Raspberry Pi-based Temperature Monitoring System

Final Report

PB14

Project Team:

Student Name(s):	Chong Yijing Isabel	Tan Xin Shi
Student ID(s):	10186106F	10185510K
E-mail(s):	IsabelChong.CYJI@gmail.com	XinShi2001@gmail.com

Project Supervisor:

Name(s):	Mr. Soon Hock Wei
E-mail(s):	SOON_Hock_Wei@np.edu.sg

Summary

This report dictates what Chong Yijing Isabel and Tan Xin Shi have done for their Biomedical Product Design (BPD) Module from September 2020 to January 2021. The report will review the project motivation, its design thinking process, and the features of the product at its final stage.

The Raspberry-Pi Temperature-Attendance Recording System, which is called RaspiTAR for short, is created to assist in temperature monitoring and contactless attendance taking in schools. The aim of the project is to create a system that is not only lower in costs, but easily implemented and accessible by the teachers. With this, the team aims to have a contactless and efficient temperature taking process for students even with face masks on, which minimises any accumulation of people during temperature taking as well as any close human contact.

This report also includes some of the limitations of the project. To conclude, the team members reflect on the values they have learnt or acquired throughout the span of the project and recommends some future implementations which can be further enhanced from the current system.

A copy of the main source codes for the project is placed in GitHub at https://github.com/IsabelChong/bmebpd_pb14.

Acknowledgement

We would first like to express our sincere gratitude towards our Project Supervisor Mr. Soon Hock Wei for carefully guiding us throughout the project. He has been very patient in explaining any doubts we had regarding subjects such as the Python Language and Cloud Databases, which was relevant to our application. He also kindly shared any tutorials and references which had been helpful as we progressed forward with our project. Thank you for giving us the opportunity to work on this project together.

We would also like to express special thanks for Mr. Kou Lip Hong for assisting us with our purchase forms. Thank you for promptly following up with our purchase orders and letting us know of the improvements to our document for a successful approval, which was very crucial for us to complete the project.

Next, we would like to thank Mr. Ong Wai Seng for providing us with helpful tips on various aspects of technicality and on our project documentation.

We would also like to thank Mr Yong for teaching us how to use the 3D printing machines and how to change the settings on the printing software.

Lasty, we would like to give special thanks to our fellow BPD module mates. They have been cooperative in assisting us in the development of our software by giving us permission to take their photos to enhance the facial recognition feature, which is utilised for attendance taking. They have also given us their valuable time to test out our projects without complains.

Table of Contents

Acronyms

Acronyms	Meaning
API	Application Programming Interface
AWS	Amazon Web Services
BPD	Biomedical Product Design
CPU	Central Processing Unit
CSI	Camera Serial Interface
CSV	Comma-Separated Values
DB	Database
FOV	Field of View
HDMI	High-Definition Multimedia Interface
I^2C	Inter-Integrated Circuit
LCD	Liquid-Crystal Display
LPDDR4	Low-Power Double Data Rate Gen4
MIPI	Mobile Industry Processor Interface
RaspiTAR	RaspiTAR Assembled Design - Top View
SES	Simple Email Service
SD	Secure Digital
SDK	Software Development Kit
SNS	Simple Notification System
SDRAM	Synchronous Dynamic Random-Access Memory
OS	Operating System
PSU	Power Supply Unit
VNC	Virtual Network Computing

List of Figures

List of Tables

Chapter 1. Introduction

1.1. Background Information

Elevating body temperatures can be an indication of the onset of an infection. As such, a fever is commonly one of the human body's first reactions to an infection, such as the novel Coronavirus Disease COVID-19. This new virus began its outbreak in Wuhan, China, in December 2019. COVID-19 has since caused a global pandemic affecting several countries all around the world, including Singapore.

In Singapore, the virus has been announced to be on DORSCON Orange in February 2020. Disease Outbreak Response System Condition (DORSCON) is a colour-coded scheme that shows the current situation and allows the government to respond appropriately to any disease outbreak (MOH, 2014). A practice on DORSCON Orange and a general advice from the government is to conduct temperature screening.

Figure 1: DORSCON Levels (TODAY Online, 2020)

Image obtained from: https://www.todayonline.com/sites/default/files/dorscon_alert_levels_0.jpg

People can catch COVID-19 or other air-borne viruses from others who have the virus. The disease spreads primarily through small droplets expelled through the

mouth or nose when a person speaks, coughs, or sneezes. One can get infected with the disease by breathing in these droplets from an infected individual. As the droplets are relatively heavy, they quickly sink to the ground without travelling far. Therefore, it is important to practice safe distancing of 1 meter between 2 individuals to minimise any risk of catching the disease through air-borne means when they are in close contact with each other (WHO, 2020).

1.2. Project Motivation

To contain the COVID-19 virus, several organisations around the world have implemented temperature monitoring systems to check for temperature. Mass temperature screening devices are used in areas with a high footfall area to reduce the time needed for a person to scan their temperature, thus conducting the temperature screening in a more efficient manner. However, these devices can be relatively expensive.

Single point temperature scanners can be a cheaper alternative, such as the one shown in *figure 2*, which costs SGD$24.99 on Amazon (Amazon.sg, n.d.). However, it then produces a problem where people can gather and form a crowd, due to the nature of single point scanning where everyone needs to pause and take their temperature one-by-one. It may even cause social distancing rules to be violated as there would be a lack of space for the many people waiting to get their temperature scanned.

Figure 2: Forehead Infrared Thermometer (amazon.sg, 2021)

Image taken from: https://www.amazon.sg/Forehead-Thermometer-Infrared-Temporal-Function/dp/B089N7F9LX/ref=asc_df_B089N7F9LX/?tag=googleshoppin-22&linkCode=df0&hvadid=408607544148&hvpos=&hvnetw=g&hvrand=15150968296916387674&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9062506&hvtargid=pla-933737628652&psc=1

More expensive solutions are being used in more crowded areas where mass screening is required. These places include shopping malls and even schools. These devices are useful for mass screening for high footfall areas. The team has found out that most malls use Fluke Infrared Cameras for the application of mass temperature screening, like shown in *figure 3*. However, for example, the Fluke Ti480 PRO Infrared Camera costs up to USD$9,999.99 (Fluke, 2021), which is translated to around SGD$13,257. Although this makes temperature screening more efficient, it is a very expensive investment. Also, this process requires the person monitoring the system to always place close attention to the crowd and screen to effectively point out the individual with high temperature when detected by the system.

Figure 3: Temperature Screening at VivoCity Mall (The Straits Times, 2020)

Image taken from: https://www.straitstimes.com/singapore/operators-scramble-to-meet-new-rules

In addition, the team has found out that attendance taking for some classes still uses the traditional pen and paper method. Students must either pass around and sign their name against the physical attendance sheet, or the teachers will call and then record the attendance based on the students' presence. Also, the teachers must personally take note of any late comers. This process can take up to 15 minutes off

the class time, depending on the class size and cooperation. Especially during the ongoing pandemic, any contact and speaking should be minimised to contain any spread of the virus.

As such, this project aims to implement a low-cost attendance taking system which can record the student's attendance and temperature. This system will not only help in targeted temperature screening, which allows the school to immediately obtain information on any students with a fever from the database but allows teachers to take their attendance with more efficiency.

The current group has completed the front-end application. It consists of two main parts. A part for the students to take their attendance, and another part for teachers to view the information from their computers of laptops.

Chapter 2. Project Management Plan

2.1. Work Breakdown Structure (WBS)

The following is the Work Breakdown Structure (WBS) of the project. The WBS is a pictorial representation of the general features which should be implemented into the system. It also highlights the responsibilities each member has for the functionality of the final product, for project progress efficiency. Boxes coloured in orange have been assigned to Isabel, and purple boxes have been assigned to Xin Shi. Tasks which require both team members' cooperation are coloured in pink.

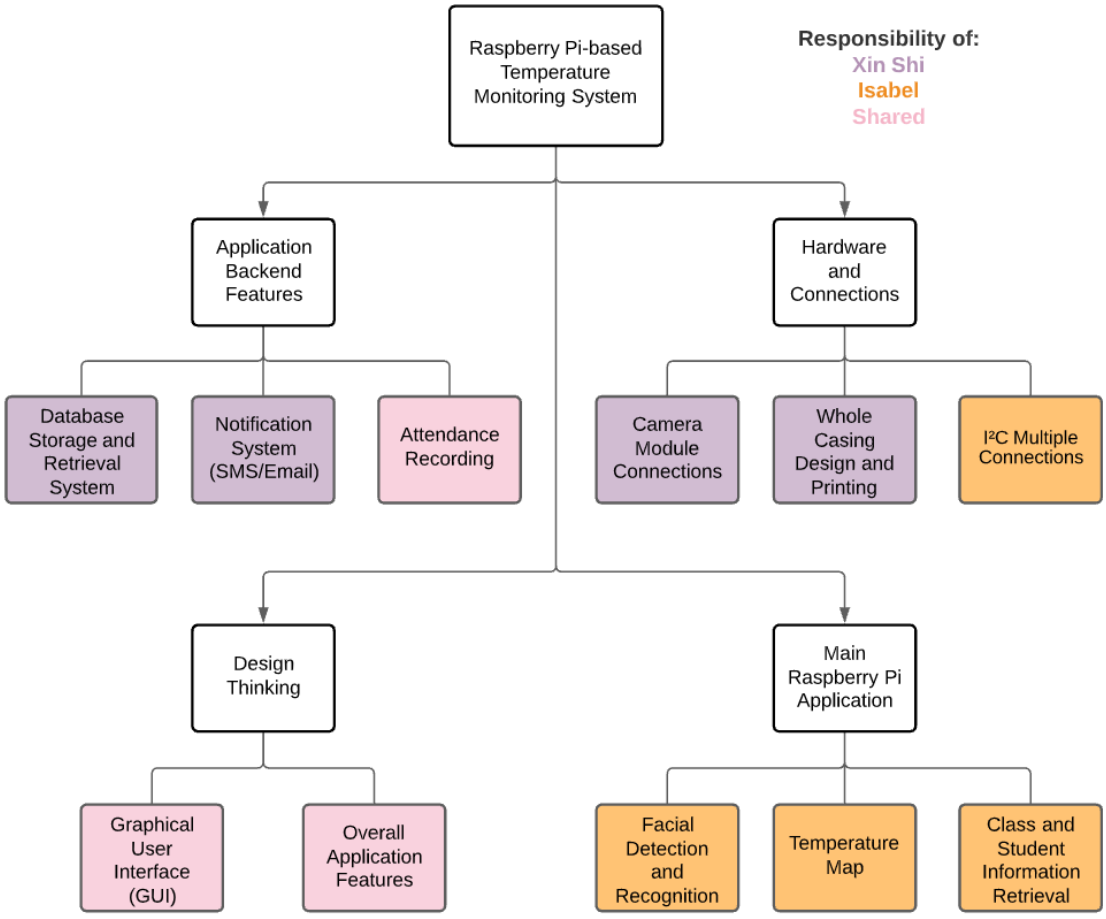


Figure 4: Work Breakdown Structure (WBS) of RaspiTAR project

2.2. Responsibility Assignment Matrix

The Responsibility Assignment Matrix, or RACI Chart, lists down all the project features and the team member in charge or responsible to carry out and finish the task. This matrix was planned with reference to the WBS.

R	Responsible
A	Accountable
C	Consulted
I	Informed

Step	Project Initiation	Isabel	Xin Shi	Mr Soon
1	Overall Project Management	A/R	R	C
2	Project Documentation	R	A/R	C
3	Logistics & Finance Management	R	A/R	C
4	Overall Graphical User Interface (GUI)	A/R	C	I
5	Product Casing Design, Printing, and Implementation (SolidWorks)	C	A/R	I
6	Facial Detection and Recognition (OpenCV & Amazon Rekognition)	A/R	C	C
7	Temperature Map (MLX90641)	A/R	I	I
8	Multiple I^2C Connections	A/R	C	I
9	Managing Raspberry Pi Electronic Connections	C	A/R	
10	Notification System using Email and Mobile Messages (AWS SES and SNS)	I	A/R	I
11	Cloud Database Read & Write (Amazon DynamoDB)	I	A/R	C
12	Obtaining Class and Student Information (Google Sheet, Amazon S3 and Rekognition)	A/R	C	I

Table 1: Responsibility Assignment Matrix

2.3. Gantt Chart

The Gantt Chart is a pictorial representation of the teams' Project Management Timeline, showcasing the ongoing progress and expected time to complete a task. With the chart, the team members were able to have a better sense of what they were supposed to complete by a deadline and when to start each task by. By the end of the project, the tasks which was allocated to each member has been completed. The yellow and pink bars are the timelines set for the tasks of Isabel and Xin Shi, respectively.

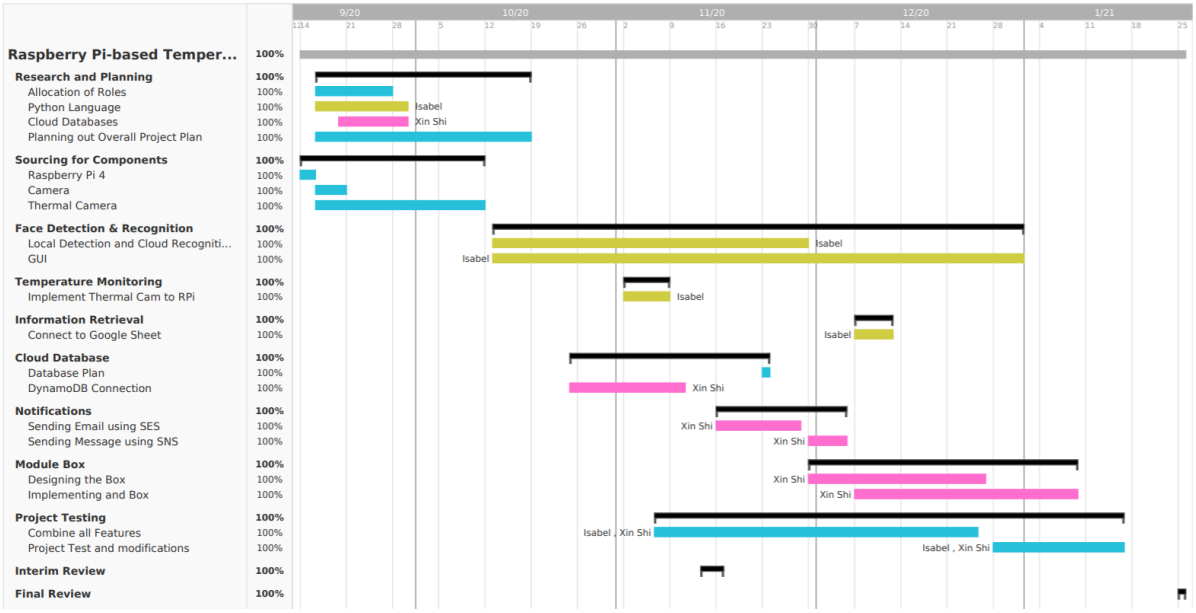


Figure 5: Gantt Chart for RaspiTAR Project

2.4. Overall Connection Plan

For a better understanding on how the system is connection with the various cloud services like Amazon Web Services (AWS) and Google Cloud, and electronic components, the overall connection plan is designed as shown in *figure 6*.

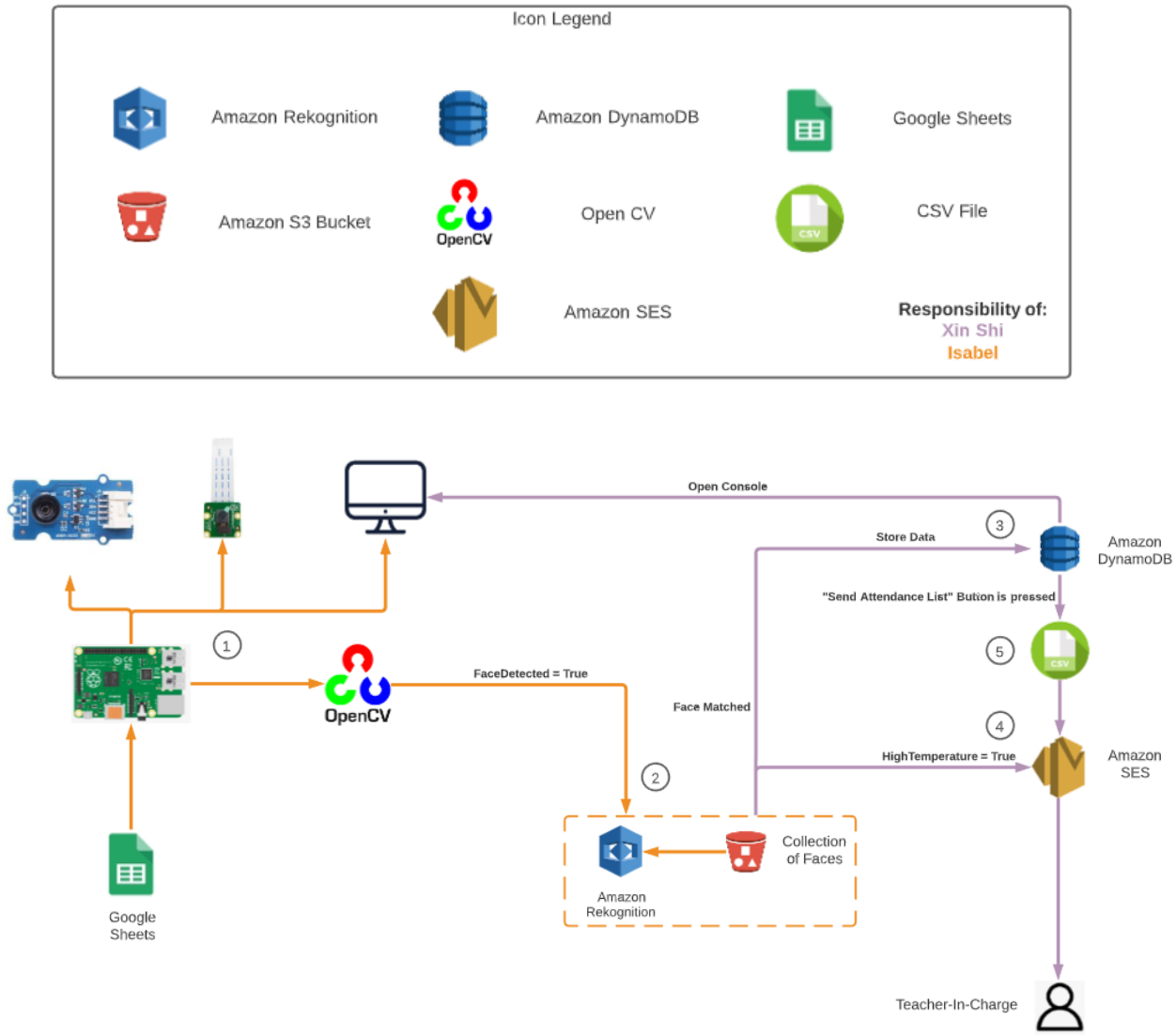


Figure 6: Cloud Processing Design Idea

2.5. Functional and Technical Requirements

The following chart shows the Functional and Technical Requirements identified for the project.

Functional Requirements	Technical Requirements
To accurately detect the temperature of a linear role of humans entering the Field of View using contactless means.	Implementation of MLX90641 IR 16X12 pixels thermal camera with accuracy of ±1°, with a NETD of 0.1K RMS at a 4Hz refresh rate.
To accurately take the attendance of a of humans entering the Field of View.	Implementation of Raspberry Pi Camera V2.1 for computer vision, with Amazon Web Services (AWS) Rekognition APIs for face recognition.
To alert the stakeholder of anyone with a fever (≥ 37.5℃).	The python application shall be capable of a notification system to send email and/or phone message notifications using Amazon SES and/or Amazon SNS to the appropriate personnel.
To capture, store and share data on temperature and attendance record.	System shall operate on a non-SQL server database DynamoDB which receives information from the Raspberry Pi and AWS.
To send the attendance to the teacher-in-charge when required.	System shall be able to send the attendance via a CSV (Comma-Separated Values) file retrieved from DynamoDB, using Amazon SES.
To obtain the class and student information and display it on the Graphical User Interface (GUI).	System shall be able to obtain and show information from Amazon Rekognition and Google Sheet using PyQt5 GUI.

