Steven A.
Zack M.
Sage M.
Matthew K.

Team 3
29 September 2025

Lab 1: Smart Thermometer

I. Introduction

ECE:4880 – Senior Design 1 at the University of Iowa tasked our team with creating a robust smart thermometer that has internet connectivity and user-friendly functionality. Our team, made up of engineers with a wide range of skillsets, was up to the challenge. Our solution provides an easy way for users to view temperature data physically on our box or remotely from a computer. There are multiple input controls used within the system, such as switches and buttons, along with cable connectors to remove sensors when there is no need for use. The user receives feedback via an OLED screen, and virtually all data is uploaded to the web. The future of smart or internet connected devices is very bright, and it will be intriguing to see what devices like what was developed in this lab evolve.

II. Design Documentation

a. Hardware Schematic

The overall thermometer system, from a hardware perspective, is rather simplistic. We utilized the Raspberry Pi 4 Model B's GPIO pins to communicate with peripherals. We decided to have our entire system operate at a voltage level of 3.3V. This way, there is no miscommunication between our peripherals and the Raspberry Pi, whose GPIO pins read 3.3V as high. The two temperature sensors use a 1-wire protocol and need a 4.7kΩ pull-up resistor to voltage for correct communication. For button interaction, a simple hardware debounce circuit was implemented for more consistent readings; the buttons were connected to the GPIO pins as well.

We power the Pi with a 5V lithium-ion battery that can be unplugged from the Pi when not in use. For our reading and display switch, we used an RC circuit to ensure a smooth switching signal is sent to the Pi. The circuit can read either ground or voltage, and, depending on the reading, we determine if temperature readings should be taken and displayed. A current-limiting resistor was also placed between one of the switch pins and 3.3V. Finally, we decided to use an I2C OLED as our display. This was easy to implement as I2C is a two-wire protocol; after adding two pull-up resistors on the data lines, communication was straightforward. Our full hardware schematic is shown below in Figure 1.

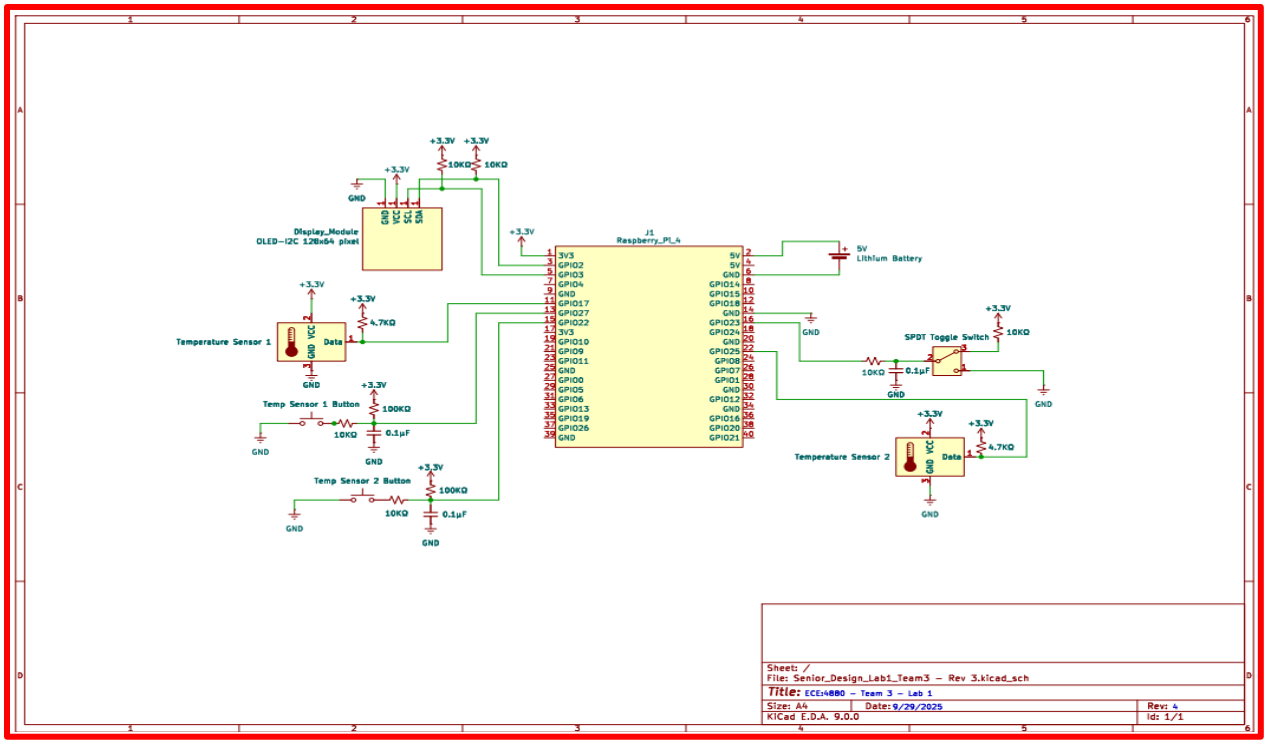

Figure 1. Smart Thermometer Hardware Schematic, Rev. 4

b. Hardware Material List

To accompany the hardware schematic, an exhaustive hardware material list has been constructed so that anyone wishing to recreate the lab can do so without having to do much digging. The list of parts can be found below in Figure 2.

