Is the A/C on? Final Project

Design Architecture Document

Microprocessors I Fall Semester 2022

Edmarie Santana Camacho

Perla N. Colón Marrero

Jonhuel A. De León Hernandez

Contents

Introduction

This document will address the background of this project and the significant functional requirements of the architecture. To carry out the project, an ESP32 microprocessor and an analog temperature sensor(LMT84) were used to read the ambient temperature. One of the main goals was to demonstrate IoT knowledge by sending an HTTP request from the ESP32 microprocessor to a cloud instance running on Node-Red. From the programming area, the C language was used, with the implementation of the MQTT library, which allows sensor data to be sent to a web server provided by Amazon Web Service. Node-Red was used next to the server to display the ambient temperature in real time via a Siri shortcut or from the correct domain with the creation of a web page. An LED light was implemented in the circuit together with ESP32 and the temperature sensor to demonstrate that the system runs without the need to be connected to the computer (it runs standalone, with battery, and without USB).

Architecture Views:

The Logical View Architecture

From a logical architecture perspective, the system used sensors, transmissions, and data processing. To achieve this, we work with communication and cloud server components. The use of temperature sensors was implemented with an ESP32 device. After the sensors measure the temperature values, the data is transmitted, sending it to a central computer in the cloud, where it can be stored in a database, using wireless communication technologies such as WiFi. Users must be able to access the system through a website or through a voice assistant such as Siri, which allows them to interact with the system to learn temperature measurements.

The Process View Architecture

The system process architecture view was split to create a better structure and ease the project process. As a first step, all the necessary tools to carry out the project were installed; Software, Hardware, Arduino IDE, MQTT, GitHub, Amazon Web Service, etc. Then, an execution plan was created, along with an Architecture and Design Document.

The Development View Architecture

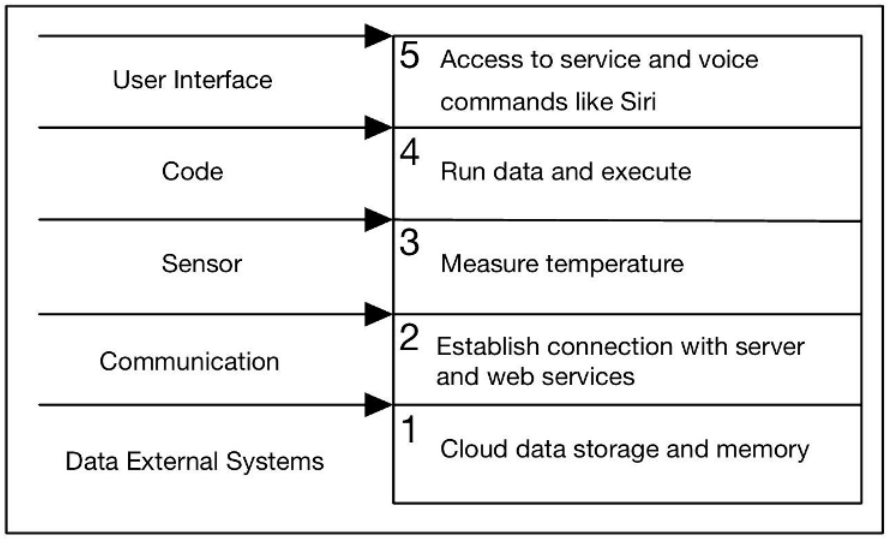

Figure 1 - Diagram for The Development View Architecture

In Figure 3, the user interface layer and the communication layer are responsible for providing a way for users to interact with the system and transmit data to other systems, respectively. The code layer is responsible for the high-level logic of the system, such as how often to read the temperature sensor and how to respond to temperature readings. The data layer processes and stores the temperature data received from the sensor layer. Finally, the sensor layer interacts with the temperature sensor and reads its data, while the data layer also provides a way to transmit data to cloud-based data storage services or other systems.

The Physical View Architecture

1. ESP32 Board: This is the main hardware component of the system that collects temperature data from the sensor and communicates it to the cloud server via the Internet.
2. Temperature Sensor: This is the hardware component that measures the temperature of the room.

