Home Automation Project Report

Index:

Foreword

In the beginning of my first year, I attended an International Conference on the LoRaWan network. I had no idea what LoRaWan back then. A company called iBhubs invited students from IITs. The main aim of the conference is Things Industry. Even though I don't know about LoRaWan, After the conference, I understood that LoRaWan is something that replaces WiFi and specifically designed for IoT products.

Please bear with this long discussion. This gonna have both technical details and challenges I encountered during the process.

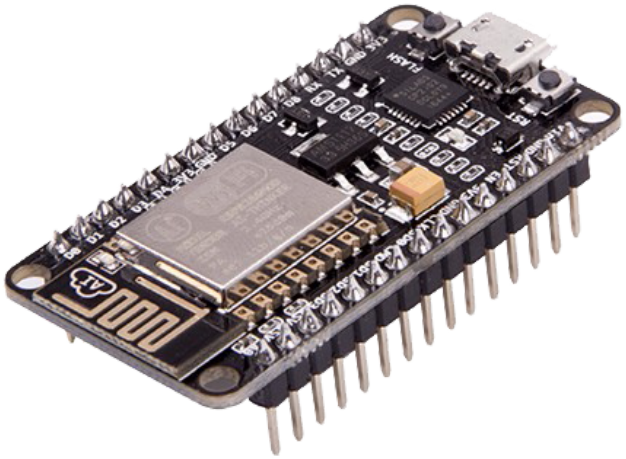
Materials Required

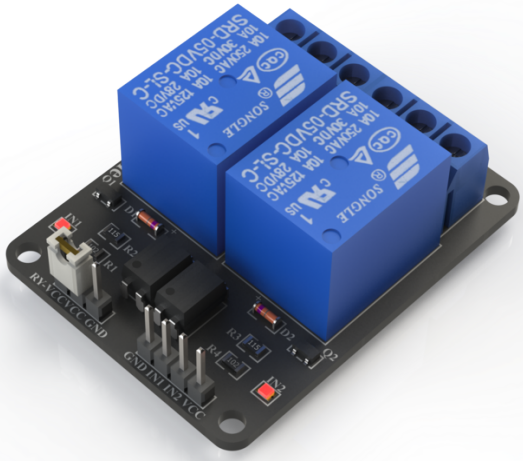
- NodeMCU Board
    - Using this board on Arduino IDE is not straightforward. There are few libraries and board softwares need to be installed before working with this.
- Relay Module
    - It provides an easy way to control high voltage appliances (220V-240V) with low voltage current (3V at max).
    - Includes a Inductor based mechanical switch which toggles based on

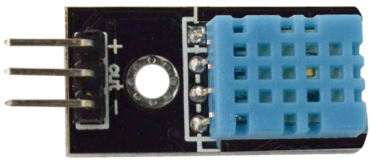
NodeMCU Input

- LDR Sensor Module
    - Light Dependent Resistor. As the name says it all, It responds to Light Intensity. This is used to know the amount of Light Intensity in this project and trigger Security Measures
- DHT11 Sensor
    - Adafruit Industries developed this Sensor and I am using their official library to communicate with this sensor.
    - Measures Temperature based on Resistance of it's electrodes
    - Measures Humidity between it's electrodes
- Electromagnetic Buzzer
    - Simple element, which produces constant high frequency volume
- Breadboard
- Jumper Wires

Images of Above Components







Coming to the project...

Initially, I searched for Arduino projects in Google. Many projects popped up. I saw one project which turns on LEDs from a webpage. But it has a drawback. Devices which are connected to the Arduino network only can control the LED. In other words, It's Intranet. Not the Internet.

Until I saw one Video on YouTube, which really seemed perfect.

As said in PPT, This project is divided into 3 independent parts. I tried to club them up in a single circuit, So, that I can have a single code base to maintain. But Number of Pins is not sufficient to arrange them properly. That's the reason I combined 2 of them in 1 Circuit and 1 of them separately.

