Electrical and Computer Engineering Department
University of Puerto Rico at Mayagüez

INEL 4206-020 (Microprocessors) Final Project ESP32-Temperature/Siri

By
<u>Team: Iced Out</u>
Alan Mercado
Ryan Melendez
Victor Hernandez

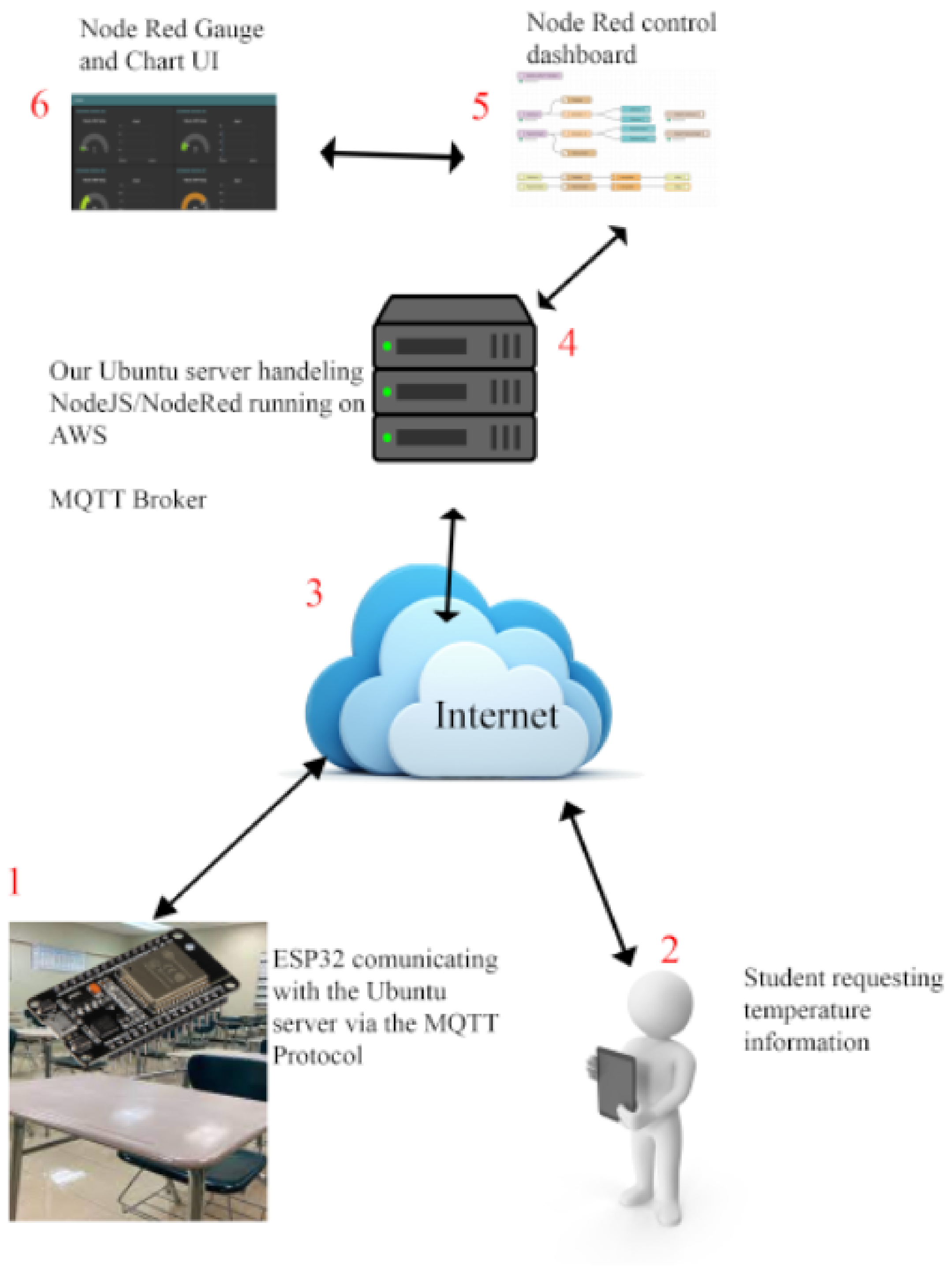
For
<u>Prof. Luis Roa</u>
Electrical and Computer Engineering Department
University of Puerto Rico at Mayagüez
luis.roa1@upr.edu

12/12/22

Table of Contents:

I. Architecture and Design Documentation
II. Project Plan
III. Tool-Set List
IV. ReadMe File

Part I - Architecture and Design Documentation


Node Red Gauge and Chart UI
6
5
Node Red control dashboard
4
Our Ubuntu server handeling NodeJS/NodeRed running on AWS
MQTT Broker
3
Internet
1
ESP32 comunicating with the Ubuntu server via the MQTT Protocol
2
Student requesting temperature information

1. Our ESP32 board with a temperature sensor will be located in the classroom and through the use of the MQTT broker it will communicate with our AWS Ubuntu server running Node Red.The server will constantly request up to date temperature information.

