

Design, Development and Testing of IoT Gateway

Major Project Report

*Submitted in partial fulfillment of the requirements
for the degree of*

**Master of Technology
in
Electronics & Communication Engineering
(Communication Engineering)**

By

**Kartik Chavda
(17MECC17)**



Department of Electronics & Communication Engineering

Institute of Technology

Nirma University

Ahmedabad-382 481

May 2019

Design, Development and Testing of IoT Gateway

Major Project Report

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology

in

Electronics & Communication Engineering

(Communication Engineering)

By

Kartik Chavda

(17MECC17)

Under the guidance of

External Project Guide:

Mr. Sandip Patel,
Product Manager,
Masibus Automation
Pvt. Ltd.,
Gandhinagar

Internal Project Guide:

Dr. Sachin Gajjar,
Associate Professor
EC Engineering,
Institute of Technology,
Nirma University,
Ahmedabad



Department of Electronics & Communication Engineering

Institute of Technology

Nirma University

Ahmedabad-382 481

May 2019

Declaration

This is to certify that

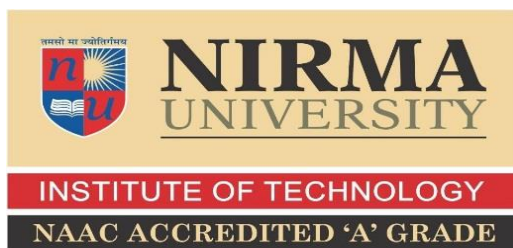
1. The thesis comprises my original work towards the degree of Master of Technology in Communication Engineering at Nirma University and Physical Research Laboratory and has not been submitted elsewhere for a degree.
2. Due acknowledgment has been made in the text to all other material used.

Kartik Chavda

17MECC17

Disclaimer

“The content of this thesis does not represent the technology, opinions, beliefs, or positions of Masibus Automation Pvt. Ltd. its employees, vendors, customers, or associates”.



Certificate

This is to certify that the Major Project entitled “**Design, Development and Testing of IOT Gateway**” submitted by **Kartik Chavda (17MECC17)**, towards the partial fulfillment of the requirements for the degree of Masters of Technology in Communication Engineering, Nirma University, Ahmedabad is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this Project, to the best of our knowledge, haven’t been submitted to any other university or institution for award of any degree or diploma.

Date:

Place: Ahmedabad

Dr. Sachin Gajjar

Internal Guide,
Associate Professor in EC Engineering,
Institute of Technology,
Nirma University, Ahmedabad

Dr. Y. N. Trivedi

Program coordinator,
Professor in EC Engineering,
Institute of Technology,
Nirma University, Ahmedabad

Dr. D. K. Kothari

Professor and Head,
EC Engineering Department,
Institute of Technology,
Nirma University, Ahmedabad

Dr. Alka Mahajan

Director,
Institute of Technology,
Nirma University, Ahmedabad



Certificate

This is to certify that the Major Project entitled “**Design, Development and Testing of IOT Gateway**” submitted by **Kartik Chavda (17MECC17)**, towards the partial fulfillment of the requirements for the degree of Masters of Technology in Communication Engineering, Nirma University, Ahmedabad is the record of work carried out by him under my supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this Project, to the best of our knowledge, haven’t been submitted to any other university or institution for award of any degree or diploma.

Date:

Place: Gandhinagar

Mr. Sandip Patel

Product Manager,

Masibus Automation Pvt. Ltd

Acknowledgement

I would like to express my gratitude and sincere thanks to **Dr. Y.N. Trivedi**, PG Coordinator of M.Tech Communication and **Dr. Sachin Gajjar** for guide-lines during the review process.

I take this opportunity to express my profound gratitude and deep regards to **Dr. Sachin Gajjar**, guide of my internship project for his exemplary guidance, monitoring and constant encouragement.

I would also like to thank **Mr. Sandip Patel**, external guide of my internship project from Masibus Automation Pvt. Ltd., for guidance, monitoring and encouragement regarding the project.

- **Kartik Chavda**

17MECC17

Contents

Declaration	iii
Disclaimer	iv
Certificate	v
Certificate	vi
Acknowledgement	vii
Abstract	x
List of Tables	xi
List of Figures	xii
Abbreviation	xiv

1. Introduction	1
1.1 Motivation	2
1.2 Problem Statement	2
1.3 Approach	2
1.4 Scope of Work	3
1.5 Organization of Thesis	3
2. Literature Survey	4
2.1 IOT Gateway	4
2.2 Application Layer Protocol	5
2.2.1 Hypertext Transfer Protocol (HTTP)	5
2.2.2 Message Queuing Telemetry Transport (MQTT)	6
2.2.3 Constrained Application Protocol (CoAP)	7
2.3 Comparison of HTTP, MQTT, CoAP	9
2.4 Comparison of NodeMCU and ESP8266	10
2.5 Programming Platform	11
2.5.1 Arduino IDE	11
2.5.2 Mongoose OS	11
2.6 Cloud Platform	12
2.6.1 Google Firebase	12
2.6.2 Amazon Web Services (AWS)	12
2.7 Mobile Application	14
2.7.1 Blynk	14
2.7.2 IoT MQTT Dashboard	14
2.8 MQTTLens Workstation for IoT	14
2.9 Eclipse MQTT Broker and Client	14
2.9.1 Mosquitto Server	14
2.9.2 Paho Mqtt client	15

3. Hardware Design	16
3.1 Hardware Block Diagram	16
3.2 IoT development hardware with ESP8266	21
3.3 Temperature monitoring system	22
4. Software Design	25
4.1 Arduino IDE for ESP8266	25
4.1.1 Install ESP8266 Board support package	25
4.1.2 ESP8266 and Blynk App communication	25
4.1.3 ESP8266 remote access from Google Firebase	26
4.2 ESP8266 and AWS IoT communication	29
4.3 Raspberry Pi remote access with IoT MQTT dashboard	33
4.4 Raspberry Pi remote access with MQTT Lens Workstation	36
4.5 Raspberry Pi remote access with AWS IoT	38
5. Testing /Results	40
5.1 IoT Development Hardware	40
5.1.1 IoT development hardware with ESP8266	40
5.1.2 Raspberry Pi IoT development kit	41
5.2 Monitor and control on android Mobile Application	42
5.2.1 Mobile App with MIT APP Inventor	42
5.2.2 Blynk Mobile Application	43
5.2.3 Remote access with MQTT Dashboard	44
5.3 Monitor and control on Firebase Cloud	44
5.3.1 DHT Sensor Real time database	44
5.3.2 IoT development hardware data on firebase	45
5.4 Monitor and control on AWS IoT	46
5.4.1 ESP8266 remotely access through AWS IOT	46
5.4.2 DHT 11 data share on AWS IoT	47
5.4.3 Raspberry pi remotely access through AWS IoT	47
5.5 Monitor and control through Workstation	48
5.5.1 Remote access with MQTTLens	48
6. Conclusion and Future Scope	49
Bibliography	50

Abstract

Internet of Things (IoT) is the most advancing technology worldwide. IoT is the technology which can connect and control physical devices remotely with the help of Internet. IoT architectural layers are sensing layer, networking layer, data processing layer and application layer. The architecture components of IoT are sensors, actuators, gateways, processing units (microcontroller or microprocessors) and cloud server. IoT gateway can be used to provide Internet connectivity to any non-IP devices. IoT gateway are connection points between devices connected using personal area network standards and the Internet. In this project, a gateway for IoT is designed, developed and tested. A hardware setup consisting of IoT gateway, end-device consisting of sensor for environmental monitoring and an end-device consisting of actuator is developed. The sensor data is send to cloud services like Google firebase, Amazon Web Services (AWS) for monitoring on mobile and on workstation. The gateway is developed using ESP8266 having Wi-Fi connectivity and Raspberry Pi development boards having both Bluetooth and Wi-Fi connectivity. Message Queuing Telemetry Transport (MQTT) is a lightweight, low-bandwidth, high-latency application layer protocol. MQTT protocol uses the main two things, Broker and Client. Eclipse mosquito is open MQTT broker and Eclipse paho is an MQTT client that can be used on a number of hardware platforms. The operating system used for the ESP8266 gateway is Mongoose Operating System (OS) and for Raspberry Pi (RPi) based gateway Raspbian stretch is used for the gateway developed. AWS and Google firebase are used as the cloud service for the gateway that is designed in the project. Device gateways share their data to the various clouds like Google firebase, Amazon AWS IoT. Mobile Application like Blynk and MQTT dashboard is used for mobile and workstation based remote control and monitoring of the devices. A customized Mobile application is also developed with MIT App Inventor for device control and monitoring. MQTT lens used is used for monitoring and control of devices through workstation. The entire hardware setup is successfully tested.

List of Tables

2.1	Comparison of HTTP, MQTT and CoAP	9
2.2	Comparison of NodeMCU and Arduino UNO.....	10
3.1	NodeMCU Board Specification	18
3.2	Relay connection with NodeMCU	19
3.3	DHT Sensor Connection with NodeMCU	20

List of Figures

1.1	IoT Architecture.	1
2.1	IoT Gateway Architecture.	4
2.2	HTTP Protocol.	6
2.3	MQTT Protocol	7
2.4	CoAP protocol Architecture	8
3.1	Hardware Block diagram	16
3.2	ESP8266 12E Wi-Fi Development Board	17
3.3	Relay Circuit	19
3.4	DHT11 Sensor	20
3.5	IoT development hardware with ESP8266	21
3.6	Raspberry Pi 3b+ Board.	23
3.7	Temperature monitoring system with Raspberry Pi.	24
4.1	Create project on firebase	26
4.2	Security rule for Cloud Firebase.	27
4.3	Firebase Host URL.	28
4.4	Firebase database Secret key.	28
4.5	AWS CLI configuration.	29
4.6	Mongoose OS git clone.	30
4.7	Navigate the mongoose OS directory.	30
4.8	Mongoose OS build the firmware for ESP8266	31
4.9	Mongoose OS flash on ESP8266	31
4.10	NodeMCU Wi-Fi configuration.	32
4.11	Mongoose OS upload certificates on AWS IoT	32
4.12	Create connection on IoT MQTT dashboard	34
4.13	IoT MQTT dashboard components.	35
4.14	Publish message with IoT MQTT Dashboard.	35
4.15	Create connection in MQTTLens	37
4.16	Publish message with MQTTLens	37

4.17	DHT 11 publish data on AWS IoT	39
5.1	IoT development hardware with ESP8266	40
5.2	Raspberry Pi IoT development kit.	41
5.3	Mobile App MIT APP Inventor.	42
5.4	Blynk App Dashboard.	43
5.5	Remote access with IoT MQTT Dashboard.	44
5.6	DHT 11 data on Firebase.	44
5.7	IoT development hardware data on firebase.	45
5.8	ESP8266 GPIO remotely access by AWS IoT	46
5.9	Temperature data publish on AWS IoT	47
5.10	Raspberry Pi GPIO remote access by AWS IoT	47
5.11	Remote access with MQTTLens	48

Abbreviation

M2M	Machine to Machine
IOT	Internet of Things
MQTT	Message Queuing Telemetry Transport
HTTP	Hypertext Transfer Protocol
CoAP	Constrained Application Protocol
GPIO	General Purpose Input Output
SPI	Serial Peripheral Interface
I2C	Inter-Integrated Circuit
CPU	Central Processing Unit
RAM	Random Access Memory
I2S	Inter-IC Sound
RISC	Reduced Instruction Set Computer
UART	Universal Asynchronous Receiver/Transmitter
AWS	Amazon Web Service
TLS	Transport Layer Security
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
IDE	Integrated development environment
Rpi	Raspberry Pi
OS	Operating System

DHT	Digital Humidity and Temperature
CLI	Command Line Interface
DTLS	Datagram Transport Layer Security
IAM	Identity and Access Management
USB	Universal Serial Bus

Chapter 1

Introduction

International Telecommunication Union defines IoT as “A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving, interoperable information and communication technologies”. IoT systems have smart devices that are used to collect, control or send data from embedded processors, sensors and communication hardware environment. IoT devices collect the data by connecting to the IoT gateway and send them to the cloud to store, control, send and remotely access from mobile or workstation kind of devices.

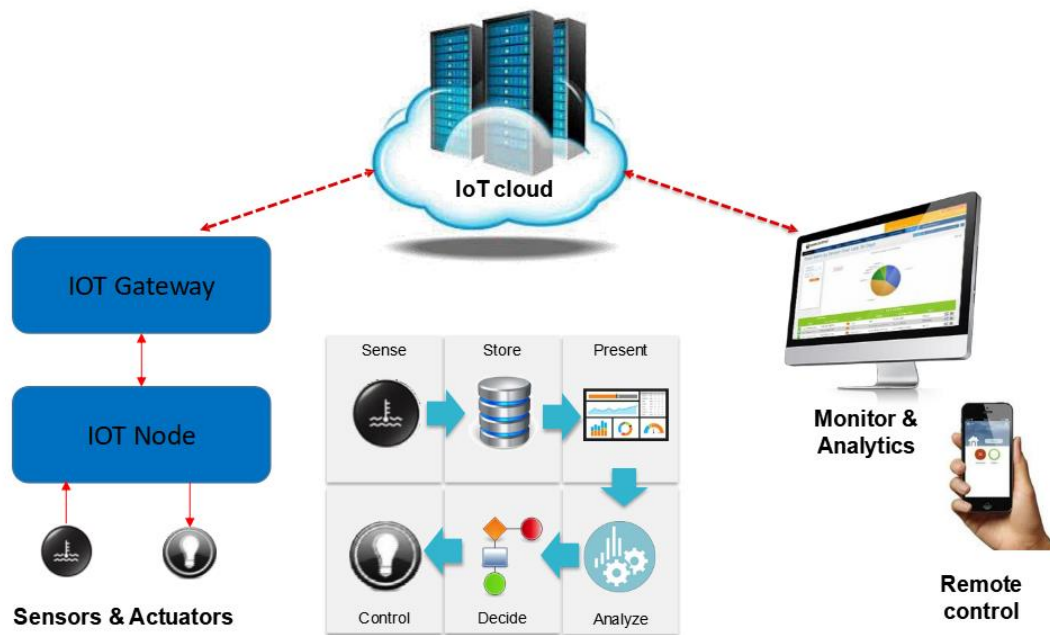


Fig. 1.1 IoT Architecture [1]

Fig 1.1 Shows Architecture of IoT, IoT Nodes are the microcontrollers which collect the data from sensor and can control the devices using relay like devices. It has capabilities like sufficient processing power and memory. IoT gateway is bridge between Internet and Local area network through which the nodes are connected. These nodes are connected using standards like WiFi, Zigbee, Bluetooth, Cellular, Lora WAN, Ethernet, etc. HTTP, MQTT, CoAP, and AMQP are the application layer protocols used in the IoT.

