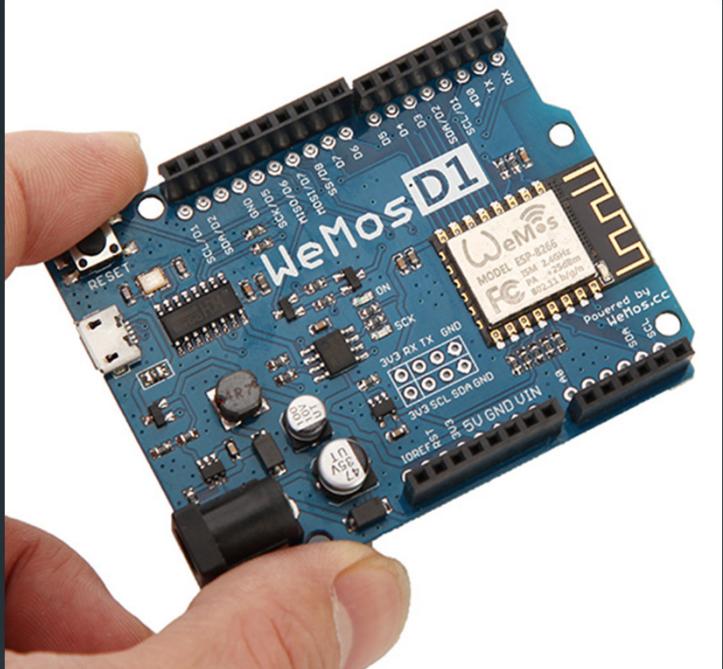


Placas para Desenvolvimento



IoT com

WeMos D1 = *Arduino* + *WiFi*

Prof. Kaw

Out/19



Outline

- Definição
- Características
- Aplicação 1: Monitor de Temperatura via Web
- Aplicação 2: Monitor de Temperatura & Umidade via *Cloud**

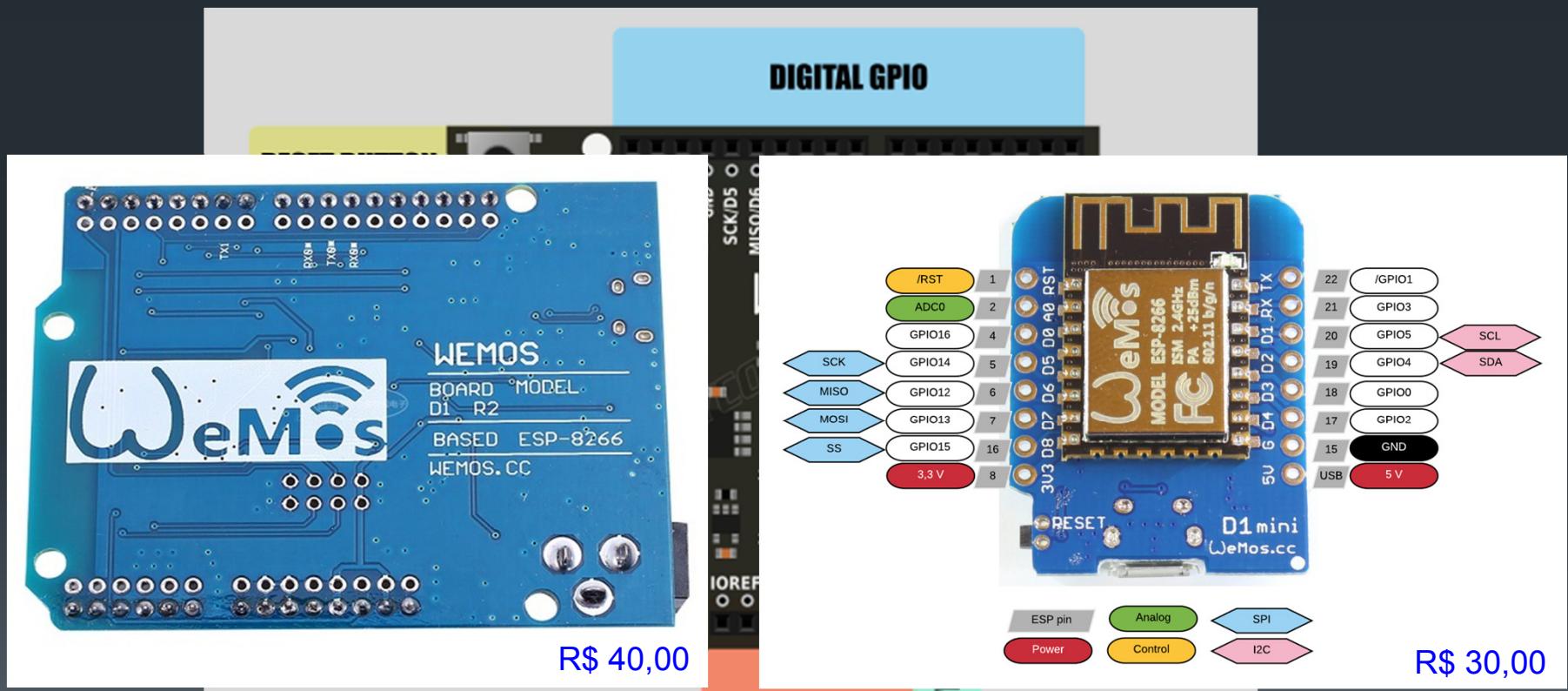
* ThingSpeak

Definição

Out/2019
Prof. Kaw

3

- **WeMos D1** - placa de desenvolvimento, com WiFi (ESP8266) que usa o layout do Arduino, com tensão operacional de 3,3 V



Características

Out/2019

Prof. Kaw

4

- WeMos D1 R2
 - Controlada pelo módulo **ESP8266EX**
 - Microcontrolador: Tensilica L106, 32 bits, RISC, encapsulamento 32 pinos QFN
 - RAM 160 kB, mas livre < 50 kB (depende do modo de funcionamento: STA/AP/STA+AP)
 - RTOS em ROM
 - *Clock*: 80 a 160 MHz
 - Entrada/Saída digitais: 11 pinos (2,5V ~ 3,6V)
 - Todos com funções de **interrupção**, **PWM**, **I2C**, **one-wire** (exceto D0)
 - Uma entrada analógica, A0 (entrada máxima de 3,6 V)
 - Uma conexão USB (micro): Serial *Bootloader*
 - Um conector de alimentação (*jack*), 7 a 12 V
 - Layout compatível com Arduino UNO R3 (somente *shields* 3,3V do Arduino)
 - Programável via IDE do Arduino (porta USB: C.I. CH340G)
 - Compatível com NodeMCU e MicroPython
 - Flash (memória de código): 4 MB (endereça até 16 MB)
 - **Comunicação WiFi embutida**
 - IEEE 802.11 b/g/n/e/i, 2,4 GHz
 - Protocolos de Rede: IPv4, TCP, UDP, HTTP, FTP (até 5 conexões simultâneas)
 - Segurança; Criptografia: WPA/WPA2; WEP/TKIP/AES
 - Programável via WiFi (**OTA - Over The Air**): WiFi *Bootloader*
 - UART, SDIO*, SPI**, I2C, I2S***, IR *Remote Control*

