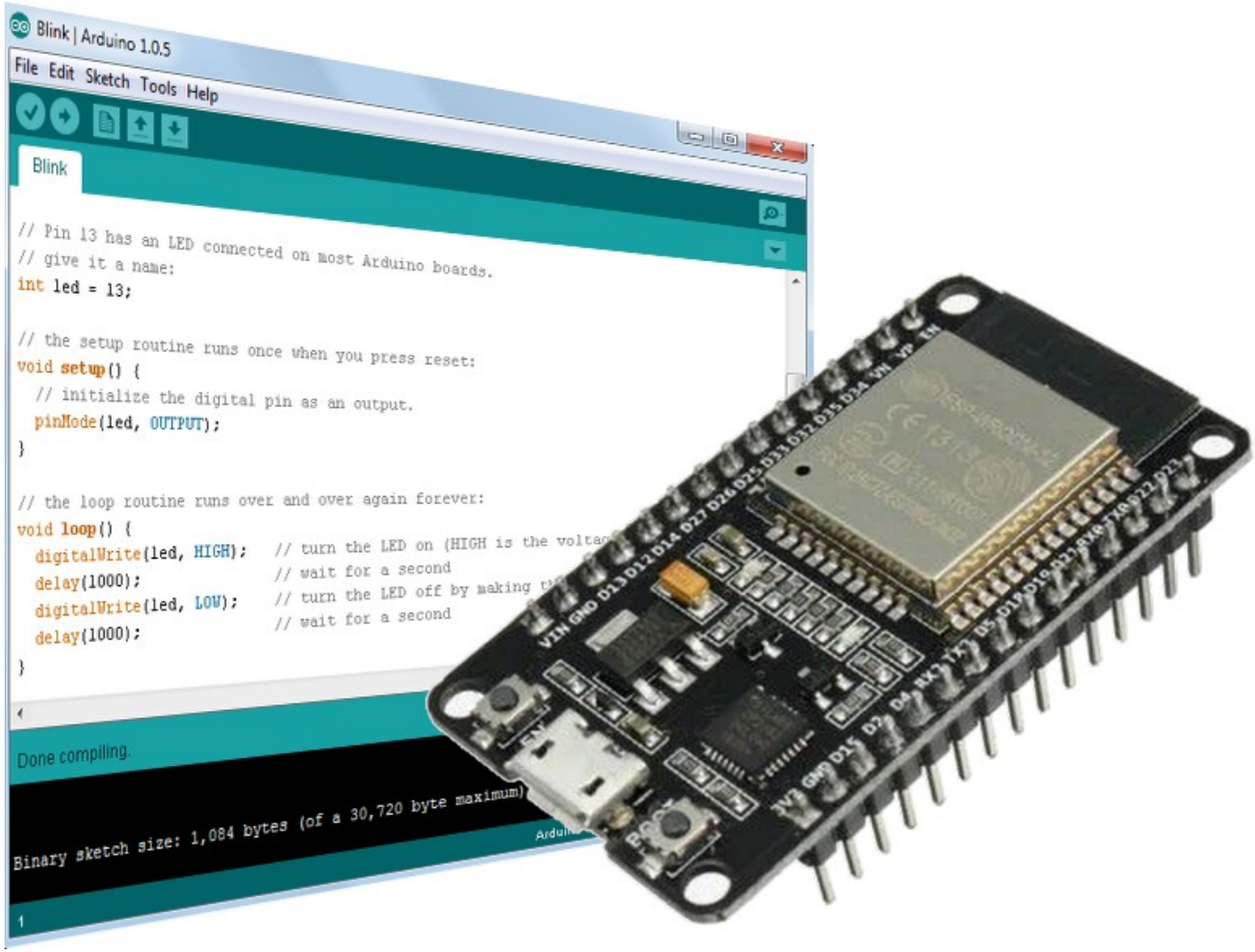
ESP32 mikrovezérlők programozása Arduino környezetben

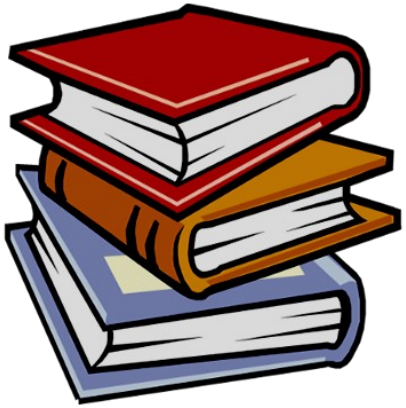


ARDUINO™

7. ESP32 webkliens IOT alkalmazások

Felhasznált és ajánlott irodalom

- LETSCODE.HU: Hogyan működik az Internet
- W3Schools.com: HTTP Request Methods
- TutorialsPoint: HTTP requests
- Időjárási adatok és előrejelzés: openweathermap.org
- ThingSpeak for IOT: thingspeak.com
- developer.mozilla.org: HTTP response status codes
- Benoit Blanchon: Mastering ArduinoJson 6
- ESPRESSIF: ESP32 Arduino Core Documentation
- ESPRESSIF: ESP32 Technical Reference Manual



Példaprogramok

ESP32_webclient – egyszerű adatlekérés

ESP32_OpenWeatherMap – Időjárási adatok lekérése

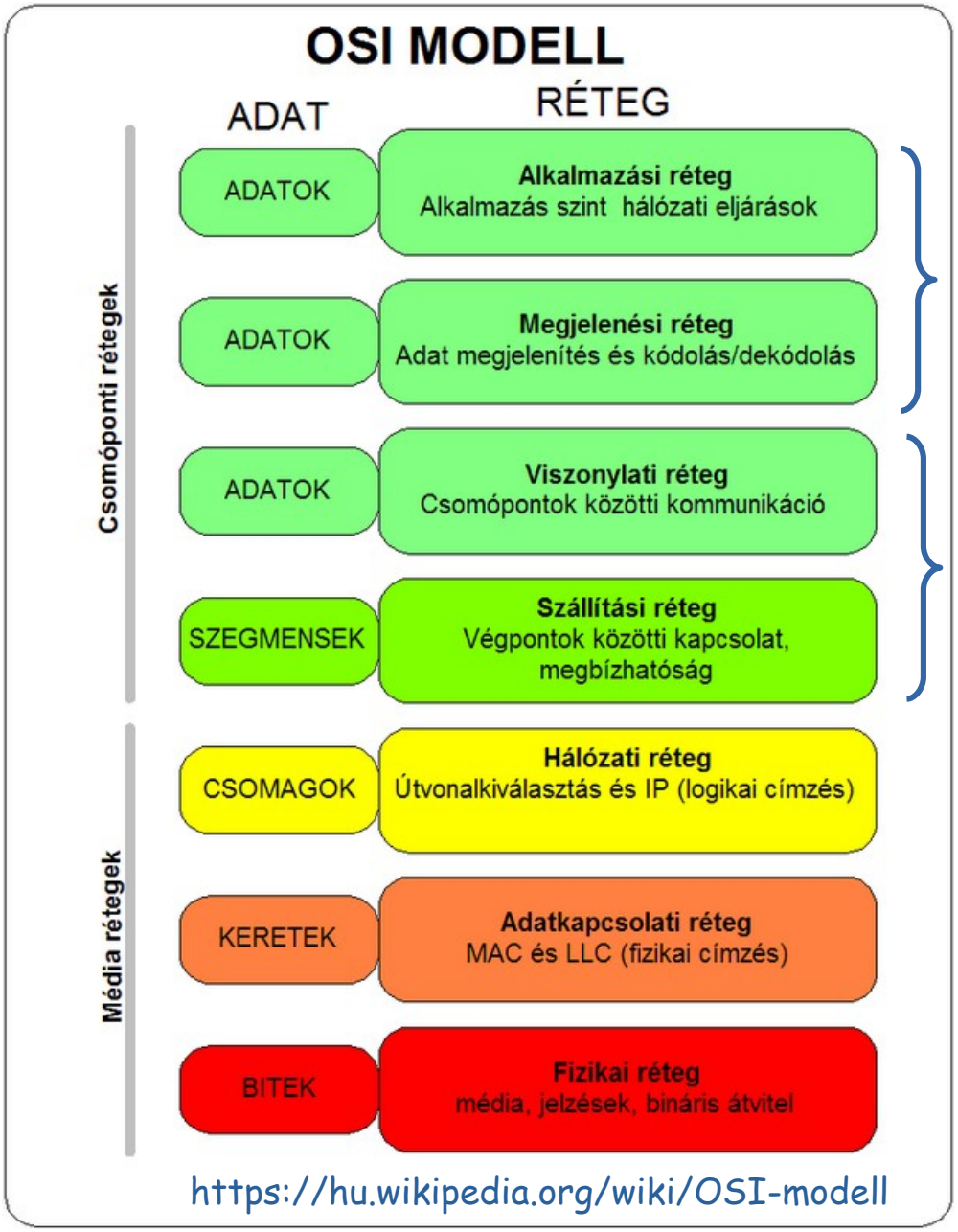
ESP32_OwMap_oled – időjárásjelző óra

ESP32_ThingSpeak – szenzor adatok küldése

ESP32_ThingSpeak_POST – adatküldés másképp

ESP32_ThingSpeak_JSON –
adatküldés JSON formátumban

Az OSI-modell (ISO 7498-1)



- Az **Open Systems Interconnection** (nyílt rendszerek összekapcsolása) referenciamodellje hét rétegbe szervezve írja le a hálózati kapcsolatot

HTTP, FTP, SMTP, Telnet, **NTP,** NFS

TCP, UDP

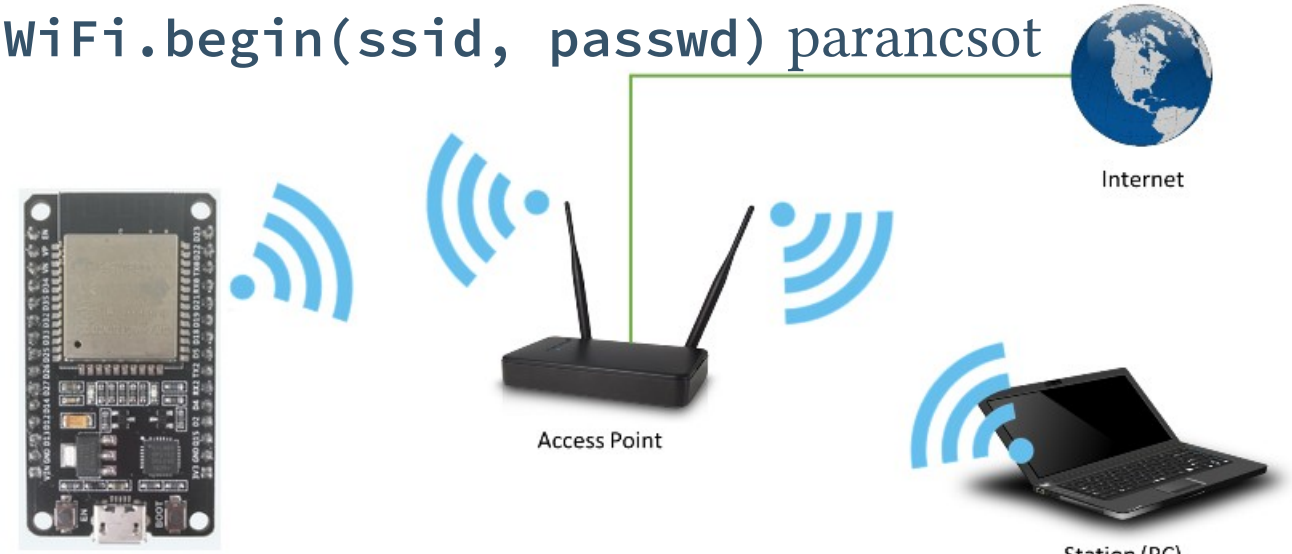
IP (**IPv4** – IPv6), ARP, IPSEC, ...

