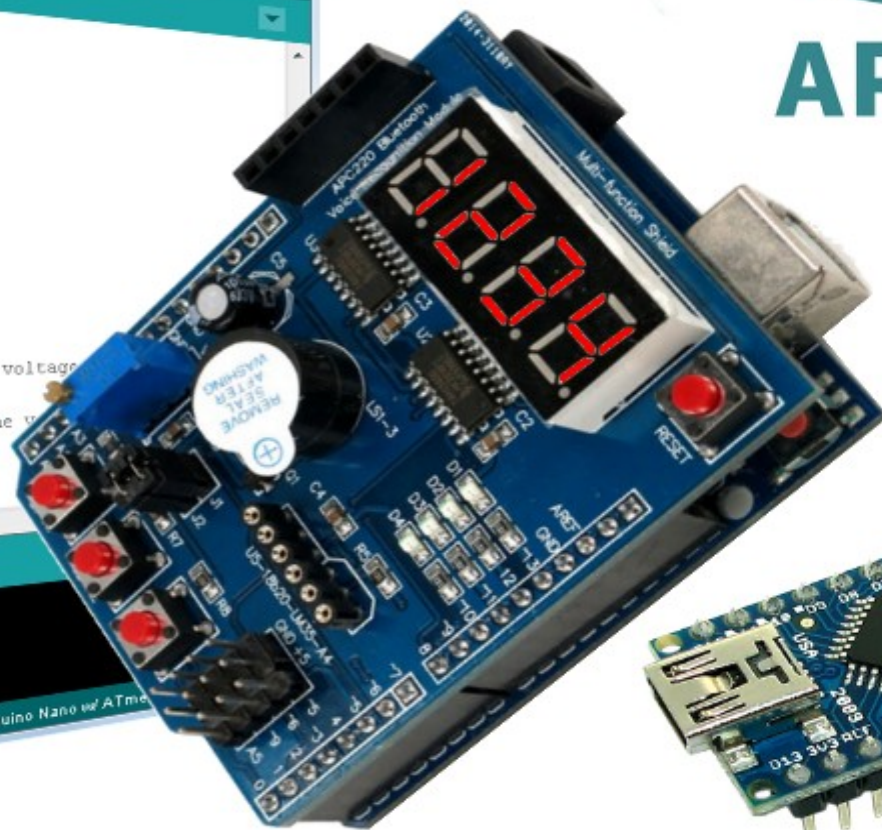
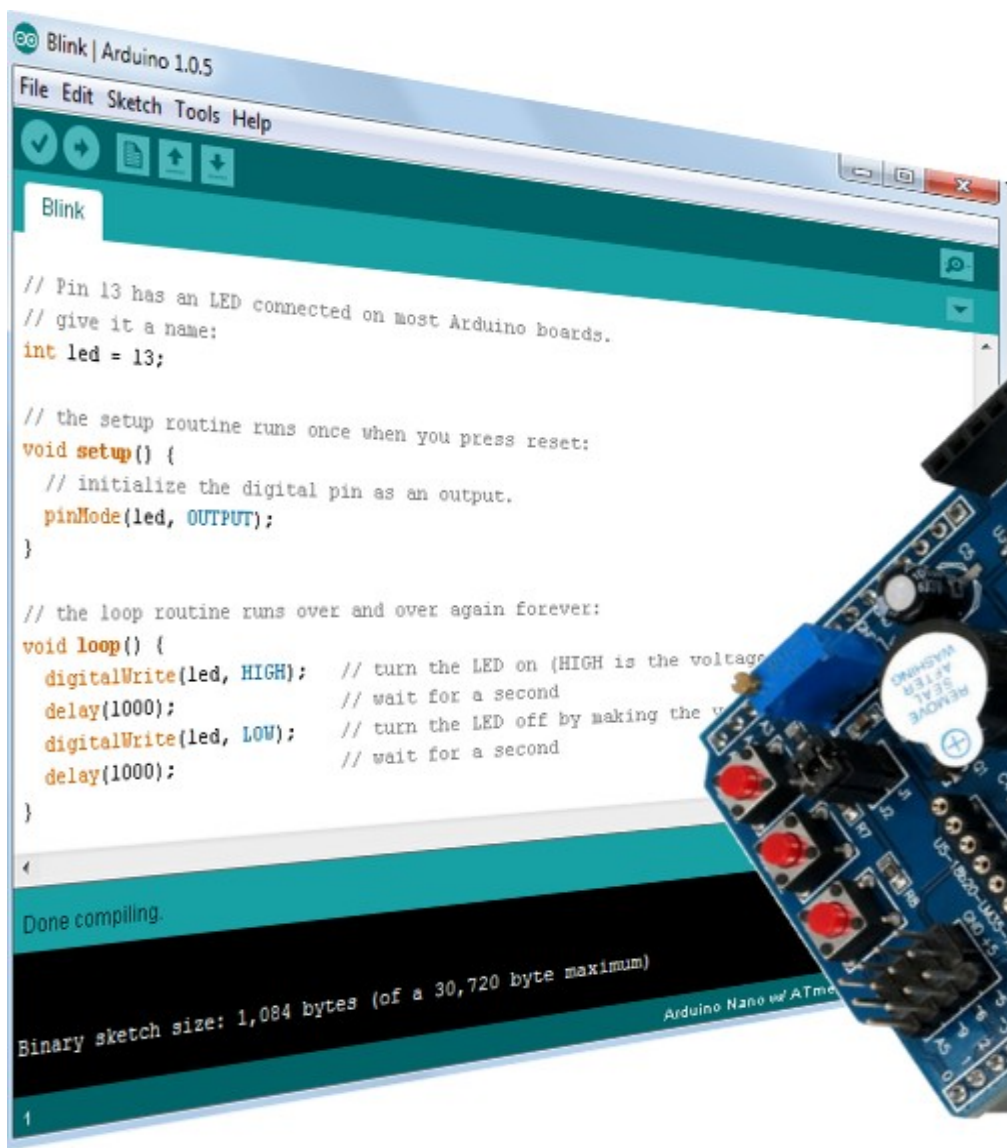


Arduino tanfolyam kezdőknek és haladóknak

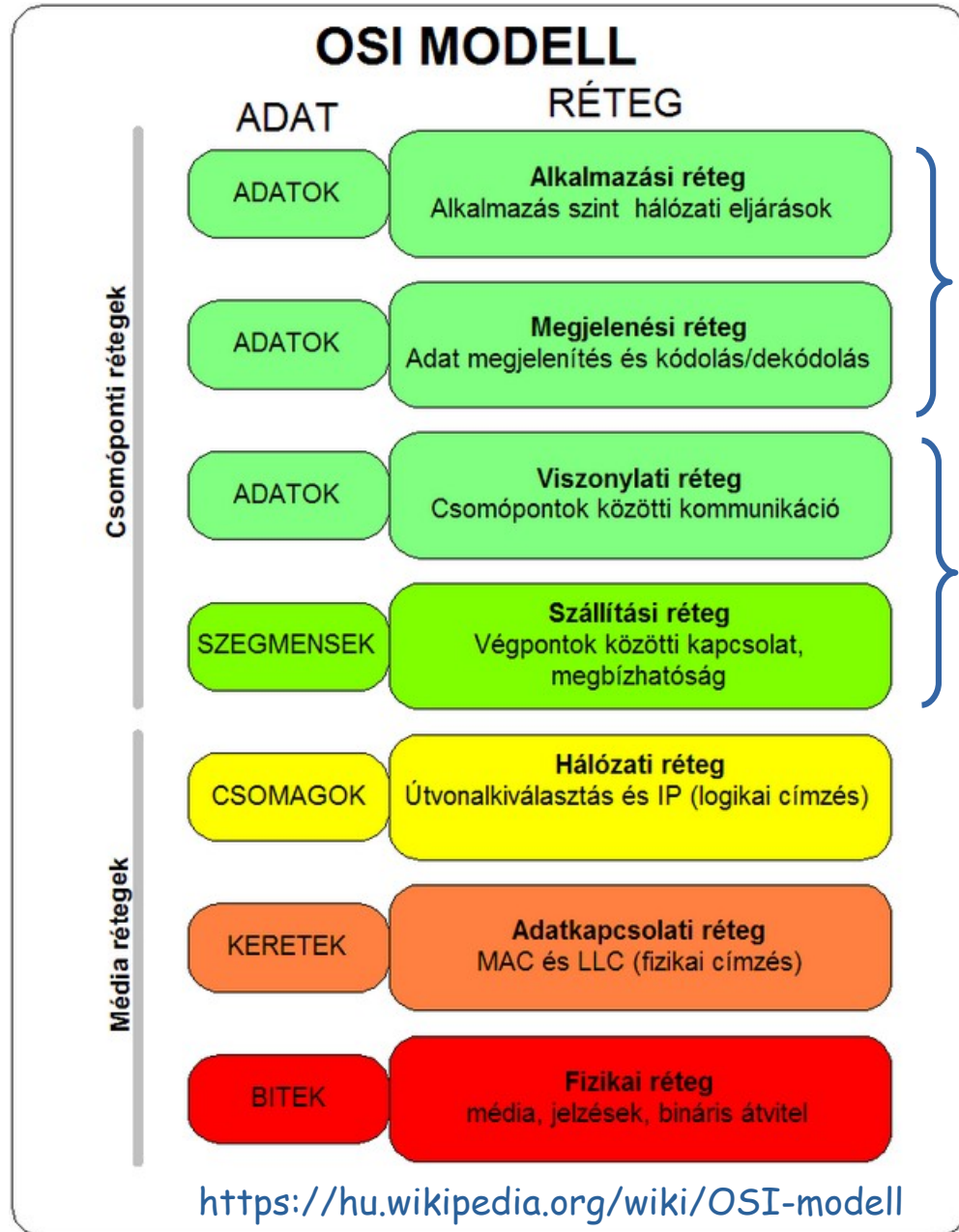


13. ESP8266 webkliens alkalmazások

Felhasznált és ajánlott irodalom

- LETSCODE.HU: [Hogyan működik az Internet](#)
- 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](#)

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.

