Team Members: Cole Arduser, Luke Farmer, Sam Loecke, Samuel Nicklaus
Team Ctrl Alt Elite
Professor Yang Liu, Yu-Jung Chu
ECE:4480
09/20/2023
Lab Report 1

Design Documentation

In this lab project, our goal was to design a portable temperature-measuring system along with a web interface. Features included a client-side user interface, with the ability to display temperature data and control certain functions through the GUI. For hardware, a third box contains a display to read out the temperature measured, a button, a battery, and a power switch. This box served as a battery-operated thermometer that helped display its temperature data on our client's GUI.

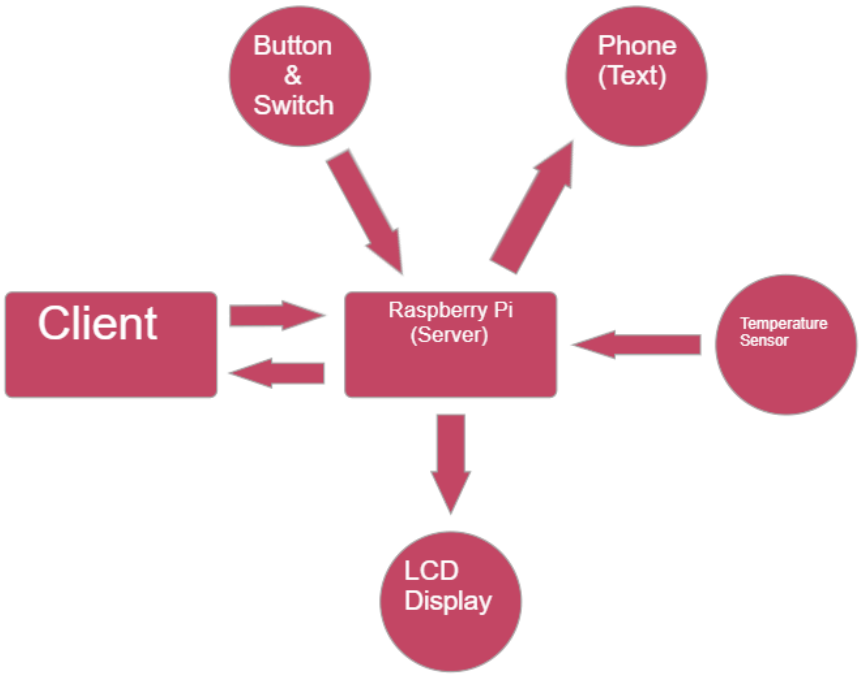


Figure 1: High-level view block diagram

Figure 1 shows us the relationship between the client, server, and hardware. The Raspberry Pi is the heart of our system, running the Python code that allows us to collect

temperature data, send out texts, and send data to the client. Attached to the Pi is a button, a switch, an LCD display, and a temperature probe. The button turns on the LCD display, showing the temperature that the probe reads in Celsius when the button is pushed down. The switch acts as an on-off switch where when flipped on, the LCD display does not show anything and the client GUI cannot receive data. In the client, the user is able to view the last 300 seconds of temperature data on a graph in Celsius. The user also has the option to change this graph to Fahrenheit, turn on the LCD on the third box remotely using a virtual button, and change text message settings such as the minimum temp to send a text, the maximum temp to send a text, and phone number.

Hardware Overview

We use a battery as our power source to allow for our device to be portable. The battery provides power to the Raspberry Pi which is the middle man for all of the hardware. We chose to use the Raspberry Pi because it was simple to interface with the physical components and has WiFi capabilities for communicating with the client-side application.

We put the button and toggle switch on Raspberry Pi GPIO pins and set the pins to trigger interrupts when the pins are pulled low. We have a hardware and software debounce on the push button with a capacitor and resistor combination. For the toggle switch, we chose to use only a software debounce to save space on the breadboard. These interrupts are then used in the software to cause other actions. We use the GPIO4 to read from the thermometer's data line to gather the temperature data. The temperature sensor communicates the temperature data to the pi using the one-wire interface. The pi has the one-wire interface communications built in, so we just needed to enable it and read from the one-wire data file. The final hardware components wired to the Raspberry Pi are the LCD I2C backpack and the LCD. We chose to use the I2C backpack because we found multiple Python libraries that used the LCD module we

used, but they all used the backpack. This saved us implementation time and pins on our Raspberry Pi.

The final part regarding our hardware is the container and thermometer connector. We considered 3D printing a box for our container, but after discussing the requirements with the TA, we decided that a Tupperware container would be the best option. We decided this because it would require less time to manufacture the container. For the thermometer connector, we used the connector that was available in the engineering electronic shop. Our final design was the battery in the bottom of the Tupperware with the Raspberry Pi and breadboard resting on top. The final design was physically robust and was able to survive drops and rotations.

