kell. A kliensek úgy működnek, mint a HiveMQ brókerrel





ESP32_sMQTT_DHT22_neopixel.ino

- Bővítsük az MQTT kliensünket még egy eszközzel, egy **WS2812** típusú, digitálisan vezérelhető RGB LED-del (neopixel), amelyet a **cspista/home/rgb** topikba érkező, **#RRGGBB** formátumú hexadecimális számmal (pl. **#203A4C**) vezérlünk!
- Az új kliens program a 2021. április 29-i előadásban bemutatott **ESP8266_uMQTT_neopixel.ino** példaprogram **ESP32** megfelelője
- A **WS2812** LED vezérléséhet az **Adafruit_Neopixel** könyvtárat használjuk
- A **WS2812** LED DIN adatbemenetét a GPIO22 kivezetéshez kötöttük



ESP32_sMQTT_DHT22_neopixel.ino 3/1.

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <Adafruit_NeoPixel.h>
#include "DHT.h"
#include "secrets.h"
const char* mqtt_server = "192.168.1.30";
#define DHTPIN          23     // GPIO23 pin for DHT sensor
#define NEOPIXEL_PIN    22     // GPIO22 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";
```

ESP32_sMQTT_DHT22_neopixel.ino 3/2.

```
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
  }
}
```

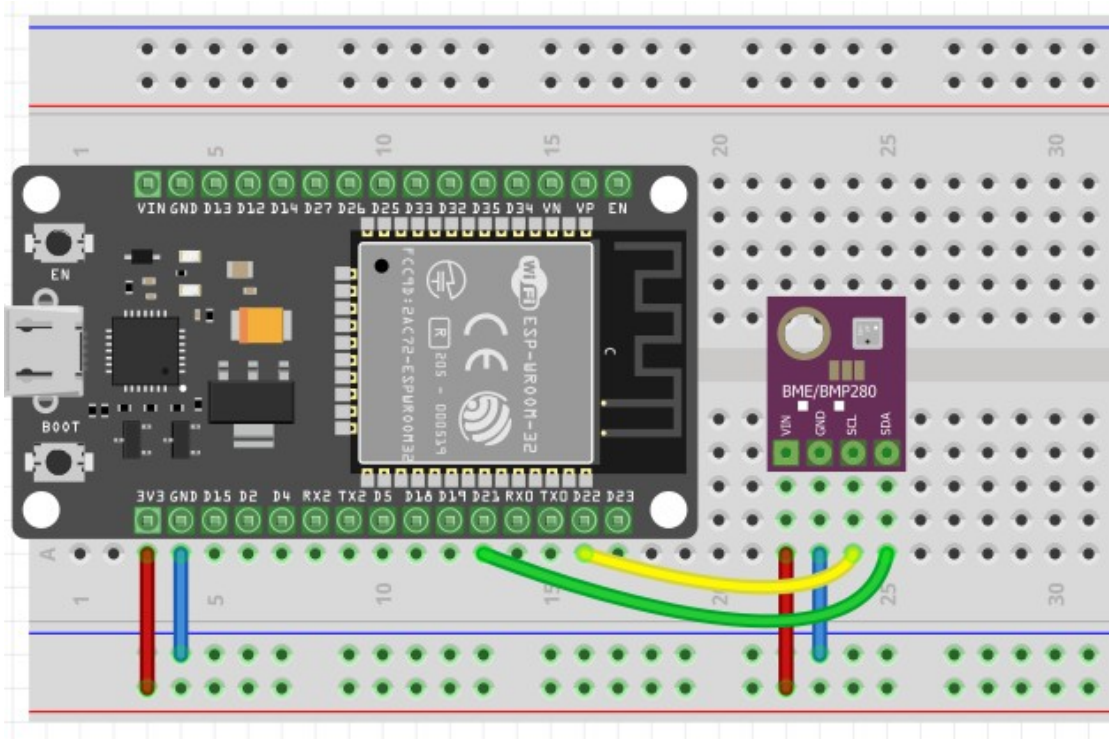
ESP32_sMQTT_DHT22_neopixel.ino 3/3.

```
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

Adatküldés JSON formátumban

- Kibővítjük a rendszert egy **BME280** szenzorral (hőmérséklet, nyomás és relatív páratartalom mérése), s a mérési adatokat egyetlen topikban (**cspista/home/bme280**), JSON formátumba csomagolva publikáljuk
- Technikai okokból nem új klienst vetünk be, hanem az **ESP32** bróker szoftverét bővítjük ki **mérő és publikáló** funkciókkal
- Bekötés:
  - SDA – GPIO21
  - SCL – GPIO22
  - GND – GND
  - VIN – 3V3



ESP8266_uMQTT_Broker_BMP280.ino

```
#include <WiFi.h>
#include "sMQTTBroker.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;

unsigned long lastMsg;
MyBroker broker;

void setup() {
  Serial.begin(115200);
  setup_wifi();
  bme.begin(0x76, &Wire);
  broker.init(1883);
}
```

```
class MyBroker: public
sMQTTBroker {

  . . .

}
```

Ugyanaz, mint korábban

ESP8266_uMQTT_Broker.ino

```
void loop() {
  broker.update();
  if ((millis() - lastMsg) > 5000) {
    //--- Read and publish BME280 sensor data ---------
    doc["temperature"] = String(bme.readTemperature(), 1);
    doc["humidity"] = (int)bme.readHumidity();
    doc["pressure"] = String((bme.readPressure()/98.58566F), 1);  // sea level corr.
    std::string payload = doc.as<std::string>();
    std::string mytopic = "cspista/home/bme280";
    broker.publish(mytopic, payload, 0, false);
    lastMsg += 5000;
  }
};
```

