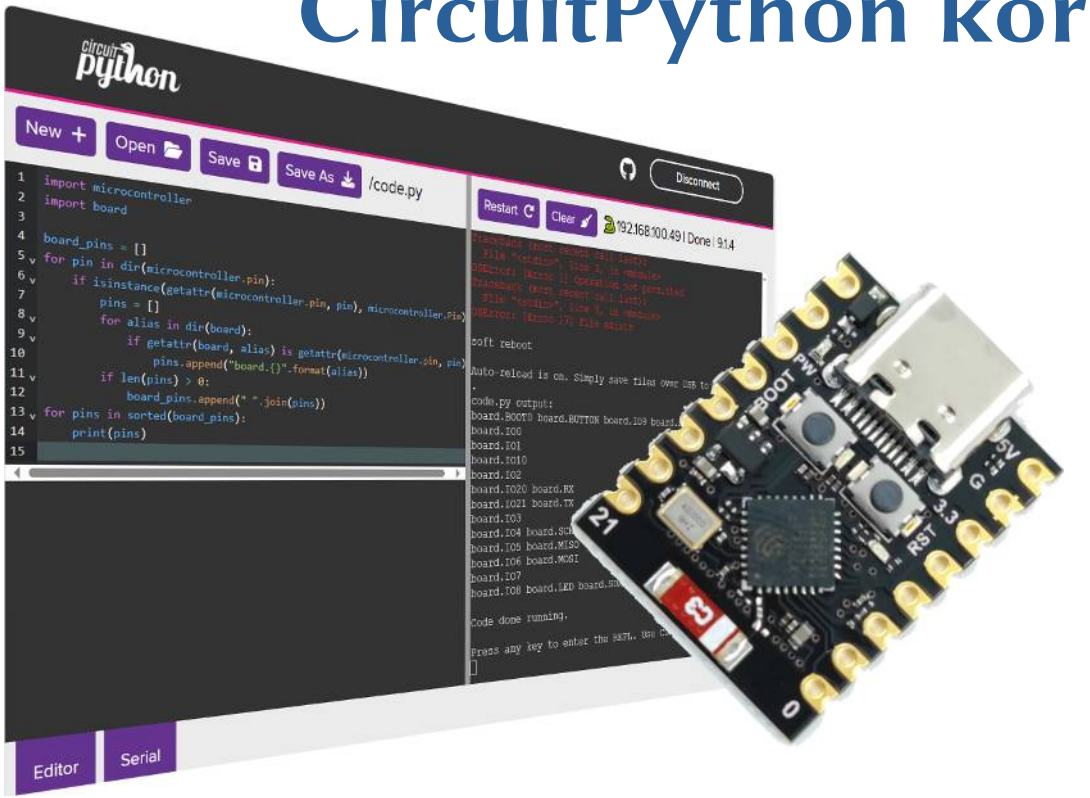
ESP32-C3 mikrovezérlők programozása CircuitPython környezetben



9. MQTT alapok: egyszerű IoT kommunikáció

Felhasznált és ajánlott irodalom

- **Python:**
  - Mark Pilgrim/Kelemen Gábor: Ugorj fejest a Python 3-ba!
  - P. Wentworth et al. (ford. Biró Piroska, Szeghalmy Szilvia és Varga Imre): Hogyan gondolkozz úgy, mint egy informatikus: Tanulás Python 3 segítségével
- **CircuitPython:**
  - Adafruit: https://circuitpython.org/downloads
  - Learn Adafruit: Welcome to CircuitPython
  - Learn Adafruit: CircuitPython Essentials
  - Adafruit: Adafruit CircuitPython API Reference
  - Adafruit: github.com/adafruit/Adafruit CircuitPython Bundle

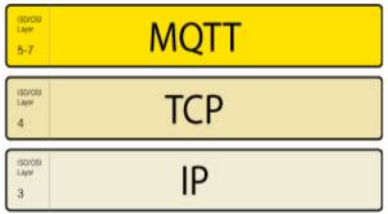
- **Online eszközök és támogatás:**
  - Learn Adafruit: CircuitPython on ESP32 Quick Start
  - Adafruit: Adafruit Web Serial ESPTool
  - Adafruit: CircuitPython Code Editor

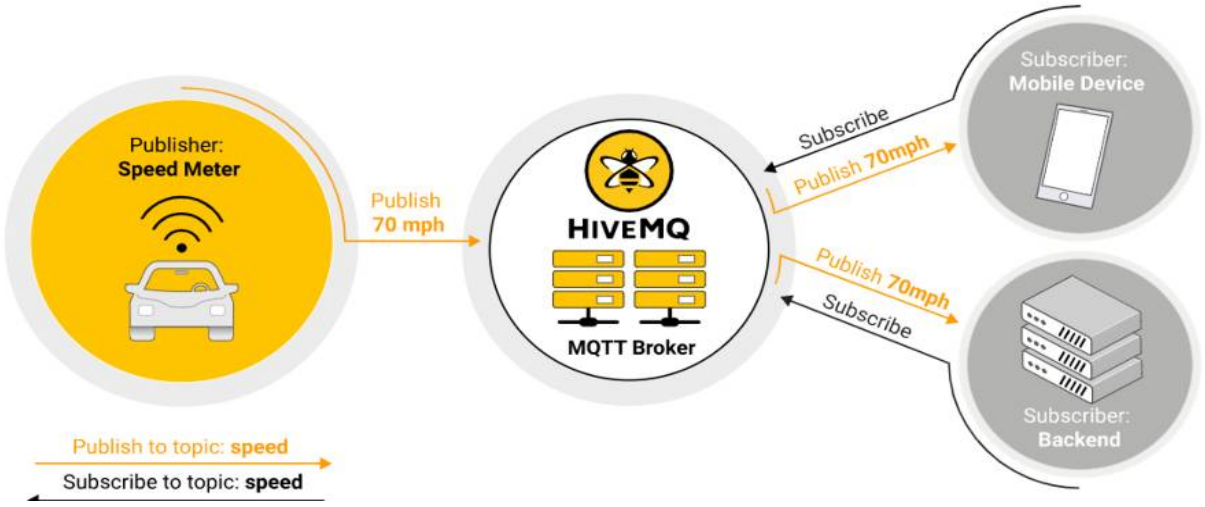
Az MQTT eredete és története

- Az MQTT protokollt 1999-ben fejlesztette ki Andy Stanford-Clark (IBM) és Arlen Nipper (Arcom, később Cirrus Link), hogy lehetővé tegye az alacsony energiafogyasztású és sávszélesség-hatékony kommunikációt olajvezetékeket felügyelő rendszereknél, műholdas kapcsolaton keresztül. Az MQTT kezdetben zárt rendszerekben használt megoldás volt, de később az IoT területén vált nyílt szabvánnyá.
- IBM 2010-ben szabadon elérhetővé tette az **MQTT 3.1** verzióját, ami növelte az IoT világban való elterjedését. A **HiveMQ** 2012-ben kezdett az MQTT-vel dolgozni, majd 2013-ban kiadta saját MQTT szoftverét. Az OASIS (Organization for the Advancement of Structured Information Standards) 2014-ben átvette az MQTT szabványosítását annak érdekében, hogy az egy nyílt, gyártófüggetlen protokoll legyen
- Ebben az előadásban az üzenetcsomagok felépítésével és a kommunikáció technikai részleteivel nem foglalkozunk, de a haladók utánanézhetnek itt:
  - OASIS: MQTT Version 3.1.1
  - OASIS: MQTT Version 5.0