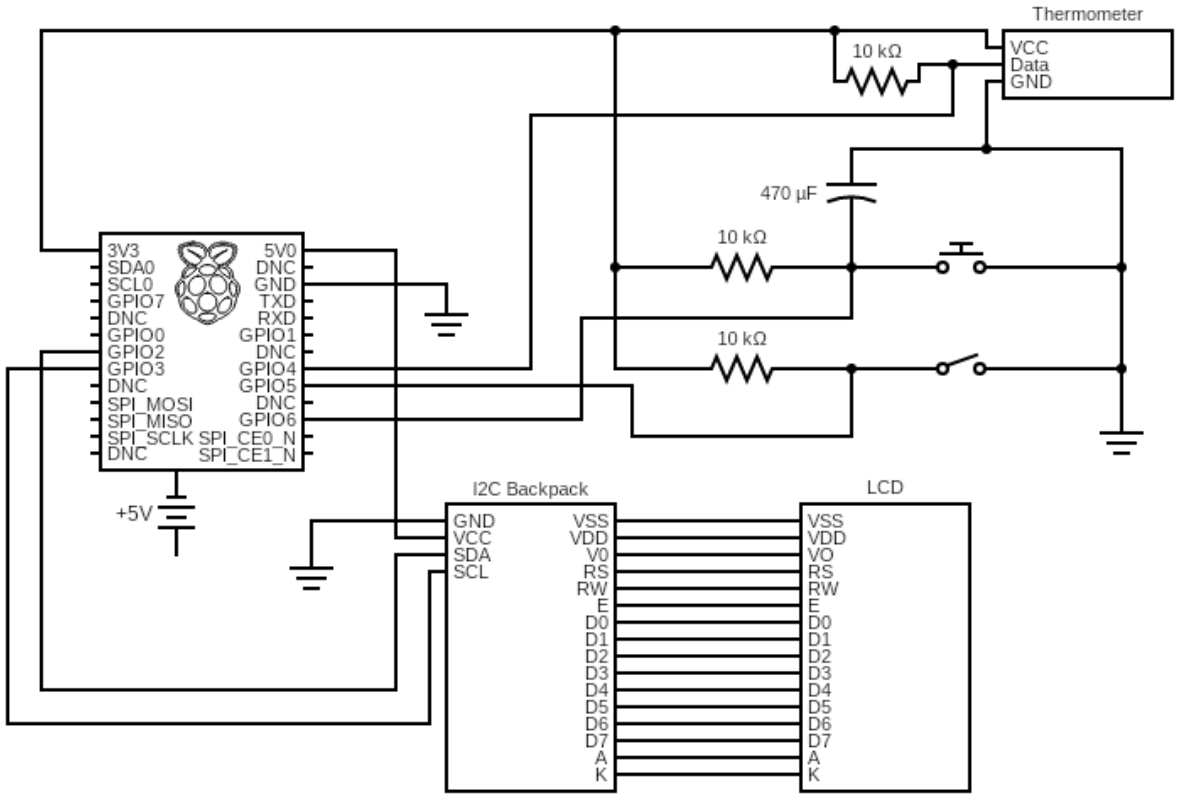


Figure 2: Schematic of all hardware components

Parts Table	
Schematic Part Name	**Part Number (Quantity)**
Raspberry Pi	Raspberry Pi 4 Model B (1)
Battery Source	Anker PowerCore 20100mAh (1)
I2C Backpack	HiLetgo 2pcs 5V 2004 1602 LCD Display IIC I2C Adapter (1)
LCD	Orient Display AMC1602AR-B-B6WTDW-I2C (1)
Thermometer	DS18B20 Temperature Sensor (1)
Male/Female Connector	4 Pin JST SM Connector LED Power Male to Female SM Wire Cable Adapter for 3528 5050 LED Light Strip (1)
Three-Pin Switch	NTE Electronics, Inc 54-305PC (1)
Push Button	CUI Devices TS02-66-50-BK-100-SCR-D (1)
Tupperware Container	24 Oz To-Go Box Black Plastic (1)
Velcro Strips	Velcro (4)
Bread Board	Solderless Breadboard 400 (1)
Cushion Tape	Cushion Tape Strips (8)

Figure 3: Parts List describing part name, number, and quantity

Client Overview:

Note: The main file that contains all the HTML, CSS, and Javascript is in Client/src/App.vue of the source code

Our client is written in Vue.js, which is a framework on top of javascript that utilizes HTML and CSS. Vuetifyjs was used as a bootstrap component to make the user design look better. Chartjs was used for the chart component. Core methods are written below:

created(): Runs on the creation of the webpage, calls updateData() and grabData() respectively.

updateData(): Runs every second. Initializes the x-axis of the graph and then pulls the most current temperature data from a global variable which is a list of 300 elements. Then it converts it to Fahrenheit if needed and pushes the information to the graph.

grabData(): Runs every second. Makes an HTTP request to the server for the last 300 seconds of temperature data. If the request is successful, this information is stored in a global variable to be used in updateData(), and the last element of the list is pushed to the screen as the current temperature. If the request fails, it enters a catch statement that shifts the graph over by one second, leaving a blank space on the rightmost side of the graph. It also sets the current temperature displayed to “no data available”.

tempToString(): Takes in temperature data and appends either °C, °F, or the correct error messages and displays it to the user on the home screen of the GUI.

swapTemp(): Changes the scale of the table, converts from C to F, and vice versa. Does this by converting the 300-element temperature data and then changing the scale of the table before updating the chart.

setPhone(): This function is used in the settings modal and sends minTemp, maxTemp, phone number, and carrier to the server. Checks are put in place to verify that min and max temp is a number and that the phone number is valid. Carrier is selected from a dropdown so the user cannot input a bad value. If the method detects that a user input is invalid, the user is prompted to enter the correct value and has to re-enter the information.

toggleBoxButton(boolVal): Function that takes in a boolean value and sends it over to the server via an HTTP post request. If true is pushed then the LCD display turns on the third box, if false is pushed then the LCD display turns off. All logic that deals with if the switch is on or off on the third box is processed server side.