1.1 Motivation

IoT is the most growing and upcoming Technology. IoT main architectural components are things, gateways, mobile devices, the cloud and the enterprise. Gateway is most crucial part for the IoT Technology. For the smart devices design very cost-effective gateway which can connect home appliance to the internet and remotely access them from anywhere in the world.

1.2 Problem Statement

The thesis aims to design, develop and test an IoT gateway that is cost effective, consumes less power and is easy to use.

1.3 Approach

Main Approach for this project is to use ESP8266 and RPi used as device gateway to connect the various devices to internal network at one end and Internet on the other. Mosquitto MQTT broker and Paho MQTT client are used for communication with light weight MQTT protocol. ESP8266 communicates with Server or Broker through internet with MQTT protocol. Google Firebase and Amazon Web Service are used as cloud broker which have many computing capability to monitor and access the ESP8266 data over the internet. To remotely monitor and control the nodes through the mobile phone a mobile application is developed using MIT app Inventor 2 and Google firebase.

1.4 Scope of Work

IoT architectural layers are sensing layer, networking layer, data processing layer and application layer. The architecture components of IoT are sensors, actuators, gateways, processing units (microcontroller or microprocessors) and cloud server. DHT11 sensor and Relays are part of sensing layer. ESP8266, Rpi and Wi-Fi routers are part of networking layer. Google firebase and AWS IoT are part of data processing layer. Blynk, MQTT Dashboard, Mobile Application are part of Application Layer.

1.5 Organization of the thesis

The rest of the thesis is organized as follows:

Chapter 2 describes the Literature review.

Chapter 3 describes the hardware design of the gateway.

Chapter 4 describes about Software Design.

Chapter 5 presents the Testing and Results.

Chapter 2

Literature Survey

2.1 IoT Gateway

IoT gateway [2], bridging communication between sensor domain and network domain, is the most important components of the IoT. Fig 2.1 gives details of IoT gateway [2] in the IoT infrastructure. The IoT gateway acts as Proxy for the sensing domain and network domain towards the ‘thing’ that are connected to it.

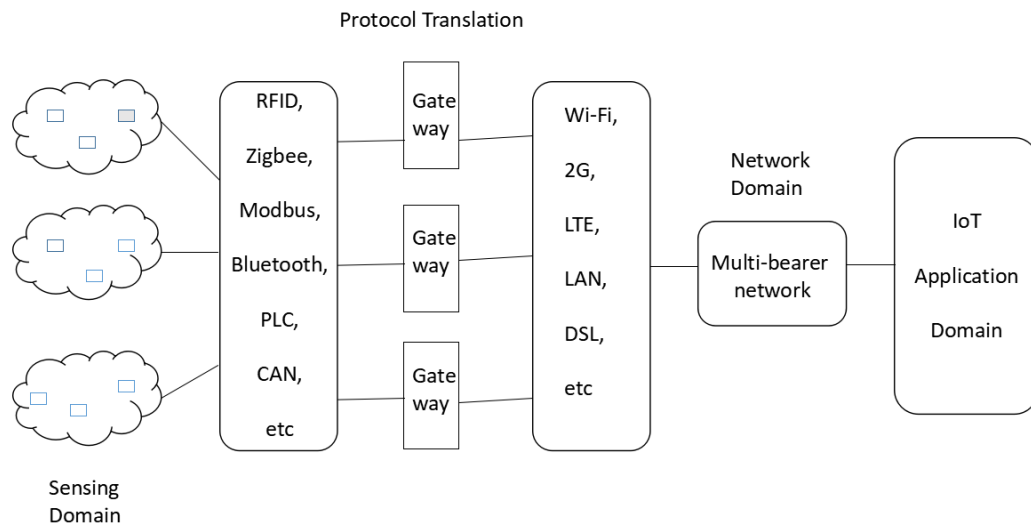


Fig. 2.1 IoT Gateway Architecture [2]

IoT gateways are quite different from application to application for various requirements. The common features are multiple interface, protocol conversion and Manageability. Smart things are connect to IoT gateway through various kind of technologies (Wi-Fi, Bluetooth, Zigbee, etc.), also IoT gateway has kind of choices to connect to the public network (2G/3G, LTE, LAN, DSL, PSTN, etc.). There are two situation that an IoT gateway needs execution protocol conversion. One is communication occurs between different sensing domain protocols (e.g. Wi-Fi and Zigbee), the other is that communication occurs between sensing domain protocol and network domain protocol

(e.g. between Zigbee and LTE). IoT gateway itself needs managed IoT servers, like subscription management, authority management, status management and mobility management, etc. Smart things attached to an IoT gateway also needs to be managed by IoT gateway. IoT gateway may have many abilities to identify, control, diagnose configure and maintain the smart things.

2.2 Application Layer Protocols

2.2.1 Hypertext Transfer Protocol (HTTP)

The Hypertext Transfer Protocol (HTTP) [3] was founded in 1990 for the data communication purpose of the World Wide Application. It was published as standard protocol in 1997 by IETF and W3F. HTTP is an application-level protocol for widely used for access multimedia content on web pages. HTTP is a request-respond TCP/IP protocol. HTTP is mostly used for transfer multimedia files. Port 80 is the default HTTP Port. It is mostly used to access web pages on the Internet. HTTP protocol is developed for transfer multimedia contents so message header size is maximum. Once data transfer overs, client needs to initiate a request for new connection.

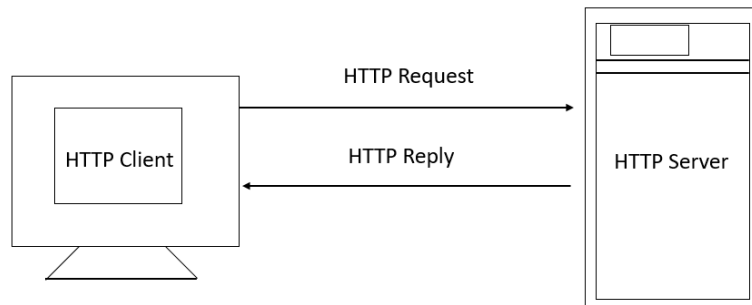


Fig. 2.2. HTTP Protocol

Fig. 2.2 shows the request-response model of HTTP protocol. Web browser (clients) initiate a request to connects to specific web pages. The web pages are managed by the Servers. Server grant access to the client and send the acknowledgement. Once connection successfully setup then data transfers.

Once data transfer is over connection is terminate each new request are treated as new connection for the client and server are new to each other. Any type of the data can be sent through HTTP can be handled data content by the client and server by each session. So, client and browser (server) cannot retain information data at different request for web pages.

2.2.2 Message Queuing Telemetry Transport Protocol (MQTT)

Message Queue Telemetry Transport (MQTT) [3] is the messaging protocol. It was developed by Andy Stanford-Clark of IBM and Arlen Nipper of Arcom in 1999. MQTT protocol is developed for machine to machine communication. Afterwards this protocol is popular for the messaging application. It is the most popular protocol used in IoT. MQTT Brokers and MQTT clients are the two main parts for this Protocol. MQTT Brokers are known as Servers used for store the data and manage clients. MQTT clients can publish or subscribe the data from MQTT Broker.

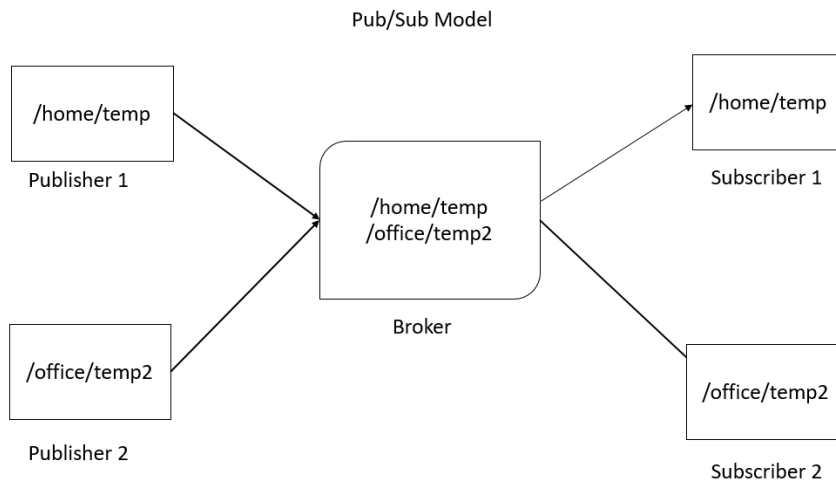


Fig. 2.3 MQTT Protocol

Fig. 2.3 shows MQTT Protocol consists of three main components: subscriber, publisher and broker. The publisher generates the data and transmits the information to subscribers through the broker. The broker ensures security by cross-checking the authorization of publishers and subscribers.

MQTT Protocol provides efficient information-routing functions to small, cheap, low-memory and power-consuming devices in vulnerable and low bandwidth based networks. MQTT protocols features are: It is light-weight message queuing and transport protocol. It is suitable for Asynchronous communication model with messages (events), It has Low overhead (2 bytes header) for low network bandwidth applications, It works on Publish / Subscribe (Pub Sub) model and knowledge of information publisher and subscriber through topics.

Advantages of MQTT protocol are: It is Simple protocol and aimed at low quality. It is low power and low footprint implementations (e.g. WSN – Wireless sensor Networks). It runs on connection-oriented transport (TCP). To be used in conjunction with 6LoWPAN (TCP header compression).

2.2.3 Constrained Application Protocol (CoAP)

CoAP derived by the IETF Constrained RESTful Environments (CoRE) working group. CoAP[3] is an Internet application protocol for constrained devices. It is designed to be used between devices on the same constrained network and general nodes on the Internet. This protocol is especially designed for IoT systems based on HTTP protocols.

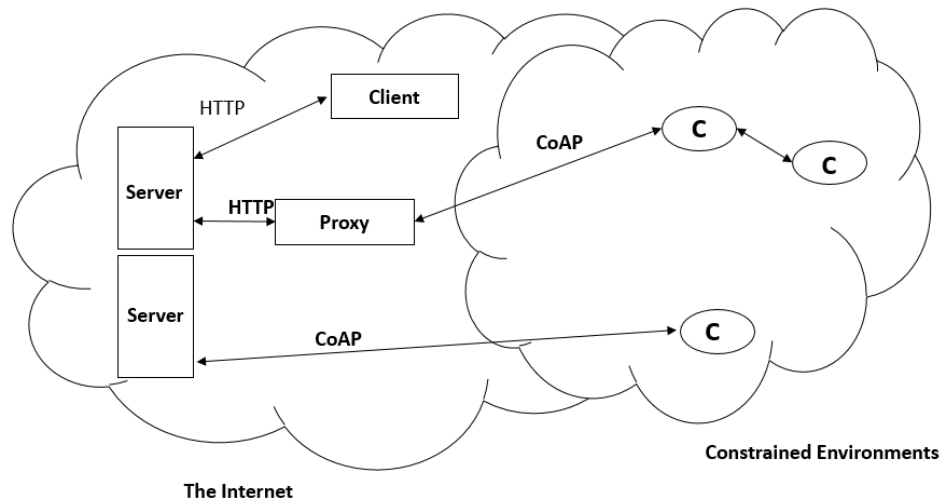


Fig. 2.4 CoAP protocol Architecture[3]

Fig. 2.4 shows, CoAP protocol architecture. CoAP makes use of the UDP protocol for lightweight implementation. It also makes use of RESTful architecture, which is very similar to the HTTP protocol. It is used within mobiles and social network based applications and eliminates uncertainty by using the HTTP get, post, put and delete methods. Apart from communicating IoT data, CoAP has been developed along with DTLS for the secure exchange of messages. It uses DTLS for the secure transfer of data in the transport layer.

CoAP protocol [\[3\]](#) Features are CoAP uses UDP, some of the TCP functions are reproduced in CoAP. For example, CoAP distinguishes between confirmable (requiring an acknowledgement) and non-confirmable messages. Requests and responses are exchanged asynchronously over CoAP messages. All the headers, methods and status codes are binary encoded, which reduces the protocol overhead. Unlike HTTP, the ability to cache CoAP responses does not depend on the request method, but the Response Code. CoAP fully addresses the need for an extremely lightweight protocol and the ability for a permanent connection.

2.3 Comparison HTTP, MQTT and CoAP

Table 2.1 Comparison of HTTP, MQTT and CoAP [3]

Details	MQTT	CoAP	HTTP
Year	1999	2010	1997
Architecture	Client/Broker	Client/Server or Client/Broker	Client/Server
Header Size	2 Byte	4 Byte	Undefined
Methods	Connect, Disconnect, Publish, Subscribe, Unsubscribe, Close	Get, Post, Put, Delete	Get, Post, Head, Put, Patch, Options, Connect, Delete
Quality of Service	QoS 0, QoS 1, QoS 2	At most once, At least-once	Not Available
Message Size	Small	Small	Large
Cache and Proxy	Partial	Yes	Yes
Transport Protocol	TCP	UDP, SCTP	TCP
Security	TLS/SSL	DTLS, IPsec	TLS/SSL
Licensing	Open Source	Open Source	Free
Default Port	1883/8883(TLS/SSL)	5683 (UDP) /5684 (DLTS)	80 / 443 (TLS/SSL)
Standards	OASIS	IETF	W3C and IETF
Organizational Support	IBM, Facebook, Eurotech, Cisco, Red Hat, AWS	Cisco, Contiki, Enka, IoTivity	Global Web Protocol Standard
Official site	https://www.w3.org/ Protocols/	mqtt.org/	https://coap.technolo gy/

Table 2.1 shows the comparison of Application layer protocols MQTT, HTTP and CoAP. HTTP is widely used for the multimedia data transfer. MQTT is the most popular protocol used for IoT because it has lowest message header size and light protocol.