* *Secure Digital Input Output* - padrão para cartão de memória

** *Serial Peripheral Interface* (4 sinal) - usado para comunicação em sistemas embarcados, curtas distâncias (1986, Motorola)

*** *Inter-IC Sound* - usado para conectar dispositivos de áudio digital (1986, Philips → NXP Semiconductors)

Características

Out/2019

5

Prof. Kaw

DC Power Jack
7-12VDC Input



USB Micro-B Port
To Computer

WeMos D1 R2 WiFi Board Pinouts:

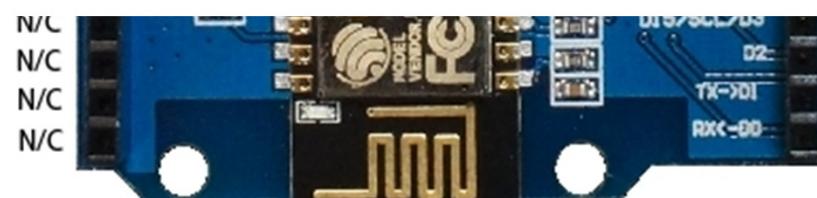
R1 Pin	Pin	Function	ESP-8266 Pin
D0	TX	TXD	TXD
D1	RX	RXD	RXD
D2	D0	IO	GPIO16
D3 (D15)	D1	IO, SCL	GPIO5
D4 (D14)	D2	IO, SDA	GPIO4
D5 (D13)	D3	IO, 10k Pull-up	GPIO0
D6 (D12)	D4	IO, 10k Pull-up, BUILTIN_LED	GPIO2
D7 (D11)	D5	IO, SCK	GPIO14
D8	D6	IO, MISO	GPIO12
D9	D7	IO, MOSI	GPIO13
D10	D8	IO, 10k Pull-down, SS	GPIO15
A0	Analog input, max 3.3V input		A0
G	Ground		GND
5V	5V		-
3V3	3.3V		3.3V
RST	Reset		RST

All of the IO pins have interrupt/pwm/I2C/one-wire support except D0

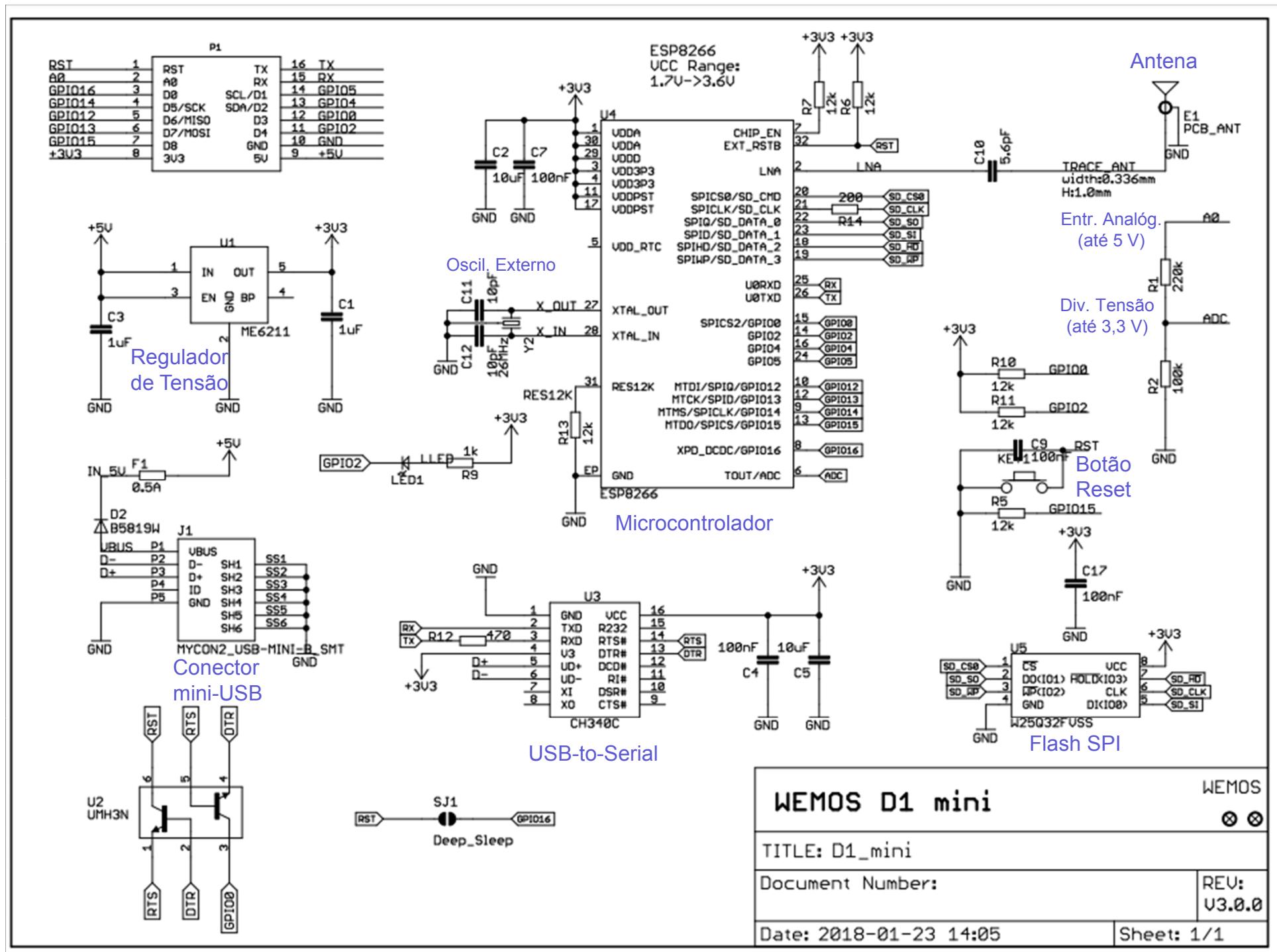
WeMos D1 "R1" WiFi Board Pinouts:

R1 Pin	Function	ESP-8266 Pin
D0	RX	GPIO3
D1	TX	GPIO1
D2	IO	GPIO16
D3 (D15)	IO, SCL	GPIO5
D4 (D14)	IO, SDA	GPIO4
D5 (D13)	IO, SCK	GPIO14
D6 (D12)	IO, MISO	GPIO12
D7 (D11)	IO, MOSI	GPIO13
D8	IO, Pull-up	GPIO0
D9	IO, Pull-up, BUILTIN_LED	GPIO2
D10	IO, Pull-down, SS	GPIO15
A0	Analog Input	A0
G	Ground	GND
5V	5V	-
3V3	3.3V	3.3V
RST	Reset	RST

*All IO have interrupt/pwm/I2C/one-wire supported(except D2)



- (D3) Digital Pin 3 / PWM / SCL
- (D2) Digital Pin 2 / PWM
- (D1) Serial Port TXD / Digital Pin 1 / PWM
- (D0) Serial Port RXD / Digital Pin 0





Monitor de Temperatura via Web

Aplicação 1

Aplicação 1

Out/2019

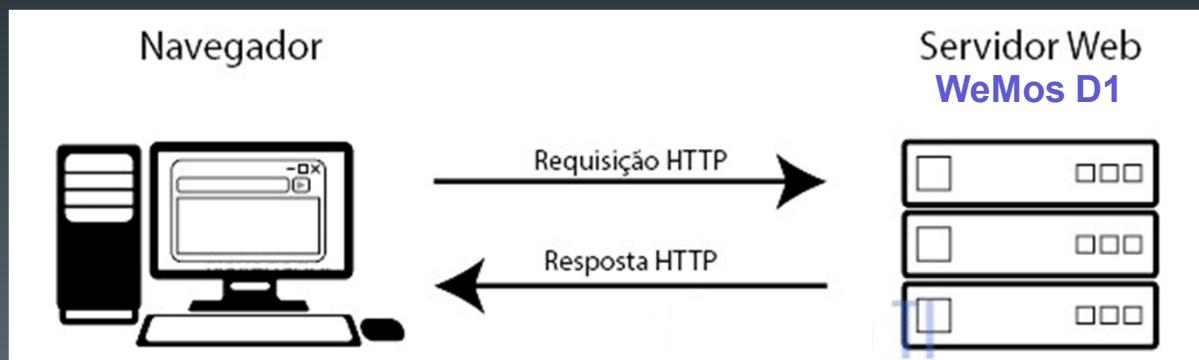
Prof. Kaw

9

- Monitor de Temperatura via Web

HTTP (*Hypertext Transfer Protocol*) - protocolo de **requisição (get)** e **resposta (put)** entre operadores: cliente e servidor

- WeMos D1 atua como **servidor Web**, aguarda requisições e responde com o conteúdo da página
- Navegador Web atua como **cliente**, solicita informações ao site (servidor)

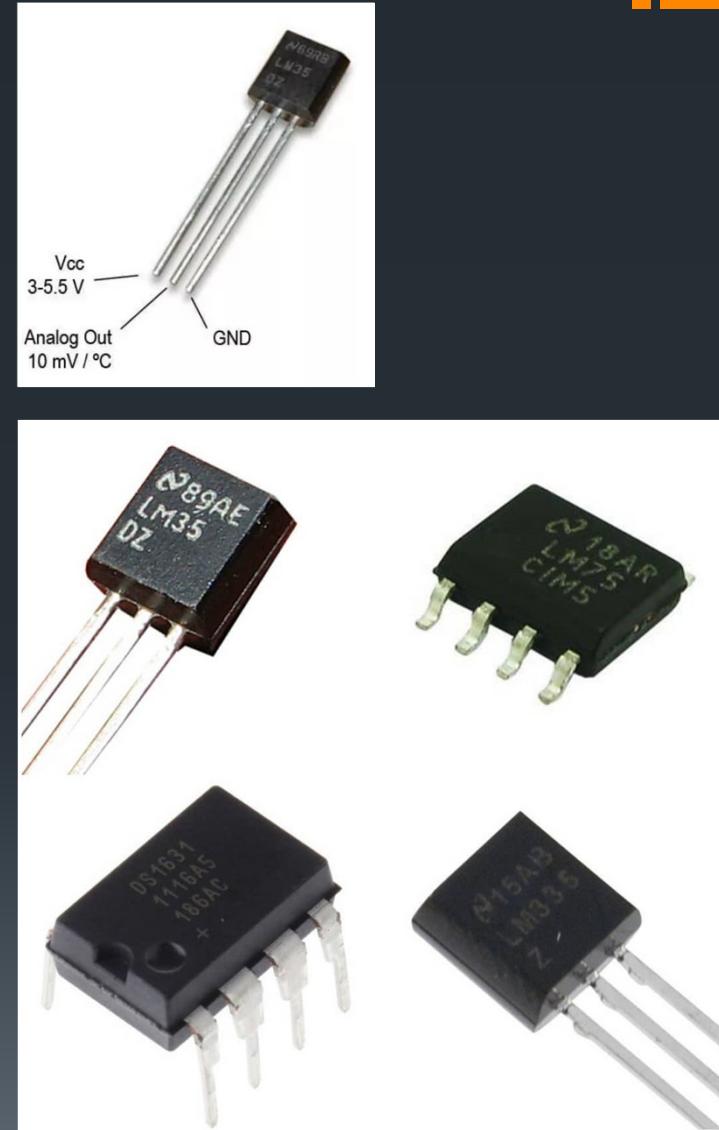
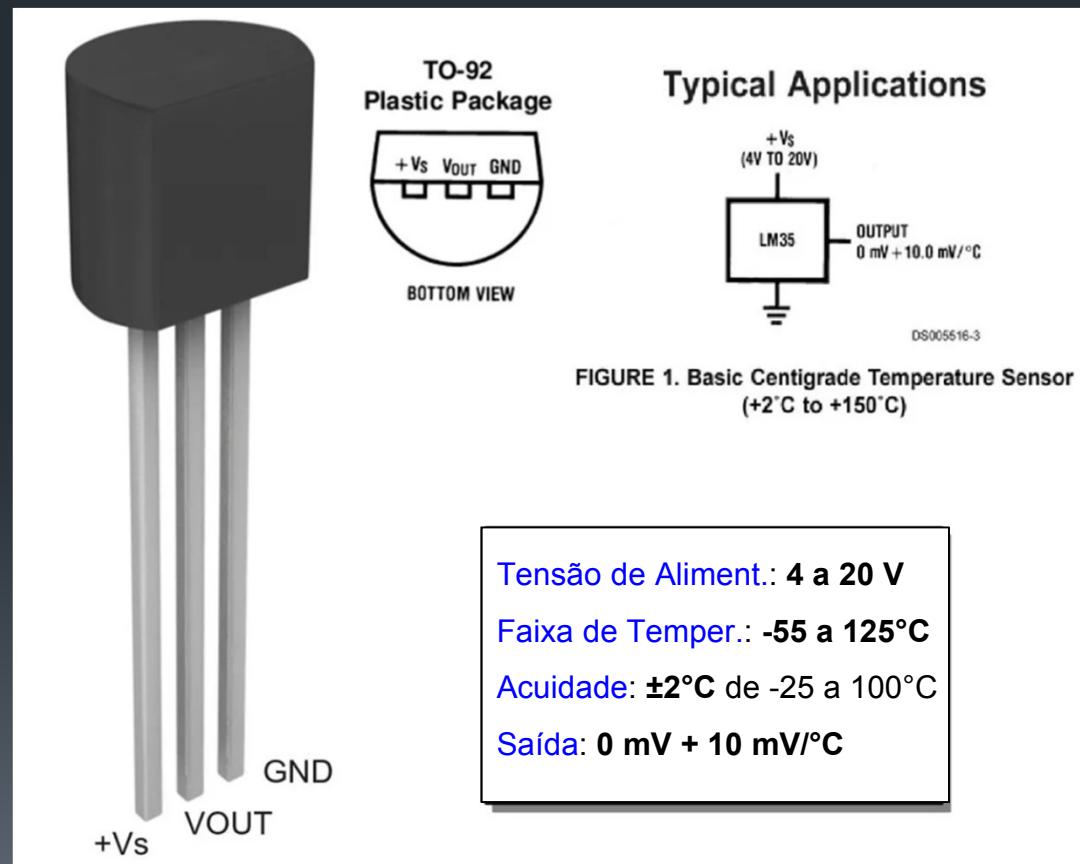


