COSMOS: A Ubiquitous Automation Solution

Final Year
Project Report

Submitted by

Jubeen Shah (70011014034)
Harshal Savla (70011014031)

Under The Guidance Of

Mr. Shreos Roychowdhury	Just Dial Limited	Chief Technology Officer (R&D)
Prof. Saurav Verma	MPSTME	Asst. Professor
Prof. Bhisaji Surve	MPSTME	Asst. Professor

In fulfillment for the award of the degree of

B.TECH.

INFORMATION TECHNOLOGY

At

Department of Information Technology
Mukesh Patel School of Technology Management & Engineering
NMIMS (Deemed –to-be University)
JVPD Scheme Bhaktivedanta Swami Marg,
Ville Parle (W), Mumbai-400 056.

April, 2018

CERTIFICATE

This is to certify that the project entitled "COSMOS: A Ubiquitous Automation Solution" is the bonafide work carried out by Jubeen Shah and Harshal Savla of B.Tech (IT), MPSTME, Mumbai, during the VIII Semester of the academic year 2017-2018, in partial fulfillment of the requirements for the award of the degree of Bachelors of Technology as per norms prescribed by NMIMS. The project work has been assessed and found to be satisfactory.

(Signature of External Mentor)
Name: Mr. Shreos Roychowdhury
Designation: Chief Technology Officer (R&D) at Justdial Limited

(Signature of Internal Mentor 1)
Name: Prof. Saurav Verma
Designation: Asst. Professor

(Signature of Internal Mentor 2)
Name: Prof. Bhisaji Surve
Designation: Asst. Professor

(Signature of External Examiner)
Name:
Designation:

HOD (IT) **Dean**

(Dr. Ketan Shah) **(Dr. N T Rao)**

DECLARATION

We, Jubeen Shah and Harshal Savla roll numbers: A032, A033 respectively, understand that plagiarism is defined as any one or the combination of the following:

1. Uncredited verbatim copying of individual sentences, paragraphs or illustrations (such as graphs, diagrams, etc.) from any source, published or unpublished, including the Internet.
2. Uncredited improper paraphrasing of pages or paragraphs (changing a few words or phrases, or rearranging the original sentence order)
3. Credited verbatim copying of a major portion of a paper (or thesis chapter) without clear delineation of who did or wrote what. (Source: IEEE, The Institute, Dec. 2004)

We have made sure that all the ideas, expressions, graphs, diagrams, etc., that are not a result of our work, are properly credited. Long phrases or sentences that had to be used verbatim from published literature have been clearly identified using quotation marks.

We affirm that no portion of our work can be considered as plagiarism and we take full responsibility if such a complaint occurs. We understand fully well that the guide of the seminar/project report may not be in a position to check for the possibility of such incidences of plagiarism in this body of work.

Signature:
Name: Jubeen Shah
Roll No.: A032
Date:

Signature:
Name: Harshal Savla
Roll No.: A033
Date:

ACKNOWLEDGEMENTS

TABLE OF CONTENT

List of Figures

List of Tables

1. Copyrights Notice and Disclaimer

1.1. Other Precaution

Please note that the technical specifications mentioned in the chapter are taken directly from the data sheet for the various sensors, for any form of reproduction of the work the official data sheet should be reviewed very carefully. The author would in no event shall in any way whatsoever be responsible for accidents happening due to the disregard of this precaution.

2. Synopsis

2.1. Abstract

Internet of Things (IoT) is a burgeoning field which is the next big thing in the world of automation. IoT devices usually have a universal application as they are not constrained to any one particular domain. It has a broad appeal in fields ranging from Hospitals, Offices, Hotels, Airports to even our very home. However, it has been known how expensive these individual devices tend to get. For example, a single Philips Hue bulb can cost anywhere between ₹6,000 to ₹7,000. A single household has multiple lamps, and the cost tends to exponentially increase while the willingness to pay for each of the bulbs by each of the consumers tends to decrease with each increment in the number of devices being bought.

The idea of COSMOS, in brief, is to make affordable custom IoT end-points, that can be managed via a single device or custom designed interface. These devices could be any iOS, Android or any smart device which can access the local wireless network in the desired automation environment. These devices can then manage the endpoints, most of which would be designed from scratch. Few examples of the managed end-points that will be developed are intrusion sensors, fire sensors, connected RGB LED Lights, Connected Switches & Switchboards, Connected Locks, integration of — temperature & humidity sensors Air Quality Sensor, UV Sensor, Air Pressure Sensor and so on.

The modules that have been described in this semester end report are:

1. Intrusion sensor
2. Fire sensor
3. Weather station
4. Ambient Station
5. Smart Lights
6. Connected Switches
7. Smart Lock
8. Window Blinds
9. Parking Sensor
10. Irrigation System

2.2. Concept and Plan

The idea as briefly stated earlier is to connect the Managed End-Points over the network in the installed environment via open platforms that are freely available for use through any personal device, which may include a smartphone, tablet or even any other device through which a user could access the internet. This document would be going through the entire process step by step. However, this particular chapter is to get the reader familiar with the core idea. Figure 1 Gives the block level view of the system.

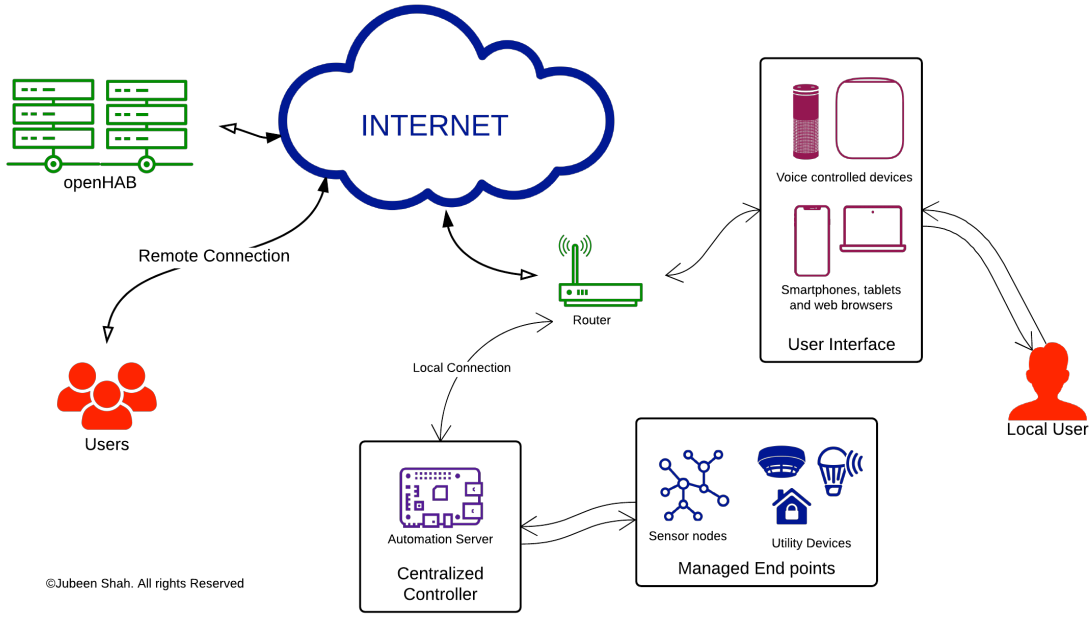

Figure 1 System Overview

2.3. System overview

As it can be seen in Figure 1 the system consists of four necessary components which are described along with the sub-components briefly below:

- Cloud: The cloud is one of the most crucial part of the system. It is the central part to which all the other components are connected and is also the gateway for the different elements to act and interact with each other, by sending information to each other and take actions based on the messages received from one device.

– Wifi Access Points: Nowadays there are usually two consumer options available for the access points, one is the faster 5GHz, and the other is 2.4 GHz option. Most devices are still not compatible with the 5GHz option. However, most modern smartphones and tablets are. The implementor of the system needs to make sure that the environment in which COSMOS is being installed or integrated with, has 2.4 GHz option available since the ESP8266 wifi modules that would be used to make the managed endpoints aren't 5GHz compatible. While prototyping the COSMOS, the author used an Apple Time Capsule with 3TB Server Grade HDD, with a Dual-Core Cortex A-9 CPU @ 1GHz. It uses internet standards 802.11 DSSS 1 & 2 Mbits/s standard, 802.11ac. It also has both the 5GHz and the 2.4GHz options available that makes it perfect for use. However, it must be noted that any access point with the 2.4 GHz option is more than enough for the system to work. It must also have USB 2.0/3.0 input port along with several Gigabit Ethernet ports for that would be used for connecting several devices to it directly. The implementor must ensure that reliable network connectivity is available

in the desired environment. Accordingly, additional Access points can be used. The author just needed one for prototyping. It is important to note down the SSID and set a secure password to the router, as it would be needed for each subsequently managed endpoints that would be integrated into the environment.

- Router: The router is the brain of the network as it assigns IP addresses and talks with the modem. This router also is one part of the network setup phase.
- Ethernet Switch: An Ethernet switch is needed only if there are several Router or modem that need to be physically connected to each other, in case of a more extensive house or an office set up, an Ethernet switch would be necessary to be able to manage all the connected network devices.

2. Managed End-Points: These components comprises of the connected devices that would be built for the COSMOS automation solution. They have been divided into two groups based on the function they would be performing. Also, this document would list down few devices that the author has thought of building for the project, however, it must be noted that this is not an exhaustive list, devices could be added/ subtracted from the list without a formal notice to either the university or Just dial Limited unless otherwise stated.

a. Sensors: These devices would send information to the automation server when an event occurs. Eg. Opening or Closing of a door/window. Few devices in this category are as follows:
   - Intrusion Sensor
   - Temperature and Humidity, Air-Quality, UV radiation, Air pressure Sensors
   - Fire Sensor

These devices would send a push notification to the authorized smart devices and be followed up by taking up specific actions. For example, if a door opens you might want the lights to turn on if it is evening or night time. Another use-case will be to automatically turn on the air conditioner if the temperature drops a certain threshold level.

b. Utility Devices: These are primarily any device in the house that can be automated. Some of these devices are
   - Connected Lock
   - Connected LED Strips
   - Connected Lights, Fan, Sockets
   - Window Blinds

3. User Interfaces: These are the different mediums through which the smart devices can be managed or monitored. As mentioned in Figure 1, and briefly stated earlier, the COSMOS automation system would be available with any device connected to the wifi network of the automation environment over any preconfigured IP address. Also, it could be accessed through an external interface from any part of the world, as long as it is connected to the internet. Additionally, Voice controlled devices such as Amazon's Alexa, or Apple HomePod, or the already available Siri on the iPhone can be used to control the devices. Finally, Wall mounted devices can be used to directly manage the smart devices which would have a touch screen and can be used to manage intelligent devices directly, would mainly be the switchboard replacement.

4. Server: The server that the author has used is a Raspberry Pi 3 with a heat sink installed, since the server would be running 24/7, additionally a fan would help keep the processor from overheating. After considering several open platforms for the Automation Server, such as Home Assistant, Calaos, OpenMotics and OpenHAB, It was found that OpenHAB was most suited.

3. Setting up the automation Server

3.1. Hardware Requirements

Assuming that the network connections are already in place, Table 1 gives the additional components needed for the process.

Sr.	Hardware	Cost	Link
1	Raspberry Pi 3	₹ 3,448.00	goo.gl/WG2eN1
2	RPi 3 Case with Heat Sink and Fan	₹ 620.00	goo.gl/frwyja
3	16 GB Micro SD card	₹ 610.00	goo.gl/u9ri6D

Table 1 Hardware requirements for setting up automation server

The entire automation server would be running on the Raspberry Pi 3 (RPi3) and is connected to the automation environment network. It hosts the OpenHAB 2 service as well as the Mosquitto MQTT (MQ Telemetry Transport) service. It is advisable to use a case to house the Raspberry Pi 3, and with heat sinks installed and a fan to cool off the RPi3. Heat sinks are needed since the server would be run on 24/7 throughout the lifetime of the project. The subsequent section gets into the details of installing the OpenHABian and Mosquitto MQTT service onto the Server.

3.2. Setting up the micro SD Card for the server

The initial steps in the project involve flashing a prebuilt image called OpenHABian as well as Mosquitto MQTT onto the RPi3 server. Once downloaded any suitable unarchiver software can be used to unarchive the files which house the image file that is to be used to flash onto the micro SD card. Once completed, the next step is to flash the image files onto the RPi3. There are several online software available for flashing, for Mac users — Apple Pi Baker, for Windows users — win32 disk imager. Once the software is downloaded, the onscreen instructions were helpful enough to install the software.

Once the SD card is flashed, RPi3 needs to be added to the Address reservation List in the Routers web page. The IP address has been assigned manually as 192.168.1.20 with its Mac address. Once the RPi3 is configured over the network, one can directly SSH into the server with its IP address and begin setting the server, such as the date and time and edit the password.

3.3. Testing the MQTT server

To test out whether the MQTT server is online or not, software — MQTT.fx can be used. This software will be needed in the future to test out other devices before finally physically installing them in my house. We only need to change the MQTT broker address to the IP address of the Raspberry Pi, for example 192.168.X.XX and just make sure that the port is 1883 which is the port for MQ Telemetry Transport (MQTT). Once that is done, we just need to do one final step, and that is to change the broker URL in the RPi3 to “broker.url=tcp://192.168.X.XX:1883”.

Once complete, only the final installation of the RPi3 physically in the network remains.

3.4. Design Details

This section entails the details regarding two aspects of the project — Design and Timelines. The Design Details section comprises of the Work Breakdown Structure (WBS) and the Product Breakdown Structure (PBS). The Timeline Details Section provides the Gantt charts for each of the modules in detail with emphasis on clarity being the primary objective of the project. This clarity was needed not only regarding deliverables but also to set milestones at each step.

3.4.1. Product Breakdown Structure (PBS)

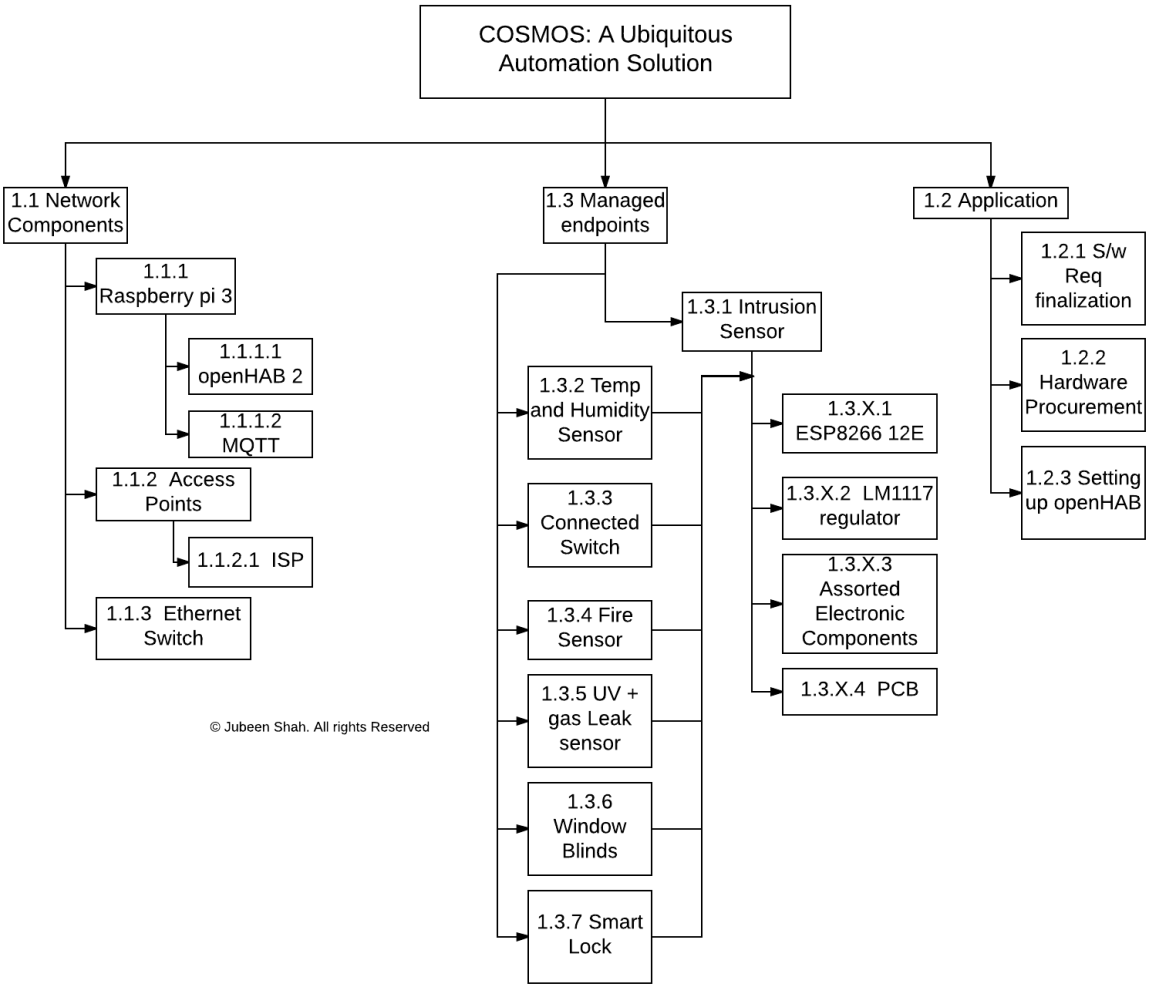

Figure 2 Product Breakdown Structure (PBS)

3.4.2. Work Breakdown Structure (WBS)

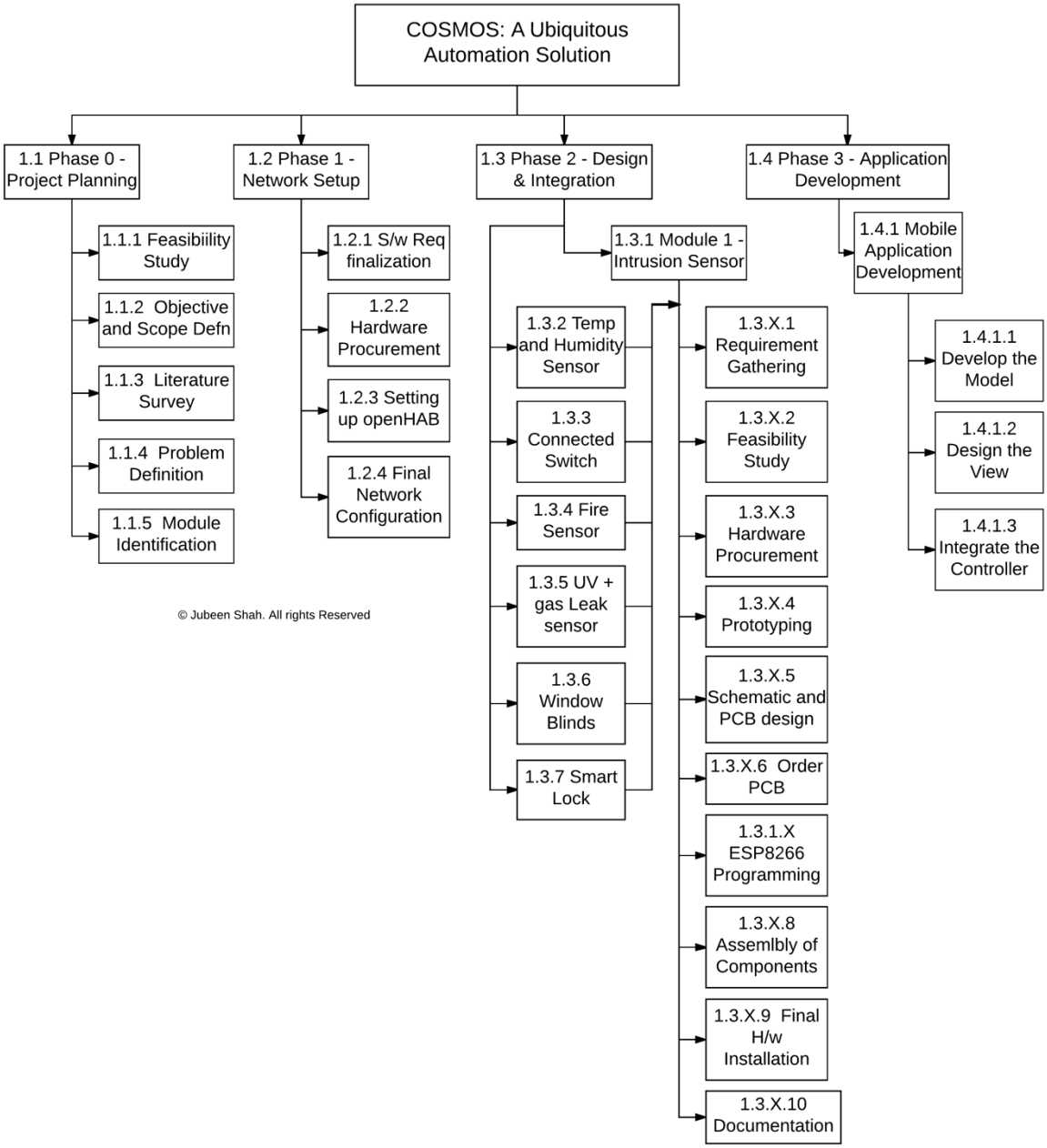

Figure 3 Work Breakdown Structure (WBS)