Hardware	Quantity	Description
Raspberry Pi 4 Model B	1	Single Board Computer
DS18B20	2	1-Wire Digital Temperature Sensor
Enable Low Push Button	2	Temperature Sensor Toggling
Single Pole Toggle Switch	1	Toggling Temperature Reading & Display
KBT 5V 4Ah Battery Pack	1	5V Lithium Ion Battery
Male 3-Pin Terminal Housing	2	Cable Connectors
Female 3-Pin Terminal Housing	2	Cable Connectors
Electrical Wire Crimp Terminal	6	Cable Connectors
128 x 64 OLED	1	I2C OLED Display
4.7kΩ Resistor	2	1-Wire Pullup Resistors
10kΩ Resistor	6	I2C Pullup Resistors & Debounce Circuit
100kΩ Resistor	2	Debounce Circuit
0.1µF Capacitor	2	Debounce Circuit

Figure 2. Hardware Material List

c. Third Box & Cable Connector Design

The design of our team's third box was all based around fitting essential components within the box. It was crucial for our design to be able to house the Raspberry Pi, breadboard, and the 5V lithium-ion battery. We decided against using PCB and soldering all connections, as staying with the breadboard would save lots of time working at the soldering iron. Below the interior of the third box is shown (Figure 3).

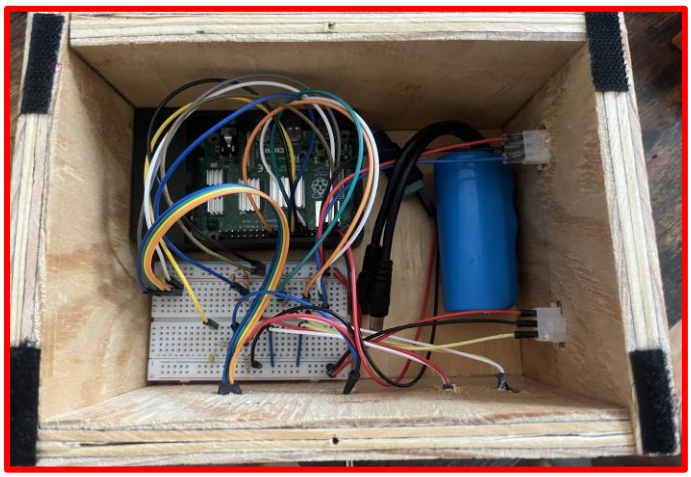

Figure 3. "Third Box" Interior View

To construct the third box, the team decided to use plywood for its physical robustness. We first measured out interior dimensions that were needed for there to be sufficient room for hardware. The interior dimensions that we decided were best fit for the application were as follows, 12.5 x 16 x 9 cm^3 with exterior dimensions of 15 x 18.5 x 10.25 cm^3. To cut the wood we used a circular saw, and then to get the specific openings needed for the cable connectors, buttons, switch, and display, a combination of a coping saw, wood files, and a drill was used. When assembling the box, multiple screws were driven into the walls and base of the box. To mount peripherals, we utilized super glue and careful measurements so that we could achieve a nice, tight fit. Below in Figure 4, the front face of the box is shown.

Figure 4. "Third Box" Front Face

For easy access to the contents of the box and to plug and unplug the battery, Velcro was attached to both the top of the walls of the box and the lid. This created a

clean and easy way for one to open the box and close it again without having to undo any screws or use any tools.

Another part of the box that required careful design considerations was the cable connectors. It was crucial our cable connectors were user friendly but yet secure enough so temperature sensors can get accurate measurements, even with the box being upside down. These requirements led us to make the decision to use connectors we are familiar with, something that is already used in cars!

We settled on using 3-pin electrical wire housings. To assemble these connectors, we first had to crimp wire terminals onto the end of the three wires we wanted to use for both male and female ends. After crimping the terminals onto the wires, all that had to be done was slide the terminals into the housing. Now, by pushing down on a tab on the top of the female connectors and inserting the male end, we have a solid connection. These connectors can also be easily unattached by once again utilizing the tab on the top of the female end. This provides a solid connection that gives accurate measurements even when moving the temperature sensors around. In Figure 5 below, an image of both the male and female ends of the connectors is shown.

