In God we trust

School of Computer Engineering

Sharif University of Technology

Embedded Systems

Project Report

Smart Garage Remote

Professor Mohsen Ansari

Presented by: Saee Saadat, Neda Taghizadeh,

Negar Nobakhti, Hamila Mailee

Contents

Table of Figures

Introduction

In this project, we design a framework that enables us to control our garage door with a remote webpage/application using the internet. To this end, we set up a server, and from the user's side, a message containing desired instructions will be sent to this server. The protocol for transferring these messages is MQTT, and NodeMCU is the intermediary that receives the client's messages. A pair of transmitter/receiver modules are needed to interpret the received signals and open/close the door. The state machine below shows an overall view of our project.

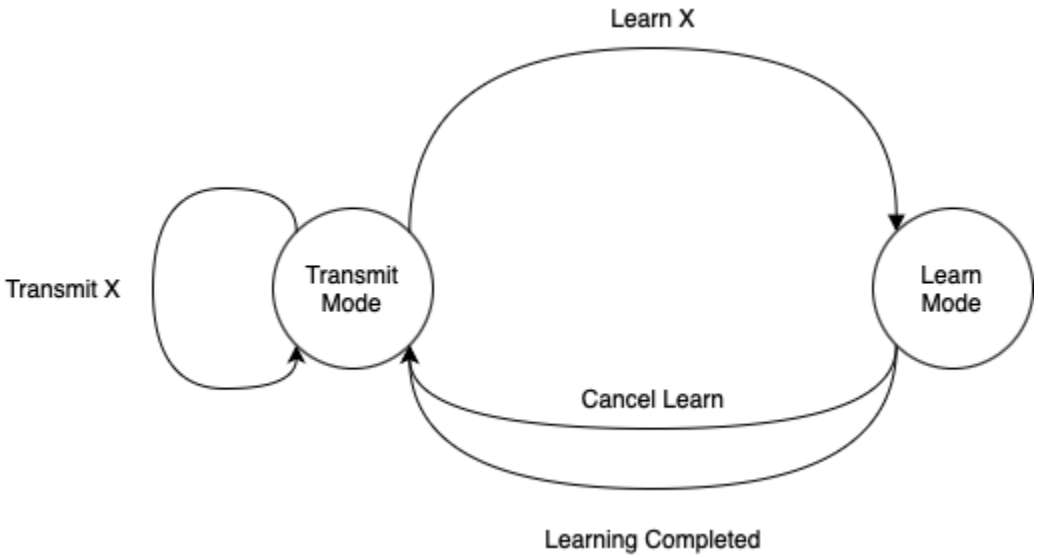

Figure 1 – The state machine representing our project.

Internet Connections

As mentioned in the introduction, the protocol used for connection between server and clients is MQTT. In this section, we first provide a brief description of this protocol. After that, we introduce our server and client and how they connect.

MQTT

MQTT is a TCP-based protocol and the standard in IoT messaging due to its reliability and transfer speed. After setting up the MQTT broker, the clients who are in charge of sending data (in our case, users pressing buttons) will define some "Topics", and the other clients (RF Transmitter) will subscribe to the ones related to them. One advantage of MQTT is that it allows the client to not only receive a message but also send it. In our project, the messages transmitted between different parts are in JSON form, following the below format:

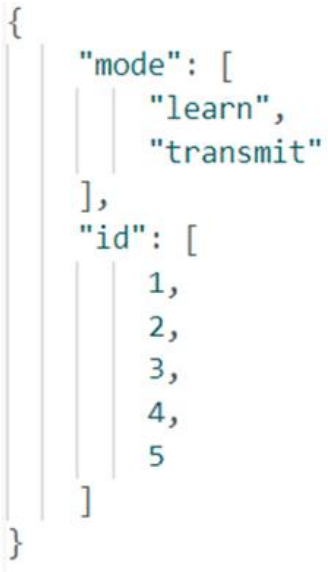

Figure 2 - Messages from client to server: (with the intention of reaching the end devices)

The mode “learn” is for the initiation of the connection, saving the values for opening/closing the door in the memory of our NodeMCU. In the “transmit” mode, we will use the saved values in memory to operate accordingly.

Figure 3 - Messages from devices to server: (reporting the state of devices after receiving a signal)

The mode “error” means that the id had no value, or the learning process was not successful. “Message” mode is equivalent to success and is mostly used for tracking the actions performed by end devices. The “message” field is an extra explanation that distinguishes events, e.g., in the “message” mode, this field can be “Signal for id: 3 has been learned”, or “Signal transmitted successfully”.