Ethernet, **WiFi,** PPP, ...

RS-232, V35, DSL, ISDN, 10BASE-T, 100BASE-TX stb.

Kapcsolódás az Internetre

- Az **ESP32 Wi-Fi** perifériája és az **ESP32 Arduino Core** részét képező WiFi programkönyvtár lehetővé teszi, hogy a helyi hálózatra csatlakozzunk (**STA** módban, kliensként, vagy **AP** módban, elérési pontként, vagy egyidejűleg mindkét módot használva)
- Mi most kliens alkalmazást készítünk, így **STA** (station) módban a helyi routerhez (vagy elérési ponthoz) fogunk csatlakozni, amely az internet elérést biztosítja számunkra
- A programjainkba ehhez becsatoljuk a **WiFi.h** fejléc állományt és kiadjuk a `WiFi.begin(ssid, passwd)` parancsot



ESP32 Wi-Fi STA módban

Csatlakozás a WiFi hálózathoz

- Az **ESP32** mikrovezérlő kliensként történő használatához programjaink elején pl. így csatlakozunk a helyi **WiFi** hálózatra
- A soros porti kiíratás természetesen opcionális, a **Serial** kezdetű sorokat elhagyhatjuk

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

void setup () {
  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());
}
```

- A személyes adatokat kiszerveztük egy fejléc állományba, amelyet a **Vázlatfüzet** (Sketchbook) mappa **libraries/secrets** almappájában helyeztünk el

secrets.h

```
#define WIFI_SSID  MY_SSID
#define WIFI_PASS  MY_PASSWORD

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

String THINGSPEAK_WRITE_APIKEY =
"xxxxxxxxxxxx";
```

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, melyet szintén egy üres sor zár le
- 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 adott erőforrásából
- **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 mai előadás első ThingSpeak példáját!)
- A **GET** kérésre egy példa:

```
GET /test/demo_form.php?name1=value1&name2=value2 HTTP/1.1
```

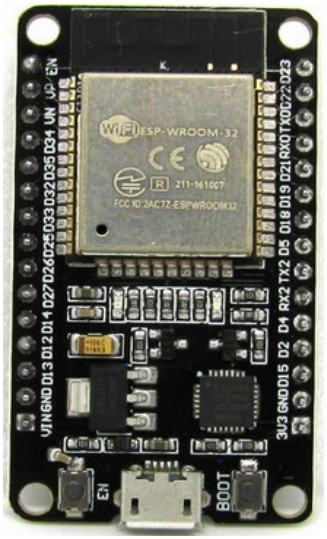
- A **POST** kérés az üzenet törzsében küldi az adatokat és többnyire adatküldésre használjuk (pl. űrlapok kitöltésénél), de a GET-hez hasonló lekérésre is használhatjuk
- A **POST** kérésre egy példa:

```
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
```

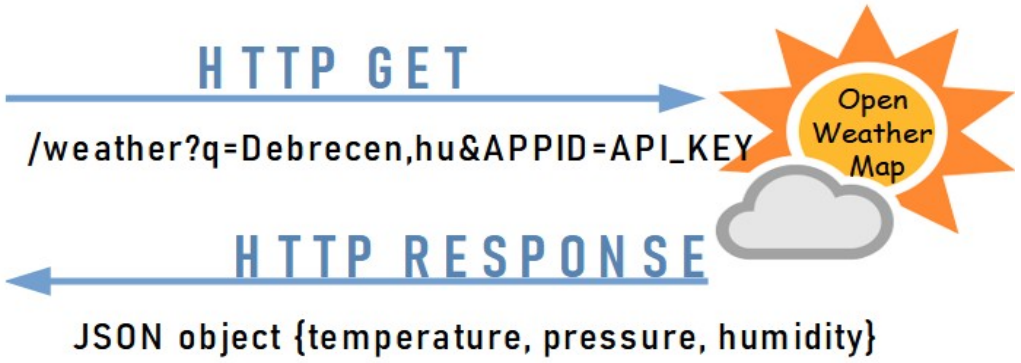
Időjárási adatok lekérése (OpenWeatherMap.org)

- Az aktuális időjárási adatokat és előrejelzéseket az openweathermap.org webszerverről tölthetjük le, HTTP kliensként
- Regisztráció (Sign in/Create Account) után az ingyenes szolgáltatásokat vehetjük igénybe (lásd Pricing, Free oszlop)
- Első bejelentkezéskor szerezzük meg és jegyezzük fel az API kulcsot!
- Mi most csak az aktuális adatok lekérdezésével foglalkozunk, például:
  `http://api.openweathermap.org/data/2.5/weather?q=Debrecen&appid=********`
- A választ alapértelmezetten JSON formátumban kapjuk meg, pl.:


```
{"coord":{"lon":21.6333,"lat":47.5333},"weather":[{"id":802,"main":"Clouds","description":"scattered
clouds","icon":"03d"}],"base":"stations","main":{"temp":283.15,"feels_like":279.93,"temp_min":283.15,
"temp_max":283.15,"pressure":1025,"humidity":46} . . .
```

Mit kezdjünk vele?



ESP32_webclient.ino: HTTP GET request

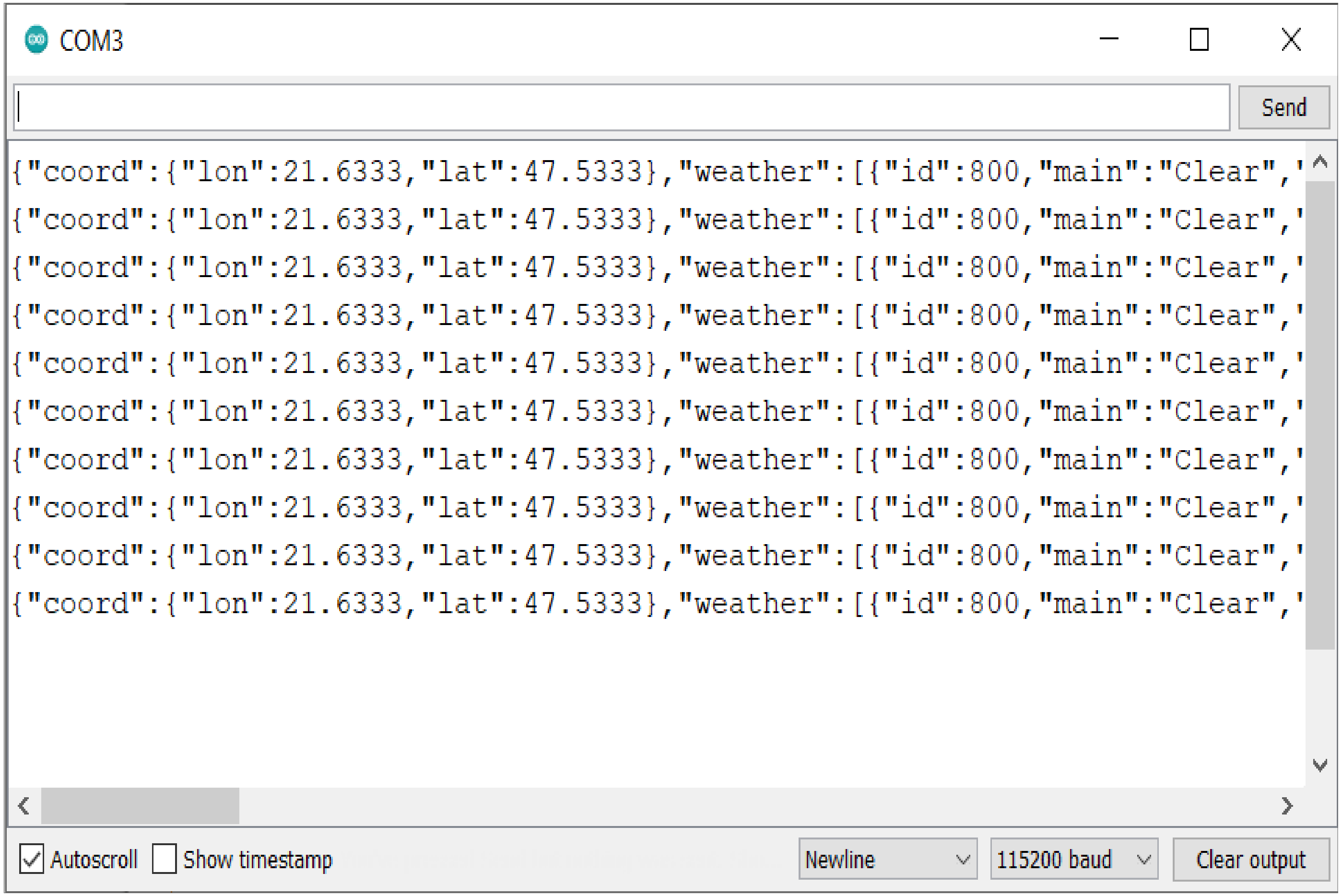
```
#include <WiFi.h>
#include <HTTPClient.h>
#include "secrets.h"
String server_url = "http://api.openweathermap.org/data/2.5/weather?q=Debrecen&APPID="
                    + OPENWEATHERMAP_APPID+"&mode=json";

void setup () {
  Serial.begin(115200);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }
}

void loop() {
  if (WiFi.status() == WL_CONNECTED) {     // Check WiFi connection status
    HTTPClient http;                       // Declare an object of class HTTPClient
    http.begin(server_url);                // Specify request destination
    int httpCode = http.GET();             // Send the request
    if (httpCode > 0) {                    // Check the returning code
      String payload = http.getString();   // Get the request response payload
      Serial.println(payload);             // Print the response payload
    }
    http.end();                            // Close connection
  }
  delay(10000);                            // Send a request every 10 seconds
}
```

HTTP GET lekérés:

Az URL-hez fűzve adjuk meg a paramétereket (nem biztonságos módszer)

ESP32_webclient.ino: eredmények

- A kapott eredmény nem túl olvasmányos, az adatok feldolgozásához célszerű beszerezni egy, a JSON formátumot kezelni tudó programkönyvtárat