3. Computer: It is used to generate and manage the software implemented in the project.
4. Battery (the power supply): provides power to the ESP32 board and the temperature sensor.
5. LED Light: This hardware component shows the operation of the system without the need to be connected to the computer (it runs independently, with the battery, and without the need for USB)
6. Cloud Server: The cloud server is responsible for receiving and storing temperature data from the ESP32 board.

Architecture Design:

Software Design

All the code used for this project was implemented using the Arduino IDE and C language. The C language libraries used include: (#include <WiFi.h>) connects the ESP32 to the Wifi, (#include <PubSubClient.h>) is an MQTT library to link the ESP32 data and Node Red via the AWS web server, (#include <Wire.h>) allows to communicate with I2C/TWI (sensors) devices, and (#include "driver/adc.h") provides functions to correct for differences in measured voltages caused by variation of ADC reference voltages (Vref) between chips.

Code in C language using Arduino IDE:

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <Wire.h>
#include "driver/adc.h"

// MQTT domain
const char* mqtt_server = "istheacon.space";

// MQTT Var in Node Red
const char* topic = "temp";

// WiFi SSID
const char* ssid = "RUMNET";

// WiFi Pasword
const char* password = "Colegio2019";

// LED pin in circ (entradas y salidas)
const int led = 2;

WiFiClient espClient; //conecta al wifi con esp32
```

```
PubSubClient     client(espClient);
//conecta esp32 con nodered
long lastMsg = 0; // 64 bits
char msg[50];

float temperature = 0;

const int sensor = 32;

void setup() {

  Serial.begin(115200);

  setup_wifi();
     client.setServer(mqtt_server,
1883);
  client.setCallback(callback);

 pinMode(sensor, INPUT);
 pinMode(led, OUTPUT);


adc1_config_channel_atten(ADC1_CHA
NNEL_4, ADC_ATTEN_DB_11);

adc1_config_width(ADC_WIDTH_BIT_12
); // default
}

void setup_wifi() {
  delay(10);

   // We start by connecting to a
WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

       while    (WiFi.status()    !=
WL_CONNECTED) {
     delay(500);
     Serial.print(".");
   }

  Serial.println("");
     Serial.println("Connected   to
WiFi");
  Serial.println("Domain: ");
  Serial.println(WiFi.localIP());
}

void  callback(char*  topic,  byte*
message, unsigned int length) {
   Serial.print("Message arrived on
topic: ");
  Serial.print(topic);
  Serial.print(". Message: ");
  String messageTemp;

   for (int i = 0; i < length; i++)
{

Serial.print((char)message[i]);
                  messageTemp     +=
(char)message[i];
  }
  Serial.println();


        if     (String(topic)     ==
"esp32/output") {
     Serial.print("Changing output
to ");
  }
```

```
}

void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
                                    if (client.connect("ESP8266Client")) {
      Serial.println("connected");
      // Subscribe

client.subscribe("esp32/output");
    } else {
      Serial.print("failed, rc=");

Serial.print(client.state());
       Serial.println(" try again in 5 seconds");
         // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}
void loop() {
   if (!client.connected()) {
    reconnect();
  }
  client.loop();

  long now = millis();
  if (now - lastMsg > 5000) {
    lastMsg = now;

        float voltage = (float) adc1_get_raw(ADC1_CHANNEL_4) / 4096 * 3.3; //getrawdata
    float temperature = (voltage - 0.5) * 100; //celsius
      temperature = temperature * 1.8 + 32; //farenheit (si se comenta, los grados seran celcios)

    char tempString[8];
       dtostrf(temperature, 1, 2, tempString); // float to string
    Serial.print("Temperature: ");
    Serial.print(tempString);
         Serial.println("°F");  // cambiar por grados C

            client.publish("temp", tempString);