Aplicação 1

Out/2019
Prof. Kaw

10

- Sensor de Temperatura
 - LM335, **LM35**, LM75, DS1631



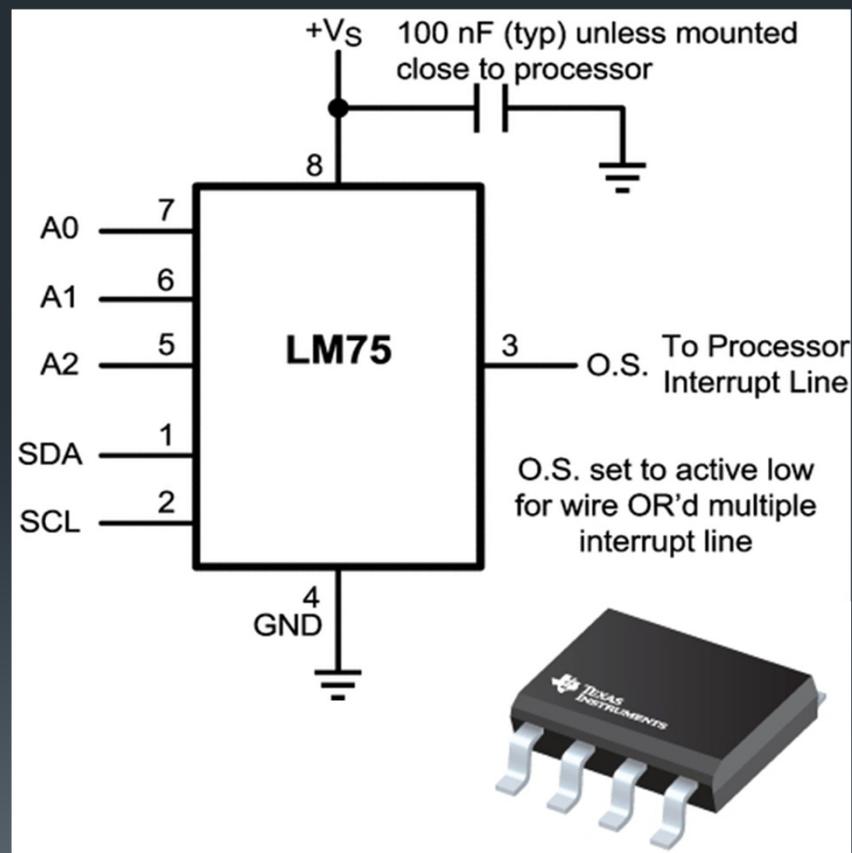
Aplicação 1

Out/2019

11

Prof. Kaw

- Sensor de Temperatura
 - LM75



LM75A – Sensor de temperatura digital, com **ADC** sigma-delta e **interface I2C**

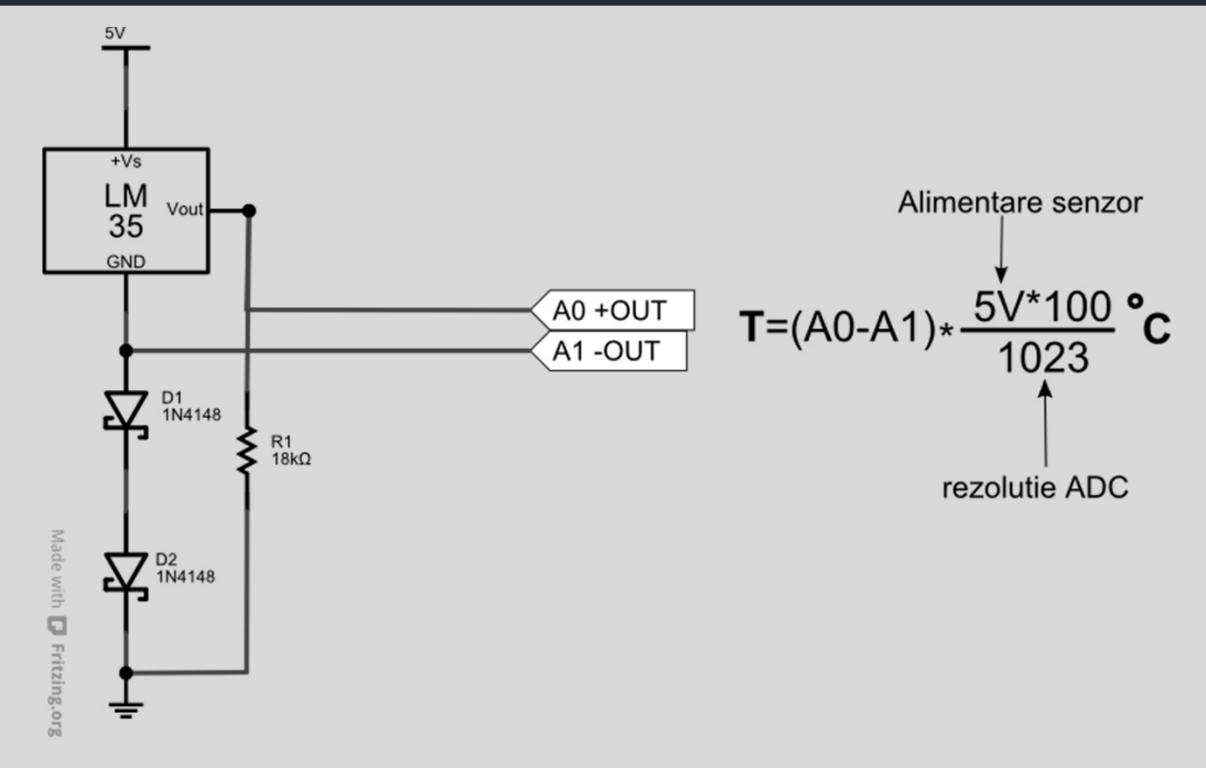
- Leituras digitais de temperatura de **9 bits** com precisão de $\pm 2^\circ\text{C}$ na faixa de -25°C a 100°C e $\pm 3^\circ\text{C}$ de -55°C a 125°C
- Alimentação única de 2,7 V a 5,5 V
- A comunicação via I2C a 400 kHz, com 3 pinos de endereço → até 8 LM75A operando no mesmo barramento
- Possui uma saída de superaquecimento dedicada (O.S.), com limite programável e histerese
- Corrente em modo normal: 280 μA (típ.), em modo "shutdown": 4 μA (típ.)