```json
{
  "coord": {
    "lon": 21.6333,
    "lat": 47.5333
  },
  "weather": [
    {
      "id": 800,
      "main": "Clear",
      "description": "clear sky",
      "icon": "01d"
    }
  ],
  "base": "stations",
  "main": {
    "temp": 280.15,
    "feels_like": 272.85,
    "temp_min": 280.15,
    "temp_max": 280.15,
    "pressure": 1021,
    "humidity": 31
  },
  "visibility": 10000,
  "wind": {
    "speed": 6.17,
    "deg": 50
  },
  "clouds": {
    "all": 0
  },
  "dt": 1616583662,
  "sys": {
    "type": 1,
    "id": 6665,
    "country": "HU",
    "sunrise": 1616560097,
    "sunset": 1616604662
  },
  "timezone": 3600,
  "id": 721472,
  "name": "Debrecen",
  "cod": 200
}
```

A JSON adatcsere-formátum

- **JSON** (JavaScript Object Notation) szöveg alapú, adatcseréhez való formátum ami C, C++, C#, Java, JavaScript, Perl, Python és más nyelveken könnyen kezelhető (https://www.json.org/)
- A **JSON** formátum két struktúrára épül:
  - Név – érték párok kollekciója, ami pl. objektum, struktúra, asszociatív tömb formájában realizálható
  - Értékek listája ami pl. tömbként realizálható
- **Példa:** `{ "name":"John","age":31,"city":"New York" }`

- A továbbiakhoz szükségünk lesz az **ArduinoJson** könyvtárra (az Arduino IDE **Tools/Manage Libraries** menüpontjából telepíthető)
- Dokumentáció és mintapéldák az https://arduinojson.org/ honlapon találhatók
- A Deserialization tutorial lapon egy teljes könyvfejezet letölthető
- A JsonFilterExample mintapélda a bejövő adatok szűrését mutatja be

Időjárási adatok kezelése és szűrése

JSON Nyers adat Fejlécek

Mentés Másolás Összes összecsukása Összes kinyitása


```
▼ coord:
    lon:            21.6333
    lat:            47.5333
▼ weather:
  ▼ 0:
      id:           800
      main:         "Clear"
      description:  "clear sky"
      icon:         "01d"
  base:             "stations"
▼ main:
    temp:           280.15
    feels_like:     272.85
    temp_min:       280.15
    temp_max:       280.15
    pressure:       1021
    humidity:       31
  visibility:       10000
▶ wind:             {…}
▶ clouds:           {…}
  dt:               1616583662
▼ sys:
```


Egyszerű adatbeolvasás (deserialization)

```
#include <ArduinoJson.h>

StaticJsonDocument<2000> doc;
deserializeJson(doc, payload);
```

Adatbeolvasás bemeneti szűréssel

```
#include <ArduinoJson.h>

StaticJsonDocument<500> doc;
StaticJsonDocument<200> filter;

// Prepare filter
filter["main"]["temp"] = true;
filter["main"]["pressure"] = true;
filter["main"]["humidity"] = true;
deserializeJson(doc, payload,
       DeserializationOption::Filter(filter));
//get data
float tempK = doc["main"]["temp"];
int hPa = doc["main"]["pressure"];
int relHum = doc["main"]["humidity"];
```

A JsonFilterExample mintapélda alapján…

ESP32_OpenWeatherMap.ino 2/1.

- Bővítsük ki az előző programunkat a JSON objektumok kezelésével!

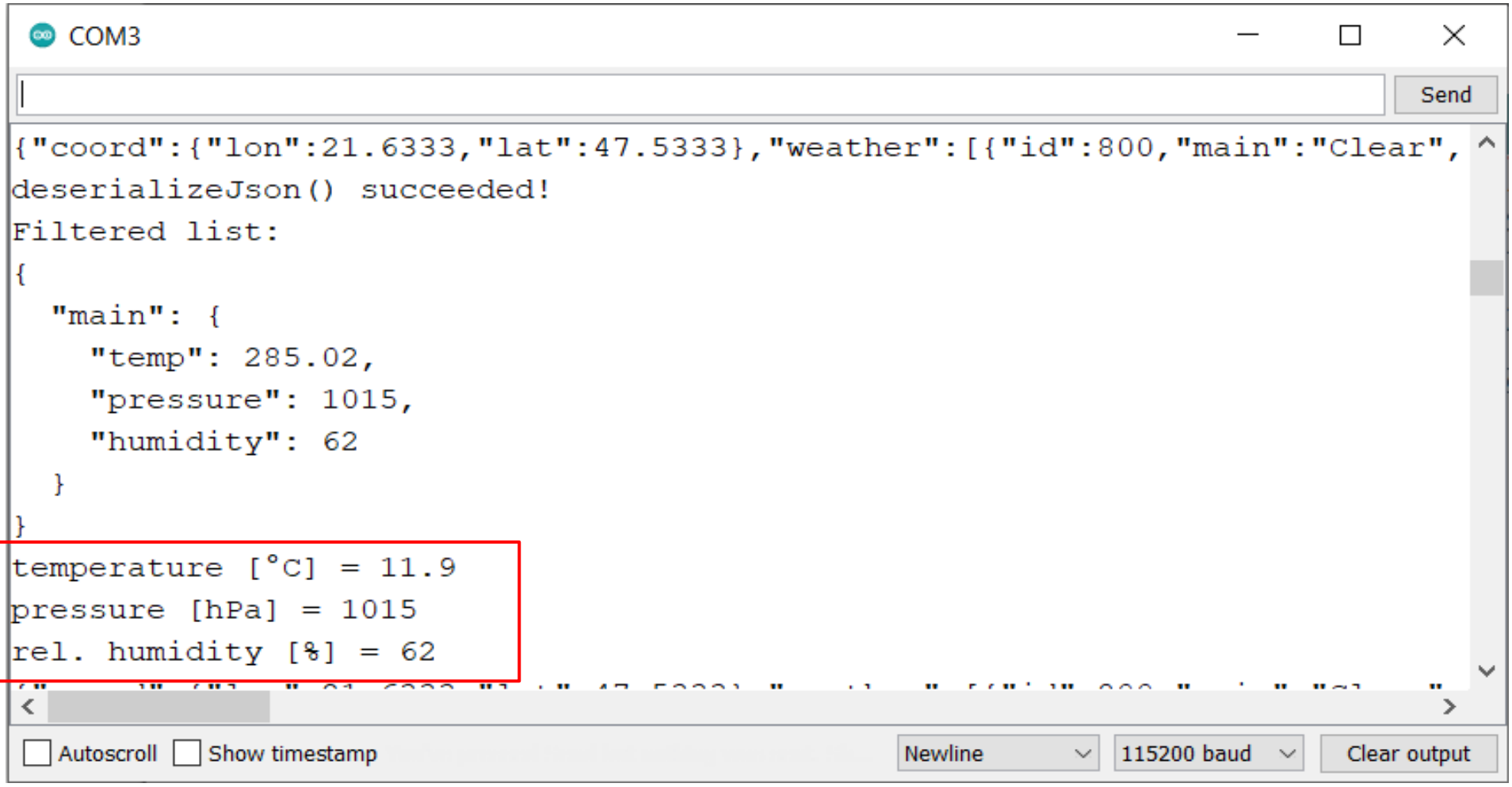
```
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>
#include "secrets.h"

StaticJsonDocument<500> doc;     // A deszerializált adatokat tartalmazza
StaticJsonDocument<200> filter; // A szűrési feltételeket tartalmazza
String server_url = "http://api.openweathermap.org/data/2.5/weather?
q=Debrecen&mode=json&APPID=" + OPENWEATHERMAP_APPID;