Bevezetés az MQTT protokollba

- Az **MQTT** egy TCP-alapú kommunikációs protokoll, amelyet az **Internet of Things** (IoT) eszközök (telemetria, ipari automatizálás, okosotthonok) számára terveztek
- Kifejezetten alacsony sávszélességű és nagy késleltetésű környezetekhez optimalizált, megbízható és hatékony adatátvitel, alacsony erőforrásigény, gyors üzenetküldés
- A **publish/subscribe** séma elkülöníti az adatküldő klienst (**publisher**) azoktól a kliensektől amelyek fogadják az üzeneteket (**subscribers**). A küldő és fogadó kliensek sohasem kerülnek egymással közvetlen kapcsolatba. A kapcsolatot egy harmadik komponens, az ún. **bróker** kezeli

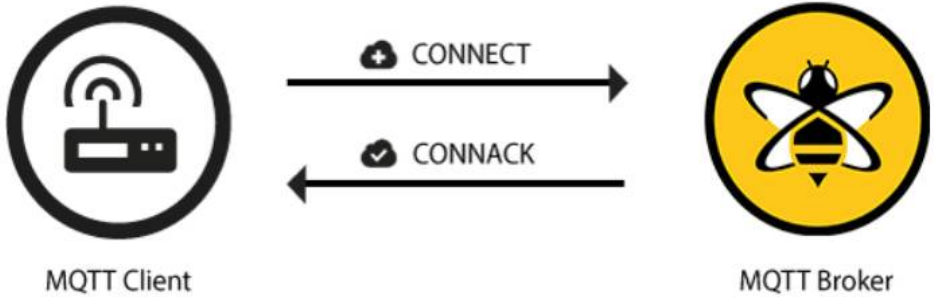




Forrás: https://www.hivemq.com/mqtt-essentials/

MQTT kliens – bróker/szerver kapcsolat

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

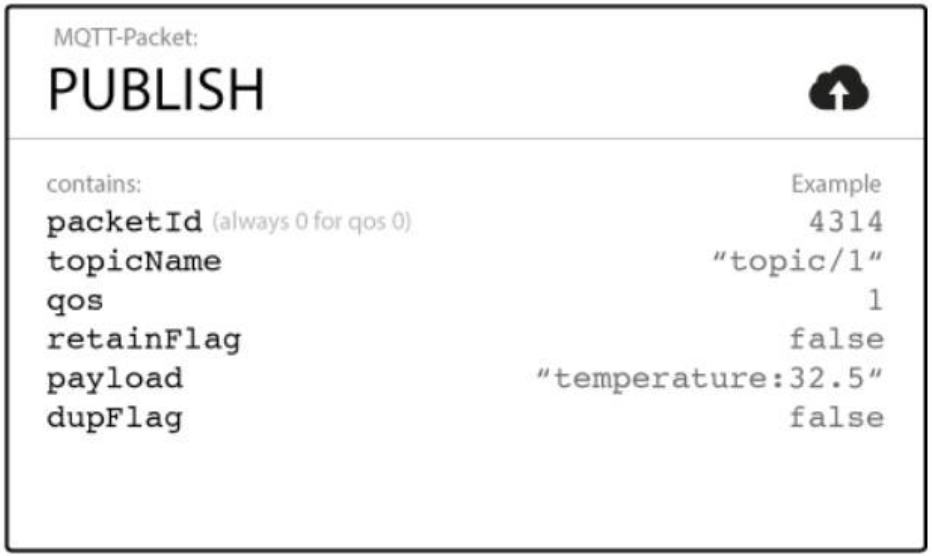


Forrás: https://www.hivemq.com/mqtt-essentials/

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

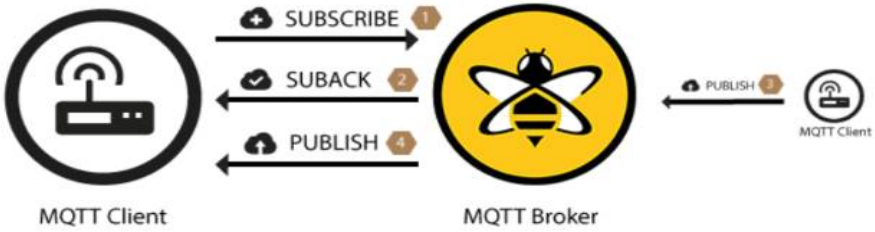
MQTT Publish - adatküldés

- 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 a `/` 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 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	2
returnCode 2	topic from SUBSCRIBE,	0
...	in the same order)	...

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: +

- Többszintű: #

Az Adafruit MiniMQTT programkönyvtár

- **MQTT kliens inicializálása:**

```
from adafruit_minimqtt.adafruit_minimqtt import MQTT
mqtt_client = MQTT(
    broker="mqtt.example.com",          # MQTT szerver címe
    port=1883,                          # MQTT szerver alapértelmezett portja
    client_id="CircuitPythonClient",    # Kliens egyedi azonosítója
    username="user",                    # Hitelesítési név
    password="pass",                    # Hitelesítési jelszó
    is_ssl=False,                       # Biztonságos kapcsolat engedélyezés
    keep_alive=60                       # Kapcsolat fenntartásának időtartama
)
```

- **Kapcsolódás és kapcsolatkezelés:**

  connect(): Kapcsolódás az MQTT brókerhez

  disconnect(): Kapcsolat bontása

  reconnect(): Újracsatlakozás hálózati hiba esetén

- **Keep Alive és hálózatfigyelés:**

  loop(): Folyamatosan figyeli az MQTT kapcsolatot és kezeli az üzeneteket

Az Adafruit MiniMQTT programkönyvtár

- **Üzenet küldése (Publish):**

topic message

```
mqtt_client.publish("iot/sensor", "Hello, MQTT!", qos=1)
```

publish(*topic, message, qos*): Üzenet küldése egy adott témára.
QoS (*Quality of Service*): Az üzenet megbízhatósági szintje (0, 1, 2).