2.4 Comparison of NodeMCU and ESP8266

Table 2.2 Comparison of NodeMCU and Arduino UNO

Parameters	ESP8266 12 E(NodeMCU)	Arduino UNO
CPU	32 Bit	8 Bit
Microcontroller	LX 106	Atmega 328P
Processor Clock	80MHz to 160MHz	16 MHz
RAM	36 KB	8 KB
Storage	16 MB	32 KB
Built in Wi-Fi	IEEE 802.11 b/g/n	N.A.
ADC Pin	1 (10 bit Resolution)	6
GPIO Pins	12	14
Operating Voltage	3 – 3.6V	7 – 12 V

Table 2.2 shows, comparison of NodeMCU and Arduino IDE board. Both board supports programming with Arduino IDE. NodeMCU board supports programming with ESPlore IDE, mongoose OS and NodeMCU Firmware. NodeMCU board have Wi-Fi chip available, high processor speed and more RAM space available compared to Arduino UNO.

2.5 Programming Platform:

2.5.1 Arduino IDE

The Arduino integrated development environment (IDE) is a cross-platform application support mac os, windows and Linux environment that is written in the programming language Java. Arduino[4] Sketch is used to write and upload programs to Arduino board. ESP8266 community provides board support packages for esp8266 board. In this Package some examples like Wi-Fi client, Wi-Fi Access points, http server, DNS Server, etc are available. In this project different libraryies viz. Arduino JSON, Blynk and firebase, etc. are used for communication with various clouds.

2.5.2 Mongoose OS:

Mongoose OS for microcontrollers was developed by Cesanta, a Dublin-based embedded software company and Advanced APN Technology Partner[5]. Mongoose OS supports many microcontrollers like STMicro: STM32 F4, L4, F7, TI: CC3200, CC3220, Espressif: ESP32, ESP8266. Mongoose OS is an open source operating system for microcontrollers that emphasizes cloud connectivity. Mongoose OS[5] Features are Reliable Over-The-Air update, secure device provisioning, and Remote management. It supports a simple networking interface for controlling devices remotely over RESTful, Web socket, or MQTT protocols. It Supports Cloud integrations like AWS IoT, Google IoT, Microsoft Azure, IBM Watson, Private MQTT / REST back end. It can be programmed using C and JavaScript. Mongoose OS provides proxy gateways for ESP8266 to connect to AWS IoT Platform.

2.6 Cloud Platform

2.6.1 Google Firebase:

Google Firebase[6] is a mobile and web app development platform that provides developers with set of tools and services for developing high-quality apps. Firebase provide different Services like Real time Database, Authentication, Test Lab, Crashlytics, Cloud Functions, Firestore, Cloud Storage, Performance Monitoring, Crash Reporting, Hosting, Firebase Analytics, Invites, Cloud Messaging, Predictions, AdMob, Dynamic Links, AdWords, Remote Configuration.

In this project, the Real time Database features is used, The Firebase[6] Real time Database is a cloud-hosted service. SQL database does not allow to store and sync between the users in real-time. The Real time Database is a JSON object that the developers can manage in real-time. With a single API, the Firebase database provides the app with both the current value of the data and any updates to that data. Real time syncing makes it easy for users to access their data from any device, on the web as well as mobile. Real time Database also helps users to collaborate with each another. Another benefit of Real time Database is that it ships with mobile and web SDKs, allowing one to build apps without the need for servers. When users go offline, the Real time Database SDKs use local cache on the device to serve and store changes. When the device comes online, the local data is automatically synchronized. The Real time Database can also be integrated with Firebase Authentication to provide a simple and intuitive authentication process.

2.6.2 Amazon Web Services (AWS)

AWS provides a wide variety of services which are cloud-based, such as compute, analytics, IoT, security, and storage. In this project, the following services of AWS[7] are being used. Amazon Web Services[7] offers a broad set of global cloud-based services including compute, storage, databases, analytics, networking, mobile, developer tools, management tools, IoT, security and enterprise applications. These services help to build up projects that are faster, lower in IT costs, and scalable. AWS is trusted universally to wide variety of workloads including: web and mobile applications, game development, data processing and warehousing, storage, archive, and many others. AWS is Easy to use,

Flexible, Cost-Effective, And Reliable, Scalable provides high-performance and is secure. In this project, the following services[\[7\]](#) of AWS are being used.

1. AWS CLI:

The AWS Command Line Interface (CLI)[\[7\]](#) is a unified tool to manage the AWS services. One tool is to be configured, to control multiple AWS services from the command line and automate them through scripts.

2. AWS IAM:

AWS Identity and Access Management (IAM)[\[7\]](#) is a web service for securely controlling access to AWS services. With IAM, it is possible to centrally manage users, security credentials such as access keys, and permissions that control which AWS resources users and applications can access

3. AWS IoT:

AWS IoT is a managed cloud platform that lets connected devices easily and securely interact with cloud applications and other devices. AWS IoT[\[7\]](#) can support billions of devices and trillions of messages, and can process and route those messages to AWS endpoints and to other devices reliably and securely. With AWS IoT, applications can keep track of and communicate with all devices, all the time, even when they aren't connected. AWS IoT makes it easy to use AWS services like AWS Lambda, Amazon Kinesis, Amazon S3, Amazon Machine Learning, and Amazon DynamoDB to build Internet of Things (IoT) applications that gather, process, analyze and act on data generated by connected devices, without having to manage any infrastructure.

2.7 Mobile Application for IoT:

2.7.1 Blynk

Blynk[\[9\]](#) is a mobile application that can control hardware remotely, it can display sensor data, it can store data, visualize it. Blynk can be run on Platform with iOS and Android operating system to control Arduino, Raspberry Pi, NodeMCU and several other boards over the Internet.

2.7.2 IoT MQTT Dashboard:

IoT MQTT Dashboard is an application to manage IoT projects using MQTT protocol. The Application features are: It supports many connections, SSL connection. It specify the data unit and display numeric values in a real-time updating chart.

2.8 MQTTLens Workstation for IoT:

MQTTLens is the Google Chrome application extension available on Google chrome store. MQTTLens workstation connects with MQTT broker and is able to subscribe and publish to MQTT topics. The following details are required for the MQTT connection with MQTTLens are: MQTT broker, ports and topic for publish and subscribe operations.

2.9 Eclipse MQTT Broker and Client:

2.9.1 Mosquitto Server:

Eclipse Mosquitto[\[11\]](#) is an open source (EPL/EDL licensed) message broker that implements the MQTT protocol versions 5.0, 3.1.1 and 3.1. Mosquitto[\[11\]](#) is lightweight and is suitable for use on all devices from low power single board computers to full servers.

The MQTT protocol provides a lightweight method of carrying out messaging using a publish/subscribe model. This makes it suitable for Internet of Things messaging such as with low power sensors or mobile devices such as phones, embedded computers or microcontrollers.

The Mosquitto project also provides a C library for implementing MQTT clients, and the mosquitto_pub and mosquitto_sub command line MQTT clients. Mosquitto is part of the Eclipse Foundation and is part of IoT.eclipse.org project.

2.9.2 Paho Mqtt client:

The Eclipse Paho project [\[12\]](#) provides open-source client implementations of MQTT and MQTT-SN messaging protocols. It is new, existing, and emerging applications for the Internet of Things(IoT). Paho provides an open-source client implementations of MQTT publish/subscribe for use on embedded platforms, along with corresponding server support. Paho Python Client provides a client class with support for MQTT v3.1, v3.1.1 on Python 2.7 or 3.4. It also provides client class which enable applications to connect to an MQTT broker to publish messages, and to subscribe to topics and receive published messages.

Paho client python library [\[12\]](#) is used to communicate many public broker like Eclipse, Hivemq and secure IoT platform like Amazon AWS IoT. Paho provides client library which is supported by many micro-controllers for IoT application.

Chapter 3

Hardware Design

3.1 Hardware Block Diagram

In the project, IoT development board is design, the aim is to connect and monitoring home appliance to the internet. IoT gateway performs an important role for remote accessing and monitoring the appliance from any place. The system block diagram is shown as Fig3.1.

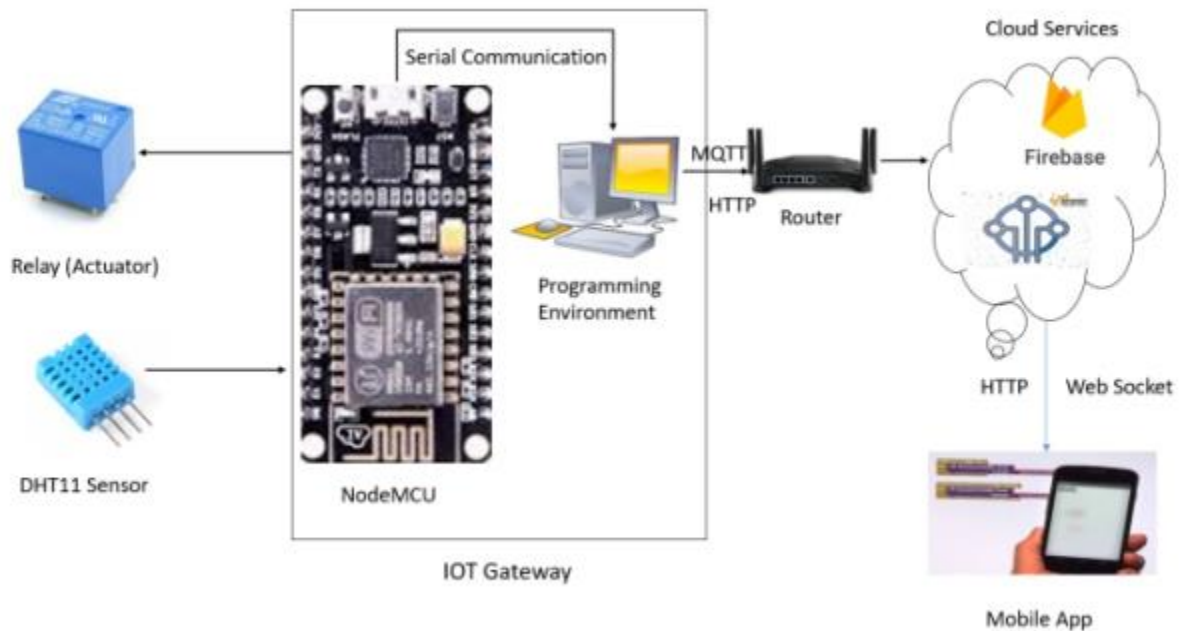


Fig. 3.1 Hardware Block diagram

Fig. 3.1 shows the Hardware block diagram for the purposed system. The system requires some hardware components are listed as below. The hardware components used are relays, DHT11 Sensor, NodeMCU, router and Smart Phone. The block diagram represents the relay and DHT11 Sensor are connected to NodeMCU GPIO. Google firebase library is installed on Arduino IDE for NodeMCU programming. The Wi-Fi and firebase client class used for create connection between ESP8266 and firebase.

DHT11 publish the humidity and temperature data and relay remotely control with the Google firebase. The Mobile Application developed with MIT APP Inventor 2 to get or send data to Google firebase.

The following list of hardware components are used in the project:

1. ESP8266 12E Wi-Fi Development Kit:

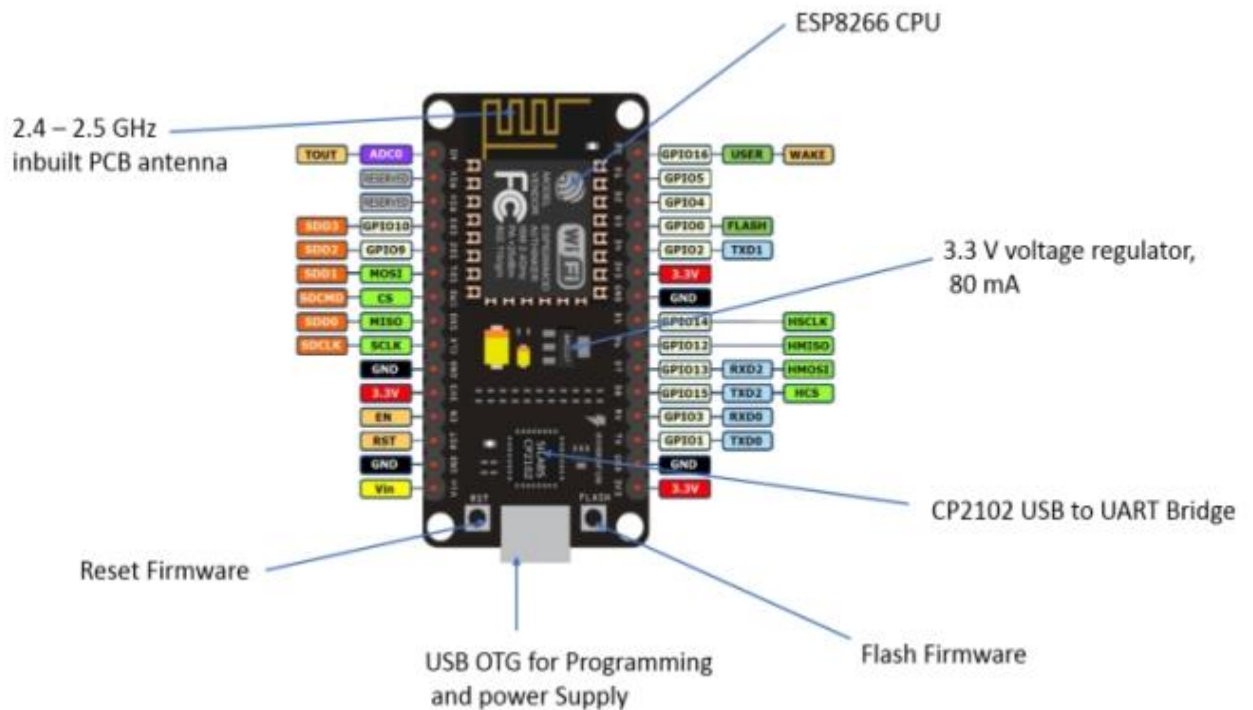


Fig. 3.2 ESP8266 12E Wi-Fi Development Board

The ESP8266 Wi-Fi development board features are, it is Breadboard Friendly, operates on 3.3 power supply. It has built-in wireless connectivity capabilities. It has built-in PCB antenna on the ESP-12E chip. It is capable of PWM, I2C, SPI, UART, 1-wire, 1 analog pin. It uses CP2102 USB serial communication interface module. It is compatible with Arduino C programming language in Arduino IDE.

