INDIAN INSTITUTE OF TECHNOLOGY PATNA

CS225 – CS226 PROJECT REPORT

NAME : ARIHANT K R

ROLL NO. : 1901CS09

PROJECT : HOME AUTOMATION

AIM:-

The main aim of this project is to show how electronic appliances and systems can be automated and monitored from remote areas and how automatic security systems warn people in case of any threats without any continuous human monitoring required. And also at the end of the project I will show a prototype of real life traffic light systems which are automatic and remotely started or stopped by knowing their current status and commanding the system based on the time and requirement.

MOTIVATION:-

The main motivation behind this project is that, this project today has a lot of real life applications, the concepts involved in this project has a significant role in almost every automatic sytem we find around us today whether the task is small or big.

COMPONENTS:-

All the components used in this project include :-

- NodeMCU ESP8266
- Micro USB cable
- LDR SENSOR
- DHT11 SENSOR
- ELECTROMAGNETIC BUZZER
- BREADBOARD
- LED BULBS(RED, BLUE, GREEN)
- MALE TO MALE JUMPER WIRES
- MALE TO FEMALE JUMPER WIRES
- SINGLE STRANDED COPPER WIRES

The above components along with the assistance of a Laptop and an android phone includes all the equipment which I have used to make this project.

MORE ABOUT EACH COMPONENT USED:-

1)NODEMCU ESP8266 WIFI DEVELOPMENT BOARD:-

ESP8266 Arduino core is a collection of software and hardware components required by the board manager and the Arduino IDE to compile an Arduino C/C++ source file for the target MCU's machine language. And NodeMCU is a low-cost open source IoT platform which is configured within this board specially to connect it to a LAN using wifi or to the WAN using mqtt service providers.

2)DHT11 SENSOR :- The DHT11 is a basic, ultra low cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the temperature of the surrounding air, and spits out a digital signal on the data pin(no analog input pins required).Its fairly simple to use, but requires careful timing to grab data. The Vcc pin and the ground pin is connected to the high voltage and the ground of the NodeMCU and the data pin to a digital signal pin of the NodeMCU board.

3)LDR SENSORS :- Photoresistors, also known as light dependent resistors(LDR), are light sensitive devices most often used to indicate the the presence or absence of light, or to measure the light intensity. The working principle of a LDR is photoconductivity, which is nothing but an optical phenomenon. When the light falls on the LDR, then the electrons in the valence band of the material excite to the conduction band and their resistance decreases as the light intensity increases: in the dark and at low light levels, the resistance of a LDR is high and little current can flow through it. The LDR sensor also has three pins the Vcc pin and ground pin connected to the high voltage and ground of NodeMCU board and the data pin connected to the analog signal pin of NodeMCU.

4)*ELECTROMAGNETIC BUZZER:-* An electromagnetic buzzer consists of an oscillator, solenoid coil, magnet, vibration diaphragm, where it produces sound through magnetism, with a frequency of 2KHz. The vibrating disk in a buzzer is attracted by the pole by the magnetic field. When an oscillating signal is moved through the coil, it produces a fluctuating manetic field which vibrates the disk at a frequency equal to that of the drive signal resulting in production of periodic sound beats.

5)*BREADBOARD*:- Breadboard is used to make quick electrical connections between components like batteries, resistors, capacitors and LEDs. In our project we will use two breadboards, one for the remote automation of simple house hold appliances and to find out humidity, temperature at that location and the other for remotely automated Traffic light system prototype.

6)MicroUSB CABLE:- MicroUSB cable is used to transfer as a data cable between the laptop and the Arduino board,which acts as a medium to transfer the data or code in this case from the system to the IoT device which would function as instructed by the given data.

7)JUMPER WIRES:- Jumper wires are wires of fixed lengths which are directly usable, we have used two types of jumper wires, Male to Male and Male to Female, Male to Male wires are used on the breadboard to make a direct connection between any two lines as they have two copper wires extended at the ends. Male to Female jumper wires have a hole at one end and an extended conducting

wire at the other end, where the hole is used to connect that end to any pin(digital, analog, ground or voltage) and the other end to the breadboard directly.