- **Feliratkozás (Subscribe) és callback kezelés:**

```
def message_callback(client, topic, message):
    print(f"Kapott üzenet: {message} a témán: {topic}")
mqtt_client.add_topic_callback("iot/sensor", message_callback)
mqtt_client.subscribe("iot/sensor")
```

  - **subscribe**(*topic*): Feliratkozás egy témára
  - **add_topic_callback**(*topic, callback*): visszahívási függvény hozzárendelése egy témához
  - **on_connect** = connected
  - **on_disconnect** = disconnected
  - **on_subscribe** = subscribe
  - **on_unsubscribe** = unsubscribe
  - **on_message** = on_message

} visszahívási függvény hozzárendelése eseményhez

MQTT alapú LED vezérlés és kommunikáció

- Az alábbi program egy **MQTT** klienst valósít meg, ami a **HiveMQ** publikus szerveréhez csatlakozik (*broker.hivemq.com*). A programmal a beépített LED (**GPIO8,** katódvezérelt) állapotát vezérelhetjük távolról, **MQTT** üzenetek segítségével
- **A program tevékenységei:**
  - WiFi kapcsolódás és hálózatkezelés
  - MQTT kapcsolatfelvétel és kommunikáció egy nyilvános szerverrel
  - Eseményvezérelt üzenetkezelés egy LED vezérlésére
  - Az eseménykezelés egy visszahívási (callback) függvény segítségével valósul meg
- A LED vezérlés a **cspista/home/led** nevű **MQTT** topikon keresztül történik
  Üzenetek: "1" (bekapcsolás) és "0" (kikapcsolás)
- Az üzeneteket a **message_callback** függvény kezeli és módosítja a LED állapotát
- **Biztonsági megfontolások:** a program egy nyilvános **MQTT** szerverhez csatlakozik, amelyre bárki küldhet és fogadhat üzeneteket az adott topikban. Nincs hitelesítés, ezért nem garantálható az adatvédelem és a manipulációmentes kommunikáció

hivemq_led.py – 2/1.

```
import time, board, digitalio, wifi, ssl, socketpool
import adafruit_minimqtt.adafruit_minimqtt as MQTT
from os import getenv
led = digitalio.DigitalInOut(board.IO8)       # Beépített LED GPIO8
led.direction = digitalio.Direction.OUTPUT
WIFI_SSID = getenv("CIRCUITPY_WIFI_SSID")     # WiFi adatok a settings.toml fájlból
WIFI_PASS = getenv("CIRCUITPY_WIFI_PASSWORD")
MQTT_TOPIC = "cspista/home/led"               # A használt témakör neve

wifi.radio.connect(WIFI_SSID, WIFI_PASS)      # WiFi csatlakozás
pool = socketpool.SocketPool(wifi.radio)      # Socket pool létrehozása
ssl_context = ssl.create_default_context()    # Csak SSL/TLS esetén kell

mqtt_client = MQTT.MQTT(                      # MQTT kliens inicializálása
    broker="broker.hivemq.com",               # MQTT Broker adatok
    port=8883,                                # 1883 vagy 8883 (SSL/TLS esetén)
    username=getenv("HIVEMQ_USER"),           } # A fenti publikus broker esetén
    password=getenv("HIVEMQ_PASS"),           } # nem kell azonosítás (None/None is jó)
    socket_pool=pool,
    is_ssl=True,                              } # is_ssl = False, ill. 1883 port normát kapcsolat
    ssl_context=ssl_context                   } # ssl_context paraméter csak SSL/TLS esetén kell
)
```

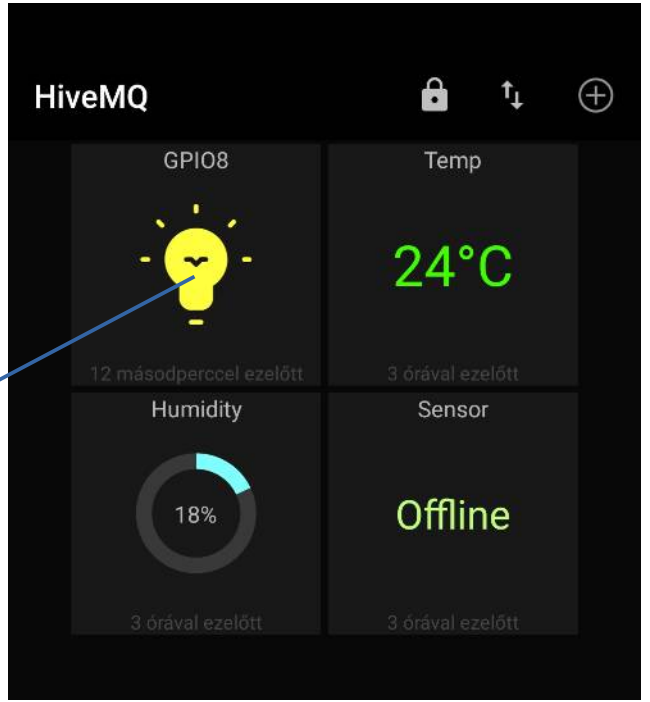
hivemq_led.py – 2/2.

```
# Callback függvény az üzenetek fogadására
def message_callback(client, topic, message):
    print(f"Kapott üzenet: {message} a témán: {topic}")
    if message == "1":
        led.value = False  # A beépített LED katódja...
        print("LED BEKAPCSOLVA")
    elif message == "0":
        led.value = True
        print("LED KIKAPCSOLVA")

mqtt_client.on_message = message_callback    # Callback hozzárendelése
mqtt_client.socket_timeout = 2               # 2 másodperces válaszvárás beállítása

mqtt_client.connect()                        # MQTT kapcsolat létrehozása
mqtt_client.subscribe(MQTT_TOPIC)            # Feliratkozás a LED vezérlő témára
led.value = True                             # Alapállapot katódvezérelt LED esetén
mqtt_client.publish(MQTT_TOPIC, "0")         # Alapállapot publokálása

# Folyamatos figyelés
while True:
        mqtt_client.loop(timeout=5)
        time.sleep(1)
```

