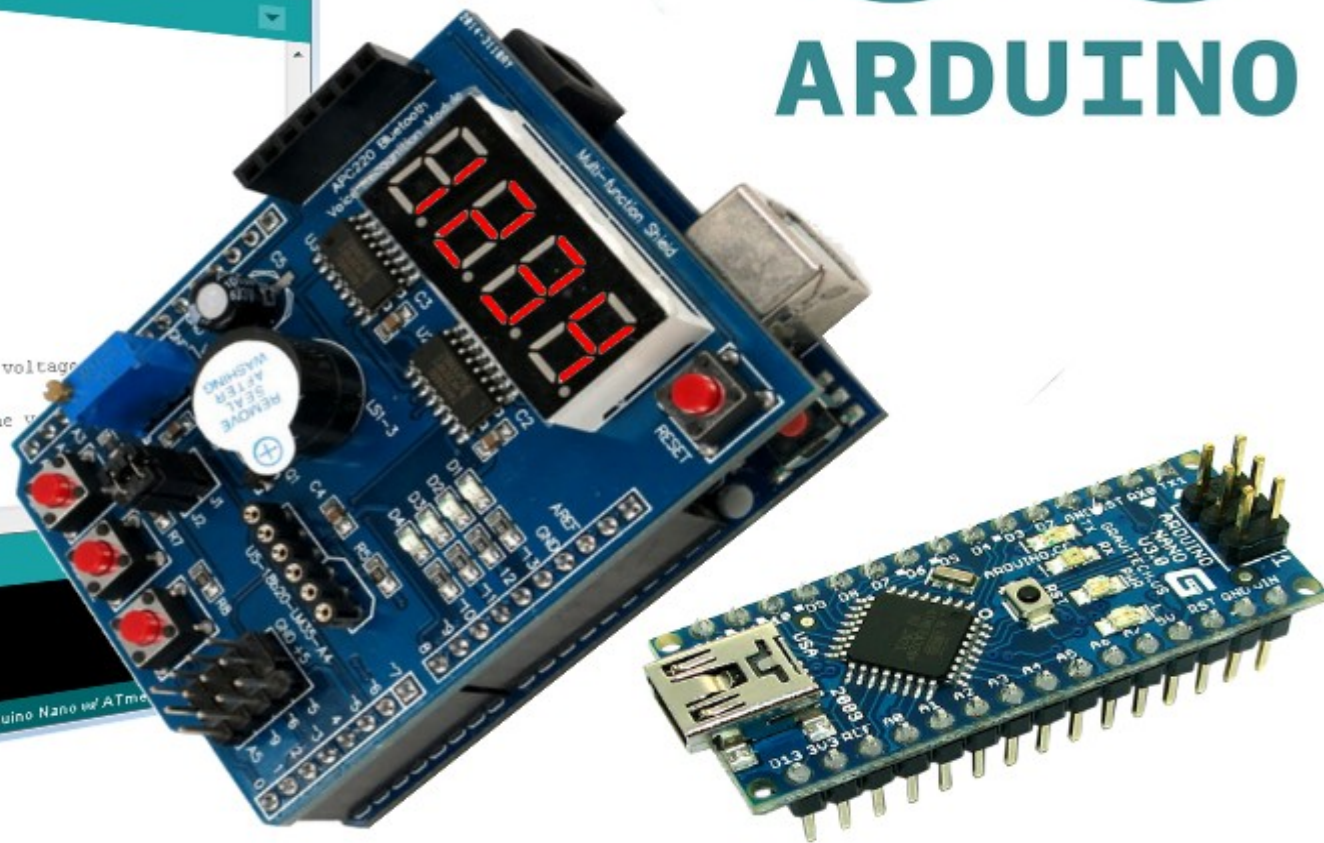
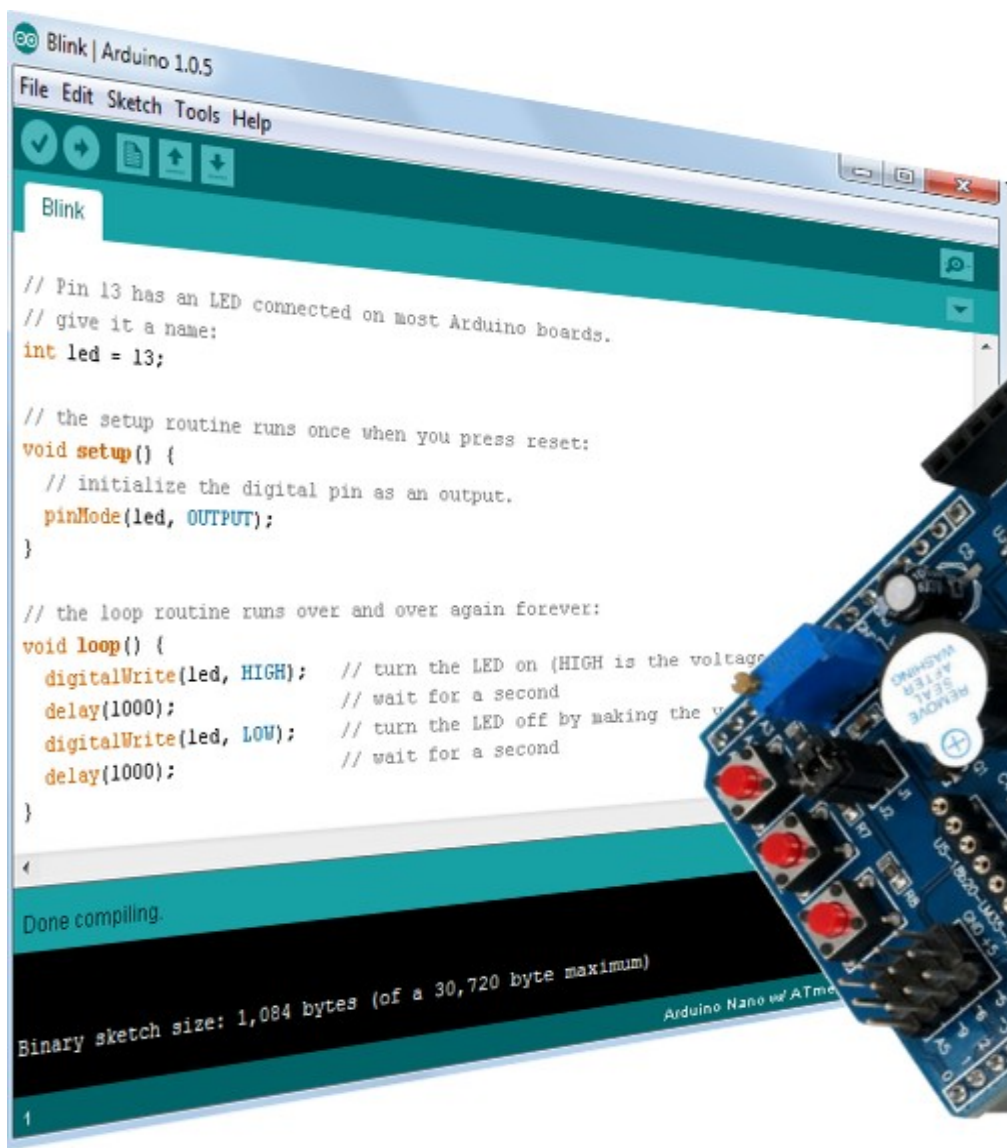