Aplicação 1

Out/2019
Prof. Kaw

12

- Sensor de temperaturas negativas também



Usando diodos típicos
(tensão direta de 0,7 V)
para levantar o pino GND
do LM35 cerca de 1,4 V
acima do GND do sistema.

Esse deslocamento
combinado com o resistor
de 18 kΩ conectado ao
terra real do sistema
permite que o dispositivo
conduza a tensões abaixo
de 0 V.

Aplicação 1

Out/2019

Prof. Kaw

13



- Programando a WeMos pela IDE Arduino
 - Acesse Arquivo → Preferências → URLs Adicionais para Gerenciadores de Placas e cole o seguinte link:
http://arduino.esp8266.com/stable/package_esp8266com_index.json
 - Vá para Ferramentas → Placas → Gerenciador de Placas... e selecione **ESP8266 by ESP8266 Community** e clique em Install e em seguida feche a janela
 - Vá para Ferramentas → Placas e selecione **WeMos D1 R2 & mini**
 - Selecione a velocidade **115200** em Ferramentas → Upload Speed
 - Instale no seu sistema operacional um driver para o chip USB-to-Serial:
<http://github.com/nodemcu/nodemcu-devkit/tree/master/Drivers>
 - Digite o código seguinte no IDE Arduino...

Aplicação 1

Out/2019

Prof. Kaw

14

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>

const char* ssid = "minhawIFI";
const char* password = "12345force";
ESP8266WebServer server(80);

void handleRoot() {
    String message = "<html><body>\n";
    message += "<h1>Servidor WEMOS</h1>";
    message += "<p>A0=";
    message += analogRead(A0);
    message += "</p>";
    message += "<p>millis =";
    message += millis();
    message += "</p>";
    message += "<meta http-equiv= 'refresh' content='3'>";
    message += "</body>";
    message += "</html>\n";
    server.send(200, "text/html", message);
}
```



Atualize com as informações da sua rede...

Aplicação 1

Out/2019

Prof. Kaw

15

```
void connectToWiFi() {  
    Serial.print("\n\nconectando a");  
    Serial.println("ssid");  
    WiFi.begin(ssid, password);  
    while (WiFi.status() != WL_CONNECTED) {  
        delay(500);  
        Serial.print(".");  
    }  
    Serial.println("\nWiFi conectado!");  
    Serial.print("Ender. IP:");  
    Serial.println(WiFi.localIP());  
}  
  
void setup() {  
    Serial.begin(115200);  
    connectToWiFi();  
    server.on("/", handleRoot);      // função executada p/ mostrar a página  
    server.begin();  
    Serial.println("Servidor HTTP iniciado!");  
}  
  
void loop() {  
    server.handleClient();  
}
```

Aplicação 1

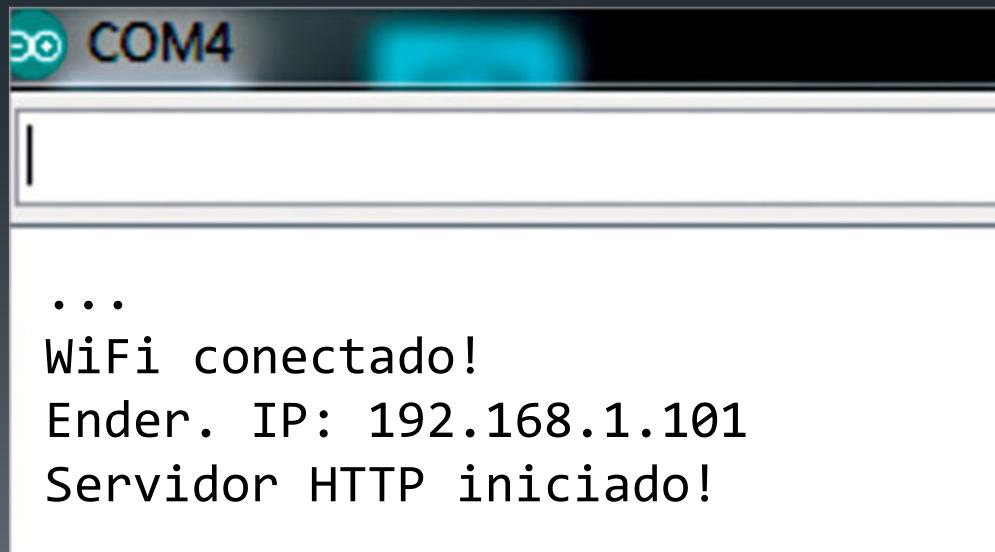
Out/2019

Prof. Kaw

16

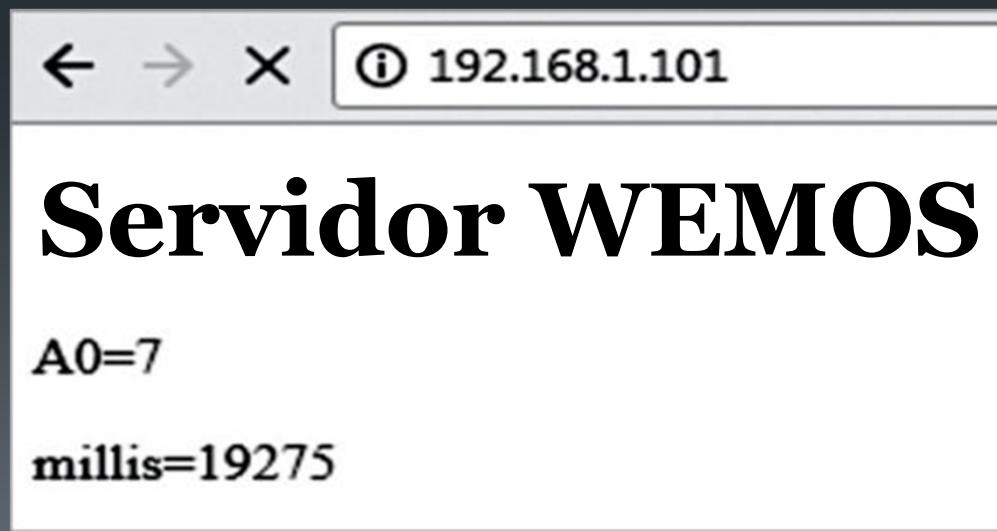


- Programando a **WeMos D1** pela IDE Arduino
 - Depois de compilar, faça o *upload* do código para a placa WeMos D1
 - Abra o **Serial Monitor** para visualizar o endereço IP da placa WeMos D1 atribuído pela sua rede





- Programando a **WeMos D1** pela IDE Arduino
 - Em seguida, abra o navegador Web (cliente) e digite o endereço IP da placa WeMos D1
 - O navegador da Web solicitará o conteúdo da página HTML. A placa WeMos D1 (servidor) responderá com o conteúdo da página, a qual terá o valor da tensão flutuante capturado no pino **A0** e o valor corrente da **millis()**





- Programando a WeMOS pela IDE Arduino
 - Os valores de A0 e de *millis()* são atualizados a cada três segundos
 - Você pode alterar essa taxa de atualização alterando o número 3 na seguinte linha no código-fonte:

```
message += "<meta http-equiv= 'refresh' content='3'>";
```
 - Para mostrar a temperatura ambiente na página HTML, conecte o sensor de temperatura (LM35) ao pino A0 da placa WeMOS
 - Acrescente as linhas de código e altere o conteúdo da página para mostrar a temperatura (função *handleRoot()*):

```
float tensao, temper;  
...  
tensao = 3.3 * analogRead(A0) / 1024  
temper = tensao / 0.01 // 10 mV/°C
```

Aplicação 1

Out/2019

Prof. Kaw

19

```
void handleRoot() {  
    float tensao, temper;  
    String message = "<html><body>\n";  
    message += "<h1>Servidor WEMOS</h1>";  
    message += "<p>A0=";  
    tensao = 3.3 * analogRead(A0) / 1024;  
    message += tensao;  
    message += "</p>";  
    message += "<p>millis =";  
    message += millis();  
    message += "</p>";  
    message += "<p>Temperatura (oC) =";  
    temper = tensao / 0.01 // LM35: 10 mV/oC  
    message += temper;  
    message += "</p>";  
    message += "<meta http-equiv= 'refresh' content='3'>";  
    message += "</body>";  
    message += "</html>\n";  
    server.send(200, "text/html", message);  
}
```



Monitor de Temperatura, Umidade e Ponto de Orvalho* via *Cloud*
Aplicação 2

* Temperatura na qual o vapor de água suspenso no ar começa a condensar (vira "orvalho")

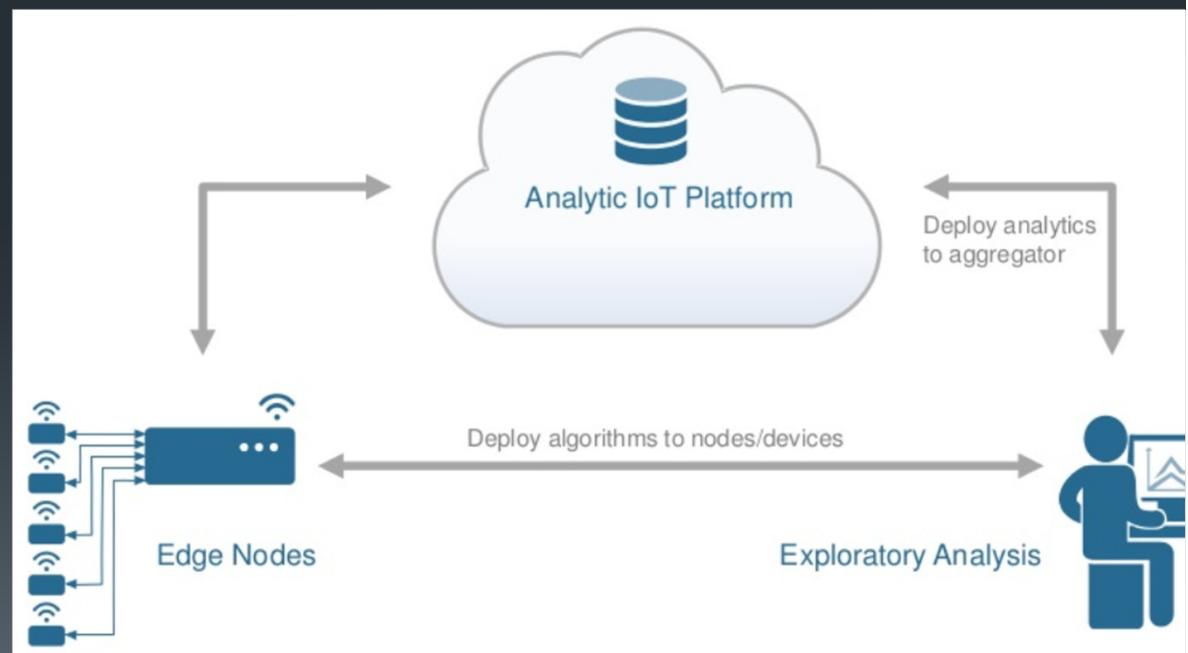
Aplicação 2

Out/2019

Prof. Kaw

21

- Monitor de Temperatura e Umidade via *Cloud*
 - ThingSpeak
 - Plataforma IoT Analítica
 - Coleta de dados de sensores (coisas)
 - Visualização de dados instantaneamente
 - Análise de Dados
 - Integração do MatLab:
execução de códigos
escalonados sobre os
dados coletados
 - Atuação nos Dados
 - Gerar notificação quando
a temperatura de um
processo monitorado
atingir certo valor...



Aplicação 2

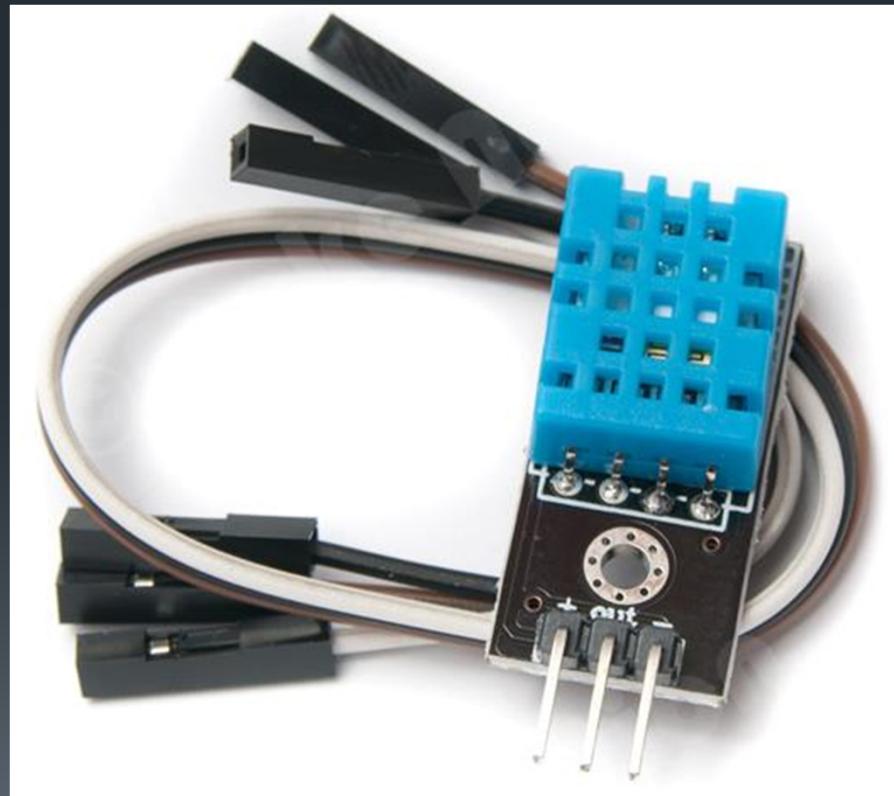
Out/2019

Prof. Kaw

22



- Módulo de Temperatura e Umidade - DHT11
 - O módulo inclui um resistor sensível à umidade e um sensor de temperatura NTC conectado a um microcontrolador de 8 bits



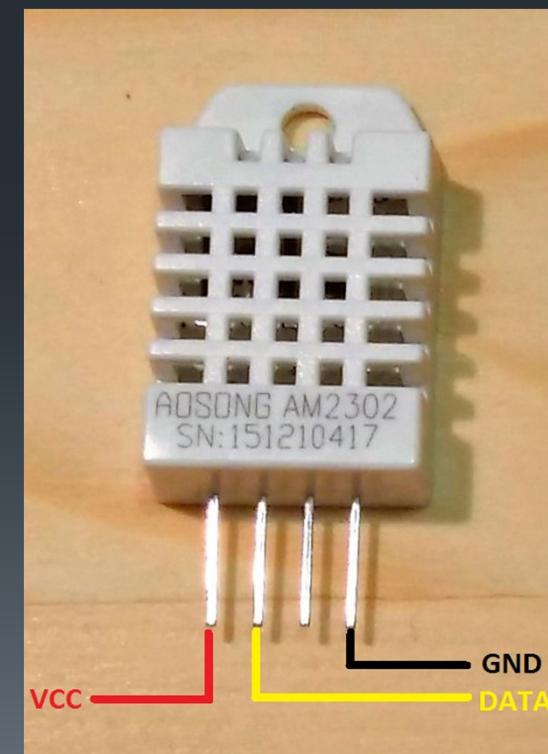
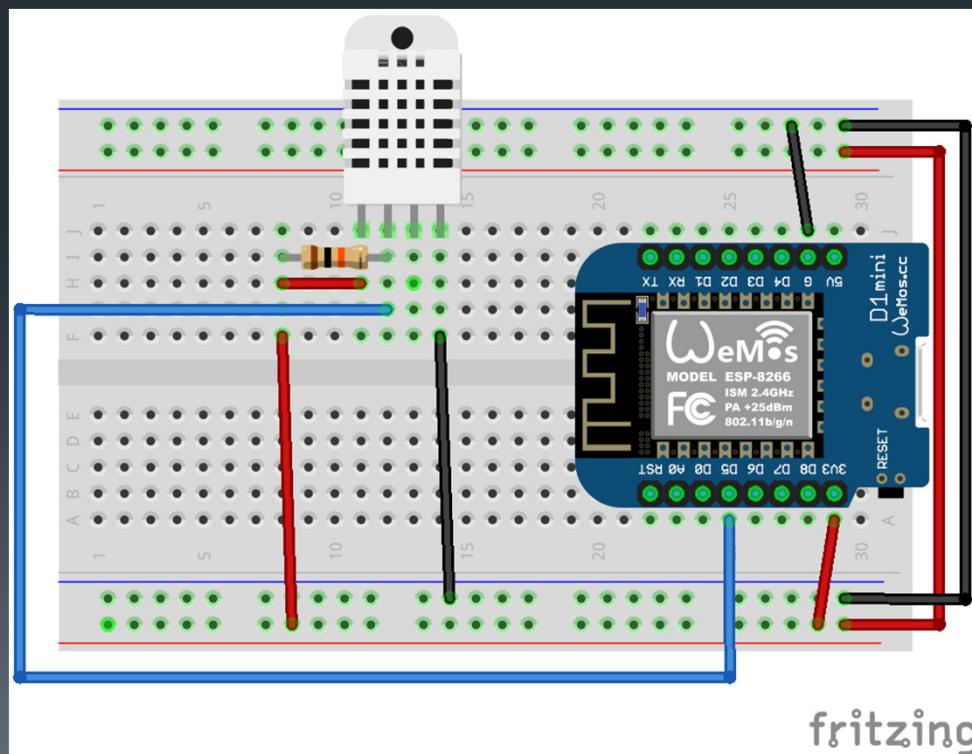
Aplicação 2

Out/2019

Prof. Kaw

23

- Sensor de Temperatura e Umidade - DHT11 (AM2302 ≡ DHT22)