3.5. Timeline Reports

ID	Task Mode	Task Name	Duration	Start
2		COSMOS: An Ubiquitous Automation Solution	110.63 days	Fri 7/14/17
3		PHASE 1 : Network Setup	9 days	Fri 7/14/17
9		Phase 2: Design and Integration	96.5 days	Mon 8/7/17
120		Mobile Application	31.13 days	Mon 1/1/18
121		Final Documentation	7 days	Sun 3/25/18

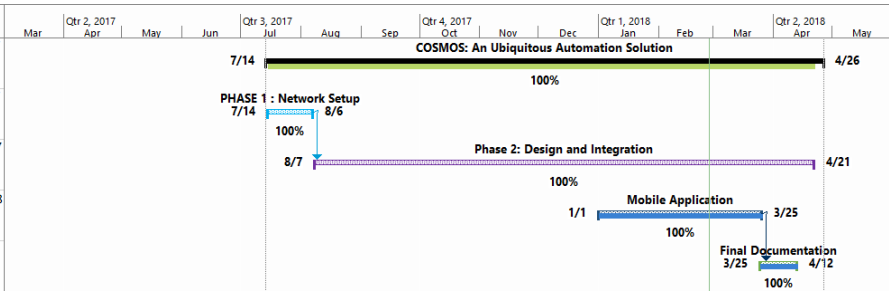

Figure 4 Overall Project Timeline

ID	Task Mode	Task Name	Duration	Start
2		COSMOS: An Ubiquitous Automation Solution	110.63 days	Fri 7/14/17
3		PHASE 1 : Network Setup	9 days	Fri 7/14/17
9		Phase 2: Design and Integration	96.5 days	Mon 8/7/17
10		Mod 1 : Intrusion Sensor	8.75 days	Mon 8/7/17
20		Mod 2 : Fire Sensor	22.5 days	Tue 8/22/17
30		Mod 3 : Weather Station	20.63 days	Sat 10/28/17
40		Mod 4: Ambient Station	10.13 days	Fri 12/15/17
50		Mod 5: LED Module	11.13 days	Thu 1/11/18
60		Mod 5 : Connected Switches	6.38 days	Fri 2/9/18
70		Mod 6: Smart Lock - I	12 days	Mon 2/26/18
80		Mod 7 : Window Blinds	11.38 days	Mon 1/1/18
90		**Mod 8 : Irrigation System**	**8 days**	**Wed 1/31/18**
100		**Mod 9 : Parking Sensor**	**12 days**	**Wed 2/21/18**
110		**Mod 10 : Smart Lock - II**	**10 days**	**Sun 3/25/18**
120		Mobile Application	31.13 days	Mon 1/1/18
121		Final Documentation	7 days	Sun 3/25/18

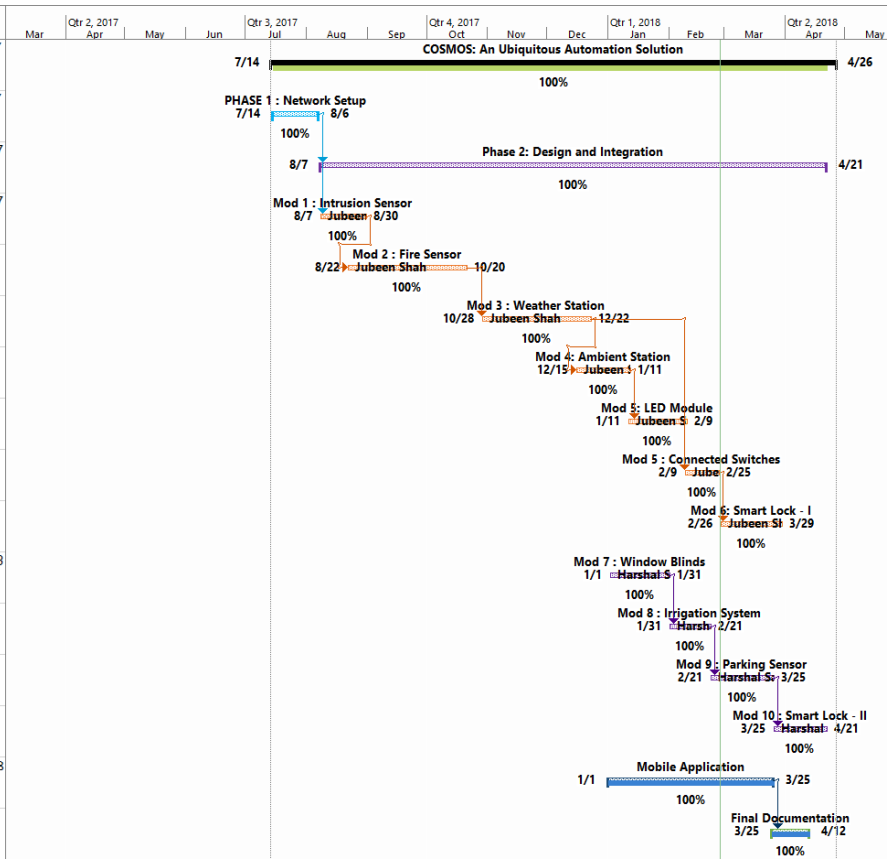

Figure 5 Detailed Project Timeline

4. OpenHAB configuration

OpenHAB (Open Home Automation Bus) is the central entity to an automation environment in which it is set up. All the functionalities and properties of the connected end-points in the environment are available to the end user for configuration. These configurations primarily comprise of the user interface, setting up the 'if-this-then-that' rule engine, and other parts. OpenHAB is an open platform installed and configured by the end user, that would run solitarily from any other online services. As a user, this gives them full control over all the aspects of their smart environment in which the platform is installed. COSMOS: A ubiquitous Automation solution, aims at utilizing this open platform for centralizing the operations and management of the endpoints.

Every device that could potentially be used to be augmented as an automation tool is logically and functionally very different from the others of its kind. OpenHAB defines several base components that COSMOS makes use of or would use. These components are of the following nature:

- Things — Abstract representation of the devices connected in the openHAB environment
- Items — Functionalities or capabilities of the Things
- Bindings — Add-ons to communicate with the devices
- Groups — Collection of Items according to a particular context
- Sitemaps — Definition of user Interfaces as visible to the end user
- Persistence — Services to store data over time

Task	Textual	Paper UI	HABmin	Console
Auto-Discover items	No	Yes	Yes	Yes
Define Things	Yes	Yes	Yes	Yes
Define Items	Yes	Yes	Yes	Yes
Define Groups	Yes	Yes	Yes	Yes
Define Sitemaps	Yes	No	No	No
Define Persistence	Yes	No	No	No
Define Rules	Yes	No	No	No
Modify openHAB	Yes	Yes	Yes	Yes

Table 2 openHAB component

4.1. Things

Things can be understood as the physical layer of the openHAB system; it constitutes of all the material entities. These physical objects include not only the devices but also consists of the web-services and other information sources. COSMOS makes use of all the physical entities to allow for as much integration of services and devices to provide the user a single consolidated view of all the essential information that the user may require to not only manage the automated environment but also act as a decision support system. For example, imagine air quality sensor embedded in the environment and an online localized air quality monitor service that is available openly, the user would not only have the air quality information in his environment but at the same time have access to the air quality information from his city or locality.

The process of defining a thing is shown in Figure 6 *Adding Things to COSMOS.*Things are connected to openHAB through bindings. An example of bindings that COSMOS makes use of is the MQTT protocol binding, which allows the device to communicate over the lightweight MQTT protocol. Items are used to manage the information and state changes of the Things, which allows this information to be accessible to the user. The Status Transition diagram is shown in Figure 7.

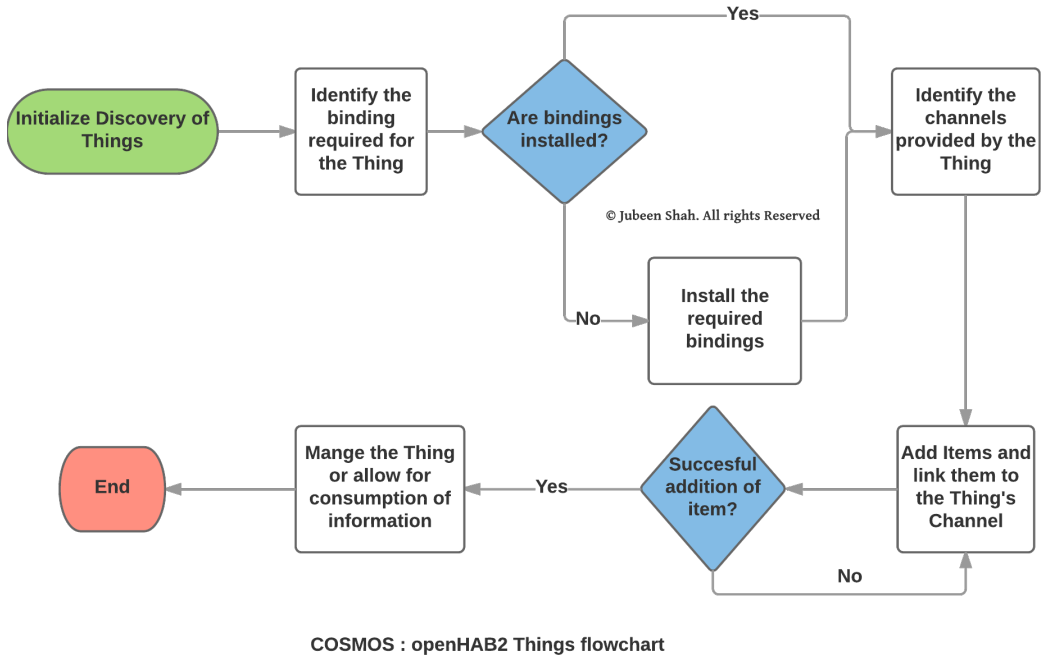

Figure 6 Adding Things to COSMOS

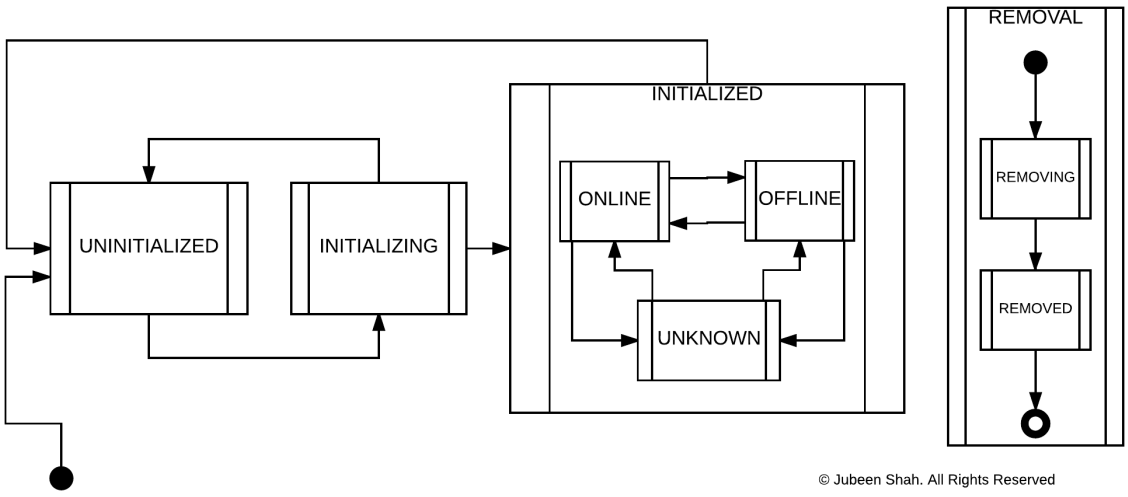

Figure 7 State transition

4.1.1. Defining Things

In COSMOS, things have been manually, by creating a **.things** text file. This text file is created directly on the automation server after we SSH into it. The location of the **.things** file, is done on the path: **/etc/openhab2/things/**. Using text files has some pros and cons. The advantage of configurable text files is to allow for a static definition of the Things, flexibility, and secure backup and restore. However, the main disadvantage is the effort involved in composing them and the probability of typing errors, as was faced in the project.

Syntax : Thing {binding_id} : {type_id} : {thing_id} “Label” @ “Location” [{parameters}]

4.2. Items

Items represent all the functionalities and capabilities of the Things. A unique feature of openHAB Items is that it allows for immediate connection to the outside world via Bindings. For example, an Item bound to a sensor receives updated sensor readings and an Item linked to a light’s dimmer channel can set the brightness of the light bulb. Similar to **.things** file, we need to create a **.items** file usually in its separate folder.

Syntax : itemtype itemname "labeltext [stateformat]" <iconname> (group1, group2, ...) ["tag1", "tag2", ...] {bindingconfig}

- Itemtype — Similar to data type of a variable in programming languages. It is used to define the type of state that can be stored in an item and which command it can allow. Several item types are made available that manage the various aspects of the Things, from Strings that store text to switch for switching on and off.
- Itemname — Like itemtype is a mandatory field and is a distinctive identifier for an Item. The naming convention is similar to that of any general programming language such as Java.
- Labeltext — This is the text that would be available to read to the user once, the Item is added and configured with openHAB.
- Stateformat — It describes the way in which a textual format is represented to the user in the user interface. For example, from the DHT22 sensor, we might get the following value as the current temperature — 25.789281727. However while displaying, we might just need precision up to 1 or 2 decimal points — 25.7 ˚C.
- Iconname — Use a particular bitmap file as an image for the item
- Group — Groups the items according to a predefined configuration
- Tag — Used for making an item identifiable by other foreign entities such as Alexa or HomeKit
- Bindingconfig — The channel description of an item

4.3. Sitemaps

Sitemaps are concerned with the visual representation of the Items and things as is visible to the user on the web-browser or the mobile application. Several elementTypes can be defined for the sitemaps, which would then be mapped to the .items and .things file for further details and configuration so that when an action is performed on the UI end of the application, the corresponding steps are taken/ performed at the backend to perform the necessary actions.

Please note that the icons and the corresponding labels will not match, since they're defined in the .items file. Sitemap defines what entities would be present in the UI, and how they're grouped. However, the backend connection with the actual things, is done by the .items file. Several elementType definitions are available ranging from images to media, to slider and switches. For example the below code snippet gives an example of the .items file.

5. Module 1- COSMOS: Intrusion Sensor

5.1. Introduction

The COSMOS: Intrusion Sensor is a simple but effective device that has been designed to detect and report suspicious movements in and around the place of installation. The idea behind the method is that it would be installed at all major and minor entry points and once armed, any unusual activity would be reported to the user via a notification on the device that has been configured with the OpenHAB server. The user can get this notification anywhere in the world, as long as the user is connected to the internet. It makes use of a custom Printed Circuit Board (PCB), along with an ESP8266 12E wifi module to make sure that the design is replicable and scalable for mass production.

The need for building this device is rather simple to understand, the use cases and the applications of the same are limited only by the imagination. The COSMOS: Intrusion Sensor, can give notification to the user and at the same time perform a subsequent activity (notify security personnel, sound an alarm, lockdown the facility or any other event that is predefined and configured). The need arises because the user would want to be notified about the suspicious activities and entries in real time so that the user can take some action against it before it's too late.

5.2. Use cases

HOME — The first and most obvious use case for the intrusion sensor is the place we live. We don't spend most of our time at our home in the day because we need to go to college, or go to work, or even sometimes when we go for a short or long vacation. However, at the same time, we feel the need to secure our home because of not only a sentimental attachment but also at the same time because of the monetary cost associated with the valuables inside the house like television, cash, jewelry, etc. The COSMOS: Intrusion Sensor, can not only be installed at the entry points of the doors and windows of the rooms but also at the same time installed at the specific locations which the user feels the need to secure such as drawers/ cupboards/ or a place where you store valuables. Imagine if the user leaves their house at the hands of a stranger such as a maid or any other houseworker, and the user does not wish that they open the doors of the cupboard where the user store their valuables. The COSMOS: Intrusion Sensor helps the user get notified when such unusual activities take place.

UNIVERSITY — In places where there are a lot of people, like a university, keeping an eye on everyone becomes increasingly difficult. There are some doors (server rooms) and especially some windows that you do not wish the students to open. If COSMOS: Intrusion Sensor is installed at such places any suspicious activities can be notified to the security personnel almost immediately and who can take the further actions necessary. Similarly, you do not want students to access cabins of the professor when the professor is not around. The professor can merely arm the sensor to detect and report any activity when he is not on his/ her desk.

OFFICES — Some places do still have their server rooms, and physical access should be allowed to authorized personnel. The COSMOS: Intrusion Sensor is useful even in such scenarios where physical access to some prohibited regions can be kept in check, and people notified if there is a security breach of some kind.

5.3. Functional Block Diagram

Figure 6 is the basic functional block diagram of the COSMOS: Intrusion Sensor device. The device comprises of four main components the ESP8266-12E wifi module, 5v power supply for the device, and 5v to 3.3v Regulator. The intrusion sensor when triggered communicates the values with the centralized controller, which then relays the corresponding message to the devices configured on receiving the notification.

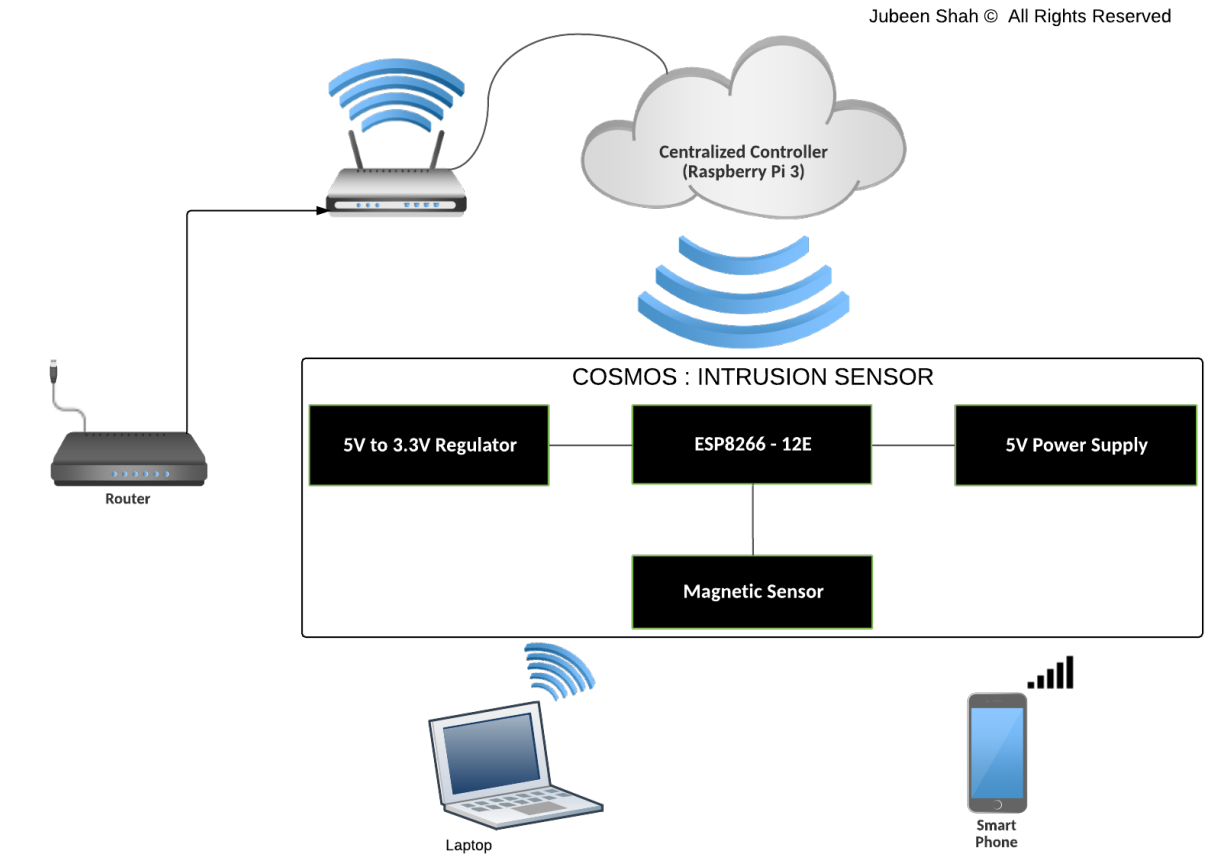

Figure 8 Functional Block Diagram of Intrusion Sensor

5.4. Schematic Design

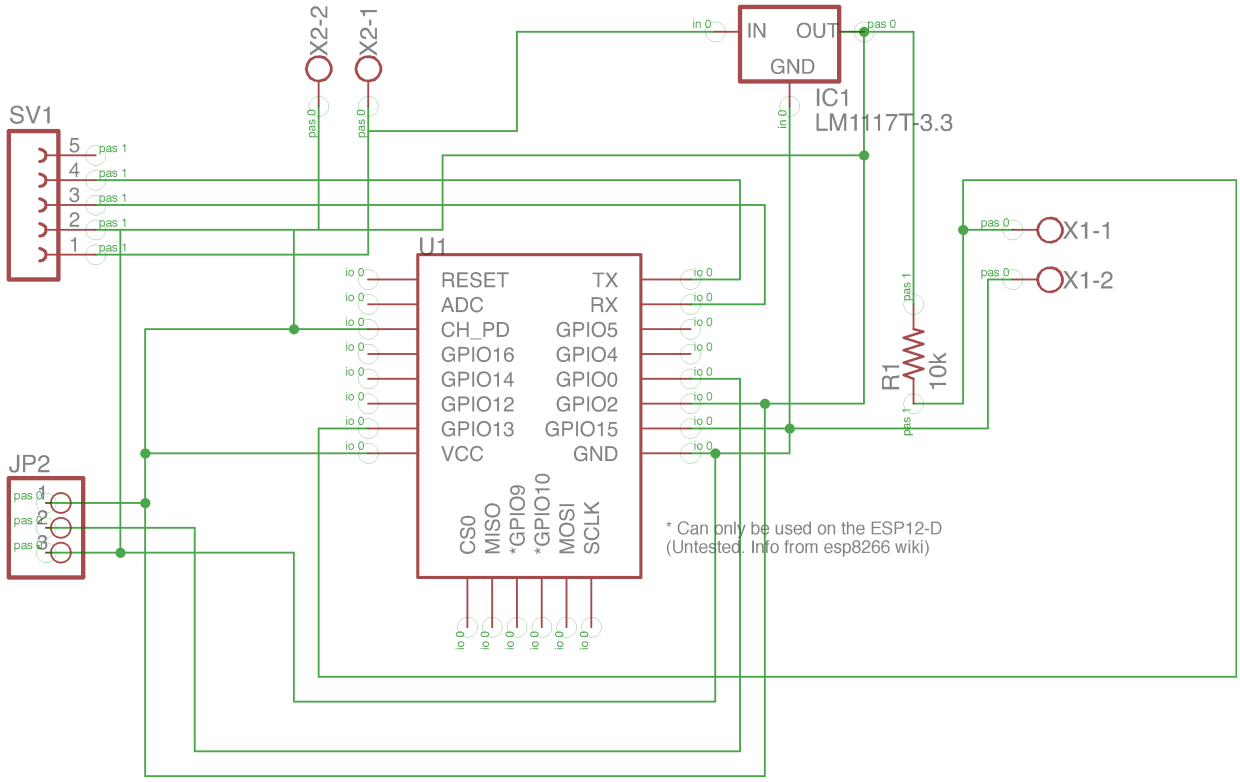

Figure 9 COSMOS: Intrusion sensor schematic diagram

The main components in the Figure 9 COSMOS: Intrusion sensor schematic diagram are as follows:

- SV1 — 4 pin Female Header
  - Pin 1 — GND
  - Pin 2 — VCC (5 volts)
  - Pin 3 — RX
  - Pin 4 — TX
- JP2 — 3 pin Male Header
  - Pin 1 + Pin 2 : Run Mode
  - Pin 2 + Pin 3 : Programmable Mode
  - Pin 1 — CH_PD and GPIO2
    - Pin 2 — GPIO0
      - Pin 3 — GND
- U1 — ESP8266-12E
- IC1 — LM1117T Voltage Regulator
  - In — 5 volts
  - Out — 3 volts
- X2-1/2 — Power Supply Module (5 volts)
- X1-1/2 — Magnetic Sensor

5.5. ESP8266 requirements and prerequisites

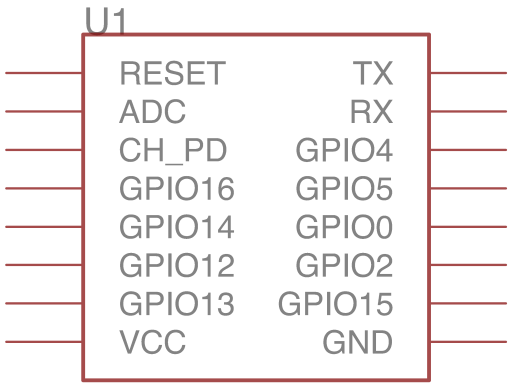

Figure 10 ESP8266 Schematic

The ESP8266 12E has the above schematic which consists of 9 GPIO pins for input and output of data values from and to the sensors. It has a Tx pin for transmission of data values and Rx pin for Receiving the data values. The VCC pin is for voltage input, which must be 3.3 volts. It should be noted that any voltage above 3.3 volts increases the risk of burning down the chip. The ESP8266 also has the limitation of working with just 2.4 GHz of network, any 5 GHz connection would not work.

5.6. Software requirements

5.6.1. EAGLE

EAGLE, which stands for Easily Applicable Graphical Layout Editor is a scriptable Electronic Design Automation (EDA) software application with schematic capture, printed circuit board (PCB) layout, auto-router and computer-aided manufacturing (CAM) features. This particular software seemed the best for the project for schematic and PCB design since it has an inbuilt schematic editor for circuit diagrams. EAGLE has been used because it not only gives an easy to use interface but also has several open source libraries available for our purpose. These libraries include that for the ESP8266, the LM1117 5v to 3.3v Regulator and in future for other components and parts that may be required. After finalizing the Gerber file layout to be used for printing the PCB via an online PCB manufacturer PCBPOWER.

5.6.2. MQTT.fx

MQTT which stands for MQ Telemetry Transport is a lightweight publish/subscribe, straightforward messaging protocol. This protocol is specially designed for constrained devices and low-bandwidth with high latency and unreliable network. So, the design principles are to minimise network bandwidth, and device resource requirements whilst also attempting to ascertain the reliability and some level of acknowledgment of delivery. As per our research, MQTT is by far the most competitive solution out there for a machine to machine (M2M) or Internet of Things (IoT) application. COSMOS revolves around the core idea of having a single interface with multiple sensor nodes and utility devices connected to it. It was found necessary to find a protocol that sits over the standard TCP/IP protocol to standardize the solution, while at the same time account for the unreliable networks and provide message transport with low-latency.

MQTT was also chosen for its support for standard security protocols. You can pass the username and password with an MQTT packet in the latest V2.1 of the protocol. Encryption across the network can be handled with Secure Socket Layer, independently of the MQTT protocol itself. Since it is a lightweight protocol, any additional security measure can be coded into the packets from the application end. This addition of security was done to ensure that the protocol follows its vision of enabling a product that can be used on any network. No additional

security has been encoded into the protocol. Though security is of primary concern for any project, the primary objective is to get the Proof of Concept (POC) ready for deployment and demo first, and later security measures can be added as a part of another future project, either by us or any other group who would like to work with it.

MQTT.fx is an open source software, available on both macOS and Windows OS. This software is primarily a JavaFX based MQTT client written in Eclipse Paho. We have used MQTT.fx for preliminary testing purpose when the ESP8266 programming is completed, and the finished module is ready to transmit MQTT messages to our centralized controller, i.e., Raspberry Pi 3.

5.7. Uploading code to ESP8266

Referring back to Figure 10, connections for the Rx and Tx of the Arduino board to the Tx and Rx of the ESP8266 via the pins 4 and 3 of the SV1 female header are made, and is seen in figure 30. Connections to the pin 1 to the GND of the Arduino and pin 2 to the VCC (+5v) of the Arduino. Once the connections are made, and the next step is to select the tools command in the Arduino IDE and the different configuration as per Table 3, shown below

Setting	Configuration	Description
Board	Generic ESP8266 Module	Board in use
Flash Mode	DIO	Dual IO — Uses 2 lines for data
Flash Frequency	40 MHz	Affects the Wifi and BLE functionality
CPU Frequency	80 MHz	Default value, but can be overridden.
Flash Size	512K (64K SPIFFS)	SPI Flash Filing System
Upload Speed	9600/ 57600/ 115200	The baud rate at which the upload takes place.
Port	(Depends on the OS)	Needed to configure the ESP8266 via the Arduino UNO

Table 3 Arduino IDE configuration

5.8. Problems and issues

One of the most common while uploading the code to ESP8266 12-E is the espcomm_uploaded_mem failed. Several reasons were identified after carrying out the tests in multiple iterations

1. Upload Speed much higher than what ESP8266 can handle
2. The port is nonoperational
3. Not enough power is being supplied to the module
4. The ESP8266 is not booted into programmable mode Tx and Rx pins are interchanged

5.9. Solutions tried and tested

1. Choose a lower serial speed selection, preferably, 9600. In the tests, the upload took significantly more time around 5-7 minutes if the serial speed selection was 9600 however at the same time the success rate was around 80-90% that the code got uploaded.
2. Make sure that the port drivers are installed and operational based on your operating system. With macOS, the steps were particularly tedious since the CH430 drivers aren't readily available.
3. Add additional power to the ESP8266. We did this by adding another 5v power source in serial with the additional input while programming it. We first tried the 3.3 v power output from the Arduino, and when that didn't work added another 5v on top of that.
4. The GPIO 0 pin from the ESP8266 has to be grounded just before power is being supplied to it, ensuring that the ESP8266 boots into a programmable mode. If the ESP8266 is not in the programmable mode, the upload tends to fail.
5. Finally, it is a crucial point, to not mix up the Tx and Rx pins on the Arduino and the ESP8266.

5.10. Final hardware installation

Figure 11 COSMOS: Intrusion Sensor

6. Module 2- COSMOS: Fire Sensor

6.1. Introduction

COSMOS: Fire Sensor is a device which augments and makes the already set up fire detection system connected to the internet. It relates to the COSMOS centralized controller which is the raspberry pi 3 through MQTT. Its primary task is to tell the centralized controller the state of the fire detector. The state of the fire detector would be monitored continuously, and any changes to the state would be notified to the user via a smartphone push notification. COSMOS: Fire Sensor makes use of the ESP8266 - 12E wifi module, with a custom PCB design, to make sure that the module is replicate and scalable for mass production. This chapter covers the use cases of the fire detector, the functional block diagram, the hardware requirements, the schematics design, and the PCB design. Finally the software requirements and the programming of the COSMOS: Fire Sensor are also discussed.

COSMOS: Fire Sensor is a device that could be potentially very useful in notifying the users remotely in case the fire detector is triggered due to fire in the place of installation. The device can be used to not only inform the user with a push notification but also at the same time perform a subsequent activity. For example, sending a message to the fire department via a GSM module, posting a tweet on twitter to notify the fire department, sounding an alarm to evacuate the facility, sending the state change information to the most critical users to help confine the fire. The need arises so that necessary steps can be taken in real time to mitigate the effects before it's too late.

6.2. Use cases

HOME — Most of the people today commute to work. Any electrical failure or short circuit that could lead to fire can burn down the house if necessary steps are not taken in advance. In most urban settings, the fire department would be notified after the blaze has grown to a considerable size after it has been noticed by the neighbors. In such cases, the latency in the communication would have already caused catastrophic damages to the setting. In the countryside, in most countries like Switzerland or alike, or even in remote locations in Mumbai, such as Ambey valley, where not a lot of people live in the surrounding area, it could become too late that the fire department is brought into the picture. COSMOS: Fire Sensor would help you get notified instantly when the fire alarm is triggered so that you can inform the fire department quickly.

OFFICES and UNIVERSITIES — In large establishments such as offices and universities with substantial campus settings, it is particularly important to get notified about the source or origin of the fire. This is needed to slow down the combustion to reduce the damages. Since the existing system, triggered all the alarms in the installed environment, identifying the source of fire becomes increasingly difficult with larger establishments. COSMOS: Fire Sensor would have its unique id, and can be configured to notify the exact starting point of the fire. For example in MPSTME, there are seven floors and two basement floors; installing a COSMOS: Fire Sensor on each level in each of the faculty area, we would be able to pinpoint the exact floor and the region as to where the fire originated.

FACTORIES — Factories are usually established in very remote locations, where access is very unfeasible. COSMOS: Fire Sensor in such establishments can help not only to notify the authorities for evacuation, but also at the same time inform other critical operations to shut down their activities before irreplaceable damage has occurred. This information can be also used to restrict the fire.

SMART CITIES — A recent move by the UAE government requires all the buildings in UAE to have mandatory connected fire alarm systems. The government plans to initiate the

movement, and have the first wide-scale connected fire alarm systems by the beginning of 2018, with all the projected installations to be complete by 2023. Gulf News society has said the following "A centralized fire-alarm receiving center that connects all UAE buildings, residential, commercial and villas, will be soon be introduced in the country." Having more than 150,000 connected existing buildings, the project is a huge undertaking by the government. These systems are but a use case of COSMOS: Fire Sensor, and the potential the system has use-cases in major and minor projects alike.

These use cases are just a small collection of scenarios where the COSMOS: Fire Sensor could come of use. Since the goal of the project is to be as general and universal as possible, they (use-cases) are limited only by the imagination.

6.3. Functional block diagram

Figure 12 gives the basic overview of the functional block diagram of COSMOS: Fire Sensor. The device contains four separate major components — ESP8266 12E wifi module, the 5V power supply module, the 5V to the 3.3V regulator and the PC 817 Optocoupler. The assumption that we've made with the use of the fire alarms is that the ones that are compatible are the hard-wired fire-alarms with a feature called interconnect. With the use of this interconnect feature when one of the fire alarms goes off, all the others connected to it goes off too. Aforementioned is made possible by the use of a cable used to transmit a 9-volt pulse to all the connected fire alarm systems, COSMOS: Fire Sensor (CFS) leverages this much functionality in the design. The CFS waits for this 9-volt pulse from the already existing fire detector system, and when it receives this pulse, CFS changes its state and consequently sends the MQTT message to the centralized RPi3 controller. This controller then broadcasts this information to all the connected devices via push notifications. Correspondingly, the RPi3 can also be used to trigger other connected modules such as the sprinkler system or posting this information on social media to inform authorities.

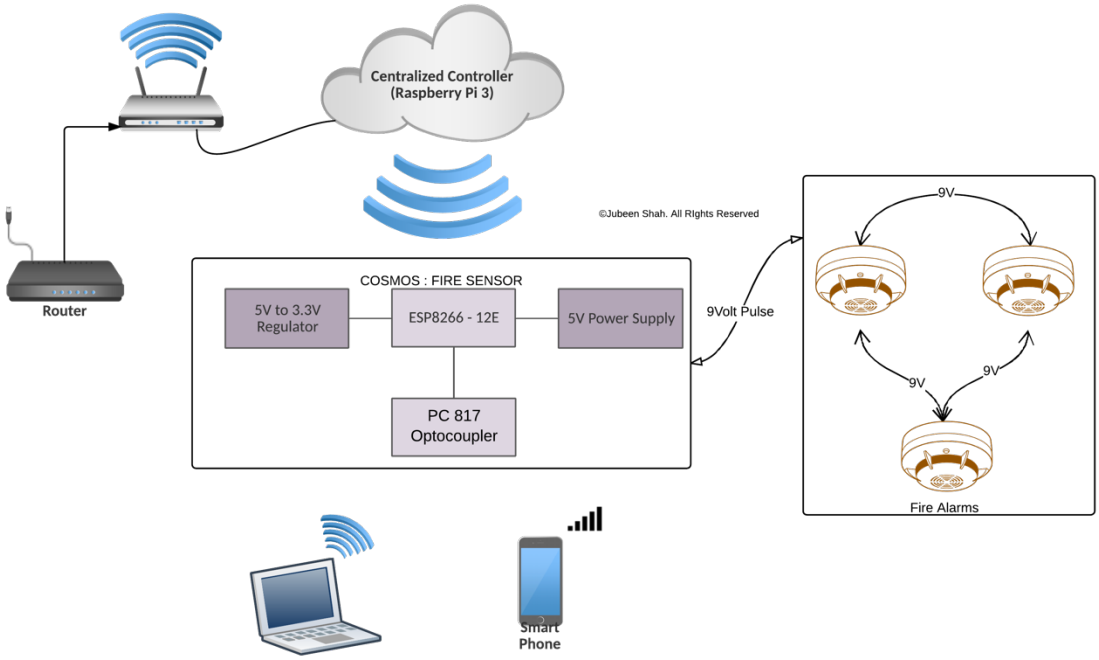

Figure 12 Functional block diagram of Fire sensor

6.4. Flowchart of COSMOS: Fire Sensor

Figure 13 is the flowchart diagram of how the system would function. Once the CFS is initialized, it would repeatedly attempt to connect with the MQTT broker, which is the centralized controller until the connection is established. Once connected, it would publish the initial state of the of the detector as "No Fire detected." Then it would continuously monitor, with a pre-configured delay (in case of CFS it is 10 seconds), whether a 9-volt pulse has been sent by the fire detector. If at any point it detects a 9-volt pulse, in which case it means that the fire alarm has been triggered, it would send an MQTT message to the broker which consequently broadcasts a "Fire detected" message over all the authenticated pre-configured devices. Once the necessary actions are taken, and the issue is resolved, the system turns itself back to the initial state and continuously monitors the circuit for another 9-volt pulse from the fire detector.

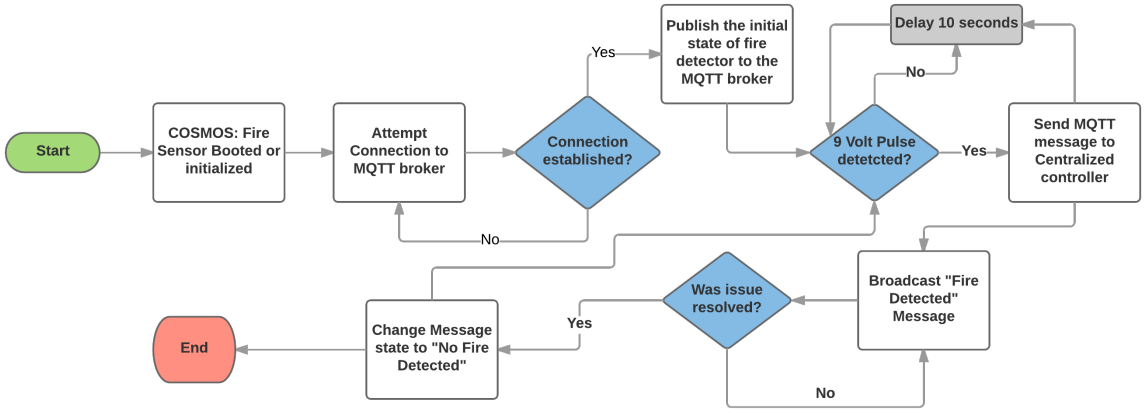

Figure 13 Flowchart of COSMOS: Fire sensor

6.5. Hardware requirements

This section focuses on the list of the hardware components required for the COSMOS: Fire Sensor. It would also provide the design perspective of the COSMOS: Fire Sensor system concerning the initial breadboard design, the schematic design, and the final PCB design. Note: The PCB costs are high because the order for the PCBs aren't done in bulk. The cost of per unit PCB can be dropped to as little as ₹16, dropping the overall per unit cost of the device to around ₹ 800.

Sr.	Componenet	Quantity	Link	Cost In Rs
1	ESP8266	1	https://goo.gl/h776Tx	₹ 319.00
2	PC 817 OptoCoupler	1	https://goo.gl/Deo1mu	₹ 119.00
3	Custom PCB	1	https://www.pcbpower.com	₹ 397.00
4	10K Resistor	1	https://goo.gl/AS2Q5p	₹ 2.23
5	1K Resistor	1	https://goo.gl/AS2Q5p	₹ 2.23
6	330 Ohm Resistor	1	https://goo.gl/AS2Q5p	₹ 2.23
7	LM1117 5v to 3.3V	1	https://goo.gl/x6LicQ	₹ 199.00
8	Plastic Enclosure	1	https://goo.gl/KWbexZ	₹ 150.00
	Total	8	**Total**	₹ 1190.69

Table 4 Fire sensor hardware component list and cost

6.6. Design

In this section we would focus on the design perspective of the entire Intrusion system from the schematics, Preliminary Breadboard design, PCB design. This is a crucial part of the project after the hardware procurement sub-task is completed.