Arduino tanfolyam kezdőknek és haladóknak



14. ESP8266 webkliens alkalmazások – 2. rész

Felhasznált és ajánlott irodalom

- W3Schools.com: [HTTP Request Methods](#)
- TutorialsPoint: [HTTP requests](#)
- HiveMQ: [MQTT Essentials](#)
- ESP8266 Community: [ESP8266 Arduino Core's documentation](#)
- Rui & Sara Santos: [Random Nerd Tutorials - ESP8266 projects](#)
- Manoj R. Thakur: [NodeMCU ESP8266 Communication Methods and Protocols](#)
- Benoit Blanchon: [Mastering ArduinoJson 6](#)

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 =
"xxxxxxxxxxxx";

#define FLESPI_TOKEN
"Ws44Ev13*****rrQk0lbQGE1NuMoDnPNk1g"

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

HTTP kérések (HTTP requests)

- Egy **HTTP kliens** kéréseket küld a **szervernek**, ezek formátuma:
 - ❖ A kérés sora CRLF-fel zárva
 - ❖ Opcionális fejléc sorok, CRLF-fel zárva
 - ❖ Egy üres sor, amely a fejléc végét jelzi
 - ❖ Opcionális üzenet törzs
- A kérés sorának formátuma:
`Method szóköz Request-URI szóköz HTTP-Version CRLF`
(URI – Uniform Resource Identifier)
- **Method:**
`GET, HEAD, POST, PUT, DELETE, CONNECT, OPTIONS, TRACE`
- **GET** – információ lekérése az adott szerver adatterőforrásából
- **POST** – adatokat küldünk a szervernek (pl. űrlapok kitöltésekor)

GET vs. POST

- A **GET** kérés is küldhet adatot (ezt a **Request-URI**-ban csatolva kell megadni) ami vagy a lekérés pontosításához ad kiegészítő információt, vagy – az eredeti koncepciótól eltérve – a szervernek küld elraktározandó adatot (lásd a ThingSpeak mintapéldát az előző előadásban!)
- A **GET** kérésre egy példa:
`GET test/demo_form.php?name1=value1&name2=value2 HTTP/1.1`
- A **POST** kérés az üzenet törzsében küldi az adatokat és többnyire adatküldésre használjuk (pl. űrlapok kitöltésénél), de a GET-hez hasonló lekérésre is használhatjuk
- A **POST** kérésre egy példa:
`POST test/demo_form.php HTTP/1.1`
`Host: w3schools.com`
`Content-Type: application/x-www-form-urlencoded`
`Content-Length: length`

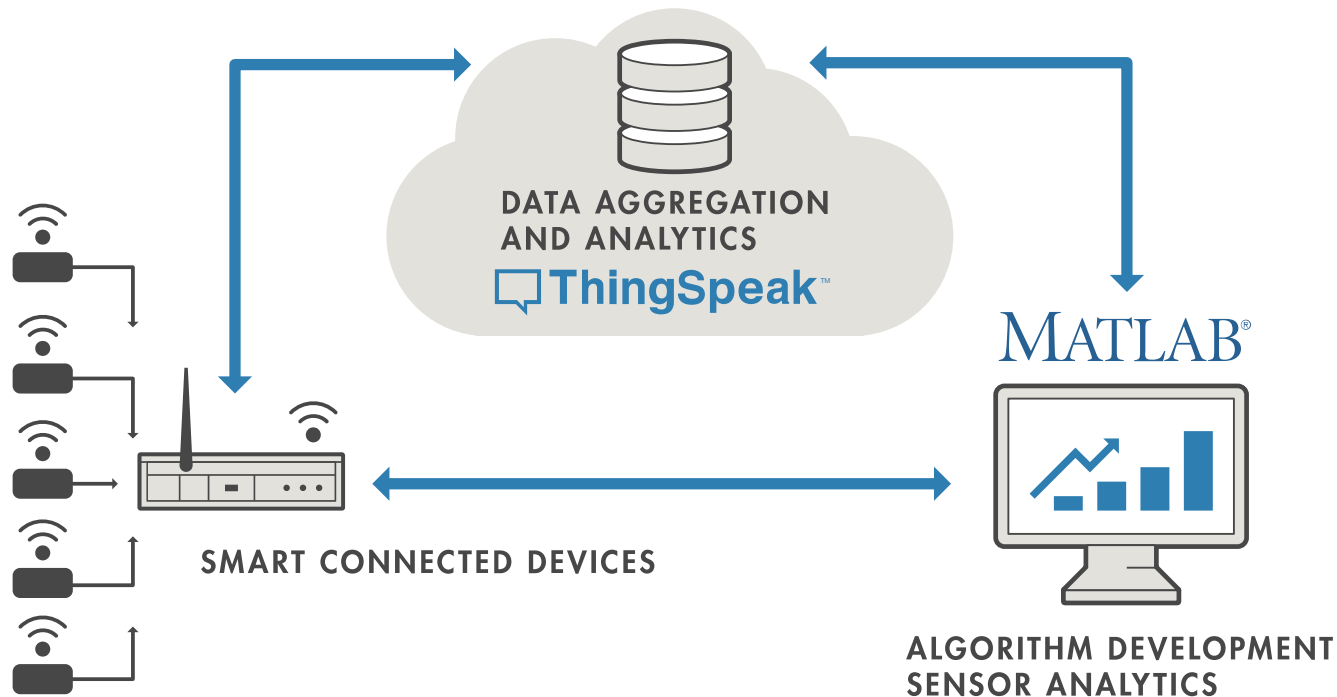
`name1=value1&name2=value2`

GET vs. POST

	GET	POST
BACK button/Reload	Harmless	Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted)
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
Encoding type	application/x-www-form-urlencoded	application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data
History	Parameters remain in browser history	Parameters are not saved in browser history
Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed
Security	GET is less secure compared to POST because data sent is part of the URL Never use GET when sending passwords or other sensitive information!	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL

Thingspeak – IoT felhő

- Mi az IoT? Internet of Things, azaz a dolgok Internetje. Különbféle szenzorok, adatgyűjtő eszközök küldhetnek adatokat privát csatornákba, melyeket a szerver tárol és megjelenít.
- Az adatok publikus vagy egyéni beállításban megtekinthetők, lekérdezhetők és akár MatLab-bal vagy számolótáblával elemezhetők, feldolgozhatók



Adatküldés HTTP GET protokollal

- Ha regisztráltunk, akkor létrehozhatunk csatornákat, amelyeknek egyedi azonosítója van. Például: thingspeak.com/channels/34244
- Csatornánként 1 – 8 mezőt definiálhatunk (pl. egy DHT22 szenzor esetén lehet field1 = hőmérséklet, field2 = páratartalom)
- Regisztráláskor kapunk egy vagy több API kulcsot. Adatot csak ennek birtokában tudunk beküldeni (xxxxxxx helyére az írás API kulcs kell!)

```
GET https://api.thingspeak.com/update?api_key=xxxxxxx&field1=adat1&field2=adat2
```

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
HTTPClient http;
String server_request = "http://api.thingspeak.com/update?api_key=DVDXXXXXXXXETD&field1=22&field2=44"
. . .
http.begin(server_request);
int httpCode = http.GET();
```

„hasból” beírt adatok



Adatküldés HTTP POST protokollal

■ Részletek egy képzeletbeli programból

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include "secrets.h"          // jelszavak, kulcsok
String body = "field1=25.7&field2=32.1";
int body_len = body.length();
HTTPClient client;
client.connect("api.thingspeak.com",80);
client.println("POST /update HTTP/1.1");
client.println("Host: api.thingspeak.com");
client.println("X-THINGSPEAKAPIKEY: *****");
client.println("Content-Type: application/x-www-form-urlencoded");
client.print("Content-Length: "); client.println(body_len);
client.println("Connection: close");
client.println();
client.println(body);
client.println();
//Wait for Server Response
while (client.available() == 0);
while (client.available()) {
    char c = client.read();
    Serial.write(c);
}
client.end();
```

- Egyszerűbb megoldás nincs?
- De igen! Használjuk inkább a
HTTPClient.POST() metódust!

