COVID-19 Attendance System

B code	Name (first middle last)
B00735132	Jamie William Buick

Background

This attendance system, which acts as a line of defence from COVID-19, can be applied to many situations, mainly where employees or visitors must enter a building or a room for a specific period of time. This system has great significance during this 'new normal' which we are currently living through. COVID-19 easily sweeps through workforces – causing staffing issues for businesses, leading to down-time and lost profits. This device will detect fevers in staff or visitors, rejecting entry to the site if detected, and recording the information to a data base.

Idea in brief

The project is an attendance system with a built-in temperature check – to provide business owners with confidence and protection inside their workplace during this pandemic. The user will bring their personal RFID card near the RFID Chip Reader and if the card is valid the system will prompt the user to provide a temperature sample. The sample is taken using a contactless infrared temperature sensor. The system will decide on whether the user is allowed entry to the site based on their card Unique ID (UID) and their temperature sample. If the temperature sample exceeds the pre-set range the user will be denied entry. The user will receive visual and audio feedback throughout the process. All data will be recorded and sent to the cloud. The data will be accessible by the relevant person/s via Google Sheets. Google sheets will store: a timestamp, the card UID, the temperature sample and the entry decision (GRANTED/DENIED).

Objectives

The primary objective of this project was to further my skills and knowledge of the Arduino microcontroller and peripherals as well as gain knowledge by implementing WiFi functionality via the Arduino IoT platform. I believe that this objective has been obtained as I now have a higher level of understanding of programming principles and the integration of hardware, firmware and cloud systems.

The second objective, set in the proposal, was to be able to implement the ideas and knowledge, gained through the development of the project, to help combat a real-world problem. I also feel that this objective has been met – this system will give companies confidence that infectious persons will not have access to their site, protecting the workforce.

The system stores data automatically, giving the company a cost and time affective solution to managing COVID-19 within their company.

I believe that with further development this product has the potential to create a start-up company. I feel with a uniquely developed database, rather than using Pushingbox.com and Google Sheets, this product has the potential to be a market leader. I also believe that with a different IC module this would be achievable. Particle IoT devices could be used to eliminate the need for the cloud API Pushingbox.com and Google Sheets. I feel that this low-cost, low-power and low-maintenance system would make a significant impact on the market. Companies around the world are desperate for a solution which provides them with the confidence that their premise is protected against COVID-19, I strongly believe that this system, which I have developed will provide this confidence.

Implementation

The COVID-19 Attendance system was initially prototyped using a Breadboard. This allowed for the system (Hardware and Firmware) to be easily built up one component at a time and any changes which were required could be easily made. Once the Breadboard prototype was complete (shown in Figure 5 and 6) I decided that it would be beneficial to the development to complete a PCB design using Altium Designer. I designed the PCB (Figures 7 – 15) and used a company based in China called JLC PCB to manufacture the PCB – they are a low-cost PCB manufacturer for hobbyists and Prototyping. Once the PCBs had been delivered, I began soldering my components and testing the final prototype.

The Adafruit contactless MLX90614 Infrared Temperature Sensor (Figure 27) is powered by 5V and uses I2C serial communication to communicate with the Microcontroller. The MLX90614 uses 0x5A as an I2C address, declared in the library .h file. This sensor uses infrared technology to provide a contactless temperature sensing solution – which is essential for hygiene and detection of COVID-19. The sensor is used to take temperature samples from the user. The sensor is manufactured in a concise package allowing it to be easily implemented into tight enclosures, this makes the sensor very cost affective (around £15). The MLX90614 has a resolution value of 0.02 °C allowing it to give accurate and reliable temperature readings. In the system the MLX90614 is connected to the 5V track, Ground Track, SDA Track and SCL Track – the two data lines have 10K Ohm Pull-up resistors to restore the lines to default state. The Arduino code reads temperature, during a 10 second countdown, by using the

mlx.readObjectTempC() function – this function reads the object temperature in °C. The Arduino code will then decide based on the temperature value regarding entry to the site.

MRFC-522 RFID Chip Reader with Card and Fob (Figure 30) is powered by 3.3V and uses SPI communication. The original plan set out in the 'Week 7 Proposal' stated that this device operate using I2C Protocol. The communication protocol was changed because tracks were required to be cut underneath the MFRC522 for I2C to be activated. The MRFC-522 operates using 13.56MHz, therefore it is very simple to add additional Cards/fobs which operate on the same frequency to the system. In this system the MFRC522 is used for person identification, each card will contain a UID, if the system does not recognise this UID the user will be denied access and the program will return to the main screen. On the other hand, if the card is recognised the system will request a temperature sample from the user and the process will continue.

To display messages and instructions to the user a 128x64 OLED Display Module (Figure 26) is used. This Module operates with 5V power and using I2C Protocol. The display uses 0x3C as an I2C address. This module is very inexpensive and there are many models available. The small package and IC2 communication allows the device to be easily mounted into a small enclosure. The module operates using the Adafruit_SSD1306 Arduino library – making configuration simple.

RGB LED (Figure 29) is used to provide visual feedback to the user - in addition to the OLED Display. The RGB LED is connected to 3 GPIO Pins, through 3x 220 Ohm resistors, on the Arduino and is controlled using the analogWrite() function. The analogWrite() function is used so that PWM signals can be applied to each pin – different duty cycle results in different colours. On start up the LED cycles through several colours – this is visual confirmation, alongside the splash screen, that the device is initializing. The red LED will display if an unrecognised card is scanned, during the temperature sample and if access is denied. The green LED will display if a recognised card is scanned and if access is granted.

An Active Buzzer (Figure 28) is also used as an audible feedback for the user. The buzzer is connected to a GPIO with a 220 Ohm Resistor. This allows the buzzer to be controlled using digitalWrite(). The buzzer will sound when access is denied due to the temperature value breaching the pre-set limits. The buzzer will also sound when access is granted. In future this component should be connected to a relay as the output is very faint due to the power consumption of the RGB LED which is activated simultaneously.

The MKR1000 which can utilize the WiFi101 library for cloud connectivity allowed for all data collected during the systems operation to be transmitted to the cloud into a simple to access Google Sheets File. This process is completed at the end of the user interaction as to not slow the system down. The device must have a WiFi connection, this can be obtained by entering WiFi credentials into the 'Arduino-secrets.h' file. In future if a start-up company were to be created, it would be beneficial to develop a dedicated database for all this information. Once the user interaction phase is complete and data has been collected the Arduino MKR1000 sends the data via a HTTP link, to the third-party cloud API called Pushingbox.com, the web link is in the following form:

http://api.pushingbox.com/pushingbox?devid=v442D0A72C12BCE7&temperature=$temperature$&CardUID=$CardUID$&Access=$Access$

Once PushingBox has received this information it translates it into Google compliant HTTPS encrypted data which can then be interpreted by our Google Sheets Script (Figure 32). In the future, an encryption algorithm could be developed that will eliminate the need for PushingBox. The device ID located in the HTTP link indicates which scenario we want to use. Figure 1 shows the scenario which is created on Pushingbox.com.

Figure 1 PushingBox Scenario

Any data collected is logged in Google Sheets in a simple to manage format, as discussed above, this can be seen in Figure 2.

97	24/11/2020	22.17	998DADB2	GRANTED
98	24/11/2020	36.93	998DADB2	DENIED
99	24/11/2020	22.37	998DADB2	GRANTED
100	24/11/2020	32.97	998DADB2	DENIED

Figure 2 Google Sheets Example

Once the prototype was fully tested, an enclosure was designed using OnShape (Free for student's CAD software). Once this was designed, it was printed using a Prusa i3 3D printer. The PCB and Electronics was then built into the enclosure – shown in Figures 16 to 25.