Csatlakozás a WiFi hálózathoz

- Az **ESP8266** kliensként történő használatához csatlakoztatnunk kell a meglévő WiFi hálózatunkra
- A kiíratás természetesen opcionális, a **Serial** kezdetű sorokat elhagyhatjuk

```
#include <ESP8266WiFi.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** almappjában helyeztünk el

secrets.h

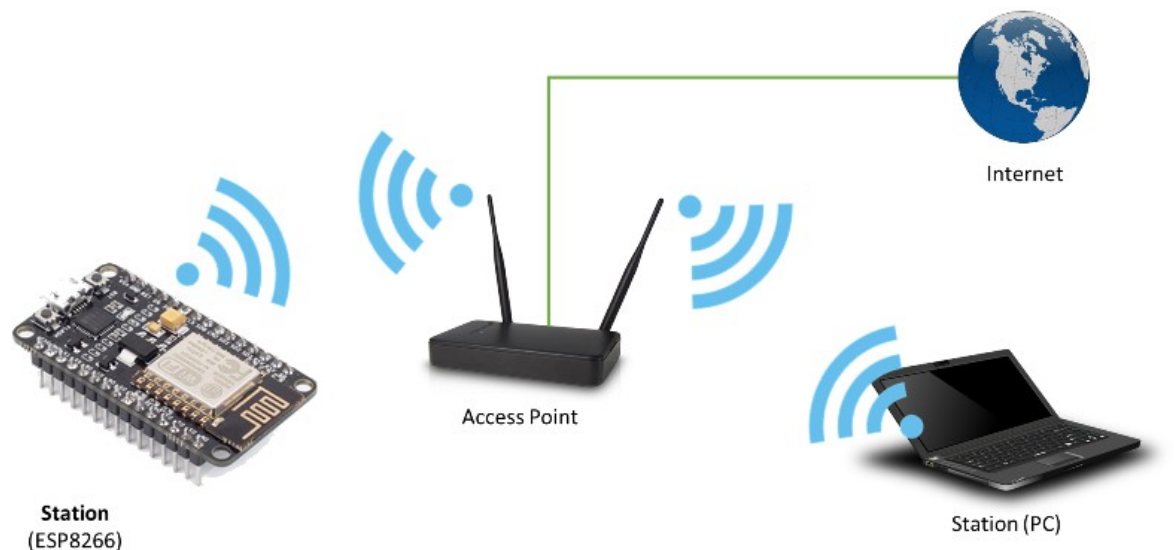
```
#define WIFI_SSID MY_SSID
#define WIFI_PASS MY_PASSWORD

String OPENWEATHERMAP_APPID =
"*****";

String THINGSPEAK_WRITE_APIKEY =
"xxxxxxxxxxxxxx";
```

UDP csomagok küldése

- **UDP** (User Datagram Protocol) kis méretű üzenetek továbbítása való
- A cél IP címén kívül egy port számot (16 bites szám) is meg kell adni (mint telefonnál a melléket), a fogadó alkalmazás várja az üzeneteket
- A következő mintaprogram a 4210-es porton várja az üzeneteket (max. 255 bájt), amelyet a forrás címével együtt kiír a soros porton.
- Válaszként egy "Hi from ESP8266!" üzenetet küldünk vissza a feladónak
- A program teszteléséhez a számítógépről küldünk üzeneteket a **Packet Sender** ingyenes alkalmazás segítségével, de annak sincs akadálya, hogy egy másik **ESP8266** kártyát használjunk helyette



ESP8266_UDP_recv.ino

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

#define UDP_PORT 4210
char packet[256];
char reply[] = "Hi from ESP8266!";

WiFiUDP UDP;

void setup() {
  Serial.begin(115200);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  Serial.print("Connecting to ");
  Serial.print(WIFI_SSID);
  while (WiFi.status() != WL_CONNECTED) {
    delay(100);
    Serial.print(".");
  }
  Serial.print("Connected! IP address: ");
  Serial.println(WiFi.localIP());
  UDP.begin(UDP_PORT);
  Serial.print("Listening on UDP port ");
  Serial.println(UDP_PORT);
}
```

```
void loop() {
  int packetSize = UDP.parsePacket();
  if (packetSize) {
    Serial.print("Received packet! Size: ");
    Serial.println(packetSize);
    int len = UDP.read(packet, 255);
    if (len > 0) {
      packet[len] = '\0';
    }
    Serial.print(UDP.remoteIP());
    Serial.write(':');
    Serial.println(UDP.remotePort());
    Serial.print("Packet received: ");
    Serial.println(packet);
    // Send return packet
    UDP.beginPacket(UDP.remoteIP(),
                   UDP.remotePort());
    UDP.write(reply);
    UDP.endPacket();
  }
}
```

```
Connecting to CSP-LINK.....
Connected! IP address: 192.168.1.101
Listening on UDP port 4210
Received packet! Size: 12
192.168.1.108:51097
Packet received: Hello world!
```

ESP8266_UDP_recv.ino

■ Futási eredmény a Packet Sender program használatakor

Packet Sender - IPs: 192.168.1.108, fe80::3c2a:6baf:6467:bba8%wireless_32768

File Tools Multicast Help

Name Hello

ASCII Hobbielektronika csoport

HEX 48 6f 62 62 69 65 6c 65 6b 74 72 6f 6e 69 6b 61 20 63 73 6f 70 6f 72 74

Address 192.168.1.255 Port 4210 Resend Delay 0 UDP Send Save

Search Saved Packets... Delete Saved Packet Persistent TCP

	Send	Name	Resend	To Address	To Port	Method	ASCII
10	Send	SMTP Gmail	0	smtp.gmail.com	465	SSL	HELO relay.example.com\r\nQUIT\r\n
11	Send	SSL cert mismatch	0	138.197.192.84	443	SSL	GET / HTTP/1.0\r\nHost: example.com\r\n\r\n
12	Send	TCP connection refused	0	138.197.192.84	200	TCP	will not connect\r\n
13	Send	TCP packetsender.com	0	packetsender.com	80	TCP	GET / HTTP/1.0\r\n\r\n
14	Send	TCP ssh ubuntu.com	0	ubuntu.com	22	TCP	SSH-2.0-OpenSSH_8.2p1 Ubuntu-4\r\n

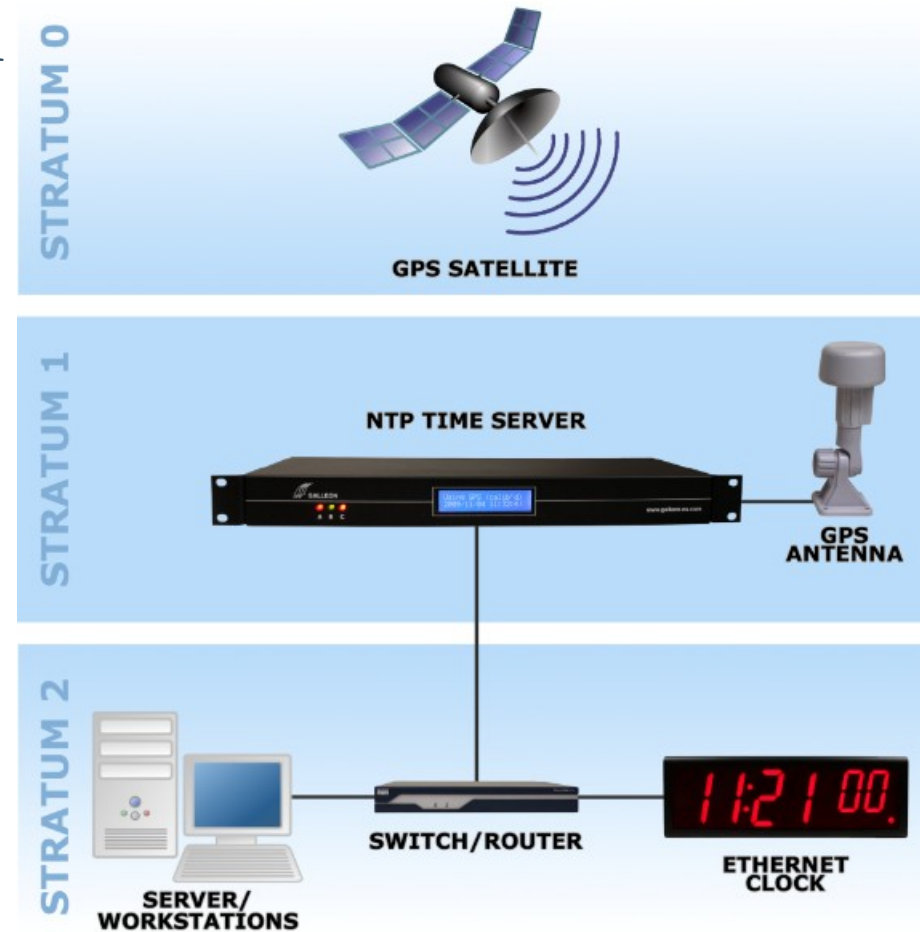
Clear Log (5) Log Traffic Save Log Save Traffic Packet Copy to Clipboard