Label	Part Type	Properties
J1	Generic female header - 2 pins	pin spacing 0.1in (2.54mm); pins 2; form ♀ (female); row single; package THT; hole size 1.0mm,0.508mm
J3	Dagu DGServo 9g header (male)	pin spacing 0.1in (2.54mm); variant - male pins; pins 3; form ♂ (male); row single; package THT
J5	Camdenboss CTB0158-2	pin spacing 0.2in (5.08mm); variant 90° 2 connector; pins 2; package THT; hole size 2.7mm; part # CTB0158-2
J6	Camdenboss CTB0158-2	pin spacing 0.2in (5.08mm); variant 90° 2 connector; pins 2; package THT; hole size 2.7mm; part # CTB0158-2
J7	Generic female header - 2 pins	pin spacing 0.1in (2.54mm); pins 2; form ♀ (female); row single; package THT; hole size 1.0mm,0.508mm
R1	1kΩ Resistor	tolerance ±0.05%; resistance 1kΩ; package 2010 [SMD]
R2	330Ω Resistor	tolerance ±0.05%; resistance 330Ω; package 2010 [SMD]
R3	10kΩ Resistor	tolerance ±0.05%; resistance 10kΩ; package 2010 [SMD]
U1	ESP8266 WiFi Module	variant variant 7; part # ESP8266
U2	Voltage Regulator	variant side; voltage 5V; package 78xxl; chip 78005
U3	Sharp PC817	part PC817; package THT

Table 5 Part list and Description for COSMOS: Fire sensor

6.6.1. Breadboard design

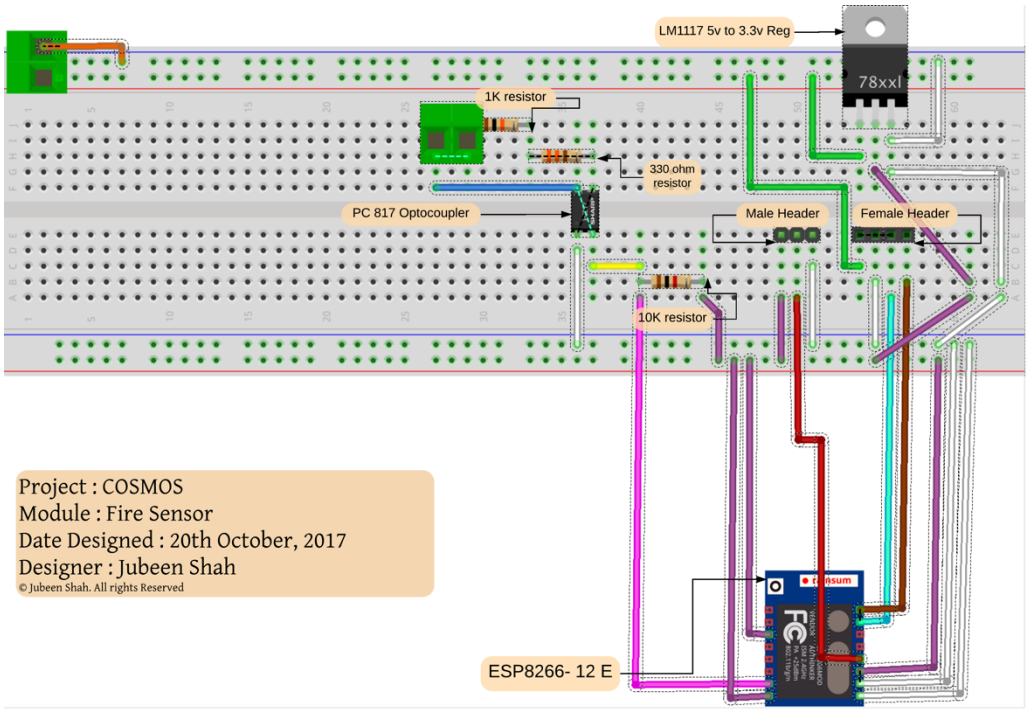

Figure 14 Breadboard design of COSMOS: Fire sensor

6.6.2. Schematic design

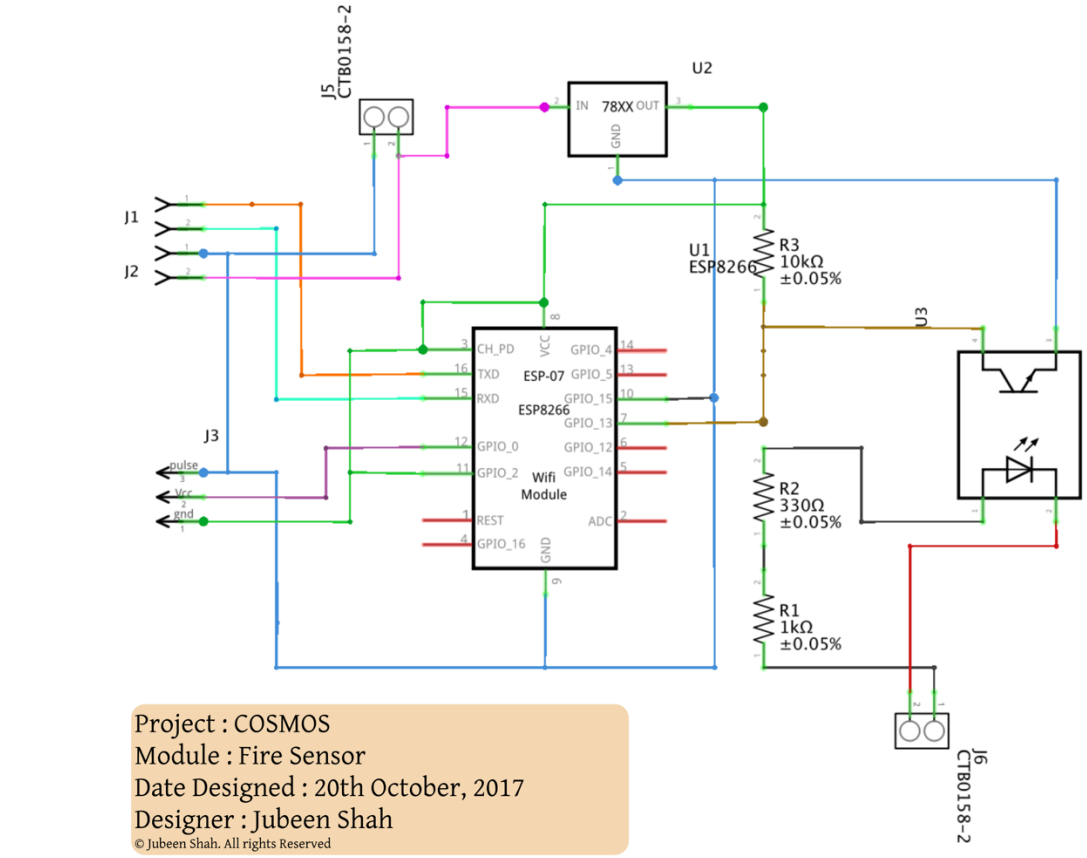

Figure 15 Schematic design of COSMOS: Fire sensor

6.7. Final hardware installation

Figure 16 Final Hardware installation of COSMOS: Fire sensor

7. Module 3- COSMOS Weather Station

7.1. Introduction

This chapter covers the design and integration of the third module of COSMOS — Weather Station. This module integrates three sensors — Temperature & Humidity Sensor (DHT11, DHT22), Gas Leak Sensor (MQ2) and the Passive Infrared (PIR) Motion Detection Sensor. It is not necessary for these devices to be implemented together, they can be installed as separate modules if needed. The primary task of this module is to announce the sensor readings to the Rpi3 by sending MQTT messages. Depending on the use case these ratings can be relayed as often or as less frequently as desired. The threshold values can be manually set, and if at any time the threshold values are reached, appropriate messages would be broadcasted to the authorized users. COSMOS: Weather Station makes use of the ESP8266 - 12E wifi module, with a custom PCB design, to make sure that the module is replicate and scalable for mass production. This chapter covers the use cases of the Weather Station, the functional block diagram, the hardware requirements, the schematics design, and the PCB design.

COSMOS: Weather Station is a utility device which relays relevant information to the end user — Temperature & Humidity, Air Quality (Gas-Leak), UV Levels, and Ambient Light levels. Even though these devices are relatively affordable, their current market prices are relatively high, and they have to be purchased separately. Also, the available devices do not have to capability to alert the user by sending a notification on their mobile phones. This COSMOS: Weather Station module will not only integrate these sensors but also send notifications to the user's devices if they're triggered and for a comparatively low price.

7.2. Use cases

Gas Sensor - In large-scale restaurants or even in factories, usually make use of natural gasses for combustion. In such environment, it becomes of utmost importance to ensure the safety of the people present. Moreover, in the consumer environment, most of the people have a stove or a water heater which run on natural gas. These devices are tested rigorously, and the probability of a defect is low. However, most of the incidences that occur are because of human fault. A lot of people sometimes forget to turn off the stove or install the devices incorrectly. The gas sensor would help to identify gas leaks and notify the administrator or any authorized user about the leak to allow them to take the required steps to contain it.

Motion Sensor – An office or a factory may have specific areas for which access is granted only to authorized personnel. The motion sensor will serve as an additional security checkpoint and will notify the administrator when motion is detected in a specific area. For example, individual server rooms have a secure perimeter, and only authorized personnel are allowed inside this boundary. The device will trigger if there's any unwarranted presence of an employee or intruder in this area and send a notification to the administrator to allow him/her to take necessary actions.

Temperature and Humidity - The DHT11 or DHT22 sensors detect temperature and humidity and pushes this information to the central server. It can be used to identify unprecedented changes in temperature in a particular area. Museums have artifacts which require a regulated temperature. This device can help in monitoring the temperature and alert the administrator as soon as the temperature or humidity goes beyond the threshold value so that he/she can take the necessary actions required to preserve these artifacts.
COSMOS: Weather station is a medley of different sensors, and since multiple sensors are part of the same device, any permutation and combination can give us additional insights into

the data being provided. For instance, if both temperature sensor and gas leak sensor get triggered at the same time, it can be inferred that there is some probability that fire has erupted from gas-leak, hence turning on the sprinkler system (if present in the automation environment) would only aggravate the problem. Figure 37 gives the use-case diagram for the partial weather station module.

7.3. Functional block diagram

Figure 17 shows the functional block diagram of the COSMOS: Weather Station. It makes use of the ESP8266-12E Wemos D1 chip to communicate with the centralized controller. The sensors which are a part of the weather station are connected directly to the ESP8266-12E WiFi chip as shown in the block diagram.

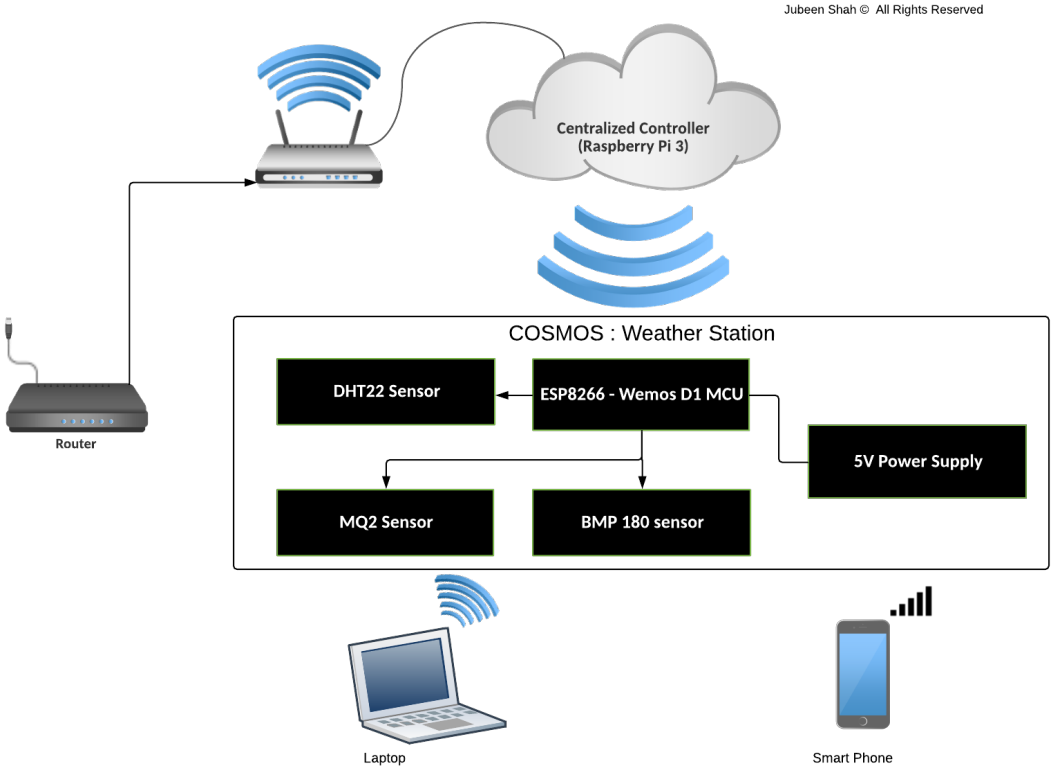

Figure 17 Functional block diagram of COSMOS: Weather station

ESP8266 chip when receives the sensor values relays the information to the centralized controller. When the RPi3 receives these values, it checks for the rules set in the .Rules file and based on the predefined conditions takes the necessary actions. These actions could be, but are not limited to:

- Send Push-Notifications
- Take Event-based actions
- If-This-Then-That Integration to interact with other modules

Once the relevant information is relayed, this information would be available to the authenticated and authorized users via the different web interfaces that were described earlier in chapter 3 of this project report. The user can also continuously monitor these values, for instance, the temperature and humidity readings directly on his smartphone and take intuitive actions based on the data interpreted. Since the module also integrates a database for persisting the values, there would be an added path in the functional block diagram to make use of that facility.

7.4. Flowchart of COSMOS: Weather Station

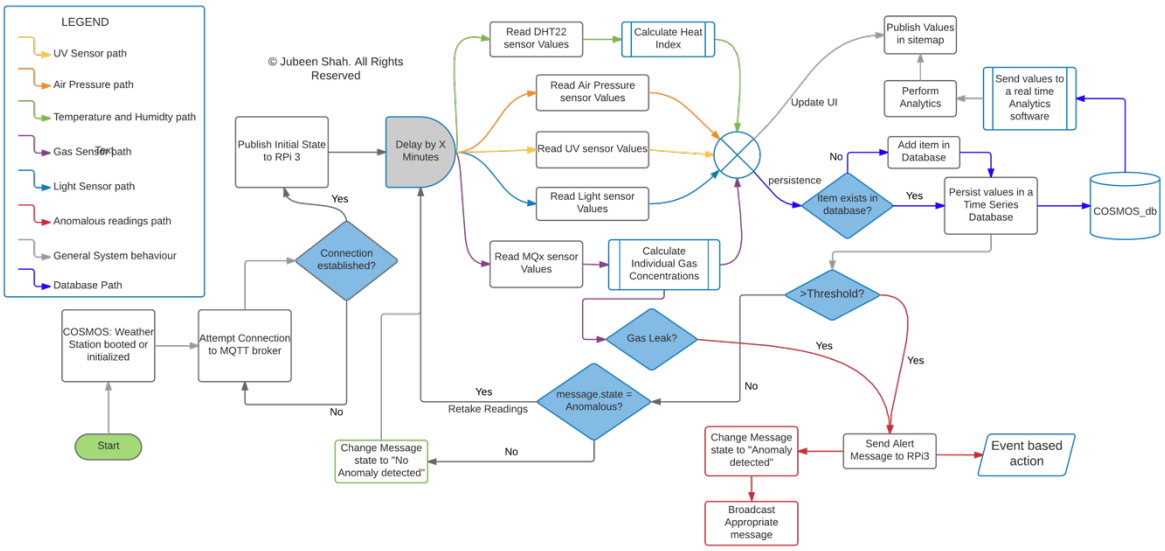

Figure 18 Flowchart of COSMOS: Weather Station

The COSMOS: Weather Station (CWS) initially boots up and repeatedly tries to connect with the MQTT broker, RPi3, till a successful connection is established. Once the link is confirmed, the RPi3 would publish the initial state of CWS. It would then monitor the values relayed by the different sensor either continuously or at regular intervals, depending on the sensor. In case of the DHT22 sensor, MQ2 sensor ,and the BMP180 sensor, the data would be consistently read and relayed every two minutes. The centralized controller would then change the state of the system to "Anomaly Detected" and the corresponding alert messages would be broadcasted to notify the authorized individuals. The system will then monitor if the issue that was risen by the CWS is resolved. If not, the system would re-affirm the presence of the problem by prompting the corresponding sensors to resend the values to double-check the occurrence of the detected error. If the issue were resolved, the controller would change the state of the system to "No Anomaly Detected".

7.5. Hardware requirements

This section focuses on the hardware requirement for the CWS module. It describes the list of the components required, a link to where to buy them, and the cost associated with those parts. The section then moves on to the schematic diagram for each of the sensors connected to the ESP8266, followed by the breadboard design.

Sr.	Component	Quantity	Price
1	ESP8266 - Wemos d1 mini	1	₹ 467.50
3	MQ2/ MQ3	1	₹ 190.00
4	DHT11/ DHT22	1	₹ 231.00
5	BMP 180 Sensor	1	₹ 240.00
6	Custom PCB	1	₹ 313.00
7	Plastic Enclosure	1	₹ 150.00
	Total	6	₹ 1591.50

Table 6 List of hardware components for COSMOS: Weather Station

Label	Part Type	Properties
220 Ohm	220Ω Resistor	bands 4; pin spacing 400 mil; resistance 220Ω; package THT; tolerance ±5%
R2	220Ω Resistor	bands 4; pin spacing 400 mil; resistance 220Ω; package THT; tolerance ±5%
R3 - 10K	10kΩ Resistor	bands 4; pin spacing 400 mil; resistance 10kΩ; package THT; tolerance ±5%
Red	Red (633nm) LED	color Red (633nm); package 5 mm [THT]; leg yes
s1	#4 Stand Off	package stand-off
Yellow	Yellow (595nm) LED	color Yellow (595nm); package 5 mm [THT]; leg yes

Table 7 Part list and description for COSMOS: Weather Station

7.6. Exploring Datasheets

7.6.1. DHT22

DHT22 output is a calibrated digital signal. It utilizes exclusive digital-signal-collecting-technique and humidity sensing technology, assuring its reliability and stability.Its sensing elements is connected with 8-bit single-chip computer. Small size & low consumption & long transmission distance (20m) enable DHT22 to be suited in all kinds of harsh application occasions.

Part	Description
Model	DHT22
Power supply	3.3-6V DC
Output signal	digital signal via single-bus
Sensing element	Polymer capacitor
Operating range	humidity 0-100%RH; temperature -40~80Celsius
Accuracy	humidity +-2%RH(Max +-5%RH); temperature <+-0.5Celsius
Resolution or sensitivity	humidity 0.1%RH; temperature 0.1Celsius
Repeatability	humidity +-1%RH; temperature +-0.2Celsius
Humidity hysteresis	+-0.3%RH
Long-term Stability	+-0.5%RH/year
Sensing period	Average: 2s
Interchangeability	fully interchangeable
Dimensions	small size 14*18*5.5mm; big size 22*28*5mm

Table 8 DHT22 - Technical Specifications

Item	Condition	Min	Typical	Max	Unit
Power supply	DC	3.3	5	6	V
Current supply	Measuring	1		1.5	mA
	Stand-by	40	Null	50	uA
Collecting period	Second		2		Second

Table 9 DHT22 Electrical Specification

7.6.2. BMP180

The BMP180 is a high precision digital pressure sensor with an I²C interface for easy and fast integration with microcontroller. It has ultra-low power, low voltage electronics which are optimized for use in mobile phones, PDAs, GPS Navigation devices and outdoor equipment. It is equipped with piezo-resistive technology for EMC robustness, high accuracy and long term stability.

Parameter	Condition	Min	Typical	Max	Units
Operating Temperature	Operational	-40		+85	°C
peak current	during conversation		650	1000	µA
Standby current	at 25°C		0.1		µA
Relative accuracy pressure	950…1050 hPa at 25°C		±0.12		hPa
Absolute accuracy pressure	300 … 1100 hPa 0…+65°C	-4.0	-1.0	+2.0	hPa
Resolution of output data	pressure		0.01		hPa

Table 10 BMP180 Electrical Charachteristics

7.6.3. MQ2

MQ2 gas-sensor makes use of SnO_2, which has lower conductivity in clean air. When different gasses are present in the higher concentration, the conductivity of the sensor increases. This electrical conductivity can be mapped to the output signal of the gas concentration. MQ2 Gas sensor has high sensitivity to LPG, Propane and hydrogen, and also other combustible steam.