In Figure 4, you can see the flow chart which shows the system process during code execution.

Block diagram of the Final System

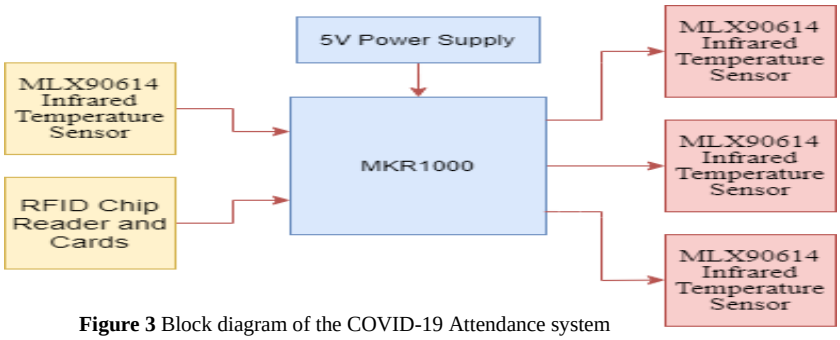

Figure 3 Block diagram of the COVID-19 Attendance system

Inputs

1. Adafruit contactless MLX90614 5V Infrared Temperature Sensor – Connected to 5V power and uses I2C protocol.
2. RFID Chip Reader 3.3V – Connected to 3.3V power and uses SPI Protocol.

Outputs

1. 128x64 OLED Display Module - Connected to 5V power and uses I2C Protocol.
2. RGB LED – Connected to 3 GPIO pins and Ground.
3. Active Buzzer - Connected via a GPIO pin and Ground.
4. The data that is recorded and sent to the cloud includes: A time stamp, the user's UID of their card, the temperature sample and the entry decision (granted or denied).