  }

  digitalWrite(led, HIGH);
  delay(100);
  digitalWrite(led, LOW);
  delay(100);
}
```

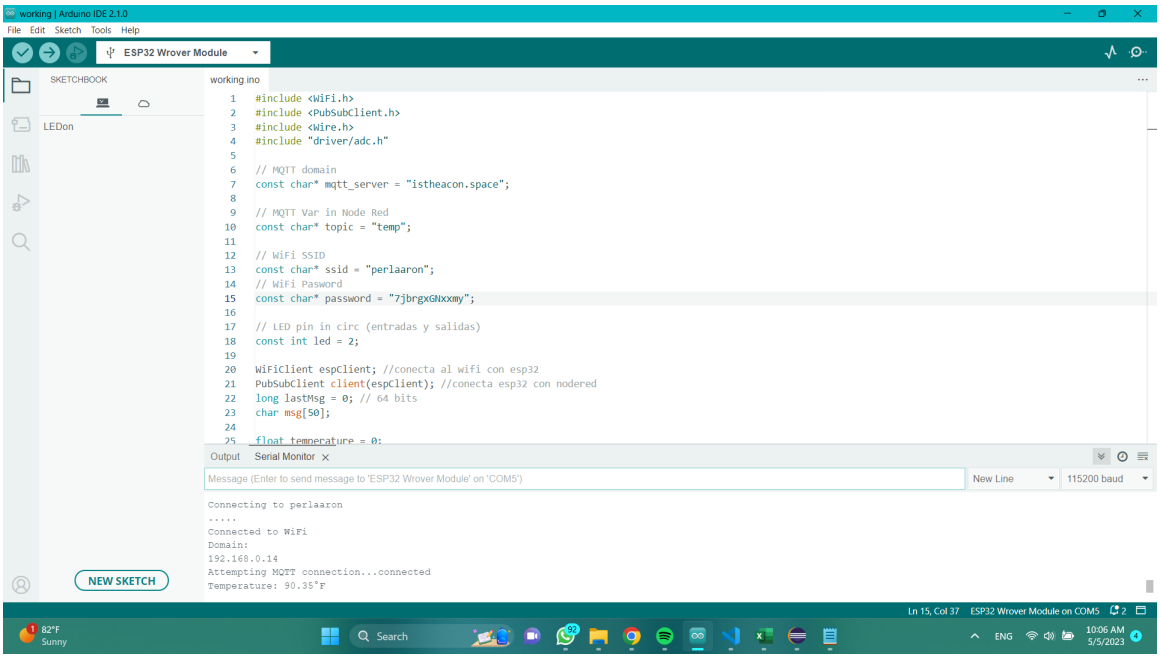

Figure 2 - Code running in Arduino IDE

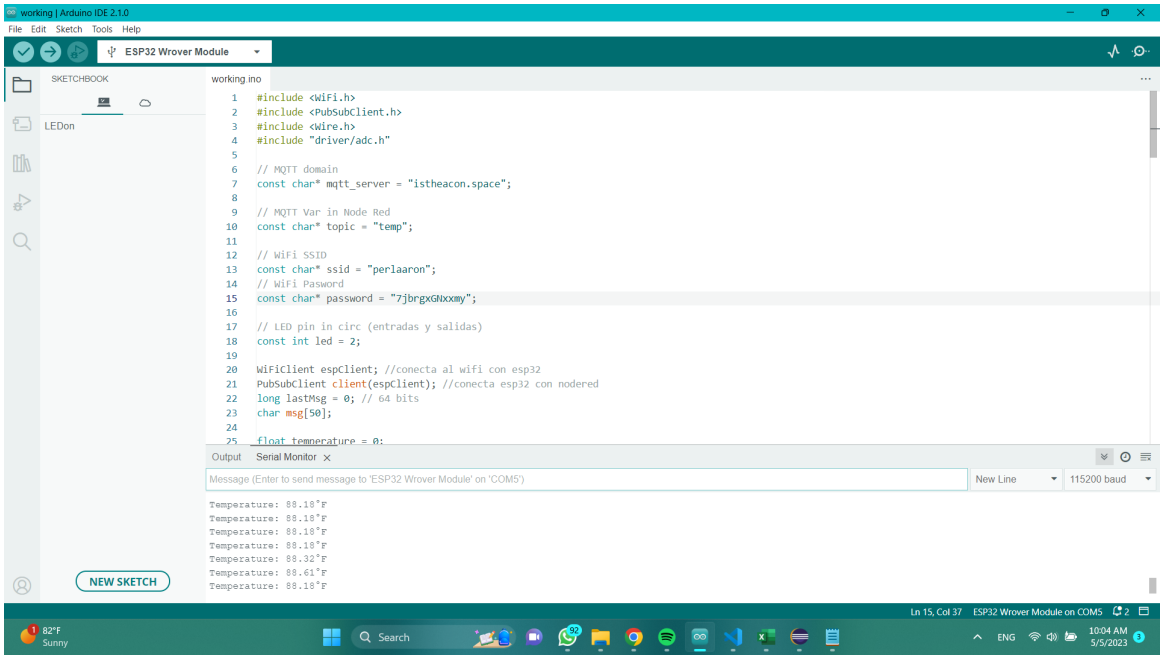

Figure 3 - Temperature printing in Arduino IDE

Node Red Implementation

To implement Node Red, we acquired a domain name (istheacon.space). Using Lightsail, a web server service provided by AWS (Amazon Web Services), we were able to use a Virtual Machine with the Ubuntu operating system. Then this was used to host Node Red, to receive the MQTT data of the ESP32 and display it on a Node Red-generated web page.

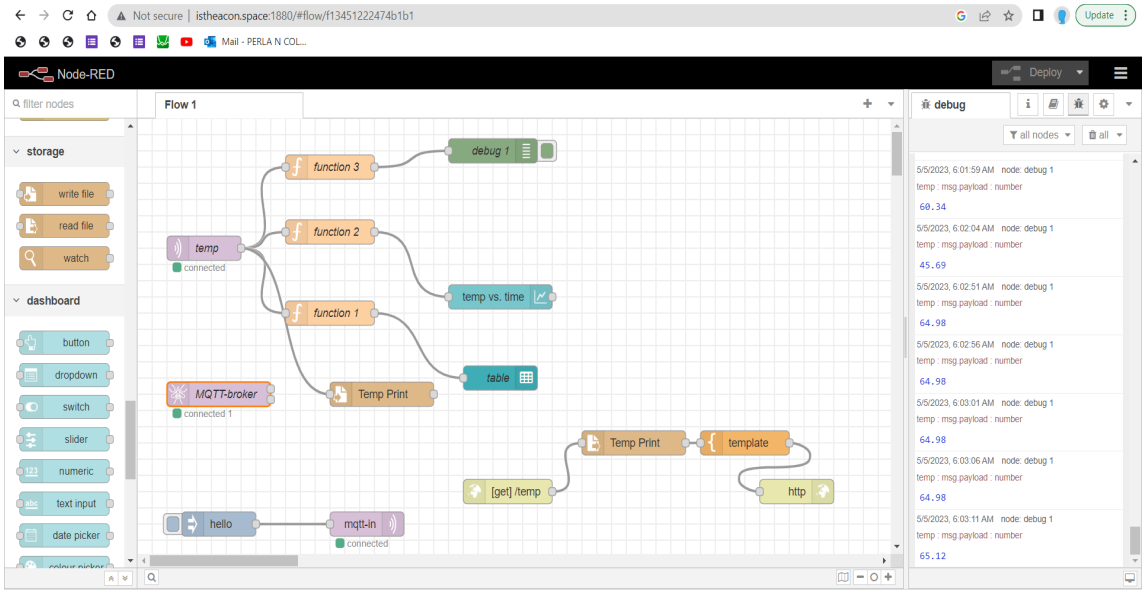

Figure 4 - MQTT Node Red Flow

The display web page with the table (Temp vs Time) can be accessed through the domain “istheacon.space:1880/ui”. The acronym UI stands for user interface, this is where the user will be able to access in real time the temperature readings. As you may notice in Figure 5, there is a noticeable change between reading, this is because it was placed in room temperature first and then in the fridge. This to prove that the temperature sensor is working correctly.