2. With the use of a created shortcut and with Siri voice commands the student is be able to access the the temperature gauges located 3.228.111.161:1880/ui and able to ask siri the current temperature in the room and receive an audible response in the desired units (C or F).

3. The internet is the bridge that connects all of our independent servers, devices and phones and allows them to communicate.

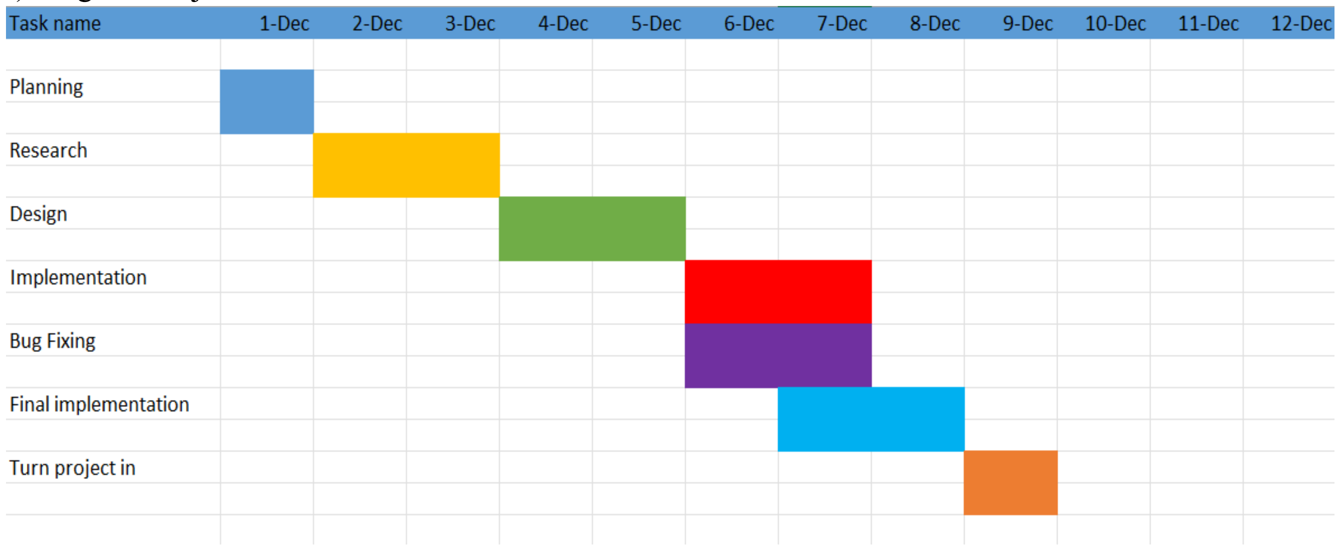
4. This is the Ubuntu server that we have running through AWS. This server is currently running the MQTT broker to communicate with our ESP32 and Node Red which we are using as the backbone to build the system that allows our phone to access the temperature information measured by the ESP32. Within The server we are running an instance of of Node Red accessible through the url: 3.228.111.161:1880 and additionally we have gauges and charts showing a live feed of the temp readouts located at the url: 3.228.111.161:1880/ui

5.Visible in the 5th image is the flow diagram of nodes we created in Node Red which outputs the temperature data in the http protocol so that we can access it through our phone.

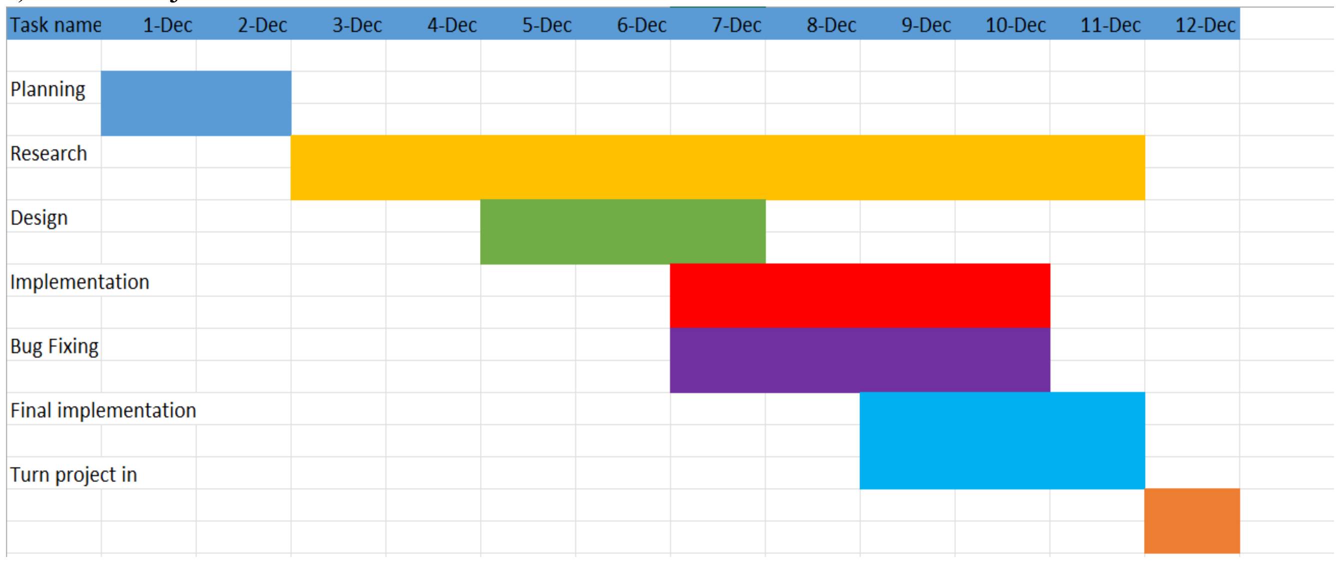
6. This is the Gauge UI generated by our nodes that allows us to live monitor the temperature of the room.

Part II - Project Plan

a) Original Project Plan Gantt Chart



b) Actual Project Plan Gantt Chart



Part III - Tool Set List

1. VSCode
2. MQTT Broker
3. NodeJS
4. Node Red
5. AWS cloud servers
6. Http protocol
7. SmartNoRA (Not used in final release but heavily used during research and development phase)
8. PIO (PlatformIO) ide for ESP32 development
9. Ubuntu
10. ESP32 board
11. iOS/Android device

Part IV - ReadMe File

Intro

The final project for the Microprocessors I - Fall 2022 course, done in the esp32 and Node-RED. The emphasis of this project is on having the user be able ask Siri for the temperature in the room where the ESP32 is. Siri will ask if one wants the temperature reading in Fahrenheit or Celsius. After that temperature is displayed in the user's preferred unit.

Our project uses Amazon's AWS services and a ESP32 that takes the temperature readings .

General Architecture

In our project, the ESP32 microcontroller is connected to a 2-pin thermistor. Our own version of analogRead was then implemented.

The ESP32 is placed in the room where it will be connected to the internet which will connect to a remote cloud computer that has an MQTT Broker protocol. The Broker is used to retrieve data and send data to the RESTl API, which will give the phone access to the data used in Siri.

Tool-Set List:

1. VSCode
2. MQTT Broker
3. NodeJS
4. Node Red
5. AWS cloud servers
6. Http protocol
7. SmartNoRA (Not used in final release but heavily used during research and development phase)
8. PIO (PlatformIO) ide for ESP32 development
9. Ubuntu
10. ESP32 board
11. iOS/Android device

Cloud Architecture

The cloud machine was created using Amazon AWS Services, where MQTT, Node js, Node-red was run on. The MQTT let us communicate with the ESP-32 faster than most other services.

Setting up node-red

For starters, one must use a cloud service like AWS, which allows us to create a machine on the cloud. Then, one must install Node-Red and create a SSH connection with a key to connect to the device of choice.

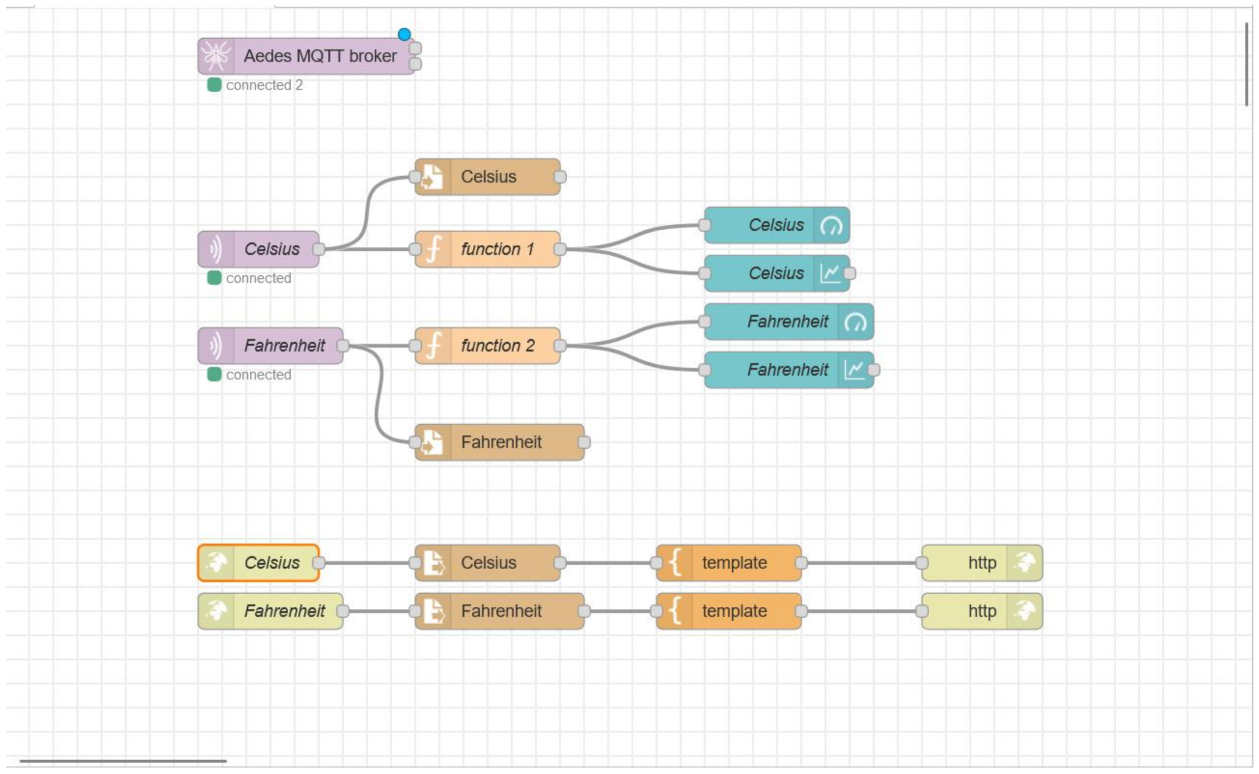
Update server:

```
sudo apt-get update
sudo apt-get upgreade
```

Install nodejs, npm and node-red:

```
sudo apt-get install nodejs
node -v
sudo apt-get install npm
npm -v
sudo npm install -g unsafe-perm node-red node-red-admin
```

We download the dependencies necessary for the connection Siri and other Voice assistants through the node red ui. After that let the automation begin.



Temperature Architecture

The calibration of certain elements of the ESP32, like the thermistor, were a little tricky since manufacturer constants needed to be used to prepare the equations that would run in our main.cpp.

The following equations were formed using manufacturer constants:

```
float adcValue = analogRead(PIN_ANALOG_IN_TEMPERATURE);


float voltage = (float)adcValue / 4095.0 * 3.3;
// read ADC pin and convert to voltage


float Rt = 10 * voltage / (3.3 - voltage);
// calculate resistance value of thermistor


double tempK = 1 / (1 / (273.15 + 25) + log(Rt / 10) / 3950.0);
// calculate temperature (Kelvin)


double tempC = tempK - 273.15;
// calculate temperature (Celsius)


double tempF = tempC * 9 / 5 + 32;
```

By instantiating the variables seen above, we could calculate useful elements of the math behind our project, such as: voltage, resistance, the adcValue, and temperature.

Part VI - ESP32 Code

```
#include <Arduino.h>
#include <wifi.h>
#include <PubSubClient.h>

#define PIN_ANALOG_IN_TEMPERATURE 36
#define PIN_ANALOG_IN_INTERRUPTOR 32

// WIFI
#define WIFI_NAME "Ghost Friend"
#define WIFI_PASSWORD "gsYmqcwR3xhv"
#define WIFI_TIMEOUT 10000

// MQTT BROKER
const char *mqttServer = "3.228.111.161";
const char *celcius = "Temperature/Celcius";
const char *farenheit = "Temperature/Farenheit";
const int mqttPort = 1883;

WiFiClient espClient;
PubSubClient client(espClient);

int calibrationTime = 0;
bool recalibrated = false;
float recalibratedVoltage = 0;
float temperatureQuotient = 0;
float recalibratingVals[5];

float recalibrate(float values[5]);
float highestTemp(float values[5]);
float lowestTemp(float values[5]);
void connectToWifi();

void connectToWifi()
{

  Serial.println("Connecting to WiFi");
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_NAME, WIFI_PASSWORD);

  unsigned long startAttempTime = millis();

  while (WiFi.status() != WL_CONNECTED && millis() - startAttempTime <
WIFI_TIMEOUT)
  {
    Serial.print(".");
    delay(100);
  }

  if (WiFi.status() != WL_CONNECTED)
  {
    Serial.println("Failed to connect to WiFi");
    return;
  }
  else
  {
    Serial.println("Connected to WiFi");
  }
}

void connectToBroker()
{
  client.setServer(mqttServer, mqttPort);

  while (!client.connected())
  {
    Serial.println("Connecting to MQTT...");

    if (client.connect("ESP32Client"))
    {
      Serial.println("connected");
    }
    else
    {
      Serial.print("failed with state ");
      Serial.print(client.state());
      delay(2000);
    }
  }
```

```
}

void setup()
{
  Serial.begin(9600);
  connectToWifi();
  connectToBroker();
}

void loop()
{
  float adcValue = analogRead(PIN_ANALOG_IN_TEMPERATURE);
  float voltage = (float)adcValue / 4095.0 * 3.3;                 // read
ADC pin and convert to voltage
  float Rt = 10 * voltage / (3.3 - voltage);                      //
calculate resistance value of thermistor
  double tempK = 1 / (1 / (273.15 + 25) + log(Rt / 10) / 3950.0); //
calculate temperature (Kelvin)
  double tempC = tempK - 273.15;
// calculate temperature (Celsius)
  double tempF = tempC * 9 / 5 + 32;
  double button = analogRead(PIN_ANALOG_IN_INTERRUPTOR);

  if (temperatureQuotient != 0)
  {
    tempC = tempC * temperatureQuotient;
  }

  if (button == 0 && calibrationTime < 5 && recalibrated == false)
  {
    Serial.println("Recalibrating...");
    recalibratingVals[calibrationTime] = tempC;
    calibrationTime++;
  }

  else if (button != 0 && calibrationTime >= 5)
  {
    temperatureQuotient = recalibrate(recalibratingVals);
    recalibrated = false;
    calibrationTime = 0;

    for (int i = 0; i < 5; i++)
    { // reset array
      recalibratingVals[i] = 0;
    }
    delay(500);
  }
  Serial.printf("Calibration Time: %d, \tTemperature in C: %.2fC,
\tTemperature in F: %.2fF\n", calibrationTime, tempC, tempF);
  delay(1000);

  if (!client.connected())
  {
    connectToBroker();
  }
  client.publish(celcius, String(tempC).c_str());
  client.publish(farenheit, String(tempF).c_str());
  client.loop();
}

float recalibrate(float values[])
{
  float lowestTemperature = lowestTemp(values);
  float highestTemperature = highestTemp(values);
  float averageTemperature = (lowestTemperature + highestTemperature) / 2;

  // Linear interpolation to find the temperature of the thermistor
  float interTemp = lowestTemperature + ((5 / 2) - 1) *
(highestTemperature - lowestTemperature) / (5 - 1);
  float temperatureQuotient = interTemp / averageTemperature;

  return temperatureQuotient;
}

float lowestTemp(float values[])
{
  float lowestTemperature = values[0];
  for (unsigned int i = 0; i < 5; i++)
  {
    if (values[i] < lowestTemperature)
    {
      lowestTemperature = values[i];
    }
  }
  return lowestTemperature;
}

float highestTemp(float values[])
{
  float highestTemperature = values[0];
  for (unsigned int i = 0; i < 5; i++)
  {
    if (values[i] > highestTemperature)
    {
      highestTemperature = values[i];
    }
  }
  return highestTemperature;
}
```