hivemq_led.py futási eredménye

- A LED-et it az MQTT Dash Android alkalmazással vezéreltük

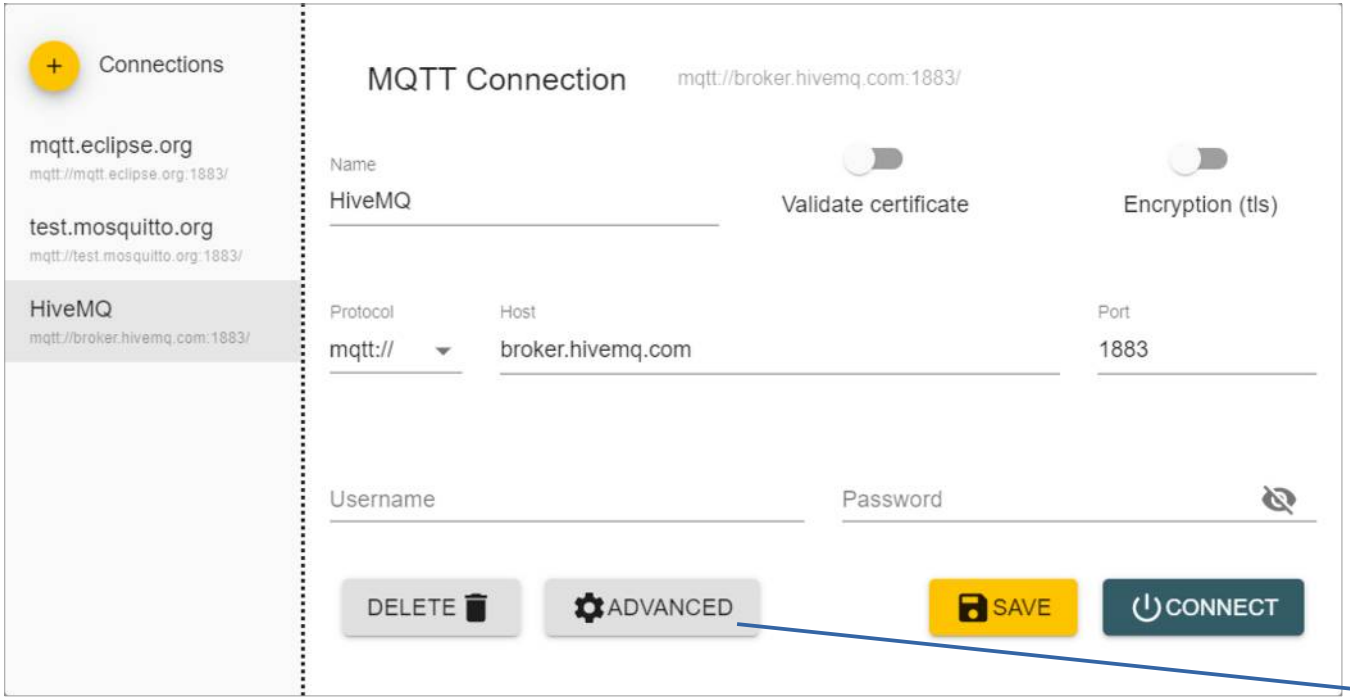
```
Shell
>>> %Run -c $EDITOR_CONTENT

 Csatlakozás a WiFi hálózathoz...
 WiFi kapcsolat létrejött!
 Csatlakozás az MQTT szerverhez...
 Kapcsolódás sikeres!
 LED kikapcsolva, állapot publikálva.
 Kapott üzenet: 0 a témán: cspista/home/led
 LED KIKAPCSOLVA
 Kapott üzenet: 1 a témán: cspista/home/led
 LED BEKAPCSOLVA
 Kapott üzenet: 0 a témán: cspista/home/led
 LED KIKAPCSOLVA
 Kapott üzenet: 1 a témán: cspista/home/led
 LED BEKAPCSOLVA
 Kapott üzenet: 0 a témán: cspista/home/led
 LED KIKAPCSOLVA
```



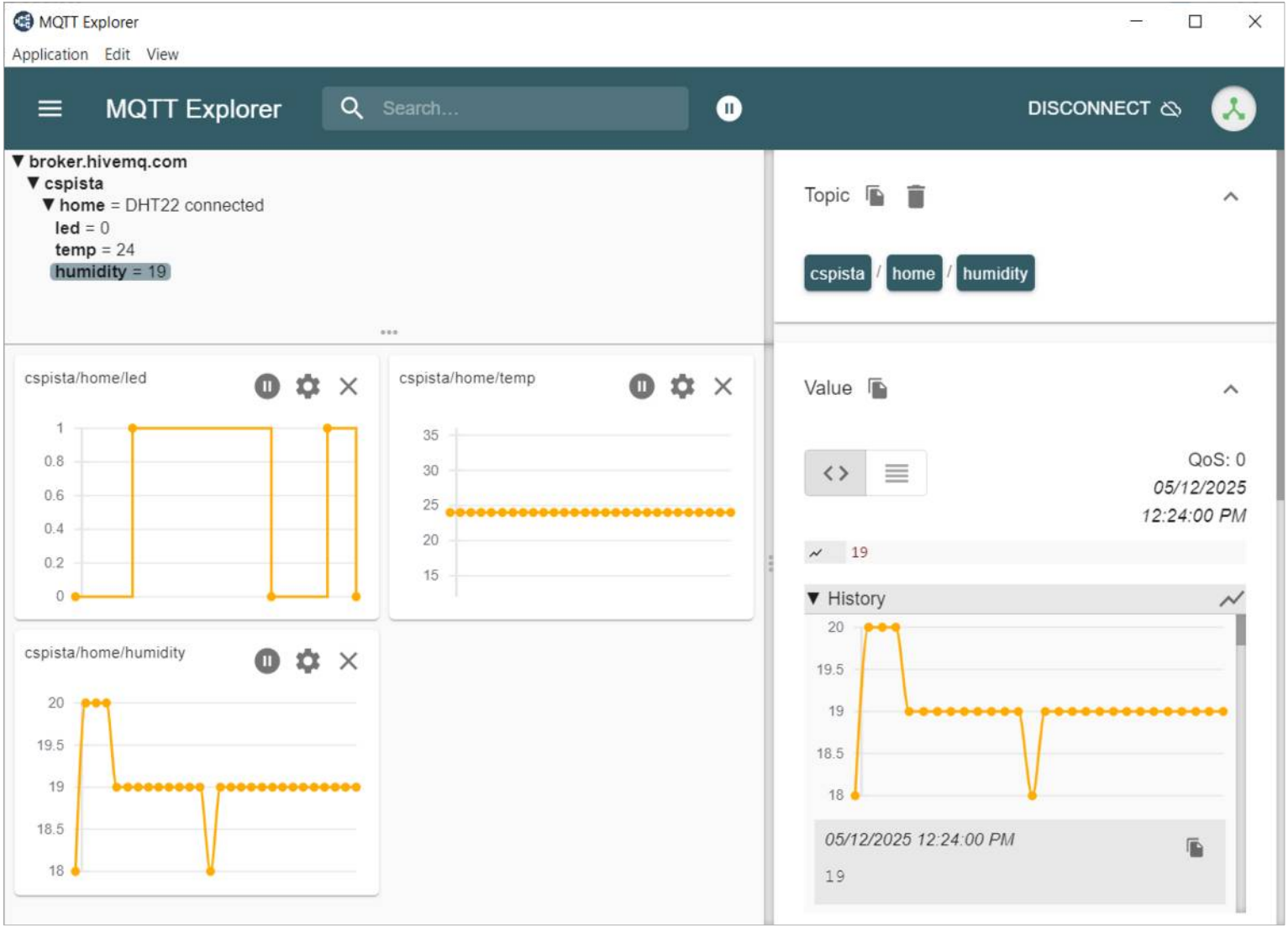
Switch/button	Text + postfix (°C)
progress+posfix (%)	Text