ESP8266_ThingSpeak_POST.ino

- Írjuk át az előző előadásban bemutatott ThingSpeak mintaprogramját POST metódusra!

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include "DHT.h"           // Adafruit DHT library
#include "secrets.h"       // Wifi jelszó, ThingSpeak Write API key

#define DHTPIN             14    // GPIO14 (D5) pin a DHT szenzorhoz
DHT dht(DHTPIN, DHT22);
float t = 0.0;              // Hőmérséklet
float h = 0.0;              // Páratartalom

const char* serverName = "http://api.thingspeak.com/update";
unsigned long lastTime;
unsigned long timerDelay = 20000;
String apiKey = String("api_key=") + THINGSPEAK_WRITE_APIKEY;

void setup() {
  Serial.begin(115200);     // Soros port inicialálás
  dht.begin();             // DHT szenzor indítása
  setup_wifi();            // Kapcsolódás a WiFi hálózatra
  lastTime = millis();
}
```

ESP8266_ThingSpeak_POST.ino

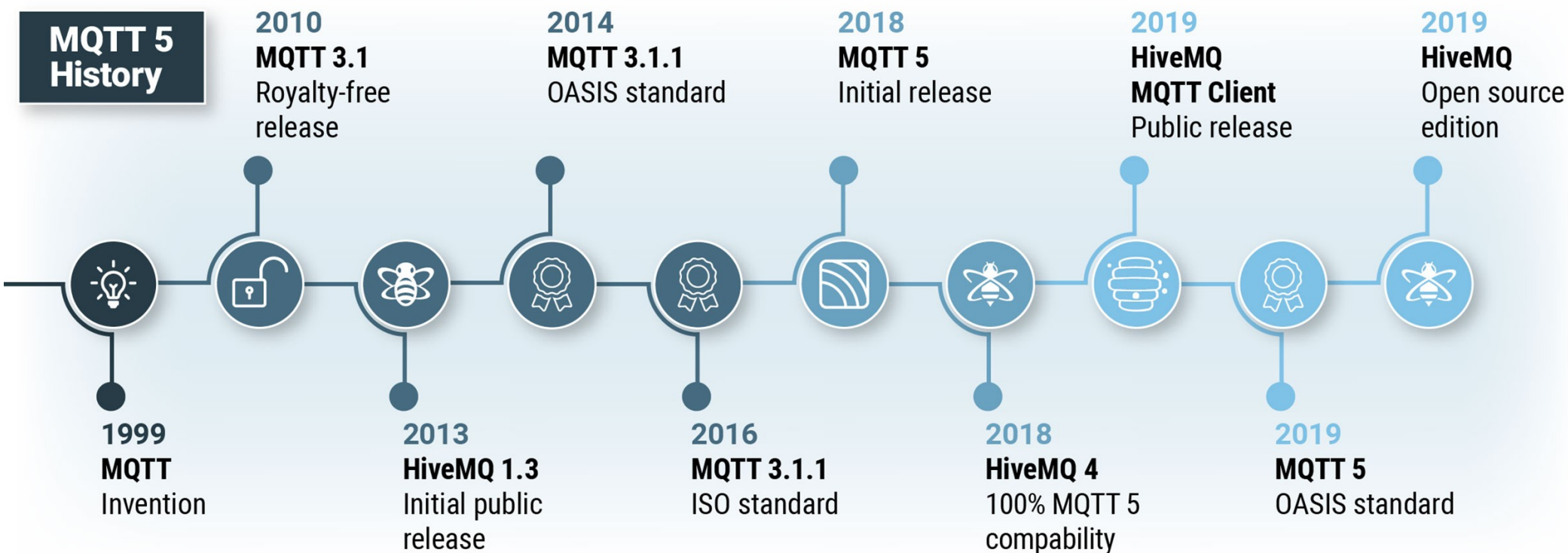
```
void loop() {
  if (millis() > lastTime) { // Ha letelt az idő ...
    float tnew = dht.readTemperature(); // Hőmérséklet kiolvasása
    if (!isnan(tnew)) t = tnew;
    float hnew = dht.readHumidity(); // Páratartalom kiolvasása
    if (!isnan(hnew)) h = hnew;
    String p1 = String("&field1=") + String(t, 1);
    String p2 = String("&field2=") + String(h, 1);
    Serial.println(p1 + p2);

