Vanier College

Computer Science Department

Team Project – IoT Final Project Report

420-531-VA IoT: Internet of Things Fall 2023

Section 00002

Mubeen Khan (1962558)

Teacher: Samad Rostampour

2023-12-13

Table of Contents

GitHub Repository

For better view of the code, please view the GitHub Repository for this project:

https://github.com/Mubeenkh/IoT_Final_Project

Project Overview and Executive Summary

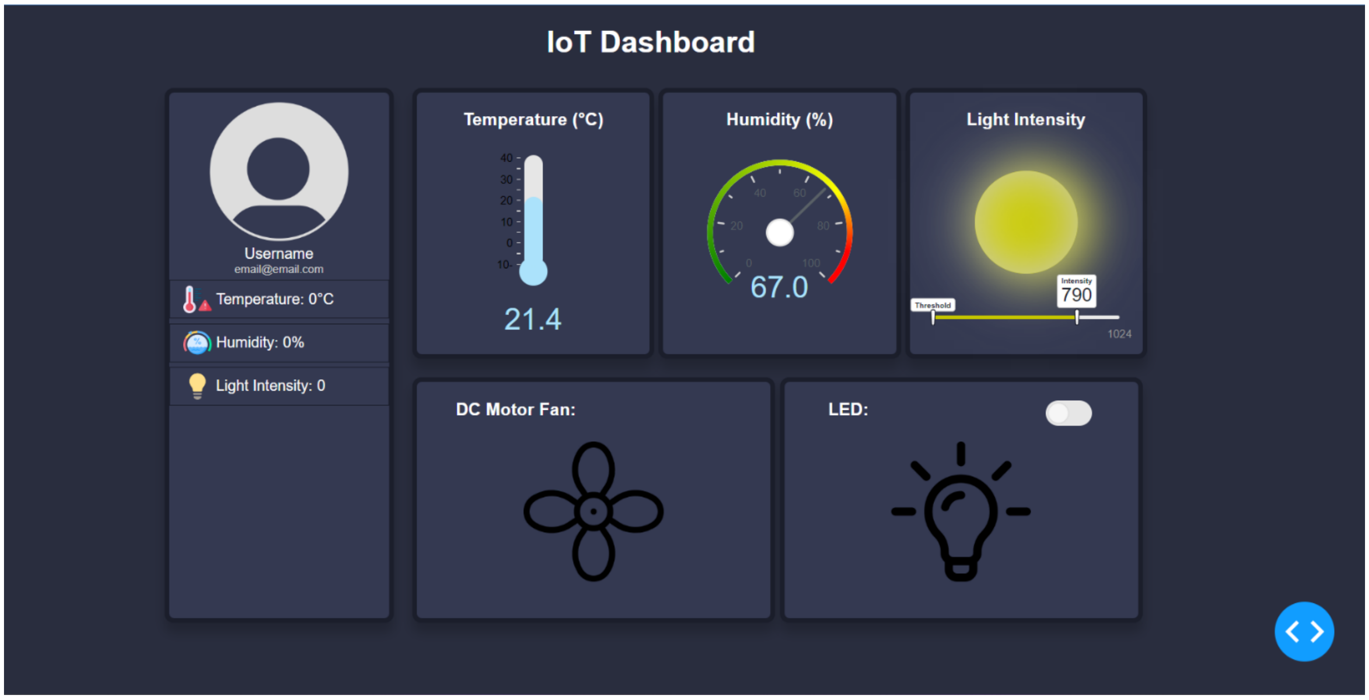
The goal of this IoT project is to connect the physical world with the virtual world, by connecting devices and enabling them to send and receive information to each other. In this project, we designed and built our own IoT system both in the physical world and the virtual world using a breadboard with components and a dashboard using html and python. The system is build using different technologies and devices we have learned throughout the course and implement them into one single project. We utilized what we learned in this course to build an application that will capture data from the sensors and make smart decisions based on that data.

Project scopes and objectives

The object of this project is to utilize the various technologies we have seen and design a smart home environment. The system must be able capture data and make decisions using those data. The different devices must also be able to communicate and exchange data with each other through the cloud or a local server. Finally, we must design a web based IoT dashboard that enables users to control and monitor the smart home system from anywhere, any place, at any time.

This project was done in a matter of four months, equivalent to an entire semester. The project was separated into four phases, each one having its own objectives and given a decent amount of time to accomplish. However, all the phases, when combined, must work as one.

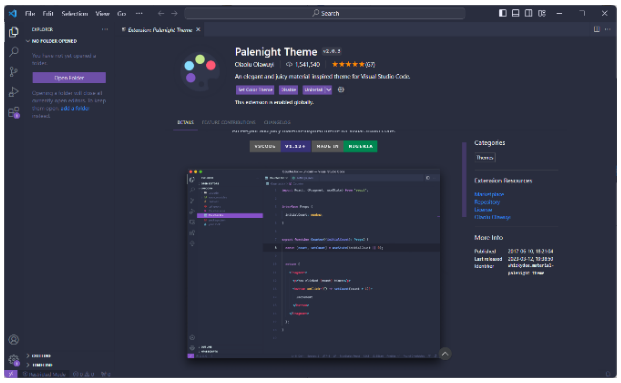
Dashboard



Dashboard Design

Color Pallet:

The color pallet was inspired by a theme which we found in the extensions of Visual Studio Code. The name of that theme was “Palenight Theme”, containing different shades of dark blue.

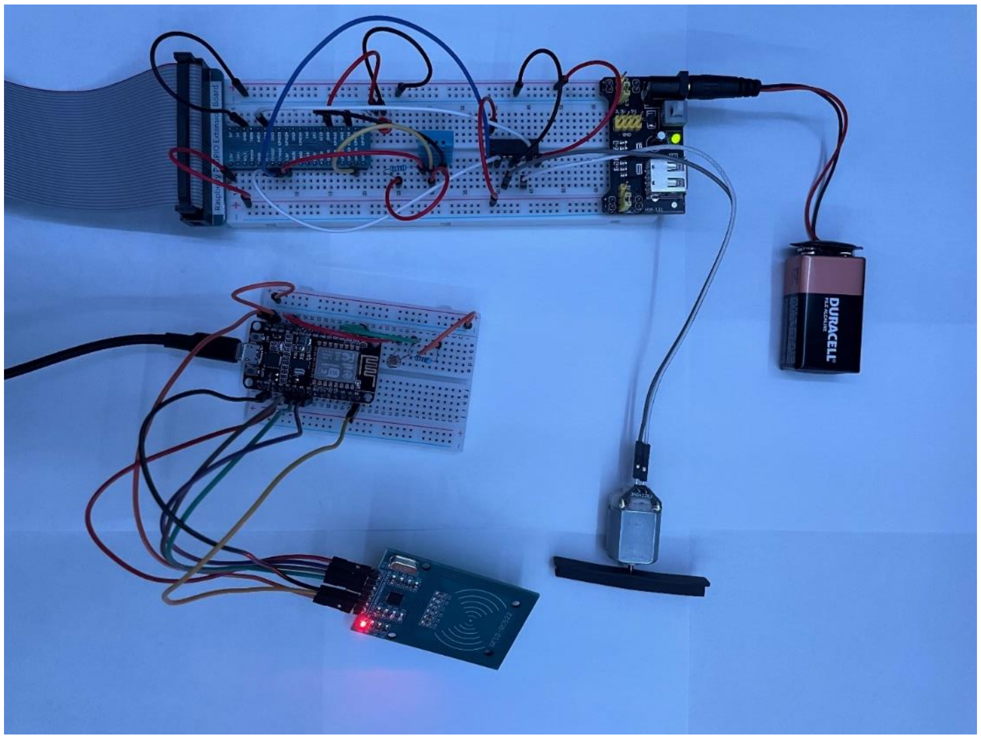

Palenight Theme
Olaolu Olawuyi 1,541,540 (67)
An elegant and juicy material-inspired theme for Visual Studio Code.
This extension is enabled globally.

Cards Layout:

The left side of the dashboard contains all the threshold information of the user as well as their username and email. We originally placed the ID of the RFID tag. However, for security reasons, we thought it would be best to not display it on the screen. We also placed images next to each threshold so the user can quickly understand what each threshold is meant for.

The top side of the dashboard contains all the data captured from the sensors. The data captured are the temperature, humidity, and the light intensity of the room. Finally, the bottom side contains all the images representing the physical hardware. These images will be manipulated using the sensors data or according to the user responses. Originally, we placed the light intensity and the LED at the bottom of the dashboard. However, it made more sense to separate the features in a way where the ones who capture data and those who act as a “output” device will be separated from each other.

Full Project Wiring



Requirement and Materials

In order to make the entire system properly work, several imports are required on Python and Arduino IDE. All hardware components are also required to have a fully functional web page with zero to little errors. We say “zero to little” because if the wiring is not properly done, or a device pin is damaged, the dashboard might display and error. We will take a closer look at the components and where each library is being used later on when we go in detail with each phase.

Components used in the project:

- Raspberry-Pi
- Breadboard
- ESP8266
- RC522 RFID Module (RFID Reader)
- RFID Tags
- DHT11
- Photoresistor
- LED
- DC Motor
- L293D motor driver
- HW-131 for power supply
- 9v battery for power supply
- 1x Resistor (220Ω)
- 2x Resistor (10kΩ)
- Male to male wires
- Male to female wires

Libraries used in the project:

Python:

- RPI.GPIO
- Dash
- dash_daq
- Freenove_DHT
- smtplib
- imaplib
- email
- secrets
- string
- time
- paho.mqtt.client
- SQLite3

Arduino IDE:

- ESP8266WiFi.h
- PubSubClient.h
- Arduino.h
- SPI.h
- MFRC522.h

Work Breakdown Structure

Phase 1:

For Phase 1, I did everything on my own. The reason for that was because my teammates were not easy to contact. One would answer late and the other would not communicate unless you contact them, otherwise they would not say anything.

Since this phase was short and simple, I decided to try working on it on my own. I was able to understand how dash work and how to connect buttons and any other feature to a callback. I even connected the dashboard and the breadboard to each other.

Phase 2:

For Phase 2, I, Mubeen, worked on sending the email to the user using an email and also reading the email reply that was sent by the user. Furthermore, I also worked on capturing the DHT11 temperature and humidity data and displaying it on the dashboard. I made the values update on the dashboard using a component called "Interval" which updates the app in real-time without having to reload the page or clicking on any button. This task was originally given to someone else. However, that person would not answer us, so I decided to take over to prevent any last-minute errors. Finally, I worked on the CSS for the dashboard since I was the most familiar with CSS.

I assigned everything related to the motor to my teammate, Damiano. He worked on the wiring and turning on the fan when needed. I had to help him on making the code work since he had some bugs that prevented the fan from opening.

Phase 3:

In Phase 3, I worked on capturing the light intensity from the Photoresistor using the ESP8266. Since our Arduino UNO had no integrated Wi-Fi module, we had to use the ESP8266. The Wi-Fi module was needed to transfer the data to the app through the network using the MQTT protocol. I was also responsible for Publishing the data to the broker on Arduino IDE and Subscribing to the Broker on Python. This was done using a topic that we named "ESP8266/Photoresister". Finally, I did the CSS for this deliverable as well.

For this phase, I assigned Damiano with the task of sending the data captured from the Photoresistor to the app file and displaying it on the dashboard. He also used the "Interval" component to constantly get the data from the broker and update the dashboard at all times.

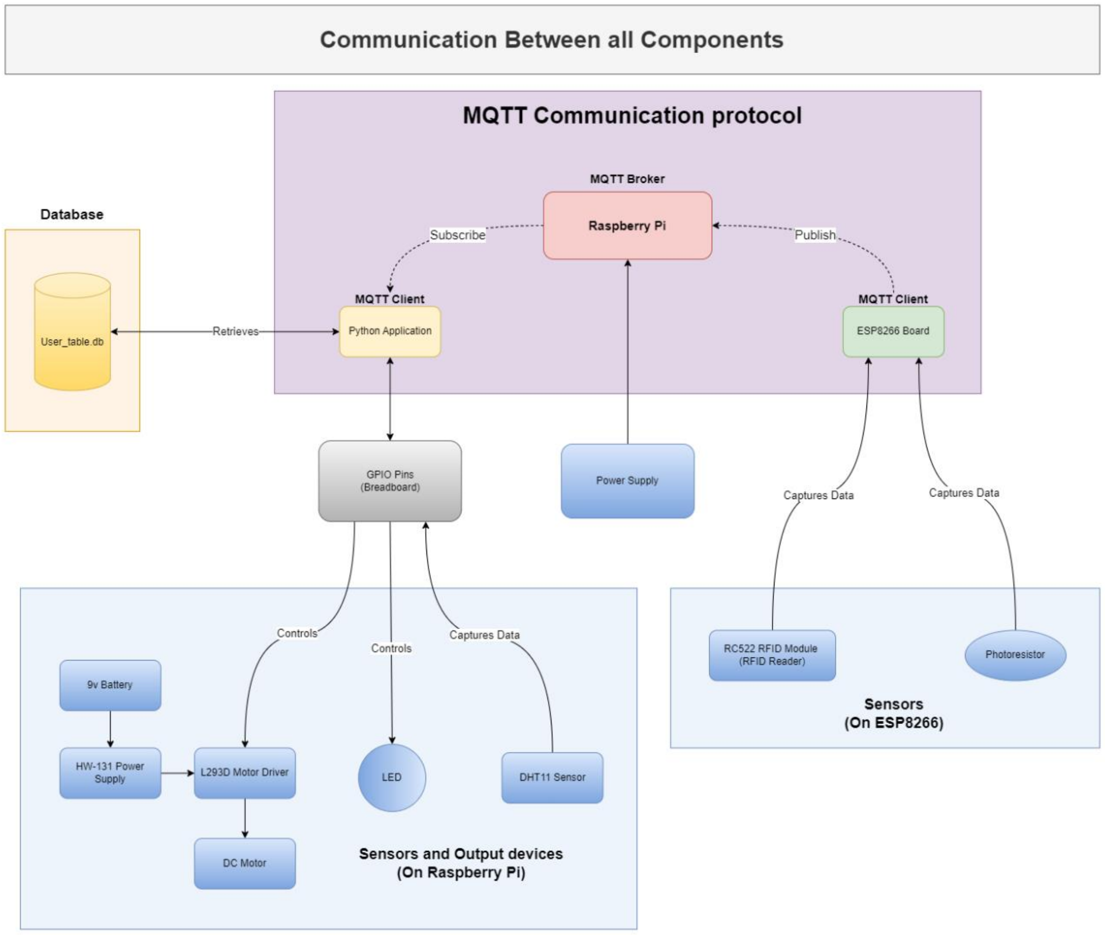
I gave him this small task because he works on things last minute. I, on the other hand, like to finish work earlier so that if we have any bugs or errors, we will have time to fix it. What happened during the phase what exactly what I said, he worked on its last minute and had several bugs in his code. I had to correct his work so that everything can work properly before we presented it.

Phase 4:

In Phase 4, I worked on reading the data from the RFID Tag and publishing it to the broker on Arduino IDE, as well as subscribing to the broker on python. I also created the SQLite database and added the necessary data in it. Furthermore, I worked on sending an email to the user who tapped the RFID tag on the reader, to let them know when they logged in.

I assigned the task of retrieving the user information from the database to Damiano and also displaying the data to the dashboard. However, he had some issues with his RFID reader, and did not do much to try fixing his issue. I ended up doing his part as well.

Project Block Diagram



Communication Protocol

The **MQTT (Message Queuing Telemetry Transport) protocol** is a lightweight messaging protocol ideal for IoT environments. It uses the **publish-subscribe model** to exchange data between the devices. It requires two **MQTT clients**, one who publishes data to the Broker using a Topic, and another client that subscribes to the Topic of a broker to retrieve the data. A client could even do both publish and subscribe in some cases. The **MQTT Broker** is the interface between the clients. It receives data and distributes it to the clients that want to view it.

Deliverables

Phase 1 activities:

Method and Solution:

In this phase, we implemented a switch on the dashboard that can manipulate an LED on both the breadboard and the dashboard. The goal was to get the data from the switch and use it to turn the LED on and off. The switch was on the dashboard while the LED was on both the dashboard and the breadboard. When the switch is on, the LED should also be on. If the user turns the switch off, the LED must turn off.

Components and Libraries

Components used:

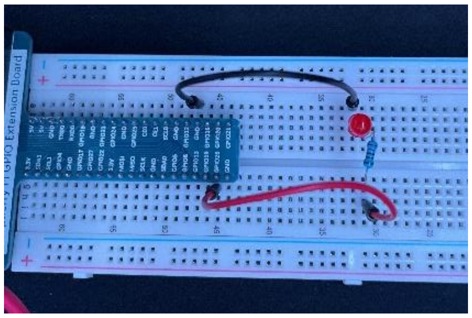
- LED
- Resistor (220Ω)
- 2x male-to-male Wires
- Breadboard
- Raspberry-Pi

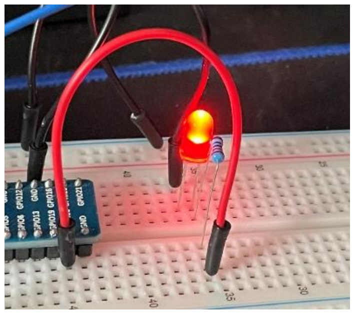
Libraries imported:

- Dash
- RPI.GPIO

Breadboard:

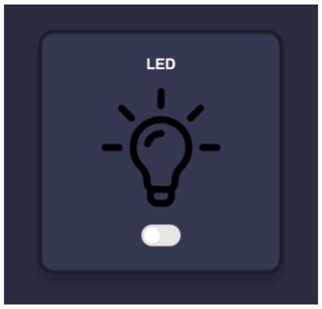
On the breadboard, we have an LED that, on its own, cannot do anything. In order to turn the LED on, the switch on the dashboard must be turned on. If we want to turn the LED off, we simply need to turn the switch off.

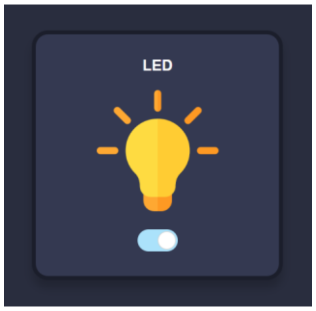




Dashboard:

On the dashboard, we can see the switch that is manipulating the LED. However, we are also displaying an image that represents the state of the LED. It will work the same way as the real LED on the breadboard by turning on and off depending on switch.





