

CLOUD SECURITY (CCF2C05)
Case Study / Project Report Submission

Practical Class : P02 Group A
Submitted by: 1900931E | GOH KAI XUAN JERICK
1905031I | HAN XIAOZHI
1900233F | MUHAMMAD ANIQ SYUKRI B MD A
1902702H | MUHAMMAD MIKAIL BIN JASMAN
1900353B | ZACHARY PHOON JUN ZE
1900313F | SO CHENG YI, AUGMEN

Date: 09/08/2021

“By submitting this work, I am / we are declaring that I am / we are the originator(s) of this work and that all other original sources used in this work have been appropriately acknowledged.

I / We understand that plagiarism is the act of taking and using the whole or any part of another person’s work and presenting it as my/ our own without proper acknowledgement.

I / We also understand that plagiarism is an academic offence and that disciplinary action will be taken for plagiarism.”

NAME AND SIGNATURE OF STUDENT: GOH KAI XUAN JERICK



NAME AND SIGNATURE OF STUDENT: HAN XIAOZHI



NAME AND SIGNATURE OF STUDENT: MUHAMMAD ANIQ SYUKRI B MD A



NAME AND SIGNATURE OF STUDENT: MUHAMMAD MIKAIL BIN JASMAN



NAME AND SIGNATURE OF STUDENT: ZACHARY PHOON JUN ZE



NAME AND SIGNATURE OF STUDENT: SO CHENG YI, AUGMEN



School of Informatics & IT School

Cloud Security (COST)

[Subject Code – CCF2C05]

PROJECT REPORT

Project Responsibilities

Member	Tasks
Zachary	IoT Sensor <ul style="list-style-type: none"> • Aircon Sensor • Motion detector safe light sensor • Telegram Bot For Alert Back
Xiaozhi	IoT Sensor <ul style="list-style-type: none"> • Temperature Sensor Simulation in Python to simulate changes in temperature virtually on the Raspberry PI. • Smart TV Sensor • A basic oven sensor Vulnerability testing <ul style="list-style-type: none"> • Nmap Scan Separate Historian DB for smart TV Sensor and Temperature <ul style="list-style-type: none"> • Implemented add and update functions with data visualisation • Implemented Login and Sign up • Implemented DB display
Aniq	MQTT <ul style="list-style-type: none"> • Implemented TLS Security Feature • Implemented Paho MQTT
Mikail	MQTT <ul style="list-style-type: none"> • Research on Paho MQTT • Implemented TLS Security Feature • Implemented Paho MQTT
Augmen	Historian DB <ul style="list-style-type: none"> • Implemented 2FA and login page • Implemented add and update functions for database
Jerick	Historian DB <ul style="list-style-type: none"> • Implemented add and update functions for database • Implemented display of DB table and calculation of total cost

Suggested Project Report Format / Contents:

Table of Contents

Project Responsibilities	3
Introduction	5
How does our IoT System work?	5
IoT Sensors	7
Aircon IOT Remote :	7
Motion Detector Light	8
THERMOSTAT	11
Sensor Vulnerability Assessment	17
Nmap Scan	17
MQTT	17
Documentation of secured implementation process with screenshots	18
MQTT Security Implementation	19
Web Application Historian DB	21
Vulnerability assessments and resolutions	222
Documentation of secured implementation process with screenshots	23
Conclusion	290
Appendix	441
Screenshot of individual completion of AWS Cloud Foundation Badge	441
Screenshot of individual completion of subject survey	474
References	485
Code or scripts developed for the respective project components	485

A. Introduction

How does our IoT System work?

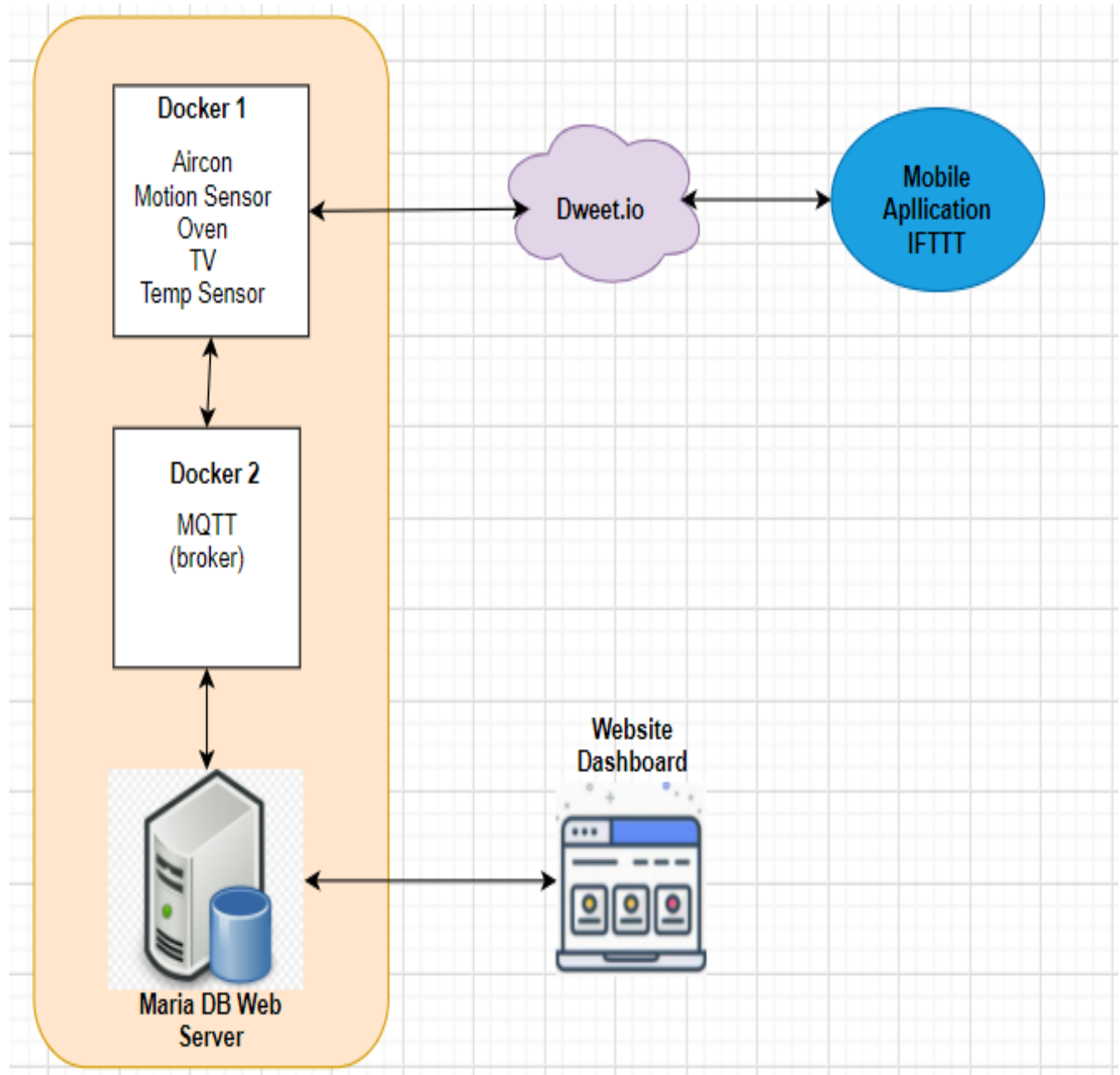
By utilizing Python scripting alongside the services of IFTTT, we managed to replicate and secure a whole smart home system. With IFTTT, applets can be created to trigger specific alerts, causing various smart home devices to function with various specifications. Our sensors include a simulated temperature sensor, aircon & oven cost sensor, motion detector and a TV.

As the simulated temperature sensor runs, it will obtain real-time temperature data and sends the data to the user via the household's dedicated Telegram channel. The message consists of the temperature recorded, a category and lastly some recommendations for the user. Any household member can then use the IFTTT application to turn on their smart devices according to pre-set settings such as turning on or off the aircon temperature and oven appliances, or to turn on the TV for personal entertainment. All these can be done on their personal mobile devices.

Next, MQTT is utilized as a broker to allow communication to take place between all IoT sensors and the Maria Web DB server. A front-end website will be present to log all the sensors' activities and displays it in a tabular format. Lastly, there will be a smart family messaging system that will trigger alerts based on any changes to any sensor's state and will post the messages in real-time, making it easy for sensor tracking purposes.

The Maria Web DB server will then subscribe to the MQTT broker to either add or update data directly to the historian DB. All household members will be able to access the database to view all records of the sensor's activities. The access will be secured via authentication function, which features login, sign up, and sessions, such that only registered users are able to view the data. Protection against SQL Injection will also be present to prevent data from being altered maliciously. Graphs and charts will also be added to allow the users to interpret the data more clearly and effectively.