Server

For our MQTT broker, we use EMQX.io which provides services in two types of cloud and enterprise. The cloud version is a fully managed MQTT service with a free trial for 14 days, which is suitable for our case.

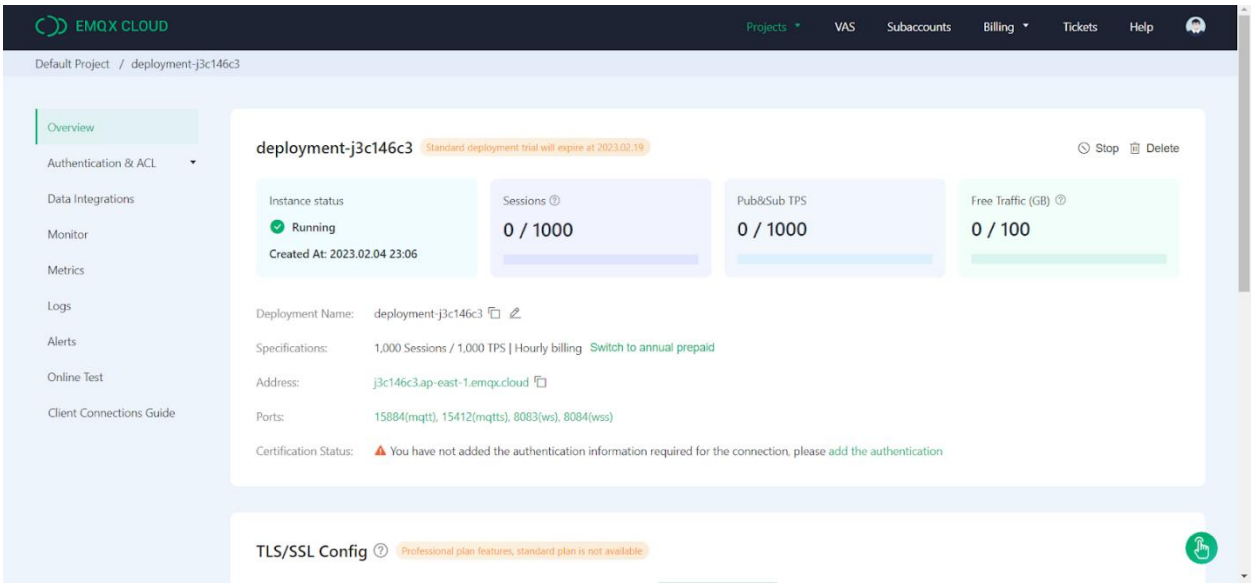

Figure 4 - An overview of the MQTT server provided by EMQX Cloud

From the Authentication section, users can be added by assigning usernames and passwords for each.

