A REPORT

ON

MQTT PROTOCOL AND IOT HOME AUTOMATION

BY

Name(s) of the Student(s)	**ID No.(s)**
Shaurya Vijayvargiya	2018A8PS0079H

AT

(Nihon Communication Solutions Pvt. Ltd. - IoT Bangalore)

A Practice School-I Station of

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

(June, 2020)

A REPORT

ON

MQTT PROTOCOL AND IOT HOME AUTOMATION

BY

Name(s) of the Student(s)	**ID No.(s)**	**Discipline(s)**
Shaurya Vijayvargiya	2018A8PS0079H	A8-Electronics and Instrumentation

Prepared in complete fulfilment of the

Practice School-I Course Nos.

BITS C221/BITS C231/BITS C241

AT

(Nihon Communication Solutions Pvt. Ltd. - IoT Bangalore)

A Practice School-I Station of

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

(June, 2020)

Acknowledgement

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

I am highly indebted to **Nihon Communication Solutions Pvt. Ltd.** for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.
I would like to express my gratitude towards my parents & faculty of BITS PILANI, **G Sai Sesha Chalapathi Sir** for their kind co-operation and encouragement which helped me in completion of this project.

I would like to express my special gratitude and thanks to industry mentors **Sanketh** and **Abhaya** sir for giving me such attention and time.

My thanks and appreciations also go to my colleagues in developing the project and people who have willingly helped me out with their abilities.

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

(RAJASTHAN)

Practice School Division

Station: Nihon Communication Solutions Pvt. Ltd. - IoT Bangalore

Duration: 6 Weeks (Till 27-06-2020) **Date of Start: 18-05-2020**

Date of Submission: 25-06-2020

Title of Project: IoT Home Automation Solutions

ID No.: 2018A8PS0079H

Name(s): Shaurya Vijayvargiya

Discipline(s) of the student(s): A8-Electronics and Instrumentation

Name of Expert(s): Sanketh Gopal

Designation(s) of Expert(s): Application Engineer

Name(s) of PS Faculty: G Sai Sesha Chalapathi

Key Words: MQTT, Automation, IoT, HC-SR04, LDR, API, Dashboard

Project Areas: Internet of Things

Abstract: MQTT stands for **Message Queueing Telemetry Transport** and is an IoT protocol. It uses publish subscribe model and has various security issues. It uses QoS for ensuring message delivery. The main project is to make a dashboard for 3 ideas which are tube light controlled using light sensors, motion detection for toggling home appliances such as fans and lights, and intruder/burglar detection using security cam. A backend made using Django has been used and the frontend has been made using ReactJS.

Signature of Student(s) **Signature of PS Faculty**

Date: 25-06-2020 **Date: 25-06-2020**

Table of Contents

Nihon Communications Solutions Pvt. Ltd.

NCS was incorporated in 2006 and focuses on IT solutions. They also provide wide range of services including development, expert and consulting and support services. They provide students with IoT kits for fast and easy development and efficient learning experiences. They also have various products such as –

1. **Monosek :** High end Network Processor based Network Packet Processing and Network Session Analysis System
2. **QualNet :** Ultra high-fidelity network evaluation software that predicts performance of wired, wireless and mixed-platform networks
3. **Exata :** Wireless emulator that lets you evaluate on-the-move communication networks by creating a digital network replica that interfaces with networks in real time, using real applications
4. **Zolertia :** Modular hardware and software platform for wireless sensor networks which can provide solutions to engineers, universities, R&D centres and companies to design create their own IOT products

They provide many services in various fields such as – embedded system design, high density FPGA design, deploying security systems, simulation of networks, etc.

They have many clients, majority of which are education institutes which use their QualNet product.

Introduction

Project Title – Home Automation Solutions

Project Description – The project is about putting together 3 home automation solutions into a dashboard and then letting the user control the devices and also control them using sensors. Main aim is to save electricity and provide convenience to the user.

The working principle protocol behind this is **MQTT** which stands for Message Queueing Telemetry Transport. The main reason behind using this is what makes it a very convenient protocol to be used for IoT projects. It has a publish-subscribe model which is good for sending live data from sensors and immediately receiving it by subscribing to a topic. MQTT is light weight and is secure by using port 8883 for secure-mqtt.

The report is organized into multiple sections as follows:

a. **Detailed Description about each solution**
b. **Explanation of sensors and their working code**
c. **Working of the security cam**
d. **MQTT Message Receiver**
e. **Controlling LEDs using the NodeMCU board**
f. **The Backend API**
g. **The Dashboard**
h. **AWS and its components**

IoT Home Automation Project

- **Description of Solutions:**

a. **Motion Detection** : Appliances such as lights and fans are often left unchecked and keep on running in our houses often wasting electricity. Sometimes the user is too lazy to go from one room to another to turn on/off the appliance. Hence I have used ultrasonic sensors to detect user motion and when a user exits the room, the lights and fans are automatically turned off and turned on when the user enters the room. The user can also switch the light on and off using a toggle provided in the UI.

b. **Light Detection** : Often when the room turns dark the user forgets to turn on the light and often strains his/her eyes. Using a light sensor to detect the light intensity in the room, the lights are automatically toggled depending on the light intensity in the room. A threshold is set according to which if the light intensity is lower than or higher than, the lights are toggled accordingly. This saves the user's eyes from strain and also saves electricity at times when the user just leaves the light on even when there is enough natural light in the room.

c. **Security Cam** : Home security is a big issue in many houses. By using many deep learning models, now a days it is very easy to detect an intruder using a simple camera and some python code. Using a library called **OpenCV**, a security cam can be created. It will detect each frame for any human face and will crop that part of the photo and save it locally on the device the code is running on (raspberry pi in this case) and alert the user in the dashboard changing the status of intruder detected to true. User can very easily access the images of the intruder using scp file protocol to view images on any device connected on the same wifi as the pi.

- **Sensors explained :** Two sensors have been used in this project. An HC-SR04 ultrasonic sensor and an LDR light dependent resistor. Both have there uses and perform very efficiently. Here is a detailed explanation of both –

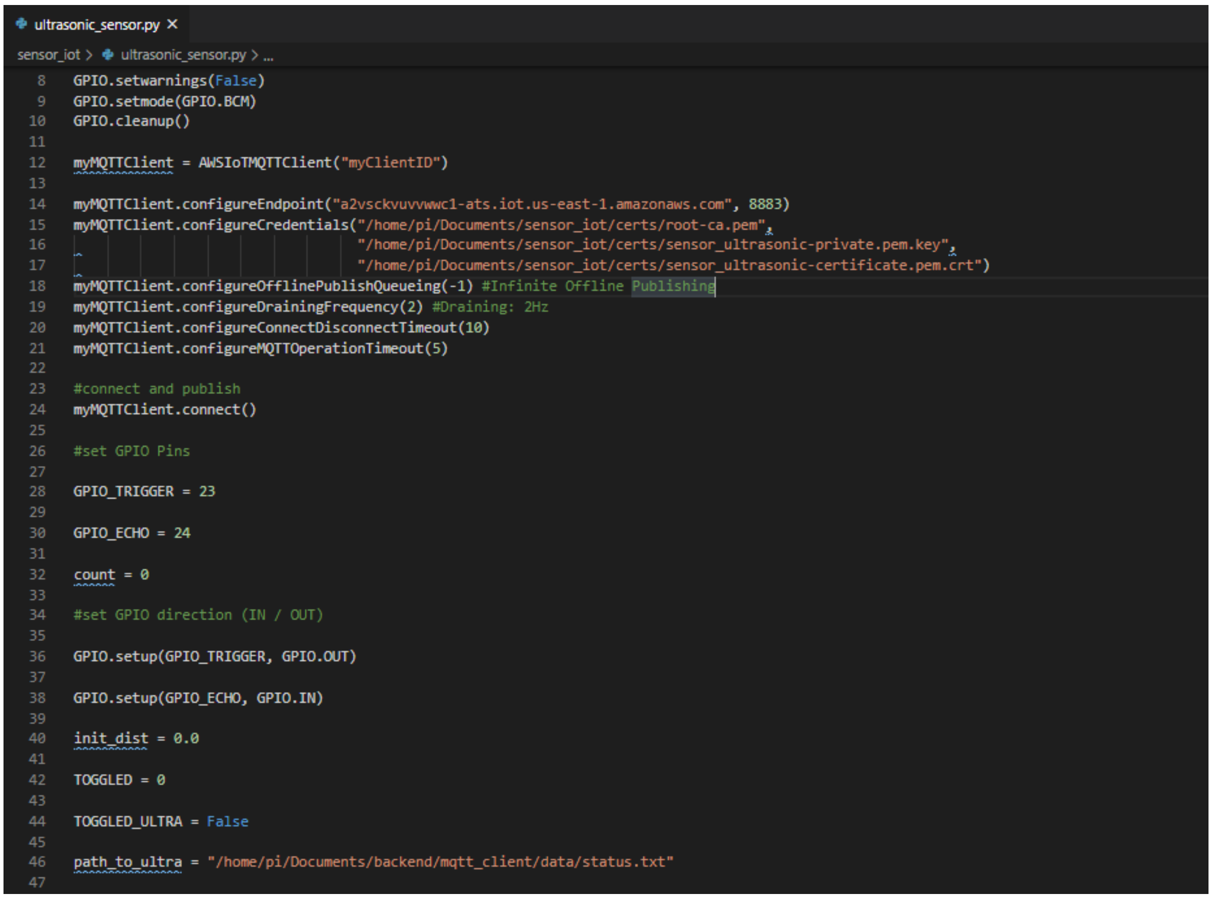
a. **HC-SR04 Ultrasonic sensor:** This sensor has 4 pins Vcc, Trig, Echo and GND pin.

Each pin has its own function.

- **Vcc** : Powers the sensor, typically +5V
- **Trig** : This is the trigger pin which usually takes the input. It is kept high for 10µs to initialize measurement by sending 8 short bursts of ultrasonic waves at 40Khz.
- **Echo** : This is the output pin. This pin goes high for a period of time which will be equal to the time taken for the ultrasonic wave to return back to the sensor.
- **GND** : This is the ground pin to connect system to ground 0V.

It's main working formula is distance = speed x time. Speed is taken as speed of sound which is 343 m/s and time is calculated using echo and trigger pin.

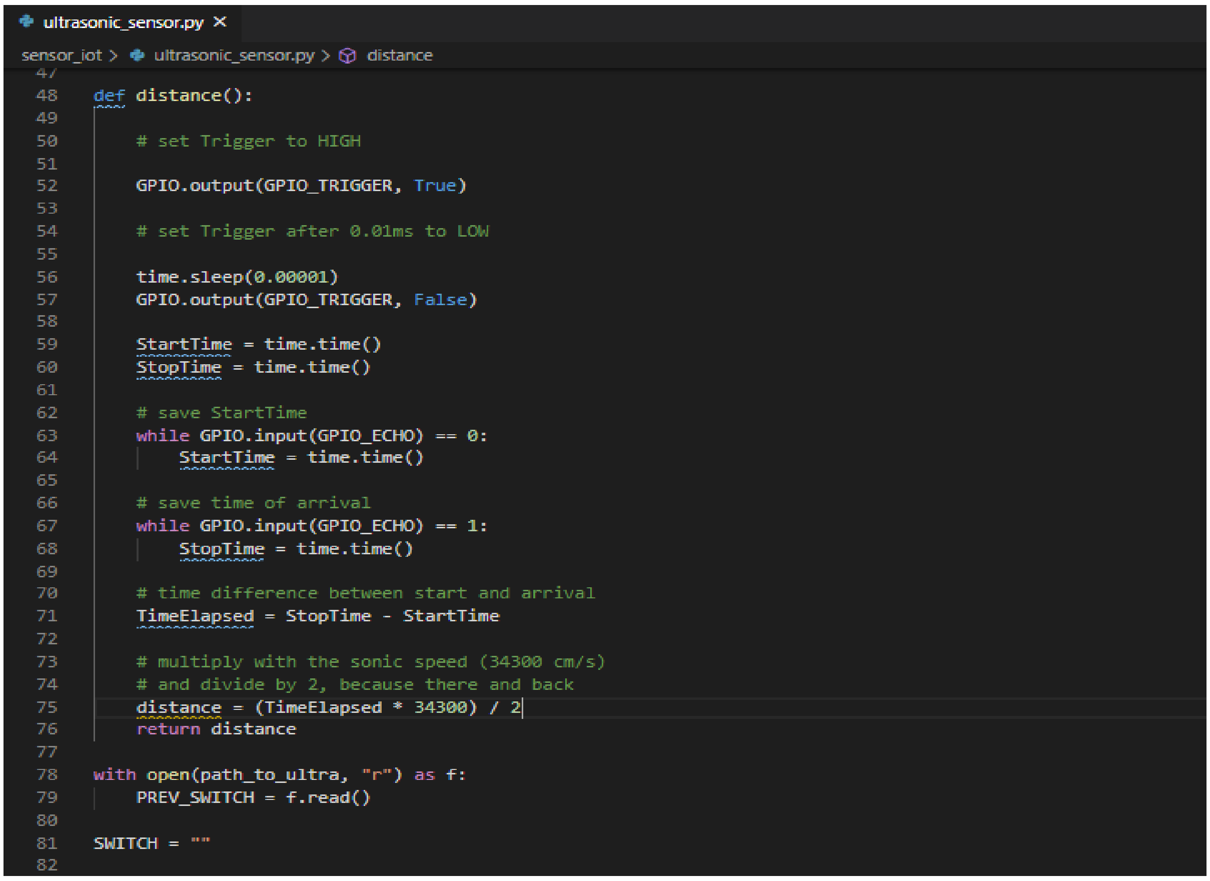
CODE:



ultrasonic_sensor.py ×

sensor_iot > ultrasonic_sensor.py > ...

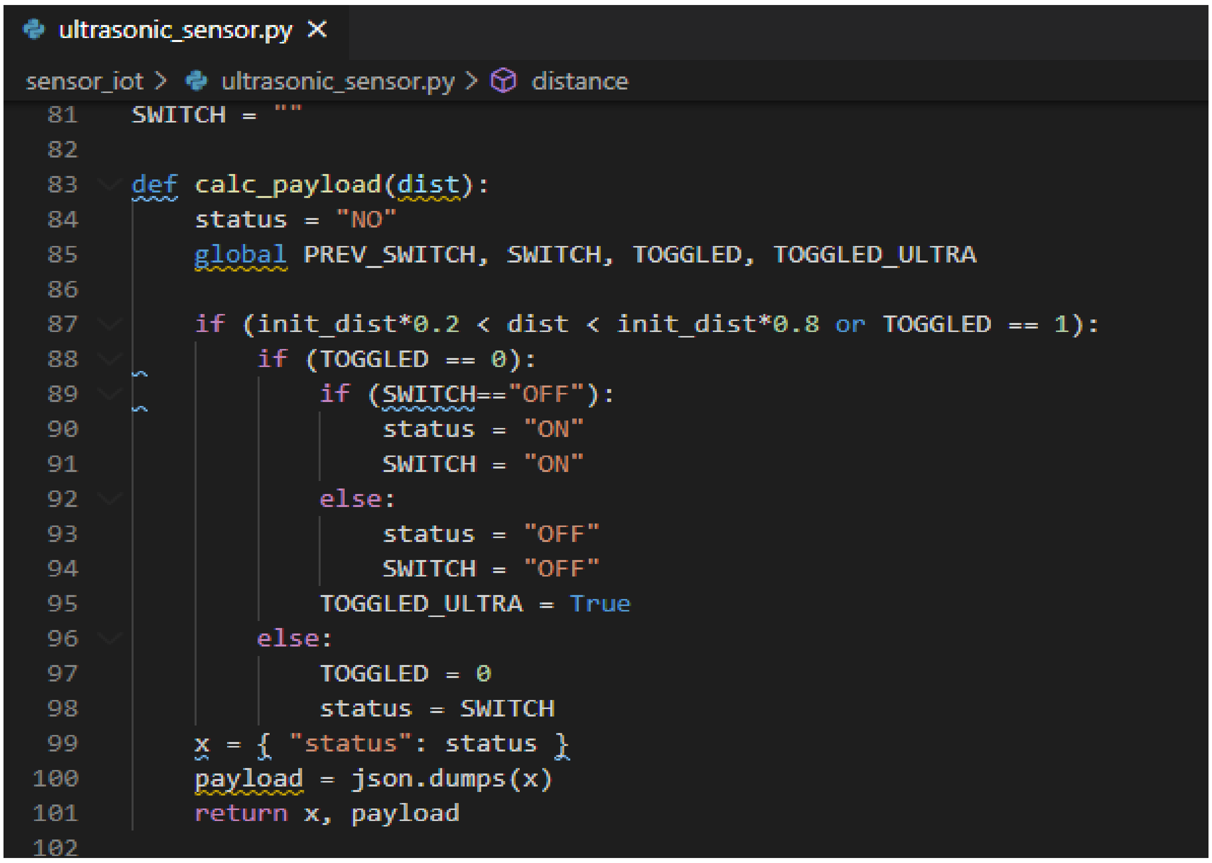
```python
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.cleanup()

myMQTTClient = AWSIoTMQTTClient("myClientID")

myMQTTClient.configureEndpoint("a2vsckvuvvwwc1-ats.iot.us-east-1.amazonaws.com", 8883)
myMQTTClient.configureCredentials("/home/pi/Documents/sensor_iot/certs/root-ca.pem",
                                  "/home/pi/Documents/sensor_iot/certs/sensor_ultrasonic-private.pem.key",
                                  "/home/pi/Documents/sensor_iot/certs/sensor_ultrasonic-certificate.pem.crt")
myMQTTClient.configureOfflinePublishQueueing(-1) #Infinite Offline Publishing
myMQTTClient.configureDrainingFrequency(2) #Draining: 2Hz
myMQTTClient.configureConnectDisconnectTimeout(10)
myMQTTClient.configureMQTTOperationTimeout(5)

#connect and publish
myMQTTClient.connect()

#set GPIO Pins

GPIO_TRIGGER = 23

GPIO_ECHO = 24

count = 0

#set GPIO direction (IN / OUT)

GPIO.setup(GPIO_TRIGGER, GPIO.OUT)

GPIO.setup(GPIO_ECHO, GPIO.IN)

init_dist = 0.0

TOGGLED = 0

TOGGLED_ULTRA = False

path_to_ultra = "/home/pi/Documents/backend/mqtt_client/data/status.txt"

