**SMART ENVIRONMENT MONITORING SYSTEM**

**COMPUTER SCIENCE**

**H**

**SURAJ KUMAR PATTANAIK**

**CERTIFICATE**

This is to certify that the dissertation work entitled  **“SMART ENVIRONMENT MONITORING SYSTEM”** is the work done by **SURAJ KUMAR PATTANAIK** submitted in partial fulfillment for the award of ‘BACHELOR OF TECHNOLOGY ’ in “**COMPUTER SCIENCE ENGINEERING”** from KONARK INSTITUE OF SCIENCE AND TECHNOLOGY affiliated to B.P.U.T , ODISHA .

The results obtained in this project have not submitted to any other University or Institution for the award of any degree or diploma.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_**

**(HEAD OF THE DEPT.) (GUIDE)**

**EXTERNAL EXAMINER**

**ACKNOWLEDGEMENT**

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mentioning of the people whose constant guidance and encouragement made it possible.

I take pleasure in presenting before you, my project, which is result of studied blend of both research and knowledge.

I express our earnest gratitude to our project guide, Professor \_\_\_\_\_\_\_\_\_\_\_\_\_\_, Department of\_\_\_\_\_\_\_, for his constant support, encouragement and guidance. We are grateful for his cooperation and his valuable suggestions. Also I are thankful to H.O.D for motivation and encouragement for completing the project successfully.

CSE

S.K PADHY

Finally, I express our gratitude to all other members who are involved either directly or indirectly for the completion of this project.

**Signature**

**DECLARATION**

I , the undersigned, declare that the project entitled ‘**SMART ENVIRONMENT MONITORING SYSTEM**, being submitted in partial fulfillment for the award of Bachelor of Technology in \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Engineering, affiliated to BPUT, is the work carried out by me/us.

**COMPUTER SCIENCE ENGINEERING”**

The results embodied in this thesis have not been submitted to any other University or Institute for the award of any diploma or degree.

(SIGNATURE OF STUDENTS WITH REGISTARTION NUMBERS)

**CONTENTS PAGE NO.**

15

15

16

17

171920

22

26

26

30

31

37

1. ABSTRACT 6
2. INTRODUCTION 7

* **History**
* **Internet of Things Communications Models**

1. HARDWARE REQUIREMENTS

4. SOFTWARE REQUIREMENTS

5. CIRCUIT DIAGRAM(NODE MCU)

6. HARDWARE DESCRIPTION

* **ESP 8266**
* **DHT11**
* **FLAME SENSOR**

7. ARDUINO IDE

* **SKETCH BOOK**
* **LIBRARIES**
* **SERIEL MONITOR etc**

1. SOURCE CODE

1. CONFIGURATIONS

1. SCREENSHOTS FROM BLYNK APP
2. ADVANTAGES,DISADVANTAGES AND CONCLUSION

**IOT based Smart Environment Monitoring system**

**ABSTRACT**

The main objective of this project is to develop a smart environment monitoring system using a Node MCU board with Internet being remotely controlled by any Android OS smart phone. As technology is advancing so houses are also getting smarter.. Along with this we can get immediate notification if our environment catches fire. Humidity and temperature of our environment can be seen in our mobile phone through IOT.

In order to achieve this, a IOT module is interfaced to the Node MCU board at the receiver end while on the transmitter end, a GUI application on the cell phone sends ON/OFF commands to the receiver where loads are connected. A flame sensor is used for fire detection. DHT11 sensor is used for measuring temperature and humidity of our environment. Output of all above described sensor is feed to the Node MCU module and Node MCU module send the sensed data to my mobile phone through blynk application.

**INTERNET OF THINGS (IOT)**

**Introduction:**

The Internet of Things (IoT) is an important topic in technology industry, policy, and engineering circles and has become headline news in both the specialty press and the popular media. This technology is embodied in a wide spectrum of networked products, systems, and sensors, which take advantage of advancements in computing power, electronics miniaturization, and network interconnections to offer new capabilities not previously possible. An abundance of conferences, reports, and news articles discuss and debate the prospective impact of the “IoT revolution”—from new market opportunities and business models to concerns about security, privacy, and technical interoperability. It is basically connecting the electronic items to the cloud and in other words internet in order to make them more useful and worthful.

The large-scale implementation of IoT devices promises to transform many aspects of the way we live. For consumers, new IoT products like Internet-enabled appliances, environment automation components,there are many uses to household people and energy management devices are moving us toward a vision of the “smart environment’’, offering more security and energy efficiency. Other personal IoT devices like wearable fitness and health monitoring devices and network enabled medical devices are transforming the way healthcare services are delivered. This technology promises to be beneficial for people with disabilities and the elderly, enabling improved levels of independence and quality of life at a reasonable cost. IoT systems like networked vehicles, intelligent traffic systems, and sensors embedded in roads and bridges move us closer to the idea of “smart cities’’, which help minimize congestion and energy consumption. IoT technology offers the possibility to transform agriculture, industry, and energy production and distribution by increasing the availability of information along the value chain of production using networked sensors. There are lots of sensors in this electronic world using iot we can make them more worthful than ever.However, IoT raises many issues and challenges that need to be considered and addressed in order for potential benefits to be realized.

A number of companies and research organizations have offered a wide range of projections about the potential impact of IoT on the Internet and the economy during the next five to ten years. Cisco, for example, projects more than 24 billion Internet–connected objects by 2019; Morgan Stanley, however, projects 75 billion networked devices by 2020. Looking out further and raising the stakes higher, Huawei forecasts 100 billion IoT connections by 2025. McKinsey Global Institute suggests that the financial impact of IoT on the global economy may be as much as $3.9 to $11.1 trillion by 2025. While the variability in predictions makes any specific number questionable, collectively they paint a picture of significant growth and influence.There are enormous number of things we could see in future.

**History:**

The term “Internet of Things” (IoT) was first used in 1999 by British technology pioneer Kevin Ashton to describe a system in which objects in the physical world could be connected to the Internet by sensors.Ashton coined the term to illustrate the power of connecting Radio-Frequency Identification (RFID) tags used in corporate supply chains to the Internet in order to count and track goods without the need for human intervention. Today, the Internet of Things has become a popular term for describing scenarios in which Internet connectivity and computing capability extend to a variety of objects, devices, sensors, and everyday items.

While the term “Internet of Things” is relatively new, the concept of combining computers and networks to monitor and control devices has been around for decades. By the late 1970s, for example, systems for remotely monitoring meters on the electrical grid via telephone lines were already in commercial use. In the 1990s, advances in wireless technology allowed “machine–to–machine” (M2M) enterprise and industrial solutions for equipment monitoring and operation to become widespread. Many of these early M2M solutions, however, were based on closed purpose–built networks and proprietary or industry–specific standards, rather than on Internet Protocol (IP)–based networks and Internet standards.

Using IP to connect devices other than computers to the Internet is not a new idea. The first Internet “device”—an IP–enabled toaster that could be turned on and off over the Internet—was featured at an Internet conference in 1990. Over the next several years, other “things” were IP–enabled, including a soda machine at Carnegie Mellon University in the US and a coffee pot in the Trojan Room at the University of Cambridge in the UK (which remained Internet–connected until 2001). From these whimsical beginnings, a robust field of research and development into “smart object networking” helped create the foundation for today’s Internet of Things.

From a broad perspective, the confluence of several technology and market trends is making it possible to interconnect more and smaller devices cheaply and easily:

• **Ubiquitous Connectivity**—Low–cost, high–speed, pervasive network connectivity, especially through licensed and unlicensed wireless services and technology, makes almost everything “connectable’’.

• **Widespread adoption of IP–based networking**— IP has become the dominant global standard for networking, providing a well–defined and widely implemented platform of software and tools that can be incorporated into a broad range of devices easily and inexpensively.