void setup () {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }

  // Prepare filter
  filter["main"]["temp"] = true;
  filter["main"]["pressure"] = true;
  filter["main"]["humidity"] = true;
}
```

ESP32_OpenWeatherMap.ino 2/2.

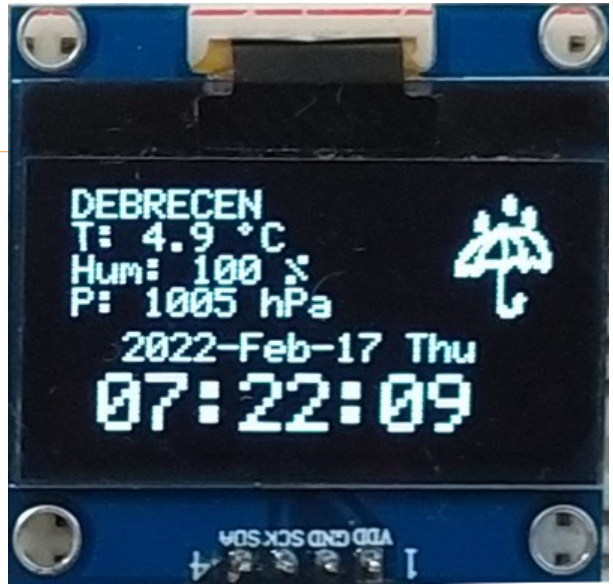
```
void loop() {
  if (WiFi.status() == WL_CONNECTED) {      // Check WiFi connection status
    HTTPClient http;                        // Declare an object of class HTTPClient
    http.begin(server_url);                 // Specify request destination
    int httpCode = http.GET();              // Send the request
    if (httpCode > 0) {                     // Check the returning code
      String payload = http.getString();    // Get the request response payload
      Serial.println(payload);              // Print the response payload
      DeserializationError error = deserializeJson(doc, payload,DeserializationOption::Filter(filter));
      if (error) {
        Serial.print(F("deserializeJson() failed: "));
        Serial.println(error.f_str());
      } else {
        float tempK = doc["main"]["temp"];
        int hPa = doc["main"]["pressure"];
        int relHum = doc["main"]["humidity"];
        Serial.println(F("deserializeJson() succeeded! \r\nFiltered list:"));
        serializeJsonPretty(doc, Serial);
        Serial.print("\r\ntemperature [°C] = "); Serial.println(tempK - 273.16, 1);
        Serial.print("pressure [hPa] = ");       Serial.println(hPa);
        Serial.print("rel. Humidity [%] = ");    Serial.println(relHum);
      }
    }
    http.end();     // Close connection
  }
  delay(10000);     // Send a request every 10 seconds
}
```

ESP32_OpenWeatherMap.ino

- A program futási eredménye az alábbi ábrán látható
- Nyomkövetési céllal az eredet nyers adatsort és a megszűrt **JSON** adatsort is kiírattuk – a gyakorlati alkalmazáshoz ez nyilván fölösleges



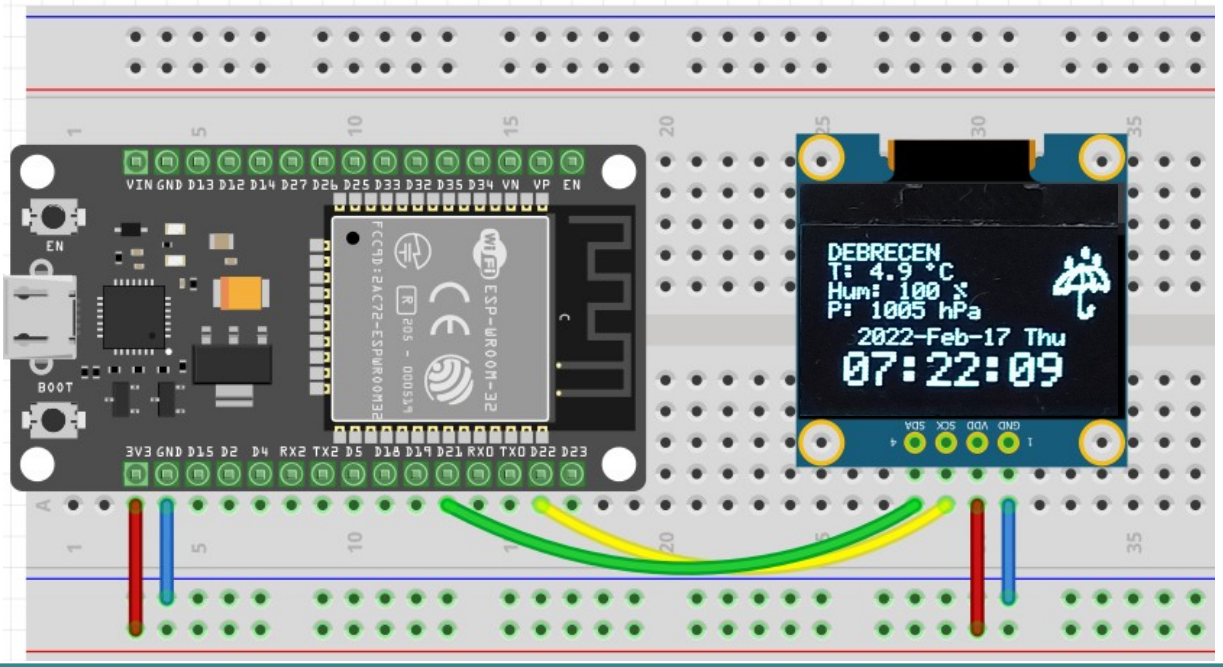
Időjárásjelző óra

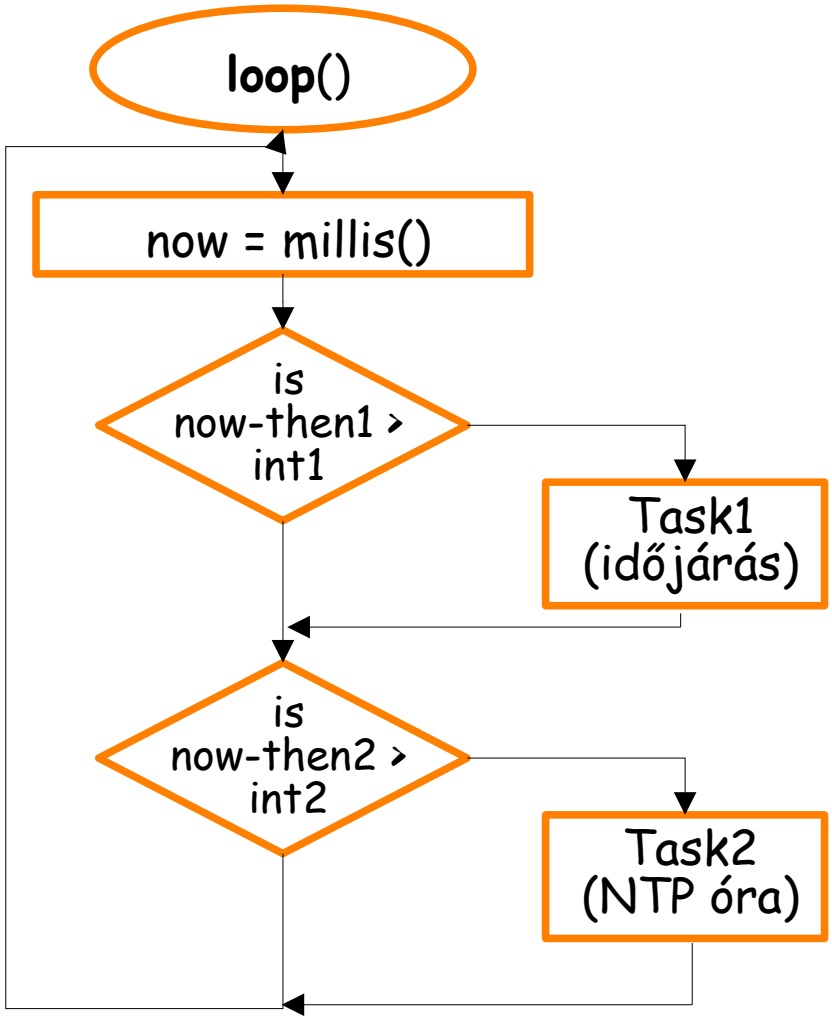


- Használjuk fel az előző előadásban bemutatott OLED kijelzőt az időjárási adatok kijelzésére!
- Az előző előadáshoz hasonlóan, most is az OLED-SSD1306-SH1106 programkönyvtárat használjuk a kijelző kezeléséhez
- Fokozzuk az élvezetet: a letöltött adatokból a **weather** szekcióból használjuk fel az ***'id'*** elemet és jelenítsünk meg a időjárást jellemző 32x32 képpont méretű ikonokat
  - Az időjárást jellemző kódok táblázata az Weather Conditions oldalon található. Például id = 800 jelentése *'tiszta idő'*
  - A programban felhasznált ikonokat az IconArchive adatbázisból töltöttük le, és a Marlin bitmap konverter segítségével konvertáltuk
- Ha 128x64 méretű kijelzőnk van, akkor annak alsó felére az előző előadásban szereplő NTP órát is odavarázsolhatjuk
- A periodikus NTP és HTTP lekéréseket a *millis()* segítségével ütemezzük

Bekötési vázlat, folyamatábra

- Az alapértelmezett I2C lábakat használjuk (**SDA – GPIO21, SCL – GPIO22**), bár ennek itt nincs jelentősége, mert az általunk választott OLED könyvtár szoftveres **I2C** kezelést használ
- A **loop**() függvény két feladatot végez ütemezetten: az időjárási adatok, ill. a pontos idő lekérését és megjelenítését





ESP32_OpenWeatherMap_oled.ino 4/1.

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>
#include <time.h>
#include "oled.h"
#include "icons.h"
#include "secrets.h"

//OLED display = OLED(21,22,NO_RESET_PIN,0x3C,128,64, false); // SSD1306
OLED display = OLED(21, 22, NO_RESET_PIN, 0x3C, 128, 64, true); // SH1106

StaticJsonDocument<500> doc;
StaticJsonDocument<200> filter;
String server_url = "http://api.openweathermap.org/data/2.5/weather?
q=Debrecen&mode=json&APPID=" + OPENWEATHERMAP_APPID;
static char msg[30];      // character buffer

// TimeZone rule for Europe/Budapest including daylight adjustment rules (optional)
// See at: https://leo.leung.xyz/wiki/Timezone
const char* time_zone = "CET-1CEST,M3.5.0,M10.5.0/3";
const char* ntpServer = "hu.pool.ntp.org";
struct tm timeinfo;
unsigned long then1, then2;     // Time of last events
#define INTERVAL1    30000UL    // Half minute intervals for OpenWeatherMap API calls
#define INTERVAL2     1000UL    // One second intervals for the clock update
```

ESP32_OpenWeatherMap_oled.ino 4/2.

```
void setup () {
  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(".");
  }
  then1 = millis();
  then2 = then1;
  configTzTime(time_zone, ntpServer);
  display.begin();
  display.clear();
  display.display();
  // Prepare filter
  filter["weather"][0]["id"] = true;
  filter["main"]["temp"] = true;
  filter["main"]["pressure"] = true;
  filter["main"]["humidity"] = true;
}
```

ESP32_OpenWeatherMap_oled.ino 4/3.

```
void loop() {
  unsigned long now = millis();
  if (WiFi.status() == WL_CONNECTED) {          // Check WiFi connection status
    if ((now - then1) > INTERVAL1) {            // Every 30 sec...
      HTTPClient http;                          // Declare an object of class HTTPClient
      http.begin(server_url);                   // Specify request destination
      int httpCode = http.GET();                // Send the request
      if (httpCode > 0) {                       // Check the returning code
        String payload = http.getString();      // Get the request response payload
   DeserializationError error = deserializeJson(doc, payload, DeserializationOption::Filter(filter));
        if (error) {                            // Test if parsing succeeds.
          Serial.print(F("deserializeJson() failed: ")); Serial.println(error.f_str());
        } else {
          int id = doc["weather"][0]["id"];  // <== Weather condition code
          float tempK = doc["main"]["temp"];
          int hPa = doc["main"]["pressure"];
          int relHum = doc["main"]["humidity"];
          display.draw_rectangle(0, 0, 127, 31, OLED::SOLID, OLED::BLACK);
          display.draw_string(2, 0, "DEBRECEN");
          display.printf(2, 8, "T: %.1f°C", tempK - 273.16);
          display.printf(2, 16, "Hum: %d %%", relHum);
          display.printf(2, 24, "P: %d hPa", hPa);
          drawIcon(96, 0, 32, 32, id, OLED::WHITE); display.display();
        }
      }
      http.end();                                   //Close connection
      then1 += INTERVAL1;
    }
```

Disply in upper half

ESP32_OpenWeatherMap_oled.ino 4/4.

```
    if ((now - then2) > INTERVAL2) {     // In every second
      if (getLocalTime(&timeinfo)) {
        display.draw_rectangle(0, 32, 127, 63, OLED::SOLID, OLED::BLACK);
        strftime (msg, 16, "%Y-%b-%d %a ", &timeinfo);
        display.draw_string(15, 35, msg);            // Display date
        strftime (msg, 10, "%T ", &timeinfo);        // Display time
        display.draw_string(8, 45, msg, OLED::DOUBLE_SIZE);
        display.display();                           // Refresh screen
      }
      then2 += INTERVAL2;
} } }

void drawIcon(int16_t x, int16_t y, int16_t w, int16_t h, int id, OLED::tColor color) {
  const unsigned char* bitmap;
  int16_t byteWidth = (w + 7) / 8;  // Bitmap scanline pad = whole byte
  uint8_t byte = 0;
  if (id == 800)  bitmap = bitmap_sun;
    else if ((id >= 200) && (id < 300)) bitmap = bitmap_lightning;
      else if ((id > 800) && (id < 900))  bitmap = bitmap_cloud;
        else bitmap = bitmap_umbrella;
  for (int16_t j = 0; j < h; j++, y++) {
    for (int16_t i = 0; i < w; i++) {
      if (i & 7) byte <<= 1;
      else byte = (*(const unsigned char *)(&bitmap[j * byteWidth + i / 8]));
      if (byte & 0x80) display.draw_pixel(x + i, y, color);
    }
  }
}
```

Ikon kiválasztás (több kellene!)