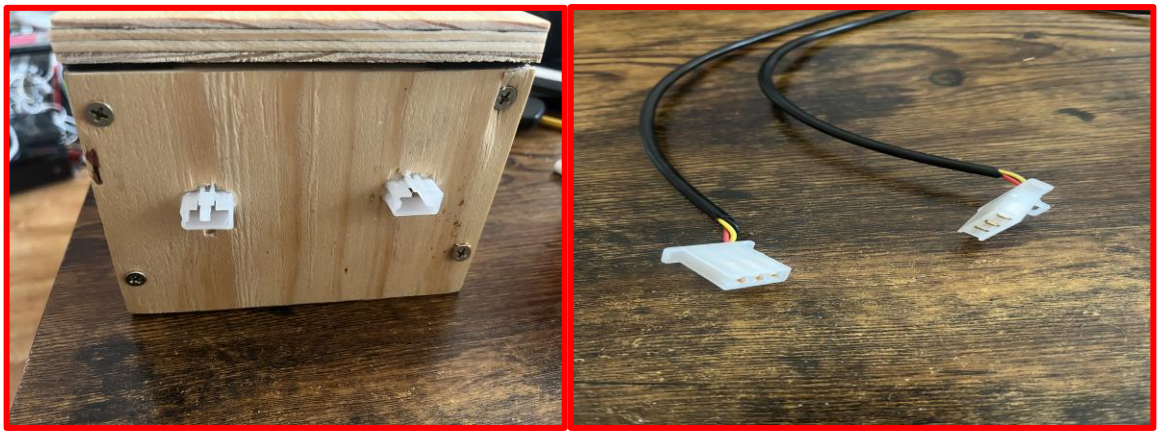

Figure 5. Male & Female Cable Connectors

d. Embedded Software

The embedded code on the Pi was written in C++ to allow for direct manipulation of GPIO pins and interrupts. The program utilized built-in timing libraries to get a temperature reading once per second. The program communicated with the sensors via 1-Wire communication protocol. Once the temperature was read, the value would be output to the OLED in the proper spot (sensor 1 or sensor 2). If the sensor was toggled off, no reading would be made, and the program would print “OFF” instead. If the sensor was unplugged, reading the temperature would throw an exception. Catching an exception was the program’s way of detecting that the sensors were unplugged.

The buttons to toggle the sensors were connected to falling-edge interrupts. We had to import a library in order to use hardware interrupts on the Pi. The interrupt service

routines would toggle a Boolean variable, which would then be polled on every iteration of the main loop.

After reading the temperatures (or null values, in the case of off/unplugged), the program would send an HTTP POST request to the server, which is also being run from the Pi. The content of the POST request is:

- Temperature value from sensor 1
- Temperature value from sensor 2
- Enabled status of sensor 1
- Enabled status of sensor 2
- Unplugged status of sensor 1
- Unplugged status of sensor 2

Once this data was sent, the server would process it and send back a three-element array. The elements are the following:

1. The current unit as determined by the user (either “C” or “F”)
2. Whether the user has toggled sensor 1
3. Whether the user has toggled sensor 2

Based on the status of the user’s toggling, the program would update the status of the sensors accordingly.