• **Computing Economics**— Driven by industry investment in research, development, and manufacturing, Moore’s law continues to deliver greater computing power at lower price points and lower power consumption.

• **Miniaturization**— Manufacturing advances allow cutting-edge computing and communications technology to be incorporated into very small objects. Coupled with greater computing economics, this has fueled the advancement of small and inexpensive sensor devices, which drive many IoT applications.

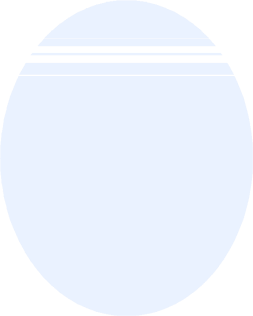
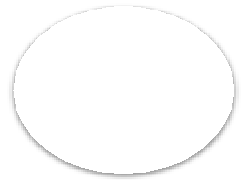
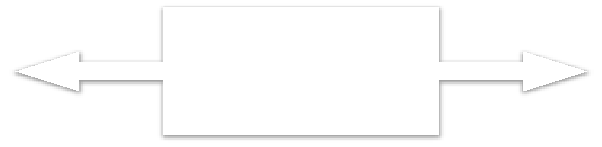
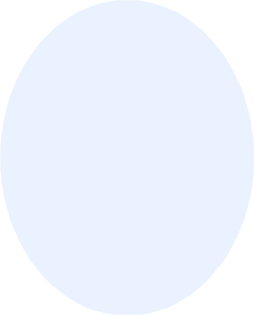
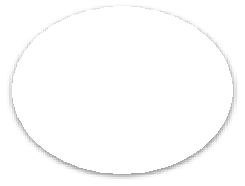
• **Advances in Data Analytics**— New algorithms and rapid increases in computing power, data storage, and cloud services enable the aggregation, correlation, and analysis of vast quantities of data; these large and dynamic datasets provide new opportunities for extracting information and knowledge.

• **Rise of Cloud Computing**– Cloud computing, which leverages remote, networked computing resources to process, manage, and store data, allows small and distributed devices to interact with powerful back-end analytic and control capabilities.

From this perspective, the IoT represents the convergence of a variety of computing and connectivity trends that have been evolving for many decades. At present, a wide range of industry sectors – including automotive, healthcare, manufacturing, environment and consumer electronics, and well beyond are considering the potential for incorporating IoT technology into their products, services, and operations.

**Internet of Things Communications Models:**

The device-to-device communication model represents two or more devices that directly connect and communicate between one another, rather than through an intermediary application server. These devices communicate over many types of networks, including IP networks or the Internet. Often, however thesedevices use protocols like Bluetooth, Z-Wave, or ZigBee to establish direct device-to-devicecommunications, as shown in Figure 1.



Light

Bulb

Wireless

Network

Light

Switch

ManufacturerA

**Bluetooth,Z+Wave,ZigBee**

ManufacturerB

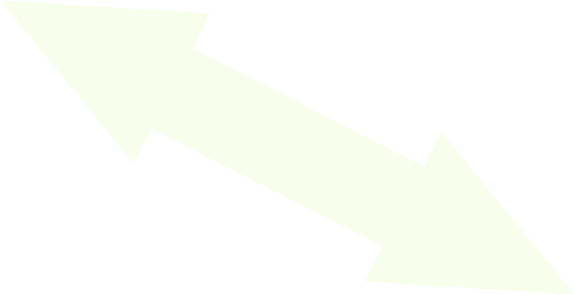
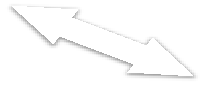
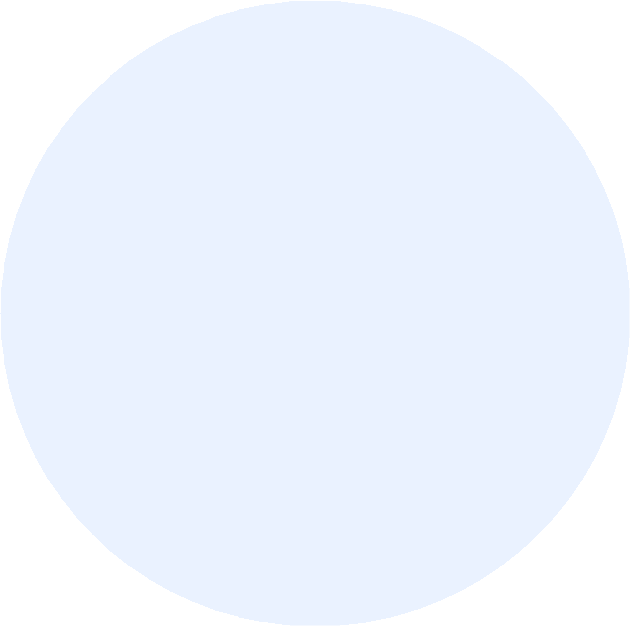
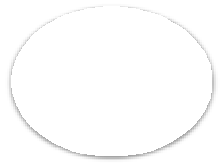
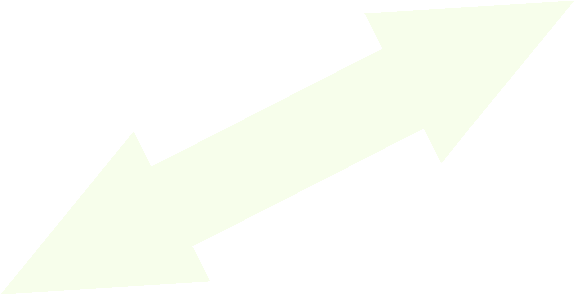
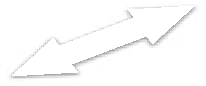
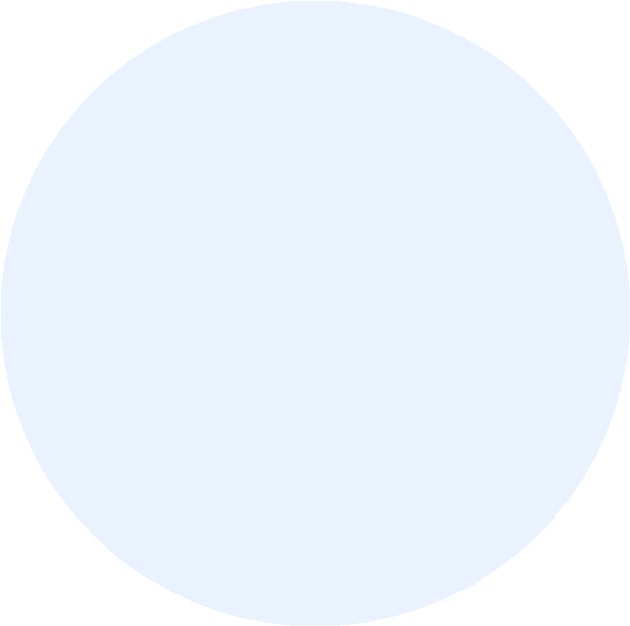
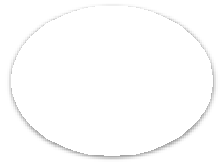
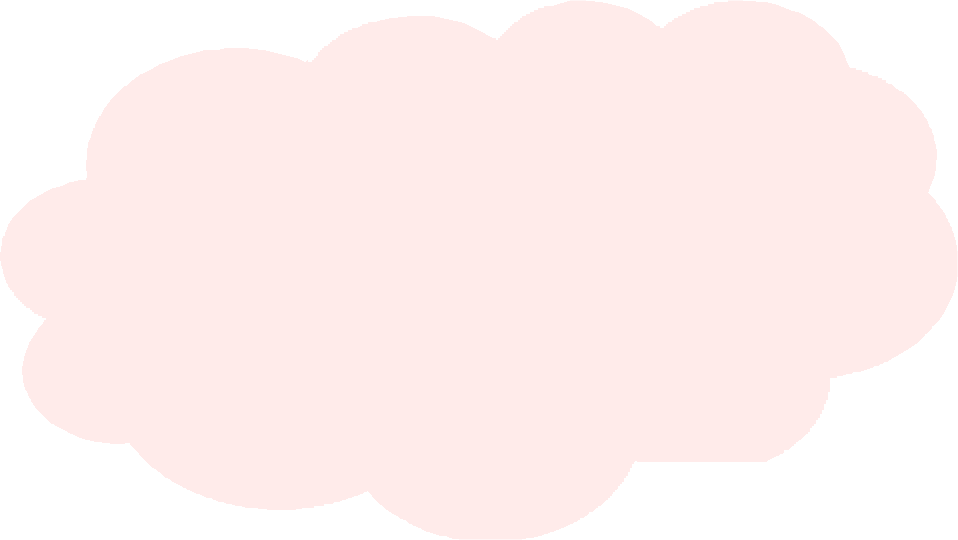
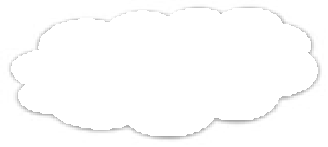
Source:Tschofenig,H.,et.al.,ArchitecturalConsidera/onsinSmartObjectNetworking.Tech.no.RFC7452.InternetArchitectureBoard,Mar.2015.Web.