Flow diagram of the Final System

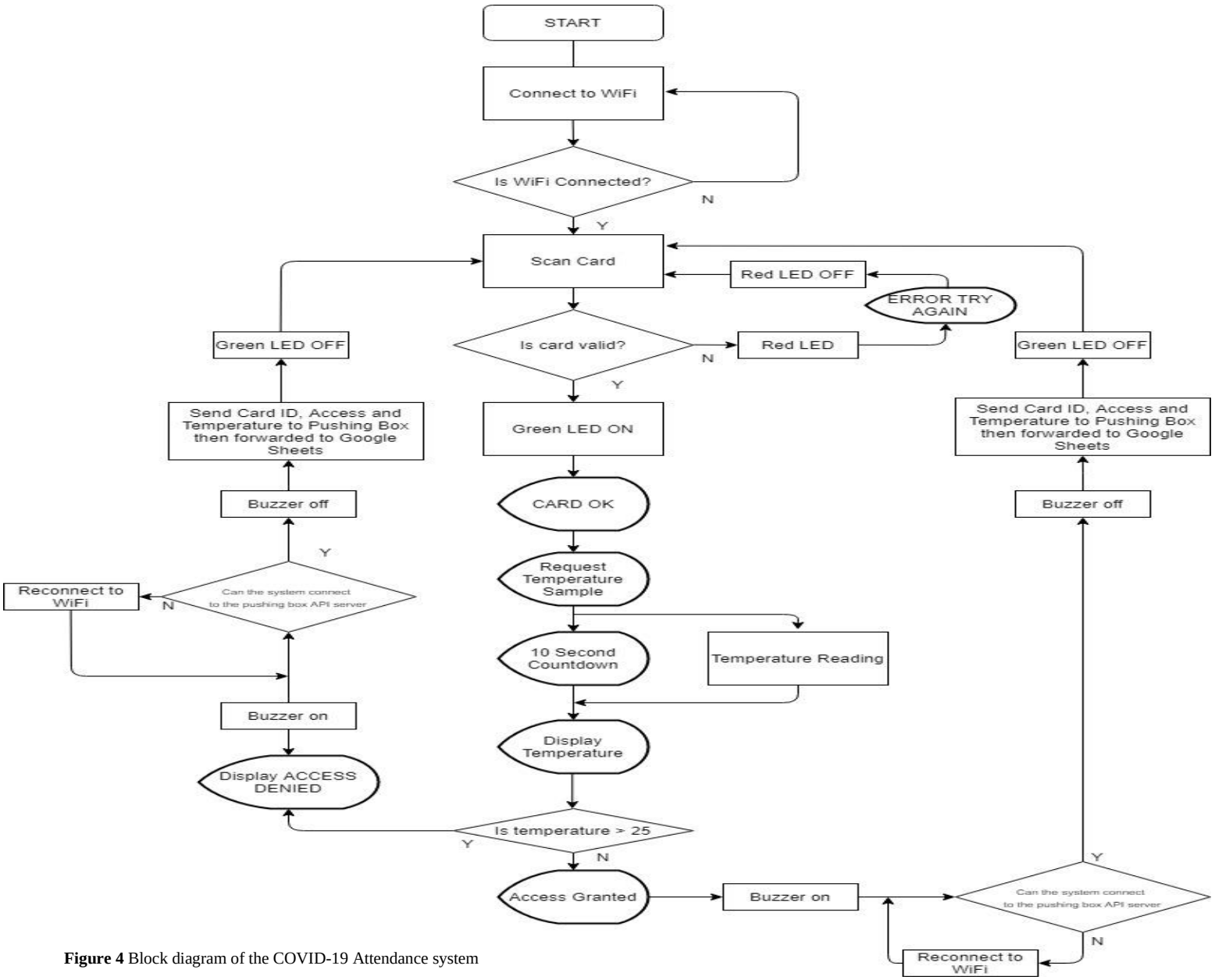

Figure 4 Block diagram of the COVID-19 Attendance system

Appendix A: Circuits and Prototype

COVID-19 Attendance System (Breadboard) - Fritzing Schematic

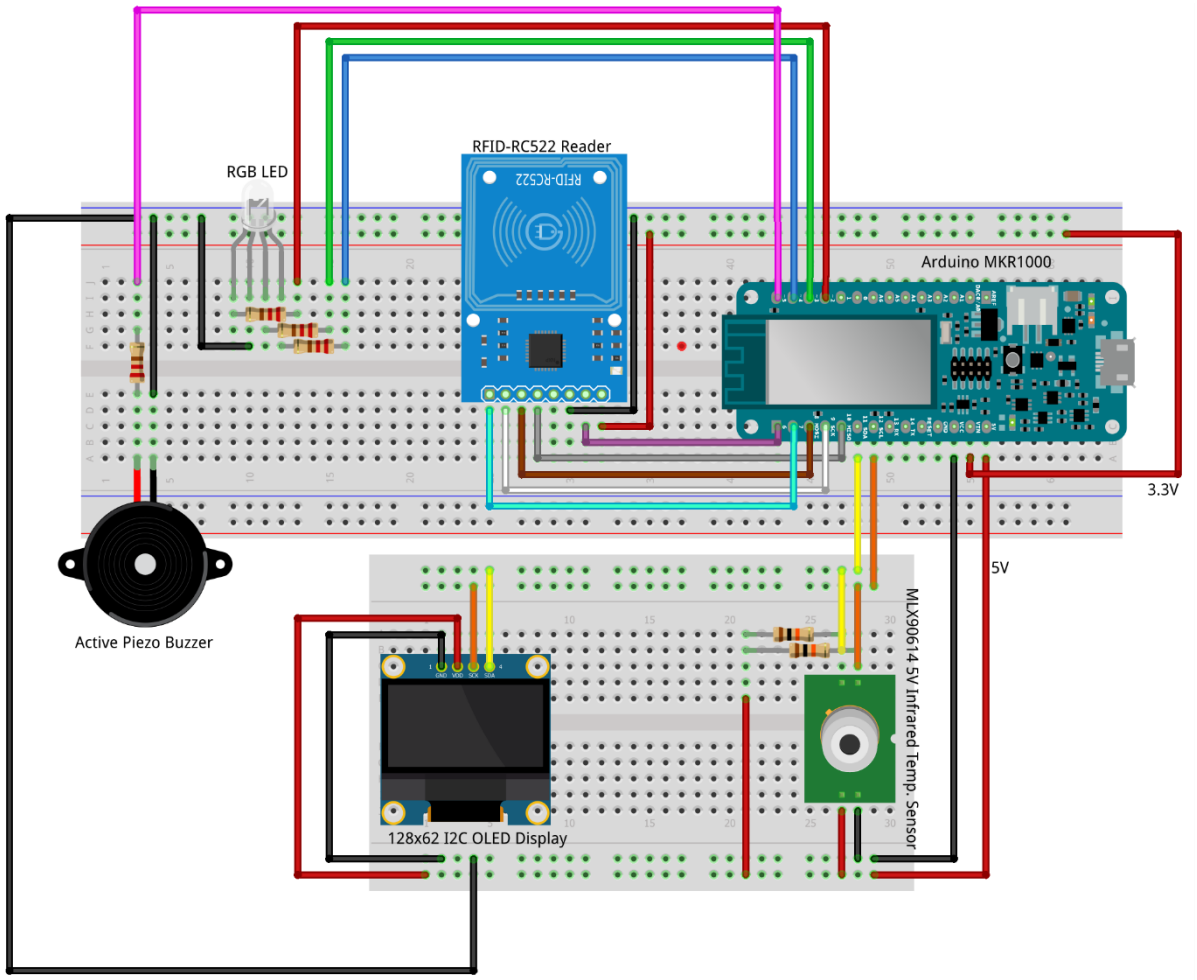

Figure 5 – COVID-19 Attendance System Fritzing Schematic

COVID-19 Attendance System (Breadboard) – Images

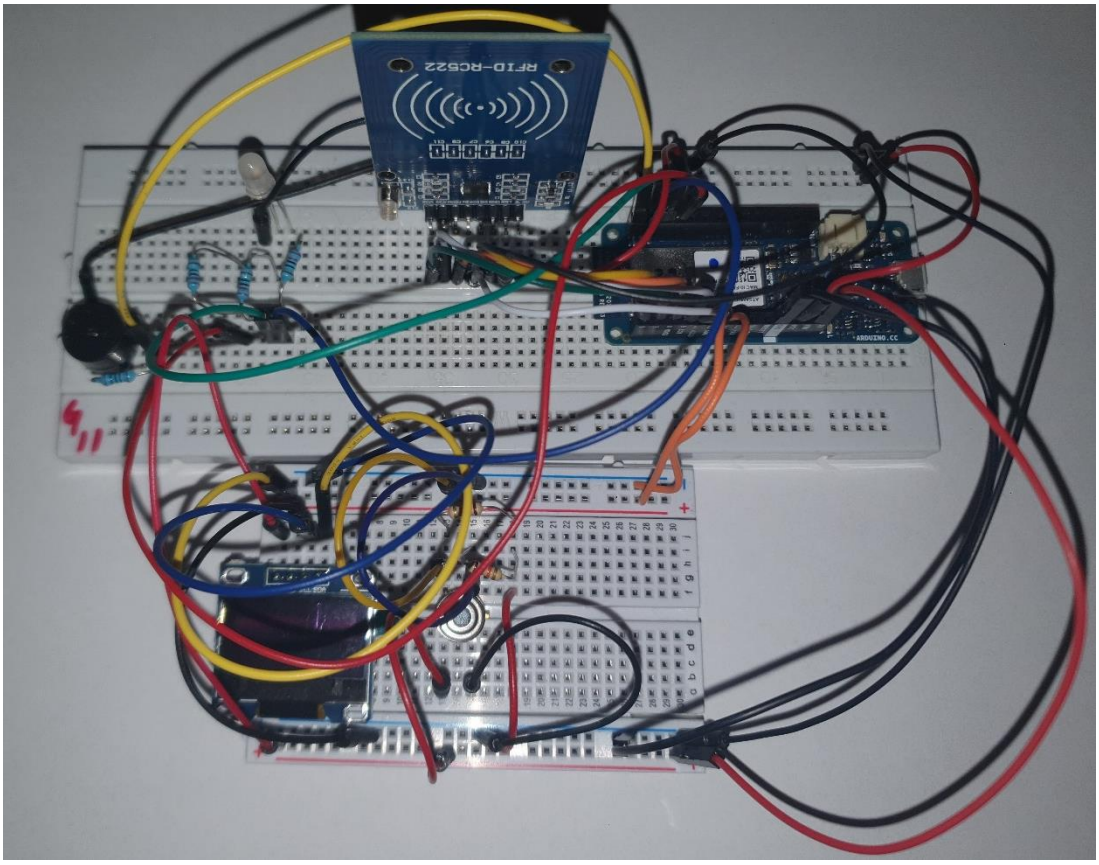

Figure 6 – COVID-19 Attendance System Breadboard construction

COVID-19 Attendance System (PCB) – Schematic and PCB

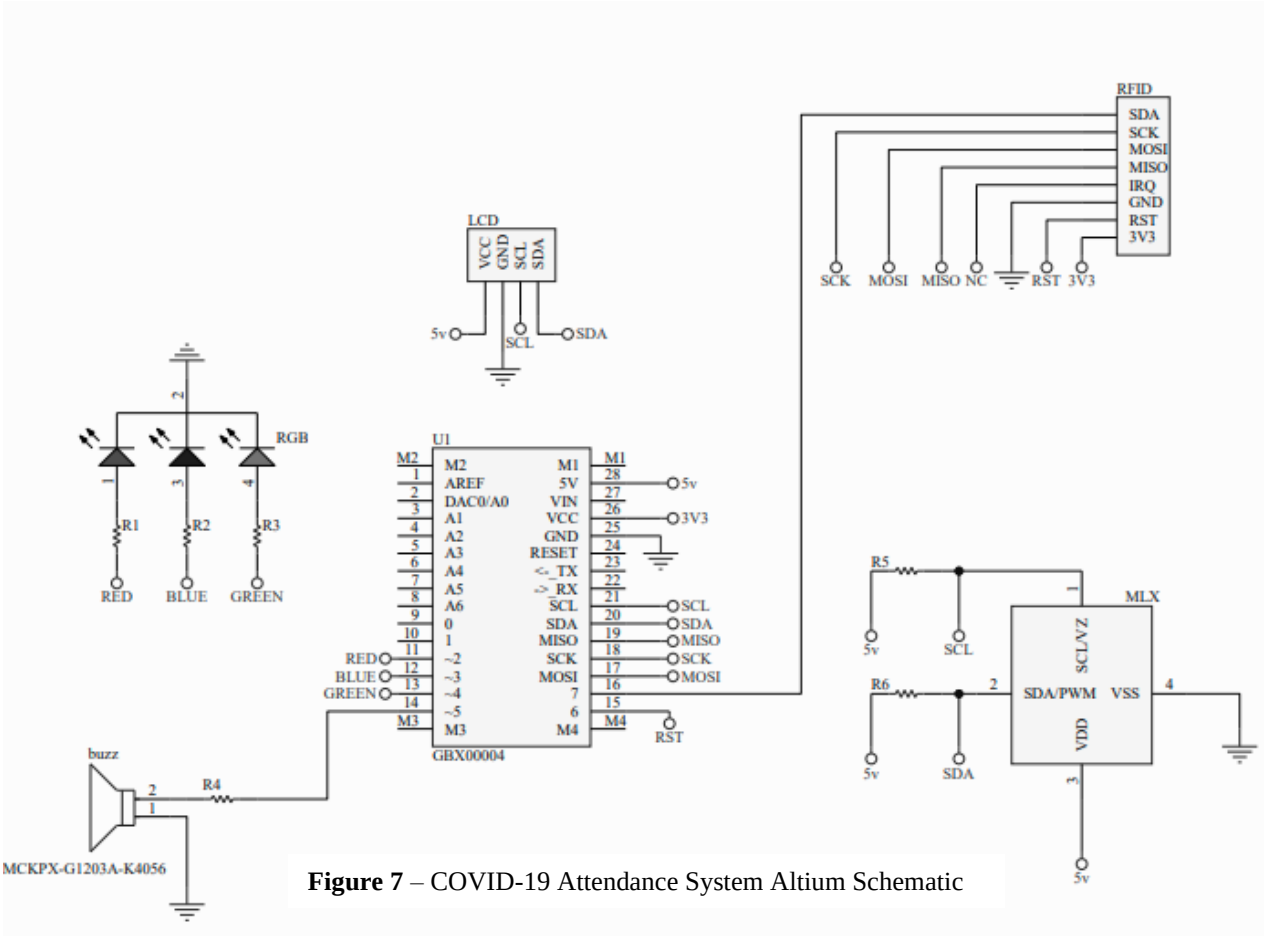

Figure 7 – COVID-19 Attendance System Altium Schematic

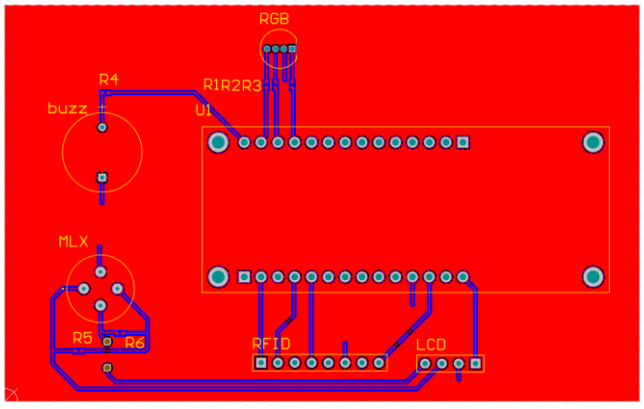

Figure 8 – Altium PCB Front Top Layer

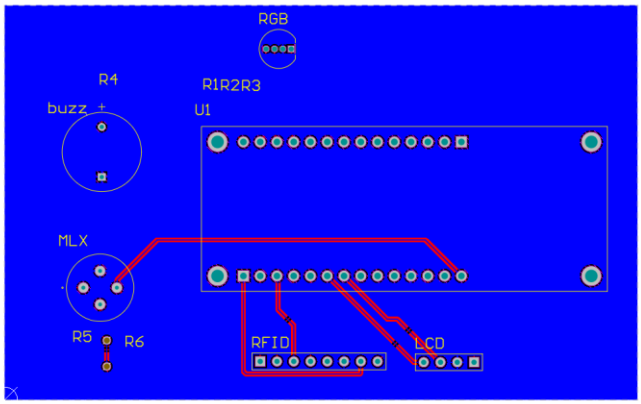

Figure 9 – Altium PCB Front Bottom Layer

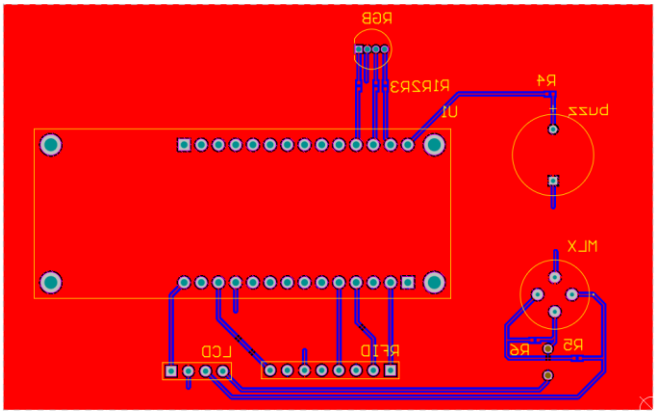

Figure 10 – Altium PCB Front Top Layer

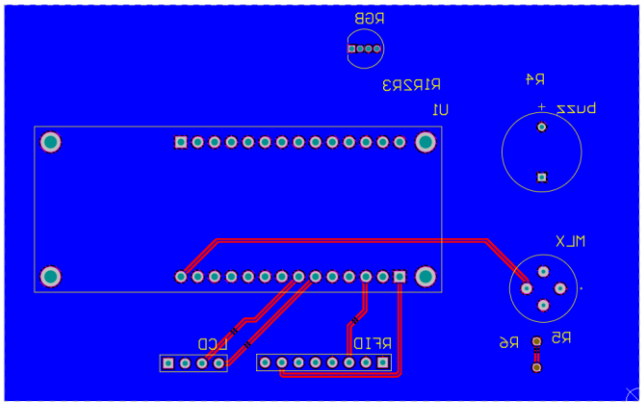

Figure 11 – Altium PCB Front Top Layer

COVID-19 Attendance System (PCB) - Images

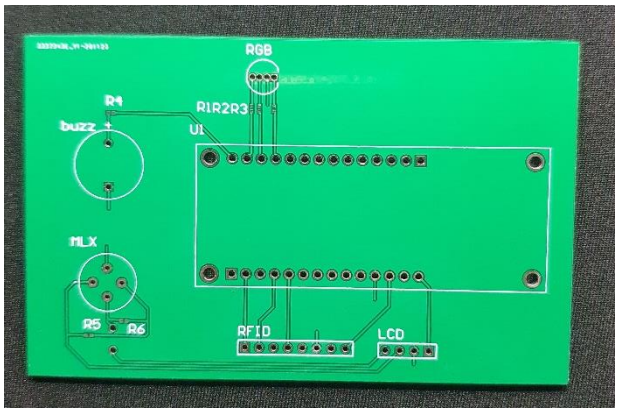

Figure 12 –PCB Front

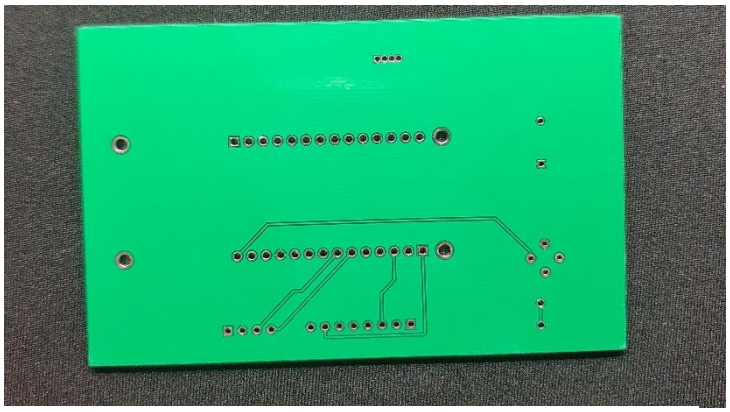

Figure 13 – PCB Back

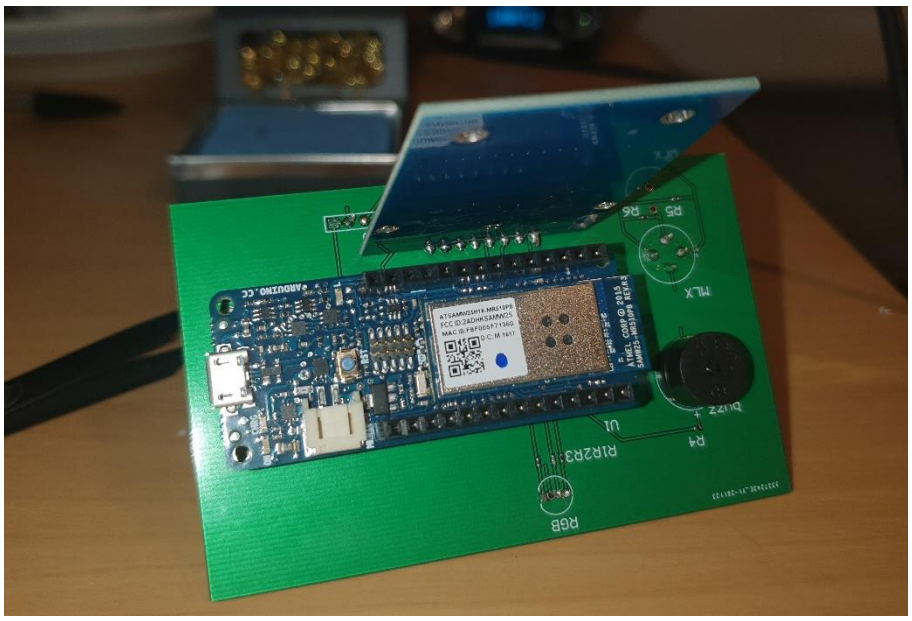

Figure 14 –PCB Front Soldered

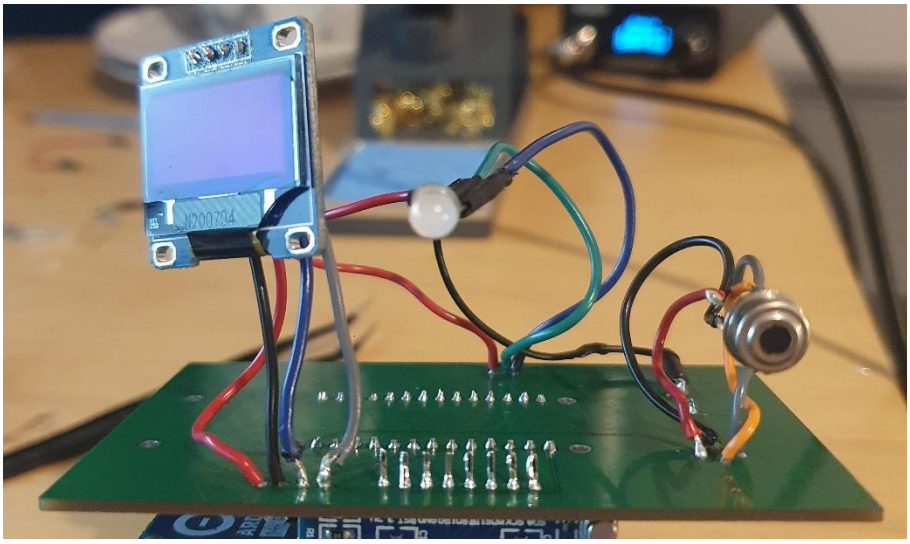

Figure 15 – PCB Back Soldered

Prototype CAD Images

Figure 16 – CAD Front View

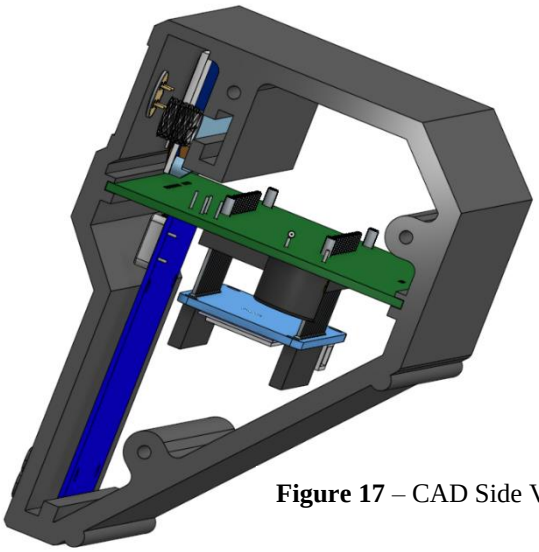

Figure 17 – CAD Side View 1

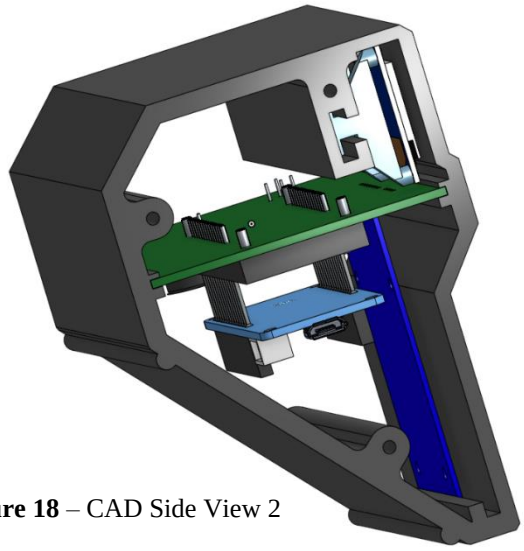

Figure 18 – CAD Side View 2

Figure 19 – USB Slot

Prototype Construction

Figure 20 – 3D Printed Enclosure Components

Figure 21 – 3D Printed Enclosure Front Face

Figure 22 – 3D Printed Side Sections

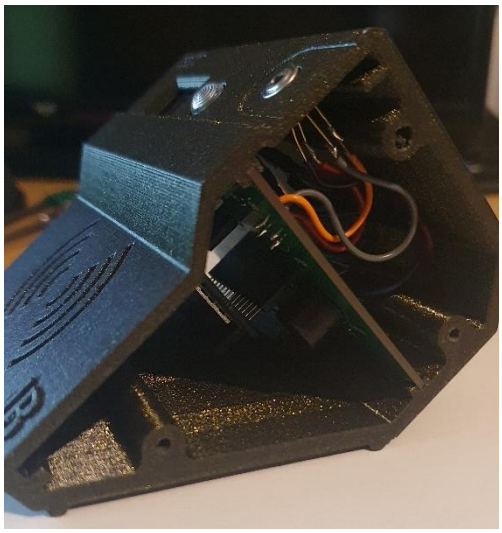

Figure 23 – PCB inside the enclosure

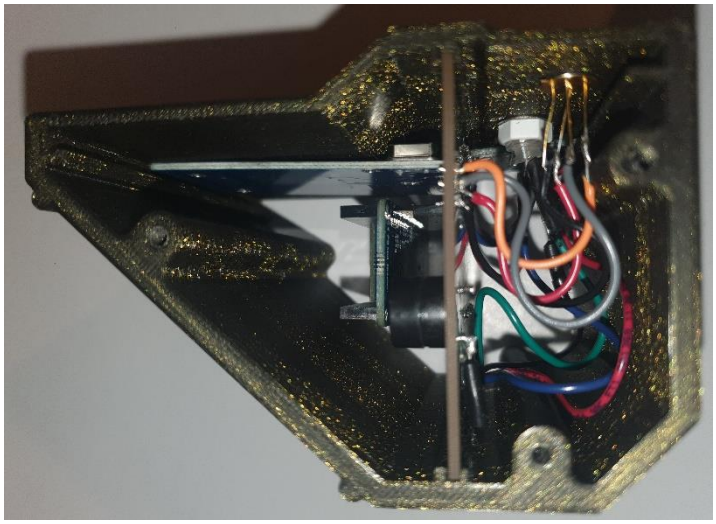

Figure 24 – PCB inside the enclosure

Figure 25 – Final Prototype

Components List

Figure 26 – I2C 128x64 OLED Display

Figure 27 – Adafruit MLX90614 5V Contactless Infrared Temperature Sensor

Figure 28 – Piezo Active Buzzer

Figure 29 – Common Cathode RGB LED

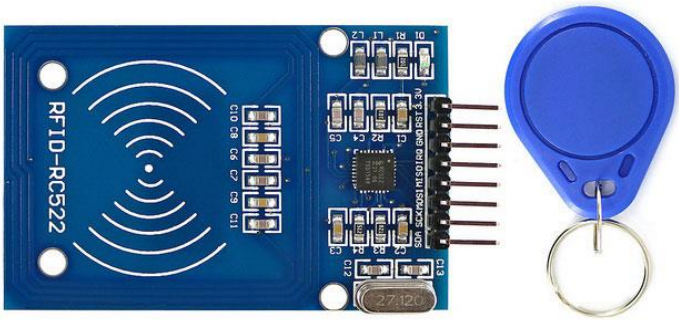

Figure 30 – RFID Scanner, Card and Fob

Appendix B: Code

Arduino Code (Figure 31)

```
/*********************************************************************************
**********************************************************************
EEE527 Project

COVID-19 Attendance System

Jamie Buick - B00735132

Code complete: 22/11/2020

*********************************************************************************
**********************************************************************/


#include<SPI.h>  // Library for communication between SPI
Devices.
#include<Wire.h>  // Library for communication between I2C Devices.
#include<WiFi101.h>  // Library to access the MKR1000 WiFi capabilites.
#include <MFRC522.h>  // Library for the RFID Card Reader.
#include <Adafruit_MLX90614.h>  // Library for the Adafruit Infrared Sensor.
#include <Adafruit_GFX.h>  // Library to provide a common set of graphics
primitives for the OLED Display.
#include <Adafruit_SSD1306.h>  // Library for the Adafruit OLED Display.
#include "arduino_secrets.h"  // Library used to store the WiFi credentials for
the chosen WiFi Network.

#define SCREEN_WIDTH 128  // Defining the OLED Screen Width - it is 128 pixels
wide.
#define SCREEN_HEIGHT 64  // Defining the OLED Screen Height - it is 64 pixels
high.

#define RST_PIN 6  // Setting the reset pin for the MFRC522.
#define SS_PIN 7  // Setting the SDA pin for the MFRC522.
MFRC522 mfrc522(SS_PIN,RST_PIN);  // Creating the MFRC522 instance.


Adafruit_MLX90614 mlx = Adafruit_MLX90614();  // Initializing the Infrared
temperature sensor.


Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);  //
Initializing the OLED Screen.

// Pushingbox API server
const String devid = "v442D0A72C12BCE7";  // Setting the device ID given on
pushingbox.com when setting the system up.
char WEBSITE[] = "api.pushingbox.com";  // Pushing box website.


/*
 * WiFi Credentials can be set in the Arduino_secrets.h
 * file.
 */

char ssid[] = SECRET_SSID;  // WiFi SSID
char pass[] = SECRET_PASS;  // WiFi Password

int status = WL_IDLE_STATUS;  // the WiFi radio's status.


float temperature;  // The temperature variable is set to a float
variable.
```

```
