|  |  |
| --- | --- |
| Ονοματεπώνυμο | ΑΕΜ |
| Παντζαρτζής Αναστάσιος | 106 |
| Τιμολέων Κομνηνάκης | 108 |

# Raspberry Pi Pico

Τα Raspberry Pi Pico είναι μια σειρά από χαμηλού κόστους microcontroller boards από την εταιρία Raspberry Pi για χρήση σε διάφορα project. Υπάρχουν 4 boards συνολικά, οι οποίες διαφέρουν με βάση την υποστήριξη και μη υποστήριξη Wi-Fi και Bluetooth και με την ύπαρξη ή όχι των (soldered) headers. Περιέχει:

* 26 GPIO Pins
* μία θύρα USB 1.1
* ενσωματωμένο αισθητήρα θερμοκρασίας
* τον RP2040 MCU
* 264 kB SRAM
* 2MB onboard flash

Επιπλέον, στην περίπτωση της υποστήριξης Wi-Fi & Blueotooth περιέχει το chip Infineon CYW43439 με το Wireless πρωτόκολλο 802.11n (single band – 2.4GHz), WPA3, Bluetooth 5.2 με υποστήριξη για Bluetooth LE με ρόλους Central και Peripheral και για Bluetooth Classic.

Ο μικροελεγκτής RP2040, έχει σχεδιαστεί από την Raspeberry Pi και περιέχει 2 πυρήνες ARM Cortex-M0+ στα 133 MHz (flexible) και 264kB SRAM. Υποστηρίζει έως 16MB μνήμης και περιέχει έναν Direct Memory Access (DMA) controller.

# Κώδικας Arduino

Το project αποτελείτε από 3 αρχεία συνολικά, τα sketch\_iot.ino, DHTReadings.h και config.h. Το αρχείο config.h είναι ένα *εικονικό* αρχείο configuration όπου περιέχει τις παραμέτρους:

* WiFi SSID
* WiFi Password
* MQTT Broker
* MQTT Port
* MQTT Topic
* Use static IP (δεν χρησιμοποιείται)

Στο αρχείο DHTReadings.h βρίσκεται η κλάση DHTReadings, η οποία κρατάει τα δεδομένα θερμοκρασίας και υγρασίας και ουσιαστικά αντιπροσωπεύει την ανάγνωση των τιμών από τον αισθητήρα DHT μαζί με κάποιες επιπλέον μεθόδους για τον έλεγχο της ορθότητας και τον υπολογισμό του heat index.

Το sketch\_iot.ino είναι το κεντρικό sketch αρχείο όπου περιέχει όλη την λειτουργικότητα και βοηθητικές μεθόδους και δεδομένα.

# ΠΑΡΑΡΤΗΜΑ

Αρχείο «sketch\_iot.ino»

#include <DHT.h>

#include <WiFi.h>

#include <ArduinoMqttClient.h>

#include "DHTReadings.h"

// DHT is connected at pin #2

#define DHTPIN 2u

// Type of DHT is DHT22

#define DHTTYPE DHT22

#define SUBQOS 2

**using** config\_id\_t **=** enum ConfigID **:** size\_t **{**

SSID**,**

PASSWORD**,**

MQTT\_BROKER**,**

MQTT\_PORT**,**

MQTT\_TOPIC**,**

USE\_STATIC\_IP

**};**

const char **\*** const config**[]** **=** **{**

// The config.h file must contain

// 2 strings (separated by comma

// of course). The first one is

// the WiFi SSID while the second

// one is the Password.

#include "config.h"

**};**

const unsigned long interval **=** 8000**;**

unsigned long previousMillis **=** 0**;**

DHT dht**(**DHTPIN**,** DHTTYPE**);**

WiFiClient wifiClient**{** **};**

MqttClient mqttClient**{** wifiClient **};**

/\*\* Compare a "bool" string to

\* to a bool value. The string

\* is NOT converted to lowercase

\* before checking.