<hRps://www.rfc&editor.org/rfc/rfc7452.txt>.

Figure 1. Example of device-to-device communication model.

**Device-to-Cloud Communications:**

In a device-to-cloud communication model, the IoT device connects directly to an Internet cloud service like an application service provider to exchange data and control message traffic. This approach frequently takes advantage of existing communications mechanisms like traditional wired Ethernet or Wi-Fi connections to establish a connection between the device and the IP network, which ultimately connects to the cloud service. This is shown in Figure 2.



**HTTP**

**TLSTCPIP**

ApplicationService

Provider

**CoAP**

**DTLSUDPIP**

Devicewith

Devicewith Carbon

Temperature Monoxide

Sensor Sensor

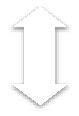
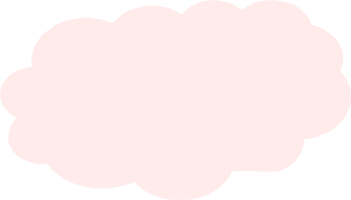
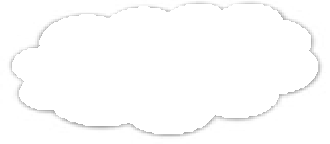
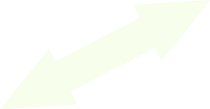
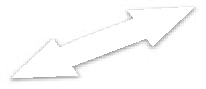
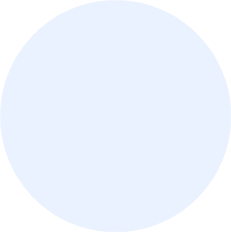
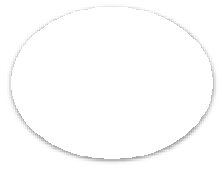
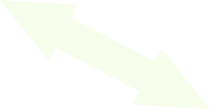
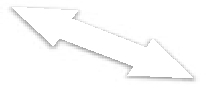
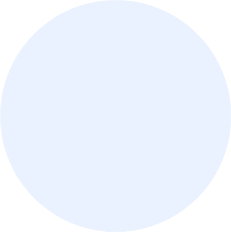
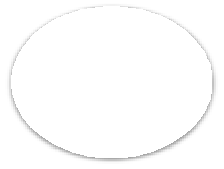
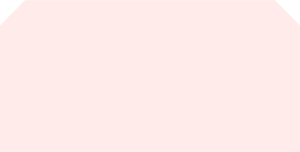
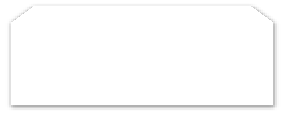
Source:Tschofenig,H.,et.al.,ArchitecturalConsidera9onsinSmartObjectNetworking.Tech.no.RFC7452.InternetArchitectureBoard,Mar.2015.Web.

<hN[ps://www.rfcQeditor.org/rfc/rfc7452.txt>.](http://www.rfcQeditor.org/rfc/rfc7452.txt)

Figure 2. Device-to-cloud communication model diagram.

**Device-to-Gateway Model:**

In the device-to-gateway model, or more typically, the device-to-application-layer gateway (ALG) model, the IoT device connects through an ALG service as a conduit to reach a cloud service. In simpler terms, this means that there is application software operating on a local gateway device, which acts as an intermediary between the device and the cloud service and provides security and other functionality such as data or protocol translation. The model is shown in Figure 3



ApplicationService

Provider

**IPv4/IPv6**

**Protocol**

**Stack**

**HTTP**

**TLSTCPIPv6**

LocalGateway

**CoAP**

**DTLSUDPIPv6**

Devicewith

TemperatureSensor

**Layer1Protocol**

**BluetoothSmartIEEE802.11(WiDFi)**

**IEEE802.15.4(LRDWPAN)**

Devicewith

CarbonMonoxideSensor

Source:Tschofenig,H.,et.al.,ArchitecturalConsidera9onsinSmartObjectNetworking.Tech.no.RFC7452.InternetArchitectureBoard,Mar.2015.Web.

<hN[ps://www.rfcQeditor.org/rfc/rfc7452.txt>.](http://www.rfcQeditor.org/rfc/rfc7452.txt)

Figure 3. Device-to-gateway communication model diagram.

**Back-End Data-Sharing Model:**

The back-end data-sharing model refers to a communication architecture that enables users to export and analyze smart object data from a cloud service in combination with data from other sources. This architecture supports “the [user’s] desire for granting access to the uploaded sensor data to third parties”. This approach is an extension of the single device-to-cloud communication model, which can lead to data silos where “IoT devices upload data only to a single application service provider’’. A back-end sharing architecture allows the data collected from single IoT device data streams to be aggregated and analyzed.

For example, a corporate user in charge of an office complex would be interested in consolidating and analyzing the energy consumption and utilities data produced by all the IoT sensors and Internet-enabled utility systems on the premises. Often in the single device-to-cloud model, the data each IoT sensor or system produces sits in a stand-alone data silo. An effective back-end data sharing architecture would allow the company to easily access and analyze the data in the cloud produced by the whole spectrum of devices in the building. Also, this kind of architecture facilitates data portability needs. Effective back-end data- sharing architectures allow users to move their data when they switch between IoT services, breaking down traditional data silo barriers.

The back-end data-sharing model suggests a federated cloud services approach or cloud applications programmer interfaces (APIs) are needed to achieve interoperability of smart device data hosted in the cloud. A graphical representation of this design is shown in Figure 4.

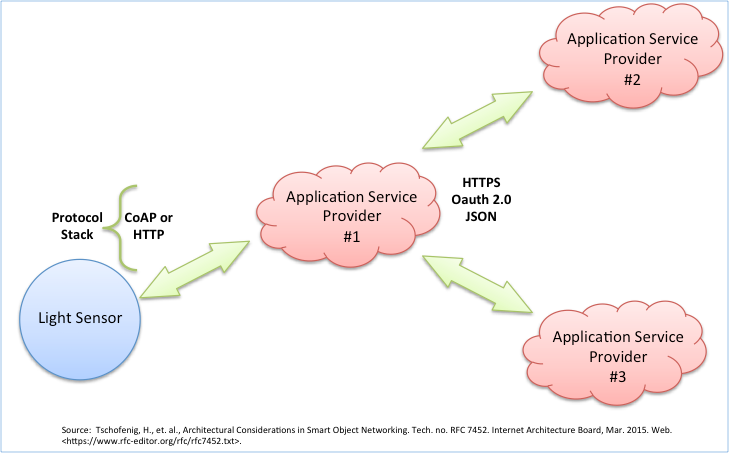


Figure 4. Back-end data sharing model diagram.

