AGH University of Science and Technology

Faculty: Computer Science, Electronics and Telecommunications

Field: Electronics & Telecommunications

Smart Home – Remote Access Control

Students: Gregório da Luz, Michał Kucharz

Index number: 305253, 290480

Date: 02.17.2021

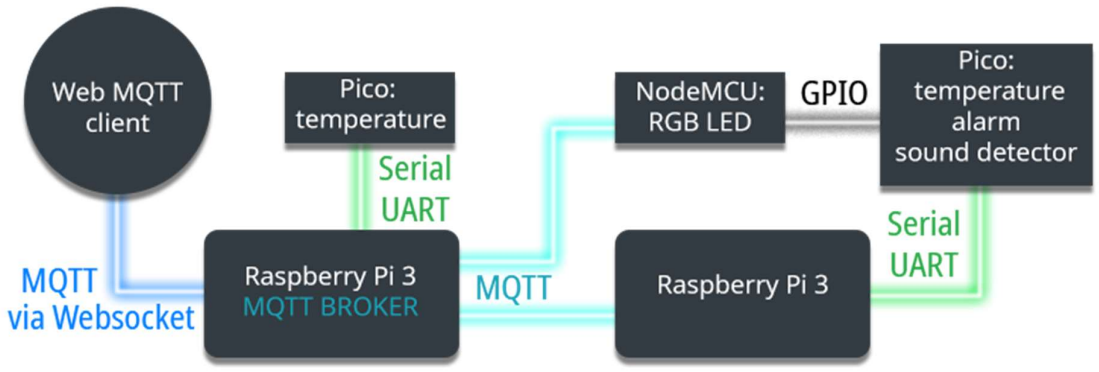
Table of Contents

Introduction

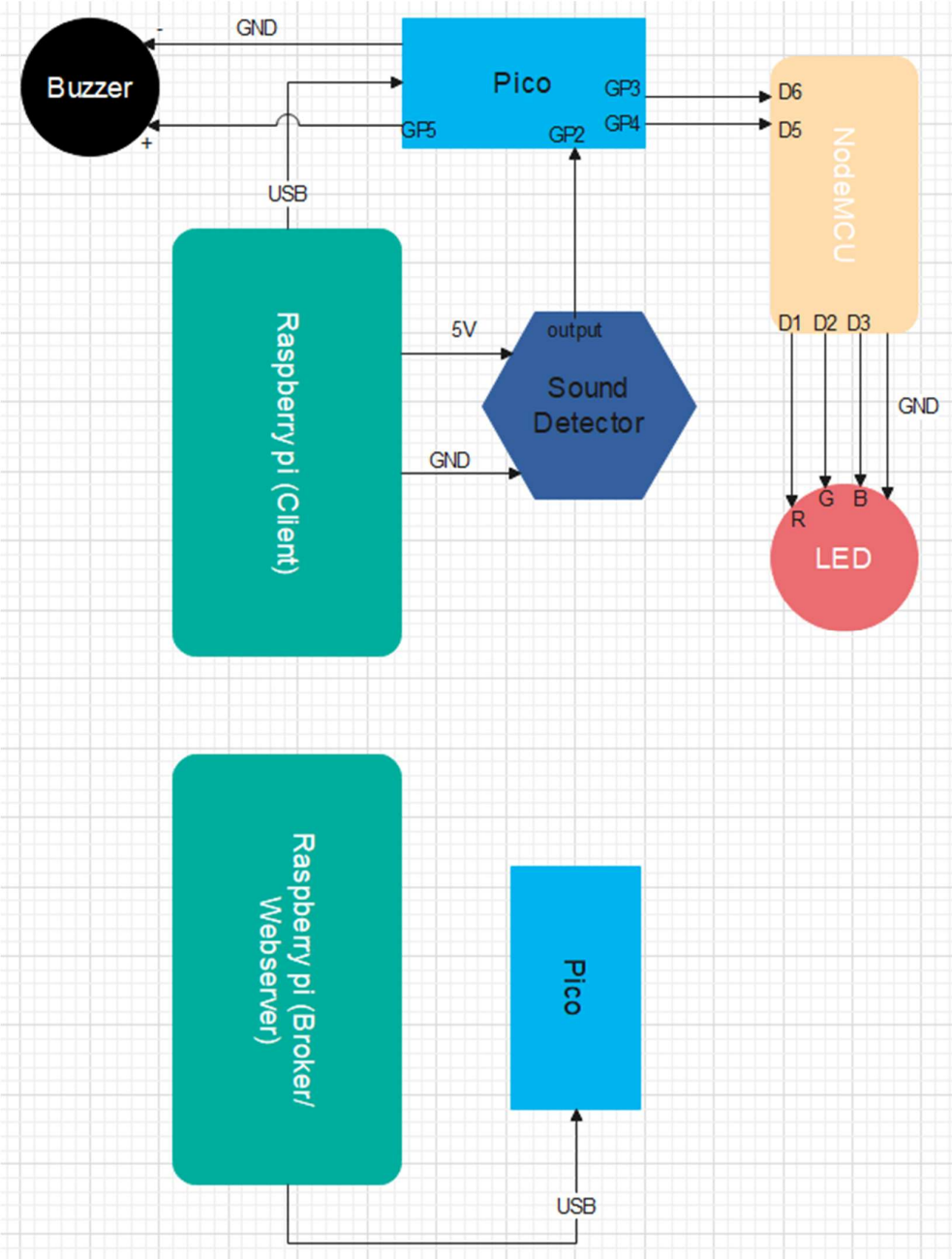
The motivation behind this project was the opportunity to delve deeper into the world of IoT, smarthomes, MQTT and other protocols and skills necessary to succeed in achieving the end goal. In our case, we opted to have our own webserver using a raspberry pi. It gave us access from anywhere in the world to the devices present in our smarthome prototype. The most interesting part was to see my (Gregório) family in Brazil being able to take control of our setup at their fingertips - and instantly. The most important details of this endeavor will be explained throughout the following sections.