String CardUID = "99 8D AD B2";  // CardUID is set to a string, more cards can be
added to this to increase the data base.
String Access;  // Access is set to a string.


const int ledRed = 2;  // The Red pin of the RGB set to pin 2 on the
MKR1000.
const int ledGreen = 3;  // The Green pin of the RGB set to pin 3 on the
MKR1000.
const int ledBlue = 4;  // The Blue pin of the RGB set to pin 4 on the
MKR1000.
const int buzz = 5;  // The Buzzer has been set to pin 5 on the MKR1000.


unsigned long time_now = 0;  // time_now is used in a millis delay below



void setup() {


  SPI.begin();  // Initiate  SPI bus.
  Wire.begin();  // Starts I2C
Communication.
  Serial.begin(9600);  // Initialize serial and wait for port to open.
  while (!Serial)
  {
  ;
  }

  mfrc522.PCD_Init();  // Initiate MFRC522


  mlx.begin();  // Initiate MLX90614


  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {  // Initiate the SSD1306 OLED
Display.
    Serial.println(F("SSD1306 allocation failed"));  // if there is no response
print message in serial monitor.
    for(;;);
    }



  display.clearDisplay();  // Clear the OLED Display.

  /*
   * Start Up Screen, Delayed with millis so that it remains
   * on screen whilst MRK1000 Connects to WiFi
   */

  JBIndustries();  // Splash Screen function.
  rgbStart();  // RGB function, runs through the colours.
  if(millis() >= time_now + 15000){  // Millis functins for a 15 second delay.
    time_now += 15000;
    }



  //  The following blocks of code are used to initialize the WiFi Network.

  // check for the presence of the shield:
  if (WiFi.status() == WL_NO_SHIELD) {  // If there is no WiFi shield present.
    Serial.println("WiFi shield not present");
    while (true);  // don't continue if there is no shield present.
  }
```

```
  // attempt to connect to WiFi network:
  while ( status != WL_CONNECTED) {  // While WiFi isn't connected.
    Serial.print("Attempting to connect to WPA SSID: ");
    Serial.println(ssid);  // Prints the SSID which the MKR1000 is attempting
connection to.
    status = WiFi.begin(ssid, pass);  // Connect to WPA/WPA2 network.
    delay(10000);  // wait 10 seconds for connection.
  }

  // you're connected now, so print out the data:
  Serial.println("You're connected to the network");  // 'Connected' message
printed to serial monitor.
  printWifiStatus();  // Start the printWiFi status function.


  // Setting up output pins for the components
  pinMode(ledRed,OUTPUT);
  pinMode(ledGreen,OUTPUT);
  pinMode(ledGreen,OUTPUT);
  pinMode(buzz,OUTPUT);


  RFID();  // Calling the RFID(); function.
}



void RFID(){
  // This block of code displays 'Please scan card!' message on the OLED.
  display.clearDisplay();  // Clear the previous from the OLED.
  display.setTextSize(2);  // Set size to 2 pixels.
  display.setTextColor(WHITE);  // Set text colour to white.
  display.setCursor(0,0);  // Set cursor to begin at 0,0.
  display.print("Please");  // Print 'Please' to OLED
  display.setCursor(0,24);  // Set cursor to begin at 0,24.
  display.print("Scan");  // Print 'Scan' to OLED
  display.setCursor(0,50);  // Set cursor to begin at 0,50.
  display.print("Card!");  // Print 'Card' to OLED.
  display.display();  // Display contents on OLED.
  display.clearDisplay();  // Clear OLED of previous.

  // The 2 following if statements deny any card which isn't correct.
  if ( ! mfrc522.PICC_IsNewCardPresent())
  {
    return;
  }
  // Select one of the cards
  if ( ! mfrc522.PICC_ReadCardSerial())
  {
    return;
  }
  //Show UID on serial monitor
  String Tag= "";
  for (byte i = 0; i < mfrc522.uid.size; i++)
  {
    Tag.concat(String(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " "));
    Tag.concat(String(mfrc522.uid.uidByte[i], HEX));
  }

  Tag.toUpperCase();  // Converts the string up uppercase letters.

  CardUID = "99 8D AD B2";

  // Comparing the scanned card to the cards which are allowed access.
  if (Tag.substring(1) == CardUID)  // if the card is accepted:
  {
    Serial.println("Card Accepted");  // Print 'Card Accepted' in serial
monitor.
```

```
    Serial.println(CardUID);  // Print the scanned card ID in serial
monitor.
    setColour(0, 0, 255);  // Turn LED to Green to show where the Temperature
sensor is located.
    display.clearDisplay();  // Clear the previous from the OLED.