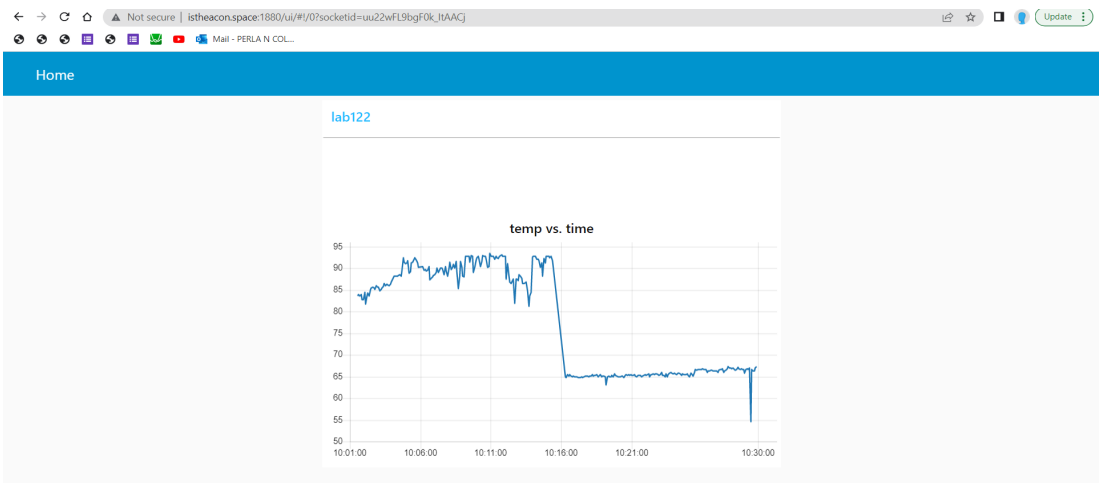

Figure 5 - Node Red Display Web Page

Node Red Documentation

Node Red uses “nodes” in series to generate a display web page. Here it displays a table with the interpolation of temperatures, as seen in Figure 5.

Nodes in the flow in Figure 4:

- Aedes MQTT broker: enables MQTT node functionality.
- MQTT-in Node: receives MQTT data from the ESP32 via the Lightsail Virtual Machine.
- Function Nodes (Function 1, Function 2, Function 3): take msg.payload as input and use it as a parameter for a JavaScript function. These functions convert the MQTT voltage data of the ESP32 into temperature values using the data from Figure 1. Function 1 reformats the converted data to fit the table node, Function 2 reformats the converted data to fit the graph node, and Function 3 debugs.
- Table: presents the input data as a table on the web page.
- Graph node: uses the input data as the y axis in a graph (the x axis is time by default).