Visuals and Diagrams of our IoT System Model



B. IoT Sensors

Documentation of implementation process with screenshots

i. Aircon IOT Remote :

It uses the emulator of a Raspberry Pie to determine if it is ON or OFF using a Red and Green LED Indicator and Dweet.io and IFTTT(If This Then That). It first requests the current state of the IOT, once it is updated, it will send it to the MQTT as well as a notification and a telegram bot to retrieve the data that is sent over.

```
Numbering style set to BroadCom Numbering style
Set warnings to False
Setup channel(s) 20 for direction "Out" Direction for Pin, with pull_up_down Pull Up/Down Resistor Disa
bled and set to initial value None
Setup channel(s) 21 for direction "Out" Direction for Pin, with pull_up_down Pull Up/Down Resistor Disa
bled and set to initial value None
Output a False value to channel(s) 20
Output a False value to channel(s) 21
All LEDs are turned OFF

Is the Device on or off ? (o)n to ON or (f) to OFF ? (q) to quit: f
Turning Off the Device
Output a False value to channel(s) 20
Output a True value to channel(s) 21
sending dweet...
Publishing ...
Published !
send dweet@ 2021-08-02T02:47:57.536Z
```

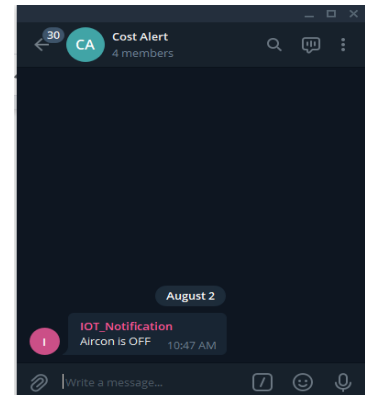
```
broker="iot-mqtt"
port = 1883
client = mqtt.Client("IOT-Sensor-Publisher")
client.username_pw_set(username= "root",password="kali")
certfilepath= "/root/cost_vol/scripts/cacert/ca.crt"
client.tls_set(certfilepath,tls_version=2)
client.tls_insecure_set(False)
```

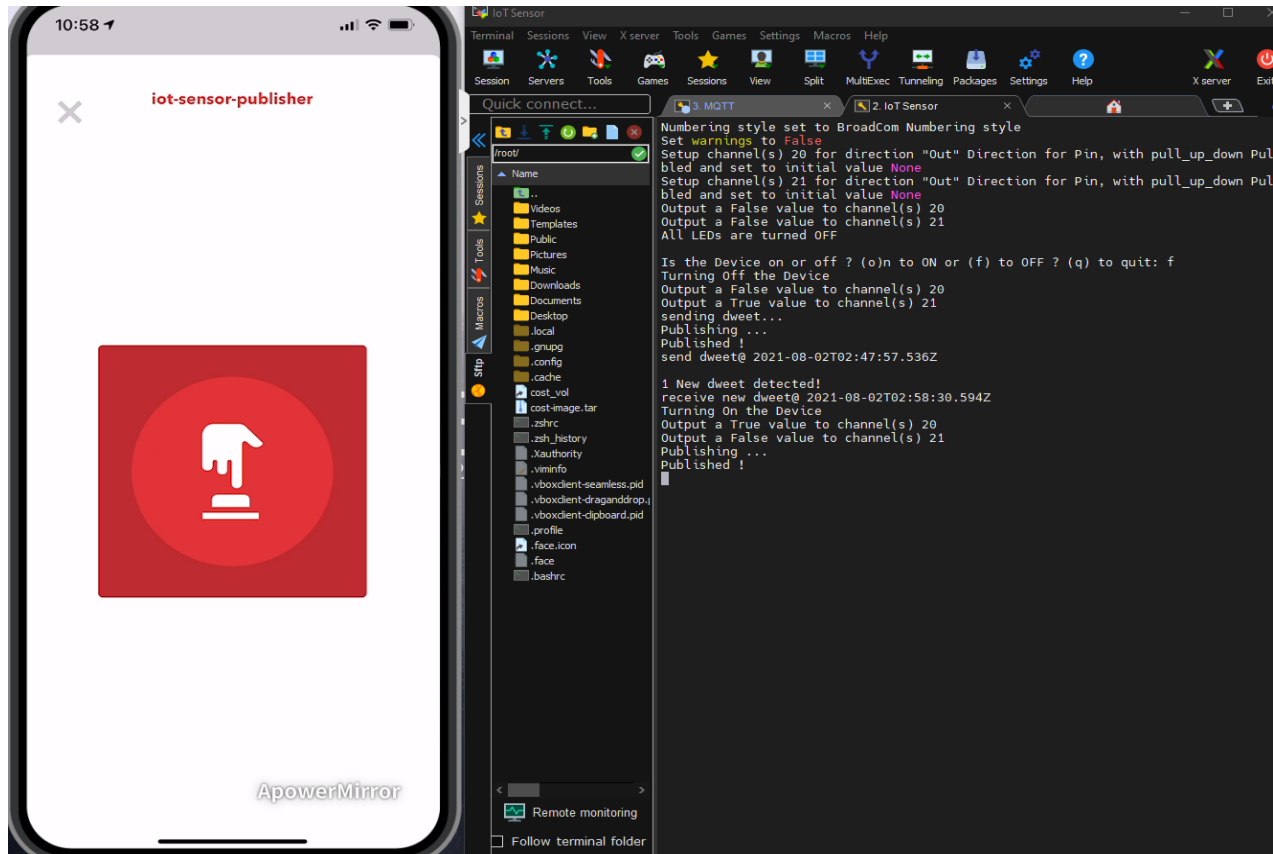
```
#MQTT codes
client.connect(broker) #Connects to broker
client.loop_start()
print('Publishing ... ')
client.publish("iot","Aircon:OFF")
print('Published !')
time.sleep(4)
client.loop_stop()
client.disconnect()
```

The two images above are codes to connect to the MQTT using the paho module. This allows the codes to be used and sent over without us connecting to the MQTT each time we need to connect.

IFTTT no longer supports webhook to send an iOS Notification, hence the other alternative I used is to use a telegram bot to echo out a notification to a chat. This will allow users to use existing texting platforms as a way of receiving notification as seen on the right.

The image below displays how the app IFTTT is used in the structure of this system.





ii. Motion Detector Light

This IOT Sensor uses a camera to detect motion. It takes the first frame, makes a grey scale and when the grey scale images in the next few frames do not match, it will determine that someone is in the room and it will create a green box and draw out a frame around the area that does not match. You can use an existing video as well with the “-v and image name” when it is being run.

```
ap = argparse.ArgumentParser()
ap.add_argument("-v", "--video", help="path to the video file")
ap.add_argument("-a", "--min-area", type=int, default=500, help="minimum area size")
args = vars(ap.parse_args())
# if the video argument is None, then we are reading from webcam
if args.get("video", None) is None:
    vs = VideoStream(src=0).start()
    time.sleep(2.0)
# otherwise, we are reading from a video file
else:
    vs = cv2.VideoCapture(args["video"])
# initialize the first frame in the video stream
firstFrame = None
```

During this, it uses an emulated Raspberry Pi , and with this it inputs. For the purpose of this project the images will not be displayed when running in the docker but below are


```

# compute the absolute difference between the current frame and
# first frame
frameDelta = cv2.absdiff(firstFrame, gray)
thresh = cv2.threshold(frameDelta, 25, 255, cv2.THRESH_BINARY)[1]

# dilate the thresholded image to fill in holes, then find contours
# on thresholded image
thresh = cv2.dilate(thresh, None, iterations=2)
cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
| cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)

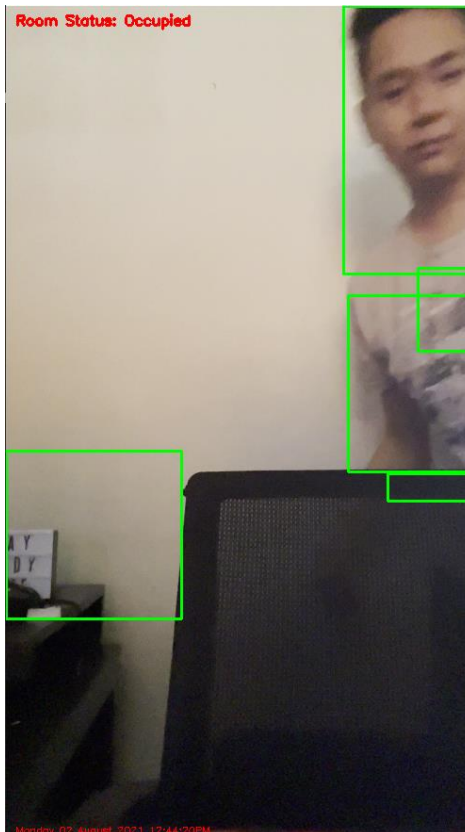
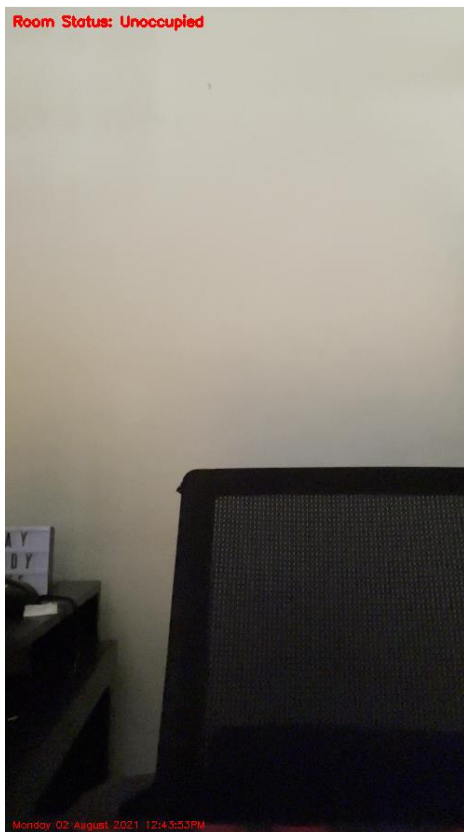
# loop over the contours
for c in cnts:
    # if the contour is too small, ignore it
    if cv2.contourArea(c) < args["min_area"]:
        | continue

    # compute the bounding box for the contour, draw it on the frame,
    # and update the text
    (x, y, w, h) = cv2.boundingRect(c)
    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
    GPIO.output(3,1)

    time.sleep(0.1)
    text = "Occupied"

```

the images of the video for better understanding. When the room status changes from Unoccupied to Occupied, it will publish to the MQTT to send that signal that it has turned ON similarly for Occupied to Unoccupied.



OVEN

to the aircon, it will request the current state of the IOT, once it is updated, it will send it to the MQTT as well as a notification and a telegram bot to retrieve the data that is sent over. I then used a telegram bot to echo out an IOS notification to a chat. This will allow users to use existing texting platforms as a way of receiving notification as seen on the right.

```
Numbering style set to BroadCom Numbering style
Set warnings to False
Setup channel(s) 20 for direction "Out" Direction for Pin, with pull_up_down Pul
l Up/Down Resistor Disabled and set to initial value None
Setup channel(s) 21 for direction "Out" Direction for Pin, with pull_up_down Pul
l Up/Down Resistor Disabled and set to initial value None
Output a False value to channel(s) 20
Output a False value to channel(s) 21
All LEDs are turned OFF

Is the Oven on or off ? (o)n to ON or (f) to OFF ? (q) to quit: o
Turning On the Oven
Output a True value to channel(s) 20
Output a False value to channel(s) 21
sending dweet...
Publishing ...
Published !
send dweet@ 2021-08-07T22:16:56.422Z
```

Telegram bot:



iii. THERMOSTAT

This section will cover my Temperature Sensor Simulation script, which is done to simulate a sensor obtaining temperature data in real time. Due to physical limitations, we decided to simulate the process by utilizing the python function “random.randrange()” which will generate random numbers. We set a realistic temperature range between 20-35 degrees, in which the “random.randrange” function will generate a random temperature within the range.

Categories have also been pre-defined to categorize the temperature that is received from the sensor.

Now, I will go into detail on how we created a Python script to simulate the process of obtaining temperature data from a sensor and transmits data through ‘dweet.io’ and MQTT.

Required libraries / modules:

- os - carry out system commands
- dweepy - to communicate with dweet.io
- time - allow a delay in each random temp generated
- random - generate random numbers
- EmulateGPIO as GPIO - virtual Raspberry Pi

After doing so, begin programming by importing the relevant libraries and modules.

```
import os
import dweepy
import time
import random
import EmulateGPIO as GPIO
import json
import paho.mqtt.client as mqtt
```

Next, utilize the “clear” function of bash to clear the terminal before printing any output. Also, label “myThing” for communication with ‘dweet.io’. Add the function that allows Telegram notifications to be sent to the user to view real-time updates on the current temperature. This code is done and integrated by Yusuf.

```
_ = os.system("clear")

#Telegram notification
#-----
def telegramText(Message):

    bot_token = '1922316795:AAEH7LfWPnFExSpkUepA1EtFyE36dunMC0U'
    bot_chatID = '-1001525214257'
    send_text = 'https://api.telegram.org/bot' + bot_token + '/sendMessage?chat_id=' + bot_chatID + '&parse_mode=Markdown&text=' + Message

    response = requests.get(send_text)

    return response.json()
#-----

# =====
myThing = "temp_change_sensor" # Label for communication with dweet.io
# =====
```

Moving on, setup the hardware related to the virtual Raspberry Pi; the desired pin numbering scheme and the proposed PIN numbers for the LEDs (Green and Red).

```
# Setup hardware
# Set the desired pin numbering scheme:
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# Create variables for the GPIO PINs the LEDs are connected to
# =====
# the PIN of the green LED
GreenLEDPin = 20 # Add values: add the pin number for the green LED
# the PIN of the red LED
RedLEDPin = 21 # Add values: add the pin number for the red LED
# =====
```

At this point, it is important to set up the direction of the GPIO pins, with the PINs connected to LEDs set to Output mode.

```
# Setup the direction of the GPIO pins - either INput or OUTput
# The PINs that connect LEDs must be set to OUTput mode:
# Prepare for initialization
GPIO.setup(GreenLEDPin, GPIO.OUT)
GPIO.setup(RedLEDPin, GPIO.OUT)
GPIO.output(GreenLEDPin, False) # False = set 0V on the pin
GPIO.output(RedLEDPin, False) # False = set 0V on the pin
print('All LEDs are turned OFF' + '\n')
print(" ")
```

Now, write a function “getCurrentTemp()” that consists of the code to simulate the temperature sensor. When the function is called, the LEDs should turn red to signify that the program is on standby mode.

```
# main function
def getCurrentTemp():

    GPIO.output(GreenLEDPin, False) # False = set 0V on the pin
    GPIO.output(RedLEDPin, True) # True = set 3.3V on the pin
    print("Activate Red LED") # Red LED for Standby Mode
    print(" ")
```

Then, use a 'for' loop to simulate temperature changes by generating random numbers from the predefined range. A random number within the range would be generated, with a print statement declaring the temperature. This value would then be renamed to 'newTempRange'. At this point, the Green LED would be lit up to show that the process is running.

Next, add some conditional statements to categorize the temperature generated. As the range is set to four, it will run the whole loop four times before stopping, simulating four occurrences of temperature change.

The loop will send the temperature and its category to the user via a Telegram notification (e.g. current temperature is 34 degrees, it is categorized as "HOT").

When a temperature change is detected by the sensor, it will also send a notification to the telegram group that was created specifically for the IoT devices. It will notify the user of information such as the temperature, category of temperature and the recommended action to take.

```
for i in range(4):
    # auto generate temp for input
    tempRange = str(random.randrange(20, 35))

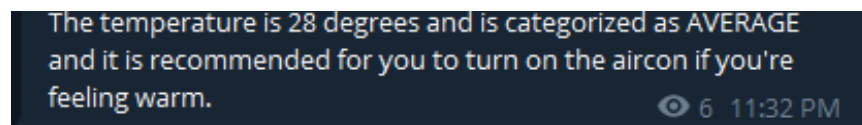
    # print temperature from input
    print("The current temperature is " + tempRange + " degrees.")
    newTempRange = int(tempRange)

    GPIO.output(GreenLEDPin, True) # True = set 3.3V on the pin
    GPIO.output(RedLEDPin, False) # False = set 0V on the pin
    print("Activate Green LED") # Green LED for Checking Temperature (working)

    # conditional statements
    if newTempRange >= 30 and newTempRange <= 35:
        category = "HOT"
        telegramText("The temperature is " + tempRange + " degrees and is categorized as " + category + " and it is recommended for you to turn on your aircon.")
    elif newTempRange >= 26 and newTempRange <= 29:
        category = "AVERAGE"
        telegramText("The temperature is " + tempRange + " degrees and is categorized as " + category + " and it is recommended for you to turn on the aircon if you're feeling warm.")
    elif newTempRange >= 23 and newTempRange <= 25:
        category = "GETTING COLD"
        telegramText("The temperature is " + tempRange + " degrees and is categorized as " + category + " and it is recommended for you to turn off your aircon if you're feeling cold.")
    elif newTempRange <= 22:
        category = "COLD"
        telegramText("The temperature is " + tempRange + " degrees and is categorized as " + category + " and it is recommended for you to turn off your aircon.")

    send_dweet = dweetpy.dweet_for(myThing, {"category": category, "temperature": newTempRange}) # this function sends data to dweet.io
    dweet_created = send_dweet['created'] # get the time stamp of the first dweet
    print("send dweet@ " + dweet_created + "\n")
    time.sleep(5)
```

Telegram:



Once the whole process has been completed, deactivate all LEDs to signify that the simulation has ended. This portion of the code is outside of the loop.

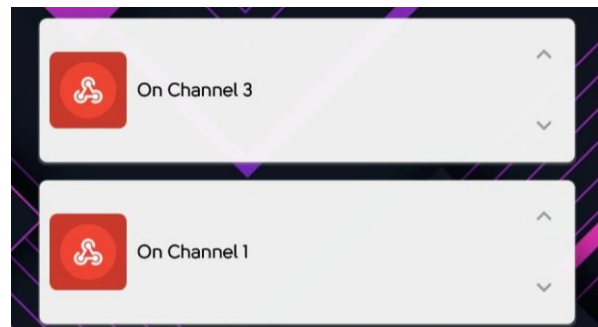
```
GPIO.output(GreenLEDPin, False) # True = set 3.3V on the pin
GPIO.output(RedLEDPin, False) #False = set 0V on the pin
print('Deactivate All LEDs' + '\n') #close all LEDs after process finishes

getCurrentTemp() # call function
```

TV

As the user uses the IFTTT to trigger the alert such as “On Channel 1”, a webhook will be sent to the dweet.io. Hence, the trigger will be relayed to the “mything” which is linked to the Samsung TV. This will cause the Samsung TV to be switched on and tuned in to the respective channel as input by the user. The IFTTT buttons can be set up as seen below. Users will be able make use of the IFTTT smart phone widget to toggle on and off the Smart TV and switch between the 3 different channels. (I configured 3 channels to replicate the functionality of a Smart TV.) This is possible as made use of the DWEET and GPIO libraries to come up with a Smart TV replica

Edit Applet



```
import dweeepy
import EmulateGPIO as GPIO
from IPython.display import clear_output
import time
import os
import sys
from tabulate import tabulate
import random
import requests
import paho.mqtt.client as mqtt
import random, threading, json
from datetime import datetime
```


Sensor Vulnerability Assessment

Nmap Scan

Nmap can be used to find undersrand more about the running services and ports of the local netowrk. I launched a reconnaissance attack to find any open ports or vulnerabilities on the local network in order to better understand the operating services and ports. This also allows us to identify any vulnerability and mark out the topology. To find the network services accessible on the Raspberry Pi, i used the terminal to run the following command:

```
└─# nmap 172.16.0.0/24 -p-
Starting Nmap 7.91 ( https://nmap.org ) at 2021-08-08 06:51 +08
Nmap scan report for 172.16.0.11
Host is up (0.0000070s latency).
All 65535 scanned ports on 172.16.0.11 are closed
MAC Address: 02:42:AC:10:00:0B (Unknown)

Nmap scan report for 172.16.0.12
Host is up (0.0000070s latency).
All 65535 scanned ports on 172.16.0.12 are closed
MAC Address: 02:42:AC:10:00:0C (Unknown)

Nmap scan report for 172.16.0.13
Host is up (0.0000070s latency).
All 65535 scanned ports on 172.16.0.13 are closed
MAC Address: 02:42:AC:10:00:0D (Unknown)

Nmap scan report for 172.16.0.1
Host is up (0.0000060s latency).
Not shown: 65534 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh

Nmap done: 256 IP addresses (4 hosts up) scanned in 5.25 seconds
```


able to gain access to the sensitive data that is being transmitted through the MQTT from the IOT Sensor to the Web DB.

MQTT Security Implementation

Firstly we import the paho.mqtt.client. Paho MQTT is a Python based MQTT which also still uses the Mosquitto Broker. Since most of this project is coded in Python we used the Paho MQTT.

```
#Import MQTT connector module.
import paho.mqtt.client as mqtt

#MQTT Details
broker="iot-mqtt"
port = 1883
client = mqtt.Client("IOT-Sensor-Publisher")
client.username_pw_set(username= "root",password="kali")
certfilepath= "/root/cost_vol/scripts/cacert/ca.crt"
client.tls_set(certfilepath,tls_version=2)
client.tls_insecure_set(False)
```

The MQTT details that have been configured for the MQTT broker, shows the port number that the devices will be connected on and also it shows the connection to the IOT Sensor which is acting as the publisher in this case. Additionally, there are a few security implementations that is applied, it is the encryption of the password of the user and TLS security. The username password set will have its “passwords” file stored within a cacert file to authenticate with the user logging in. Furthermore, there is a docker generated TLS certificate authority (CA) certificate that is linked to prevent Man-in-the-middle attacks from happening. The TLS set details specifies the certification file path and the version specifies that the SSL/TLS protocol 2 is used. The “client.tls_insecure_set(False)” is to guarantee that the host the broker is connecting to is not impersonating the broker server itself. This is to prevent third party malicious software impersonating the MQTT broker and initiate Man-in-the-middle attacks.

The codes shown below shows the functions that the MQTT broker is designed to do on startup, the first function is to establish a connection to the client of the MQTT broker, which in this case is the appliances. The second function is to establish the publishing of data to the Historian Database. The third function is a disconnect function. In the whilst “true” statement, there will be an input where it will ask the user to select the option of the LED to turn it on or off. The response will then be put into a variable, converting it to lowercase and putting it into another variable. Therefore, if the user chose to turn off the device chosen, it will print the status and it will send a dweet to the MQTT broker based on a false or true variable.

```
#MQTT Functions
def on_connect(client, userdata, flags, rc):
    print("Connected with result code", rc)

def on_publish(client, userdata,result):
    print ("Data is being Published \n")

def on_disconnect(client, userdata, rc):
    print("Broker has Disconnected!")

while True:

    # Asks the user to select the LED. Put the response into a variable.
    lit = input("Is the Device on or off ? (o)n to ON or (f) to OFF ? (q) to quit: ")

    # convert the input to lowercase and put it in another variable.
    lit1 = lit.lower()

    #Set the LED state based on the user input
    if lit1 == "f": #If the user chose the Turn OFF the Device
        print("Turning Off the Device")
        GPIO.output(GreenLEDPin, False) # False = set 0V on the pin
        GPIO.output(RedLEDPin, True) # True = set 3.3V on the pin
        print('sending dweet...')

        #MQTT codes
        client.connect(broker) #Connects to broker
        client.loop_start()
        print('Publishing ... ')
        client.publish("iot","Aircon:OFF")
        print('Published !')
        time.sleep(4)
        client.loop_stop()
        client.disconnect()
```

The codes will establish a connection to the broker and it will start publishing the variables or states of the appliance. The screenshot below, is the code function to send data to dweet.io to initiate a “Red” dweet and get the timestamp of the first dweet.

```
old_dweet = dweepy.dweet_for(myThing, {"dweet": "Red"}) #this function sends data to dweet.io
old_created = old_dweet['created'] #get the time stamp of the first dweet

print('send dweet@' , old_created + '\n')
break
```

These screenshot below has the same function for the MQTT broker but it is executed when the user chooses to turn “ON” the device chosen.

```

counter = 0

while True:
    new_dweet = dweepy.get_latest_dweet_for(myThing) #get latest dweet
    new_created = new_dweet[0]["created"] #put the created value of the latest dweet into a variable
    if new_created != old_created: #check to see if the old dweet is different from the new dweet
        counter += 1
        print(str(counter) + " New dweet detected!",end='\n')
        old_created = new_created

    if lit1 == "o":
        print('receive new dweet@' , new_created)
        print("Turning Off the Device")
        GPIO.output(GreenLEDPin, False) # False = set 0V on the pin
        GPIO.output(RedLEDPin, True) # True = set 3.3V on the pin

        #MQTT codes
        client.connect(broker) #Connects to broker
        client.loop_start()
        print('Publishing ... ')
        client.publish("iot","Aircon:OFF")
        print('Published !')
        time.sleep(4)
        client.loop_stop()
        client.disconnect()

        lit1 = "f"
        print()

    elif lit1 == "f":
        print('receive new dweet@' , new_created)
        print("Turning On the Device")
        GPIO.output(GreenLEDPin, True)
        GPIO.output(RedLEDPin, False)

        #MQTT codes
        client.connect(broker) #Connects to broker
        client.loop_start()
        print('Publishing ... ')
        client.publish("iot","Aircon:ON")
        print('Published !')
        time.sleep(4)
        client.loop_stop()
        client.disconnect()

        lit1 = "o"
        print()

    time.sleep(1)

```

D. Web Application Historian DB

i. Vulnerability assessments and resolutions

No.	Attack Surface	Vulnerability	Resolution
1.	Application	Making use of brute-force to discover weaknesses in database structure	Block the brute-force attack.
2.	Application	SQL Injection, Cross Site Scripting, XSS	Use escape string for variables sending to database Using prepared statements when sending queries to the database
3.	Application	Unauthorized Access to data	Implemented login and 2FA system
4.	Application	Bypass login page, into database page	Implement access control to force login before giving access to database.
5.	Application	Using brute force to force login	Implement 2 factor authentication (2FA)

ii. Documentation of secured implementation process with screenshots

Start webdb.sh

First, to prevent brute force attacks on the web database, we will have to turn the firewall on and only allow 8080 port requests. To do this, we configure the start_webdb.sh , which we use to turn on the mySQL database.

```

1 |#!/bin/sh
2
3 ufw allow ssh
4 ufw allow 8080/tcp
5 ufw allow 80/tcp
6 ufw enable
7 ufw limit 8080/tcp
8 cd /root/iot_vol/scripts
9 service mysql start
10 mysql < init_mysql.sql
11 ./docker_app.py & ./webappreceiving.py
12

```

This ensures that the firewall is up before the database is up. Database will be in the docker_app.py and the webappreceiving.py is to listen for messages from the sensors and turn the appliance on or off according to the message. Line 7 might need to be commented if facing an issue accessing the database.

Security Relevancy

Firewall state has been changed to up and running in this script, protecting the application from brute force attacks conducted through either skipfish or sql map.

Docker_app.py

The docker_app.py is to show the database, login and 2fa page.

```

#init programme
mysql = MySQL(app)
bootstrap= Bootstrap(app)

#checking if the db is connected
@app.route('/init')
def init():
    #Connecting to DB
    cur = mysql.connection.cursor()

    conn = mysql.connection()

    #Prepared Statement
    cur.execute('SELECT * FROM sensors')

    #Executing the query and assiging the variable.
    db_test = cur.fetchall()

    #Close DB connection
    cur.close()

    #If data is returned, redirect.
    if db_test != None:
        return redirect("/login/")
    else:
        return ("MYSQL is not running")

# login page route
@app.route("/login/")
def login():
    return render_template("login.html")

```

The first part of this script is to fetch all data from the database and enable a route to the login page. Attempting to go to /db/ or the 2fa page before logging in will result in the user being redirected to the login page.

```

# 2FA page route
@app.route("/login/2fa/")
def login_2fa():
    if session.get('logged') == True:
        print("Access Granted")
    else:
        return redirect(url_for('login'))
    # generating random secret key for authentication
    #secret = pyotp.random_base32()
    secret = "QAAGLKQZH5ACKGNX"
    return render_template("login_2fa.html", secret=secret)

if username == creds["username"] and password == creds["password"]:
    session['logged'] = True
    return redirect(url_for("login_2fa"))

```

The 2fa page checks for the session variable “logged” and if it is true, access is granted to the 2fa page. Otherwise, it will not work and will redirect users to the login page again. If there is no issue, the 2fa will continue to generate a random base32 code. For ease of presentation, we set the secret key to a fixed base32 code we got from a previous generation.

```

#Database Page
@app.route('/db', methods=["GET","PUT"] )
def default():
    if session.get('authenticated') == True:
        #Same as Init
        cur = mysql.connection.cursor()
        cur.execute('SELECT * FROM sensors')
        results = cur.fetchall()
        cur.close()
        print("Access Granted")
    else:
        return redirect(url_for('login'))

```

Same goes for the database page; the user is redirected to the login page if not logged in. The “authenticated” session variable is only set to true if the 2fa code is correct. If logged in but 2fa code is wrong, a “danger” message is recorded.

Security Relevancy

- Login page and the 2FA function provides authentication and prevents unauthorized access to the user’s sensitive data.
- Access control measures implemented prevents broken access control (The user will not be able to access the database if they are logged in.)

```

@app.route("/login/2fa/", methods=["POST"])
def login_2fa_form():

    # getting secret key used by user
    secret = request.form.get("secret")
    # getting OTP provided by user
    otp = int(request.form.get("otp"))

    # verifying submitted OTP with PyOTP
    if pyotp.TOTP(secret).verify(otp):
        # inform users if OTP is valid
        flash("The TOTP 2FA token is valid", "success")
        session["authenticated"] = True
        request.endpoint
        return redirect("/db")
    else:
        # inform users if OTP is invalid
        flash("You have supplied an invalid 2FA token!", "danger")
        return redirect(url_for("login_2fa"))

```



```

# Add Function
@app.route("/db/add/", methods=["GET","PUT"])
@app.route("/db/add/<appliance_name>", methods=["GET","PUT"])
def add(appliance_name=None):
    StatusStr = request.args.get('Status') #Ensure that there are arguments to put into the db
    if StatusStr == None or appliance_name == None:
        return "Not Valid"
    try:
        cur = mysql.connection.cursor()
        query="insert into sensors(Appliance_Name,Status) values ('" + json.dumps(appliance_name) + "','" + json.dumps(StatusStr) + "');"
        print(query)
        cur.execute(query) #Executing the query
        mysql.connection.commit() #Committing the changes into the database

    except Exception as e: #If there is error
        print("Problem inserting into db: " + str(e))
        return "FAIL"
    cur.close()
    return "OK"

```

This part of the code is to help us perform the add function. Here, my argument requires both the status string and the appliance name string, otherwise the function would not execute. Then, if we have both arguments, we would have a cursor to help us perform the query which allows us to add a new entry into the database.

Security Relevancy

```

query="insert into sensors(Appliance_Name,Status) values ('" + json.dumps(appliance_name) + "','" + json.dumps(StatusStr) + "');"
print(query)

```

Here, we used json.dumps as an escape string. Json.dumps would remove any backslashes and convert quotes into string, thereby preventing any cross site scripting attacks.

Data Table

For the database table generation, we first check if the sql database is empty or not. If not, generate the table header first.

```

#Generation of Table
if results != None :
    #Table Headers
    completeTable="<tr >"
    completeTable += "<th> No. </th>"
    completeTable += "<th> Name </th>"
    completeTable += "<th> Status </th>"
    completeTable += "<th> Up Time (Min) </th>"
    completeTable += "<th> Total Cost ($) </th>"
    completeTable+="</tr>"

```

```

for c in results:
    completeTable+="|"
    #Retreiving the first Variable from DB Query
    completeTable += "<th>" + str(c[0]) + "</th>"

    #Retreiving the second Variable from DB Query
    completeTable += "<th>" + str(c[1]) + "</th>"

    #Retreiving the third Variable from DB Query and check for Boolean, '1' means ON and '0' means OFF

    if str(c[2]) == '1': #Check if its a '1' or '0'

        completeTable += "<th> ON </th>"
        #Takes the Fourth Variable from the DB Query and Call Time Diff
        start_time = c[3]

        #Calculation of time difference & Printing it into the table.
        current_time = datetime.now()
        difference = (current_time - start_time)
        difference_in_s = difference.total_seconds()
        minutes = divmod(difference_in_s, 60)[0]
        completeTable += "<th>" + str(minutes) + "</th>"

        #Calculation of Cost & Printing it into the table.
        total_cost = (difference_in_s * 0.6)
        completeTable += "<th>" + str(total_cost) + "</th>"

    else:
        completeTable += "<th> OFF </th>"
        completeTable += "<th> 0</th>"
        completeTable += "<th> 00.00 </th>"
        completeTable+="

|  |

```

Here, we start pulling each variable from the database and putting them into the table. For each of these, we need to convert to string to output it.

c[0] and c[1] are the device ids and the appliance names. Here, c[2] is referring to the status of the device. In the database, we stored the status as “0” for OFF or “1” for ON. To generate the table, we added in an if statement to check if it is a “1” or “0”. This way, if it is “1”, it would print out that the Status is ON, and we would take the current date and time (noted as c[3]) and minus the start time from it to get the “running time” or how long the appliance has been running for. For the calculation, we need to convert it from seconds to minutes. Once converted, we then calculate the total cost by multiplying the difference in seconds with a constant of 0.6 for demo purposes.

```

return
<!DOCTYPE html>
<title>Sensify</title>
<html>

<style>
.w3-top {
  background-color: #EBEBEB;
}

.bg {
  height:1080px;
  width:1900px;
}

</style>

<head>
<link rel="stylesheet" href="{{ url_for('static', filename='css/w3css.css') }}">
<link rel="stylesheet" href="{{ url_for('static', filename='css/googleapi.css') }}">
<link rel="stylesheet" href="{{ url_for('static', filename='css/font-awesome.css') }}">
</head>

<body>
<div class="w3-top">
<div class="w3-bar w3-white w3-card" id="myNavbar">
  <a href="/" class="w3-bar-item w3-button w3-wide">Sensify</a>
  <a href="/about" class="w3-bar-item w3-button">About</a>
  <a href="/info" class="w3-bar-item w3-button">Code Info</a>
  <a href="/logout" class="w3-bar-item w3-button">Logout</a>
</div>
</div>

```

Following this, we input the html code for the /db page. This code also outputs out the table onto the webpage

```

#Running the programme on localhost
if __name__ == "__main__":
    host = os.getenv("IP", "0.0.0.0")
    port = int(os.getenv("PORT", 8080))
    app.run(host=host, port=port, debug=False, use_reloader=True, use_evalex=False)

```

Lastly, this final code is to run the programme on localhost, on port 8080.

webappreceiving.py

```
def on_message(client, userdata, msg):
    Messagereceived = True
    #Converting the payload to string first then stripping the first 2 characters and the last one
    apayload = str(msg.payload)
    newpayload = apayload[2:-1]
    appliance_msg = newpayload
    #print (appliance_msg) //For testing
```

webappreceiving.py is a script designed to automate the subscription to mqtt broker, receiving messages sent over and to perform the update and add functions.

At the start of the function, we need to first convert the payload to string and remove the first 2 and last character of the message. This would print out the appliance message without the quotation marks and brackets.

```
#Splitting the message by the delimiter and sorting it
split_msg = appliance_msg.split(":")
split_msg_sorted = sorted(split_msg)
#print(split_msg_sorted) //For testing

#Appending the splitted messages into variables
appliance_name = split_msg_sorted[0].split(":")
appliance_status = split_msg_sorted[1].split(":")
#print (appliance_list) //For testing

#If appliance name inside
#Converting to string and stripping first 2 and last 2 characters
appliance_name = str(appliance_name)
appliance_name = appliance_name[2:-2]
#print (appliance_name) //For testing
```

The message sent over looks like this, "Appliance_Name : Status". Therefore, we would first need to split the message by the delimiter ":". Once splitted, we can sort the message into the variables appliance_name and appliance_status and perform a stripping of the first 2 and last 2 characters from these variables.

```

if appliance_name in appliance_list:
    #Translating the position of the appliance name into the appliance_list
    position = appliance_list.index(appliance_name)
    #Converting the status into string and stripping first 2 and last 2 characters
    appliance_status = str(appliance_status)
    appliance_status = appliance_status[2:-2]
    #print (appliance_status) //For testing

    #Since python arrays start with 0, we start with 1 to make it same as the device_id
    position = position + 1
    device_id = str(position)

    if appliance_status == "OFF":
        deviceStatus = "0"
        os.system('curl http://172.16.0.13:8080/db/update/'+device_id+'?Status='+deviceStatus+'')
        Messagereceived=False #Loop breaking
    else:
        deviceStatus = "1"
        os.system('curl http://172.16.0.13:8080/db/update/'+device_id+'?Status='+deviceStatus+'')
        Messagereceived=False #Loop breaking

```

Once split, if the appliance_name is in the list that has been predefined at the top of the script, we would first translate the position of the appliance_name into the appliance_list to see if it appears in the list. Next, we get the device_id by adding 1 to the position variable. The logic behind this is that since python arrays start with a 0, by adding a 1 it would make it the same as the device_id of the appliances.

Now, if the status message sent over is “OFF”, it would define the variable deviceStatus as “0” and perform a curl update into the db. The query would get the variables that have been defined above and use it in its curl statement.

```

else:
    deviceStatus= "0"
    print("Appliance name not recognised")
    print ("Adding Appliance into the Database") #Since appliance not recognised, we need to add it into the db
    os.system('curl http://172.16.0.13:8080/db/add/'+appliance_name+'?Status='+deviceStatus+'')
    appliance_list.append(appliance_name) #Once added, append the new appliance name into the appliance_list to make the list updated and remove double adding
    Messagereceived=False
    print(appliance_list)

```

In the event that the appliance_name is not found in the list, the curl statement would add the appliance into the database. Once added, it would also append the new appliance’s name into the list, thereby updating it and preventing double entry.

Security Relevancy

The sql queries used in both the functions have prepared statements, thereby preventing sql injection attacks.

Vulnerability assessments and resolutions for DB for temperature detector sensor and tv sensor

No.	Attack Surface	Vulnerability	Resolution
1.	SQL Injection	Attackers would be able to issue malicious SQL queries to make changes to the database. This could result in the whole table being deleted, rendering the web application unusable.	The usage of prepared queries will ensure that the parameters are only sent as literal values, instead of executable portions of an SQL query. This would ensure that no malicious SQL queries can be injected using parameters.
2.	Unauthenticated access to the database	Attackers would be able to view the data despite not being authenticated, so long as they know the path of the site which displays the database tables.	The usage of sessions will ensure that users must be authenticated, through logging in using a valid account, before they can have access to view the database tables. This would ensure that outsiders would not have access to sensitive data.

3.	Brute forcing and Denial of Service attacks (DOS)	Attackers would be able to brute force the server to identify application vulnerabilities. They could also flood the server with requests which could render it unusable.	The usage of UFW firewall will ensure that attackers would not be able to send a huge number of requests in a short period of time, by limiting a host's IP address to initiate only 5 connections to the database in 30 seconds.
4	Cross Site Scripting (XSS)	Attackers would be able to insert malicious client-side JS scripts (i.e <script>) to the database. This could potentially render the web application unusable.	The usage of escape strings will remove special characters such as < and >, by converting them to < and >, which will not be interpreted as a valid code. This will ensure that no malicious codes can get executed if an attacker injects such scripts to the database which is intended to be exploit XSS vulnerable web applications.

With the help of Flask, I created a signup function. This allows users to register for an account which will then be added to the users table of my database.

This process is done via a HTML form, where users will fill in their username and password to create an account. The HTML file 'signup.html' is referenced in the 'render_templates' section, which then allows the file to be displayed when users visit '/signup'.

```
#sign up page
@app.route("/signup", methods=["GET", "POST", "PUT"])
return render_template('signup.html', error=error) #render signup.html page from templates folder
```

As the signup function is being built, I added some measures in place to strengthen the security of my database.

The key functions include:

- Ensuring that no duplicate account exists. Users should not be able to create accounts with the same usernames.

A 'SELECT' statement is used to check if the account created has a username that exists. If it doesn't, 'None' will be returned.

```
checkquery=('select * from users where username=%s;') #query to check if account with same username exists.
checktuple = ([username])
cursor.execute(checkquery,checktuple) #prepared query
info = cursor.fetchone() #fetch output returned
if info == None:
```

- A confirm password field. This is especially useful in the case where a user makes an error within the password field, which could potentially lock them out of the created account.

If the user input from both fields do not match, an error is thrown.

```
if request.form['password'] != request.form['confirmpassword']: #checking if password matches confirm password
    error = 'Passwords do not match'
```

- A common authorization password. This will act as an additional layer of security so that only actual members of the household will obtain access to the IoT Sensors

If the authorization password is valid, then a user account can be created.


```
if request.form['authorisationpw'] == "MyHomeIOTSensorDBAccessC0de": #checking if authorisation code matches string MyHomeIOTSensorDBAccessC0de
    try:
```

- Create password requirements. They are set at a minimum of at least 8 characters.

Sign Up

user1

•

Please use at least 8 characters (you are currently using 1 characters).

.....

Sign Up

```
<input type="password" placeholder="Password" name="password" value="{{
request.form.password }}" minlength=8 required> <br><br>
```

- Password hashing. In this project, I used Bcrypt hashing which is a very secure hashing function. It has been proven to be effective against many attacks. In the rare event of a data leak, it can also prevent the user accounts from being accessed without authorization, as attackers would have no way of deciphering the hashed passwords.

Bcrypt hashing uses 'salt', which consists of random bits, to ensure that all the hashed passwords are unique, even if multiple users sign up with the same passwords.

```
password = request.form['password'] #get password from signup form
salt = bcrypt.gensalt() #generate salt
hashedpw = bcrypt.hashpw(password.encode('utf8'),salt) #hash password using bcrypt with salt
```

The bottom image is an example of password being hashed using Bcrypt hashing.

No.	Username	Password
1	hello	\$2b\$12\$OfxsevtVyDeNpB7KAhAYouKJLeKxfI8b9fBTNFCEfcpz5Hzm/5IK

- Prepared SQL queries to safeguard against SQL Injection.

The query would insert the username and password specified by user from the sign-up form into the users table in the database.

```
query="insert into users(username,password) values (%s,%s);" #query to add user to db
tuple1 = (escape(username),hashedpw)
cursor.execute(query,tuple1) #prepared query
db.commit()
```

- Escape string to safeguard against cross site scripting XSS.

I made use of escape function in the jinja2 module, which escapes special characters such as the angled brackets, which is used to enclose HTML tags. This converts them from < and > to < and > respectively. This would effectively protect my web application against XSS attacks.

Example of the successful conversion of "<script>"

```
&lt;script&gt;
```

escape(username) will prevent special characters from being added to the database, which could result in XSS.

```
query="insert into users(username,password) values (%s,%s);"
tuple1 = (escape(username),hashedpw)
```

Login and Sessions Function

Next, I created a login function. This is to authenticate users so that they will be able to gain access to view the various database tables. Like the sign-up feature, I made use of 'render_templates' to display the login form.

When the user is logged in, there will be a session initiated, based on the username of the user. This will allow them to have access to view the different database tables. Users without a valid session initiation would not be able to view the tables. This would enhance the security of the database web application, as unauthenticated users would not be able to have access to the sensitive information of the different IoT sensors.

Finally, when the user clicks on logout, the session will be terminated, and they can no longer have access to the tables.

```
@app.route("/login", methods=["GET", "POST", "PUT"])
```

```
return render_template('login.html', error=error)
```

As the login function is being built, I have added some measures in place to strengthen the security of my database.

Key functions include:

- Checking the passwords from user input in the form against the database records, based on the username specified in the same form.

A 'SELECT' query is used to check for password of username specified by user in login form. A 'Prepared' query is also being used.

Then, `bcrypt.checkpw` is used to check password specified to the hashed password in the database(`hashedPwInDB[0]` will return the password stored in DB).

```
query=('select password from users where username=%s;')
tuple1 = ([username])
cursor.execute(query,tuple1) #prepared query
hashedPwInDB = cursor.fetchone() #fetch output returned
if bcrypt.checkpw(password.encode('utf8'),hashedPwInDB[0].encode('utf8')): #checking password returned against hashed password in db
    session['username'] = request.form['username'] #set session username
    return redirect('/') #redirect if login success
else: #password doesn't match
    error = 'Invalid password'
db.close()
```

```
password = request.form['password'] #get password from login form
```

- Initiating a session, based on the username of the user, if authentication is successful.

If the password specified is valid, there will be a session initiated, and users will be redirected to the index page ('/').

```

if bcrypt.checkpw(password.encode('utf8'), hashedPwInDB[0].encode('utf8')): #checking password returned against hashed password in db
    session['username'] = request.form['username'] #set session username
    return redirect('/') #redirect if login success
else: #password doesn't match
    error = 'Invalid password'
db.close()

```

Data will be displayed only if a valid username is present the user session. Otherwise, there will be a prompt for users to login.

Logged in as user
[click here to log out](#)

Smart TV data LIVE

No.	Appliance	Status	Tv Channel	Edited By	Timestamp
-----	-----------	--------	------------	-----------	-----------

- Termination of user session upon logout

After logout, users will be redirected to the index page, and be prompted to login again. 'Session.pop' will then end the session for the user.

```

@app.route("/logout", methods=["GET", "POST"])
def logout():
    session.pop('username', None)
    return redirect('/')

```

Protection against brute force and denial of service attacks.

To help protect against brute force and DOS attacks, I implemented a limit to the number of requests being sent to the web server using ufw. It is a simple firewall that

leverages on iptables for configuration. It requires just a few commands to set firewall rules.

I allow SSH(port 22) and HTTP(port 80) service, as limiting it would cause issues to the web application, as it is common for hosts to send multiple requests to the server with these essential services to view webpages or initiate connections.

I limit port 8080, which is the port for my DB web application service. This will prevent an overwhelming number of requests sent to the server, effectively protecting against brute force attacks aimed at scanning for application vulnerabilities and DOS attack which affect the availability of the service.

```
root@iot-db:~/iot_vol/iotsec# ufw allow ssh
Rules updated
Rules updated (v6)
root@iot-db:~/iot_vol/iotsec# ufw allow 8080/tcp
Rules updated
Rules updated (v6)
root@iot-db:~/iot_vol/iotsec# ufw allow 80/tcp
Rules updated
Rules updated (v6)
root@iot-db:~/iot_vol/iotsec# ufw enable
Firewall is active and enabled on system startup
root@iot-db:~/iot_vol/iotsec# ufw limit 8080/tcp
Rule updated
```

The following shows the status of the ufw, and the different rules set. It is used to verify that the firewall is running and whether the rules set are in effect.

```
root@iot-db:~/iot_vol/iotsec# ufw status
Status: active

To Action From
--
22/tcp ALLOW Anywhere
8080/tcp LIMIT Anywhere
80/tcp ALLOW Anywhere
22/tcp (v6) ALLOW Anywhere (v6)
8080/tcp (v6) LIMIT Anywhere (v6)
80/tcp (v6) ALLOW Anywhere (v6)
```

Adding, Updating and Displaying Data to the TV Tables

The TV table stores the information of the different activities for my Smart TV IoT device, based on the different actions triggered from IFTTT. This includes the following:

- Appliance name – name of TV

- Status – Whether TV is on or off
- Channel – Current channel TV is on
- Edited By – User which performed a certain action, such as change channel and turn TV On/Off.
- Timestamp – Timestamp of activity.

It consists of 2 components:

- Live table – displays the most recent activity
- Logs table – displays the rest of the activities

Adding data to the DB:

The add function checks to see whether the appliance is the Samsung TV, it will then request the status, channel, who it was edited by, and the timestamp from the MQTT broker.

```
elif appliance == "samsungTv":
    # return "usage: update/sensorid?temperature=VALUE"
    try:
        db = MySQLdb.connect(db_address, db_user, db_password, db_database)
    except:
        return "MYSQL not running"
    cursor = db.cursor()
    try:
        status = request.args.get('status')
        channel = str(request.args.get('channel'))
        editedBy = request.args.get('editedBy')
        timestamp = datetime.now()
```

It then inserts the data with the appliance name into the db using this prepared insert query.

```
query="insert into costDevices.samsungTv(appliance,status,tvChannel,editedBy,timeStamp) values (%s,%s,%s,%s,%s);"
print(query)
tvTuple = (appliance,status,channel,editedBy,timestamp)
cursor.execute(query,tvTuple)
db.commit()
```

Updating data to the DB:

The function checks to see what the appliance is first, it then requests the status, channel and the user who edited it

```

elif appliance == "samsungTv":
    # return "usage: update/sensorid?temperature=VALUE"
    try:
        db = MySQLdb.connect(db_address, db_user, db_password, db_database)
    except:
        return "MYSQL not running"
    cursor = db.cursor()
    try:
        status = request.args.get('status')
        tvChannel = request.args.get('channel')
        editedBy = request.args.get('editedBy')

```

It will then update the data in the database using the prepare update query

```

query="update costDevices.samsungTv set status = '' + status + '', tvChannel = '' + tvChannel + '', editedBy = '' + editedBy + '', timestamp = now() where num = 1;"
print(query)
cursor.execute(query)
db.commit()

```

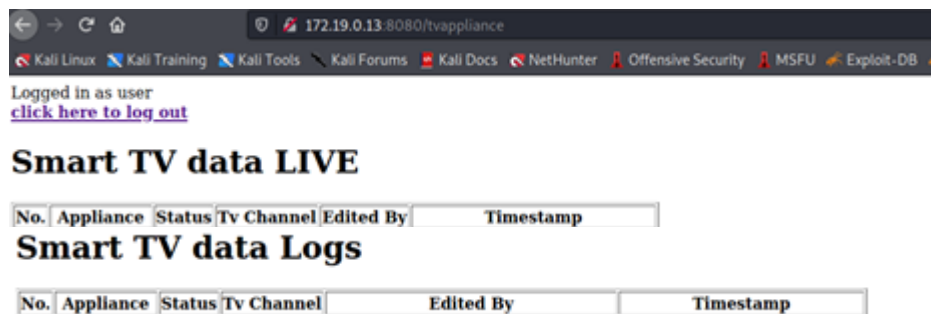
Displaying data to the DB:

```

cursor = db.cursor()
cursor.execute('''select * from samsungTv where num = 1''')
comments = cursor.fetchall()
db.close()
completeTable=<tr >
completeTable += "<th> No. </th>"
completeTable += "<th> Appliance </th>"
completeTable += "<th> Status </th>"
completeTable += "<th> Tv Channel </th>"
completeTable += "<th> Edited By </th>"
completeTable += "<th> Timestamp </th>"
completeTable+="</tr>"
for c in comments:
    completeTable+="<tr >"
    completeTable += "<th>" + str(c[0]) + "</th>"
    completeTable += "<th>" + str(c[1]) + "</th>"
    completeTable += "<th>" + str(c[2]) + "</th>"
    completeTable += "<th>" + str(c[3]) + "</th>"
    completeTable += "<th>" + str(c[4]) + "</th>"
    completeTable += "<th>" + str(c[5]) + "</th>"
    completeTable+="</tr>"

```

The DB will then fetch the data from the database table and display it in a table along with the previous inputs made in a second table



The screenshot shows a web browser window with the address bar displaying '172.19.0.13:8080/tvappliance'. The page has a navigation bar with links to Kali Linux, Kali Training, Kali Tools, Kali Forums, Kali Docs, NetHunter, Offensive Security, MSFU, and Exploit-DB. Below the navigation bar, it says 'Logged in as user' with a link to 'click here to log out'.

The main content area has two sections:

Smart TV data LIVE

No.	Appliance	Status	Tv Channel	Edited By	Timestamp

Smart TV data Logs

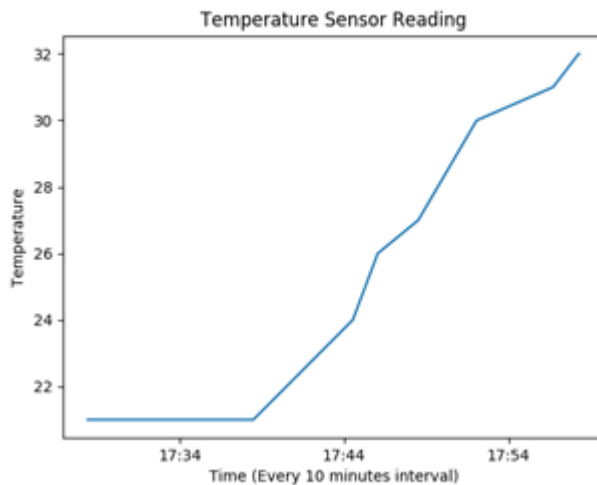
No.	Appliance	Status	Tv Channel	Edited By	Timestamp

Data Visualization

To aid the users with the ease of visualizing a wide range of data that is supplied to them from the various sensors, 1 chart and 1 graph have been created to serve that purpose.

The graph displays the temperature recorded over the time based on a 10-minute interval.

Temperature Reading Graph



```
@app.route("/temp/image")
def showImage():
    """
    Show list of comments with form to submit comments
    """
    try:
        db = MySQLdb.connect(db_address, db_user, db_password, db_database)
    except:
        return "MySQL not running"
    cursor = db.cursor()
    cursor.execute('select * from temperaturesensor')
    comments = cursor.fetchall()
    index = 0
    timestampList = []
    temperatureList = []
    for c in comments:
        index = index + 1
        if index <= len(comments):
            dateTime = datetime.strptime(str(c[1]), "%Y-%m-%d %H:%M:%S")
            temperatureList.append(int(c[2]))
            timestampList.append(dateTime)

    figure = plt.figure()
    axes = figure.add_subplot(1,1,1)
    axes.plot(timestampList, temperatureList)
    axes.set_title("Temperature Sensor Reading")
    timeformat = DateFormatter("%H:%M")
    axes.set_xlabel('Time (Every 10 minutes interval)')
    axes.set_ylabel('Temperature')
    axes.xaxis.set_major_formatter(timeformat)
    axes.xaxis.set_major_locator(MinuteLocator(interval=10))
    stream = BytesIO()
    figure.savefig(stream)
    stream.seek(0)
    return send_file(stream, mimetype='image/png')
```

The pie chart displays the frequency of the temperature range recorded over the time. Of which, the categories have respective temperature range assigned to them. ("COLD"

refers to the temperature below 22°C, “GETTING COLD” refers to the temperature range of 23°C to 25°C, “AVERAGE” refers to the temperature range of 26°C to 29°C, and “HOT” refers to the temperature range of 30°C to 35°C.

```
@app.route("/temp/image2")
def showImage2():
    """
    Show list of comments with form to submit comments
    """
    try:
        db = MySQLdb.connect(db_address, db_user, db_password, db_database)
    except:
        return "MySQL not running"
    cursor = db.cursor()
    cursor.execute('select * from temperaturesensor')
    comments = cursor.fetchall()
    index = 0
    hotCounter = 0
    averageCounter = 0
    gettingColdCounter = 0
    coldCounter = 0
    for c in comments:
        print(c)
        index = index+1
        if index <= len(comments):
            if int(c[2]) >= 30 and int(c[2]) <= 35:
                hotCounter = hotCounter+1
                print(hotCounter)
            elif int(c[2]) >= 26 and int(c[2]) <= 29:
                averageCounter = averageCounter+1
                print("I am working2")
            elif int(c[2]) >= 23 and int(c[2]) <= 25:
                gettingColdCounter = gettingColdCounter+1
            elif int(c[2]) <= 22:
                coldCounter = coldCounter+1
    temperatureRangeList = [(hotCounter), (averageCounter), (gettingColdCounter), (coldCounter)]
    category = ['HOT \n(30°C - 35°C)', 'AVERAGE \n(26°C - 29°C)', 'GETTING COLD \n(23°C - 25°C)', 'COLD \n(<22°C)']
```

```
figure, ax = plt.subplots()
ax.pie(temperatureRangeList, labels = category, autopct='%1.1f%%', startangle=90)
ax.axis('equal')
ax.set_title('Temperature Range Chart')
stream = BytesIO()
figure.savefig(stream)
stream.seek(0)
return send_file(stream, mimetype='image/png')
```

E. Conclusion

Summarize what you have learnt from this project.

Through this project, we learnt how to set up a simulated IOT system architecture. We've also brushed up on our python code and honed our debugging skills. Now that we've experienced how to set up an IOT system, it made us realise how simple and yet complicated it is to actually do so. The final goal may look complicated but once we have broken down the project into smaller parts for each of us to handle, it turns out to be simpler than we thought.

F. Appendix

i. Screenshot of individual completion of AWS Cloud Foundation Badge

GOH KAI XUAN JERICK:



HAN XIAOZHI:



MUHAMMAD ANIQ SYUKRI B MD A:



MUHAMMAD MIKAIL BIN JASMAN:



ZACHARY PHOON JUN ZE:

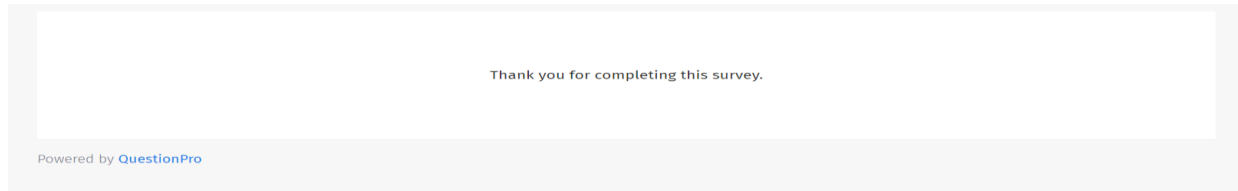


SO CHENG YI, AUGMEN:

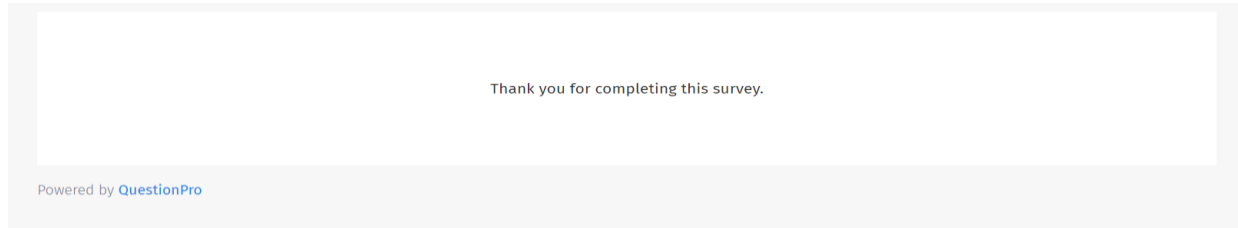


ii. Screenshot of individual completion of subject survey

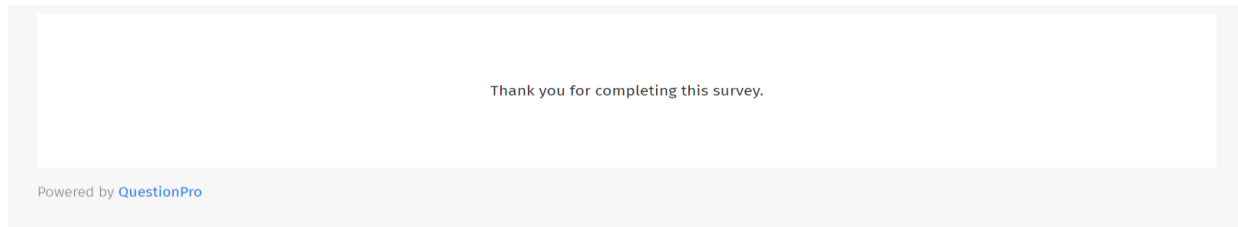
GOH KAI XUAN JERICK:

A screenshot of a survey completion screen. The background is light gray. In the center, there is a white rectangular box containing the text "Thank you for completing this survey." in a small, black, sans-serif font. Below this box, at the bottom of the screen, is a smaller white rectangular box containing the text "Powered by QuestionPro" in a small, blue, sans-serif font.

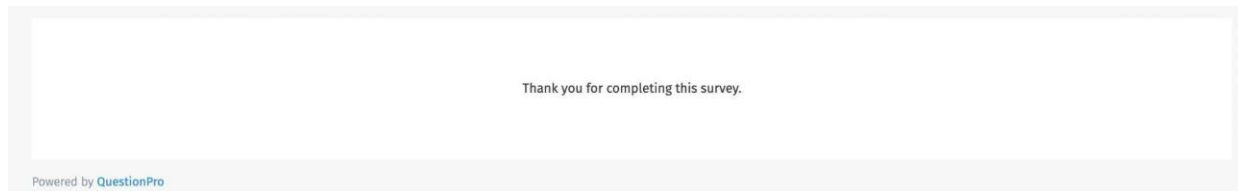
HAN XIAOZHI:

A screenshot of a survey completion screen. The background is light gray. In the center, there is a white rectangular box containing the text "Thank you for completing this survey." in a small, black, sans-serif font. Below this box, at the bottom of the screen, is a smaller white rectangular box containing the text "Powered by QuestionPro" in a small, blue, sans-serif font.

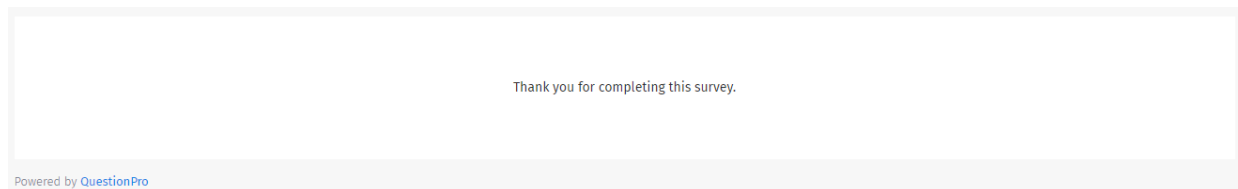
MUHAMMAD ANIQ SYUKRI B MD A:

A screenshot of a survey completion screen. The background is light gray. In the center, there is a white rectangular box containing the text "Thank you for completing this survey." in a small, black, sans-serif font. Below this box, at the bottom of the screen, is a smaller white rectangular box containing the text "Powered by QuestionPro" in a small, blue, sans-serif font.

MUHAMMAD MIKAIL BIN JASMAN:

A screenshot of a survey completion screen. The background is light gray. In the center, there is a white rectangular box containing the text "Thank you for completing this survey." in a small, black, sans-serif font. Below this box, at the bottom of the screen, is a smaller white rectangular box containing the text "Powered by QuestionPro" in a small, blue, sans-serif font.

ZACHARY PHOON JUN ZE:

A screenshot of a survey completion screen. The background is light gray. In the center, there is a white rectangular box containing the text "Thank you for completing this survey." in a small, black, sans-serif font. Below this box, at the bottom of the screen, is a smaller white rectangular box containing the text "Powered by QuestionPro" in a small, blue, sans-serif font.

SO CHENG YI, AUGMEN:

Thank you for completing this survey.

Powered by [QuestionPro](#)

iii. References

Paho MQTT Guide and Documentation:

<http://www.steves-internet-guide.com/into-mqtt-python-client/>

IFTTT Website:

<https://ifttt.com/home>

Telegram Bot Creation Guide:

https://medium.com/@ManHay_Hong/how-to-create-a-telegram-bot-and-send-messages-with-python-4cf314d9fa3e

iv. Code or scripts developed for the respective project components

<https://github.com/Serade12/PowerCal>

END