```



ultrasonic_sensor.py ×

sensor_iot > ultrasonic_sensor.py > distance

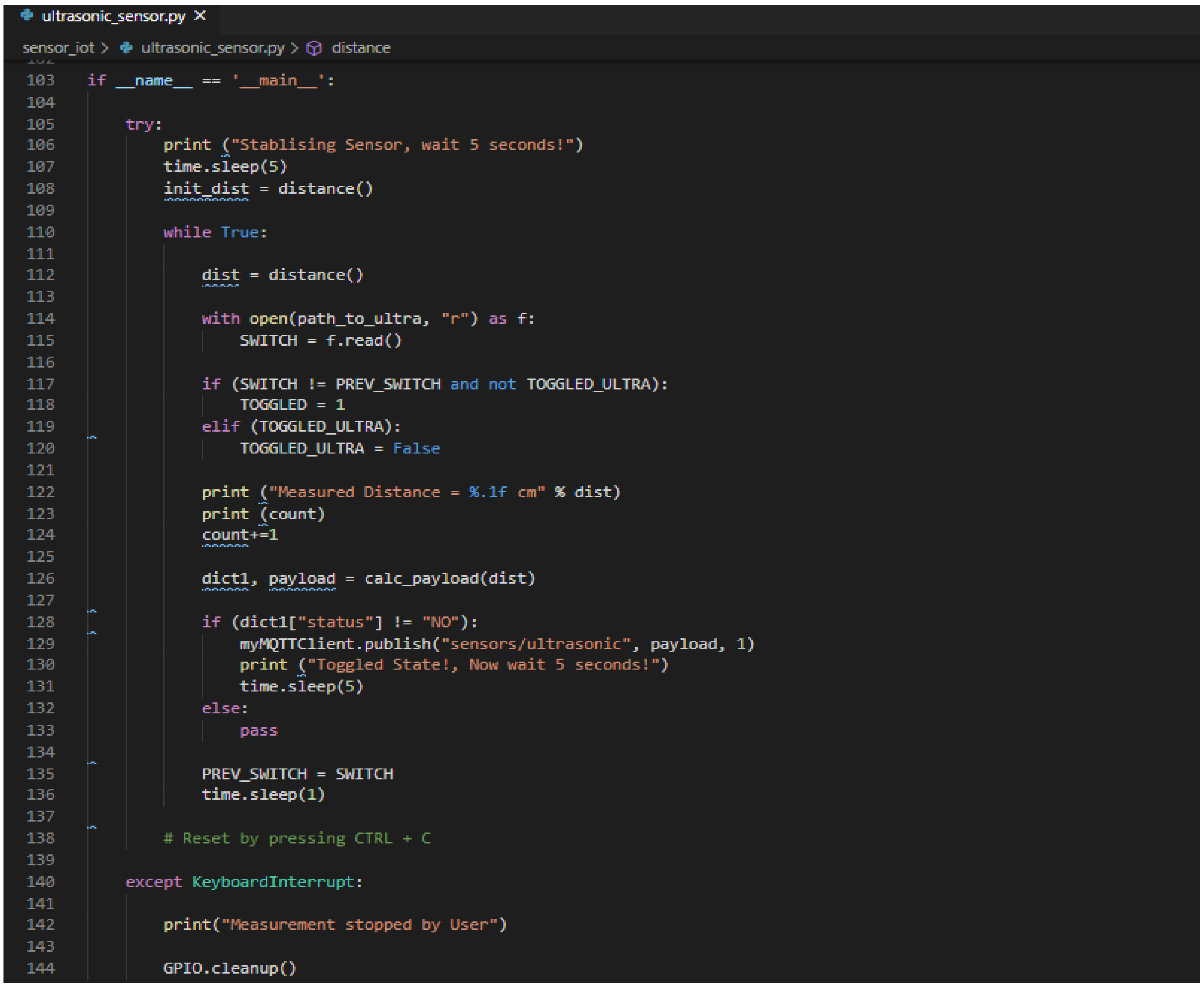
```python

def distance():

    # set Trigger to HIGH

    GPIO.output(GPIO_TRIGGER, True)

    # set Trigger after 0.01ms to LOW

    time.sleep(0.00001)
    GPIO.output(GPIO_TRIGGER, False)

    StartTime = time.time()
    StopTime = time.time()

    # save StartTime
    while GPIO.input(GPIO_ECHO) == 0:
        StartTime = time.time()

    # save time of arrival
    while GPIO.input(GPIO_ECHO) == 1:
        StopTime = time.time()

    # time difference between start and arrival
    TimeElapsed = StopTime - StartTime

    # multiply with the sonic speed (34300 cm/s)
    # and divide by 2, because there and back
    distance = (TimeElapsed * 34300) / 2
    return distance

with open(path_to_ultra, "r") as f:
    PREV_SWITCH = f.read()

SWITCH = ""

```



ultrasonic_sensor.py

sensor_iot > ultrasonic_sensor.py > distance

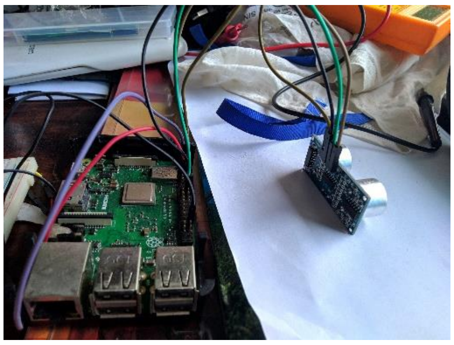
```python
SWITCH = ""

def calc_payload(dist):
    status = "NO"
    global PREV_SWITCH, SWITCH, TOGGLED, TOGGLED_ULTRA

    if (init_dist*0.2 < dist < init_dist*0.8 or TOGGLED == 1):
        if (TOGGLED == 0):
            if (SWITCH=="OFF"):
                status = "ON"
                SWITCH = "ON"
            else:
                status = "OFF"
                SWITCH = "OFF"
            TOGGLED_ULTRA = True
        else:
            TOGGLED = 0
            status = SWITCH
    x = { "status": status }
    payload = json.dumps(x)
    return x, payload

```

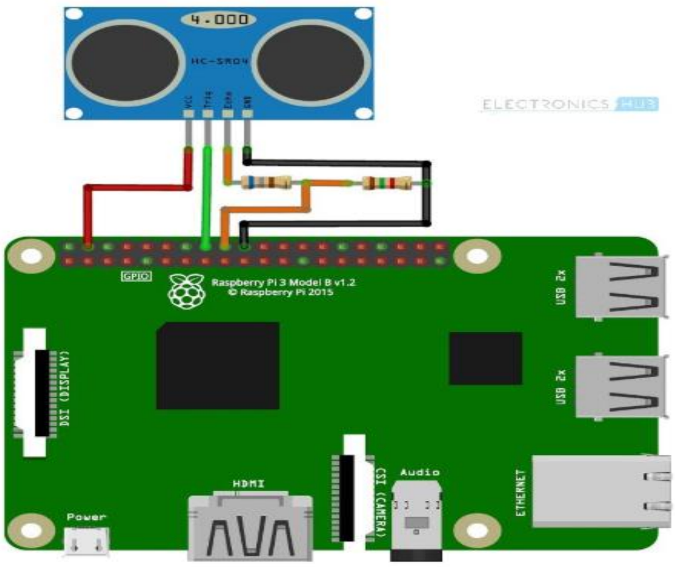


ultrasonic_sensor.py

sensor_iot > ultrasonic_sensor.py > distance

```python
if __name__ == '__main__':

    try:
        print ("Stablising Sensor, wait 5 seconds!")
        time.sleep(5)
        init_dist = distance()

        while True:

            dist = distance()

            with open(path_to_ultra, "r") as f:
                SWITCH = f.read()

            if (SWITCH != PREV_SWITCH and not TOGGLED_ULTRA):
                TOGGLED = 1
            elif (TOGGLED_ULTRA):
                TOGGLED_ULTRA = False

            print ("Measured Distance = %.1f cm" % dist)
            print (count)
            count+=1

            dict1, payload = calc_payload(dist)

            if (dict1["status"] != "NO"):
                myMQTTClient.publish("sensors/ultrasonic", payload, 1)
                print ("Toggled State!, Now wait 5 seconds!")
                time.sleep(5)
            else:
                pass

            PREV_SWITCH = SWITCH
            time.sleep(1)

        # Reset by pressing CTRL + C

    except KeyboardInterrupt:

        print("Measurement stopped by User")

        GPIO.cleanup()
```





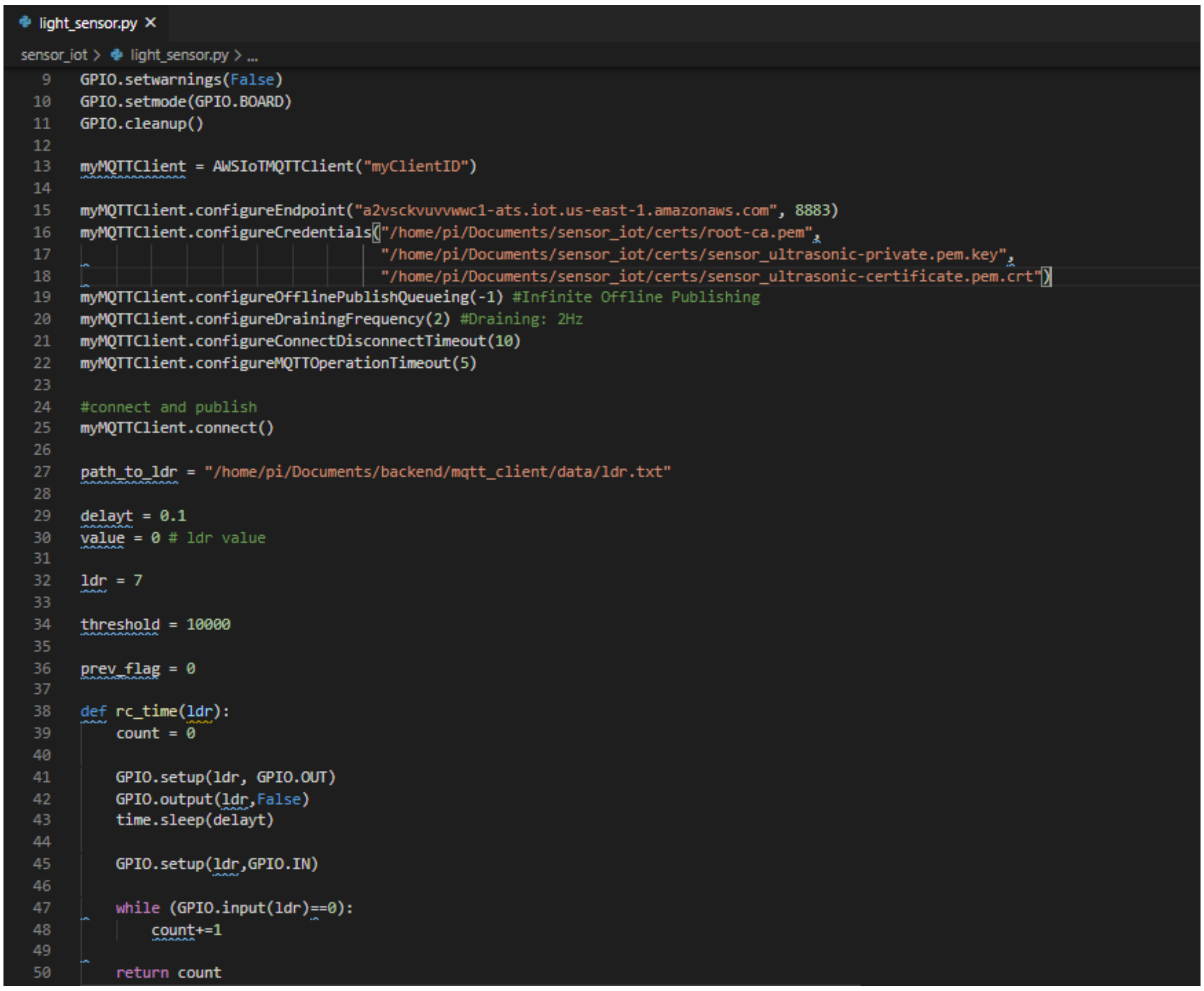
Here the sensor is connected to GPIO pins 23 and 24 of the pi. We calculate the distance using the trigger and echo pin as mentioned above. Then accordingly an MQTT message is published under the topic ‘**sensors/ultrasonic’** to AWS IoT client.

b. LDR Light Dependent Resistor:

An LDR or a photoresistor is a device that is made up of high resistance semiconductor material.

The working principle of an LDR is photoconductivity, that is nothing but an optical phenomenon. When the light is absorbed by the material then the conductivity of the material reduces. When the light falls on the LDR, then the electrons in the valence band of the material are eager to the conduction band. But, **the photons in the incident light must have energy superior than the bandgap of the material to make the electrons jump from one band to another band** (valance to conduction). Hence, when light having ample energy, more electrons are excited to the conduction band which grades in a large number of charge carriers. When the effect of this process and the flow of the current starts flowing more, the resistance of the device decreases. A **12mm** sensor has been used in this case.

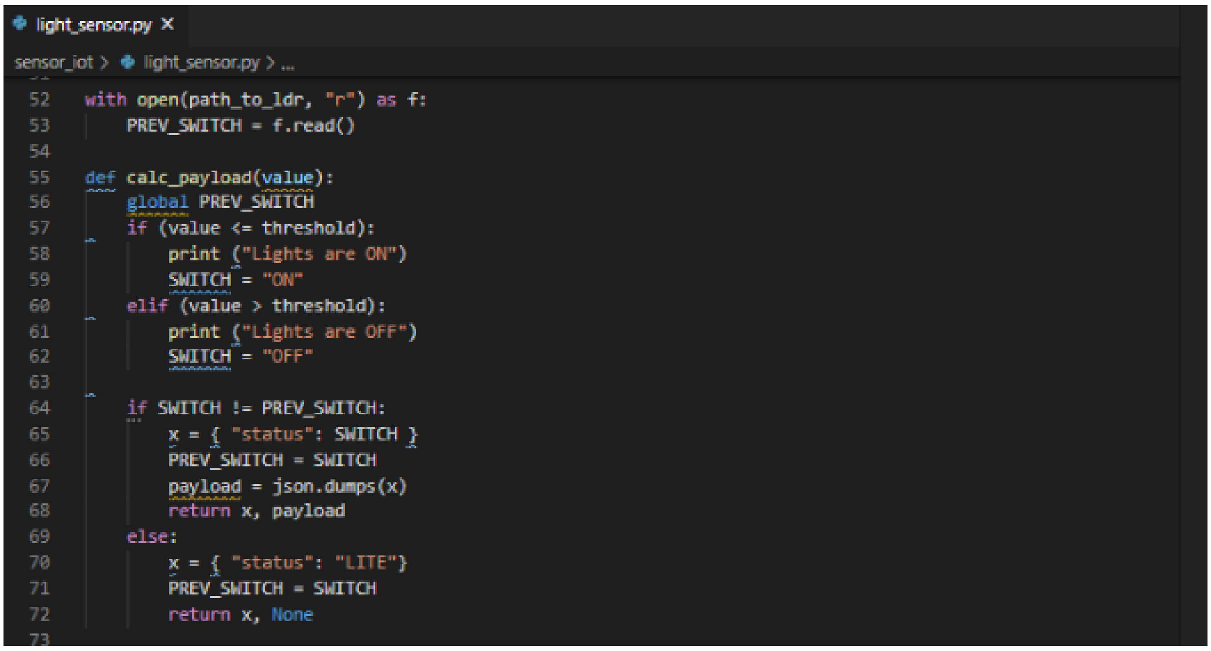
CODE:



light_sensor.py ×

sensor_iot > light_sensor.py > ...