A **drawIcon** függvény csak azért kell, mert az ikonjaink bitsorrendje nem egyezik meg azzal, amit az **oled.draw_bitmap()** tagfüggvény elvárna

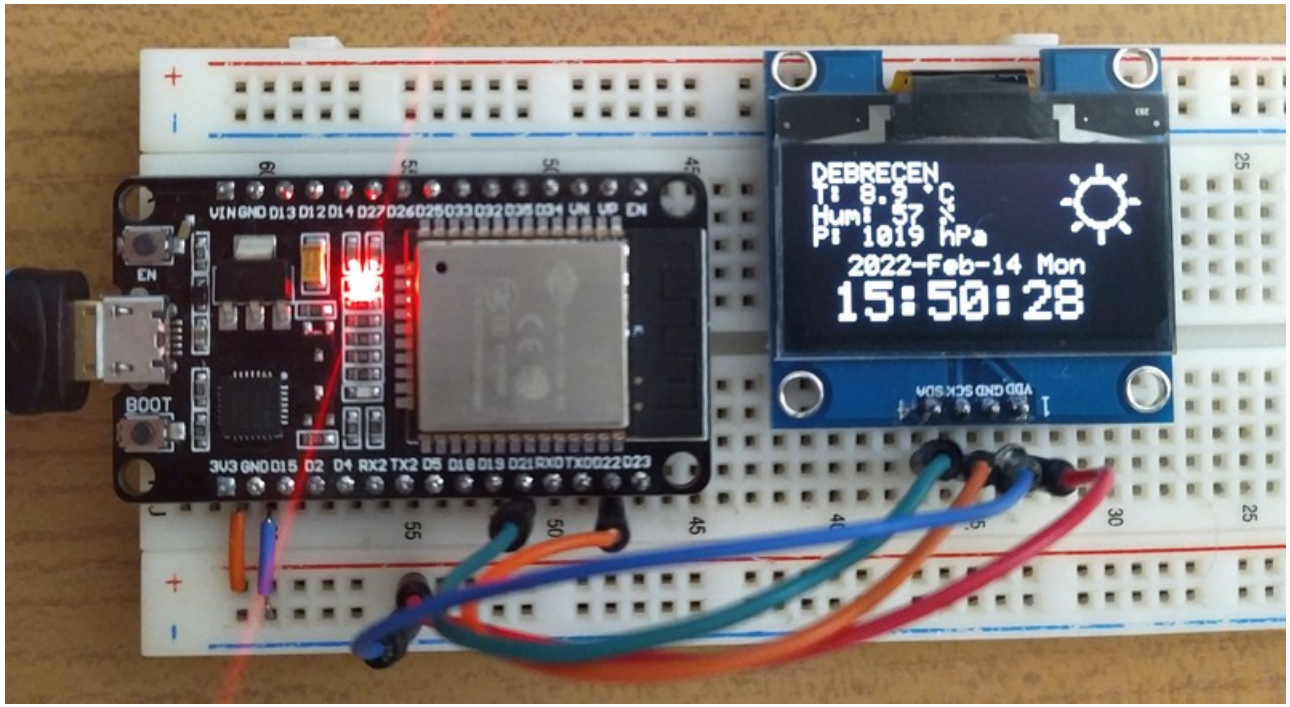
icons.h (részlet)

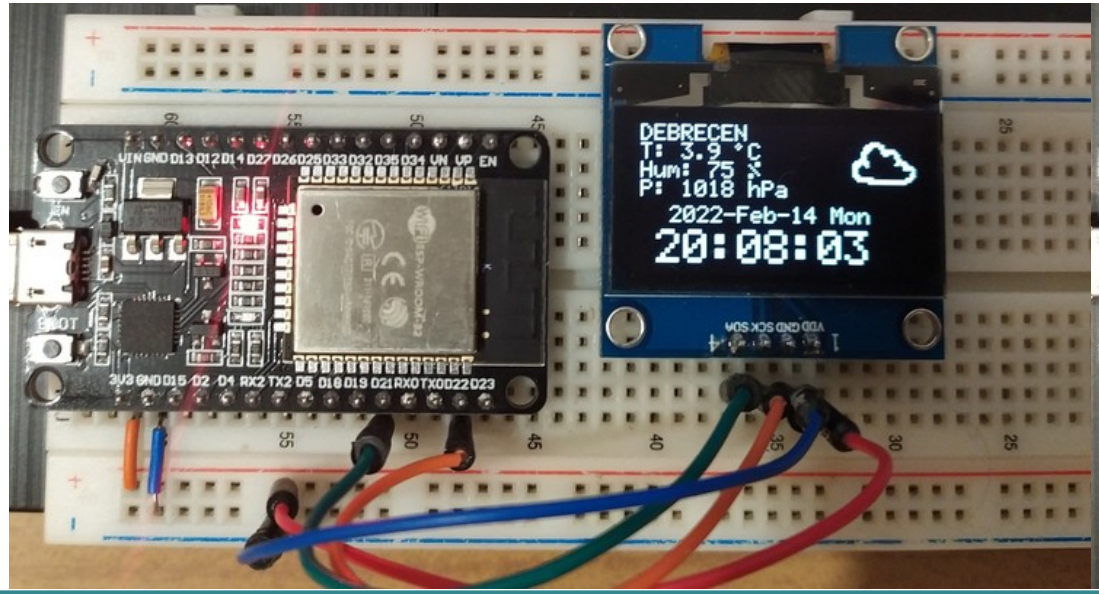
```
const unsigned char bitmap_sun[] PROGMEM = {
  0x00,0x00,0x00,0x00, // ................................
  0x00,0x01,0x80,0x00, // ...............##...............
  0x00,0x01,0x80,0x00, // ...............##...............
  0x00,0x01,0x80,0x00, // ...............##...............
  0x00,0x01,0x80,0x00, // ...............##...............
  0x06,0x00,0x00,0xC0, // .....##..................##......
  0x07,0x0F,0xF1,0xC0, // .....###....########...###......
  0x03,0x1F,0xF9,0x80, // ......##...##########..##.......
  0x00,0x70,0x0E,0x00, // .........###........###.........
  0x00,0xE0,0x07,0x00, // ........###..........###........
  0x00,0xC0,0x03,0x00, // ........##............##........
  0x01,0x80,0x01,0x80, // .......##..............##.......
  0x01,0x80,0x01,0x80, // .......##..............##.......
  0x01,0x00,0x00,0x80, // .......#................#.......
  0x3B,0x00,0x00,0xDC, // ..###.##................##.###..
  0x7B,0x00,0x00,0xDE, // .####.##................##.####.
  0x03,0x00,0x00,0xC0, // ......##................##.......
  0x01,0x80,0x01,0x80, // .......##..............##.......
  0x01,0x80,0x01,0x80, // .......##..............##.......
  0x01,0x80,0x01,0x80, // .......##..............##.......
  0x00,0xC0,0x03,0x00, // ........##............##........
  0x00,0x60,0x06,0x00, // .........##..........##.........
  0x00,0x38,0x1C,0x00, // ...........###......###...........
  0x03,0x9F,0xF9,0x80, // ......###..##########..##.......
  0x07,0x07,0xE1,0xC0, // .....###.....######....###......
  0x06,0x00,0x00,0xE0, // .....##..................###.....
  0x00,0x01,0x80,0x60, // ...............##........##.....
  0x00,0x01,0x80,0x00, // ...............##...............
  0x00,0x01,0x80,0x00, // ...............##...............
  0x00,0x01,0x80,0x00, // ...............##...............
  0x00,0x01,0x80,0x00, // ...............##...............
  0x00,0x00,0x00,0x00  // ................................
};
```

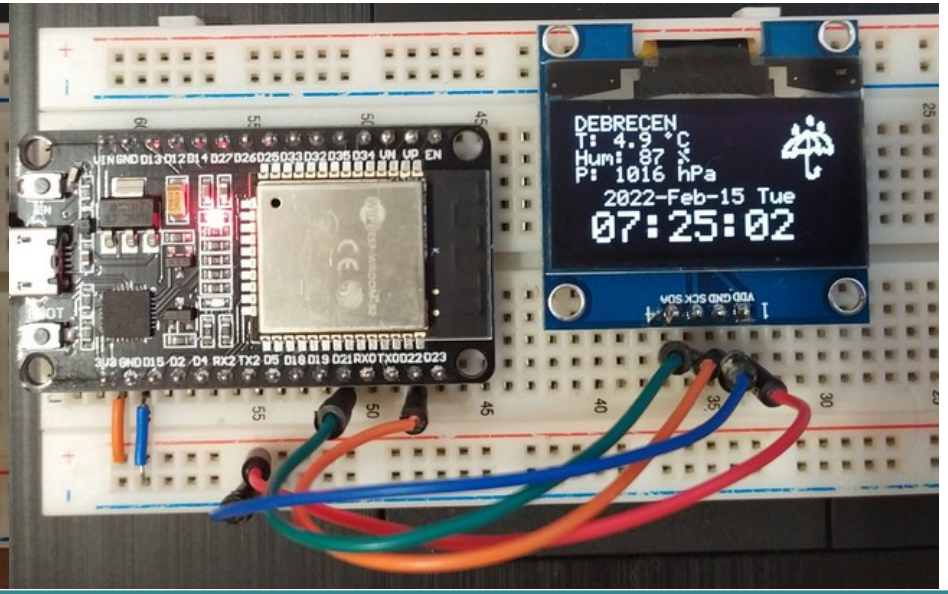
- Mindegyik ikon 32x32 képpont, azaz 4x32 bájt
- Sajnos, a bitek sorrendje nem egyezik meg azzal, amit az OLED kijelzőt kezelő programkönyvtár vár, így nem tudjuk használni a meglevő **draw_bitmap()** függvényt
- A probléma gyors áthidalására készítettük a **drawIcon()** függvényt, ami az **oled.draw_pixel()** függvény segítségével képpontonként rajzolja ki az ikont