- Debug Node: displays the input data in the debugger tab of Node Red.
- Inject Node: injects information to the display web page.

- Template Node: prints the msg string for the display.
- Get/temp Node: displays msg to webpage with readings.

Shortcut Implementation

This shortcut was created on the shortcut app from Apple. The private intelligent assistant known as Siri will be the one communicating the readings. It prompts a query for the user, then goes to the function in the domain http://istheacon.space:1880/temp?room and copies it to the clipboard. It gets the content and prompts a new query for continuity. At last it displays via sound and text the contents of the URL. The URL will include the temperature reading from the ESP32 and Node Red taken with the temperature sensor.

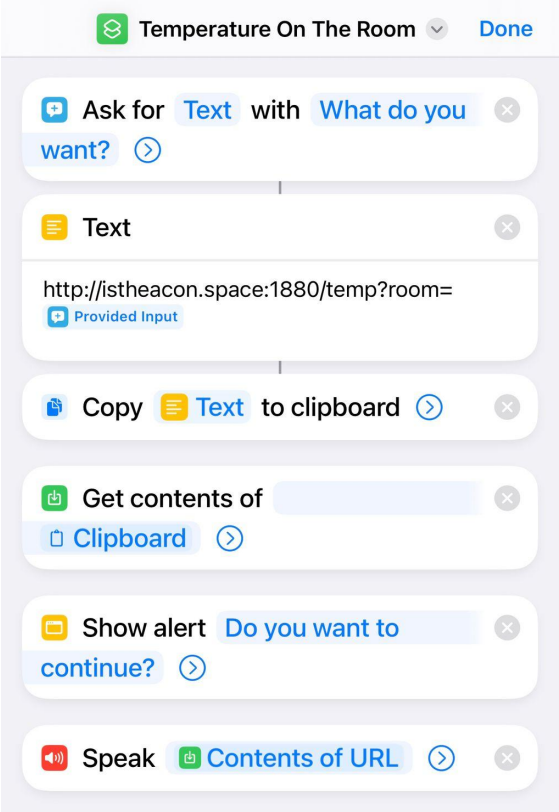

Figure 5 - Siri Shortcut Temperature On The Room

Hardware Design

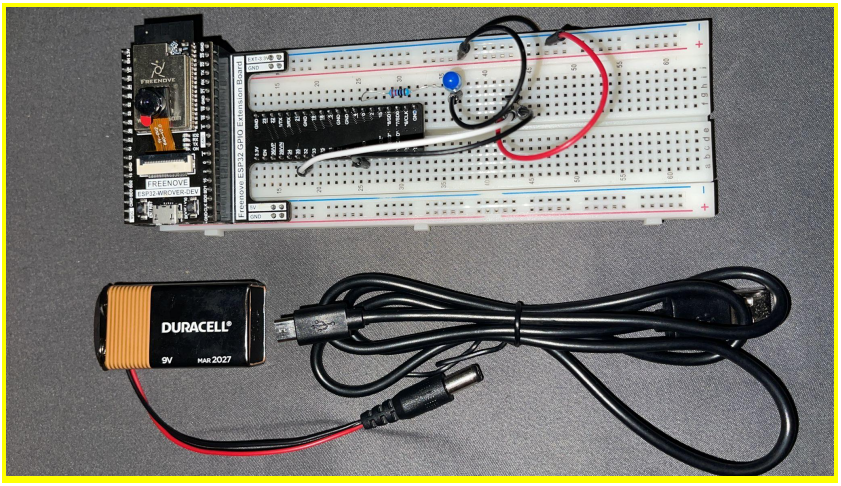

Figure 6 - The hardware used for implementation

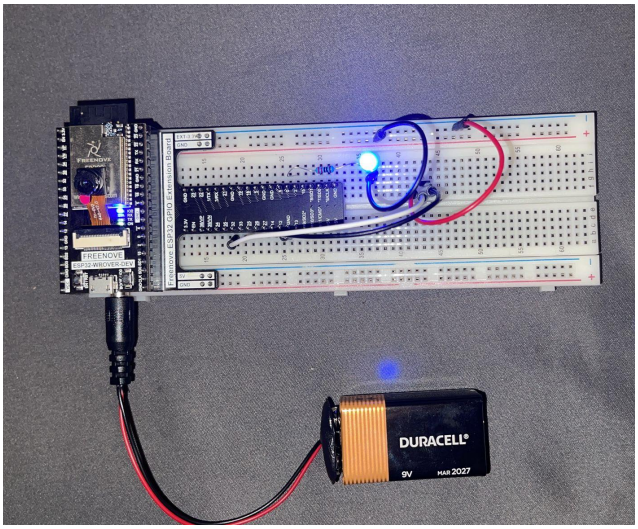

Figure 7 - Implementation of the ESP32 with the temperature sensor, LED and battery

In this hardware we see how the ESP32 is configured with the temperature sensor (LMT84), the LED and a battery. The circuit connections were based on connecting an LED

with a 220 ohm resistance to pin number 2 of the ESP32. We decided to use this pin to drive the LED since the GPIO02 can be configured as an input or output pin. What we want with the LED is to show that everything is working. Also in the circuit we have a temperature sensor (LMT84) which is an analog sensor. This sensor has three pins each of them has its functions. Pin 1 is connected to the 3.3v input. Pin 2 is the output, this is connected to GPIO32 or ESP32. We decided to take this pin since our sensor is an analog one and this pin can be used for digital input/output or analog input. And finally Pin 3 is used as ground to complete the circuit. Finally, we have a 9v battery to power the circuit to demonstrate that the system can operate independently without the need to be connected to the computer via USB.