Time	From IP	From Port	To Address	To Port	Method	Error	ASCII	Hex
17:52:21.510	192.168.1.101	4210	You	51097	UDP		Hi from ESP8266!	48 69 20 66 72 6f 6d 20 45 53 50 38 32 36 36 21
17:52:21.416	You	51097	192.168.1.255	4210	UDP		Hobbielektronika csoport	48 6f 62 62 69 65 6c 65 6b 74 72 6f 6e 69 6b 61 20 63 73 6f 70 6f 72 74
17:52:02.432	You	51097	192.168.1.255	4210	UDP		Hobbielektronika csoport	48 6f 62 62 69 65 6c 65 6b 74 72 6f 6e 69 6b 61 20 63 73 6f 70 6f 72 74
17:51:45.150	192.168.1.101	4210	You	51097	UDP		Hi from ESP8266!	48 69 20 66 72 6f 6d 20 45 53 50 38 32 36 36 21
17:51:45.040	You	51097	192.168.1.255	4210	UDP		Hello world!	48 65 6c 6c 6f 20 77 6f 72 6c 64 21

UDP:51097 TCP:54250 SSL:54251 IPv4 Mode

Pontos idő lekérdezése NTP kéréssel

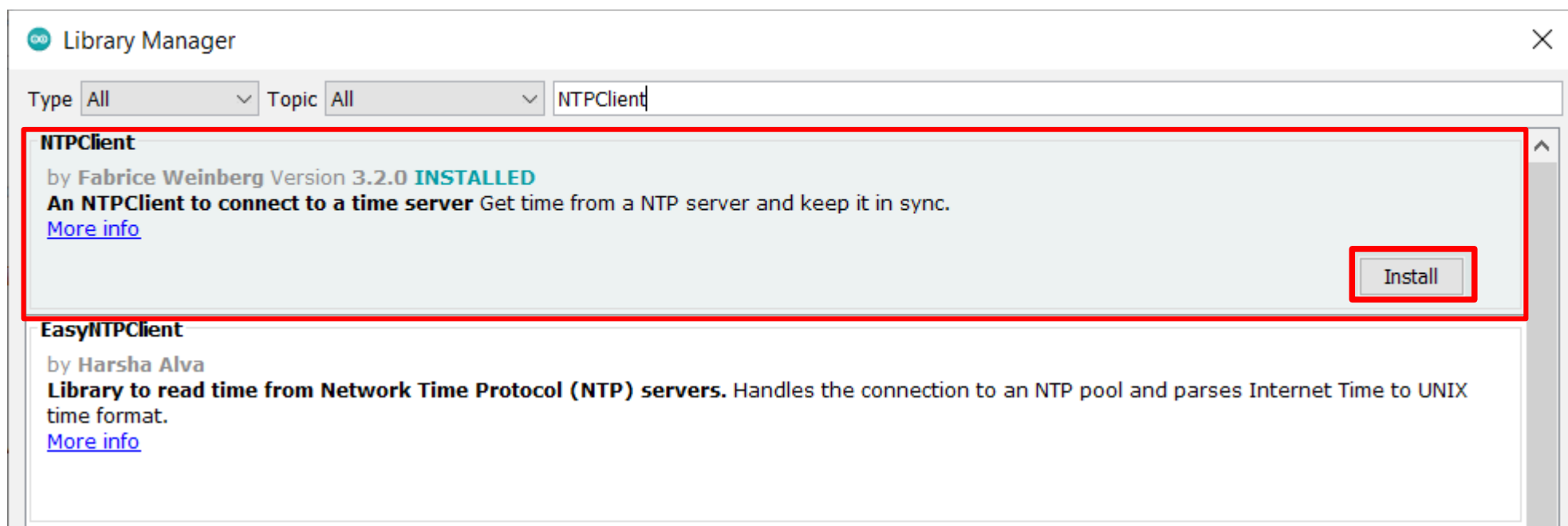
- A világot behálózó **NTP** (Network Time Protocol) szerverek is UDP csomagokkal kommunikálnak
- **NTP** szerver lehet helyi vagy távoli
- A nyilvános szervereket az **pool.ntp.org** fogja össze (lásd: <https://www.ntppool.org/en/>)
- Az **NTP** szerverek általában a 123-as portot használják
- Az üzenetcsomag formátumát (amely többnyire 48 bájt) és a protokollt az **RFC958** írja le
- A legegyszerűbb **NTP** kérelem: 0x1B és 47 db nulla



Kép forrása: www.galsys.co.uk/news/

Az NTPClient programkönyvtár

- Az előzőek alapján is megoldható az idő lekérdezése, de mégis kényelmesebb az **NTPClient** programkönyvtár használata
- Nyissuk meg a **Tools** menüben a **Manage Libraries** menüpontot és írjuk be keresőszóként, hogy: **NTPClient**
- A *Fabrice Weinberg* névvel fémjelzett **NTPClient** kell nekünk, kattintsunk az **Install** gombra!



Az NTPClient programkönyvtár

- **NTPClient**(*udp, poolServerName, timeOffset, updateInterval*) – a konstruktor fv. default paramétereit: *server= pool.ntp.org, timeOffset = 0, updateInterval = 60 s*
- **begin**(port) – a default port 1337, ami **Packet Senderrel** nem, de a programmal működik
- **update**() - frissíti az idő adatokat, ha *interval* már letelt, a közbenső időt pedig a **millis()** függvénnel mérjük
- **getEpochTime**() - POSIX típusú Epoch idő (1970-JAN-01 0:0:0 óta eltelt másodpercek száma , GMT időzóna szerint
- **getFormattedTime**() - string-ként adja meg az időt mint pl. `hh:mm:ss`

```
#include <NTPClient.h>
#include <ESP8266WiFi.h>
#include <WiFiUdp.h>
#include <secrets.h>

WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP);

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

void loop() {
  timeClient.update();
  Serial.println(timeClient.getFormattedTime());
  delay(1000);
}
```

ESP8266_ntp_client.ino

- Az **esp8266.com** fórum egyik hozzászólásának szerzője bemutatta, hogy az Epoch adatból hogyan állíthatjuk elő POSIX-szerű függvényhívásokkal a kívánt formátumú idő és dátum kiírásokat:
<https://www.esp8266.com/viewtopic.php?f=29&t=21327>
- A programban használt **strftime** függvény leírása itt található:
<https://www.cplusplus.com/reference/ctime/strftime/>
- A **localtime** függvény leírása pedig itt található:
<https://www.cplusplus.com/reference/ctime/localtime/>
- Fentiek ismeretében például ezt írhatjuk:

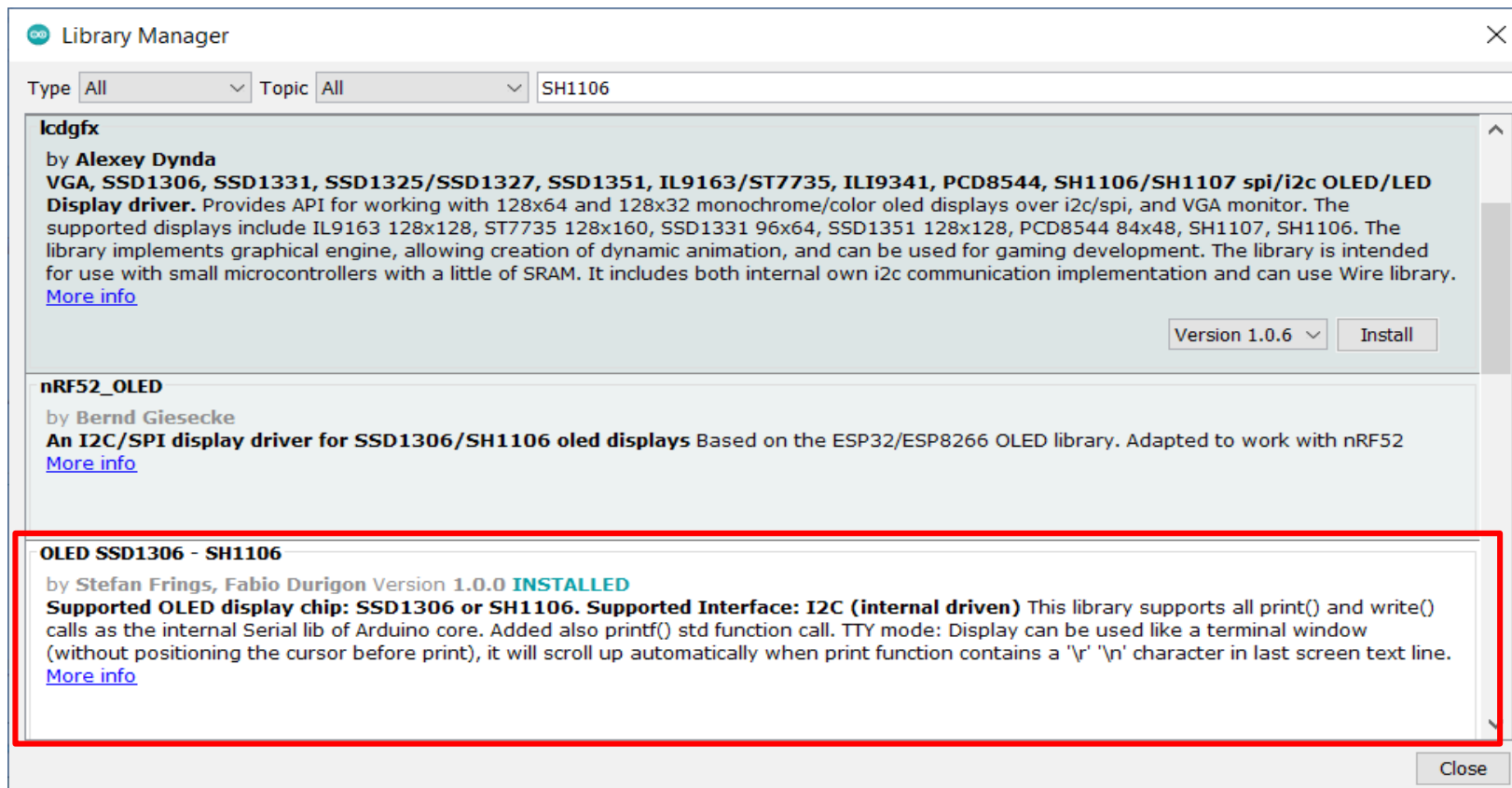
```
tm* timeinfo = localtime(&Epoch);  
strftime (Time, 10, "%T", timeinfo);  
strftime (Date, 12, "%d/%m/%Y", timeinfo);
```

ahol **Time** és **Date** karaktertömbök
- Ezekkel az ismeretekkel felvértezve készítsünk NTP órát egy OLED kijelző felhasználásával!

Az OLED programkönyvtár

- Az OLED SSD1306 – SH1106 könyvtárat választottam, de az `oled.cpp` állományban a `display()` metódust SH1106-hoz javítani kell:

```
if (isSH1106) {  
    i2c_send(0xB0 + page); // set page  
    i2c_send(0x02);        // lower columns address =2  
    i2c_send(0x10);        // upper columns address =0  
}
```



The screenshot shows the Arduino Library Manager interface. The search filter is set to 'SH1106'. Three libraries are listed:

- lcdgfx** by Alexey Dynda: Provides API for working with 128x64 and 128x32 monochrome/color oled displays over i2c/spi, and VGA monitor. The supported displays include IL9163 128x128, ST7735 128x160, SSD1331 96x64, SSD1351 128x128, PCD8544 84x48, SH1107, SH1106. The library implements graphical engine, allowing creation of dynamic animation, and can be used for gaming development. The library is intended for use with small microcontrollers with a little of SRAM. It includes both internal own i2c communication implementation and can use Wire library. [More info](#) Version 1.0.6 Install
- nRF52_OLED** by Bernd Giesecke: An I2C/SPI display driver for SSD1306/SH1106 oled displays Based on the ESP32/ESP8266 OLED library. Adapted to work with nRF52 [More info](#)
- OLED SSD1306 - SH1106** by Stefan Frings, Fabio Durigon Version 1.0.0 **INSTALLED** Supported OLED display chip: SSD1306 or SH1106. Supported Interface: I2C (internal driven) This library supports all print() and write() calls as the internal Serial lib of Arduino core. Added also printf() std function call. TTY mode: Display can be used like a terminal window (without positioning the cursor before print), it will scroll up automatically when print function contains a '\r' '\n' character in last screen text line. [More info](#)

A kijelző bekötése és inicializálása

- A kapcsolást egyszerűen úgy állítottuk össze, hogy egy I2C SSD1306 OLED kijelzőt „mellétűztünk” a NodeMCU kártyának
- **SDA** így GPIO0 lesz, **SCLK** pedig GPIO2
- A kivezetéseket, az I2C címet és az egyéb paramétereket (grafikai felbontás, van-e RESET kivezetés, SH1106 vagy SSD1306 vezérlő) a konstruktornak kell megadni, például:

```
OLED display=OLED(0,2,NO_RESET_PIN,0x3C,128,32,false);
```

SDA pin SCL pin RESET pin I2C address resolution in pixels isSH1106?

A fenti példa egy SSD1306 128 x 32 pixel felbontású kijelzőt definiál

SDA
SCL
VCC
GND



ESP8266_ntp_clock.ino

- A programban mellőztük a soros portra történő írást, mivel használat közben nem csatlakoztatjuk számítógéphez az órát

```
#include <NTPClient.h>    // NTP Client by Fabrice Weinberg
#include <ESP8266WiFi.h>  // default from Espressif
#include <WiFiUdp.h>      // default from Espressif
#include "secrets.h"      // WiFi login secrets
#include <oled.h>

#define TZ                1    // (utc+) Timezone in hours
#define DST_MN           60    // Daylight Saving Time shift in minutes
#define GMT_OFFSET_SEC  3600 * TZ
#define DAIYLIGHT_OFFSET_SEC 60 * DST_MN
#define NTP_SERVER "pool.ntp.org" // Time server cluster

// Variables
tm*      timeinfo;    // localtime returns a pointer to a tm structure
time_t   Epoch;
static char msg[20];  // character buffer

// Constructors
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, NTP_SERVER, GMT_OFFSET_SEC); // Configure NTP client
OLED display=OLED(0,2,NO_RESET_PIN,0x3C,128,32,false);    // Configure display
```