ESP32_OpenWeatherMap_oled.ino futási eredmény

- Hétfőn délután napos idő volt
- Estére felhők gyülekeztek
- Kedd reggel eleredt az eső

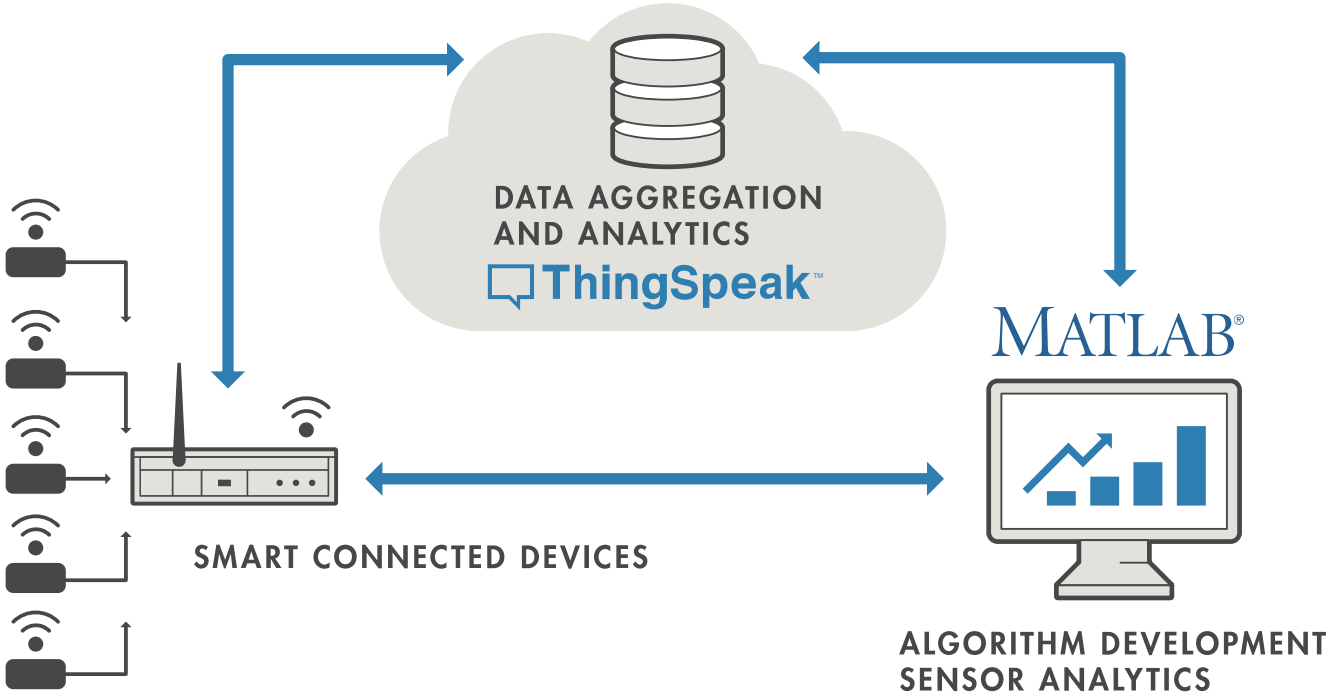






Thingspeak – IoT felhő

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

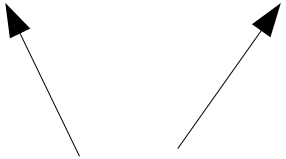


Adatküldés HTTP GET protokollal

- Ha regisztráltunk, akkor létrehozhatunk csatornákat, amelyeknek egyedi azonosítója van. Például: thingspeak.com/channels/34 244
- 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&filed2=adat2
```

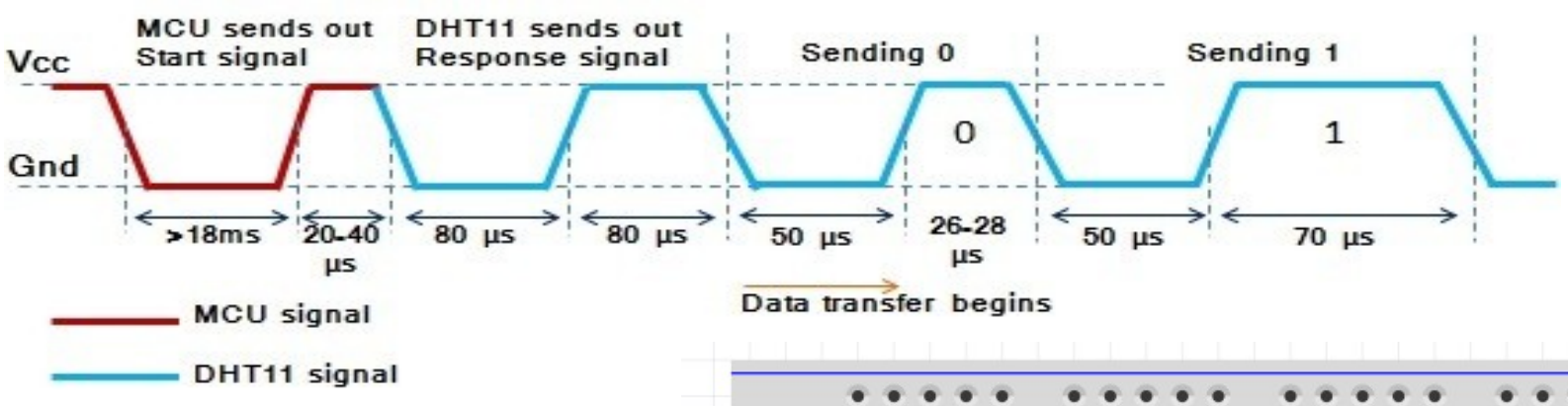
```
#include <WiFi.h>
#include <HTTPClient.h>
HTTPClient http;
String server_request = "http://api.thingspeak.com/update?\
        api_key=DVDXXXXXXXETD&field1=22&field2=44"
. . .
http.begin(server_request);
int httpCode = http.GET();
```



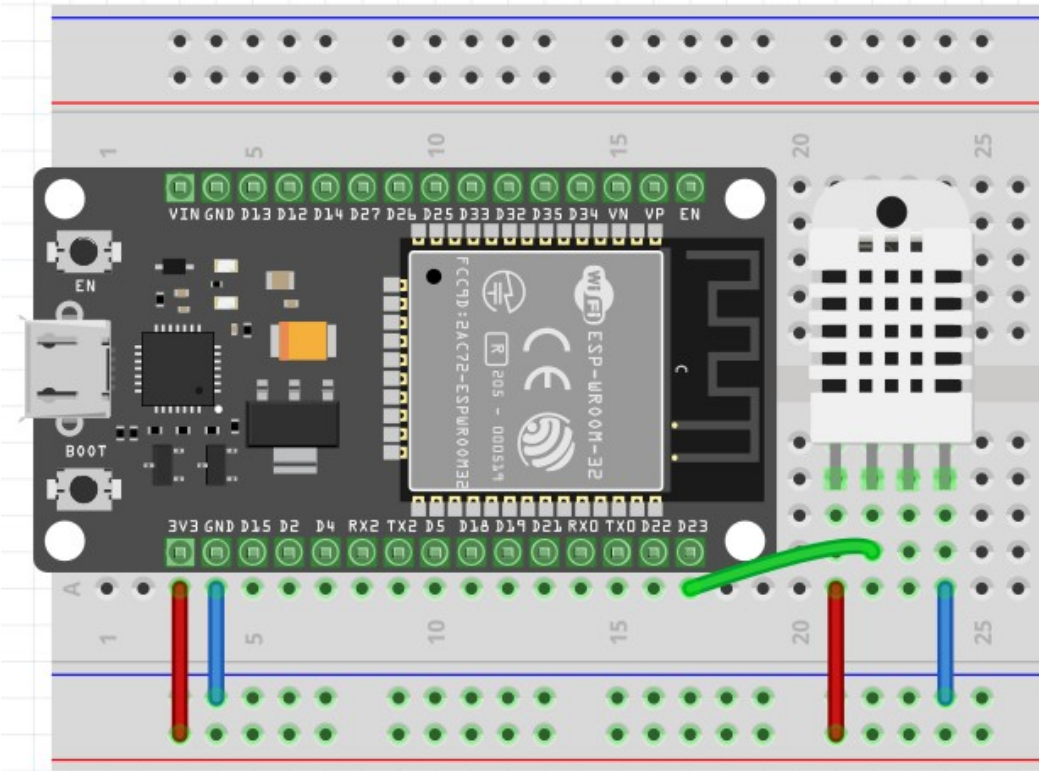
DHT11/DHT22 szenzorok

- A **DHT22** szenzor hőmérséklet és relatív páratartalom mérésére szolgál

`DH22 DATA = 16-bits RH data + 16-bits Temperature data + 8-bits checksum`



- Tápfeszültség: 3.3-5.5V
- Mérési tartomány:
  - RH = 0 – 100 %
  - Temp = –40 – 80 °C
- Felbontás:
  - RH = 0.1 % (DHT22)
  - Temp = 0.1 °C (DHT22)



ESP32_ThingSpeak.ino 2/1.

- Egy **DHT22** szenzor adatait küldjük a **ThingSpeak** szerverre

Az Adafruit DHT és az Adafruit unified sensor könyvtárakat telepítsük hozzá!

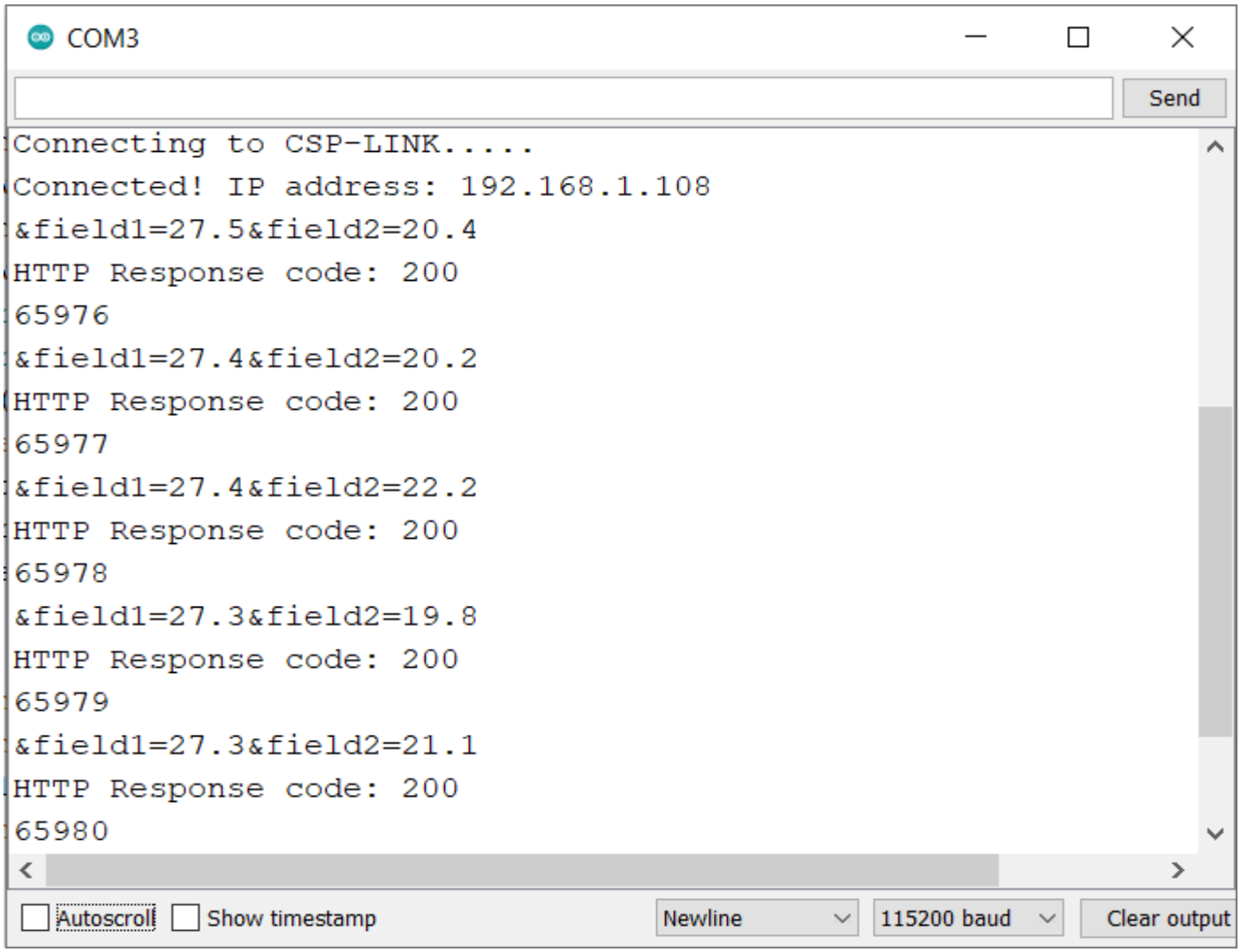
```
#include <WiFi.h>
#include <HTTPClient.h>
#include "DHT.h"
#include "secrets.h"
#define DHTPIN          23     // GPIO23 pin for DHT sensor
DHT dht(DHTPIN, DHT22);
float t = 0.0;                 // temperature
float h = 0.0;                 // humidity
String server_url = "http://api.thingspeak.com/update?api_key=" +
THINGSPEAK_WRITE_APIKEY;
void setup () {
  Serial.begin(115200);
  dht.begin();
  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());
}
```

ESP32_ThingSpeak.ino 2/2.