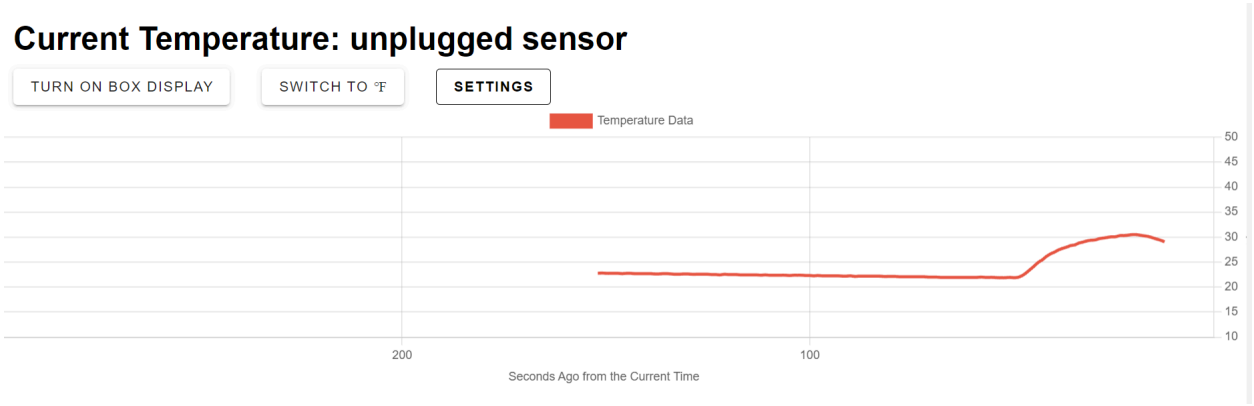


Figure 3: Web application home screen when the temperature sensor is unplugged

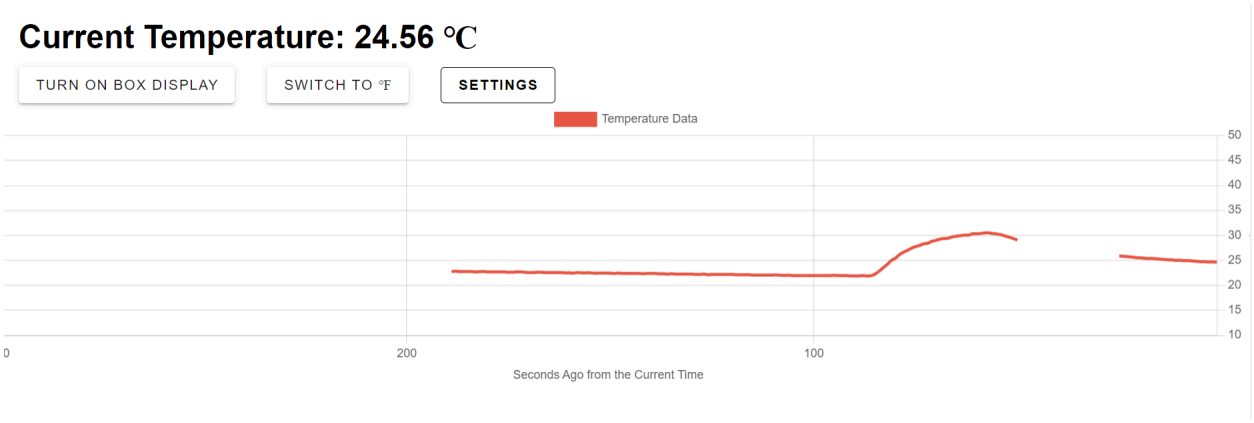


Figure 4: Web application home screen when the user first starts the application

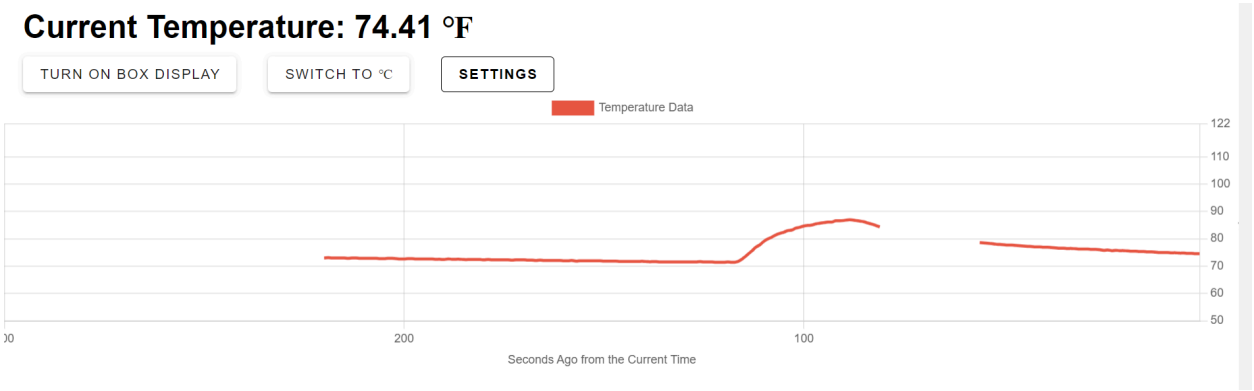


Figure 5: Web application home screen after the user clicks the “Switch to F” button

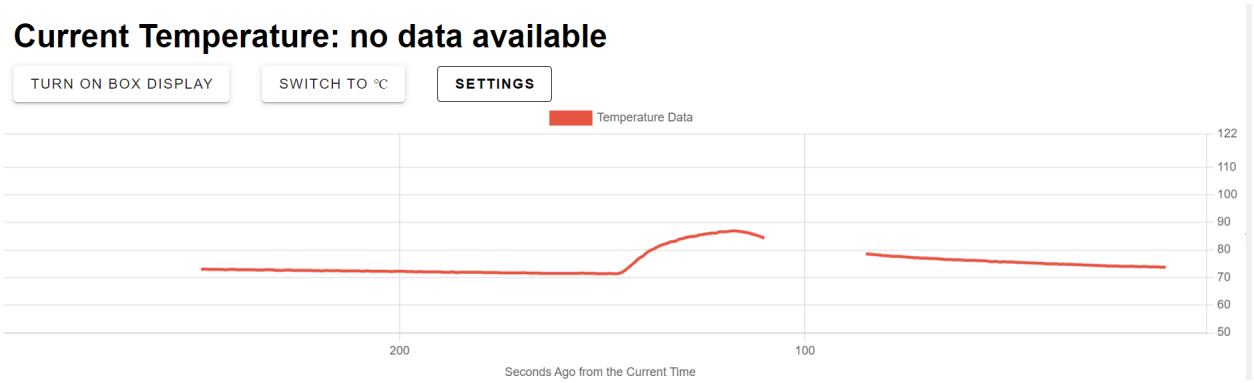


Figure 6: Web application home screen after the switch is flipped on the third box

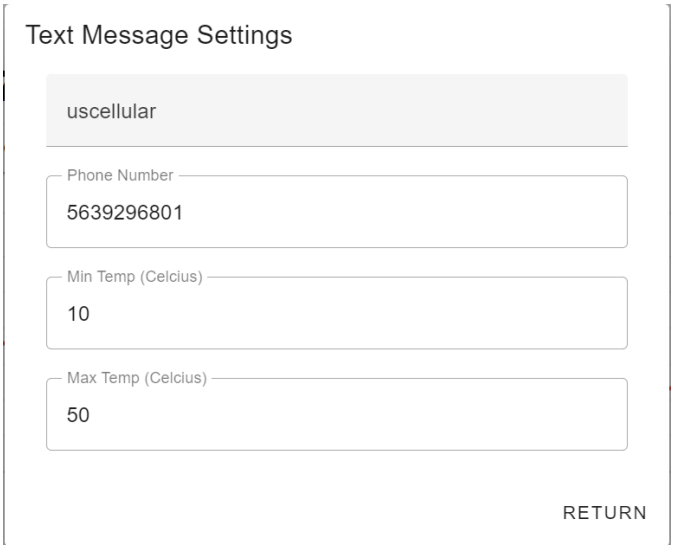


Figure 7: Web application modal after the "Settings" button is clicked

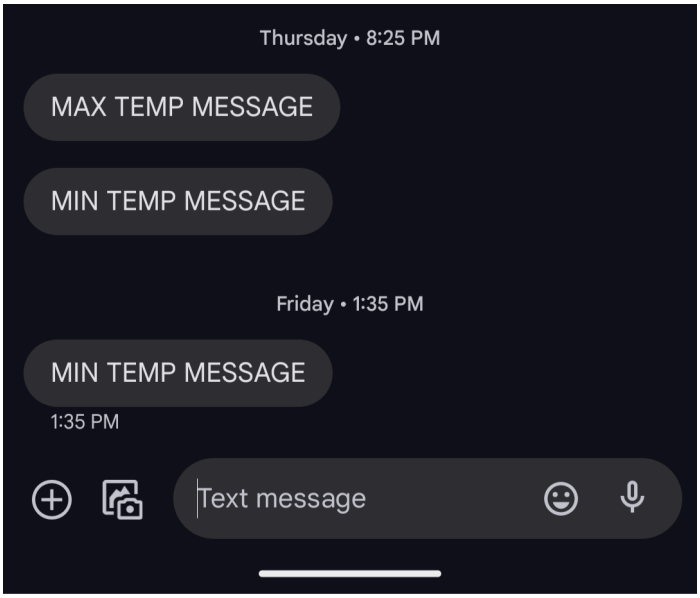


Figure 8: Screenshot of the device receiving the text message notification from the server

Server Overview: All classes, functions, and variables are well documented in code

The server is started by executing the main.py file which initially creates the main pi object and then creates and starts the 3 main threads. When the Pi object is created it initializes all variables and sets the GPIO settings. This enables the button and switch interrupts which are constantly waiting for pin changes. The main thread runs the Flask API app which is continuously waiting for requests. The two other threads are the temperature loop thread and the LCD loop thread.

The temperature thread runs a constant loop that grabs the current temperature every second and adds it to the pi objects temp_data array. If no data is available because the sensor is unplugged a null value is appended. Also if the current temp is outside the min and max temp bounds a text message alert is sent. (temp loop is in Server/thermometer.py file)

The LCD thread runs a constant loop that checks the status of the switch, physical button, and computer button. Depending on the status the LCD is turned on or off and displays

the correct information. (LCD loop is in the Server/main.py file and accesses Server/lcd.py class and functions)

The Flask API server has the following endpoints (located in the Server/main.py file):

POST /button/<status>

Set button_status_comp to input status

Input:

- status: True or False

Response:

- 200: Button status set to True/False
- 400: Invalid Input Status

POST /settings

Set the Pi's phone number, carrier, max_temp, and min_temp

Inputs:

- request body: {
  "phone_number": <phone_number>,
  "carrier": <carrier>,
  "max_temp": <max_temp>,
  "min_temp": <min_temp>
  }

Response:

- 200: Success
- 400: Input min_temp > max_temp, Invalid Carrier, Invalid Phone Number, Missing Input Parameters

GET /settings

Get the Pi's phone number, carrier, max_temp, and min_temp

Response:

- 200: Success

GET /temp

Get the Pi's temperature data

Response:

- 200: Success
- 403: Switch is off

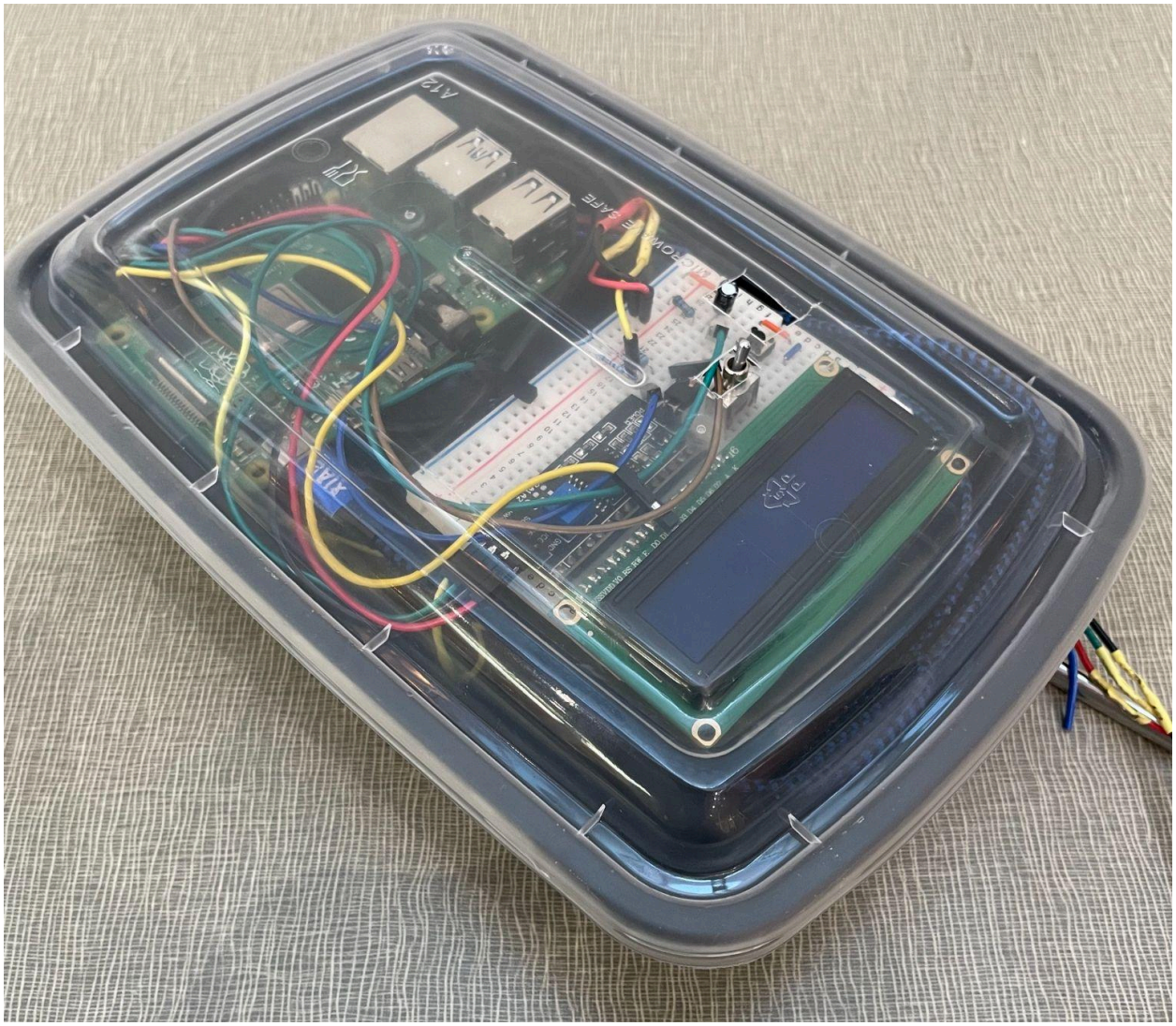


Figure 9: Picture of the final device

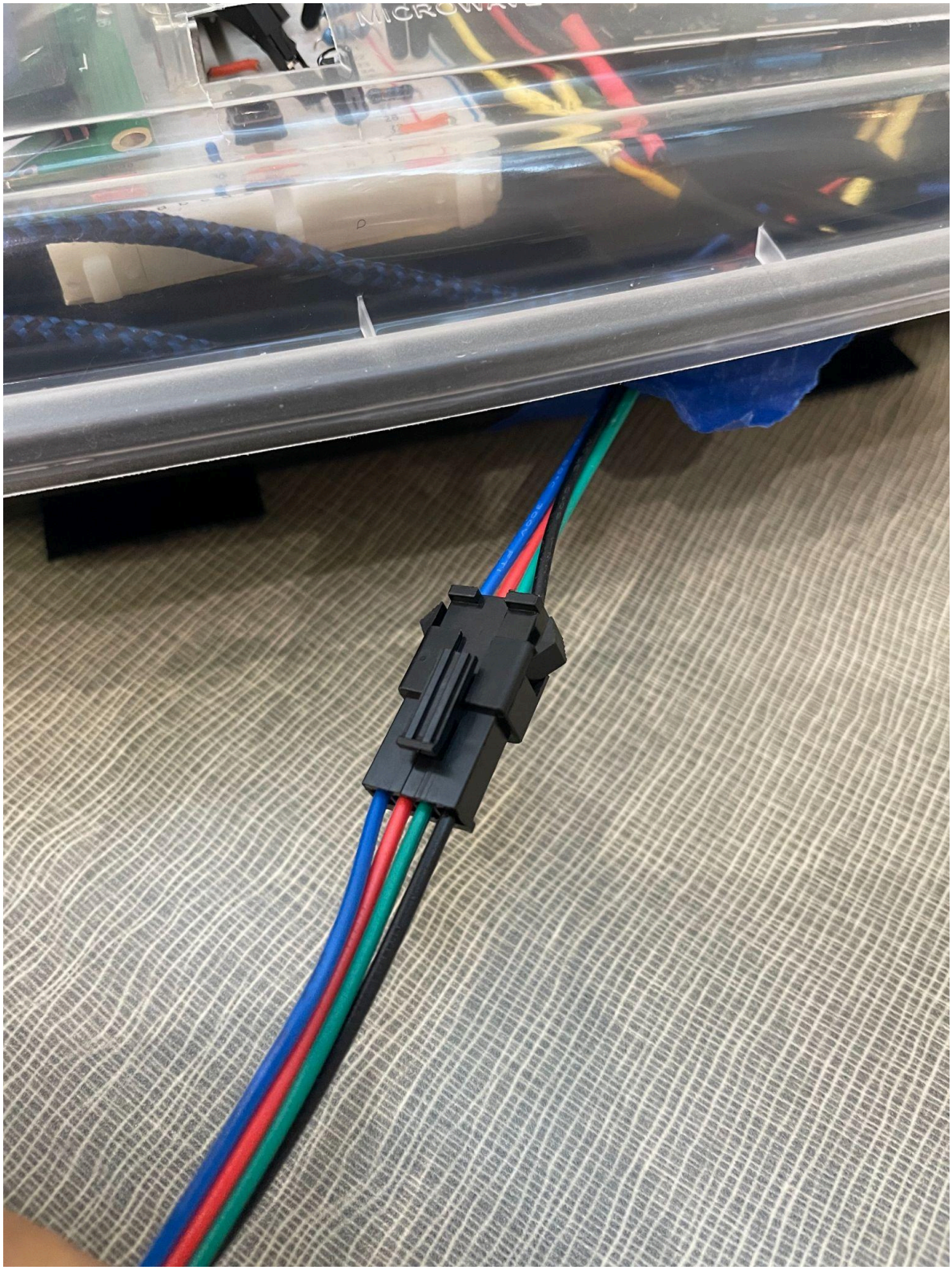


Figure 10: Picture of the Male/Female Connector used for the Thermometer

Figure 11: Picture showing the LCD being on with the web app button being pressed

Figure 12: Picture showing the LCD being off when the push button is pressed because the toggle switch is set to off

Figure 13: Picture showing the LCD being on because the push button is pressed and the toggle switch is on

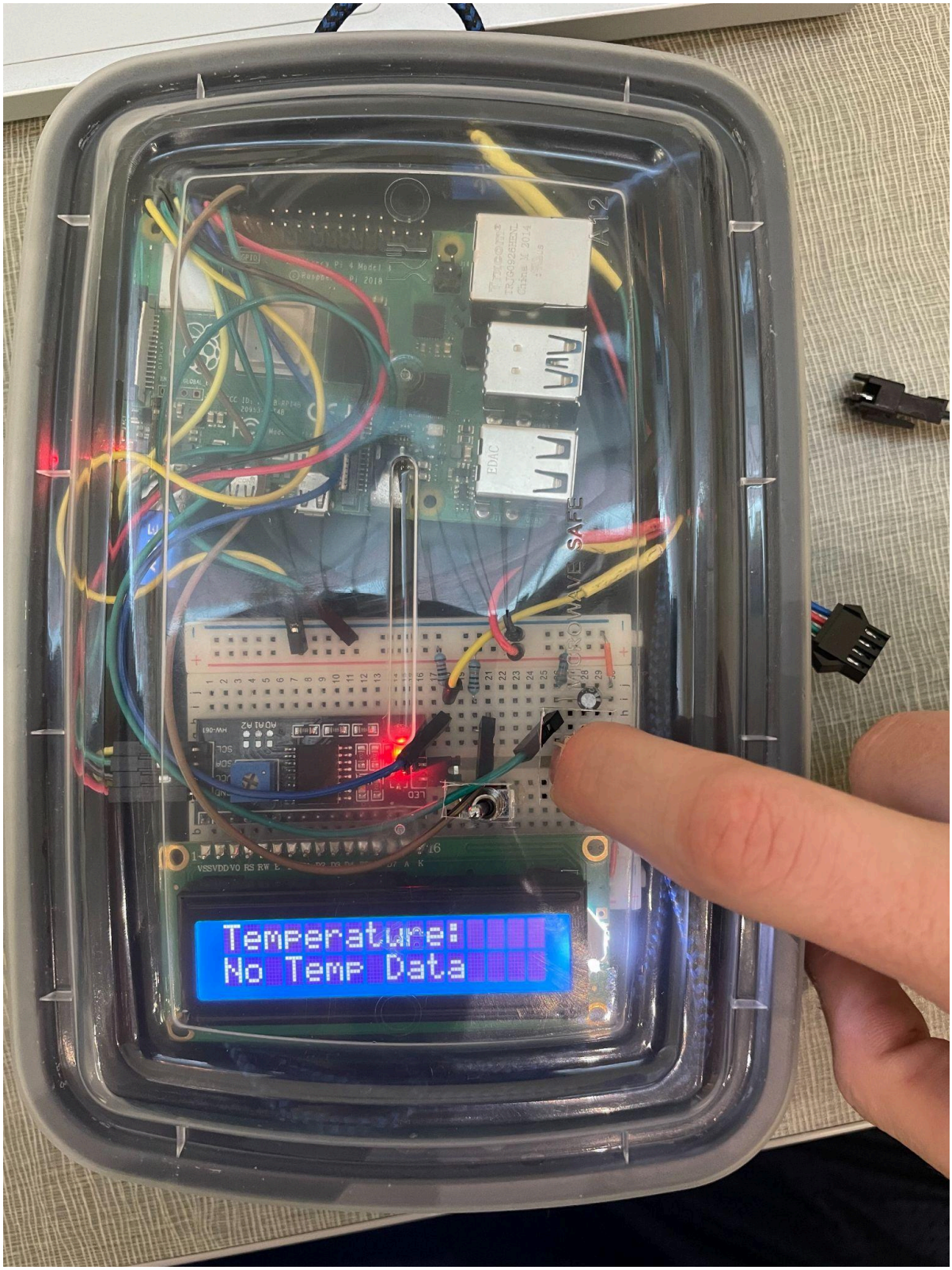


Figure 14: Picture showing the LCD displaying there is no temperature data because the thermometer is unplugged

Design Process and Experimentation

Server-Side Design:

The server-side design considerations we needed to figure out were:

Temperature collection/sensor

Microcontroller or computer to facilitate collecting and storing values

Internet enabled device to send data to the client side

Push button, toggle switch, and a display

Web based communication

3D Box or Tupperware

Temperature collection/sensor:

Solution	Implemented?	Reason
Make our own thermocouple from wire	No	More work than buying an already put-together solution. Would need to use the microcontroller to manually collect data. Would need to make the thermocouple waterproof
DS18B20 Temperature Sensor	Yes	Available at the CoE electronics shop. Well packaged and put together. Waterproof. Has built-in one-wire communication capability for easy temperature reading

Device to facilitate collecting and keeping values:

Solution	Implemented?	Reason
Arduino ATMEGA328P	No	Have familiarity collecting data from Embedded systems. Would require sending data with Bluetooth and a Pi to connect to the internet.
Raspberry Pi	Yes	The Pi has WiFi capabilities which allows for API calls and simple interfacing with the display computer. In addition, we had familiarity with this type of interfacing already because of our Internet of Things class.

Internet-enabled device to send data to the client side:

Solution	Implemented?	Reason
Normal computer	No	Would be easy to work with. Could communicate over Bluetooth with a microcontroller. Wouldn't be able to be included in the box itself as it is large too.
Raspberry Pi	Yes	Smaller option. Easy to fit inside the box itself so no outside device is needed to communicate with the server. Also has GPIO pins for interfacing with the hardware. Good all-in-one unit to interface with hardware as well as communicate with client-side

Push button:

Solution	Implemented?	Reason
CUI Devices TS02-66-50-BK-100-SCR-D	Yes	Small and compact button. Already available from our embedded kits. Easy to work with as we already have experience with it.
Other button designs	No	We looked at a few other push button designs. A larger button could've been easier to work with for the user as our button is pretty small. Didn't choose anything else because we already had the other push button

LCD:

Solution	Implemented?	Reason
AMC1602AR-B-B6WTDW-I2C	Yes	Easy to interface with the Pi. Also has an extra "backpack" to limit the number of IO pins. Simple LCD. All team members were already familiar with it as we all used it in embedded systems class. Free since we already had it
Waveshare 2-inch IPS LCD Module 240×320 Resolution Display for Raspberry Pi	No	Better screen quality and can display different colors. Much more complicated to work with and the library we found was written entirely in Arduino code, but we wanted to work in Python

Web communication protocol

Solution	Implemented?	Reason
FastAPI	No	We tried the FastAPI python library first and ran into errors when trying to multithread our application. We needed to be able to create threads to run the temperature loop and the LCD loop while also having the API server up at all times. We were going to try to split up the work and have two separate servers running at the same time, one for the APIs and one for the rest of the pi's work. This didn't work however because the API endpoints needed to be able to access the pi object's data so we needed them to be run on the same server.
Flask API	Yes	Flask solved all the problems we had with FastAPI and allowed us to host our API server on the main thread of the application while also letting us create threads for the other loops.

Box:

Solution	Implemented?	Reason
3D Printed Box	No	Would be able to customize the fit to our components. Would be expensive and time-consuming as none of our members are familiar with 3D printing.
Tupperware	Yes	Would be cheap and structurally sound. Provides the ability to have multiple prototypes because of the widespread availability of different Tupperware container sizes and types.

<u>Client Side Design:</u>

The client-side design considerations were:

Protocol for web-based communication with server-side

Framework for building web app

Communication Protocol

Solution	Implemented?	Reason
HTTP Requests	Yes	Asynchronous requests allow us to efficiently and effectively grab and post data.

Packet Sending / Hard Connection	No	In a web application environment packet passing is more complicated and more difficult to develop for

Framework for web app

Solution	Implemented?	Reason
Web Application	Yes	Software as a service model allows many clients to be connected to the Pi at the same time. Easier to deploy and spread. More flexibility for developing apps (Javascript libraries such as Chartjs allow us to make graphs easier as an example)
Local Application	No	Using C#.Net or Java Scenebuilder would work, but we don't have as much experience making a full application with it vs. Javascript. Local applications could also run differently depending on the operating system, and environment setup.

Test Report

Software Tests - test_api.py and test_pi.py

Description	Results
test_pi_init: Tested the constructor of the Pi class. Tested that all class member variables are set.	Pass
test_button_interrupt: Tested the function called when the button is pressed. Mocked the button pin value and then tested that button_status_phys is set correctly	Pass
test_switch_interrupt: Tested the function called when the switch is flipped. Mocked the switch pin value and then tested that switch_status is set correctly	Pass
test_pi_lcd_loop: Tested the lcd_loop function. Mocked the pi values for the switch, button Phys, and button comp statuses. Using a combination of different values for those statuses as well as giving the pi fake temp data the lcd_loop function was called. lcd.message and lcd.clear functions were tested to make sure they were called the correct amount of times and used the correct input values from the mocked values.	Pass
test_post_button_true:	Pass

Tested POST /button/True API call returns the correct response	
test_post_button_false: Tested POST /button/False API call returns the correct response	Pass
test_post_button_invalid: Tested POST /button/<status> API call with invalid inputs returns the correct error response	Pass
test_post_settings_valid: Tested POST /settings API call with multiple combinations of valid input data expecting success response	Pass
test_post_settings_invalid_inputs: Tested POST /settings API call with invalid input data expecting correct error response	Pass
test_post_settings_invalid_request: Tested POST /settings API call with missing input data expecting correct error response	Pass
test_get_settings: Tested GET /settings API call by first mocking the values then making the GET request and validating the response values match	Pass
test_get_temp: Tested GET /temp API call by first mocking the temp values then making the GET request and validating the response values match	Pass
test_get_temp_switch_off: Tested GET /temp API call doesn't returns the correct error code when the switch is off	Pass
test_return_response: Tested return_response function returns correct output for various inputs	Pass
test_return_error: Tested return_error function returns correct output for various inputs	Pass

Client User Tests

On startup, the real-time temperature is displayed in large font	Pass
When the temperature sensor is unplugged, an "unplugged sensor" message appears instead of the real time temperature	Pass
If the client cannot receive data from the server, "no data available" appears instead of the real-time temperature	Pass

The temperature display on the box can be turned on or turned off	Pass
When the computer is connected to the internet, and the switch on the third box is on, a graph of the past temperature readings from the third box can be displayed on the computer screen	Pass
The graph is the temperature in degrees C/F and there is a mechanism for users to switch between degrees C and degrees F. The top of the graph corresponds to 50 degrees C(122 degrees F), and the bottom, 10 degrees C(50 degrees F)	Pass
The physical size of the graph should be scalable with the mouse	Pass
If the third box is off or the temperature sensor is not plugged in, the graph continues to scroll and the graph data should be shown as missing	Pass
The user can change text message settings such as the phone number the text gets sent to, the min and max temp bounds, and the carrier the user is on	Pass

Hardware tests

The button itself that there is no continuity when unpressed	Pass
The button itself that there is continuity when pressed	Pass
The push button pulls the node low when pressed	Pass
The node is high when the button is not pressed	Pass
Raspberry Pi pin reads low when the button is pressed	Pass
Raspberry Pi pin reads high when the button is unpressed	Pass
Interrupt triggered when button pressed	Pass
Interrupt triggered when the button is released	Pass
Interrupt triggered only once when pressed and when released	Pass
Switch has no continuity when in off position	Pass
Switch has continuity when in on position	Pass
Switch node is high when switch is in the off position	Pass
Switch node is low when switch is in the on position	Pass
Raspberry pi pin reads high when switch is in on position	Pass
Raspberry pi pin reads low when switch is in off position	Pass

Interrupt triggered when switch is flipped	Pass
Interrupt triggered only once when the switch is flipped	Pass
Display “Hello World” to LCD	Pass
Clear the LCD	Pass
Turn the LCD backlight on	Pass
Turn the LCD backlight off	Pass

Project Retrospective

In lab 1 we successfully completed and fulfilled all requirements given to us while staying on time and on budget. We played to our strengths, picking languages we were already familiar with and that worked well for the type of project we were planning (Python backend, Vue.js frontend). We split the project up as follows:

Samuel Nicklaus - Front-end web design, HTTP requests

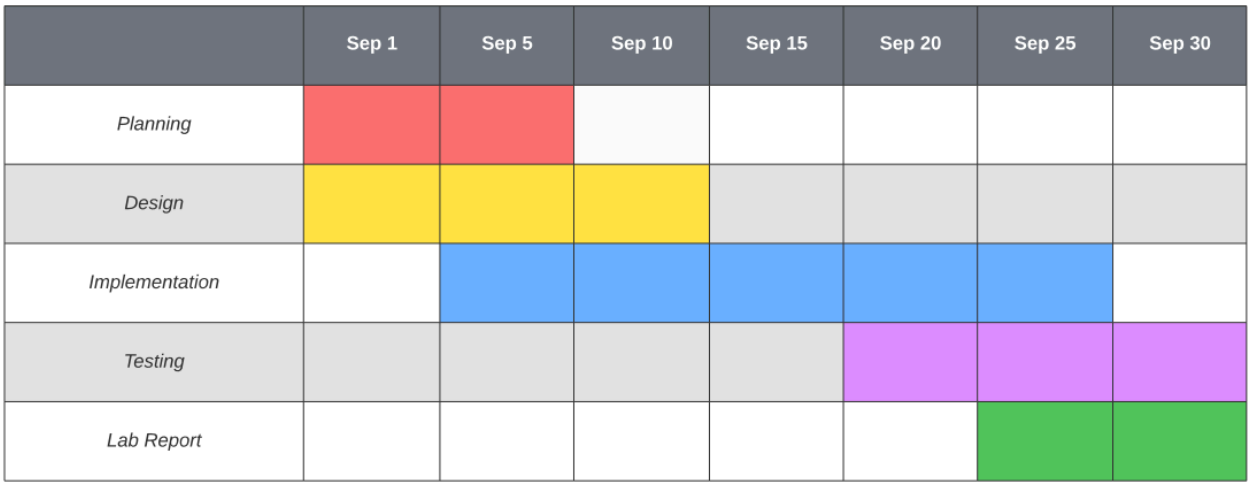
Cole Arduser - Back-end Server (APIs and Pi implementation) and unit testing for both

Luke Farmer - Circuit design, implementing LCD code, text messaging library sourcing, and script to start the main program on Pi boot

Sam Loecke - Circuit design, server code for button and switch interrupts, soldering for temperature probe and connector

Every individual contributed about equal with each person playing to their strengths. We implemented a Waterfall methodology as we were already given strict requirements and guidelines. This helped a lot as we started our project by having a long meeting going over how we wanted to design the client and the server. What we were most concerned with was the communication and the types of data that are passed between them (We eventually settled on Flask, setting up strict guidelines for the data types passed in and out).

One thing our group delayed was designing the box. When considering where improvements could be made, the box design is definitely more hacked together than what we originally planned. Our final product is made by cutting a Tupperware box to access the switches and buttons, but we wanted to custom 3D print the chassis in our original plan. The reason we opted for our current solution was due to time and monetary constraints. Another design decision was choosing the Raspberry Pi over the microcontroller. This in turn helped our software development a ton due to the Pi having more documentation and available libraries for what we need. Also having the pi have wifi enabled us to use Flask to have API requests straight from the Pi. We did not have any additional factors that contributed to a less-than-complete success, I believe we found the most optimal way to approach the lab and made good compromises where needed (in the case of the Tupperware housing).



For the most part, we were able to stick to our plan and timeline. One problem we encountered was the inability to SSH into the Raspberry Pi on UI-DeviceNet. To work around this, we had to purchase a USB-Ethernet connector which we didn't account for in our budget. In addition, it increased the time needed to test functionalities as they were being implemented because only one team member could be connected to the Pi at a time. Apart from that, we stuck to our timeline. The planning time and meetings were the reason we were able to stay on track.

Appendix & References

Full source code can be found HERE

HD44780U “Dot Matrix Liquid Crystal Display Controller/Driver” Hitachi, 1998

DS18B20 “Programmable Resolution 1-Wire Digital Thermometer” Maxim Integrated, 2019

Raspberry Pi 4 documentation

https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/, 2023

Python Raspberry Pi GPIO library documentation https://pypi.org/project/RPi.GPIO/, 2022

Flask Python Module https://flask.palletsprojects.com/en/3.0.x/

Python Unittest framework https://docs.python.org/3/library/unittest.html

Pytest Python framework https://docs.pytest.org/en/7.4.x/

Vue.js documentation https://vuejs.org/, 2014

Chartjs documentation https://www.chartjs.org/, NA

Vue chartjs documentation https://vue-chartjs.org/, 2018

Vuetify component framework documentation https://vuetifyjs.com/en/, 2016

NPM package manager https://www.npmjs.com/, NA

Vite developer environment https://vitejs.dev/, 2019

LCD Installation and Code: https://github.com/the-raspberry-pi-guy/lcd, 2023