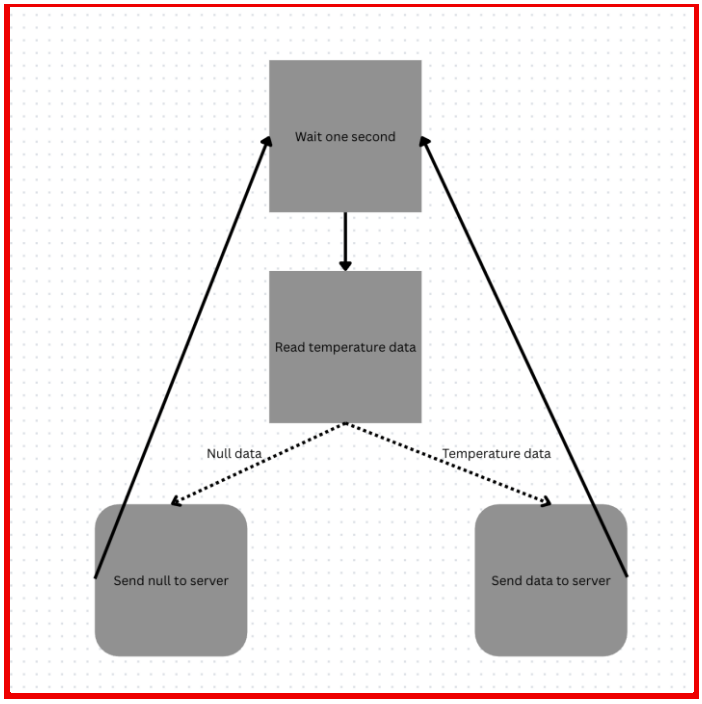

Figure 6. Software Flowchart

e. Server and Dashboard

The server was run using Dash and Flask in Python. Components for toggling the buttons and the graph were set up for the user to interact with. A POST request listener was set up to listen for data from the embedded program. When triggered, the server would first perform necessary "None" checks to check for no data. It then uses the data to update the "last 300 seconds" DataFrame, which plots the data on the dashboard.

The method of updating the DataFrame was time sensitive. The simplest solution would be to move all values of an array and add one at the end. However, this operation would take linear time in the size of the array (which is 301 in this case). Instead, we used the DataFrame columns as a queue. This is done by simply appending to the end of the lists, then shifting everything down by one, effectively acting as a queue. Enqueuing and dequeuing take constant time, which effectively speeds up the operations by a factor of 301.

The server looks at the new data coming in and compares it with the minimum and maximum values set by the user. If greater than the maximum, or less than the minimum, it will send out an email to the address the user puts in. More about emailing will be discussed later.

In the settings page, the user can enter their email address, minimum value, and maximum value. These values are stored as class variables, which are accessed by the server app when sending out emails.

The emails are sent out from the University of Iowa's ns-mx.uiowa.edu server, which is a lightweight email server that requires little verification. This was perfect for the lab because we did not want to go through the hassle of authentication on a full email server. The only requirements needed to use this server are:

- The user must be connected to the University network (e.g., eduroam).
- The receiver must be a uiowa.edu address

Given these two requirements, sending the emails was straightforward.

III. Design Process & Experimentation

a. Hardware

Throughout the hardware design process, many decisions needed to be made based on testing. Many of these decisions affected the hardware we wanted to use. A large majority of the hardware was provided for this lab which made the decision process easy. We decided to use the simple pushbuttons provided in our Embedded Systems kits, as we were familiar with those and how to make their respective debounce circuits.

The one area that gave us uncertainty was deciding which display we wished to implement to show sensor data. Our original plan for the display was to use the same LCD that could be found in the embedded systems kits. However, after doing some digging, we discovered that the LCD operated at a 5V logic level while the Raspberry Pi 4's GPIO pins utilize a 3.3V logic level. This realization left us with a couple of options: use a bidirectional logic level converter or a different display. Since the LCD has four data pins, using a logic level converter could get messy and take up more room on our

breadboard. These factors led us to search for a new display; we quickly discovered an OLED that suited our needs. This OLED uses the I2C communication protocol, which is a simplistic two-wire protocol and also operated at 3.3V. This decision saved us a good amount of time in both wiring and programming.

b. "Third Box"

Our "Third Box" design went through multiple iterations. The original plan was to 3D model and print a box that would be structurally sound enough to withstand the drop from the workbench. The main constraint encountered was the ability to fit the Raspberry Pi 4 Model B, a small breadboard, and our 5V lithium-ion battery within the box. This posed an issue as the dimensions of the box proved to be relatively large for 3D printing. After modeling the box, we discovered that a print would take around a day and a half. This discovery led us to create our final third box design out of wood. This felt like a better solution as wood is sturdier than 3D printed material, and we would not have to run a printer for multiple days. The wood gave us the opportunity to make changes on the fly by using a drill, coping saw, and wood files to create openings of various sizes for our display, switch, buttons, and cable connectors. The original 3D model is pictured below in Figure 6. The dimensions of our final box were very similar to the 3D model.