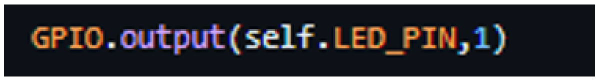
Breakdown of the code:

The code is written in a way that we are separating the application and the code that is manipulating the LED on the breadboard. In the **app.py**, we are importing a file called **LED.py** which contains all the needed imports and commands to manipulate the LED on the breadboard.

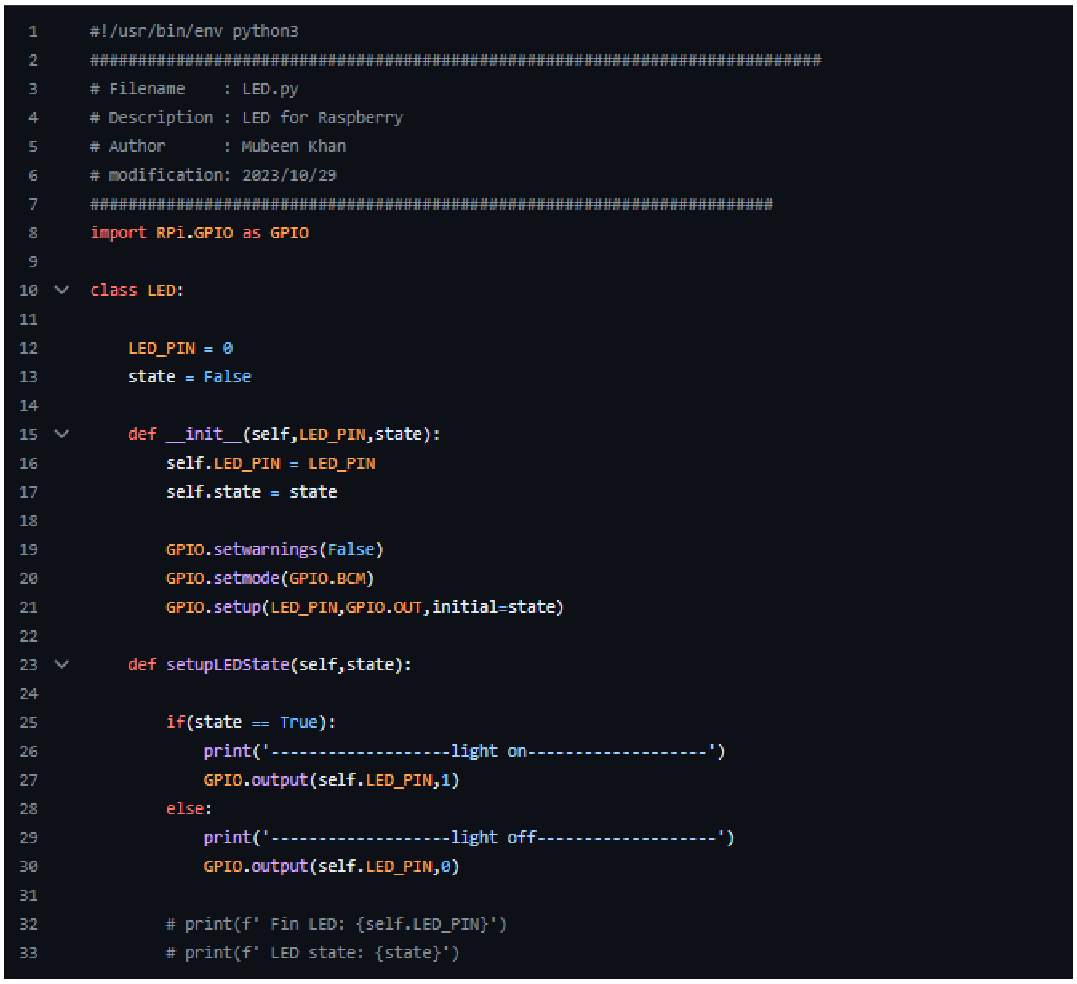
LED.py

The function turning the LED on and off is called **setupLEDSate** where we pass a Boolean parameter named "**state**". This parameter is used to let the system know when to turn the LED on or off.

GPIO.output(Pin #, 0 or 1) is the line that's turning the LED on or off



```python
GPIO.output(self.LED_PIN,1)
```

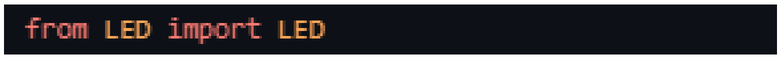


```python
#!/usr/bin/env python3
##############################################################################
# Filename    : LED.py
# Description : LED for Raspberry
# Author      : Mubeen Khan
# modification: 2023/10/29
#########################################################################
import RPi.GPIO as GPIO

class LED:

    LED_PIN = 0
    state = False

    def __init__(self,LED_PIN,state):
        self.LED_PIN = LED_PIN
        self.state = state

        GPIO.setwarnings(False)
        GPIO.setmode(GPIO.BCM)
        GPIO.setup(LED_PIN,GPIO.OUT,initial=state)

    def setupLEDState(self,state):

        if(state == True):
            print('------------------light on------------------')
            GPIO.output(self.LED_PIN,1)
        else:
            print('------------------light off------------------')
            GPIO.output(self.LED_PIN,0)

        # print(f' Fin LED: {self.LED_PIN}')
        # print(f' LED state: {state}')
```

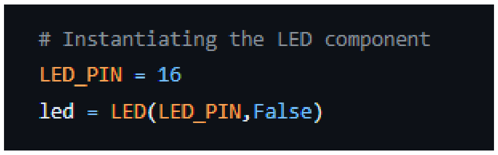
You need to import RPI.GPIO in order to use the GPIO pins of the Raspberry Pi.

app.py

In the app.py file, we need to import and instantiate the LED class. It requires the LED_PIN which is the GPIO pin, and a Boolean value to change the state of the LED. You can use any of the GPIO pins, as long as you change the LED_PIN value to the pin you are using.

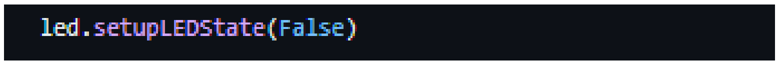


```python
from LED import LED
```

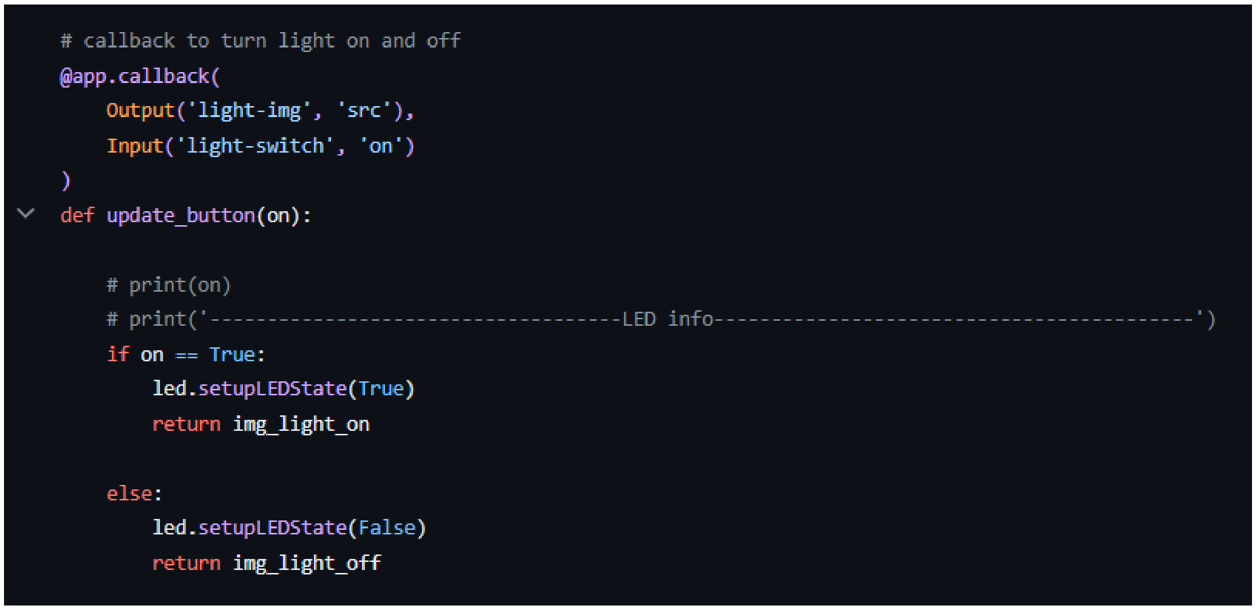


```python
# Instantiating the LED component
LED_PIN = 16
led = LED(LED_PIN,False)
```

In **update_button**, we are changing the state of the LED using the “on” value as an input. This value returns a Boolean value. When the switch is on, “on” becomes True, but when the switch is off, “on” becomes False. This function will use the LED class to change the state of the LED.



```python
led.setupLEDState(False)
```

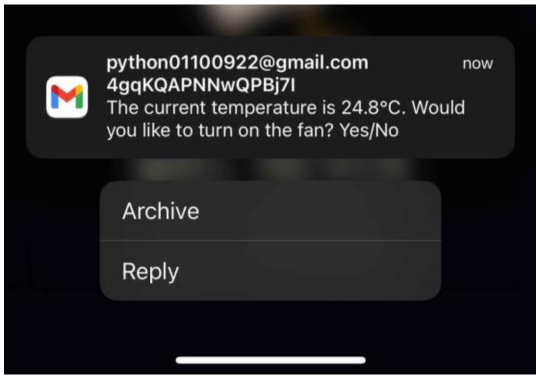


```python
# callback to turn light on and off
@app.callback(
    Output('light-img', 'src'),
    Input('light-switch', 'on')
)
def update_button(on):

    # print(on)
    # print('-------------------------------------LED info------------------------------------------')
    if on == True:
        led.setupLEDState(True)
        return img_light_on

    else:
        led.setupLEDState(False)
        return img_light_off
```

Phase 2 activities:

Method and Solution:

The goal for phase 2 was to capture the temperature and humidity of the room using a DHT11 sensor and display the data on the dashboard. We created two gauges on the dashboard where we display the data from both the temperature and humidity. Furthermore, if the temperature surpasses the temperature threshold, which was hardcoded to 24°C at this time, the system sends an email to the user. The email contains the temperature and a question asking them if they would like to open the fan. If the user were to reply "yes", the fan would open. Otherwise the fan will remain off.


python01100922@gmail.com now
4gqKQAPNNwQPBj7I
The current temperature is 24.8°C. Would you like to turn on the fan? Yes/No

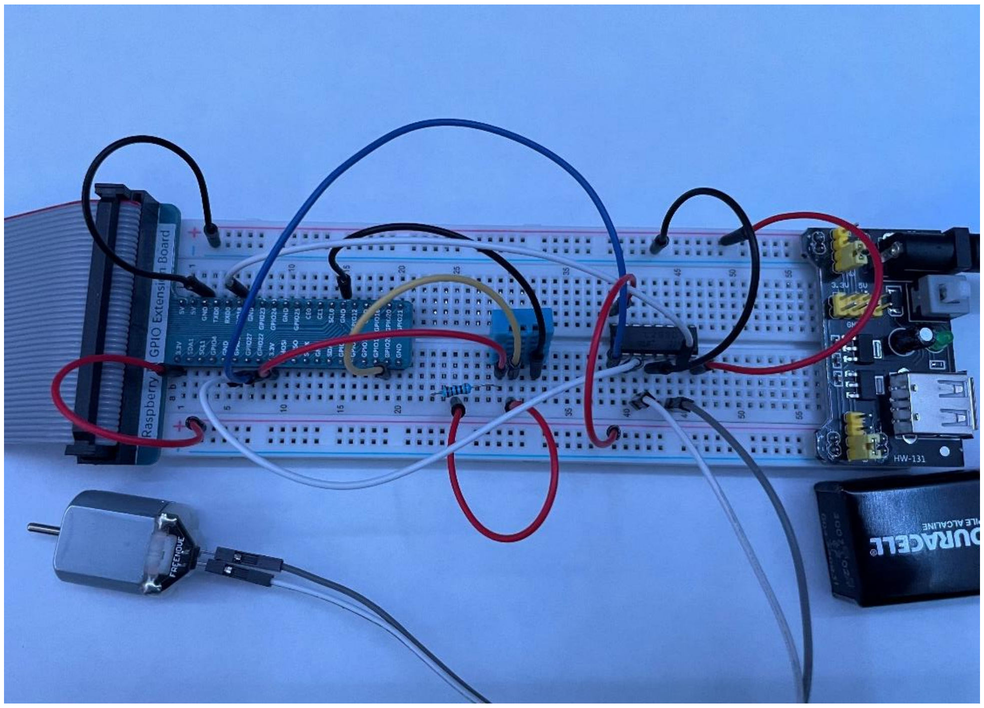
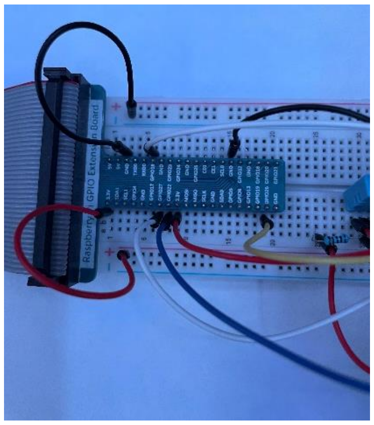
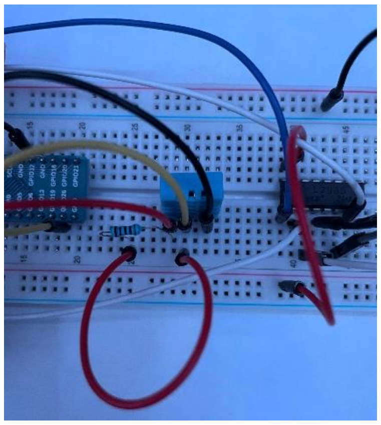
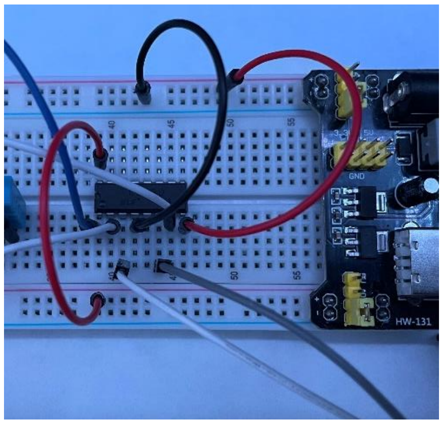


Figure 1: Phase 2 Wiring

Components and Libraries

Components introduced:

- DHT11 Temperature and Humidity sensor
- Resistors (10k Ω)
- DC Motor
- L293D motor driver
- HW-131 for power supply
- 9v battery for power supply
- Wires

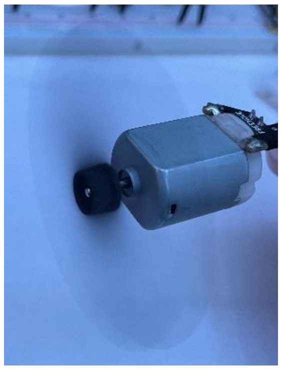
Libraries Imported:

- dash_daq
- Freenove_DHT
- smtplib (send emails)
- imaplib (access emails)
- email (manage email messages)
- secrets
- string
- time

Breadboard:

On the breadboard, we have multiple new components. We have a DHT11 with a 10k Resistor for capturing the temperature and the humidity of the room. Also, we have a L293D motor driver specifically used to control the DC motor. The DC motor will act as a fan, and it will require some male-to-female wires.

Like I previously mentioned, when the temperature goes above the threshold, which is 24°C, the user will receive an email asking them “Would you like to turn on the fan?”. If the user replies “yes”, the DC motor will start to spin. Otherwise, it will remain off.



DHT11 Pins Connection		
#	**Name:**	**Connection to RPI GPIO pins:**
1	VCC	3.3v
2	Data	GPIO26
3	NC	
4	GND	GND

View **Figure 1: Phase 2 Wiring**

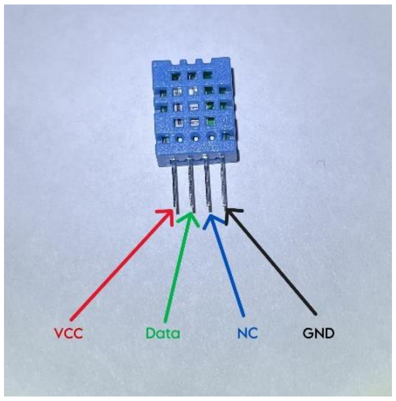


Figure 2: DHT11

L293D Pins Connection		
#	**Name:**	**Connection to RPI GPIO pins and HW-131:**
1	EN1	GPIO22
2	IN1	GPIO27
3	OUT1	DC Motor pin (one of the two pins)
4	0V	GND
6	OUT2	DC Motor pin (pin that is free)
7	IN2	GPIO18
8	Vs	On HW-131 LEFT Jumper side (+)
16	Vss	On HW-131 RIGHT Jumper side (+)

View **Figure 1: Phase 2 Wiring**

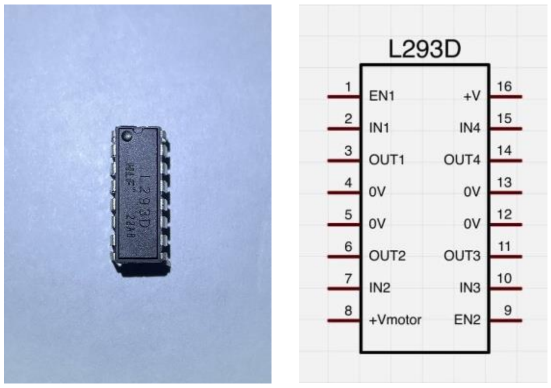


Figure 3: L293D

Figure 4: Power Supply

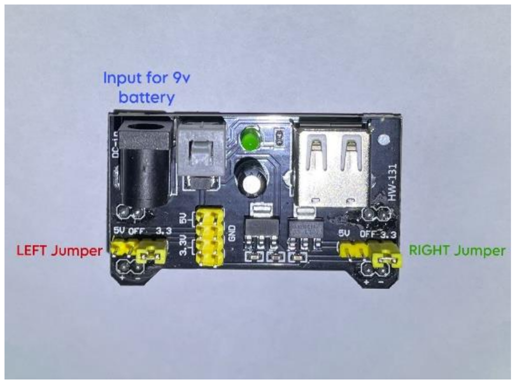


Figure 5: HW-131

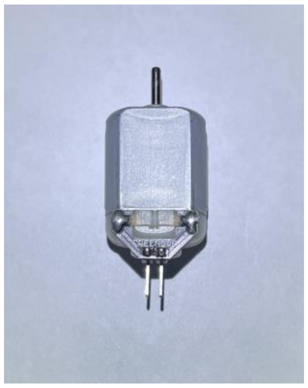


Figure 6: DC motor

Dashboard:

On the dashboard, we are using a Dash component called "Interval" which updates the app in real-time without having to reload the page. The purpose of this component is to update the data on the screen which is being captured from the DHT11 sensor. Once, the temperature goes above the threshold, and that the conditions for opening the fan are met, the image representing the Fan on the dashboard will start to spin.