ESP8266_ntp_clock.ino

```
void setup() {
  display.begin(); // Initialise OLED display

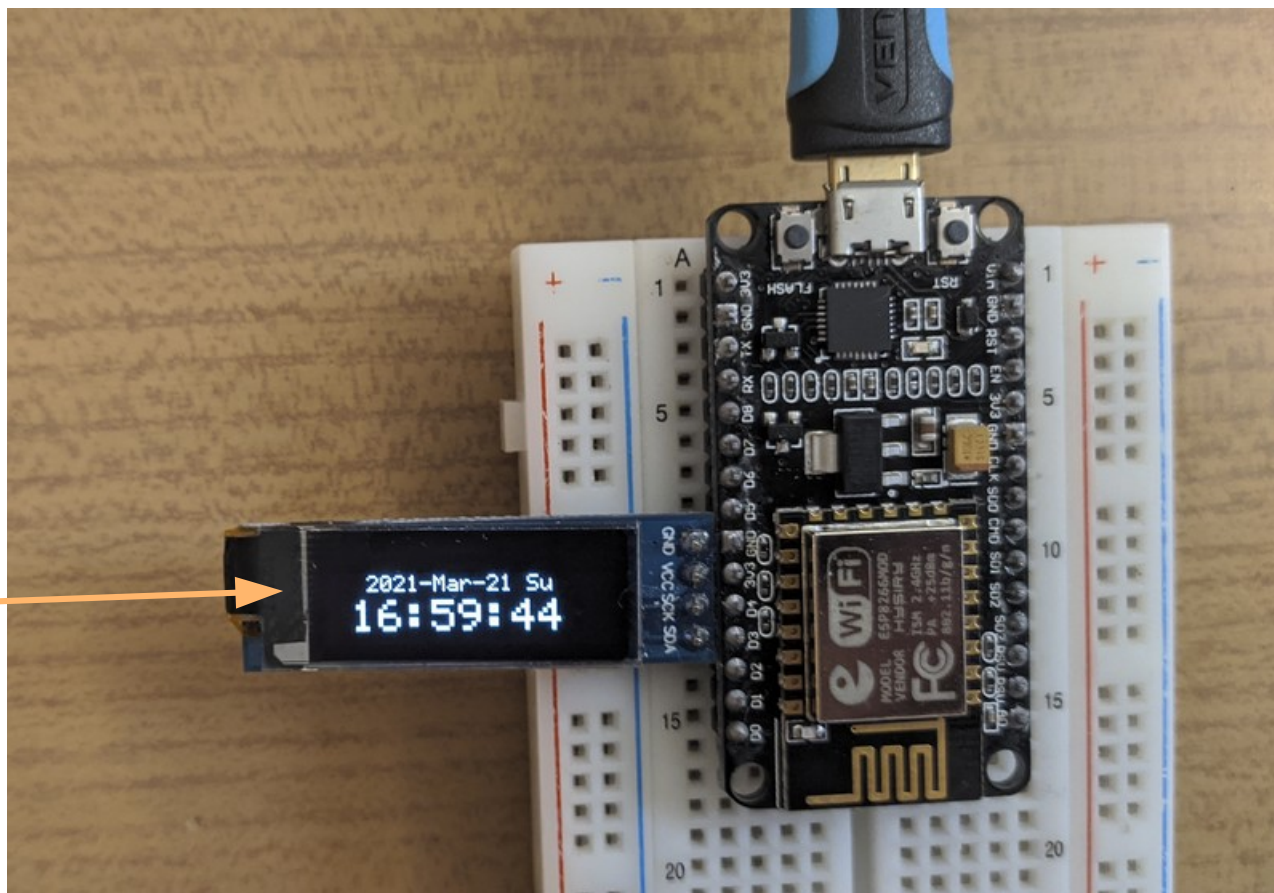
  WiFi.begin(WIFI_SSID, WIFI_PASS); // Connect to WiFi network
  while ( WiFi.status() != WL_CONNECTED ) {
    delay ( 500 );
  }

  timeClient.begin(); // Start NTP client
  timeClient.update();
}

void loop() {
  timeClient.update();
  Epoch = timeClient.getEpochTime(); // Get Epoch time
  timeinfo = localtime(&Epoch); // Convert to local time
  display.clear();
  strftime (msg, 15, "%Y-%b-%d %a ", timeinfo);
  display.draw_string(16,1,msg); // Display date
  strftime (msg, 10, "%T ", timeinfo); // Display time
  display.draw_string(8,12,msg,OLED::DOUBLE_SIZE);
  display.display(); // Refresh screen
  delay(1000);
}
```


ESP8266_ntp_clock.ino

- A program futási eredménye az alábbi képen látható
- A kép állása az **oled.cpp** állományban az **A0/C0** vagy **A1/C8** inicializáló parancsokkal választható ki. Bővebben információ az 2021. február 18-i előadásban található (STM32 tanfolyam)

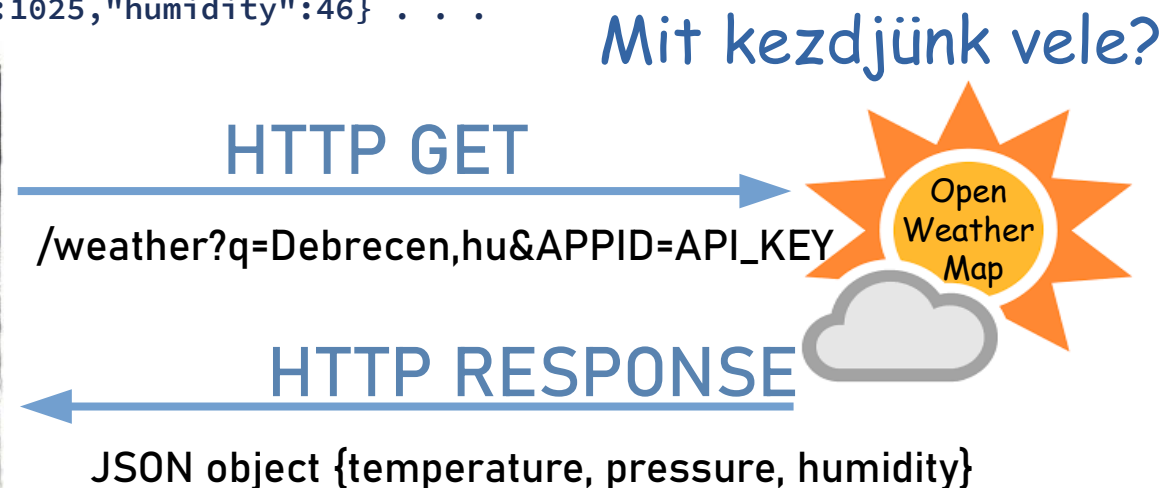
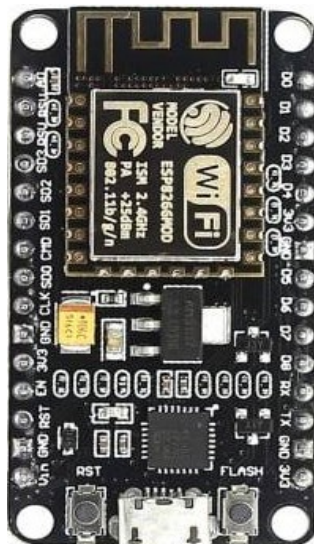


Ez az **A0/C0**
parancsok által
kiválasztott
állás

Időjárási adatok lekérése (OpenWeatherMap.org)

- Az időjárási adatokat HTTP kliensként tölthetjük le az openweathermap.org webszerverről
- 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}} . . .
```



ESP8266_webclient.ino: HTTP GET request

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.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) {
    HTTPClient http;
    http.begin(server_url);
    int httpCode = http.GET();
    if (httpCode > 0) {
      String payload = http.getString();
      Serial.println(payload);
    }
    http.end();
  }
  delay(10000);
}
```

HTTP GET lekérés:

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

```
// Check WiFi connection status
// Declare an object of class HTTPClient
// Specify request destination
// Send the request
// Check the returning code
// Get the request response payload
// Print the response payload

// Close connection

// Send a request every 10 seconds
```


A JSON adatsere-formátum

- **JSON** (JavaScript Object Notation) szöveg alapú, adatszeréhez 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 kollekcója, 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 található
- A **Deserialization tutorial** lapon egy teljes könyvfejezet letölthető
- A **JsonFilterExample** mintapélda a bejövő adatok szűrését mutatja be

Időjárási adatok kezelése

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 tempC = doc["main"]["temp"];
int hPa = doc["main"]["pressure"];
int relHum = doc["main"]["humidity"];
```

ESP8266_OpenWeatherMap.ino 2/1.

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include "secrets.h"
#include <ArduinoJson.h>

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

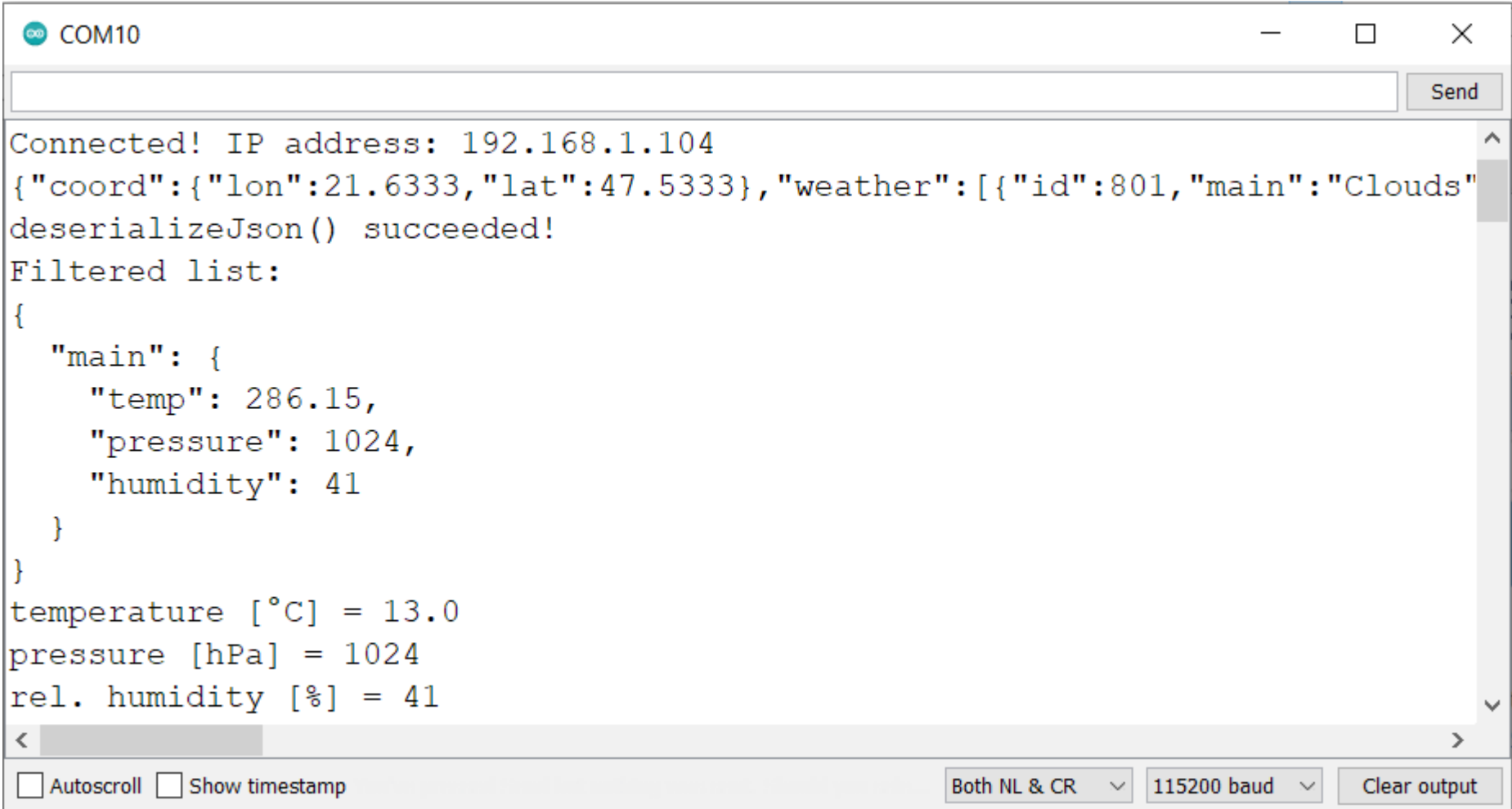
void setup () {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  Serial.print("Connecting to ");
  Serial.print(WIFI_SSID);
  while (WiFi.status() != WL_CONNECTED) {
    Delay(500);
    Serial.print(".");
  }
  Serial.print("\r\nConnected! IP address: ");
  Serial.println(WiFi.localIP());
  // Prepare filter
  filter["main"]["temp"] = true;
  filter["main"]["pressure"] = true;
  filter["main"]["humidity"] = true;
}
```