    display.setTextSize(2);  // Set size to 2 pixels.
    display.setTextColor(WHITE);  // Set text colour to white.
    display.setCursor(20,0);  // Set cursor to begin at 20,0.
    display.print("Card OK");  // Display 'Card OK' on OLED for user to see.
    display.drawLine(0,25,128,25, WHITE);  // Draw a line - to divide the OLED
Screen into two sections.
    display.setTextSize(1);  // Set size to 1 pixel.
    display.setCursor(0,50);  // Set cursor to begin at 0,50.
    display.print(CardUID);  // Display the card ID on OLED for user to see.
    display.display();  // Display contents on OLED.
    delay(5000);  // 5 Second delay
    setColour(0, 0, 0);  // Turning RGB off.

    Serial.println("Sampling Temperature");  // Print 'Sampling Temperature' in
serial monitor
    RequestTemperatureLCD();  // Calling the RequestTemperatureLCD(); function to
now run.
    temperatureSample();  // Then calling the temperatureSample(); function to
now run.
    }

  else if (Tag.substring(1) != CardUID){  // If the card is declined:
    setColour(255, 0, 0);  // Set RGB to Red.
    Serial.println("Card Declined");  // Print the 'Card Declined' in serial
monitor.
    Serial.println(CardUID);  // Print the scanned card ID in serial monitor.
    display.clearDisplay();  // Clear the previous from the OLED.
    display.setTextSize(2);  // Set text size to 2.
    display.setTextColor(WHITE);  // Set text colour to white.
    display.setCursor(30,10);  // Set cursor to 30,10.
    display.print("ERROR");  // Print the 'Error' on OLED
    display.setCursor(10,30);  // Set cursor to 10,30.
    display.print("TRY AGAIN");  // Print the 'TRY AGAIN' on OLED.
    display.display();  // Display contents on OLED.
    delay(5000);  // Delay of 5 seconds.
    setColour(0, 0, 0);  // Set RGB LED to off.
    RFID();  // Return to the RFID(); function.
    }
    else
    {
      RFID();  // Return to the RFID(); function.
    }
}




void loop() {
  RFID();  // void loop() only contains one function
           // which is required to be constantly looped - the device
  }        // is always looking for a card to be scanned so that the other
process can follow.
```

```
/*********************************************************************************
**************************************************************************
This following section contains all of the functions created for this code. They
are as follows:

1. JBIndustries();
2. RequestTemperatureLCD();
4. temperatureSample();
5. temperatureReading();
6. setColour();
7. rgbStart()
8. temperatureTransmission();
9. printWifiStatus();


*********************************************************************************
**************************************************************************/

/*
 * This is the splash screen used when the device is power cycled, this screen is
used
 * when the device is connecting to the internet and initializing the components.
 */

void JBIndustries(void){
  display.clearDisplay();  // Clear the previous from the OLED.
  display.setTextSize(2);  // Set size to 2 pixels.
  display.setTextColor(WHITE);  // Set text colour to white.
  display.setCursor(50,10);  // Set cursor to 50,10.
  display.print("JB");  // Display 'JB' on the OLED.
  display.setCursor(0,30);  // Set cursor to 0,30.
  display.print("Industries");  // Display 'Industries' on the OLED.
  display.drawLine(0,45,128,45, WHITE);  // Draw a line to split the OLED into 2
sections.
  display.setCursor(0,55);  // Set cursor to 0,55.
  display.setTextSize(1);  // Set text size to 1 pixel.
  display.print("EEE527      B00735132");  // Display 'EEE527 B00735132' on the
OLED
  display.display();  // Display contents on OLED.
}

/*
 * This function simply displays a message asking the user to provide a
temperature
 * sample.
 */

void RequestTemperatureLCD(void){
  display.clearDisplay();  // Clear the previous from the OLED.
  display.setCursor(0,0);  // Set cursor to 0,0.
  display.setTextSize(1);  // Set text size to 1 pixel.
  display.print("Please provide a     ");  // Display 'Please provide a' on OLED.
  display.println("temperature sample");  // Display 'temperature sample' on
OLED.
  display.display();  // Display contents on OLED.
  delay(3000);  // 3 Second delay.