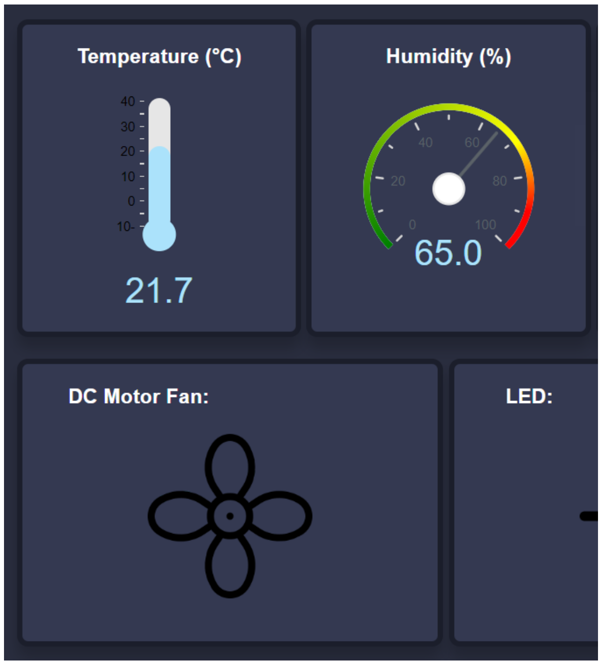


Figure 7: Before surpassing Temp threshold

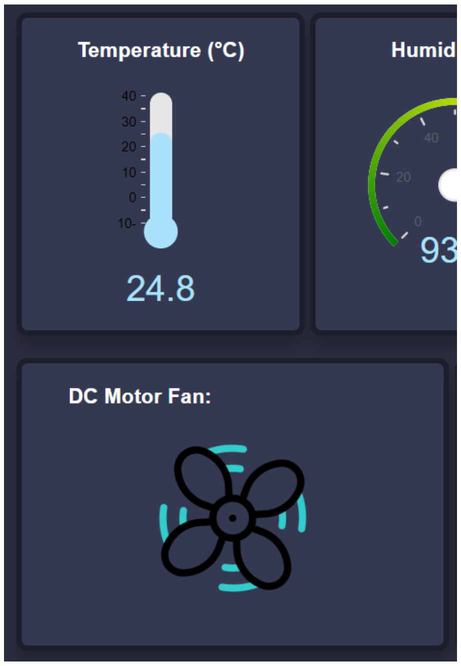


Figure 8: After surpassing Temp threshold

Breakdown of the code:

Just like the **LED.py** file, we also have a **DCMotor.py** file which is responsible for manipulating the state of the DC motor. In the **app.py**, we imported the DCMotor file containing all the necessary imports and commands to manipulate the Motor. Furthermore, we also imported the **Freenove_DHT** file which contains everything we need to capture the data from the DHT11 sensor. This can be found online.

- Freenove_DHT.py link: https://github.com/Freenove/Freenove_RFID_Starter_Kit_for_Raspberry_Pi/blob/master/Code/Python_Code/21.1.1_DHT11/Freenove_DHT.py

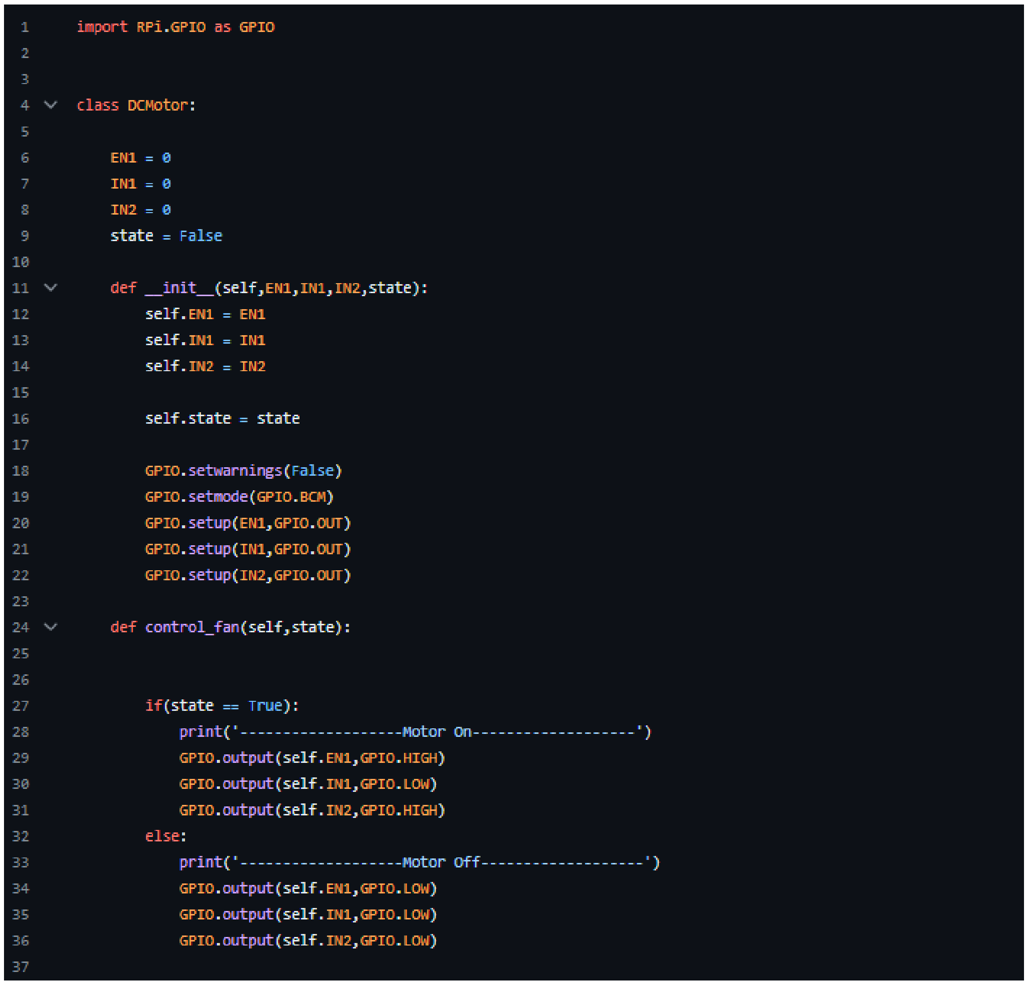
```
from DCMotor import DCMotor
import Freenove_DHT as DHT
```

The component responsible for reloading the page will be written as followed in the **app.py** file along with gauges and other components:

```
dcc.Interval(id='refresh', interval=3*1000,n_intervals=0)
```

Interval=3*1000 simply means that the page will reload every 3 seconds

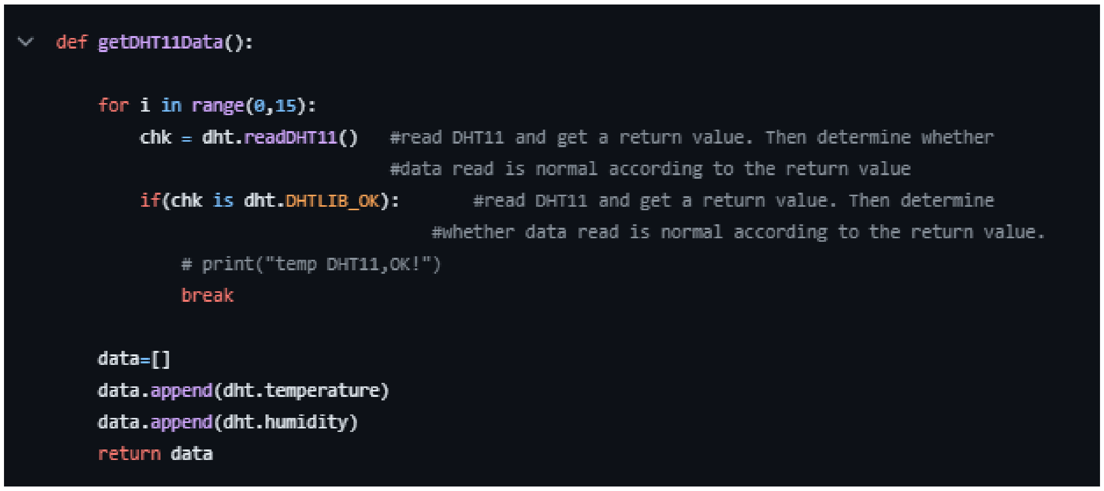
DCMotor.py



```python
import RPi.GPIO as GPIO


class DCMotor:

    EN1 = 0
    IN1 = 0
    IN2 = 0
    state = False

    def __init__(self,EN1,IN1,IN2,state):
        self.EN1 = EN1
        self.IN1 = IN1
        self.IN2 = IN2

        self.state = state

        GPIO.setwarnings(False)
        GPIO.setmode(GPIO.BCM)
        GPIO.setup(EN1,GPIO.OUT)
        GPIO.setup(IN1,GPIO.OUT)
        GPIO.setup(IN2,GPIO.OUT)

    def control_fan(self,state):


        if(state == True):
            print('------------------Motor On------------------')
            GPIO.output(self.EN1,GPIO.HIGH)
            GPIO.output(self.IN1,GPIO.LOW)
            GPIO.output(self.IN2,GPIO.HIGH)
        else:
            print('------------------Motor Off------------------')
            GPIO.output(self.EN1,GPIO.LOW)
            GPIO.output(self.IN1,GPIO.LOW)
            GPIO.output(self.IN2,GPIO.LOW)

```

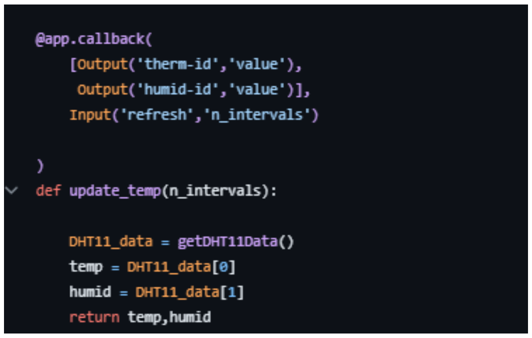
app.py

getDHT11Data is the function that uses the Freenove_DHT components to capture the data from the DHT11 sensor. The data is placed in an array before being returned.



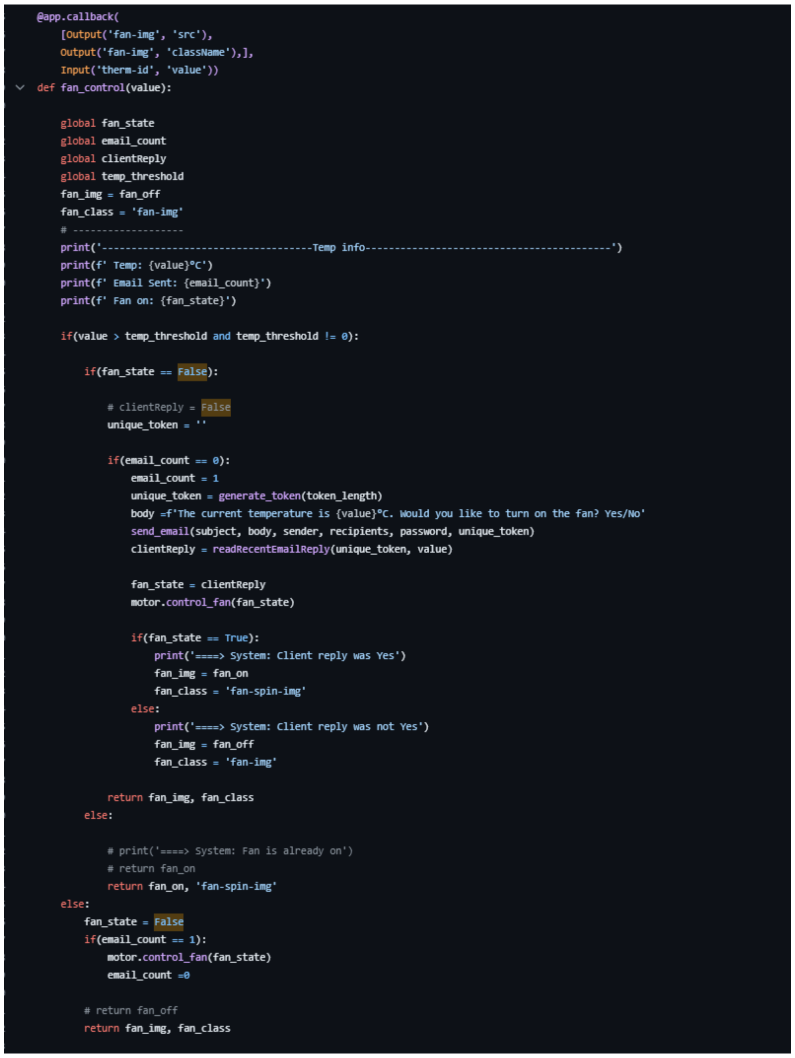
```python
def getDHT11Data():

    for i in range(0,15):
        chk = dht.readDHT11()   #read DHT11 and get a return value. Then determine whether
                                #data read is normal according to the return value
        if(chk is dht.DHTLIB_OK):      #read DHT11 and get a return value. Then determine
                                  #whether data read is normal according to the return value.
            # print("temp DHT11,OK!")
            break

    data=[]
    data.append(dht.temperature)
    data.append(dht.humidity)
    return data
```

update_temp uses the Interval as an input, meaning during every reload, this function will be called. It outputs the data captured from the DHT11 sensor.



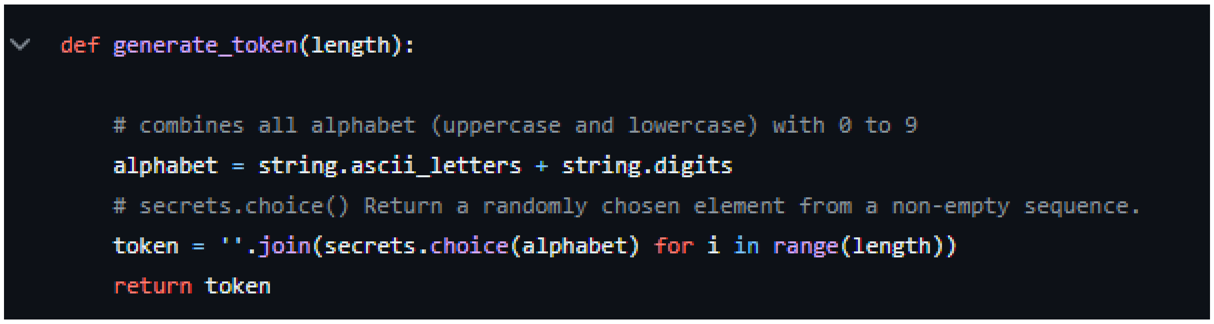
```python
@app.callback(
    [Output('therm-id','value'),
     Output('humid-id','value')],
    Input('refresh','n_intervals')

)
def update_temp(n_intervals):

    DHT11_data = getDHT11Data()
    temp = DHT11_data[0]
    humid = DHT11_data[1]
    return temp,humid
```

fan_control uses the thermometer value as an input. It checks the value and makes a decision whether to send an email to the user or not. Before sending an email, we also check whether the fan is already on or not, so that we don't spam the user with emails. We also use a counter to make sure an email is only sent once. Once the email is sent, the system will wait for a reply. After that, it will make a decision to open the fan or leave the fan off.



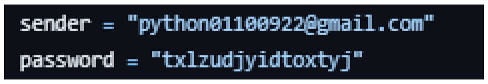
```python
@app.callback(
    [Output('fan-img', 'src'),
    Output('fan-img', 'className'),],
    Input('therm-id', 'value'))
def fan_control(value):

    global fan_state
    global email_count
    global clientReply
    global temp_threshold
    fan_img = fan_off
    fan_class = 'fan-img'
    # ------------------
    print('-----------------------------------Temp info-----------------------------------------')
    print(f' Temp: {value}°C')
    print(f' Email Sent: {email_count}')
    print(f' Fan on: {fan_state}')

    if(value > temp_threshold and temp_threshold != 0):

        if(fan_state == False):

            # clientReply = False
            unique_token = ''

            if(email_count == 0):
                email_count = 1
                unique_token = generate_token(token_length)
                body =f'The current temperature is {value}°C. Would you like to turn on the fan? Yes/No'
                send_email(subject, body, sender, recipients, password, unique_token)
                clientReply = readRecentEmailReply(unique_token, value)

                fan_state = clientReply
                motor.control_fan(fan_state)

                if(fan_state == True):
                    print('====> System: Client reply was Yes')
                    fan_img = fan_on
                    fan_class = 'fan-spin-img'
                else:
                    print('====> System: Client reply was not Yes')
                    fan_img = fan_off
                    fan_class = 'fan-img'

            return fan_img, fan_class
        else:

            # print('====> System: Fan is already on')
            # return fan_on
            return fan_on, 'fan-spin-img'
    else:
        fan_state = False
        if(email_count == 1):
            motor.control_fan(fan_state)
            email_count =0

        # return fan_off
        return fan_img, fan_class
```

generate_token is used to make sure that when the system reads the emails, it will search and read the specific email containing the same token that was sent as a subject.

