A PROJECT REPORT

ON

NODEMCU AND MQTT BASED HOME AUTOMATION SYSTEM

Submitted in partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

SUBMITTED BY

SURAJ GUPTA G (1601-14-735-111)

SAI VINEETH P (1601-14-735-107)

VIVEK REDDY K (1601-14-735-119)

Under the Esteemed guidance of

Sri. T. ARAVINDA BABU

Assistant Professor

Dept. of ECE, CBIT, Hyderabad.

Chaitanya Bharathi Institute of Technology (A)

Gandipet, Hyderabad – 500075

2017-2018

CERTIFICATE

Chaitanya Bharathi Institute of Technology
(Regd. No. 855/2009)
(Affiliated to Osmania University, Accredited by NBA (AICTE), Accredited by NAAC (UGC), ISO Certified 9001 : 2008)
Chaitanya Bharathi P.O., CBIT Campus, Gandipet, Kokapet (V),
Rajendranagar Mandal, Ranga Reddy District, Hyderabad - 500 075
e-mail : principal@cbit.ac.in; principalcbit1979@gmail.com; Website : www.cbit.ac.in
✆ : 040 - 24193276; 24193277; 24193280; Fax : 040 - 24193278

This is to certify that the project report entitled **"NodeMCU AND MQTT BASED HOME AUTOMATION SYSTEM"** being submitted by **Mr. SURAJ GUPTA (1601-14-735-111), Mr. SAI VINEETH PENUKULA (1601-14-735-107), Mr. VIVEK REDDY KARNE (1601-14-735-119),** in partial fulfilment of the requirement for the degree Bachelor of Engineering in Electronics and Communication Engineering from **CHAITANYA BHARATHI INSITUTE OF TECHNOLOGY** by the Osmania University, Hyderabad during the academic year 2017-2018. The results embodied in this report have not been submitted to any other university or institution for the award of any degree or diploma.

[illegible]

Project Guide

Sri. T. Aravinda Babu

Assistant Professor

Dept of ECE

CBIT, Hyderabad

[illegible] 6/4/18

Head of the department

Dr. N. V. Koteswara Rao

Professor

Dept of ECE

CBIT, Hyderabad

[illegible] 12/04/2018

CHAPTER 1
INTRODUCTION

1.1 AIM

Our project aims in realizing low cost yet flexible NodeMCU based IoT system that can control the electronic appliances of your home remotely.

1.2 OBJECTIVE

To implement MQTT protocol using NodeMCU board and control relays which in turn control our appliances and then provide access to our appliances over internet using cloud solutions like Firebase and Cloudmqtt. Design an interactive websocket client using jquery and Paho supported by majority of browsers

1.3 MOTIVATION

As automating of the appliances in our home makes the daily life easier to get on with, and because of its possibility of remote access of the appliances we find it highly useful. Advantages of home automation typically fall into a few categories, including savings, safety, convenience, and control. Additionally, some consumers purchase home automation for comfort.

Savings: Smart thermostats and smart light bulbs save energy, cutting utility costs over time. Some home automation technologies monitor water usage, too, helping to prevent exorbitant water bills. Certain devices even offer rebates.

Safety: Many home automation technologies fall under the umbrella of home security. Consumers purchase these devices because they want to make their homes safer and more secure. Automated lighting thwarts would-be burglars, and motion sensors help people enter doors and walk hallways late at night.

Convenience: Because home automation technology performs rote tasks automatically, end users experience great convenience. Lots of smart gadgets are compatible with one another, and you can set different triggers between devices to automate regular home processes. For instance, you could set your smart locks to turn on your smart lighting when you unlock the front door.

Control: Consumers also choose smart home devices to better control functions within the home. With home automation technology, you can know what’s happening inside your home at all times.

Comfort: Some people use smart technology to record shows or to play music throughout the home. Connected devices can also help create a comfortable atmosphere—they provide intelligent and adaptive lighting, sound, and temperature, which can all help create an inviting environment.

Implementing this project will provide us the control of all appliances at our finger tips and the booming automation market wherein everyone wants a smart and automated house will find our project inspiring.

1.4 SCOPE

In recent years, wireless systems like Wi-Fi have become more and more common in home networking. Also, in home and building automation systems, the use of wireless technologies gives several advantages that could not be achieved using a wired network only.

Reduced installation costs: First and foremost, installation costs are significantly reduced since no cabling is necessary. Wired solutions require cabling, where material as well as the professional laying of cables (e.g. into walls) is expensive.

System scalability and easy extension: Deploying a wireless network is especially advantageous when, due to new or changed requirements, extension of the network is necessary. In contrast to wired installations, in which cabling extension is tedious. This makes wireless installations a seminal investment.

Aesthetical benefits: Apart from covering a larger area, this attribute helps to full aesthetical requirements as well. Examples include representative buildings with all-glass architecture and historical buildings where design or conservatory reasons do not allow laying of cables.

Integration of mobile devices: With wireless networks, associating mobile devices such as PDAs and Smartphones with the automation system becomes possible everywhere and at any time, as a device's exact physical location is no longer crucial for a connection (as long as the device is in reach of the network).

For all these reasons, wireless technology is not only an attractive choice in renovation and refurbishment, but also for new installations.

1.5 LITERATURE SURVEY

Konglong Tang, Yong Wang, Hao Liu, Yanxiu Sheng, Xi Wang and Zhiqiang Wei, **"Design and Implementation of Push Notification System Based on the MQTT Protocol",** International Conference on Information Science and Computer Applications (ISCA 2013) This paper described a method of pushing notification system based on the MQTT protocol. It can be used to solve the problem of instant pushing various messages from the server to the mobile client. This method is suitable for developing the smartphone applications. This paper also carried out research and design on the realization of MQTT protocol. With support of the proxy server message broker, the protocol put forward solutions for the server sending the specified messages to the specified mobile devices and the mobile clients receiving pushed messages from the server.

Nidhi Barodawala, Barkha Makwana, Yash Punjabi, Chintan Bhatt, **“Home Automation Using IoT”,** Springer 15th August 2017”. The main agenda of this paper is to enable us to monitor and control physical environment by collecting, processing and analysing the data generated by smart objects, which is an advancement of automation technologies, making our life simpler and easier. The presented paper introduces the home automation system using BASCOM, which also includes the components, flow of communication, implementation and limitations.

Ramón Alcarria, Borja Bordel, Diego Martín, Diego Sanchez De Rivera, **"Rule-based monitoring and coordination of resource consumption in smart communities"**, *Consumer Electronics IEEE Transactions on*, vol. 63, pp. 191-199, 2017, ISSN 0098-3063.This paper proposes a monitoring system based on the provision of monitoring services for the real-time checking of the correct behaviour of sensors and actuators. Monitoring services consist of a set of user-defined rules, which can be registered in a cloud domain and directly executed on a controller device. An implementation of this system and an evaluation of the system's response time in detecting unexpected behaviour of monitored devices is presented.

T. Thaker**,"ESP8266 based implementation of wireless sensor network with Linux based web-server**," 2016 Symposium on Colossal Data Analysis and Networking (CDAN), *Indore*, 2016, pp. 1-5. This paper uses an ARM based Linux (Raspberry Pi) board which acts as the central server of a wireless sensor network, this sensor network is comprised of ESP8266 modules instead of usual ZigBee and Rf modules. IEEE 801.11n protocol has been used in this paper. Various Internet accesses were offered by using Wi-Fi wireless networks communication technology. Raspberry Pi use as a main server in the system and which connects the sensor nodes via Wi-Fi in the wireless sensor network and collects sensors data from different sensors, and supply multi-clients services including
data display through an Embedded Linux based Web-Server.

J. Boman, J.Taylor and A. H. Ngu**, "Flexible IoT middleware for integration of things and applications**,*"* 10th IEEE International Conference on Collaborative Computing: Networking, Applications and Work sharing, Miami, FL, 2014, pp. 481-488. This paper describes the use of IoT middleware. IoT applications must be supported by a middleware that allows IoT consumers and IoT application developers to interact in a user-friendly way, despite the differences in each user's perspective of IoT system. To that end, our software attempts to bridge the gap between IoT consumers and IoT application developers. Through the coupling of GSN (an existing open source IoT middleware), Firebase (a cloud storage service), and an IoT data interpreter developed by us, we have created a software system that takes the first step towards a ubiquitous middleware for IoT.

1.6 ORGANISATION OF THESIS

CHAPTER 2: NodeMCU has been introduced and a brief description about its pin layout has been given.

CHAPTER 3: The various modules used in our project have been described along with their contribution.

CHAPTER 4: The MQTT Protocol has been introduced and few details about its architecture has also been mentioned.

CHAPTER 5: An overview of all the tools and platforms used in our project has been given.

CHAPTER 6: The implementation procedure has been explained along with the code.

CHAPTER 7: The results have been described and verified.

CHAPTER 2

OVERVIEW OF NodeMCU

2.1 INTRODUCTION

An open-source firmware and development kit that helps you to prototype your IOT product within a few Lua script lines. It is based on the eLua project, and built on the ESP8266 SDK 1.4. It uses many open source projects, such as lua-cjson, and spiffs. It includes firmware which runs on the ESP8266 Wi-Fi SoC, and hardware which is based on the ESP-12 module [5].

Features: Open-source, Interactive, Programmable, Low cost, Simple, Smart, WI-FI enabled

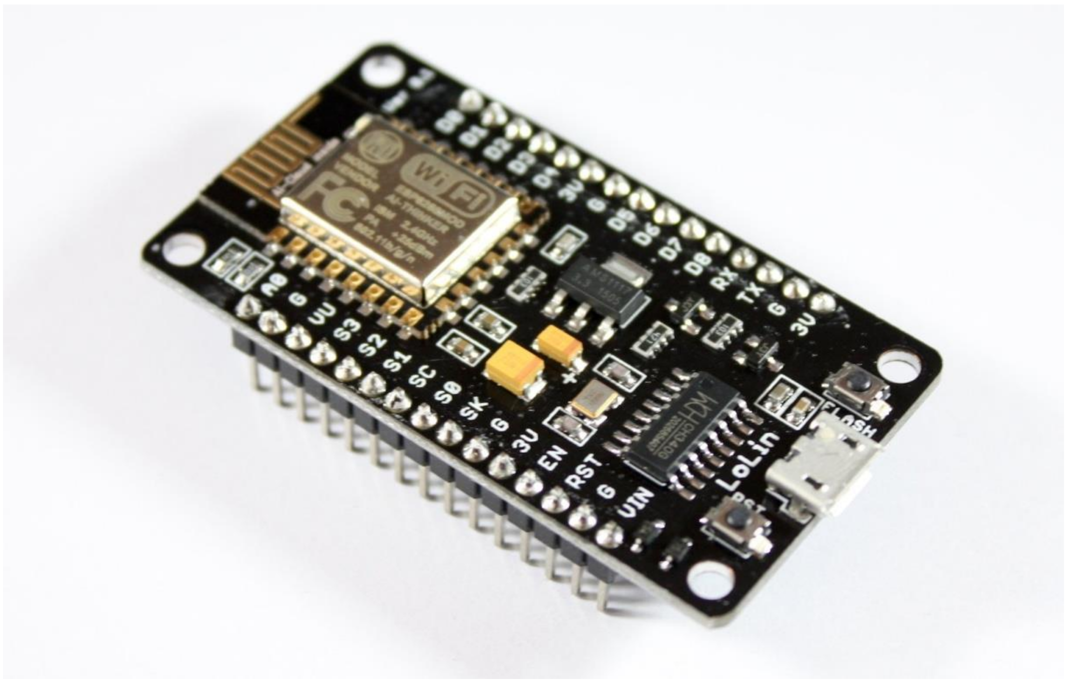


Fig 2.1: NodeMCU

Arduino-like hardware IO: Advanced API for hardware IO, which can dramatically reduce the redundant work for configuring and manipulating hardware. Code like arduino, but interactively in Lua script.

Nodejs style network API: Event-driven API for network applicaitons, which facilitates developers writing code running on a 5mm*5mm sized MCU in Nodejs style. Greatly speed up your IOT application developing process.

Lowest cost WI-FI Less than $2 WI-FI MCU ESP8266 integrated and easy to prototyping development kit. We provide the best platform for IOT application development at the lowest cost.

2.2 SPECIFICATIONS

- Integrated low power 32-bit MCU
- Integrated TCP/IP protocol stack
- Integrated TR switch, balun, LNA, power amplifier and matching network
- 802.11 b/g/n Wi-Fi 2.4 GHz, support WPA/WPA2
- Support STA/AP/STA+AP operation modes
- 10-bit ADC, SDIO 2.0, (H) SPI, UART, I2C, I2S, IR Remote Control, PWM,
- Deep sleep power <10uA, Power down leakage current < 5uA
- Wake up and transmit packets in < 2ms
- Standby power consumption of < 1.0mW (DTIM3)
- +20 dBm output power in 802.11b mode
- Operating temperature range -40C ~ 125C
- FCC, CE, TELEC, Wi-Fi Alliance, and SRRC certified

2.3 PINOUT

NodeMCU has general purpose input output pins on its board as shown in above below diagram. We can make it digital high/low and control things like LED or switch on it. Also, we can generate PWM signal on these GPIO pins.

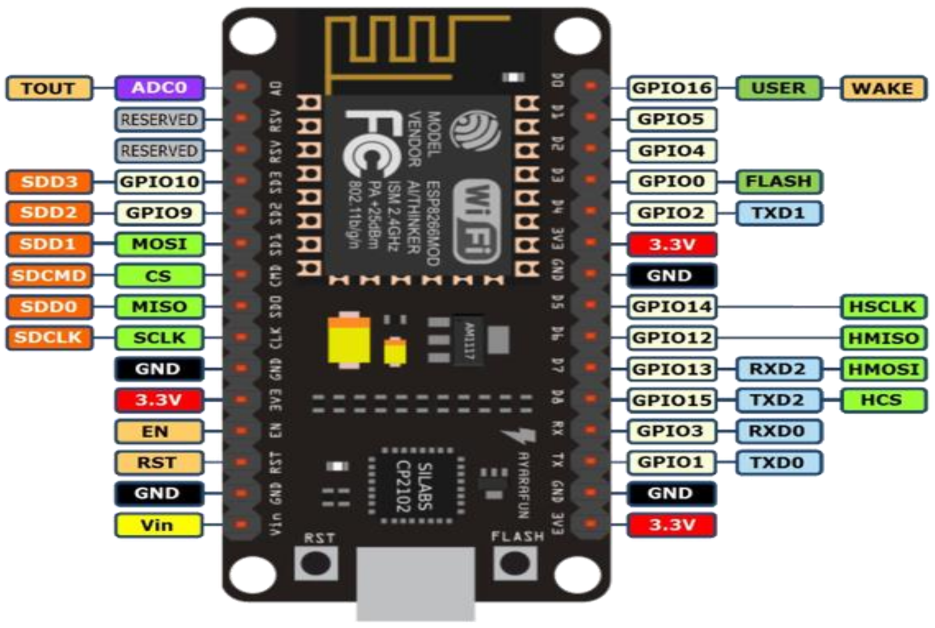
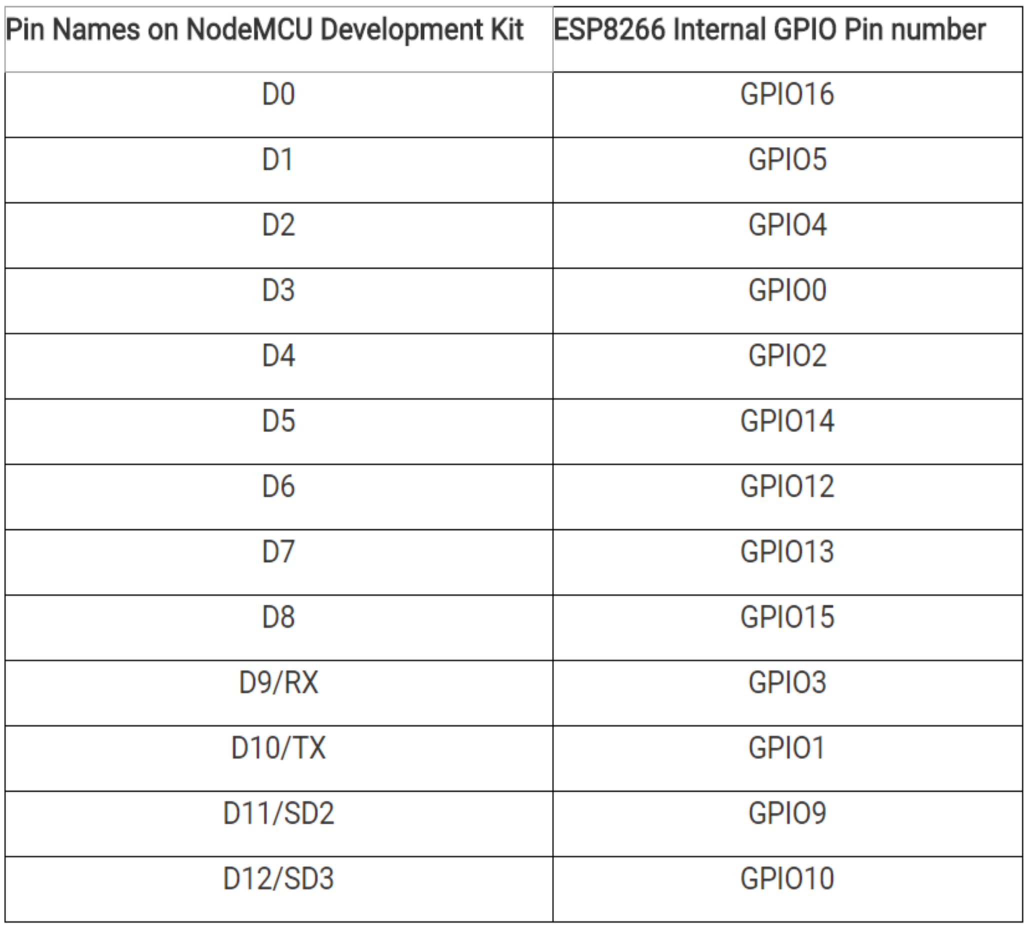


Fig2.2: NodeMCU Pinout

While writing GPIO code on NodeMCU you can't address them with actual GPIO Pin Numbers. There are different I/O Index numbers assigned to each GPIO Pin which is used for GPIO Pin addressing. Refer following table to check I/O Index of NodeMCU GPIO Pins –

Table 2.1: I/O Index of NodeMCU GPIO pins



Pin Names on NodeMCU Development Kit	ESP8266 Internal GPIO Pin number
D0	GPIO16
D1	GPIO5
D2	GPIO4
D3	GPIO0
D4	GPIO2
D5	GPIO14
D6	GPIO12
D7	GPIO13
D8	GPIO15
D9/RX	GPIO3
D10/TX	GPIO1
D11/SD2	GPIO9
D12/SD3	GPIO10

2.3.1 GPIO PINS

NodeMCU has general purpose input output pins on its board as shown in above pinout diagram. We can make it digital high/low and control things like LED or switch on it. Also, we can generate PWM signal on these GPIO pins. General-purpose input/output (GPIO) is a pin on an IC (Integrated Circuit). It can be either input pin or output pin, whose behaviour can be controlled at the run time.

NodeMCU Development kit provides access to these GPIOs of ESP8266. The only thing to take care is that NodeMCU Dev kit pins are numbered differently than internal GPIO notations of ESP8266 as shown in below figure and table. For example, the D0 pin on the NodeMCU Dev kit is mapped to the internal GPIO pin 16 of ESP8266.

The GPIO's shown in blue box (1, 3, 9, 10)(Table 1) are mostly not used for GPIO purpose on Dev Kit. ESP8266 is a System on Chip (SoC) design with components

like the processor chip. The processor has around 16 GPIO lines, some of which are used internally to interface with other components of the SoC, like flash memory.
Since several lines are used internally within the ESP8266 SoC, we have about 11 GPIO pins remaining for GPIO purpose. Now again 2 pins out of 11 are generally reserved for RX and TX in order to communicate with a host PC through which compiled object code is downloaded. Hence finally, this leaves just 9 general purpose I/O pins i.e. D0 to D8.
As shown in figure2 of NodeMCU Dev Kit. We can see RX, TX, SD2, SD3 pins are not mostly used as GPIOs since they are used for other internal process. But we can try with SD3 (D12) pin which mostly like to respond for GPIO/PWM/interrupt like functions [6] .
Note that D0/GPIO16 pin can be only used as GPIO read/write, no special functions are supported on it.
Some of the NodeMCU GPIO functions are as follows:

- pin Mode - Used to set the pin type weather input or output.
  Syntax - pinMode(PIN_NO , INPUT/OUTPUT)
- digitalWrite - Used to write HIGH/LOW value to the GPIO pin.
  Syntax - digitalWrite(PIN_NO , HIGH/LOW)
- digitalRead - Used to read from a GPIO pin, returns 1 if HIGH or else 0 if LOW.
  Syntax - digitalRead(PIN_NO).
- analogWrite - Used to write PWM values to GPIO pins.
  Syntax - analogWrite(PIN_NO, 0-1023).

2.3.2 ADC (Analog to Digital Converter) channel (A0)

NodeMCU has one ADC channel/pin on its board. Analog to Digital Converter(ADC) is used to convert analog signal into digital form. ESP8266 has inbuilt 10-bit ADC with only one ADC channel i.e. it has only one ADC input pin to read analog voltage from external device/sensor.
The ADC channel on ESP8266 is multiplexed with the battery voltage. Hence, we can set it to measure either on board supply voltage or external analog voltage. The input voltage range for ADC pin is 0–3.3V while reading external analog voltage.

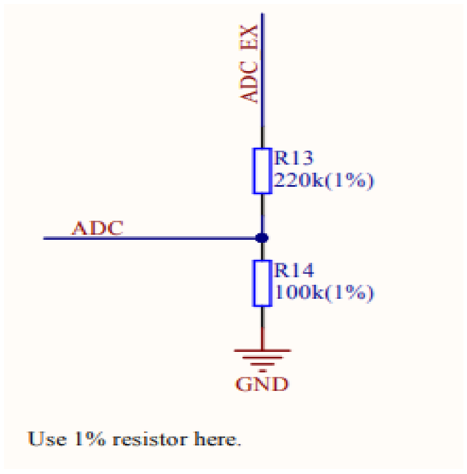


Fig. 2.3: ADC Schematic

The setting for ADC mode i.e. whether system voltage or external voltage is being measured is available in 107th byte of "esp_init_data_default.bin" (0-127 byte) of firmware.

The 107th byte of esp_init_data_default.bin (0 - 127 byte) is "vdd33_const". It must be set to 0xFF i.e. 255 to read system voltage i.e. voltage on VDD pin of ESP8266.

And to read external voltage on ADC pin it must be set to power supply voltage on VDD pin of ESP8266. The working power voltage range of ESP8266 is between 1.8V and 3.6V, and the unit of "vdd33_const" is 0.1V, therefore, the value range of "vdd33_const" is in between 18 to 36.

The NodeMCU has on board register divider network which provide 1.0V from 3.3V to the ADC pin of ESP8266. Hence, we can use 0–3.3V range for ADC input voltage for NodeMCU Dev Kit. Since 10-bit resolution, it will give 0-1023 value range for ADC input voltage 0-3.3V on Dev Kit.

2.3.3 SPI (Serial Peripheral Interface) Pins:

The Serial Peripheral Interface (SPI) is a bus interface connection protocol originally started by Motorola Corp.

- SPI Interface uses four wires for communication. Hence it is also known as four wire serial communication protocol.
- SPI is a full duplex master-slave communication protocol. This means that only a single master and a single slave can communicate on the interface bus at the same time.
- SPI enabled devices work in two basic modes of SPI operation i.e. SPI Master Mode and SPI Slave Mode.

- Master Device is responsible for initiation of communication. Master Device generates Serial Clock for synchronous data transfer. Master Device can handle multiple slave devices on the bus by selecting them one by one.

NodeMCU based ESP8266 has Hardware SPI with four pins available for SPI communication. With this SPI interface, we can connect any SPI enabled device with NodeMCU and make communication possible with it.

ESP8266 has SPI pins (SD1, CMD, SD0, CLK) which are exclusively used for Quad-SPI communication with flash memory on ESP-12E, hence, they can't be used for SPI applications. We can use Hardware SPI interface for user end applications

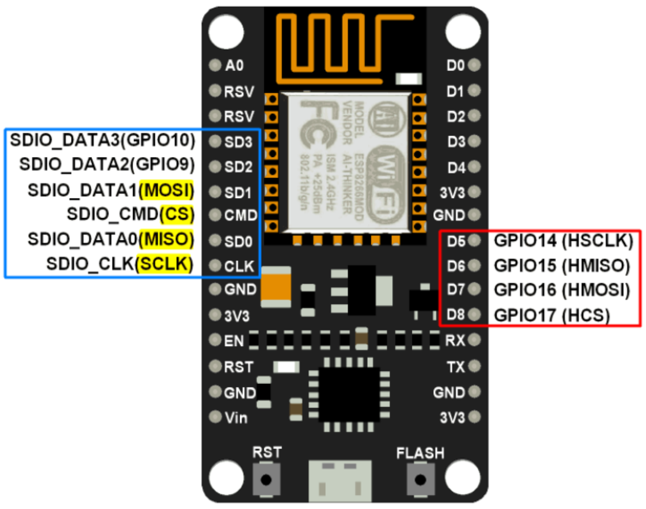


Fig. 2.4: NodeMCU SPI pins

2.3.4 I2C (Inter-Integrated Circuit) Pins:

I2C (Inter Integrated Circuit) is serial bus interface connection protocol. It is also called as TWI (two wire interface) since it uses only two wires for communication. Those two wires are SDA (serial data) and SCL (serial clock).

I2C is acknowledgement-based communication protocol i.e. transmitter waits for acknowledgement from receiver after transmitting data to know whether data is received by receiver successfully.

I2Cworks in two modes namely,

- Master mode
- Slave mode

SDA (serial data) wire is used for data exchange in between master and slave device. SCL (serial clock) is used for synchronous clock in between master and slave device.

Master device initiates communication with slave device. Master device requires slave device address to initiate conversation with slave device. Slave device responds to master device when it is addressed by master device.

NodeMCU has I2C functionality support on its GPIO pins. Due to internal functionality on ESP-12E we cannot use all its GPIOs for I2C functionality. So, do tests before using any GPIO for I2C applications.

UART (Universal Asynchronous Receiver/Transmitter) is a serial communication protocol in which data is transferred serially bit by bit at a time. Asynchronous serial communication is widely used for byte-oriented transmission. In Asynchronous serial communication, a byte of data is transferred at a time.

2.3.5 UART (Universal Asynchronous Receiver Transmitter) Pins:

UART serial communication protocol uses a defined frame structure for their data bytes. Frame structure in Asynchronous communication consists:

- **START bit:** It is a bit with which indicates that serial communication has started and it is always low.
- **Data bits packet**: Data bits can be packets of 5 to 9 bits. Normally we use 8-bit data packet, which is always sent after the START bit.
- **STOP bit**: This usually is one or two bits in length. It is sent after data bits packet to indicate the end of frame. Stop bit is always logic high.

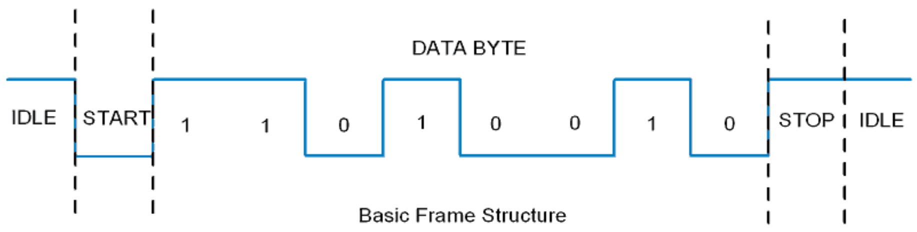


Fig. 2.5: UART Frame Structure

Usually, an asynchronous serial communication frame consists of a START bit (1 bit) followed by a data byte (8 bits) and then a STOP bit (1 bit), which forms a 10-bit frame as shown in the figure above. The frame can also consist of 2 STOP bits instead of a single bit, and there can also be a PARITY bit after the STOP bit.

NodeMCU based ESP8266 has two UART interfaces, UART0 and UART1. The ESP8266 data transfer speed via UART interfaces can reach 40 times of 115200 i.e. 4.5Mbps. By default, UART0 baud rate is 115200 for the oscillator of 40MHz. It can be changed to user defined value according to need of application.

2.4 ESP8266 ARDUINO CORE

NodeMCU is Lua based firmware of ESP8266. Generally, ESPlorer IDE is referred for writing Lua scripts for NodeMCU. It requires to get familiar with ESPlorer IDE and Lua scripting language. But there is another way of developing NodeMCU with a well-known IDE i.e. Arduino IDE. We can also develop NodeMCU applications using Arduino development environment. This makes things easy for Arduino developers than learning new language and IDE for NodeMCU.

Steps Involved:

- First **Download Arduino IDE**https://www.arduino.cc/en/Main/Software
- **Open Arduino IDE** and **Go to File -> Preference**.
- Now on Preference window, **Enter below link in Additional Boards Manager URL** http://arduino.esp8266.com/stable/package_esp8266com_index.json
- Now close Preference window and **go to Tools -> Board -> Boards Manager**
- In Boards Manager window, **Type esp in the search box, esp8266 will be listed there below. Now select latest version of board and click on install.**
- After installation of the board is complete, **open Tools->Board->and select NodeMCU 1.0(ESP-12E Module).**
- **Now Your Arduino IDE is ready for NodeMCU**

This ESP8266 Arduino core framework brings support for ESP8266 chip to the Arduino environment. It lets you write sketches using familiar Arduino functions and libraries, and run them directly on ESP8266, no external microcontroller required.

ESP8266 Arduino core comes with libraries to communicate over WiFi using TCP and UDP, set up HTTP, mDNS, SSDP, and DNS servers, do OTA updates, use a file system in flash memory, work with SD cards, servos, SPI and I2C peripherals.

2.4.1 TIMING AND DELAYS

The functions millis() and micros() return the number of milliseconds and microseconds elapsed after reset, respectively.

Delay(ms) pauses the sketch for a given number of milliseconds and allows WiFi and TCP/IP tasks to run. DelayMicroseconds(us) pauses for a given number of microseconds.

Remember that there is a lot of code that needs to run on the chip besides the sketch when WiFi is connected. WiFi and TCP/IP libraries get a chance to handle any pending events each time the loop() function completes, OR when delay is called. If you have a loop somewhere in your sketch that takes a lot of time (>50ms) without calling delay, you might consider adding a call to delay function to keep the WiFi stack running smoothly.

There is also a yield() function which is equivalent to delay(0).

The delayMicroseconds function, on the other hand, does not yield to other tasks, so using it for delays more than 20 milliseconds is not recommended.

2.4.2 SERIAL OBJECT

Serial object works much the same way as on a regular Arduino. Apart from hardware FIFO (128 bytes for TX and RX) Serial has additional 256-byte TX and RX buffers. Both transmit and receive is interrupt-driven. Write and read functions only block the sketch execution when the respective FIFO/buffers are full/empty. Note that the length of additional 256-bit buffer can be customized.
Serial uses UART0, which is mapped to pins GPIO1 (TX) and GPIO3 (RX).
Serial may be remapped to GPIO15 (TX) and GPIO13 (RX)
by calling Serial.swap() after Serial.begin. Calling swap again maps UART0 back to GPIO1 and GPIO3.
Serial1 uses UART1, TX pin is GPIO2. UART1 cannot be used to receive data because normally it's RX pin is occupied for flash chip connection. To use Serial1, call Serial1.begin(baud rate).
If Serial1 is not used and Serial is not swapped - TX for UART0 can be mapped to GPIO2 instead by calling Serial.set_tx(2) after Serial.begin or directly
With Serial.begin (baud, config, mode, 2).

2.4.3 PROGMEM

PROGMEM is a Arduino AVR feature that has been ported to ESP8266 to ensure compatability with existing Arduino libraries, as well as, saving RAM. On the esp8266 declaring a string such as const char * xyz = "this is a string" will place this string in RAM, not flash. It is possible to place a String into flash, and then load it into RAM when it is needed. On an 8bit AVR this process is very simple. On the 32bit ESP8266 there are conditions that must be met to read back from flash.

PROGMRM the ESP8266 PROGMEM is a macro which places the variable in the .irom.text section in flash. Placing strings in flash requires using any of the methods above.

Syntax:

Constdatatype variableName[] PROGMEM = {data0, data1, data3…};

dataType - any variable type

variableName - the name for your array of data

It is mainly used to store the html code in flash memory instead of storing it in the RAM. It is loaded onto the RAM whenever it is needed.

2.4.4 ESP8266WIFI LIBRARY

The Wi-Fi library for ESP8266 has been developed basing on ESP8266 SDK, using naming convention and overall functionality philosophy of Arduino WiFi library. Over time the wealth Wi-Fi features ported from ESP9266 SDK to esp8266 / Adruino outgrew Arduino WiFi library.

CODE:

```
#include <ESP8266WiFi.h>
voidsetup()
{
Serial.begin(115200);
Serial.println();
WiFi.begin("network-name","pass-to-network");
Serial.print("Connecting");
while(WiFi.status()!=WL_CONNECTED)
{
delay(500);
Serial.print(".");
}
Serial.println();
Serial.print("Connected, IP address: ");
Serial.println(WiFi.localIP());
}
voidloop(){}
```

In the line WiFi.begin("network-name", "pass-to-network") replace network-name and pass-to-networkwith name and password to the Wi-Fi network you like

to connect. Then upload this sketch to NodeMCU module and open serial monitor.The while() loop will keep looping while WiFi.status() is other than WL_CONNECTED.

The loop will exit only if the status changesto WL_CONNECTED.

The return value of function WiFi.status() specifies:

- 0 : WL_IDLE_STATUS when Wi-Fi is in process of changing between status
- 1 : WL_NO_SSID_AVAILin case configured SSID cannot be reached
- 3 : WL_CONNECTED after successful connection is established
- 4 : WL_CONNECT_FAILED if password is incorrect
- 6 : WL_DISCONNECTED if module is not configured in station mode

Devices that connect to Wi-Fi network are called stations (STA). Connection to Wi-Fi is provided by an access point (AP), that acts as a hub for one or more stations. The access point on the other end is connected to a wired network. An access point is usually integrated with a router to provide access from Wi-Fi network to the internet. Each access point is recognized by a SSID (**S**ervice **S**et IDentifier), that essentially is the name of network you select when connecting a device (station) to the Wi-Fi.

ESP8266 module can operate as a station, so we can connect it to the Wi-Fi network. It can also operate as a soft access point (soft-AP), to establish its own Wi-Fi network. Therefore we can connect other stations to such ESP module. ESP8266 is also able to operate both in station and soft access point mode. This provides possibility of building e.g. mesh networks. The ESP8266WiFi library provides wide collection of C++ methods (functions) and properties to configure and operate an ESP8266 module in station and / or soft access point mode.

2.5 CLASSES PRESENT IN ESP8266WIFILIBRARY

Station

Station (STA) mode is used to get ESP module connected to a Wi-Fi network established by an access point. Station class has several features to facilitate management of Wi-Fi connection. In case the connection is lost, ESP8266 will automatically reconnect to the last used access point, once it is again available. The same happens on module reboot. This is possible since ESP is saving credentials to last used access point in flash (non-volatile) memory. Using the saved data ESP will also reconnect if sketch has been changed but code does not alter the Wi-Fi mode or credentials.

Soft Access Point

An access point (AP) is a device that provides access to Wi-Fi network to other devices (stations) and connects them further to a wired network. ESP8266 can provide similar functionality except it does not have interface to a wired network. Such mode of operation is called soft access point (soft-AP). The maximum number of stations connected to the soft-AP is five.

Scan

To connect a mobile phone to a hot spot, you typically open Wi-Fi settings app, list available networks and pick the hot spot you need. Then enter a password (or not) and you are in. You can do the same with ESP. Functionality of scanning for, and listing of available networks in range is implemented by the Scan Class.

Client

The Client class creates clients that can access services provided by servers in order to send, receive and process data.

Client Secure

The Client Secure is an extension of Client Class where connection and data exchange with servers is done using a secure protocol. It supports TLS 1.1. The TLS 1.2 is not supported.

Server

The Server Class creates servers that provide functionality to other programs or devices, called clients.

UDP

The UDP Class enables the User Datagram Protocol (UDP) messages to be sent and received. The UDP uses a simple “fire and forget” transmission model with no guarantee of delivery, ordering, or duplicate protection. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram.

Generic

There are several functions offered by ESP8266’s SDK and not present in Arduino WiFi library. If such function does not fit into one of classes discussed above, it will likely be in Generic Class. Among them is handler to manage Wi-Fi events like connection, disconnection or obtaining an IP, Wi-Fi mode changes, functions to manage module sleep mode, hostname to an IP address resolution, etc.

CHAPTER 3

OVERVIEW OF MODULES

3.1 RELAY

3.1.1 INTRODUCTION

A relay is an electrically operated switch. Many relays use an electromagnet to mechanically operate a switch, but other operating principles are also used, such as solid-state relays. Relays are used where it is necessary to control a circuit by a separate low-power signal, or where several circuits must be controlled by one signal. The first relays were used in long distance telegraph circuits as amplifiers: they repeated the signal coming in from one circuit and re-transmitted it on another circuit. Relays were used extensively in telephone exchanges and early computers to perform logical operations.

A type of relay that can handle the high power required to directly control an electric motor or other loads is called a contactor. Solid-state relayscontrol power circuits with no moving parts, instead using a semiconductor device to perform switching. Relays with calibrated operating characteristics and sometimes multiple operating coils are used to protect electrical circuits from overload or faults; in modern electric power systems these functions are performed by digital instruments still called "protective relays" [7] .

Magnetic latching relays require one pulse of coil power to move their contacts in one direction, and another, redirected pulse to move them back. Repeated pulses from the same input have no effect. Magnetic latching relays are useful in applications where interrupted power should not be able to transition the contacts.

Magnetic latching relays can have either single or dual coils. On a single coil device, the relay will operate in one direction when power is applied with one polarity, and will reset when the polarity is reversed. On a dual coil device, when polarized voltage is applied to the reset coil the contacts will transition. AC controlled magnetic latch relays have single coils that employ steering diodes to differentiate between operate and reset commands.

3.1.2 WORKING

When power flows through the input circuit it activates the electromagnet , generating a magnetic field that attracts a contact and activates the output circuit. When the power is switched off, a spring pulls the contact back up to its original position, switching the second circuit off again. This is an example of a "normally open" (NO) relay: the contacts in the second circuit are not connected by default, and switch on only when a current flows through the magnet.

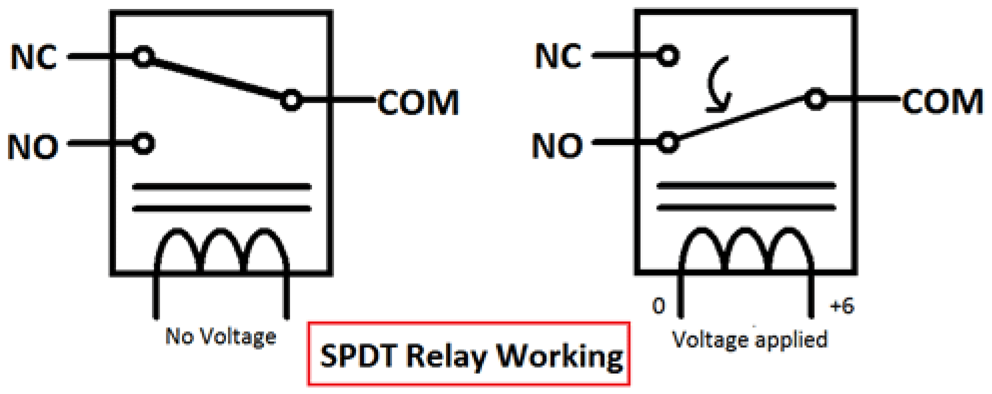


Fig 3.1: SPDT Relay working

Other relays are "normally closed" (NC; the contacts are connected so a current flows through them by default) and switch off only when the magnet is activated, pulling or pushing the contacts apart.

3.1.3 RELAY MODULE

we will use the HL-52S 2 channel relay module, which has 2 relays with rating of 10A @ 250 and 125 V AC and 10A @ 30 and 28 V DC. The high voltage output connector has 3 pins, the middle one is the common pin and as we can see from the markings one of the two other pins is for normally open connection and the other one for normally closed connection [8] .

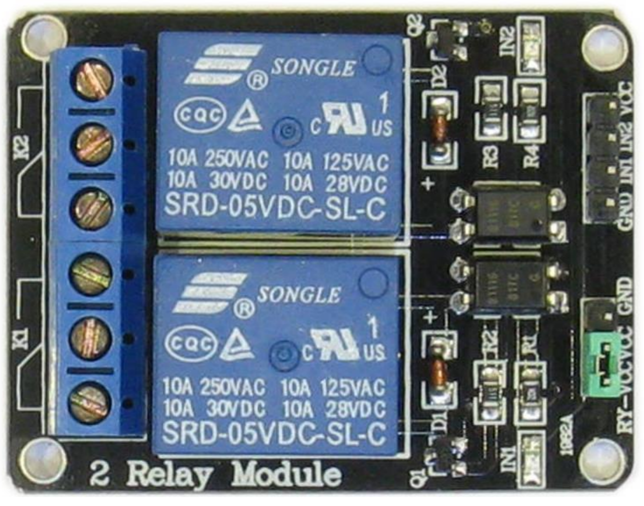


Fig 3.2: Relay module

On the other side of the module we have these 2 sets of pins. The first one has 4 pins, a Ground and a VCC pin for powering the module and 2 input pins In1 and In2. The second set of pins has 3 pins with a jumper between the JDVcc and the Vcc pin. With a configuration like this the electromagnet of the relay is directly powered from the Arduino Board and if something goes wrong with the relay the microcontroller could get damaged.

3.1.4 CIRCUIT SCHEMATIC

For better understanding let's see the circuit schematics of the relay module in this configuration. So we can see that the 5 volts from our microcontroller connected to the Vcc pin for activating the relay through the Optocoupler IC are also connected to the JDVcc pin which powers the electromagnet of the relay. So in this case we got no isolation between the relay and the microcontroller [9] .

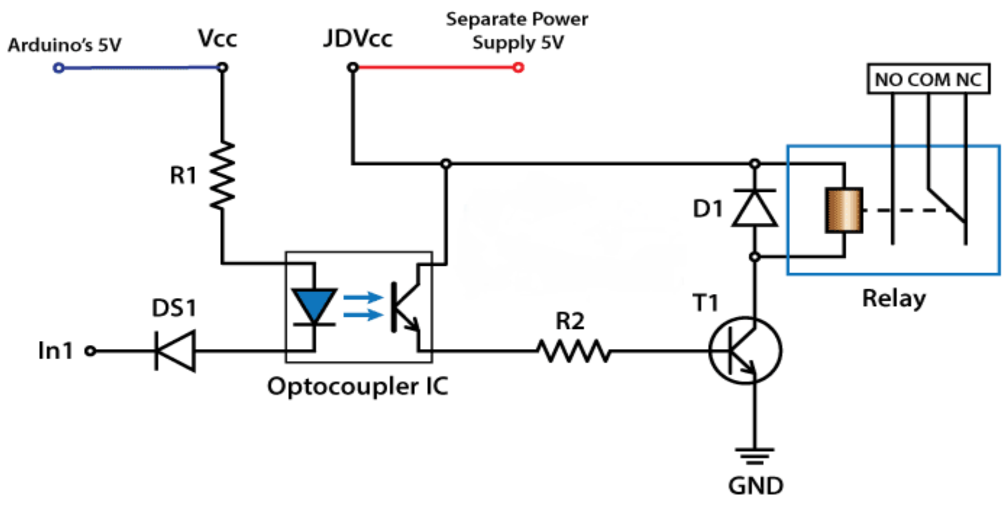


Fig3.3 : Interfacing relay to MCU without Isolation

In order to isolate the microcontroller from the relay, we need to remove the jumper and connect separate power supply for the electromagnet to the JDVcc and the Ground pin. Now with this configuration the microcontroller doesn't have any physical connection with the relay, it just uses the LED light of the Optocoupler IC to activate the relay.

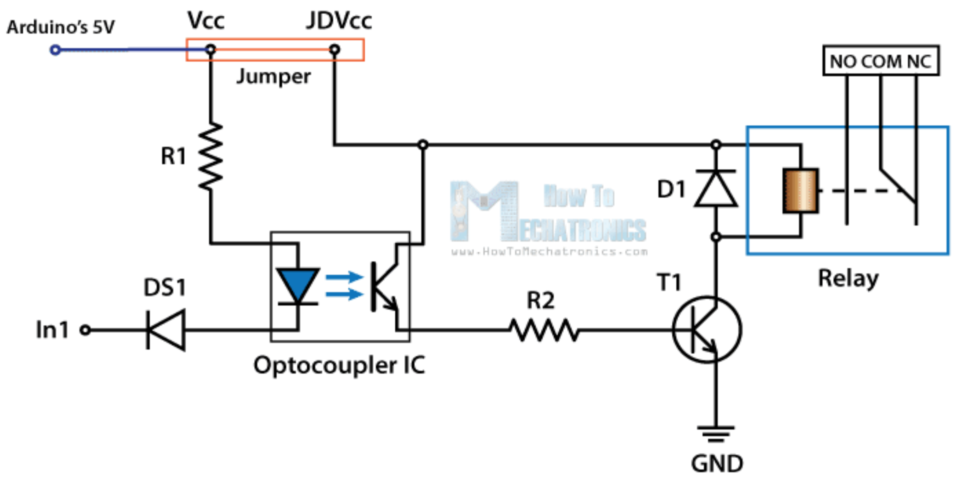


Fig 3.4: Interfacing relay to MCU with isolation

There is one more thing to be noticed from this circuit schematics. The input pins of the module work inversely. As we can see the relay will be activated when the input pin will be LOW because in that way the current will be able to flow from the VCC to the input pin which is low or ground, and the LED will light up and active the relay. When the input pin will be HIGH there will be no current flow, so the LED will not light up and the relay will not be activated.

3.1.5 APPLICATONS

Relays are used wherever it is necessary to control a high power or high voltage circuit with a low power circuit, especially when galvanic isolation is desirable.

The first application of relays was in long telegraph lines, where the weak signal received at an intermediate station could control a contact, regenerating the signal for further transmission. In automobile, a starter relay allows the high current of the cranking motor to be controlled with small wiring and contacts in the ignition key.

Electromechanical switching systems including Strowger and Crossbar telephone exchanges made extensive use of relays in ancillary control circuits.

Because relays are much more resistant than semiconductors to nuclear radiation, they are widely used in safety-critical logic, such as the control panels of radioactive waste-handling machinery. Electromechanical protective relays are used to detect overload and other faults on electrical lines by opening and closing circuit breakers.

3.2 MOTOR DRIVER

Motor drives are circuits used to run a motor. In other words, they are commonly used for motor interfacing. These drive circuits can be easily interfaced with the motor and their selection depends upon the type of motor being used and their ratings (current, voltage) [12] .

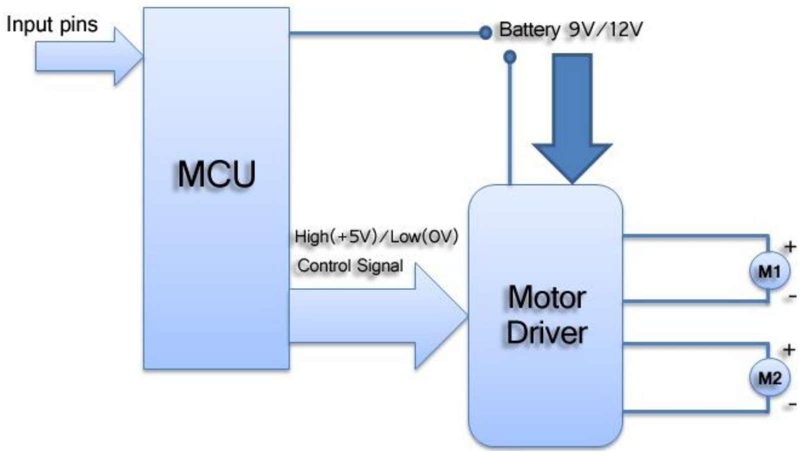


Fig 3.5: Block diagram

3.2.1 MAJOR COMPONENTS IN MOTOR DRIVES

The major motor drive components for DC motors are: a controller, a motor driver IC or a motor driver circuit, the desired DC motor being used, power supply unit and the necessary connections to the motor.

Controller: The controller can be a microprocessor or a microcontroller.

Motor Driver IC: These basically are current amplifiers_which accept the low current signal from the controller and convert it into a high current signal which helps to drive the motor.

Motor: Motor is defined as an electric or mechanic device that can create a motion. While interfacing with the controller; some of the motors like DC motor, stepper motor and brushless dc motor may require a driver IC or driver circuit. DC motor is a type of motor that can convert DC into a mechanical power. In a brushless DC motor, it consists of a DC power source, an inverter producing an AC signal to drive the motor. While stepper motor is a brushless DC electric motor that converts electrical pulses into discrete mechanical motions.

Power Supply Unit: Provides the required power to the motor drive.

3.2.2 NEED FOR MOTOR DRIVER CIRCUITS/ICS

In motor interfacing with controllers, primary requirement for the operation of the controller is low voltage and small amount of current. But the motors require a high voltage and current for its operation. In other words we can say the output of the controller or processor is not enough to drive a motor. In such a case direct interfacing of controllers to the motor is not possible. So we use a Motor Driver Circuit or Motor Driver IC.

Not only in the case of controllers, while connecting motors with 555 timer ICs or 74 series ICs; they also cannot provide the large current required by the motor. If direct connection is given, there might be a chance of damage to the IC.

3.2.3 H Bridge Circuit

H bridge circuit is one of the other commonly used motor driver circuit. In robotic applications, were the DC motor has to run in backward and forward direction; H bridge circuits play a major role. The name H Bridge is used because of the diagrammatic representation of the circuit. Usually the H bridge circuit contains 4 switches S1, S2, S3 and S4. These switches can be relays, or P channel and N channel BJTs, MOSFETs, or they can be N channel MOSFETs only. Here a basic H bridge circuit is shown in the figure below were NPN transistors are placed at the high voltage and PNP transistors to the low voltage.

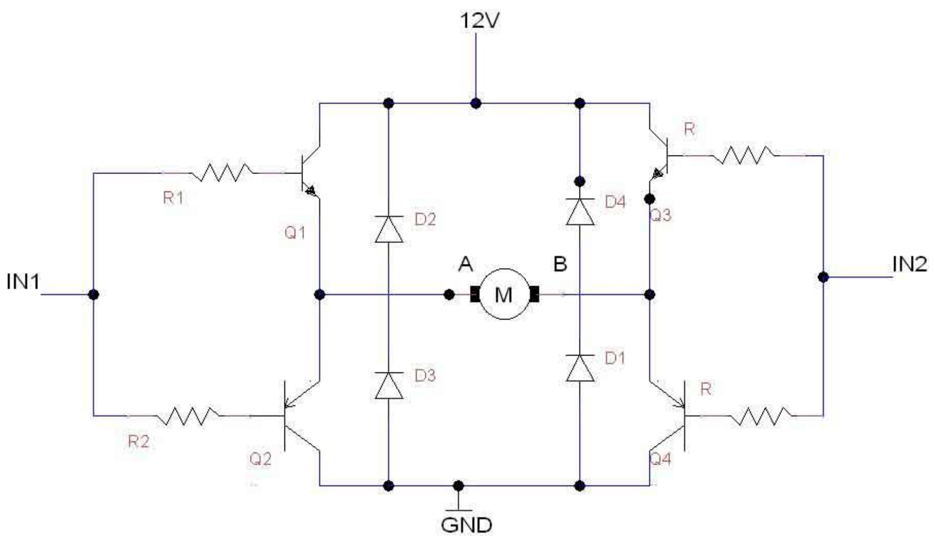


Fig: 3.6: Transistor based H bridge circuit

3.3 L293D

L293D is a dual H bridge motor driver IC. This 16 pin motor driver IC can drive the motors in anti-clockwise and clockwise direction. The connection of the DC motors to L293D IC is given below.

3.3.1 PIN DESCRIPTION OF L293D

- 1 and 9 are Enable Pins.
- 2, 7, 10, 15 are Input pins.
- 3,6,11,14 are output pins
- 4, 5, 12, 13 are the Ground pins.
- 8 and 16 pins are for Vcc.

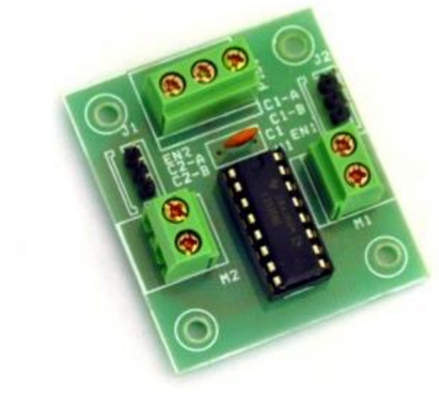


Fig3.7: L293D Motor drive

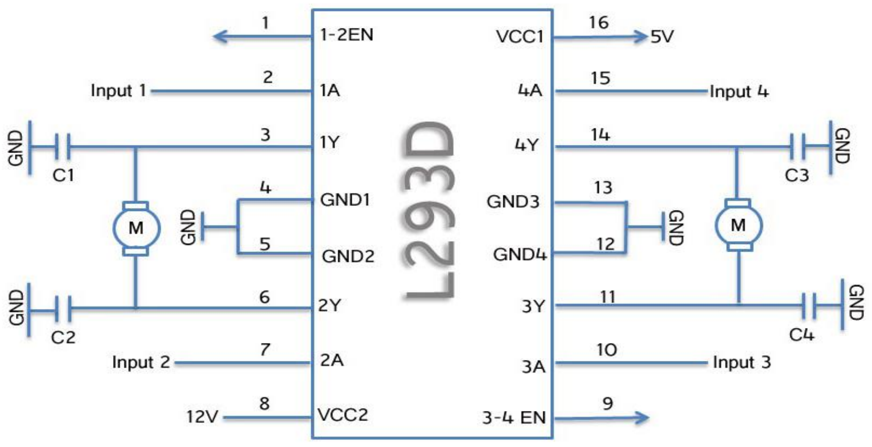


Fig 3.8: L293D Circuit diagram

3.3.2 OPERATION OF L293D CONTROLLING TWO DC MOTORS

Enable pins should be connected to +5v for the motor driver to start its operation. If these pins are connected to GND then the motors will stop its operation. Enable 1, 2 drives the H bridge circuit on the left side while the Enable 3, 4 drives the H Bridge on the right side.

Consider the motor placed at the left (to pin 3 and 6). The operation is described in the table below. Its enable pin is Pin1. Input pins are pin2 and pin7.

Table 3.1: Motor action with various Pin inputs

Pin1	Pin2	Pin7	Action of Motor
+5 volt	0 volt	0 volt	Stop
+5 volt	0 volt	+5 volt	Clockwise
+5 volt	+5 volt	0 volt	Anti-clockwise
+5 volt	+5 volt	+5 volt	Stop

The same operation takes place while controlling the motor placed at the right side (pin11 and 14). Here the enable pin will be pin9; input pins are pin 10 and 15.

3.4 LM2596S

The LM2596 regulator is monolithic integrated circuit ideally suited for easy and convenient design of a step−down switching regulator (buck converter) [4] . It is capable of driving a 3.0 A load with excellent line and load regulation. This device is available in adjustable output version and it is internally compensated to minimize the number of external components to simplify the power supply design.

Since LM2596 converter is a switch−mode power supply, its efficiency is significantly higher in comparison with popular three−terminal linear regulators, especially with higher input voltages.

The LM2596 operates at a switching frequency of 150 kHz thus allowing smaller sized filter components than what would be needed with lower frequency switching regulators. Available in a standard 5−lead TO−220 package with several different lead bend options, and D2PAK surface mount package. he other features include a guaranteed 4% tolerance on output voltage within specified input voltages and output load conditions, and 15% on the oscillator frequency. External shutdown is included, featuring 80 A (typical) standby current. Self-protection features include switch cycle−by−cycle current limit for the output switch, as well as thermal shutdown for complete protection under fault conditions.

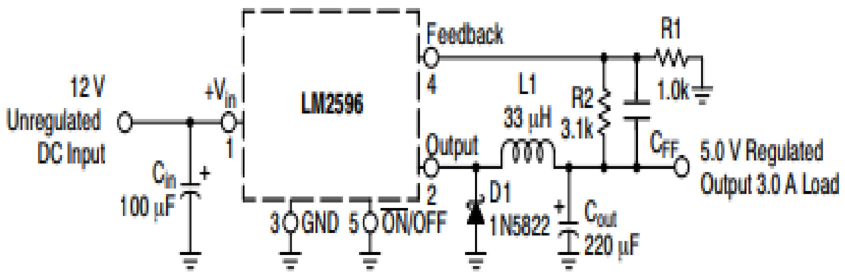


Fig 3.9: Power module Schematic

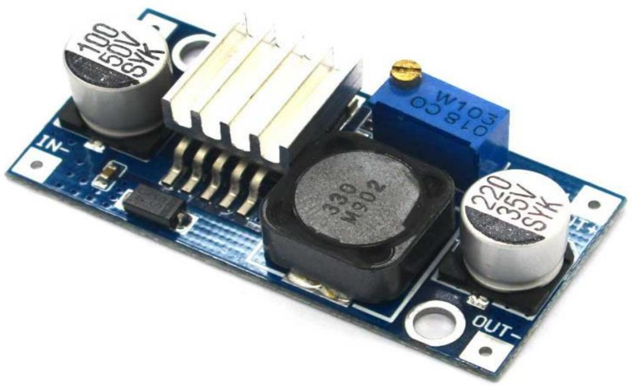


Fig 3.10: Power Module LM2596

3.4.1 FEATURES

• Adjustable Output Voltage Range 1.23 V − 37 V

• Guaranteed 3.0 A Output Load Current

• Wide Input Voltage Range up to 40 V

• 150 kHz Fixed Frequency Internal Oscillator

• TTL Shutdown Capability

• Low Power Standby Mode, type 80 A

• Thermal Shutdown and Current Limit Protection

• Internal Loop Compensation

• Moisture Sensitivity Level (MSL) Equals 1

• Pb−Free Packages are Available

3.4.2 APPLICATIONS

• Simple High−Efficiency Step−Down (Buck) Regulator

• Efficient Pre−Regulator for Linear Regulators

• On−Card Switching Regulators

• Positive to Negative Converter (Buck−Boost)

• Negative Step−Up Converters

• Power Supply for Battery Chargers

CHAPTER 4

MQTT

4.1 INTRODUCTION

Home automation refers to remotely monitoring the conditions of home and performing the required actuation. Through home automation, household devices such as TV, light bulb, fan, etc. are assigned a unique address and are connected through a common home gateway. These can be remotely accessed and controlled from any PC, mobile or laptop. This can drastically reduce energy wastage and improve the living conditions besides enhancing the indoor security. Owing to the rapid growth in technology, the devices in the recent past are becoming smart. The real world devices are being equipped with intelligence and computing ability so that they can configure themselves accordingly. Sensors connected to embedded devices along with the low power wireless connectivity is facilitates to remotely monitor and control the devices. This forms an integral component of Internet of Things(IoT) network. Internet of Things can be considered as a network of devices that are wirelessly connected so that they communicate and organize themselves based on the predefined rules. However these devices are constrained in terms of their resources. Hence light weight protocols such as MQTT, CoAP etc. are used for the data transmission over wireless connectivity. There are so many kinds of radio modules out of which GSM, 3G, WiFi, Bluetooth, Zigbee, etc. are common. However, owing to the surging number of WiFi hotspots and range sufficient to perform the required control and monitoring, WiFi is chosen as the mode of communication in the prototype and the devices are controlled through MQTT protocol implemented using ESP8266.

4.2 MESSAGE QUEUING TELEMETRY TRANSPORT(MQTT)

Message Queuing Telemetry Transport (MQTT) is a light weight transport protocol that efficiently uses the network bandwidth with a 2 byte fixed header [11]. MQTT works on TCP and assures the delivery of messages from node to the server. Being a message oriented information exchange protocol, MQTT is ideally suited for the IoT nodes which have limited capabilities and resources. MQTT was initially developed by IBM in 1999 and recently has been recognized as standard by Organization for the Advancement of Structured Information Standards (OASIS) . MQTT is a publish/subscribe based protocol. Any MQTT connection typically involves two kinds of agents: MQTT clients and MQTT public broker or MQTT server. Data that is being transported by MQTT is referred to as application message. Any device or program that is connected to the network and exchanges application messages through MQTT is called as an MQTT client. MQTT client can be either publisher or subscriber. A publisher publishes application messages and subscriber requests for the application messages. MQTT server is a device or program that interconnects the MQTT clients. It accepts and transmits the application messages among multiple clients connected to it. Devices such as sensors, mobiles etc. are considered as MQTT client. When an MQTT client has certain information to broadcast, it publishes the data to the MQTT broker. MQTT broker is responsible for data collection and organization. The application messages that are published by MQTT client is forwarded to other MQTT clients that subscribe to it. MQTT is designed to simplify the implementation on client by concentrating all the complexities at the broker. Publisher and subscriber are isolated, meaning they need not have to know the existence or application of other.

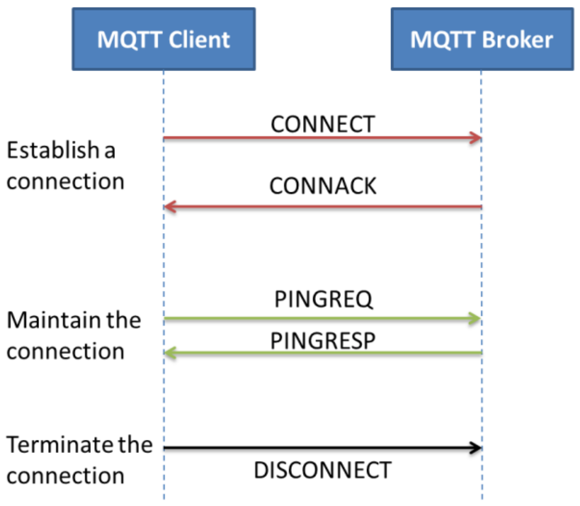


Fig.4.1: Establishing, maintaining and terminating MQTT connection

Before transmitting the application messages, control packets are exchanged based on the QoS associated with them. An MQTT control packet consists of a fixed header, a variable header and payload. CONNECT, CONNACK, PUBLISH, PUBACK, PUBREC, PUBREL, SUBSCRIBE, SUBACK, etc. are some of the MQTT control packets exchanged between MQTT clients and MQTT server. ”Topic” in MQTT provide the routing information. Each topic has a topic name and topic levels associated with it. There may be multiple topic levels separated by / in a topic tree. Wildcard characters such as # and + are used to match multiple levels in a topic. Featuring the queuing system, MQTT server buffers all the messages if client is offline and delivers them to the client when the session is enabled.

A. Establishing a connection

Upon the successful establishment of network between the MQTT client and the MQTT server, control packets are exchanged between the client and the server. The client that wishes to connect to the MQTT server sends a CONNECT packet to the server specifying its identifier, flags, protocol level and other fields. The server acknowledges the client with the specified identifier through CONNACK packet with a return code denoting the status of connection [2].

B. Publishing the application messages

If the client desires to be a publisher, it sends a PUBLISH packet to the server. This packet contains details about the QoS level of transmission, topic name, payload.

C. Subscribing to a topic

If the MQTT client want to subscribe to the application messages published on topic,A topic is a UTF-8 string, which is used by the broker to filter messages for each connected client. A topic consists of one or more topic levels. Each topic level is separated by a forward slash (topic level separator).

In comparison to a message queue a topic is very lightweight. There is no need for a client to create the desired topic before publishing or subscribing to it, because a broker accepts each valid topic without any prior initialization, it sends the SUBSCRIBE packet along with the topic name indicated in UTF-8 encoding. The server acknowledges the subscription with SUBACK packet along with a return code denoting the status of request. Once the subscription is successful, the application messages on the specified topic are forwarded to the client with the maximum QoS. To unsubscribe a topic, the client sends an UNSUBSCRIBE packet to the server which acknowledges it with the UNSUBACK packet.

D. Maintaining the connection alive

After a certain time-out, the connection between the client and the server is terminated. To maintain the connection, the client indicates that it is alive by transmitting a PINGREQ packet to the server. The MQTT server responds to the client with the indicated identifier with a PINGRESP packet and maintains the connection alive.

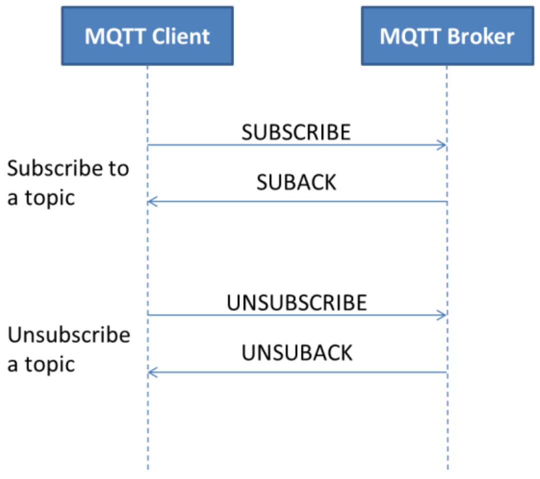


Fig. 4.2: Client subscribing and unsubscribing to the topic

E. **Retained messages**

In MQTT, the messages are retaining in the broker after distributing it to all present clients. At the point when another membership is gotten for the identical subject, then retained messages of those topics are transmitted to the new customer.

F**. Clean sessions and reliable connections**

At the point when a subscriber associates with the broker, clean session association is considered as permanent, if its value is false. In this task, consecutive messages which come out conveying a highest QoS assignment are reserved for delivery when the association is resumed . Use of these flag is optional.

G. **Wills**

A client can inform the broker that it contains a will (message) which should be distributed to a particular topic or topics in case of an unanticipated detach. These will be especially valuable in the system such as security or alarm settings where managers instantly notified just as a sensor has extinct connection with the system.

Terminating the connection

To terminate the connection, the MQTT client sends a DISCONNECT packet to the server. The server does not acknowledge this packet. However all the application messages related to the client will be flushed off and the client is disconnected from the server.

4.3 ESSENTIALS OF MQTT

4.3.1 THE PUBLISH/SUBSCRIBE PATTERN

The publish/subscribe pattern (pub/sub) is an alternative to the traditional client-server model, where a client communicates directly with an endpoint. However, Pub/Sub decouples a client, who is sending a particular message (called publisher) from another client (or more clients), who is receiving the message (called subscriber). This means that the publisher and subscriber don’t know about the existence of one another. There is a third component, called broker, which is known by both the publisher and subscriber, which filters all incoming messages and distributes them accordingly. So let’s dive into a little bit more details about the just mentioned aspects. Remember this is still the basic part about pub/sub in general, we’ll talk about MQTT in coming sections [13].

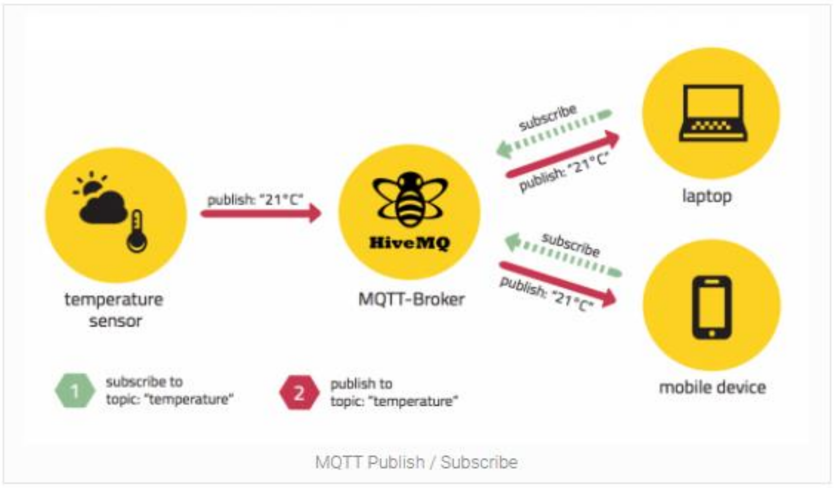


Fig 4.3: Publish / subscribe pattern

4.3.2 MQTT PUBLISH / SUBSCRIBE

As already mentioned the main aspect in pub/sub is the decoupling of publisher and receiver, which can be differentiated in more dimensions:

Space decoupling: Publisher and subscriber do not need to know each other (by ip address and port for example)

Time decoupling: Publisher and subscriber do not need to run at the same time.

Synchronization decoupling: Operations on both components are not halted during publish or receiving

In summary publish/subscribe decouples publisher and receiver of a message, through filtering of the messages it is possible that only certain clients receive certain messages.

The decoupling has three dimensions: Space, Time, Synchronization.

4.3.3 SCALABILITY

Pub/Sub also provides a greater scalability than the traditional client-server approach. This is because operations on the broker can be highly parallelized and processed event-driven. Also often message caching and intelligent routing of messages is decisive for improving the scalability. But it is definitely a challenge to scale publish/subscribe to millions of connections. This can be achieved using clustered broker nodes in order to distribute the load over more individual servers with load balancers. (We will discuss this in detail in a separate post, this would go beyond the scope).

4.3.4 MESSAGE FILTERING

So what’s interesting is, how does the broker filter all messages, so each subscriber only gets the messages it is interested in?

Option 1: Subject-based filtering

The filtering is based on a subject or topic, which is part of each message. The receiving client subscribes on the topics it is interested in with the broker and from there on it gets all message based on the subscribed topics. Topics are in general strings with an hierarchical structure, that allow filtering based on a limited number of expression.

Option 2: Content-based filtering

Content-based filtering is as the name already implies, when the broker filters the message based on a specific content filter-language. Therefore clients subscribe to filter queries of messages they are interested in. A big downside to this is, that the content of the message must be known beforehand and cannot be encrypted or changed easily.

Option 3: Type-based filtering

When using object-oriented languages it is a common practice to filter based on the type/class of the message (event). In this case a subscriber could listen to all messages, which are from type Exception or any subtype of it.

4.3.5 QUALITY OF SERVICE

The Quality of Service (QoS) level is an agreement between sender and receiver of a message regarding the guarantees of delivering a message.

There are 3 QoS levels in MQTT:

At most once (0)

At least once (1)

Exactly once (2).

When talking about QoS there are always two different parts of delivering a message: publishing client to broker and broker to subscribing client. We need to look at them separately since there are subtle differences. The QoS level for publishing client to broker is depending on the QoS level the client sets for the particular message. When the broker transfers a message to a subscribing client it uses the QoS of the subscription made by the client earlier. That means, QoS guarantees can get downgraded for a particular receiving client if subscribed with a lower QoS.

IMPORTANCE OF QOS

QoS is a major feature of MQTT, it makes communication in unreliable networks a lot easier because the protocol handles retransmission and guarantees the delivery of the message, regardless how unreliable the underlying transport is. Also it empowers a client to choose the QoS level depending on its network reliability and application logic.

QOS IMPLEMENTATION IN MQTT

QoS 0 – at most once

The minimal level is zero and it guarantees a best effort delivery. A message won't be acknowledged by the receiver or stored and redelivered by the sender. This is often called "fire and forget" and provides the same guarantee as the underlying TCP protocol.

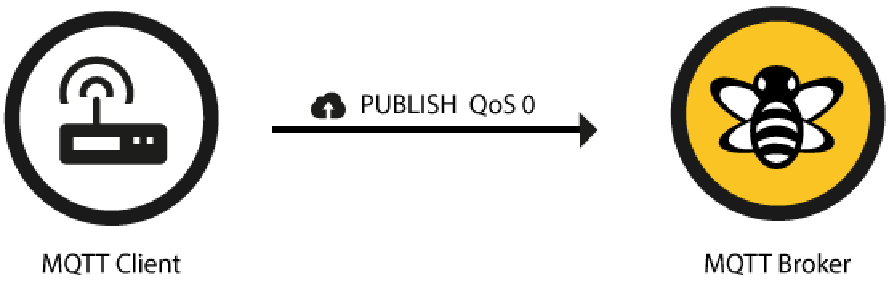


Fig 4.6: QoS 0 pattern

QoS 1 – At least once

– at least once When using QoS level 1, it is guaranteed that a message will be delivered at least once to the receiver. But the message can also be delivered more than once.

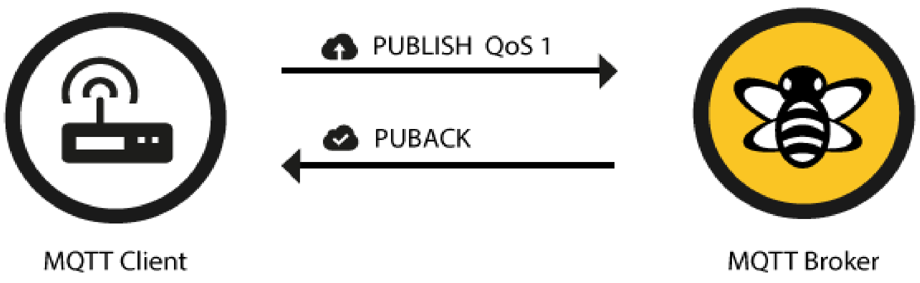


Fig 4.7: QoS1 pattern

The sender will store the message until it gets an acknowledgement in form of a PUBACK command message from the receiver.

The association of PUBLISH and PUBACK is done by comparing the packet identifier in each packet. If the PUBACK isn't received in a reasonable amount of time the sender will resend the PUBLISH message. If a receiver gets a message with QoS 1, it can process it immediately, for example sending it to all subscribing clients in case of a broker and then replying with the PUBACK.

The duplicate (DUP) flag, which is set in the case a PUBLISH is redelivered, is only for internal purposes and won't be processed by broker or client in the case of QoS 1. The receiver will send a PUBACK regardless of the DUP flag.

QoS 2 - exactly once

The highest QoS is 2, it guarantees that each message is received only once by the counterpart. It is the safest and also the slowest quality of service level. The guarantee is provided by two flows there and back between sender and receiver.

If a receiver gets a QoS 2 PUBLISH it will process the publish message accordingly and acknowledge it to the sender with a PUBREC message.

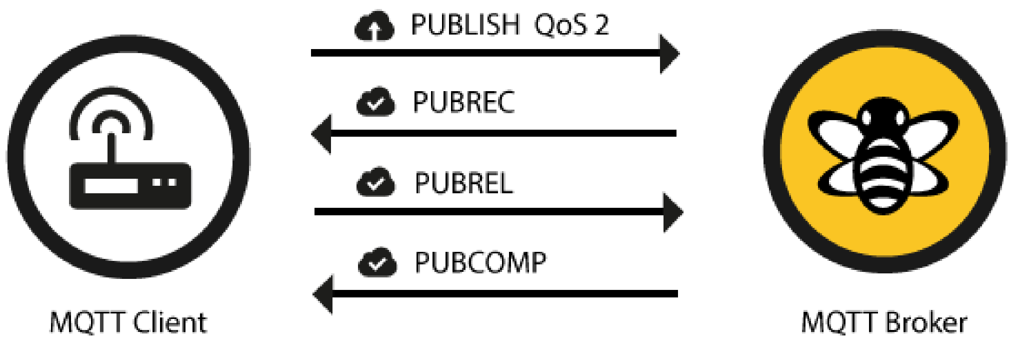


Fig 4.8: QoS2 pattern

The receiver will store a reference to the packet identifier until it has send the PUBCOMP. This is important for avoid processing the message a second time. When the sender receives the PUBREC it can safely discard the initial publish, because it knows that the counterpart has successfully received the message. It will store the PUBREC and respond with a PUBREL.

After the receiver gets the PUBREL it can discard every stored state and answer with a PUBCOMP. The same is true when the sender receives the PUBCOMP.

When the flow is completed both parties can be sure that the message has been delivered and the sender also knows about it.

Whenever a packet gets lost on the way, the sender is responsible for resending the last message after a reasonable amount of time. This is true when the sender is a MQTT client and also when a MQTT broker sends a message. The receiver has the responsibility to respond to each command message accordingly.

4.4 MQTT ARCHITECTURE

The typical MQTT architecture can be divided into two main components as shown in figure 2. Each component briefly described below.

1)**Client**: Client could be a Publisher or Subscriber and it always establishes the network connection to the Server (Broker). It can do the following things [7]:

- Publish messages for the interested users.
- Subscribe in interested subject for receiving messages.
- Unsubscribe to extract from the subscribed subjects.
- Detach from the Broker.

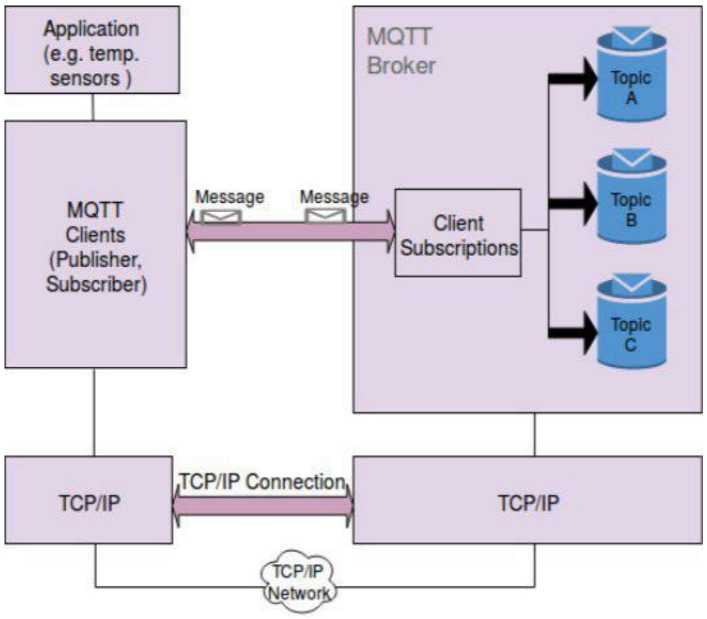


Figure 4.4: MQTT Architecture

*2) **Broker**:* Broker controls the distribution of information and mainly responsible for receiving all messages from publisher, filtering them, decide who is interested in it and then sending the messages to all subscribed clients. It can do the following things:

- Accept Client requests.
- Receives Published messages by Users.
- Processes different requests like Subscribe and Unsubscribe from Users.

After receiving messages from publisher sends it to the interested Users.

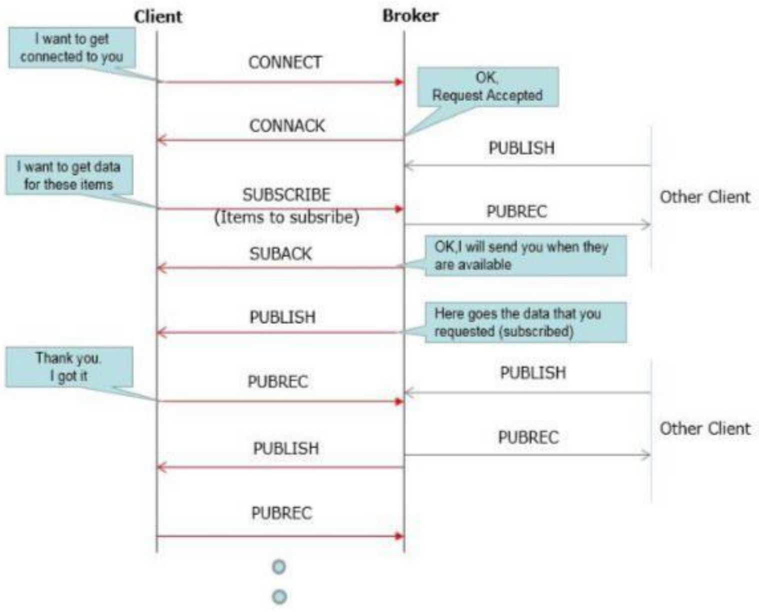

Client
Broker
I want to get connected to you
CONNECT
OK, Request Accepted
CONNACK
PUBLISH
I want to get data for these items
SUBSCRIBE (Items to subsribe)
PUBREC
Other Client
SUBACK
OK,I will send you when they are available
PUBLISH
Here goes the data that you requested (subscribed)
Thank you. I got it
PUBREC
PUBLISH
PUBLISH
PUBREC
Other Client
PUBREC

Figure 4.5: Working of MQTT

4.5 WHERE TO USE MQTT

Constrained environment like embedded devices which has confined processing capacity or devices which are connected to unstable network are most fitted for MQTT protocol. MQTT is being used in many situations [1] which are described below,

1. Healthcare*:* By using MQTT, a healthcare association needed to create a flexible checking solution. Following are the arrangement expected to address of victim care:

- Keeping track of victims besides they go away from the clinic.
- Upgrading the effectiveness of subsequent tests.
- Achieving advanced industry information catch principles.

The organization worked with IBM to make an answer in which an MQTT customer is inserted in a home observing machine that gathers diagnostics at whatever point the victim is in nearness to a base system. Then it forwards the indicative information through the web to the main domain, which is given to an application which analyses the measurements and aware the healthcare team if there are hints the victim perhaps carrying trouble. It spares cash for the association also its victims, as there is constrained requirement for victims to go hospital for regular check-ups if they are doing fine.

2. Energy and utilities: A service organization was confronted with increasing expenses to deliver power among with rising interest for electricity by their client root, which was not able, normally, to spend forever expanding amounts. Therefore instead of quickly carried out generation charges that their clients possibly couldn't spend, the organization first looked for an answer for decrease general request for power by putting smart meters in clients' apartments to remotely manage the application of definite power absorbing device. In any case, the arrangement expected to minimize utilization of accessible information network, for that the organization salaried according to the quantity of information transferred.

Making of Virtual power plant (VPP) was the arrangement which sits between the organization's producing origins and their clients. At that point, apartment gateway examines, furnished with a progressed MQTT customer, distribute the utilization information to the VPP at normal interval throughout the nearby cell phone arrange.

3. ***Social Networking**:* A long range interpersonal communication organization experienced latency issues during transferring information. The strategy the organization utilized to deliver data was stable yet time-consuming, and if it remained to utilize the similar mechanism then the solutions were restricted. Another structure for a constant.
association among the servers lacking absorbing more battery power was required, that is basic to clients of the organization's civic communication site
Using MQTT protocol the organization's designers tackled the issue in social networking. With keeping up a MQTT association also directing data via MQTT's conversational channel (chat pipeline), the organization was capable to accomplish data distribution by speeds of 1×105 microseconds, instead of several minutes,

4.6 BROKERS OF MQTT

The MQTT broker is the heart of each MQTT arrangement. It provides connecting link between applications or physical devices and enterprise systems. Brokers are in charge of subscription, determined sessions, missed messages and general security, including authentication and authorization. The follow table describes mostly used brokers with their features and limitations.

Table 4.1 various MQTT brokers and their features:

Broker	About	Features	Limitations
Mosquitto	It supports MQTT version 3.1 and it is an open source	-All QoS - Authentication - Bridge - Dynamic topics - Web sockets	- Clustering - Fewer Configuration Not allowing simultaneous connection with using authentication
RSMB(Really small message broker)	It is a tiny broker which supports V3 and V3.1	- All QoS - Bridge - Dynamic topic	- Security - Web sockets - Cluster
MQTT.js	It is an MQTT broker among client/server API Production recorded in JavaScript	- All QoS - Dynamic topics - Web sockets - SSL	- Bridge - Authentication - Cluster
HiveMQ	HiveMQ empowers organization to attach all devices and services with nominal effort by victimization the de-facto	- All QoS - Bridge - Dynamic topics - Web sockets - TLS/SSL - Cluster	- Open Standard - Performance degradation because of TLS
VerneMQ	VerneMQ could be a superior, distributed MQTT message broker supports MQTT version 3.1 and 3.1.1	- All QoS - Bridge - Authentication - Dynamic topics - Web sockets - Encryption	-Performance degradation because of TLS

CHAPTER 5

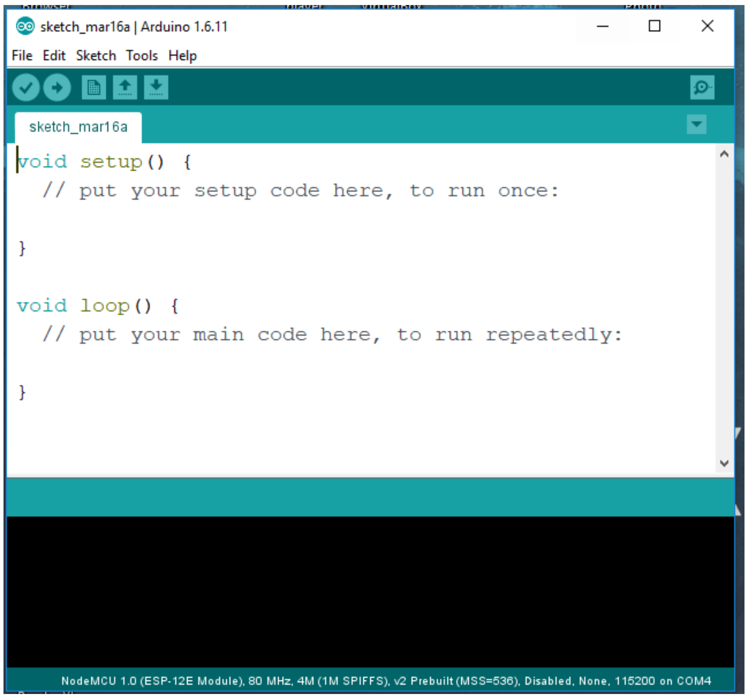
TOOLS AND PLATFORMS

5.1 ARDUINO IDE

The Arduino Integrated Development Environment - or Arduino Software (IDE) - contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions and a series of menus. It connects to the Arduino and Genuino hardware to upload programs and communicate with them.

WRITING SKETCHES

Programs written using Arduino Software (IDE) are called sketches. These sketches are written in the text editor and are saved with the file extension .ino. The editor has features for cutting/pasting and for searching/replacing text. The message area gives feedback while saving and exporting and also displays errors. The console displays text output by the Arduino Software (IDE), including complete error messages and other information. The bottom righthand corner of the window displays the configured board and serial port. The toolbar buttons allow us to verify and upload programs, create, open, and save sketches, and open the serial monitor.



```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

```
NodeMCU 1.0 (ESP-12E Module), 80 MHz, 4M (1M SPIFFS), v2 Prebuilt (MSS=536), Disabled, None, 115200 on COM4
```

Fig 5.1 :Arduino IDE

SKETCHBOOK

The Arduino Software (IDE) uses the concept of a sketchbook: a standard place to store our programs (or sketches). The sketches in our sketchbook can be opened from the File > Sketchbook menu or from the Open button on the toolbar. The first time we run the Arduino software, it will automatically create a directory for our sketchbook. We can view or change the location of the sketchbook location from with the Preferences dialog.

UPLOADING

Before uploading our sketch, we need to select the correct items from the Tools > Board and Tools > Port menus.On Windows, the ports are COM1 or COM2 (for a serial board) or COM4, COM5, COM7, or higher (for a USB board) - to find out, we look for USB serial device in the ports section of the Windows Device Manager.

On Linux, it should be /dev/ttyACMx , /dev/ttyUSBx or similar. Once we've selected the correct serial port and board, press the upload button in the toolbar or select the Upload item from the Sketch menu. On most boards, we'll see the RX and TX LEDs blink as the sketch is uploaded. The Arduino Software (IDE) will display a message when the upload is complete, or show an error.

When we upload a sketch, we're using the Arduino bootloader, a small program that has been loaded on to the microcontroller on our board. It allows us to upload code without using any additional hardware. The bootloader is active for a few seconds when the board resets; then it starts whichever sketch was most recently uploaded to the microcontroller. The bootloader will blink the on-board LED when it starts (i.e. when the board resets).

LIBRARIES

Libraries provide extra functionality for use in sketches, e.g. working with hardware or manipulating data. To use a library in a sketch, select it from the Sketch > Import Library menu. This will insert one or more #include statements at the top of the sketch and compile the library with our sketch. Because libraries are uploaded to the board with our sketch, they increase the amount of space it takes up. If a sketch no longer needs a library, simply delete its #include statements from the top of the code.

SERIAL MONITOR

This displays serial sent from the Arduino or Genuino board over USB or serial connector. To send data to the board, enter text and click on the "send" button or press enter. Choose the baud rate from the drop-down menu that matches the rate passed to Serial.begin in the sketch. On Windows, Mac or Linux the board will reset (it will rerun our sketch) when we connect with the serial monitor. Serial Monitor does not process control characters.

BOARDS

The board selection has two effects: it sets the parameters (e.g. CPU speed and baud rate) used when compiling and uploading sketches; and sets and the file and fuse settings used by the burn bootloader command. Arduino Software (IDE) includes the built in support for the boards in the following list, all based on the AVR Core. The Boards Manager included in the standard installation allows to add support for the growing number of new boards based on different cores like Arduino Due, Arduino Zero, Edison, Galileo and so on.

We select the NodeMCU board from the Board Manager menu as we are using that particular board in this project.

5.2 MOSQUITTO BROKER

Mosquitto provides standards compliant server and client implementations of the MQTT messaging protocol. MQTT uses a publish/subscribe model, has low network overhead and can be implemented on low power devices such microcontrollers that might be used in remote Internet of Things sensors. As such, Mosquitto is intended for use in all situations where there is a need for lightweight messaging, particularly on constrained devices with limited resources [3] .

The Mosquitto project is a member of the Eclipse Foundation There are three parts to the project. • The main mosquitto server.
• The mosquitto_pub and mosquitto_sub client utilities that are one method of communicating with an MQTT server.
• An MQTT client library written in C, with a C++ wrapper.

Mosquitto allows research directly related to the MQTT protocol itself, such as comparing the performance of MQTT and the Constrained Application Protocol (CoAP) (Thangavel et al. 2014) or investigating the use of OAuth in MQTT (Fremantle et al. 2014). Mosquitto supports other research activities as a useful block for building larger systems and has been used to evaluate MQTT for use in Smart City Services (Antonić et al. 2015), and in the development of an environmental monitoring system (Bellavista, Giannelli, and Zamagna 2017). Mosquitto has also been used to support research less directly as part of a scheme for remote control of an experiment (Schulz, Chen, and Payne 2014). The mosquitto broker has reasonable high throughput andlow delay [21] .

5.3 ECLIPSE PAHO CLIENT

The Paho project has been created to provide scalable open-source implementations of open and standard messaging protocols aimed at new, existing, and emerging applications for Machine-to-Machine (M2M) and Internet of Things (IoT) [10] .

Paho reflects the inherent physical and cost constraints of device connectivity. Objectives include effective levels of decoupling between devices and applications, designed to keep markets open and encourage the rapid growth of scalable Web and Enterprise middleware and applications. Paho initially started with MQTT publish/subscribe client implementations for use on embedded platforms, and in the future will bring corresponding server support as determined by the community.

Eclipse Paho is an umbrella project for several MQTT and MQTT-SN client implementations in different programming languages. The Eclipse Paho project was one of the first open source MQTT client implementations available and is actively maintained by a huge community. Paho also features a Java client which is suited for embedded use, Android applications and Java applications in general. The initial code base was donated to Eclipse by IBM in 2012.

The Java version of Eclipse Paho is rock-solid and is used by a broad range of companies from different industries around the world to connect to MQTT brokers. The synchronous/blocking API of Paho makes it easy to implement the applications MQTT logic in a clean and concise way while the asynchronous API gives the application developer full control for high-performance MQTT clients that need to handle high throughput. The Paho API is highly call back based and allows to hook in custom business logic to different events, e.g. when a message is received or when the connection to the broker was lost. Paho supports all MQTT features and a secure communication with the MQTT Broker is possible via TLS.

The basic Procedure of creating a Mqtt client using the paho library is as follows

1.) Download the specific paho client library for the platform under use.

2.) Create a MQTT client object using the MqttClient class, All MQTT operations are exposed via a MqttClient object (Before connecting, publishing or subscribing, the user needs to create an instance of this MqttClient first).

3.) Connect to the MQTT broker using client.connect() function.

4.) Use options.setUserName and options.setPassword if the connection requires authentication.

5.) Then publish and subscribe using client.publish() and client.subscribe() routines respectively.

6.) Use client.unsubscribe() to unsubscribe to a topic and client.disconnect() to disconnect from the broker.

5.4 CLOUDMQTT

CloudMQTT are managed Mosquitto servers in the cloud. Mosquitto implements the MQ Telemetry Transport protocol, MQTT, which provides lightweight methods of carrying out messaging using a publish/subscribe message queueing model [14] .

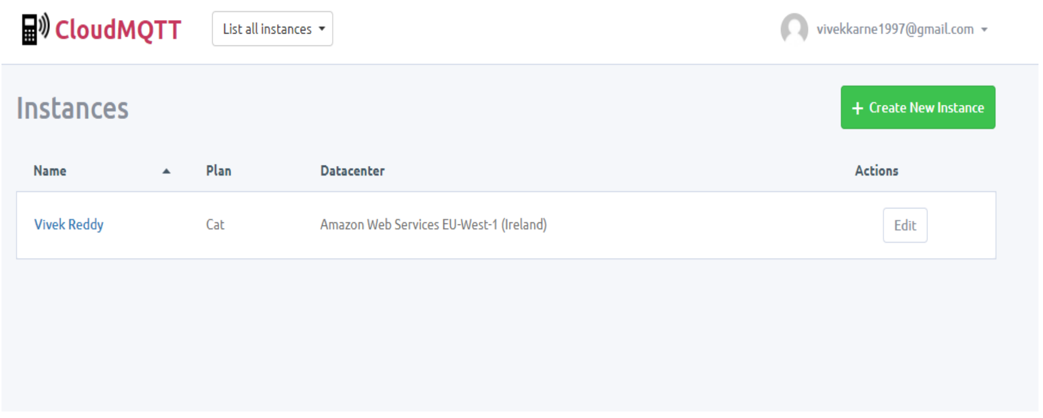
MQTT is the machine-to-machine protocol of the future. It is ideal for the "Internet of Things" world of connected devices. Its minimal design makes it perfect for built-in systems, mobile phones and other memory and bandwidth sensitive applications.

Message queues provide an asynchronous communications protocol, the sender and receiver of the message do not need to interact with the message queue at the same time. Messages placed onto the queue are stored until the recipient retrieves them or until the messages times out. MQTT and Mosquitto are for good use by bandwidth sensitive applications.

CloudMQTT let us focus on the application instead of spending time on scaling the broker or patching the platform.

CREATE A CLOUDMQTT INSTANCE

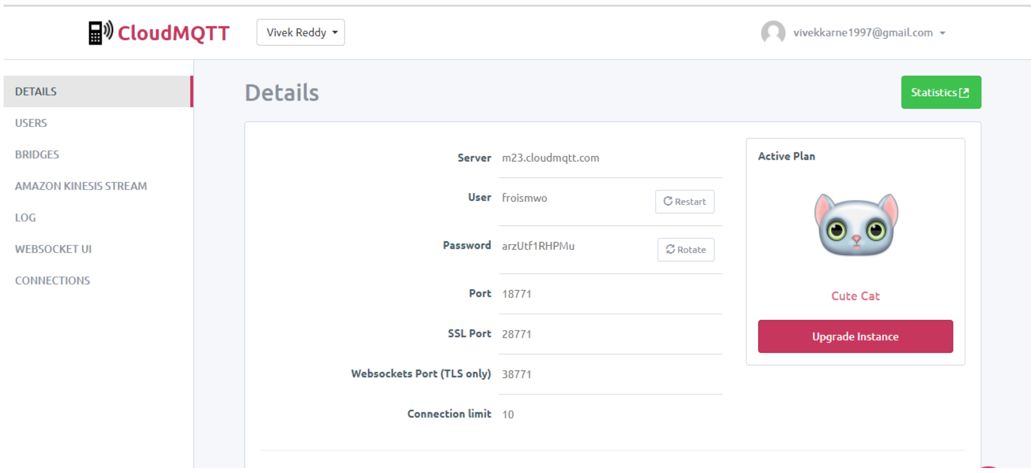
Create an account and login to the control panel and press + Create New Instance to create a new instance.



Name	Plan	Datacenter	Actions
Vivek Reddy	Cat	Amazon Web Services EU-West-1 (Ireland)	Edit

Fig 5.2: Cloud MQTT Instance

The instance is immediately provisioned after sign up and we can view the instance details, such as connection information, at the details page. We are sometimes forced to format a connection URL while connecting via client libraries, it should look like mqtt://user:password@server:port



Server	m23.cloudmqtt.com
User	froismwo
Password	arzUtf1RHPMu
Port	18771
SSL Port	28771
Websockets Port (TLS only)	38771
Connection limit	10

Fig 5.3: Cloud MQTT Credentials

BRIDGES

Bridges are essentially a way for one MQTT broker to connect to another MQTT broker. This is a very useful feature and enables us to setup our own HA "cluster".

We can create our own bridge from the console page.

In the "Connection uri" field we must enter a connection string in thisformat: mqtt://USER:PASSWORD@host:port If we don't enter a port the default 1883 will be used.

In the "Direction" dropdown we can choose between:

- Out - the local broker will forward all messages to the remote broker.
- In - the local broker will subscribe to all topics on the remote broker.
- Both - combining in and out.

With the 'Both' option we can create loops. This means that the brokers are forever forwarding each other the same message.

To get around the loop problem we can specify a "Local prefix" and a "Remote prefix". Local in this context means the broker where the bridge is defined. This means that only a topic that matches the prefix will be forwarded.

We have two brokers, A and B, where A is the local broker hosted by CloudMQTT.

On the console page we enter the connection options to broker B and select the direction "both". We then enter "B/" into the "Local prefix" field. We enter "A/" into the "Remote prefix" field.

Then we publish the current time to the topic "A/time" on broker B. On broker A we can subscribe to "B/time" and we will see the times published to broker B. This works in both directions.

5.5 FIREBASE

Firebase is a Backend-as-a-Service—BaaS—that started as a YC11 startupand grew up into a next-generation app-development platform on Google Cloud Platform. It combines Analytics,Database,Authentication,Storage,Hosting,Crash Reports,AdMob etc. Google is trying to Integrate all basic services needed for an android app through Firebase [20] .

Analytics-Provides Insights about our app like number of users, what they doing with our app.

5.5.1 AUTHENTICATION

FirebaseUI is a library built on top of the Firebase Authentication SDK that provides drop-in UI flows for use in our app.FirebaseUI provides the following benefits:

- **Multiple Providers** - sign-in flows for email, phone authentication, Google, Facebook, Twitter and GitHub sign-in.
- **Account Linking** - flows to safely link user accounts across identity providers.
- **Customization** - override CSS styles of FirebaseUI to match our app requirements. Also, because FirebaseUI is open source, we can fork the project and customize it exactly to our needs.
- **One-tap sign-up and automatic sign-in** - automatic integration with One-tap sign-up for fast cross-device sign-in.
- **Localized UI** - internationalization for over 40 languages.

5.5.2 FIREBASE REALTIME DATABASE

It stores and syncs data with a NoSQL cloud database. Data is synced across all clients in realtime, and remains available when our app goes offline.

The Firebase Realtime Database is a cloud-hosted database. Data is stored as JSON and synchronized in realtime to every connected client. When we build cross-platform apps with our iOS, Android, and JavaScript SDKs, all of our clients share one Realtime Database instance and automatically receive updates with the newest data.

WORKING

The Firebase Realtime Database lets us build rich, collaborative applications by allowing secure access to the database directly from client-side code. Data is persisted locally, and even while offline, realtime events continue to fire, giving the end user a responsive experience. When the device regains connection, the Realtime

Database synchronizes the local data changes with the remote updates that occurred while the client was offline, merging any conflicts automatically.

The Realtime Database provides a flexible, expression-based rules language, called Firebase Realtime Database Security Rules, to define how our data should be structured and when data can be read from or written to. When integrated with Firebase Authentication, developers can define who has access to what data, and how they can access it.

The Realtime Database is a NoSQL database and as such has different optimizations and functionality compared to a relational database. The Realtime Database API is designed to only allow operations that can be executed quickly. This enables us to build a great realtime experience that can serve millions of users without compromising on responsiveness. Because of this, it is important to think about how users need to access our data and then structure it accordingly.

5.5.3 FIREBASE CLOUD STORAGE

Cloud Storage is built for app developers who need to store and serve user-generated content, such as photos or videos.

Cloud Storage for Firebase is a powerful, simple, and cost-effective object storage service built for Google scale. The Firebase SDKs for Cloud Storage add Google security to file uploads and downloads for our Firebase apps, regardless of network quality. We can use our SDKs to store images, audio, video, or other user-generated content. On the server, we can use Google Cloud Storage, to access the same files.

WORKING

Developers use the Firebase SDKs for Cloud Storage to upload and download files directly from clients. If the network connection is poor, the client is able to retry the operation right where it left off, saving our users time and bandwidth.

Cloud Storage stores y\our files in a Google Cloud Storage bucket, making them accessible through both Firebase and Google Cloud. This allows us the flexibility to upload and download files from mobile clients via the Firebase SDKs, and do server-side processing such as image filtering or video transcoding using Google Cloud Platform. Cloud Storage scales automatically, meaning that there's no need to migrate to any other provider. Learn more about all the benefits of our integration with Google Cloud Platform.

The Firebase SDKs for Cloud Storage integrate seamlessly with Firebase Authentication to identify users, and we provide a declarative security language that lets us set access controls on individual files or groups of files, so we can make files as public or private as we want.

5.5.4 FIREBASE HOSTING

Firebase Hosting provides fast and secure static hosting for our web app. Firebase Hosting is production-grade web content hosting for developers. With Hosting, we can quickly and easily deploy web apps and static content to a global content-delivery network (CDN) with a single command.

WORKING

Firebase Hosting is built for the modern web developer. Static sites are more powerful than ever with the rise of front-end JavaScript frameworks like Angular and static generator tools like Jekyll. Whether we are deploying a simple app landing page or a complex Progressive Web App, Hosting gives us the infrastructure, features, and tooling tailored to deploying and managing static websites.

Hosting gives our project a subdomain on the firebaseapp.com domain. Using the Firebase CLI, we can deploy files from local directories on our computer to our Hosting server. Files are served over an SSL connection from the closest edge server on firebase's global CDN.

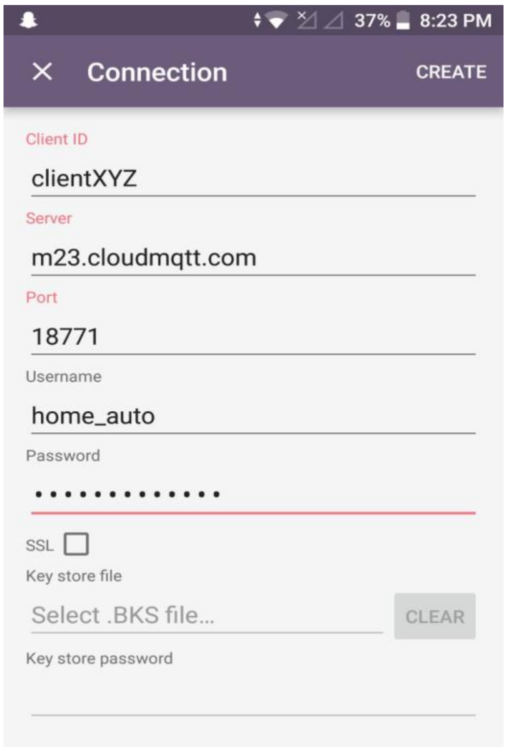
In addition to static content hosting, Firebase Hosting offers lightweight configuration options for us to be able to build sophisticated Progressive Web Apps. We can easily rewrite URLs for client-side routing or set up custom headers.

Once we're ready to take a site to production, we can connect our own domain name to Firebase Hosting. Firebase automatically provides an SSL certificate for our domain so all our contents are served securely.

CHAPTER 6

IMPLEMENTATION

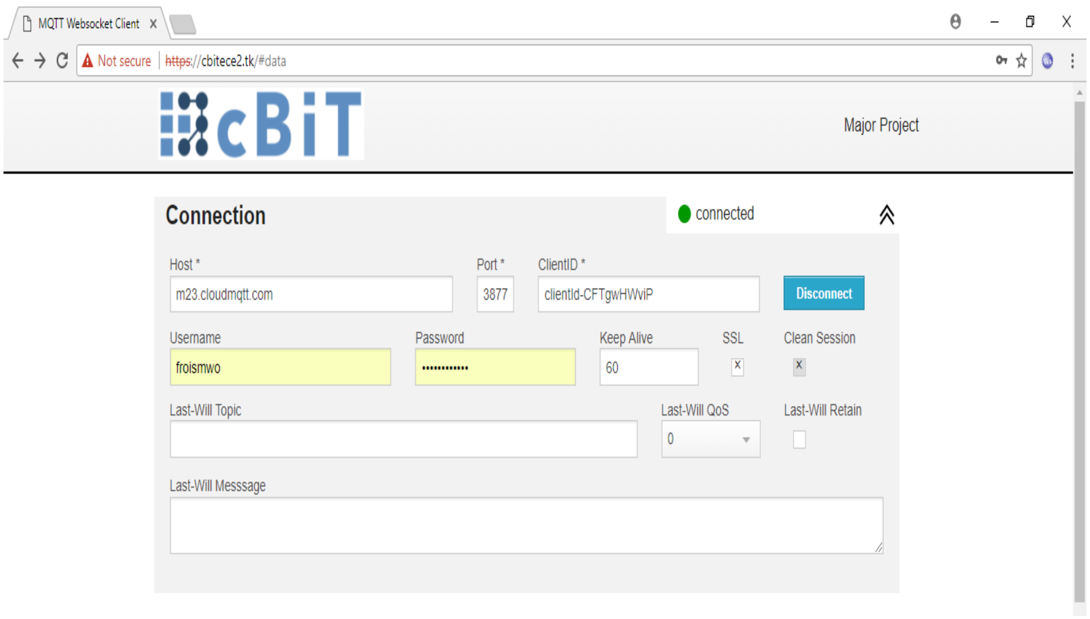
There are two ways to access our Home automation system, one through a website and another through an Android application.



Client ID
clientXYZ
Server
m23.cloudmqtt.com
Port
18771
Username
home_auto
Password
••••••••••••••
SSL ☐
Key store file
Select .BKS file...

Key store password

Fig6.1: Android App user interface



Connection ● connected

Host *	Port *	ClientID *
m23.cloudmqtt.com	3877	clientId-CFTgwHWviP

Username	Password	Keep Alive	SSL	Clean Session
froismwo	••••••••••••	60	X	X

Last-Will Topic	Last-Will QoS	Last-Will Retain
	0	☐

Last-Will Messsage

Fig6.2:Website user interface

These two user interfaces are used to give the commands to the home automation system according to the requirement of the user.

6.1 BLOCK DIAGRAM

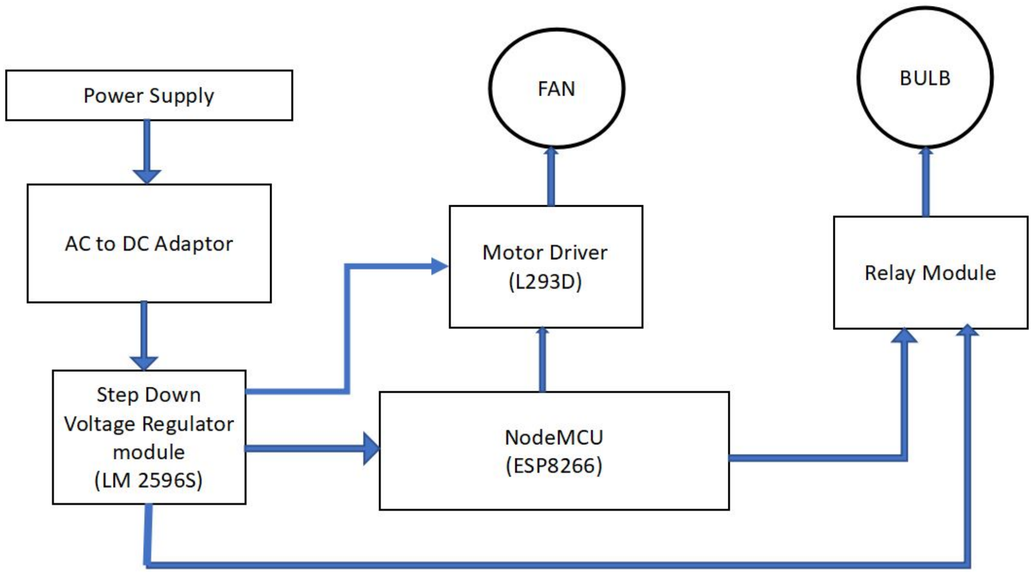


Fig 6.3 :Block Diagram

The commands to switch ON/OFF the appliances are given either from the android app or from the website.

NodeMCU takes these commands and does the appropriate action, i.e sending a logic 1 or 0 to the intermediate modules (Motor driver and Relay) which ultimately control the Fan and Bulb.

The power supply of 5v is provided to all the modules through a Stepdown voltage regulator (LM 2596S).

6.2 Hardware Implementation

First we have to setup Arduino ide so that we can burn the code into NodeMCU board. The following steps are to be followed:

- Start Arduino and open Preferences window.
- Enter http://arduino.esp8266.com/stable/package_esp8266com_index.json into Additional Board Manager URLs field. You can add multiple URLs, separating them with commas.

- Open Boards Manager from Tools > Board menu and find esp8266 platform.
- Select the version needed from a drop-down box.
- Click install button.
- Select ESP8266 board from Tools > Board menu after installation.

In order to make our ESP8266 a MQTT client we use PubSubClient. The pubsubclient provides many functions like connecting to a mqtt broker, subscribing to a topic, publishing to a topic, setCallback, reconnect, etc. We create a pubsubclient object then use that object to call the functions defined in pubsubclient library. So the code above uses this library to implement the functionality required [18] .

6.3 CODE

```
#include <ESP8266WiFi.h>

#include <PubSubClient.h>

// Update these with values suitable for your network.

const char* ssid = "linksys";

const char* password = "9052345605";

const char* mqtt_server = "m23.cloudmqtt.com";

const char* mqttUser="froismwo";

const char* mqttPass="arzUtf1RHPMu";

int var1=0;

int var2=0;

WiFiClient espClient;

PubSubClient client(espClient);

long lastMsg = 0;

char msg[50];

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(".");

}

randomSeed(micros());

Serial.println("");

Serial.println("WiFi connected");

Serial.println("IP address: ");

Serial.println(WiFi.localIP());

}

void callback(char* topic, byte* payload, unsigned int length) {

Serial.print("Message arrived [");

Serial.print(topic);

Serial.print("] ");

if(strcmp(topic,"BULB")==0)

{

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

Serial.print((char)payload[i]);

}

Serial.println();

// Switch on the LED if an 1 was received as first character

if ((char)payload[0] == '1') {

digitalWrite(D3, LOW);

client.publish("checkBULB","BULB IS ON IN HOUSE");

// Turn the LED on (Note that LOW is the voltage level but actually the LED is
on; this is because it is acive low on the ESP-01)

} else {

digitalWrite(D3, HIGH);
client.publish("checkBULB","BULB IS OFF IN HOUSE"); // Turn the LED off
by making the voltage HIGH

} }

if(strcmp(topic,"FAN") == 0)

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

Serial.print((char)payload[i]);

}

// Switch on the LED if an 1 was received as first character

if ((char)payload[0] == '0') {

analogWrite(D1 , 0);

digitalWrite(D2 , LOW);

client.publish("checkFan","Fan is OFF");

} else if((char)payload[0] == '1') {

analogWrite(D1,204);

digitalWrite(D2,LOW);

client.publish("checkFan","Fan is ON and speed is 1"); // Turn the LED off by
making the voltage HIGH

}

else if((char)payload[0] == '2') {

digitalWrite(D2,LOW);

analogWrite(D1, 408);

client.publish("checkFan","Fan is ON and speed is 2");// Turn the LED off by
making the voltage HIGH

}

else if((char)payload[0] == '3') {

analogWrite(D1, 612);

digitalWrite(D2,LOW);

client.publish("checkFan","Fan is ON and speed is 3");// Turn the LED off by
making the voltage HIGH

}
```

```
else if((char)payload[0] == '4') {
analogWrite(D1,816);
digitalWrite(D2,LOW);
client.publish("checkFan","Fan is ON and speed is 4");// Turn the LED off by
making the voltage HIGH
}
else if((char)payload[0] == '5') {
analogWrite(D1, 1023  );
digitalWrite(D2, LOW);
client.publish("checkFan","Fan is ON and speed is 5");// Turn the LED off by
making the voltage HIGH
} } }
void reconnect() {
// Loop until we're reconnected
while (!client.connected()) {
Serial.print("Attempting MQTT connection...");
// Create a random client ID
String clientId = "ESP8266Client-";
clientId += String(random(0xffff), HEX);
// Attempt to connect
if (client.connect(clientId.c_str(),mqttUser,mqttPass)) {
Serial.println("connected");
// Once connected, publish an announcement and resubscribe
client.subscribe("BULB");
Serial.println("subs");
client.subscribe("FAN");
} else {
Serial.print("failed, rc=");
Serial.print(client.state());
Serial.println(" try again in 5 seconds");
// Wait 5 seconds before retrying
delay(5000);
} } }
void setup() {
pinMode(D3, OUTPUT);
pinMode(D1, OUTPUT);
pinMode(D2, OUTPUT);
digitalWrite(D3, HIGH);
digitalWrite(D2, LOW);
digitalWrite(D0, HIGH);
// Initialize the D3 pin as an output
Serial.begin(115200);
setup_wifi();
client.setServer(mqtt_server,18771);
client.setCallback(callback);
}
void loop() {
if (!client.connected()) {
reconnect();
}
client.loop();
long now = millis();
if (now - lastMsg > 50000) {
lastMsg = now;
snprintf (msg, 75, "Server OK ");
Serial.print("Publish message: ");
Serial.println(msg);
```

```
client.publish("checkServer", msg);
var1=digitalRead(D0);
  }
}
```

6.4 WEB SOCKETS CLIENT UI IN WEB-BROWSER

We will be using cloudmqtt, which as described in the previous chapter is nothing but a mosquito broker hosted in the cloud. Inorder to communicate with the broker and tell it to send commands to our NodeMcu we have designed a Websockets UI which will act as a client to publish data to cloudmqtt. Then this published data will be routed to our NodeMCU which is using a pubsubclient library and appropriate action will be taken [16].

The steps required to designing a websocket client which works on all major browsers is as follows:

- First we will require the paho java script client which can be downloaded from the official website.
- This script includes all the major functions required to connect to a MQTT broker.
- We have to include this “mqttws31.js” javascript library in the pages where we want to use the functionality using : <script src="js/mqttws31.js"></script>.
- Now we can include jquery library by downloading it from jquery.com or use one of the many CDNs available.
- Include relevant cascaded style sheets which make the webpage look attractive.
- Design the website using css box model and include boxes, tables, dropdown menus as per requirements.
- Now the website is ready to be deployed. To test the proper working we first hosted it on our computer using xampp.

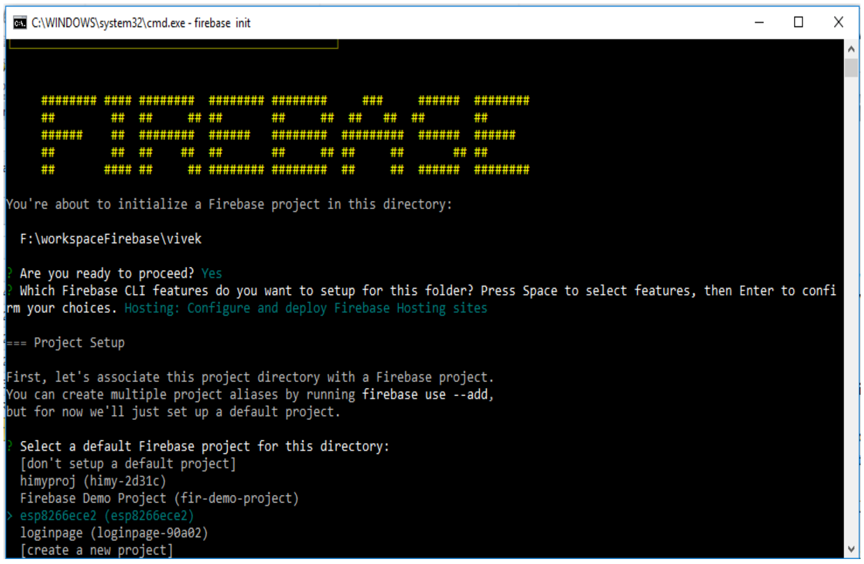
After testing it extensively and eliminating all possible errors we were able to access it using local hosting. Since it is a front end application we did not require any data bases. After this process we searched for a place which would provide hosting for free. We came up across Firebase which is a platform which was implemented to make building mobile applications easier. It provided many services as listed in the previous chapter. We needed hosting and firebase was capable of hosting front end websites which did not require back end functionalities.

Deploying website on Firebase servers:

First we have to login into our google account on firebase.google.com. Then go to our console where we will have to create a new firebase project. After creating we will find many services which firebase offers [21] .

These are the steps we followed to deploy our website:

- After creating a new project we have to install firebase command line tools using npm. For npm visit nodejs.org and download nodejs.
- Use the command: npm install -g firebase-tools to install the firebase tools.
- Create a folder which will contain the remote offline version of the website.
- Open command prompt and go to the location of newly created folder. Use the command firebase login and enter your credentials.
- Initiate a new project using firebase init. While initiating we should select hosting and link this remote folder to the actual project created on firebase server, as shown in this picture.



```
C:\WINDOWS\system32\cmd.exe - firebase init

     ######## #### ########  ######## ########     ###     ######  ########
     ##        ##  ##     ## ##       ##     ##  ##   ##  ##       ##
     ######    ##  ########  ######   ########  #########  ######  ######
     ##        ##  ##    ##  ##       ##     ## ##     ##       ## ##
     ##       #### ##     ## ######## ########  ##     ##  ######  ########

You're about to initialize a Firebase project in this directory:

  F:\workspaceFirebase\vivek

? Are you ready to proceed? Yes
? Which Firebase CLI features do you want to setup for this folder? Press Space to select features, then Enter to confi
rm your choices. Hosting: Configure and deploy Firebase Hosting sites

=== Project Setup

First, let's associate this project directory with a Firebase project.
You can create multiple project aliases by running firebase use --add,
but for now we'll just set up a default project.

? Select a default Firebase project for this directory:
  [don't setup a default project]
  himyproj (himy-2d31c)
  Firebase Demo Project (fir-demo-project)
> esp8266ece2 (esp8266ece2)
  loginpage (loginpage-90a02)
  [create a new project]
```

Fig 6.4: Firebase command line tools

- Public directory will be the default directory, we didn't make any changes to it.
- Copy and paste the website folders into the public directory and use the command firebase deploy to host the website.

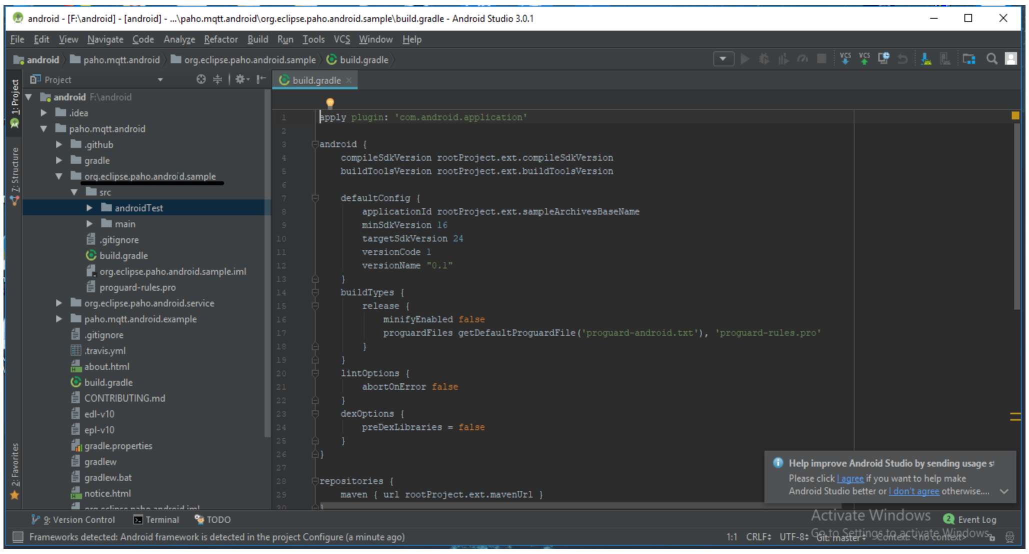
Now after going back to firebase console and we will see that a website has been hosted and we will have a website link to access that website. We created a free custom domain name using www.dot.tk, named cbitece2.tk. Now we have to connect this domain name to the website hosted on firebase.

- Click on connect domain and add your domain name. Go to your DNS manager in my.freenom.com and add a txt record with the information provided by firebase.
- It will take some time for firebase to verify that it your domain name and after that both the domain names will be linked together.
- When we entered our .tk domain name it got redirected under the hood to our firebase domain name.
- Thus we have deployed our website on firebase.

6.5 WEB SOCKETS CLIENT ANDROID APPLICATION

We have used Eclipse Paho android service which is implemented already by eclipse developers, the entire source code is present on GitHub so just cloning it was sufficient. The Paho Android Service is an MQTT client library written in Java for developing applications on Android. We used android studio for testing the source code if it is meeting the requirements or not [17].

We had to add the Eclipse Maven repository to your build.gradle file and then add the Paho dependency to the dependencies section. The source code comes along with a sample app which is contained in the 'org.eclipse.android.sample' package so we have used this sample app in our project as it was sufficient to control the NodeMCU. Pictures of this app have been included in the results section of this report.



android - [F:\android] - [android] - ...\paho.mqtt.android\org.eclipse.paho.android.sample\build.gradle - Android Studio 3.0.1
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
android
paho.mqtt.android
org.eclipse.paho.android.sample
build.gradle
Project
android F:\android
.idea
paho.mqtt.android
.github
gradle
org.eclipse.paho.android.sample
src
androidTest
main
.gitignore
build.gradle
org.eclipse.paho.android.sample.iml
proguard-rules.pro
org.eclipse.paho.android.service
paho.mqtt.android.example
.gitignore
.travis.yml
about.html
build.gradle
CONTRIBUTING.md
edl-v10
epl-v10
gradle.properties
gradlew
gradlew.bat
notice.html
9: Version Control
Terminal
TODO
Event Log

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion rootProject.ext.compileSdkVersion
    buildToolsVersion rootProject.ext.buildToolsVersion

    defaultConfig {
        applicationId rootProject.ext.sampleArchivesBaseName
        minSdkVersion 16
        targetSdkVersion 24
        versionCode 1
        versionName "0.1"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
    lintOptions {
        abortOnError false
    }
    dexOptions {
        preDexLibraries = false
    }
}

repositories {
    maven { url rootProject.ext.mavenUrl }
```

Help improve Android Studio by sending usage s
Please click I agree if you want to help make Android Studio better or I don't agree otherwise...

Frameworks detected: Android framework is detected in the project Configure (a minute ago)

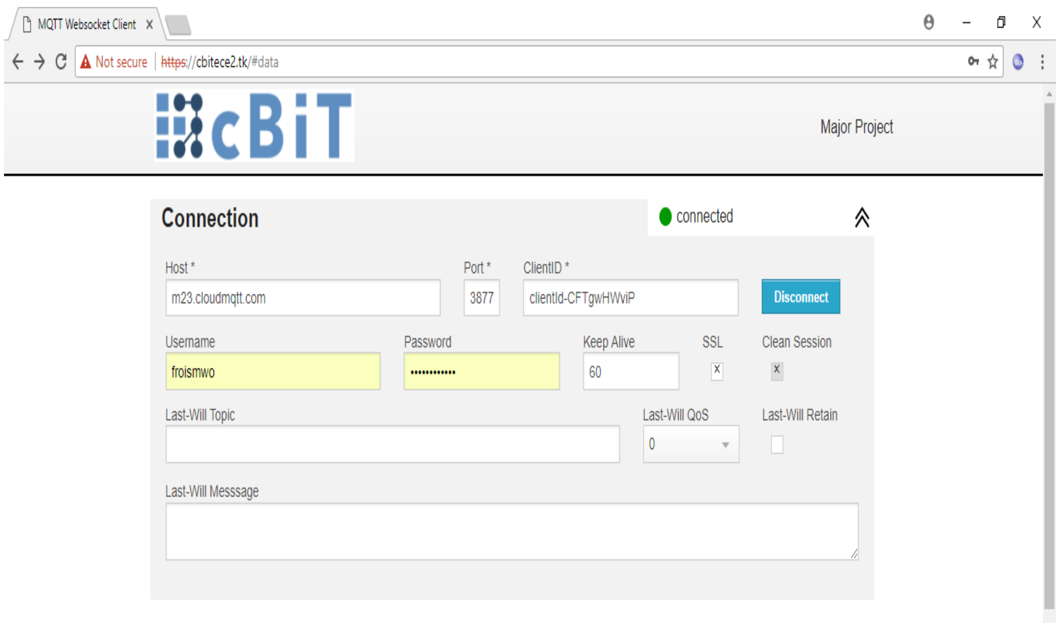
1:1 CRLF: UTF-8: Git: master

Fig 6.5: Paho client example application code on Android Studio

CHAPTER 7

RESULTS

7.1 WEBSITE



Host *: m23.cloudmqtt.com
Port *: 3877
ClientID *: clientId-CFTgwHWviP
Username: froismwo
Password: ••••••••••••
Keep Alive: 60
SSL: x
Clean Session: x
Last-Will Topic:
Last-Will QoS: 0
Last-Will Retain:
Last-Will Messsage:

Fig 7.1: Homepage of websocket client

This is the webpage of the site created by us so as to get connected to the NodeMCU via cloud MQTT Broker [19] .

The website designed by us can be divided into 4 major sections, they are

1.) Connection 2.) Publish 3.) Subscribe 4.) Messages

The first section (Connection) fulfils the function of connecting our websocket client to the cloud mqtt broker which is being used by us. Here there are certain input fields like username, password, host, port number, etc which are to be filled by the user.

The second section(Publish) contains the topic name and the QOS with which the message has to be delivered to the broker. There is also an option if we want the message to be retained.

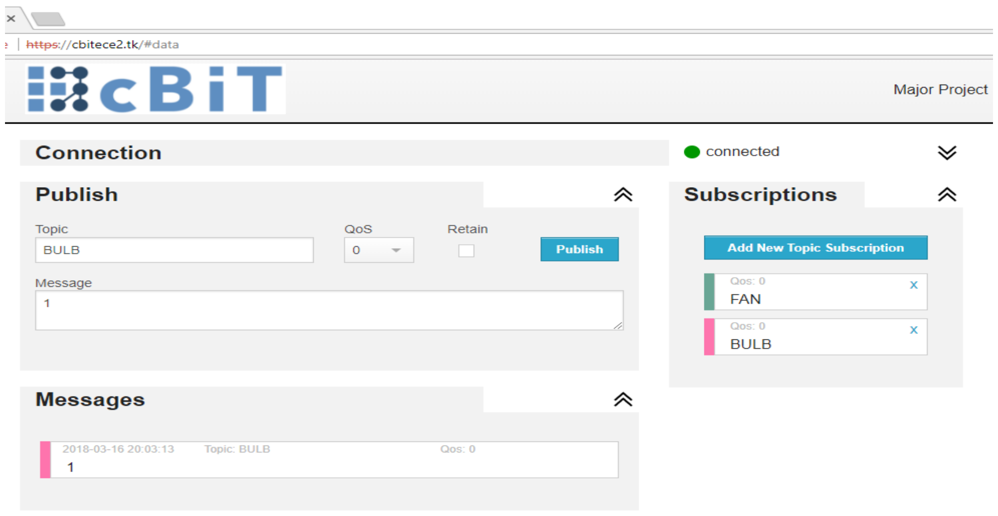
The third section (Subscribe) is used to Subscribe to any topic and will give the current message that is present in that topic.

The fourth section is the messages section and it shows all the messages which are received by the client from the time of its connection.

7.2 VERIFYING THE FUNCTIONALITY OF HOME AUTOMATION SYSTEM

7.2.1 CONTROLLING THE BULB

By subscribing to the topic “bulb” and publishing the message “1 “ in the website the bulb is switched ON and similarly by publishing the message “0 “ the bulb is switched OFF.


https://cbitece2.tk/#data

Major Project

Connection

connected

Publish

Topic: BULB

QoS: 0

Retain

Publish

Message: 1

Subscriptions

Add New Topic Subscription

Qos: 0 FAN

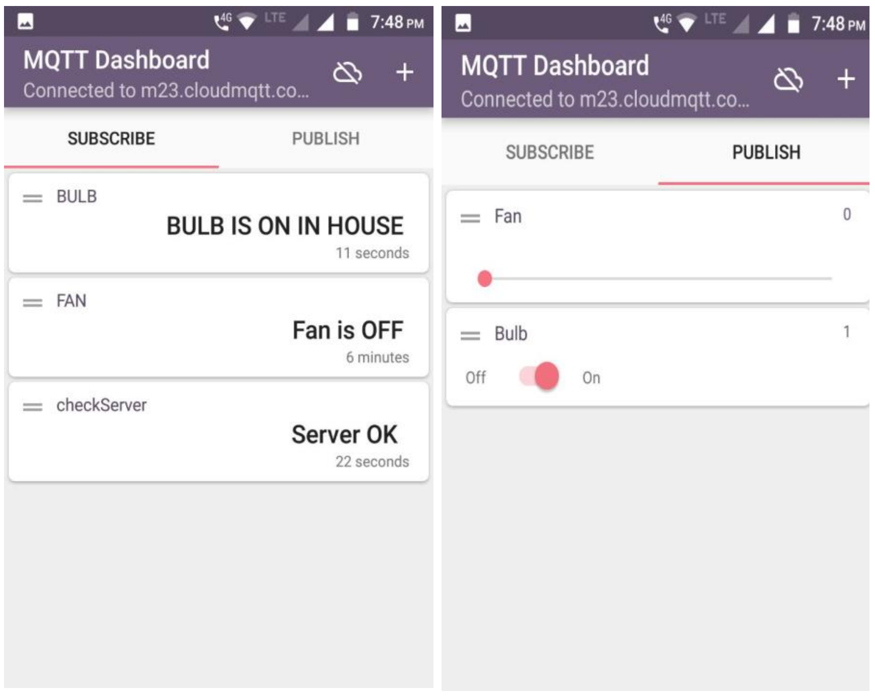
Qos: 0 BULB

Messages

2018-03-16 20:03:13 Topic: BULB Qos: 0

1

Fig7.2: Switching ON the Bulb from the website


7:48 PM

MQTT Dashboard

Connected to m23.cloudmqtt.co...

SUBSCRIBE PUBLISH

BULB — BULB IS ON IN HOUSE — 11 seconds

FAN — Fan is OFF — 6 minutes

checkServer — Server OK — 22 seconds

7:48 PM

MQTT Dashboard

Connected to m23.cloudmqtt.co...

SUBSCRIBE PUBLISH

Fan 0

Bulb 1

Off On

Fig 7.3: Switching ON the Bulb from the application

By subscribing to the topic bulb and by keeping the button in ON state in the application the bulb is switched on.

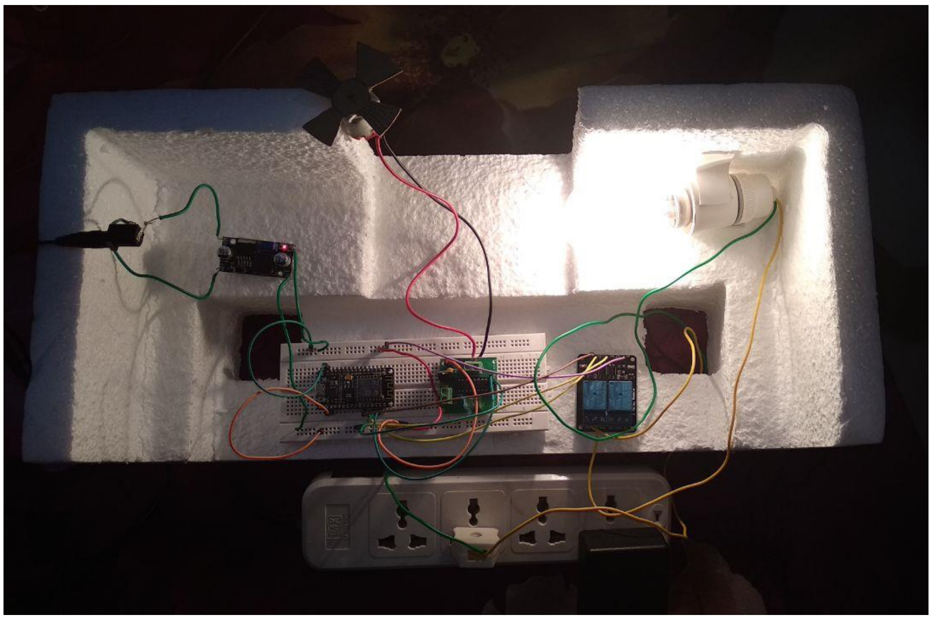
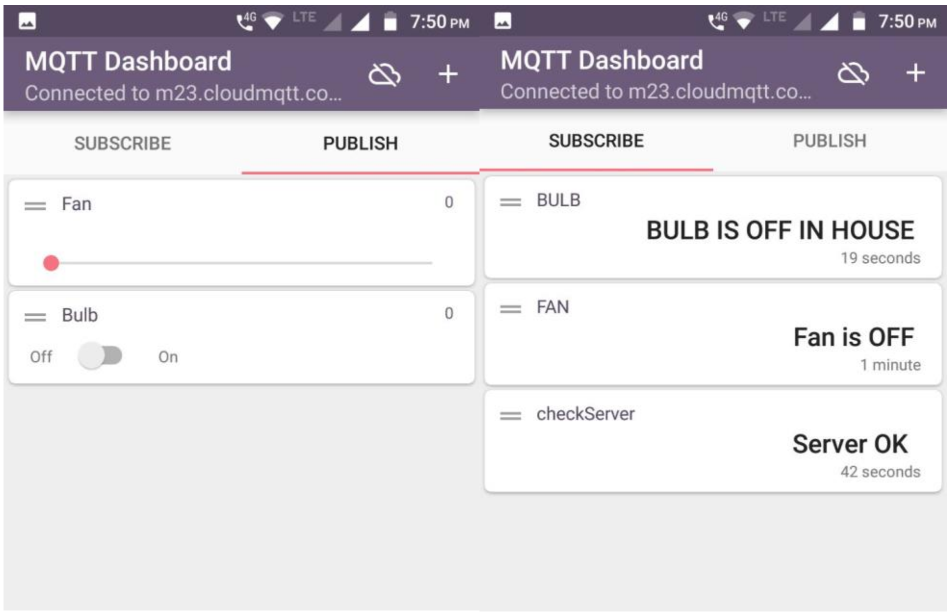


Fig 7.4 : Bulb switched ON

The command “ON “ sent from the website or application is received by the NodeMCU and the bulb is switched on.



7:50 PM

MQTT Dashboard

Connected to m23.cloudmqtt.co...

SUBSCRIBE PUBLISH

Fan 0

Bulb 0

Off On

7:50 PM

MQTT Dashboard

Connected to m23.cloudmqtt.co...

SUBSCRIBE PUBLISH

BULB

BULB IS OFF IN HOUSE

19 seconds

FAN

Fan is OFF

1 minute

checkServer

Server OK

42 seconds

Fig 7.5 :Switching OFF the bulb from the application

By subscribing to the topic bulb and by keeping the button in the application in OFF state, the bulb is switched OFF.

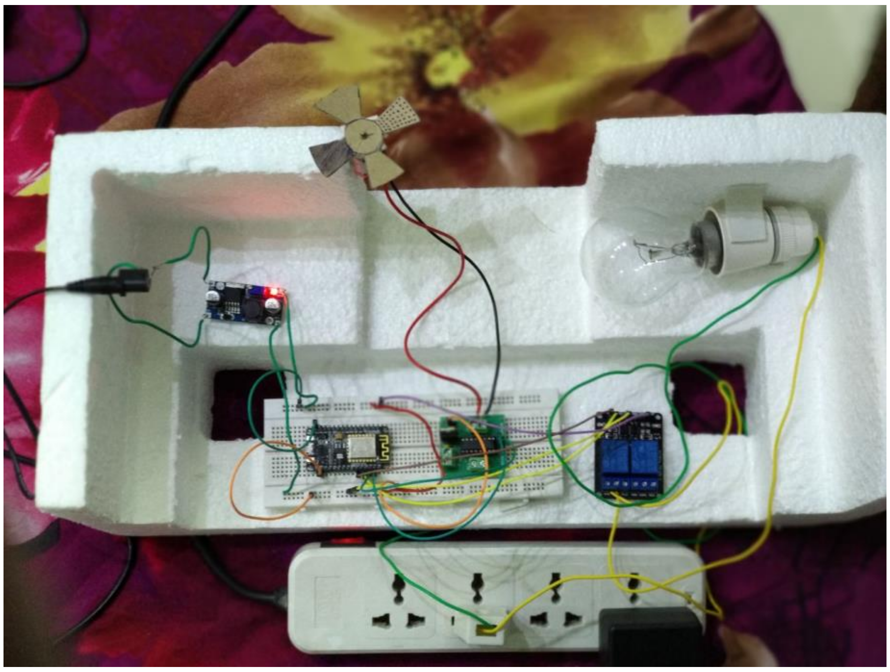
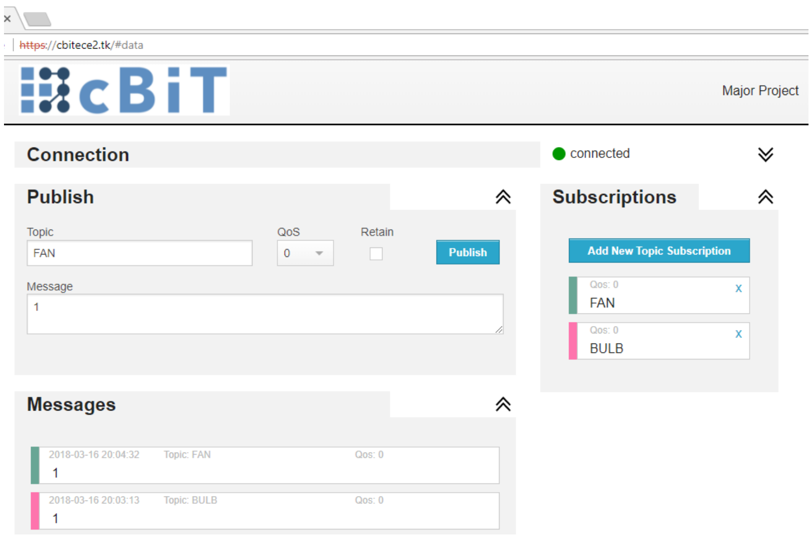


Fig 7.6: Bulb switched OFF

The command “OFF “ sent from the website or the application is received by the NodeMCU and the bulb is switched off.

7.2.2 CONTROLLING THE FAN



https://cbitece2.tk/#data

cBiT

Major Project

Connection

● connected

Publish

Topic: FAN

QoS: 0

Retain

Message: 1

Subscriptions

Add New Topic Subscription

Qos: 0 FAN

Qos: 0 BULB

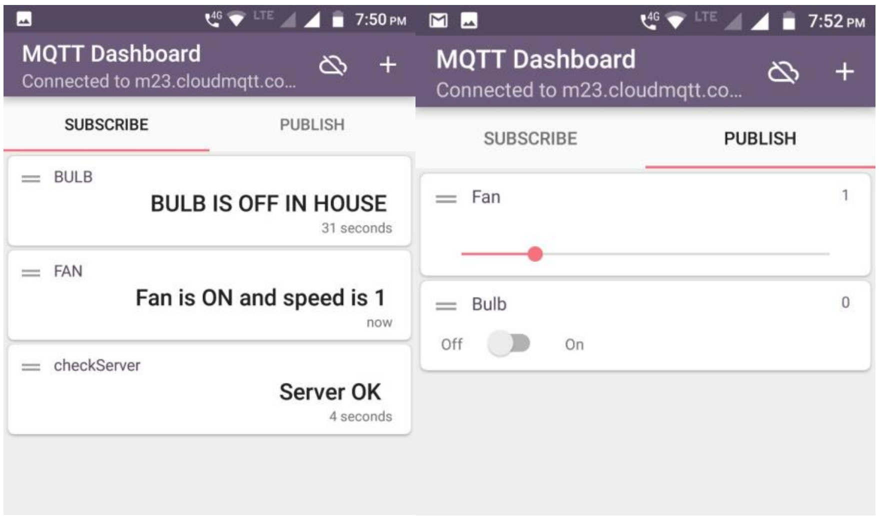
Messages

2018-03-16 20:04:32 Topic: FAN Qos: 0
1

2018-03-16 20:03:13 Topic: BULB Qos: 0
1

Fig 7.7: Switching on the fan from website.

By subscribing to the topic fan and by publishing message with values “1 “ to “ 5 “ in the website the fan is switched on and its speed is varied respectively and similarly by publishing the message “0 “ the fan is switched off.



BULB
BULB IS OFF IN HOUSE
31 seconds

FAN
Fan is ON and speed is 1
now

checkServer
Server OK
4 seconds

Fan 1

Bulb 0
Off On

Fig 7.8: Switching on the fan from the application

By subscribing to the topic "fan" and by varying the value published by using the slider in the application the fan is switched on and its speed is varied respectively.

Five various levels of speed have been provided.

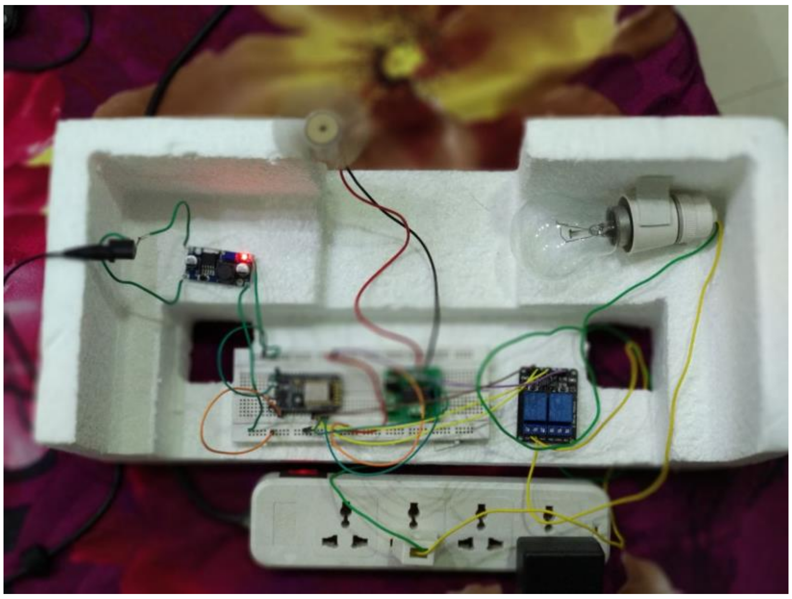


Fig 7.9: Fan switched on

The command "ON " sent from either the website or the application is received by the NodeMCU and the fan is turned on.

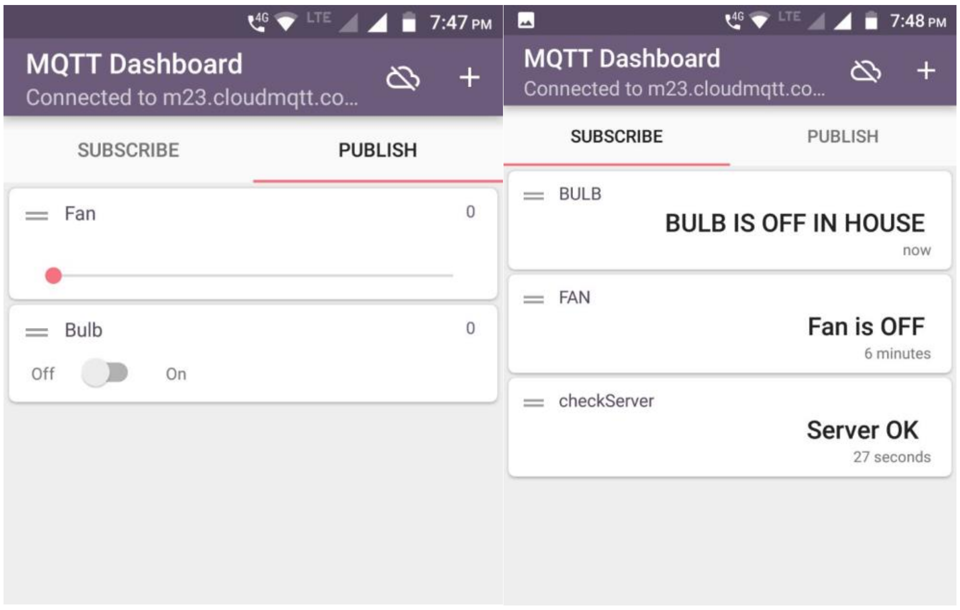


Fig 7.10: Switching off the fan from application

By subscribing to the topic fan and by keeping the slider in zero position in the application the fan is switched off.

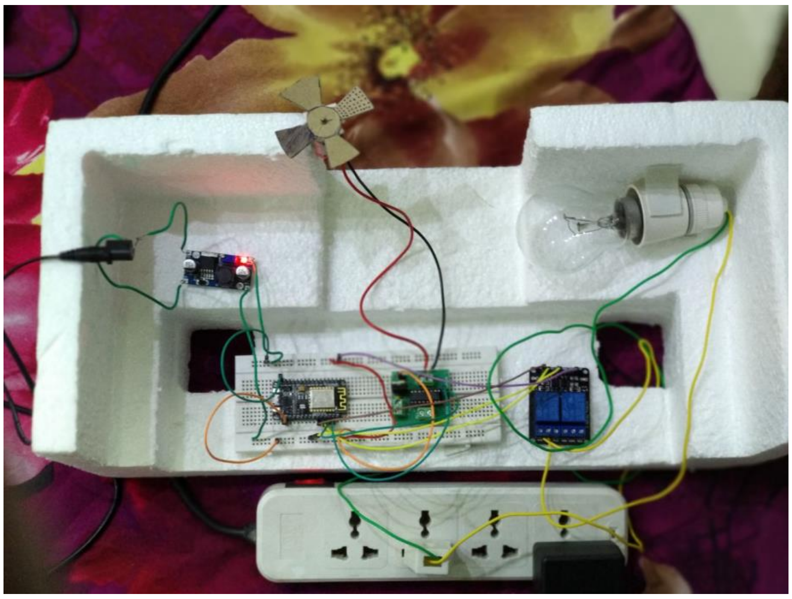


Fig 7.11: Fan switched off

The command “OFF “ sent from either the website or the application is received by the NodeMCU and the fan is switched off.

CONCLUSION

The field of IOT is rapidly advancing, many modern-day devices are being connected to the internet which enables the users to access them at their convenience. The ever-growing research in developing new algorithms, applications and protocols for resource constrained devices is a boon to the field of IoT. In our project we used NodeMCU which is very cost effective and is a highly scalable device. We have used modules like relays and motor drivers which aren't that costly either keeping the total cost required to implement the project at a minimum. The NodeMCU has unused pins which can be used in the future for addons making the device highly scalable.

The protocol which is being used by us is the MQTT protocol which is a light weight protocol and has many client service libraries readily available for NodeMCU. Instead of installing a broker service on a local server we have used a cloud server which will ensure that the server is always up and also eliminates the problem of NAT implemented by ISP. We have also used the paho client library developed by eclipse which helped us to create client applications on the mobile platform as well as web. PubSubClient has been used for making the NodeMCU a MQTT client. The websocket client is being hosted by firebase which is another cloud solution provided by google. Therefore integrating intelligent hardware like NodeMCU with existing protocols and cloud solutions provides a seamless access to our IoT system from anywhere in the world.

FUTURE SCOPE

In hardware perspective this smart IoT system can be extended using different sensor and actuator modules present in the industry. It can be used in wireless sensor networks where each NodeMCU will act as a node and collection of these individual nodes can be used to report various useful data using our cloud solution. It is to be noted that these nodes can be placed as far as possible from each other provided they are internet enabled.

Coming to the software perspective the data sent from the device can be accumulated in a database and can be fed to a analysing software which can be used to predict the future trends of the environment. The GUI of the website can be improved to provide a master dashboard to control our device.

REFERENCES

[1] Soni, Dipa & Makwana, Ashwin, A SURVEY ON MQTT: A PROTOCOL OF INTERNET OF THINGS(IOT), International Conference on Telecommunication, Power Analysis and Computing Techniques, 2017.

[2] R. K. Kodali and S. Soratkal, "MQTT based home automation system using ESP8266", IEEE Region 10 Humanitarian Technology Conference, 2016.

[3] Light, Roger A. (2017) Mosquitto: server and client implementation of the MQTT protocol. Journal of Open Source Software, 2017.

[4] https://www.onsemi.cn/PowerSolutions/document/LM2596-D.PDF

[5] https://nodemcu.readthedocs.io/en/master/

[6] http://www.nodemcu.com/index_en.html

[7] http://www.circuitbasics.com/wp-content/uploads/2015/11/SRD-05VDC SL-C-Datasheet.pdf

[8] http://modtronix.com/mod-rly2-5v.html

[9] https://www.elprocus.com/h-bridge-motor-control-circuit-using-l293d-ic/

[10] https://www.eclipse.org/paho/clients/js/

[11] http://mqtt.org/documentation

[12] https://www.mepits.com/tutorial/379/electrical/motor-driver

[13] https://www.hivemq.com/mqtt-essentials/

[14] https://www.cloudmqtt.com/docs.html

[15] https://firebase.google.com/

[16] https://github.com/hivemq/hivemq-mqtt-web-client

[17] https://github.com/eclipse/paho.mqtt.android

[18] https://pubsubclient.knolleary.net/api.html

[19] https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

[20] https://www.quora.com/What-is-firebase

[21] https://mosquitto.org/api/files/mosquitto-h.html

[22] https://www.arduino.cc

APPENDIX

LIST OF ABBREVATIONS

IOT -Internet of things

MCU - Micro Controller unit

MQTT - Message Queueing Telemetry Transport

IEEE -Institute of Electrical and Electronics Engineers

I2C -Inter Integrated communication

SPI -Serial Peripheral Interface

RAM – Random Access Memory

API -Application programming Interface

UART -Universal Asynchronous Receive and Transmit

PWM -Pulse Width Modulation

GPIO - General Purpose Input Output

ADC -Analog to Digital Converter

IDE - Integrated Development Environment

TCP - Transmission Control Protocol

SDK - Software Development Kit

VPP - Virtual Power Plant

TLS -Transport Layer Security

SSL- Secured Socket Layer

SSID- Service Set Identifier

UDP- User Datagram Protocol

CLI- Command Line Interface

CDN - Content Delivery Network

QOS- Quality of Service