Model No.			MQ-2
Sensor Type			Semiconductor
Standard Encapsulation			Bakelite (Black Bakelite)
Detection Gas			Combustible gas and smoke
Concentration			300-10000ppm (Combustible gas)
Circuit	Loop Voltage	V_C	≤24V DC
	Heater Voltage	V_H	5.0V±0.2V ACorDC
	Load Resistance	R_L	Adjustable
Character	Heater Resistance	R_H	31Ω±3Ω(Room Tem.)
	Heater consumption	P_H	≤900mW
	Sensing Resistance	R_S	2KΩ-20KΩ(in 2000ppm C3H8)
	Sensitivity	S	Rs(in air)/Rs(1000ppm isobutane)≥5
	Slope	α	≤0.6(R5000ppm/R3000ppm CH4)
Condition	Tem. Humidity		20°C±2°C;65%±5%RH
	Standard test circuit		Vc:5.0V±0.1V; V_H: 5.0V±0.1V
	Preheat time		Over 48 hours

Table 11MQ2 Technical details

7.7. Design

7.7.1. Breadboard design

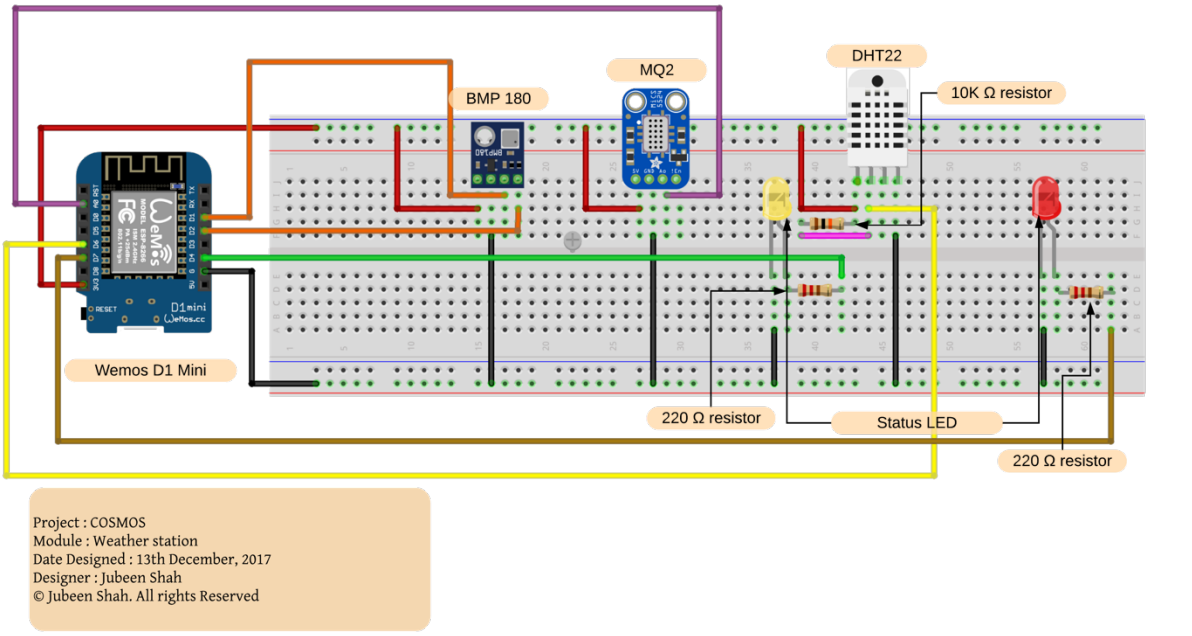

Figure 19 Breadboard design of COSMOS: Weather Station

7.7.2. Schematic design

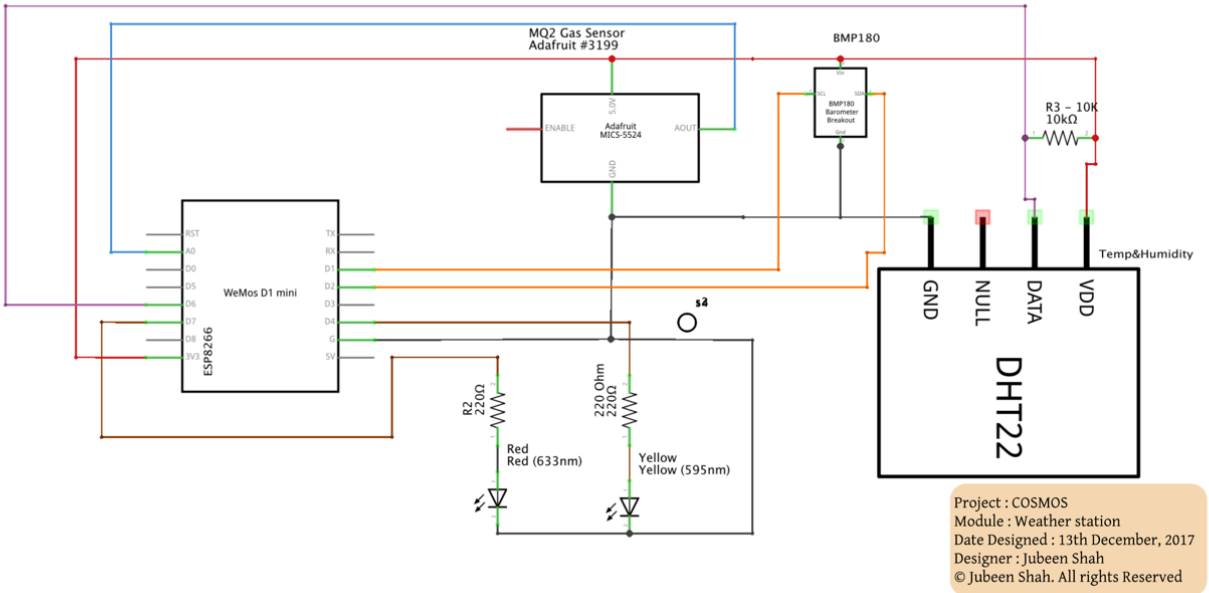

Figure 20 Schematic design of COSMOS: Weather Station

7.8. InfluxDB and Grafana

The idea to establish a graphical dashboard and a persistent storage database came from the roots of trying to integrate Artificial Intelligence into the project. Though the AI integration itself would be a part of much later improvisation into the project, this phase of setting up the database constructs a baseline environment for the same. This section wouldn't go into the details of setting up the database, however it is meant to give an overview on what has been accomplished.

7.8.1. InfluxDB

InfluxDB is an open platform that allows for easy integration into other open projects. It is a time series, metrics, and analytics database written in Go. This database was chosen for integration into the project since it has no external dependencies, which means that once influxDB is installed on the server or network, no additional management is required. Features like but not limited to HTTP API, Database managed retention policies and built in management interface were key in establishing the need for InfluxDB as the primary persistence mechanism in COSMOS.

Once InfluxDB is installed and configured, a database instance was created, with user privileges defined. It must be noted that for advanced application, InfluxDB can be installed directly onto a separate docker image to ensure application independency.

7.8.2. Grafana

Grafana is another open source software for time series analytics. With Grafana COSMOS has gained capabilities for visualization, alerting, notifications and annotations. Like InfluxDB, Grafana can either be installed on the broker, a separate docker image, or separate device but on the same network. Once the Grafana and InfluxDB are connected through modifications in the configuration files, the only step that remains is to connect the COSMOS MQTT broker with the two.

Finally add a persistence job so openHAB sends your item states to InfluxDB. Add all items or item groups you want to persist including fitting strategies with the creation of an influxdb.persist file :

```
Strategies {
   everyMinute : "0 * * * * ?"
   everyHour   :  "0 0 * * * ?"
   everyDay    :   "0 0 0 * * ?"
}

Items {
  //Items that needs persistence
}
```

7.8.3. COSMOS: Dashboards

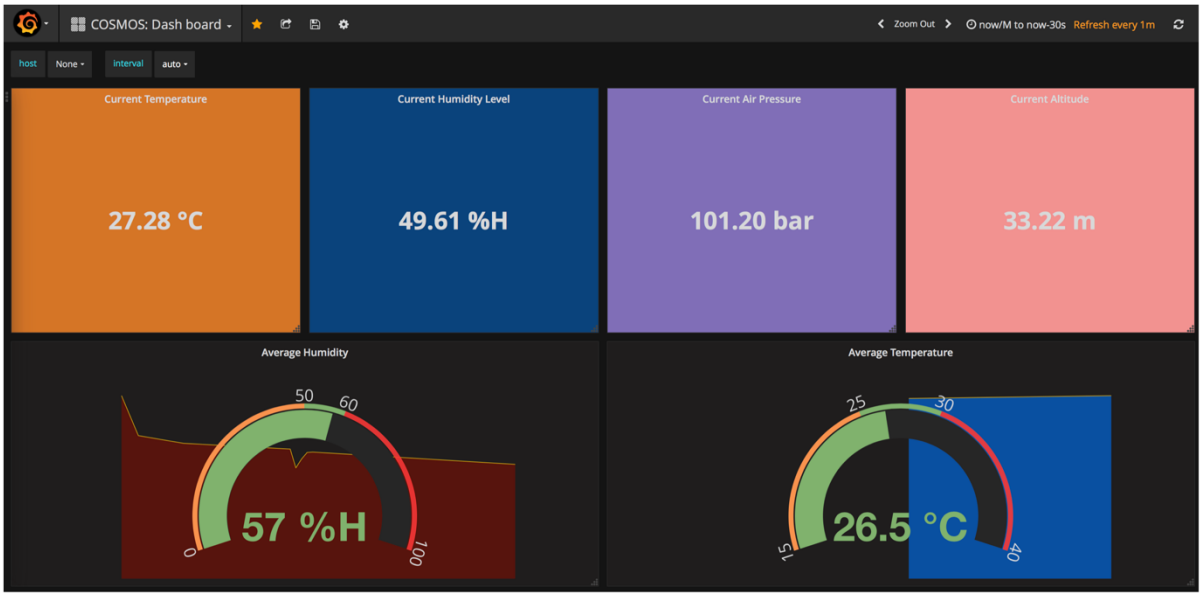

Figure 21 COSMOS: Dashboard 1

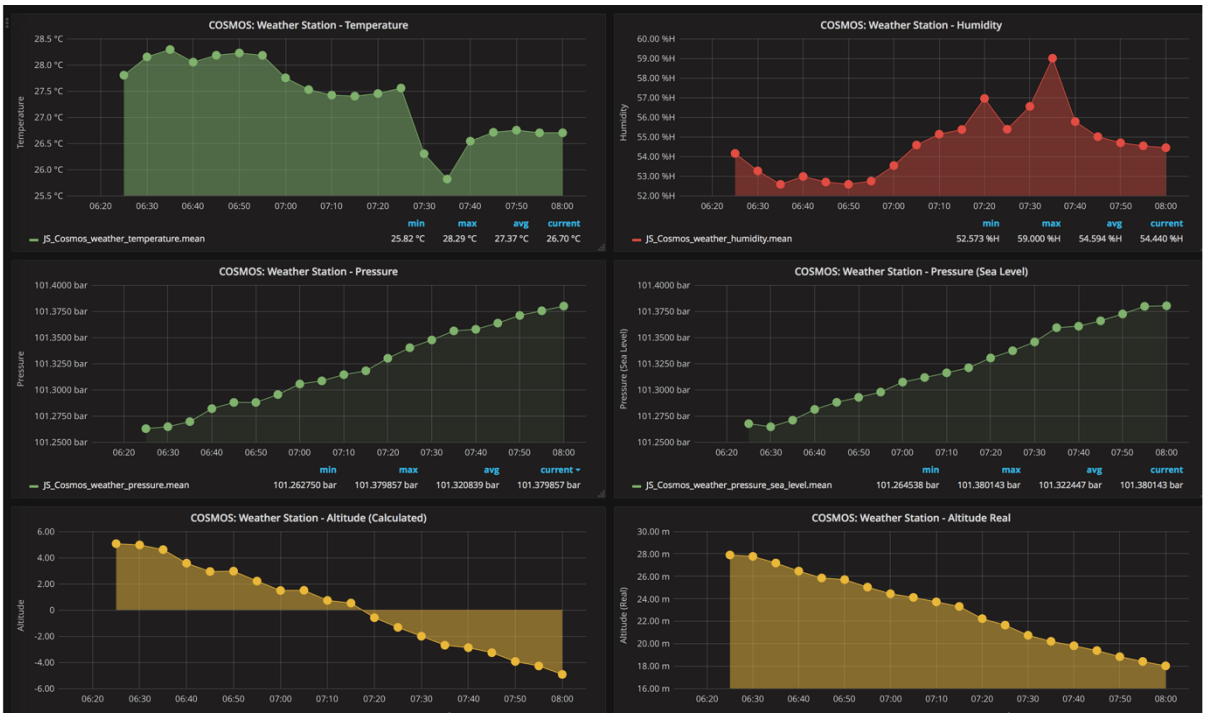

Figure 22 COSMOS: Dashboard2

7.9. Final Installation

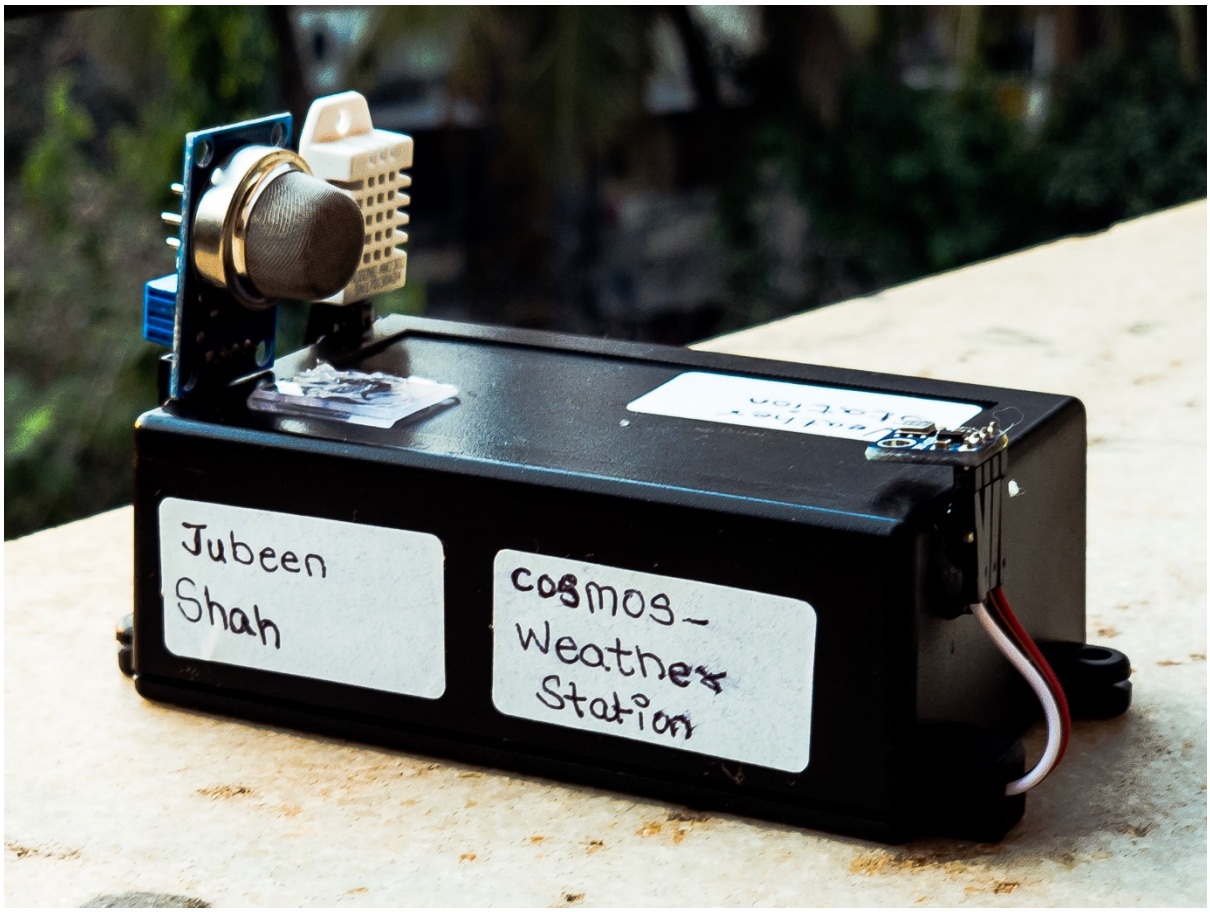

Figure 23 COSMOS: Weather Station

8. Module 4- COSMOS: Ambient Station

8.1. Introduction

COSMOS: Ambient Station (CAS) is a device that complements the COSMOS: Weather Station. CAS provides information that could be relevant to outdoor application where knowing the UV radiation levels, the light intensity levels, and Soil Moisture Levels are of prime importance — Agricultural domain for example. The sensors used in CAS are BH1750 Ambient Light Sensor, ML8511 UV sensor, and a Soil Moisture Sensor. An ESP8266 Wemos D1 MCU unit has been used for this module, which gives the module the capability to communicate the values from the sensors over a wireless connection to the MQTT broker, like in the previous modules. Having developed a dashboard for visualizing the data in the previous module, i.e the COSMOS: Weather Station module, CAS also integrates itself to the existing database, giving a central repository for all the data that is stored locally on the broker. This was done, to align CAS module with the project's goal of maintaining user security. This module was particularly difficult to develop, because both ML8511 and the Soil Moisture Sensor have analog outputs, which means the output from these sensors is active high output in terms of voltage, and each voltage value is mapped to a particular value that the sensor reads. The ESP8266 has only one Analog to Digital Converter pin, addressing this issue was of prime concern with the CAS module. Several solutions were tested, and are explained in the coming sections. This chapter following sections — Use Cases, Functional Block Diagram and flow chart of CAS, followed by the hardware components needed for the module. A detailed description of the bread-board diagram and the schematic followed by an in-depth overview of the code is provided towards the end of this chapter.

8.2. Use cases

COSMOS: Ambient Station is a utility device which relays relevant information to the end user — stated in the previous section. Since, CAS is integrated with the COSMOS environment which is already setup, it provides all the features that current industrial standard devices fail to do, or provide a very expensive solution. The advantage of integrating CAS into the COSMOS environment is the remote alert, and continuous monitoring of the ambient conditions in the appropriate environment.

BH1750 — This is a digital Ambient Light Sensor with typical use cases, as per the data sheet, in the Personal Digital Assistant domain. However, the use cases can be much wider than that and be expanded to even solar farms for instance. The typical solar farms have solar cells that are static — the main disadvantage being that since the sun does not always shine in one place, the efficiency of such systems is greatly reduced. With multiple BH1750 sensors, strategically positioned, solar tracking systems can be improvised at a much lower cost to increase the efficiency of the system by 30-40%.

ML8511 — UV radiation or UVR has extreme effects on the spectrum when considering human health, crops, terrestrial ecosystem, aquatic ecosystem and biogeochemical cycles. UV-B Radiation causes skin cancer and mutation in DNA of Humans. It also surpasses the immune system of the body. While, when considering plants, it could be beneficial for some species while growth stunting in some. Similarly, overexposure to UV-B impairs the productivity of phytoplankton in aquatic ecosystems. Using the ML8511 in such environments can help take event-based actions to either expose more UV to a certain plant type while restricting the exposure to some.

Soil Moisture — Soil Moisture sensor can simply, tell the moisture levels in the soil. This information can be used for better and more efficient irrigation systems. For instance, in a place where it rains heavily, irrigation requirement would be infrequent, whereas in location with scanty rainfall, irrigation requirement would be frequent. Depending on the soil moisture levels, the irrigation system can be automatically turned on when required.

CAS is a ubiquitous device which provides a culmination of all the aforementioned sensors; and, their advantages when permuted and computed, provides a standalone system, ready for deployment in farms, crop fields, or the backyard garden of a residential house.

8.3. Functional block diagram

Figure 24**,** shows the functional block diagram of the COSMOS: Ambient Station. It makes use of the ESP8266-12E Wemos D1 chip to communicate with the centralized controller. The sensors which are a part of the Ambient station are connected directly to the ESP8266-12E WiFi chip as shown in the block diagram. The additional component in the COSMOS: Ambient Station module is the IC 4051 multiplexer/demultiplexer which is needed since both ML8511 and the FC-28 sensor have analog outputs, where has the ESP8266 has only one ADC.

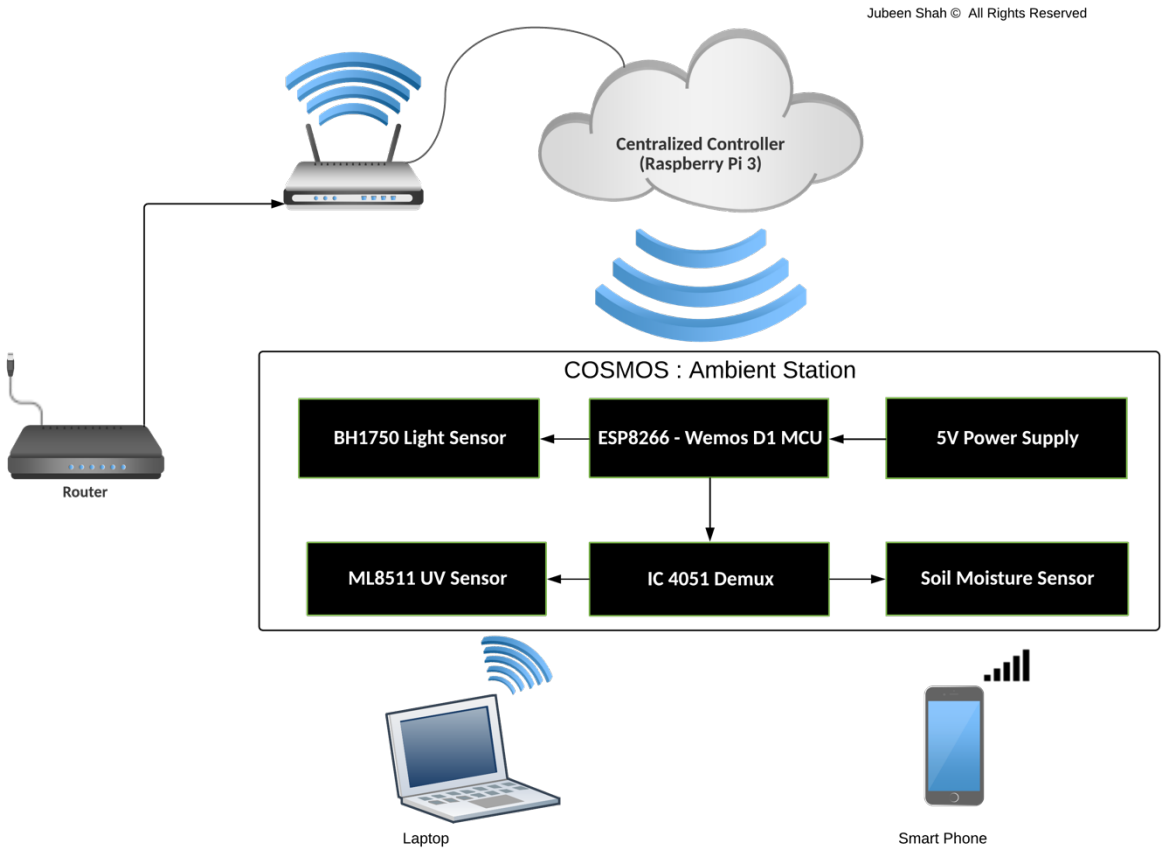

Figure 24 Functional block diagram of COSMOS: Ambient station

ESP8266 chip when receives the sensor values relays the information to the centralized controller. When the RPi3 receives these values, it checks for the rules set in the **.Rules** file and based on the predefined conditions takes the necessary actions. These actions could be, but are not limited to:

- Send Push-Notifications
- Take Event-based actions
- If-This-Then-That Integration to interact with other modules

Once the relevant information is relayed, this information would be available to the authenticated and authorized users via the different web interfaces that were described earlier in chapter 3 of this project report. The user can also continuously monitor these values. Since a database integration is already setup in the COSMOS: Weather Station Module, the COSMOS: Ambient Station values would simply be appended to the existing database by creating the required columns.

