IR Temp Team 4 Members: Andrew Hang, Cameron Lein, Christian Ovchinikov

Instructor: Brandilyn Coker

TA: Jacob Cook

ECE 342: Junior Design II

System Verification

The system verification was the final leg of the project. Our team was now working to bridge connections between each of our blocks that we have been working on during the term. The enclosure was designed using Blender and printed using school resources. The manufactured PCB was then installed into the enclosure and our final portion of testing was started. We initially were having issues with figuring out how to host an online database for the stored temperature readings. However, after some brainstorming, our team decided to use Google Sheets and APIs to communicate via the serial communication port. This was implemented using a python script and would be the final key component of our project before demonstrations.

Top-Level Block Diagrams:

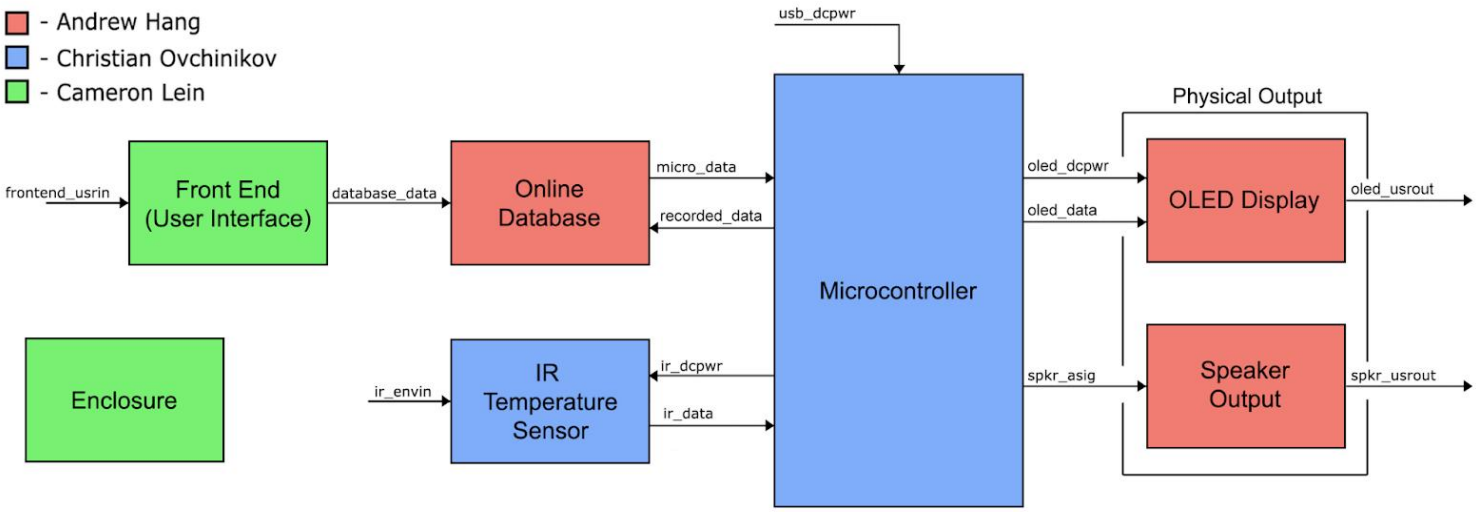


Figure 1: Top Level Block Diagram of the contactless IR temperature sensor showing each member's block.

Interface Definitions:

Interface Name	Interface Type	Specifics
usb_dcpwr	DC Power	V_{MAX} = 5V V_{MIN} = 3.3V I_{MAX} = 40mA
frontend_usrin	User Input	1) Start temperature reading 2) Switch between °F and °C 3) User ID HTML language.
database_data	Data (code)	Front End sends user selections to Google Sheets using APIs. HTML language.
ir_envin	Environmental Input	-70°C to +380°C ± 0.5°C Accuracy 90° Field of view-
micro_data	Data (code)	Baud Rate = 9600 Serial Communication “F” = Read in Fahrenheit “C” = Read in Celsius Python language.
recorded_data	Data (code)	Baud Rate = 9600 Serial Communication Python language.
ir_dcpwr	DC Power	V_{MAX} = 5.5V V_{MIN} = 4.5V I_{MAX} = 2 mA $I_{TYPICAL}$ = 1.3 mA
ir_data	Data (code)	Baud Rate = 9600 I2C Communication (SDA/SCL). Data is sent through the serial data line. Arduino IDE language.
oled_dcpwr	DC Power	V_{MAX} = 5V V_{MIN} = 3.3V
oled_data	Data (code)	I2C Communication Digital Write through SDA pin. Arduino IDE language.
spkr_asig	Analog Signal	V_{MAX} = 9V V_{MIN} = 3V I_{MAX} = 25mA
oled_usrout	User Output	Outputs read temperature to OLED Display.

spkr_usrout	User Output	V_{MAX} = 16V V_{MIN} = 3V Min Output = 80 dB

Table 1: Interface name, type, and properties for each block in the diagram.

Bill of Materials:

Reference Designators	Device	Datasheet	Part No.	Cost/unit
U1	IR Temperature Sensor	https://www.melexis.com/en/documents/documentation/datasheets/datasheet-mlx90614	MLX90614ESF	$11.98
M1	Microcontroller	https://www.arduino.cc/en/uploads/Main/ArduinoNanoManual23.pdf	Arduino Nano 3.0 (ATmega328P)	$10.99
U2	OLED Display	https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf	SSD1306	$8.59
SG1	Speaker Output	https://www.kaili-buzzer.com/Buzzer-Datasheet/Piezo-Buzzer/HP1470X.pdf	HP147012X	$0.73
R1	1/4W Resistor (100Ω)	https://www.futurlec.com/Resistors/Res14W_Technical.shtml	CFR0W4	$0.22

Table 2: Bill of materials for each electrical or mechanical component in the project.

Mechanical Drawings:

The mechanical drawings in the figures below were found on the internet and referenced in the figure captions. These drawings were useful in the creation of the enclosure because they gave accurate measurements of each component.

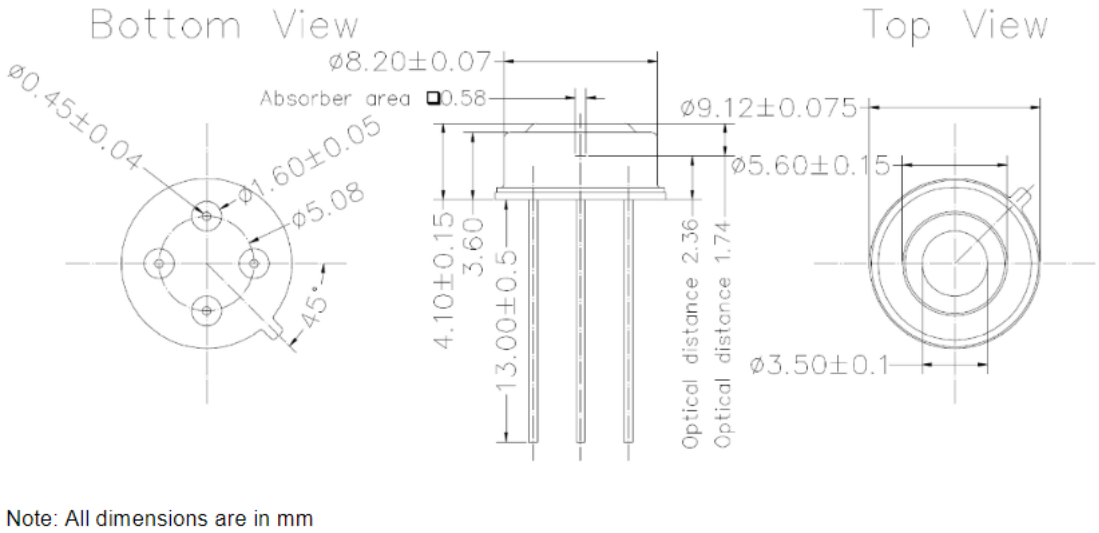


Figure 2: Mechanical drawings of the MLX90614 infrared sensor from the Melexis datasheet (page 47), all measurements in millimeters *(mm)*.