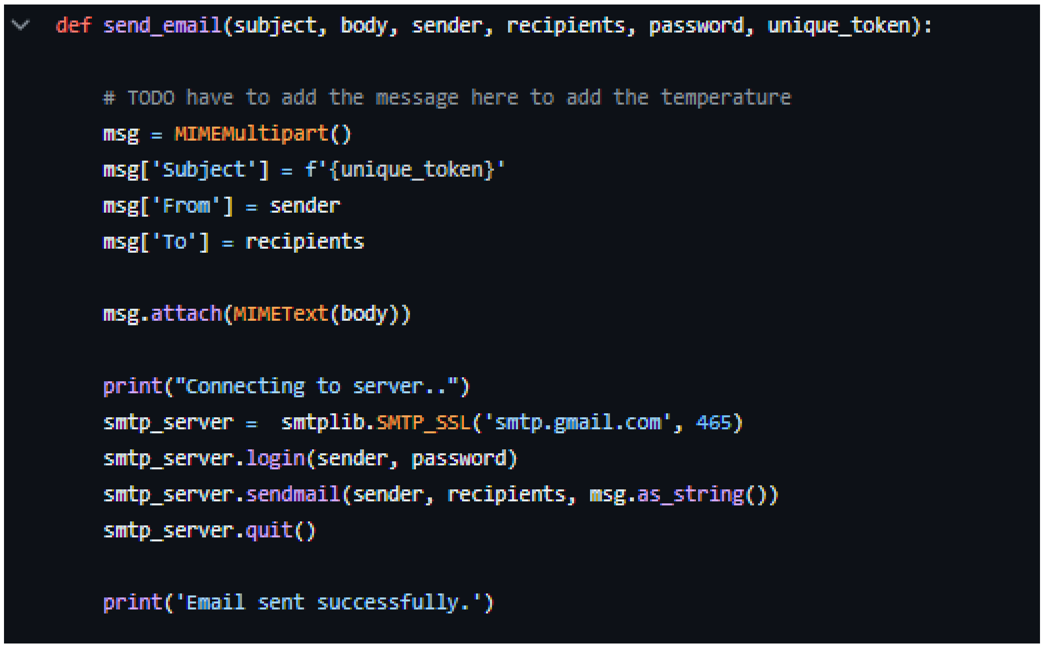


```python
def generate_token(length):

    # combines all alphabet (uppercase and lowercase) with 0 to 9
    alphabet = string.ascii_letters + string.digits
    # secrets.choice() Return a randomly chosen element from a non-empty sequence.
    token = ''.join(secrets.choice(alphabet) for i in range(length))
    return token
```

send_email uses the **smtplib library** to login into a specific google account using the email address and password generated with googles 2-Step Verification. After login in, we simply send the email and logout.

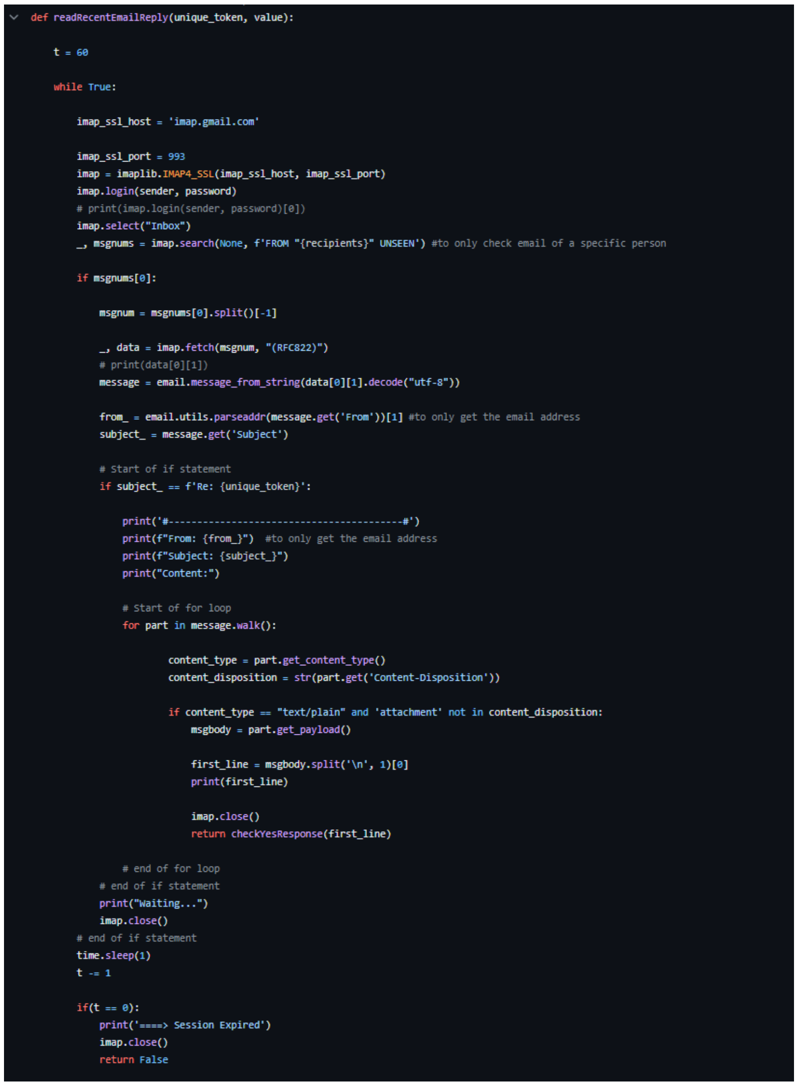


```python
sender = "python01100922@gmail.com"
password = "txlzudjyidtoxtyj"
```



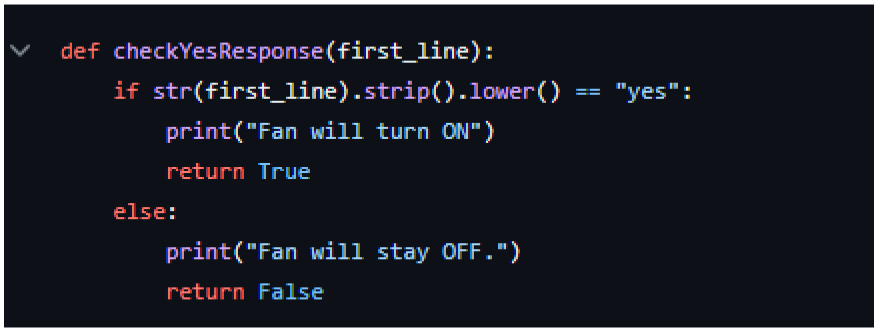
```python
def send_email(subject, body, sender, recipients, password, unique_token):

    # TODO have to add the message here to add the temperature
    msg = MIMEMultipart()
    msg['Subject'] = f'{unique_token}'
    msg['From'] = sender
    msg['To'] = recipients

    msg.attach(MIMEText(body))

    print("Connecting to server..")
    smtp_server =  smtplib.SMTP_SSL('smtp.gmail.com', 465)
    smtp_server.login(sender, password)
    smtp_server.sendmail(sender, recipients, msg.as_string())
    smtp_server.quit()

    print('Email sent successfully.')
```

readRecentEmaiReply uses **imaplib library** to read the most recent email sent by the user using the same token that was generated while sending the email to the user. it only reads the first line of the email. Furthermore, the user only has 60 seconds to reply to the email. If they do not reply in time, the function will return false, leaving the fan off.



```python
def readRecentEmailReply(unique_token, value):

    t = 60

    while True:

        imap_ssl_host = 'imap.gmail.com'

        imap_ssl_port = 993
        imap = imaplib.IMAP4_SSL(imap_ssl_host, imap_ssl_port)
        imap.login(sender, password)
        # print(imap.login(sender, password)[0])
        imap.select("Inbox")
        _, msgnums = imap.search(None, f'FROM "{recipients}" UNSEEN') #to only check email of a specific person

        if msgnums[0]:

            msgnum = msgnums[0].split()[-1]

            _, data = imap.fetch(msgnum, "(RFC822)")
            # print(data[0][1])
            message = email.message_from_string(data[0][1].decode("utf-8"))

            from_ = email.utils.parseaddr(message.get('From'))[1] #to only get the email address
            subject_ = message.get('Subject')

            # Start of if statement
            if subject_ == f'Re: {unique_token}':

                print('#---------------------------------------#')
                print(f"From: {from_}")  #to only get the email address
                print(f"Subject: {subject_}")
                print("Content:")

                # Start of for loop
                for part in message.walk():

                        content_type = part.get_content_type()
                        content_disposition = str(part.get('Content-Disposition'))

                        if content_type == "text/plain" and 'attachment' not in content_disposition:
                            msgbody = part.get_payload()

                            first_line = msgbody.split('\n', 1)[0]
                            print(first_line)

                            imap.close()
                            return checkYesResponse(first_line)

                # end of for loop
            # end of if statement
            print("Waiting...")
            imap.close()
        # end of if statement
        time.sleep(1)
        t -= 1

        if(t == 0):
            print('====> Session Expired')
            imap.close()
            return False
```

checkYesResponse is simply used to make sure we strip the message and remove any black spaces. It also sets the users response to lower case. If its "yes", we return true, otherwise we return false.

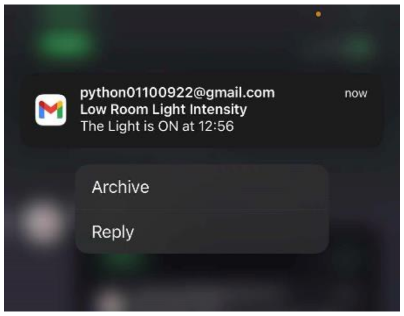


```python
def checkYesResponse(first_line):
    if str(first_line).strip().lower() == "yes":
        print("Fan will turn ON")
        return True
    else:
        print("Fan will stay OFF.")
        return False
```

Phase 3 activities:

Method and Solution:

In phase 3, the goal was to use the same LED used in phase 1, but this time we disable the switch and change the state of the LED depending on the light intensity of the room. In short, we implemented a smart LED system. The light intensity is being captured using a Photoresistor sensor connected to a ESP8266 board. The data is being exchanged with the application using the MQTT publish/subscribe communication protocol. When the light intensity goes below the set threshold, the LED will turn on and an email will be sent to the user. Furthermore, the data must be displayed on the dashboard and update when the light intensity changes. Finally, an email will be sent to the user to let them know the LED is on.



python01100922@gmail.com now
Low Room Light Intensity
The Light is ON at 12:56

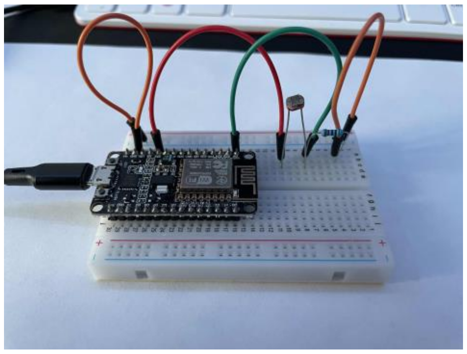


Figure 9: ESP8266 and Photoresistor Connection

Components and Libraries:

Components used:

- LED
- Photoresistor
- ESP8266
- 1x Resistor (10kΩ)
- 1x Resistor (220Ω)
- Wires

Libraries imported:

Python:

- paho.mqtt.client

Arduino IDE:

- ESP8266WiFi.h
- PubSubClient.h
- Arduino.h

Breadboard:

On the breadboard, we are introducing the ESP8266 board and Photoresistor, which, if needed, can be placed on a separate breadboard. The Photoresistor requires a 10kΩ Resistor and some wires to connect to the ESP8266. In our case, the ESP8266 board requires a cable for it to be connected to the Raspberry Pi. The LED will only turn on when the light intensity is below the threshold. If the intensity goes back up, the LED will turn off.

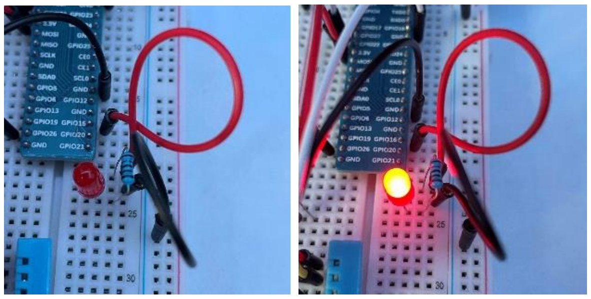


Figure 10: LED Wiring

LED Pins Connection	
Lead Pins:	RPi GPIO pins:
Long lead (+)	GPIO16 with 220Ω resistor
Short lead (-)	GND

View **Figure 10** for wiring

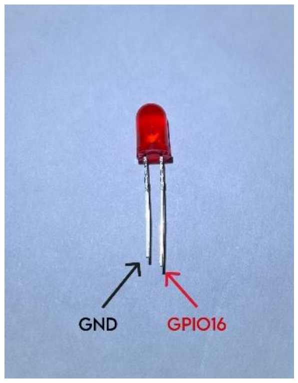


Figure 11: LED

Pin Connection to ESP8266 board	
ESP8266 Pins:	Components:
GND	10kΩ Resistor (to one pin)
3.3v	Photoresistor (to one pin)
A0	Photoresistor and 10kΩ Resistor

View **Figure 9** for wiring

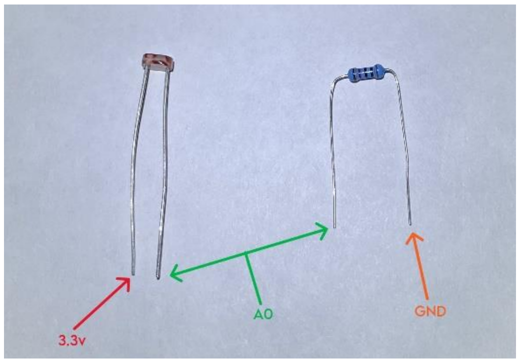


Figure 12: Photoresistor and 10k Resistor

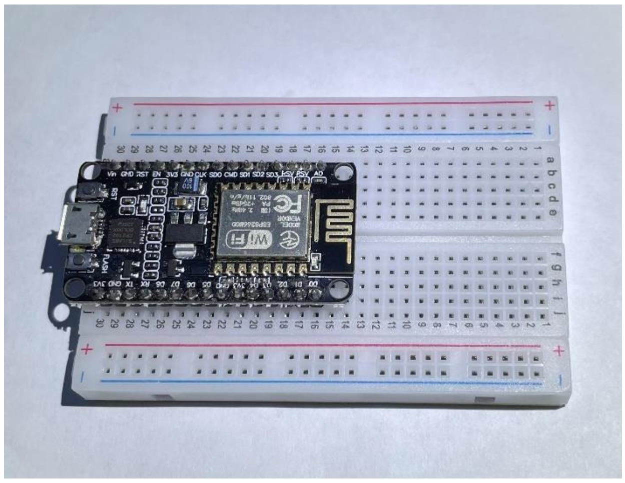


Figure 14: ESP8266 Board

Figure 13: Cable

Dashboard:

On the dashboard, we are using the same “Interval” component from phase 2 to update the light intensity value. The Sun and the Slider represents the light intensity. When the intensity goes down, the Sun gets darker and the Slider value will go down. When the intensity goes under the threshold, the condition for turning the LED on will be met, and the image representing the LED will switch to a lit up LED. The switch will also change depending on the state of the LED. However, the user does not have access to the switch.

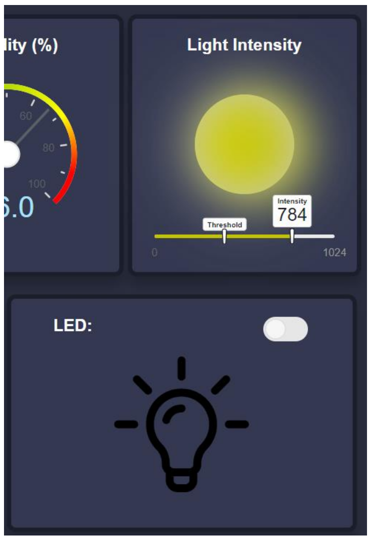


Figure 15: above intensity threshold

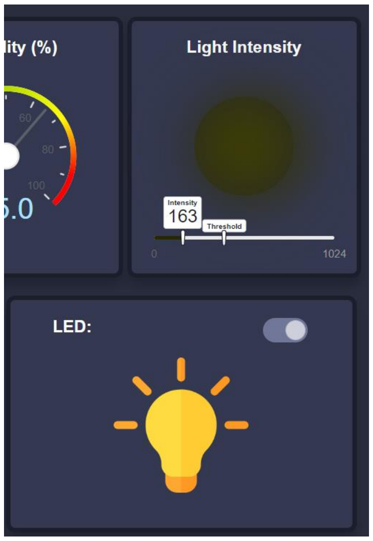


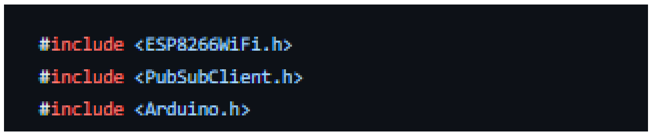
Figure 16: below intensity threshold

Breakdown of the code:

The MQTT connection will all be done inside of the **IoTController.py** file and **MQTT_For_Project.ino** file. In the **app.py**, we will instantiate and subscribe to the broker using the Broker address and the Topic. In our case, the IP address of the broker is the IP address given to the Raspberry Pi.