8.4. Flowchart

Figure 25, describes the working of the COSMOS: Ambient Station with a flowchart.

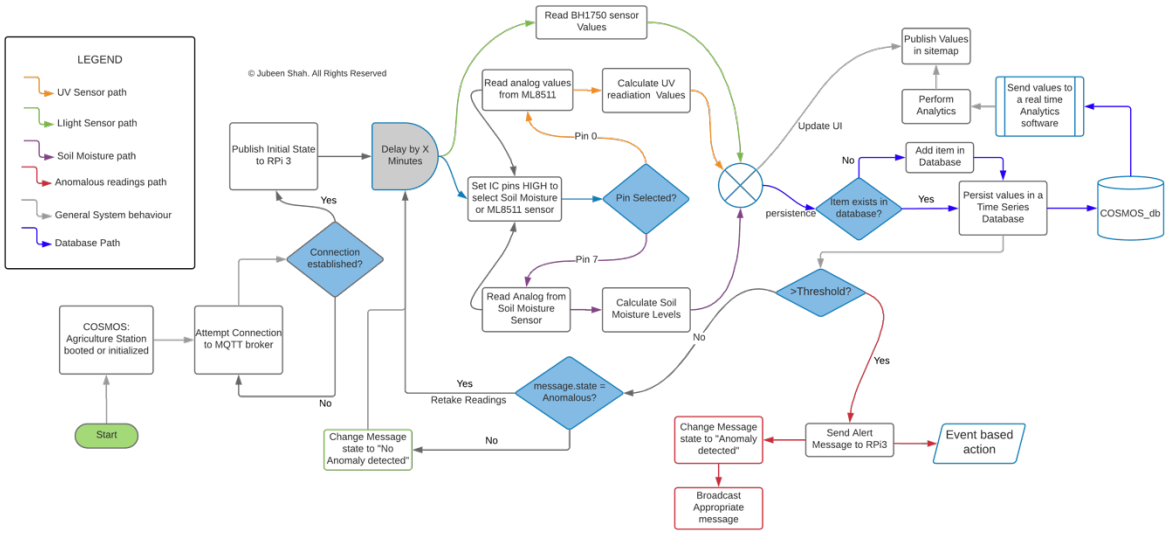

Figure 25 Flowchart of COSMOS: Ambient Station

The COSMOS: Ambient Station (CAS) initially boots up and repeatedly tries to connect with the MQTT broker, RPi3, till a successful connection is established. Once the link is confirmed, the RPi3 would publish the initial state of CAS. It would then monitor the values relayed by the different sensor at regular intervals. If the sensor values being read, are anomalous in ways that are predefined, the centralized controller would change the state of the system to "Anomaly Detected" and the corresponding alert messages would be broadcasted to notify the authorized individuals. The system will then monitor if the issue that was risen by the CAS is resolved. If not, the system would re-affirm the presence of the problem by prompting the corresponding sensors to resend the values to double-check the occurrence of the detected error. If the issue were resolved, the controller would change the state of the system to "No Anomaly Detected". The switching between the ML8511 and the FC-28 happens in the loop itself, while the sensor values are being read. A delay of 5 seconds allows for the optimal result, such that the values from either sensor don't interfere with each other.

8.5. Hardware requirements

The components required for the CWS are easily and readily available. This was necessary to ensure the scalability of the project. They have also been tried and tested and are found to be reliable. Table 12, lists down the hardware components needed with their corresponding links and price associated with each.

Sr.	Component	Quantity	Link	Price
1	ESP8266 - Wemos d1 mini	1	https://goo.gl/oJDvTY	₹ 467.50
3	BH1750	1	https://goo.gl/sNGMFA	₹ 276.00
4	ML8511	1	https://goo.gl/tpTDBi	₹ 599.00
5	FC-28	1	https://goo.gl/8ZDcSY	₹ 163.00
6	Custom PCB	1	https://www.pcbpower.com	₹ 313.00
7	IC 4051	1	https://goo.gl/KWbexZ	₹ 349.00
	Total	6	Total	₹ 2167.50

Table 12 List of Hardware components - COSMOS: Ambient station

Table 13, below, describes the part required for the COSMOS: Ambient Station, with its properties and PCB description.

Label	Part Type	Properties
IC4051	IC 4051	chip label IC 4051; true; package DIP (Dual Inline) [THT]; pin spacing 300mil; hole size 1.0mm,0.508mm; pins 16
J1	Generic female header - 2 pins	package THT; pin spacing 0.1in (2.54mm); row single; hole size 1.0mm,0.508mm; pins 2; form ♀ (female)
JP1-4	#4 Stand Off	package stand-off-tight; variant tight
LED1	Red (633nm) LED	leg yes; package 5 mm [THT]; color Red (633nm)
LED2	Yellow (595nm) LED	leg yes; package 5 mm [THT]; color Yellow (595nm)
R1	330Ω Resistor	bands 4; package THT; pin spacing 400 mil; resistance 330Ω; tolerance ±5%
R2	330Ω Resistor	bands 4; package THT; pin spacing 400 mil; resistance 330Ω; tolerance ±5%

Table 13 Part List and description for COSMOS: Ambient station

8.6. Exploring Datasheets

8.6.1. BH1750

BH1750's output is a calibrated digital signal. As stated earlier, it is a digital Ambient Light sensor integrated circuit. Like the BMP180 sensor from the weather station module, it communicates over an I^2C interface, one of the primary reasons, the COSMOS: Weather Station Module and COSMOS: Ambient Station Module had to be different. It has a very high resolution that gives values ranging from 1 lux to 65535 lux.

Part	Description	Ratings			Units
		Min.	Typ.	Max.	
Model	BH1750	-	-	-	
Power supply	Max Voltage	-	-	4.5	V
Operating Temperature	Operating temperatures of BH1750	-40	-	85	°C
VCC Voltage	Vcc	2.4	3.0	3.6	V
I^2C Reference Voltage	VDVI	1.65	-	VCC	V
Current supply	When Measuring	-	120	190	µA
Current supply	When in Stand-by mode	-	0.01	1.0	µA

Table 14 Technical specification and operating conditions of BH1750

Typical measurement conversion looks like this in the BH1750
High Byte = "1000_0011"
Low Byte = "1001_0000"
$(2^{15}+2^9+2^8+2^7+2^4)/1.2 \fallingdotseq 28067[lx]$ //As defined in the data sheet

8.6.2. ML8511

The ML8511 UV sensor according to the data sheet is suitable for measuring UV intensity indoors and outdoors. It consists of an internal amplifier that converts the photo-current to voltage depending the UV intensity. This configuration allows the ML8511 to be easily configured with an ADC which makes it feasible for integration into the COSMOS: Ambient Station.

Part	**Description**	**Ratings**			**Units**
		Min.	**Typ.**	**Max.**	
Model	ML8511	-	-	-	V
Operating Temperature	Minimum to Maximum	-30	25	85	°C
VCC Voltage	V_{cc}	2.7	3.3	3.6	V
Output Voltage (Shading)	V_{REF}	0.95	1.0	1.05	V
Output Voltage ($10mW/cm^2$)	V_o	2.08	2.2	2.32	V
Current supply	When Measuring	-	300	500	µA
Current supply	When in Stand-by mode	-	0.1	1.0	µA

Table 15 Electrico-optical characteristics of ML8511

8.6.3. FC-28

The FC-28 soil moisture sensor is designed to detect the moisture of soil or judge if there is water around the sensor. Table 16, gives the electrical characteristics of FC-28.

Part	**Description**	**Ratings**			**Units**
		Min.	**Typ.**	**Max.**	
Model	FC-28	-	-	-	V
Operating Temperature	Minimum to Maximum	-30	25	85	°C
VCC Voltage	V_{cc}	3.3	-	5	V
Output	Sensor in dry soil	0	-	300	-
	Sensor in humid soil	300	-	700	-
	Sensor in water	700	-	950	-
Current supply	When Measuring	0	-	35	mA

Table 16 Electrical characteristics of FC-28

8.7. Design

8.7.1. Breadboard design

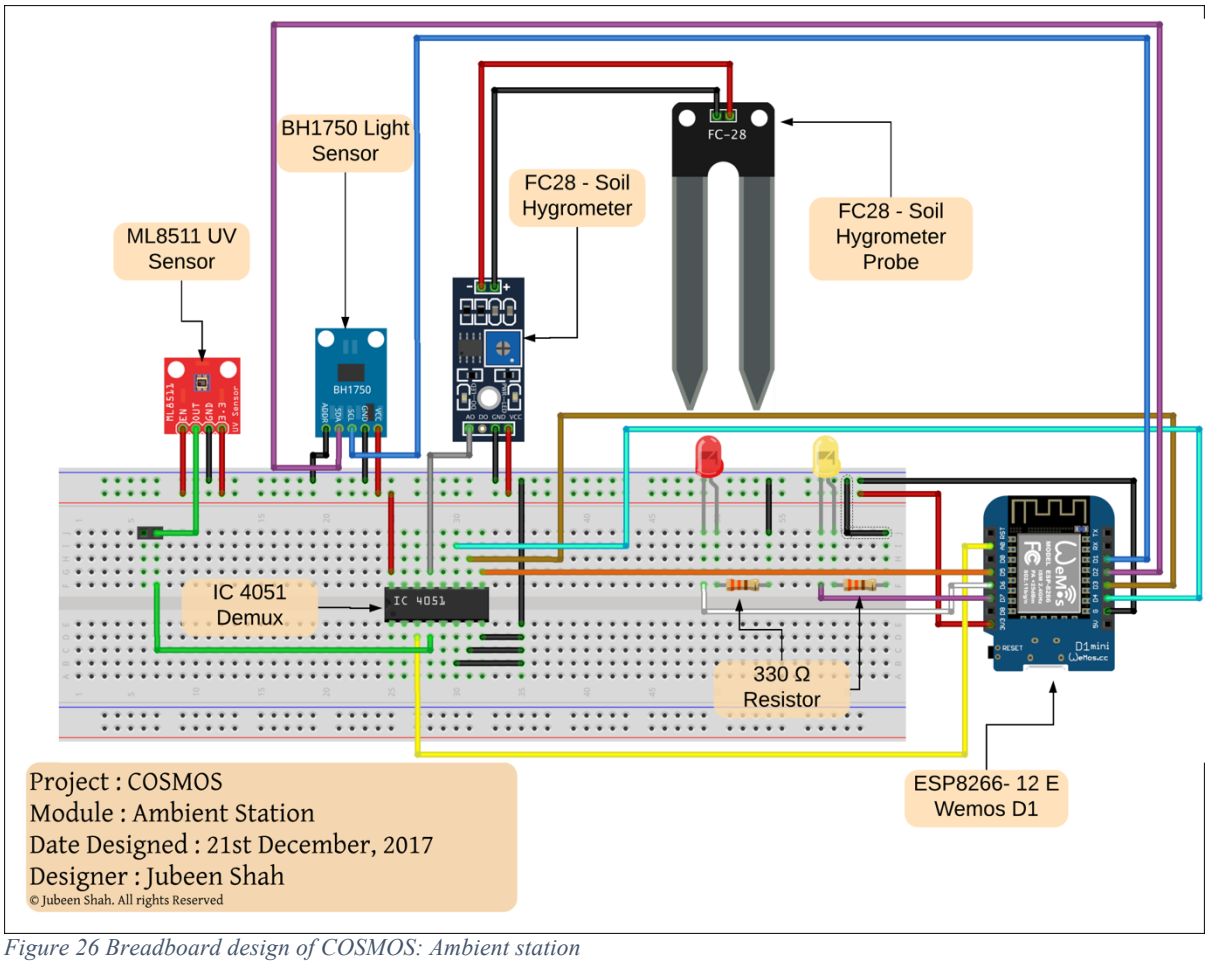

Figure 26 Breadboard design of COSMOS: Ambient station

8.7.2. Schematic design

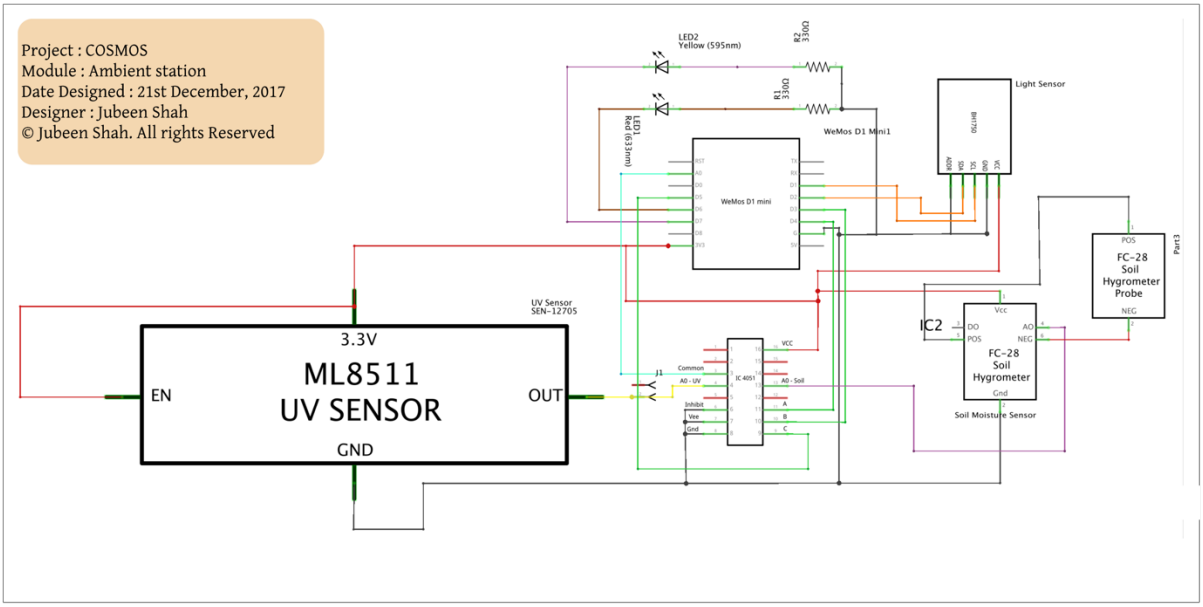

Figure 27 Schematic design of COSMOS: Ambient station

8.8. De-multiplexing alternatives attempted

One limitation of the ESP8266 MCU is that, there is only one onboard ADC available for use. As it has been stated several times in the chapter, the ML8511 and FC-28 Sensors have analog outputs, using them with one ESP8266 was not possible. This section briefly covers the three alternatives that were tried to overcome this hindrance. The alternatives are as follows:

- Programming Modifications
- Using a diode
- IC 4051 multiplexer/demultiplexer

8.8.1. Programming modifications

In the initial stages of the project, when the COSMOS: Ambient Station module was in the feasibility study phase, a simple solution was thought to be able to read analog values from both the sensors. The ESP8266's digital pins have an output voltage of 3.3 volts, when set HIGH. This voltage output was within the specified limits of the sensors as per the data sheet. As a result, due to lack of familiarity with electronics component, the digital pins of the ESP8266 were directly to supply voltage to the sensors. Using simple programming constructs, the following logic was implemented.

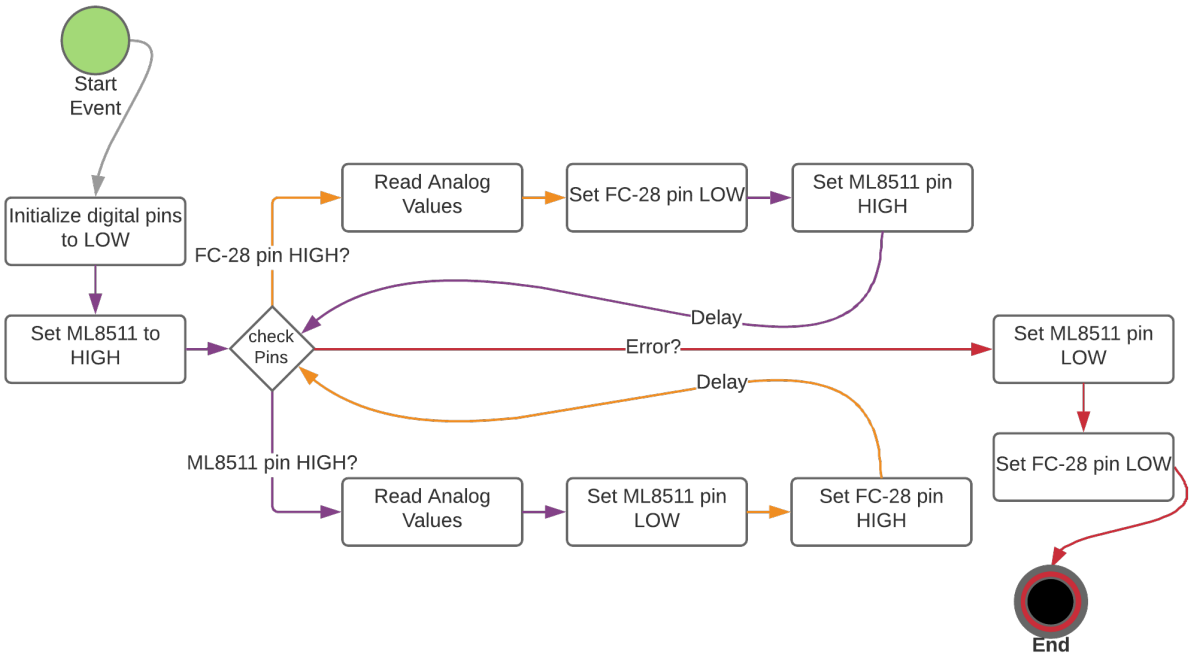

Figure 28 Programming modifications flow chart

However, this did not solve the problem, since it was observed that even when the sensor's input Vcc were set to LOW, a current kept flowing through the closed circuit, which interfered with the readings.

8.8.2. Using a diode

After scraping off the initial idea, and after doing some more study regarding the electronic components, the features for a diode made it appealing to be used as tool to allow the flow of current in one direction only. This seemed promising, since the flaw in the previous logic was the presence of interfering current. After using a low reverse leakage current enabled diode — IN4007, and using the logic as before, another issue had risen. With the use of diode, came a voltage drop of approximately 0.7 Volts in the test circuit. This reduced supply voltage from the digital pins to 2.6 Volts, which was below the minimum operating requirement of the sensors. This alternative was discarded as well.

8.8.3. IC 4051 multiplexer/demultiplexer

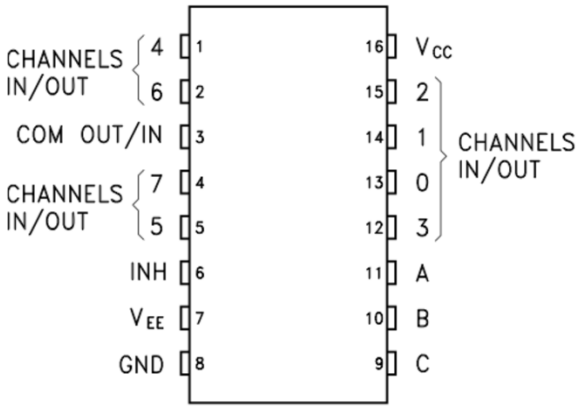

Figure 29 IC 4051 pin diagram

After studying more about multiplexing and demultiplexing of signals, the IC 4051 was chosen to overcome the restriction imposed by ESP8266 of a single ADC. Pin 3 of IC 4051, as seen in the pin diagram is the COMMON pin that is connected to CHANNELS IN/ OUT pins 1, 2, 4, 5, 12, 13, 14 and 15. To choose one of the pins to connect with the COMMON pin, the pins 9, 10, 11 which correspond to C, B, and A pins on the IC to high or low. This expands the capabilities to connect up to eight additional analog sensors. The value of A, B, and C are set according to the following truth table.

A	B	C	Value	Pin Selected
0	0	0	0	13
0	0	1	1	14
0	1	0	2	15
0	1	1	3	12
1	0	0	4	1
1	0	1	5	5
1	1	0	6	2
1	1	1	7	4

Table 17 IC 4051 Truth Table

The advantage of using IC 4051 was, that no reverse current was observed in the circuit, and moreover since there is a single V_{cc} pin for the IC, no significant voltage drop was experienced. The only precaution that needed to be taken was to set appropriate delay between selecting the pins to connect to the COMMON pin and reading the analog values.