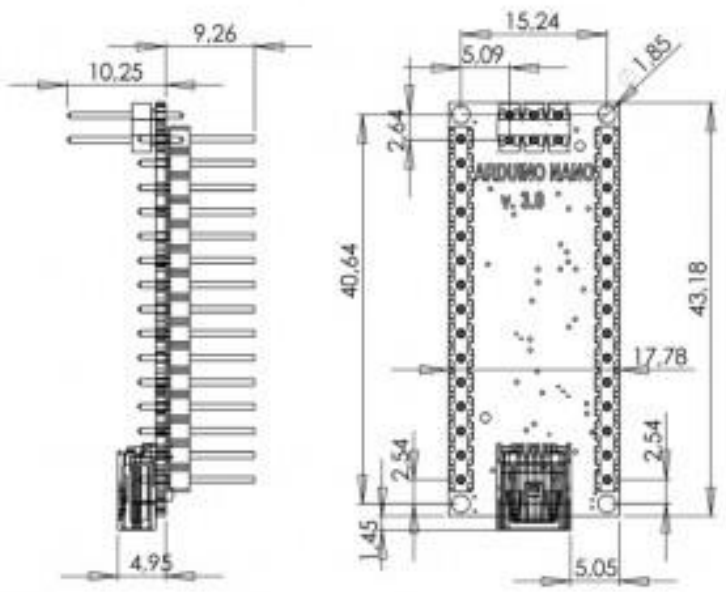


Figure 3: Mechanical drawings of the Arduino Nano v3.0 provided by JakartaNotebook, all measurements in millimeters *(mm)*.

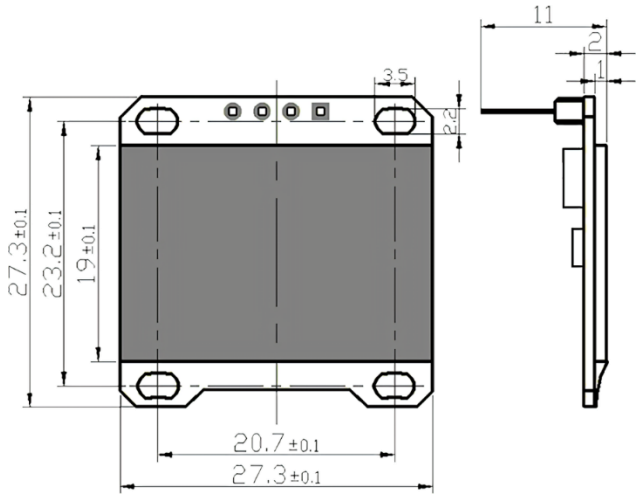


Figure 4: Mechanical drawings of the SSD1306 OLED Display provided by an open-source site, all measurements in millimeters *(mm)*.

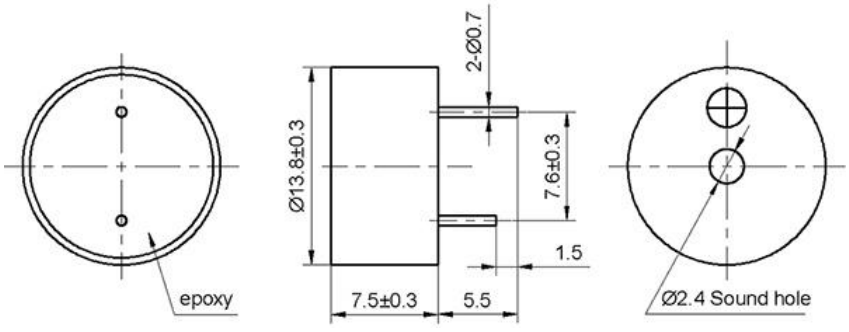


Figure 5: Mechanical drawings of the Piezo Buzzer provided by Kaili Buzzer, all measurements in millimeters *(mm)*.

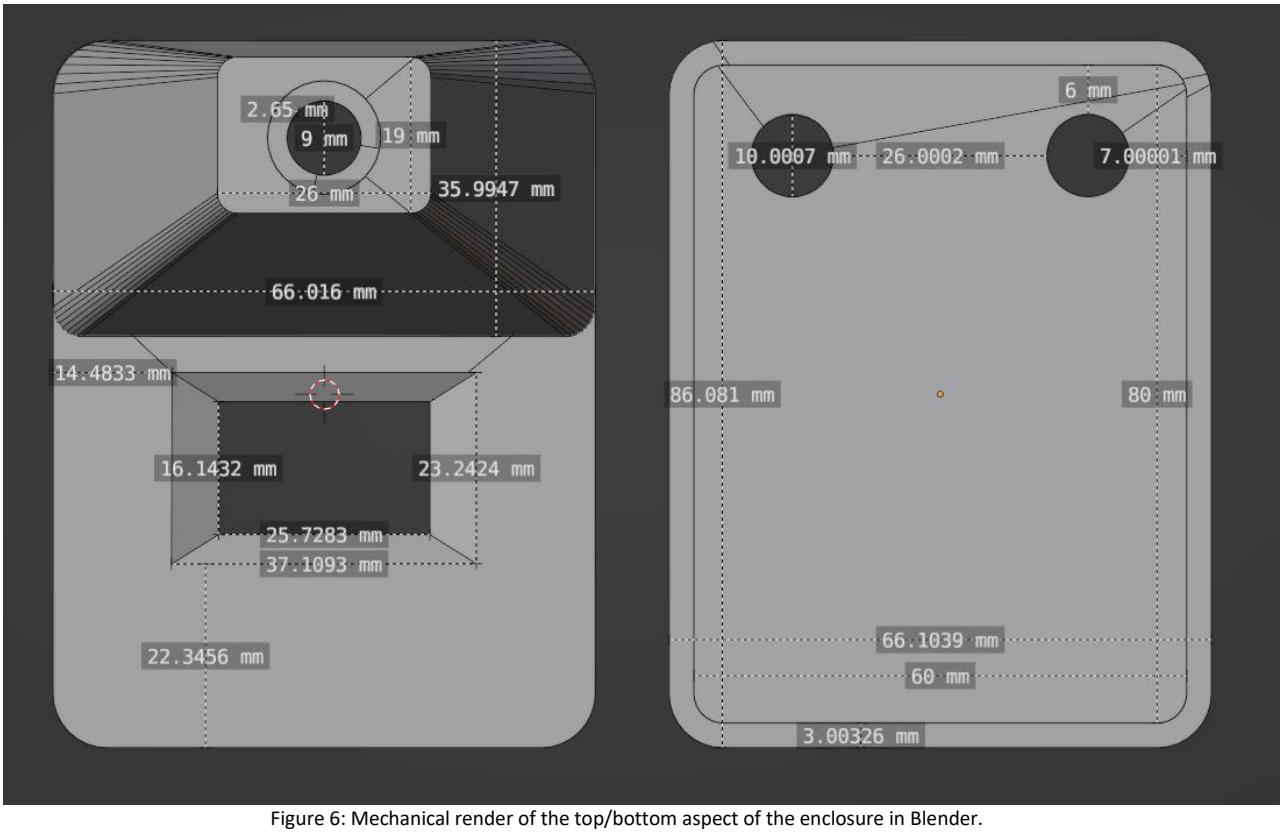


Figure 6: Mechanical render of the top/bottom aspect of the enclosure in Blender.