```
void loop() {
  float tnew = dht.readTemperature();
  if (!isnan(tnew)) t = tnew;
  float hnew = dht.readHumidity();
  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) {     // Check WiFi connection status
    HTTPClient http;                       // Declare an object of class HTTPClient
    String server_request = server_url + p1 + p2;
    http.begin(server_request);            // Specify request destination and fields
    int httpCode = http.GET();             // Send the request
    if (httpCode > 0) {                    // Check the returning code
      Serial.print("HTTP Response code: ");
      Serial.println(httpCode);
      String payload = http.getString();   // Get the request response payload
      Serial.println(payload);             // Print the response payload
    }
    else {
      Serial.print("Error code: ");
      Serial.println(httpCode);
    }
    http.end();                            // Close connection
  }
  delay(20000);                            // Send a request every 20 seconds
}
```

ESP32_ThingSpeak.ino

- Szerencsés esetben az alábbihoz hasonló kiírásokat látunk
- A válaszkódok ismertetése itt: HTTP response status codes



Az eredmény megtekintése böngészőben

ThingSpeak™ Channels Apps Devices Support

Pista szoba

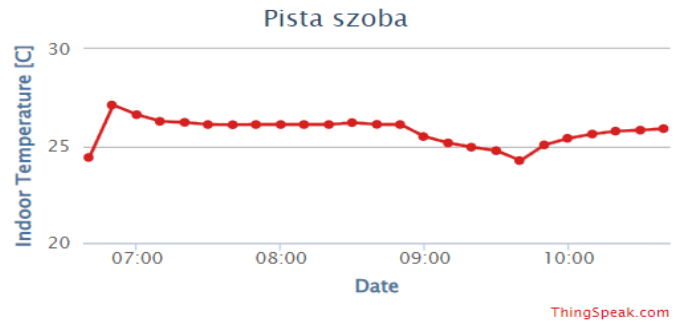
Channel ID: **34244**
Author: icserny
Access: Public

Experimental channel running in the room of Pista.
Hadware: DHT22 sensor + ESP32 WiFi module.
Software: Arduino IDE
dht22, esp32, arduino

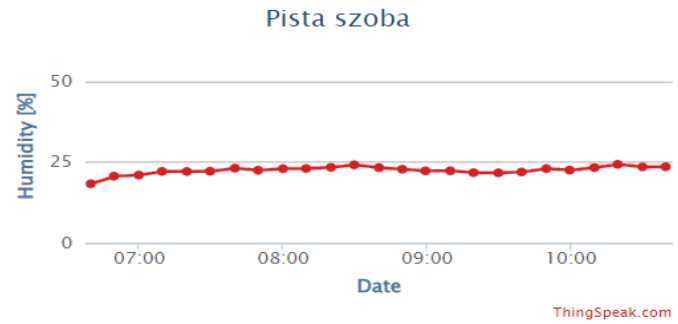
Channel Stats

Created: 6 years ago
Last entry: less than a minute ago
Entries: 66642

Field 1 Chart



Field 2 Chart

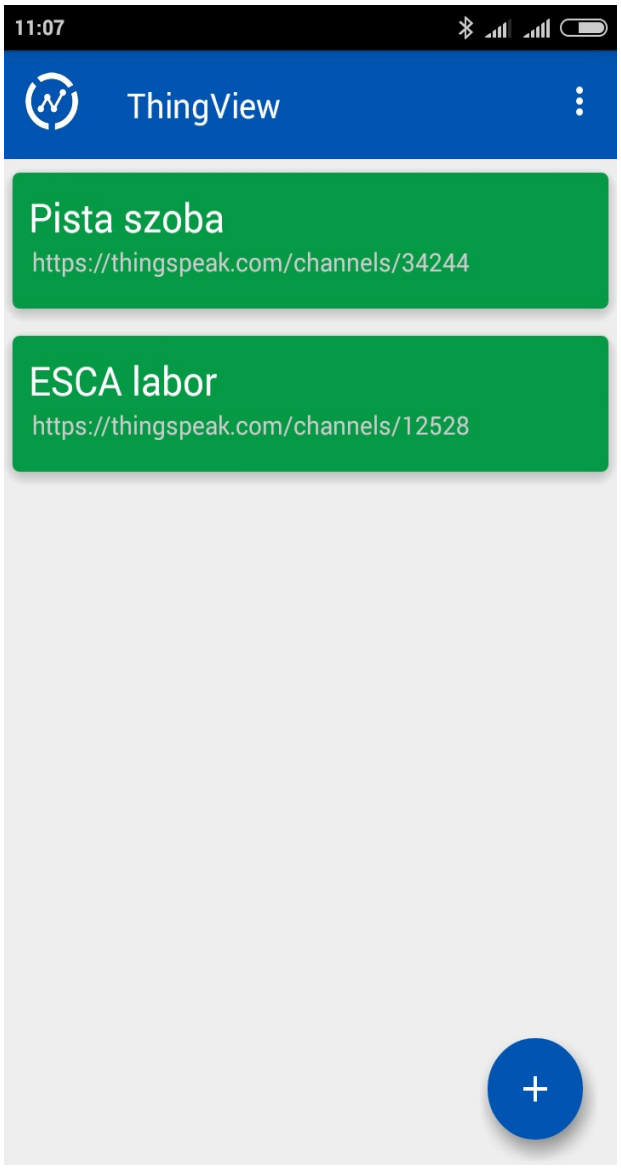


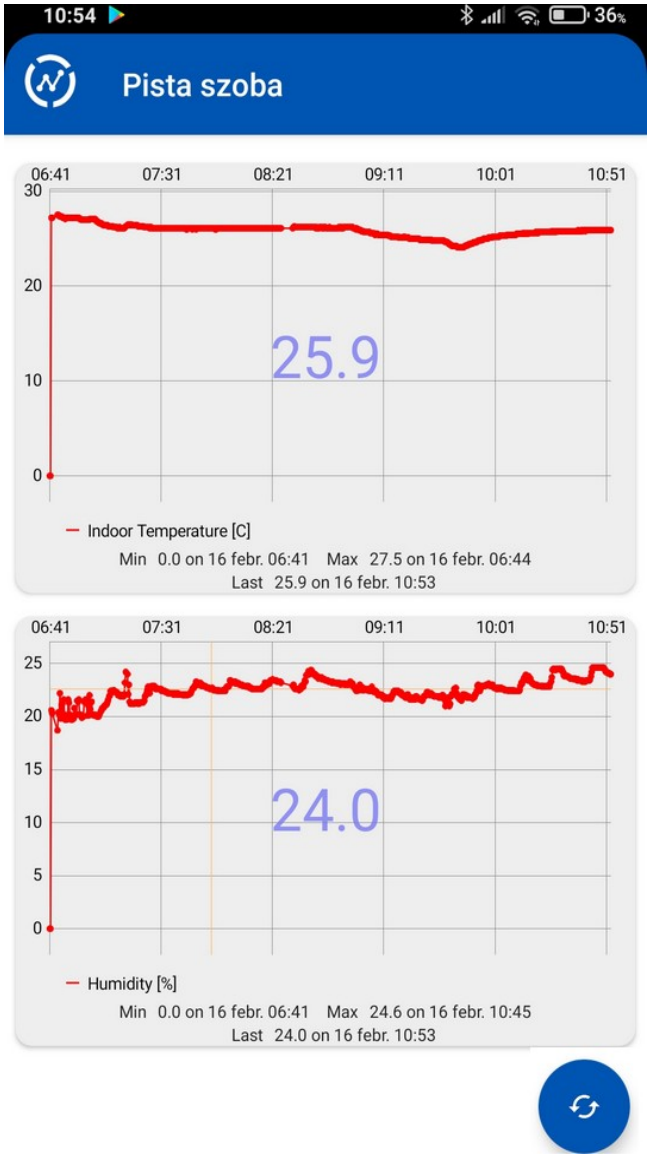
Channel Location

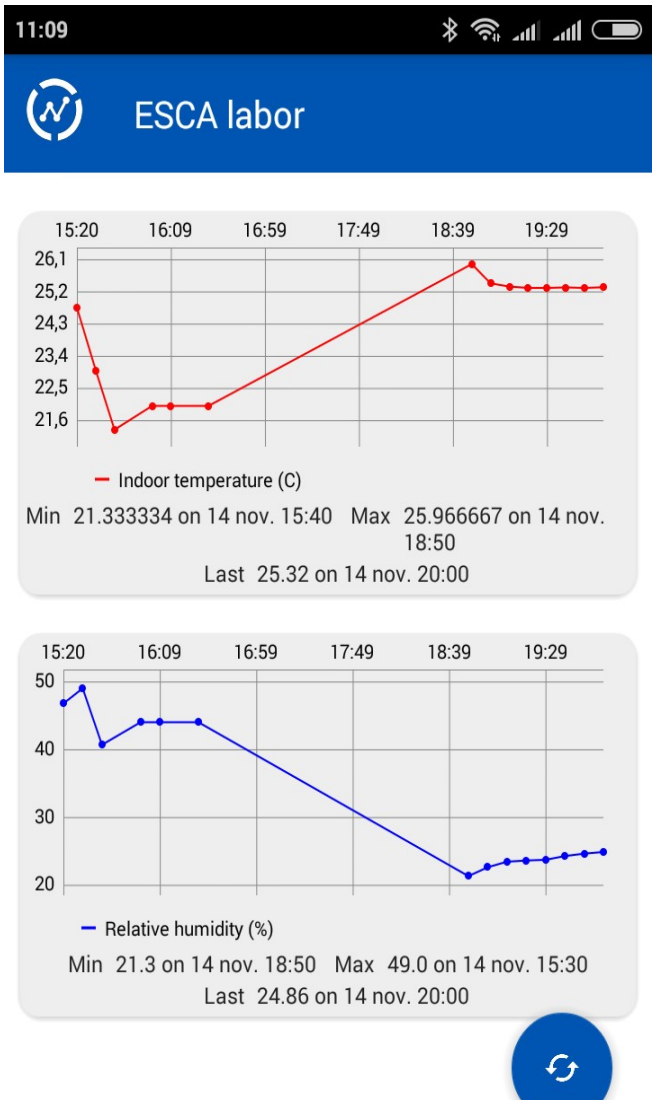
Channel Status Updates

Az eredmény megtekintése applikációban

- **ThingView** - ThingSpeak megjelenítő (Google Play áruház)







Adatküldés HTTP POST protokollal

- Részletek egy képzeletbeli **ThingSpeak** kliens programból