Schematics:

Software:



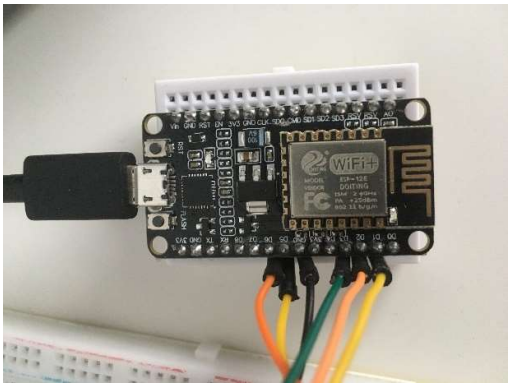
Hardware:



Installation and Setup

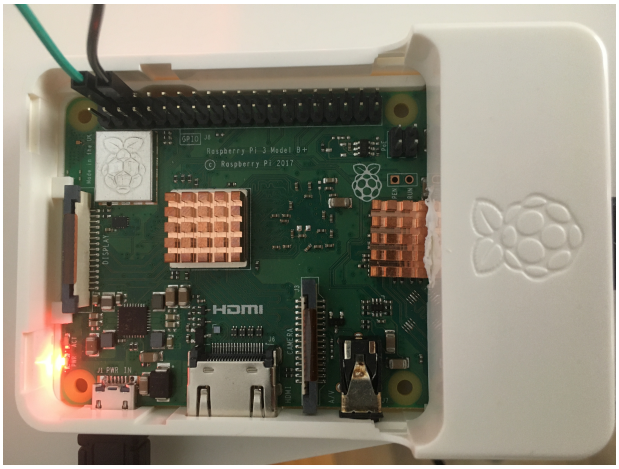
The goal of this section is rather to give a brief explanation of the most important steps towards installing the system and understanding it. We have no intention to get inside every detail of the settings. Doing so would overwhelm this documentation/report.

NodeMCU(ESP-12E):



In regards to NodeMCU board, we opted to program it using LUA language. It's not a very popular language, however the modules available for the firmware are very well documented. This made the final goal of turning the board into a MQTT client much easier.
In the section called ***references***, there will be the address for the website with the documentation for the NodeMCU LUA language modules. There it is also possible to get acquainted with all the procedures from building the firmware, flashing it into the board for the first time and downloading your LUA code into the board.