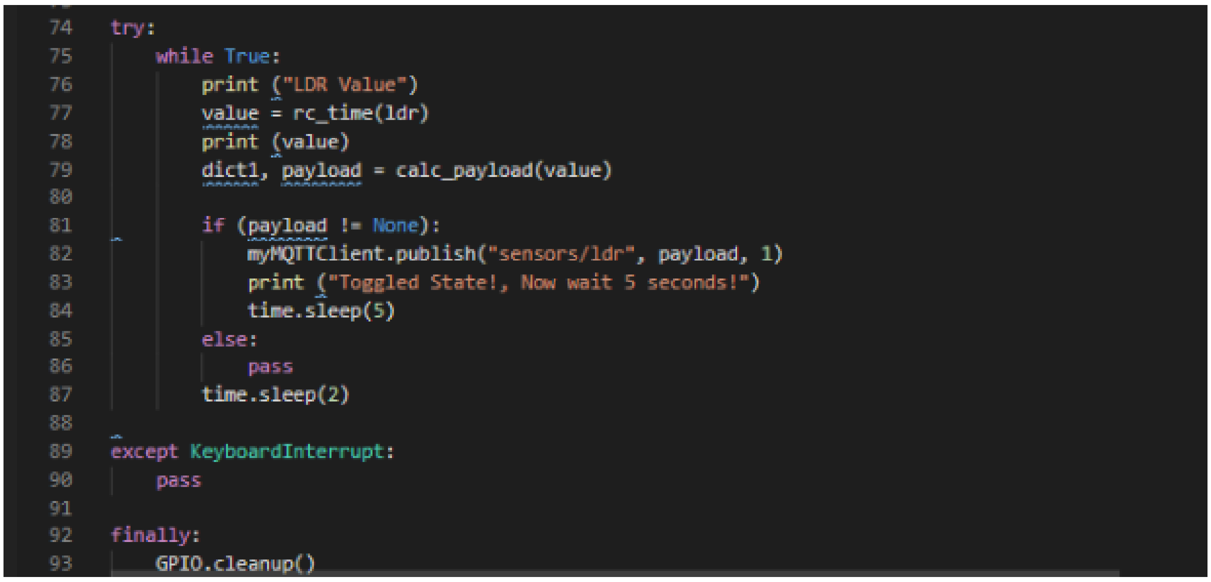
```python
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
GPIO.cleanup()

myMQTTClient = AWSIoTMQTTClient("myClientID")

myMQTTClient.configureEndpoint("a2vsckvuvvwwc1-ats.iot.us-east-1.amazonaws.com", 8883)
myMQTTClient.configureCredentials("/home/pi/Documents/sensor_iot/certs/root-ca.pem",
                                  "/home/pi/Documents/sensor_iot/certs/sensor_ultrasonic-private.pem.key",
                                  "/home/pi/Documents/sensor_iot/certs/sensor_ultrasonic-certificate.pem.crt")
myMQTTClient.configureOfflinePublishQueueing(-1) #Infinite Offline Publishing
myMQTTClient.configureDrainingFrequency(2) #Draining: 2Hz
myMQTTClient.configureConnectDisconnectTimeout(10)
myMQTTClient.configureMQTTOperationTimeout(5)

#connect and publish
myMQTTClient.connect()

path_to_ldr = "/home/pi/Documents/backend/mqtt_client/data/ldr.txt"

delayt = 0.1
value = 0 # ldr value

ldr = 7

threshold = 10000

prev_flag = 0

def rc_time(ldr):
    count = 0

    GPIO.setup(ldr, GPIO.OUT)
    GPIO.output(ldr,False)
    time.sleep(delayt)

    GPIO.setup(ldr,GPIO.IN)

    while (GPIO.input(ldr)==0):
        count+=1

    return count
```



light_sensor.py ×

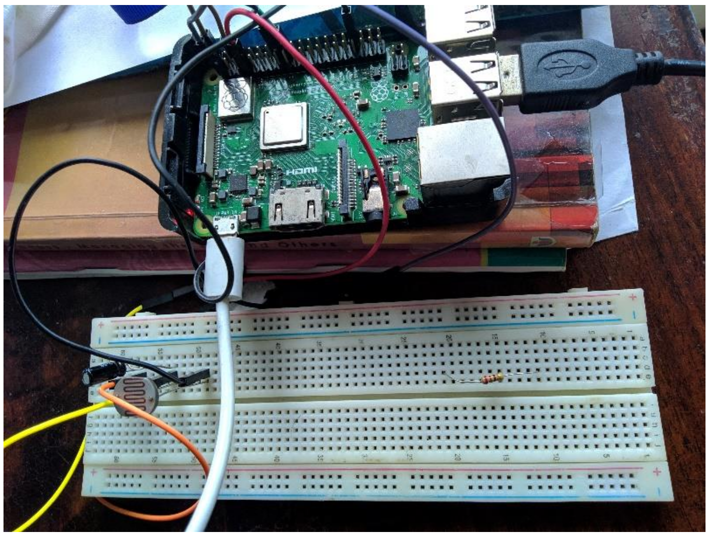
sensor_iot > light_sensor.py > ...

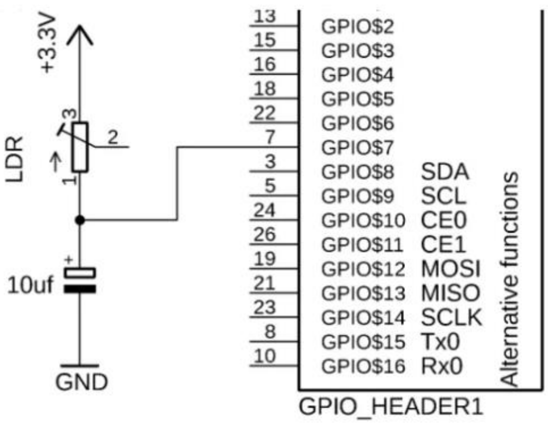
```python
with open(path_to_ldr, "r") as f:
    PREV_SWITCH = f.read()

def calc_payload(value):
    global PREV_SWITCH
    if (value <= threshold):
        print ("Lights are ON")
        SWITCH = "ON"
    elif (value > threshold):
        print ("Lights are OFF")
        SWITCH = "OFF"

    if SWITCH != PREV_SWITCH:
        x = { "status": SWITCH }
        PREV_SWITCH = SWITCH
        payload = json.dumps(x)
        return x, payload
    else:
        x = { "status": "LITE"}
        PREV_SWITCH = SWITCH
        return x, None

```



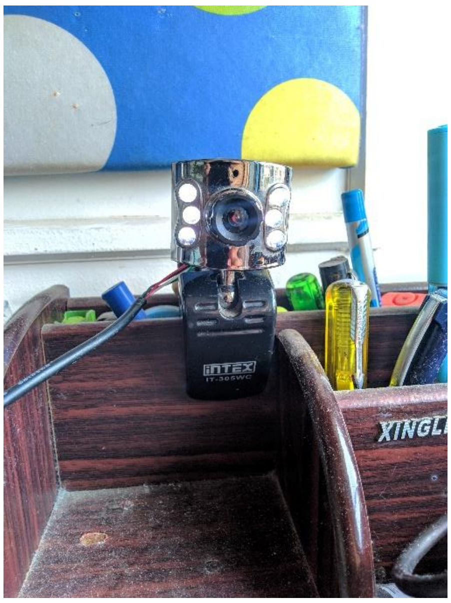
```python
try:
    while True:
        print ("LDR Value")
        value = rc_time(ldr)
        print (value)
        dict1, payload = calc_payload(value)

        if (payload != None):
            myMQTTClient.publish("sensors/ldr", payload, 1)
            print ("Toggled State!, Now wait 5 seconds!")
            time.sleep(5)
        else:
            pass
        time.sleep(2)

except KeyboardInterrupt:
    pass

finally:
    GPIO.cleanup()
```

Here the LDR is connected to pin 7 or GPIO 4 pin (according to Broadcom SOC Channel style numbering) on the board. A threshold value is then set (in this case to 10000). This value decides if the room is bright/dark. If the light reading is **above** the threshold value that means the room is **dark**, whereas if it is **below** the threshold value, then the room is **bright**. Accordingly, when there is a change detected in the brightness of the room, an MQTT message is triggered and sent to the IoT client under the topic **'sensors/ldr'**.





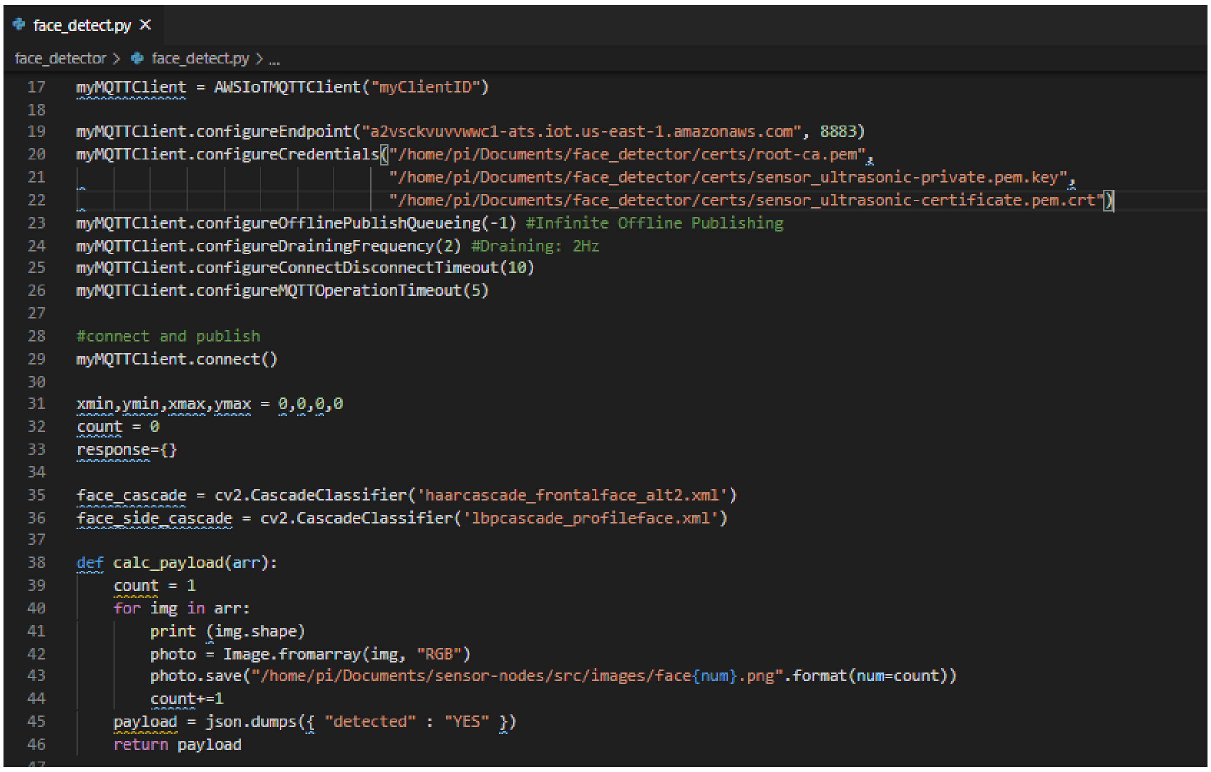
- **Working of the Security Cam:**



For the security cam, a USB camera is used along with the pi. We use OpenCV here to detect intruder using face detection models. These pre-trained models are often used by adding **haar-cascades** to our code. These then give us the precise location of the intruder's face in the frame. Haar cascade work on the principle that in any image there are multiple sets of simple features that combine together to form complex features. We can use these cascade files to detect many features such as eyes, nose, ears, etc.

In our case we use them to identify faces in a frame.

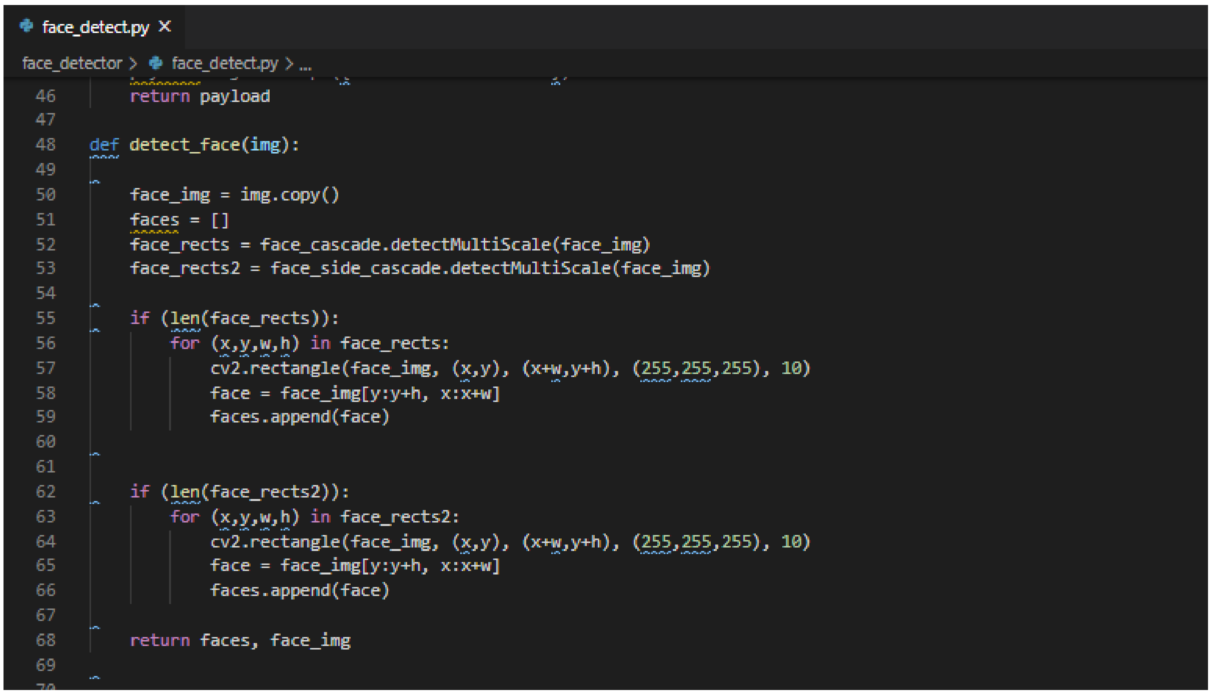
CODE:



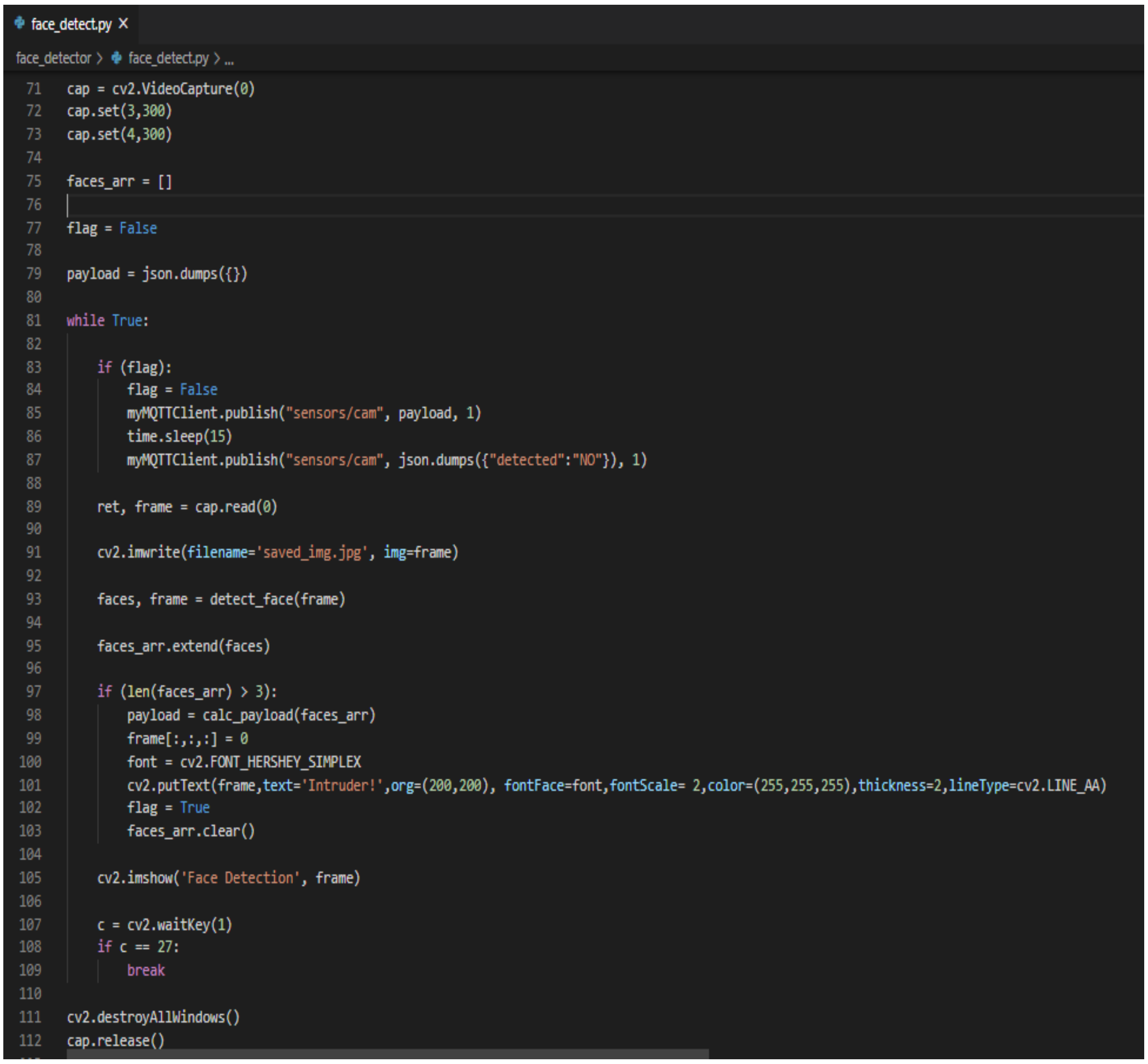
face_detect.py ×

face_detector > face_detect.py > ...

```python
myMQTTClient = AWSIoTMQTTClient("myClientID")

myMQTTClient.configureEndpoint("a2vsckvuvvwwc1-ats.iot.us-east-1.amazonaws.com", 8883)
myMQTTClient.configureCredentials("/home/pi/Documents/face_detector/certs/root-ca.pem",
                                  "/home/pi/Documents/face_detector/certs/sensor_ultrasonic-private.pem.key",
                                  "/home/pi/Documents/face_detector/certs/sensor_ultrasonic-certificate.pem.crt")
myMQTTClient.configureOfflinePublishQueueing(-1) #Infinite Offline Publishing
myMQTTClient.configureDrainingFrequency(2) #Draining: 2Hz
myMQTTClient.configureConnectDisconnectTimeout(10)
myMQTTClient.configureMQTTOperationTimeout(5)

#connect and publish
myMQTTClient.connect()

xmin,ymin,xmax,ymax = 0,0,0,0
count = 0
response={}

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_alt2.xml')
face_side_cascade = cv2.CascadeClassifier('lbpcascade_profileface.xml')

def calc_payload(arr):
    count = 1
    for img in arr:
        print (img.shape)
        photo = Image.fromarray(img, "RGB")
        photo.save("/home/pi/Documents/sensor-nodes/src/images/face{num}.png".format(num=count))
        count+=1
    payload = json.dumps({ "detected" : "YES" })
    return payload
```

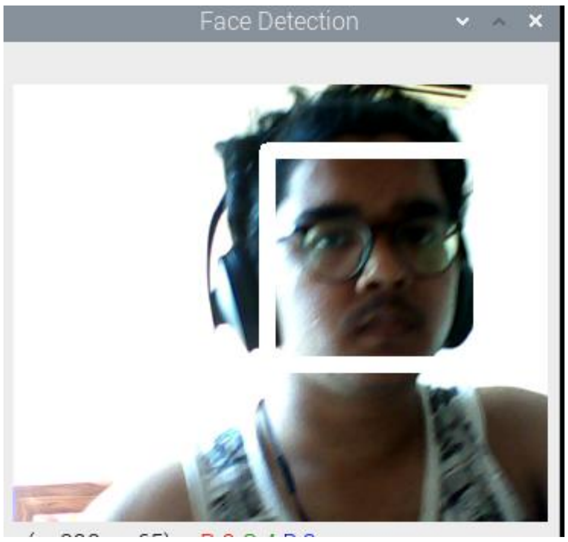


```python
    return payload

def detect_face(img):

    face_img = img.copy()
    faces = []
    face_rects = face_cascade.detectMultiScale(face_img)
    face_rects2 = face_side_cascade.detectMultiScale(face_img)

    if (len(face_rects)):
        for (x,y,w,h) in face_rects:
            cv2.rectangle(face_img, (x,y), (x+w,y+h), (255,255,255), 10)
            face = face_img[y:y+h, x:x+w]
            faces.append(face)


    if (len(face_rects2)):
        for (x,y,w,h) in face_rects2:
            cv2.rectangle(face_img, (x,y), (x+w,y+h), (255,255,255), 10)
            face = face_img[y:y+h, x:x+w]
            faces.append(face)

    return faces, face_img

```



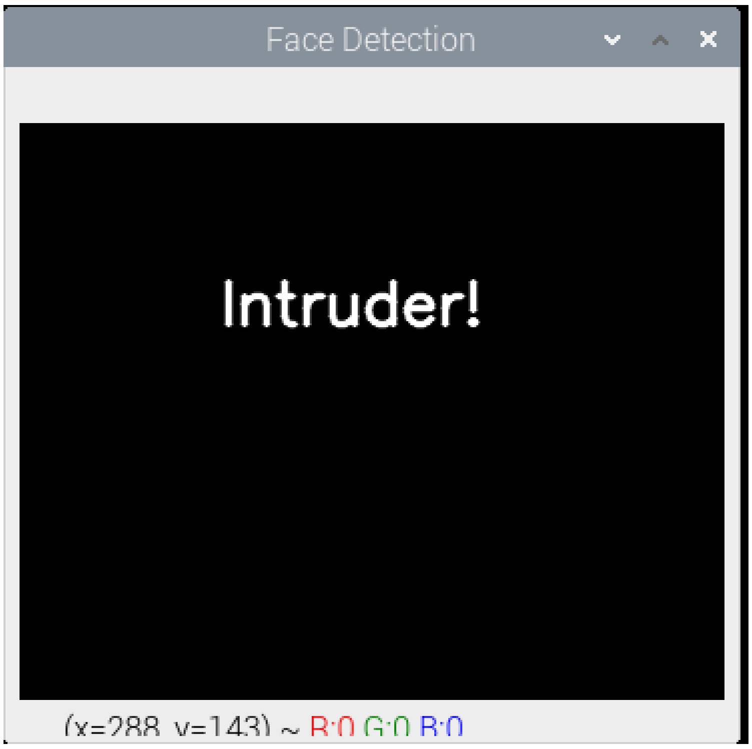
```python
cap = cv2.VideoCapture(0)
cap.set(3,300)
cap.set(4,300)

faces_arr = []

flag = False

payload = json.dumps({})

while True:

    if (flag):
        flag = False
        myMQTTClient.publish("sensors/cam", payload, 1)
        time.sleep(15)
        myMQTTClient.publish("sensors/cam", json.dumps({"detected":"NO"}), 1)

    ret, frame = cap.read(0)

    cv2.imwrite(filename='saved_img.jpg', img=frame)

    faces, frame = detect_face(frame)

    faces_arr.extend(faces)

    if (len(faces_arr) > 3):
        payload = calc_payload(faces_arr)
        frame[:,:,:] = 0
        font = cv2.FONT_HERSHEY_SIMPLEX
        cv2.putText(frame,text='Intruder!',org=(200,200), fontFace=font,fontScale= 2,color=(255,255,255),thickness=2,lineType=cv2.LINE_AA)
        flag = True
        faces_arr.clear()

    cv2.imshow('Face Detection', frame)

    c = cv2.waitKey(1)
    if c == 27:
        break

cv2.destroyAllWindows()
cap.release()
```

Once the code is executed, the security cam begins to analyse each frame. Once a face is detected, a white square frame is drawn around it. Then that part of the image is cropped out and a copy of that is saved, in the local storage of the pi. Since ssh is enabled in the pi, the user can access the images from the pi, if connected to the same wifi network by typing this command –



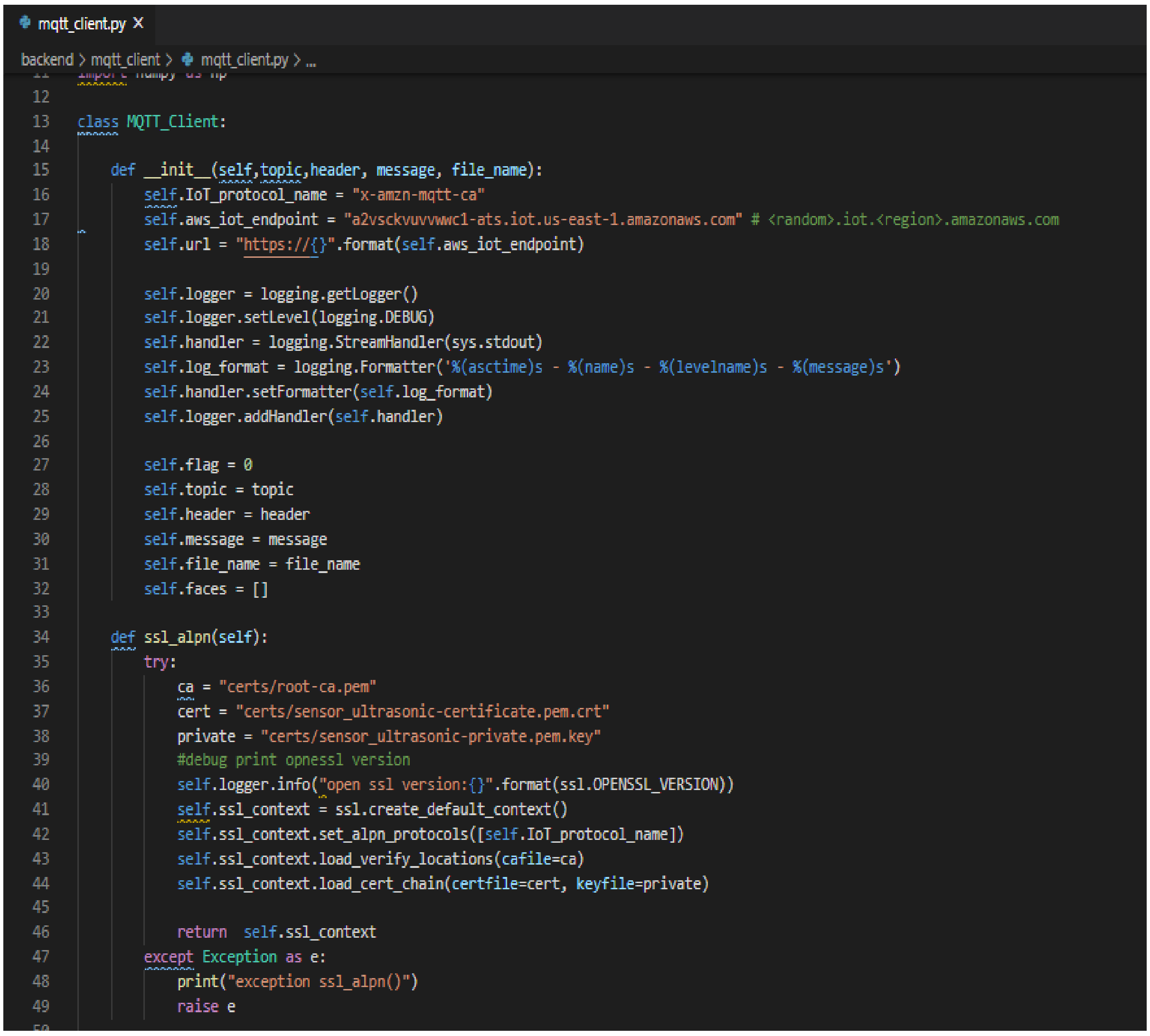
scp pi@<ip-address>:<path_to_images> .

Once a face is detected, the video frame turns into a black frame with the word 'intruder' written on it indicating the same. This stays on there for 15s. At the beginning of those 15s, an MQTT message is sent to the topic **'sensors/cam'.** This message indicates that an intruder is present. After the 15s ends, another message is sent to the IoT client under the same topic, changing the status of intruder detected to NO and the frame returns back to the normal video feed.



- **MQTT Message Receiver :** Once the sensors described above send the messages to the IoT client, there should be a way to confirm if the client has received these messages. Hence we subscribe to these topics and display the received messages in the command line if any. Thus, if any sensor sends a message to the AWS IoT client, that message is received here because of subscribing to the same topics the sensors are sending the messages to.

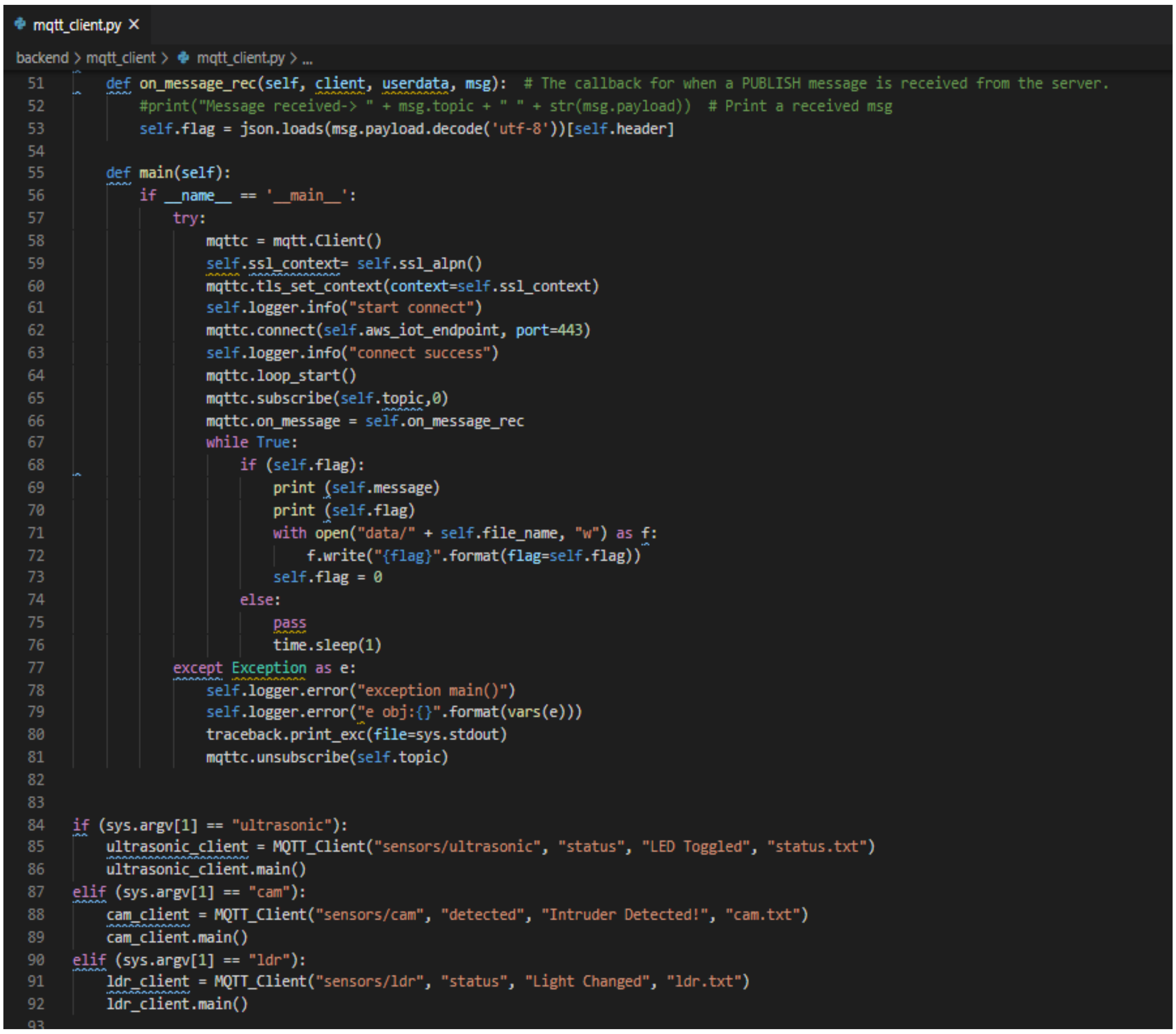
CODE:



mqtt_client.py

backend > mqtt_client > mqtt_client.py > ...

```python
import numpy as np

class MQTT_Client:

    def __init__(self,topic,header, message, file_name):
        self.IoT_protocol_name = "x-amzn-mqtt-ca"
        self.aws_iot_endpoint = "a2vsckvuvvwwc1-ats.iot.us-east-1.amazonaws.com" # <random>.iot.<region>.amazonaws.com
        self.url = "https://{}".format(self.aws_iot_endpoint)

        self.logger = logging.getLogger()
        self.logger.setLevel(logging.DEBUG)
        self.handler = logging.StreamHandler(sys.stdout)
        self.log_format = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
        self.handler.setFormatter(self.log_format)
        self.logger.addHandler(self.handler)

        self.flag = 0
        self.topic = topic
        self.header = header
        self.message = message
        self.file_name = file_name
        self.faces = []

    def ssl_alpn(self):
        try:
            ca = "certs/root-ca.pem"
            cert = "certs/sensor_ultrasonic-certificate.pem.crt"
            private = "certs/sensor_ultrasonic-private.pem.key"
            #debug print opnessl version
            self.logger.info("open ssl version:{}".format(ssl.OPENSSL_VERSION))
            self.ssl_context = ssl.create_default_context()
            self.ssl_context.set_alpn_protocols([self.IoT_protocol_name])
            self.ssl_context.load_verify_locations(cafile=ca)
            self.ssl_context.load_cert_chain(certfile=cert, keyfile=private)

            return  self.ssl_context
        except Exception as e:
            print("exception ssl_alpn()")
            raise e
```



mqtt_client.py ×

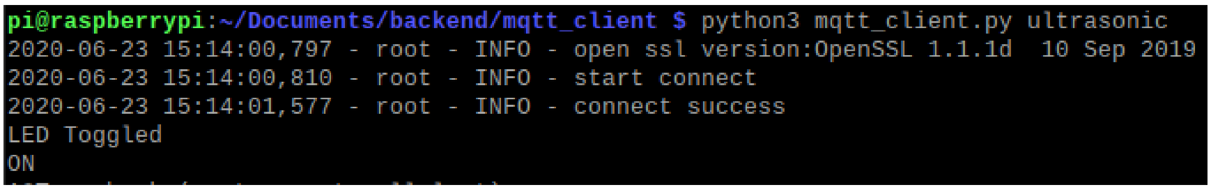
backend > mqtt_client > mqtt_client.py > ...

```python
    def on_message_rec(self, client, userdata, msg):  # The callback for when a PUBLISH message is received from the server.
        #print("Message received-> " + msg.topic + " " + str(msg.payload))  # Print a received msg
        self.flag = json.loads(msg.payload.decode('utf-8'))[self.header]

    def main(self):
        if __name__ == '__main__':
            try:
                mqttc = mqtt.Client()
                self.ssl_context= self.ssl_alpn()
                mqttc.tls_set_context(context=self.ssl_context)
                self.logger.info("start connect")
                mqttc.connect(self.aws_iot_endpoint, port=443)
                self.logger.info("connect success")
                mqttc.loop_start()
                mqttc.subscribe(self.topic,0)
                mqttc.on_message = self.on_message_rec
                while True:
                    if (self.flag):
                        print (self.message)
                        print (self.flag)
                        with open("data/" + self.file_name, "w") as f:
                            f.write("{flag}".format(flag=self.flag))
                        self.flag = 0
                    else:
                        pass
                        time.sleep(1)
            except Exception as e:
                self.logger.error("exception main()")
                self.logger.error("e obj:{}".format(vars(e)))
                traceback.print_exc(file=sys.stdout)
                mqttc.unsubscribe(self.topic)


if (sys.argv[1] == "ultrasonic"):
    ultrasonic_client = MQTT_Client("sensors/ultrasonic", "status", "LED Toggled", "status.txt")
    ultrasonic_client.main()
elif (sys.argv[1] == "cam"):
    cam_client = MQTT_Client("sensors/cam", "detected", "Intruder Detected!", "cam.txt")
    cam_client.main()
elif (sys.argv[1] == "ldr"):
    ldr_client = MQTT_Client("sensors/ldr", "status", "Light Changed", "ldr.txt")
    ldr_client.main()
```

A python class called **MQTT_Client** which takes several arguments

1. topic to subscribe to
2. key in the json data
3. alert message to show when a message is received
4. data file in which status of each sensor is saved

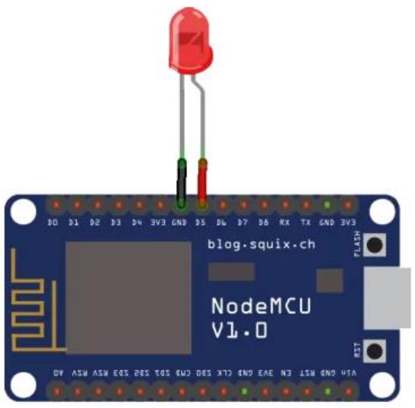
It also accepts command line arguments so that we can tell which sensor to listen to for messages. For ex –



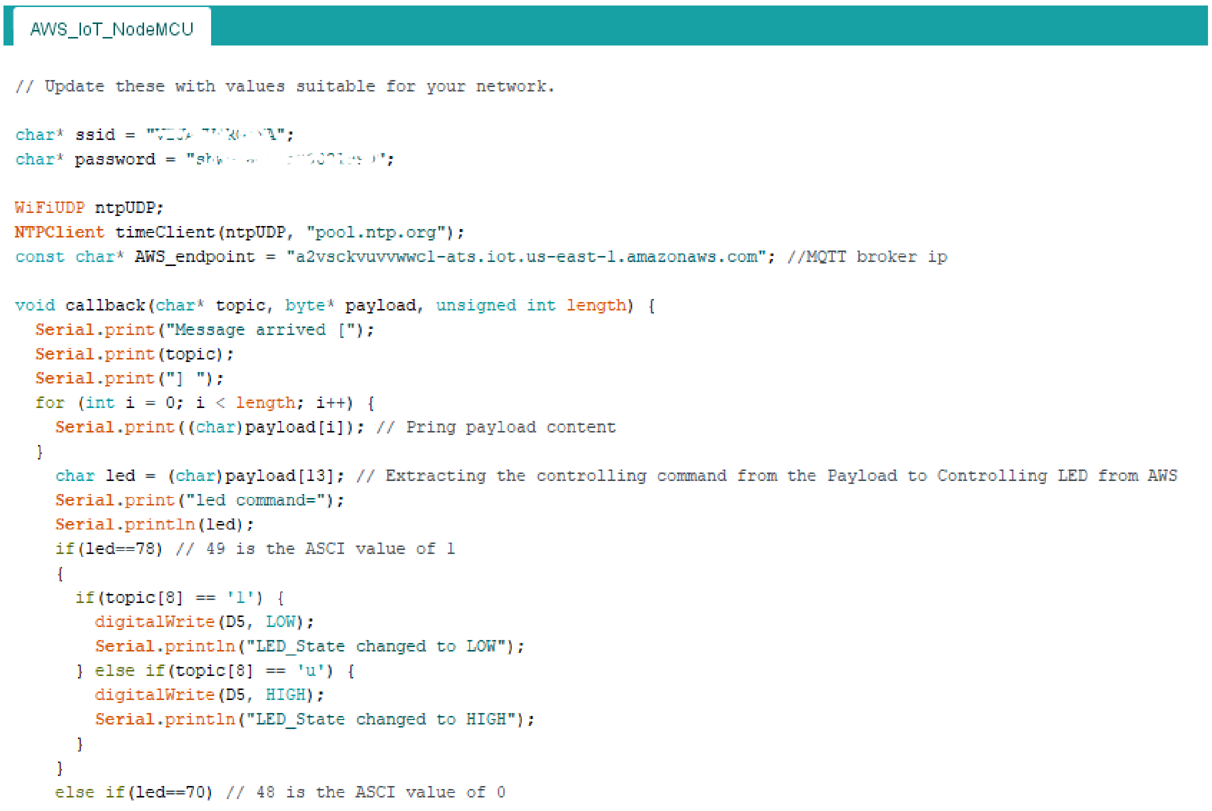
```
pi@raspberrypi:~/Documents/backend/mqtt_client $ python3 mqtt_client.py ultrasonic
2020-06-23 15:14:00,797 - root - INFO - open ssl version:OpenSSL 1.1.1d  10 Sep 2019
2020-06-23 15:14:00,810 - root - INFO - start connect
2020-06-23 15:14:01,577 - root - INFO - connect success
LED Toggled
ON
```

- **Controlling LEDs using NodeMCU Board:**

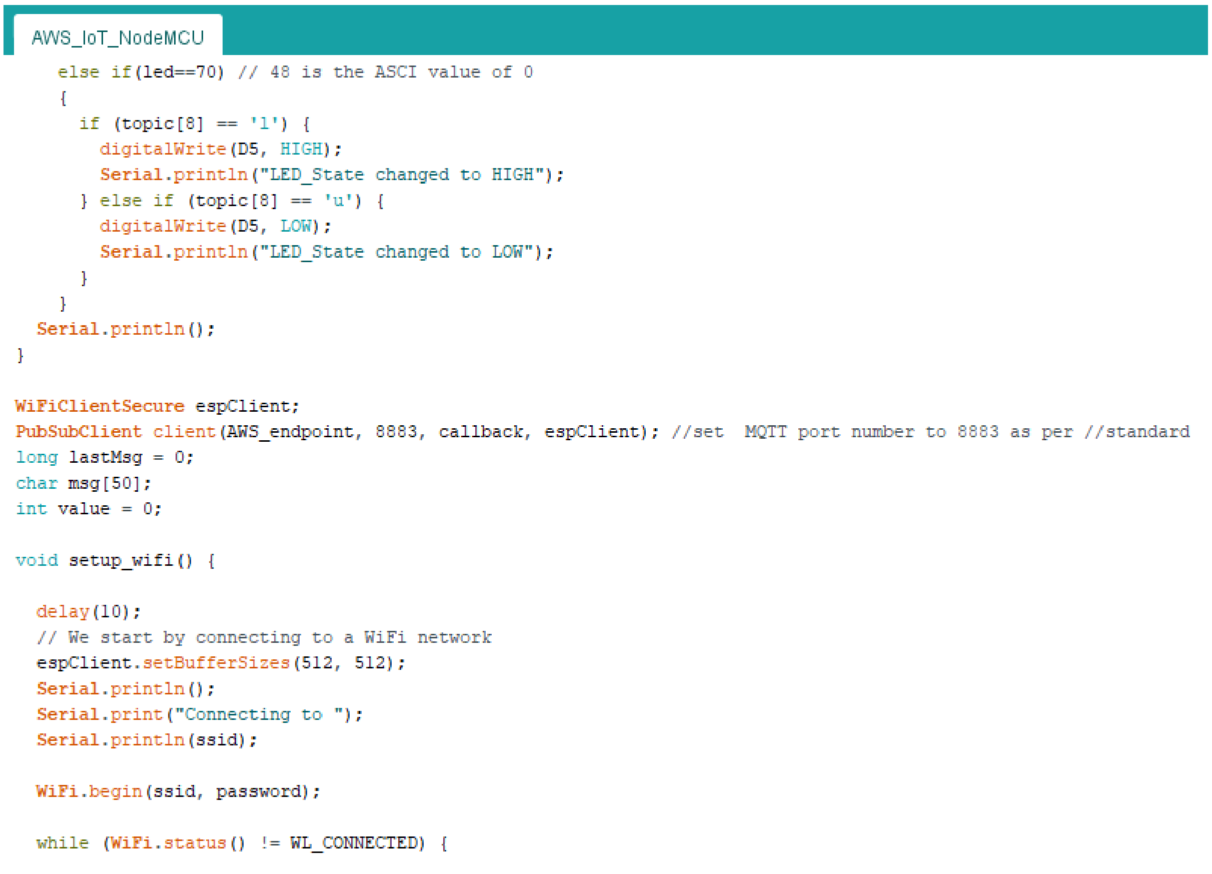
The NodeMCU board has a wifi module inbuilt which connects to wifi and subscribes to various MQTT topics. The LED light is connected to the D5 and the ground pin of the NodeMCU board. Then the logic is added as to when to give the high and low signals for toggling the LED lights.



CODE:

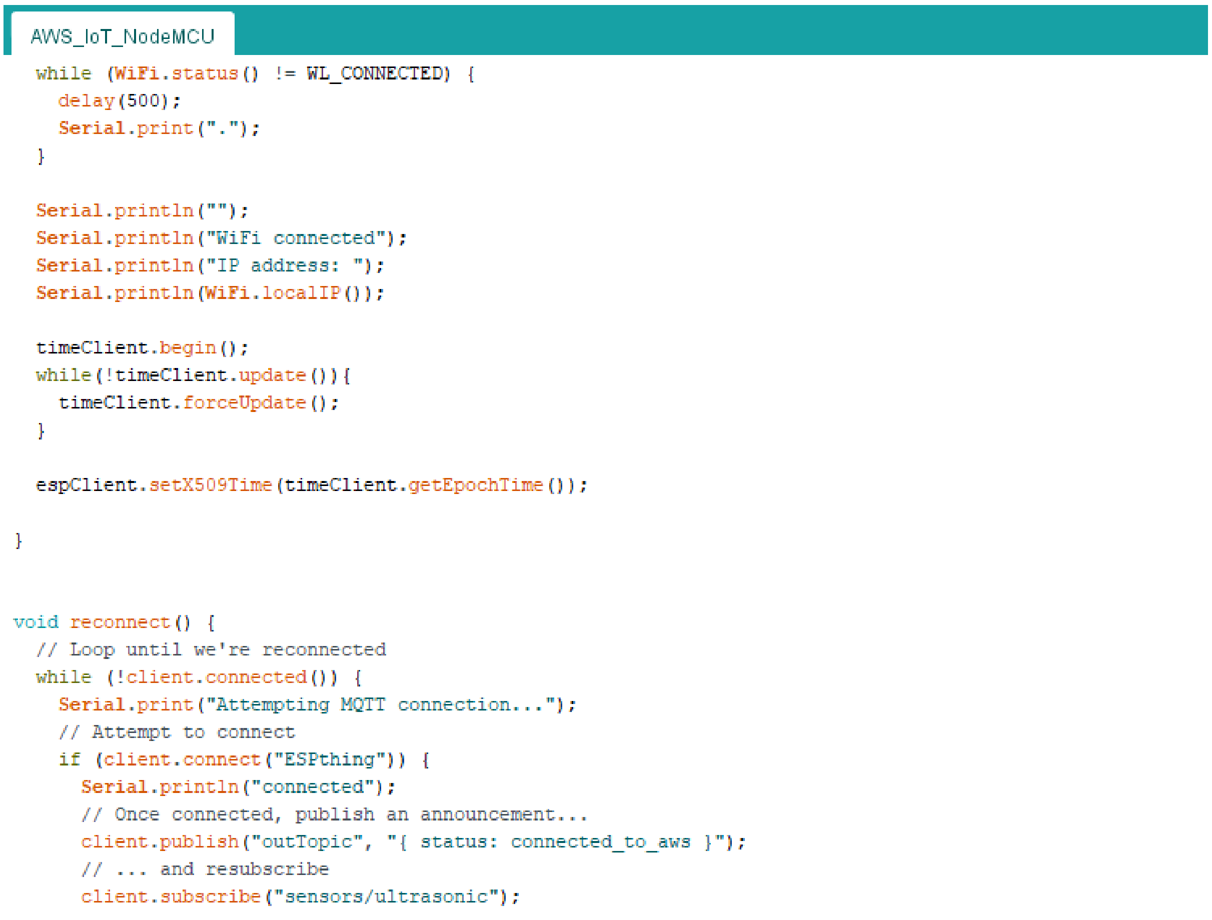


```
// Update these with values suitable for your network.

char* ssid = "[redacted]";
char* password = "[redacted]";

WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "pool.ntp.org");
const char* AWS_endpoint = "a2vsckvuvvwwcl-ats.iot.us-east-1.amazonaws.com"; //MQTT broker ip

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]); // Pring payload content
  }
    char led = (char)payload[13]; // Extracting the controlling command from the Payload to Controlling LED from AWS
    Serial.print("led command=");
    Serial.println(led);
    if(led==78) // 49 is the ASCI value of 1
    {
      if(topic[8] == '1') {
        digitalWrite(D5, LOW);
        Serial.println("LED_State changed to LOW");
      } else if(topic[8] == 'u') {
        digitalWrite(D5, HIGH);
        Serial.println("LED_State changed to HIGH");
      }
    }
    else if(led==70) // 48 is the ASCI value of 0
```