```
#include <WiFi.h>
#include <HTTPClient.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!

ESP32_ThingSpeak_POST.ino 2/1.

- **DHT22** szenzor adatait POST metódussal küldjük **ThingSpeak** szerverre

```
#include <WiFi.h>
#include <HTTPClient.h>
#include "DHT.h"                // Adafruit DHT library
#include "secrets.h"            // Wifi jelszó, ThingSpeak Write API key
#define DHTPIN          23      // GPIO23 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";
String apiKey = String("api_key=") + THINGSPEAK_WRITE_APIKEY;
void setup() {
  Serial.begin(115200);         // Soros port inicialálás
  dht.begin();                  // DHT szenzor indítása
  WiFi.mode(WIFI_STA);          // Station (client) mód
  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());
}
```

ESP32_ThingSpeak_POST.ino 2/2.

```
void loop() {
  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: 200-as kód
    http.end();                                // Erőforrások felszabadítása
  }
  else {
    Serial.println("WiFi Disconnected");
  }
  delay(20000);
}
```

JSON formátumú adatküldés ThingSpeak-re

- A **ThingSpeak** szerverre JSON formátumban is elküldhetjük az adatainkat, ekkor:
  - Az API szervercím: **https://api.thingspeak.com/update.json**
  - A **Content-Type** elem értéke: **application/json**
  - A **httpRequestData** tartalma pedig egy **JSON** objektum (itt formázva mutatjuk, de serializált formában küldjük) :

```
{
  "api_key": "**************",
  "field1": 26,
  "field2": 24.6
}
```

- Az elküldendő **JSON** objektum opcionális adatokkal is kiegészíthető (*created_at, lat, long, elevation* stb.)
- A **JSON** formátumú adatküldés fő előnye - az **ArduinoJson** programkönyvtárnak köszönhetően - az áttekinthetőbb kód

ESP32_ThingSpeak_JSON.ino 2/1.

- A **DHT22** szenzor adatait **POST** metódussal, **JSON** formátumban küldjük a **ThingSpeak** szerverre (Update channel data with HTTP GET or POST)

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>
#include "DHT.h"                // Adafruit DHT library
#include "secrets.h"            // Wifi jelszó, ThingSpeak Write API key
#define DHTPIN          23      // GPIO23 pin a DHT szenzorhoz

DHT dht(DHTPIN, DHT22);
StaticJsonDocument<500> doc;
const char* serverName = "https://api.thingspeak.com/update.json"; // formátumfüggő cím!

void setup() {
  Serial.begin(115200);         // Soros port inicialálás
  dht.begin();                  // DHT szenzor indítása
  WiFi.mode(WIFI_STA);          // Station (client) mód
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  while (WiFi.status() != WL_CONNECTED) {
    Delay(500); Serial.print(".");
  }
  doc["api_key"] = THINGSPEAK_WRITE_APIKEY;
  doc["field1"] = 0.0;          // Hőmérséklet Celsius fokokban
  doc["field2"] = 0.0;          // Relatív páratartalom [%]
}
```

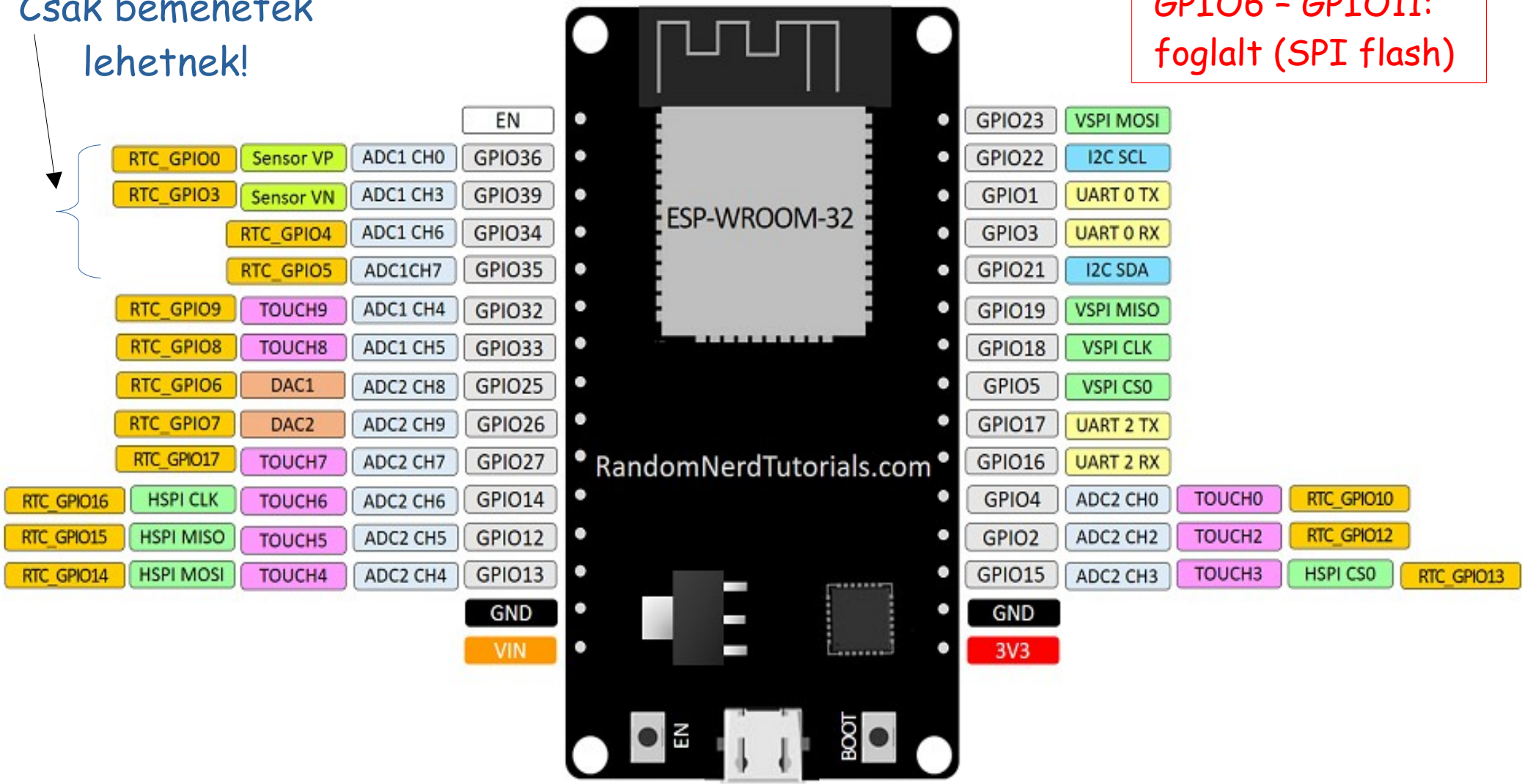
ESP32_ThingSpeak_POST.ino 2/2.

```
void loop() {
  String httpRequestData;
  float tnew = dht.readTemperature();          // Hőmérséklet kiolvasása
  if (!isnan(tnew)) doc["field1"]  = tnew;
  float hnew = dht.readHumidity();             // Páratartalom kiolvasása
  if (!isnan(hnew)) doc["field2"]  = hnew;
  serializeJson(doc, httpRequestData);
  // serializeJsonPretty(doc, Serial);

  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/json");
    int httpResponseCode = http.POST(httpRequestData); // adatok kiküldése
    Serial.print("HTTP Response code: ");      // Válaszkód kiírása
    Serial.println(httpResponseCode);          // Sikeres küldés: 200-as kód
    http.end();                                // Erőforrások felszabadítása
  }
  else {
    Serial.println("WiFi Disconnected");
  }
  delay(20000);
}
```

A DOIT ESP32 Devkit-1 kártya kivezetései

Csak bemenetek lehetnek!

GPIO6 - GPIO11: foglalt (SPI flash)



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

Ellenállás színkódok