Raspberry Pi 3B+(Mosquitto Broker):



The procedures of how to install an OS into Raspberry pi won't be explained here, there are many tutorials available on the internet. Jumping further, we can certainly affirm that this is the core of the project. This hardware contains the broker, which is the device where all the messages are published in the topics by the clients and are delivered to all clients that are subscribed to a specific topic.

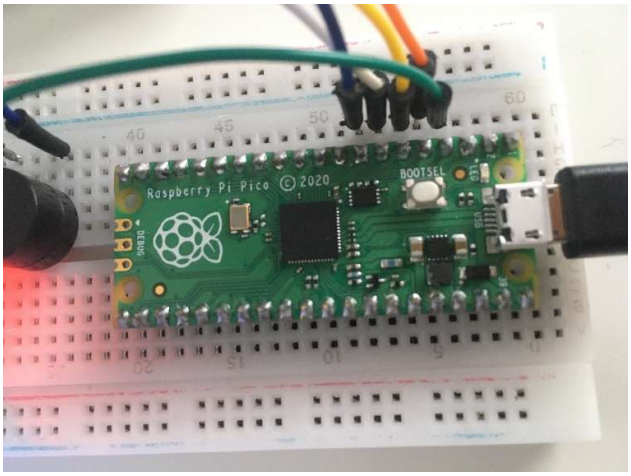
To install the Mosquitto broker, the steps are as follows:

1- Set a static IP address to our Raspberry pi;

2- Install Mosquitto using the following command on terminal:
sudo apt install mosquittto mosquitto-clients
mosquitto-clients is installed to be used for debugging purposes. With its help, we can check if the broker is well configured and react properly to the messages sent to it;

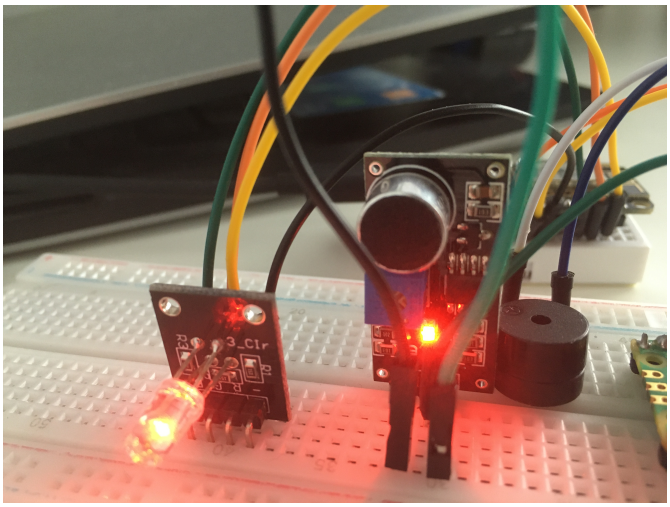
3- To give some security to our broker, we need to configure the *mosquitto.config* file and set a username and a password for clients wanting to connect to the broker. Besides from that, we also want to configure two ports to be used for communication with the broker. One for regular clients (port 1883) and another for the webserver (port 9001).

Raspberry pi pico:



Two Raspberry Pi pico were used for the project. One was programmed using MicroPython and the other one using C. Both of then send the temperature readings from the in chip temperature sensor through UART serial communication. Those temperature readings are then received by a Python script that has a MQTT client and sends the readings to the broker.
The pico programmed using C is also responsible for the sound detection interpretation, alarm setting and, buzzer triggering. All the codes are include in the

GitHub repository. Concerning the pico using C, pooling method and interrupts were used to handle the alarm setting and sound detection.
The alarm is set by a HIGH sent by the NodeMCU. When the pin designated for the alarm is LOW, it means the alarm is off. If the alarm is on and the pico detects an edge rise or fall in the pin connected to the Sound Detector module output, it triggers a HIGH on the pin connected to the + of the Buzzer. This causes the beep/buzz sound to happen. Looking at the Hardware diagram (earlier in the documentation) things become clearer.



Webserver(http://89.79.127.216/) – remote access control

1 – MQTT over Websockets

We opted to have a MQTT client open connection inside the website to make it work as both an http website over internet and an html file locally opened on PC. Because it is not possible to open raw tcp connection in a browser, we have used built-in features present in MQTT libraries and in mosquitto broker to encapsulate MQTT messages by websocket frames.