  display.clearDisplay();  // Clear the previous from the OLED.
  }




/*
 * This function is used to take the temperature sample from the user,
 * there is a 10 second countdown (which is displayed on the LCD)
 * and during this 10 samples are taken. The final temperature sample is used for
 * the decision stage as this allows the temperature sensor to stabilize.
 */
```

```
void temperatureSample(){
  setColour(255, 0, 0);  // Set RGB to Red.
  for(int t=10; t>=0; t--){  // For loop which is used as a simple 10 second
countdown.
    Serial.print(temperature);  // Print a temperature to serial monitor every
second.
    temperature = (mlx.readObjectTempC());  // Setting temperature to the object
temperature from the MLX library.
    display.clearDisplay();  // Clear the previous from the OLED.
    display.setTextSize(2);  // Set text size to 2 pixels.
    display.setTextColor(WHITE);  // Set text colour to white.
    display.setCursor(10,0);  // Set cursor to 10,0.
    display.print("READING..");  // Display 'Reading' on the OLED.
    display.drawLine(0,20,128,20, WHITE);  // Draw a line to split the OLED into
2 sections.
    Serial.println(t);  // Print the countdown timer in serial monitor.
    display.setTextSize(4);  // Set text size to 4 pixels.
    display.setCursor(50,30);  // Set cursor to 50,30.
    display.setTextColor(WHITE);  // Set text colour to white.
    display.print(t);  // Display the countdown timer on the OLED.
    delay(1000);  // Delay for the for loop to repeat every second.
    display.display();  // Display contents on OLED.
    }
    setColour(0, 0, 0);  // RGB set back to OFF.
    temperatureReading();  // Calling the temperateReading(); function to now
run.
    delay(5000);  // 5 second delay.

}

/*
 * This function below displays the temperature sample onto the LCD display
 * for the user to see. It is also used to make a decision,
 * based on the temperature sample, whether the user is allowed
 * to access the building.
 */

void temperatureReading(){
  display.clearDisplay();  // Clear the previous from the OLED.
  display.setTextColor(WHITE);  // Set text colour to white.
  display.setTextSize(3);  // Set text size to 3 pixels.
  display.setCursor(6,10);  // Set cursor to 6,10.
  display.print(temperature);  // Display temperature onto the OLED.
  display.setTextSize(1);  // Set text size to 1 pixel.
  display.setCursor(100,8);  // Set cursor to 100,8.
  display.print("o");  // Display 'o' on the OLED.
  display.setTextSize(3);  // Set text size to 3 pixels.
  display.setCursor(108,10);  // Set cursor to 108,10.
  display.print("C");  // Display 'C' on the OLED.
  display.display();  // Display contents on OLED.
  delay(3000);

  if (temperature > 25){  // If the temperature is greater than 25:
    Serial.print("ACCESS DENIED");  // Print the 'ACCESS DENIED' in serial
monitor.
    Access = "DENIED";  // Setting the variable Access to "DENIED".
    display.clearDisplay();  // Clear the previous from the OLED.
    display.setTextSize(2);  // Set text size to 2 pixel.
    display.setCursor(24,20);  // Set cursor to 24,20.
    display.print("ACCESS");  // Dislay "ACCESS" onto the OLED.
    display.setCursor(24,50);  // Set cursor to 24,50.
    display.print("DENIED");  // Dislay "DENIED" onto the OLED.
    setColour(255, 0, 0);  // Set RGB to Red.
    digitalWrite(buzz,HIGH);  // Buzzer on
    display.display();  // Display contents on OLED.
    delay(5000);  // 5 Second delay
```

```
    setColour(0, 0, 0);  // Set RGB to OFF.
    digitalWrite(buzz,LOW);  // Buzzer off
    }
    else if (temperature < 25){  // If the temperature is less than 25:
      Serial.println("ACCESS GRANTED ");  // Print the 'ACCESS GRANTED' in serial
monitor.
      Access = "GRANTED";  // Setting the variable Access to "GRANTED".
      display.clearDisplay();  // Clear the previous from the OLED.
      display.setTextSize(2);  // Set text size to 2 pixels.
      display.setCursor(24,20);  // Set cursor to 24,20.
      display.print("ACCESS");  // Dislay "ACCESS" onto the OLED.
      display.setCursor(24,50);  // Set cursor to 24,50.
      display.print("GRANTED");  // Dislay "GRANTED" onto the OLED.
      setColour(0, 0, 255);  // Set RGB to Green.
      digitalWrite(buzz,HIGH);  // Buzzer on