 - Pull-up de 10 kOhm para o sinal de dados (leituras a cada 2 s)
 - Sensor de umidade relativa do ar
 - Tipo capacitivo (2,5 mA), 0 a 100%, acuidade $\pm 2\%$
 - Sensor de Temperatura (DS18B20)
 - Faixa de -40°C a 80°C ; acuidade $\pm 0,5^{\circ}\text{C}$



Aplicação 2

Out/2019

24

Prof. Kaw



■ Acesso à *Cloud*

1. Criar conta no site thingspeak.com
2. Configuração de Canal (até 8 campos de dados por canal)
 - a) Exibição dos dados publicados pelo nosso dispositivo (WeMos)
 - b) Criação de três campos: temperatura, umidade, ponto de orvalho (calc.)

ThingSpeak.com é um aplicativo e API de Internet das Coisas (IoT) de código aberto para armazenar e recuperar dados de coisas usando o protocolo HTTP pela Internet via LAN ou WLAN

The screenshot shows the 'Channel Settings' page of a ThingSpeak channel. The top navigation bar includes 'Private View', 'Public View', 'Channel Settings' (which is active), 'Sharing', 'Chiavi API', and 'Data Import / Export'. The main section is titled 'Channel Settings' and displays the following information:

- Percentage complete: 50%
- ID Canale: 634036
- Nome: Sample Wemos D1 Temperature
- Descrizione: Example
- Campo 1: TEMPERATURE (C) (checkbox checked)
- Campo 2: HUMIDITY (%) (checkbox checked)
- Campo 3: DEW POINT (C) (checkbox checked)

To the right, there is a sidebar titled 'Aiuto' (Help) with a detailed description of channels and their fields. Below the help section is another sidebar titled 'Channel Setting' with a bulleted list of instructions for setting up fields.

Aiuto

Channels store all the data in eight fields that can hold any status data. Once you collect enough data, you can visualize it.

Channel Setting

- **Channel Name:** Enter a name for your channel.
- **Description:** Enter a brief description of your channel.
- **Field#:** Check the boxes to see which channel can have up to eight fields.
- **Metadata:** Enter information about your channel.
- **Tags:** Enter keywords describing your channel.
- **Link to External Site:** Enter the URL of a website related to your ThingSpeak channel.

Aplicação 2

- Código WeMos