2 – Apache

To host the website on our raspberry pi 3, we have used apache web server software and forwarded the required ports on the router.

3 – JavaScript MQTT Client

The interface is divided into html file and JavaScript file, which contains our client. Making use of the Eclipse Paho JavaScript Client library and taking advantage of the websocket features it has, we could connect the server to our broker. The html file

contains data fields and buttons. The communication between those and the JavaScript client is handled by Paho library.

4 – JavaScript file in detail

Once the html document finishes loading, we use Paho connect function to have our link between broker and the webserver.

```
$(document).ready(function () {
    MQTTconnect();
});
```

```
let host = '89.79.127.216';
let port = 9001;
let topic = 'smarthome/info/#';
let reconnectTimeout = 3000;
let mqtt;
let previousMsg;

function MQTTconnect() {
    mqtt = new Paho.MQTT.Client(host, port, '/ws', "mqtt_panel" + parseInt(Math.random() * 1000, 10));
    let options = {
        timeout: 3,
        userName: "brazil",
        password: "inpoland2021",
        useSSL: false,
        cleanSession: true,
        onSuccess: onConnect,
        onFailure: function (message) {
            $('#status').html("Connection failed: " + message.errorMessage + "Retrying...")
                .attr('class', 'alert alert-danger');
            setTimeout(MQTTconnect, reconnectTimeout);
        }
    };
    mqtt.onConnectionLost = onConnectionLost;
    mqtt.onMessageArrived = onMessageArrived;
    mqtt.connect(options);
};
```

As seen above, the procedure does not differs from any other client – we are setting up options such as timeout, username, password, and functions to execute on success, failure, connection loss, and new message, i.e. When connection fails, we are using jQuery to modify *alert* in the html file to display appropriate message by referencing to a specific ID of the html element.

```
<div id="status" class="alert alert-dark">
    <small>Loading...</small>
</div>
```

A new client is created every time a webpage is opened, so it is necessary to retain all of the data messages. For that purpose we split our topics into *commands* and *information*:

smarthome/info/#
smarthome/cmd/#

The JavaScript client is only subscribed to ***info***. When the message is captured, the topic levels are split into array and then compared using ***switch*** like so:

```
function onMessageArrived(message) {
    let topic = message.destinationName;
    let payload = message.payloadString;
    let topics = topic.split('/');
    let area = topics[2];

    switch (area) {
        case 'device1':
            if (payload == 'online') {
                $('#label1').text('Online');
                $('#label1').removeClass('badge-danger').addClass('badge-success');
            } else {
                $('#label1').text('Offline');
                $('#label1').removeClass('badge-success').addClass('badge-danger');
            }
            break;
```

Above, once again, jQuery's $() is used to reflect the message data in html file.

As a final remark, let us take a look at the behavior of the buttons:

```
<button type="button" onclick="onButton('beep')" style="padding: 12px 8px"  class="btn btn-outline-danger">RING Alarm</button>
<button type="button" onclick="onButton('on')" style="padding: 12px 8px"  class="btn btn-outline-primary">Alarm: ON</button></
<button type="button" onclick="onButton('off')" style="padding: 12px 8px"  class="btn btn-outline-primary">Alarm: OFF</button>
<button type="button" onclick="onButton('form')" style="padding: 12px 7px"  class="btn btn-outline-info">Submit Form</button><
```

Since the js file is included in the html file, we can directly call onButton function with a specified argument.