    if (WiFi.status() == WL_CONNECTED) { // Ha a WiFi hálózat elérhető
      HTTPClient http; // Webkliens példányosítása
      http.begin(serverName); // Szerver név megadása
      http.addHeader("Content-Type", "application/x-www-form-urlencoded");
      String httpRequestData = apiKey + p1 + p2; // HTTP request adatai
      int httpResponseCode = http.POST(httpRequestData); // adatok kiküldése
      Serial.print("HTTP Response code: "); // Válaszkód kiírása
      Serial.println(httpResponseCode); // Sikeres küldés esetén 200-as kód
      http.end(); // Erőforrások felszabadítása
    }
    else {
      Serial.println("WiFi Disconnected");
    }
    lastTime += timerDelay; // A következő eseményidő előjegyzése
  }
}
```

MQTT – MQ Telemetry Transport protokoll

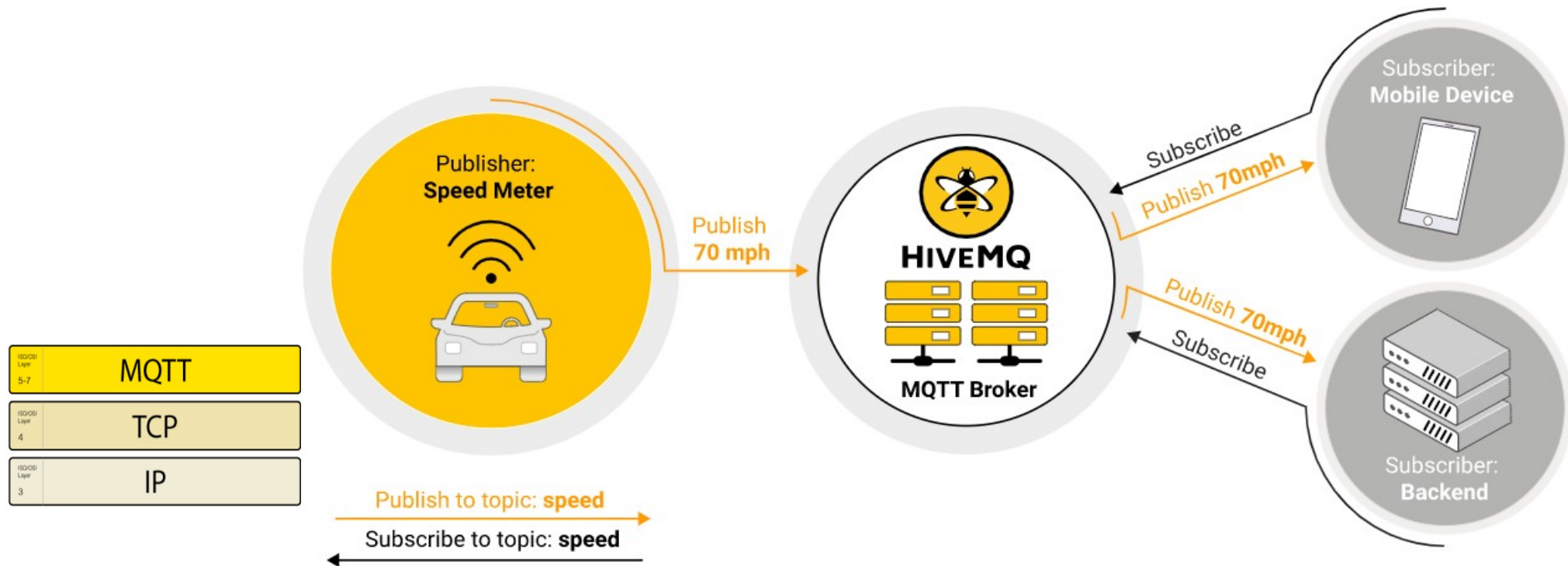
- Eredetileg az **IBM** szakemberei dolgozták ki ezt a protokollt telemetriai adatok továbbítására, amit 2010-ben nyilvánossá tettek
- **OASIS** - Organization for the Advancement of Structured Information Standards – egy globális nonprofit konzorcium, amely oroszlánrészt vállalt az **MQTT** protokoll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

MQTT 5 History



MQTT alapok

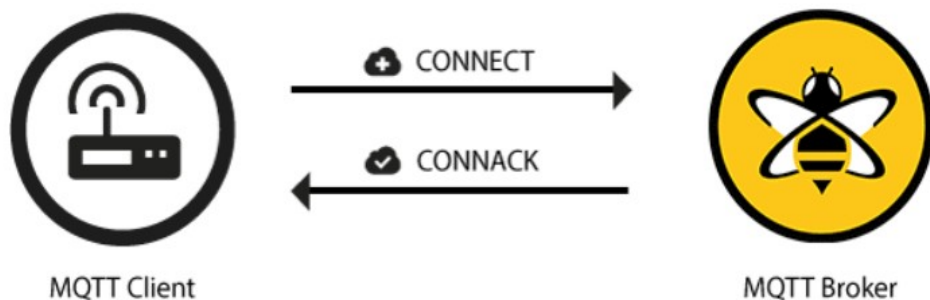
- A **publish/subscribe** séma a hagyományos kliens/szerver sémától eltérően elkülöníti az adatküldő klienst (**publisher**) azoktól a kliensektől amelyek fogadják az üzeneteket (**subscribers**).
- A küldő és fogadó kliensek sohasem kerülnek közvetlen kapcsolatba. A kapcsolatot egy harmadik komponens,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

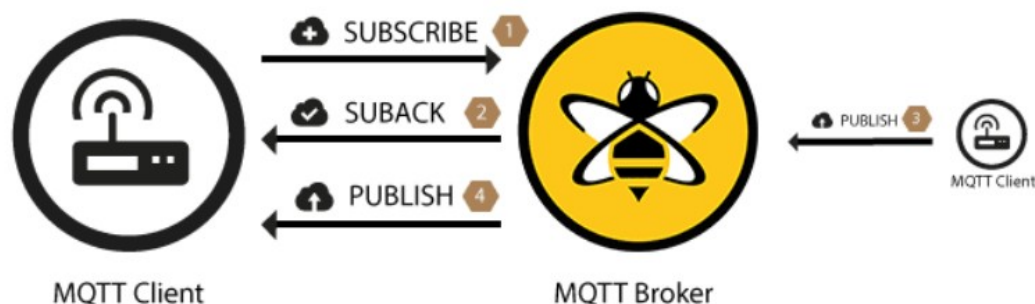
- Sikeres kapcsolódás után a kliens **adatokat küldhet** (publish) a brókernek
- Minden üzenetnek tartalmaznia kell egy **témakör** (topic) megnevezését, amely alapján a bróker szétküldi az adatot mindazon klienseknek, amelyek feliratkoztak (subscribe) az adott témakörre
- **topicName** - egyszerű karakterfüzér, 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

MQTT-Packet:	
PUBLISH 	
contains:	Example
packetId (always 0 for qos 0)	4314
topicName	"topic/1"
qos	1
retainFlag	false
payload	"temperature:32.5"
dupFlag	false

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

MQTT Subscribe - feliratkozás

- A kliens **feliratkozhat** (subscribe) témakörök adatainak 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 } (list of topic + qos)	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

MQTT témakörök

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



- Helyettesítő karakterek:

- ❖ Egyszintű: +



- ✓ myhome / groundfloor / livingroom / temperature
- ✓ myhome / groundfloor / kitchen / temperature
- ✗ myhome / groundfloor / kitchen / brightness
- ✗ myhome / firstfloor / kitchen / temperature
- ✗ myhome / groundfloor / kitchen / fridge / temperature

- ❖ Többszintű: #



- ✓ myhome / groundfloor / livingroom / temperature
- ✓ myhome / groundfloor / kitchen / temperature
- ✓ myhome / groundfloor / kitchen / brightness
- ✗ myhome / firstfloor / kitchen / temperature

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_mqtt.ino

- Ez a program a **PubSubClient** programkönyvtár mintapéldája, apró módosításokkal
- A bróker: **mqtt.flespi.io** (Gmail accounttal be lehet lépni) az API token a *secrets.h* állományban **FLESPI_TOKEN** néven definiáljuk
- **Publish**: 2 s-onként küldünk az **outTopic** témakörbe egy "hello world #*n*" üzenetet
- **Subscribe**: feliratkozunk az *inTopic* témakörre és a *callback* nevű visszahívási függvényben várjuk az üzeneteket
- Ha a beérkező üzenet első karaktere '1', akkor felkapcsoljuk, ha pedig '0', akkor lekapcsoljuk a beépített LED-et (**GPIO2**)
- A **loop()** ciklusban ellenőrizzük, hogy a kliens kapcsolódik-e a brókerhez. Ha nem, akkor újra kapcsolódunk a **reconnect()** függvényben kiadott **connect()** fv. hívással

ESP8266_mqtt.ino

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include "secrets.h"
const char* mqtt_server = "mqtt.flespi.io";

WiFiClient espClient;
PubSubClient client(mqtt_server, 1883, espClient);
unsigned long lastMsg = 0;
#define MSG_BUFFER_SIZE (50)
char msg[MSG_BUFFER_SIZE];
int value = 0;
int nClient = 0;

void setup() {
  pinMode(BUILTIN_LED, OUTPUT);
  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
  }
}
```

Megjegyzés:

Az itt bemutatott listából a jobb áttekinthetőség érdekében kihagytuk a soros porti kiírásokat!