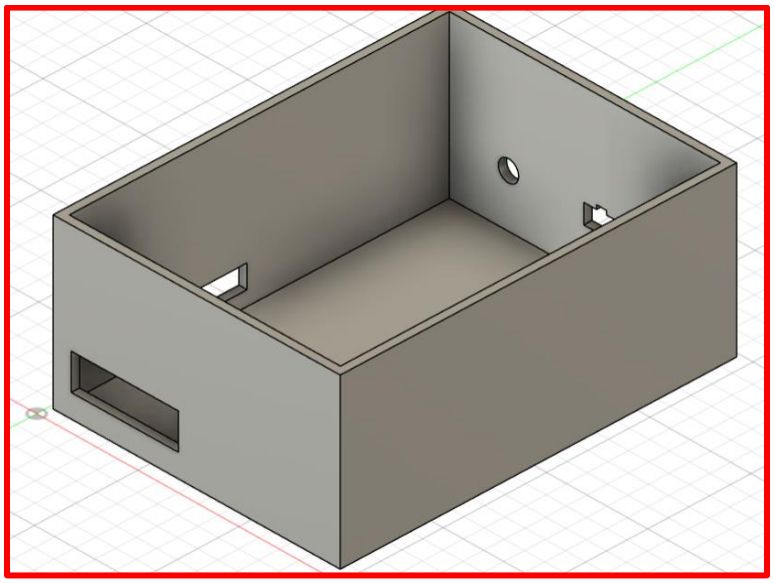

Figure 7. Initial "Third Box" 3D Model

The overall cost of wood material was around $10, which was more than that of 3D printing our box, but this investment proved to save time and helped improve the stability of our final product.

c. Server & Dashboard

Our dashboard was built with ease-of-use in mind. Its design was minimal and effective. There were two cards used for toggling the buttons, simple text boxes used for

displaying the temperature, and a graph that updates every second. On the settings screen, there were three text boxes for setting the email, maximum temperature threshold, and minimum temperature threshold.

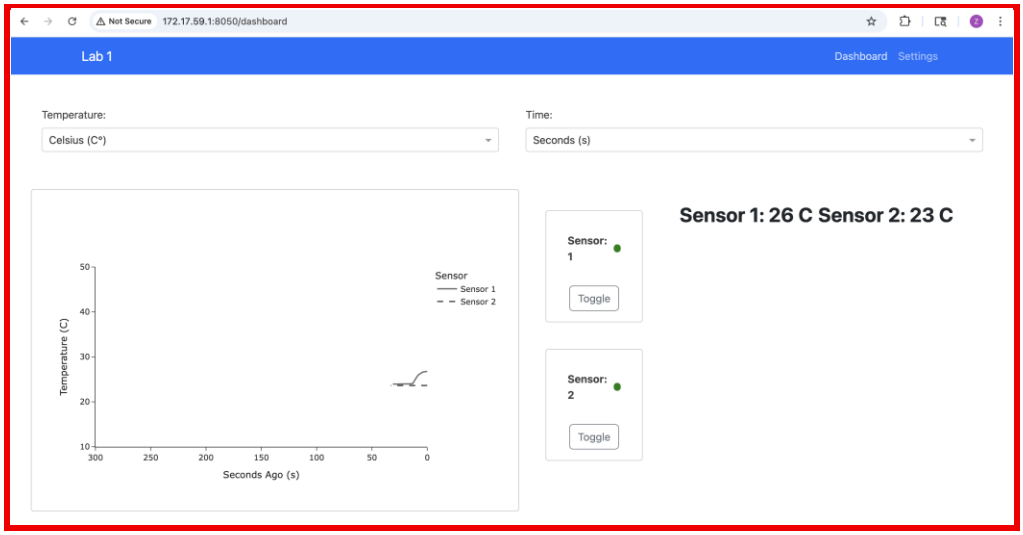

Figure 8. Online Thermometer Dashboard

To move past a prototype and build a more robust system, a second application was created with support for further iterations of the project in mind. Having already completed the base requirements, this application was purely experimental and aimed at how we could improve the system.

Keeping the same embedded system, this software portion of application was modified to consist of five components: a dedicated entry point for streaming data (1) into a Redis cache (2), a PostgreSQL database to store temperature readings and user information (3), and an asynchronous task queue to handle processes with high overhead – such as database calls and SMTP - in the background (4), and a web application that serves a dashboard designed for mobile devices (5). The components were containerized using Docker to for ease of development, deployment, and scalability.

With this extensible foundation, any smart-home application is straightforward and highly configurable; Adding an additional sensor (thermometer, humidity, air quality, etc) just means adding new endpoints to handle sensor information within the Streamer API service, creating a new stream, and creating a new table within the database. The additional sensor information can then be used for analytics and real-time operations in response to external sensor readings.