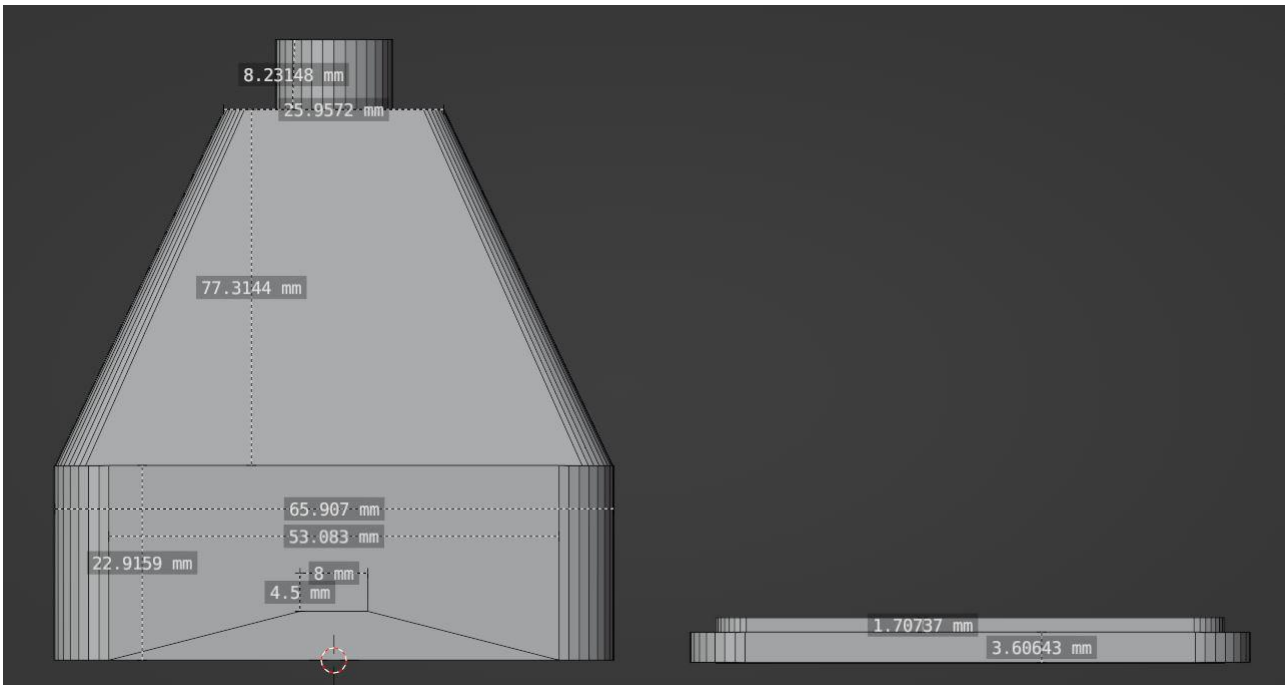


Figure 7: Mechanical render of the side profile of the enclosure in Blender.

Schematics:

Below are two industry-standard schematics, the first in *Figure 4* shows the full project connections and the second in *Figure 5* shows the PCB schematic. The schematics below also correlate to the values for Reference Designators found in the bill of materials from *Table 2*.

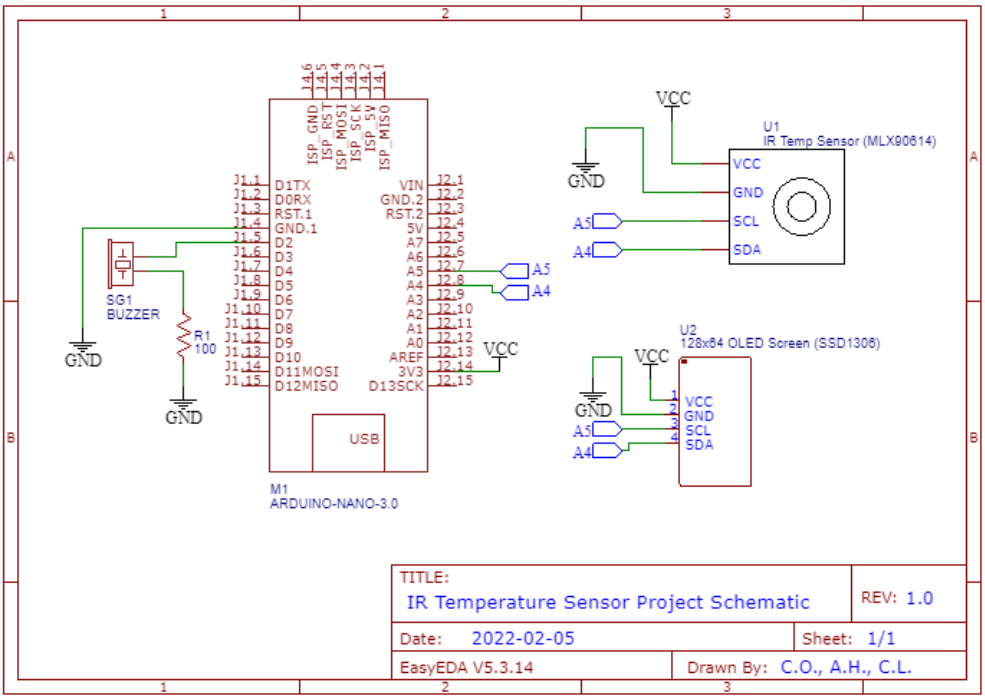


Figure 8: A schematic of the complete IR Temperature Sensor Project showing all connections between components.

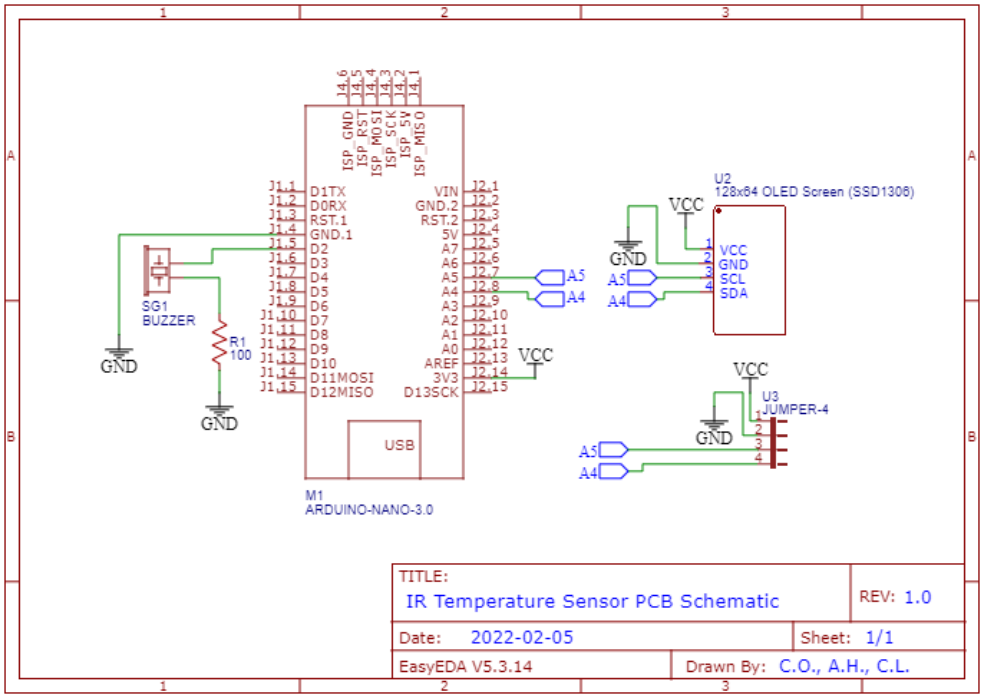


Figure 9: A schematic of the complete IR Temperature Sensor PCB Design.