8.9. Final Installation

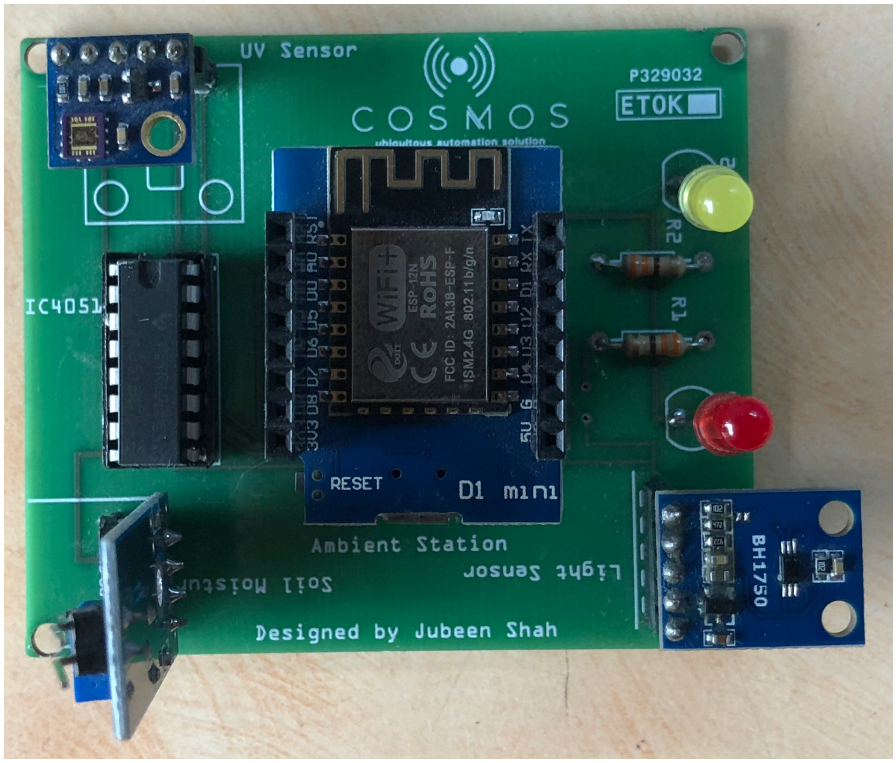

Figure 30 COSMOS: Ambient station

9. Module 5 – COSMOS: Smart LEDs

9.1. Introduction

COSMOS: Smart LEDs is multipurpose device which manages the state of standard RGB LEDs. The device makes use of the standard ESP8266 -12E WiFi chip to receive commands over the air and relay the respective information to the LED lights that is connected directly to the device. The benefit of having WiFi connected LEDs is, that they can be triggered by an event that happens on other devices in the environment once the rules are set up. For instance, consider the COSMOS: Fire Sensor Module. Once the fire has been detected and the notification sent to the user of the event, a simultaneous message can be relayed by the MQTT broker to turn on the COSMOS: Smart LEDs and set it to RED indicating the path to the nearest Fire Escape. The COSMOS: Smart LEDs also have other use cases, such as that involving recreational use to set the ambience of an environment. Users can set the colour of the lights to one of the 1,67,77,216 colour variants possible with the COSMOS: Smart LEDs. These environments are not limited to home, but use cases can be easily extended to Restaurants, Pubs, Concerts, Universities and college auditoriums, Luxury Cars, etc.

9.2. Functional block diagram

Figure 31**,** shows the functional block diagram of the COSMOS: Smart LEDs. It makes use of the ESP8266-12E WiFi chip to communicate with the centralized controller. The obstacle in this module that had to be overcome, was to deal with the 12 volt input requirement of the LEDs and the 3.3V requirement of the ESP8266. In contrast to the LM1117; LM2596 was used to satisfy both the requirements. The TIP-122 transistor is used for amplification of current from the ESP8266 to allow for translation of the digitally encoded values of the red, blue and green signals into an amplified version that can be recognised by the RGB LED strip. It simultaneously also works as a switch, when only one of the colours in the strip are to be made visible.

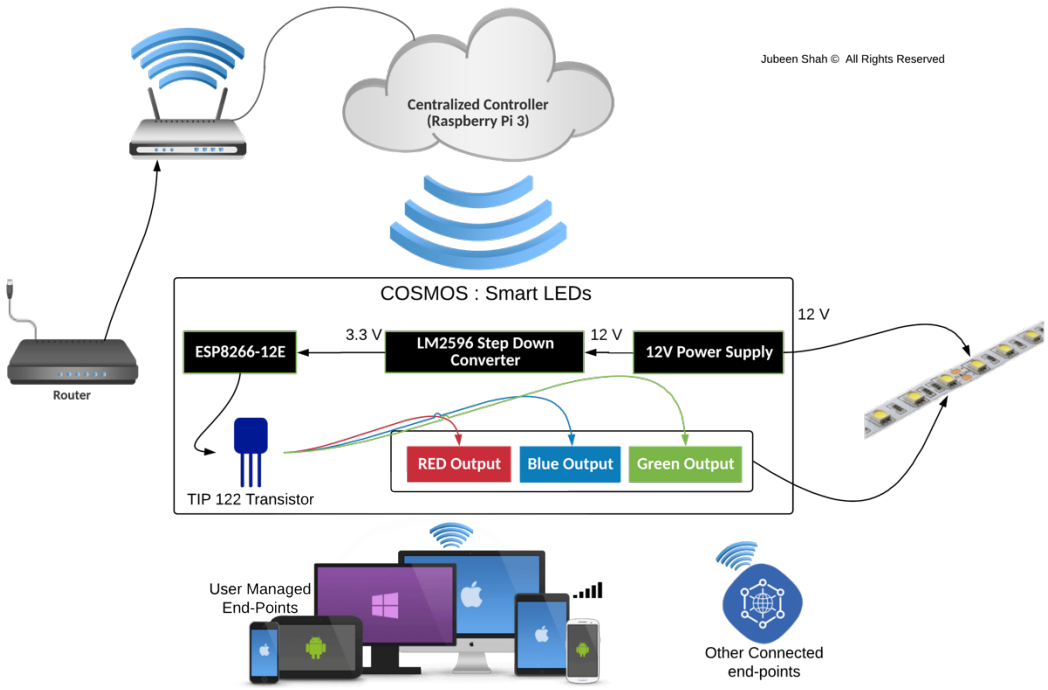

Figure 31 Functional block diagram of COSMOS: Smart LEDs

9.3. Flowchart

ESP8266 chip, when it receives the RGB values from the centralized controller, sends the corresponding signal onto the Red, Green, and Blue channel of the light strip. This helps to manually control the light strip. When the broker receives a message from other connected end-points, for instance the COSMOS: Fire Sensor, it checks for the rules set in the **.Rules** file. Based on the predefined conditions takes the necessary actions. These actions could be, but are not limited to:

– Send Push-Notifications
– Take Event-based actions
– If-This-Then-That Integration to interact with other modules

This allows for an ecosystem of devices that can communicate with each other, through the broker.

Figure 32, describes the working of the COSMOS: Smart LEDs with a flowchart.

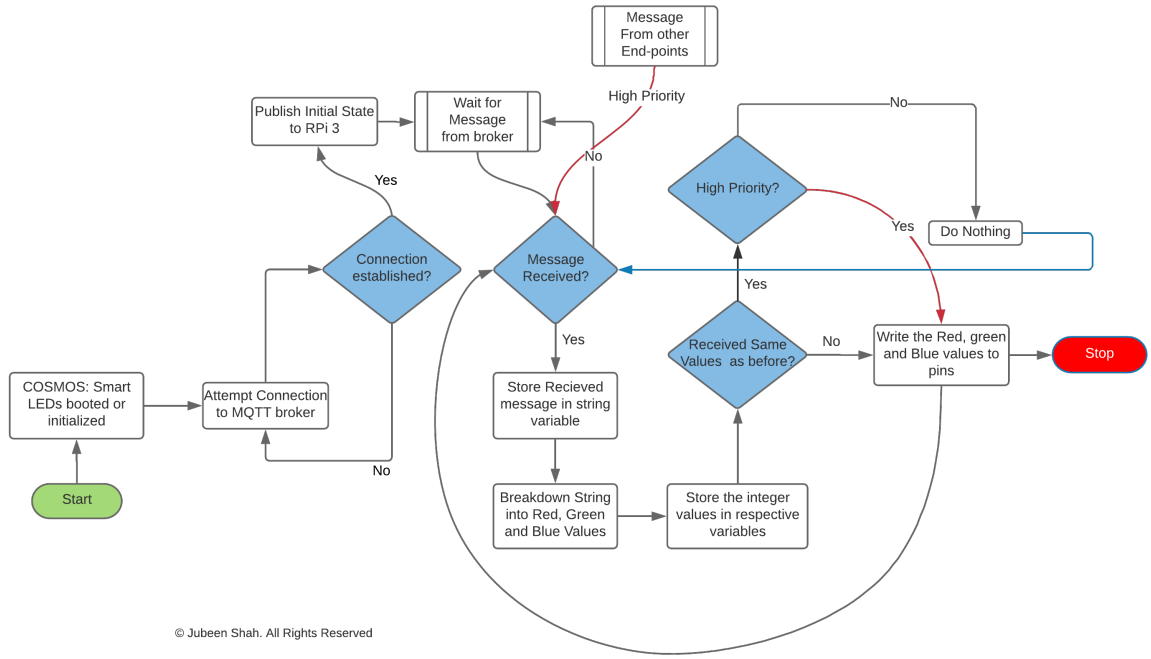

Figure 32 Flowchart of COSMOS: Smart LEDs

The COSMOS: Smart LED (CSL) initially boots up and repeatedly tries to connect with the MQTT broker, RPi3, till a successful connection is established. Once the link is confirmed, the RPi3 would publish the initial state of CSL. The CSL, then waits for a message from the broker, which would indicate the action that it has to perform. For instance, consider the following message being sent by the broker to the CSL module —`100;100;100` as mesgRecvd, which translates to a white light light. The mesgRecvd is stored as a string, and broken down by the separator ‘;’ and the corresponding red, green and blue values are stored as an integer. This integer value is checked, if it is the same as the previous message sent, to prevent redundant message transfer. The digital values of the corresponding colours are then mapped to the values of the signal that is to be sent to the light strip. Once the signal is processed, it checks if another message has been received. If the next mesgRecvd is `0;0;0`, it would simply turn off the strip.

It is useful to know that, since the database is already in place, and all state changes are persisted, the states of CSL are persisted as well. However, they are not represented on the dashboard due to insufficient transference of meaning.

9.4. Hardware requirements

The components required for the CSL are easily and readily available. This was necessary to ensure the scalability of the project. They have also been tried and tested and are found to be reliable.Table 18, lists down the hardware components needed with their corresponding links and price associated with each.

Sr.	Component	Quantity	Link	Price
1	ESP8266 12-E	1	https://goo.gl/yUFSpz	₹ 249.00
3	LM2596 module	1	https://goo.gl/qJQZjw	₹ 165.00
4	TIP 122 Transistor	3	https://goo.gl/P4ZTKT	₹ 65.40
5	470 Ohm Resistor	3	https://goo.gl/jKmaQv	₹ 3.00
6	Custom PCB	1	https://www.pcbpower.com	₹ 374.00
7	LED Strip	1	https://goo.gl/XvaGU1	₹ 1299.00
	Total	**10**	**Total**	**₹ 2155.40**

Table 18 List of Hardware components for COSMOS: Smart LEDs

Label	Part Type	Properties
J1	Power Jack	variant pth_lock; type 5.5mm barrel; package power_jack_pth_lock
J2	Generic female header - 2 pins	package THT; pin spacing 0.1in (2.54mm); row single; hole size 1.0mm,0.508mm; pins 2; form ♀ (female)
JP1-4	Jumper	package stand-off-tight; variant tight
R1	470Ω Resistor	resistance 470Ω; pin spacing 400 mil; bands 4; tolerance ±5%; package THT

Table 19 Part list and description of COSMOS: Smart LEDs

9.5. Exploring datasheets

The schematic shown in figure xx gives the schematic of a generic LED strip which needs 12V of power. As per the Datasheet these are analog type RGB LED Strips. Each of the marked sections have a tricolour type LED consisting of a red, green, and a blue LED. Each LED draws approximately 15-20 milliAmperes of current from a 12V power supply, as long as one of the LED is only working. When all three are working together, it would consume upto 60 milliAmperes of current per segment. The TIP 122 was chosen for its resilience to high voltage upto 100 Volts, and unlike it counter parts like TIP 120, has a higher capacity for current handling. Being able to handle upwards 5 A of current, more than 250 LEDs can at any time be connected to the power supply, and the transistors would be able to handle the load.

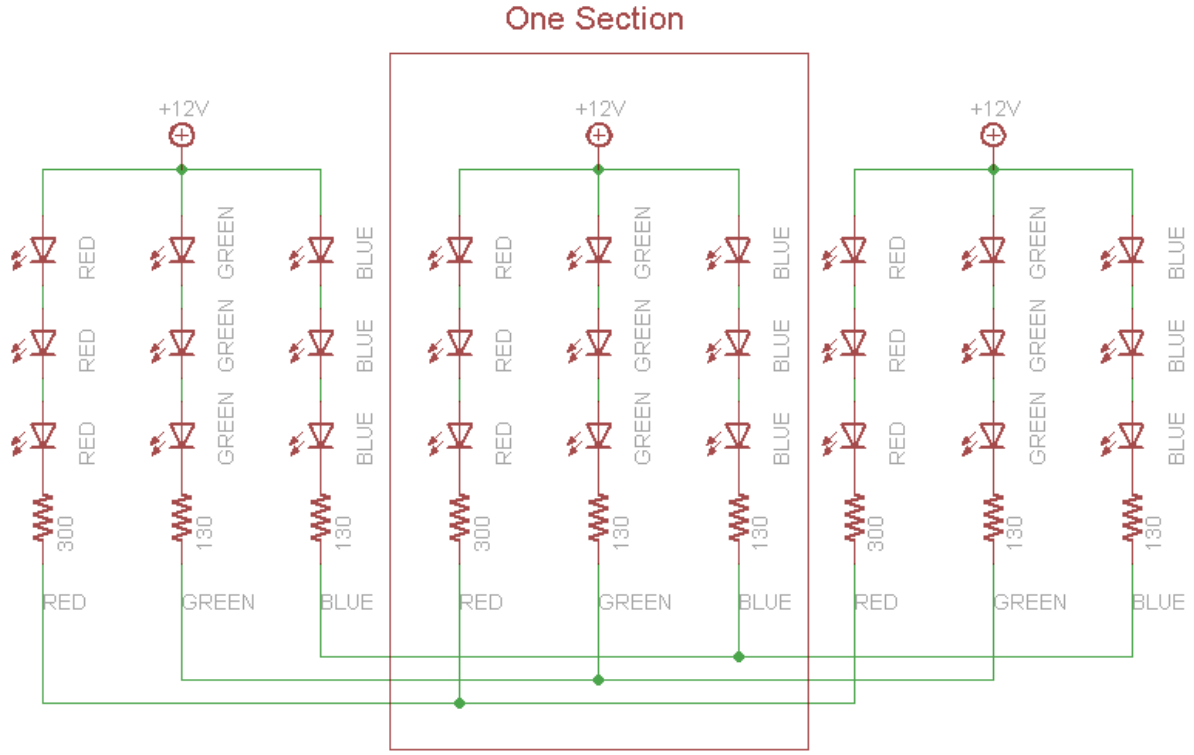

Figure 33 LED Strip Schematic

Having the schematic ready, for the COSMOS: Smart LEDs, the last thing to perform before concluding the module is the programming of the module. Unlike most modules completed so far, this module relies solely on being able to handle the received commands from the broker. Breaking down the client-server paradigm with respect to this module, the broker would first act as a server, when it is waiting for the arrival for messages regarding colour values from the client. The broker than acts as a client, transferring the received messages to the COSMOS: Smart LEDs, which are acting as the client. The CSL, would then perform the necessary actions on the connected strip.

9.6. Design

9.6.1. Breadboard design

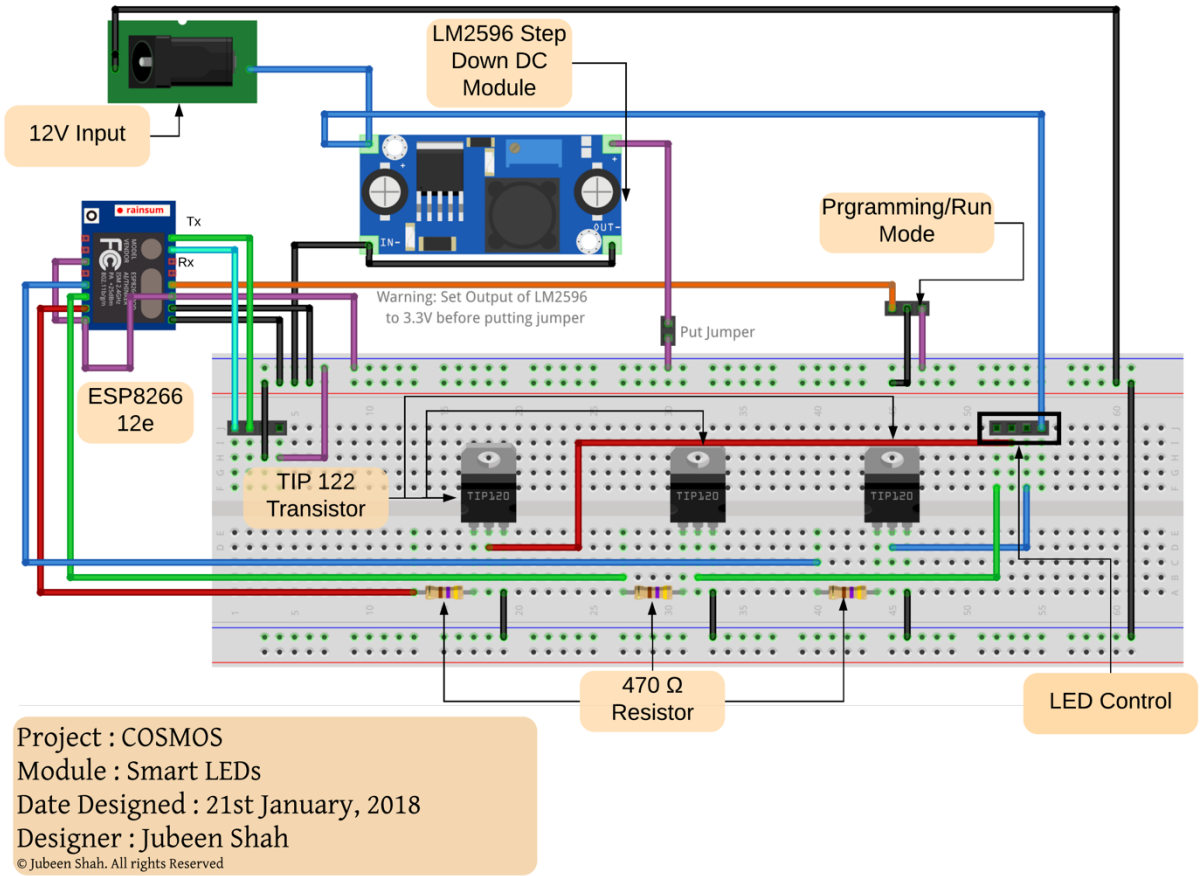

Figure 34 Breadboard design of COSMOS: Smart LEDs

9.6.2. Schematic Design

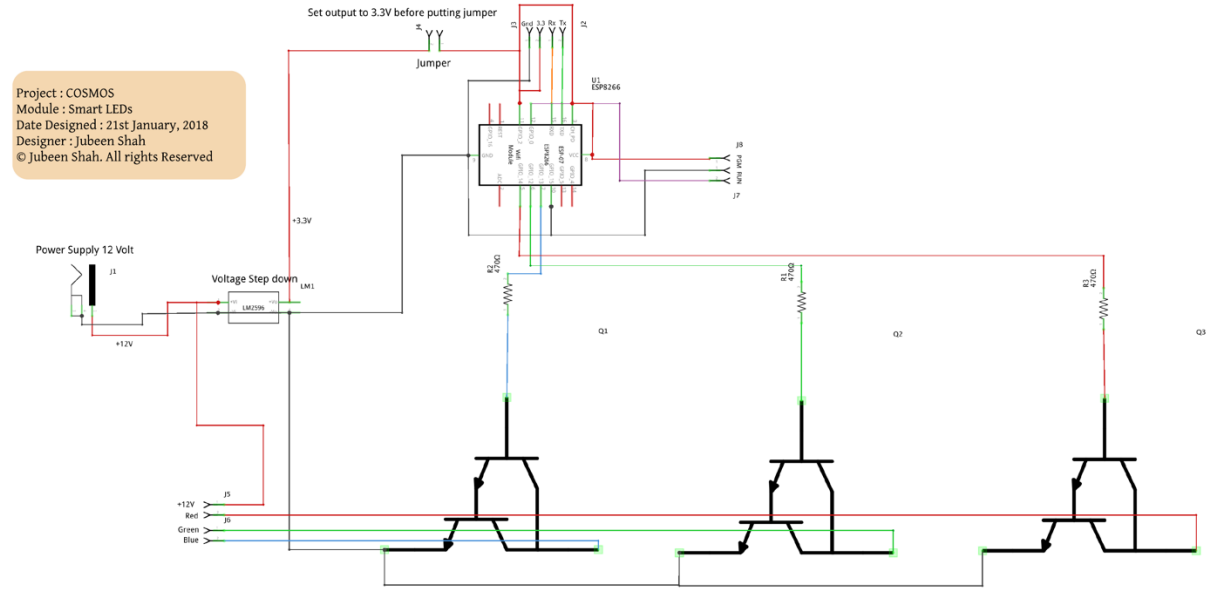

Figure 35 Schematic Design of COSMOS: Smart LEDs

9.7. Configuration

9.7.1. Items File

```
// COSMOS: Smart LEDs
Color  fWIFI_RGB_1 "RGB LED Light"    <slider>
String WIFI_RGB_1_RGB (WIFI_RGB_1)
{ mqtt=">[broker:COSMOS-SmartHouse/lights/COMSMOS-
LEDStripControl1:command:*:default]" }
String SetRed (WIFI_R_1)
{ mqtt=">[broker:COSMOS-SmartHouse/lights/COMSMOS-
LEDStripControl1:command:*:default]" }
String SetBlue  (WIFI_B_1)
{ mqtt=">[broker:COSMOS-SmartHouse/lights/COMSMOS-
LEDStripControl1:command:*:default]" }
String SetGreen (WIFI_G_1)
{ mqtt=">[broker:COSMOS-SmartHouse/lights/COMSMOS-
LEDStripControl1:command:*:default]" }
```