8)SINGLE STRANDED COPPER WIRES:- Their function is also same as the male to male jumper wiresbut these wires come in long lengths like around 2m and more, one only difference is that these wires can be cut into smaller parts for our usage just like male to male jumper wires.

9)LEDs:- LEDs also known as light emitting diodes which emit good amount of light using small voltage difference are used in our project.

SETTING UP ARDUINO IDE FOR NODEMCU:-

Once the Arduino IDE is downloaded and installed, it is absolutely free to use it for most of the traditional Arduino cores such as Arduino UNO or Arduino Nano. But for boards like NodeMCU additional drivers and packages needs to be installed for this we will have to follow the following steps.

- In your Arduino IDE go to **File>Preferences**
- Enter the following URL in Additional Boards Manager URLs. (http://arduino.esp8266.com/stable/package_esp8266com_index.json)

- Open the Boards Manager. Go to **Tools>Board>Boards Manager**
- Search for ESP8266 and press install for the “ESP8266 by ESP8266 Community”.
- Now if you go to **Tools>Board** you will find all the NodeMCU options which were not present previously.
- Now select NodeMCU 1.0(ESP-12E Module)
- And now when you connect the MicroUSB connected to your NodeMCU board to the system, in **Tools> (it must show you the port)**. In case it doesn’t show you the port you must once check if it the USB cable you are using allows data transfer also or it allows only charge transfer.

ABOUT THIS PROJECT:-

This project has been divided into three parts

- ❖Automate appliances and systems from anywhere in the world.
- ❖Automatic Security Systems
- ❖Automatic Traffic Lights System Prototype

In the part 1 of this project I would control and operate over simple home appliances from any part of the world using MQTT protocol which is the most important part of this project and here I would be

using UBIDOTS, a MQTT service provider as a broker between to communicate with the servers as a client.

In the part 2 of this project, we develop a simple model of home security system, showing how automatic security systems react to various threats instantaneously by themselves without any human monitoring.

In the part 3 of this project, we show a real life traffic light system prototype, which is actually a small prototype of how traffic light systems work actually in real life which we see around us.

PART 1:-

As described above in this part I will control and operate a LED bulb at my home using internet and also note that it can be operated from anywhere in the world.

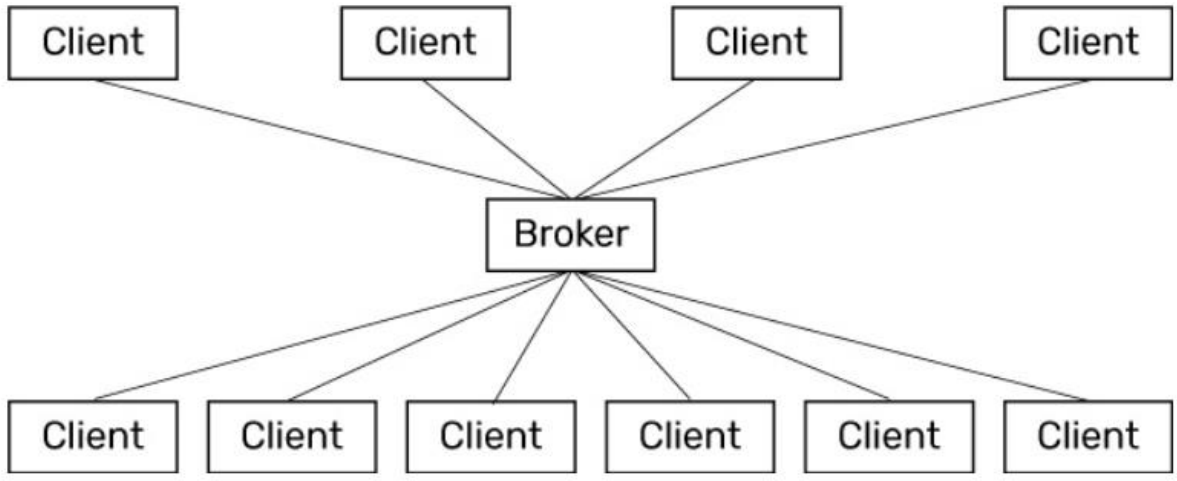
In this part I would be using the MQTT protocol service provider Ubidots to act as a broker between me(client) and the server.