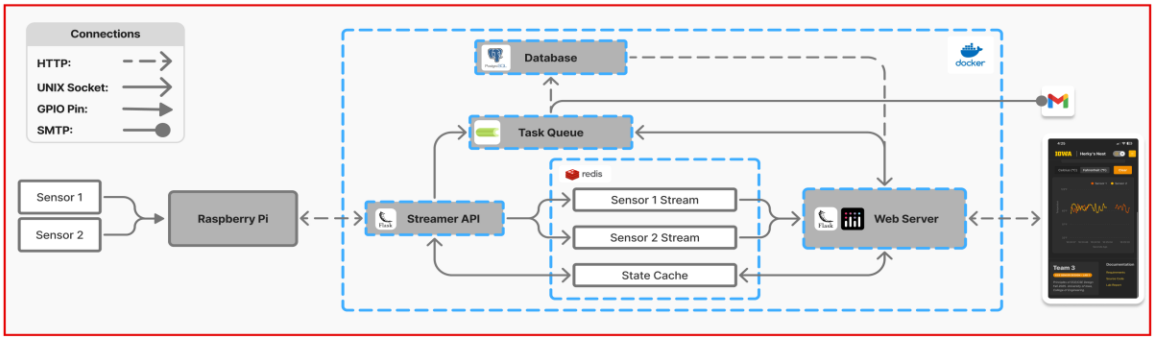

Figure 9. Extended Prototype System Architecture

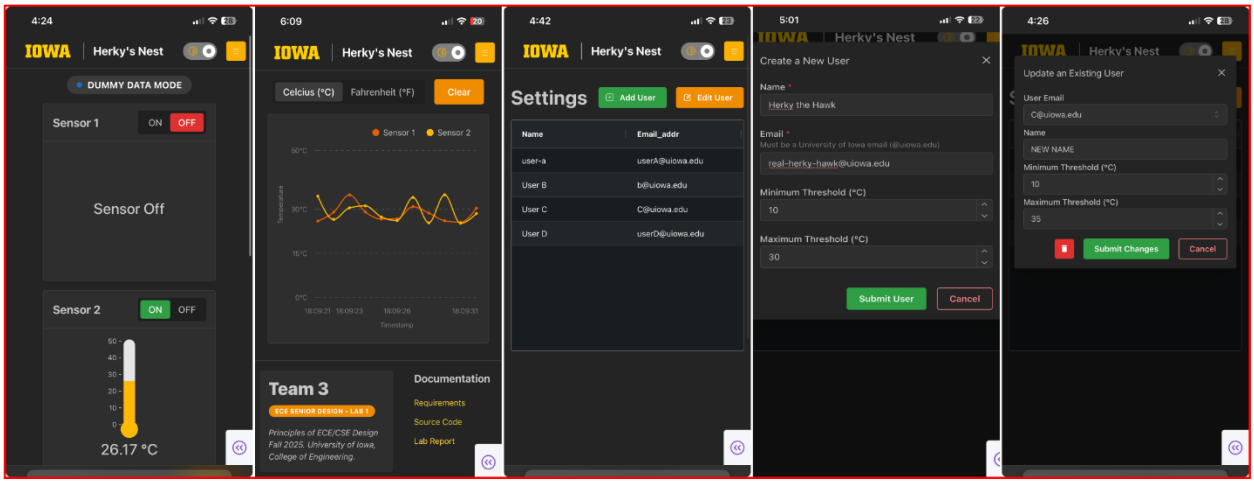

Figure 10. Extended Prototype Application UI

Comparing the two applications showed that both performed the same as the refresh rates were both set to 1Hz per the requirements. Ultimately, we opted not to use the experimental application for checkoff as the original application completed the base requirements.

d. Embedded Code

C++ was chosen for the language. Other languages such as Python would not give us as direct control of the hardware as we desired. Still, hardware interrupts are not immediately native to the Raspberry Pi, so we had to import an external library that gave us access to the GPIO interrupts.

The timing of the OLED update was crucial. Updating it inside of the interrupt was impractical, because it could be mid-update when the interrupt is called, which would cause the OLED to glitch out. Therefore, we just polled the Boolean inside of the main program loop. To satisfy the 20 ms button response time requirement, we had to use special libraries to make sure all HTTP requests and OLED updates could be performed within that interval.

Even though the server and embedded program were both run from the Raspberry Pi, we still used a POST request to transfer the data instead of in-device transfer options. This introduced unnecessary overhead for the lab, because the data had to be sent through the WiFi network just to end up back in the same device. However, we wanted to generalize this lab. It would be straightforward for us to change the server to be a different computer. All we would need to do is change the IP address in the source code of the embedded program.

IV. Test Report

To document and ensure all lab requirements were satisfied, we created a test report. This report addresses specific requirements and outlines the passing criteria. If a test is passed, that is reported in the pass/fail box, with specific values if those are needed. Having a solid test report with well-designed tests is paramount to meeting requirements and having a properly functioning final product.