Table 2: Functional and Technical Requirements

Chapter 3. Components & Casing Design

3.1. Electronic Components & Connections

This subchapter describes the components that was selected for the use of the project. While the team was designing the overall idea, the overall price of the system was an important factor to take note of. After much research on the components' reliability, price, accuracy and long-term performance, the team has chosen to add on the following components to our main Raspberry Pi system. The physical components we have acquired are the Raspberry Pi 4 Model B, MLX90641 Thermal Imaging Sensor, LCD 1602A Display, LCD I2C Adapter and Raspberry Pi Camera v2.1.

1) Raspberry Pi 4 Model B

The Raspberry Pi 4 Model B is the newest addition to Raspberry Pi released in June 2019. The Raspberry Pi is a small credit card-sized desktop computer. The Raspberry Pi 4 Model B consists of Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC at 1.5GHz with a 5V DC via USB-C connector. It also consists of 4 USB ports with 40pin GPIO headers. Next, it has a 2-lane Mobile Industry Processor Interface (MIPI) Camera Serial Interface (CSI) camera port which was used to connect to the PiCamera V2.1 (Raspberry Pi, n.d.).

The team has gotten a Raspberry Pi 4 Model B with 8GB LPDDR4-3200 SDRAM. Before starting the Raspberry Pi 4 Model B, the Raspberry Pi OS, which is the operating system for Raspberry Pi models, was downloaded into a class 10 micro-SD card using a software called Raspberry Pi Imager. This was inserted into the Raspberry Pi 4.

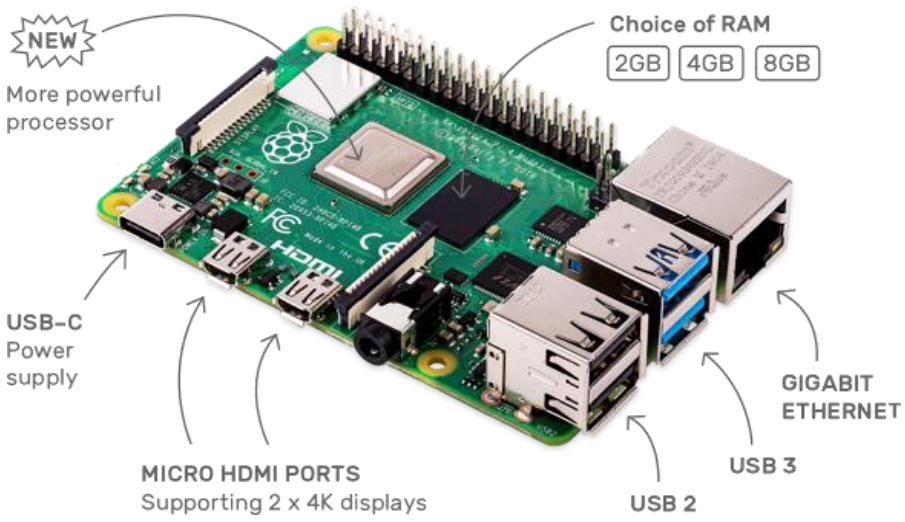


Figure 7: Raspberry Pi 4 Model B (Raspberry Pi, n.d.)

Image obtained from: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/

2) MLX90641 Grove Thermal Imaging Sensor

Temperature recording is the system's main feature. The four main keywords that were thought up for the feature is: contactless, fast, accurate and low-cost. As such, an infrared (IR) thermal camera was the direction to go for as it satisfies all the keywords. IR cameras are sensitive heat sensors which can easily detect any tiny differences in temperatures without being in direct contact with the temperature medium.

The original plan for the project was to get a MLX90640 Thermal Camera, instead of the MLX90641. The main difference between the two cameras is in their resolution. MLX90640 has a 32x24 pixels resolution, and MLX90641 has a 16x12 pixels resolution. However, this camera was globally out of stock and was not able to be shipped in time before the end of the project semester. As such, the only alternative which is the MLX90641 was chosen instead. The MLX90641 Thermal Imaging Sensor has a 16X12 Resolution and a 110° Field of View (FOV).

The Grove MLX90641 Thermal Imaging Sensor is connected to the Raspberry Pi 4 using Inter-Integrated Circuit (I^2C) communication protocol, via Grove cables. The connections to Raspberry Pi will be shown in *chapter 3.1.1.*

3) LCD 1602A Display with LCD I²C Adapter

The LCD 1602A is an LCD module with a display format of 16 characters with 2 lines. It has a white character on blue display and requires a 5V single power supply (SHENZHEN EONE ELECTRONICS CO.,LTD).

For ease of connection, an LCD I²C Adapter was first soldered onto the LCD 1602A. After soldering, instead of the 16 individual connections from the LCD, only the GND, SDA, SCL and VCC connections from the LCD I²C Adapter were required to be connected to the Raspberry Pi computer for communication using I²C protocol, as shown in *figure 8*. The connections to Raspberry Pi will be shown in *chapter 3.1.1.*

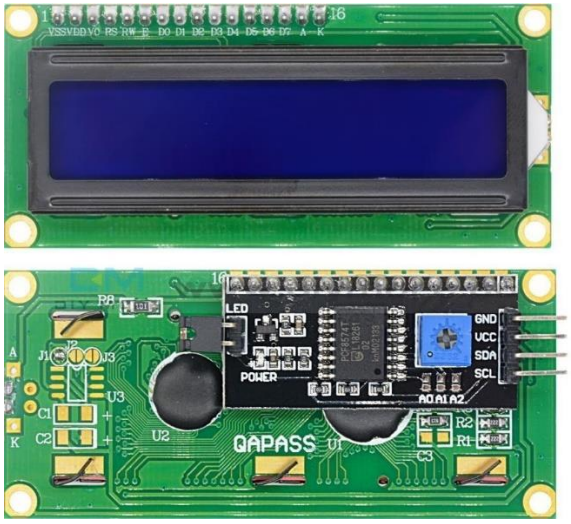


Figure 8: LCD 1602A Frontal and Back Connections to LCD I²C Adapter

4) Raspberry Pi Camera v2.1

The Raspberry Pi Camera Module v2.1 can be used to take high-definition photos and videos. The camera is chosen as it can be easily attached to the Raspberry Pi 4 via a ribbon cable to the CSI port, and is compatible with the Raspberry Pi computer.

This is an 8-megapixel Sony IMX219 image sensor which is capable of providing 3280 x 2464 pixel static images and high-definition video of up to 1080p30.

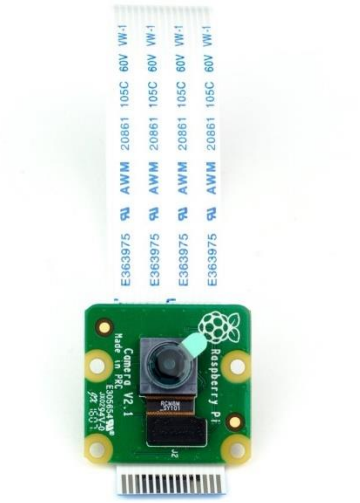


Figure 9: Raspberry Pi Camera v2.1 (PIMORONI, n.d.)

Image obtained from: https://shop.pimoroni.com/products/raspberry-pi-camera-module-v2-1-with-mount?variant=19833929735

3.1.1. Multiple I²C Connections on Raspberry Pi

To connect multiple I²C devices to the Raspberry Pi computer, additional I²C buses were created to make use of other GPIO pins on the computer as I²C SDA and SCL pins. Following the Raspberry Pi OS release, the bus number has to be configured in the following order – 7, 6, 5, 4, 3, with 3 being the lowest bus. With only two I²C devices required to be connected, only buses 4 and 3 were created.

The figure below shows the additions made to the config.txt file, which is a file which is read by the GPU before the ARM CPU and OS are initialised. This allows the Raspberry Pi computer to create the I²C-Bus 4 and I²C-Bus 3 while it boots. GPIO 23, 24, 17 and 27 were chosen as SDA and SCL pins as they are alternate pins which were not utilised. The GPIO delay was configured through trial and error. Initially, 1µs delay was too short or too quick and prompted the LCD display to display a set of zeroes. 2µs delay was found to be the most efficient setting.

```
dtoverlay=i2c-gpio,bus=4,i2c_gpio_delay_us=1,i2c_gpio_sda=23,i2c_gpio_scl=24
dtoverlay=i2c-gpio,bus=3,i2c_gpio_delay_us=1,i2c_gpio_sda=17,i2c_gpio_scl=27
```

Change pic, the us diff ald

Below shows the connection between the two I²C devices – MLX90641 Thermal Imaging Sensor and the LCD 1602A Display. MLX9061 Thermal Imaging Sensor is connected to I²C-Bus 4 and the LCD 1602A Display is connected to I²C-Bus 3. It is to note that the MLX90641 Thermal Imaging Sensor is connected to a 3.3V DC Power Supply and the LCD 1602A Display is connected to a 5V DC Power Supply.

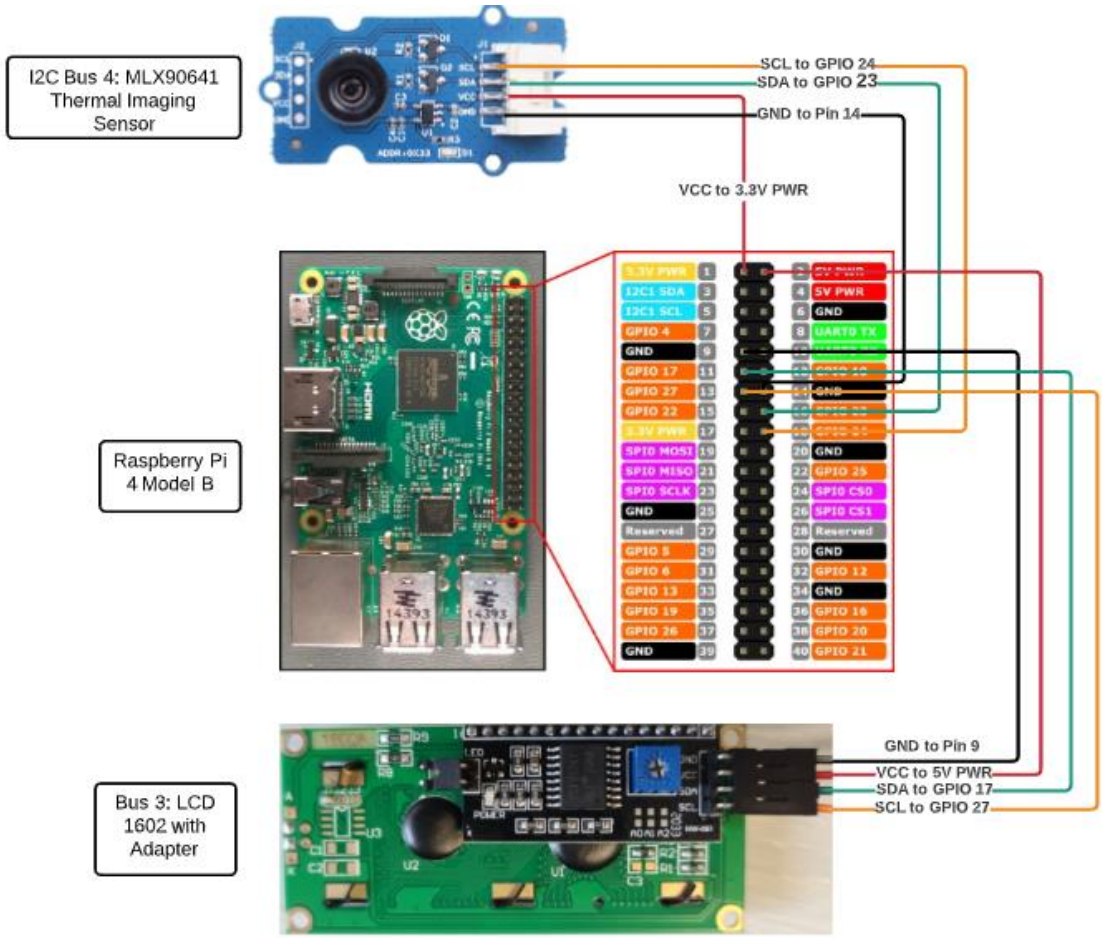


Figure 10: Multiple I²C Connections to Raspberry Pi 4 Model B

The figure below shows the connections to the Raspberry Pi after creating the additional buses, shown from the CLI. The computer recognises the two buses created from “i2c-3” and “i2c-4”. It can be further seen that bus 4 consists of a device with address “33”, which is the MLX90641 Thermal Imaging Sensor, and bus 3 consists of a device with address “27”, which is the LCD 1602A Display.

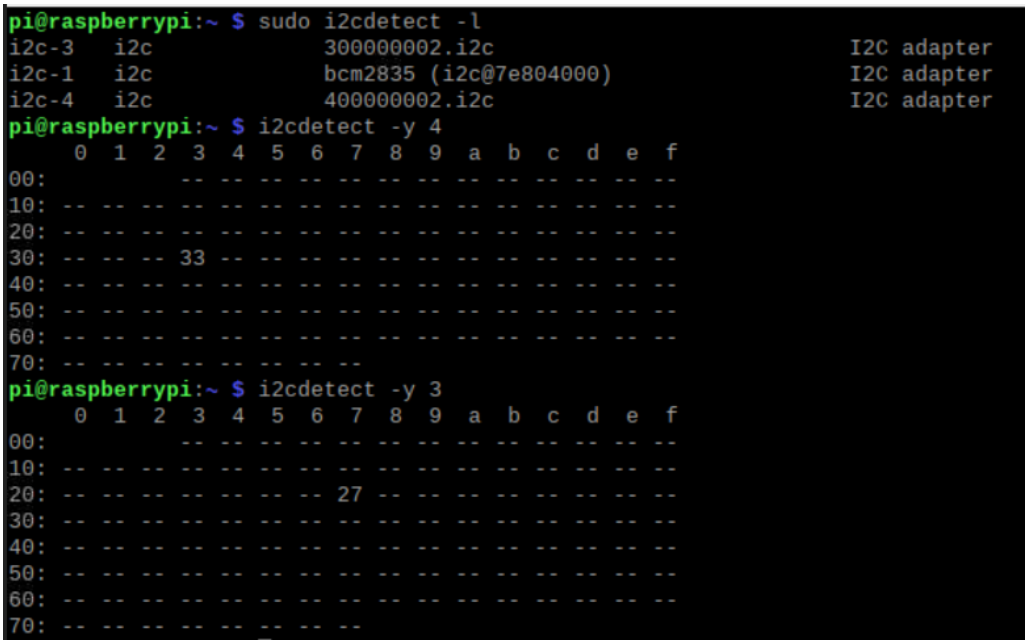


Figure 11: I²C Multiple Bus Creation shown on CLI

After making sure that the devices are connected properly, the backend code for connecting to the two I²C devices were updated to read from their respective I²C-Buses.

3.1.2. Others

Figure 12: Power Supply

According to the Raspberry Pi website, the Raspberry Pi 4 Model B recommends a current capacity of 3.0A, and a 5.1V supply. For this, the team utilised a SAMSUNG charger with a Power Supply Unit (PSU) current capacity of 3.0A, and a 5.0V supply.

3.2. RaspiTAR Outer Case

Figure 13: Reference Raspberry Pi Case

As the team continued with the project, the team realized that the MLX90641 Thermal Imaging Sensor and the Raspberry Pi Camera v2.1 module needed a box with its designated positions to place both modules in.

The store-bought case was unable to hold the two modules, making it difficult for the product to be implemented in real trials. Furthermore, the cameras are sensitive to heat as high temperatures generated from the Raspberry Pi 4 Model B CPU (Central Processing Unit) would spoil the cameras. Also, the jumper wires and ribbon wire connecting the two modules would require a space for them to be contained in. The case was also unable to hold the LCD display was implemented show the student's name, temperature and the last 4 digits of their student ID. This will allow the students to know if they have scanned. Hence, the team has decided to design a casing to solve these problems.

3.2.1. SOLIDWORKS Rx 2019

Figure 14: SOLIDWORKS Rx Logo (Cadimensions, 2019)

Image taken from: https://www.cadimensions.com/wp-content/uploads/2019/08/How-to-Use-Solidworks-RX-to-Create-a-Problem-Capture.jpg

SOLIDWORKS is a popular software for mechatronic engineers used for solid modelling. It is a computer-aided design and engineering software (CAPITOL Technology University, 2020). For the casing, the team has decided to use 3-Dimensional (3D) printing to create the case. The design was done on SOLIDWORKS.

3.2.2. Prototypes

The initial plan was to only redesign the cover of the Raspberry Pi and utilize the base of the store-bought casing. However, this proved to be a challenge

as the general shape of the case is curved and had precise fitting features to place the base and cover together. As such, it was difficult to get measurements to duplicate the fitting features on the new cover to fit the old base casing. Thus, the team decided to create a full casing with both the cover and the base.