```
#include <ESP8266WiFi.h>
#include <stdio.h>

#include <ThingSpeak.h>
#include <Wire.h>
#include <DHT.h>
#include <Arduino.h>
#include <cctype.h>           // para verificação 'isNumber'

// Configurações Wifi
char ssid[] = "XXXXXXXXXXXXXX";    // SSID (nome) da sua rede
char pass[] = "XXXXXXXXXXXXXX";    // senha da sua rede
float dewpt = 0;                  // ponto de orvalho
WiFiClient client;

// Inicia sensor DHT11
#define DHTPIN 14                 // pino digital em que está conectado

// retire o comentário do tipo de sensor que vc estiver usando!
#define DHTTYPE DHT11             // DHT11
//#define DHTTYPE DHT21           // DHT21 (AM2301)
//#define DHTTYPE DHT22           // DHT22 (AM2302), AM2321

// DHT11 - Conecte o pino 1 (o da esquerda) do sensor em +5V
// Obs.: se usar placa com lógica 3.3V tal qual Arduino Due conecte o pino 1 em 3.3V
// Conecte o pino 2 do sensor ao pino colocado em DHTPIN
// Conecte o pino 4 (o da direita) do sensor ao Terra (GND)
// Conecte um resistor de 10 kOhm do pino 2 (dados) ao pino 1 (alim.) do sensor
```



Aplicação 2

Out/2019

Prof. Kaw

26

```
// Inicia o sensor DHT
// Versões antigas dessa biblioteca toma um terceiro parâmetro opcional para
// ajustar os tempos para processadores mais rápidos. Este parâmetro não é mais necessário
// pois o algoritmo de leitura DHT atual ajusta-se autom. ao trabalho em processadores mais rápidos
DHT dht(DHTPIN,DHTTYPE);

// Configurações ThingSpeak
unsigned long myChannelNumber = 895100;                      // seu canal ThingSpeak
const char *myWriteAPIKey = "UUYPB0NLBWLLS5I5";

void setup() {
    Serial.begin(9600);
    Serial.print("\nSketch: "); Serial.println(__FILE__);
    Serial.print("Uploaded: "); Serial.println(__DATE__);
    Serial.println("\nESTACAO CLIMATICA - Temperatura via Sensor DHT11 - By MR WATT");
    Serial.println("BootStrap...");

    Serial.print("Conectando a "); Serial.println(ssid); // inicia pela conexão da rede WiFi
    WiFi.begin(ssid, pass);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500); Serial.print(".");
    }
    Serial.println("\nWiFi conectado.\nEndereço IP: "); Serial.println(WiFi.localIP());
    pinMode(LED_BUILTIN, OUTPUT);                         // inicia o pino do LED_BUILTIN como saída

    Serial.println("\nIniciando DHT11..."); // testa o sensor DHT11
    dht.begin();
    ThingSpeak.begin(client);                  // inicia ThingSpeak (nuvem)
}
```

Aplicação 2

Out/2019

Prof. Kaw

27



```
void loop() {
    float temperature = dht.readTemperature();
    float humidity = dht.readHumidity();

    // Checa se falhou alguma leitura e abandona precocemente (para tentar novamente).
    if (isnan(humidity) || isnan(temperature)) {
        Serial.println("Falhou a leitura do sensor DHT!"); return; }

    // Calcula o índice de calor em Celsius (isFahrenheit = false)
    float hic = dht.computeHeatIndex(temperature, humidity, false);
    dewpt = dewPoint(temperature, humidity);

    Serial.print("\t Temperatura DHT11: "); Serial.print(temperature); Serial.print("C ");
    Serial.print("\t Umidade: "); Serial.print(humidity); Serial.print("%");
    Serial.print("\t Ponto de Orvalho: "); Serial.println(dewpt);

    // Grava em ThingSpeak
    ThingSpeak.setField(1,temperature); ThingSpeak.setField(2,humidity);
    ThingSpeak.setField(3,dewpt); ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);

    delay(1000); client.stop(); delay(5000); // espera um poucos segundos entre medidas
}

void printWifiStatus() {                                // mostra o status da rede Wifi
    Serial.print("SSID: "); Serial.println(WiFi.SSID());
    IPAddress ip = WiFi.localIP(); Serial.print("IP Address: "); Serial.println(ip);
    Serial.print("signal strength (RSSI):"); Serial.print(WiFi.RSSI()); Serial.println(" dBm");
}
```



Aplicação 2

Out/2019

28

Prof. Kaw



```
// Calcula o ponto de orvalho (Dew Point)
double dewPoint(double temper, double h) {
    double A0 = 373.15/(273.15 + temper);
    double SUM = -7.90298 * (A0-1);
    SUM += 5.02808 * log10(A0);
    SUM += -1.3816e-7 * (pow(10, (11.344*(1-1/A0))) - 1);
    SUM += 8.1328e-3 * (pow(10, (-3.49149*(A0-1))) - 1);
    SUM += log10(1013.246);
    double VP = pow(10, SUM-3) * h;
    double T = log(VP/0.61078);
    return (241.88 * T)/(17.558 - T);
}
```