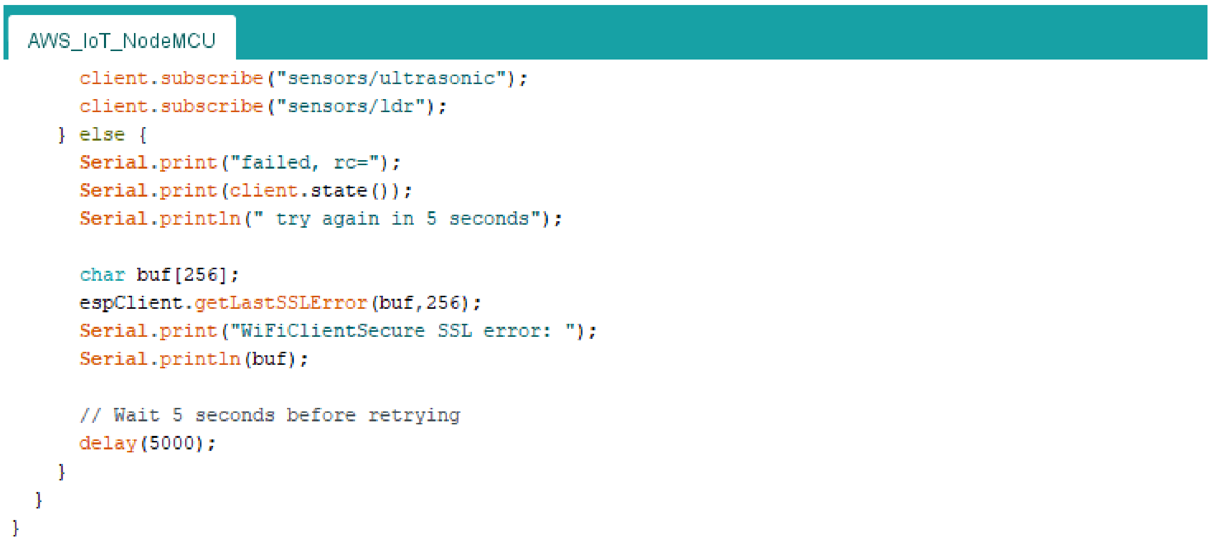
AWS_IoT_NodeMCU

```
    else if(led==70) // 48 is the ASCI value of 0
    {
      if (topic[8] == 'l') {
        digitalWrite(D5, HIGH);
        Serial.println("LED_State changed to HIGH");
      } else if (topic[8] == 'u') {
        digitalWrite(D5, LOW);
        Serial.println("LED_State changed to LOW");
      }
    }
  Serial.println();
}

WiFiClientSecure espClient;
PubSubClient client(AWS_endpoint, 8883, callback, espClient); //set  MQTT port number to 8883 as per //standard
long lastMsg = 0;
char msg[50];
int value = 0;

void setup_wifi() {

  delay(10);
  // We start by connecting to a WiFi network
  espClient.setBufferSizes(512, 512);
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
```

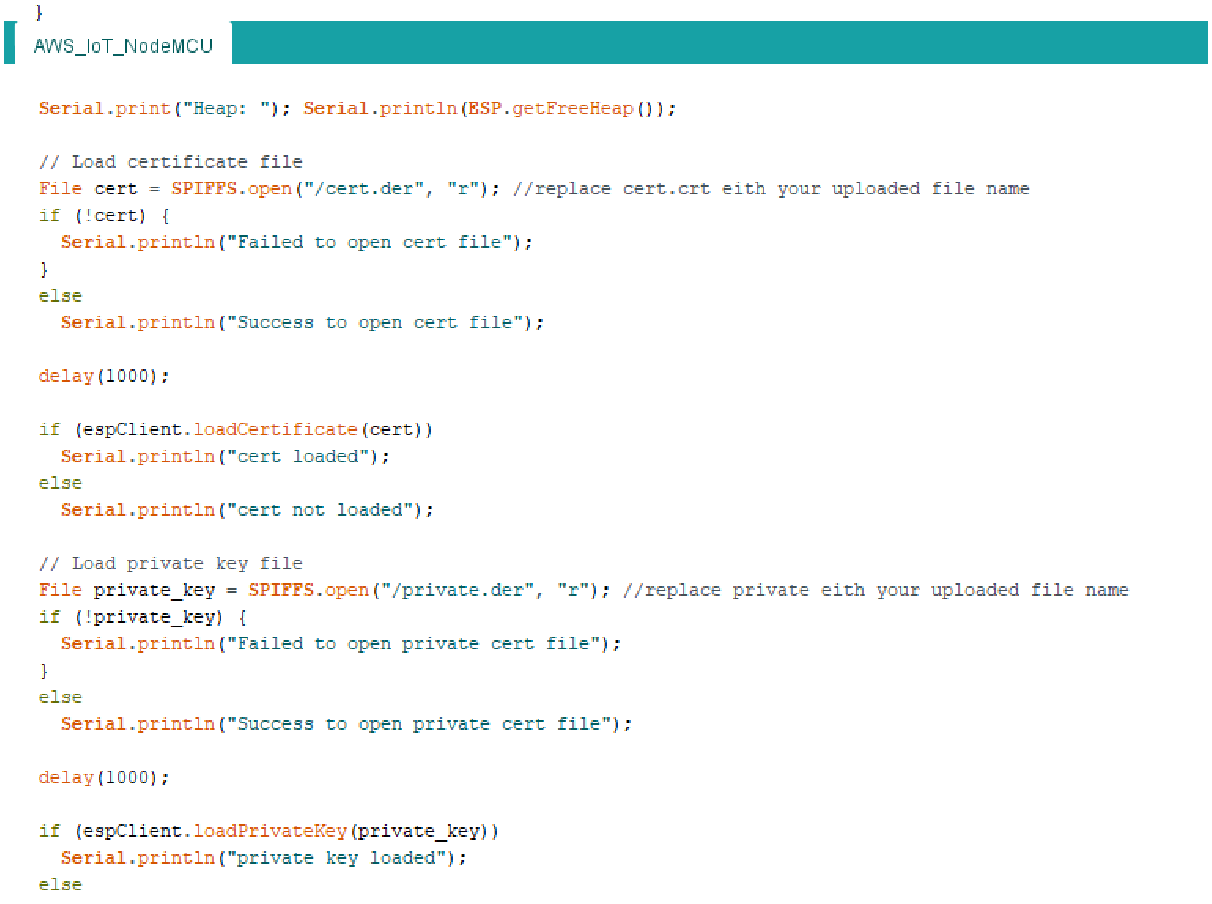


AWS_IoT_NodeMCU

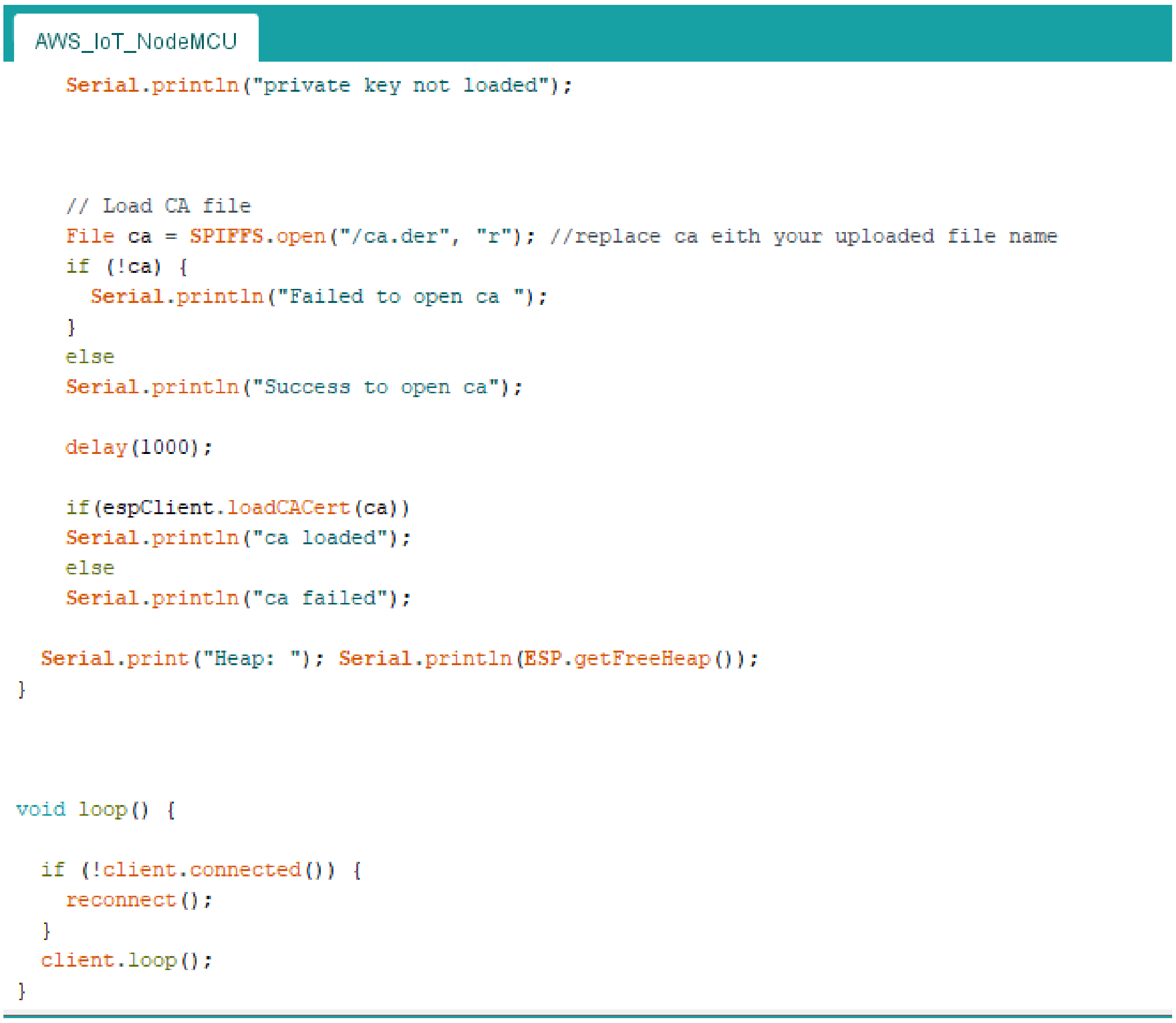
```
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());

  timeClient.begin();
  while(!timeClient.update()){
    timeClient.forceUpdate();
  }

  espClient.setX509Time(timeClient.getEpochTime());

}


void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (client.connect("ESPthing")) {
      Serial.println("connected");
      // Once connected, publish an announcement...
      client.publish("outTopic", "{ status: connected_to_aws }");
      // ... and resubscribe
      client.subscribe("sensors/ultrasonic");
```


AWS_IoT_NodeMCU

```
      client.subscribe("sensors/ultrasonic");
      client.subscribe("sensors/ldr");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");

      char buf[256];
      espClient.getLastSSLError(buf,256);
      Serial.print("WiFiClientSecure SSL error: ");
      Serial.println(buf);

      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}

void setup() {

  Serial.begin(9600);
  Serial.setDebugOutput(true);
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(D5, OUTPUT);
  setup_wifi();
  delay(1000);
  if (!SPIFFS.begin()) {
    Serial.println("Failed to mount file system");
    return;
  }
```


AWS_IoT_NodeMCU

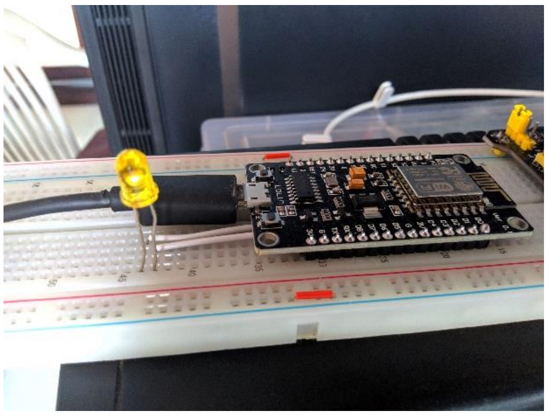
```
  Serial.print("Heap: "); Serial.println(ESP.getFreeHeap());

  // Load certificate file
  File cert = SPIFFS.open("/cert.der", "r"); //replace cert.crt eith your uploaded file name
  if (!cert) {
    Serial.println("Failed to open cert file");
  }
  else
    Serial.println("Success to open cert file");

  delay(1000);

  if (espClient.loadCertificate(cert))
    Serial.println("cert loaded");
  else
    Serial.println("cert not loaded");

  // Load private key file
  File private_key = SPIFFS.open("/private.der", "r"); //replace private eith your uploaded file name
  if (!private_key) {
    Serial.println("Failed to open private cert file");
  }
  else
    Serial.println("Success to open private cert file");

  delay(1000);

  if (espClient.loadPrivateKey(private_key))
    Serial.println("private key loaded");
  else
```



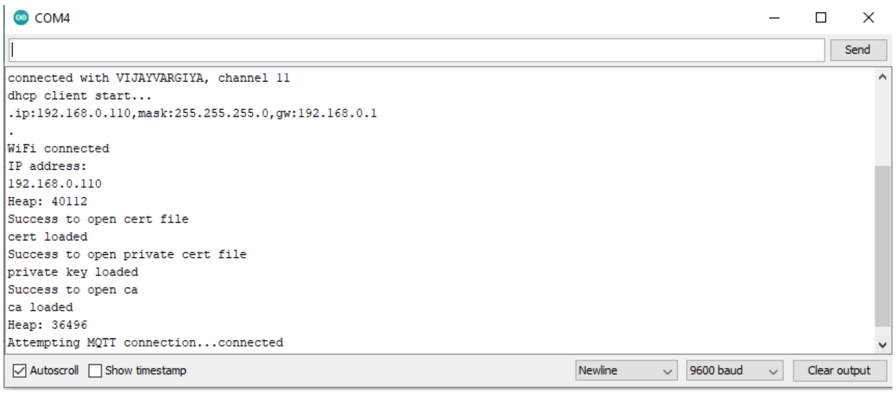
AWS_IoT_NodeMCU

```
    Serial.println("private key not loaded");



    // Load CA file
    File ca = SPIFFS.open("/ca.der", "r"); //replace ca eith your uploaded file name
    if (!ca) {
      Serial.println("Failed to open ca ");
    }
    else
    Serial.println("Success to open ca");

    delay(1000);

    if(espClient.loadCACert(ca))
    Serial.println("ca loaded");
    else
    Serial.println("ca failed");

  Serial.print("Heap: "); Serial.println(ESP.getFreeHeap());
}



void loop() {

  if (!client.connected()) {
    reconnect();
  }
  client.loop();
}
```

The topics subscribed to are **‘sensors/ldr’** and **‘sensors/ultrasonic’.** When a message is received from either of these, the 13th character is checked, if it is either N(78) (in case of ON) or F(70) (in case of OFF). In the case of the ultrasonic sensor the LED works as expected and is switched on case of ON and vice versa. But in the case of the ldr sensor, it is switched off in the case of ON (room is bright) and switched off in the case of OFF (room is dark).



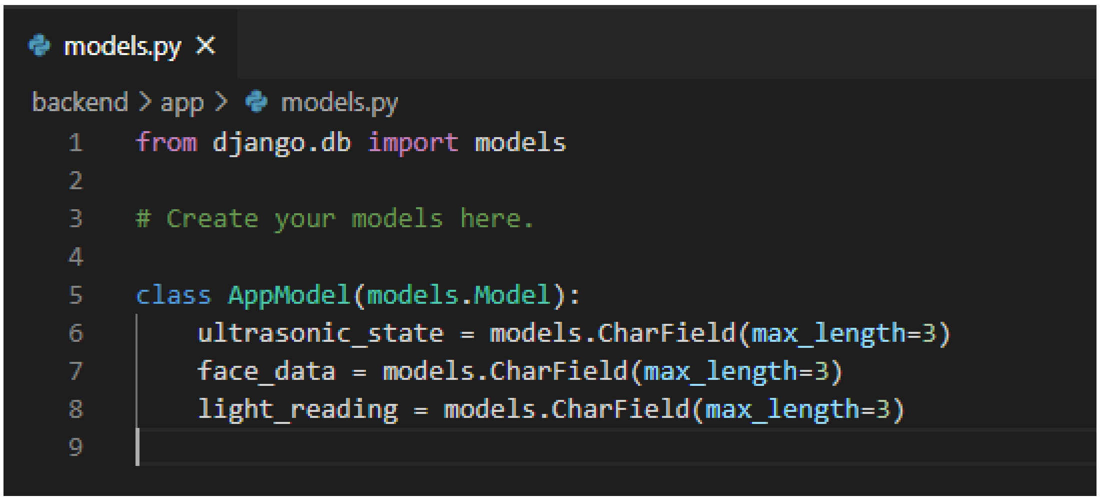
One can even see the messages in the COM screen in the Arduino IDE.



```
connected with VIJAYVARGIYA, channel 11
dhcp client start...
.ip:192.168.0.110,mask:255.255.255.0,gw:192.168.0.1
.
WiFi connected
IP address:
192.168.0.110
Heap: 40112
Success to open cert file
cert loaded
Success to open private cert file
private key loaded
Success to open ca
ca loaded
Heap: 36496
Attempting MQTT connection...connected
```

- **The Backend API :**

For the backend API Django REST API has been used which uses the Django Rest framework. A user first creates a table for the data to be stored in the SQL database. This can be done by adding models in the **models.py** file.