Base

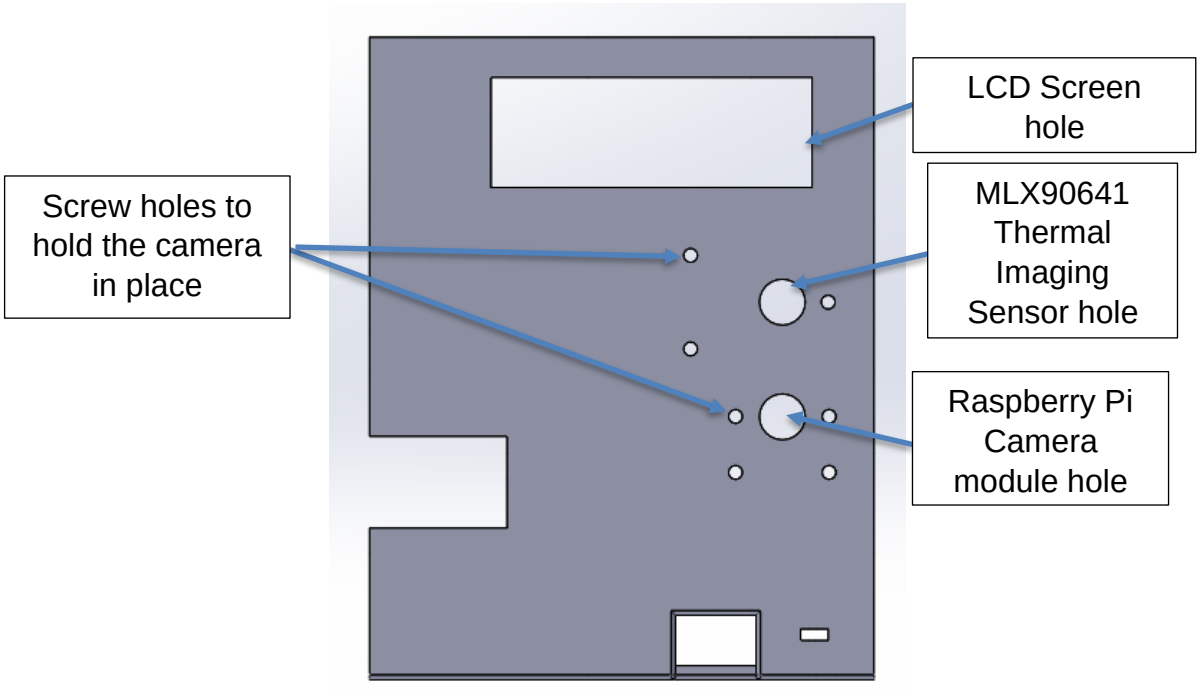


Figure 15: RaspiTAR Casing Base – Front View

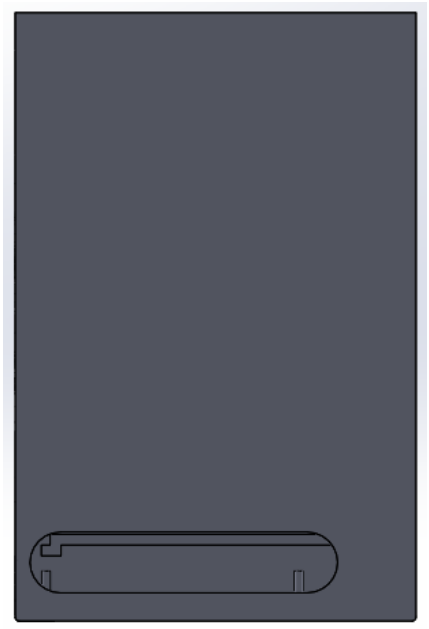


Figure 16: RaspiTAR Casing Base – Right Side View

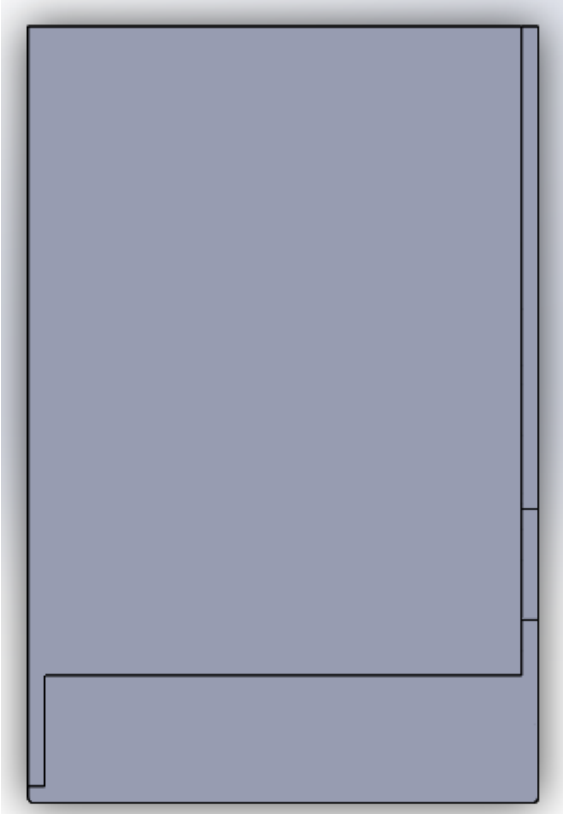


Figure 17: RaspiTAR Casing Base – Left Side View

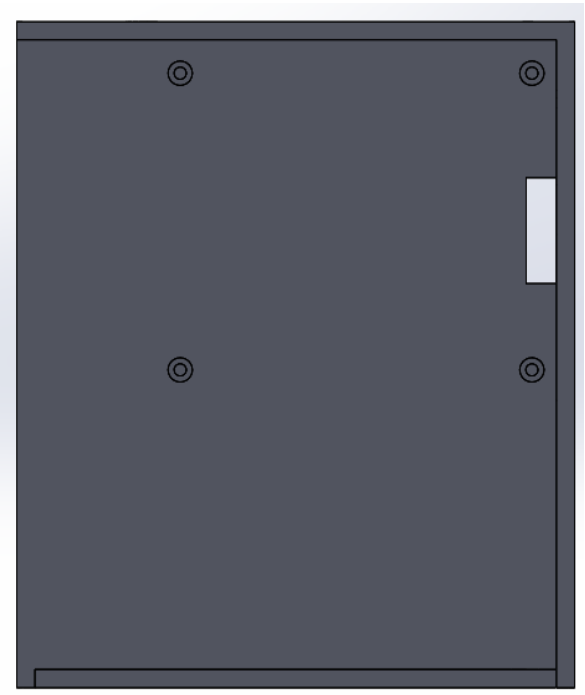


Figure 18: RaspiTAR Casing base – Top View

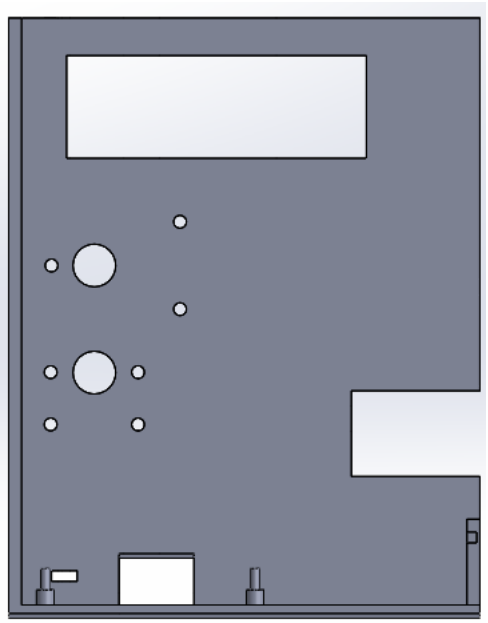


Figure 19: RaspiTAR Casing Base – Back View

Below is a rotatable 3D model. To view this model, ensure that you are viewing this on Microsoft Word. To pan the model, click on the image. After that, click and hold on the  icon while pulling it to the direction you wish.

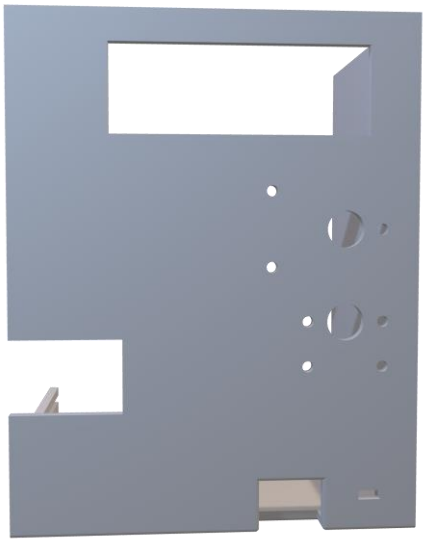


Figure 20: RaspiTAR Casing Base – 3D View

Cover

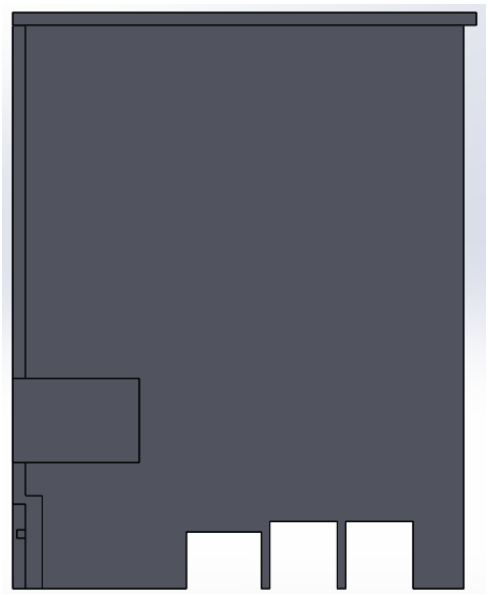


Figure 21: RaspiTAR Casing Cover – Front View

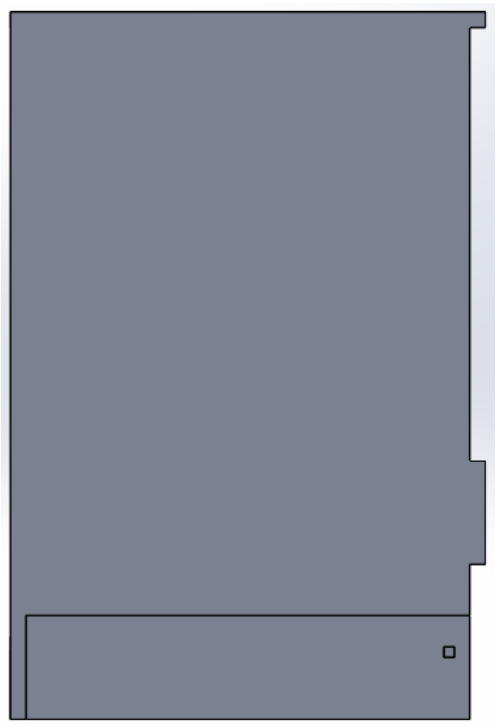


Figure 22: RaspiTAR Casing Cover – Left Side View

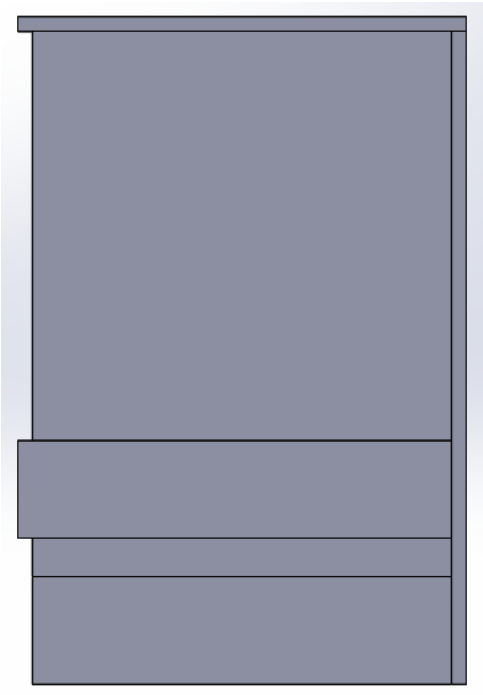


Figure 23: RaspiTAR Casing Cover – Right Side View

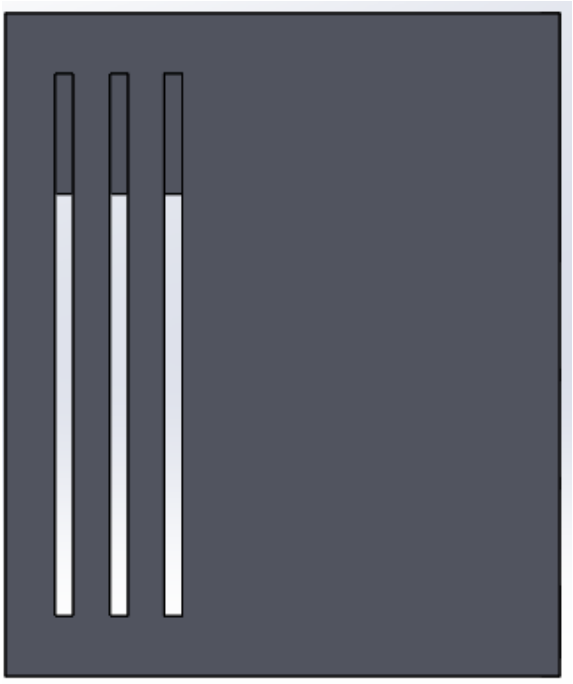


Figure 24: RaspiTAR Casing Cover – Top View

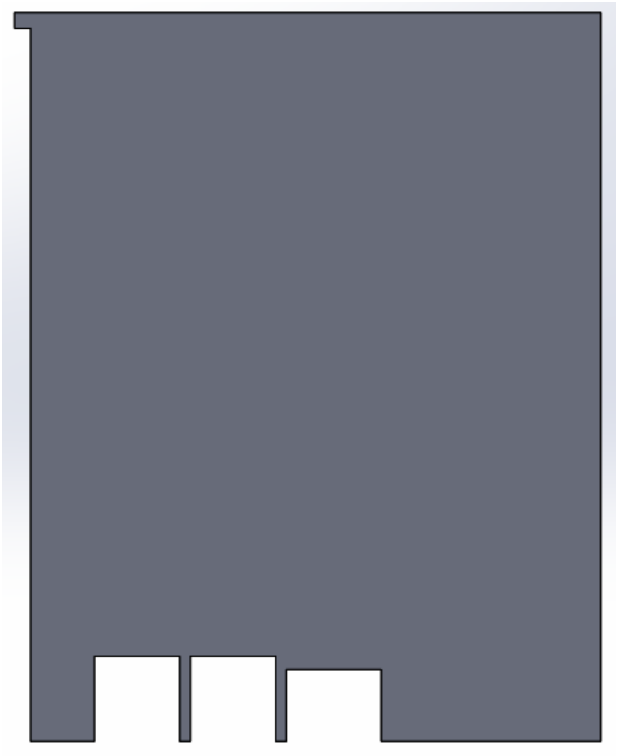


Figure 25: RaspiTAR Casing Cover – Back View

Below is a rotatable 3D model. To view this model, ensure that you are viewing this on Microsoft Word. To pan the model, click on the image. After that, click and hold on the icon while pulling it to the direction you wish.

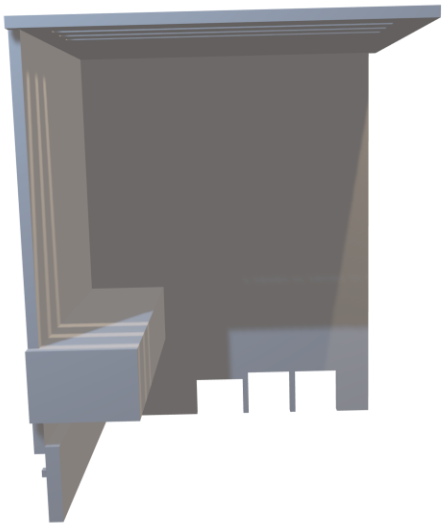


Figure 26: RaspiTAR Casing Cover – 3D View

Fully Assembled Design

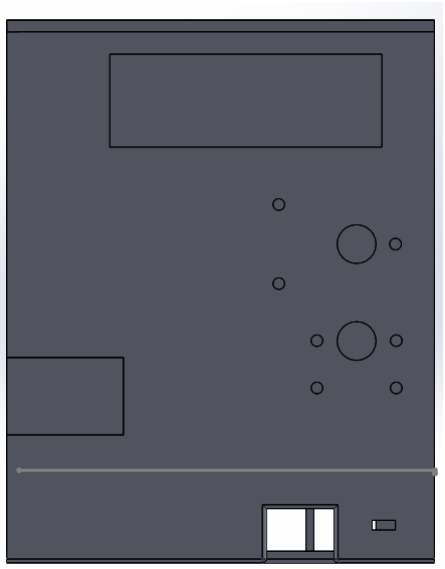


Figure 27 RaspiTAR Assembled Design - Front view

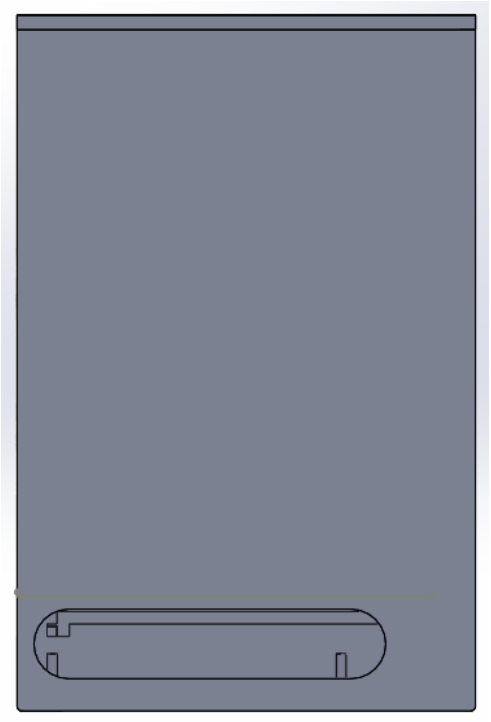


Figure 28 RaspiTAR Assembled Design - Left Side View

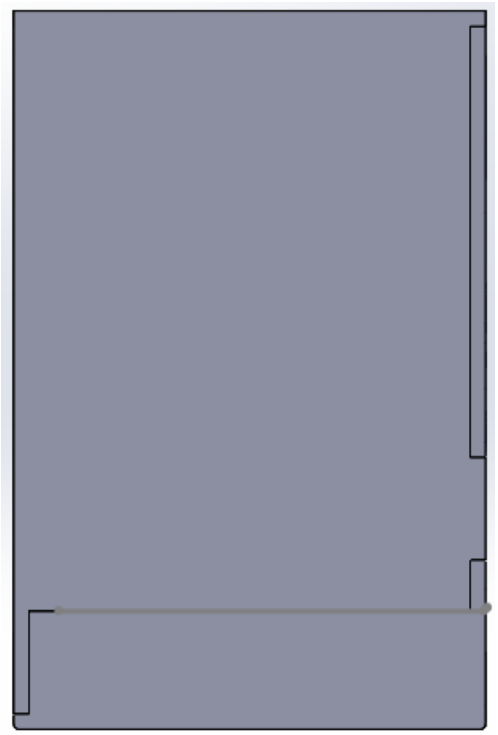


Figure 29 RaspiTAR Assembled Design - Right Side View

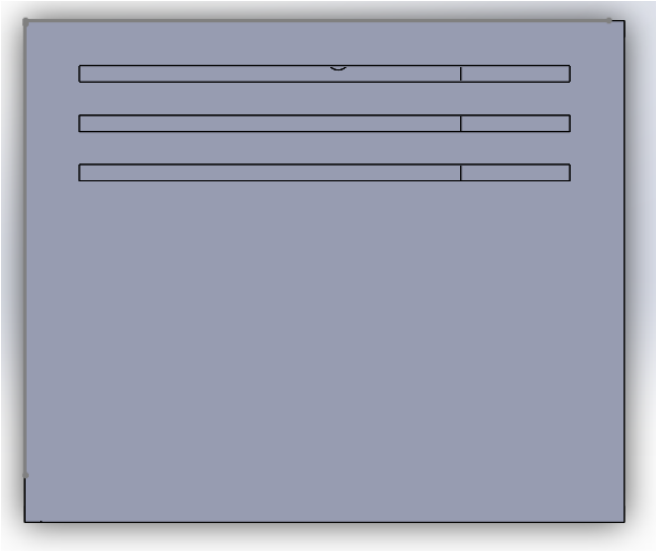


Figure 30 RaspiTAR Assembled Design - Top View

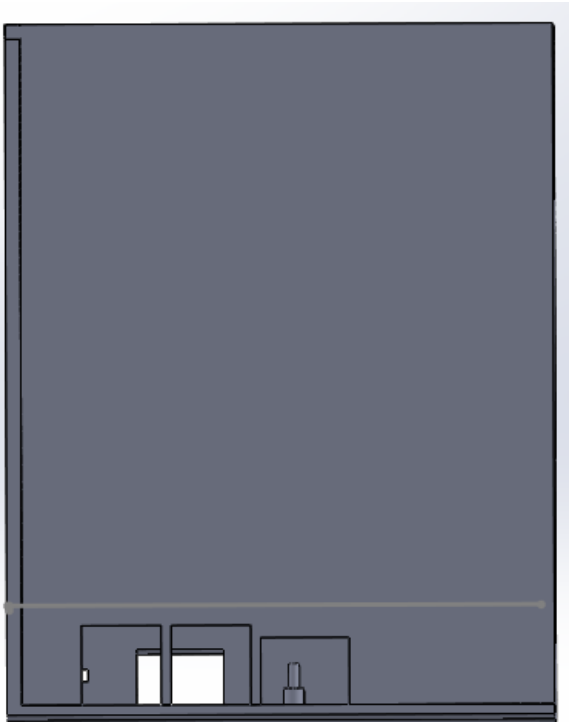


Figure 31 RaspiTAR Assembled Design - Back View

The 3D printing machine, Ultimaker, required the software Ultimaker Cura to slice the model for printing. The settings for the 3D printer can also be changed in this software. The team learnt about that different settings that are required based on the model. For example, if a model is tall, Brim should be selected in the Build Plate Adhesion type instead of skirt as brim is able to stabilize the model more.