ESP8266_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 tempC = 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(tempC - 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
}
```


ESP8266_OpenWeatherMap.ino

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

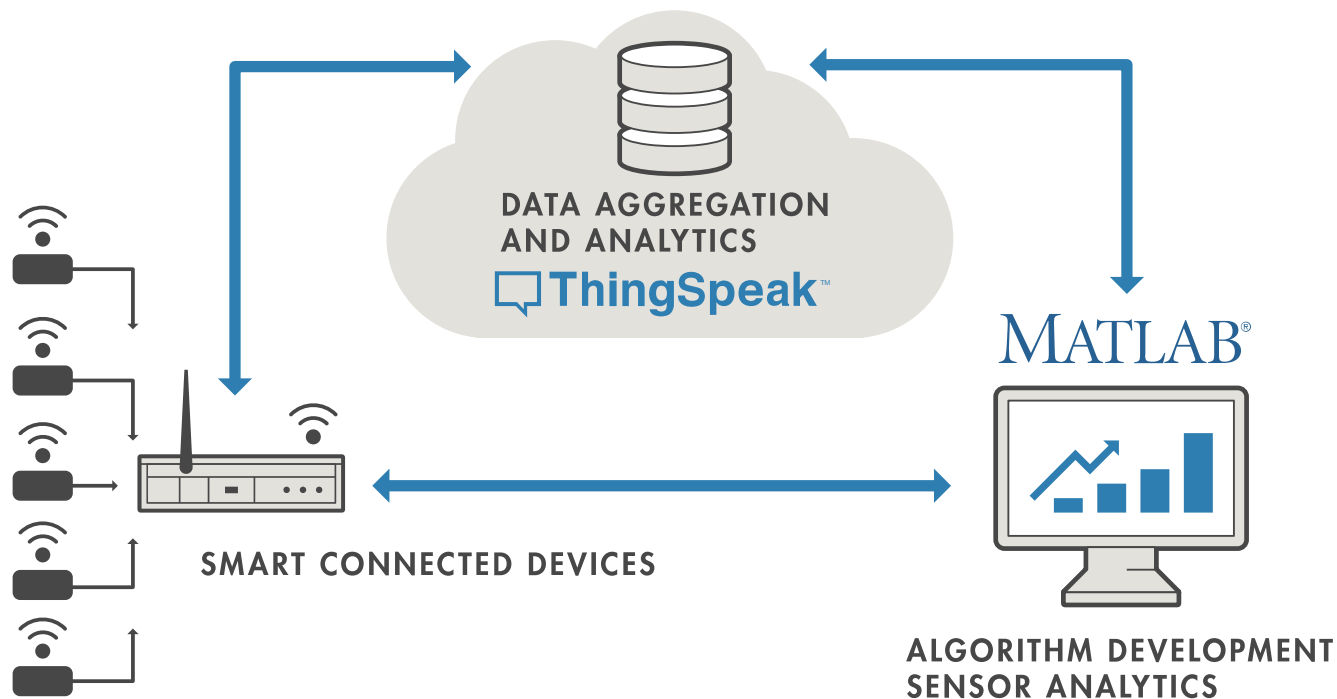


```
COM10
Connected! IP address: 192.168.1.104
{"coord":{"lon":21.6333,"lat":47.5333},"weather":[{"id":801,"main":"Clouds"}]}
deserializeJson() succeeded!
Filtered list:
{
  "main": {
    "temp": 286.15,
    "pressure": 1024,
    "humidity": 41
  }
}
temperature [°C] = 13.0
pressure [hPa] = 1024
rel. humidity [%] = 41
```

Autoscroll Show timestamp Both NL & CR 115200 baud Clear output

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 számológéppel 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();
```

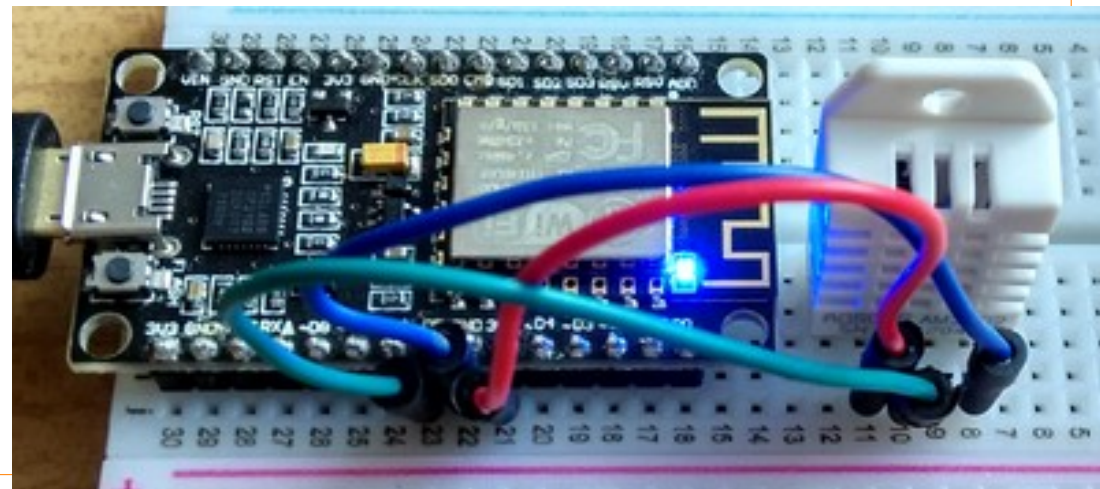
„hasból” beírt adatok

ESP8266_ThingSpeak.ino 2/1.

- Egy DHT22 szenzor adatait küldjük a ThingSpeak szerverre

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include "DHT.h"
#include "secrets.h"
#define DHTPIN 14 // GPIO14 (D5) pin for DHT sensor
DHT dht(DHTPIN, DHT22);
String server_url = "http://api.thingspeak.com/update?api_key=" + THINGSPEAK_WRITE_APIKEY;
float t = 0.0; // temperature
float h = 0.0; // humidity