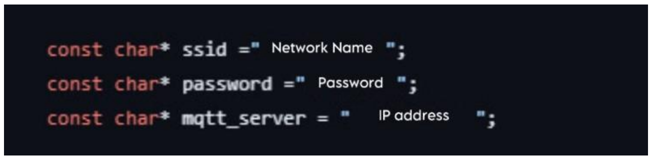
Before even subscribing to the broker, we must first publish the light intensity data to the broker with a topic, such as “**ESP8266/Photoresister**”. This will be done on Arduino IDE in a file called **MQTT_For_Project.ino.**

MQTT_For_Project.ino



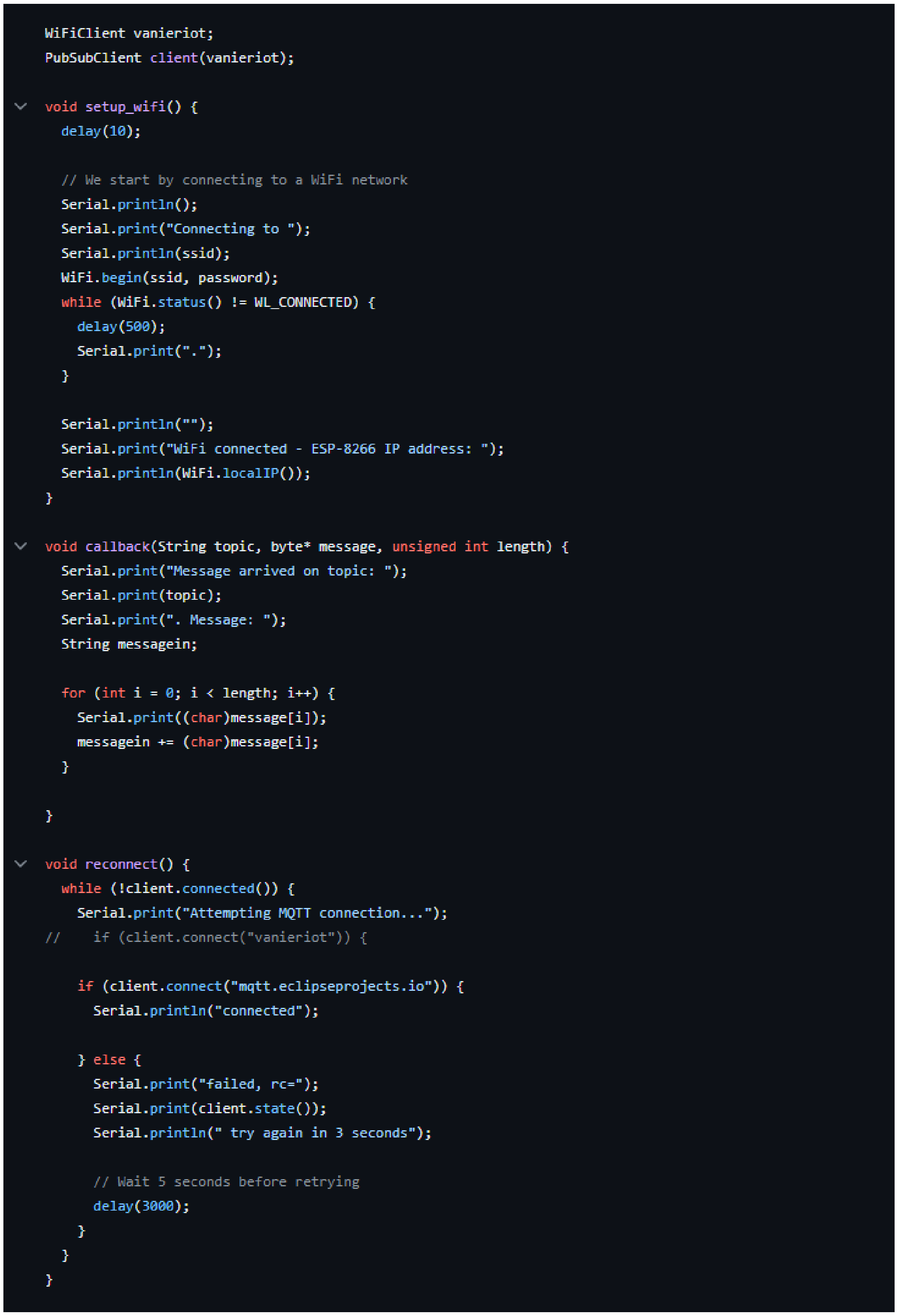
```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <Arduino.h>
```

The **mqtt_server** IP address must be the same as when the client is subscribing to the broker.

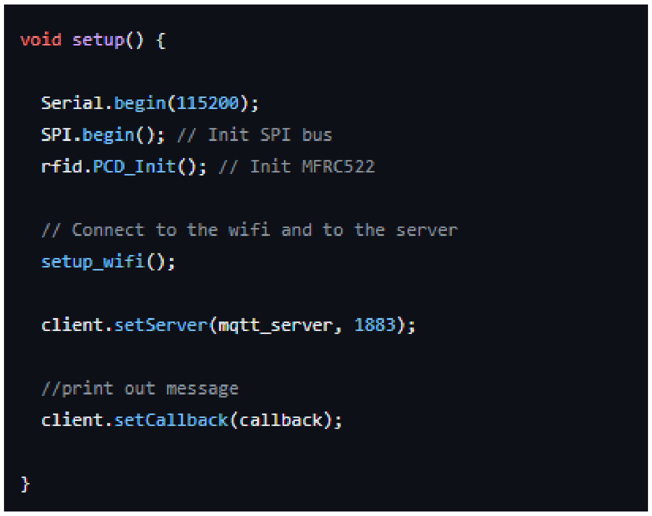


```
const char* ssid =" Network Name ";
const char* password =" Password ";
const char* mqtt_server = " IP address ";
```

These functions will be called to establish a connection with the broker. However, before that, the system must connect to the network. Make sure you connect to the same network in both the Raspberry Pi and the ESP8266. Otherwise, the message exchange will not work.

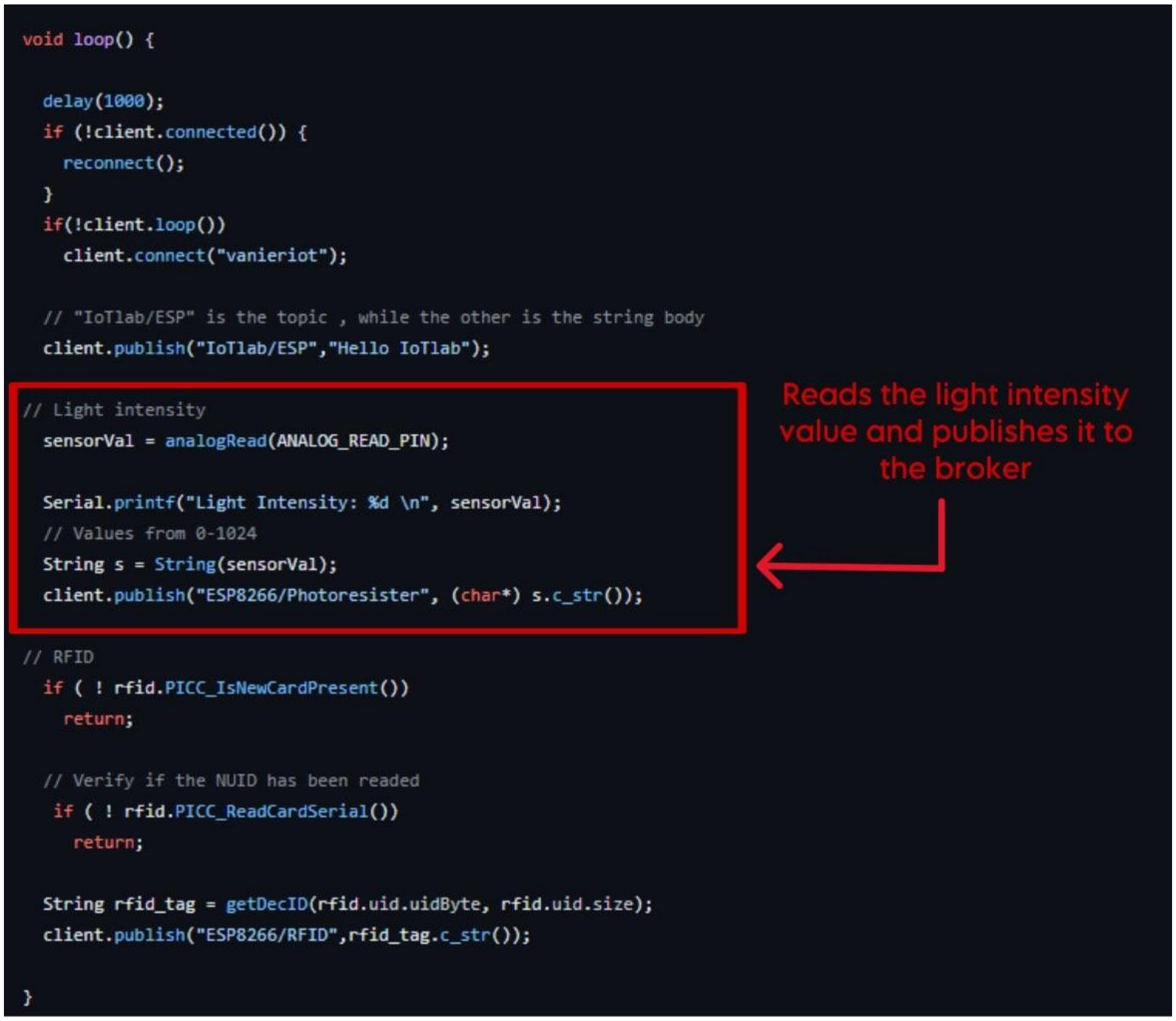


```
WiFiClient vanieriot;
PubSubClient client(vanieriot);

void setup_wifi() {
  delay(10);

  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.print("WiFi connected - ESP-8266 IP address: ");
  Serial.println(WiFi.localIP());
}

void callback(String topic, byte* message, unsigned int length) {
  Serial.print("Message arrived on topic: ");
  Serial.print(topic);
  Serial.print(". Message: ");
  String messagein;

  for (int i = 0; i < length; i++) {
    Serial.print((char)message[i]);
    messagein += (char)message[i];
  }

}

void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
//    if (client.connect("vanieriot")) {

    if (client.connect("mqtt.eclipseprojects.io")) {
      Serial.println("connected");

    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 3 seconds");

      // Wait 5 seconds before retrying
      delay(3000);
    }
  }
}
```



```
void setup() {

  Serial.begin(115200);
  SPI.begin(); // Init SPI bus
  rfid.PCD_Init(); // Init MFRC522

  // Connect to the wifi and to the server
  setup_wifi();

  client.setServer(mqtt_server, 1883);

  //print out message
  client.setCallback(callback);

}
```

In the **loop**, If the client is not connected to the network, it will keep trying to reconnect until the connection is done. If, the connection does not work, the system will not be able to publish the data.

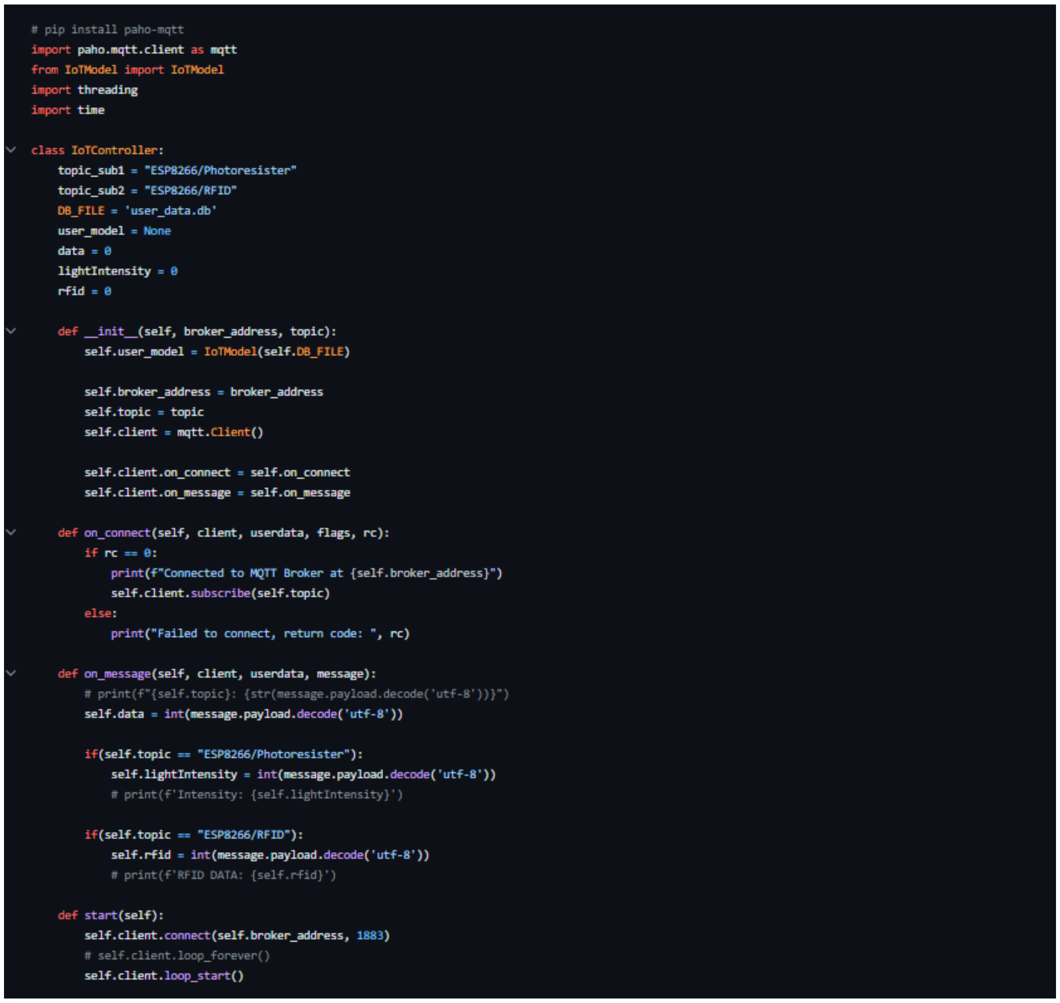


```
void loop() {

  delay(1000);
  if (!client.connected()) {
    reconnect();
  }
  if(!client.loop())
    client.connect("vanieriot");

  // "IoTlab/ESP" is the topic , while the other is the string body
  client.publish("IoTlab/ESP","Hello IoTlab");

// Light intensity
  sensorVal = analogRead(ANALOG_READ_PIN);

  Serial.printf("Light Intensity: %d \n", sensorVal);
  // Values from 0-1024
  String s = String(sensorVal);
  client.publish("ESP8266/Photoresister", (char*) s.c_str());

// RFID
  if ( ! rfid.PICC_IsNewCardPresent())
    return;

  // Verify if the NUID has been readed
   if ( ! rfid.PICC_ReadCardSerial())
     return;

  String rfid_tag = getDecID(rfid.uid.uidByte, rfid.uid.size);
  client.publish("ESP8266/RFID",rfid_tag.c_str());

}
```

Reads the light intensity value and publishes it to the broker

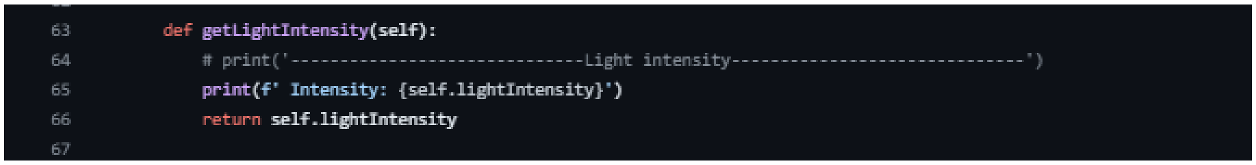
IoTController.py

In the **IoTController class**, once instantiated, the client will subscribe to the broker using the topic "**ESP8266/Photoresister**" (**on_connect**) and retrieve the message payload (**on_message**).



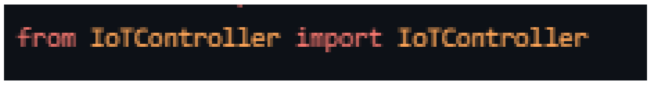
```python
# pip install paho-mqtt
import paho.mqtt.client as mqtt
from IoTModel import IoTModel
import threading
import time

class IoTController:
    topic_sub1 = "ESP8266/Photoresister"
    topic_sub2 = "ESP8266/RFID"
    DB_FILE = 'user_data.db'
    user_model = None
    data = 0
    lightIntensity = 0
    rfid = 0

    def __init__(self, broker_address, topic):
        self.user_model = IoTModel(self.DB_FILE)

        self.broker_address = broker_address
        self.topic = topic
        self.client = mqtt.Client()

        self.client.on_connect = self.on_connect
        self.client.on_message = self.on_message

    def on_connect(self, client, userdata, flags, rc):
        if rc == 0:
            print(f"Connected to MQTT Broker at {self.broker_address}")
            self.client.subscribe(self.topic)
        else:
            print("Failed to connect, return code: ", rc)

    def on_message(self, client, userdata, message):
        # print(f"{self.topic}: {str(message.payload.decode('utf-8'))}")
        self.data = int(message.payload.decode('utf-8'))

        if(self.topic == "ESP8266/Photoresister"):
            self.lightIntensity = int(message.payload.decode('utf-8'))
            # print(f'Intensity: {self.lightIntensity}')

        if(self.topic == "ESP8266/RFID"):
            self.rfid = int(message.payload.decode('utf-8'))
            # print(f'RFID DATA: {self.rfid}')

    def start(self):
        self.client.connect(self.broker_address, 1883)
        # self.client.loop_forever()
        self.client.loop_start()
```

getLightIntensity will be called on the **app.py** file to pass the data to the application.