MQTT Explorer

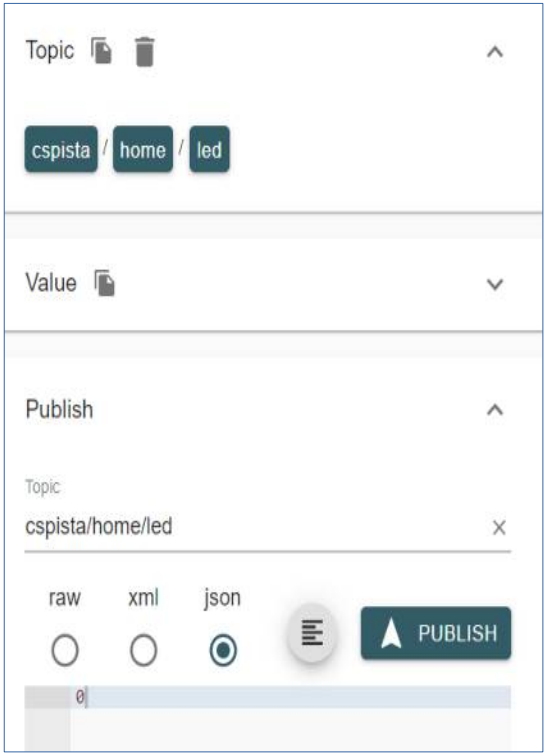
- Az MQTT Explorer egy grafikus kliens alkalmazás, amely segít az MQTT protokollt használó rendszerek megfigyelésében és kezelésében. Lehetővé teszi az MQTT topikok struktúrájának vizuális megjelenítését, az üzenetek küldését és fogadását



A program telepítése után az alábbi beállítással vehetjük használatba a **broker.hivemq.com** publikus **MQTT** szervert a **CONNECT** gomb megnyomása után



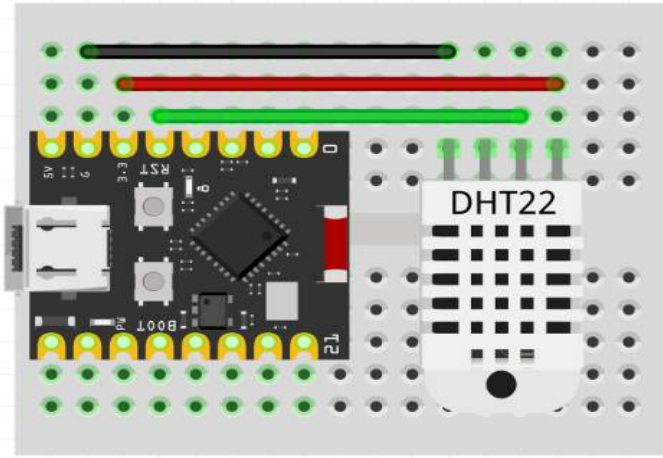
A **PUBLISH** panelon a kiválasztott topikba adatokat küldhetünk



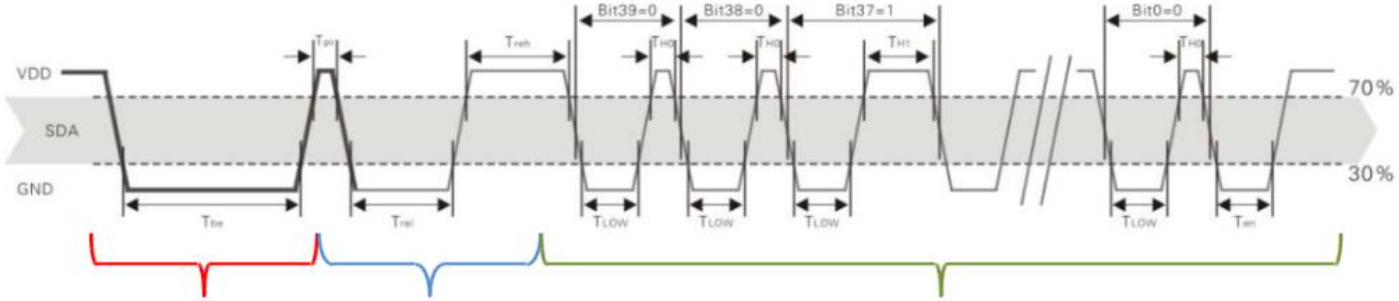
Hőmérséklet és páratartalom mérése

Az Am2302 (DHT22) szenzor jellemzői:

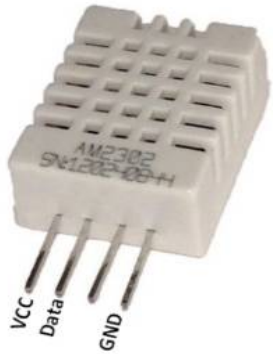
- Hőmérséklet és rel. páratartalom mérésére
- Felbontás: 0.1 °C, illetve 0.1 %
- 1-wire, nem szabványos protokol
- Mintavételezési gyakoriság: 2 s
- Tápfeszültség: 3,3 – 6 V



- Programkönyvtár: Adafruit_CircuitPython_DHT
- Adatlap: sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf



Összesen tehát 85 időzítést tartalmaz egy-egy tranzakció …



HiveMQ_DHT22

- Az előző programot kibővítve most egy **DHT22** szenzor segítségével mérjük a hőmérsékletet és a páratartalmat, s a kiolvasott adatokat 10s időközönként publikáljuk az **mqtt://broker.hivemq.com** szerverre. Eközben megtartjuk a beépített LED (**GPIO8,** katódvezérelt) **MQTT** üzenetekkel történő vezérlését is
- **A felhasznált MQTT topikok és funkcióik:**
  - **cspista/home** → Általános rendszerállapot (pl. "DHT22 connected", "Offline")
  - **cspista/home/led** → LED vezérlés (küldhető: "1" BE, vagy "0" KI)
  - **cspista/home/temp** → Hőmérséklet adatok publikálása °C fokokban (pl. "22")
  - **cspista/home/humidity** → Relatív páratartalom %-ban kifejezve (pl. "55")
- Főbb funkciók:
  - WiFi kapcsolat kezelése (automatikus újracsatlakozás)
  - **MQTT** alapú adatküldés és LED vezérlés
  - **DHT22** szenzor adatainak publikálása
  - Kritikus hiba esetén teljes mikrovezérlő újraindítás
  - Nyomkövetéshez tájékoztató üzenetek kiírása a terminálra