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

ESP8266_mqtt.ino

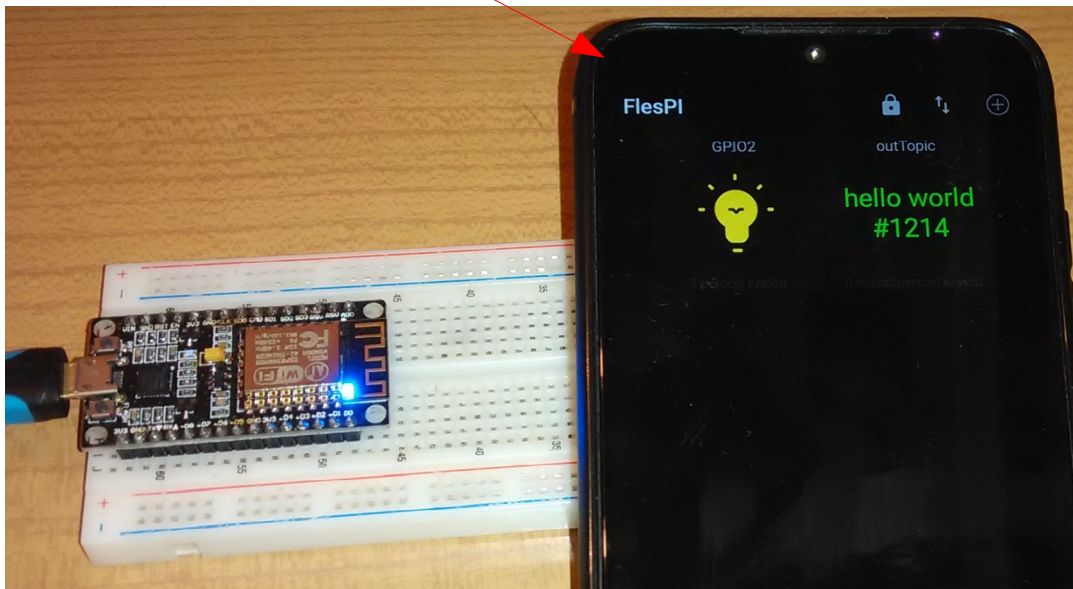
```
void reconnect() {
  while (!client.connected()) {           // Loop until we're reconnected
    String clientId = "ESP8266Client-";
    nClient++;
    clientId += String(nClient, HEX);     // Create a client ID
    if (client.connect(clientId.c_str(), FLESPi_TOKEN, "" )) { // Attempt to connect
      client.publish("outTopic", "hello world"); // Once connected, publish...
      client.subscribe("inTopic");          // ... 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 > 2000) {
    lastMsg = now;
    ++value;
    sprintf (msg, MSG_BUFFER_SIZE, "hello world %ld", value);
    client.publish("outTopic", msg);
  }
}
```

ESP8266_mqtt.ino: futási eredmény

- A program működését a soros porti terminál ablakon kívül különféle online, vagy offline (Windows, Android) MQTT kliens segítségével is nyomon követhetjük, mint pl.
- Flespi.io - MQTT BOARD (online)
- MQTT Explorer (Windows)
- MQTT Dash (Android)



COM10 - terminál ablak

```
..WiFi connected
IP address: 192.168.1.106
Attempting MQTT connection...connected
Publish message: hello world #1
Publish message: hello world #2
Publish message: hello world #3
Publish message: hello world #4
Publish message: hello world #5
Publish message: hello world #6
Publish message: hello world #7
Publish message: hello world #8
Publish message: hello world #9
Attempting MQTT connection...connected
Publish message: hello world #10
Publish message: hello world #11
Publish message: hello world #12
Message arrived [inTopic] 1
Publish message: hello world #13
Publish message: hello world #14
Publish message: hello world #15
Message arrived [inTopic] 0
```

ESP8266_mqtt.ino: futási eredmény

- A **flespi.io** honlapon bejelentkezve az **MQTT Board** komponens segítségével egyszerűen kipróbálhatjuk az eszköznket
- A **+** gombra kattintva adhatunk hozzá **Publish**, ill. **Subscribe** klienst

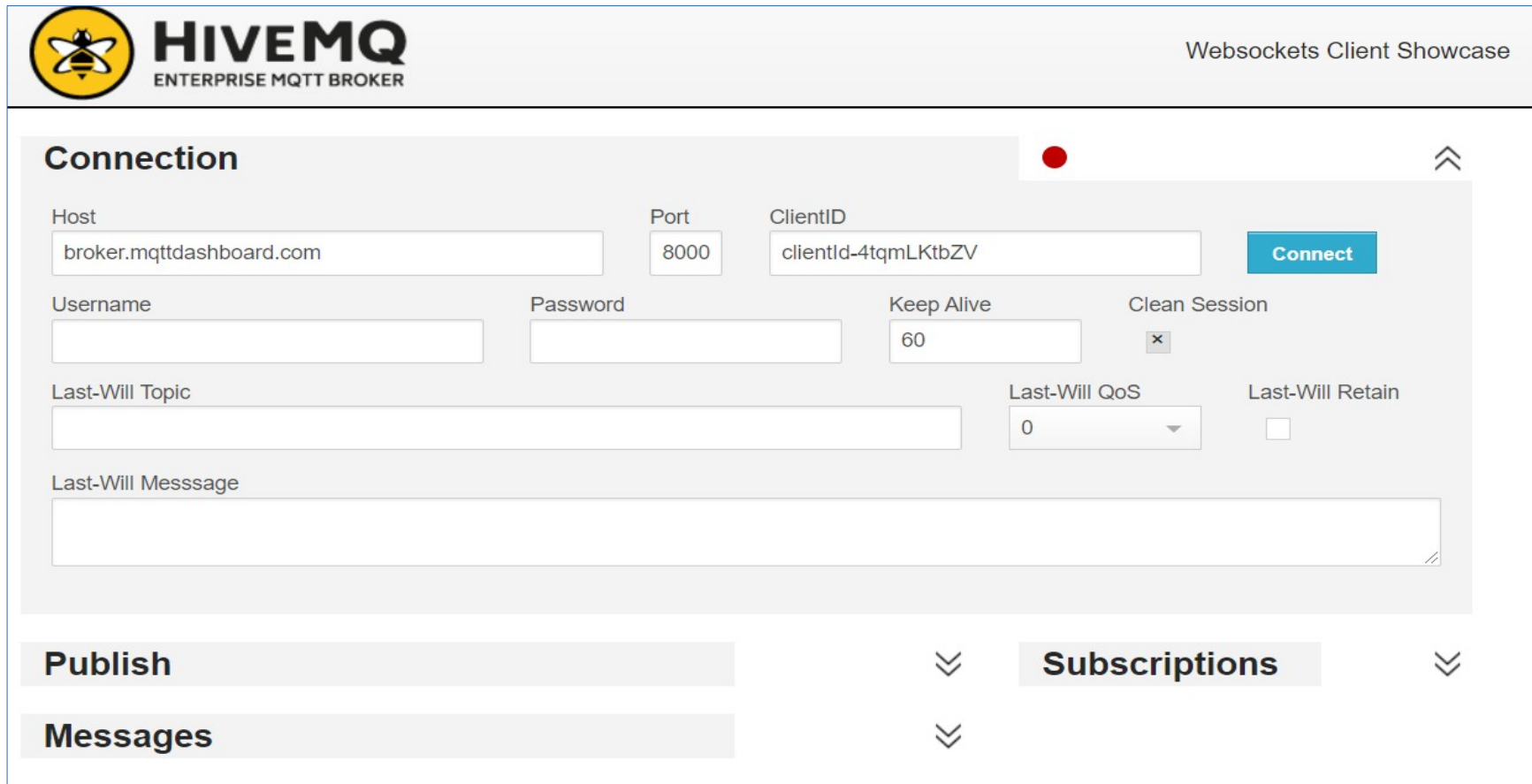
The screenshot displays the MQTT Board interface. At the top, there's a navigation bar with 'MQTT BOARD' and a close button. Below it, the 'mqtt-board-panel-23603a56' is shown as 'online' with ID '894061'. The main area is divided into three panes:

- Publisher (left):** Shows 'inTopic' as the topic and '1' as the message. Options include QoS (0, 1, 2), Retain, and Duplicate flag. Properties are also visible.
- outTopic (middle):** Displays a list of published messages. Each entry shows the topic 'outTopic', QoS, dup, retain, timestamp, and the message content 'hello world #411' through '#414'. Each entry also includes user properties like 'timestamp' and 'cid'.
- Subscriber (right):** Shows 'outTopic' as the subscribed topic.

At the bottom right, there is a 'REMOVE ALL' button. An orange arrow points to the '+' icon in the top right corner of the interface.

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/>



The screenshot shows the HiveMQ Websockets Client Showcase interface. At the top left is the HiveMQ logo (a bee) and the text "HIVEMQ ENTERPRISE MQTT BROKER". At the top right is the text "Websockets Client Showcase". The main content area is titled "Connection" and contains several input fields and a "Connect" button. The fields are: Host (broker.mqttdashboard.com), Port (8000), ClientID (clientId-4tqmLKtbZV), Username, Password, Keep Alive (60), Clean Session (checkbox), Last-Will Topic, Last-Will QoS (0), Last-Will Retain (checkbox), and Last-Will Message. Below the connection fields are three expandable sections: "Publish", "Subscriptions", and "Messages".

ESP8266_HiveMQ.ino

- Az előző programot módosítottuk az alábbiakkal:
- A bróker neve: `broker.hivemq.com` a belépéshez szükséges `HIVEMQ_USER` és `HIVEMQ_PASS` a `secrets.h` állományban definiált
- A publikáláshoz választott témakör neve: `cspista/home/temp`
- 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: `hobbi@cspista.hu`
- A program a `cspista/home/temp` témakörbe most csak egy fix szöveget publikál (25.7 °C), amit később majd valódi mérési adattal helyettesítünk
- A témakörre `cspista/home/led` feliratkozva, a beérkező üzenetek első karakterét figyeljük, s ugyanúgy vezéreljük a LED-et, mint az előző programban az `inTopic` üzenetekkel

ESP8266_HiveMQ.ino

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include "secrets.h"
WiFiClient espClient;
PubSubClient client(espClient);
const char* mqtt_server = "broker.hivemq.com";
unsigned long lastMsg = 0;
#define MSG_BUFFER_SIZE (50)
char msg[MSG_BUFFER_SIZE];
int value = 0;
int nClient = 0;

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

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 by making the voltage HIGH
  }
}
```

ESP8266_HiveMQ.ino

```
void reconnect() {
  while (!client.connected()) { // Loop until we're reconnected
    char clientId[] = "hobbi@cspista.hu";
    if (client.connect(clientId, HIVEMQ_USER, HIVEMQ_PASS )) { // Attempt to connect
      client.publish("cspista/home/temp", "25 °C"); // Once connected, publish...
      client.subscribe("cspista/home/led"); // ... and resubscribe
    } else delay(5000); // Wait 5 seconds before retrying
  }
}