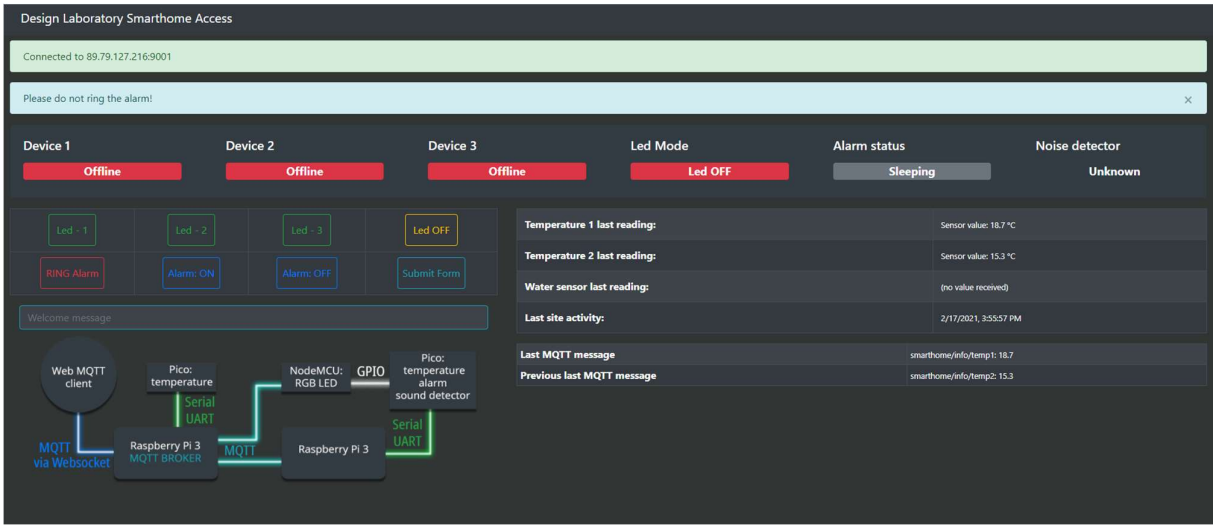
```
function onButton(but) {
    let message
    let date = new Date();
    if (but != 'form'){
      message = new Paho.MQTT.Message(but);
      if (but == "beep" || but == "on" || but == "off")
          message.destinationName = "smarthome/cmd/alarm";
      else
          message.destinationName = "smarthome/cmd/led";
      mqtt.send(message);
    }
```

Functionalities

Throughout the explanations about the project, much of the functionalities of our smart home prototype were already revealed. However, here it is a list with its features:

- Two temperature readings;
- 4 LED modes;
- Alarm (sound detector + buzzer);
- Ringing Buzzer;

A view from our website show how those functionalities were set in our web design:



List of MQTT Topics

command topics

smarthome/cmd/led:
Payloads: “0”,”1”,”2”,”3”;
Subscriber: NodeMCU;
Aim: triggering NodeMCU to change the LED states;

smarthome/cmd/alarm:
Payloads: “beep”, “on”, “off”
Subscriber: NodeMCU
Aim: triggering NodeMCU to set on and off the alarm and ring the buzzer

data topics

smarthome/info/ledstate:
Return message sent by nodemcu on succesful led mode change.

smarthome/info/ledstate/temp1:
smarthome/info/ledstate/temp2:
Temperature values from raspberry pico devices, sent by python clients on raspberry pi 3B+ after pico sends them through UART.

smarthome/info/alarm:
Return message of alarm state
.
smarthome/info/activitiy:
It is exclusively published to and subscribed by the website clients. It updates whenever a button is pressed. It has a timestamp as a payload.

smarthome/info/device1:
smarthome/info/device2:
smarthome/info/device3:
It keeps the “online”/”offline” connection status of devices. Due to all of the info messages being retained, MQTT *last-will* ensures that unexpected closing of connection will send an “offline” payload.

smarthome/info/welcome:
Any string message that is displayed and can be sent on the website interface.

Files Summary

NodeMCU:

- config.lua;
- init.lua;
- init_man.lua;
- main.lua;

Pico in C language:

- alarmtempsmart.c;

Raspberry pi 3B+(Client):

- temperature_mqtt.py;

Pico in MicroPython:

- michal-pico.py;

MQTT LED tests:

- mqtt_commander.py;

Raspberry pi 3B+ (Webserver/Broker/Client):

- index.html;
- index.js;
- michal-pico-client;

Mosquitto:

- mosquitto.conf;

References and Links

Github repository:
https://github.com/gregorio1212/Remote_Access_Smart_Home

NodeMCU docs:
https://nodemcu.readthedocs.io/

Raspberry Pi Pico documentation for C:
https://raspberrypi.github.io/pico-sdk-doxygen/modules.html

Paho – Python – Documentation:
https://pypi.org/project/paho-mqtt/

Steve's Internet Guide:
http://www.steves-internet-guide.com/

Bootstrap:
https://getbootstrap.com/docs/4.0/getting-started/introduction/

Paho – JavaScript Client:
https://www.eclipse.org/paho/index.php?page=clients/js/index.php