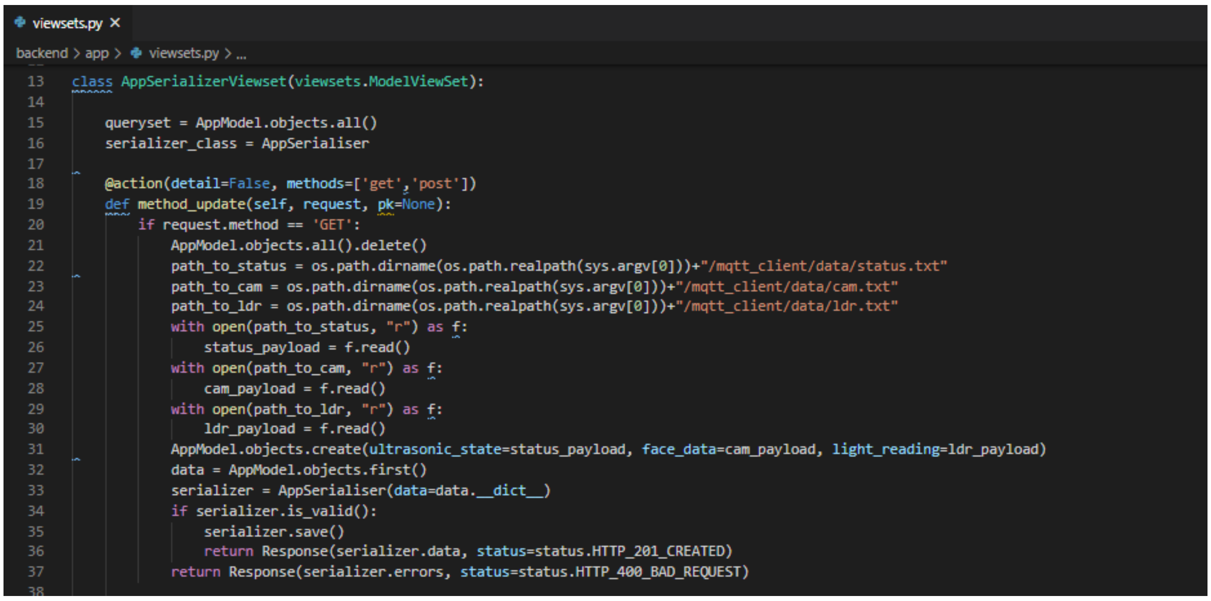
```python
from django.db import models

# Create your models here.

class AppModel(models.Model):
    ultrasonic_state = models.CharField(max_length=3)
    face_data = models.CharField(max_length=3)
    light_reading = models.CharField(max_length=3)

```

The table has the following columns:

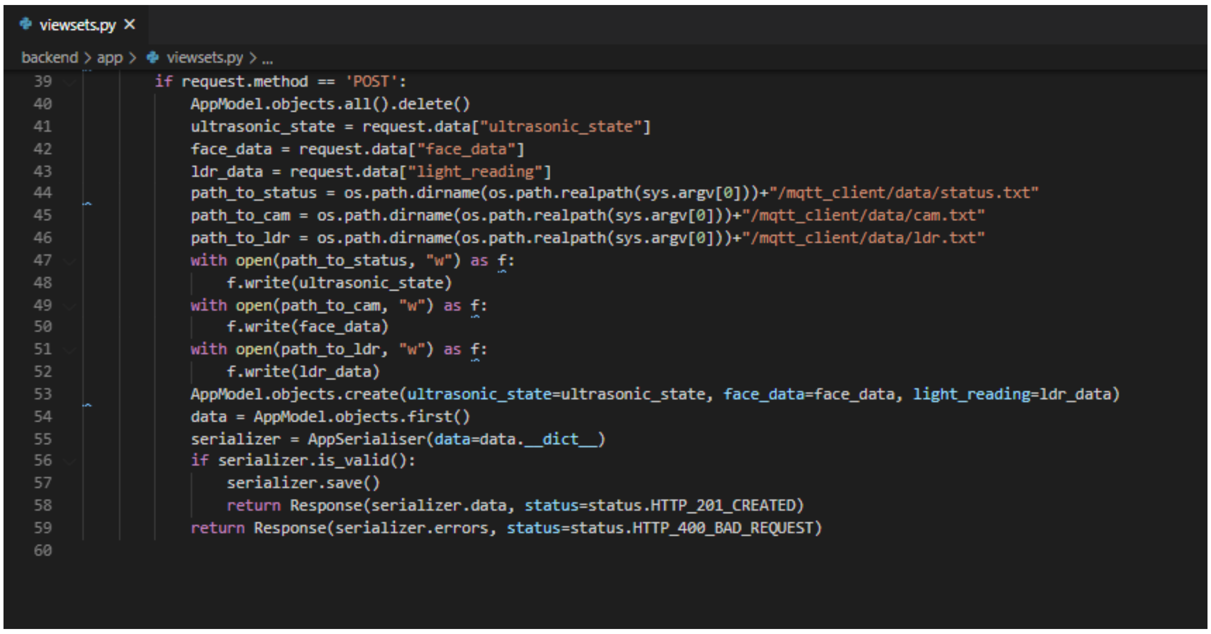
1. *ultrasonic_state*
2. *face_data*
3. *light_reading*

Once that is done a **serializer** is created which allows parsing of data.

The most crucial part of the API is how the views are defined. Viewsets can be used in this case for managing multiple types of HTTP requests.

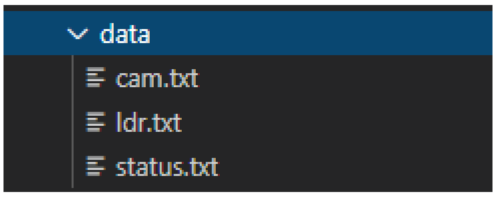


```python
class AppSerializerViewset(viewsets.ModelViewSet):

    queryset = AppModel.objects.all()
    serializer_class = AppSerialiser

    @action(detail=False, methods=['get','post'])
    def method_update(self, request, pk=None):
        if request.method == 'GET':
            AppModel.objects.all().delete()
            path_to_status = os.path.dirname(os.path.realpath(sys.argv[0]))+"/mqtt_client/data/status.txt"
            path_to_cam = os.path.dirname(os.path.realpath(sys.argv[0]))+"/mqtt_client/data/cam.txt"
            path_to_ldr = os.path.dirname(os.path.realpath(sys.argv[0]))+"/mqtt_client/data/ldr.txt"
            with open(path_to_status, "r") as f:
                status_payload = f.read()
            with open(path_to_cam, "r") as f:
                cam_payload = f.read()
            with open(path_to_ldr, "r") as f:
                ldr_payload = f.read()
            AppModel.objects.create(ultrasonic_state=status_payload, face_data=cam_payload, light_reading=ldr_payload)
            data = AppModel.objects.first()
            serializer = AppSerialiser(data=data.__dict__)
            if serializer.is_valid():
                serializer.save()
                return Response(serializer.data, status=status.HTTP_201_CREATED)
            return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```



```python
        if request.method == 'POST':
            AppModel.objects.all().delete()
            ultrasonic_state = request.data["ultrasonic_state"]
            face_data = request.data["face_data"]
            ldr_data = request.data["light_reading"]
            path_to_status = os.path.dirname(os.path.realpath(sys.argv[0]))+"/mqtt_client/data/status.txt"
            path_to_cam = os.path.dirname(os.path.realpath(sys.argv[0]))+"/mqtt_client/data/cam.txt"
            path_to_ldr = os.path.dirname(os.path.realpath(sys.argv[0]))+"/mqtt_client/data/ldr.txt"
            with open(path_to_status, "w") as f:
                f.write(ultrasonic_state)
            with open(path_to_cam, "w") as f:
                f.write(face_data)
            with open(path_to_ldr, "w") as f:
                f.write(ldr_data)
            AppModel.objects.create(ultrasonic_state=ultrasonic_state, face_data=face_data, light_reading=ldr_data)
            data = AppModel.objects.first()
            serializer = AppSerialiser(data=data.__dict__)
            if serializer.is_valid():
                serializer.save()
                return Response(serializer.data, status=status.HTTP_201_CREATED)
            return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

A viewset class is made for handling get and post requests. As mentioned before, we store the state of each sensor in a text file and that file is read by the class. In case of a **get** request, it returns the current status stored in the text files. In case of a **post** request, it updates the current status in the text files with what the



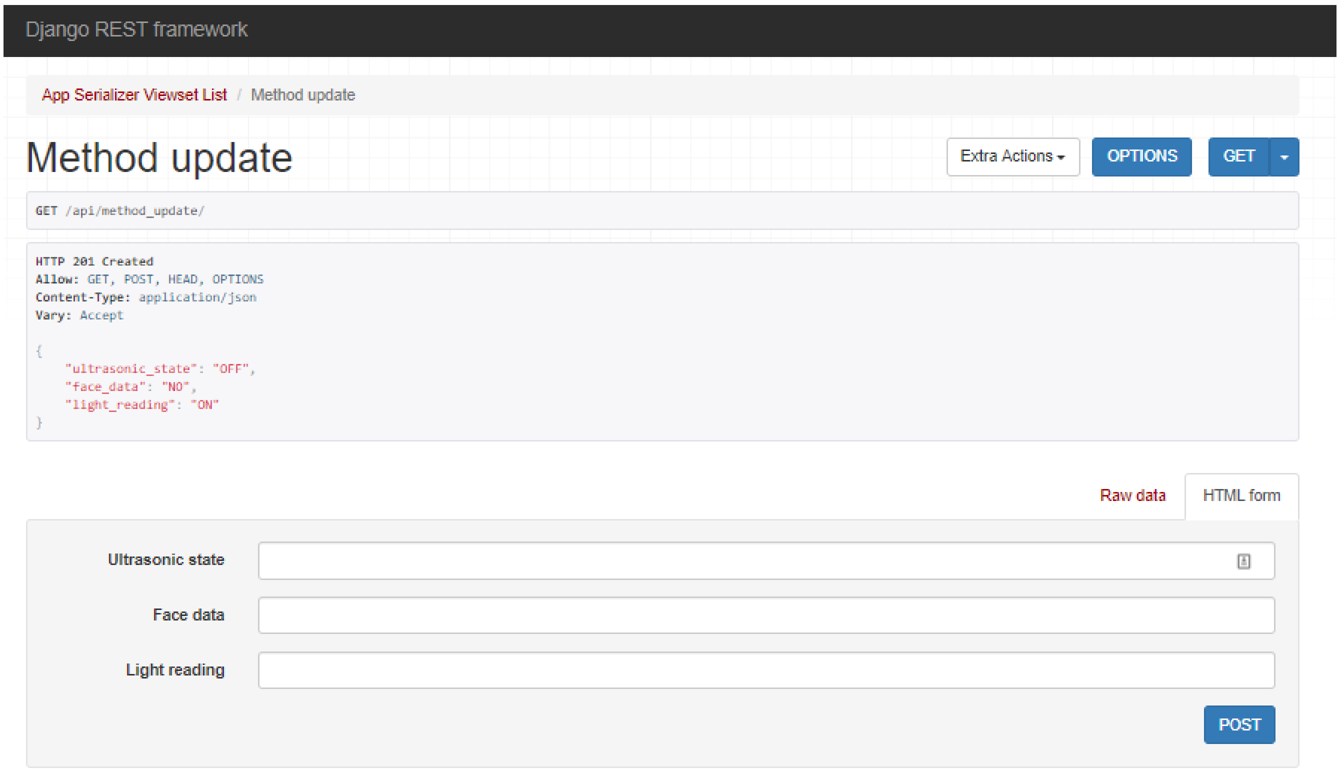
user has posted. This will be helpful later for controlling the status of LED light using the toggle in the dashboard.

The user can start the API by using the command :

python manage.py runserver 0.0.0.0:8000

By using this command any device can access the API if on the same wifi network.

Once we enter the URL, the API shows the current status of the sensors and the camera.



```
GET /api/method_update/
```

```
HTTP 201 Created
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "ultrasonic_state": "OFF",
    "face_data": "NO",
    "light_reading": "ON"
}
```

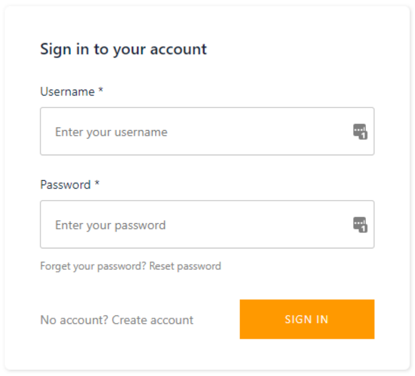
We can test the API by updating the status of each field by entering data in the text fields and clicking on POST.

This API can now be used by the dashboard frontend to extract data or update it too.

- **The Dashboard :**

This is the main UI designed for the user to control all the sensors and keep a check on the status of each sensor. Once the website is done compiling using ReactJS, the user is welcomed with a sign in screen.

This website is accessible on any device connected to the same wifi.



Sign in to your account

Username *

Enter your username

Password *

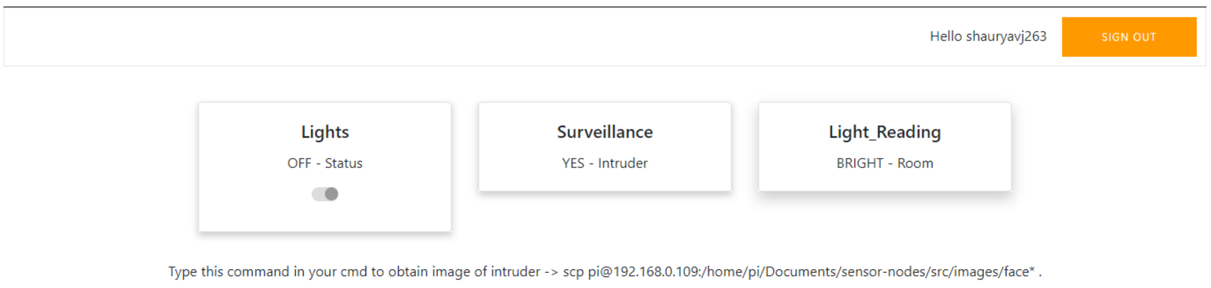
Enter your password

Forget your password? Reset password

No account? Create account

SIGN IN

Once a user creates the account and signs in they can see the status of all the sensors.



Hello shauryavj263

SIGN OUT

Lights

OFF - Status

Surveillance

YES - Intruder

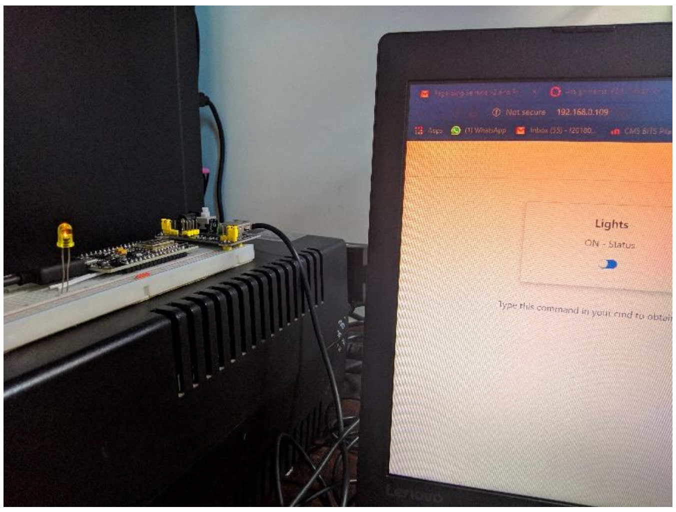
Light_Reading

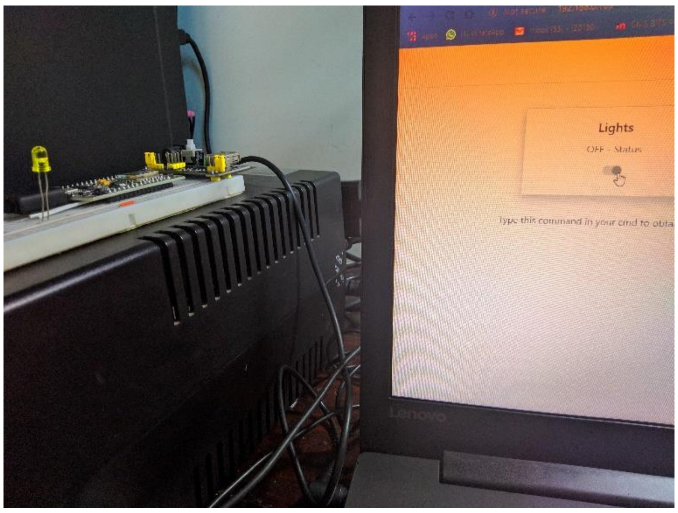
BRIGHT - Room

Type this command in your cmd to obtain image of intruder -> scp pi@192.168.0.109:/home/pi/Documents/sensor-nodes/src/images/face* .