Coming in bit detail about the MQTT protocol, unlike HTTP request-response model this follows a publish-subscribe model, which means that client can publish certain

messages to the server and it can subscribe to receive some messages when there are some published messages. Clients can also subscribe to some topics, so that whenever there are any published messages over the topic, the client also receives the message. These are all the important features of MQTT protocol which makes it more useful for IoT devices.

	MQTT	HTTP
Design orientation	Data centric	Document centric
Pattern	Publish/subscribe	Request/response
Complexity	Simple	More complex
Message size	Small, with a compact binary header just two bytes in size	Larger, partly because status detail is text-based
Service levels	Three quality of service settings	All messages get the same level of service
Extra libraries	Libraries for C (30 KB) and Java (100 KB)	Depends on the application (JSON, XML), but typically not small
Data distribution	Supports 1 to zero, 1 to 1, and 1 to *n*	1 to 1 only

Above is a small image which gives us a simple comparision between MQTT protocol and HTTP.



And the above image here, is just an example of how brokers facilitate communication between clients and servers in the publish/subscribe model.

And in this project *Ubidots* is the website used as a broker which would act as a MQTT service provider to enable publish/subscribe model communication between the clients and servers.

PART 2:-

As described above in the part 2, of this project we build a mini home security system, using a LDR, electromagnetic buzzer and DHT11 sensor.

Here consider a case of burglary where, someone has entered your home at your absence, and in search of something he has bought some flashlights to find out things, now as the LDR sensor is interrupted due to the light of these flashlights, the electromagnetic buzzer starts beeping, sending a signal of security threat to the entire neighbourhood and also it sends me a mail notifying me that the LDR sensor has been interrupted, signaling an external unknown threat, along with the temperature and humidity of the place at that time as recorded by DHT11 sensor.

And also in this part, there is a requirement of Arduino sending us an e-mail, when the LDR sensor gets interrupted, for the NodeMCU board to perform this operation, just like in the part 1 we will have to use the SMTP protocol.

SMTP protocol just like MQTT and HTTP protocols is also an Application layer IoT protocol expanded as Simple Mail Transfer Protocol. It is also used to send, receive and relay outgoing mails between senders and receivers. When an

email is sent, its transferred over the internet from one server to other using SMTP.

Just as in part 1, here also we use a SMTP service provider for brokerage that is *smtp2go* an other online website just like ubidots as in part 1 for email communication.

The Part 1 and Part 2 of our project are executed in the same code, hence we will look into our code for part 1 and part 2 now.

CODE FOR PART 1 AND PART 2:-

INITIALISATION:- The below picture shows the initialization part of code, where macros for all the pins being used are defined. Apart from the traditional libraries you can also find the ubidots library which is used to link the NodeMCU to the Ubidots account of that client whose various details have been coded such as the “TOKEN” and his devices “DEVICE_LABEL” along with a variable “VARIABLE_TO_SUBSCRIBE”, apart from these the library used also provides the NodeMCU a facility to send mail along with several other class functions which are used further.

And also the SSID and PASSWORD of the wifi network to which the nodemcu has to be connected should also be provided as done below. The DHT library included is for the sake of DHT sensor functioning.

The most important thing is that you can also notice that using the token provided at my ubidots account the ubidots is creating a bridge between the client and the code(i.e nodemcu)for data transfer.

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

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

#define WIFISSID "Arihant KR"                                  // Put your WifiSSID here
#define PASSWORD "333333333333"                              // Put your wifi password here
#define TOKEN "BBFF-nPzQ3r0ckEHHb3v3IJpcfkQTXvoFiK"    // Ubidots Token
#define VARIABLE_TO_SUBSCRIBE "home_bulb"
#define DEVICE_LABEL "esp8266"