**Different Definitions, Similar Concepts:**

Despite the global buzz around the Internet of Things, there is no single, universally accepted definition for the term. Different definitions are used by various groups to describe or promote a particular view of what IoT means and its most important attributes. Some definitions specify the concept of the Internet or the Internet Protocol (IP), while others, perhaps surprisingly, do not. For example, consider the following definitions.

The Internet Architecture Board (IAB) begins RFC 7452, “Architectural Considerations in Smart Object Networking’’, with this description:

*The term "Internet of Things" (IoT) denotes a trend where a large number of embedded devices employ communication services offered by the Internet protocols. Many of these devices, often called "smart objects,’’ are not directly operated by humans, but exist as components in buildings or vehicles, or are spread out in the environment.*

Within the Internet Engineering Task Force (IETF), the term *“smart object networking”* is commonly used in reference to the Internet of Things. In this context, “smart objects” are devices that typically have significant constraints, such as limited power, memory, and processing resources, or bandwidth. Work in the IETF isorganized around specific requirements to achieve network interoperability between several types of smart objects.

Published in 2012, the International Telecommunication Union (ITU) ITU–T Recommendation Y.2060, Overview of the Internet of things, discusses the concept of interconnectivity, but does not specifically tie the IoT to the Internet:

*3.2.2 Internet of things (IoT): A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies*.

*Note 1—Through the exploitation of identification, data capture, processing and communication capabilities, the IoT makes full use of things to offer services to all kinds of applications, whilst ensuring that security and privacy requirements are fulfilled.*

*Note 2—From a broader perspective, the IoT can be perceived as a vision with technological and societal implications.*

This definition in a call for papers for a feature topic issue of IEEE Communications Magazine links the IoT back to cloud services:

*The Internet of Things (IoT) is a framework in which all things have a representation and a presence in the Internet. More specifically, the Internet of Things aims at offering new applications and services bridging the physical and virtual worlds, in which Machine-to-Machine (M2M) communications represents the baseline communication that enables the interactions between Things and applications in the cloud.*

The Oxford Dictionaries offers a concise definition that invokes the Internet as an element of the IoT:

*Internet of things (noun): The interconnection via the Internet of computing devices embedded in everyday objects, enabling them to send and receive data*

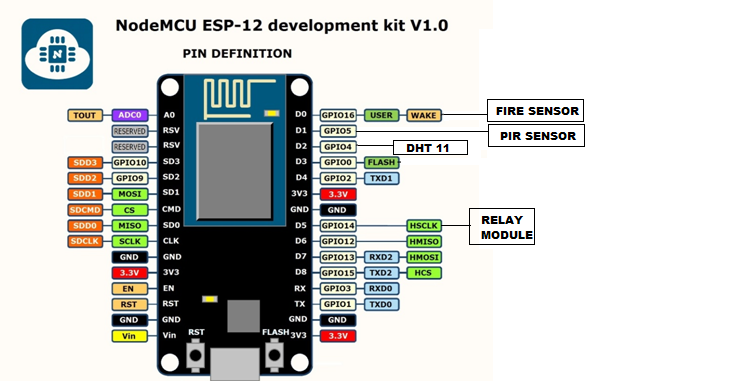
**HARDWARE REQUIREMENTS:**

* **5V POWER SUPPLY(MOBILE CHARGE/USB PORT)**
* **USB TO MINI USB DATA CABLE**
* **FLAME SENSOR**
* **DHT11 SENSOR**
* **NODE MCU**
* **FEMALE TO FEMALE CONNECTING WIRE**
* **PLY COVER**
* **BUZZER**

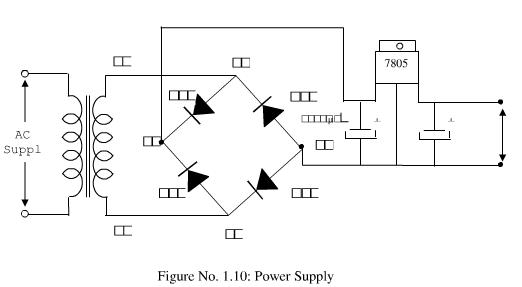
**SOFTWARE REQUIREMENTS:**

* **ARDUINO IDE**
* **USB TO SERIAL DRIVER**
* **ESP8266 BORAD DRIVER**
* **ESP8266 LIBRARY**
* **EMBEDDED C**
* **SOURCE CODE**

**Circuit Diagram :**



***POWER SUPPLY***



**HARDWARE DESCRIPTION:**

**ESP8266-12E:**

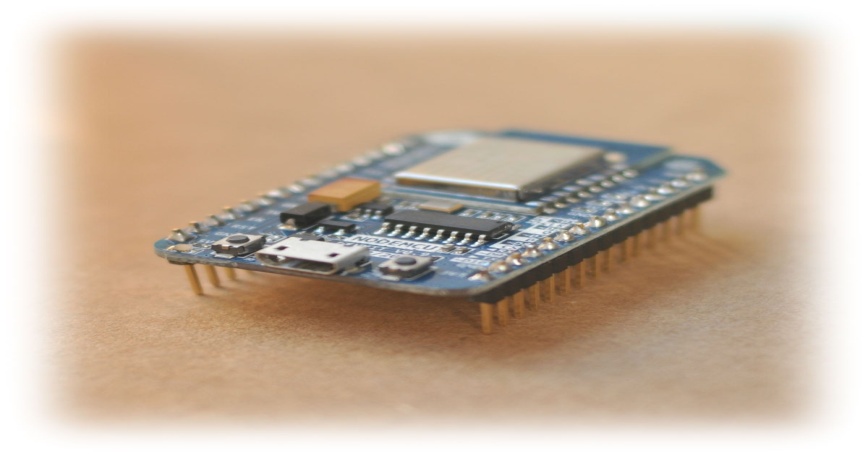
****

Fig. 8 ESP8266-12E

Your ESP8266 is an impressive, low cost WiFi module suitable for adding WiFi functionality to anexisting microcontroller project via a UART serial connection. The module can even be reprogrammedto act as a standalone WiFi connected device–just add power!The feature list is impressive and includes:802.11 b/g/n protocolWi-Fi Direct (P2P), soft-APIntegrated TCP/IP protocol stack.

**Summary:**

Microcontroller ESP-8266EX

Operating Voltage 3.3V

Digital I/O Pins 11

Analog Pins 1(Max input: 3.2V)

Clock Speed 80MHZ/160MHZ

Flash 4MB

Length 34.2mm

Width 25.6mm

Weight 10gm

**Board:**

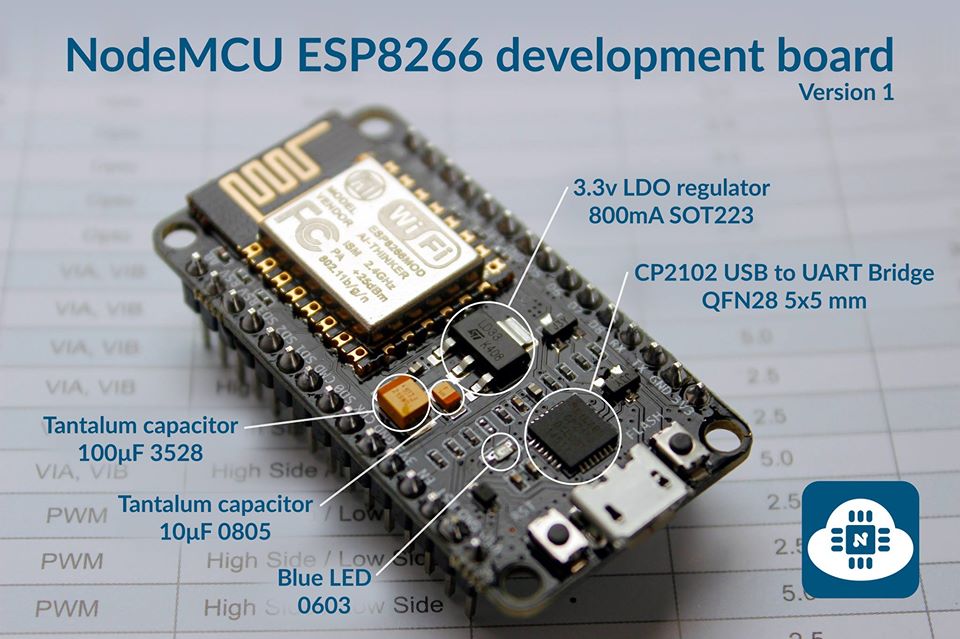
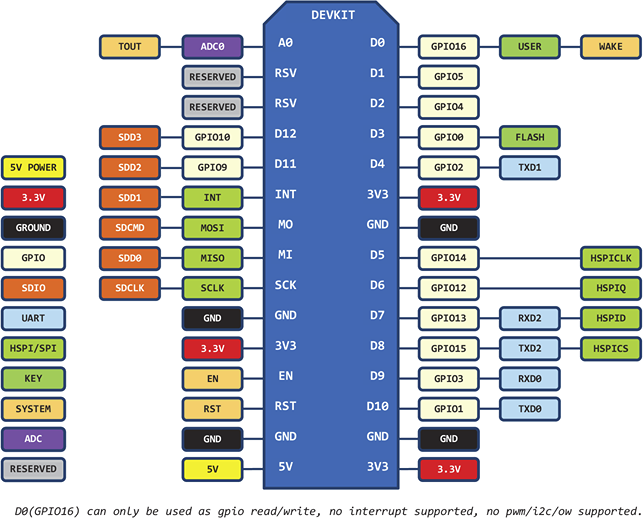
****

Fig. 8 ESP8266-12E board Description

ESP8266EX has been designed for mobile, wearable electronics and Internet of Things applicationswith the aim of achieving the lowest power consumption with a combination of several proprietarytechniques. The power saving architecture operates mainly in 3 modes: active mode, sleep modeand deep sleep mode.By using advance power management techniques and logic to power-down functions not requiredand to control switching between sleep and active modes, ESP8266EX consumes about than 60uA indeep sleep mode (with RTC clock still running) and less than 1.0mA (DTIM=3) or less than 0.5mA(DTIM=10) to stay connected to the access point.When in sleep mode, only the calibrated real-time clock and watchdog remains active. The real-timeclock can be programmed to wake up the ESP8266EX at any required interval.The ESP8266EX can be programmed to wake up when a specified condition is detected. Thisminimal wake-up time feature of the ESP8266EX can be utilized by mobile device SOCs, allowingthem to remain in the low-power standby mode until WiFi is needed.In order to satisfy the power demand of mobile and wearable electronics, ESP8266EX can beprogrammed to reduce the output power of the PA to fit various application profiles, by trading offrange for power consumption.

**Pin Definition:**

****

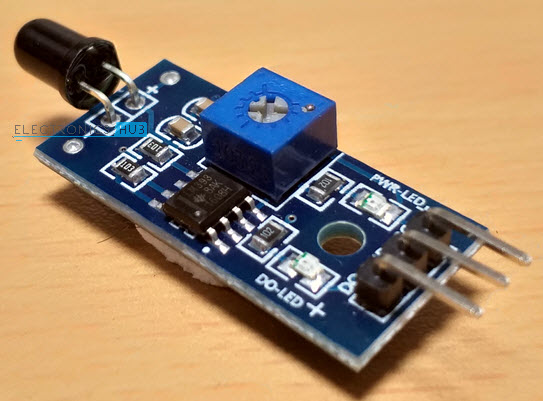
**DHT11:(Humidity/Temp Sensor)**

DHT11 digital temperature and humidity sensor is a composite Sensor contains a calibrated digital signal output of the temperature and humidity. Application of a dedicated digital modules collection technology and the temperature and humidity sensing technology, to ensure that the product has high reliability and excellent long-term stability. The sensor includes a resistive sense of wet components and an NTC temperature measurement devices, and connected with a high-performance 8-bit microcontroller.

### A Brief Note on Flame Sensor

A Flame Sensor is a device that can be used to detect presence of a fire source or any other bright light sources. There are several ways to implement a Flame Sensor but the module used in this project is an Infrared Radiation Sensitive Sensor.

The following image shows an Infrared type Flame Sensor.



This particular flame sensor is based on YG1006 NPN Photo Transistor. The black object at the front of the module is this Photo Transistor.

The YG1006 Photo Transistor looks like a black LED but it is a three terminal NPN Transistor, where the long lead is the Emitter and the shorter one is the collector (there is no base terminal as the light it detects will enable the flow of current).

This photo transistor is coated with black epoxy, making it sensitive to Infrared radiations and this particular Photo Transistor (YG1006) is sensitive to Infrared Radiation in the wavelength range of 760nm to 1100nm.

Using this particular type of Flame Sensor, you can detect Infrared Light up to a distance of 100cm within its 60 degrees of detection angle.

There are two types of implementations of Flame Sensors using YG1006 Photo Transistor: one is with both Analog Output and Digital Output while the other is with only the Digital Output.

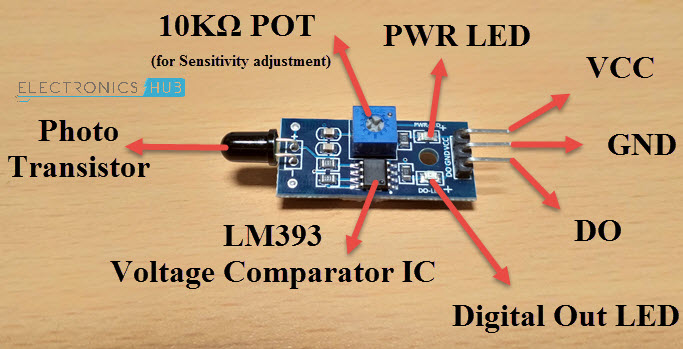
Both these implementations require same components but the difference is that one module (the one with the Analog Output) provides the Sensor output as Analog Output.

The Flame Sensor that I am using in this project has only Digital Output.

**NOTE:** In the circuit diagram of the Flame Sensor, I have pointed out where to get the Analog Output if your module doesn’t have that option.

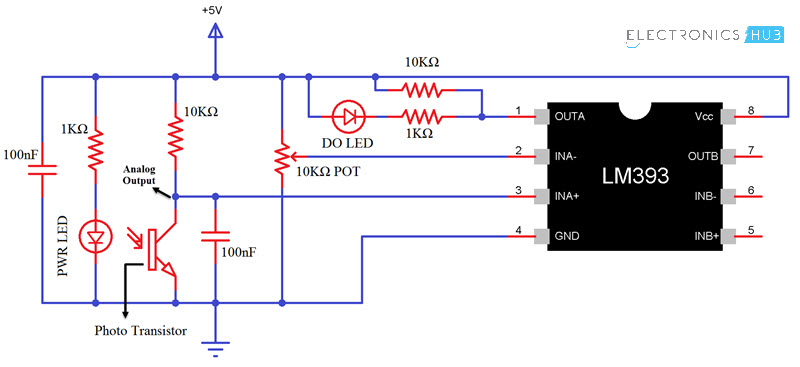
#### Components of Flame Sensor Module

The following image shows all the components of a typical Flame Sensor Module.



#### Circuit Diagram of Flame Sensor Module

If you want to know a little bit more about the Flame Sensor Module, then analyzing its circuit will probably help you. The following image shows the circuit diagram of a Flame Sensor.



### Arduino Flame Sensor Interface

Whether you are using either of the Flame Sensors (with or without Analog Output), interfacing it with Arduino (or any other Microcontroller) is very easy. Since I do not have an Analog Output, I won’t go into that details.

By interfacing a Flame Sensor with Arduino, you can detect fire and activate a Buzzer (simple and easy implementation) or any other emergency safety measurements (like activating a sprinkler system or shutting off gas valves etc.).

**SOFTWARE DESCRIPTION**

**Arduino IDE:**

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

### Writing Sketches

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

### Sketchbook

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

**Beginning with version 1.0, files are saved with a .ino file extension. Previous versions use the .pde extension. You may still open .pde named files in version 1.0 and later, the software will automatically rename the extension to .ino**.

### Tabs, Multiple Files, and Compilation

Allows you to manage sketches with more than one file (each of which appears in its own tab). These can be normal Arduino code files (no visible extension), C files (.c extension), C++ files (.cpp), or header files (.h).

### Uploading

Before uploading your sketch, you need to select the correct items from the **Tools > Board** and **Tools > Port** menus. The[boards](https://www.arduino.cc/en/Guide/Environment#boards) are described below. On the Mac, the serial port is probably something like **/dev/tty.usbmodem241** (for an Uno or Mega2560 or Leonardo) or **/dev/tty.usbserial-1B1** (for a Duemilanove or earlier USB board), or**/dev/**tty.USA19QW1b1P1**.1** (for a serial board connected with a Keyspan USB-to-Serial adapter). On Windows, it's probably COM1 or COM2 (for a serial board) or COM4, COM5, COM7, or higher (for a USB board) - to find out, you look for USB serial device in the ports section of the Windows Device Manager. On Linux, it should be **/dev/ttyACMx** ,**/dev/ttyUSBx** or similar. Once you've selected the correct serial port and board, press the upload button in the toolbar or select the **Upload** item from the **File** menu. Current Arduino boards will reset automatically and begin the upload. With older boards (pre-Diecimila) that lack auto-reset, you'll need to press the reset button on the board just before starting the upload. On most boards, you'll see the RX and TX LEDs blink as the sketch is uploaded. The Arduino Software (IDE) will display a message when the upload is complete, or show an error.

When you upload a sketch, you're using the Arduino **boot-loader**, a small program that has been loaded on to the microcontroller on your board. It allows you to upload code without using any additional hardware. The boot-loader is active for a few seconds when the 57board resets; then it starts whichever sketch was most recently uploaded to the microcontroller. The boot-loader will blink the on-board (pin 13) LED when it starts (i.e. when the board resets).

### Libraries

Libraries provide extra functionality for use in sketches, e.g. working with hardware or manipulating data. To use a library in a sketch, select it from the **Sketch > Import Library** menu. This will insert one or more **#include** statements at the top of the sketch and compile the library with your sketch. Because libraries are uploaded to the board with your sketch, they increase the amount of space it takes up. If a sketch no longer needs a library, simply delete its **#include**statements from the top of your code. Some libraries are included with the Arduino software. Others can be downloaded from a variety of sources or through the Library Manager. Starting with version 1.0.5 of the IDE, you do can import a library from a zip file and use it in an open sketch

### Third-Party Hardware

Support for third-party hardware can be added to the **hardware** directory of your sketchbook directory. Platforms installed there may include board definitions (which appear in the board menu), core libraries, bootloaders, and programmer definitions. To install, create the **hardware** directory, then unzip the third-party platform into its own sub-directory. (Don't use "arduino" as the sub-directory name or you'll override the built-in Arduino platform.) To uninstall, simply delete its directory.

### Serial Monitor

Displays serial data being sent from the Arduino or Genuino board (USB or serial board). To send data to the board, enter text and click on the "send" button or press enter. Choose the baud rate from the drop-down that matches the rate passed to **Serial.begin** in your sketch. Note that on Windows, Mac or Linux, the Arduino or Genuino board will reset (rerun your sketch execution to the beginning) when you connect with the serial monitor.

You can also talk to the board from Processing, Flash, MaxMSP, etc (see the [interfacing page](http://www.arduino.cc/playground/Main/Interfacing) for details).

Wearable Electronics

* The Arduino IDE is the computer software that will make it possible to create, modify, and upload programs called ***sketches*** on the computer.
* The IDE window contains a ***Menu bar***, a ***Toolbar*** with buttons for common functions, a ***Text editor*** for writing the sketch, ***a Message area*** and a ***Text console***
* ***Sketches***  are written on the Text editor area of the IDE window. Sketches are saved with the file extension ***.ino*** which are taken from the last three letters of the word ACEduino.
* the ***Toolbar*** buttons allow short cut clicks to verify and upload programs, create, open and save sketches, and open the serial monitor.
* The ***Message Area*** displays the IDE feedback like when saving and uploading files.
* The ***Text console*** also displays text output by the Arduino environment such as complete ***error*** and other information.
* The bottom right hand corner of the window displays the currently connected type of Arduino board and the designated computer serial port.
* The ***number*** in the bottom left hand corner is the row count of the cursor.

**Source code :**

**#define BLYNK\_PRINT Serial**

**#include <ESP8266WiFi.h>**

**#include <BlynkSimpleEsp8266.h>**

**#include <DHT.h>**

**#define DHTPIN D2**

**#define DHTTYPE DHT11**

**DHT dht(DHTPIN, DHTTYPE);**

**uint8\_t h, t;**

**int fire;**

**//Blynk Timer timer;**

**// You should get Auth Token in the Blynk App.**

**// Go to the Project Settings (nut icon).**

**char auth[] = "** **vukGsVAZT0QPOR7fQjvQ80Q1HgipynSb ";**

**// Your WiFi credentials.**

**// Set password to "" for open networks.**

**char ssid[] = "IOT";**

**char pass[] = "12345678";**

**// What digital pin we're connected to**

**// This function sends Arduino's up time every second to Virtual Pin (5).**

**// In the app, Widget's reading frequency should be set to PUSH. This means**

**// that you define how often to send data to Blynk App.**

**void sendSensor()**

**{**

**h = dht.readHumidity();**

**t = dht.readTemperature(); // or dht.readTemperature(true) for Fahrenheit**

**if (isnan(h) || isnan(t)) {**

**Serial.println("Failed to read from DHT sensor!");**

**return;**

**}**

**// You can send any value at any time.**

**// Please don't send more that 10 values per second.**

**Blynk.virtualWrite(V5, h);**

**Blynk.virtualWrite(V6, t);**

**Serial.println(h);**

**Serial.println(t);**

**}**

**void setup()**

**{**

**// Debug console**

**Serial.begin(115200);**

**dht.begin();**

**delay(10);**

**Blynk.begin(auth, ssid, pass);**

**pinMode(D7,OUTPUT);//buzzer**

**pinMode(D5,INPUT); // flame/fire sensor**

**}**

**void loop()**

**{**

**Blynk.run();**

**fire=digitalRead(D5);**

**if(fire==0)**

**{**

**digitalWrite(D7,HIGH); // buzzer activate**

**Serial.println("FIRE DETECTED");**

**Blynk.virtualWrite(V2,"FIRE DETECTED\n");**

**Blynk.notify("FIRE DETECTED");**

**Blynk.email("surajkumarpattanaik3@gmail.com","FIRE ALERT","FIRE DETECTED");**

**}**

**else**

**{**

**digitalWrite(D7,LOW); // buzzer deactivate**

**Serial.println("FIRE NOT DETECTED");**

**//Blynk.print(V2,"FIRE DETECTED\n");**

**}**

**delay(1000);**

**//timer.run();**

**sendSensor();**

* *You must write the correct unique authentication token received from your Blynk mobile application at the “auth” space.*
* *Local Network Id name and Password must be correct.*

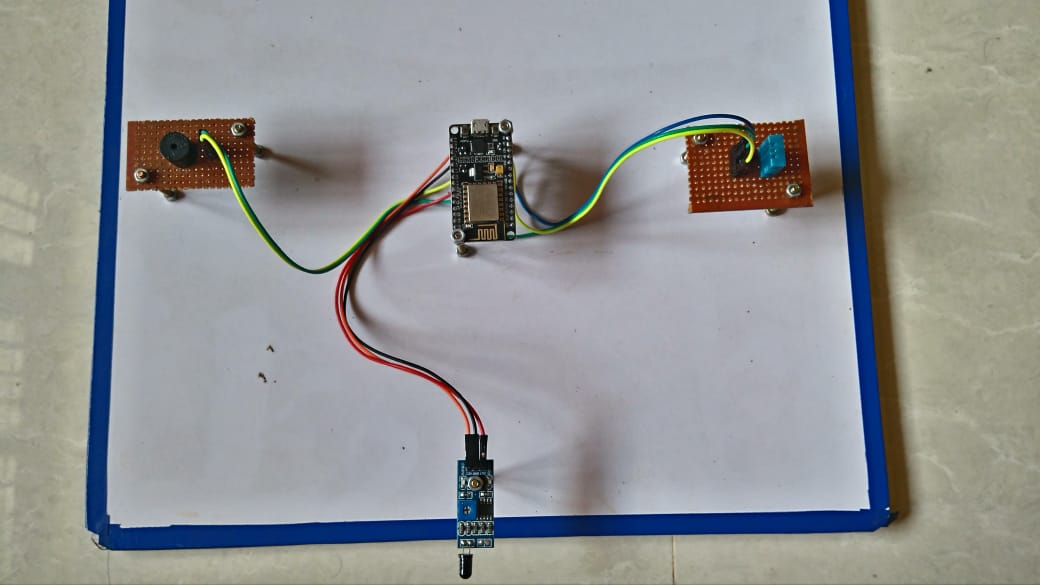
**HARDWARE AND SOFTWARE CONFIGURATIONS:**

**Connections:**

**1: Flame Sensor:-** It has 3 pins. Those are OUTPUT ,GND and Vcc. Connect the OUT pin to the D5 of the Node Mcu (as defined in the source code). GND to gnd and Vcc to 3v3 Of the nodeMcu.

**2: DHT11:-** Despite of having 4 pins , one of the pin(3rd no) is of no use here. Hence Connect the remaining three pins(i.e: Vin,GND and Input ) with the respective pins of Node Mcu.

**3:Buzzer:-** Connect the OutPut(+) pin to D7 and Gnd to Gnd of NodeMcu.



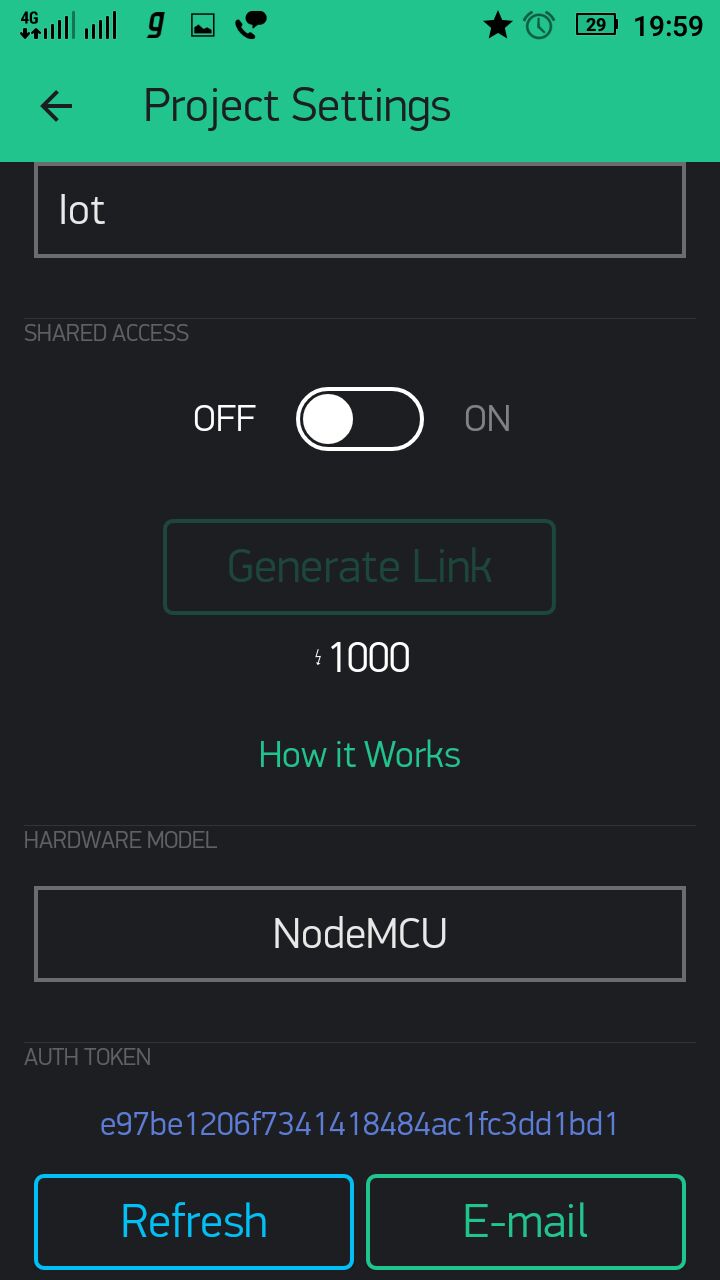
**In the Blynk App:-**

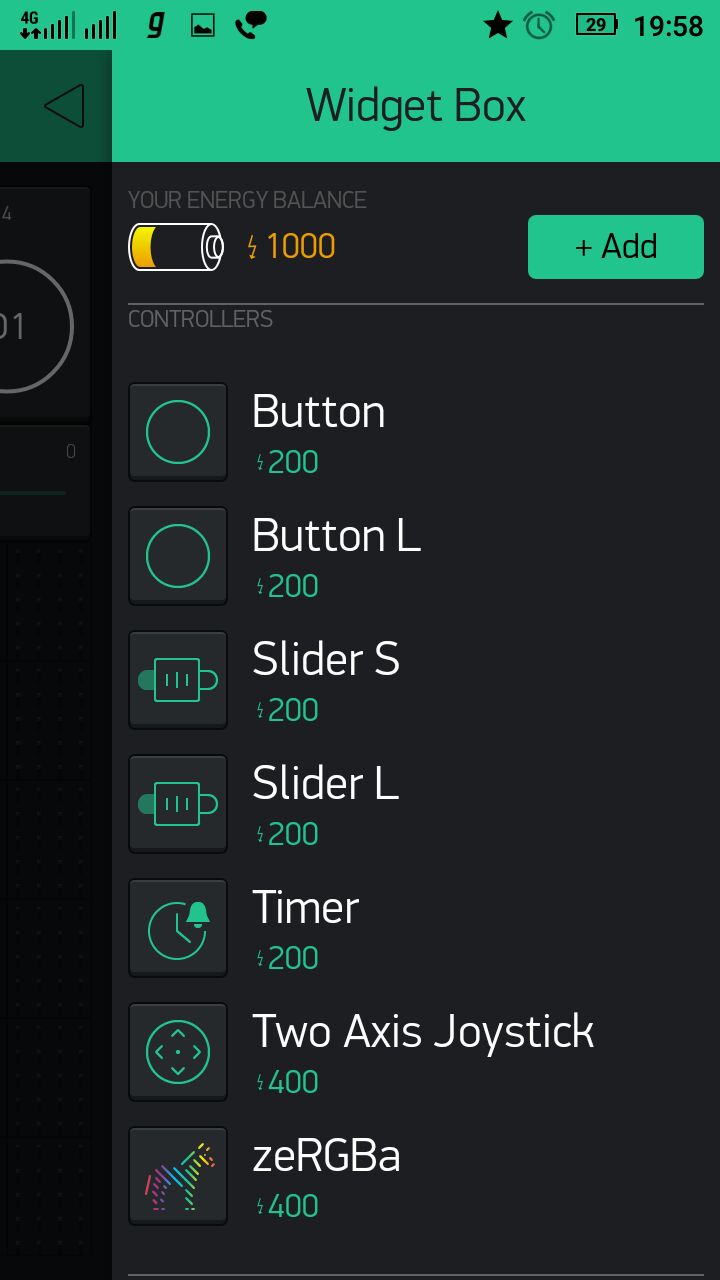
* Create a new project ,and choose device to ESP8266.Your Authentication token will be sent to your given email address.
* Thereafter add two buttons with virtual pins V5 and V6. Name them as HUMIDITY and TEMPARATURE respectively. Add a Terminal with Virtual pin v2,that will show real time alerts.
* Add Two more switches for Notification and Email alert.
* If the Source Code and Over all Configuration Is perfect , you can see the real time humidity and temp in your Blynk app and can get live fire alerts in your Phone.

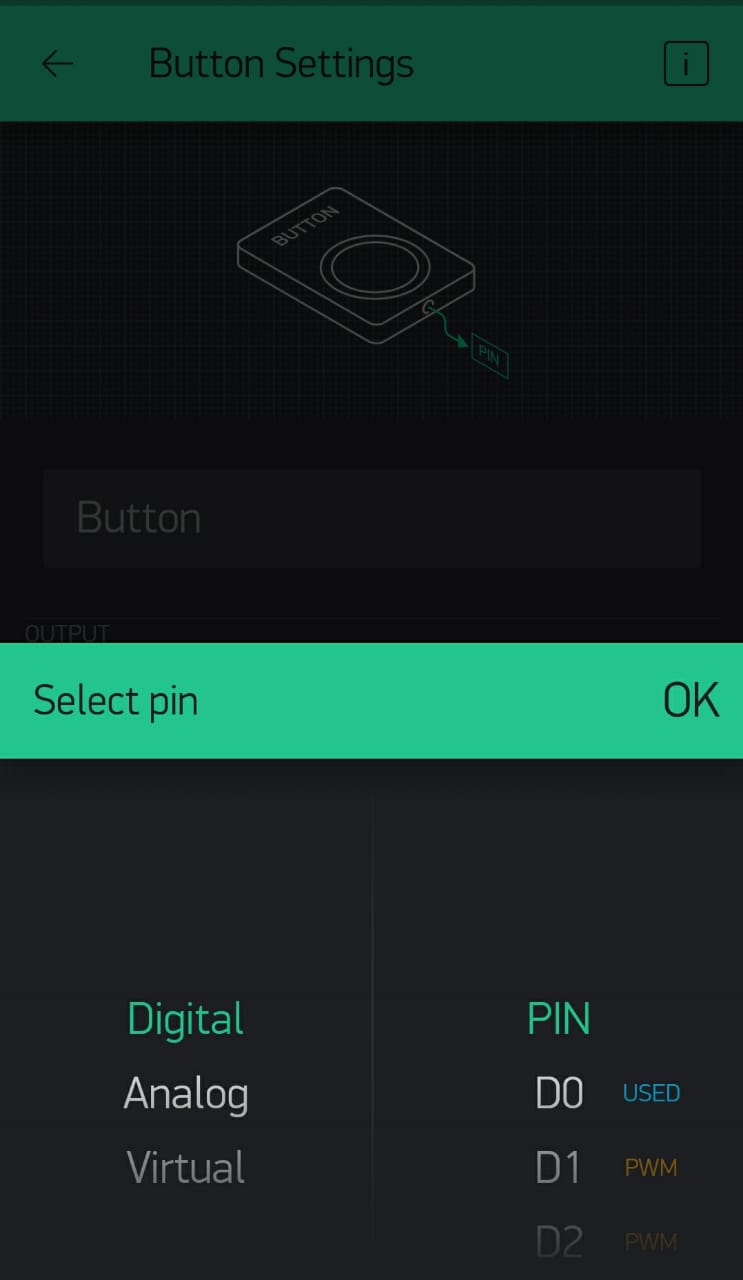
Buzzer

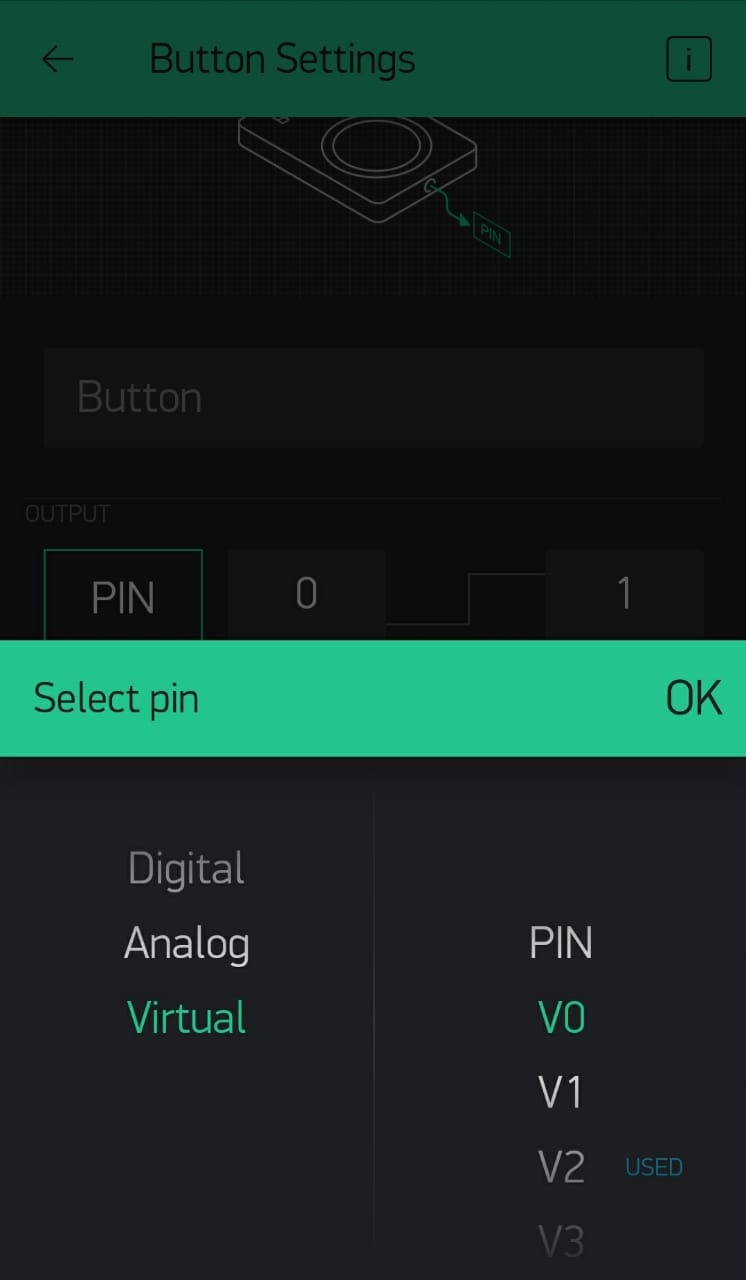
NODE MCU DHT11

FLAME SENSOR

**Blynk APP :**

****

****

****

****

**Advantages:-**

* Fully automatic no human interface is there.
* Our environment becomes fully secure
* Fire alarm on IOT
* Low cost investment

**Disadvantages:-**

* Always power supply required
* Internet connection must be there.
* future scope
* This system can be integrated with a CCTV so that we see live video at time of thief enters.

**Conclusion:-**

* We successfully designed and test its all working condition of power supply unit. It works as per our described with our objectives.