\*/

bool boolstrcmp**(**const char **\***str**,** bool value**)** **{**

**return** **(**bool**)(**String**(**str**)** **==** **((**value**)** **?** "true" **:** "false"**));**

**}**

bool get\_validated\_port**(**int **\***outPort**)** **{**

**if** **(**outPort **!=** **nullptr)** **{**

String strPort **=** String**(**config**[**MQTT\_PORT**]);**

long bufPort **=** strPort**.**toInt**();**

**\***outPort **=** **(**int**)**bufPort**;**

**return** **!(**

bufPort **==** 0l **&&**

strPort **==** "0" **&&**

bufPort **!=** **((**long**)\***outPort**)**

**);**

**}**

**return** **false;**

**}**

bool should\_publish**()** **{**

unsigned long currentMillis **=** millis**();**

**if** **(**currentMillis **-** previousMillis **>=** interval**)** **{**

previousMillis **=** currentMillis**;**

**return** **true;**

**}**

**return** **false;**

**}**

void publish\_sensor\_data**(**const DHTReadings**&** readings**,** const float cHeatIndex**,** const float fHeatIndex**)** **{**

// Publich humidity

mqttClient**.**beginMessage**(**String**(**config**[**MQTT\_TOPIC**])** **+** "/humidity"**);**

mqttClient**.**print**(**readings**.**humidity**());**

mqttClient**.**endMessage**();**

// Publich temperature (Celsius)

mqttClient**.**beginMessage**(**String**(**config**[**MQTT\_TOPIC**])** **+** "/ctemp"**);**

mqttClient**.**print**(**readings**.**cTemperature**());**

mqttClient**.**endMessage**();**

// Publich temperature (Fahrenheit)

mqttClient**.**beginMessage**(**String**(**config**[**MQTT\_TOPIC**])** **+** "/ftemp"**);**

mqttClient**.**print**(**readings**.**fTemperature**());**

mqttClient**.**endMessage**();**

// Publich head index (Celsius)

mqttClient**.**beginMessage**(**String**(**config**[**MQTT\_TOPIC**])** **+** "/cheatindex"**);**

mqttClient**.**print**(**cHeatIndex**);**

mqttClient**.**endMessage**();**

// Publich head index (Fahrenheit)

mqttClient**.**beginMessage**(**String**(**config**[**MQTT\_TOPIC**])** **+** "/fheatindex"**);**

mqttClient**.**print**(**fHeatIndex**);**

mqttClient**.**endMessage**();**

**}**

void print\_sensor\_data**(**const DHTReadings**&** readings**,** const float cHeatIndex**,** const float fHeatIndex**)** **{**

Serial**.**print**(**F**(**"Humidity: "**));**

Serial**.**print**(**readings**.**humidity**());**

Serial**.**print**(**F**(**"% Temperature: "**));**

Serial**.**print**(**readings**.**cTemperature**());**

Serial**.**print**(**F**(**"°C "**));**

Serial**.**print**(**readings**.**fTemperature**());**

Serial**.**print**(**F**(**"°F Heat index: "**));**

Serial**.**print**(**cHeatIndex**);**

Serial**.**print**(**F**(**"°C "**));**

Serial**.**print**(**fHeatIndex**);**

Serial**.**println**(**F**(**"°F"**));**

**}**

void on\_mqtt\_message**(**int messageSize**)** **{**

Serial**.**println**();**

Serial**.**print**(**"Received a message with topic '"**);**

Serial**.**print**(**mqttClient**.**messageTopic**());**

Serial**.**print**(**"', duplicate = "**);**

Serial**.**print**(**mqttClient**.**messageDup**()** **?** "true" **:** "false"**);**

Serial**.**print**(**", QoS = "**);**

Serial**.**print**(**mqttClient**.**messageQoS**());**

Serial**.**print**(**", retained = "**);**

Serial**.**print**(**mqttClient**.**messageRetain**()** **?** "true" **:** "false"**);**

Serial**.**print**(**"', length "**);**

Serial**.**print**(**messageSize**);**

Serial**.**println**(**" bytes:"**);**

// use the Stream interface to print the contents

char msg**[(**size\_t**)** messageSize **+** 1u**]** **=** **{** '\0' **};**

size\_t i **=** 0**;**

**while** **(**mqttClient**.**available**()** **&&** i **<** messageSize**)** **{**

msg**[**i**]** **=** **(**char**)**mqttClient**.**read**();**

**++**i**;**

**}**

Serial**.**println**(**msg**);**

**if** **(**String**(**msg**)** **==** "alert"**)** **{**

digitalWrite**(**LED\_BUILTIN**,** HIGH**);**

**}** **else** **if** **(**String**(**msg**)** **==** "stopalert"**)** **{**

digitalWrite**(**LED\_BUILTIN**,** LOW**);**

**}** **else** **{**

Serial**.**println**(**"Unknown message"**);**

**}**

Serial**.**println**();**

**}**

void setup**()** **{**

// The baud rate is ignored as is

// a USB connection. The method

// has a default value of 115200

// for parameter "baud" which is

// later ignored.