Test Step	Requirement Addressed	Passing Criteria	Pass/Fail	Value	Date
Hardware Tests					
1. Cable length	The two thermometer sensors must be at the end of a cable at least 0.95 to 1m in length.	The length of the cables exceeds 0.95 m.	Pass	0.95 m & 1.0 m	9/30
2. Third Box Mechanical Construction	The third box must be enclosed & physically robust and can be turned upside down while remaining operational.	Temperature measurements can still be taken and uploaded to web app when box is upside down.	Pass	-	9/30
3. Cable connectors	Connections to the third box are securely mounted & can be easily connected/disconnected by the casual user.	Cables can be unplugged in under 5 seconds.	Pass	-	9/30
4. Third Box Drop	When dropped to the floor, cables and connectors should not break.	After a drop from a desk in the classroom connectors are still operational, can take temperature measurements.	Pass	-	10/2
5. Sensor Unplugging	The box is still operational when the sensor is unplugged and plugged back in.	Display shows when sensor is unplugged and uploads null data. When the sensor is plugged back in the data is once again displayed and uploaded.	Pass	-	9/30
6. On/Off Switch	Switch on the third box operates as an on and off switch. When off the system temperature data is not available via the internet and no data is displayed.	With the switch off, no data is displayed on OLED & no data (blank) is being uploaded to the internet.	Pass	-	10/1
7. Toggle Buttons On	When the box is on, and both sensors are on (buttons) two temperatures are displayed in degrees C.	The display shows two temperatures, one from each sensor when the buttons are pressed to turn sensors on.	Pass	-	10/1

8. Toggle Buttons Off	When the box is on, and the buttons are toggled to off, the screen should display which sensors are off and not show any data.	Display shows sensors as off after buttons are toggled.	Pass	-	10/1
9. Button Delay	The correct temperature should be displayed when the button is pressed with no noticeable delay.	When the button is pressed for either sensor the corresponding temperature is shown in degrees C with no noticeable delay.	Pass	-	10/1
10. Physical Display	Display is legible under normal lighting conditions and temperatures are within the normal range of operation.	The OLED display is legible in the lab room. Temperatures do not exceed 63 C and do not fall below – 10 C.	Pass	-	10/1
11. Button Combinations	Buttons can be both on, both off, or one on and one off.	Buttons can be in a combination of states, and the display is updated accordingly with OFF or the corresponding sensor temperature	Pass	-	10/1
12. Unplugged Sensor Error	When a sensor is unplugged, the display should notify the user that there is an error.	When a sensor is unplugged, the display should read the sensor number: unplugged.	Pass	-	10/1
Software Tests					
13. Real Time Temperature Display	The temperature of both sensors is displayed prominently on the screen and updated every second in degrees C or degrees F.	With sensors plugged in the temperature at both should be shown on the screen in either degrees F or degrees C and is updated once every second.	Pass	-	10/1
14. Unplugged Temperature Sensor Display	When a sensor is unplugged a “unplugged sensor” message should appear instead of the real time temperature	When a sensor is unplugged, within a second, a message notifying the user of which sensor is unplugged should replace the real time temperature.	Pass	-	10/1
15. Third Box Off Display	When the third box is off, the computer should show a no data available message instead of the real time temperatures.	After turning off the third box the real time temperatures should be replaced with no data available within a second.	Pass	-	10/1
16. Computer Button Presses	User action on the computer can toggle the buttons on and off when the third box is turned on, within one second.	When the box is on after toggling the buttons on the computer the sensors should turn on or off within one second of the press.	Pass	-	10/1
17. Graph Start Up Time	When the computer is on and the third box is on a graph of the real time temperature data should be available within 10 seconds.	After turning the box off and back on the temperature data should be available on the graph within 10 seconds.	Pass	-	10/1
18. Graph Units Changing	The graph should be able to toggle between degrees C and degrees F.	Within a dropdown menu the data can be displayed in either degrees C or degrees	Pass	-	10/1

		F. The graph data is updated accordingly and accurately.			
19. Graph Y-axis	The top of the graph corresponds to 50 degrees C, and the bottom of the graph corresponds to 10 degrees C. these are the limits for the temperature sensor readings.	The limits on the y-axis of the graph are 50 degrees C and 10 degrees C. These limits never change.	Pass	-	10/1
20. Graph Scrolling	The graph scrolls horizontally from right to left, latest temperature on the right and oldest on the left. After 300 s old values scroll off the graph.	The graph scrolls from right to left having the new values coming in on the right side and after 300 s the values are no longer available on the graph.	Pass	-	10/1
21. Graph X-axis	On the x-axis of the graph, measurements from the last 300 seconds should be shown with a label of "seconds ago".	The graph shows only the last 300 seconds on the x-axis with a label of seconds ago.	Pass	-	10/1
22. Missing data	Missing data is shown as blank and is discernable from actual data. When data is available again the graph should start up at that time.	When the third box is off, a sensor is unplugged, or a sensor is off the data on the graph is blank for that period of time. When plugging back in the graph depicts accurate temperature readings at the corresponding time.	Pass	-	10/1
23. Email or Text Notifications	When the third box is on a specified email or phone number will receive a notification when the temperature exceeds or is lower than a specified value. The message and the number or email can be altered within the computer display.	When the temperature exceeds or is lower than the value specified by the user a message is sent to the email provided by the user notifying them that the temperature fell below or exceeded the specific value.	Pass	-	10/1