1. Controlling Home Appliances over the Internet
2. Home Security System
3. Traffic Lights ProtoType

Prework

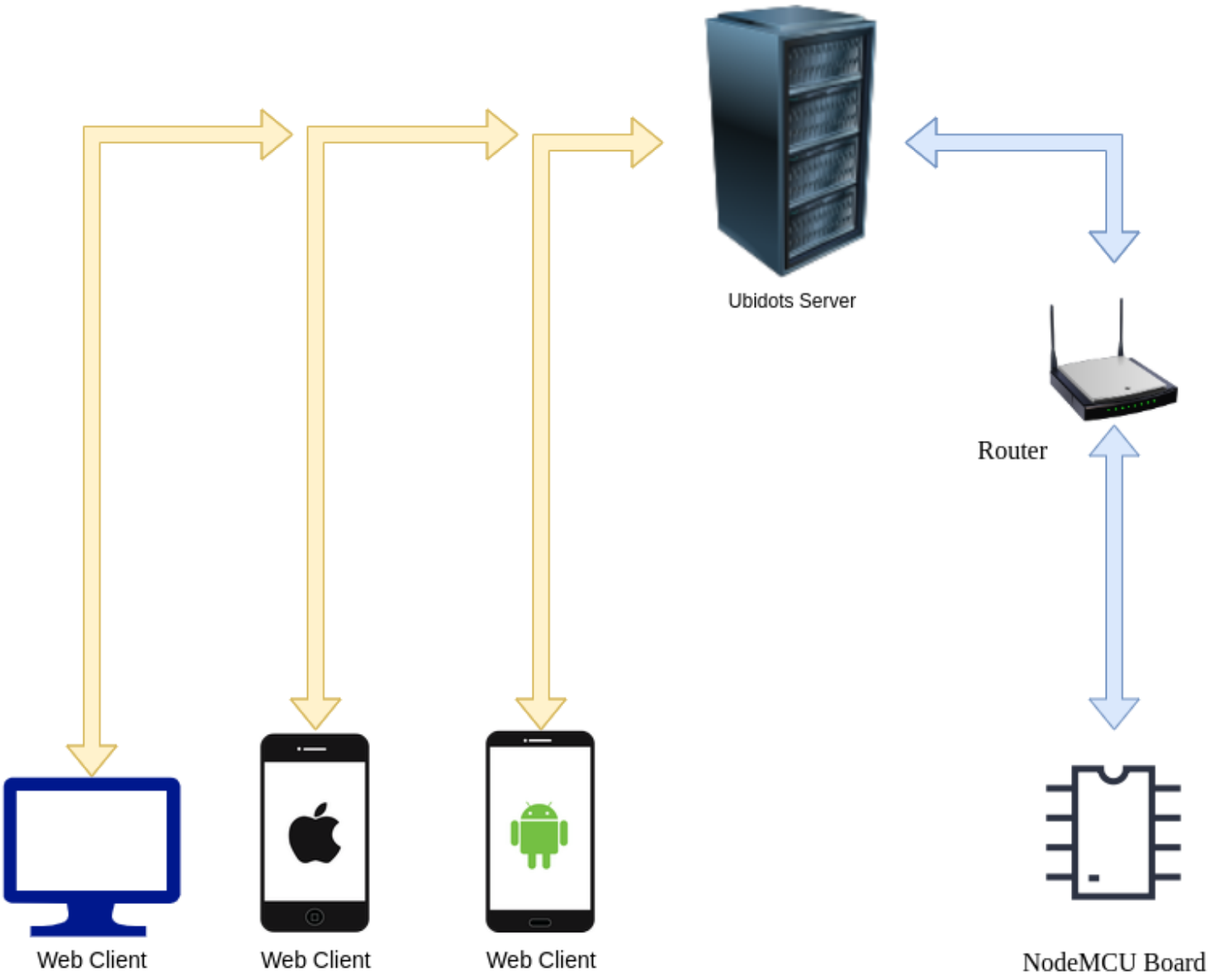
There are few prerequisites, before diving into the code.