PCB Design:

The PCB design we decided to go with was placing the microcontroller, the OLED display, the buzzer, and 4 jumpers to connect the IR temperature sensor. We decided to do this to centralize the components that we could into one area of the enclosure. The jumpers will be connected with wires up to the IR temperature sensor which will be up above the display so that the user can see the display when they are getting their reading done. The buzzer will also be around ear level so there won't be much amplification needed for the tone to be heard. The PCB was created using AutoDesk Eagle and was broken down into two layers. We also decided to have components on both sides of the PCB to allow the screen to sit outward with the Arduino Nano and other components tucked behind in the enclosure. The figure below was the drawing in AutoDesk Eagle of the PCB. There are two layers, red being the top PCB layer and blue being the bottom. This is to avoid any overlapping in wires. We also added some silk text with our names and the project title for some personalization.

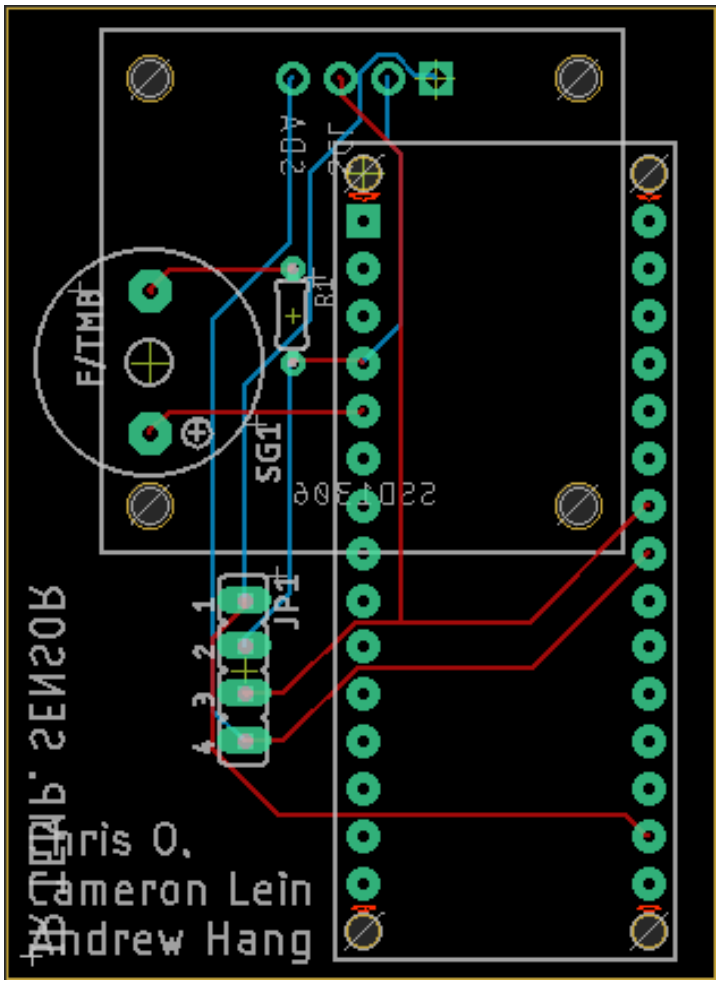


Figure 10: Screenshot of the PCB design in AutoDesk Eagle.

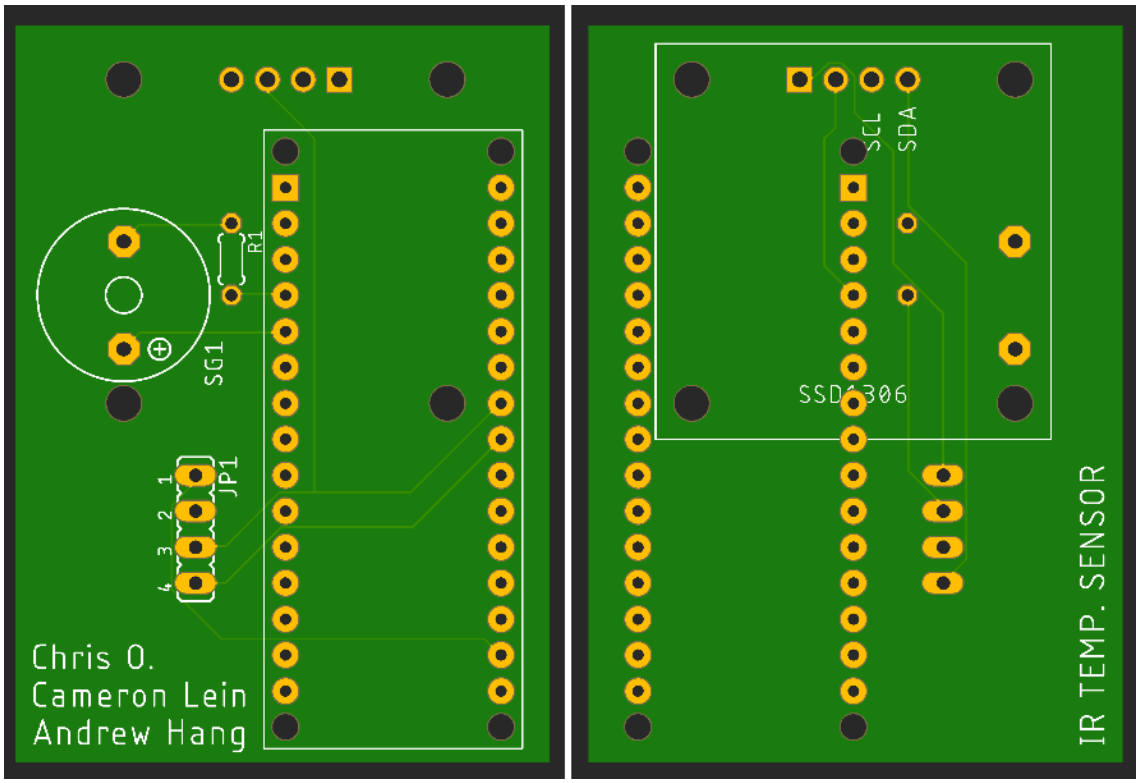


Figure 11: Front *(right)* and back *(left)* views of the PCB design before exporting for manufacturing.