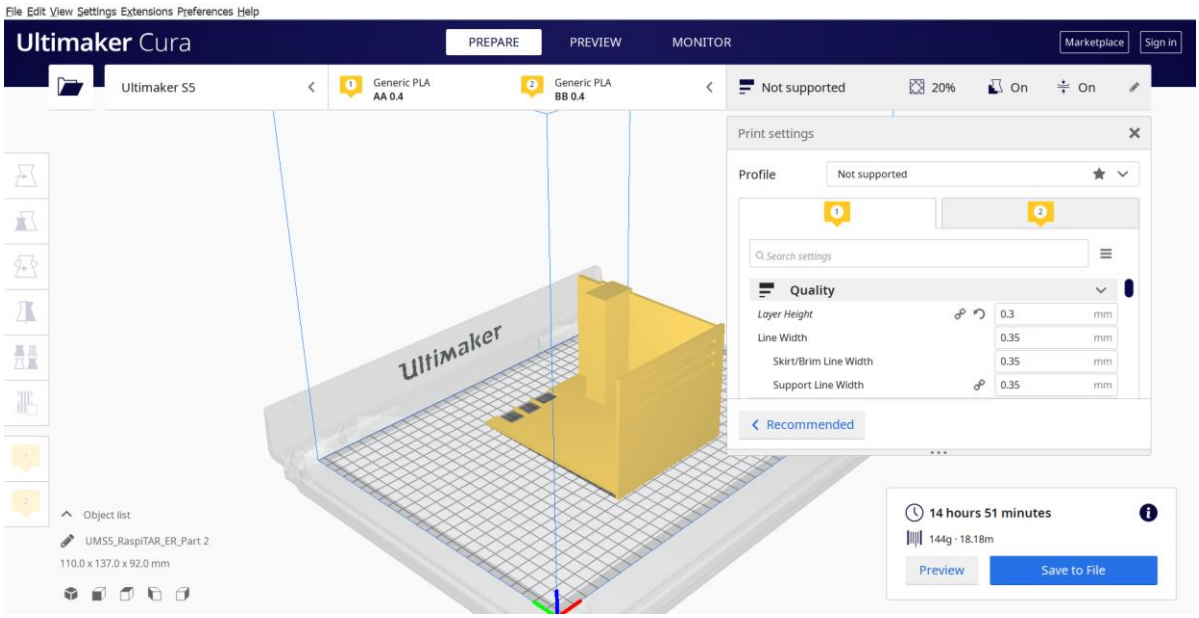


Figure 32: Ultimaker Cura Application Window for RaspiTAR Casing

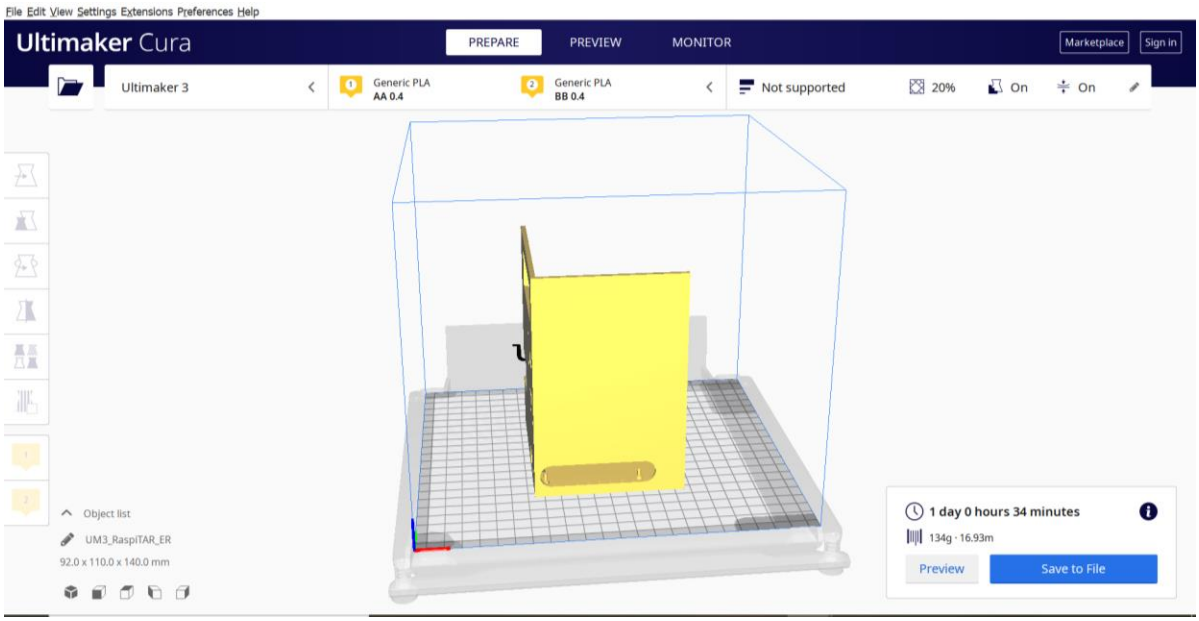


Figure 33: Ultimaker Cura Application Window for RaspiTAR Base

Chapter 4. User Walkthrough

4.1. Initialising the System

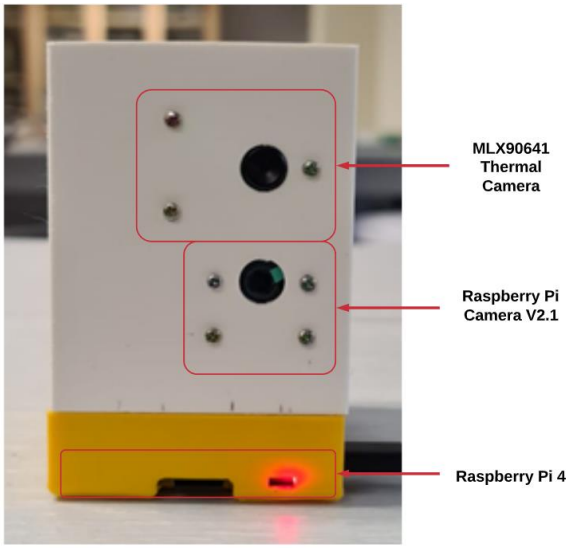


Figure 34: Prototype 3 with all modules in place

Figure 35: Prototype 3 Side View

Note: A new prototype for the casing has been designed, but not shown in this chapter.

The device is first powered on by plugging in a USB-C cable from the power supply. The LED light will light up as shown in *figure 34*. After which, the device interface can be viewed using a laptop via VNC (Virtual Network Computing), after ensuring that the Raspberry Pi 4 Model B and laptop are connected to the same network. VNC

viewer for Google Chrome was used on the laptop for the connection. The IP address is initially obtained from the RealVNC app from the Raspberry Pi 4 Model B shown below in *figure 36*. The address is keyed into the application on the laptop and the user will then be prompted to enter a password.

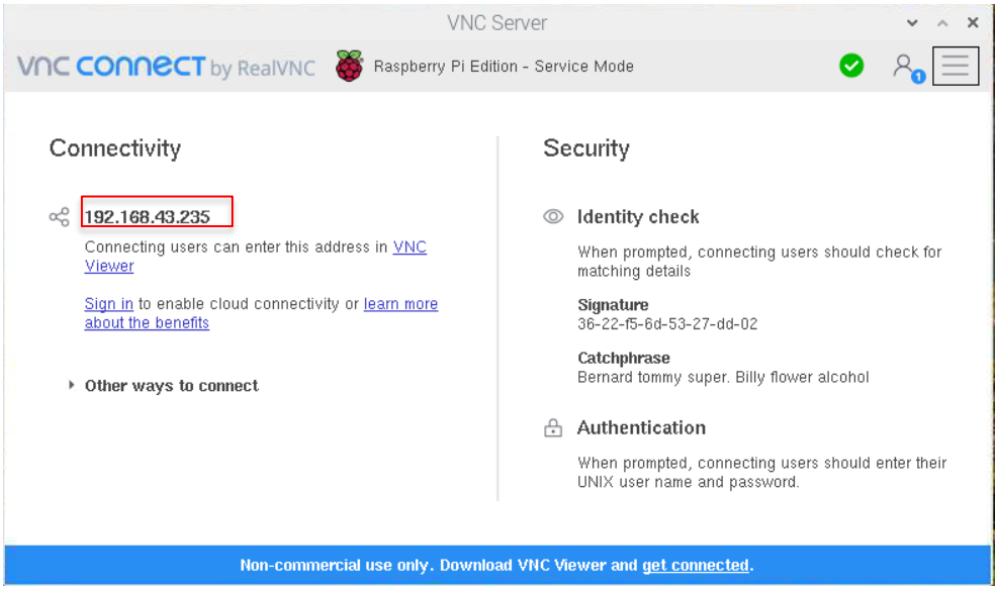


Figure 36: Raspberry Pi RealVNC Application

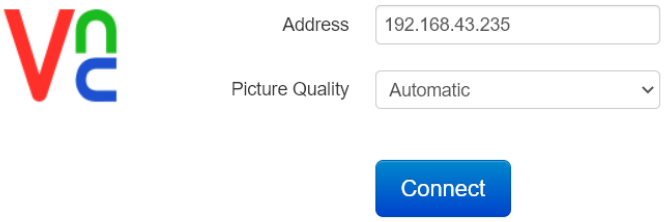


Figure 37: VNC Viewer for Google Chrome Startup Page on Laptop

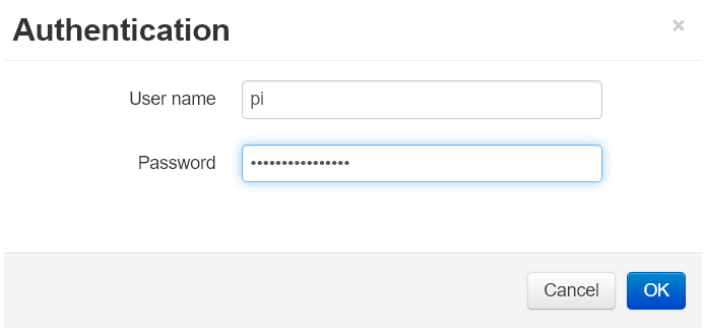


Figure 38: VNC Viewer for Google Chrome Authentication Page on Laptop

After authentication, the user will be brought to the Raspberry OS home page. The program can be ran from the Command Line Interface (CLI) as shown in the figure below. The RaspiTAR application will start initialising and the GUI will initialise.

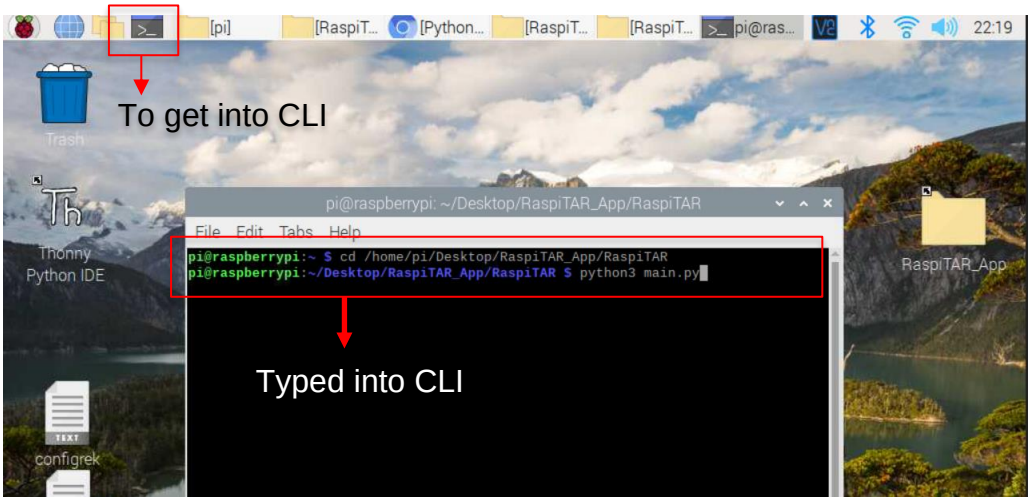


Figure 39: Raspberry Pi Home Page with CLI open

4.2. Understanding the Graphical User Interface (GUI)

When the RaspiTAR application starts running, the following features will first initialise on the GUI. They are the Live Camera, Live Temperature Map, Current Class, Current Time and Current Date.

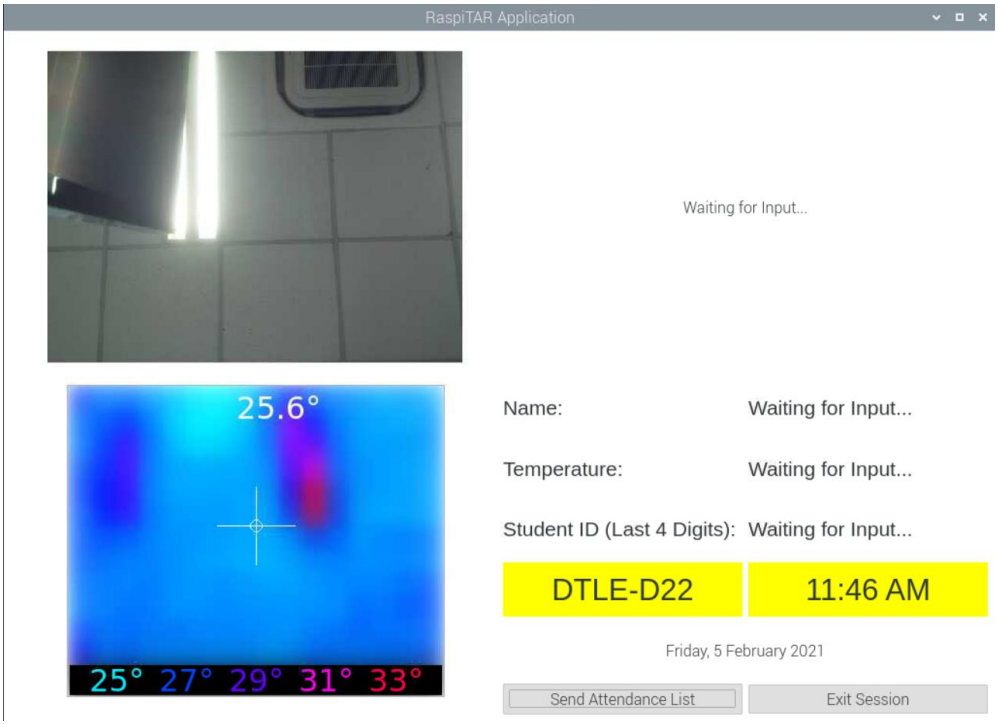


Figure 40: RaspiTAR GUI First Initialisation

After initialisation, the system will start waiting for a face to be detected on the frames inputted from the live camera, which will later be displayed in the labels with “Waiting for Input…” texts.

When a registered student face is detected, the frame is captured and shown on the right-hand side of the GUI. The Student’s Name, Student ID, and Temperature will then be inputted on the GUI, also at the right-hand side of the GUI. Students who are attending the class then follow and go through the process of standing in front of the Live Frame to have their temperature and attendance taken. Their information will be displayed onto an LCD screen to let them know that their attendance is taken and show them their temperature.