Serial**.**begin**(**/\*115200\*/**);**

pinMode**(**LED\_BUILTIN**,** OUTPUT**);**

// Set the WiFi mode to STATION

WiFi**.**mode**(**WIFI\_STA**);**

// Start the WiFi connection

WiFi**.**begin**(**config**[**SSID**],** config**[**PASSWORD**]);**

Serial**.**print**(**F**(**"Connecting"**));**

// Wait till the device connects to

// the WiFi. Wait at least 1 second

// between status checks.

**while** **(**WiFi**.**status**()** **!=** WL\_CONNECTED**)** **{**

delay**(**500u**);**

Serial**.**print**(**F**(**"."**));**

delay**(**500u**);**

**}**

// After the device connects print the SSID

Serial**.**println**(**F**(**""**));**

Serial**.**println**(**F**(**"Pico W is connected to WiFi network"**));**

Serial**.**println**(**WiFi**.**SSID**());**

**if** **(**boolstrcmp**(**config**[**USE\_STATIC\_IP**],** **true))** **{**

// ...

**}**

// Also, print the IP address of the device

Serial**.**print**(**F**(**"Assigned IP Address: "**));**

Serial**.**println**(**WiFi**.**localIP**());**

int port **=** 0**;**

**if** **(!**get\_validated\_port**(&**port**))** **{**

Serial**.**printf**(**"Validating port %s failed\n"**,** config**[**MQTT\_PORT**]);**

**}** **else** **{**

**if** **(!**mqttClient**.**connect**(**config**[**MQTT\_BROKER**],** port**))** **{**

Serial**.**print**(**F**(**"MQTT connection failed! Error code: "**));**

Serial**.**println**(**mqttClient**.**connectError**());**

**}** **else** **{**

mqttClient**.**onMessage**(**on\_mqtt\_message**);**

mqttClient**.**subscribe**(**String**(**config**[**MQTT\_TOPIC**])** **+** "/alerts"**,** SUBQOS**);**

Serial**.**println**(**"Connected to the MQTT broker!"**);**

**}**

**}**

// Start measuring

dht**.**begin**();**

**}**

void loop**()** **{**

// Wait 2 seconds between measurements

//delay(2000u);

mqttClient**.**poll**();**

**if** **(**should\_publish**())** **{**

// Reading temperature or humidity

// takes about 250 ms (sensor reading

// can be up to 2 seconds old).

DHTReadings readings**{**

// Read humidity

dht**.**readHumidity**(),**

// Read temperature as Celsius (the default)

dht**.**readTemperature**(),**

// Read temperature as Fahrenheit (isFahrenheit = true)

dht**.**readTemperature**(true)**

**};**

// Check if any of the readX calls

// failed. If any one of them failed

// return (in order to try again).

**if** **(**readings**.**validate**())** **{**

Serial**.**println**(**F**(**"Failed to read from DHT22 sensor!"**));**

**return;**

**}**

// Compute heat index in Fahrenheit

float fHeatIndex **=** readings**.**computeHeatIndex**(&**dht**,** **true);**

// Compute heat index in Celsius

float cHeatIndex **=** readings**.**computeHeatIndex**(&**dht**);**

publish\_sensor\_data**(**readings**,** cHeatIndex**,** fHeatIndex**);**

print\_sensor\_data**(**readings**,** cHeatIndex**,** fHeatIndex**);**

**}**

**}**

Αρχείο «DHTReadings.h»

#define DHTREADINGS\_H 1

#include <cmath>

#include <DHT.h>

class DHTReadings **{**

public**:**

inline DHTReadings**(**float humidity**,** float cTemperature**,** float fTemperature**)**