hivemq_dht22.py – 3/1.

```
import time, board, digitalio, wifi, ssl, socketpool, microcontroller, adafruit_dht
import adafruit_minimqtt.adafruit_minimqtt as MQTT
from os import getenv
led = digitalio.DigitalInOut(board.IO8)    # Beépített LED GPIO8
led.direction = digitalio.Direction.OUTPUT
dht_sensor = adafruit_dht.DHT22(board.IO4) # DHT22 inicializálása (GPIO4)
WIFI_SSID = getenv("CIRCUITPY_WIFI_SSID")  # WiFi adatok a settings.toml fájlból
WIFI_PASS = getenv("CIRCUITPY_WIFI_PASSWORD")
MQTT_TOPIC_HOME = "cspista/home"
MQTT_TOPIC_LED = "cspista/home/led"
MQTT_TOPIC_TEMP = "cspista/home/temp"
MQTT_TOPIC_HUMIDITY = "cspista/home/humidity"
wifi.radio.connect(WIFI_SSID, WIFI_PASS)     # WiFi csatlakozás
pool = socketpool.SocketPool(wifi.radio)     # Socket pool létrehozása
ssl_context = ssl.create_default_context()   # Csak SSL/TLS esetén kell
mqtt_client = MQTT.MQTT(                     # MQTT kliens inicializálása
    broker="broker.hivemq.com",              # MQTT Broker adatok
    port=8883,                               # 1883 vagy 8883 (SSL/TLS esetén)
    username=getenv("HIVEMQ_USER"),          # A fenti publikus broker esetén
    password=getenv("HIVEMQ_PASS"),          # nem kell azonosítás (None/None is jó)
    socket_pool=pool,
    is_ssl=True,                             # is_ssl = False, ill. 1883 port normát kapcsolat
    ssl_context=ssl_context                  # ssl_context paraméter csak SSL/TLS esetén kell
)
```

hivemq_dht22.py – 3/2.

```
# Callback függvény az üzenetek fogadására
def message_callback(client, topic, message):
    print(f"Kapott üzenet: {message} a témán: {topic}")
    if message == "1":
        led.value = False  # A beépített LED katódja...
        print("LED BEKAPCSOLVA")
    elif message == "0":
        led.value = True
        print("LED KIKAPCSOLVA")

mqtt_client.on_message = message_callback    # Callback hozzárendelése
mqtt_client.socket_timeout = 2               # 2 másodperces válaszvárás beállítása
mqtt_client.will_set(                        # „Végakarat” megadása: ha megszakad a kapcsolat
    topic="cspista/home",                    # ebbe a topikba az alábbi üzenetet küldjük ki
    msg="Offline",
    qos=1,
    retain=False
)

mqtt_client.connect()                        # MQTT kapcsolat létrehozása
mqtt_client.subscribe(MQTT_TOPIC)            # Feliratkozás a LED vezérlő témára
led.value = True                             # Alapállapot katódvezérelt LED esetén
mqtt_client.publish(MQTT_TOPIC, "0")         # Alapállapot publokálása
```

A „last will” beállítását már az előző programban is megtehettük volna...

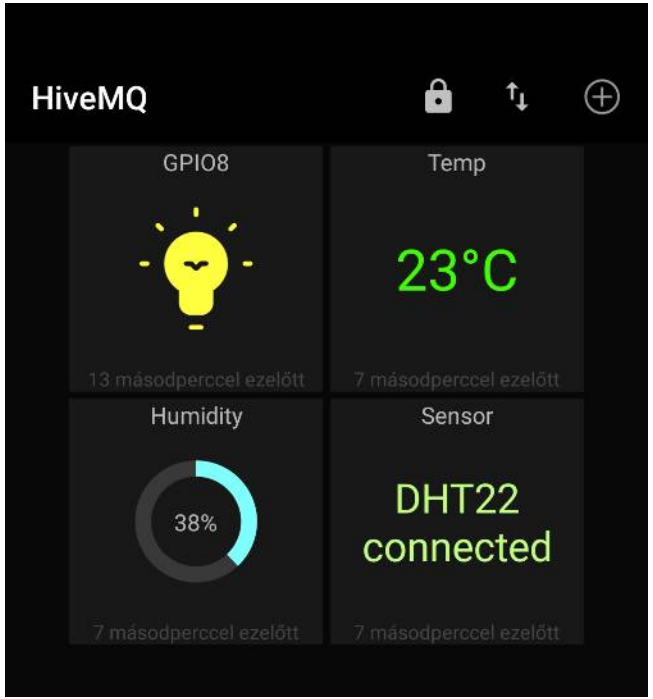
hivemq_dht22.py – 3/3.

```
last_read_time = time.monotonic()       # Induláskor rögzítjük az időt
interval = 10                           # DHT22 adatkiolvasás intervalluma másodpercben
while True:
    try:
        mqtt_client.loop(timeout=5)     # Folyamatos MQTT kezelési ciklus
        current_time = time.monotonic()
        if current_time - last_read_time >= interval:  # Csak ha letelt az idő
            last_read_time = current_time  # Frissítjük az utolsó olvasási időt
            temperature = round(dht_sensor.temperature)
            humidity = round(dht_sensor.humidity)
            mqtt_client.publish(MQTT_TOPIC_HOME, "DHT22 connected")
            mqtt_client.publish(MQTT_TOPIC_TEMP, str(temperature))
            mqtt_client.publish(MQTT_TOPIC_HUMIDITY, str(humidity))

    except RuntimeError as error:
        print(f"DHT22 olvasási hiba: {error}")
        mqtt_client.publish(MQTT_TOPIC_HOME, "DHT22 read error")
        time.sleep(2)  # Ha hiba van, várunk egy kicsit és újrapróbáljuk

    except Exception as e:        # Általános hiba (pl. hálózati vagy MQTT kapcsolat)
        print(f"Hiba történt: {e}")
        time.sleep(5)
        microcontroller.reset()  # Teljes újraindítás
```