      display.display();  // Display contents on OLED.
      delay(5000);  // 5 Second delay
      setColour(0, 0, 0);  // Set RGB to off.
      digitalWrite(buzz,LOW);  // Buzzer off
      }
      else {
        setColour(255, 0, 0);  // Set RGB to red.
        display.clearDisplay();  // Clear the previous from the OLED.
        Serial.println("Error Please Try again"); // Print "Error Please Try
again" on serial monitor.
        display.setTextSize(2);  // Set text size to 2 pixels.
        display.setCursor(30,20);  // Set cursor to 30,20.
        display.print("ERROR");  // Display "ERROR" on the OLED.
        display.display();  // Display contents on OLED.
        delay(5000);  // 5 Second delay
        setColour(0, 0, 0);  // Set RGB to off.
        RFID();  // Return to the RFID() function.
      }
      CardUID = "998DADB2";
      temperatureTransmission(); // Calling the temperatureTransmission();
function to now run.


}

  // The two functions below are used for operation of the RGB LED.

/*
 * The function below takes 3 arguments relating to each pin
 * on the RGB LED which areused to store the analog value
 * created in the rgbStart() function. AnalogWrite is used to apply
 * a PWM signal to the pins which represent the brightness applied.
 */

void setColour(int red_light_value, int green_light_value, int blue_light_value)
 {
  analogWrite(ledRed, red_light_value);  // analogWrite using the ledRed pin and
the analog value.
  analogWrite(ledGreen, green_light_value);  // analogWrite using the ledGreen
pin and the analog value.
  analogWrite(ledBlue, blue_light_value);  // analogWrite using the ledBlue pin
and the analog value.
}

/* The RGB LED works by applying different intensities of voltage
 *  to the R, G, B, pins. Red, Green and Blue are created by having 100% duty
cycle
 *  on the Red, Green or Blue pins. Other colours like purple or yellow are made
 *  my adding different duty cycle to each pin as shown below in the function.
 */

void rgbStart(){
  setColour(255, 0, 0);  // Red Color
```

```
  delay(500);
  setColour(0, 255, 0);  // Green Color
  delay(500);
  setColour(0, 0, 255);  // Blue Color
  delay(500);
  setColour(255, 255, 255);  // White Color
  delay(500);
  setColour(170, 0, 255);  // Purple Color
  delay(500);
  setColour(0, 255, 255);  // aqua
  delay(500);
  setColour(255, 255, 0);  // yellow
  delay(500);
}


/*
 * This function is used to transmit the three variables: temperature,
 * CardUID & Access to the google sheets document which has been directly
 * linked using the third-party service called 'Pushing Box'
 */

void temperatureTransmission(){
WiFiClient client;  // Instantiating the WiFi object

  // API service using WiFi Client through PushingBox
 if (client.connect(WEBSITE, 80))
      {
         client.print("GET /pushingbox?devid=" + devid
       + "&temperature=" + (float) temperature
       + "&CardUID="     + (String) CardUID
       + "&Access="      + (String) Access
       );

/* Data is sent to Pushingbox using the following format:
 *  ?temperature=$temperature$&CardUID=$CardUID$&Access=$Access$
 *
 * The URL which is sent to Pushingbox.com looks like this:
 *
http://api.pushingbox.com/pushingbox?devid=v442D0A72C12BCE7&temperature=$temperature$&CardUID=$CardUID$&Access=$Access$
 *
*/

      /*
       * HTTP 1.1 provides a persistent connection, allowing batched requests
       * or pipelined to an output buffer
      */
      client.println(" HTTP/1.1");
      client.print("Host: ");
      client.println(WEBSITE);
      client.println("User-Agent: MKR1000/1.0");
      client.println();
      Serial.println("\nData Sent");
      delay(5000);
      }


}


/* This function is used to display the WiFi SSID, Local IP Address
 * and WiFi RSSI allowing the user to check which network they have
 * successfully been connected to as well as the signal strengh of the Wifi.
 */

void printWifiStatus() {
  // print the SSID of the network you're attached to:
```

```
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // print your WiFi shield's IP address:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);

  // print the received signal strength:
  long rssi = WiFi.RSSI();
  Serial.print("signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");
}
```