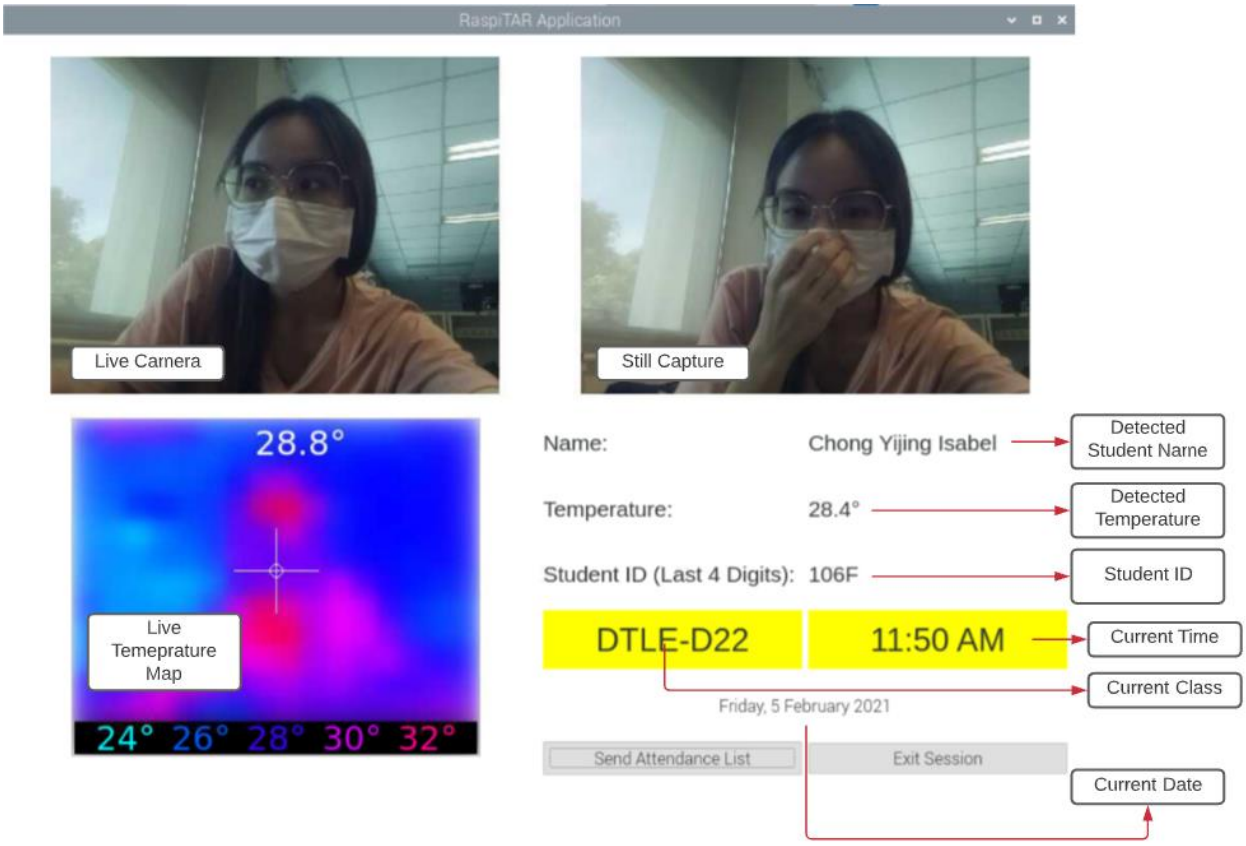


Figure 41: RaspiTAR Application with Labels

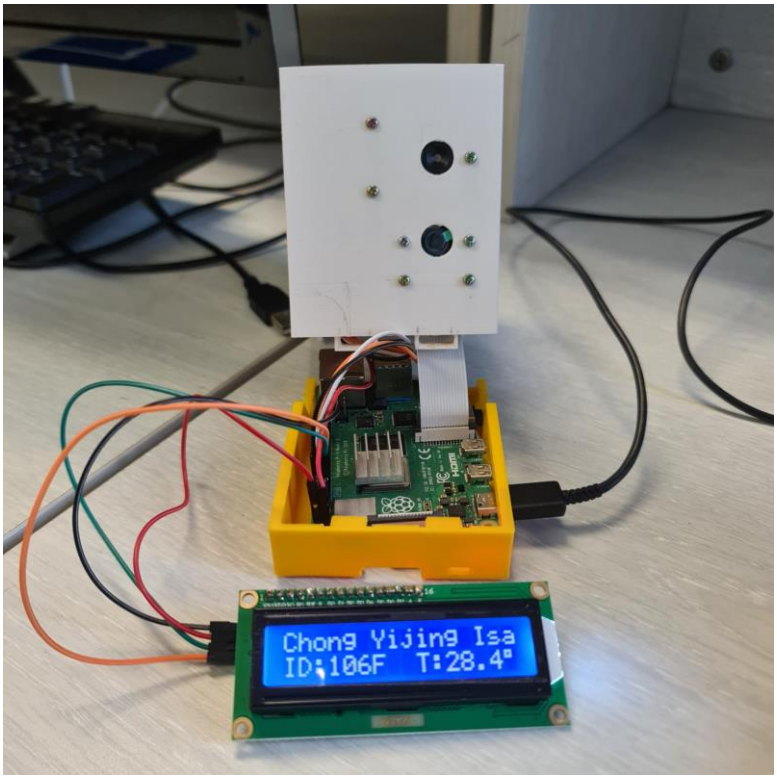


Figure 42: LCD Display after detecting student

4.3. General Step Through

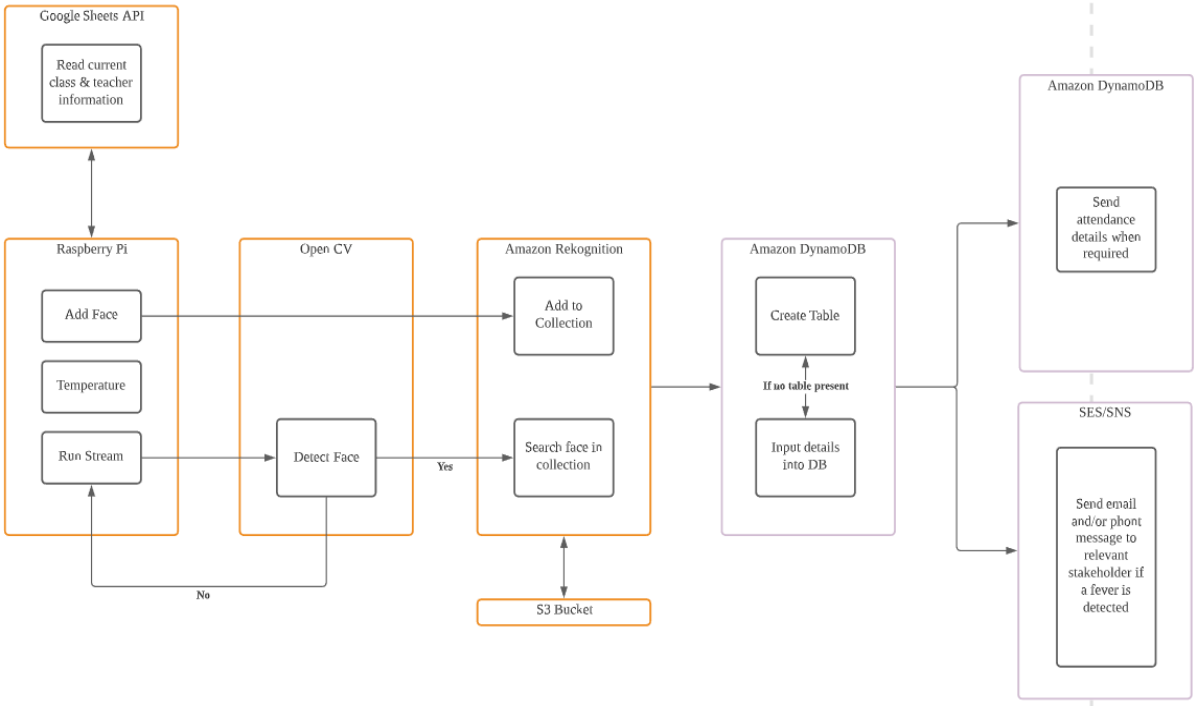


Figure 43: Block Diagram of RaspiTAR

Below is a walkthrough of what happens behind the GUI:

1. Updates current class and obtains teacher information from the connected Google Sheet.
2. Detects for a face using HAAR Cascade from the live frames.
3. Sends the frame to Amazon Rekognition and gets facial bounding metadata. Compares this data to a set of present face metadata sets stored in a collection.
4. If a face is recognised, their names will be retrieved and stored. The captured frame will be shown on the left-hand side of the GUI, at “Still Capture”, as reference to *figure 41*.
5. If a new table has not been created yet, DynamoDB will first create a new table. If the table is present, the information will be stored successfully. DynamoDB will also delete any tables which are 7 months old.
6. If a fever is detected (eg. 38°C), Amazon SES will be called to send an email to the lecturer-in-charge with the relevant student and class details.
7. When the “Send Attendance List” button is called, DynamoDB will obtain the information for that current class and send it to the teacher’s email using Amazon SES via a CSV file.

4.4. Limitations

This subchapter introduces some of the limitations which poses a few challenges to users of this system.

4.4.1. Surrounding Light Settings

As we are using computer vision for facial detection, it is difficult for the algorithm to calculate the facial cascades when they are unclear on the As such, the setup will have to be appropriately lit up with clear facial features shown for the system to work efficiently.

4.4.2. Types of Face Masks

The system can accurately record and take the attendance of students even when only half of their facial features are detected, when they are wearing face masks. This is possible as HAAR Cascade is still able to recognise the overall

shape of a face, and because Amazon Rekognition is still able to calculate the relevant bounding data. However, we have found out that only some colours or patterns works well with the system. Surgical blue, white, and pale coloured masks allow the student' face to be recognised almost instantaneously. However, black, and patterned masks have unstable and varying results. It will either take a very long time for the system to recognise, or the system will not recognise the face at all.

Chapter 5. Code Walkthrough

This chapter explains and walks through the code for the RaspiTAR System. The Graphical User Interface (GUI) of the application is shown in *figure 41.*

5.1. Importing Libraries

main.py

```
1  import faulthandler; faulthandler.enable()
2  import os
3  import io
4  import sys
5  import cv2
6  import boto3
7  import numpy
8  import imutils
9  import argparse
10 import threading
11 import traceback
12 import numpy as np
13 from time import sleep
14 from PIL import Image
15 from PyQt5.QtGui import *
16 from PyQt5.QtCore import *
17 from PyQt5.QtWidgets import *
```

Lines 1-17: Imports required libraries for operating system, functional operations, PyQt5 and threading functions.

```
19 import I2C_LCD_driver
20 from time import *
```

Lines 19-20: Imports required libraries for reading and writing to the LCD 1602A display, which is done via Inter-Integrated Circuit (I^2C) Communication Protocol.

```
22 import seeed_mlx9064x
23 from serial import Serial
```

Lines 22-23: Imports required libraries for reading and converting MLX90641 Thermal Imaging Sensor input, which is done via I^2C Communication Protocol.

```
25 from DynamoAdd import AddItems
26 from SESEmail import SESEmail
27 from CSVSend import CSVEmail
```

Lines 25-27: Imports self-made threading codes to use in threading.

```
29  import gspread
30  from datetime import datetime
31  from oauth2client.service_account import ServiceAccountCredentials
```

Lines 29-31: Imports required libraries for authorising, connecting, and reading Google Sheets, and using Google Sheet and Google Drive APIs.

5.2. Main Graphical User Interface (GUI)

main.py

```
381 class MainWindow(QMainWindow):
382     def __init__(self, parent = None, *args, **kwargs):
383         super().__init__(parent = parent, *args, **kwargs)
384         self.setStyleSheet("background-color: white")
385         self.counter = 0
386 
387         self.setWindowTitle("RaspiTAR Application")
388         self.showMaximized()
389         self.blockLabel = QLabel(" ")
390         self.cameraLabel = QLabel("Initialising Camera...")
391         self.cameraLabel.setAlignment(Qt.AlignCenter)
392         self.snapLabel = QLabel("Waiting for Input...")
393         self.snapLabel.setAlignment(Qt.AlignCenter)
394         # To make up for the layout, IR camera space user
395 
396         self.bigFrame = QFrame(self)
397 
398         self.frame = QFrame()
399         self.frame.resize(100, 200)
400         #self.frame.setStyleSheet("border: 1px solid black")
401         self.namenameLabel = QLabel("Name:")
402         self.namenameLabel.setFont(QFont('Arial', 16))
403         self.nameInputLabel = QLabel("Waiting for Input...")
404         self.nameInputLabel.setFont(QFont('Arial', 16))
405         self.countLabel = QLabel("Waiting for Input...")
406         self.classLabel = QLabel("Loading Current Class...")
407         self.classLabel.setStyleSheet("background-color: yellow")
408         self.classLabel.setFont(QFont('Arial', 23))
409         self.classLabel.setAlignment(Qt.AlignCenter)
410         self.idLabel = QLabel("Student ID (Last 4 Digits):")
411         self.idLabel.setFont(QFont('Arial', 16))
412         self.idInputLabel = QLabel("Waiting for Input...")
413         self.idInputLabel.setFont(QFont('Arial', 16))
```

```
415         self.temptempLabel = QLabel("Temperature:")
416         self.temptempLabel.setFont(QFont('Arial', 16))
417         self.tempInputLabel = QLabel("Waiting for Input...")
418         self.tempInputLabel.setFont(QFont('Arial', 16))
419 
420         self.quitButton = QPushButton("Exit Session", self)
421         self.quitButton.clicked.connect(self.close)
422 
423         self.sendAttendanceButton = QPushButton("Send Attendance
423 List", self)
424         self.sendAttendanceButton.setStyleSheet("padding: 3px;")
425         self.sendAttendanceButton.clicked.connect(self.sendCSV)
426 
427         self.dateLabel = QLabel("")
428         self.dateLabel.setAlignment(Qt.AlignCenter)
429         self.timeLabel = QLabel(" ")
430         self.timeLabel.setStyleSheet("background-color: yellow")
431         self.timeLabel.setFont(QFont('Arial', 23))
432         self.timeLabel.setAlignment(Qt.AlignCenter)
433         # A
434         self.vBoxOne = QVBoxLayout()
435         self.vBoxOne.addWidget(self.cameraLabel)
436         self.vBoxOne.addWidget(self.frame)
437 
438         # B
439         self.vBoxTwo = QVBoxLayout()
440         self.vBoxTwo.addWidget(self.snapLabel)
441         self.gBox = QGridLayout()
442         self.gBox.addWidget(self.namenameLabel, 1,0)
443         self.gBox.addWidget(self.nameInputLabel, 1,1)
444         self.gBox.addWidget(self.temptempLabel, 2,0)
445         self.gBox.addWidget(self.tempInputLabel, 2,1)
446         self.gBox.addWidget(self.idLabel, 3,0)
447         self.gBox.addWidget(self.idInputLabel, 3,1)
448         self.gBox.addWidget(self.classLabel, 4,0)
449         self.gBox.addWidget(self.timeLabel, 4,1)
450         self.gBox.addWidget(self.dateLabel, 5,0, 1,0)
451         self.gBox.addWidget(self.sendAttendanceButton, 6,0)
452         self.gBox.addWidget(self.quitButton, 6,1)
453         self.vBoxTwo.addLayout(self.gBox)
454 
455         # X
456         self.hBoxOne = QHBoxLayout()
457         self.hBoxOne.addLayout(self.vBoxOne)
458         self.hBoxOne.addLayout(self.vBoxTwo)
459 
460         self.widget = QWidget()
461         self.widget.setLayout(self.hBoxOne)
462 
463         # Set the central widget of the Window. Widget will expand
464         # to take up all the space in the window by default.
465         self.setCentralWidget(self.widget)
466         self.show()
```

Lines 384-461: Initialises the Graphical User Interface (GUI) of the RaspiTAR application using PyQt5 using QMainWindow, with the class *MainWindow*. This part of the code sets up a series of layouts and main widgets following the plan as shown in the *figure 44* below. The styles such as font size and font are also initiliased here.

Lines 465-466: Initialises the layout widget as a central QMainWindow widget.

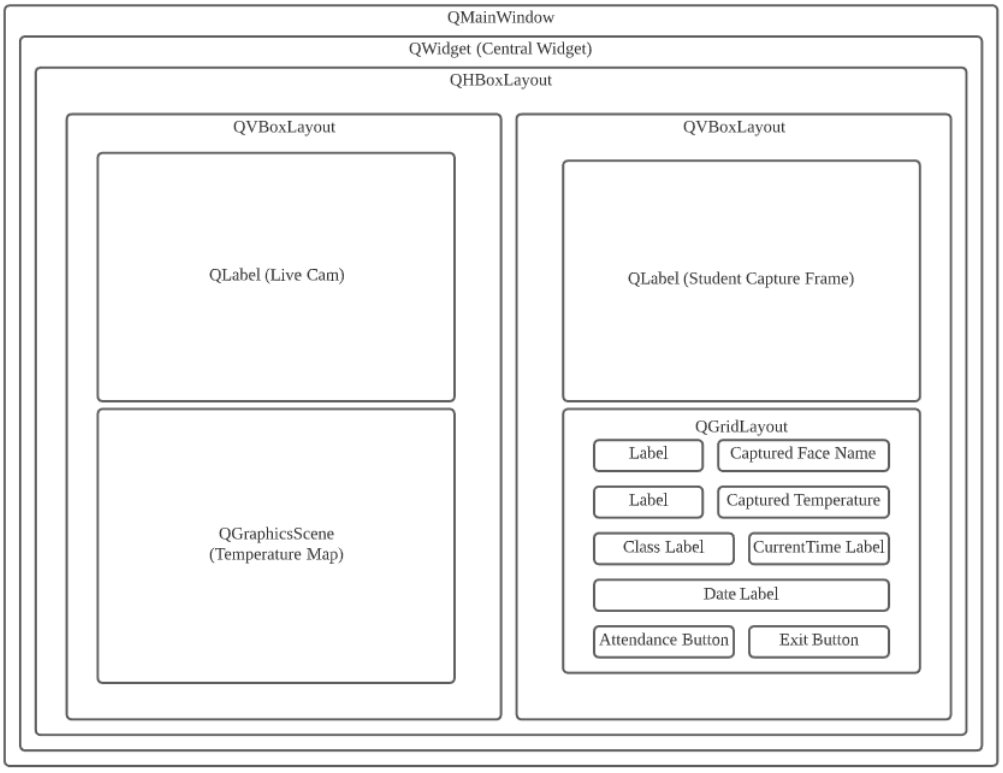


Figure 44: RaspiTAR Main Application Design Layout

```
468         self.threadpool = QThreadPool()
469         print("Multithreading with maximum %d threads" %
469 self.threadpool.maxThreadCount())
470 
471         """ Simulate temperature map running in the background """
472 
473         self.thread = VideoThread()
474         self.thread.change_pixmap_signal.connect(self.init_video)
475         self.thread.detectface_signal.connect(self.worker_function)
476         self.thread.start()
477 
478         self.timer = QTimer()
479         self.timer.setInterval(1000)
480         self.timer.timeout.connect(self.recurring_timer)
481         self.timer.start()
482 
483         self.thread_class = ClassThread()
484         self.thread_class.start()
```

Lines 468: Initialises a threadpool for the facial recognition thread, with a maximum of four thread workers.

Lines 473-484: Initialises two QThreads – *VideoThread* and *ClassThread* and connect the signals to the relevant functions. *VideoThread* continuously refreshes and updates the frame for *cameraLabel* which is used to show the live cam on the application. *ClassThread* continuously reads and updates the current class that is held based on the time given on the connected google worksheet.

Lines 478-481: Initialises *self.timer* which updates the clock and date in the GUI, by calling the *self.recurring_timer* function in the *MainWindow* class.

```
488 scope = ['https://spreadsheets.google.com/feeds',
488 'https://www.googleapis.com/auth/drive']
489         cred = ServiceAccountCreden-
489 tials.from_json_keyfile_name('/home/pi/RaspiTAR_App/RaspiTAR/cli-
489 ent_key.json', scope)
490         client = gspread.authorize(cred)
491 
492         spr = client.open_by_url('https://docs.google.com/spread-
492 sheets/d/1h8BkhqdnqTmtVdKOV13dpC3Is70OiEJ5VmvSFGQuYz4/edit#gid=0 ')
493         wks = spr.worksheet('05-01')
494         global sheet_all_records
495         sheet_all_records = wks.get_all_records()
496         global newlist
497         newlist = dict()
498 
499         for ud in sheet_all_records:
500             newlist[ud.pop('Start Time')] = ud
501 
502         ts = datetime.datetime.now()
503         current_time = ts.strftime("%I:%M %p")
504         current_min = ts.strftime("%M")
505         current_min_int = int(current_min)
```

```
507 if current_min_int < 30:
508             decide_class = "00"
509         else:
510             decide_class = "30"
511 
512         decide_current_time = ts.strftime("%I:" + decide_class +
512 " %p")
513         global current_class
514         global current_teacher
515         global current_teacher_email
516         global current_teacher_phone
517         current_class = str(newlist[decide_current_time]['Class'])
518         current_teacher = str(newlist[decide_current_time]['Teach-
518 er'])
519         current_teacher_email = str(newlist[decide_cur-
519 rent_time]['Email'])
520         current_teacher_phone = str(newlist[decide_cur-
520 rent_time]['Number'])
521         self.classLabel.setText(current_class)
```

Lines 488-521: Initialises and update the current class information including *current_class*, *current_teacher*, *current_teacher_email* and *current_teacher_phone* variables at start-up of application. This is done by connecting the variables to Google Sheet using its API.

Lines 488-493: Initialises the account credentials and google worksheet used.

Lines 494-497: Sets the *sheet_all_records* and *newlist* variables to global so that the system can acquire the teacher information from outside the function. This part of the code is first called to get the variables at system startup.

Lines 499-521: Uses “Start Time” as key to be searched from the Google Sheet. The system will round down the current real time to the nearest 30 minute and match it to the time from the Google Sheet, to get details of the current class. The relevant information such as Class, Teacher Name, Email and Phone Number will be obtained for the calculated time. The google sheet is shown in *figure 45*.