**:**\_humidity**{** humidity **},** \_cTemperature**{** cTemperature **},** \_fTemperature**{** fTemperature **}**

**{** **}**

DHTReadings**(**const DHTReadings**&)** **=** **default;**

virtual inline **~**DHTReadings**()** **{** **}**

DHTReadings**&** **operator=(**const DHTReadings**&)** **=** **default;**

inline float humidity**()** const **{**

**return** **this->**\_humidity**;**

**}**

inline float cTemperature**()** const **{**

**return** **this->**\_cTemperature**;**

**}**

inline float fTemperature**()** const **{**

**return** **this->**\_fTemperature**;**

**}**

inline float computeHeatIndex**(**DHT **\***const dht**,** bool forFahrenheit **=** **false)** **{**

float temp **=** **(**forFahrenheit**)** **?** **this->** \_fTemperature **:** **this->**\_cTemperature**;**

**return** dht**->**computeHeatIndex**(**temp**,** **this->**\_humidity**,** forFahrenheit**);**

**}**

inline bool validate**()** **{**

**return** **(**

std**::**isnan**(this->**\_humidity**)** **||**

std**::**isnan**(this->**\_cTemperature**)** **||**

std**::**isnan**(this->**\_fTemperature**)**

**);**

**}**

private**:**

float \_humidity**;**

float \_cTemperature**;**

float \_fTemperature**;**

**};**

#endif // DHTREADINGS\_H

Αρχείο «config.h»

// WiFi SSID:

// Replace <WiFi SSID> with the

// actual SSID

"<WiFi SSID>"**,**

// WiFi Password:

// Replace <WiFi Password> with

// the actual password

"<WiFi Password>"**,**

// MQTT Broker:

// Replace <MQTT Broker> with

// the URL of the broker

"<MQTT Broker>"**,**

// MQTT Port:

// Replace <MQTT Port> with

// the port at which the broker

// listens to

"<MQTT Port>"**,**

// MQTT Topic:

// Replace <MQTT Topic> with

// the \*main\* topic

"<MQTT Topic>"**,**

// Use Static IP:

// ...

"false"

Node-RED

|  |  |
| --- | --- |
| Node | Λειτουργικότητα |
| Mqtt in (humidity, ctemp, cheatindex):  authIoTTestTopic/(humidity, ctemp, cheatindex) | Λαμβάνουν δεδομένα από τον server:  test.mosquitto.org στο port 1883 |
| Switch (Check humidity, Check temp) | Ελέγχουν αν οι τιμές υγρασίας > 70% και θερμοκρασία > 30 |
| Timestamp | Ορίζει τους χρόνους κατά το οποίους εκτελούνται διαδικασίες |
| Chart | Για την εμφάνιση των διαγραμμάτων |
| Functions | Συναρτήσεις σε JavaScript για την εκτέλεση της λογικής του προγράμματος |
| SQLite node | Για την αποθήκευση των δεδομένων σε τοπική βάση δεδομένων SQLite |
| Debug | Για την δοκιμή (debugging) του προγράμματος |
| Join | Συγχωνεύει δυο ή περισσότερα μηνύματα σε ένα |

Περιγραφή Συναρτήσεων

* Parse data (temp): Λαμβάνει και επιστρέφει την τιμή της θερμοκρασίας

var temp = msg.payload;

msg.payload = temp;

return msg;

* Parse data (heat index): Λαμβάνει και επιστρέφει την τιμή του heat index

var heat = msg.payload;

msg.payload = heat;

return msg;

* Prepare ctemp:

msg.topic = msg.topic.split("/")[1]; //Διαχωρίζει το ‘msg.topic’ με βάση τον χαρακτήρα ‘/’ και επιλέγει το δεύτερο στοιχείο του πίνακα.

msg.payload = msg.payload > 30.0; //Γίνεται έλεγχος αν το ‘msg.payload’ > 30

return msg; //Επιστρέφει το αντικείμενο του μηνύματος

* Prepare humidity:

msg.topic = msg.topic.split("/")[1]; //Διαχωρίζει το ‘msg.topic’ με βάση τον χαρακτήρα ‘/’ και επιλέγει το δεύτερο στοιχείο του πίνακα.

msg.payload = msg.payload > 70.0; //Γίνεται έλεγχος αν το ‘msg.payload’ > 70

return msg; //Επιστρέφει το αντικείμενο του μηνύματος

* Extract data:

Ανάλυση τύπου δεδομένων: Η πρώτη γραμμή κώδικα χωρίζει το θέμα του μηνύματος για να εξάγει τον τύπο των δεδομένων που περιέχει το μήνυμα. Αυτός ο τύπος δεδομένων αναμένεται να βρίσκεται στο δεύτερο μέρος του θέματος.