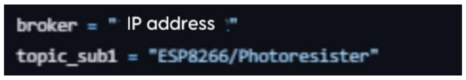


```python
    def getLightIntensity(self):
        # print('-----------------------------Light intensity-----------------------------')
        print(f' Intensity: {self.lightIntensity}')
        return self.lightIntensity
```

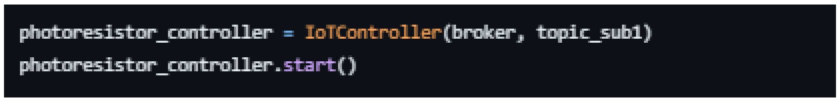
app.py



```python
from IoTController import IoTController
```

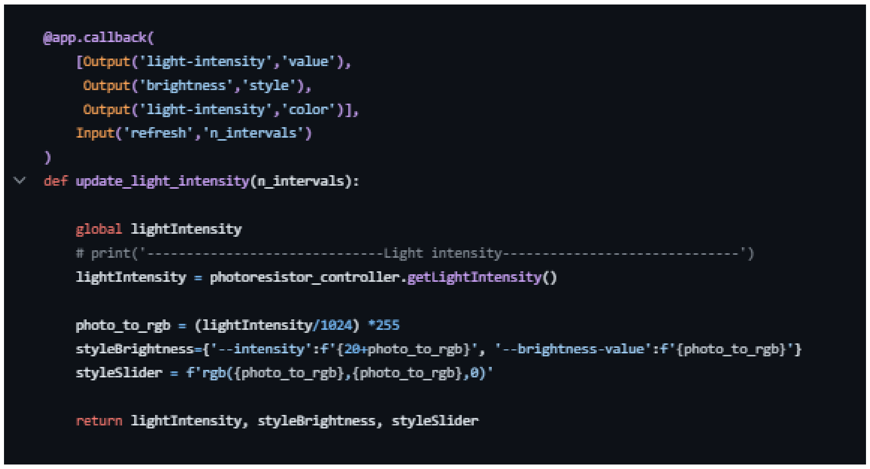


```python
broker = " IP address "
topic_sub1 = "ESP8266/Photoresister"
```



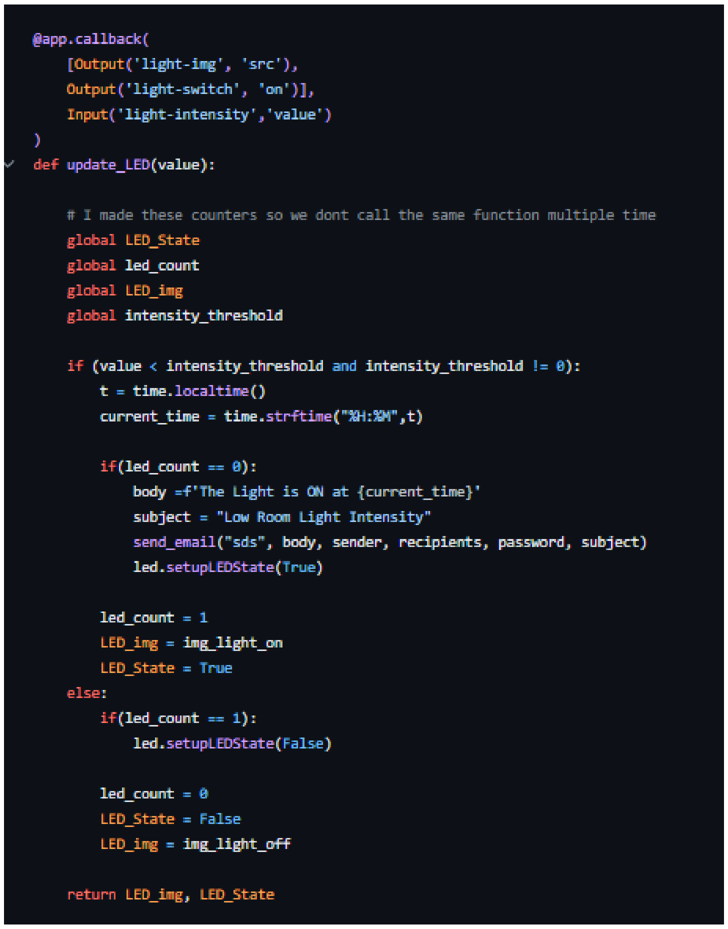
```python
photoresistor_controller = IoTController(broker, topic_sub1)
photoresistor_controller.start()
```

update_light_intensity retrieves the light intensity data from the **IoTController** and displays it on the dashboard along with the CSS for the Sun and Slider.



```
@app.callback(
    [Output('light-intensity','value'),
     Output('brightness','style'),
     Output('light-intensity','color')],
    Input('refresh','n_intervals')
)
def update_light_intensity(n_intervals):

    global lightIntensity
    # print('-----------------------------Light intensity-----------------------------')
    lightIntensity = photoresistor_controller.getLightIntensity()

    photo_to_rgb = (lightIntensity/1024) *255
    styleBrightness={'--intensity':f'{20+photo_to_rgb}', '--brightness-value':f'{photo_to_rgb}'}
    styleSlider = f'rgb({photo_to_rgb},{photo_to_rgb},0)'

    return lightIntensity, styleBrightness, styleSlider
```

update_LED looks at the value of the light intensity displayed on the dashboard and makes a decision whether to turn on or off the LED. If intensity goes below the threshold, the system will send an email to the user containing the time the LED opened. We also have a counter here to make sure the email is only sent once, and that the user does not get multiple emails at once.

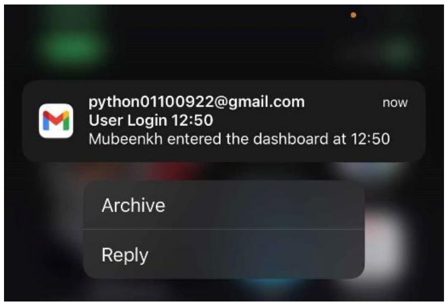


```
@app.callback(
    [Output('light-img', 'src'),
    Output('light-switch', 'on')],
    Input('light-intensity','value')
)
def update_LED(value):

    # I made these counters so we dont call the same function multiple time
    global LED_State
    global led_count
    global LED_img
    global intensity_threshold

    if (value < intensity_threshold and intensity_threshold != 0):
        t = time.localtime()
        current_time = time.strftime("%H:%M",t)

        if(led_count == 0):
            body =f'The Light is ON at {current_time}'
            subject = "Low Room Light Intensity"
            send_email("sds", body, sender, recipients, password, subject)
            led.setupLEDState(True)

        led_count = 1
        LED_img = img_light_on
        LED_State = True
    else:
        if(led_count == 1):
            led.setupLEDState(False)

        led_count = 0
        LED_State = False
        LED_img = img_light_off

    return LED_img, LED_State
```

Phase 4 activities:

Method and Solution:

In phase 4, the goal was to use RFID Reader to capture the RFID Tag's information and update the user's information on the dashboard. The information being updated will be the username, email, temperature threshold, humidity threshold, and light intensity threshold. This information will be store in a database table called **user**. If the user does not exist in the database, the information displayed on the dashboard will be set back to default. The data exchange between the devices will be done using the same MQTT communication protocol used in phase 3. The sensor publishes the data do the broker, and the application will subscribe to the topic of the broker. Furthermore, an email will be sent to the user who taps the RFID Reader, letting them know what time they entered the dashboard. This helps with security.


python01100922@gmail.com now
User Login 12:50
Mubeenkh entered the dashboard at 12:50

Components and Libraries:

Components used:

- RC522 RFID Module
- ESP8266
- RFID Tags
- Wires

Libraries imported:

Python:

- SQLite3

Arduino IDE:

- SPI.h
- MFRC522.h

Breadboard:

On the breadboard, we added the RFID Reader to the ESP8266. If you have space on your breadboard, you can place the reader on it, otherwise, use some male-to-female wires to connect the pins of the RFID Reader to the ESP8266 pins. Another component needed for this phase are the RFID Tags. Make sure the tags are readable by the RC522 RFID Module. It is recommended to use tags of type MIFARE-mini, MIFARE-1K, and MIFARE-4K for this system.

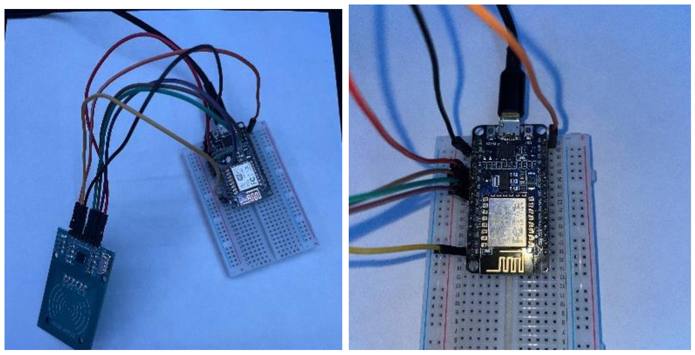


Figure 17: RFID Reader Wiring

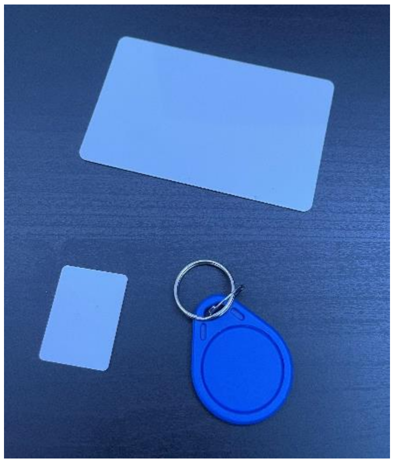


Figure 18: RFID Reader

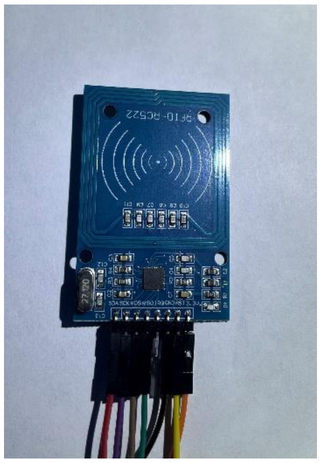


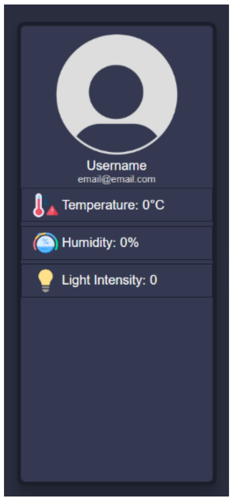
Figure 19: RFID Tags

Pin Connection to ESP8266 board	
ESP8266 board:	RC522 RFID Module:
D8	SDA
D5	SCK
D7	MOSI
D6	MISO
GND	GND
D0	RST
Vin	3.3v

Dashboard:

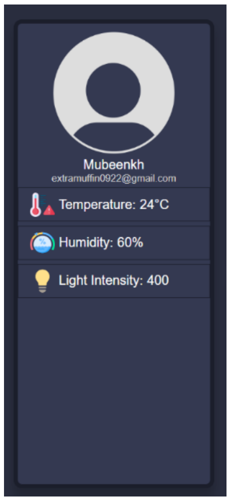
On the dashboard, the user information is set to default when no RFID Tag has been tapped. The information is also set to default when the user does not exist in the database. Once a Tag ID exist in the database, the data on the screen will change to the information of the user who is linked to that tag. The information being updated is the Username, the email address, and the thresholds for Temperature, Humidity, and Light Intensity. Furthermore, an email will be sent to the user of that Tag to let them know what time they entered the dashboard.

Figure 20: Default information



Username
email@email.com
Temperature: 0°C
Humidity: 0%
Light Intensity: 0

Figure 21: User's information



Mubeenkh
extramuffin0922@gmail.com
Temperature: 24°C
Humidity: 60%
Light Intensity: 400

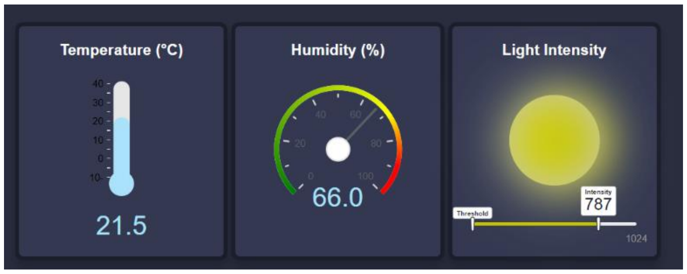


Figure 22: Default Light Intensity threshold

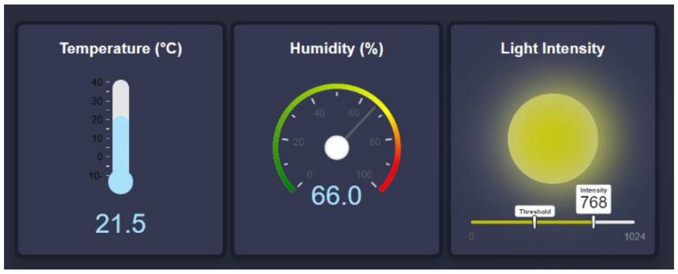


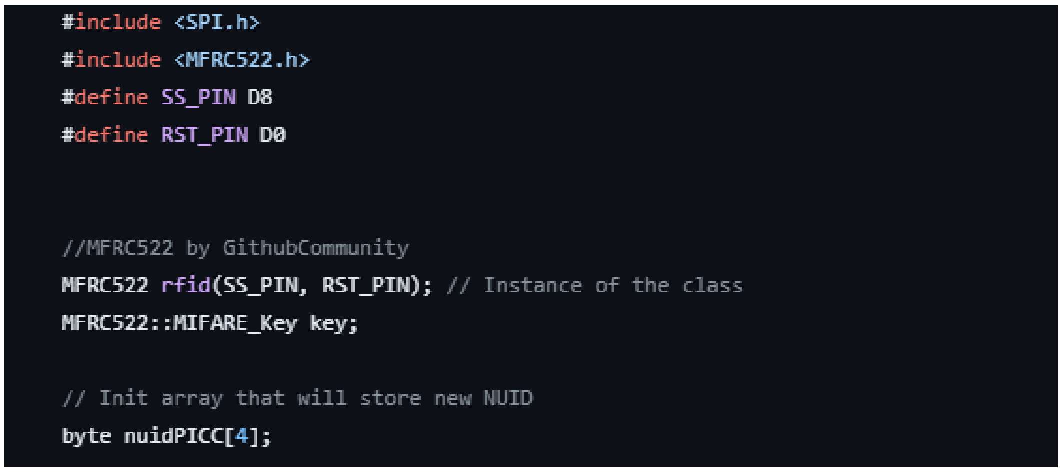
Figure 23: User's Light Intensity threshold

Breakdown of the code:

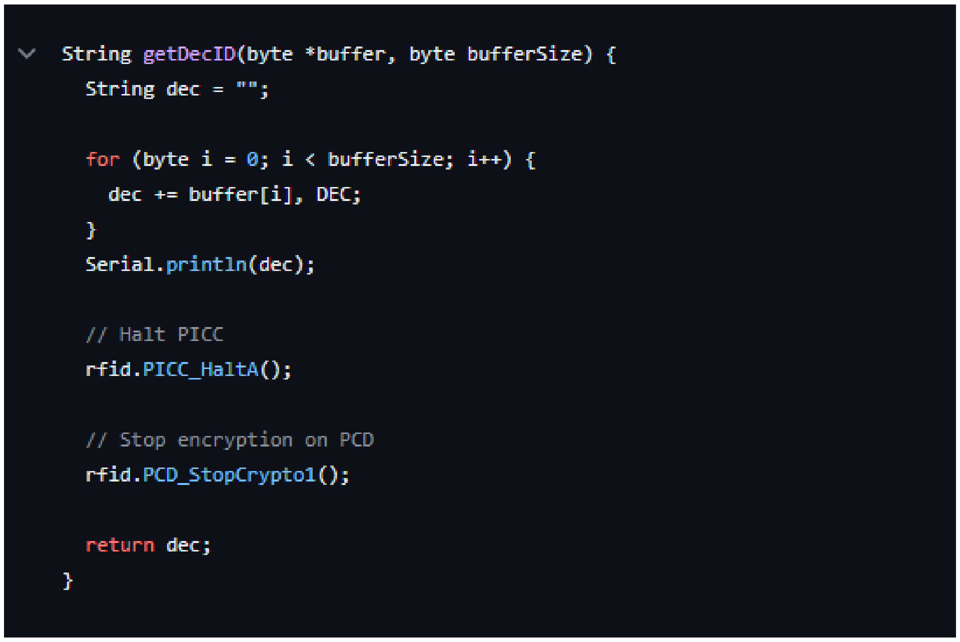
The MQTT Connection will be done in the same IoTController.py file as phase 3. However, this time, the IoTController will be responsible for subscribing to the broker and receiving the message payload, as well as communicating with the IoTModel.py file. The IoTModel is where communication with the database will occur. My idea was to separate the file by following the MVC framework. We have the IoTModel that gets the data from the database, the IoTController which communicates with the Model and the Broker to retrieve data. Finally, we have the app.py file which works like the "View". It get the data from the IoTController.

Once the RFID Tag has been read, the data will be published to the broker and the system will have to subscribe to the topic to check which card was recently tapped. The data will automatically update on the dashboard.

MQTT_For_Project.ino

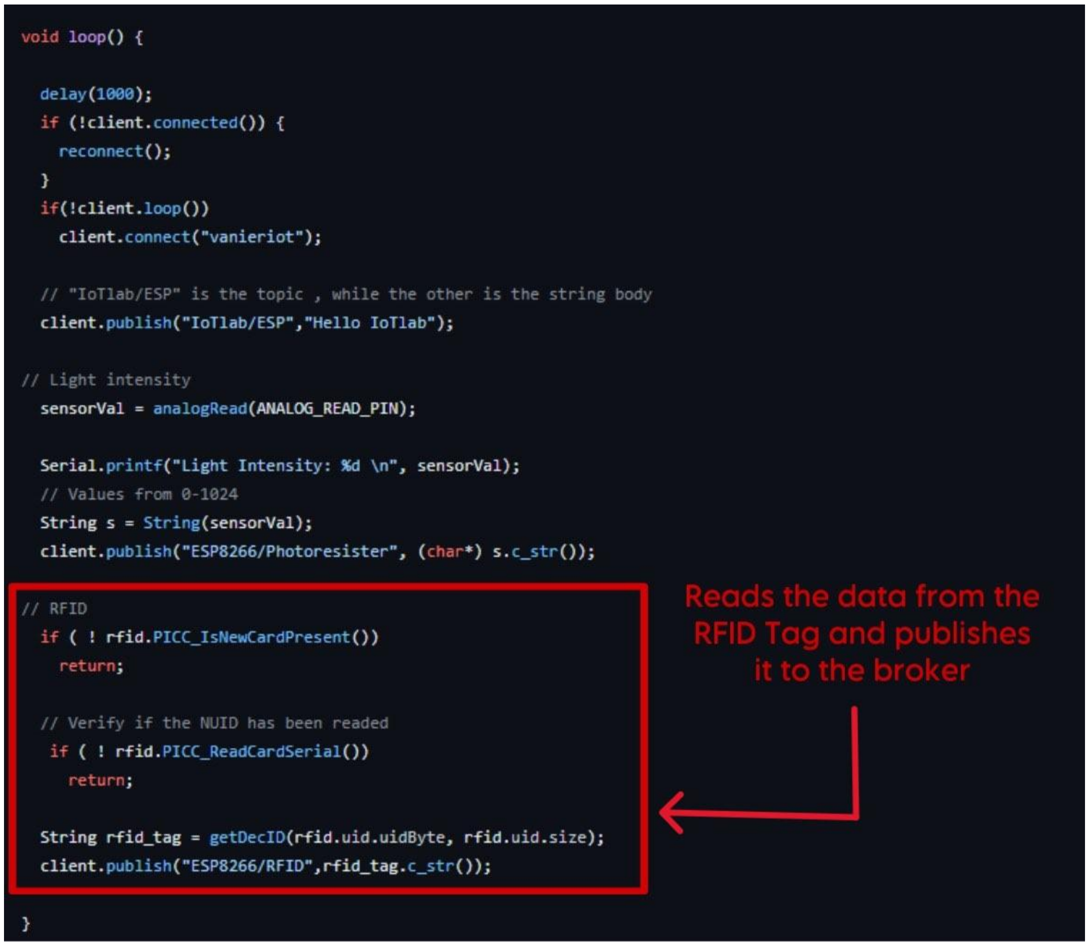


```
#include <SPI.h>
#include <MFRC522.h>
#define SS_PIN D8
#define RST_PIN D0


//MFRC522 by GithubCommunity
MFRC522 rfid(SS_PIN, RST_PIN); // Instance of the class
MFRC522::MIFARE_Key key;

// Init array that will store new NUID
byte nuidPICC[4];
```



```
String getDecID(byte *buffer, byte bufferSize) {
  String dec = "";

  for (byte i = 0; i < bufferSize; i++) {
    dec += buffer[i], DEC;
  }
  Serial.println(dec);

  // Halt PICC
  rfid.PICC_HaltA();

  // Stop encryption on PCD
  rfid.PCD_StopCrypto1();

  return dec;
}
```

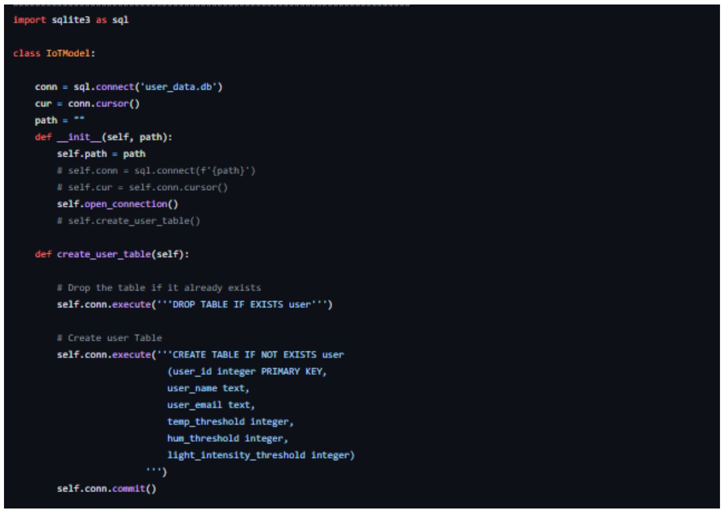
The data will be published to the Broker with the topic "**ESP8266/RFID**"


Reads the data from the RFID Tag and publishes it to the broker

```
void loop() {

  delay(1000);
  if (!client.connected()) {
    reconnect();
  }
  if(!client.loop())
    client.connect("vanieriot");

  // "IoTlab/ESP" is the topic , while the other is the string body
  client.publish("IoTlab/ESP","Hello IoTlab");

// Light intensity
  sensorVal = analogRead(ANALOG_READ_PIN);

  Serial.printf("Light Intensity: %d \n", sensorVal);
  // Values from 0-1024
  String s = String(sensorVal);
  client.publish("ESP8266/Photoresister", (char*) s.c_str());

// RFID
  if ( ! rfid.PICC_IsNewCardPresent())
    return;

  // Verify if the NUID has been readed
   if ( ! rfid.PICC_ReadCardSerial())
     return;

  String rfid_tag = getDecID(rfid.uid.uidByte, rfid.uid.size);
  client.publish("ESP8266/RFID",rfid_tag.c_str());

}
```

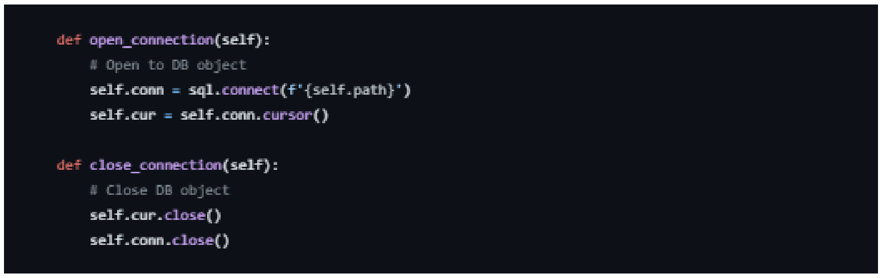
IoTModel.py

In the IoTModel file, we will mainly focus on the functions shown below. When the IoTModel is instantiated, you can decide to recreate the user table in case you already have an existing one. In our case, we commented it out because we already have the data inserted into the database.

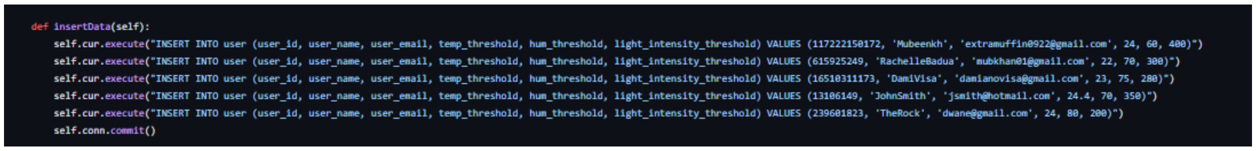


```python
import sqlite3 as sql

class IoTModel:

    conn = sql.connect('user_data.db')
    cur = conn.cursor()
    path = ""
    def __init__(self, path):
        self.path = path
        # self.conn = sql.connect(f'{path}')
        # self.cur = self.conn.cursor()
        self.open_connection()
        # self.create_user_table()

    def create_user_table(self):

        # Drop the table if it already exists
        self.conn.execute('''DROP TABLE IF EXISTS user''')

        # Create user Table
        self.conn.execute('''CREATE TABLE IF NOT EXISTS user
                          (user_id integer PRIMARY KEY,
                          user_name text,
                          user_email text,
                          temp_threshold integer,
                          hum_threshold integer,
                          light_intensity_threshold integer)
                        ''')
        self.conn.commit()
```

open_connection simply starts a connection with the database you are trying to access, while **close_connection** will simply close the connection.

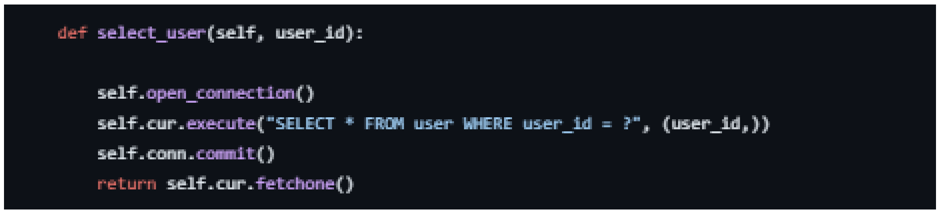


```python
    def open_connection(self):
        # Open to DB object
        self.conn = sql.connect(f'{self.path}')
        self.cur = self.conn.cursor()

    def close_connection(self):
        # Close DB object
        self.cur.close()
        self.conn.close()
```



```python
    def insertData(self):
        self.cur.execute("INSERT INTO user (user_id, user_name, user_email, temp_threshold, hum_threshold, light_intensity_threshold) VALUES (117222150172, 'Mubeenkh', 'extramuffin0922@gmail.com', 24, 60, 400)")
        self.cur.execute("INSERT INTO user (user_id, user_name, user_email, temp_threshold, hum_threshold, light_intensity_threshold) VALUES (615925249, 'RachelleBadua', 'mubkhan01@gmail.com', 22, 70, 300)")
        self.cur.execute("INSERT INTO user (user_id, user_name, user_email, temp_threshold, hum_threshold, light_intensity_threshold) VALUES (16510311173, 'DamiVisa', 'damianovisa@gmail.com', 23, 75, 280)")
        self.cur.execute("INSERT INTO user (user_id, user_name, user_email, temp_threshold, hum_threshold, light_intensity_threshold) VALUES (13106149, 'JohnSmith', 'jsmith@hotmail.com', 24.4, 70, 350)")
        self.cur.execute("INSERT INTO user (user_id, user_name, user_email, temp_threshold, hum_threshold, light_intensity_threshold) VALUES (239601823, 'TheRock', 'dwane@gmail.com', 24, 80, 200)")
        self.conn.commit()
```

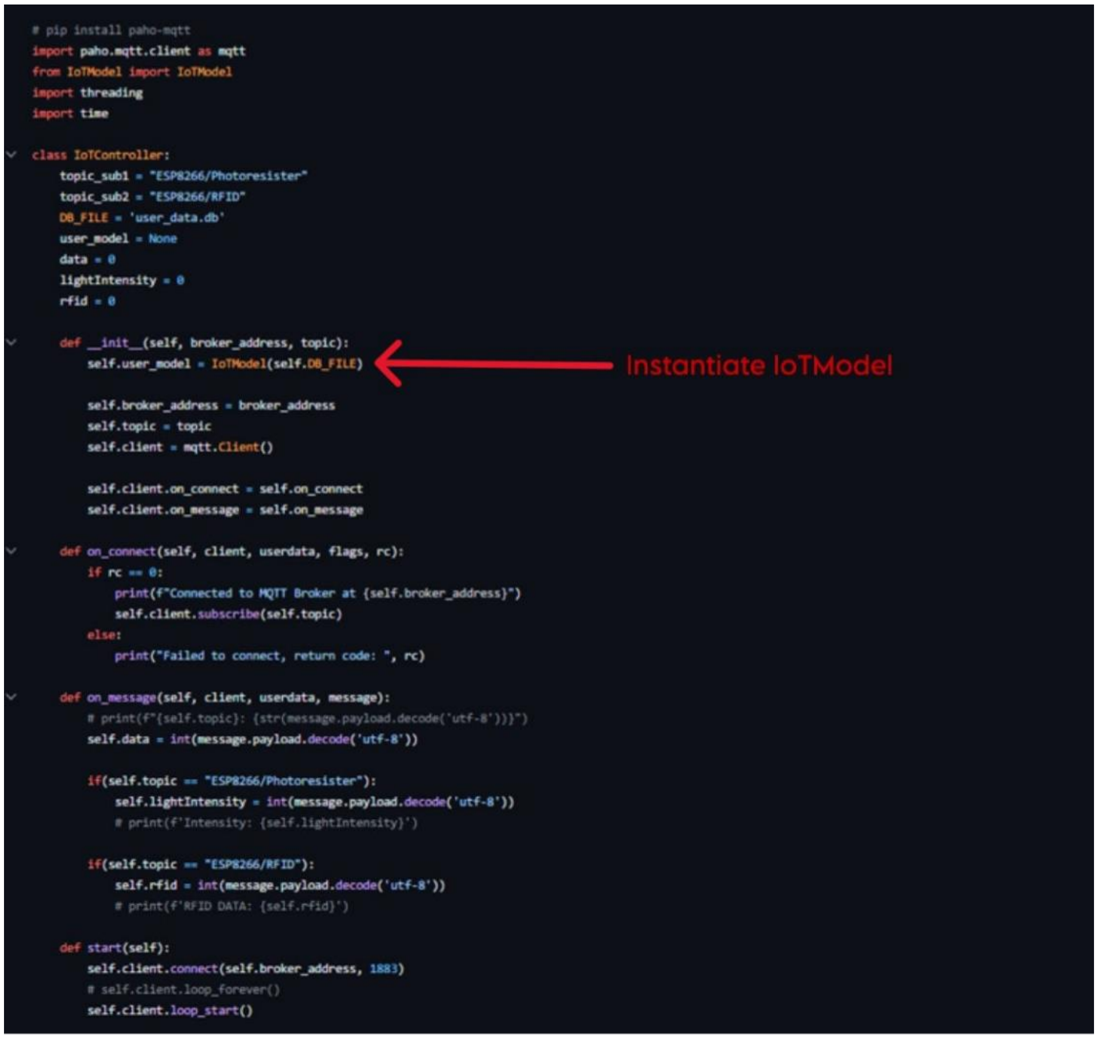
select_user will be called from the IoTController with the Tag ID as a parameter to retrieve the specific users information from the database using the Tag ID.



```python
    def select_user(self, user_id):

        self.open_connection()
        self.cur.execute("SELECT * FROM user WHERE user_id = ?", (user_id,))
        self.conn.commit()
        return self.cur.fetchone()
```

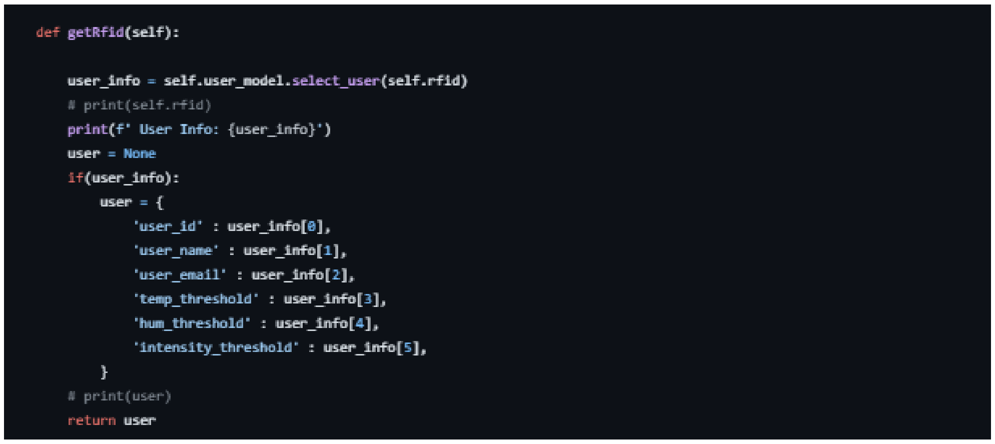
IoTController.py

This is the same code we have seen in the previous phase. However, one new function will be added to this class. This time, we will be subscribing to the broker using the topic “**ESP8266/RFID**”.



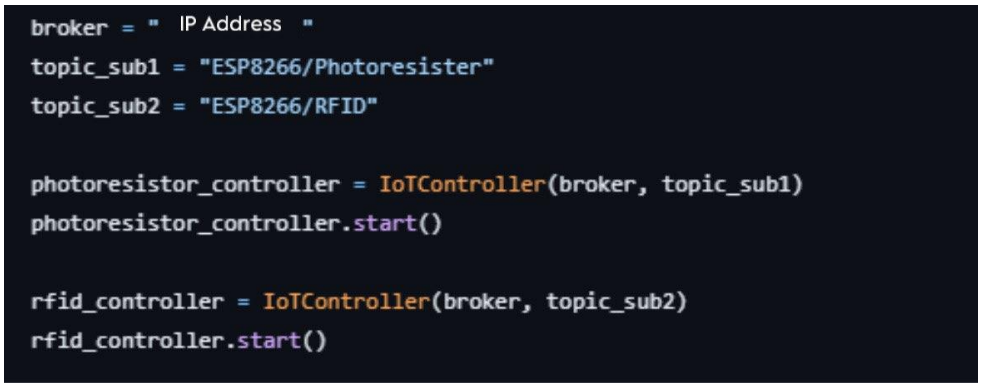
```
# pip install paho-mqtt
import paho.mqtt.client as mqtt
from IoTModel import IoTModel
import threading
import time

class IoTController:
    topic_sub1 = "ESP8266/Photoresister"
    topic_sub2 = "ESP8266/RFID"
    DB_FILE = 'user_data.db'
    user_model = None
    data = 0
    lightIntensity = 0
    rfid = 0

    def __init__(self, broker_address, topic):
        self.user_model = IoTModel(self.DB_FILE)

        self.broker_address = broker_address
        self.topic = topic
        self.client = mqtt.Client()

        self.client.on_connect = self.on_connect
        self.client.on_message = self.on_message

    def on_connect(self, client, userdata, flags, rc):
        if rc == 0:
            print(f"Connected to MQTT Broker at {self.broker_address}")
            self.client.subscribe(self.topic)
        else:
            print("Failed to connect, return code: ", rc)

    def on_message(self, client, userdata, message):
        # print(f"{self.topic}: {str(message.payload.decode('utf-8'))}")
        self.data = int(message.payload.decode('utf-8'))

        if(self.topic == "ESP8266/Photoresister"):
            self.lightIntensity = int(message.payload.decode('utf-8'))
            # print(f'Intensity: {self.lightIntensity}')

        if(self.topic == "ESP8266/RFID"):
            self.rfid = int(message.payload.decode('utf-8'))
            # print(f'RFID DATA: {self.rfid}')

    def start(self):
        self.client.connect(self.broker_address, 1883)
        # self.client.loop_forever()
        self.client.loop_start()
```

getRfid will get the data from the model using **select_user**. We then proceed to place the data in a set of key-value pairs. This will make it easier to retrieve the data when it gets passed to the application.