Start Time	Class	Teacher	Email	Number
08:00 AM	CE-D21	CHONG Isabel	xin.isa.raspberry@gmail.com	81667004
08:30 AM	CE-D21	CHONG Isabel	xin.isa.raspberry@gmail.com	81667004
09:00 AM	CE-D21	CHONG Isabel	xin.isa.raspberry@gmail.com	81667004
09:30 AM	EG2-D22	CHONG Isabel	xin.isa.raspberry@gmail.com	81667004
10:00 AM	EG2-D22	CHONG Isabel	xin.isa.raspberry@gmail.com	81667004
10:30 AM	EG2-D22	CHONG Isabel	xin.isa.raspberry@gmail.com	81667004
11:00 AM	DTLE-D22	TAN Xin Shi	xin.isa.raspberry@gmail.com	81667004
11:30 AM	DTLE-D22	TAN Xin Shi	xin.isa.raspberry@gmail.com	81667004
12:00 PM	EM3-D21	ONG Shi En	xin.isa.raspberry@gmail.com	81667004
12:30 PM	EM3-D21	ONG Shi En	xin.isa.raspberry@gmail.com	81667004
01:00 PM	DAELN-T21	LEM Winnie	xin.isa.raspberry@gmail.com	81667004
01:30 PM	DAELN-T21	LEM Winnie	xin.isa.raspberry@gmail.com	81667004
02:00 PM	7WISP-T56	TAN Xin Shi	xin.isa.raspberry@gmail.com	81667004
02:30 PM	7WISP-T56	TAN Xin Shi	xin.isa.raspberry@gmail.com	81667004
03:00 PM	7WISP-T56	TAN Xin Shi	xin.isa.raspberry@gmail.com	81667004
03:30 PM	7WISP-T56	TAN Xin Shi	xin.isa.raspberry@gmail.com	81667004
04:00 PM	7EN-T30	LEM Winnie	xin.isa.raspberry@gmail.com	81667004
04:30 PM	7EN-T30	LEM Winnie	xin.isa.raspberry@gmail.com	81667004
05:00 PM	CE-D21	PANG Yi Yi	xin.isa.raspberry@gmail.com	81667004
05:30 PM	CE-D21	PANG Yi Yi	xin.isa.raspberry@gmail.com	81667004
06:00 PM	CE-D21	PANG Yi Yi	xin.isa.raspberry@gmail.com	81667004

Figure 45: Google Sheet Example

```
524     def recurring_timer(self):
525         self.dateDate = QDate.currentDate()
526         self.timeTime = QTime.currentTime()
527         self.date = self.dateDate.toString(Qt.DefaultLocaleLong-
527 Date)
528         global time_now
529         time_now = self.timeTime.toString(Qt.Default-
529 LocaleShortDate)
530         self.dateLabel.setText(self.date)
531         self.timeLabel.setText(time_now) #variable to be read
532         self.classLabel.setText(current_class)
```

Lines 524-532: Initialises the function *recurring_timer* which is called at an interval of one second starting from the application run. This will update the current time, date, and current class and display them on the GUI. *time_now* is set as a global variable as it is shared across the code in main.py.

```
534     def init_video(self, cv_img):
535         """Updates the labelCam with a new opencv image, Continuous
535 showing"""
536         qt_img = self.convert_cv_qt(cv_img)
537         self.cameraLabel.setPixmap(qt_img)
538 
539     def convert_cv_qt(self, cv_img):
540         """Convert from an opencv image to QPixmap"""
541         rgb_image = cv2.cvtColor(cv_img, cv2.COLOR_BGR2RGB)
542         h, w, ch = rgb_image.shape
543         bytes_per_line = ch * w
544         convert_to_Qt_format = QImage(rgb_image.data, w, h,
544 bytes_per_line, QImage.Format_RGB888)
545         p = convert_to_Qt_format.scaled(431, 321, Qt.KeepAspectRa-
545 tio)
546         return QPixmap.fromImage(p)
```

Lines 534-546: Initialises the functions *init_video* and *convert_cv_qt* which are called for the *VideoThread* QThread class. *Init_video* updates *cameraLabel* with the live camera view and *convert_cv_qt* converts the input image using OpenCV to QPixmap which is returned to the *init_video* function and displayed in *cameraLabel*.

```
548     def closeEvent(self, event):
549         result = QMessageBox.question(self,
550                       "Confirm Exit",
551                       "Are you sure you want to exit?",
552                       QMessageBox.Yes| QMessageBox.No)
553         event.ignore()
554 
555         if result == QMessageBox.Yes:
556             self.thread.stop()
557             event.accept()
```

Lines 548-557: Initialises the *closeEvent* function. When the application is prompted to terminate or close, it will confirm the event with a messagebox.

```
559     def execute_this_fn(self, current_frame):
560         image = cv2.resize(current_frame, (500, 500))
561         is_success, im_buf_arr = cv2.imencode(".jpg", image)
562         byte_im = im_buf_arr.tobytes()
563 
564         try:
565             client=boto3.client('rekognition')
566             response=client.search_faces_by_image(
567                 CollectionId = 'bmecentertrial',
568                 Image={
569                     'Bytes': byte_im
570                 },
571                 MaxFaces = 1,
572                 FaceMatchThreshold=95.0,
573                 QualityFilter='AUTO'
574             )
575 
576             cv2.imwrite('/home/pi/Desktop/RaspiTAR_App/Lo-
576 calFile/Image.jpg', image)
577             #FaceId = response['FaceMatches'][0]['Face']['FaceId']
578             #Confidence = response['SearchedFaceConfidence']
579             global Name
580             global StudID
581             iniName = response['FaceMatches'][0]['Face']['Exter-
581 nalImageId']
582             splitvar = iniName.split("-", 1)[0]
583             StudID = iniName.split("-", 1)[1]
584             Name = splitvar.replace("_", " ")
585             return current_frame
586 
587         except:
588             print("No match/No internet connection")
```

Lines 559-588: Initialises the *execute_this_function* function. This function is called as the execution function of the threadpool initialised in line 445.

Lines 565-574: Calls AWS Rekognition API *search_faces_by_image* to compare faces to a face collection.

Lines 576: Saves the current frame into a local file, as a .jpg file. This will be retrieved to be sent together as an email to the teacher using *current_teacher_email,* when a student with fever is detected.

Lines 576-588: Sets the global variable for *Name,* which will be used to display on the GUI, and passed on to Amazon DynamoDB, Amazon SES, and Amazon SNS to send relevant information including the detected students' name.

```
590 def print_output(self, s):
591         temp = cneter
592         global preName
593         preName = ""
594 
595         try:
596             ts = datetime.datetime.now()
597 
598             if Name is not preName:
599                 self.nameInputLabel.setText(Name)
600                 self.idInputLabel.setText(StudID)
601 
602                 snapimage = self.convert_cv_qt(s)
603                 self.snapLabel.setPixmap(snapimage)
604                 self.tempInputLabel.setText(str(temp)+ "°")
605                 x = threading.Thread(target=print_lcd,
605 args=(temp,)) #thread not working
606                 x.start()
607                 preName = Name
608             else:
609                 pass
610 
611             try:
612                 AddItems(current_class, StudID, Name, temp).start()
613             except:
614                 pass
615 
616             if temp > 37.5:
617                 SESEmail(ts.strftime("%Y-%m-%d %H:%M"), Name,
617 StudID, temp).start()
618 
619         except:
620             pass
```

Lines 561-581: Initialises the *print_output* function. This function is called when a thread from the threadpool is receiving results from AWS and its response after the thread is completed. When there is a successful recognised face, *print_output* updates the information shown on the GUI.

Lines 598-697: Makes sure that the LCD does not reprint the same information.

Lines 612: Calls the *AddItems* thread to add items into DynamoDB.

Lines 616-617: Calls the *SESEmail* thread to send an email to a teacher when high fever (37.5° in this code) is detected.

```
622     def thread_complete(self):
623         print("--> Rekognition Thread Completed")
624         self.thread_class.start()
625 
626     def worker_function(self, flip):
627         worker = Worker(self.execute_this_fn, flip)
628         worker.signals.result.connect(self.print_output)
629         worker.signals.finished.connect(self.thread_complete)
630         self.threadpool.start(worker)
```

Lines 622-624: Initialises the *thread_complete* function which is called whenever a worker function sends a “finished” signal.

Lines 626-630: Initialises the *worker_function* function which initialises the worker thread roles upon receiving certain signals which were connected and called in class *VideoThread*. The *self.execute_this_fn* function is called when a worker is initialised. *self.print_output* is called while the worker is still working. *self.thread_complete* is called when a worker finishes its functions and returns a “finished” signal.

```
632     def sendCSV(self):
633         Present_Month = datetime.datetime.now().month
634         Month_dict = { 1: "January", 2: "February", 3: "March", 4:
634 "April", 5: "May", 6: "June", 7: "July", 8: "August", 9: "Septem-
634 ber", 10: "October", 11:"November", 12:"December"}
635         strPresent_Month = Month_dict.get(Present_Month)
636         try:
637             CSVEmail(strPresent_Month, current_class, cur-
637 rent_teacher, current_teacher_email).start()
638             QMessageBox.information(self,
639                   "Attendance Sheet",
640                   "Attendance Sheet is sent for: \nTeacher-In-
640 Charge: "+ current_teacher + " \nEmail: "+ current_teacher_email
640 +"\nClass: "+ current_class +"\nTime Sent:" + time_now)
641 
642         except:
643             QMessageBox.warning(self,
644                   "Error",
645                   "Failed to send attendance sheet. No Entry.
645 Please try again.")
```

Lines 632-645: Initialises the *sendCSV* function which is called when the *sendAttendanceButton* is pushed. This function creates the template email and attaches the relevant information such as a snapshot, name, class and time sent to the teacher as notification.

```
749 def run():
750     global minHue
751     global maxHue
752     global ChipType
753 
754     port = 'I2C'
755     ChipType = 'MLX90641'
756     minHue = 180
757     maxHue = 360
758 
759 
760     app = QApplication(sys.argv)
761     GUI = MainWindow()
762     view = painter(GUI)
763 
764     dataThread = DataReader(port,ChipType)
765     dataThread.drawRequire.connect(view.draw)
766     dataThread.start()
767 
768     GUI.show()
769     sys.exit(app.exec_())
770 
771 if __name__ == "__main__":
772     run()
```

Lines 749-772: Initialises the *run* function. This function is called from the main thread. It initialises the global variables used for calling the *DataReader* and later the *painter* class to initialise the Temperature Map formed using input from the MLX90641 Thermal Imaging Sensor Sensor Array.

Lines 771-772: These two lines calls the run function to start up the application in the main thread.

5.3. Threading (QThread)

main.py

```
649 class VideoThread(QThread):
650     change_pixmap_signal = pyqtSignal(np.ndarray)
651     detectface_signal = pyqtSignal(object)
652 
653     def __init__(self):
654         super().__init__()
655         self._run_flag = True
656         self.flagA = False
657 
658     def run(self):
659         # capture from web cam
660         cascPath = "haarcascade_frontalface_default.xml"
661         faceCascade = cv2.CascadeClassifier(cascPath)
662         cap = cv2.VideoCapture(0)
663 
664         while self._run_flag:
665             # flip orientation of camera 180
666             ret, cv_img = cap.read()
667             flip = cv2.flip(cv_img, -1)
668             gray = cv2.cvtColor(flip, cv2.COLOR_BGR2GRAY)
669             faces = faceCascade.detectMultiScale(
670                 gray,
671                 scaleFactor=1.2,
672                 minNeighbors=4,
673                 minSize=(100, 100),
674                 flags=cv2.CASCADE_SCALE_IMAGE
675             )
676 
677             if ret:
678                 self.change_pixmap_signal.emit(flip)
679 
680                 if type(faces) == numpy.ndarray:
681                     if self.flagA == True:
682                         None
683                     else:
684                         self.flagA = True
685                         try:
686                             self.detectface_signal.emit(flip)
687 
688                         except:
689                             print("Bounding Error")
690                             pass
691 
692                 else:
693                     self.flagA = False
694 
695         cap.release()
696 
697     def stop(self):
698         """Sets run flag to False and waits for thread to finish"""
699         self._run_flag = False
700         self.wait()
```

Lines 649-700: Initialises the *VideoThread* Qthread class. This class is called from lines 473-476.

Lines 650-651: Initialises the two signals *change_pixmap_signal* and *detectface_signal* which will be connected to functions *init_video* and *worker_function* in the *MainWindow* class respectively, when a signal is sent from the *run* function in *VideoThread*.

Lines 653-656: Initialises the *_run_flag* and *flagA* variables after super() initialising the class. This is required as it ensures that the next method in line according to the Method Resolution Order (MRO) is called.

Lines 658-695: Initialises the *run* function which is the main running code for the *VideoThread* class.

Lines 660-662: Initialiases the files and variables required for face detection.

Lines 662: Initialises the mode of capture to the PiCamera.

Lines 664-675: Captures the frames and orientates it to an upright position to be shown on the GUI on *cameraLabel* and *snapLabel* asQ QPixMaps. After which, it detects the frame for any faces, using the "*Haarcascase_frontalface_default.xml"* file downloaded from github.

Lines 677-693: When a face is detected, it will emit and connect the function *init_video* with the given signal *flip.* Inside, it ensures that consequent frames which already has faces detected will not send the *detectface_signal* to the *worker_function* function continuously.

Lines 697-700: Initialises the *stop* function which can be used to stop the *VideoThread* thread from running.

```
703 class ClassThread(QThread):
704     """
705         Class that reads time variable and call "read_class" func-
705 tion every 30min or when min shows "00" or "30"
706     """
707     read_class_signal = pyqtSignal(object)
708 
709     def __init__(self):
710         super().__init__()
711         self._run_flag = True
712 
713     def run(self):
714         while self._run_flag:
715             try:
716                 time_string = str(time_now)
717                 minute_long = time_string.split(":", 1)[1]
718                 minute = minute_long.split(" ")[0]
719                 str_min = str(minute)
720 
721                 try:
722                     if (str_min == "30") or (str_min == "00"):
723                         ts = datetime.datetime.now()
724                         decide_current_time = ts.strftime("%I:" +
724 str_min + " %p")
725 
726                         global newlist
727                         global current_class
728                         global current_teacher
729                         global current_teacher_email
730                         global current_teacher_phone
731                         current_class = str(newlist[decide_cur-
731 rent_time]['Class'])
732                         current_teacher = str(newlist[decide_cur-
732 rent_time]['Teacher'])
733                         current_teacher_email = str(newlist[de-
733 cide_current_time]['Email'])
734                         current_teacher_phone = str(newlist[de-
734 cide_current_time]['Number'])
735 
736                 except:
737                     print("Key Error: Time Key Not Found In
737 List\nKey Given: " + time_now)
738 
739             except:
740                 pass
741 
742     def stop(self):
743         """Sets run flag to False and waits for thread to finish"""
744         self._run_flag = False
745         self.wait()
```

Lines 703-745: Initialises the *ClassThread* Qthread class. This class is called from lines 483-484.

Lines 666-668: Initialises the *_run_flag* variable after super() initialising the class. This is required as it ensures that the next method in line according to the Method Resolution Order (MRO) is called.

Lines 710-711: Initialises the *run* function which is the main running code for the *ClassThread* class.

Lines 716-734: After initial initialisation of *time_now* in line 505, then these lines will be executed. It will obtain the current time and convert it to a string. After which, it obtains the minute of the current time at *str_min*. Then, it checks if the minute is at “30” or “00”. If so, it will update the current class session information for *current_class, current_teacher, current_teacher_email* and *current_teacher_phone*, which is information for the current class, teacher-in-charge of the current class, the teacher’s email, and the teacher’s phone number respectively.

Lines 742-745: Initialises the *stop* function which can be used to stop the *ClassThread* thread from running.

5.4. Temperature Map

main.py

```
33  # For Temperature Map
34  hetaData = []
35  lock = threading.Lock()
36  minHue = 180
37  maxHue = 360
38  ChipType = 'MLX90641'
39  port = 'I2C'
```

Lines 34-40: Initialises the variables to be used in DataReader class later.

```
46  def map_value(value, curMin, curMax, desMin, desMax):
47      curDistance = value - curMax
48      if curDistance == 0:
49          return desMax
50      curRange = curMax - curMin
51      direction = 1 if curDistance > 0 else -1
52      ratio = curRange / curDistance
53      desRange = desMax - desMin
54      value = desMax + (desRange / ratio)
55      return value
56
57  def constrain(value, down, up):
58      value = up if value > up else value
59      value = down if value < down else value
60      return value
61
62  def is_digital(value):
63      try:
64          if value == "nan":
65              return False
66          else:
67              float(value)
68          return True
69      except ValueError:
70          return False
```

Lines 46-70: Defines the functions which are utilised in the temperature map classes.

Lines 46-55: Defines the “mapValue” function. “mapValue” is used to categorise the temperature range retrieved from the MLX90641 Thermal Imaging Sensor. The value returned is used by the painter class to draw out the pixel map or temperature map.

Lines 57-60: Defines the “constrain” function. “constrain” is used to define the limits of the Temperature Map.

Lines 62-70: Defines the “isDigital” function. “isDigital” ensures that during the temperature input reading process, no null or “nan” values are fed to the function, which will produce errors as the system is not able to read “nan” values. It returns digital or Boolean True and False only.

```
 91 class DataReader(QThread): #dont care about this, its just a thread
 91 to read the temperature
 92     drawRequire = pyqtSignal()
 93     I2C = "I2C"
 94     SERIAL = 0
 95     MODE = I2C
 96     pixel_num = 192
 97     def __init__(self,port,ChipType):
 98         #Initialise the backend running processes for mlx90641
 99         super(DataReader, self).__init__()
100         self.frameCount = 0
101         # i2c mode
102         DataReader.pixel_num = 192
103         self.dataHandle = seeed_mlx9064x.grove_mxl90641()
104         self.dataHandle.refresh_rate = seeed_mlx9064x.Re-
104 freshRate.REFRESH_8_HZ
105         self.readData = self.i2c_read
106 
107     def i2c_read(self):
108         #Read from cam and get value
109         hetData = [0]*DataReader.pixel_num
110         self.dataHandle.getFrame(hetData)
111         return hetData
112 
113     def run(self):
114         # throw first frame
115         self.readData()
116 
117         while True:
118             maxHet = 0
119             minHet = 500
120             tempData = []
121             nanCount = 0
122 
123             hetData = self.readData()
124             if  len(hetData) < DataReader.pixel_num :
125                 continue
126 
127             for i in range(0, DataReader.pixel_num):
128                 curCol = i % 32
129                 newValueForNanPoint = 0
130                 curData = None
131 
132                 if i < len(hetData) and is_digital(hetData[i]):
133                     curData = float(format(hetData[i],'.2f'))
134                 else:
135                     interpolationPointCount = 0
136                     sumValue = 0
137                     print("curCol",curCol,"i",i)
138 
139                     abovePointIndex = i-32
```

```
140                     if (abovePointIndex>0):
141                         if hetData[abovePointIndex] is not "nan" :
142                             interpolationPointCount += 1
143                             sumValue += float(hetData[abovePointIn-
143 dex])
144 
145                     belowPointIndex = i+32
146                     if (belowPointIndex<DataReader.pixel_num):
147                         print(" ")
148                         if hetData[belowPointIndex] is not "nan" :
149                             interpolationPointCount += 1
150                             sumValue += float(hetData[belowPointIn-
150 dex])
151 
152                     leftPointIndex = i -1
153                     if (curCol != 31):
154                         if hetData[leftPointIndex]  is not "nan" :
155                             interpolationPointCount += 1
156                             sumValue += float(hetData[leftPointIn-
156 dex])
157 
158                     rightPointIndex = i + 1
159                     if (belowPointIndex<DataReader.pixel_num):
160                         if (curCol != 0):
161                             if hetData[rightPointIndex] is not
161 "nan" :
162                                 interpolationPointCount += 1
163                                 sumValue += float(het-
163 Data[rightPointIndex])
164 
165                     curData =  sumValue /interpolationPointCount
166                     # For debug :
167                     # print(abovePointIndex,belowPointIn-
167 dex,leftPointIndex,rightPointIndex)
168                     # print("newValueForNanPoint",newValueForNan-
168 Point," interpolationPointCount" , interpolation-
168 PointCount ,"sumValue",sumValue)
169                     nanCount +=1
170 
171                 tempData.append(curData)
172                 maxHet = tempData[i] if tempData[i] > maxHet else
172 maxHet
173                 minHet = tempData[i] if tempData[i] < minHet else
173 minHet
174 
175             if maxHet == 0 or minHet == 500:
176                 continue
177             # For debug :
178             # if nanCount > 0 :
179             #     print("____@@@@@@@ nanCount " ,nanCount , "
179 @@@@@@@____")
```

```
181                 lock.acquire()
182             hetaData.append(
183                 {
184                     "frame": tempData,
185                     "maxHet": maxHet,
186                     "minHet": minHet
187                 }
188     )
189     lock.release()
190     self.drawRequire.emit()
191     self.frameCount = self.frameCount + 1
192 self.com.close()
```

Lines 91-192: Defines the “DataReader” class. “DataReader” translates the input with an 8Hz refresh rate from the MLX90641 Thermal Imaging Sensor which will be returned to the “painter” class.

Lines 92-96: Initialises the variables used for the class, which are unchanged throughout the program. It sets the code to communicate with the MLX90641 Thermal Imaging Sensor using I^2C in Serial mode. It also initialises the number of pixels to 192, which is obtained from the 16 X 12 pixels resolution from the datasheet.

Lines 97-105: Initialises the pixel numbers, camera and refresh rate used for this application, using the “seeed_mlx9064x” library.

Lines 107-111: Defines the “i2c_read” function. “i2c_read” reads the hetData, or heterozygote encoding data.

Lines 113-192: Defines the “run” function. “run” reads the hetData and sorts it accordingly before passing it into the “painter” class.

Lines 171: This line appends the data onto a list called “tempData” which will be utilised by the “painter” class.

5.5. SES Email

SESEmail.py

```
1  import os
2  import boto3
3  from threading import Thread
4
5  from botocore.exceptions import ClientError
6  from email.mime.text import MIMEText
7  from email.mime.application import MIMEApplication
8  from email.mime.multipart import MIMEMultipart
9
10 class SESEmail:
11     """
12     Class that calls SES using a dedicated thread
13     """
14     def __init__(self, Date, Name, StudentID, Temp):
15         self.Date = Date
16         self.Name = Name
17         self.StudentID = StudentID
18         self.Temp = Temp
19         self.stopped = False
20
21     def start(self):
22         Thread(target=self.send_email, args=()).start()
23         return self
24
25     def send_email(self):
26         # This address must be verified with Amazon SES.
27         SENDER = "xin.isa.raspberry@gmail.com"
28
29         # Replace recipient@example.com with a "To" address. If
30 your account
31         # is still in the sandbox, this address must be verified.
32         RECIPIENT = "xin.isa.raspberry@gmail.com"
33
34         # The subject line for the email.
35         SUBJECT = "Amazon SES Test (SDK for Python)"
36
37         client = boto3.client('ses')
38
39         msg = MIMEMultipart()
40         msg['Subject'] = SUBJECT
41         msg['From'] = SENDER
42         msg['To'] = RECIPIENT
43
42         part = MIMEText('Hello, You are receiving this email be-
42 cause there is a high temperature recorded.Please kindly ask the
42 student to exit the campus and seek medical attention.\n\n\nDate:'+
42 str(self.Date)+ '\nName:'+ self.Name+' \nStudent ID:'+self.Studen-
42 tID+'\nTemperature:'+ str(self.Temp))
43         msg.attach(part)
```

```
44
45          msg_body = MIMEMultipart('alternative')
46
47          att = MIMEApplication(open('/home/pi/Codes/AWS-Services/Lo-
47  cal File/Image.jpeg','rb').read())
48          att.add_header('Content-Disposition','attachment',filename
48  = os.path.basename('/home/pi/Codes/AWS-Services/Local File/Im-
48  age.jpeg'))
49          msg.attach(msg_body)
50          msg.attach(att)
51
52          try:
53              response = client.send_raw_email(
54                  Source = SENDER,
55                  Destinations = [RECIPIENT],
56                  RawMessage = {
57                      'Data' : msg.as_string(),
58                      },
59                  )
60          except ClientError as e:
61                  print(e.response['Error']['Message'])
```

Lines 1-8: Import required library for operational system, AWS SES and threading.

Lines 10-22: Initialise *SESEmail* class.

Lines 24-33: Initialise variables for sender email, recipient email and the subject of the email.

Line 35-50: Creates a template for email with attachment of the latest image that is taken from the application to the email.

Line 52-61: Sends email to recipient. If unsuccessful, prints error message.

5.6. DynamoDB Database

DynamoAdd.py

```
import boto3
import time
import datetime
from threading import Thread
from decimal import Decimal
from boto3.dynamodb.conditions import Key
import cv2
import os
import io
import sys
import numpy
import boto3

class AddItems:
    """
    Class that calls Dynamo add_items API
    """
    def __init__(self, add_Class, add_StuID, add_Name, add_Temp):
        self.add_Class = add_Class
        self.add_StuID = add_StuID
        self.add_Name = add_Name
        self.add_Temp = add_Temp
        self.stopped = False

    def start(self):
        Thread(target=self.add_items, args=()).start()
        return self

    def add_items(self):
        #Get the present month
        ts = datetime.datetime.now()
        Present_Month = datetime.datetime.now().month
        #Convert month into string
        Month_dict = { 1: "January", 2: "February", 3: "March", 4:
"April", 5: "May", 6: "June", 7: "July", 8: "August", 9: "Septem-
ber", 10: "October", 11:"November", 12:"December"}
        strPresent_Month = Month_dict.get(Present_Month)

        try:
            client = boto3.client('dynamodb')
            dynamodb = boto3.resource('dynamodb')
            #Creating DynamoDB
            table = dynamodb.create_table(
                TableName= strPresent_Month,
                KeySchema=[
                    {
                        'AttributeName': 'Date',
                        'KeyType': 'HASH'
                    },
```

```
                    {
                        'AttributeName': 'Name',
                        'KeyType': 'RANGE'
                    }],
                AttributeDefinitions=[
                {
                        'AttributeName': 'Date',
                        'AttributeType': 'S'
                    },
                    {
                        'AttributeName': 'Name',
                        'AttributeType': 'S'
                    }],
                ProvisionedThroughput={
                    'ReadCapacityUnits': 4,
                    'WriteCapacityUnits': 4
                },
                GlobalSecondaryIndexes=[{
                    "IndexName":"Class_Index",
                    "KeySchema": [
                        {
                        "AttributeName": "Class",
                        "KeyType": "HASH"
                        }
                     ],
                    "Projection": {
                        "ProjectionType": "ALL"
                    },
                    # Global secondary indexes have read and write
capacity separate from the underlying table.
                    "ProvisionedThroughput": {
                        "ReadCapacityUnits": 1,
                        "WriteCapacityUnits": 1
                    }
                    }
                ]
            )

            #Set Created_Table = True when a table is created
            Created_Table = 1

        except:
        #Get existing table
            Created_Table = 0
            table = dynamodb.Table(strPresent_Month)

        # To get the get the 7th month in the past
        if Created_Table is 1:

            Del_Month = Present_Month - 6
            if Del_Month < 1:
                Del_Month = 12 + Del_Month

            #Convert month into string
            strDelMonth = Month_dict.get(Del_Month)
            print(strDelMonth)

```

```
104             #Delete table
105             try:
106                 response = client.delete_table(
107                 TableName= DelMonth)
108 
109             except:
110                 print("No Past Table exists")
111 
112             #Changing the provisioned throughput
113             try:
114                 Past_Month = Present_Month - 1
115                 if Past_Month < 1:
116                     Past_Month = 12 + Past_Month
117 
118                 strPast_Month = Month_dict.get(Past_Month)
119                 response = client.update_table(
120                     TableName = strPast_Month,
121                     ProvisionedThroughput = {
122                         'ReadCapacityUnits': 1,
123                         'WriteCapacityUnits': 1
124                     }
125                 )
126                 print(strPast_Month)
127 
128             except:
129                 print("No previous table")
130 
131         table = dynamodb.Table(strPresent_Month)
132         table.put_item(
133            Item={
134             'Date': ts.strftime("%Y-%m-%d %H-%M"),
135             'Name': self.add_Name,
136             'Class': self.add_Class,
137             'Student ID': self.add_StuID,
138             'Temperature': round(Decimal(self.add_Temp),2)
139             }
140         )
```

Lines 1-11: Imports required libraries for operating system, DynamoDB, time and threading.

Lines 13-25: Initialises the class and calls the functions needed for threading.

Lines 29-35: Obtains the present month and converts it to a string.

Lines 37-87: Try to create a table using the current table month. If the table already exists, it will skip these steps. A secondary index called "Class_Index" will also be created. If table is created, "Created_Table" variable will be set as 1.

Lines 89-112: If a new table is created, it will delete the table that has been stored for 7 months.

Lines 114 - 130: Try to downgrade the capacity units of the previous month's table from 5 to 1. This helps to reduce cost.

Lines 132 - 140: Add items into current DynamoDB table.

Chapter 6. Functions and Services

This introduces and explains the functions and services used throughout the RaspiTAR Application.

6.1. Threading System

The Raspberry Pi does not have enough processing power to process all the codes efficiently in a main thread. As such, the codes are programmed as threads to the main program. A thread is a separate executable flow of code. Using threads not only improves clarity of the code but allows them to run “separately” such that there are two events happening at once in the main thread. This has significantly increased the speed of the whole system.

```
class CSVEmail:

    def __init__(self, Find_Table, Class, Recipient, Teacher):
        self.Find_Table = Find_Table
        self.Class = Class
        self.stopped = False
        self.Recipient = str(Recipient)
        self.Teacher = Teacher

    def start(self):
        Thread(target=self.CSV, args=()).start()
        return self
```

Figure 46: CSVEmail.py Threading code

6.2. Haar Cascade

Haar Cascade is a Haar feature-based cascade classifier. This is a popular method for computer vision object detection proposed in 2001 by Paul Viola and Michael Jones.

In the face detection cascade used, it uses haar features, which are digital features used for object recognition. The features are shown below in *figure 47*. Each haar feature is a single value which is obtained from the subtraction of pixels under the white rectangle form the sum of pixels under the black rectangle. The cascade has already been pre-trained to detect faces in an image.

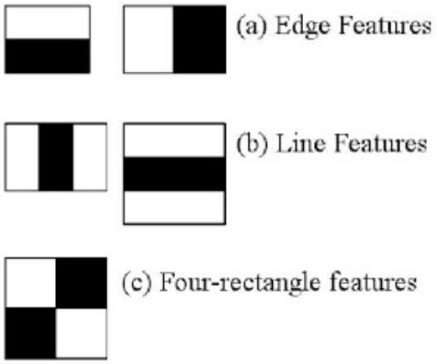


Figure 47: Haar Features

```
def run(self):
    # capture from web cam
    cascPath = "/home/pi/RaspiTAR_App/RaspiTAR/haarcascade_frontalface_default.xml"
    faceCascade = cv2.CascadeClassifier(cascPath)
    cap = cv2.VideoCapture(0)

    while self._run_flag:
        # flip orientation of camera 180
        ret, cv_img = cap.read()
        flip = cv2.flip(cv_img, -1)
        gray = cv2.cvtColor(flip, cv2.COLOR_BGR2GRAY)
        faces = faceCascade.detectMultiScale(
            gray,
            scaleFactor=1.2,
            minNeighbors=4,
            minSize=(100, 100),
            flags=cv2.CASCADE_SCALE_IMAGE
        )
```

Figure 48: main.py Haar Cascade call

To sum it up, Face Detection Haar Cascade model sums up the pixel intensities from regions at a specific location on the detection window then calculates the differences between these sums. This cascade is taken from the OpenCV (Open-Source Computer Vision Library) library using Python, which is a library for computer vision machine learning which has been extensively used by well-known companies such as Google, Intel and Microsoft (OpenCV, n.d.).

6.3. Amazon Web Services (AWS)

Figure 49: Amazon Web Services (AWS) Logo

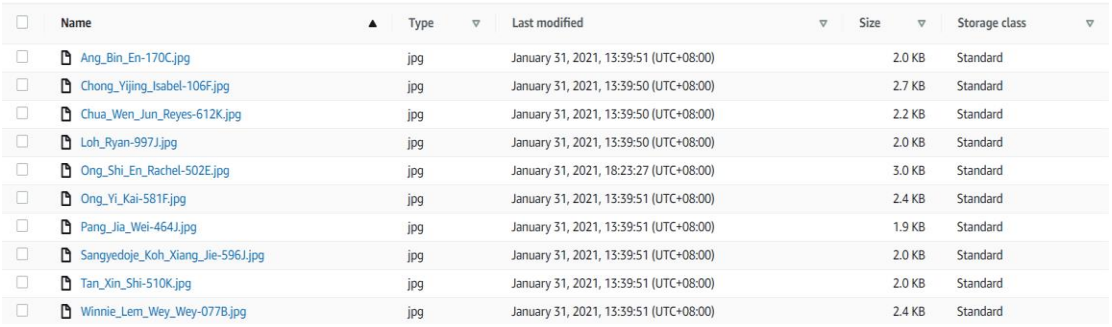
Image obtained from: https://upload.wikimedia.org/wikipedia/commons/thumb/9/93/Amazon_Web_Services_Logo.svg/1200px-Amazon_Web_Services_Logo.svg.png

Amazon Web Services (AWS) is the most all-inclusive and most widely adopted cloud services platform. They present up to 175 fully equipped services from global data centers. AWS has been used by fast-growing and leading enterprises such as Netflix, LinkedIn and Facebook for low-cost agile cloud computing services (Amazon Web Services (AWS), 2020). Five services have been implemented into our project – Amazon S3, Amazon Rekognition, Amazon DynamoDB, Amazon SES, and Amazon SNS.

6.3.1. Amazon S3 – Internet Storage

Amazon S3 stands for Amazon Simple Storage Service. This is an easy-to-use service on AWS which allows users to store and retrieve data easily from the web. S3 is specifically used in conjunction with Amazon Rekognition. Amazon Rekognition will be discussed in *chapter 6.3.2*.

Amazon S3 creates *buckets* in specific regions and contains *objects*. To upload the photos to Amazon S3, a bucket called “bmecenter” was created in a global region. After which, the objects which are the student photos are uploaded into the bucket as shown in *figure 50*.



Name	Type	Last modified	Size	Storage class
Ang_Bin_En-170C.jpg	jpg	January 31, 2021, 13:39:51 (UTC+08:00)	2.0 KB	Standard
Chong_Yijing_Isabel-106F.jpg	jpg	January 31, 2021, 13:39:50 (UTC+08:00)	2.7 KB	Standard
Chua_Wen_Jun_Reyes-612K.jpg	jpg	January 31, 2021, 13:39:50 (UTC+08:00)	2.2 KB	Standard
Loh_Ryan-997J.jpg	jpg	January 31, 2021, 13:39:50 (UTC+08:00)	2.0 KB	Standard
Ong_Shi_En_Rachel-502E.jpg	jpg	January 31, 2021, 18:23:27 (UTC+08:00)	3.0 KB	Standard
Ong_Yi_Kai-581F.jpg	jpg	January 31, 2021, 13:39:51 (UTC+08:00)	2.4 KB	Standard
Pang_Jia_Wei-464J.jpg	jpg	January 31, 2021, 13:39:51 (UTC+08:00)	1.9 KB	Standard
Sangyedoje_Koh_Xiang_Jie-596J.jpg	jpg	January 31, 2021, 13:39:51 (UTC+08:00)	2.0 KB	Standard
Tan_Xin_Shi-510K.jpg	jpg	January 31, 2021, 13:39:51 (UTC+08:00)	2.0 KB	Standard
Winnie_Lem_Wey_Wey-077B.jpg	jpg	January 31, 2021, 13:39:51 (UTC+08:00)	2.4 KB	Standard

Figure 50: Amazon S3 Console View of "bmecenter"

6.3.2. Amazon Rekognition – Face Recognition

Amazon Rekognition is a service that makes adding visual analysis to applications accessible. This is done using deep leaning technology which are highly scalable and requires no prior machine learning expertise. Using Amazon Rekognition, provides highly accurate facial analysis and face search functions which is used to help analyse and compare faces for attendance taking for the project.

Amazon Rekognition stores information about the detected face in server-side containers called collections. After creating a collection called "collectionbmebpd", the images are retrieved from Amazon S3, then analysed, and placed into the collection. To store the analysis information, the IndexFaces operation is used as shown in *figure 51*.

In S3, the images are saved as the student's names, with any spaces in between replaced with underscores "_". This is because the IndexFaces operation does not allow spaces during the operation call. This underscore will be replaced with spaces before storing into the database.

```
def add_faces_to_collection(bucket,photo,collection_id, Name):

    client=boto3.client('rekognition')

    response=client.index_faces(CollectionId=collection_id,
                                Image={'S3Object':{'Bucket':bucket,'Name':photo}},
                                ExternalImageId=Name,
                                MaxFaces=1,
                                QualityFilter="AUTO",
                                DetectionAttributes=['ALL'])
```

Figure 51: IndexFaceIntoCollection.py IndexFace operation call

Figure 52 below shows the information stored in the “collectionbmebpd” collection in Amazon Rekognition. Other than the facial analysis data, there is an “ExternalImageId” which was used for easy comprehension of who the matched face belongs to. This “ExternalImageId” is directly obtained from the file name of the objects stored in Amazon S3 bucket “bmecenter”.

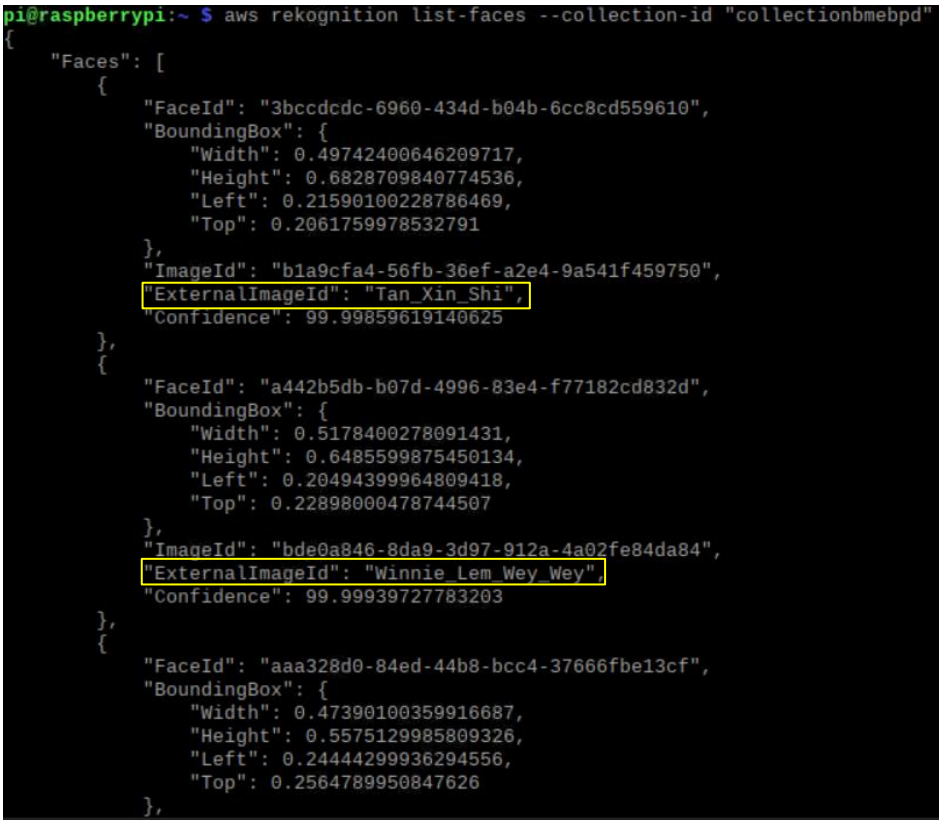


Figure 52: Faces in “collectionbmebpd” collection

To test the functionality of the attendance taking system, various kinds of face images have been tested on the recognition. There are casual photos, ID photos, unfiltered selfies, and faces with masks on. Other than the photos with masks on, the other three types of photos work well after Amazon Rekognition translate the facial bounding features in metadata to be compared to from the live camera. However, it is tested that the selfies or casual photos taken at a frontal angle, where it showcases the student’s casual and day-to-day appearance, provides more accurate facial analysis information. These photos makes the recognition process faster.

Figure 53: Types of Images Uploaded onto “bmecenter” bucket

6.3.3. Amazon DynamoDB – Cloud Database

A cloud database was needed to store the information of each student when they enter the premises and retrieve the data when needed. Information that will be stored will include date, time, student name or student number and temperature records. Having a cloud database allows more data to be stored without needing to manage a local database, which reduces the cost and time needed to manage the data.

DynamoDB is a NoSQL database and is widely used by many companies such as Samsung, Airbnb and Toyota (Amazon Web Services (AWS), 2020). It is also low cost when properly programmed and managed.

A table must be created before inserting the data. There are many ways to create a table: manually from the console, using the AWS command line or using AWS Software Development Kits (SDKs). This project also focuses on being user friendly, which means that the handling of data should be automatic and will not be noticed by the user. Thus, Python SDK was used to interact with DynamoDB and provide more flexibility with the code.

The current program is able to create a new table, access an existing table and delete any tables that have been stored for more than 6 months to be automatically every first day of a new month as well as creating a secondary index called “Class_index”. It is also able to automatically add new data when it is triggered from the Amazon Rekognition service and retrieve the available data from the database. The code will also change the provisioned read and write from 4 to 1 for the previous month so that the full program will be more cost effective. As the secondary index also uses 1 provisioned read and write units each, the total provisioned read and write units used is 5 each.

When creating a new table, the read and write capacity modes must be determined. The two ways available in the AWS service is On-demand or Provisioned Capacity mode. The Provisioned capacity mode is usually the default but can be changed to On-demand mode if needed. Using the Provisioned capacity mode means that there is a pre-allocated read and write units per second. Should the demand for any one of the units be higher, it will be ignored (Amazon Web Services, n.d.). For On-demand mode, the read and write units are decided based on the workload's traffic (Amazon Web Services, n.d.). Thus, it good for tables that have unknown workloads. This was one of the challenging parts of using DynamoDB as the modes can only be switched once every 24 hours (Amazon Web Services, n.d.).

With consideration to the cost, the team has decided to use On-demand capacity mode for the official product. The current code uses Provisioned capacity mode as it is free under the educational account.

6.3.4. Amazon SES (Simple Email Service) – Email Notification

The email notification system will send out an email with information about the student who has high fever. This will allow the school personnel to be notified immediately and take the necessary precautions.

To send an email, Amazon Simple Email Service (SES) was used. This service is relatively cheap and secure. For SES to send an email, the recipient and sender email both have to be verified, which is an added layer of security. *Figure* 53 shows an example of a verified email address on the AWS console.

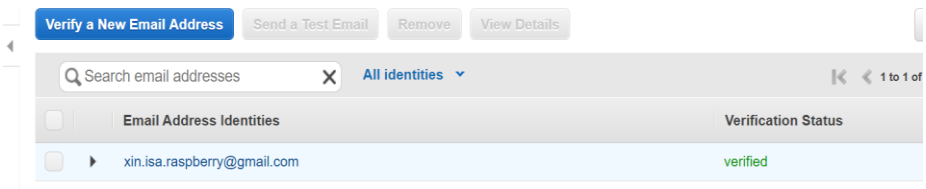


Figure 54: Email Verification

To better allow the staff to identify the student with high temperature, an attachment of the image taken will be included in the email along with the student's name, temperature, time, and date. This email can be sent to the staff or lecturer. This ensures that the student will be immediately escorted out

and seek medical attention. *Figure 54* shows a sample of the email that will be sent to the staff when a high temperature is detected.

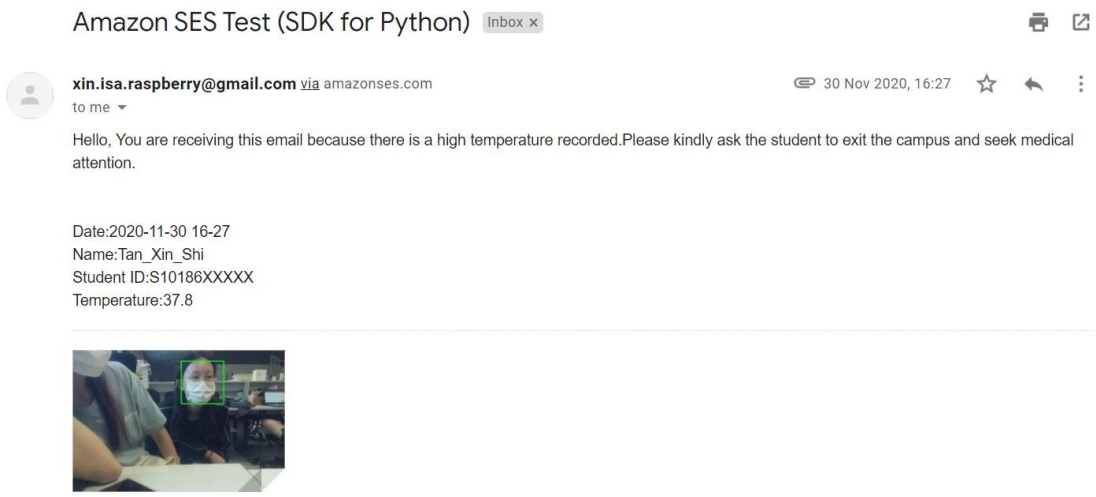


Figure 55: High Temperature Email Notification

To help lecturers with attendance taking, the team has implemented a function to send the list of students that attended class to the lecturer's email. This list will be sent in the form of a Comma-Separated Values (CSV) file. The data would be retrieved from DynamoDB and will only contain the students that entered the class during the set time. The program will first check the google sheet for the class according to the time. When the "Send Attendance List" button is pressed, the program retrieves the records according to the class by triggering the get CSV function. This function will retrieve the records using DynamoDB's secondary index called "Class_Index".

As DynamoDB can only search for records by using a primary key or a primary key and sorting key, to search for other attributes, a secondary index has to be created. The secondary index, "Class_Index" uses the attribute of class to find matches and contains it in a list. After that, it will insert the list into a newly created list and send it to the lecturer using SES service. It has a similar concept to the email notification system. *Figure 55* shows how the email that will be sent to the lecturer will look like and *figure 57* shows a sample of how the CSV file will look like.

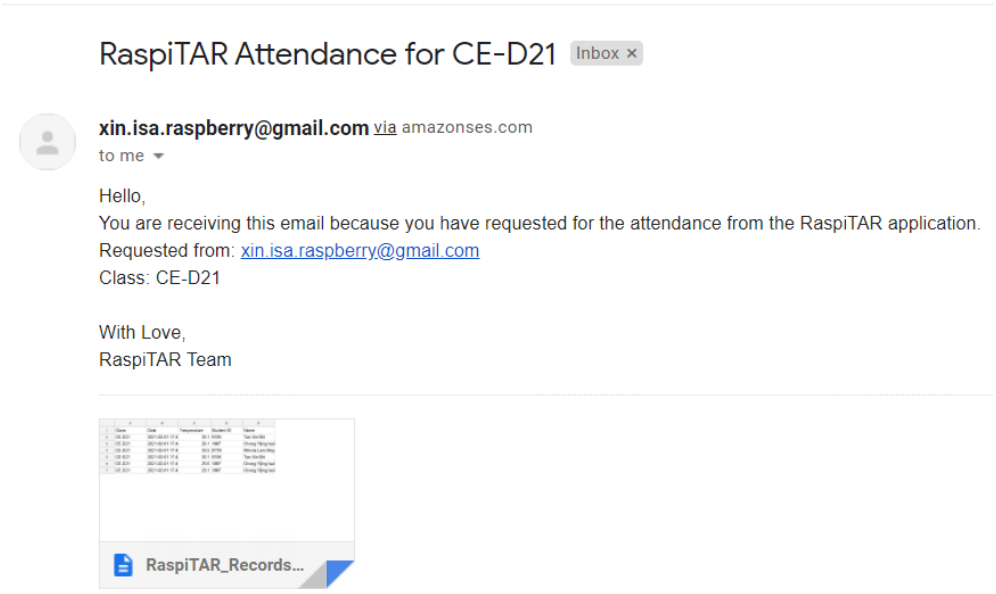


Figure 56: Send Attendance to Email

RaspiTAR_Records01-02-2021_CE-D21.csv

	A	B	C	D	E
1	Class	Date	Temperature	Student ID	Name
2	CE-D21	2021-02-01 17:43	30.1	510K	Tan Xin Shi
3	CE-D21	2021-02-01 17:44	26.1	106F	Chong Yijing Isabel
4	CE-D21	2021-02-01 17:42	30.5	077B	Winnie Lem Wey We
5	CE-D21	2021-02-01 17:42	30.1	510K	Tan Xin Shi
6	CE-D21	2021-02-01 17:41	25.6	106F	Chong Yijing Isabel

Figure 57: Example of Sent Attendance List

6.3.5. Amazon SNS (Simple Notification System)

Another option of notifying the lecturer or staff is via text messages sent to their phone. The lecturer will need to register their phone number into the google sheet. Similar to the SES system, when a high temperature is detected, a text message will be sent to the registered phone numbers. This system can send the same message to all the registered phone numbers or only a specific phone number.

In this way, staff and lecturers can be notified without having the need to open their emails. However, there are limitations to using only this feature as it is costly and cannot send images for the lecturers to easily identify the student

with high temperature. Phone numbers may not need to be verified before using and as such, it may leak out student information if it is implemented into the text message to other parties when a wrong number is registered. *Figure 58* shows a sample of the text message that will be sent to those who have registered their phone number.

Figure 58: Sample of SMS Notification

This feature is available should there be a need to implement it. However, it is not implemented into the current project trial due to its cost and is not included in the subscription costs for AWS in *chapter 7.1*.

6.4. Google Cloud

Google cloud is a public cloud-based machine containing sets of physical assets, such as hard disk drivers and computers, and virtual resources like virtual machines (VMs). These are contained in Google's global data centers, which are each located in a specified region.

Google Cloud services using Google Sheet API has been implemented in the project. The team has chosen to interact with Google Cloud using Google Cloud Platform. For authorization of the account admin, Googe Drive API is used. After which, a Google Cloud Project named "bmebpd" is created. After which, the Google Sheets API and Google Drive API is chosen to be used for the project. Google Cloud Platform easily performs analysis and displays the errors and or successes on the interfaces for ease of error solving.

Google Sheet has been chosen as it can be easily accessible by all teachers who are allowed permissions to either view or edit the sheets as required when there is any change in classes for the day.

Chapter 7. Financial Planning and Costs

7.1. Subscription Costs

Below are the estimated yearly costs to subscribe to the AWS services and Google Cloud Service we have adopted for the completed system.

Service Used	Unit Price	Estimated Quantity	Annual Cost
Amazon Rekognition	For first million photos processed in a month $0.00125 per image	~32,200 units per year	$40.25
Amazon DynamoDB	$1.4231 per million write request units	1,000,000 units	$17.04
	$0.285 per million read request units	1,000,000 units	$3.36
	First 25 GB stored per month is free $0.285 per GB-month thereafter	25GB	Free
Amazon S3	Storage for first 50 TB in a Month $0.025 per GB	~32,200 units x 0.0000238419 GB = 0.76770 GB	~$0.0191925 per year
	Transfers within same region $0.00 per GB	~32,200 units x 261 x 0.0000238419 GB = 200.37209598 GB	Free
Amazon SES	$0.10 per 1,000 emails	< 1,000 emails	$0.10
Google Cloud (Google Drive and Google Sheet APIs)	Free	Free	Free
		Total Costs	~$60.77 per year

Table 3: Subscription Costs

7.2. Overall Finance Documentation

Component	Cost
Raspberry Pi 4 Model B 8GB	$106.90
Raspberry Pi 4B, Red, White Raspberry Pi Case	$6.85
Raspberry Pi micro-HDMI to Standard-Male Cable, 1mtr White	$6.92
MLX90641 Camera Sensor Grove Platform Evaluation	$132.59
LCD I^2C Adapter	$4.00
LCD 16x02 / White on Blue /	$6.00
Total Spent	**$263.26**

Table 4: Non-Service Total Spent

Chapter 8. Conclusion

8.1. Reflections

This is the biggest project that we have taken up into our hands during our time at Ngee Ann Polytechnic. By the end of the module semester, we have managed to finish the RaspiTAR main application using something we have not known about before – a computer called the Raspberry Pi 4! We also went through some product trials in the BME Center, with the cooperation of our friends and module mates.

Throughout the time we spent completing RaspiTAR, we faced a lot of challenges such as sourcing for the best alternatives for our infrared camera and solving errors in our codes. And we could not have done this without mutual support and help from each other. We have learnt how to be resilient and independent individuals by seeking help from internet sources and trying out the different methods to solve the various problem.

Of course, when we reach a dead-end with our personal means of seeking answers, Mr. Soon was glad to extend his helping hand to us and share tutorials and probable solutions we can take on. One such instance is when we were unsure on how to move on when the initial local processing for face recognition was slow and unstable, due to the nature of the processing speed of Raspberry Pi 4. We have thought about purchasing an additional USB processor, but it is very expensive. After consulting Mr. Soon, we found out about Amazon Web Services (AWS) and cloud computing services. We have implemented various services from AWS into RaspiTAR, which solves the initial problem of the code running too slowly for our application.

By the end of the project, we have learnt and successfully implemented the following basis into our project – cloud databases, temperature mapping, using AWS services with python, facial recognition with computer vision and SolidWorks designing.

The project was entirely new to us and we both learnt a lot on new technology and services as well as how to manage our time better to cope with our other assignments. We have gained unforgettable memories working together to solve problems, brainstorm and discuss our project and schoolwork.

8.2. Future Recommendations

This subchapter includes some of the recommendations that can be done to improve the project.

1. Web Application

Firstly, a web application can be built for lecturers to easily view current and past attendances. This application should be able to authenticate the lecturers and view or query student attendance records. In addition, attendance lists can be downloaded when required. To help lecturers in attendance taking, the application should be able to filter the records by comparing the recorded attendances to the class list. Filtering will allow lecturers to easily view absent students and update their daily attendance records with ease. This web application can further expand on the attendance taking feature of this project.

2. Casing Material

Secondly, the material of the Raspberry Pi case should be changed to aluminium to allow better heat dissipation. As the processor for the raspberry pi can get very hot, it is important to ensure that heat is properly dissipated. Otherwise, it may cause the raspberry pi, the thermal camera, or the camera module to spoil easily. Although there is a vent hole implemented for the current 3D printed design, the material of the casing is Polyactic Acid plastic which may not last long when subjected to high heat from the processer. Thus, metals such as aluminium will be better choices as material for the RaspiTAR casing as they are good conductors of heat, which means that they can conduct heat away faster, and are more durable.

Chapter 9. References

Adzik, G. (10 July, 2019). *severless.pub.* Retrieved 13 November, 2020, from S3 or DynamoDB?: https://serverless.pub/s3-or-dynamodb/

Amazon Web Services (AWS). (2020). *Amazon S3 pricing.* Retrieved from AWS: https://aws.amazon.com/s3/pricing/

Amazon Web Services (AWS). (2020). *AWS.* Retrieved from Amazon Simple Storage Service: https://docs.aws.amazon.com/AmazonS3/latest/dev/Welcome.html

Amazon Web Services (AWS). (2020). *AWS.* Retrieved from Amazon Rekognition pricing: https://aws.amazon.com/rekognition/pricing/

Amazon Web Services (AWS). (2020). *AWS.* Retrieved from Amazon SES pricing: https://aws.amazon.com/ses/pricing/

Amazon Web Services (AWS). (2020). *AWS.* Retrieved from Amazon DynamoDB pricing: https://aws.amazon.com/dynamodb/pricing/

Amazon Web Services (AWS). (2020). *AWS.* Retrieved from AWS Lambda Pricing: https://aws.amazon.com/lambda/pricing/

Amazon Web Services. (n.d.). *Read/Write Capacity Mode*. Retrieved 4 February, 2021, from https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.ReadWriteCapacityMode.html#HowItWorks.OnDemand

amazon.sg. (24 January, 2021). *Forehead Thermometer, Digital Infrared Temporal and Ear Thermometer with Fever Alarm and Memory Function for Baby Adults and Kids- Thermometer for Adults.* Retrieved from amazon.sg: https://www.amazon.sg/Forehead-Thermometer-Infrared-Temporal-Function/dp/B089N7F9LX/ref=asc_df_B089N7F9LX/?tag=googleshoppin-22&linkCode=df0&hvadid=408607544148&hvpos=&hvnetw=g&hvrand=15150968296916387674&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=

Amazon.sg. (n.d.). *Forehead Thermometer, Digital Infrared Temporal and Ear Thermometer with Fever Alarm and Memory Function for Baby Adults and Kids- Thermometer for Adults*. Retrieved from Amason.sg: https://www.amazon.sg/Forehead-Thermometer-Infrared-Temporal-Function/dp/B089N7F9LX/ref=asc_df_B089N7F9LX/?tag=googleshoppin-22&linkCode=df0&hvadid=408607544148&hvpos=&hvnetw=g&hvrand=15150968296916387674&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=

Cadimensions. (August, 2019). *How-to-Use-Solidworks-RX-to-Create-a-Problem-Capture.* Retrieved from Cadimensions: https://www.cadimensions.com/wp-

content/uploads/2019/08/How-to-Use-Solidworks-RX-to-Create-a-Problem-Capture.jpg

CAPITOL Technology University. (2020). *What is SOLIDWORKS?* Retrieved from CAPITOL Technology University: https://www.captechu.edu/blog/solidworks-mechatronics-design-and-engineering-program

Drew, H. (12 May, 2020). *Thermal scanners are the latest technology being deployed to detect the coronavirus. But they don't really work.* Retrieved from The Washington Post: https://www.washingtonpost.com/technology/2020/05/11/thermal-scanners-are-latest-technology-being-deployed-detect-coronavirus-they-dont-really-work/

Fluke. (2021). *Fluke Ti480 PRO Infrared Camera*. Retrieved from Fluke: https://www.fluke.com/en-us/product/thermal-cameras/ti480-pro#country-picker-mobile

gov.sg. (6 February, 2020). *What do the different DORSCON levels mean.* Retrieved from gov.sg: https://www.gov.sg/article/what-do-the-different-dorscon-levels-mean

Lazada. (24 January, 2021). *Thermometer For Baby,Ready Stock,High Quality K3 PRO Non-Contact Infrared Temperature Measurement Forehead with Fever Alarm On Sale.* Retrieved from Lazada: https://www.lazada.sg/products/thermometer-for-babyready-stockhigh-quality-k3-pro-non-contact-infrared-temperature-measurement-forehead-with-fever-alarm-on-sale-i1446788944-s6680934311.html?exlaz=d_1:mm_150050845_51350205_2010350205::12:1025267241!5448320

MOH. (2014). *MOH PANDEMIC READINESS AND RESPONSE PLAN FOR INFLUENZA AND OTHER ACUTE RESPIRATORY DISEASES.* Retrieved from Ministry Of Health (MOH): https://www.moh.gov.sg/docs/librariesprovider5/diseases-updates/interim-pandemic-plan-public-ver-_april-2014.pdf

OpenCV. (n.d.). *About.* Retrieved from OpenCV: https://opencv.org/about/

PIMORONI. (n.d.). *Raspberry Pi Camera v2.1 – Standard*. Retrieved from PIMORONI: https://www.google.com/url?sa=i&url=https%3A%2F%2Fshop.pimoroni.com%2Fproducts%2Fraspberry-pi-camera-module-v2-1-with-mount&psig=AOvVaw2_Oz4dIq0_PXqjyNgJ74Ms&ust=1611579633241000&source=images&cd=vfe&ved=0CA0QjhxqFwoTCNDE147QtO4CFQAAAAAdAAAAABAF

Raspberry Pi. (n.d.). *Raspberry Pi 4*. Retrieved from Raspberry Pi: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/

Raspberry Pi. (n.d.). *Raspberry Pi 4 Tech Specs*. Retrieved from Raspberry Pi: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/

SHENZHEN EONE ELECTRONICS CO.,LTD. (n.d.). *1602A-1 LCD Module Specification Ver1.0 .* Retrieved from SHENZHEN EONE ELECTRONICS CO.,LTD: https://www.openhacks.com/uploadsproductos/eone-1602a1.pdf

shopee.sg. (24 January, 2021). *Smartfuture Contactless Temperature Stand for Office.* Retrieved from shopee.sg: https://shopee.sg/Smartfuture-Contactless-Temperature-Stand-for-Office-i.67264398.4528460050?gclid=Cj0KCQiA0rSABhDlARIsAJtjfCcCddSQyPmYFK5z_PYZMWBQ-1M4UBbacWQ0C0w3qyy--XueFL4EX1waAjuPEALw_wcB

The Straits Times. (26 March, 2020). *Coronavirus: Operators scramble to meet new rules.* Retrieved from The Straits Times: https://www.straitstimes.com/singapore/operators-scramble-to-meet-new-rules

TODAY Online. (2020). *DORSCON ALERT LEVELS.* Retrieved from TODAY Online: https://www.todayonline.com/sites/default/files/dorscon_alert_levels_0.jpg

WHO. (15 April, 2020). *Coronavirus disease (COVID-19).* Retrieved from World Health Organization (WHO): https://www.who.int/emergencies/diseases/novel-coronavirus-2019/question-and-answers-hub/q-a-detail/q-a-coronaviruses#:~:text=protect

WHO. (9 July, 2020). *Coronavirus disease (COVID-19): How is it transmitted?* Retrieved from World Health Organisation (WHO): https://www.who.int/emergencies/diseases/novel-coronavirus-2019/question-and-answers-hub/q-a-detail/coronavirus-disease-covid-19-how-is-it-transmitted