Table 3.1 NodeMCU Board Specification

Parameters	Specifications
Microcontroller	ESP8266 12-E
Memory	Tensilla 32 bit
Processor Clock	80MHz – 160 MHz
RAM	36 KB
Storage	16 MB
Wireless Standard	IEEE 802.11 b/g/n
Operating Voltage	3.3 V
Operating Current	12 - 200 mA
Operating Temperature	-40 to 125°C
GPIO Capability	UART, I2C, PWM, GPIO, 1 ADC
Serial Transmission	110 - 921600 bps, TCP Client 5

Table 3.1 shows NodeMCU Board Specifications. The NodeMCU Board has Wi-Fi chip with ESP8266 CPU. It can operate on 5V USB power Supply. It can programming with Arduino IDE and mongoose OS.

2. Relay

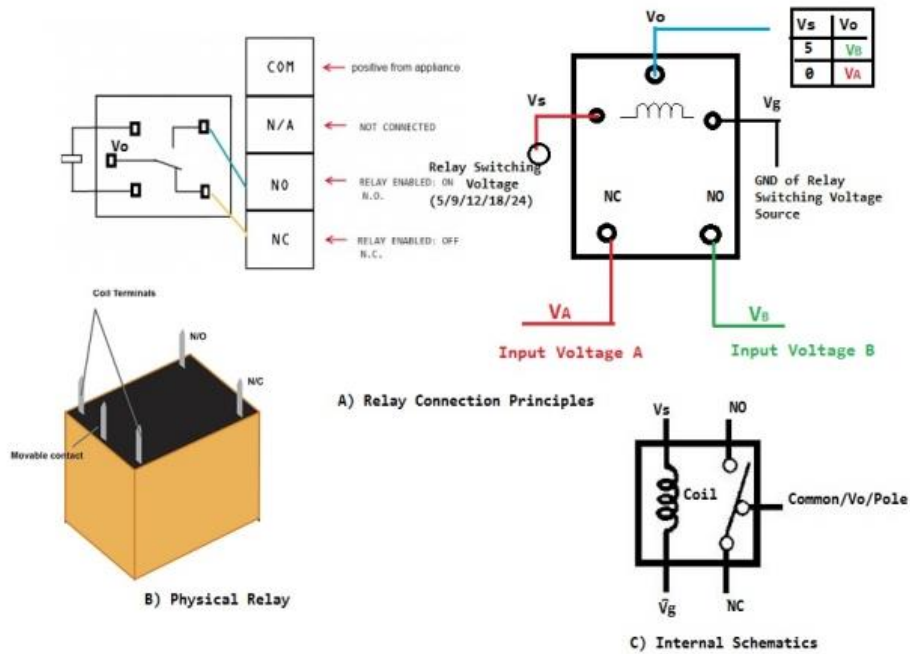


Fig. 3.3 Relay Circuit

Relays are known as electromagnetic switches. Relays can receive signal from an ESP8266 Wi-Fi development board and perform switching operation on the other side. The main operation of this device is to make or break contact with the help of a signal without any human involvement in order to switch it ON or OFF. It is mainly used to control a high power circuit using a low power signal. The relay is interfaced to the ESP8266 board's digital GPIO pins to remotely control 220 V A.C. power supply appliances.

Table 3.2 Relay connection with NodeMCU

Relay	NodeMCU Board Pin
1	D1
2	D2
3	D4
4	D5

The Table 3.2 shows the connection between the NodeMCU and Relays.

3. DHT11 Temperature and Humidity sensor

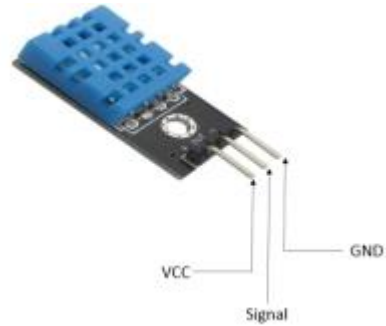


Fig. 3.4 DHT11 Sensor

Fig 3.4 shows DHT 11 Sensor, DHT11 temperature & Humidity Sensor is a temperature & humidity sensor with a calibrated digital signal output. The sensor includes a resistive type humidity measurement component and NTC (Negative Temperature Coefficient) temperature component. It offers excellent quality, fast response, anti-interference ability and cost effectiveness. It can measure temperature from 0-50 °C with an accuracy of $\pm 2^{\circ}\text{C}$ and relative humidity ranging from 20-90% with an accuracy of $\pm 5\%$. The sensor provides fully calibrated digital outputs for the humidity and temperature measurements

Table 3.3 DHT Sensor Connection with NodeMCU

DHT 11 Pin	NodeMCU Pins
VCC	3.3V
Signal	D4
GND	GND

The Table 3.3 NodeMCU and DHT 11 Sensor connection. The D4 pin is the digital input and output GPIO is connected with the signal pin of DHT 11.

4. Router:

Wireless routers are the hardware devices used to connect to Internet. A wireless router, also called a Wi-Fi router, combines the networking functions of a wireless access point and a router.

NodeMCU board can use Wi-Fi client library to connect Wi-Fi router by entering correct Wi-Fi SSID and password details. Router provides the Internet connectivity to the NodeMCU for IoT applications.

5. Smart Phone with Android OS:

In an IoT based application Smart phones with the Android OS can be used to monitor and control the devices from any place and at any time. The mobile applications are installed on the Android smart phone for IoT operations.

3.2 IoT Development Hardware

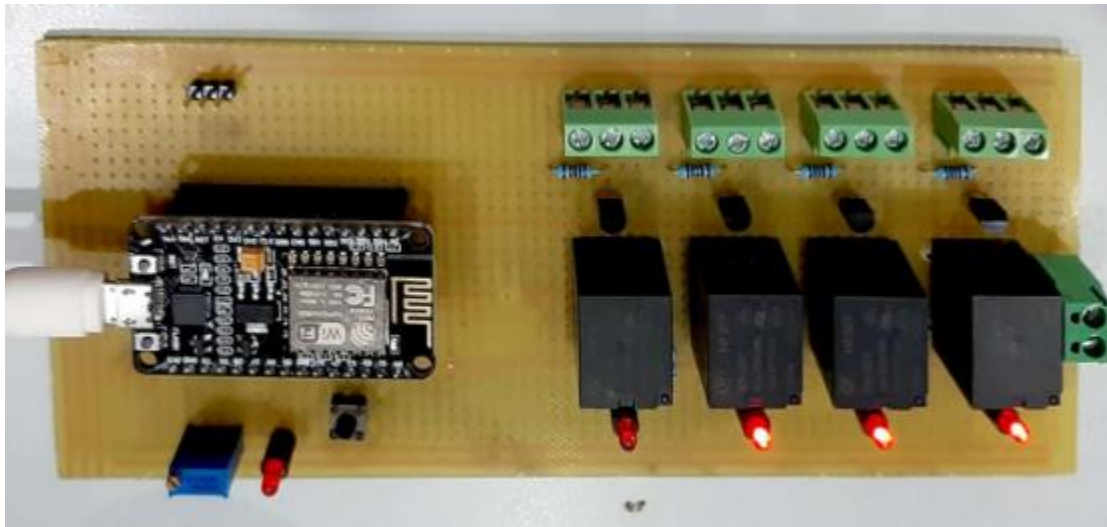


Fig. 3.5 IoT development Hardware with NodeMCU

Fig. 3.5 shows hardware design of prototype consists various components such as resistor, LED, wires, button, BC547 transistor, 1N4007 diode, potentiometer, terminals block and NodeMCU. The design of automated system should be simple and easy to

control the home appliances remotely and also be monitored their status at the same time with wireless access gateways. The Android app designed here is use App with graphical buttons to control the home appliances by the means of wireless media. The code is uploaded with required libraries to the NodeMCU using the Arduino IDE. Once the ESP8266 finds the matched combination of that particular SSID and Password, it connects to the access point and becomes part of the WLAN.

The hardware part of the developed system is programmed using Arduino IDE and the android application is developed using MIT APP Inventor. The hardware components have been programmed to communicate directly with the Firebase database. All the values of the sensors and the current status of each and every electrical appliance is continuously updated on the Firebase database using pre-configured Wi-Fi device.

3.3 Temperature monitoring system

Raspberry Pi 3b+ Board:

Raspberry Pi 3B+ have following specifications [\[13\]](#):

It has SoC: Broadcom BCM2837B0 quad-core A53 (ARMv8) 64-bit @ 1.4GHz, GPU: Broadcom Video core-IV, RAM: 1GB LPDDR2 SDRAM, Networking: Gigabit Ethernet(via USB channel), 2.4GHz and 5GHz 802.11b/g/n/ac Wi-Fi, Bluetooth: Bluetooth 4.2, Bluetooth Low Energy (BLE), Storage: Micro-SD, GPIO: 40-pin GPIO header, populated, Ports: HDMI, 3.5mm analogue audio-video jack, 4x USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)

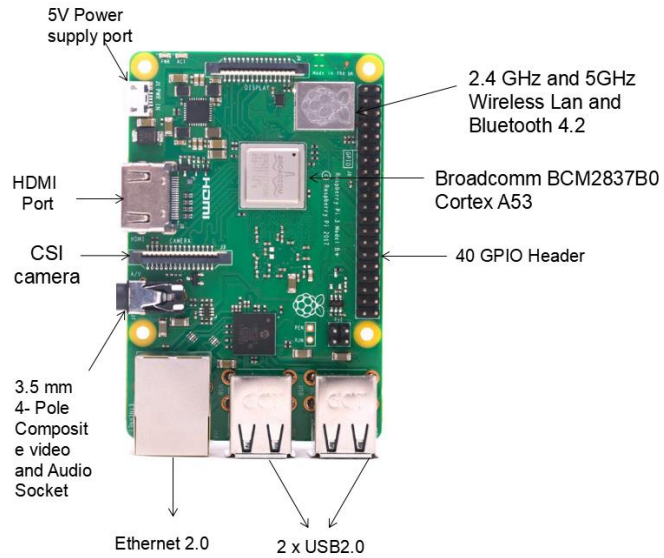


Fig. 3.6 Raspberry Pi 3b+ Board

Fig. 3.6 shows the raspberry pi 3b+ board. The board have Broadcom CPU, 40 pin GPIO header, Wi-Fi and Ethernet connectivity and HDMI and 4 USB port. This board can be work as computer. This board is operate on 5V DC and require 2.5 A USB power supply. CSI port available for camera interface and HDMI port available for video output.

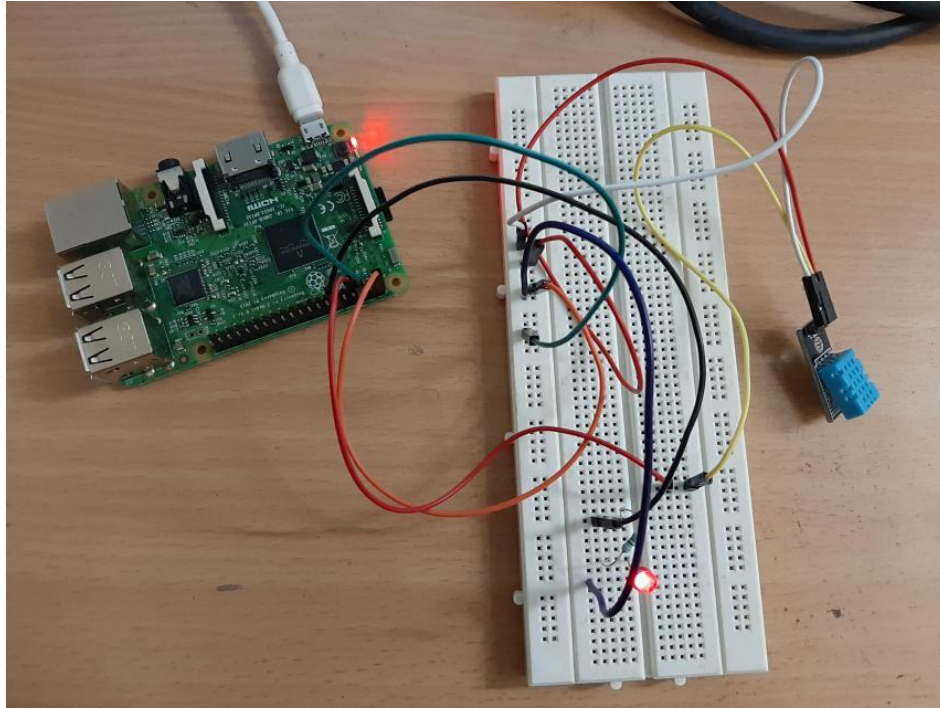


Fig. 3.7 Temperature monitoring system with Raspberry Pi

Fig. 3.7 shows hardware design of prototype consists various components such as DHT11 Sensor, Breadboard, wires, LED, resistor and Raspberry Pi 3b. The design of automated system should be simple and easy to control the home appliances remotely and also be monitored their status at the same time with wireless access gateways. The code is uploaded with paho MQTT libraries to the Raspberry Pi using the Python IDE. The IoT MQTT dashboard is use App with graphical buttons to remote control the LED.

Chapter 4

Software Design

4.1 Arduino IDE for ESP8266:-

The Arduino IDE software does not include ESP8266 Board support package. So, following steps are required to install the ESP8266 packages in the Arduino IDE.

4.1.1 Install ESP8266 Board support package [4]:

Open Arduino IDE and File < Preferences < Addition Boards URL Manager and copy below URL on the empty box:

http://arduino.esp8266.com/stable/package_esp8266com_index.json

Tool < Boards < Board Mangers and select esp8266 community and Install

Tool < Boards < NodeMCU 1.0 and Select appropriate COM Port

Now open File < Examples < Examples for NodeMCU 1.0 and set Wi-Fi router SSID and Password and run the code. Now ESP8266 connects to the router and it has internet connectivity.