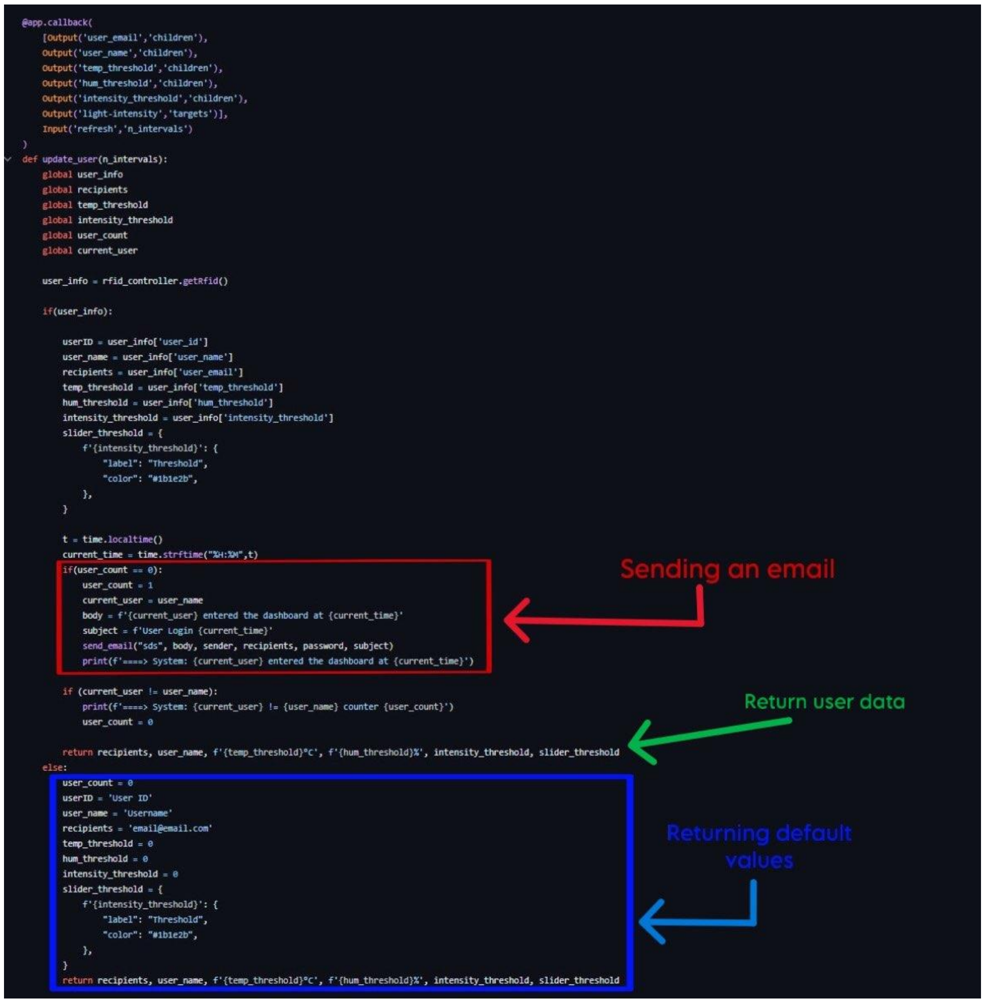
```
def getRfid(self):

    user_info = self.user_model.select_user(self.rfid)
    # print(self.rfid)
    print(f' User Info: {user_info}')
    user = None
    if(user_info):
        user = {
            'user_id' : user_info[0],
            'user_name' : user_info[1],
            'user_email' : user_info[2],
            'temp_threshold' : user_info[3],
            'hum_threshold' : user_info[4],
            'intensity_threshold' : user_info[5],
        }
    # print(user)
    return user
```

app.py



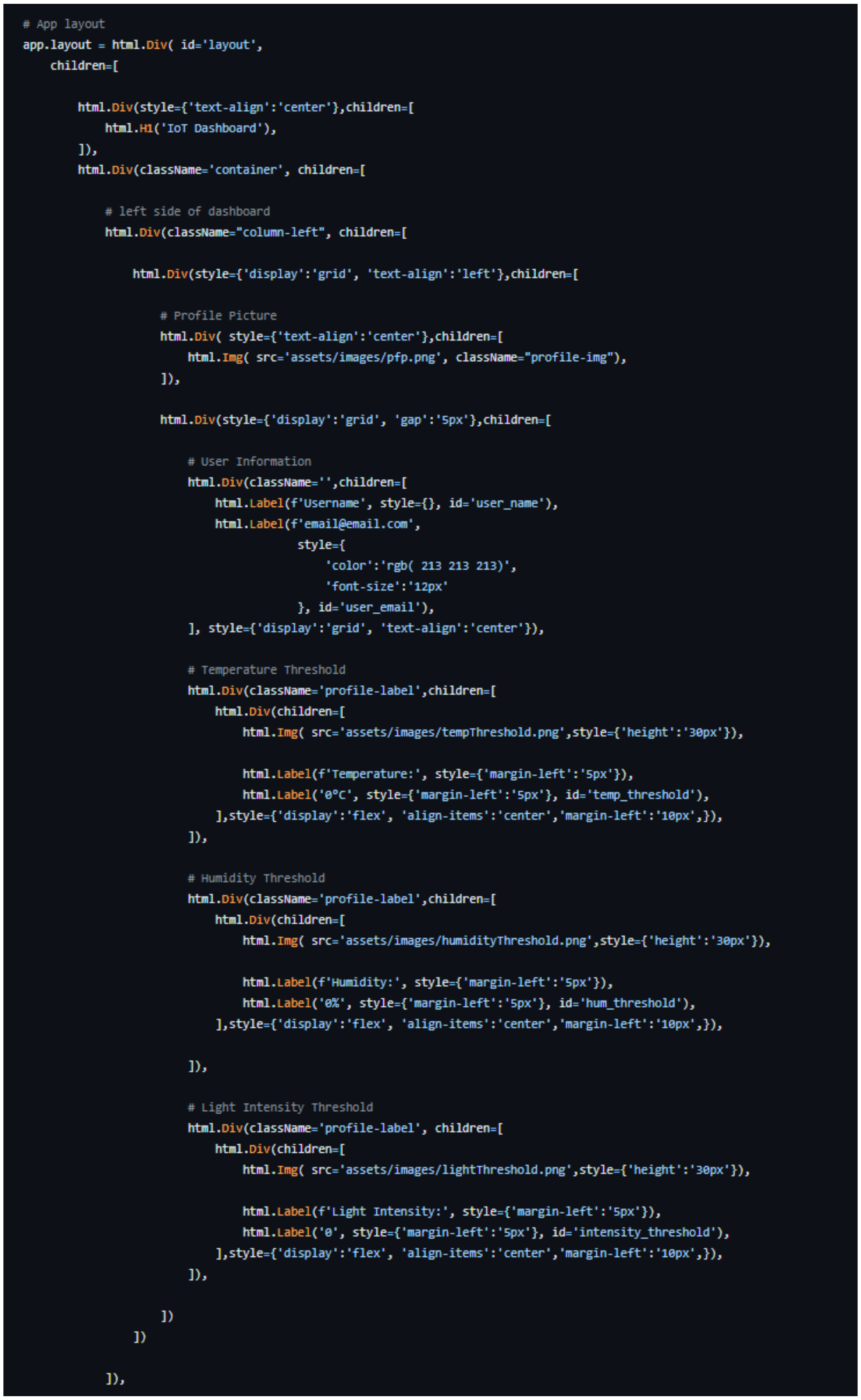
```
broker = "  IP Address  "
topic_sub1 = "ESP8266/Photoresister"
topic_sub2 = "ESP8266/RFID"

photoresistor_controller = IoTController(broker, topic_sub1)
photoresistor_controller.start()

rfid_controller = IoTController(broker, topic_sub2)
rfid_controller.start()
```

update_user gets the user data from the IoTController. If the user exists, it will return the user data, otherwise it will return default values. Furthermore, an email will be sent to the user. A counter is used once again to prevent the system from spamming the user with emails.

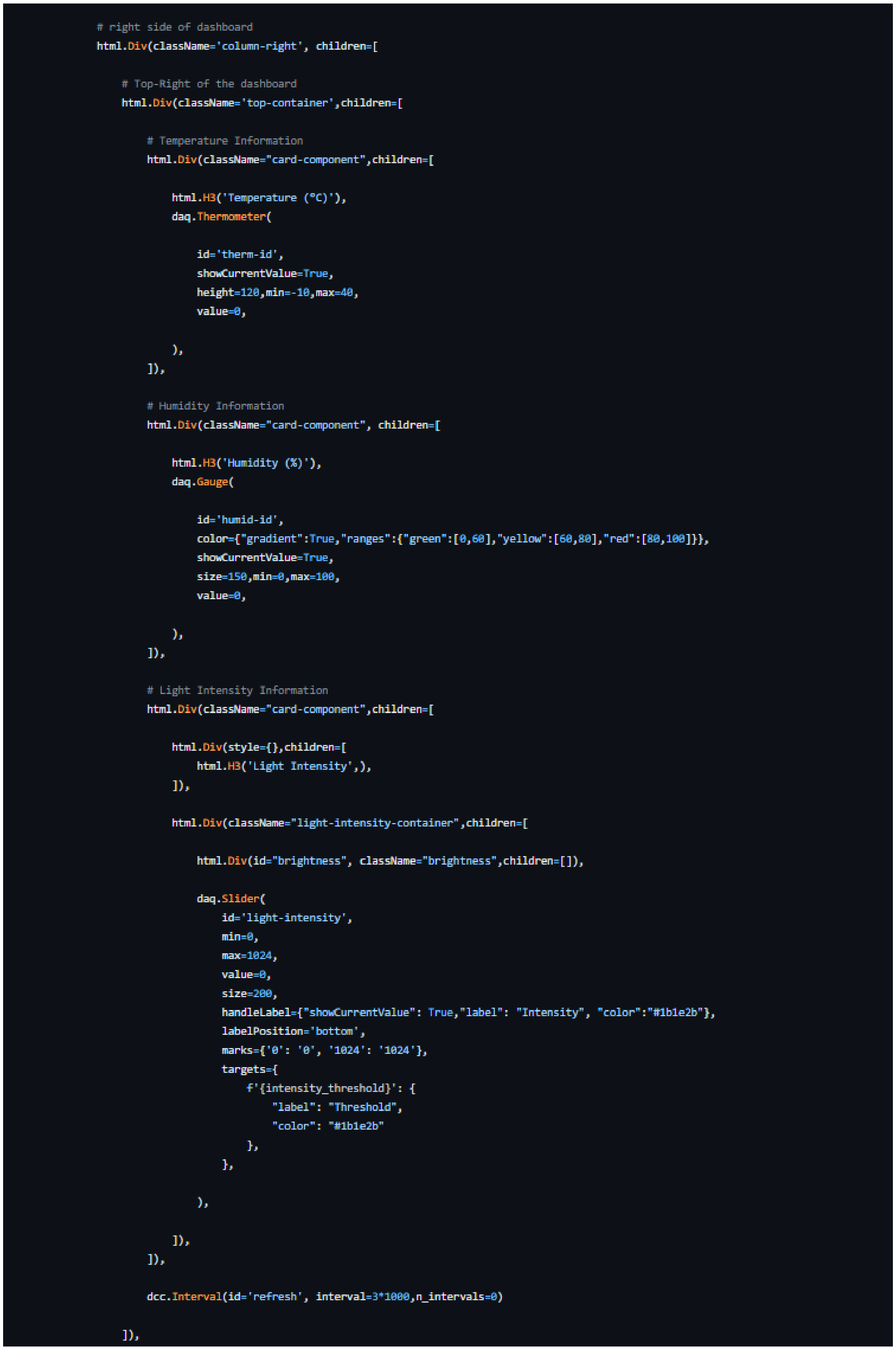


```
@app.callback(
    [Output('user_email','children'),
    Output('user_name','children'),
    Output('temp_threshold','children'),
    Output('hum_threshold','children'),
    Output('intensity_threshold','children'),
    Output('light-intensity','targets')],
    Input('refresh','n_intervals')
)
def update_user(n_intervals):
    global user_info
    global recipients
    global temp_threshold
    global intensity_threshold
    global user_count
    global current_user

    user_info = rfid_controller.getRfid()

    if(user_info):

        userID = user_info['user_id']
        user_name = user_info['user_name']
        recipients = user_info['user_email']
        temp_threshold = user_info['temp_threshold']
        hum_threshold = user_info['hum_threshold']
        intensity_threshold = user_info['intensity_threshold']
        slider_threshold = {
            f'{intensity_threshold}': {
                "label": "Threshold",
                "color": "#1b1e2b",
            },
        }

        t = time.localtime()
        current_time = time.strftime("%H:%M",t)
        if(user_count == 0):
            user_count = 1
            current_user = user_name
            body = f'{current_user} entered the dashboard at {current_time}'
            subject = f'User Login {current_time}'
            send_email("sds", body, sender, recipients, password, subject)
            print(f'====> System: {current_user} entered the dashboard at {current_time}')

        if (current_user != user_name):
            print(f'====> System: {current_user} != {user_name} counter {user_count}')
            user_count = 0

        return recipients, user_name, f'{temp_threshold}°C', f'{hum_threshold}%', intensity_threshold, slider_threshold
    else:
        user_count = 0
        userID = 'User ID'
        user_name = 'Username'
        recipients = 'email@email.com'
        temp_threshold = 0
        hum_threshold = 0
        intensity_threshold = 0
        slider_threshold = {
            f'{intensity_threshold}': {
                "label": "Threshold",
                "color": "#1b1e2b",
            },
        }
        return recipients, user_name, f'{temp_threshold}°C', f'{hum_threshold}%', intensity_threshold, slider_threshold
```

Final Application HTML Layout



```
# App layout
app.layout = html.Div( id='layout',
    children=[

        html.Div(style={'text-align':'center'},children=[
            html.H1('IoT Dashboard'),
        ]),
        html.Div(className='container', children=[

            # left side of dashboard
            html.Div(className="column-left", children=[

                html.Div(style={'display':'grid', 'text-align':'left'},children=[

                    # Profile Picture
                    html.Div( style={'text-align':'center'},children=[
                        html.Img( src='assets/images/pfp.png', className="profile-img"),
                    ]),

                    html.Div(style={'display':'grid', 'gap':'5px'},children=[

                        # User Information
                        html.Div(className='',children=[
                            html.Label(f'Username', style={}, id='user_name'),
                            html.Label(f'email@email.com',
                                        style={
                                            'color':'rgb( 213 213 213)',
                                            'font-size':'12px'
                                        }, id='user_email'),
                        ], style={'display':'grid', 'text-align':'center'}),

                        # Temperature Threshold
                        html.Div(className='profile-label',children=[
                            html.Div(children=[
                                html.Img( src='assets/images/tempThreshold.png',style={'height':'30px'}),

                                html.Label(f'Temperature:', style={'margin-left':'5px'}),
                                html.Label('0°C', style={'margin-left':'5px'}, id='temp_threshold'),
                            ],style={'display':'flex', 'align-items':'center','margin-left':'10px',}),
                        ]),

                        # Humidity Threshold
                        html.Div(className='profile-label',children=[
                            html.Div(children=[
                                html.Img( src='assets/images/humidityThreshold.png',style={'height':'30px'}),

                                html.Label(f'Humidity:', style={'margin-left':'5px'}),
                                html.Label('0%', style={'margin-left':'5px'}, id='hum_threshold'),
                            ],style={'display':'flex', 'align-items':'center','margin-left':'10px',}),

                        ]),

                        # Light Intensity Threshold
                        html.Div(className='profile-label', children=[
                            html.Div(children=[
                                html.Img( src='assets/images/lightThreshold.png',style={'height':'30px'}),

                                html.Label(f'Light Intensity:', style={'margin-left':'5px'}),
                                html.Label('0', style={'margin-left':'5px'}, id='intensity_threshold'),
                            ],style={'display':'flex', 'align-items':'center','margin-left':'10px',}),
                        ]),

                    ])
                ])

            ]),
```



```
        # right side of dashboard
        html.Div(className='column-right', children=[

            # Top-Right of the dashboard
            html.Div(className='top-container',children=[

                # Temperature Information
                html.Div(className="card-component",children=[

                    html.H3('Temperature (°C)'),
                    daq.Thermometer(

                        id='therm-id',
                        showCurrentValue=True,
                        height=120,min=-10,max=40,
                        value=0,

                    ),
                ]),

                # Humidity Information
                html.Div(className="card-component", children=[

                    html.H3('Humidity (%)'),
                    daq.Gauge(

                        id='humid-id',
                        color={"gradient":True,"ranges":{"green":[0,60],"yellow":[60,80],"red":[80,100]}},
                        showCurrentValue=True,
                        size=150,min=0,max=100,
                        value=0,

                    ),
                ]),

                # Light Intensity Information
                html.Div(className="card-component",children=[

                    html.Div(style={},children=[
                        html.H3('Light Intensity',),
                    ]),

                    html.Div(className="light-intensity-container",children=[

                        html.Div(id="brightness", className="brightness",children=[]),

                        daq.Slider(
                            id='light-intensity',
                            min=0,
                            max=1024,
                            value=0,
                            size=200,
                            handleLabel={"showCurrentValue": True,"label": "Intensity", "color":"#1b1e2b"},
                            labelPosition='bottom',
                            marks={'0': '0', '1024': '1024'},
                            targets={
                                f'{intensity_threshold}': {
                                    "label": "Threshold",
                                    "color": "#1b1e2b"
                                },
                            },

                        ),

                    ]),
                ]),

                dcc.Interval(id='refresh', interval=3*1000,n_intervals=0)

            ]),
```

```
                # Bottom-Right of the dashboard
                html.Div(className="bottom-container", children=[

                    # DC Motor Fan
                    html.Div(className="card-component", children=[

                        html.Div(style={'display':'grid', 'grid-template-columns':'auto auto auto', },children=[
                            html.H3('DC Motor Fan:'),
                        ]),

                        html.Div(children=[
                            html.Img( src=fan_off, id='fan-img', className="fan-img" ),
                        ]),

                    ]),

                    # LED
                    html.Div(className="card-component", children=[

                        html.Div(style={'display':'grid', 'grid-template-columns':'auto auto auto', },children=[
                            html.H3('LED:'),
                            html.Div(style={ 'grid-column': '3', 'padding-top':'20px'},children=[
                                daq.BooleanSwitch(
                                    disabled=True,
                                    on=False,
                                    id='light-switch',
                                    className='dark-theme-control',
                                    color='#707798'
                                ),
                            ])
                        ]),

                        html.Div([
                            html.Img( src=img_light_off, id='light-img',  className="feature-img" )
                        ]),

                    ]),

                ])

            ],)
        ]),

    ]
)
```