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

void loop() {
  if (!client.connected()) reconnect();
  client.loop(); ← Rendszeresen meg kell hívni!
  unsigned long now = millis();
  if (now - lastMsg > 5000) {
    lastMsg = now;
    sprintf (msg, MSG_BUFFER_SIZE, "25.7 °C");
    client.publish("cspista/home/temp", msg);
  }
}
```

Próba a HiveMQ MQTT Broker-rel

- A kipróbáláshoz kapcsolódjunk a HiveMQ nyilvános webklienssel a <http://www.hivemq.com/demos/websocket-client/> címre látogatva!
- A képen látható hostnév vagy a `broker.hivemq.com` egyaránt jó, a port a webes hozzáférés miatt itt **8000**, a clientID tetszőleges
- A belépési név és jelszó beírása után a **Connect** gombbal kapcsolódjunk!

Connection

● disconnected

Host

broker.mqttdashboard.com

Port

8000

ClientID

cspista-001

Connect

Username

icserny

Password

.....

Keep Alive

60

Clean Session

×

Last-Will Topic

Last-Will QoS

0

Last-Will Retain

Last-Will Message

Próba a HiveMQ MQTT Broker-rel



Websockets Client Showcase

Connection

● connected



Publish



Topic

QoS

0

Retain



Publish

Message

Subscriptions



Add New Topic Subscription

Qos: 2



cspista/home/temp

Messages



2021-04-14 07:36:45

Topic: cspista/home/temp

Qos: 0

25.7 °C

2021-04-14 07:36:40

Topic: cspista/home/temp

Qos: 0

25.7 °C

2021-04-14 07:36:35

Topic: cspista/home/temp

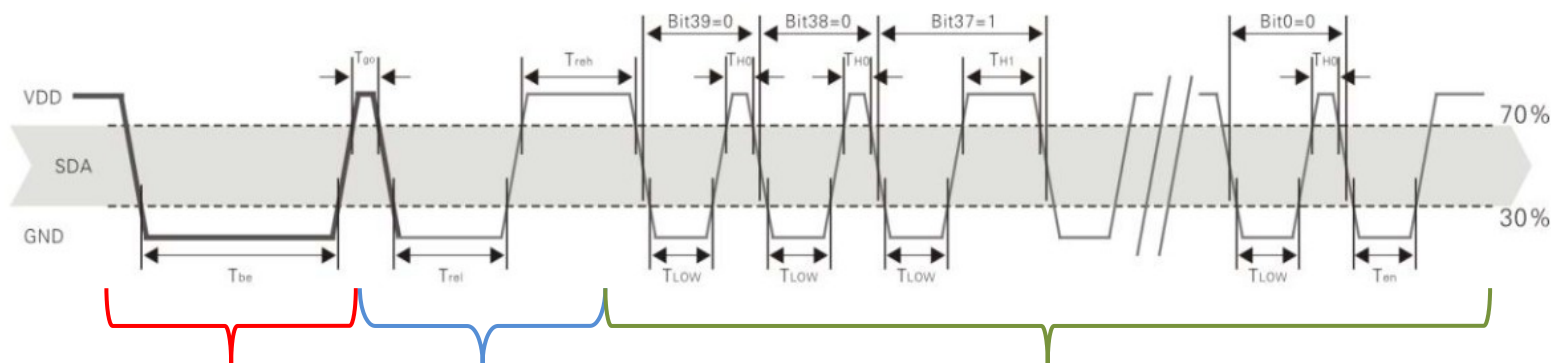
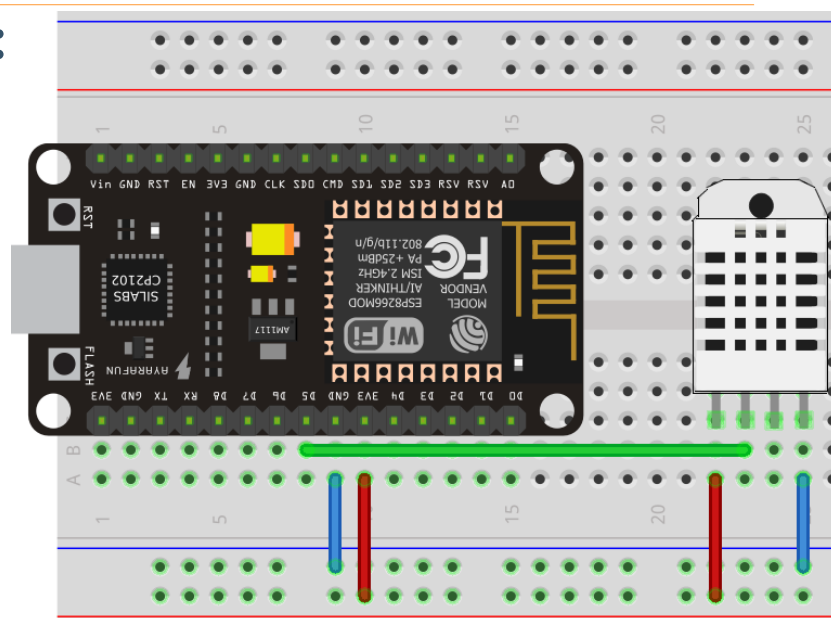
Qos: 0

25.7 °C

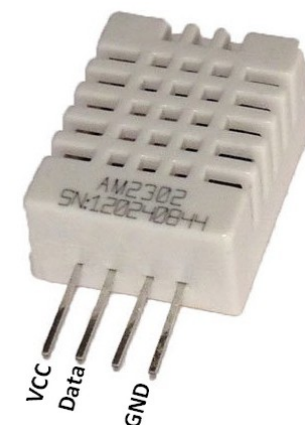
Hőmérséklet és páratartalom mérés DHT22 szenzorral

Az Am2302 (DHT22) szenzor jellemzői:

- ❖ Hőmérséklet és rel. páratartalom mérésére
 - ❖ Felbontás: $0.1\text{ }^{\circ}\text{C}$, illetve 0.1%
 - ❖ 1-wire, nem szabványos protokoll
 - ❖ Mintavételezési gyakoriság: 2 s
 - ❖ Tápfeszültség: $3,3 - 6\text{ V}$
- Programkönyvtár: [Adafruit_DHT](https://www.adafruit.com/library/dht)
 - Adatlap: [sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf](https://www.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 ...



ESP8266_HiveMQ_DHT22.ino

- Az előző programot bővítjük tovább
- A publikáláshoz két témakör használunk:
 - ❖ `cspista/home/temp`
 - ❖ `cspista/home/humidity`
- Változás, hogy az adatoknak csak a mérőszámát írjuk ki, hogy numerikus adatként lehessen kezelni
- Kapcsolódáskor vagy újrakapcsolódáskor a `cspista/home` témakörbe küldünk egy "DHT22 connected" üzenetet
- A feliratkozásra (és a **GPIO2**-re kötött LED vezérlésére) használt témakör neve változatlanul:
 - ❖ `cspista/home/led`

ESP8266_HiveMQ.ino

```
#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);
  }
}
```

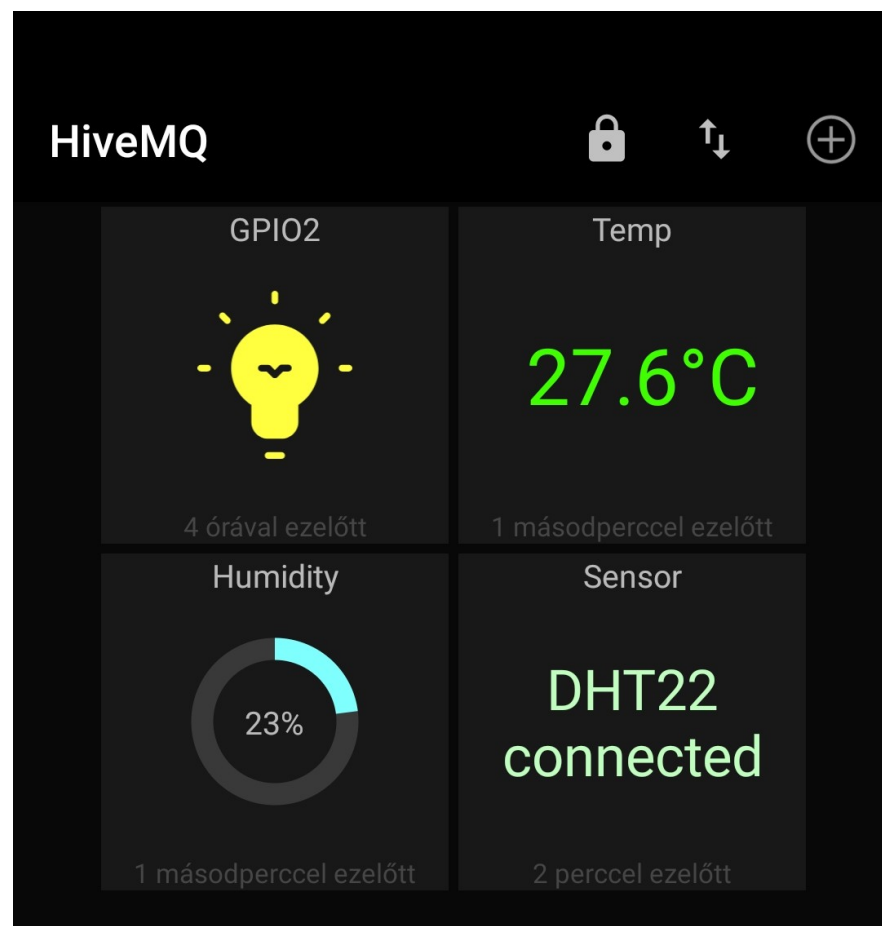

ESP8266_HiveMQ.ino

```
void reconnect() {
  while (!client.connected()) { // Loop until we're reconnected
    char clientId[] = "hobbi@cspista.hu";
    if (client.connect(clientId, HIVEMQ_USER, HIVEMQ_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_HiveMQ.ino: futási eredmények