9.7.2. Sitemap File

```
Frame label="COSMOS: RGB Lights" {
    Colorpicker item=fWIFI_RGB_1 icon="colorwheel"
    Switch item=SetRed icon="rgb"
    Switch item=SetGreen icon="rgb"
    Switch item=SetBlue icon="rgb"
}
```

9.7.3. Rules File

```
rule "Set RGB 1 value"
  when
  Item fWIFI_RGB_1 changed
  then
  hsbValue = fWIFI_RGB_1.state as HSBType
  redValue = hsbValue.red.intValue
  greenValue = hsbValue.green.intValue
  blueValue = hsbValue.blue.intValue
  RGBvalues= redValue.toString + ";" + greenValue.toString + ";" + blueValue.toString
  sendCommand( WIFI_RGB_1_RGB, RGBvalues )
  logInfo( "fWIFI_RGB_1", RGBvalues )
 end

rule "Set BLUE 1 value"
  when
  Item SetBlue changed
  then
  hsbValue = fWIFI_RGB_1.state as HSBType
  redValue = hsbValue.red.intValue
  greenValue = hsbValue.green.intValue
  blueValue = hsbValue.blue.intValue
  RGBvalues= "0;0;100"
  sendCommand( SetBlue, RGBvalues )
  logInfo( "SetBlue", RGBvalues )
 end


rule "Set RED 1 value"
```

```
  when
  Item SetRed changed
  then
  hsbValue = fWIFI_RGB_1.state as HSBType
  redValue = hsbValue.red.intValue
  greenValue = hsbValue.green.intValue
  blueValue = hsbValue.blue.intValue
  RGBvalues= "100;0;0"
  sendCommand( SetRed, RGBvalues )
  logInfo( "SetRed", RGBvalues )
end

rule "Set GREEN 1 value"
  when
  Item SetGreen changed
  then
  hsbValue = fWIFI_RGB_1.state as HSBType
  redValue = hsbValue.red.intValue
  greenValue = hsbValue.green.intValue
  blueValue = hsbValue.blue.intValue
  RGBvalues= "0;100;0"
  sendCommand( SetGreen, RGBvalues )
  logInfo( "SetGreen", RGBvalues )
end
```

9.8. Final Installation

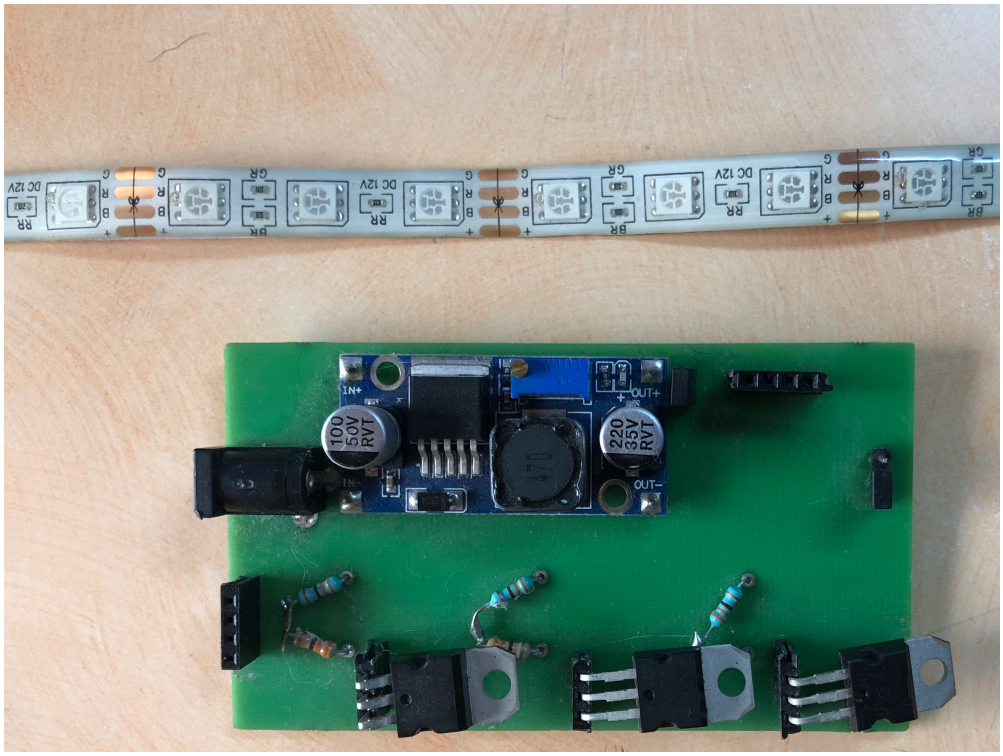

Figure 36 COSMOS: Smart LEDs

10. Module 6 – COSMOS: Connected Switches

10.1. Introduction

COSMOS: Connected switches is a module that ties together several modules that have been previously mentioned. Moving away from the norm of just gathering data from the environment like COSMOS: Smart LEDs, COSMOS: Connected Switches (CCS) provide a way to interact with the environment. The central idea behind CCS is to allow for a way to bring both manual and automatic state management of devices. The user can manage the state of the connected devices from anywhere in the world, as long as the user is connected to the internet. It makes use of a custom PCB, with an ESP8266 WiFi module to make sure that the design is replicable and scalable for mass production. CCS, begins to close the loop which the project started by incorporating functionalities to allow, automatic state management of switches, when it receives MQTT messages from other devices. For example, if the COSMOS: Weather station reports unusually high temperature in the installed environment, the CCS module can be used to switch on a connected fan or air conditioner.

10.2. Functional block diagram

Figure 37 shows the functional block diagram of CCS. The device contains four separate major components — ESP8266 12E wifi module, the 5V power supply module, PC 817 Optocoupler, and a bipolar transistor. The optocoupler is used for optical isolation of the circuit to prevent probable damage from the high voltage side of the relay. The NPN transistor is used for switching the voltage from the ESP8266 to the relay. To the end of the relays, would be connected the high voltage devices such as fans, lights, air conditioners etc. The switches' states can be manually managed by User Managed End-Points, or can be triggered by other connected end-points

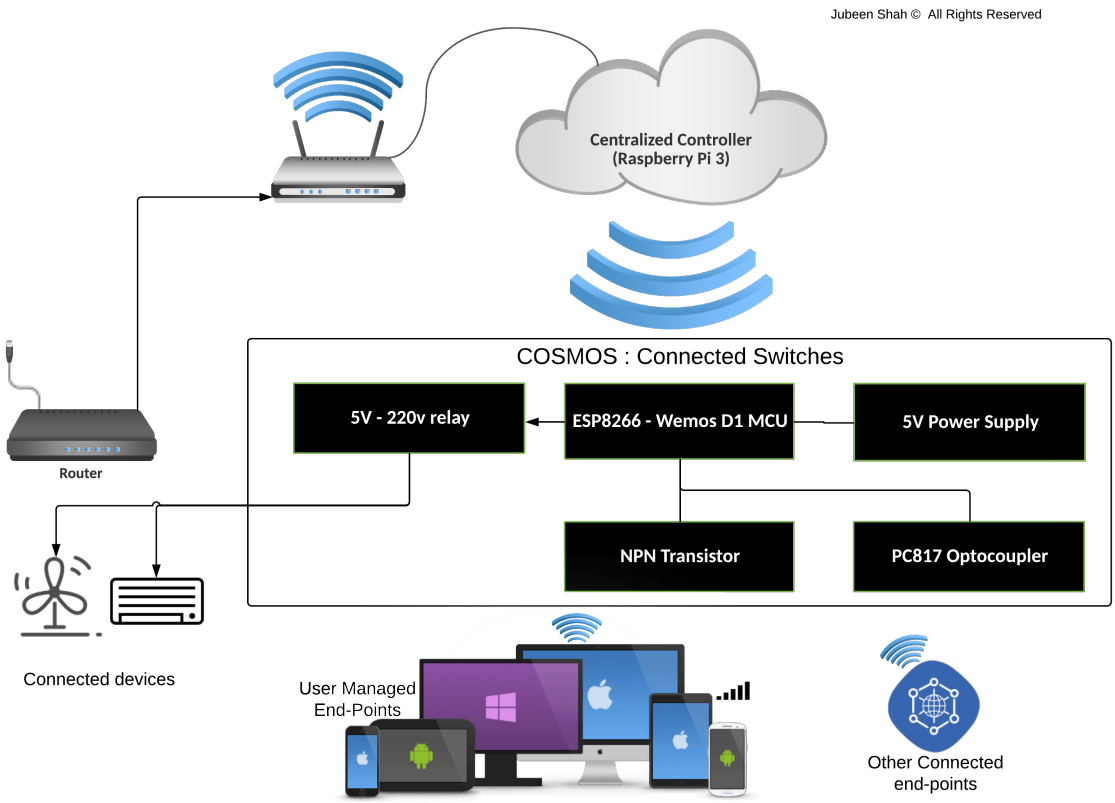

Figure 37 Functional block diagram of Connected switches

10.3. Flowchart

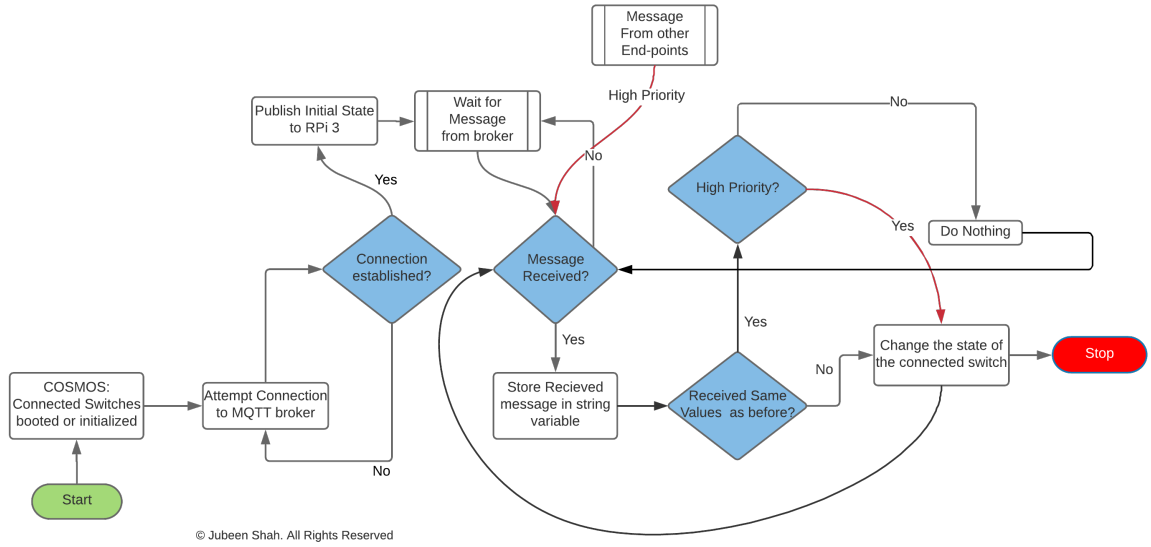

Figure 38 Flowchart of COSMOS: Connected Switches

The COSMOS: Connected Switches (CCS) initially boots up and repeatedly tries to connect with the MQTT broker, RPi3, till a successful connection is established. Once the link is confirmed, the RPi3 would publish the initial state of CCS. The CCS, then waits for a message from the broker, which would indicate the action that it has to perform. For instance, consider the following message being sent by the broker to the CCS module —`C1ON` as mesgRecvd, which translates to a turning on switch 1. The mesgRecvd is stored as a string, and compared using simple if-else conditions stored on the ESP8266 module. This mesgSent value is checked, if it is the same as the previous message sent, to prevent redundant message transfer. Once any one of the conditional statements return a true value, the ESP8266 would send an active high signal to the corresponding pin which is predefined in the schematic. This in turn would result in the relay to be activated, allowing the live wire to be in contact with the commonly closed pin, to complete the circuit.

10.4. Hardware requirements

Amount	Part Type	Properties
4	100Ω Resistor	tolerance ±5%; resistance 100Ω; package THT; bands 4; pin spacing 400 mil
4	330Ω Resistor	tolerance ±5%; resistance 330Ω; package THT; bands 4; pin spacing 400 mil
4	Rectifier Diode	type Rectifier; package 300 mil [THT]; part # 1N4001
4	Camdenboss CTB0158-3	variant 90° 3 connector; hole size 2.7mm; pins 3; package THT; pin spacing 0.2in (5.08mm); part # CTB0158-3
1	Generic female header - 2 pins	hole size 1.0mm,0.508mm; pins 2; form ♀ (female); row single; package THT; pin spacing 0.1in (2.54mm)
4	Relay T73	voltage 5V
4	NPN-Transistor	type NPN (ECB); package TO92 [THT]
4	Sharp PC817	part PC817; package THT

Table 20 List of parts for COSMOS: Connected switches and description

10.5. Design

10.5.1. Breadboard design

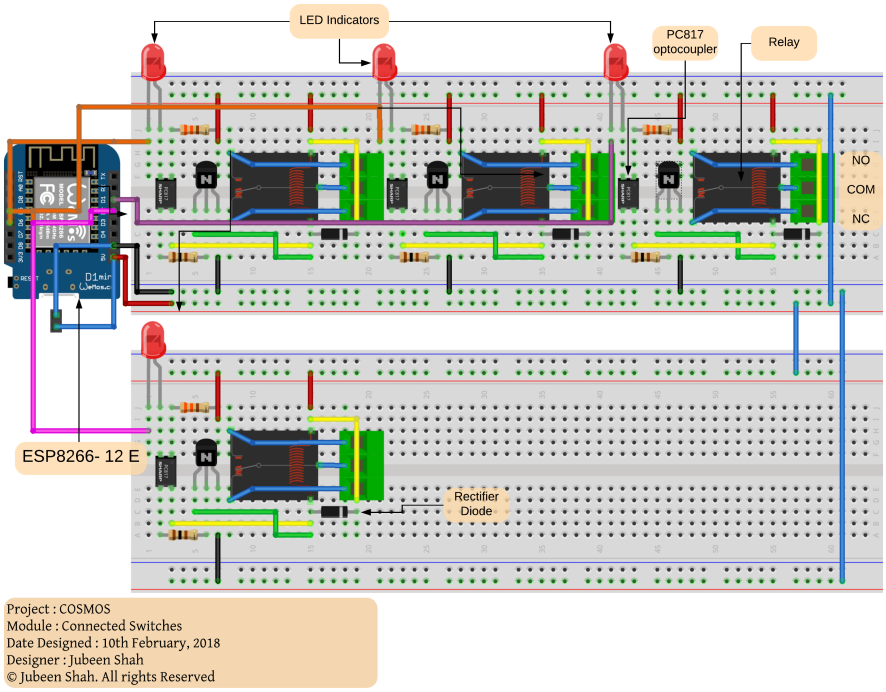

Figure 39 Breadboard design of COSMOS: Connected switches

10.5.2. PCB Design

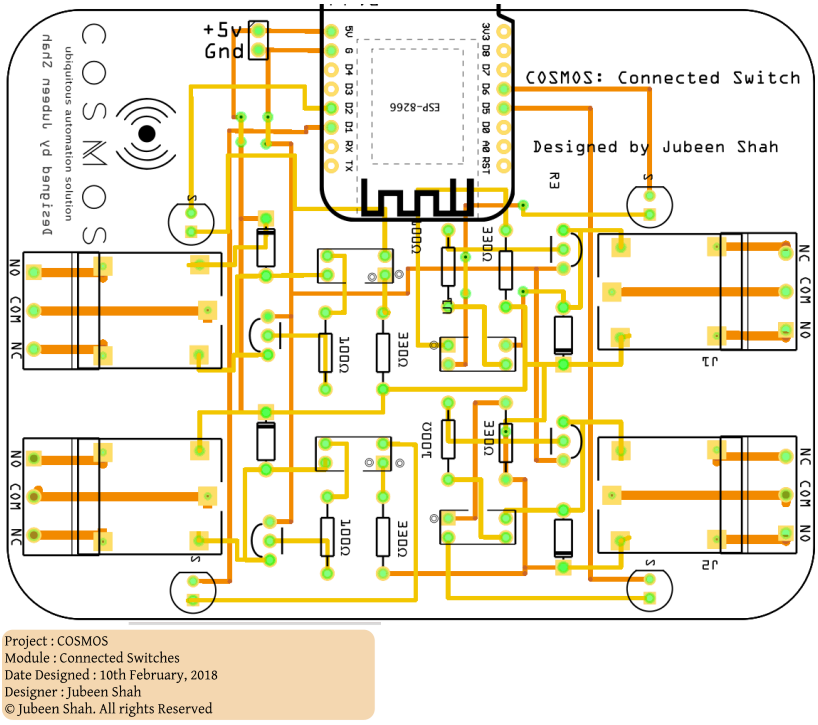

Figure 40 PCB design of COSMOS: Connected switches

10.6. Adding third party integration

10.6.1. Amazon's Alexa

In the initial scope of the project, it was indicated that voice commands would be integrated into COSMOS. Using standard API calls as defined by Amazon for Amazon echo, simple commands like, "Turn on Switch 1", "Turn off Switch 2", can be used to trigger events without using either the phone or a web user interface. Please note that the name of the device, is easily customizable, and can be sent to something generic such as "Lights", or something more specific like "Desk Lights".

Figure 41 Compatible Amazon Echo's family

Img src - https://alexacommander.com/device-guide/

10.6.2. Apple's Siri

Since, the mobile operating system platform chosen for development was iOS, integration of Apple's voice assistant was taken into consideration. Making use of the HomeKit library, integration with the native 'Home' app on iPhone was carried out to allow for a more flexible approach for state management of the COSMOS: Connected switches. This allows for easy voice based commands to be executed from any apple device – iPhone, iPad, HomePod, and Apple Watch. Simply saying "Hey Siri, Turn on Switch 1" or "Hey Siri, Turn off Switch 2" can be used to trigger events. Please note that the name of the device, is easily customizable, and can be sent to something generic such as "Lights", or something more specific like "Desk Lights".

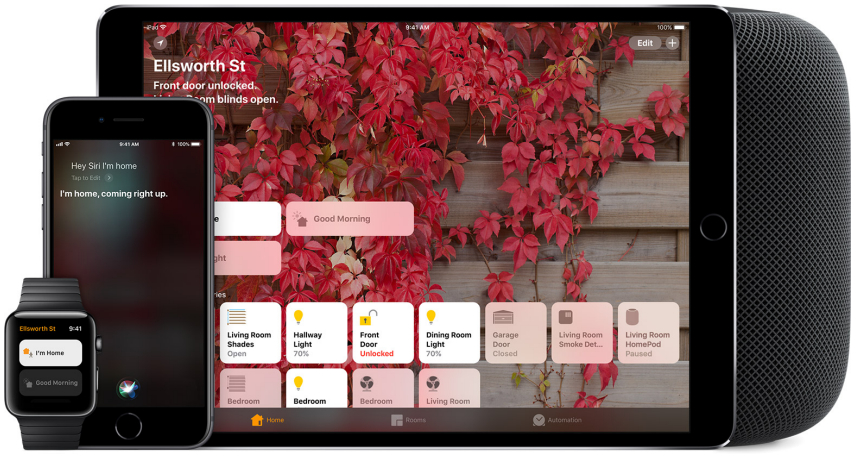

Figure 42 Compatible Apple device's family

Img src - https://www.apple.com/

10.7. Final installation

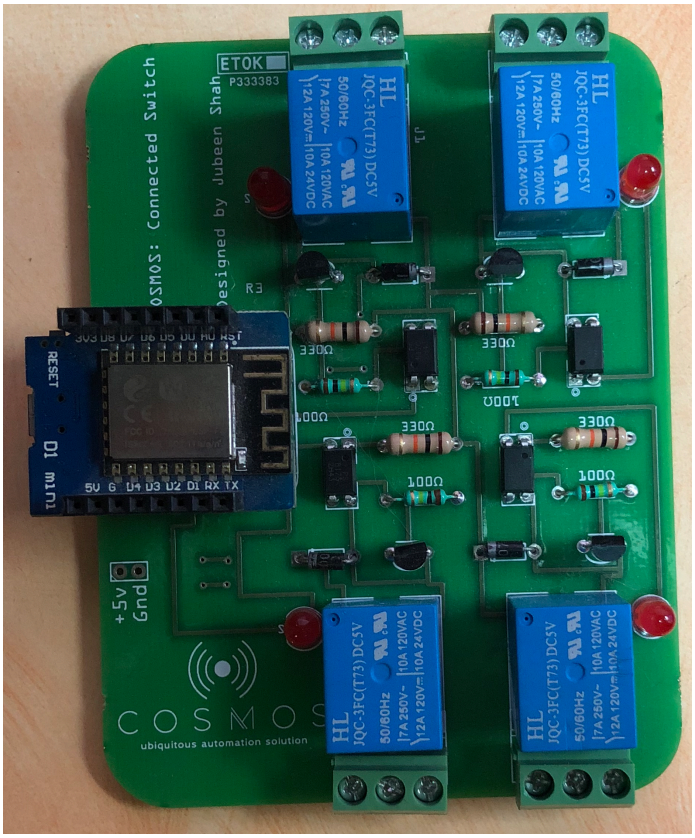

Figure 43 COSMOS: Connected Switch