Ubidots client(TOKEN);
DHT dht = DHT(DHTPIN, DHTTYPE);
bool ismailsent=0;
```

AUXILIARY FUNCTIONS:-

These are some utility functions which are very important to operate the appliances from ubidots, the one and only most important function here is the callback function which is very important for us to keep checking if there is any command being passed out by us in the server. If it is being passed out the home appliances would reconfigure based on the commands it receives there on the server and as per that the devices reconfigure.

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

SETUP FUNCTION:- Setup is one of the main functions of the project where I setup the entire important pins telling what their function is like whether they take input or the output and also this is the same place where the entire ubidots account of mine gets connected to my nodemcu using all the details I have used in the initialization part above. And this is the same place which activates the serial monitor and also starts up the DHT11 sensor as you can see in my setup code below.

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

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

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

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

  dht.begin();
}
```

LOOP FUNCTION:- As this function is pretty big, I have divided this function into two images below. This is also one of the main function. At the very start of the function, what I'm doing is from that I read the resistance reading at the LDR sensor. At the LDR sensor as the light intensity increases its resistance decreases as its resistance decreases I would switch on the Buzzer if the resistace goes less than a given value and send the mail with a message as wriiten in the string below, else if the resistance doesn't go less than that value I would not make any other changes in the circuit. And below this if else statement I again try to reconnect to get the latest updates from the ubidots website because I might have made some changes there in the meanwhile. And after this in the second image below which is also the continuation in the same function for better understanding I read the values of temperature and humidity to write them in the serial monitor for better understanding which is actually redundant.

```
void loop() {

  // Controlling Buzzer with LDR Sensor
  if (analogRead(LDR) < 200) {
    Serial.println("Light is falling on LDR. Buzzer will be turned on.");
    digitalWrite(BUZZER, HIGH);
    if(!ismailsent)
    {
      String maildetail = "Seems like your Home is under some security threat, an unknown source of unrecognised light has disturbed the LDR sensor.\n";
      maildetail += "I think you must check what happened!\n\n";
      maildetail += "The current Room temperature is " + String(dht.readTemperature()) + " Degree Celsius" + "\n";
      maildetail += "The current Room Relative Humidity is " + String(dht.readHumidity()) + " %\n";
      client.sendMail("mail.smtp2go.com",2525, "QXJpaGFudGty","YXJpaGFudDEyMw==","nodemcu@gmail.com","arihanteshwar@gmail.com","Security Breach",maildetail);
      ismailsent=1;
    }
  }
  else {
    Serial.println("Light is not falling on LDR");
    digitalWrite(BUZZER, LOW);
    ismailsent=0;
  }

  // Controlling Bulb with Website
  if (!client.connected()) {
    client.reconnect();
    client.ubidotsSubscribe(DEVICE_LABEL, VARIABLE_TO_SUBSCRIBE);
  }
  client.loop();
```

```
  // Reading Temperature and Humidity
  int humidity_data = (int)dht.readHumidity();
  int temperature_data = (int)dht.readTemperature();

  Serial.print("Current Room Temperature: ");
  Serial.println(humidity_data);
  Serial.print("Current Room Humidity: ");
  Serial.println(temperature_data);
  Serial.print("\n\n");

  delay(100);
}
```

PART 3:-

In the part 3 of my project I have made a simple real life traffic light system prototype, which is similar to the real life Traffic light systems we find around us in our day to day life. So using the ubidots account similar to the one in the above code I am remotely able to control my traffic light system

from any part of the world, and in this case which can be considered as a control room.

And also most importantly in this project of mine I have used a blue led bulb instead of a yellow led bulb which we find in the actual traffic lights because the yellow led bulbs use a special kind of white LED bulb which requires more than 3V (i.e the Arduino capacity) which is not possible, whereas a blue LED bullb takes less than 3V which can be supplied by the Arduino.

PART 3 CODE:-

INITIALISATION AND MACROS:- In the first picture below of the traffic lights code I have all the macros defined which enables my nodemcu to connect to the internet and also for the nodemcu to connect to my ubidots account from the place where I would be monitoring my traffic light system.