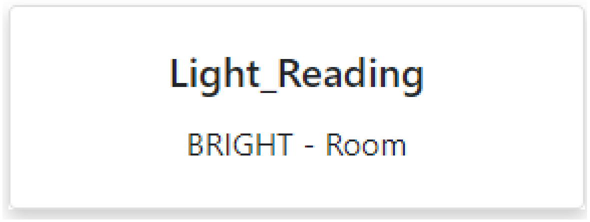
Let’s go over the working of each sensor and its integration in the dashboard UI.

a. **Ultrasonic sensor** : Whenever the sensor detects motion, it sends an MQTT message to AWS IoT client, which is received by the message receiver code which changes the data inside the text file **status.txt**. The API reads the file and updates the data in the database. This is picked up by the frontend and the changes the state as shown. The NodeMCU also receives the message and toggles the LED accordingly.

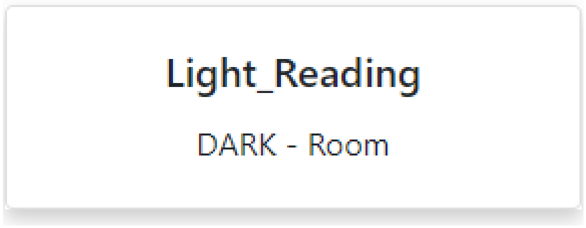




b. **LDR Sensor** : Once there is a change in the brightness of the room, a message is sent to the client and received by the message receiver code. This changes the data in the text file **ldr.txt**. The API makes the necessary change in the database and once the frontend receives the updated data, it changes the state of the sensor. If the room is dark, the LED is turned on by the NodeMCU and vice versa.

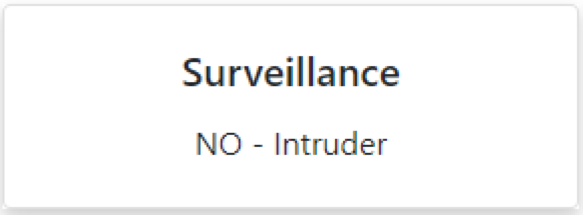

Light_Reading

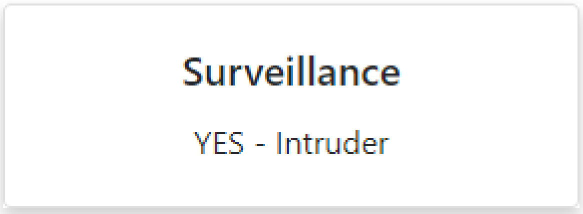
BRIGHT - Room


Light_Reading

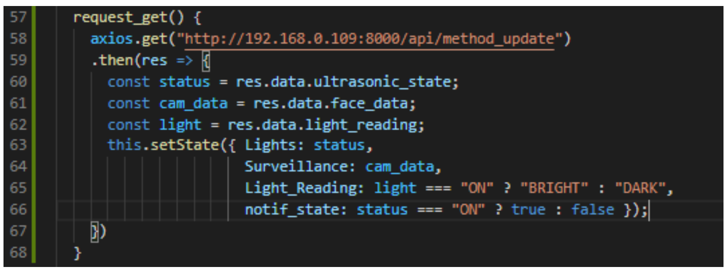
DARK - Room

c. **Camera** : As explained before, once the camera detects a face, it sends the message to client and the **cam.txt** file is updated by the message receiver code. The API reads the file and updates the data in the database. Similarly, the frontend makes the necessary changes.

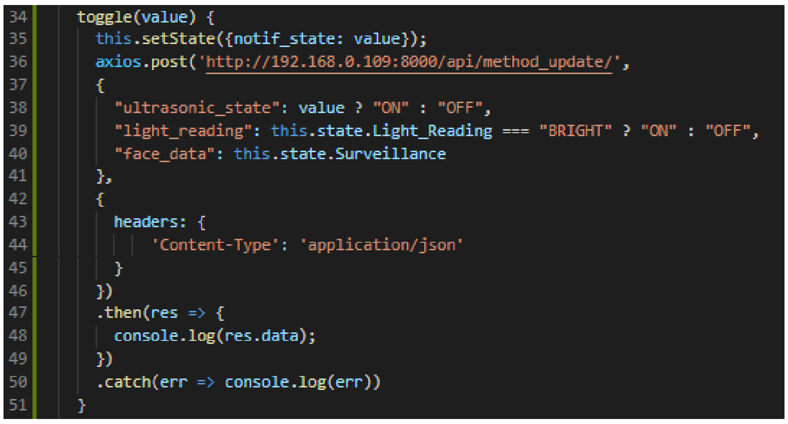




The dashboard accesses the backend database by calling HTTP requests to the API such as **GET** and **POST** using the axios library in JavaScript.



```
request_get() {
  axios.get("http://192.168.0.109:8000/api/method_update")
  .then(res => {
    const status = res.data.ultrasonic_state;
    const cam_data = res.data.face_data;
    const light = res.data.light_reading;
    this.setState({ Lights: status,
                    Surveillance: cam_data,
                    Light_Reading: light === "ON" ? "BRIGHT" : "DARK",
                    notif_state: status === "ON" ? true : false });
  })
}
```



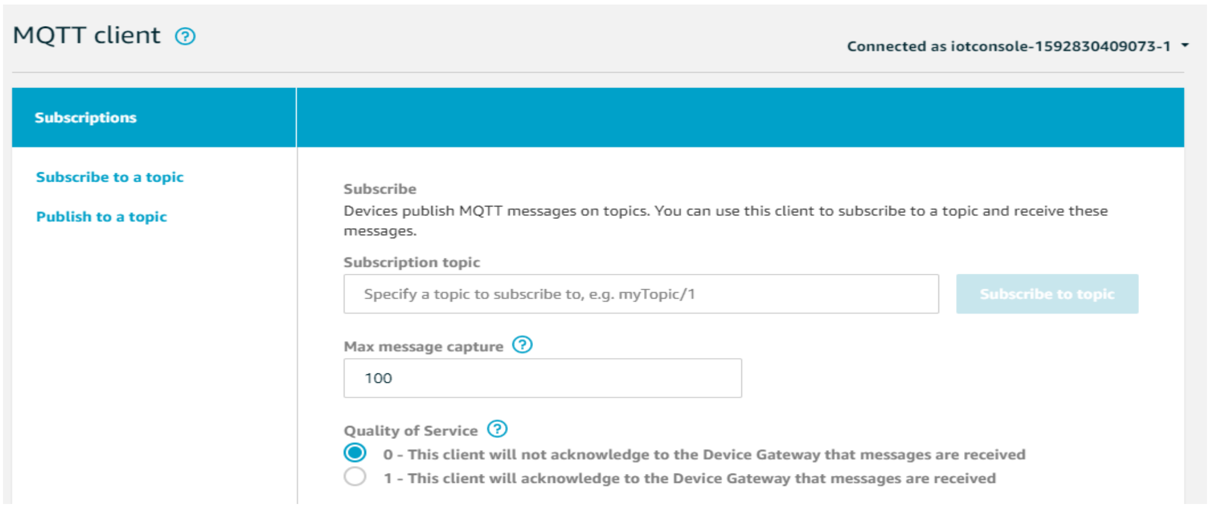
```
toggle(value) {
  this.setState({notif_state: value});
  axios.post('http://192.168.0.109:8000/api/method_update/',
  {
    "ultrasonic_state": value ? "ON" : "OFF",
    "light_reading": this.state.Light_Reading === "BRIGHT" ? "ON" : "OFF",
    "face_data": this.state.Surveillance
  },
  {
    headers: {
        'Content-Type': 'application/json'
    }
  })
  .then(res => {
    console.log(res.data);
  })
  .catch(err => console.log(err))
}
```

- **AWS and its Components:**

AWS stands for Amazon web services. They provide multiple cloud computing services and the ones used in this project are –

a. **IoT Core** : It allows one to easily connect any number of devices to the cloud and to other devices. It supports HTTP, WebSockets, and MQTT, a lightweight communication protocol specifically designed to tolerate intermittent connections, minimize the code footprint on devices, and reduce network bandwidth requirements.

It has its own unique endpoint so the user can send messages to the client. There is even a testing feature where the user can subscribe and publish in various topics.


MQTT client

Connected as iotconsole-1592830409073-1

Subscriptions

Subscribe to a topic

Publish to a topic

Subscribe

Devices publish MQTT messages on topics. You can use this client to subscribe to a topic and receive these messages.

Subscription topic: Specify a topic to subscribe to, e.g. myTopic/1

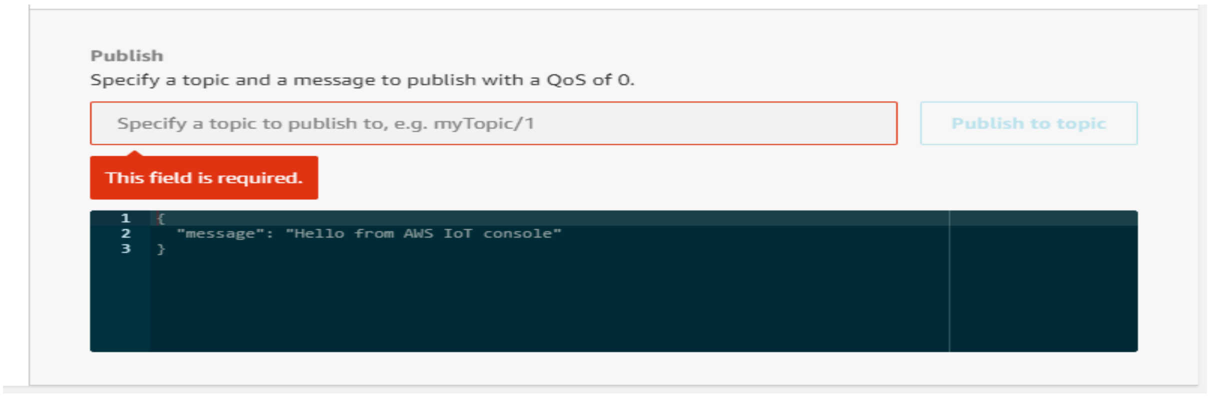
Subscribe to topic

Max message capture: 100

Quality of Service

[✓ 0 - This client will not acknowledge to the Device Gateway that messages are received]

1 - This client will acknowledge to the Device Gateway that messages are received


Publish

Specify a topic and a message to publish with a QoS of 0.

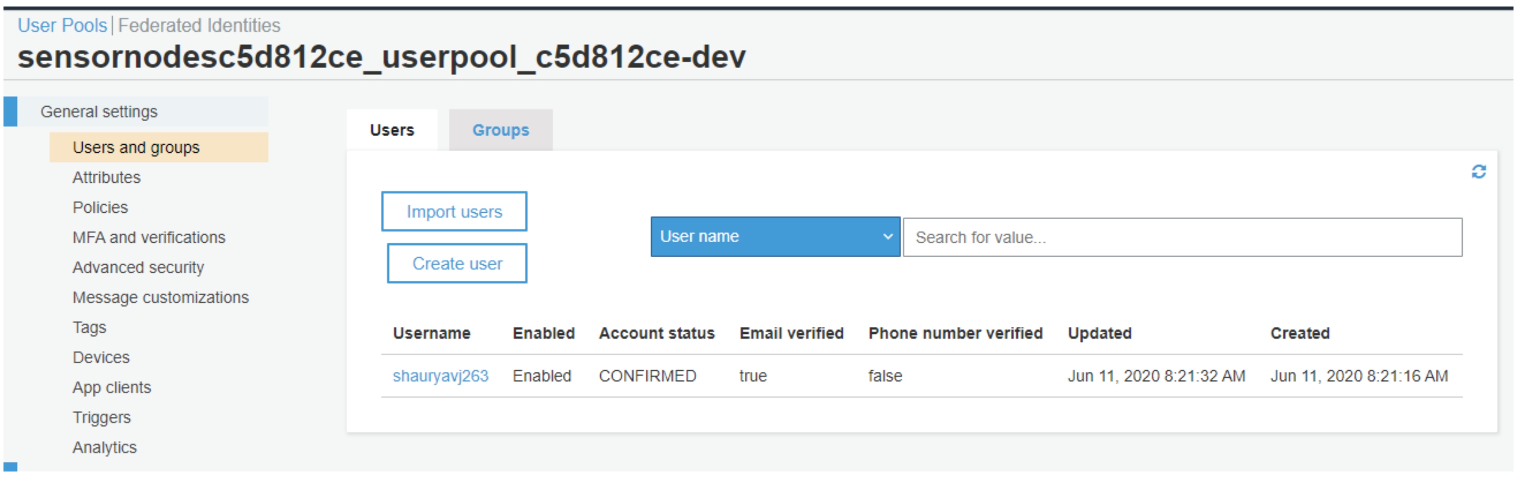
Specify a topic to publish to, e.g. myTopic/1

Publish to topic

This field is required.

```
{
  "message": "Hello from AWS IoT console"
}
```

b. **Cognito** : This service adds the sign up and sign in features in the dashboard. Cognito user pools can be used to talk to the AWS IoT Core by linking them to Cognito Federated Identities.


User Pools | Federated Identities

sensornodesc5d812ce_userpool_c5d812ce-dev

General settings
- Users and groups
- Attributes
- Policies
- MFA and verifications
- Advanced security
- Message customizations
- Tags
- Devices
- App clients
- Triggers
- Analytics

Users | Groups

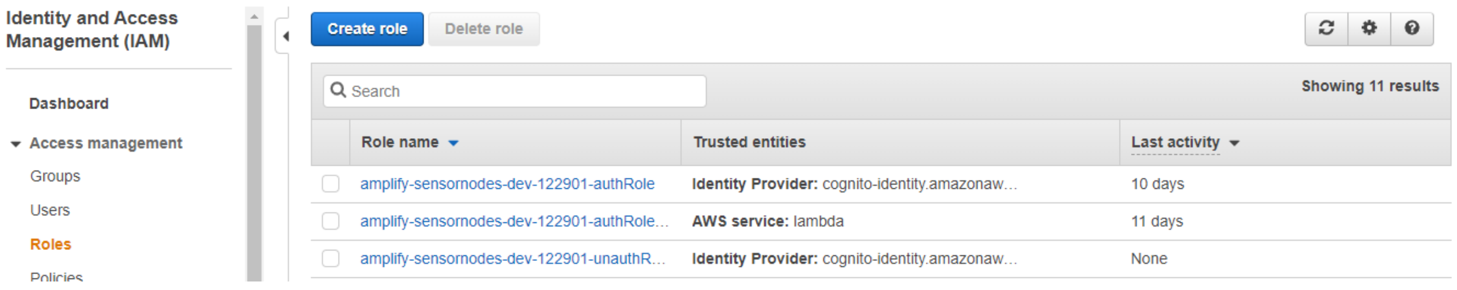
Import users

Create user

User name: Search for value...

Username	Enabled	Account status	Email verified	Phone number verified	Updated	Created
shauryavj263	Enabled	CONFIRMED	true	false	Jun 11, 2020 8:21:32 AM	Jun 11, 2020 8:21:16 AM

c. **IAM** : Identity and Access Management enables user to manage access to AWS services and resources securely. Using IAM, user can create and manage AWS users and groups, and use permissions to allow, and deny their access to AWS resources.



Role name	Trusted entities	Last activity
amplify-sensornodes-dev-122901-authRole	**Identity Provider:** cognito-identity.amazonaw...	10 days
amplify-sensornodes-dev-122901-authRole...	**AWS service:** lambda	11 days
amplify-sensornodes-dev-122901-unauthR...	**Identity Provider:** cognito-identity.amazonaw...	None

One example could be how much access an unauthorized user can have compared to an authorized user. Multiple roles can be created and each role can be given certain access over the data.

Conclusion

IoT Home Automation solutions help save electricity and are convenient for the user. This project is about making a dashboard which lets the user monitor the sensors and the current status of the devices. It also adds security to the user’s house by providing a security cam which scans for faces. The ultrasonic sensor checks for motion and toggles the lights when a user walks by. The light sensor toggles the lights on when it’s too dark and switches them off when its bright. These 3 solutions are added and displayed in the dashboard for the user to monitor/control. Django rest framework is used for building the API and ReactJS for the frontend of the dashboard.

The cloud client used here is AWS (Amazon Web Services) and it has multiple services which are used here such as IoT Core (for the MQTT communication), Cognito (for user pools), and IAM (auth roles).

Another useful library used is AWS Amplify which makes adding auth in the dashboard very easy.

There can be more improvements such as deploying the dashboard to the web so that the user can toggle the lights even when outside the house and improving the framerate of the camera. These can be added later on in the future.

References

1. http://www.electroniclinic.com/raspberry-pi-3-ldr-sensor-circuit-and-python-programming/

2. http://crazykoder.com/2019/01/13/let-raspberry-pis-talk-using-aws-iot-shadow/

3. https://electronicsinnovation.com/controlling-led-from-aws-using-nodemcu-arduino-ide/

4. https://theskenengineering.com/building-a-react-js-app-with-aws-iot/

5. https://tutorials-raspberrypi.com/raspberry-pi-ultrasonic-sensor-hc-sr04/