Αποθήκευση δεδομένων: Η συνέχεια του κώδικα πραγματοποιεί μια σειρά ελέγχων χρησιμοποιώντας τον τύπο δεδομένων. Ανάλογα με τον τύπο δεδομένων, η τιμή του μηνύματος αποθηκεύεται σε ένα αντίστοιχο πεδίο δεδομένων.

Αποθήκευση σε ροή: Οι τιμές αποθηκεύονται σε ένα πεδίο δεδομένων με τη χρήση της λειτουργίας ροής (flow). Αυτό επιτρέπει την αποθήκευση των τιμών για μελλοντική χρήση στο πλαίσιο της εφαρμογής.

// Extract the type of data from the topic

let dataType = msg.topic.split("/")[1]; // Extracts the second part of the topic

// Depending on the data type, the value into the corresponding array

if (dataType === "humidity") {

let humidity = msg.payload;

let humidityValues = flow.get("humidityValues") || [];

humidityValues.push(humidity);

flow.set("humidityValues", humidityValues);

} else if (dataType === "ctemp") {

let temperature = msg.payload;

let tempValues = flow.get("tempValues") || [];

tempValues.push(temperature);

flow.set("tempValues", tempValues);

} else if (dataType === "cheatindex") {

let heatIndex = msg.payload;

let heatIndexValues = flow.get("heatIndexValues") || [];

heatIndexValues.push(heatIndex);

flow.set("heatIndexValues", heatIndexValues);

}

return null;

* passToDB: Προετοιμάζει τους μέσους όρους για να τα περάσει στην βάση δεδομένων.

let theTimeStamp = Math.round(Date.now() / 1000);

let tempValues = flow.get("tempValues") || [];

let heatIndexValues = flow.get("heatIndexValues") || [];

let humidityValues = flow.get("humidityValues") || [];

// Calculate averages

let averageTemp = tempValues.reduce((acc, val) => acc + val, 0) / tempValues.length || 0;

let averageHeatIndex = heatIndexValues.reduce((acc, val) => acc + val, 0) / heatIndexValues.length || 0;

let averageHumidity = humidityValues.reduce((acc, val) => acc + val, 0) / humidityValues.length || 0;

// Construct SQL query to insert averages

let sql = `INSERT INTO hourly\_averages (timestamp, average\_temp, average\_heat\_index, average\_humidity) VALUES (${theTimeStamp}, ${averageTemp}, ${averageHeatIndex}, ${averageHumidity});`;

// Reset the arrays in the context for the next hour

flow.set("tempValues", []);

flow.set("heatIndexValues", []);

flow.set("humidityValues", []);

// Prepare the message object to send to the SQLite node

msg.topic = sql;

return msg;

* Send data: Προετοιμάζει τα δεδομένα και το query για την αποθήκευση των δεδομένων της θερμοκρασίας και του heat index στην SQLite βάση δεδομένων.

let theTimeStamp = Math.round(Date.now() / 1000);

// Retrieve context variables

let temp = flow.get("temp") || null;

let heatIndex = flow.get("heatIndex") || null;

// Determine message topic and store the value

if (msg.topic === "authIoTTestTopic/ctemp") {

temp = msg.payload;

flow.set("temp", temp);

} else if (msg.topic === "authIoTTestTopic/cheatindex") {

heatIndex = msg.payload;

flow.set("heatIndex", heatIndex);

}

// Check if both values are available

if (temp !== null && heatIndex !== null) {

// Construct SQL query

let sql = `INSERT INTO sensor\_data (timestamp, temperature, heat\_index) VALUES ('${theTimeStamp}', '${temp}', '${heatIndex}');`;

msg.topic = sql; // Ensure to set your SQL query in msg.payload or msg.topic as per your requirement

// Reset context variables

flow.set("temp", null);

flow.set("heatIndex", null);

return msg; // Return the msg object with SQL query

}

// Return null if both values are not yet received

return null;

* Check: Ελέγχει αν η θερμοκρασία ή η υγρασία έχει ξεπεράσει του 30 βαθμούς ή το 70% υγρασίας.

if (msg.payload.humidity === true || msg.payload.ctemp === true) {

msg.payload = "alert";

} else {

msg.payload = "stopalert";

}

return msg;