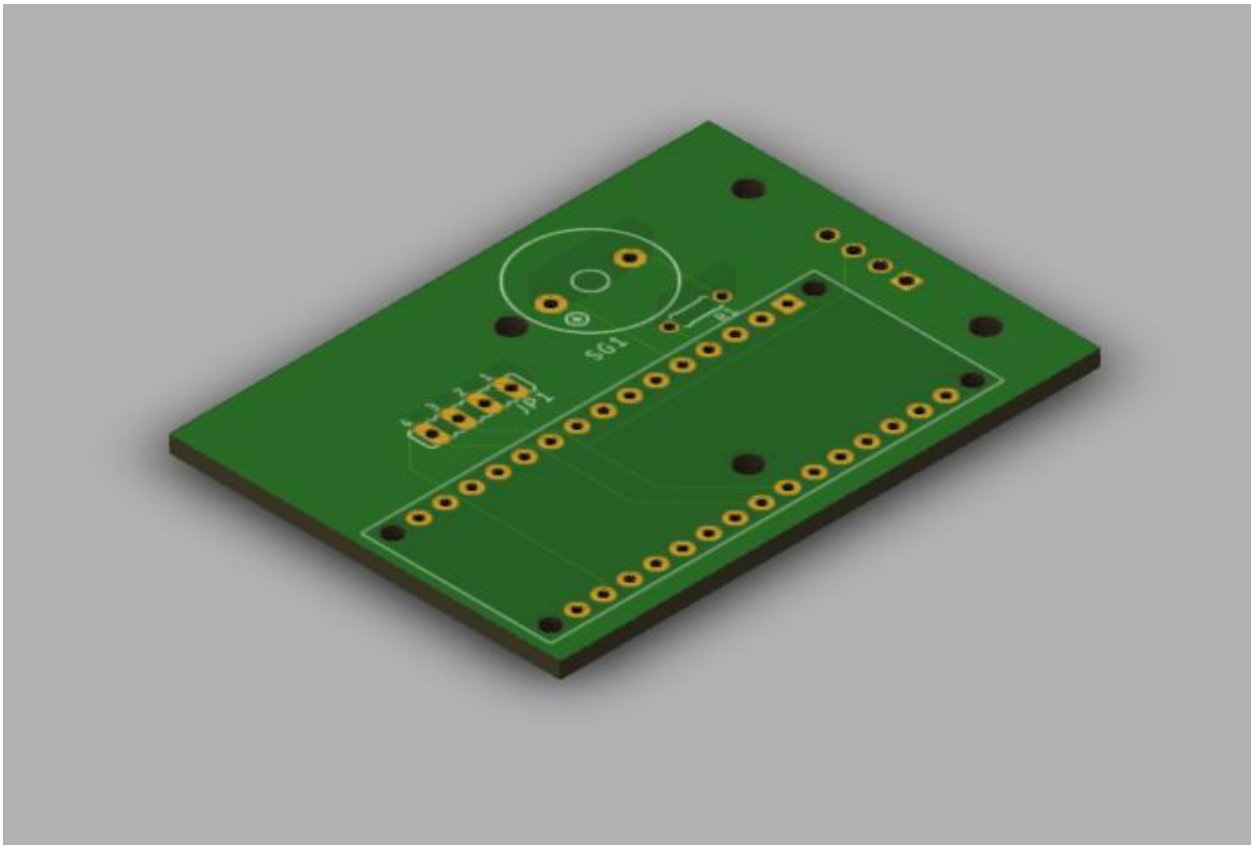


Figure 12: Orthographic view of the PCB board 3D modeled in AutoDesk Fusion 360.

Enclosure Design:

The enclosure features 3mm-thick walls, a Mini-B USB porthole, a beveled slot for the LCD, room to house the PCB with the microcontroller, an elongated neck for readability, a telescoped hole for reducing the FOV of the IR sensor, and a snap-in backplate with tack holes for wall-mounting. It was modeled in Blender, with the dimensions X = 66, Y = 86, Z = 113 (combined front and back plates). It was printed in white PLA at the TekBots store, then sanded and filed to smoothness.

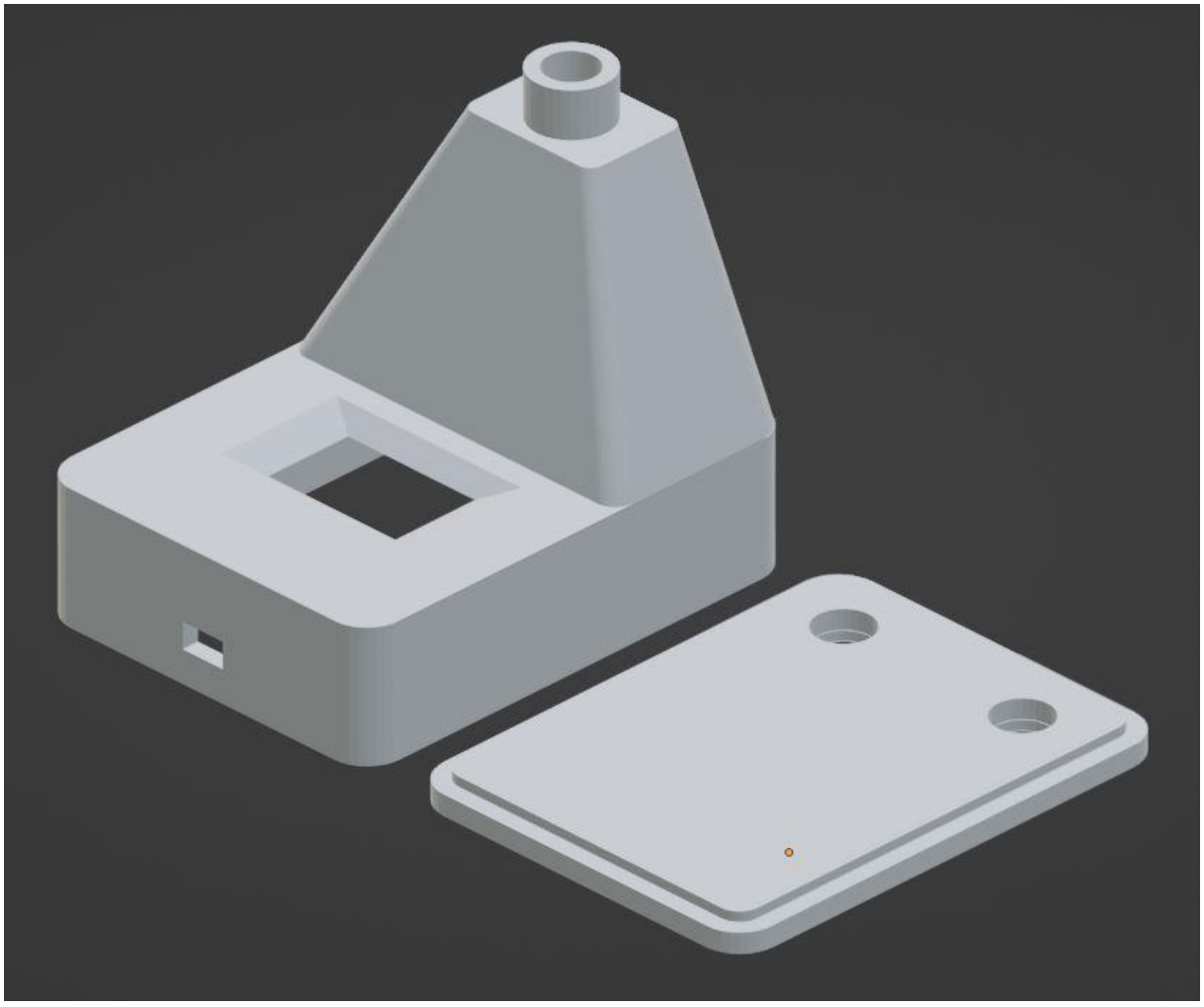


Figure 13: An isometric screenshot of the 3D model of the enclosure in Blender.

Remote Server & Online Database:

Scanning the QR code in the right pane takes the user to the HTML control panel, hosted on a remote Debian server configured with Nginx. This was chosen to avoid the headache of installing an app. The web page contains form options for customizing settings, including a temperature unit toggle and a field to enter a unique ID number, such as your ONID. Upon submitting this information, the form data is sent to the database, and the scanner begins its reading. The figure below shows a screenshot of the website which is viewable on desktop and mobile devices. The link is also listed below in the figure caption.