Google Sheets Script (Figure 32)

```
//-----------------------------------------------
//Originally published by Mogsdad@Stackoverflow
//Modified for by Jamie Buick for EEE527
//-----------------------------------------------
/*

GET request query:

https://script.google.com/macros/s/<gscript id>/exec?celData=data_here
--------------------------------------------------------------------------

GScript, PushingBox and Arduino MRK1000 Variables in order:

temperature
CardUID
Access
-----------------------------------------------------
*/


/* Using spreadsheet API */

function doGet(e) {
  Logger.log( JSON.stringify(e) );  // view parameters

  var result = 'Success'; // assume success

  if (e.parameter == undefined) {
    result = 'No Parameters';
  }
  else {
    var id = ('1RdE4Fv-qCHV-b-
r7X4sElEmn2ByY_TaB5GJMLQdjOmc');//docs.google.com/spreadsheetURL/d
    var sheet = SpreadsheetApp.openById(id).getActiveSheet();
    var newRow = sheet.getLastRow() + 1;
    var rowData = [];
    //var waktu = new Date();

    rowData[0] = new Date(); // Timestamp in column A

    for (var param in e.parameter) {
      Logger.log('In for loop, param='+param);
      var value = stripQuotes(e.parameter[param]);
      //Logger.log(param + ':' + e.parameter[param]);
      switch (param) {
        case 'temperature':
          rowData[1] = value;
```

```
          break;
         case 'CardUID':
          rowData[2] = value;
          break;
        case 'Access':
          rowData[3] = value;
          break;
        default:
          result = "unsupported parameter";
      }
    }
    Logger.log(JSON.stringify(rowData));

    // Write new row below
    var newRange = sheet.getRange(newRow, 1, 1, rowData.length);
    newRange.setValues([rowData]);
  }

  // Return result of operation
  return ContentService.createTextOutput(result);
}

/**
* Remove leading and trailing single or double quotes
*/
function stripQuotes( value ) {
  return value.replace(/^["']|['"]$/g, "");
}
```