AUXILIARY FUNCTIONS:- As it can also be seen that just like in the code of part 1 and part 2, even here I have the callback function which enables the nodemcu to check with the ubidots command if I have passed any new command and based on the signal it receives it woul change the bool variable which I have defined below. If the signal is ‘1’ the bool variable is set to true, so that it turns on the traffic light system, else if it is ‘0’ it turns off the signals if they are currently on.

```
#include "UbidotsESPMQTT.h"
#include <stdio.h>
#define WIFISSID "Arihant KR"                                   // Put your WifiSSID here
#define PASSWORD "333333333333"                              // Put your wifi password here
#define TOKEN "BBFF-nPzQ3r0ckEHHb3v3IJpcfkQTXvoFiK"    // Ubidots Token
#define VARIABLE_TO_SUBSCRIBE "home_bulb"
#define DEVICE_LABEL "esp8266"
Ubidots client(TOKEN);
bool ifon=0;
/****************************************
   Auxiliar Functions
 ****************************************/
void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  if ((char)payload[0] == '0')
  {
    Serial.println("Since, Output is 0, I'm going to turn off LED");
    ifon=0;
  }
  else if ((char)payload[0] == '1')
  {
    Serial.println("Since, Output is 1, I'm going to turn on LED");
    ifon=1;
  }
  else {
    Serial.print(" Ubidots Value is ");
    Serial.println((char)payload[0]);
  }
}
```

SETUP FUNCTION:- In the setup function here I have setup all the digital pins to the output mode as it is required in the circuit and its following code enables the nodemcu's connection with the ubidots account using the details I have already provided above.

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  Serial.begin(115200);
  pinMode(D0, OUTPUT);
  pinMode(D1, OUTPUT);
  pinMode(D2, OUTPUT);
  pinMode(D3, OUTPUT);
  pinMode(D4, OUTPUT);
  pinMode(D5, OUTPUT);
  pinMode(D6, OUTPUT);
  pinMode(D7, OUTPUT);
  client.ubidotsSetBroker("industrial.api.ubidots.com");
  client.setDebug(true);
  client.wifiConnection(WIFISSID, PASSWORD);
  client.begin(callback);
  client.ubidotsSubscribe(DEVICE_LABEL, VARIABLE_TO_SUBSCRIBE);
}
```

LOGIC:- The following image contains my final logic for the traffic light system. The logic is completely written in the loop function. The logic is very simple if the signal received by the Arduino is ‘1’ it sets on the entire systems. I have used 8 pins to connect the green led in a particular direction and blue led in the next direction (4 directions and 4 pins) as they work together, as the four red LEDs are independent I have connected them in four independent pins which finally makes it 8 total pins used in the project and later what I did

was in a particular direction if the green led or the blue led is on I turn of the red LED at that duration else I turn it on.

```
void loop() {
  if (!client.connected()) {
    client.reconnect();
    client.ubidotsSubscribe(DEVICE_LABEL, VARIABLE_TO_SUBSCRIBE);
  }
  client.loop();
  if(ifon)
  {
    digitalWrite(D6, HIGH);
  digitalWrite(D7, HIGH);
  digitalWrite(D3, LOW);
  digitalWrite(D0, HIGH);// turn the LED on (HIGH is the voltage level)
  digitalWrite(D5, LOW);
  delay(5000);                       // wait for a second
  digitalWrite(D1, HIGH);// turn the LED on (HIGH is the voltage level)
  digitalWrite(D0, LOW);
  digitalWrite(D6, LOW);
  digitalWrite(D4, HIGH);
  delay(5000);
  digitalWrite(D2, HIGH);// turn the LED on (HIGH is the voltage level)
  digitalWrite(D7, LOW);
  digitalWrite(D1, LOW);
  digitalWrite(D5, HIGH);
  delay(5000);
  digitalWrite(D4, LOW);
  digitalWrite(D3, HIGH);    // turn the LED on (HIGH is the voltage level)
  digitalWrite(D2, LOW);
  digitalWrite(D6, HIGH);
  delay(5000);
  }
  else
  {
    digitalWrite(D0,LOW);
    digitalWrite(D1,LOW);
    digitalWrite(D2,LOW);
    digitalWrite(D3,LOW);
    digitalWrite(D4,LOW);
    digitalWrite(D5,LOW);
    digitalWrite(D6,LOW);
    digitalWrite(D7,LOW);
  }
  loop();

}
```

THANK YOU

THIS IS THE END OF MY REPORT