4.1.2 ESP8266 and Blynk App communication:

Step 1: Download the Blynk APP library zip file from github and install in Arduino IDE:

Sketch < Include Library < Add Blynk App.zip library.

Step2: Download Blynk APP from Application store.

Step3: Now create a Blynk account, and login to the account.

Step4: Create new project by enter the project Name and the choose NodeMCU device.

Step5: Now Auth Token is send to the register email id.

Step6: Now Open File < Examples <Blynk < Boards_WiFi < NodeMCU

Step7: Enter the WiFi SSID, Password, Auth Key and upload code on NodeMCU Board.

Step8: Now Open Blynk App and add the widgets from widget box to control NodeMCU remotely from Blynk App.

4.1.3 ESP8266 remote access from Google Firebase:

Step 1: Download the Firebase library zip file from github and install in Arduino IDE:

Sketch < Include Library < Add Google Firebase .zip library.

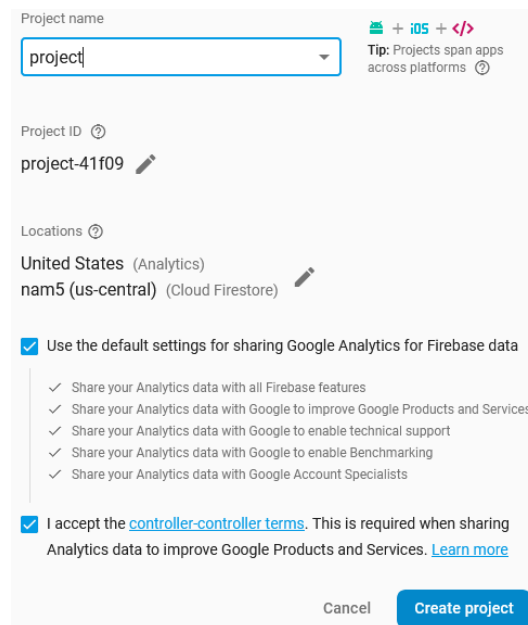
Step 2: Open the Google firebase console with following URL:

<https://firebase.google.com/>

Step 3: Now create a Google firebase account, and login to the account.

Step 4: Now select on get started to access firebase console page.


Step 5. Now click on the Add project and enter the project name, location and accept the agreement to create the project.





Project name

project


+ iOS + </>
Tip: Projects span apps across platforms

Project ID 

project-41f09 

Locations 

United States (Analytics)

nam5 (us-central) (Cloud Firestore) 

Use the default settings for sharing Google Analytics for Firebase data

- Share your Analytics data with all Firebase features
- Share your Analytics data with Google to improve Google Products and Services
- Share your Analytics data with Google to enable technical support
- Share your Analytics data with Google to enable Benchmarking
- Share your Analytics data with Google Account Specialists

I accept the [controller-controller terms](#). This is required when sharing Analytics data to improve Google Products and Services. [Learn more](#)

Cancel **Create project**

Fig. 4.1 Create project on firebase

Fig. 4.1 shows, the information require like project name, location for create new project in the Google firebase platform.

Step6. Now click on continue and select database < Create database and enable lock mode

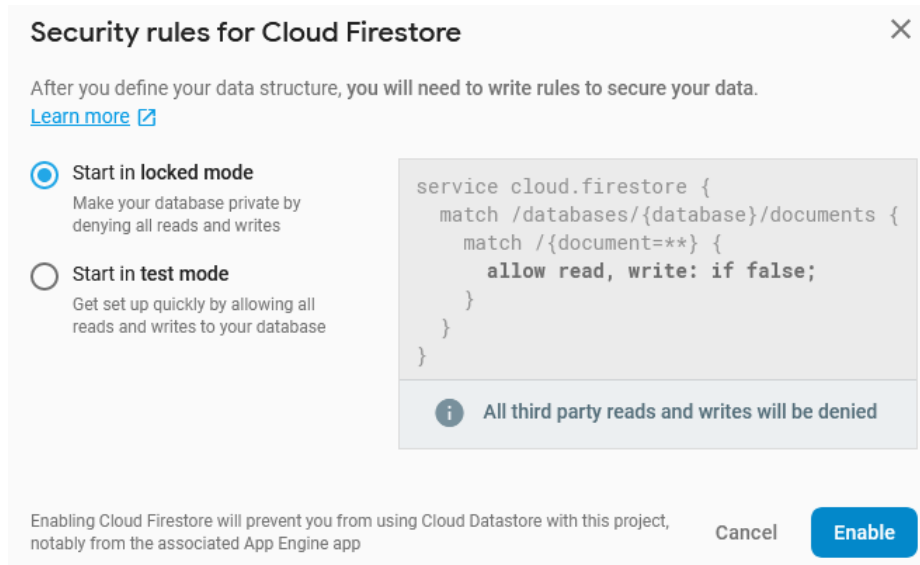


Fig. 4.2 Security rule for Cloud Firebase

Fig. 4.2 shows, Lock and Test mode security rule in Google Firebase.

Step7. Now select Real time database and copy the project URL:

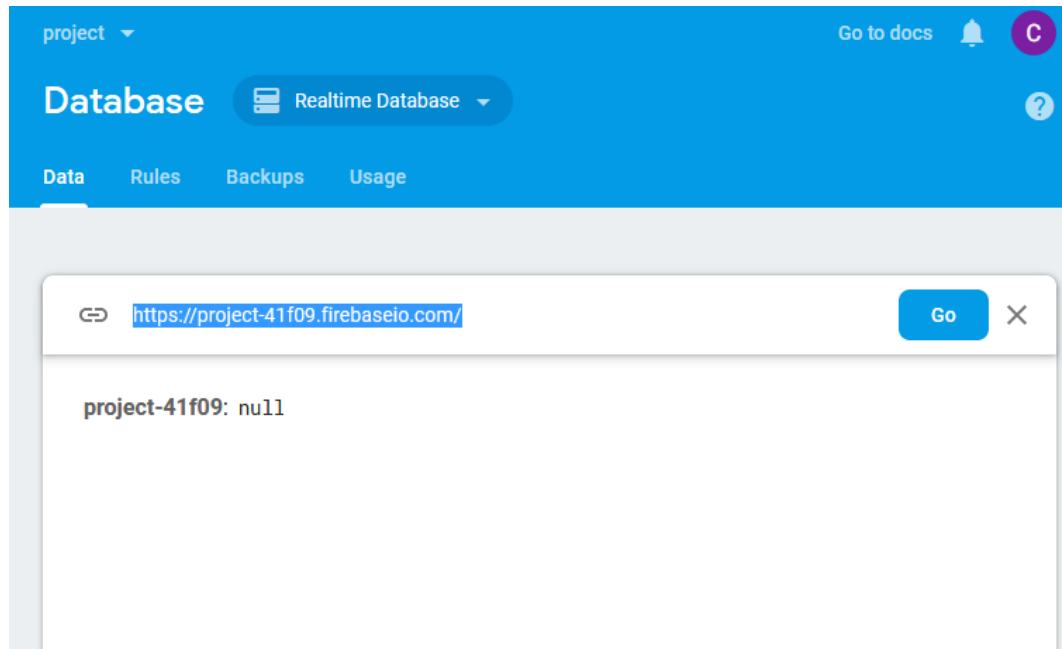


Fig. 4.3 Firebase Host URL

Fig. 4.3 shows the Google firebase project URL. The URL is pasted as host address in NodeMCU programming in Arduino IDE. The default port is 1883.

Step8. Now select on setting symbol < open project setting < select service account < Database secret and copy the secret key.

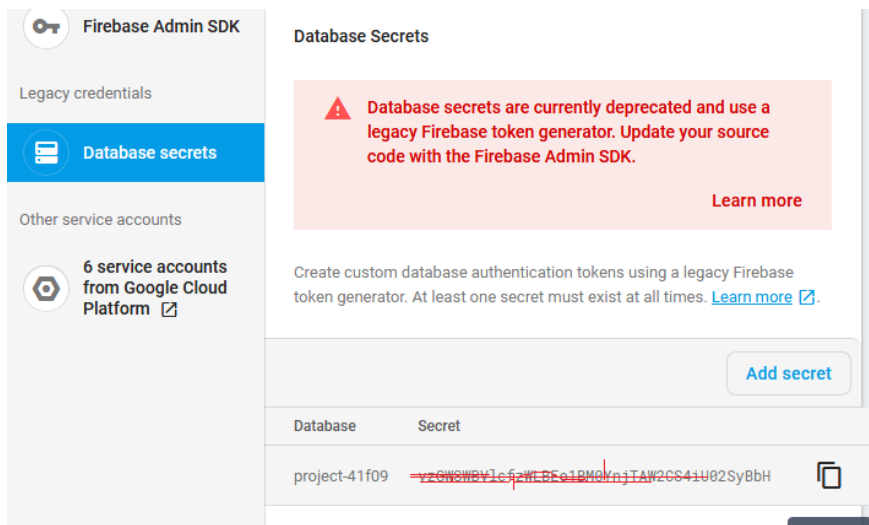


Fig. 4.4 Firebase database Secret key

The Fig. 4.4 shows the database secret is hidden on this page. It can show by click on show button. The database secret key is the work as security key to authenticate the user. The firebase host address and secret key is require for connecting with ESP8266.

Step9. Open Arduino IDE < Examples < FirebaseArduino <FirebaseDemo_ESP8266 and paste the URL and secret key, enter Wi-Fi ssid and password and upload the code on NodeMCU board. NodeMCU and firebase is connected to each other.

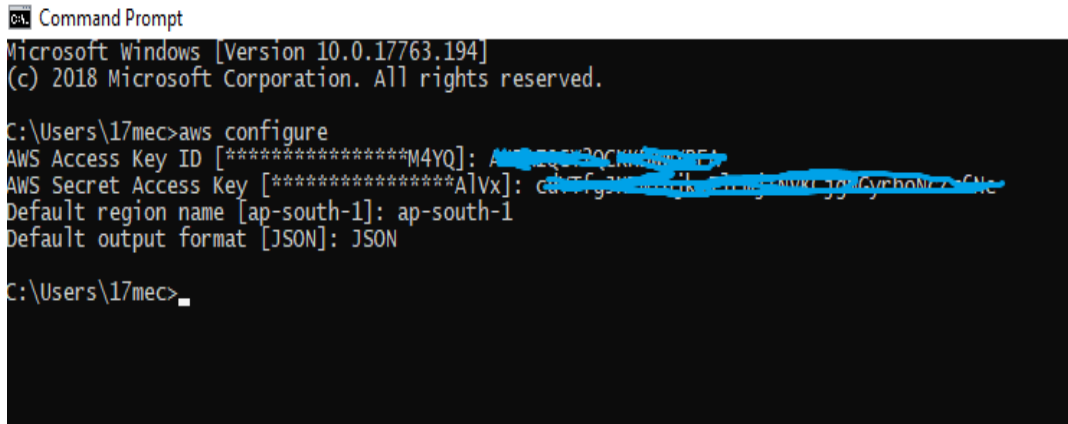
4.2 ESP8266 and AWS IoT communication:

The following steps are required for mongoose OS[8] to communication between ESP8266 and AWS IoT.

Prerequisites:

Python environment require for AWS CLI installation.

AWS CLI configuration in command Line:



```
Command Prompt
Microsoft Windows [Version 10.0.17763.194]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\17mec>aws configure
AWS Access Key ID [*****M4YQ]: *****
AWS Secret Access Key [*****A\vx]: *****
Default region name [ap-south-1]: ap-south-1
Default output format [JSON]: JSON

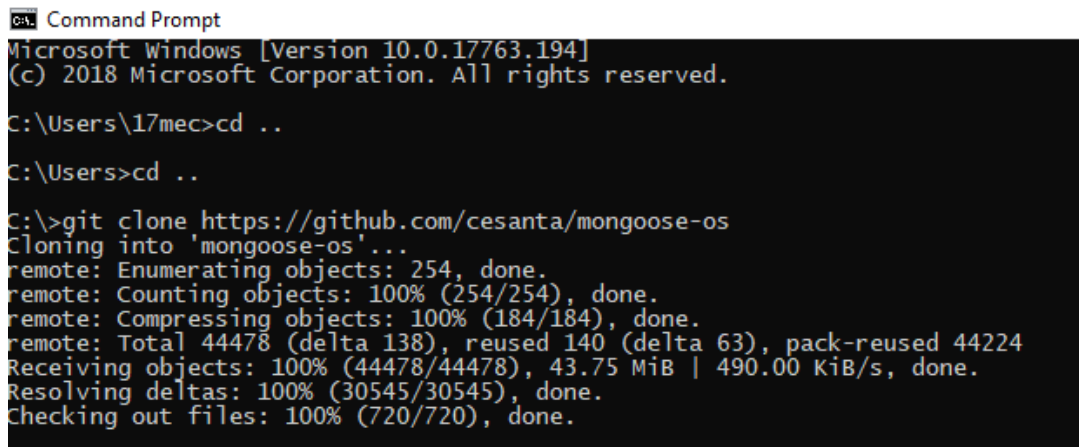
C:\Users\17mec>
```

Fig. 4.5 AWS CLI configuration

Fig. 4.5 shows the AWS CLI configuration steps, it requires access key id, secret access key, AWS region name and output format. Access key id and secret access key are getting from the AWS IAM user.

Step 1 – Mongoose OS [\[8\]](#) git repository

A git clone of the Mongoose OS repository from GitHub, which can obtain with the git clone `https://github.com/cesanta/mongoose-os` command. Navigate to the git repository in the shell.



```
Command Prompt
Microsoft Windows [Version 10.0.17763.194]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\17mec>cd ..

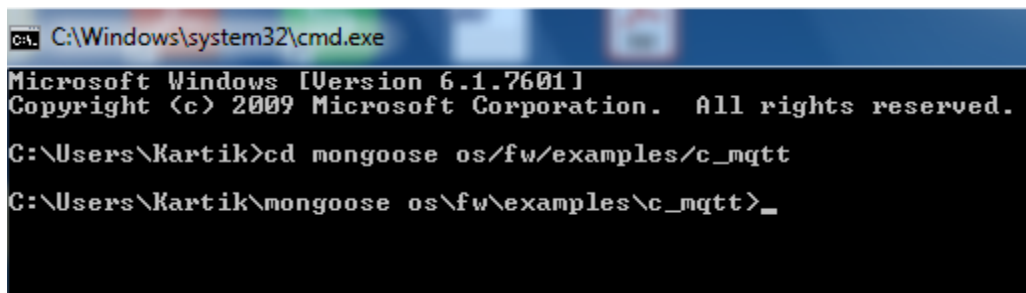
C:\Users>cd ..

C:\>git clone https://github.com/cesanta/mongoose-os
Cloning into 'mongoose-os'...
remote: Enumerating objects: 254, done.
remote: Counting objects: 100% (254/254), done.
remote: Compressing objects: 100% (184/184), done.
remote: Total 44478 (delta 138), reused 140 (delta 63), pack-reused 44224
Receiving objects: 100% (44478/44478), 43.75 MiB | 490.00 KiB/s, done.
Resolving deltas: 100% (30545/30545), done.
Checking out files: 100% (720/720), done.
```

Fig. 4.6 mongoose OS git clone

Fig4.6. shows the git clone of the mongoose OS downloaded. The mongoose OS git clone some default setting and demo examples are available in C and Javascript.

Step2. Navigate to the c_mqtt firmware example directory: \$ cd mongoose-os/fw/examples/c_mqtt



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Kartik>cd mongoose os/fw/examples/c_mqtt
C:\Users\Kartik\mongoose os\fw\examples\c_mqtt>_
```

Fig. 4.7 Navigate the mongoose OS directory

Fig. 4.7 show the steps for navigate the mongoose OS directory. The c_mqtt is the demo code available to for the demonstration of c programming in mongoose OS.

Step3. Build the firmware: \$ mos build --arch esp8266

```
C:\ Command Prompt
Microsoft Windows [Version 10.0.17763.194]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\17mec>cd mongoose-os/fw/examples/c_mqtt

C:\Users\17mec\mongoose-os\fw\examples\c_mqtt>mos build --arch esp8266
Connecting to https://mongoose.cloud, user test
Uploading sources (3763 bytes)
Firmware saved to C:\Users\17mec\mongoose-os\fw\examples\c_mqtt\build\fw.zip
```

Fig. 4.8 mongoose OS build the firmware for ESP8266

Fig. 4.8 shows the mongoose OS build the firmware for the ESP8266.

Step4. Flash the firmware: \$ mos flash

```
C:\Users\17mec\mongoose-os\fw\examples\c_mqtt>mos flash
Loaded c_mqtt/esp8266 version 1.0 (20181212-175654)
Using port COM3
Opening COM3 @ 115200...
Connecting to ESP8266 ROM, attempt 1 of 10...
  Connected, chip: ESP8266EX
Running flasher @ 0...
  Flasher is running
Flash size: 4194304, params: 0x0240 (dio,32m,40m)
Deduping...
  2320 @ 0x0 -> 0
  262144 @ 0x8000 -> 0
  711104 @ 0x100000 -> 0
  128 @ 0x3fc000 -> 0
Writing...
  4096 @ 0x7000
  4096 @ 0x3fb000
Wrote 8192 bytes in 0.08 seconds (826.15 KBit/sec)
Verifying...
  2320 @ 0x0
  4096 @ 0x7000
  262144 @ 0x8000
  711104 @ 0x100000
  4096 @ 0x3fb000
  128 @ 0x3fc000
Booting firmware...
All done!
```

Fig. 4.9 mongoose os flash on ESP8266

Fig. 4.9 shows the mongoose OS firmware is flash on the ESP8266 architecture. When the firmware is flashed the ESP8266 reboot and booting code from firmware.

Step5. ESP8266 Wi-Fi Configuration: \$ mos wifi WIFI_SSID WIFI_PASSWORD

Wi-Fi configure by replacing WIFI_SSID and WIFI_PASSWORD with the appropriate values for the environment:

```
C:\Users\17mec\mongoose-os\fw\examples\c_mqtt>mos wifi GIONEE [REDACTED]22
Using port COM3
Getting configuration...
Setting new configuration...
Saving and rebooting...
```

Fig. 4.10 NodeMCU Wi-Fi configuration

Fig. 4.10 shows the Wi-Fi configuration steps for the ESP8266 in the mongoose OS. The Wi-Fi SSID and password is require to connect the Wi-Fi access point.

Step6. AWS IoT Setup: \$ mos aws-IoT-setup --aws-region REGION--aws-IoT-policy mos-default

Generate certificates, upload them to the NodeMCU board, and set up the MQTT parameters by replacing REGION with the name of the region that use with AWS IoT:

```
Attaching policy "mos-default" to the certificate...
2018/12/12 23:36:16 This operation, AttachPrincipalPolicy, has be
Attaching the certificate to "test"...
writing certificate to aws-test.crt.pem...
Uploading aws-test.crt.pem (1120 bytes)...
writing key to aws-test.key.pem...
Uploading aws-test.key.pem (227 bytes)...

Updating config:
  aws.thing_name = test
  mqtt.enable = true
  mqtt.server = a5k8k70fc67ex.iot.ap-south-1.amazonaws.com:8883
  mqtt.ssl_ca_cert = ca.pem
  mqtt.ssl_cert = aws-test.crt.pem
  mqtt.ssl_key = aws-test.key.pem
Setting new configuration...
Saving and rebooting...
```

Fig. 4.11 Mongoose OS upload certificates on AWS IoT

Fig. 4.11 shows the AWS IoT setup with the mongoose OS. Mongoose OS setup generates default setup, device certificate, region and policy and upload on AWS IoT. ESP8266 and AWS IoT is connection is setup successfully.

4.3 Raspberry Pi remote access with IoT MQTT dashboard:

Free online broker used is broker.hivemq.com.

Following are the steps to install paho MQTT Client on Raspberry Pi:

Step1 Execute the following command to install python pip

```
sudo apt-get install python-pip
```

Step2 Execute the following command to install paho MQTT client library

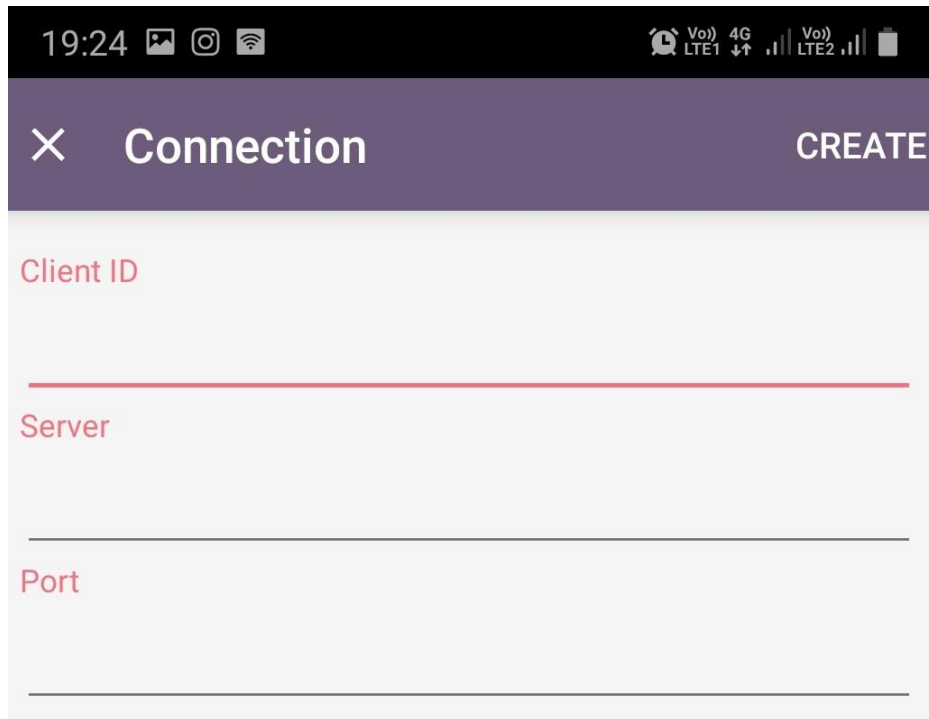
```
sudo pip install paho-mqtt
```

Step3 Execute the following command to install Rpi Gpio library:

```
sudo pip install RPi.GPIO
```

Step4 Now open the subscribe.py script from paho-mqtt < site-packages and add host address and port number and subscription topic.

Step5 Open IoT MQTT dashboard Application and enter the client id, server and port details



The screenshot shows a mobile application interface for creating a connection. At the top, there is a purple header bar containing a close button (X), the title "Connection", and a "CREATE" button. Below the header, there are three input fields: "Client ID" (with a red underline), "Server" (with a black underline), and "Port" (with a black underline). The status bar at the top of the phone shows the time 19:24, signal strength, and battery level.

Fig. 4.12 Create connection on IoT MQTT dashboard

Fig. 4.12 shows the main page of IoT MQTT dashboard Application. The topic, broker and port details can be fill here.

Step6 Now select available component as per requirement.

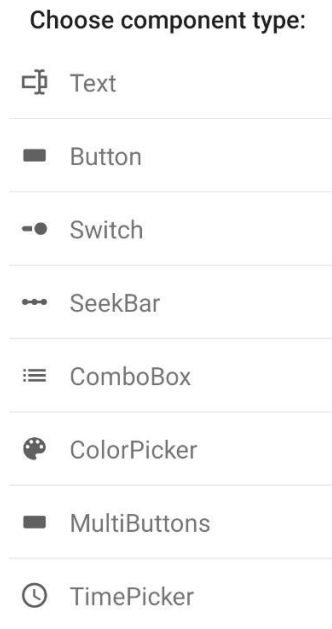


Fig. 4.13 IoT MQTT dashboard components

Fig. 4.13 shows the components available in the IoT MQTT dashboard Application. The components are Text, Button, Switch, Time Picker, etc.

Step7 Edit the details of topic and start publish the message:

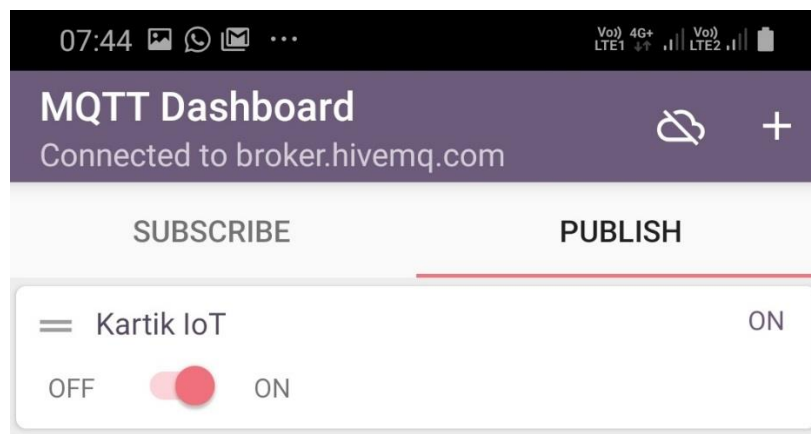


Fig. 4.14 Publish message with IoT MQTT Dashboard

The Fig. 4.14 shows the on and off button switch to remote access Led connected with raspberry pi

Step8. Now enter the same topic in subscribe.py script and run the script. The raspberry pi subscribe the message from the IoT MQTT dashboard.

4.4 Raspberry Pi remote access with MQTT Lens Workstation:

Free online broker used here is: broker.hivemq.com

Following are the steps to install paho MQTT Client on Raspberry Pi:

Step1 Execute the following command to install python pip

```
sudo apt-get install python-pip
```

Step2 Execute the following command to install paho MQTT client library

```
sudo pip install paho-mqtt
```

Step3 Execute the following command to install Rpi Gpio library:

```
sudo pip install RPi.GPIO
```

Step4 Now open the subscribe.py script from paho-mqtt < site-packages and add host address and port number and the subscription topic.

Step5 Open IoT MQTTLens Application and enter the client id, server and port details

Connection Details

Connection name: Kartik

Connection color scheme: [Green bar]

Hostname: tcp:// broker.hivemq.com

Port: 1883

Client ID: lens_hoRHTzXUeD4KcGXz8Gd9Z42MCdf

Session: Clean Session

Automatic Connection: Automatic Connection

Keep Alive: 120 seconds

Generate a random ID

Fig. 4.15 Create connection in MQTTLens

Step6 Edit the details of topic and start publish the message:

MQTTLens Version 0.0.14

Connections + ^

Kartik

Connection: Kartik

Subscribe

raspi/1 0 - at most once SUBSCRIBE

Publish

raspi/1 0 - at most once Retained PUBLISH

Message

OFF

Fig. 4.16 Publish message with MQTTLens

Fig. 4.16 shows the MQTTLens Web Application can publish and subscribe raspi/1 topic from the broker.hivemq.com MQTT broker to remote access raspberry pi.

4.5 Raspberry Pi remote access with AWS IoT:

Following are the steps to install paho MQTT Client on Raspberry Pi:

Step1 Execute the following command to install python pip

```
sudo apt-get install python-pip
```

Step2 Execute the following command to install paho MQTT client library

```
sudo pip install paho-mqtt
```

Step3 Execute the following command to install Rpi Gpio library:

```
sudo pip install RPi.GPIO
```

Step4 Execute the following command to install AWS IoT Python SDK and download AWS IoT certificates and private key:

```
sudo pip install AWSIoTPythonSDK
```

Step5 Open the Aws_publish.py and Aws_subscribe.py script from AWS IoT python SDK in raspberry pi.

Step6 Add the AWS IoT endpoint and 8883 MQTT port number and TLS set class

with the rootCA, private certificates and private key. Now connect the AWS IoT

with AWS IoT end point , port number and the TLS class.

Step7 Execute the Aws_publish.py python script with following command:

```
sudo python Aws_publish.py
```

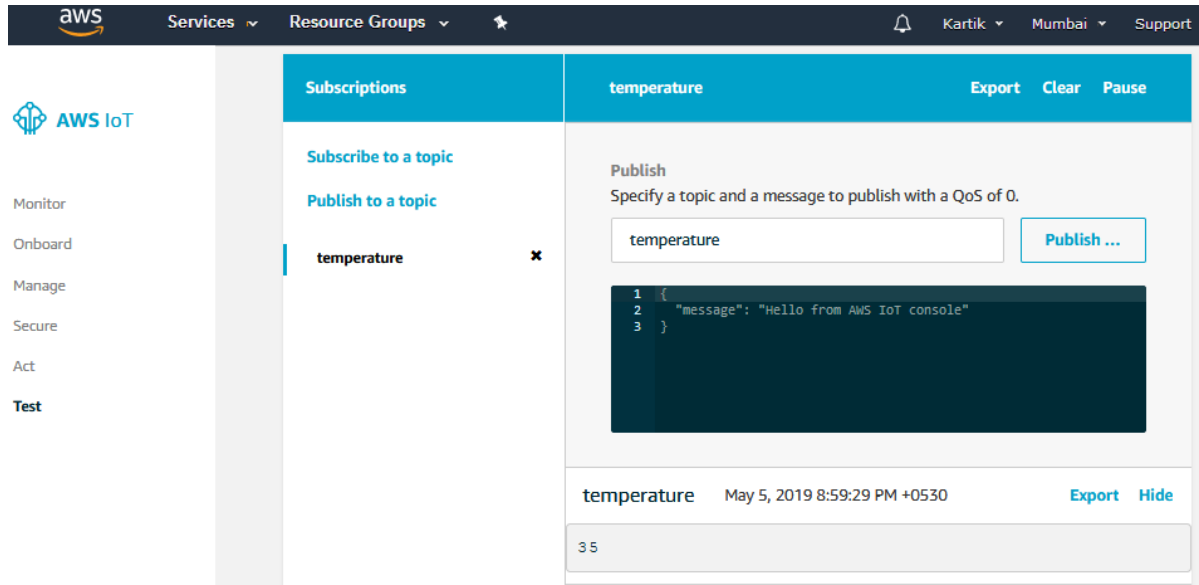


Fig. 4.17 DHT 11 publish data on AWS IoT

The Fig. 4.17 shows the page of AWS IoT MQTT Test in AWS IoT. Temperature topic is subscribe to get DHT11 sensor data on AWS IoT.

Chapter 5

Testing and Results

5.1 IoT Development Hardware:

5.1.1 IoT development hardware with NodeMCU:



Fig. 5.1 IoT development hardware with NodeMCU

Fig 5.1 shows the home automation development kit with NodeMCU. The relays and sensor are connect are connected with the ESP8266. The Hardware kit is design with button and led for digital input and output operations.

5.1.2 Raspberry Pi IoT development kit:

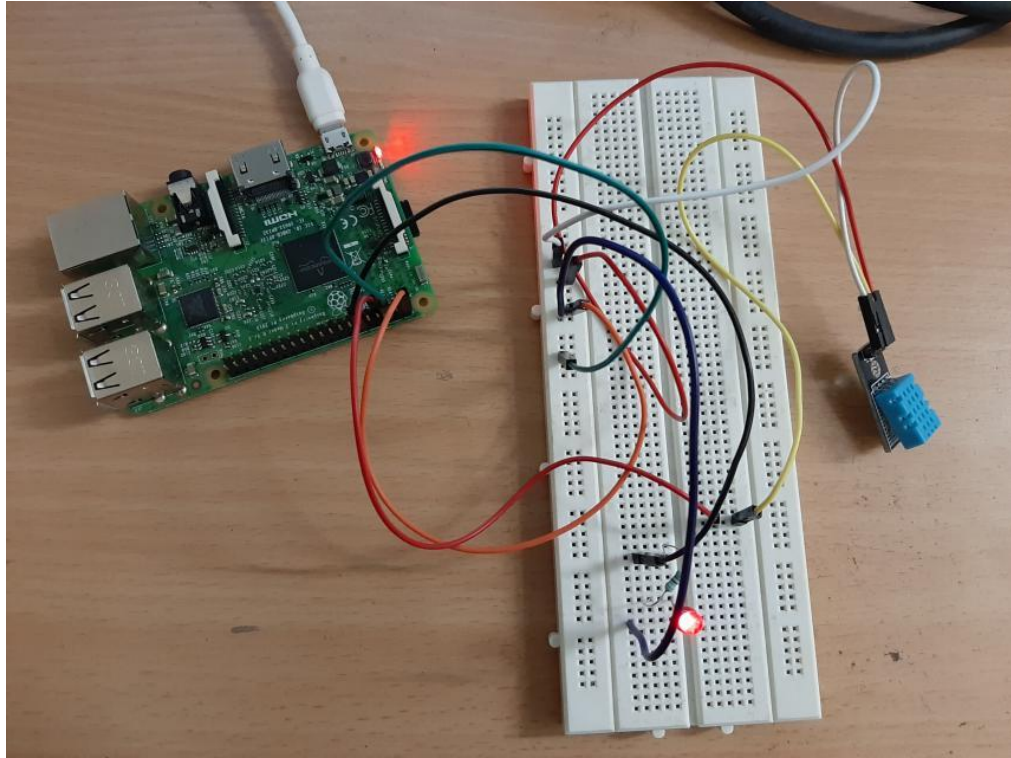


Fig. 5.2 Raspberry Pi IoT development kit

Fig. 5.2 shows the IoT development kit with the Raspberry Pi board. DHT 11 sensor publish temperature and humidity data with the temperature and humidity topic. Led is connected with raspberry pi GPIO pin 4 and subscribe the topic from the broker or mobile app. This hardware is tested successfully with IoT MQTT dashboard, AWS IoT MQTT client and MQTTLens.

5.2 Monitor and control on android Mobile Application:

5.2.1 Mobile App with MIT APP Inventor



Fig. 5.3 Mobile App MIT APP Inventor

Fig. 5.3 shows the graphical user interface of the firebase automation mobile application. The mobile application developed by MIT App Inventor and Google firebase application programming interface. The button and display widget are used for application development. Relay ON and Relay OFF are used to remote control the relays on ESP8266 and analog sensor data can share on display. The application successfully tested on IoT development kit of ESP8266.

5.2.2 Blynk Mobile Application

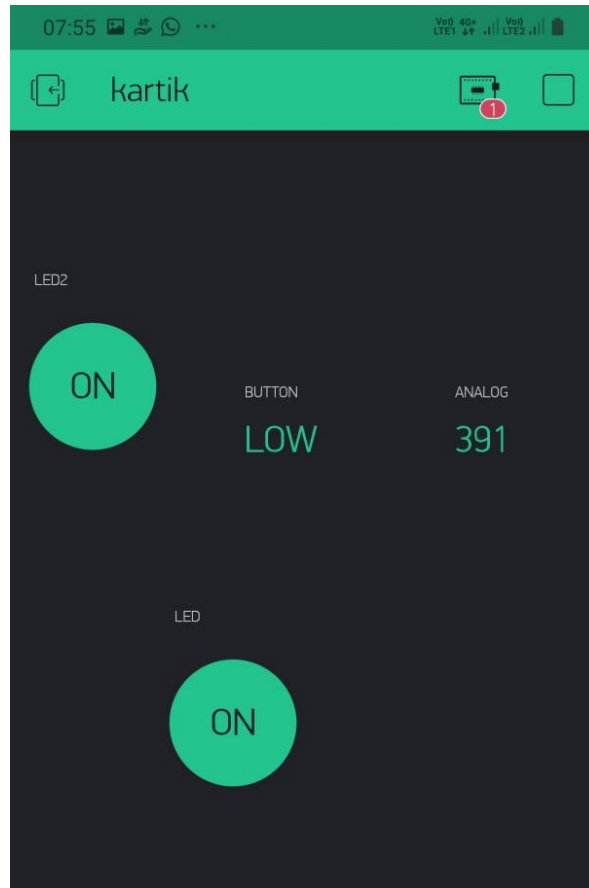


Fig 5.4 Blynk App Dashboard

Fig5.4 shows the Blynk Mobile App Dashboard. Button and display widgets are used to remote control NodeMCU Board. There 2 LEDs, 1 Button are controlled with the digital GPIO of ESP8266 and the display show the analog sensor real time data.

5.2.3 Remote access with MQTT Dashboard

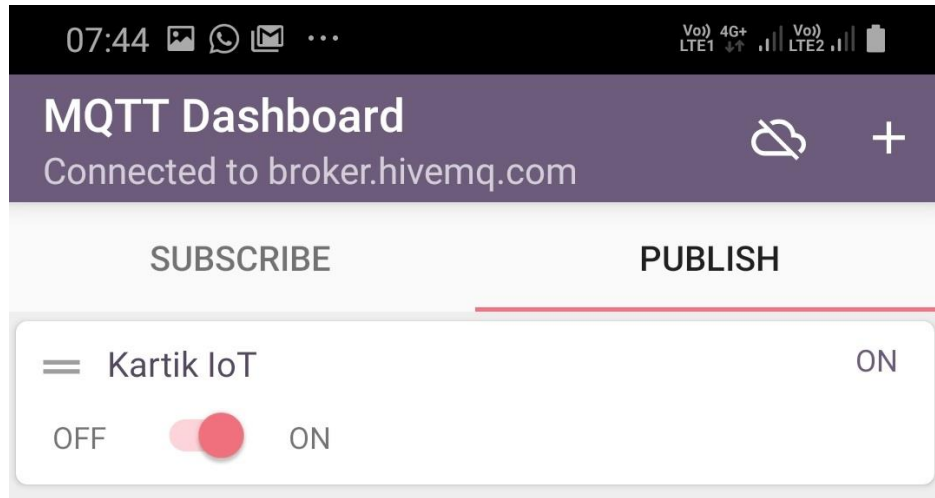


Fig. 5.5 Remote access with IoT MQTT Dashboard

Fig. 5.5 shows the simple toggle button for ON and OFF button with IoT MQTT dashboard to control the raspberry pi GPIO.

5.3 Monitor and control on Firebase Cloud:

5.3.1 DHT Sensor Real time database:

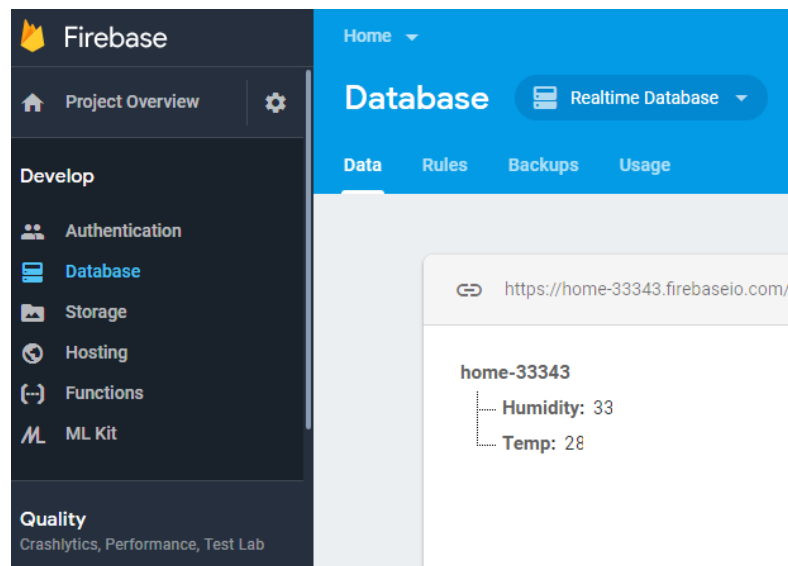


Fig. 5.6 DHT 11 data on Firebase

Fig. 5.6 shows the real time data base service is used in google firebase. Temperature and humidity data are continuously update on this page. This data value changes with few milliseconds and so, it is called the real time database.

5.3.2 IoT development hardware data on firebase

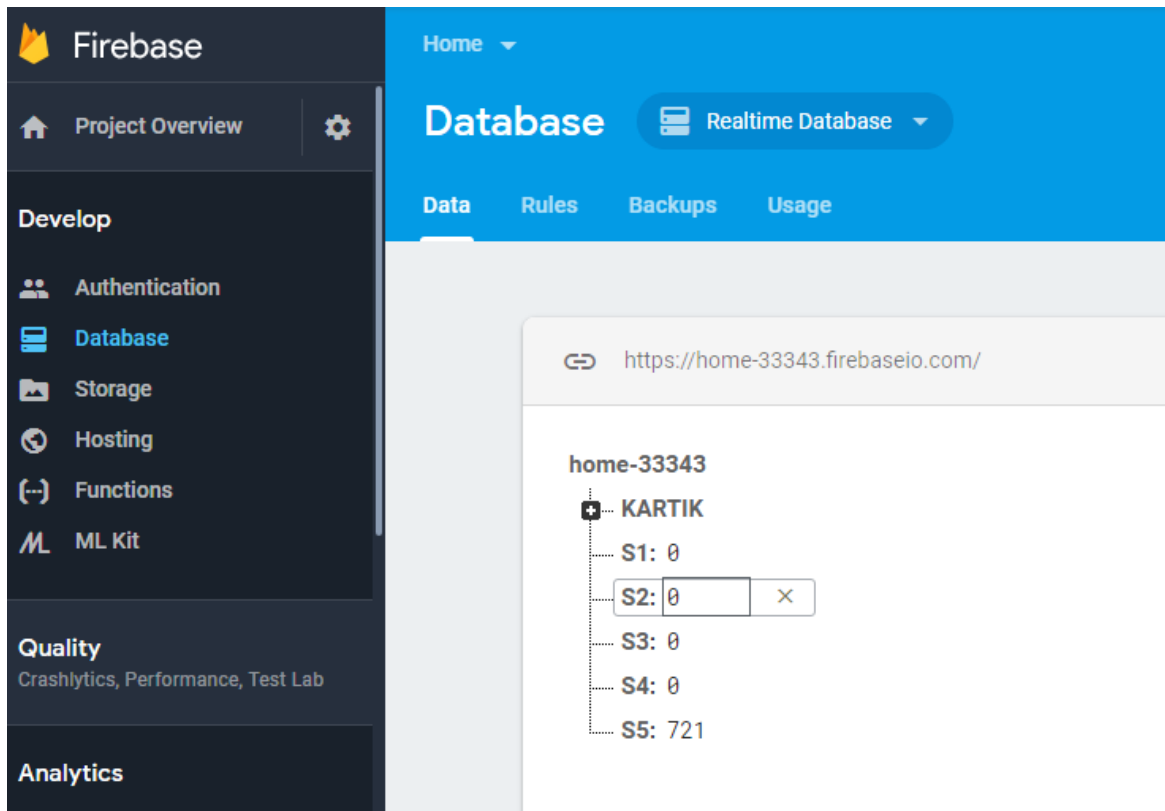


Fig. 5.7 IoT development hardware data on firebase

Fig. 5.7 Shows Home Automation development kit data share on Google firebase database. S1, S2, S3, S4 are the relays. Relays can be on or off by changing this value as 1 and 0. S5 shows the analog sensor value update on real time basis.

5.4 Monitor and control on AWS IoT:

5.4.1 ESP8266 remotely access through AWS IoT:

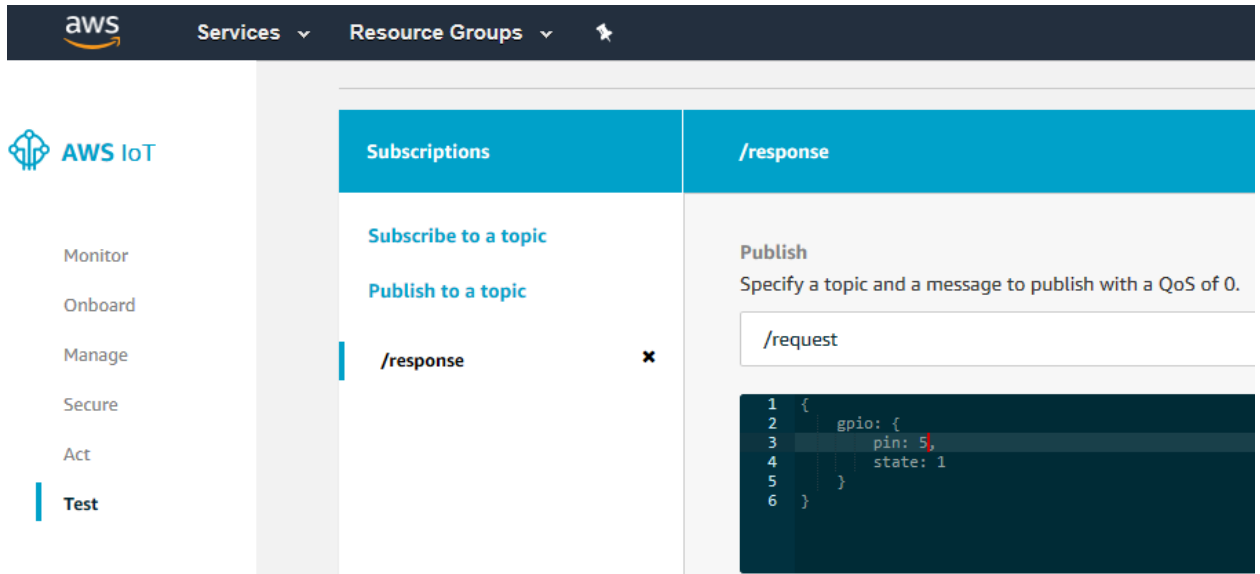


Fig. 5.8 ESP8266 GPIO remotely access by AWS IoT

Fig. 5.8 shows the GPIO and Pin status of the ESP8266 with the topic /request and it can change by the changing the pin 1 for ON and 0 for OFF.

5.4.2 DHT 11 data share on AWS IoT:-

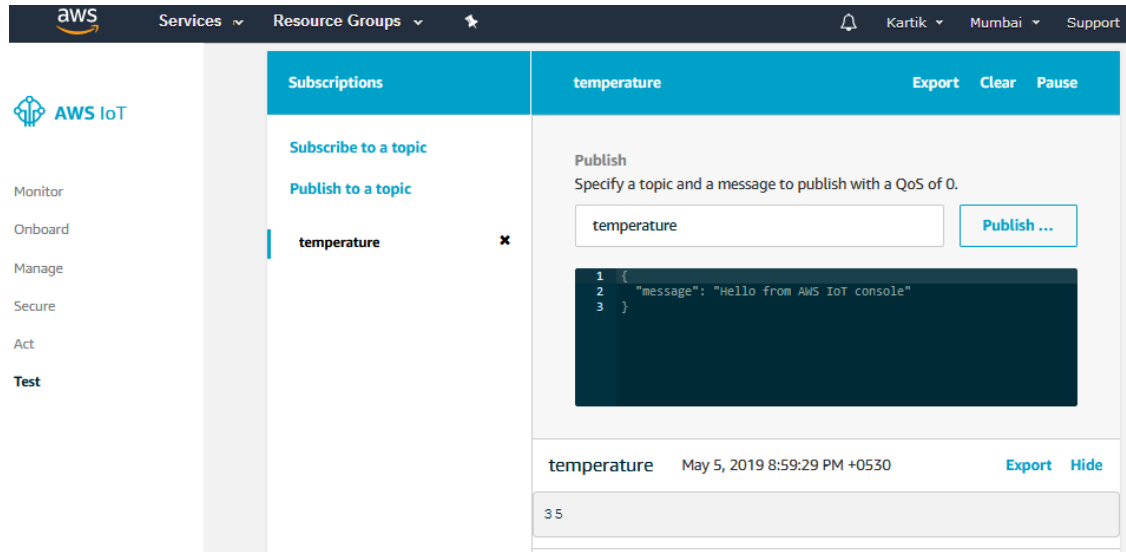


Fig. 5.9 Temperature data publish on AWS IoT

Fig. 5.9 shows the mqtt client test page of the AWS IoT. Temperature topic subscribe the value of temperature data of DHT 11 sensor share by raspberry pi on the AWS IoT Platform.

5.4.3 Raspberry pi remotely access through AWS IoT

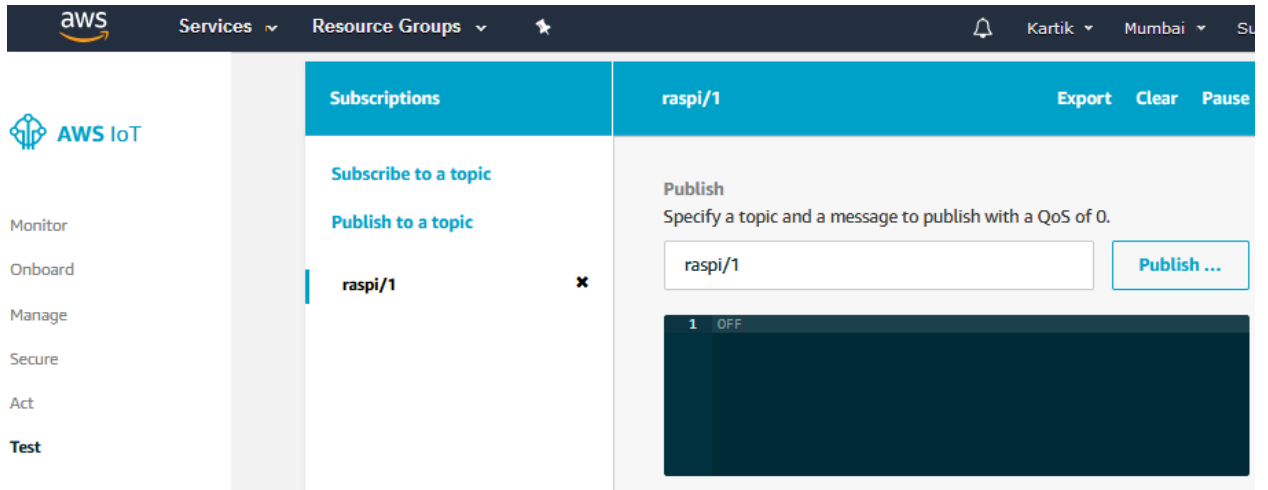


Fig. 5.10 Raspberry Pi GPIO remotely access by AWS IoT

Fig. 5.10 shows the AWS IoT MQTT client test page, the raspberry pi GPIO can be control with the publishing ON and OFF on the raspi/1 topic.

5.5 Monitor and control through Workstation:

5.5.1 Remote access with MQTTLens:-

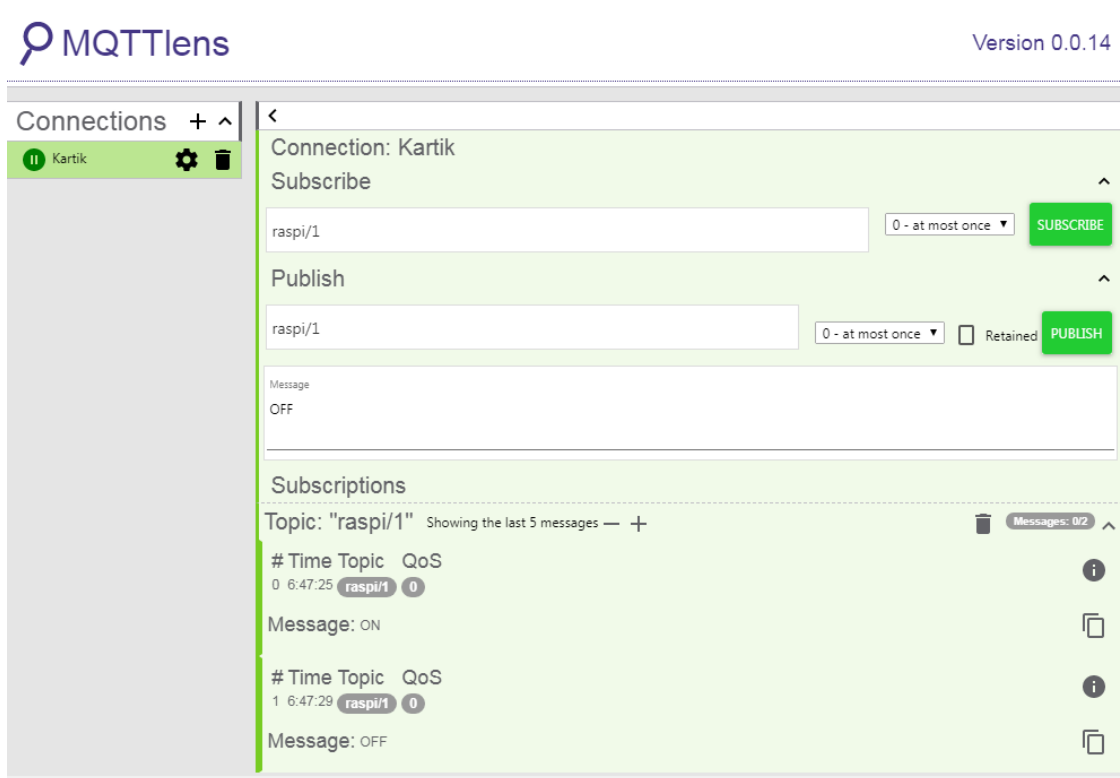


Fig. 5.11 Remote access with MQTTLens

Fig5.11 shows the MQTTLens Web Application can publish and subscribe raspi/1 topic from the broker.hivemq.com MQTT broker to remote access raspberry pi.

Chapter 6

Conclusion and Future Scope

Conclusion

In this thesis, ESP8266 Wi-Fi Development Kit is used to design IoT gateway. Arduino IDE and Mongoose OS are used for programming with ESP8266 Wi-Fi development board. The appliance is connected with ESP8266 board can remotely control with Blynk Application and Google firebase. Google firebase provides backend service for developing mobile with MIT APP Inventor. AWS IoT is used as secured IoT platform for remotely control ESP8266 and Raspberry Pi. DHT 11 Sensor publish the data on google firebase and AWS IoT platform. Paho MQTT library provides MQTT client supports for the various microcontroller. IoT MQTT dashboard provides mobile dashboard service to remote control and monitoring raspberry pi. The gateway is developed and tested successfully.

Future Scope

Schematic and PCB design for the various IoT Application.

Machine learning and Artificial intelligence can be applied with IoT application.

Bibliography

- [1].P. Srinivasan, “TM4C-IoT-Gateway-with-Security Protection in Texas Instruments, 2016.
- [2].H. Chen, X Jia “A brief introduction to IoT gateway”, IET International Conference on Communication Technology and Application (ICCTA 2011)
- [3].N.Nitin “Choice of Effective Messaging Protocols for IOT Systems: MQTT, CoAP, AMQP and HTTP” 2017 IEEE International Systems Engineering Symposium (ISSE) 2017.
- [4].Arduino IDE “Arduino core for ESP8266 WiFi chip” [Online] Available <https://github.com/esp8266/Arduino> [Access: 16- October -2018]
- [5].Mongoose OS “Mongoose OS Features” [Online] Available: <https://mongoose-os.com/features.html> [Access: 13-October-2018].
- [6].Google Firebase Services “Firebase by Platform” [Online] Available <https://firebase.google.com/docs/> [Access: 10-October-2018].
- [7].Amazon Web Services “AWS IoT Core” [Online] Available: <https://aws.amazon.com/IoT-core/> [Access: 18–October-2018].
- [8].T.Mattison “AWS IoT on Mongoose OS” [Online] Available: <https://aws.amazon.com/blogs/apn/aws-IoT-on-mongoose-os-part-1/> [Access: 20–December-2018].
- [9].Blynk “Blynk APP” [Online] Available: <http://docs.blynk.cc/> [Access: 20–December-2018].
- [10].M.Tharaniya soundhari, Ms.S.Brilly Sangeetha, "Intelligent Interface Based Speech Recognition for Home Automation using Android Application", in 2nd International Conference on Innovations in Information Embedded and Communication Systems (ICIIECS), 2015.
- [11].Eclipse Foundation “Eclipse Mosquitto” [Online] Available: <https://mosquitto.org/> [Access: 10–March-2019].
- [12].Steves “Beginners Guide to the Paho MQTT Python Client” [Online] Available: <http://www.steves-internet-guide.com/into-mqtt-python-client/> [Access: 20–March-2019].

[13].Raspberry pi Documentation “Raspberry Pi 3b+” [Online] Available:
<https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/> [Access: 20-February-
2019].