Qos Retain flag

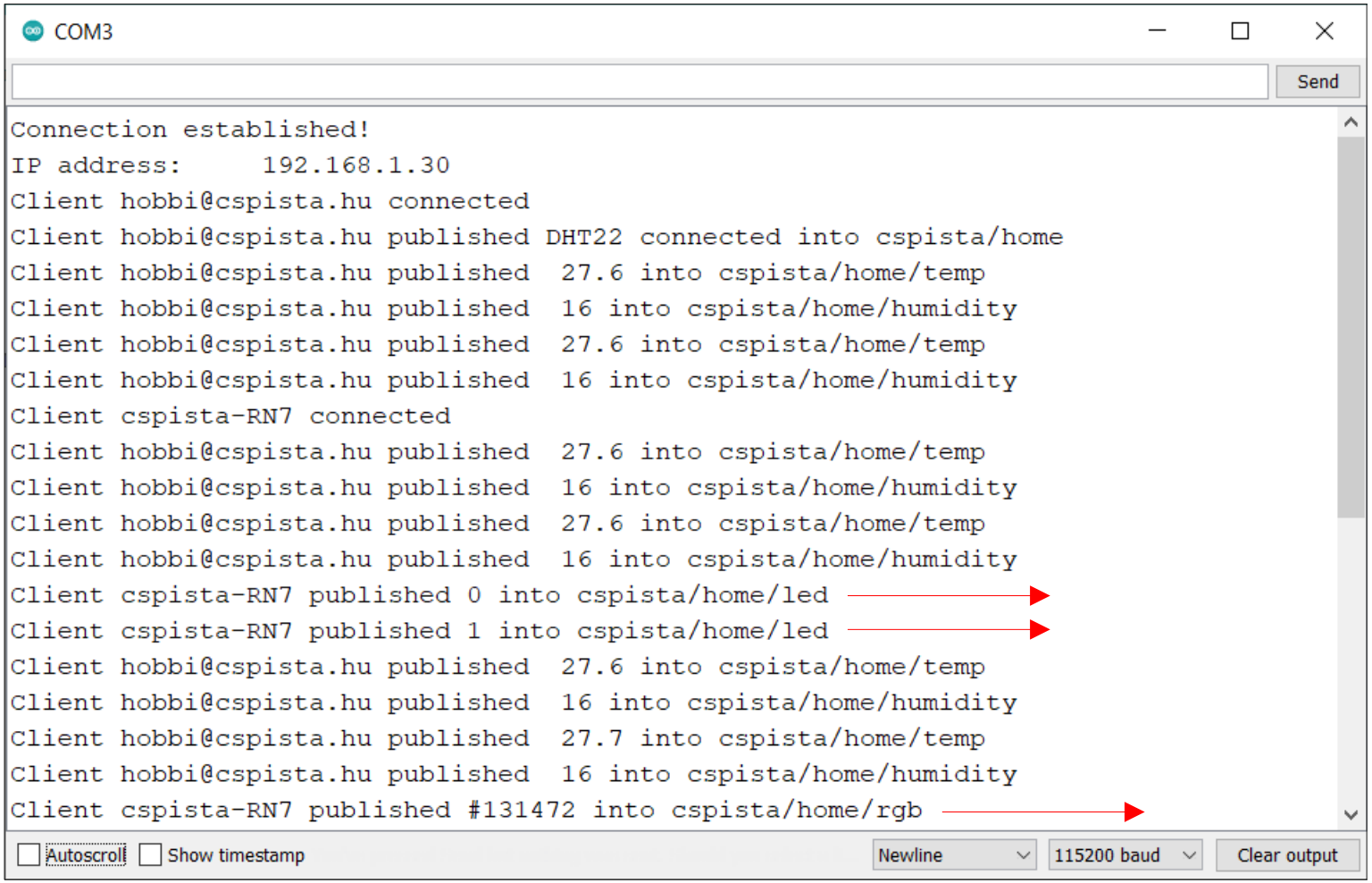
- Arra ügyeljünk, hogy a **serializeJson()** adatfolyamként kezeli a ***String*** objektumokat, azaz hozzáfűz... Például:

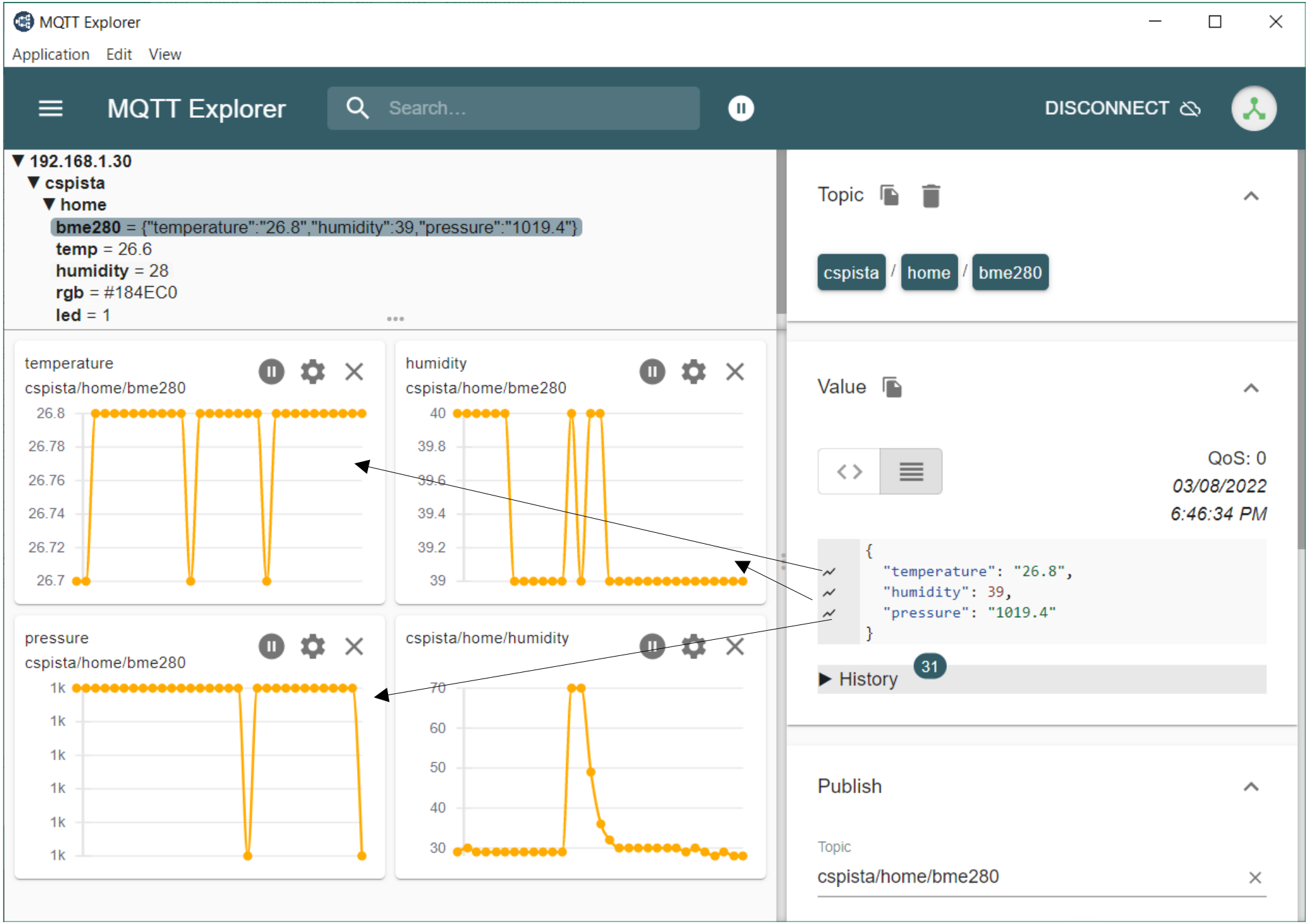
```
String output = "JSON = ";
serializeJson(doc, output);
```

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

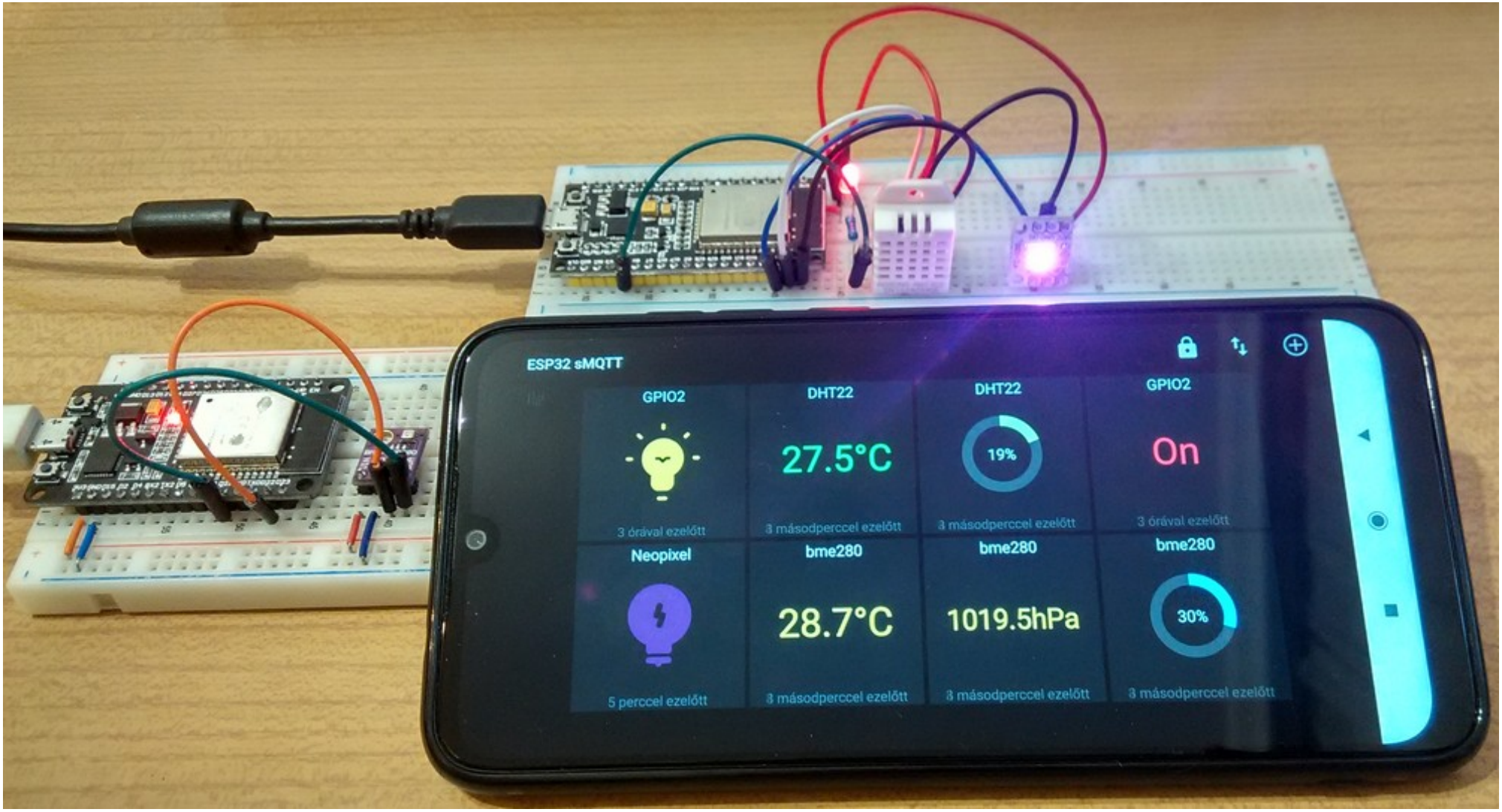
ESP8266_sMQTTBroker_BME280 futási eredmény

- A **cspista/home/bme280** témakörbe a brókerről helyileg küldött üzenetek láthatóan nem jutnak el az ***onPublish()*** visszahívási függvénybe, ezek naplózását nekünk kellett volna megoldani

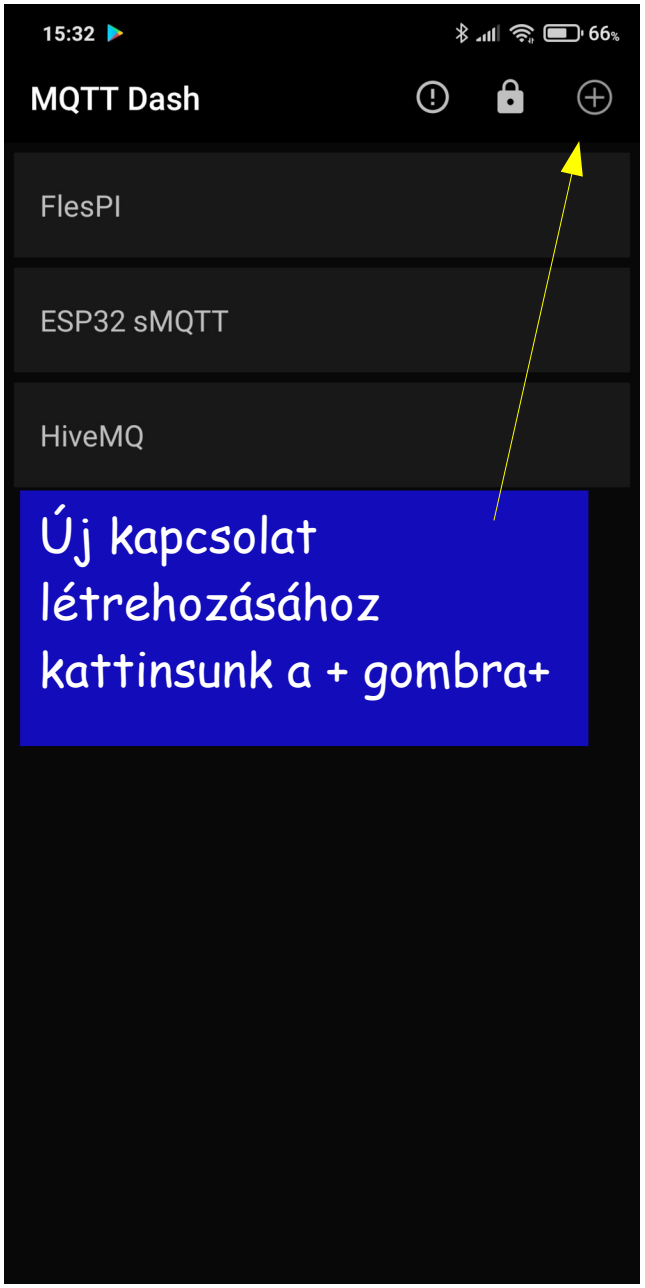


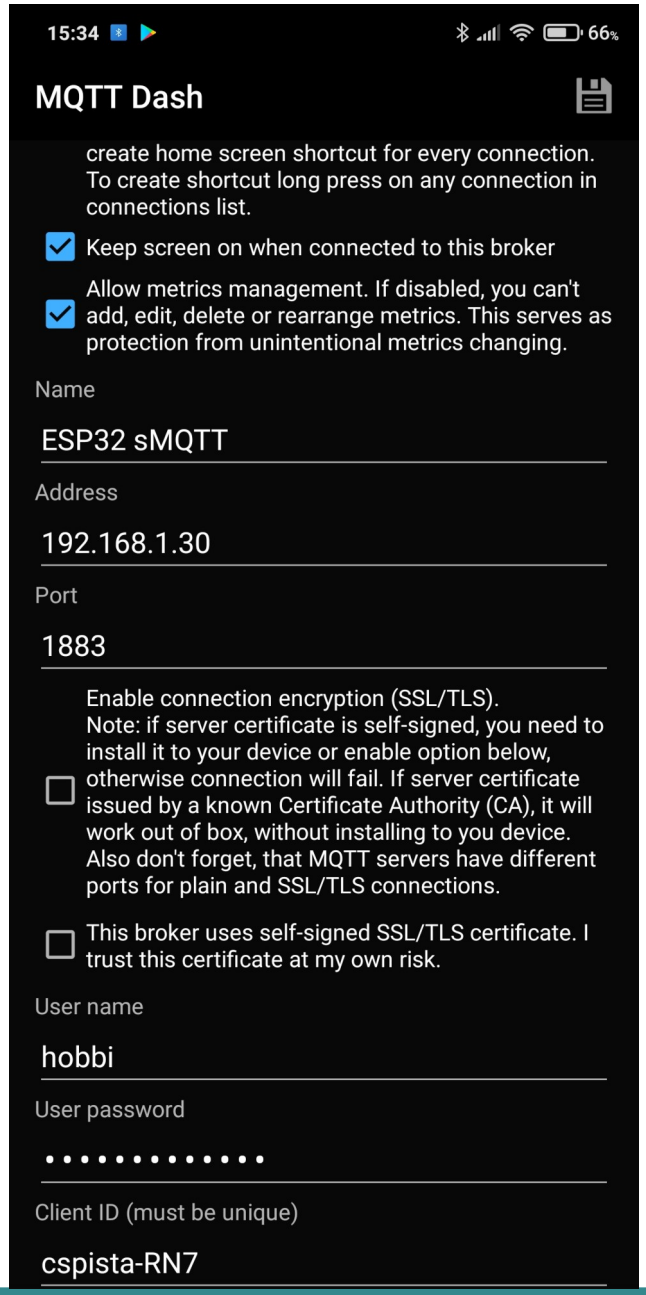

MQTT Explorer
Application Edit View
MQTT Explorer
Search...
DISCONNECT
192.168.1.30
cspista
home
bme280 = {"temperature":"26.8","humidity":39,"pressure":"1019.4"}
temp = 26.6
humidity = 28
rgb = #184EC0
led = 1
temperature
cspista/home/bme280
26.8
26.78
26.76
26.74
26.72
26.7
humidity
cspista/home/bme280
40
39.8
39.6
39.4
39.2
39
pressure
cspista/home/bme280
1k
cspista/home/humidity
70
60
50
40
30
Topic
cspista / home / bme280
Value
QoS: 0
03/08/2022
6:46:34 PM
{
"temperature": "26.8",
"humidity": 39,
"pressure": "1019.4"
}
History
31
Publish
Topic
cspista/home/bme280

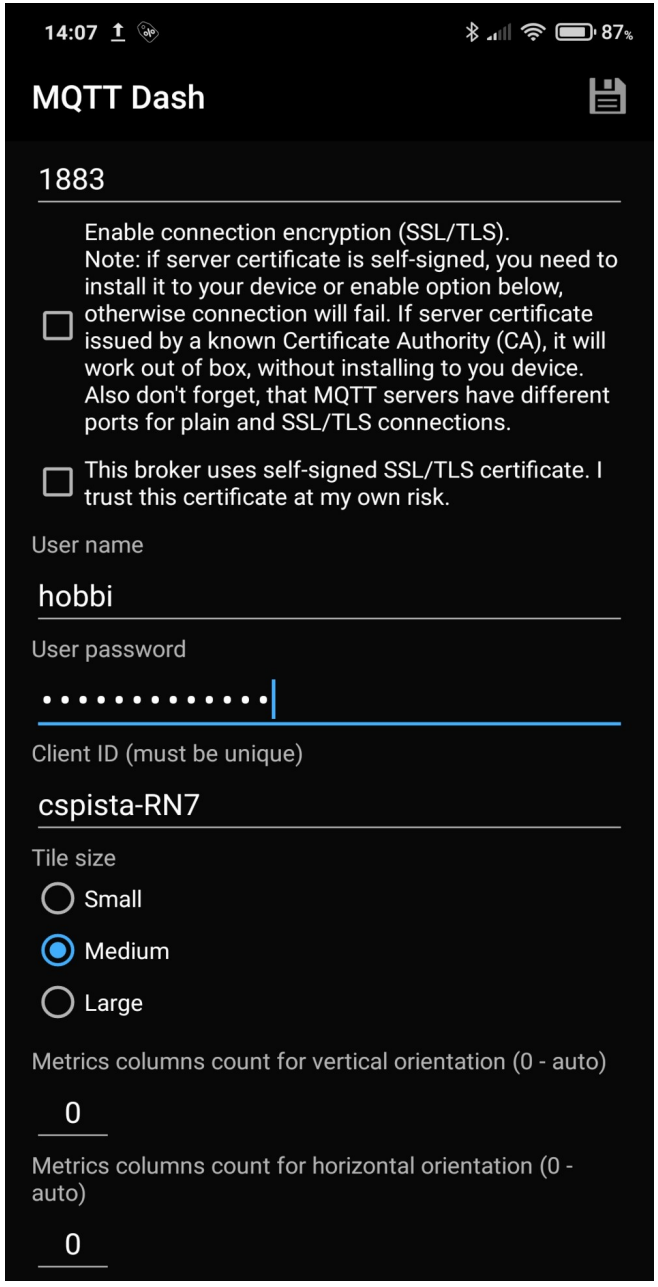
Ellenőrzés és használat MQTT Dash klienssel



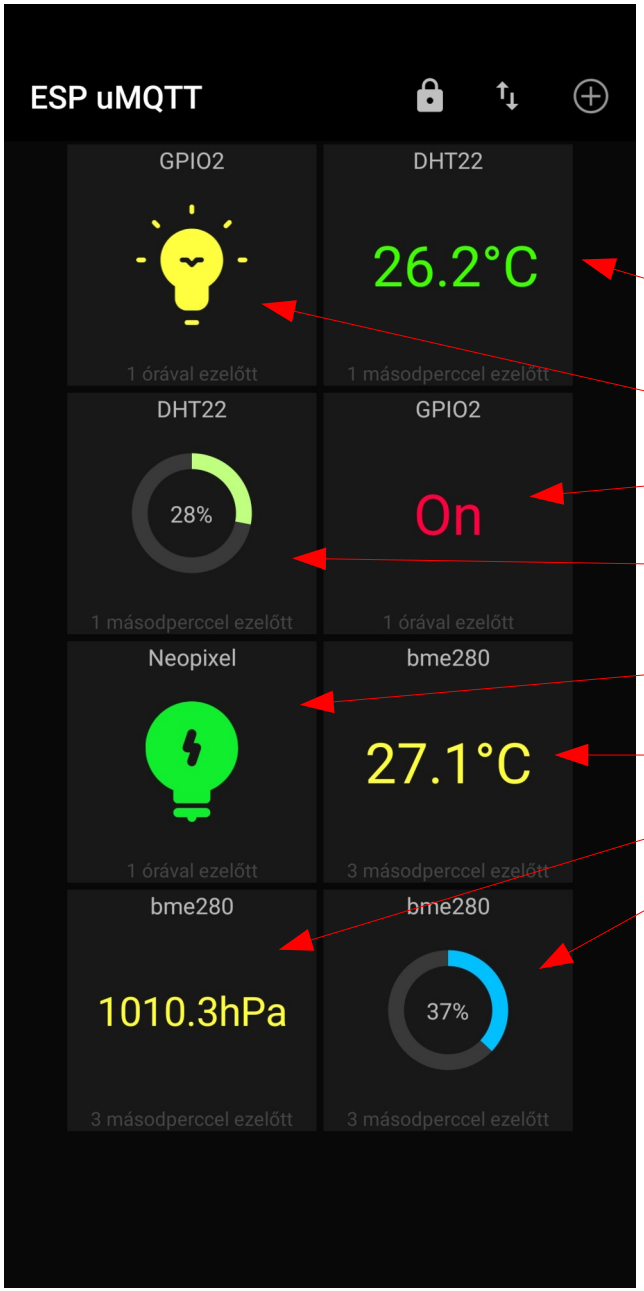
MQTT Dash konfigurálás







MQTT Dash konfigurálás



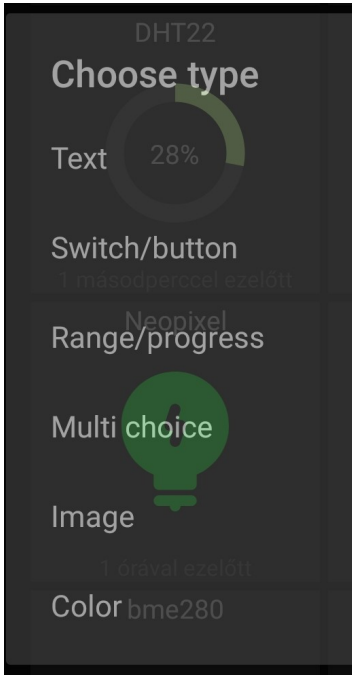
cspista/home/temp

cspista/home/led

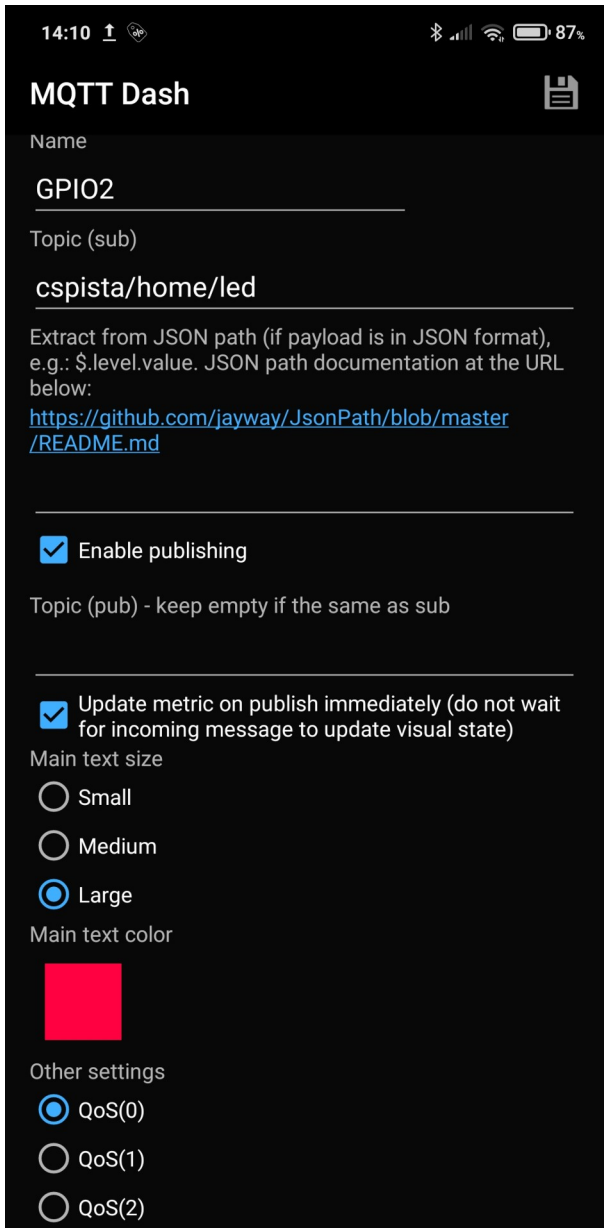
cspista/home/humidity

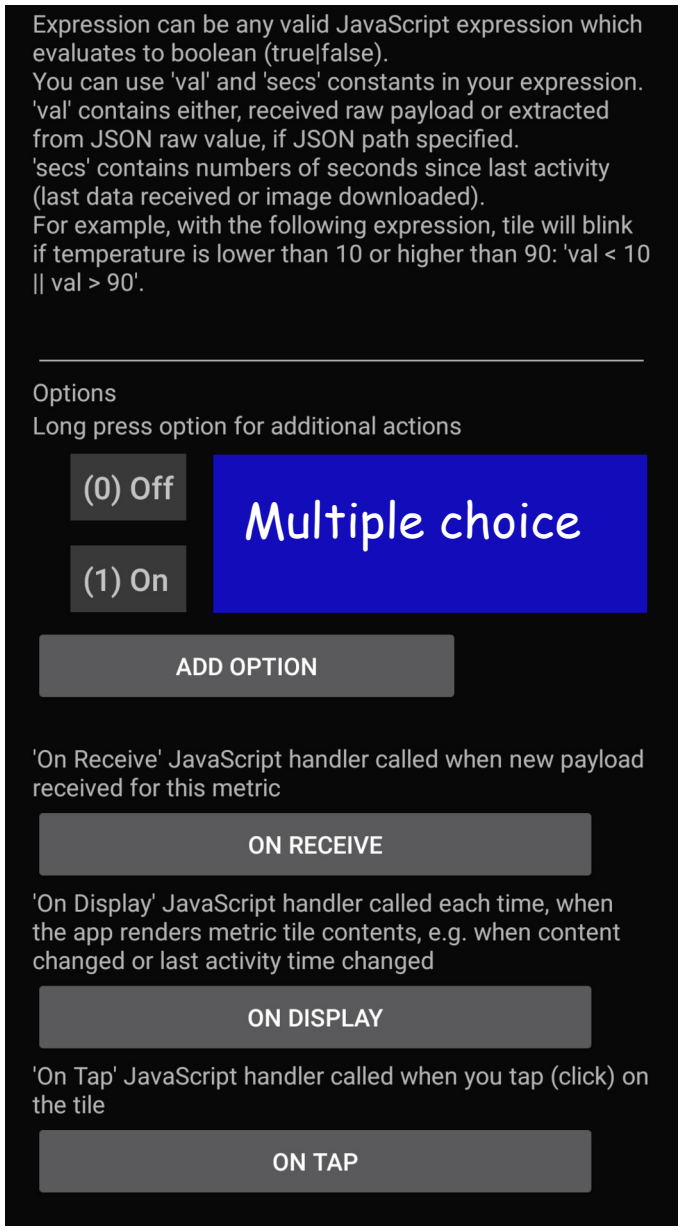
cspista/home/rgb

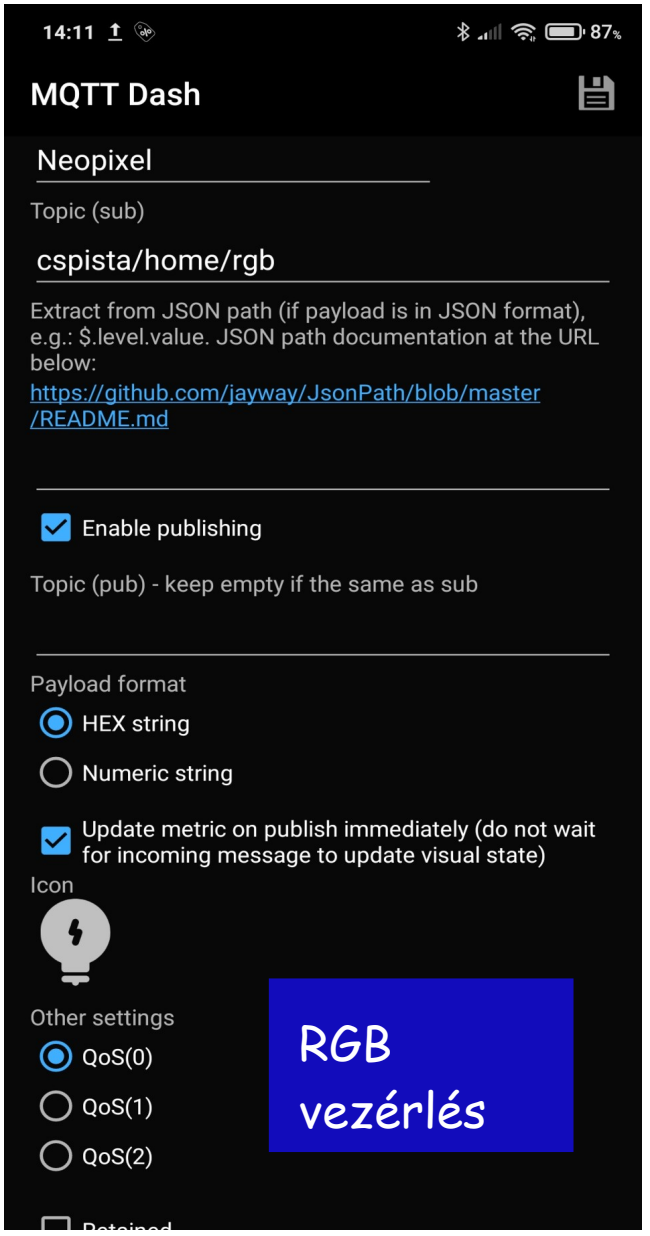
cspista/home/bme280



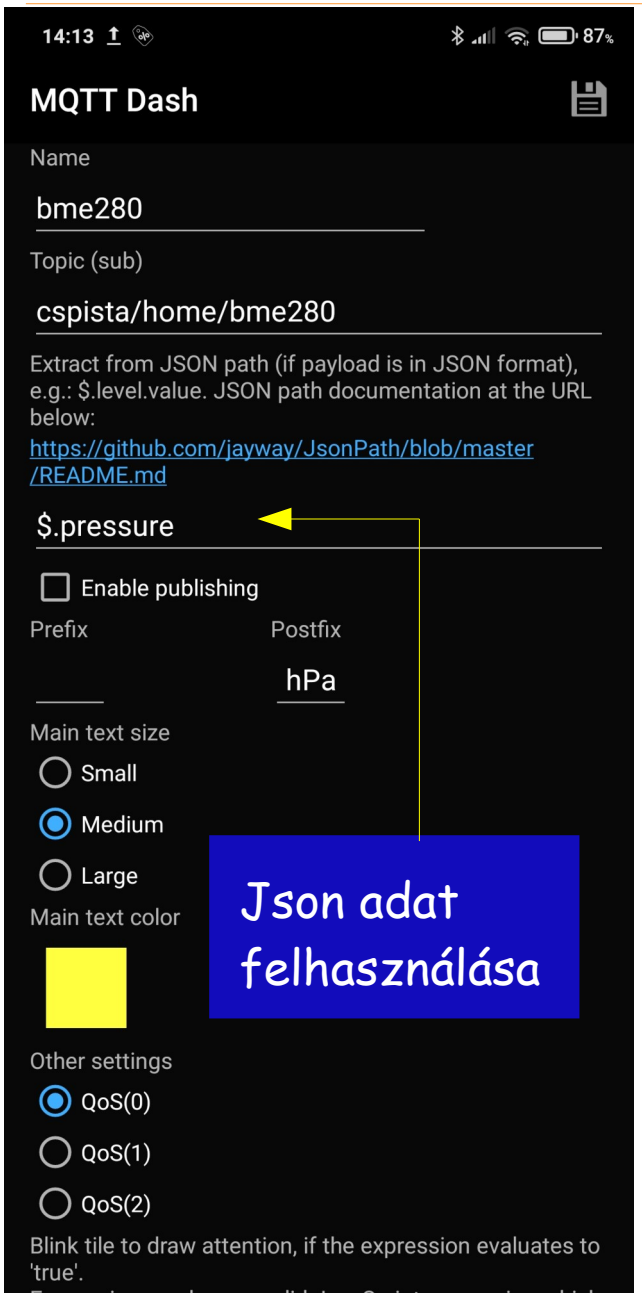
MQTT Dash konfigurálás







MQTT Dash konfigurálás

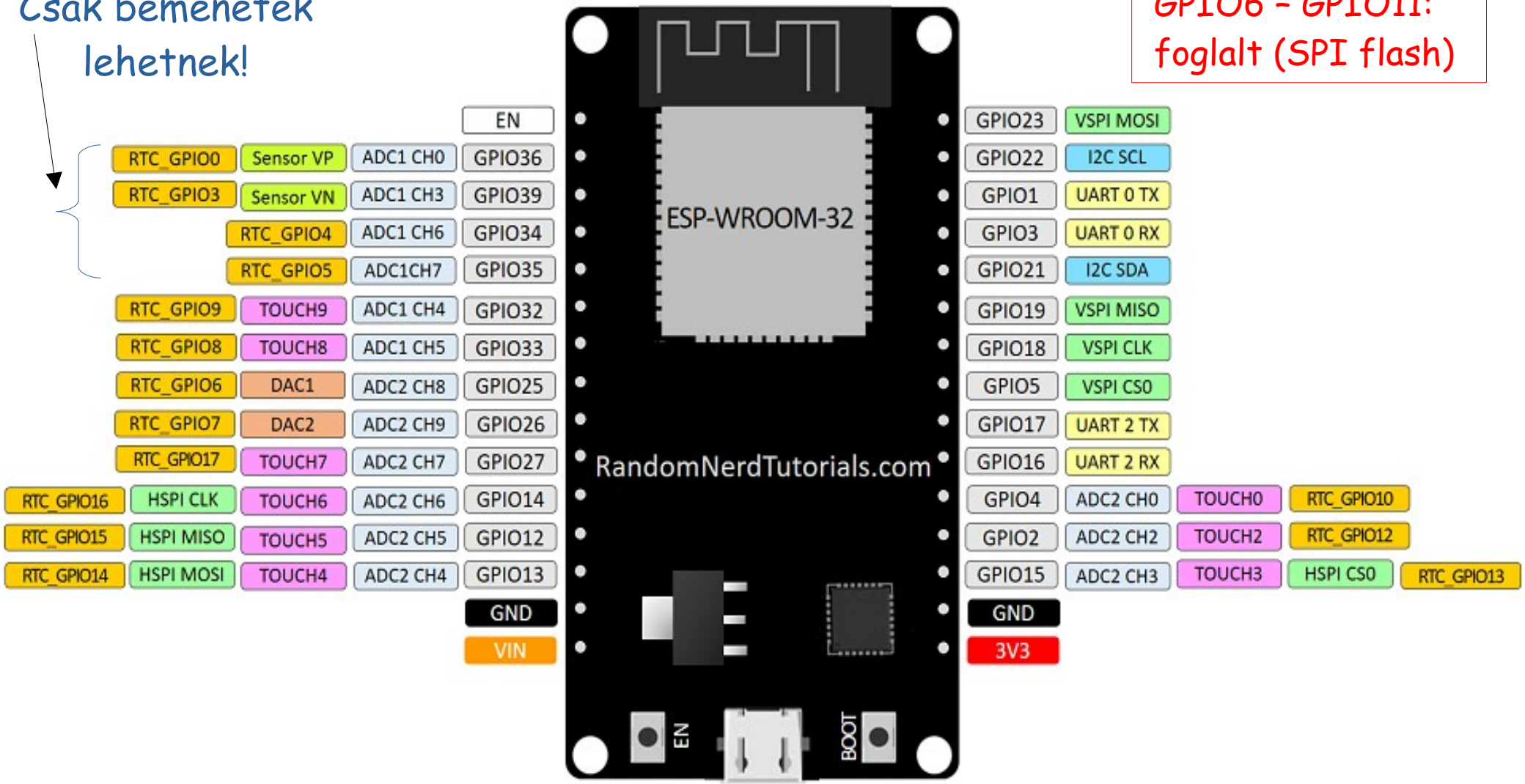


- Többszintű adatstruktúra esetén a neveket ponttal elválasztva adhatjuk meg
- A **$.** a kiindulási szintet jelenti (gyökér)

A DOIT ESP32 Devkit-1 kártya kivezetései

Csak bemenetek lehetnek!

GPIO6 - GPIO11: foglalt (SPI flash)



- **Forrás:** randomnerdtutorials.com/getting-started-with-esp32/

Ellenállás színkódok