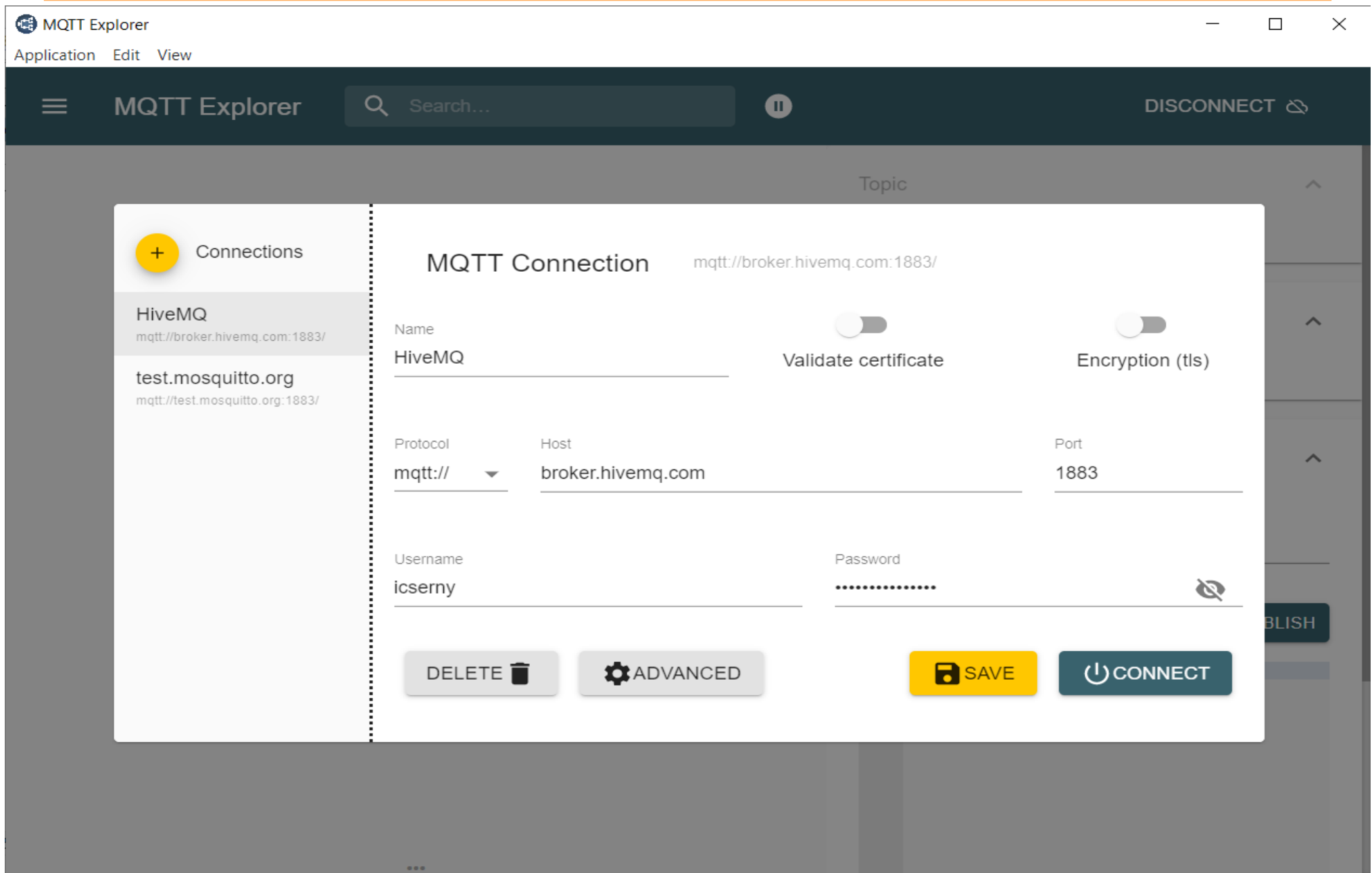
```
COM10 - terminál ablak
COM10
IP address: 192.168.1.103
Attempting MQTT connection...connected
Publish [cspista/home/temp]: 26.5
Publish [cspista/home/humidity]: 27
Publish [cspista/home/temp]: 26.5
Publish [cspista/home/humidity]: 25
Message arrived [cspista/home/led] : 0
Publish [cspista/home/temp]: 26.5
Publish [cspista/home/humidity]: 26
Publish [cspista/home/temp]: 26.5
Publish [cspista/home/humidity]: 26
Publish [cspista/home/temp]: 26.5
Publish [cspista/home/humidity]: 26
Message arrived [cspista/home/led] : 1
Publish [cspista/home/temp]: 26.5
Publish [cspista/home/humidity]: 26
Publish [cspista/home/temp]: 26.5
Publish [cspista/home/humidity]: 26
Publish [cspista/home/temp]: 26.5
```



MQTT Dash (Android)

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

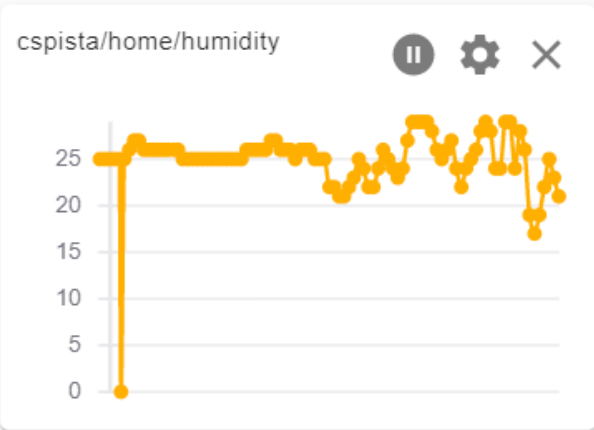
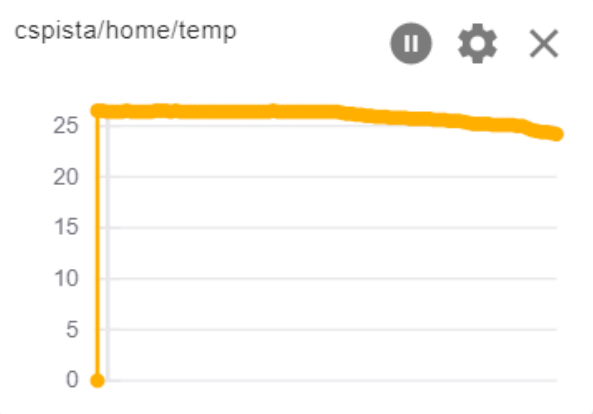
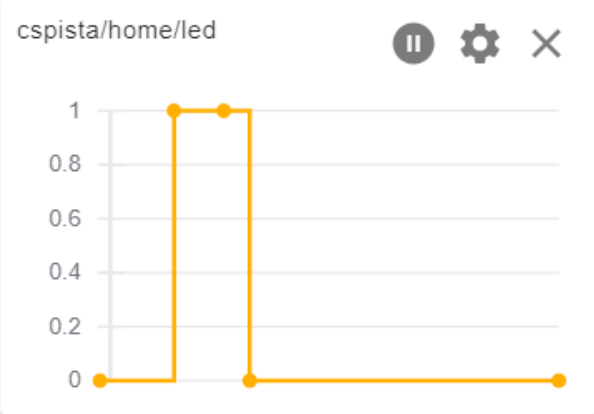
MQTT Explorer (Windows alkalmazás)





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

Publish
Topic: cspista/home/humidity
raw xml json
PUBLISH

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

A Flash memóriát kezelő kivezetések foglaltak!