hivemq_dht22.py futási eredménye

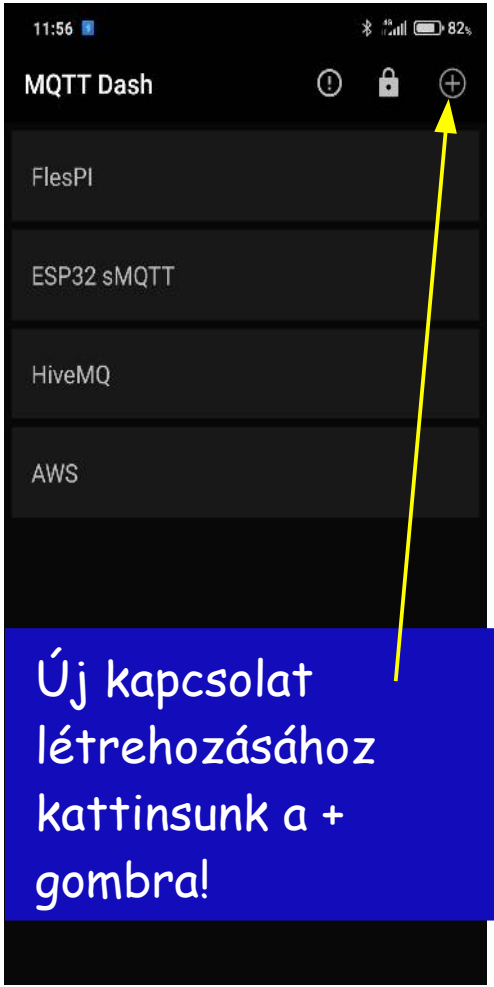
- A publikált adatokat az MQTT Dash Android alkalmazással néztük

```
Shell
>>> %Run -c $EDITOR_CONTENT
 Csatlakozás a WiFi hálózathoz...
 WiFi kapcsolat létrejött!
 Csatlakozás az MQTT szerverhez...
 Kapcsolódás sikeres!
 LED kikapcsolva, állapot publikálva.
 Kapott üzenet: 0 a témán: cspista/home/led
 LED KIKAPCSOLVA
 Kapott üzenet: 1 a témán: cspista/home/led
 LED BEKAPCSOLVA
 Hőmérséklet: 24°C, Páratartalom: 32%
 Hőmérséklet: 24°C, Páratartalom: 32%
 Hőmérséklet: 24°C, Páratartalom: 33%
 Hőmérséklet: 24°C, Páratartalom: 33%
 Hőmérséklet: 24°C, Páratartalom: 32%
```

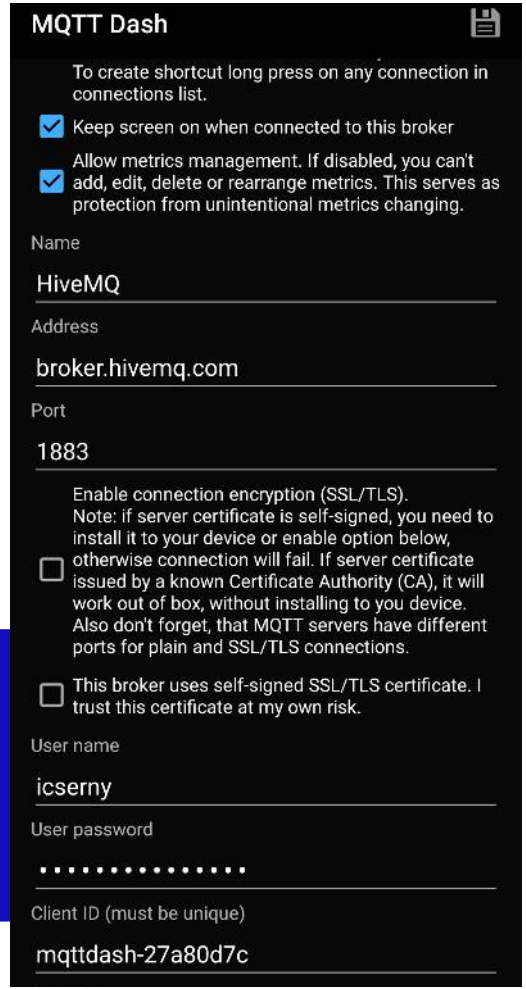


Switch/button	Text + postfix (°C)
progress+posfix (%)	Text

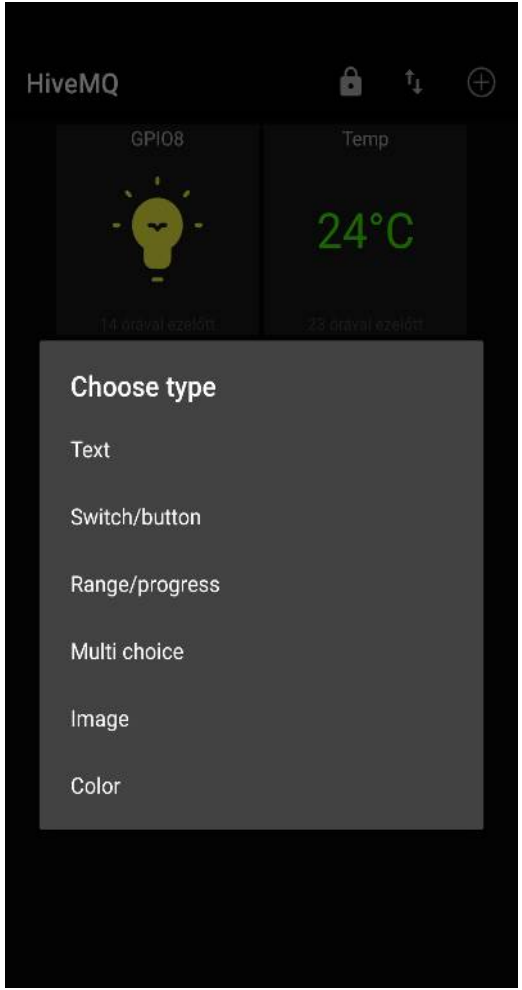
MQTT Dash konfigurálása



Új kapcsolat létrehozásához kattinsunk a + gombra!

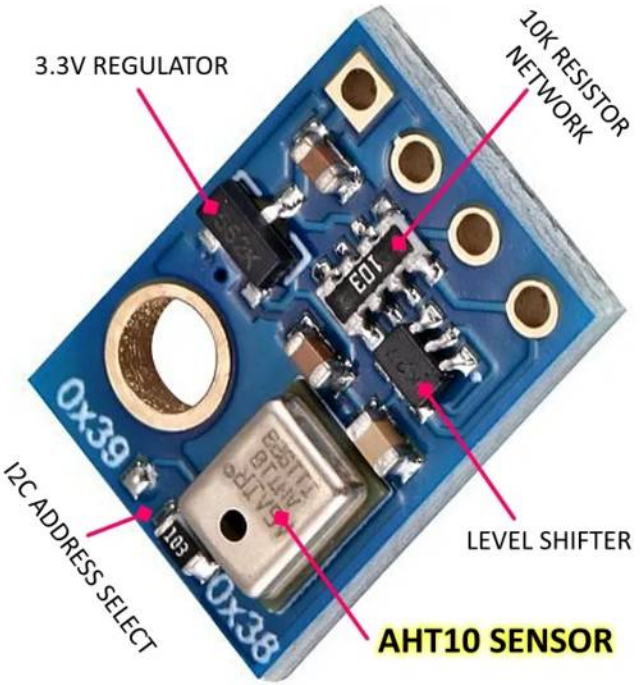


Új elem létrehozásához kattinsunk a + gombra!