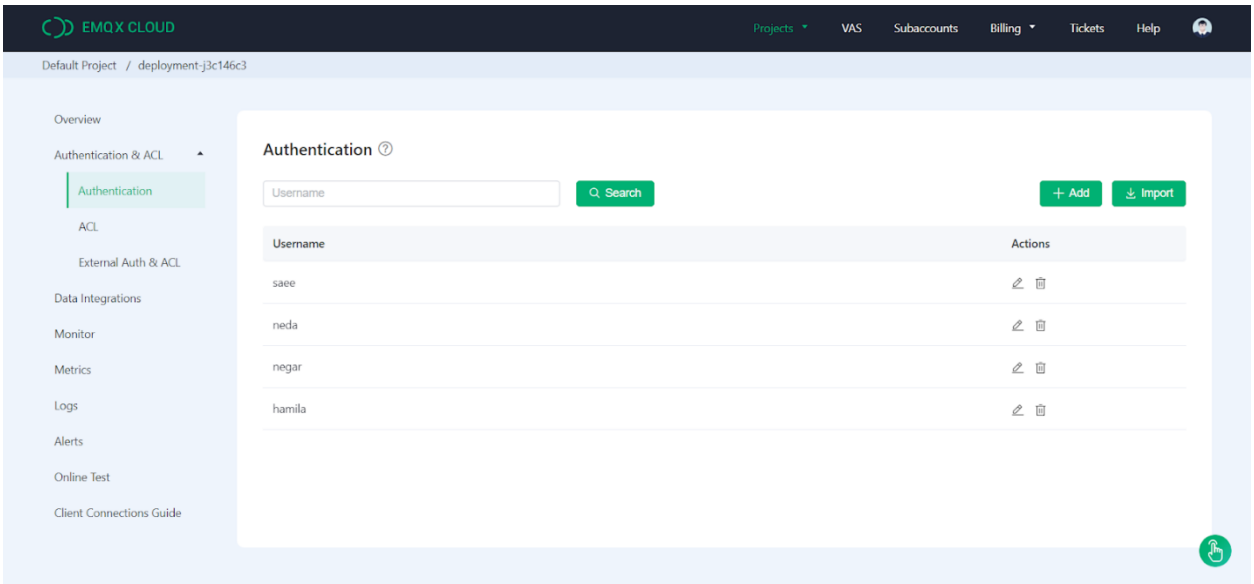

Figure 5 - Authenticated users that can subsribe/publish

Client

left

Hardware Modules

The needed modules for the hardware section are NodeMCU, RF Transmitter, and RF Receiver. A description of each one is provided in this section.

NodeMCU

NodeMCU is an open-source software and hardware development environment, containing the crucial elements of a computer: CPU, RAM, and networking (Wi-Fi). These properties make it an excellent choice for Internet of Things (IoT) projects of all kinds.

In order to check the functionality of our chip, we programmed a simple code to turn an LED on for 1 second and then turn it off in a loop. The code for this program and a picture of its implementation are shown below.

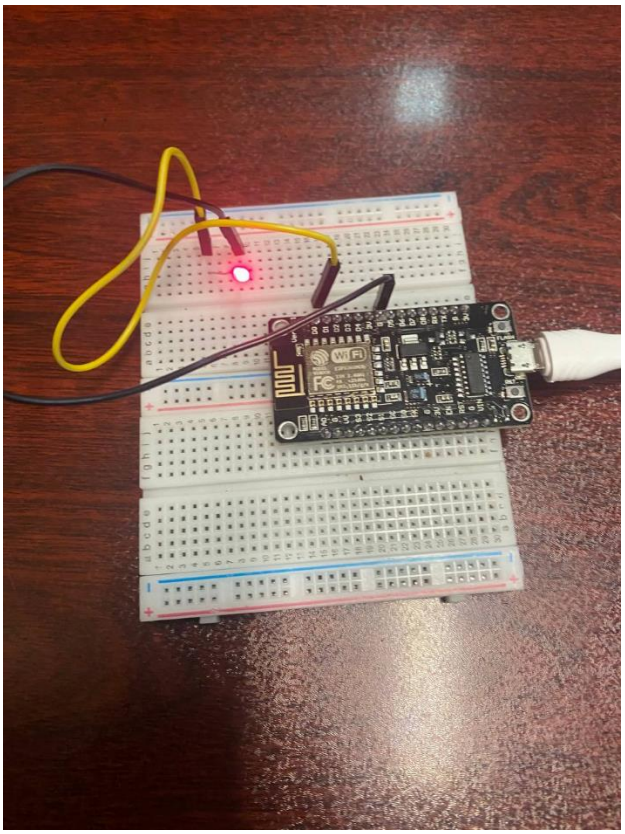

Figure 6 – Using NodeMCU to light the LED periodically.

```
int ledPin = D0;

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(ledPin, OUTPUT);
  Serial.print("SETUP\n");
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(ledPin, HIGH);  // turn the LED on (HIGH is the voltage level)
  Serial.print("HIGH\n");
  delay(1000);                       // wait for a second
  digitalWrite(ledPin, LOW);   // turn the LED off by making the voltage LOW
  Serial.print("LOW\n");
  delay(1000);                       // wait for a second
}
```

Figure 7 – The code used for the above implementation.

The main responsibility of NodeMCU in our project is connecting to Wi-Fi and the MQTT server, and sending respective signals to RF Receiver through the RF Transmitter. Our NodeMCU should be connected to both Receiver and Transmitter, so it should handle both functionalities in its loop function. To do this, we use an RCSwitch that can receive and send data. Whenever a message is received from the MQTT broker, NodeMCU calls the "callback" function, in which the JSON message is deserialized, and the proper message is sent to RF Transmitter.

```
void callback(char *topic, byte *payload, unsigned int length)
{
  Serial.printf("Call back %s %s \n", topic, payload);
  DynamicJsonDocument data(length + 10);
  DeserializationError err = deserializeJson(data, payload);
  if (err)
  {
    Serial.print(("deserializeJson() failed: "));
  }
  if (data["signals"].containsKey("learn"))
  {
    int mode = data["signals"]["mode"];
    learn(mode);
  }
  else if (data["signals"].containsKey("operate"))
  {
    int mode = data["signals"]["mode"];
    operate(mode);
  }
  return;
}
```

Figure 8 – The "callback" function.

RF Module

RF module operates at Radio Frequency. This frequency range varies between 30 kHz & 300 GHz. The RF module we used is a combination of an RF Transmitter and an RF Receiver. The transmitter/receiver (Tx/Rx) pair operate at a frequency of 433 MHz.

Transmitter

Below is a figure of our Transmitter. When NodeMCU calls the function "operate", RF Transmitter has to send the corresponding signal that has been saved in the NodeMCU's memory to the RF Receiver. Since the saved signal had been sent from the client in the first place, there is no need to decode it.

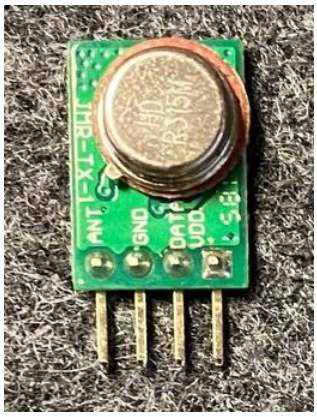

Figure 9 – RF Transmitter.

Receiver

Below shows the RF Receiver used. Since the Receiver has to send a message regarding the result of the operation signal, it has to be connected to our NodeMCU. This message is the "message" mode mentioned in the MQTT section.

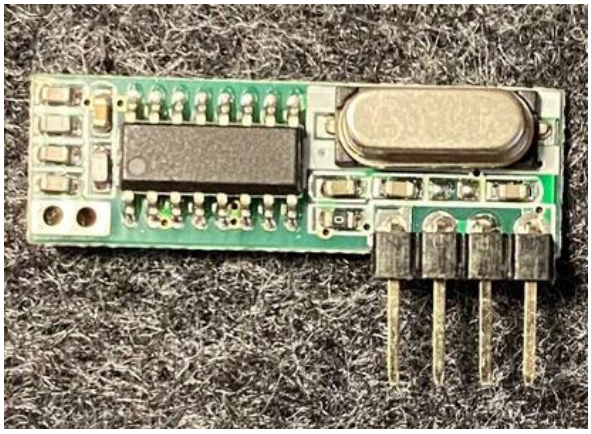

Figure 10 – RF Receiver.

Simulation

To test the functionality of our design with the RF Transmitter/Receiver, we used the modules implemented in Proteus8 with two Arduino Unos, simulating a simple project of turning on an LED connected to the RF Receiver using the signals sent by the RF Transmitter. The successful implementation of this simulation was a stepping stone for the rest of our project.

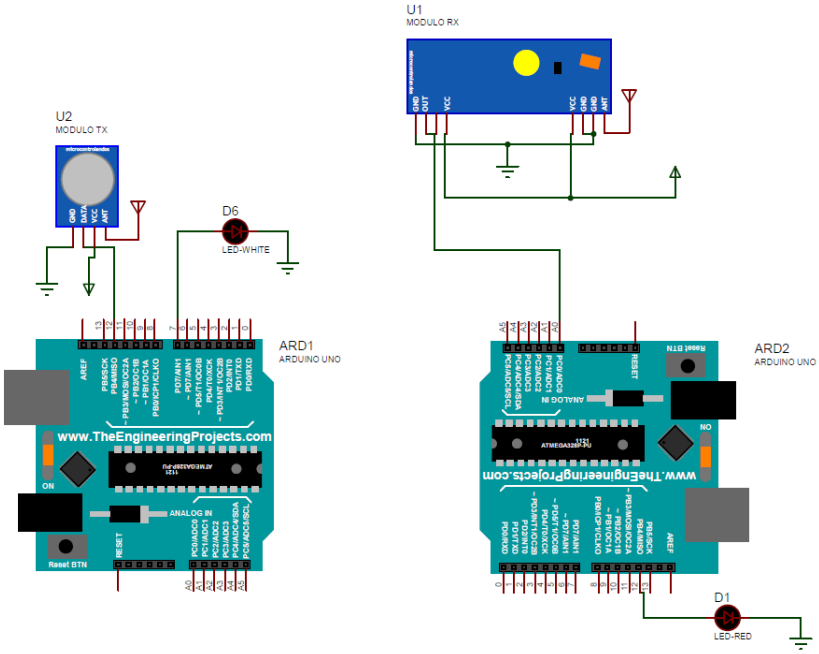

Figure 11 – Simulation in Proteus8, using RF modules and two Arduinos.

Resources

[1] https://mqtt.org/

[2] https://www.emqx.io/

[3] https://www.make-it.ca/nodemcu-details-specifications/

[4] https://robu.in/what-is-rf-transmitter-and-receiver/

[5] https://maker.pro/arduino/projects/how-to-simulate-arduino-projects-using-proteus