I'll start with MQTT, which in turn implies Ubidots used in this project. ***The following flow chart summarizes the idea***. Unlike, Typical HTTP request, response model, MQTT has a 2-way communication strategy.

It follows the Publish-Subscribe Model. It means, client can publish certain messages to the server, And, **It can subscribe to receive some messages when certain messages are published.**

Clients can choose what messages they can subscribe to. Whenever a new message is published to the subscribed topic, Clients receive a message.

It may seem quite abstract now. Things get a little bit clear, when writing code.



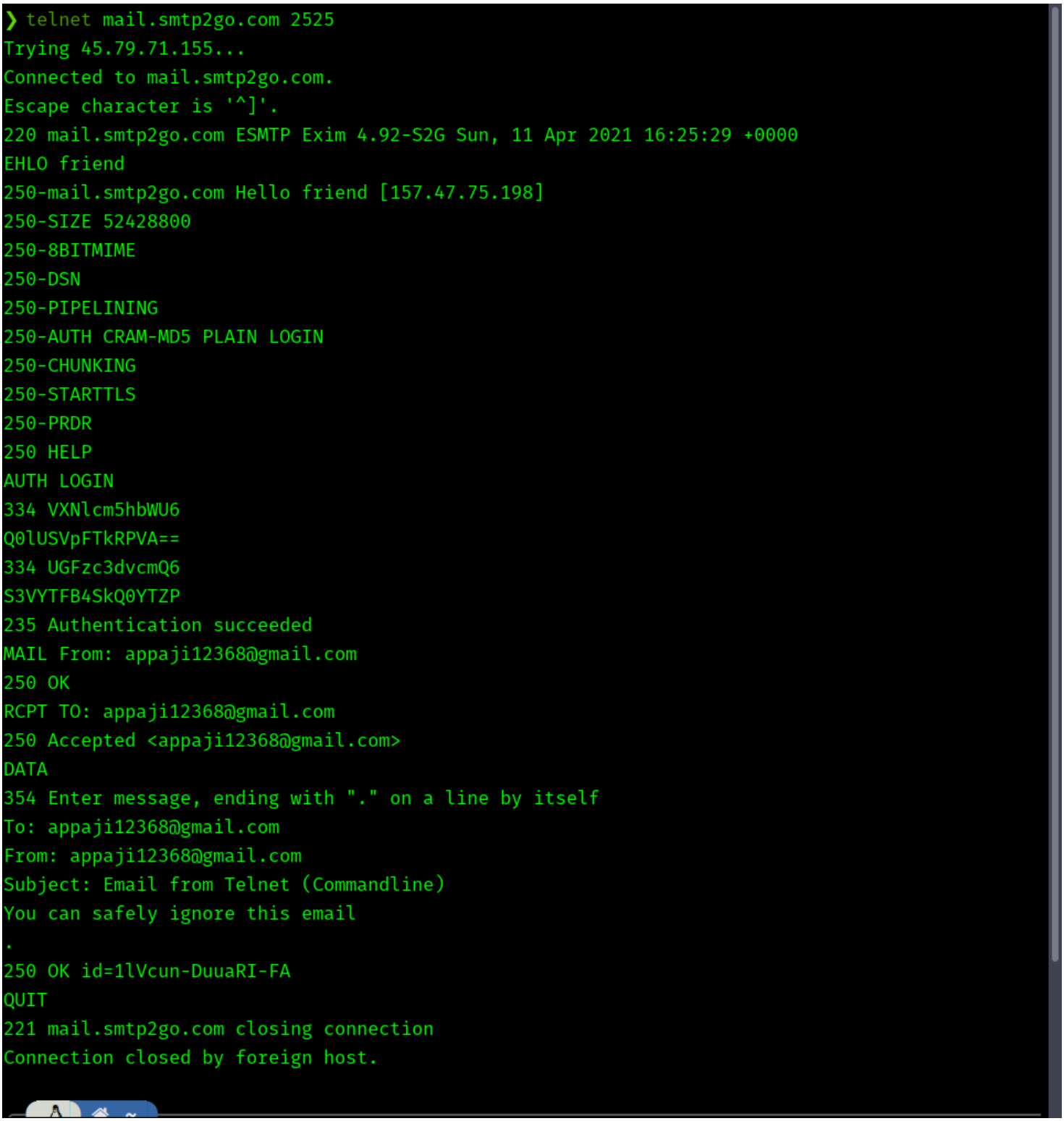
Let’s move on to the Relay Module.
Relay Module is a wonderful device, which enables us to control high voltage devices with low voltage current without messing with high voltage currents. That doesn’t mean it is super safe. Obviously, one has to be careful while handling it.