Hőmérséklet és páratartalom mérés AHT10 szenzorral

- Az előző programot átírhatjuk az **AHT10** integrált hőmérséklet- és páratartalom-érzékelő használatához
- A szenzor az I2C interfészt használ (cím: 0x38), ami a **GPIO8, GPIO9** lábakat lefoglalja, ezért a távvezérelt LED-et a **GPIO10**-re kötjük
- **Főbb jellemzők:**
  - Felbontás: hőmérséklet: 0,01℃
    Páratartalom: 0,024% RH
  - Pontosság: ±3% relatív páratartalom (RH) és ±0,5℃ hőmérséklet
  - Működési tartomány: 0-100% RH és –40℃-tól 85℃-ig.
  - Interfész: I²C (alapértelmezett cím: 0x38)
  - I2C sebesség: 100 kHz, vagy 400 kHz



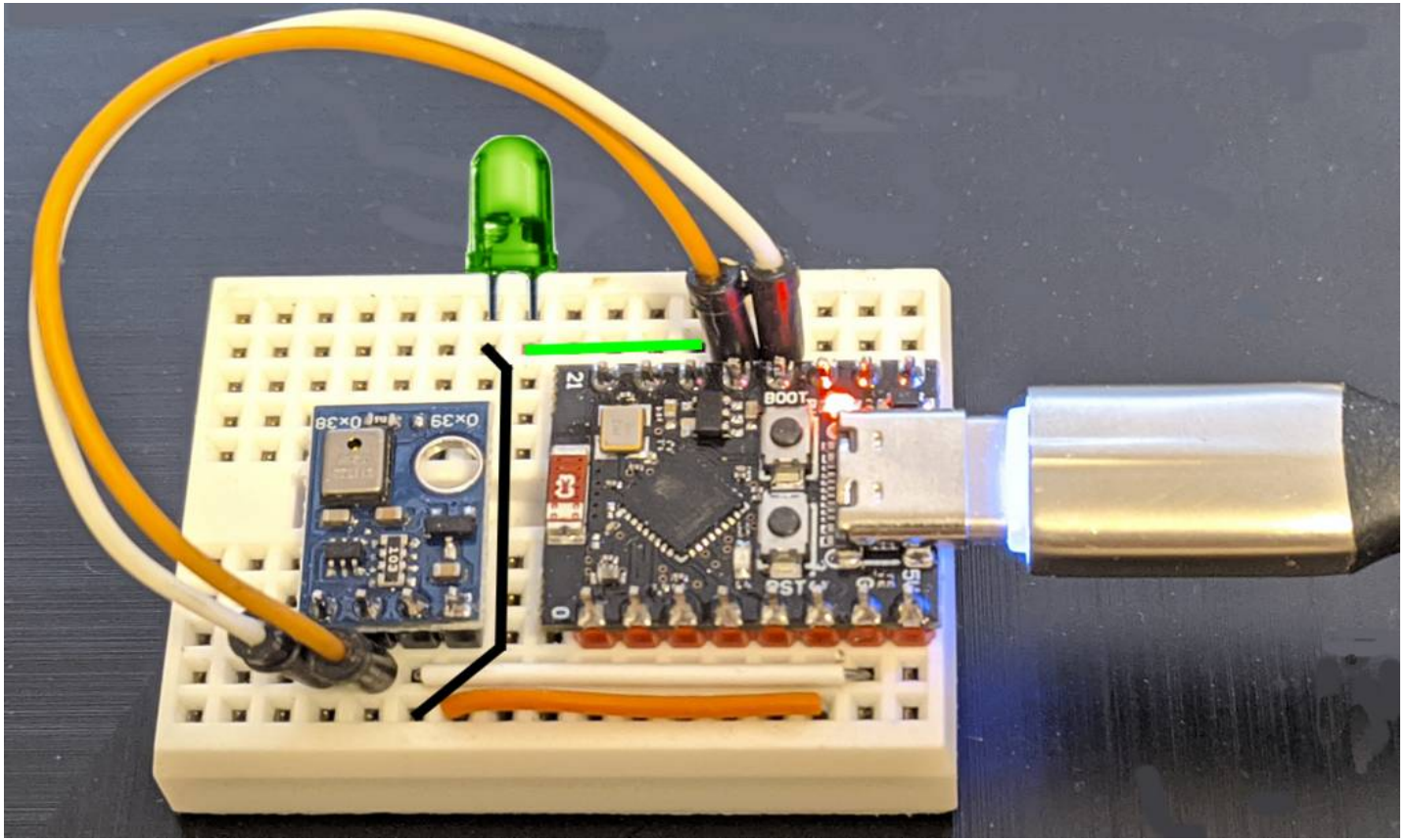
AHT10 támogatói könyvtár

- Az **AHT10** szenzor kezeléséhez az **adafruit_ahtx0** könyvtárat telepítsük az Adafruit CircuitPython Library Bundle gyűjteményből a **lib** mappába
- **Forráskód:**
  - **Adafruit_CircuitPython_AHTx0**
- **Dokumentáció:** docs.circuitpython.org/projects/ahtx0
- **Importálás:** `import adafruit_ahtx0`
- **Inicializálás:** `sensor = adafruit_ahtx0.AHTx0(i2c,0x38)`
- **Tagfüggvények/tulajdonságok:**
  - **reset()** - soft reset
  - **calibrate()** - önkalibrálást indít
  - **status** - státuszbájt kiolvasása
  - **temperature** – hőmérséklet kiolvasása
  - **relative_humidity** – páratartalom kiolvasása

Opcionális paraméter (address)

Bekötési vázlat

- A Vin bemenetre 3,3 V-ot kötöttünk, SCL ⟶ GPIO9, SDA ⟶ GPIO8, LED+ ⟶ GPIO10, LED- és AHT10 Gnd ⟶ G



hivemq_aht10.py (részletek)

- Az alábbaikban csak a hivemq_dht22.py programtól való eltéréseket listáztuk ki

```
import adafruit_ahtx0  # AHT10 könyvtár

# AHT10 inicializálása
i2c = board.I2C()                          # (I2C: GPIO8, GPIO9)
aht_sensor = adafruit_ahtx0.AHTx0(i2c)  # AHT10 inicializálás I2C-n
aht_sensor.calibrate()                     # önkalibrálás

# A vezérelt LED GPIO10
led = digitalio.DigitalInOut(board.IO10)
led.direction = digitalio.Direction.OUTPUT
. . .

# AHT10 adatkiolvasás és publikálás
temperature = round(aht_sensor.temperature)
humidity = round(aht_sensor.relative_humidity)
```

Az ESP32 C3 Super Mini kártya kivezetései