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());
}
```

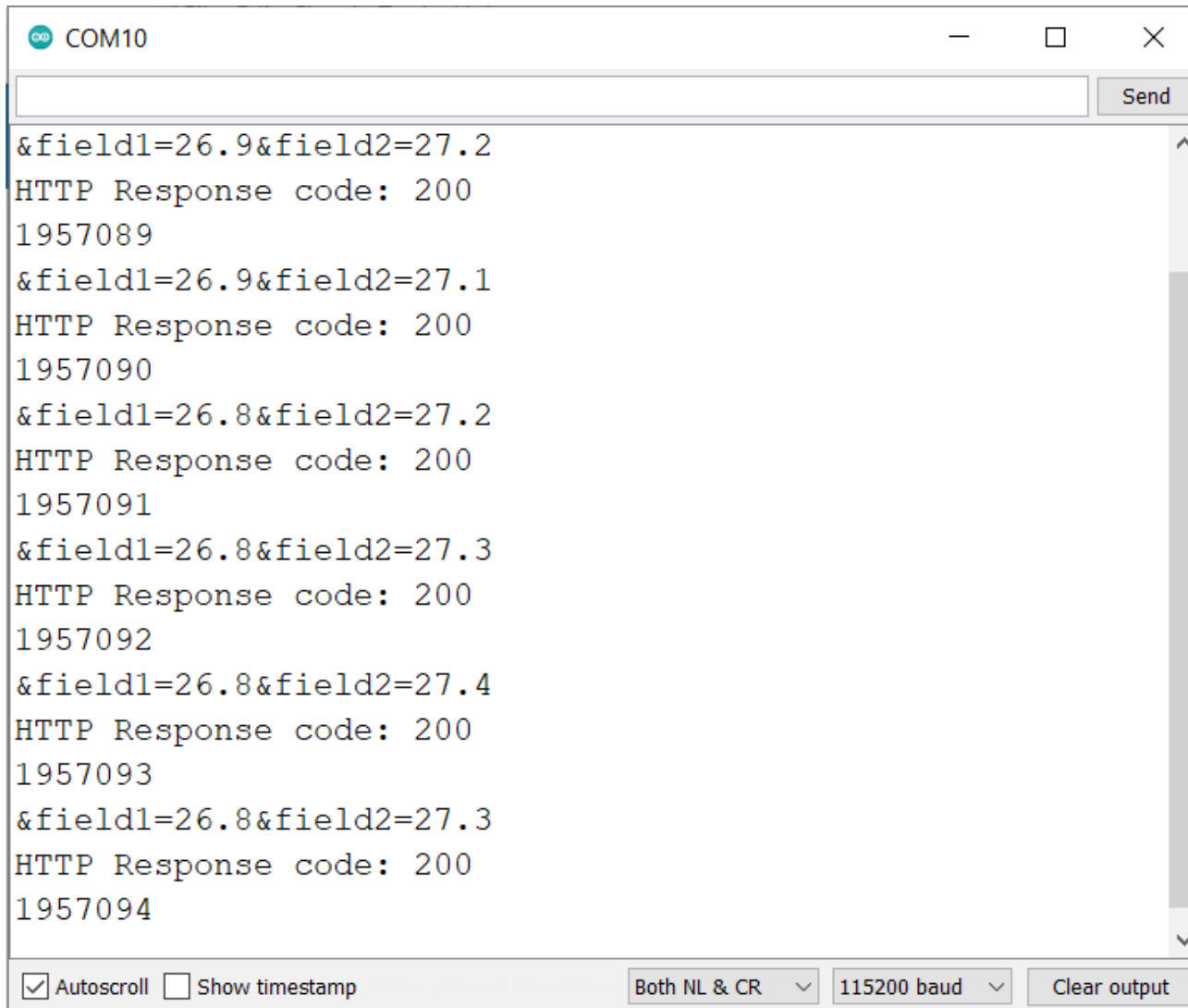


ESP8266_ThingSpeak.ino 2/2.

```
void loop() {
  float tnew = dht.readTemperature();
  if (!isnan(tnew)) t = tnew;           // Check if data is valid
  float hnew = dht.readHumidity();
  if (!isnan(hnew)) h = hnew;          // Check if data is valid
  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
}
```

ESP8266_ThingSpeak.ino

- Szerencsés esetben az alábbihoz hasonló kiírásokat látunk a terminálablakban



The screenshot shows a terminal window titled 'COM10'. The window contains a text input field with a 'Send' button. Below the input field, the terminal displays a series of HTTP response codes and timestamps. The output is as follows:

```
&field1=26.9&field2=27.2
HTTP Response code: 200
1957089
&field1=26.9&field2=27.1
HTTP Response code: 200
1957090
&field1=26.8&field2=27.2
HTTP Response code: 200
1957091
&field1=26.8&field2=27.3
HTTP Response code: 200
1957092
&field1=26.8&field2=27.4
HTTP Response code: 200
1957093
&field1=26.8&field2=27.3
HTTP Response code: 200
1957094
```

At the bottom of the terminal window, there are several controls: a checked 'Autoscroll' checkbox, an unchecked 'Show timestamp' checkbox, a dropdown menu set to 'Both NL & CR', a dropdown menu set to '115200 baud', and a 'Clear output' button.

Az eredmény megtekintése böngészőben

The screenshot shows a web browser window with the address bar containing <https://thingspeak.com/channels/34244>. The page header features the ThingSpeak logo and navigation links for Channels, Apps, Community, and Support. The channel name 'Pista szoba' is prominently displayed.

Pista szoba

Channel ID: **34244**

Author: [icserny](#)

Access: Public

Experimental channel running in the room of Pista.

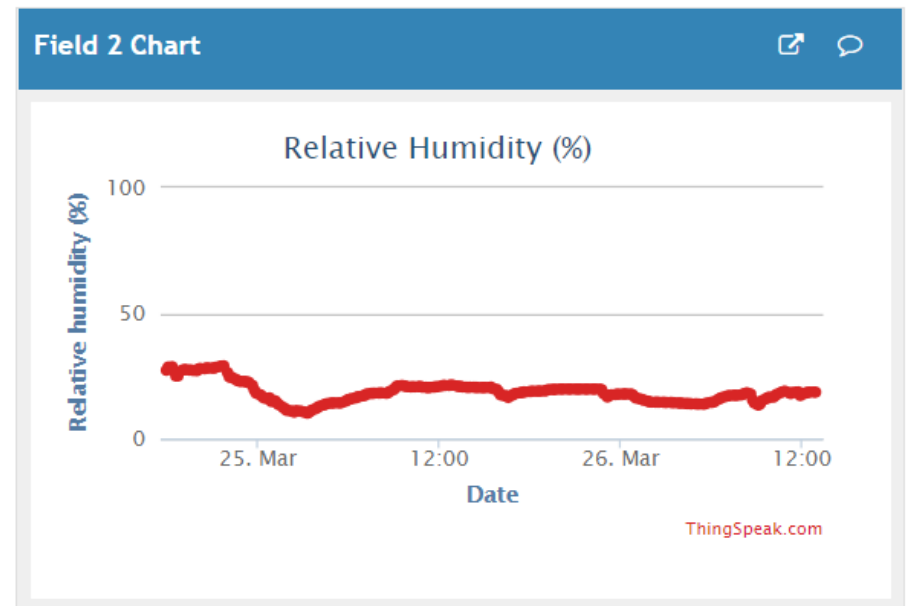
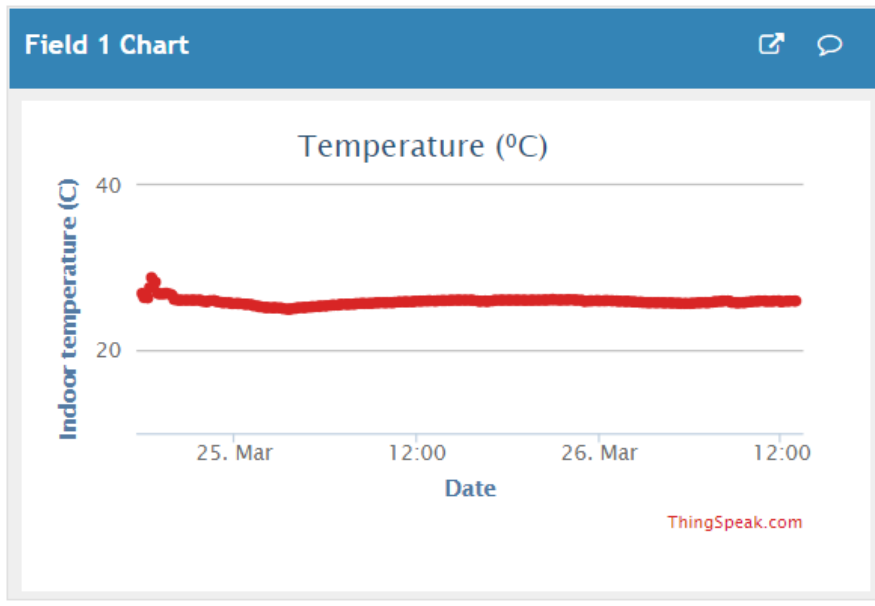
Hardware: DHT22 sensor + NodeMCU with ESP-12E

ESP8266 WiFi module. Software: NodeMCU Lua

float_0.9.6-dev_20150704 firmware + Simple

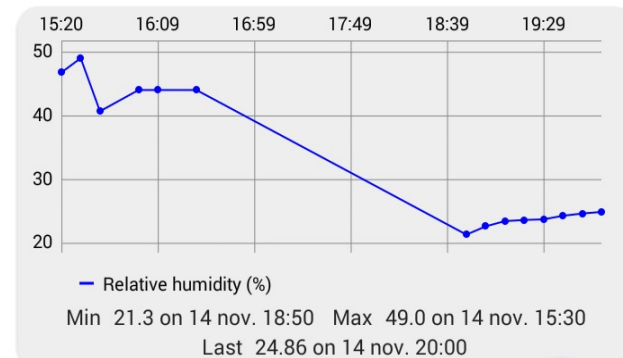
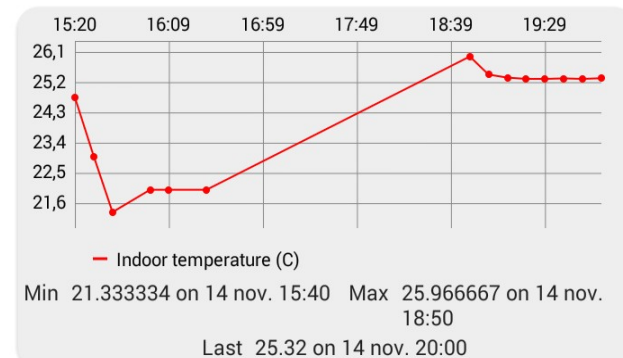
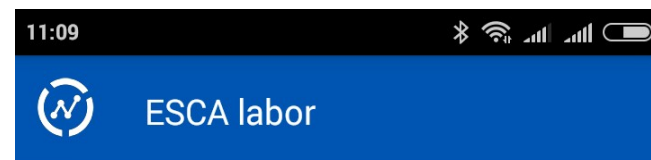
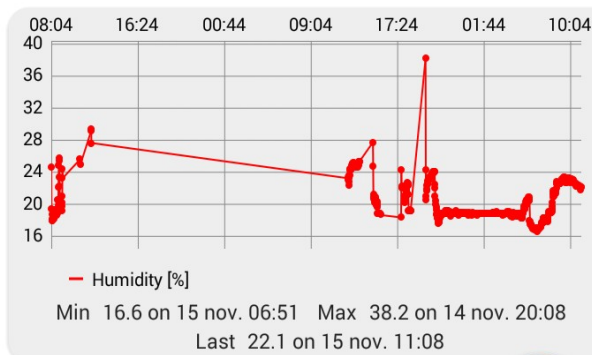
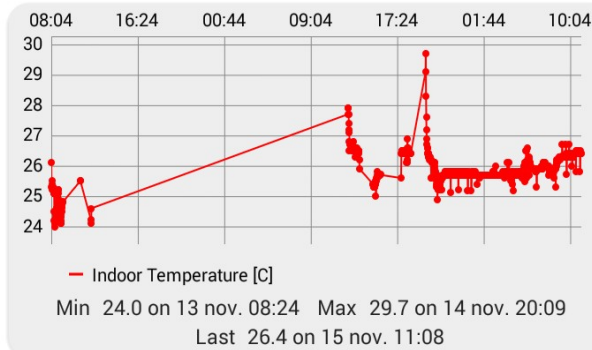
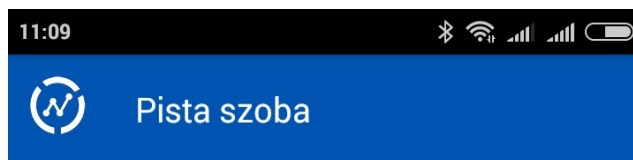
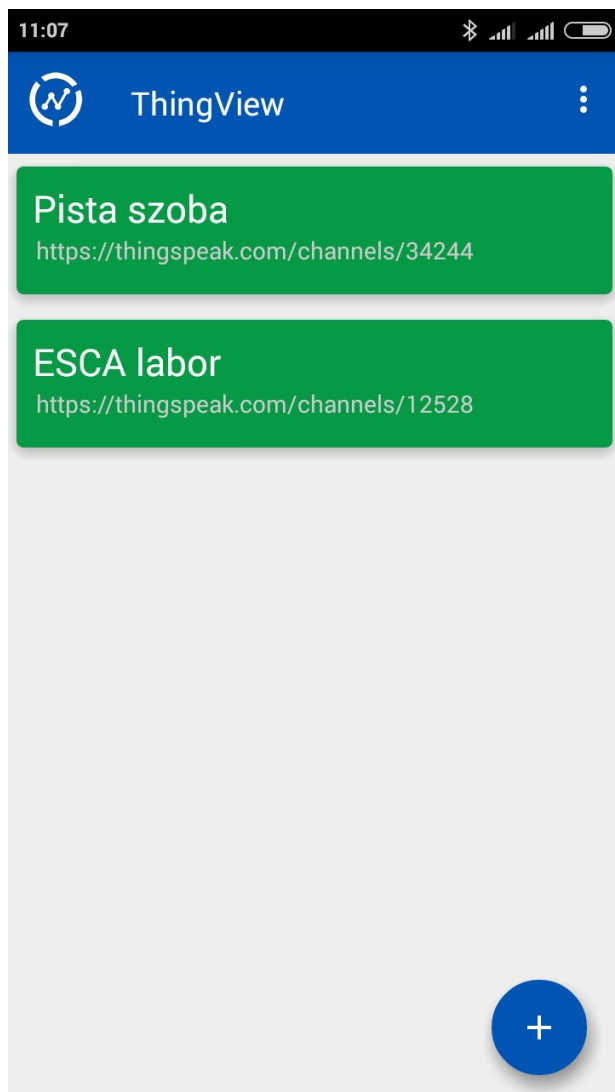
Thingspeak Client published by Jankop at

esp8266.com



Az eredmény megtekintése applikációban

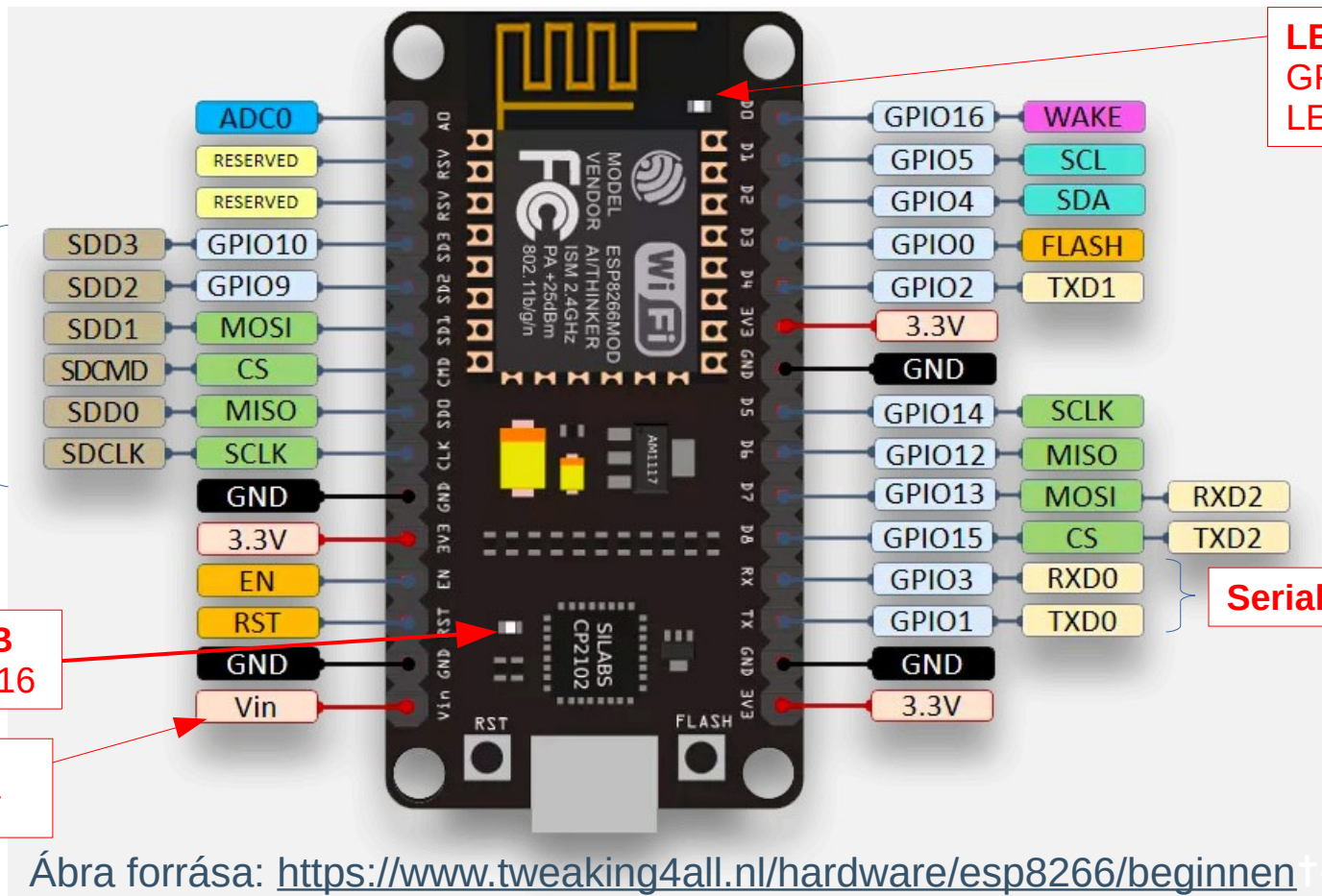
- ThingView - ThingSpeak megjelenítő (Google Play áruház)



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!



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

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