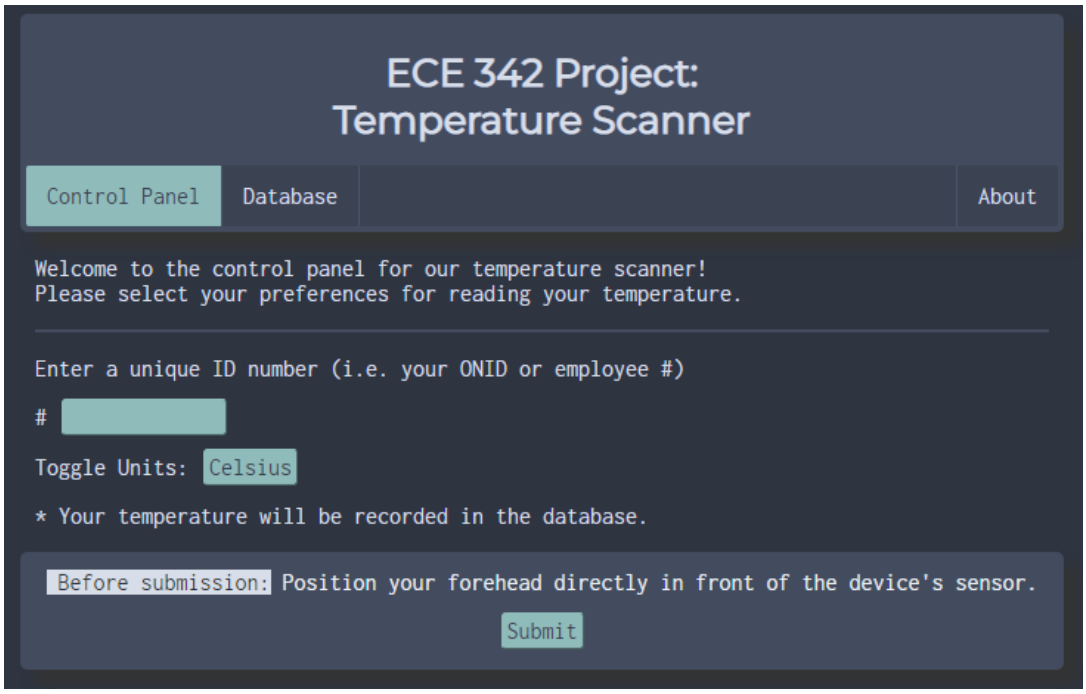
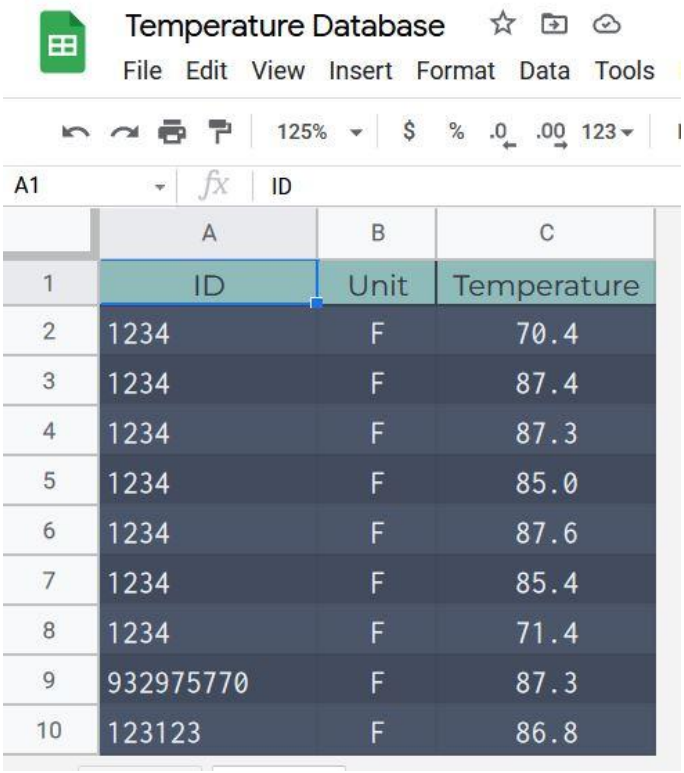


Figure 14: A screenshot of the website seen from the QR Code or the provided link.

The external database communicates with the microcontroller when a user selects Submit on the website. This will then update the Google Sheets database with a new row indicating the user's ID and desired units. The unit choice is then sent via serial communication to the Arduino Nano in bytes to trigger the reading for that particular temperature unit. The temperature is then outputted through the same serial connection back to the database, updating the corresponding Temperature cell.



	A	B	C
1	ID	Unit	Temperature
2	1234	F	70.4
3	1234	F	87.4
4	1234	F	87.3
5	1234	F	85.0
6	1234	F	87.6
7	1234	F	85.4
8	1234	F	71.4
9	932975770	F	87.3
10	123123	F	86.8

Figure 15: A screenshot of the database which can be found on the website from the QR Code or the provided link.

Software *(Microcontroller, Database, Website Code)*:

The microcontroller code was designed in the Arduino IDE. The input of the infrared temperature sensor and the output of the OLED display screen were the first objects to be implemented into the code. This allowed for testing and designing the displayed output early on in the project. Soon after the code was then set up to output to the buzzer and to configure the conditional statements for detecting a fever. The code was added once more to utilize the communication between the Google Sheets file and the Arduino Nano. The microcontroller gains the ability to manipulate the data on the online file through the serial communication port using a python script and APIs. The Arduino code was written to continuously loop waiting for an incoming byte through the serial port. This byte will come from the updated cells on the database which will be either an “F” or a “C” telling the microcontroller to either run in Fahrenheit or Celsius. The code utilizes a handful of libraries for each component. The library used for the infrared sensor allows us to read in the temperature using a function. This number was then pushed through a regression function to output a more accurate reading. This output is finally transmitted to the serial to be sent back to the Google Sheets file and will also be written to the OLED display.

Microcontroller Code:

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_MLX90614.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT   64
#define OLED_RESET   -1        // Reset pin # (or -1 if sharing Arduino reset pin)
#define SCREEN_ADDRESS  0x3C   // Screen Address in memory

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET); // Initializes the OLED screen
Adafruit_MLX90614 mlx = Adafruit_MLX90614();                              // Initializes the ir sensor

int incomingByte;    // Reads incoming serial data

const int buzzer = 2;  // Constant value of 2 indicating D2 output on Arduino Nanoe

float finalTemp; // Stores final temp
char finalUnit;   // Stores unit used

double tempRead;
double tempAdj;

void setup() {
  Serial.begin(9600);                                 // Starts serial comm with baud rate of 9600
  mlx.begin();                                        // Starts the IR temperature reading
  display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS);  // Start the OLED display
  display.clearDisplay();
  pinMode(buzzer, OUTPUT);                                      // Sets D2 to output to Buzzer
}

void loop() {
  int counter = 200;           // Counter used to record temperatures for around 6.5 seconds

  display.clearDisplay();

  // Text at the top of the screen
  display.setTextSize(2);
  display.setTextColor(WHITE);
  display.setCursor(0, 16);
  display.println("Please use \n  QR Code");
  display.display();
```

```
// see if there's incoming serial data:
if (Serial.available() > 0) {
        // read the oldest byte in the serial buffer:
        incomingByte = Serial.read();

        // if it's a capital C (ASCII 67), print to display in Celsius:
        if (incomingByte == 'C') {
        // Initial beep to let the user know that their temperature is being recorded
        digitalWrite(buzzer, HIGH);  // Turn on Buzzer
        delay(100);                     // Delay for 1/10 sec
        digitalWrite(buzzer, LOW);  // Turn on Buzzer

        // Loops until counter = 0, which is approximately 5 seconds
        while (counter) {
        display.clearDisplay();

        // Text at the top of the screen
        display.setTextSize(1);
        display.setTextColor(WHITE);
        display.setCursor(0, 0);
        display.println("Temperature:");

        // Text in the middle of the temperature reading
        display.setTextSize(3);
        display.setCursor(10, 16);

        tempRead = mlx.readObjectTempC();
        tempAdj = 2.52 + tempRead;
        display.println(tempAdj, 1);
        display.setCursor(90, 16);
        display.println("C");

        // Wait text at the bottom of the screen
        display.setTextSize(1);
        display.setTextColor(WHITE);
        display.setCursor(0, 45);
        display.println("Wait for the tone");
        finalTemp = tempAdj;

        counter = counter - 1;
        if (counter % 2 == 0)
        display.display();
        }
        finalUnit = 'C';
        Serial.print(finalTemp, 1);
        FinalPrint();
        }

        // if it's an F (ASCII 70), print to display in Fahrenheit:
        if (incomingByte == 'F') {
        // Initial beep to let the user know that their temperature is being recorded
        digitalWrite(buzzer, HIGH);  // Turn on Buzzer
        delay(100);                     // Delay for 1/10 sec
        digitalWrite(buzzer, LOW);  // Turn on Buzzer

        // Loops until counter = 0, which is approximately 5 seconds
        while (counter) {
        display.clearDisplay();

        // Text at the top of the screen
        display.setTextSize(1);
        display.setTextColor(WHITE);
        display.setCursor(0, 0);
        display.println("Temperature:");

        // Text in the middle of the temperature reading
        display.setTextSize(3);
        display.setCursor(10, 16);
        tempRead = mlx.readObjectTempF();
```

```
          tempAdj = 4.53 + tempRead;
          display.println(tempAdj, 1);
          display.setCursor(90, 16);
          display.println("F");

          // Wait text at the bottom of the screen
          display.setTextSize(1);
          display.setTextColor(WHITE);
          display.setCursor(0, 45);
          display.println("Wait for the tone");
          finalTemp = tempAdj;

          counter = counter - 1;
          if (counter % 2 == 0)
          display.display();
          }
          finalUnit = 'F';
          Serial.print(finalTemp, 1);
          FinalPrint();
          }
  }
}

// Function used to print the final display with the final temperature, unit and indication of a fever
void FinalPrint() {
  // Final print screen with the buzzer in the background to tell the user they are completed
  display.clearDisplay();
  display.display();

  // Prints final temperature of user
  display.setTextSize(3);
  display.setCursor(10, 16);
  display.println(finalTemp, 1);
  display.setCursor(90, 16);

  if (finalUnit == 'C') {
          display.println("C");

          // Checks and prints if the user has a fever of over 38 C
          if (finalTemp > 33.7) {
          display.setTextSize(2);
          display.setTextColor(WHITE);
          display.setCursor(0, 45);
          display.println("   FEVER");
          display.display();
          // If fever beep faster
          digitalWrite(buzzer, HIGH);  // Turn on Buzzer
          delay(250);                    // Delay for 1/4 sec
          digitalWrite(buzzer, LOW);   // Turn off Buzzer
          delay(250);
          digitalWrite(buzzer, HIGH);
          delay(250);
          digitalWrite(buzzer, LOW);
          delay(250);
          digitalWrite(buzzer, HIGH);
          delay(250);
          digitalWrite(buzzer, LOW);
          delay(250);
          digitalWrite(buzzer, HIGH);
          delay(250);
          digitalWrite(buzzer, LOW);
          delay(250);
          digitalWrite(buzzer, HIGH);
          delay(250);
          digitalWrite(buzzer, LOW);
          delay(2500);                    // Total delay of 5 seconds for user to see their final
temperature
          }
          else {
```

```
          display.display();
          digitalWrite(buzzer, HIGH);  // Turn on Buzzer
          delay(500);                    // Delay for 1/2 sec
          digitalWrite(buzzer, LOW);   // Turn off Buzzer
          delay(500);
          digitalWrite(buzzer, HIGH);
          delay(500);
          digitalWrite(buzzer, LOW);
          delay(500);
          digitalWrite(buzzer, HIGH);
          delay(500);
          digitalWrite(buzzer, LOW);
          delay(2500);                   // Total delay of 5 seconds for user to see their final
temperature
          }
  }
  else if (finalUnit == 'F') {
          display.println("F");

          // Checks and prints if the user has a fever of over 100.4 F
          if (finalTemp > 92.7) {
          display.setTextSize(2);
          display.setTextColor(WHITE);
          display.setCursor(0, 45);
          display.println("   FEVER");
          display.display();

          // If fever beep faster
          digitalWrite(buzzer, HIGH);  // Turn on Buzzer
          delay(250);                    // Delay for 1/4 sec
          digitalWrite(buzzer, LOW);   // Turn off Buzzer
          delay(250);
          digitalWrite(buzzer, HIGH);
          delay(250);
          digitalWrite(buzzer, LOW);
          delay(250);
          digitalWrite(buzzer, HIGH);
          delay(250);
          digitalWrite(buzzer, LOW);
          delay(250);
          digitalWrite(buzzer, HIGH);
          delay(250);
          digitalWrite(buzzer, LOW);
          delay(250);
          digitalWrite(buzzer, HIGH);
          delay(250);
          digitalWrite(buzzer, LOW);
          delay(2500);                   // Total delay of 5 seconds for user to see their final
temperature
          }
          else {
          display.display();
          digitalWrite(buzzer, HIGH);  // Turn on Buzzer
          delay(500);                    // Delay for 1/2 sec
          digitalWrite(buzzer, LOW);   // Turn off Buzzer
          delay(500);
          digitalWrite(buzzer, HIGH);
          delay(500);
          digitalWrite(buzzer, LOW);
          delay(500);
          digitalWrite(buzzer, HIGH);
          delay(500);
          digitalWrite(buzzer, LOW);
          delay(2500);                   // Total delay of 5 seconds for user to see their final
temperature
          }
  }
}
```

The database code was designed using python language to bridge the connection between the online database and the microcontroller. Since the microcontroller is being powered through the USB port, we were able to use the serial communication port to transmit and receive data. The python script uses a handful of libraries such as serial, gspread, and pandas. These libraries provided functions that allowed us to pull and push through the Google Sheets API, manipulate and read cell data, and communicate via serial communication. The script initially loads in the Sheets file and reads the last updated cell for units. This cell is updated by the website we created that would allow the user to control the device free of contact. Once the user submits their information with the desired units for the temperature reading, a new row is added to the Sheets file with the student information and the desired unit. This unit is read in by the python script and sent through the serial communication port to the Arduino Nano which will trigger the corresponding reading. The script then waits and then reads the serial communication port again to receive the final read temperature to update the Sheets file. This is how we implemented an automated online database that virtually holds as many users as the Google Sheets file will hold rows.

Database Code:

```
# db.py - Manages exchange between microcontroller and database.
#         Also sends the activation signal to begin the readout.

import serial
import time
import gspread
import pandas as pd
from oauth2client.service_account import ServiceAccountCredentials

#define the scope
scope = ['https://spreadsheets.google.com/feeds','https://www.googleapis.com/auth/drive']

# add credentials to the account
creds = ServiceAccountCredentials.from_json_keyfile_name('C:/Users/Lein/Desktop/temperature-scanner-
342921-17ef87090809.json', scope)

# authorize the clientsheet
client = gspread.authorize(creds)

# get the instance of the Spreadsheet
sheet = client.open('Temperature Database')

# get the primary subsheet of the Spreadsheet
sheet_instance = sheet.get_worksheet(0)

# set up serial on COM4 (Cameron's laptop) and 9600 baud
ser = serial.Serial('COM4', 9600, timeout = 1)
time.sleep(2)

# init row update vars
rows = sheet_instance.row_count
rowCnt = rows

while True:
    # update worksheet to compare current row
    sheet_instance = sheet.get_worksheet(0)
    rows = sheet_instance.row_count
    time.sleep(1)

    if (rows != rowCnt):
        # get user-preferred unit (deg F or C)
        unitRead = sheet_instance.cell(col=2,row=rows).value
        ser.write(unitRead.encode())

        # wait for temp scanner to finish
```

```
        time.sleep(8)

        # read, decode, and export temp to database
        temp = ser.read(4)
        output = temp.decode("utf-8")
        sheet_instance.update_cell(rows, 3, output)

        # increment row count
        rowCnt = rows
```

The website code, index.html, contains the HTML/CSS/Javascript used to provide the user with a remote control panel for operating the system. The Javascript functions entail initializing JQuery and the Google Sheets API, adding entries to the database, and unit toggle button functionality. The CSS contains several font/margin/padding/border/color settings for each module of the HTML body, which contains the readable text on the website. The site is formatted primarily for use on variable-sized mobile phone screens, but scales and adjusts to a standard computer monitor as well. Form data, including the user's ID and the temperature unit preference, are sent as form data to the database.

Website Code:

```
<!DOCTYPE html>
<html lang="en-us">
<head>
<title>TS Control Panel</title>
<link href="https://fonts.googleapis.com/css2?family=Inconsolata&family=Montserrat&display=swap"
rel="stylesheet">
<meta name="description" content="Enter user preferences">
<meta name="author" content="Cameron Lein">
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<link rel="icon" type="image/x-icon" href="favicon.ico">
<script
  src="https://code.jquery.com/jquery-3.4.1.js"
  integrity="sha256-WpOohJOqMqqyKL9FccASB9O0KwACQJpFTUBLTYOVvVU="
  crossorigin="anonymous">
</script>
<script>
function SubForm () {
          $.ajax({
          url:'https://api.apispreadsheets.com/data/DwTm2i4OjsOKPgTW/',
          type:'post',
          data:$("#submission").serializeArray(),
          success: function(){
          alert("Submission accepted :)")
          },
          error: function(){
          alert("There was an error :(")
          }
          });
}
</script>
<script>
  function toggle() {
          var elem = document.getElementById("tempUnit");
          var hide = document.getElementById("Unit");
          if (elem.value=="Fahrenheit") {
          elem.value = "Celsius";
          hide.value = "C"
          } else {
          elem.value = "Fahrenheit";
          hide.value = "F"
          }
  }
</script>
```

```
<style>
* {
  box-sizing:  border-box;
  font-family: 'Inconsolata', monospace;
  font-size: 16px;
  color: #d8dee9;
}
.header {
  background-color: #434c5e;
  color: #eceff4;
  padding: 0px;
  margin: 10px;
  border: 2px solid #434c5e;
  border-radius: 5px;
  box-shadow: 10px 10px 8px #333;
}

footer {
  bottom: 0;
  left: 0;
  margin-bottom: 10px;
  position: sticky;
  width: 100%;
}
.footer {
  background-color: #434c5e;
  color: #eceff4;
  padding: 10px 10px 0 10px;
  margin: 10px;
  border: 2px solid #434c5e;
  border-radius: 5px;
  width: auto;
  height: auto;
  text-align: center;
  box-shadow: 10px 10px 8px #333;
}

.main {
  margin: 0 20px 0 20px;
  text-align: justify;
}

h2 {
  font-family: 'Montserrat', sans-serif;
  padding: 0px 2px 0px 2px;
  text-align: center;
  font-size: 26px;
}

ul {
  list-style-type: none;
  margin: -8px 0 0 0;
  padding: 0;
  overflow: hidden;
  background-color: #3b4252;
  border: 2px solid #434c5e;
  border-radius: 4px;
}
li {
  float: left;
  border-right: 1px solid #4c566a;
}
li a {
  display: inline-block;
  color: #d8dee9;
  text-align: center;
  padding: 12px 14px;
  text-decoration: none;
```

```
}
.active {
  color: #3b4252;
  background-color: #8fbcbb;
}
li:last-child {
  border-right: none;
  border-left: 1px solid #4c566a;
}

.button {
  color: #3B4252;
  background-color: #8fbcbb;
  border: 1px outset #8fbcbb;
  border-radius: 3px;
  font-weight: 500;
  padding: 3px;
  margin: -6px;
}
mark {
  color: #2e3440;
  background-color: #d8dee9;
}

body {
  background-color: #2e3440;
}
</style>
</head>
<body>
<div class="header">
  <h2>ECE 342 Project:
  <br>Temperature Scanner</h2>
  <ul>
          <li><a class="active" href="#">Control Panel</a></li>
          <li><a href="https://docs.google.com/spreadsheets/d/1_KqDl0Hk-BMej6iBPifuAkL-
buHj4C5pySd2Ofcvjus/edit?usp=sharing" target="_blank">Database</a></li>
          <li style="float:right"><a href="#">About</a></li>
  </ul>
</div>

<div class="main">
  <p>Welcome to the control panel for our temperature scanner!
  <br>Please select your preferences for reading your temperature.</p>
  <hr style="height:2px;border-width:0;background-color:#434c5e">

  <p>Enter a unique ID number (i.e. your ONID or employee #)</p>
  <p>
          <form id="submission">
          #  <input class="button" type="number" id="ID" name="ID" min="1" max="999999999"><br><br>
          Toggle Units:  <input class="button" onclick="toggle()" type="button" value="Celsius"
id="tempUnit" name="Unit"></input>
          <input type="hidden" id="Unit" name="Unit" value="C"></radio>
          </form>
  </p>
  <p>* Your temperature will be recorded in the database.</p>
</div>

<footer>
  <div class="footer">
          <mark> Before submission:</mark> Position your forehead directly in front of the device's
sensor.
          <p><button class="button" onclick="SubForm()">Submit</button></p>
  </div>
<footer>
</body>
</html>
```

Time Report:

Our group used a Gantt chart on our shared drive that allowed us to keep track of each teammate's contributed time. We usually worked on the project together and had regular meetings in the library. This along with our organization of tasks, our team was able to keep the contributed hours relatively equal. The figure below shows a snippet of the Gantt chart that we used and the hours spent on each step.

WBS NUMBER	TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION	PCT OF TASK COMPLETE
1	**Project Conception**					
1.1	Eng Requirements Submission	[All]	1/10/22	1/11/22	1	100%
1.2	Block Diagram Submission	[All]	1/12/22	1/14/22	2	100%
1.3	Project Timeline	[All]	1/12/22	1/14/22	2	100%
1.4	Research: Infrared capture and mC	Christian	1/14/22	1/19/22	5	100%
1.5	Research: Database and outputs	Andrew	1/14/22	1/19/22	5	100%
1.6	Research: Enclosure & Inputs	Cameron	1/14/22	1/19/22	5	100%
1.7	Solidified Game Plan	[All]	1/19/22	1/21/22	2	100%
2	**Project Definition**					
2.1	Part Purchasing & Shipping	[All]	1/21/22	1/24/22	3	100%
2.2	IR Sensor completion	Christian	1/24/22	1/28/22	4	100%
2.3	LCD & Soundsystem completion	Andrew	1/24/22	1/28/22	4	100%
2.4	Control Buttons completion	Cameron	1/24/22	1/28/22	4	100%
3	**Project Realization**					
3.1	Block 1 Check-off	[All]	1/26/22	1/28/22	2	100%
3.2	Microcontroller	Christian	1/31/22	2/11/22	11	100%
3.3	Database	Andrew	1/31/22	2/11/22	11	100%
3.4	Enclosure	Cameron	1/31/22	2/11/22	11	100%
3.5	Design Feedback Session	[All]	1/31/22	2/4/22	4	100%
3.6	Block 2 Check-off	[All]	2/9/22	2/11/22	2	100%
3.7	System Compilation	[All]	2/13/22	2/18/22	5	95%
4	**Project Performance / Monitoring**					
4.1	Teamwork Reflection	[All]	2/17/22	2/18/22	1	0%
4.2	System Verification	[All]	3/2/22	3/4/22	2	0%
4.3	Project Showcase Presentation	[All]	2/28/22	3/11/22	11	0%

Figure 14: A screenshot displaying the hourly contributions of each member from our team's Gantt Chart.

Team Member	Overall Time Spent
Andrew	43 hours
Cameron	43 hours
Christian	43 hours

Table 3: Displaying the overall time spent by each teammate.