It has 3 Pins :-

- CO - Commonly Output
- NO - Normally Open
- NC - Normally Closed

Here, We’ll break appliance one of the Power Source wire of the appliance and connect it’s two pieces to CO and NC pins.

Let’s discuss the SMTP mail sending process.
SMTP as any protocol has certain instructions to follow. It has few commands to properly send mail. SMTP commands can perform various tasks, like reversing sender & receiver, spoof email address, encryption of the mail (TLS/SSL) etc... We’re not going to look at all of them though. I wrote the library just to send a mail. It does one job and does it well. UNIX Philosophy😅



I sent a mail using the SMTP service provided by https://www.smtp2go.com/. In the above figure, I am using a telnet client to send mail from Command-line. I am sending those exact commands via ESP8266 WiFi Client in my Library.
Now, I guess going through the code will be a breeze, as most of the concepts are covered above.

Code Overview (Part1 & Part2)

I'll be discussing code structure in this section. This is divided into a few sub-sections.

- ### Initialization [*Refer Fig.*]

Here, I defined all the macros for Pins I am using. Just for convenience. Some Constants & Secrets like WiFi Credentials, Ubidots Token etc... are also defined here. The first library **#include "UbidotsESPMQTT.h"** isn't the official library. It is my fork of it with additional mail sending capability.

Notice ***Ubidots*** and ***DHT*** classes initialized below. ***Ubidots*** class contains several utility functions like, connect, ubidotsSubscribe, publish, reconnect etc... which we'll use in **setup()**.
They help us to focus on the logic instead of handling Byte data from brokers etc...
DHT sensor exposes two class functions **readTemperature** and **readHumidity**. They return float type values of the same.

```
#include "UbidotsESPMQTT.h"
#include <DHT.h>
#include <stdio.h>

/****************************************
   Define Constants
 ****************************************/
#define BUZZER D7
#define LDR A0
#define TEMP D5
#define RELAY D4
#define DHTTYPE DHT11                                   // DHT 11
#define DHTPIN D5

#define WIFISSID "My A"                                 // Put your WifiSSID here
#define PASSWORD "appajichintimi"                       // Put your wifi password
#define TOKEN "BBFF-YSfLdMkT52zPM7GbMCFGank2tJXVWj"     // Ubidots Token
#define VARIABLE_TO_SUBSCRIBE "home_bulb"
#define DEVICE_LABEL "esp8266"


Ubidots client(TOKEN);
DHT dht = DHT(DHTPIN, DHTTYPE);
bool isMailSent = false;
```

- **Auxiliary Functions** [*Refer Fig.*]

Some utility functions which don't directly deal with the main program, are defined here. This term is adopted from Ubidots documentation. Here, A function named **callback** is defined. This function is called whenever Ubidots Client receives a message from Broker. So, Essentially, We have to write the code which decides how to control the Pins based on the message received.

Another thing to mention. We don't call this function in the code anywhere. ***Ubidots*** class takes care of this. We just need to pass this function to the class, And it gets called whenever it's required.

```
/****************************************
   Auxiliary Functions
 ****************************************/
void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  if ((char)payload[0] == '0')
  {
    Serial.println("Since, Output is 0, I'm going to turn off LED");
    digitalWrite(RELAY, LOW);
  }
  else if ((char)payload[0] == '1')
  {
    Serial.println("Since, Output is 1, I'm going to turn on LED");
    digitalWrite(RELAY, HIGH);
  }
  else {
    Serial.print(" Ubidots Value is ");
    Serial.println((char)payload[0]);
  }
}
```

- **Main Functions** [*Refer Fig.*] [*Refer Fig.*]

Finally Here's the main logic. We'll start off with the setup function. **setup()** is called just one time, after the Power is supplied to Arduino.

So, In our case, Below tasks are performed in this setup() function :-

- Setting appropriate pinModes
- Setting Ubidots broker address
- Enabling Debug Messages (Optional, but recommended)
- Attempting Wi-Fi connection with given credentials above
- Subscribing to required variables
- Initializing DHT Sensor

Not much logic is going on here. But this function is vital for upcoming tasks.

```
/**************************************
   Main Functions
 **************************************/

void setup() {
  Serial.begin(115200);

  pinMode(BUZZER, OUTPUT);
  pinMode(RELAY, OUTPUT);
  pinMode(TEMP, INPUT);

  client.ubidotsSetBroker("industrial.api.ubidots.com");
  client.setDebug(true);
  client.wifiConnection(WIFISSID, PASSWORD);
  client.begin(callback);
  client.ubidotsSubscribe(DEVICE_LABEL, VARIABLE_TO_SUBSCRIBE);

  dht.begin();
}
```

loop() contains the heart and soul of this project. As the name suggests, this function runs indefinitely, by default without delay. Users are responsible for issuing delays in this function.

Below tasks are being performed in this function :-

- Conditionally Switching on Buzzer
- Sending Alert Mail conditionally using function provided by my library
- Ensuring that repeated mails are not being sent
- Reconnecting when connection seems to be lost
- Re-subscribing after reconnect
- Decent delay of 100ms

```
void loop() {
  // Controlling Buzzer with LDR Sensor
  if (analogRead(LDR) < 200) {
    // Serial.println("Light is falling on LDR. Buzzer will be turned on.");
    digitalWrite(BUZZER, HIGH);
    if (!isMailSent) {
      String mailSubject = "Hi,\nI am your personal asssistant at Home, monitoring LDR Sensor.\n";
      mailSubject += "It looks like someone interuppted the Signal.\n";
      mailSubject += "Could you check what happened?\n";

      // Reading Temperature and Humidity
      mailSubject += "For your Information, The current Room temperature is " + String(dht.readTemperature()) + "Degree Celsius" + "\n";
      mailSubject += "The current Room Relative Humidity is " + String(dht.readHumidity()) + "\n";
      Serial.print("Starting to Send Mail...");
      isMailSent = client.sendMail("mail.smtp2go.com", 2525, "Q0lUSVpFTkRPVA==", "S3VYTFB4SkQ0YTZP",
                                   "appaji12368@gmail.com", "appaji12368@gmail.com",
                                   "Mail from NodeMCU", mailSubject);
      if (isMailSent) {
        Serial.println("Mail Sent Successfully!");
      }
    }
  }
  else {
    // Serial.println("Light is not falling on LDR");
    digitalWrite(BUZZER, LOW);
    if (isMailSent) {
      isMailSent = false;
    }
  }

  // Controlling Bulb with Website
  if (!client.connected()) {
    client.reconnect();
    client.ubidotsSubscribe(DEVICE_LABEL, VARIABLE_TO_SUBSCRIBE);
  }
  client.loop();
  // Serial.print("\n\n");

  delay(100);
}
```

That concludes Part1 & Part2 Code Overview.

Takeaways

The key takeaway from doing this project is, Building a IoT project is not some Rocket Science and requires a little patience and intuition. I chose this project because it has real life significance. Infact, I use the project I made in my daily life now. Call me lazy but, I prefer to switch off/on the bulb from a website than to go the switch board and turn it off/on.