Table 1. Smart Thermometer Test Report

V. Project Retrospective

a. Project Outcome

As a collective, our team is proud of the final product we developed during Lab 1. We were able to develop a robust smart thermostat that has user friendly inputs and an easy-to-use UI for the internet available data. The team was able to get most tasks done in a timely manner and felt on schedule throughout the design and testing process. Despite this there are some choices, that if given the chance to make again, we may go in a different direction.

The first of these choices was using the Raspberry Pi 4 as the main computer for our box. We were provided with an ESP32 for this lab by the department, but we decided to use the Pi as it has more features. We quickly figured out that in many ways the Pi was overkill for this project and caused us a bit of grief. It also is much more power hungry

than the ESP32 and takes up more room in the box, which all had to be accounted for in our design. The Pi ended up working out for us, however, things may have gone a little smoother if we went with the ESP32 from the jump.
Another example of this was in the decision to make the "Third Box" out of wood. This is dependent on the choice to use the Raspberry Pi 4. With the dimensions of the Pi, breadboard, and battery, the volume of our box proved to be a bit too large to print quickly. If we were to do it again, using a different microcontroller, we would have much less to fit within the box and would be able to use a 3D printed version. Another challenge with wood was finding the correct tools for the project. The labs do not have many tools to work with, so we spent a good amount of time in the hardware store or running errands to find saws and drills. This all could have been avoided if we 3D printed the box.
Other than these two choices we felt the overall design process went smoothly. As a team we will reflect on and learn from these choices so in the future we do not run into any of these problems. It serves as a good reminder that we as a team need to consider how decisions will affect other aspects of the project, even ones that may not seem related at the time. These lessons will be very useful and important to remember when it comes time to begin working on our final project.

b. Division of Work

The division of work for this project was fairly even. Sage and Steven spent the majority of their time working on hardware wiring, constructing the third box, and helping with embedded code when time permitted. Matt and Zack spent their time developing embedded code, data transfer, server code, and email notifications. This was the logical division of work as Sage and Steven are electrical and computer engineers, while Matt and Zack have a bigger focus in computer science. We feel the way work was divided for this project worked well, and we were all able to utilize our areas of expertise to create a satisfactory final product. Every group member was allowed to broaden their horizons and contribute to other aspects of the project. The plan is to employ similar strategies for the remainder of the labs and then into the senior design project.

c. Project Management

Throughout this lab we employed two project management tools, Gantt Chart and Jira. These tools operate in similar ways; you assign tasks and then define subtasks, outlining the initial timeline tasks should be completed in. We organized tasks between hardware & embedded, and software & web server. Using these project management tools made it very easy to see who was working on what and how far they had progressed. It also made it easy to remind group members if they need to get something done. We may have not stayed exactly on schedule, but the tools helped keep us grounded and proved to be an asset throughout design and testing. Below in Figure 7 is the Gantt Chart used for our project. You can see the planned schedule in gray and the actual timeline of work in green.

Figure 11. Lab 1 Gantt Chart

We used a combination of project management strategies for design and development throughout the lab. For the hardware side we utilized a waterfall approach, setting out strict requirements and goals before design began. This proved to be useful as we always were able to keep ourselves grounded and remind ourselves what we are aiming for. For the software side of things, we used more of an agile approach, working towards a goal and then reevaluating and maybe changing our target. This proved to be useful as well as new functionality or ideas were often discovered while reevaluating options. Without project management strategies and a clear goal, it would have been very difficult to complete our design and project on time while meeting all the requirements. This lab drove home the importance of planning and organization; we will take these lessons and carry them into the labs throughout the rest of the semester and into our senior design project.

VI. Appendix & References

- Source Code: https://github.com/Senior-Design-2025-2026
- Libraries:
  - WiringPi Interrupts: https://github.com/WiringPi/WiringPi
  - Fonts: https://andygock.github.io/glcd-documentation/font5x7_8h.html
  - C++ HTTP Requests: https://github.com/yhirose/cpp-httplib
  - C++ JSON Library: https://github.com/nlohmann/json
- Extension Application Source Code: https://github.com/Senior-Design-2025-2026/ECE-Senior-Design-Lab-1-EXTENSION

- Raspberry Pi:
    - Raspberry Pi (Trading) Ltd., "Raspberry Pi 4 Model B Datasheet, Release 1.1," 12 March 2024. [Online]. Available: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf.