# Laboratory #1 Report:
# Temperature Sensor with Physical and Virtual Interface, Control, and Live-time Warning Features

*ECE:4880 Principles of Electrical and Computer Engineering Design*




Team AccessDenied:

Maneesh John, Kiana Erickson, Ming He, Werner Bayas

# Table of Contents

**Design Documentation** **4**
1.1 - Overall Function 4
1.2 - Hardware Documentation 5
1.3 - Mechanical Documentation 8
1.4 - Software Documentation 10

**Design Process and Experimentation** **15**
2.1 - Design Process 15
2.2 - Hardware Design Choices 15
2.3 - Software Design Choices 17

**Test Report** **19**
3.1 - Test Specifications 19
3.2 - Test Plan and Pass Criteria 21
3.3 - Test Report 23

**Project Retrospective** **28**
4.1 - Roles of Team Members 28
4.2 - Gantt Chart 28
4.3 - Project Management 28
4.4 - Workload Distribution 29
4.5 - Timeline/Plan 29
4.6 - Critical Assessment 29

**Appendix & References** **31**
5.1 - Hardware Datasheets 31
5.2 - ESP32 Libraries 31
5.3 - Backend Development 31
5.4 - Front-End Development 31

# Design Documentation

## 1.1 - Overall Function

For this project, the goal was to implement a digital temperature measurement system that could stand alone as a “third box” while also offering certain capabilities for connection with a user’s computer and phone. From a high-level view, as shown in Fig. 1.1, there are three devices communicating with each other: a box containing a circuit and a temperature sensor, a computer accessing a website, and a mobile device capable of receiving SMS messages.

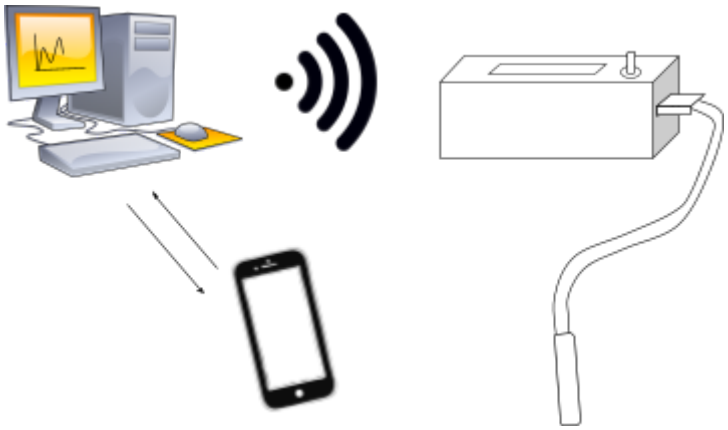


*Figure 1.1: A high-level view of how the devices are communicating*

The third box itself can display the current temperature and send temperature data over WiFi. The computer can view the temperature data from the last 300 seconds as a graph, and can send an SMS alert to the phone if the temperature exceeds a user-defined upper or lower bound. This high-level view can be broken down in more detail as shown in Fig. 1.2.

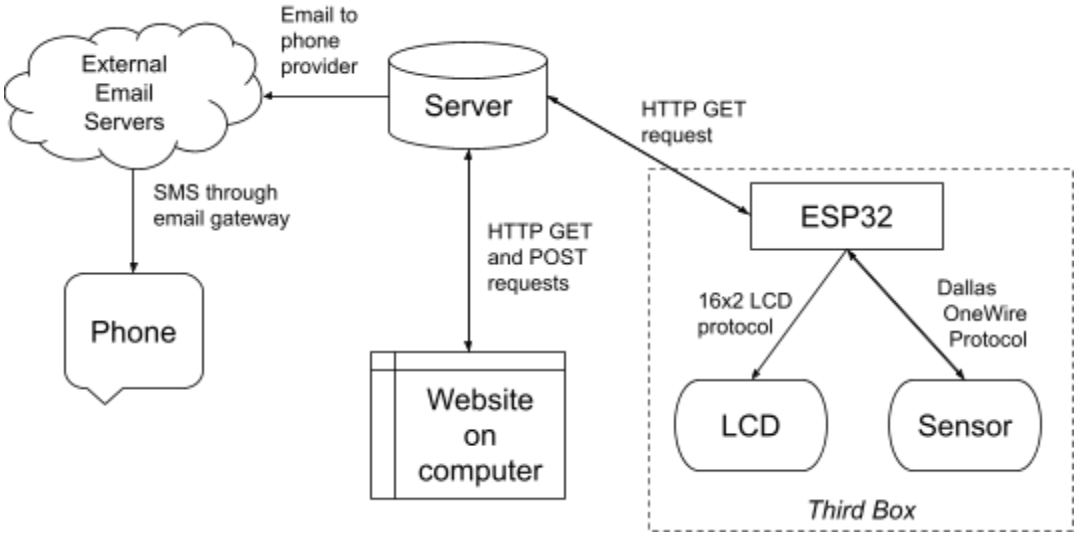



*Figure 1.2: A diagram of major subsystems and communications between them*

The third box consists of the subsystems enclosed by the dashed gray line in Fig. 1.2. These three components can function on their own, without the connection to the rest of the system. The design of the third box had to fulfill a combination of hardware, mechanical, and software goals. The hardware goals consisted of connecting the ESP32 microprocessor to an LCD and a temperature sensor. The software goals included programming the ESP32, connecting the ESP32 to WiFi, taking temperature readings through the ESP32, and sending data from the ESP32 to the server using HTTP requests. The mechanical side of the design focused on constructing a robust container to house the hardware.

The systems shown in Fig. 1.2 outside of the third box were entirely software-based. The goals for these systems consisted of setting up a website and database on a Web server, developing backend software, and developing frontend software. The backend software goals can be separated into the following: ability to send and receive data by communicating with the ESP32, ability to preprocess data and save it to storage on the server, ability to send SMS messages at certain temperature thresholds, and ability to provide accurate data to the website frontend. The frontend software goals consist of displaying accurate data in the appropriate format, allowing users to modify certain aspects of the website appearance, allowing users to change the phone alert settings, and the ability to turn the LCD on or off.

These general goals guided our design of the third box and the overall temperature measurement system. The resulting design was rigorously tested and was shown to meet the user requirements provided in the lab description.

## 1.2 - Hardware Documentation

The hardware design of the box required the interconnection of multiple subsystems. The first subsystem is the temperature sensor. The hardware of the sensor consists of a probe encased in metal, connected to one end of a 2 meter cable; on the other end of the cable is a pluggable terminal with three inputs: GND, VCC, and Data. The Data line is connected to IO pin D4 on the ESP32, which allows the microcontroller to communicate with the sensor and read data. VCC and GND are connected to the ESP's 3.3V and ground pins, respectively.

The Power supply of the system consists of four 1.5V AA batteries in a plastic case with a switch included within the case. We connected the power supply to the VIN pin of the microcontroller and to Pin 1 of the push button. Thus the power supply switch is able to control whether both the ESP32 and the LCD are in an "on" or "off" state.

The push button also plays an important role in our design.We chose a SPST push button with 4 pins to control the LCD. Pin 1 was connected to power; pin 3 was connected to LCD in order to control its backlight; and pin 4, since it is shorted to pin 3, was connected to the microcontroller in order to implement the "virtual button" capability allowing the website to control the LCD as well. With that configuration, whether the push button is being pressed or not, the ESP32 is able to turn the LCD on by setting the IO pin to high.

The final subsystem is the LCD. We used a 16x2 LCD which we obtained from a previous course's lab kit. By referencing the datasheet, we determined the functionality of each of the LCD pins and connected them to their respective IO pins or to power or ground. The LCD will turn on when the push button is being pressed or if the IO pin D27 is set to high.

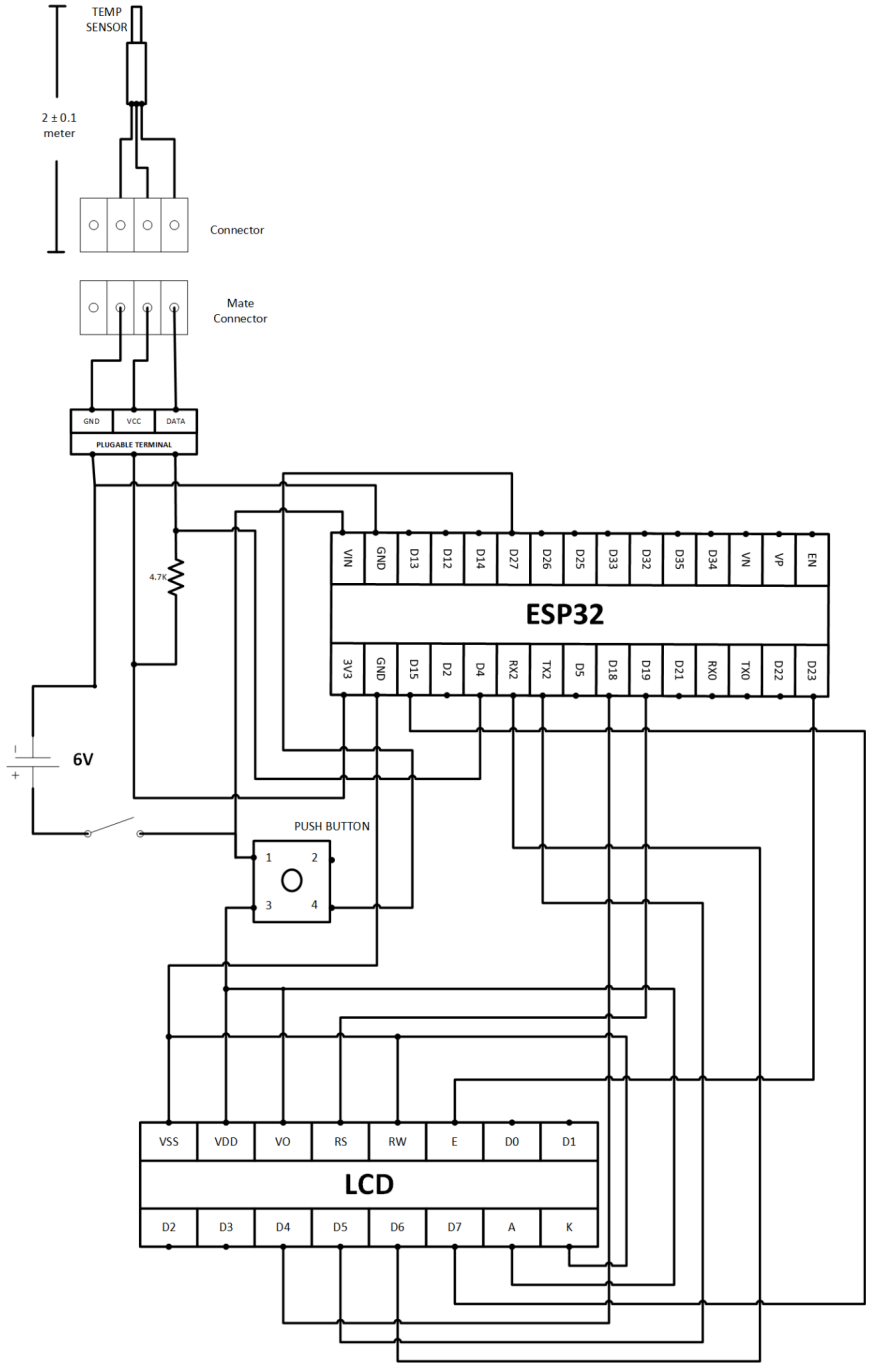



*Figure 1.3: Schematic of circuit design of the third box*

We used four 1.5V batteries connected in series, which provides 6V to the circuit. The 6V terminal of the power supply is is connected to the VIN pin on the ESP32, and the ground is connected to the GND pin on the ESP32. The ESP32's internal hardware regulates the voltage and steps it down to 3.3V, which can be accessed from the 3.3V pin. This lower voltage is used as the power supply of the rest of the circuit.

Connecting the sensor required some planning. Based on information from the sensor datasheet, we picked 4.7 kΩ as the pull-up resistor for 3.3V DC input voltage. To connect the sensor cable to our box, we used a four-wire connector, as shown in Fig. 1.4. Because the sensor only has three lines, we removed one of the wires of the connector. Since the cable that originally came with the sensor was about one meter long, it was necessary to buy a separate 3-wire cable to extend the length to two meters. We soldered both ends of the cables and taped each wire with electrical tape to make sure none of the wires would short. For mechanical stability, we used heat shrink to cover the section where the two cables were connected.



*Figure 1.4: Four-wire connector, modified for three-wire cable*

| ***Hardware Parts List*** | *Quantity* |
|---|---|
| 16x2 LCD | 1 |
| ESP32 Devkit v1 | 1 |
| Push Button | 1 |
| 1.5V AA Battery | 4 |
| Power Supply Case with Switch included | 1 |
| 4.7 K Ohms Resistor | 1 |
| DS18B20 Temperature Sensor | 1 |
| 3-wire Cable | 1 |
| 4-wire Connector | 1 |
| Tupperware Container | 1 |

The completed hardware implementation is shown in the figures below. In Fig 1.5a, the inside of the box is visible, showing the ESP32 and the power supply. Fig. 1.5b shows the outside of the box, with the LCD, push button, and power switch visible.

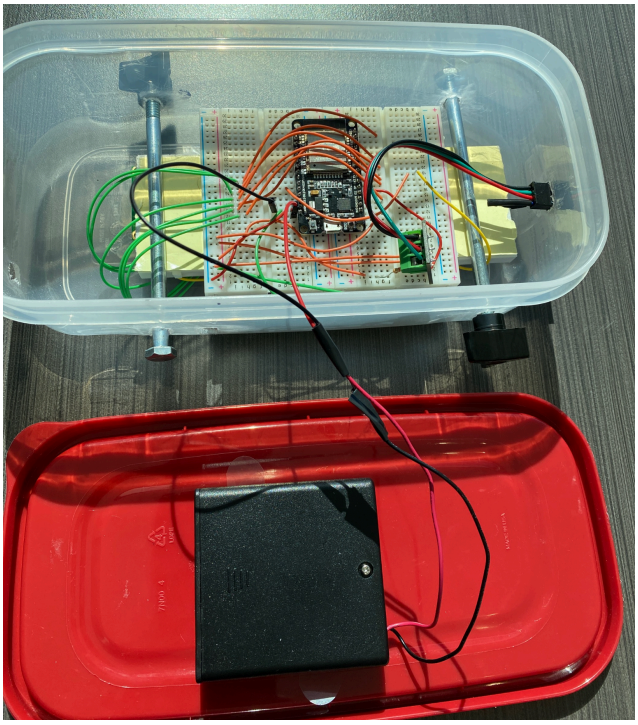


*Figure 1.5a: The complete project hardware, view of the interior*




*Figure 1.5b: The complete project hardware, view of the exterior*

## 1.3 - Mechanical Documentation

When first thinking about how to construct the box, we considered many options, including 3D printing, wood, and cardboard. 3D printing would ultimately be far more expensive than the other options and after briefly looking into 3D printing materials, we were not confident that it would withstand the drop test if it landed on a hardwood floor. Cardboard was cheap, but the safety of our circuit was dependent on the thickness of the material. We landed on the idea of using a tupperware container and cutting holes within the surface so the user could interact with both the buttons and see the LCD clearly. Tupperware is sturdy and provides a large volume for a reasonably cheap price. To make sure our LCD screen and button sat on the top, we created a “shelf” with two long screws that go across the box, as shown in Fig. 1.6.

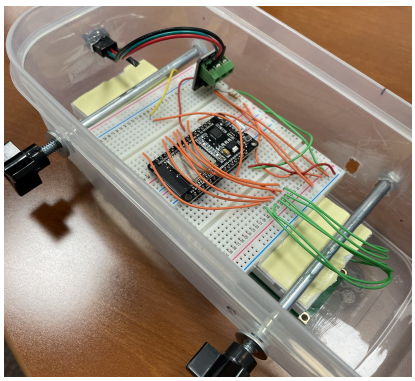


*Figure 1.6: View of the “shelving” mechanism*

This created a secure barrier that won’t allow them to move, while also making it easy to access the breadboards when needed. One concern about the drop test was that the LCD might shatter, so we added a “shield” using glue to protect the LCD and button. This can be seen in Fig. 1.7 below. If dropped on the side with the display, the LCD and button will not hit the ground and the superglue segments will absorb the impact instead.



*Figure 1.7: Superglue shield around LCD*

Placing the battery pack required some trial and error. The first location we selected didn’t work because of the added shelf mechanism, and the side of the box didn’t work due to the curved edges. Thus, we decided to use command strips to secure the battery pack to the back of the container lid, as this was the only location large enough to hold the battery pack.

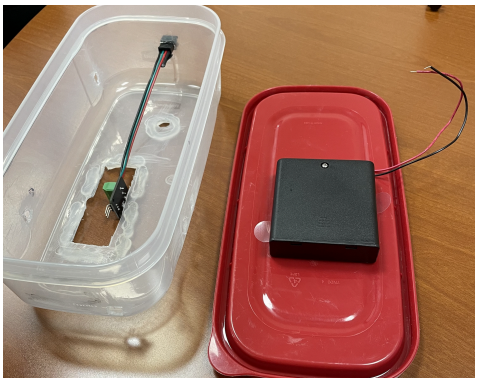


*Figure 1.8: Empty tupperware and battery placement under lid*

## 1.4 - Software Documentation

The software design of this project consists of three main parts: an Arduino C++ program on the ESP32, a Python backend running on the server, and a JavaScript frontend running on the user's web browser. All of these subsystems communicate through HTTP GET and POST requests, and their functionality is interconnected.

The C++ program on the ESP32 was written in the Arduino IDE, which was also used to compile and upload the program to the ESP32. This program can be divided into four main tasks which run in a loop, as shown in Fig 1.9. First, the program must read temperatures from the sensor. The second task is displaying temperature readings to the LCD. The third task is uploading the temperature reading to the Internet. The fourth task is checking if the user has turned on the LCD from the website, and updating the LCD state.

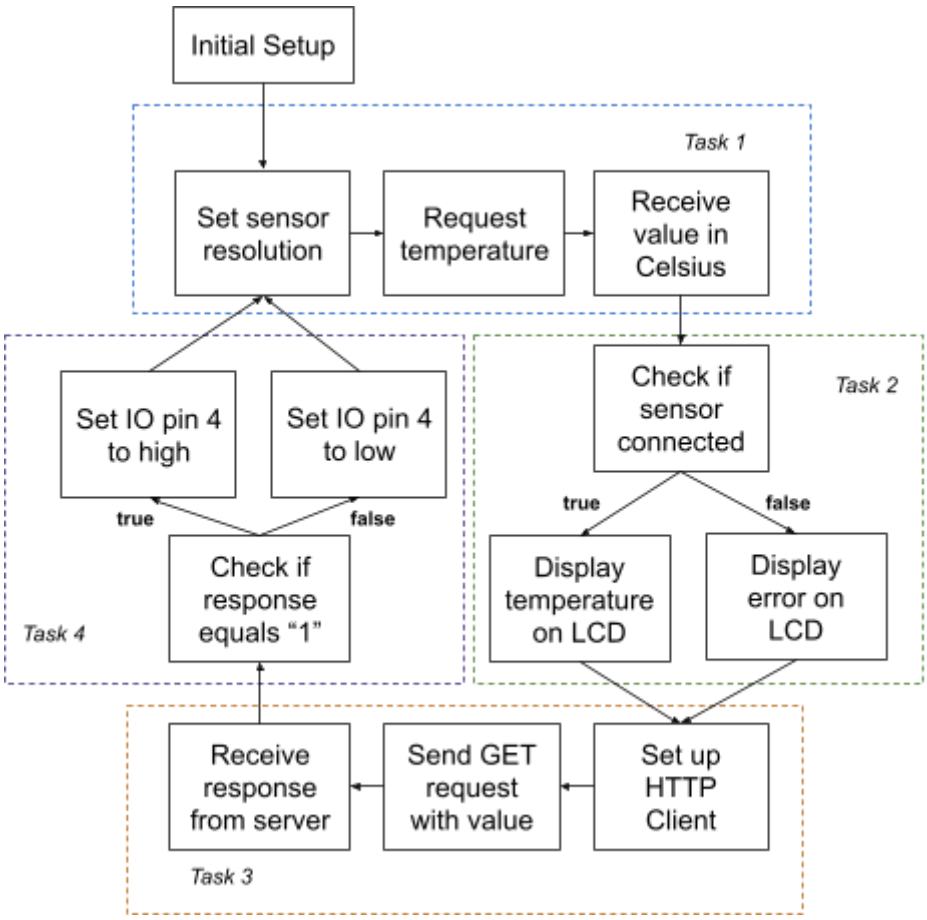



*Fig. 1.9: Flowchart of the main loop*

The first task is accomplished using the OneWire library and the DallasTemperature library. These two libraries allow the ESP32 to communicate with the DS18B20 and request temperature readings. During the initial setup, an instance of OneWire is set up on IO pin 4, which is connected to the sensor Data line. During the main loop, the process of temperature readings is as follows: set a 9-bit resolution for the reading using the *setResolution* function,

request the reading using the *requestTemperatures* function, and receive the reading as a Celsius value using the *getTempCByIndex* function.

The second task is accomplished using the LiquidCrystal library. This library provides an API for communicating with LCDs, using the *setCursor* and *print* functions to move the cursor to a position and send a string to display. If the sensor is disconnected, the temperature reading will be very low (less than -100 deg C). Thus, if the temperature read in the first task is very low, the LCD displays an “Error: Sensor Disconnected” message. Otherwise, the sensor displays “Current temp: X deg C” where X is the current temperature reading.

The third task is accomplished by using the WiFi library, the WiFiMulti library, and the HTTPClient library. These libraries are used to establish a connection to a WiFi network, to check if the ESP32 is connected, to make an HTTP GET request that uploads the temperature reading to a specified URL, and to read the GET response sent from a web server. The URL for the GET request is “http://accessdenied.pythonanywhere.com/up?val=X”, where X represents an integer value read from the sensor. This URL is processed by server-side software, which will be explained further on in this section.

The fourth task is accomplished by wiring the LCD backlight to IO pin 27 on the ESP32. This pin is controlled by the ESP program as follows: if the WiFi is connected and the HTTP GET request returns a value of “1”, then IO pin 27 is set to high. Otherwise, it is set to low. This value is sent by the server, which we have configured to send a “1” when the user turns on the LCD from the web interface, and “0” when the user turns it off from the web interface.

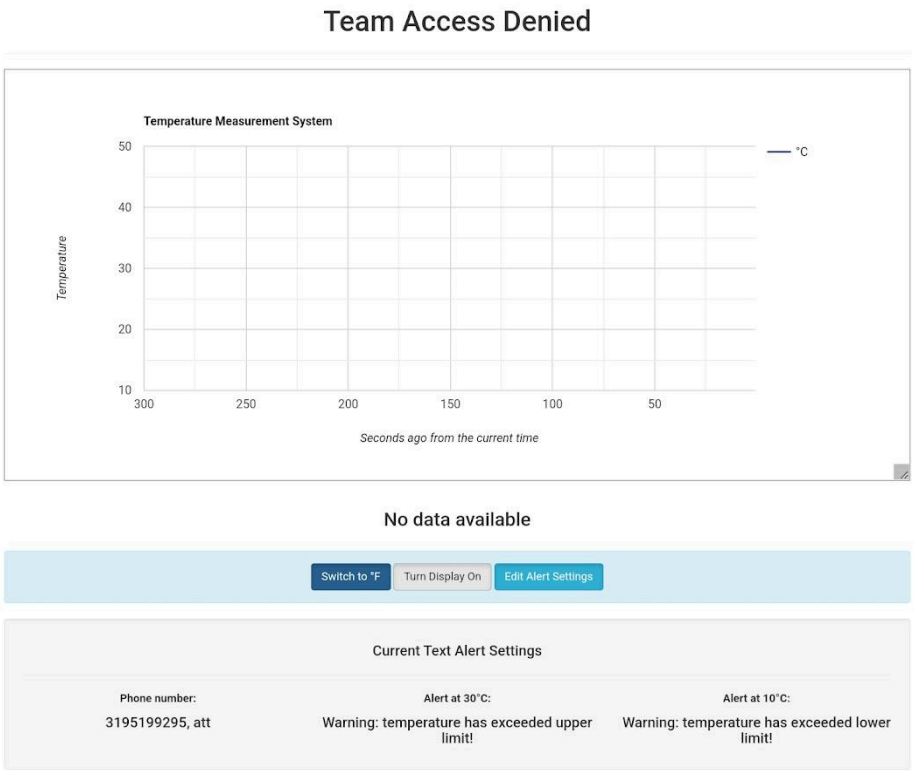



*Figure 1.10: User interface on main web page (main_page.html)*

The web interface component of the project is contained within four files: **settings.html**, **main_page.html**, **flask_app.py**, and **data.json**. All of these files are stored on a web server provided by the PythonAnywhere platform, which is a service by Anaconda that allows users to design Python web apps for free. For this project, a web app was implemented in Flask, which is a micro web framework that provides a minimalist web server, with features such as routing and database access. The main server-side features are all contained within **flask_app.py**, and the user front-end is contained within **settings.html** and **main_page.html**. All other data, such as temperatures and user settings, are stored in **data.json**.

The main user interface is **main_page.html**. As shown in Fig. 1.10, the interface consists of a graph, a text element displaying either the current temperature or an error, a row of buttons, and another display for current user settings. The visual appearance of the website, such as the page layout and the look of the buttons, was created using the Bootstrap 3 CSS framework. In addition, CSS is used to make the graph display scalable with the mouse.

Within this HTML file, we use JavaScript to provide front-end functionality, and we also use the JQuery and Google Chart libraries in order to display the temperature data in a graph and add functionality to buttons on the website. Using the *setInterval* function, the browser reruns the *timingLoad* function once every second to get data from the **data.json** file on the server and redraw the chart.

Within **data.json**, the temperature data is stored as a list of pairs. Each pair contains a timestamp (in seconds from the epoch) and a temperature reading in degrees Celsius. When drawing the graph, the *getData* function iterates over the list of pairs and determines where on the graph to plot each temperature value. The “number of seconds ago” for a temperature value is found by comparing the timestamp with the current time in seconds. If the user turns on the Fahrenheit setting, the temperature readings are converted to Fahrenheit before being graphed.

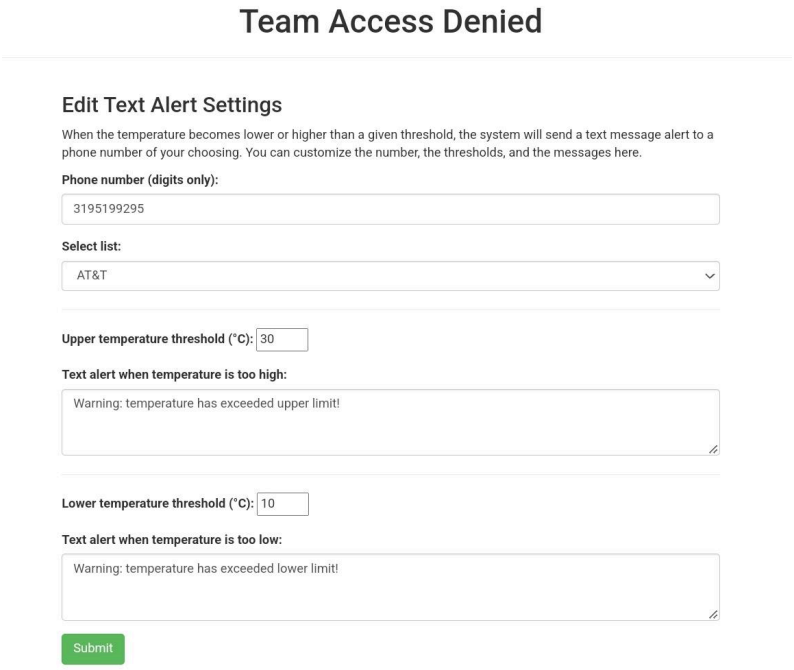



*Figure 1.11: Form to change text alert settings (settings.html)*

Pressing the “Edit Alert Settings" button on the main page reroutes the user to **settings.html**, which is a web page that allows the user to change the text alert settings. As shown in Fig. 1.11, the user can change the phone number, phone provider, upper and lower temperature thresholds for sending text alerts, and the content of the text alerts. Submitting the form will send the data to the server as an HTTP POST request, and the server will process and save the data as fields within the **data.json** file.

The ability to send SMS messages is implemented in a roundabout fashion, as there is no way to directly send messages from a web application without using a paid API such as Twilio. In order to avoid such APIs, we used a method known as “SMS via email gateway.” In this method, an email sent to a phone provider company’s mail servers can be used to send a text message. For example, given a 10-digit number 123-456-7890 provided by AT&T, sending an email to *1234567890@txt.att.net* will trigger an SMS message to be sent to that phone number, containing the same content as the email. The entire process of sending an email and receiving a text is shown as a diagram in Fig. 1.12.

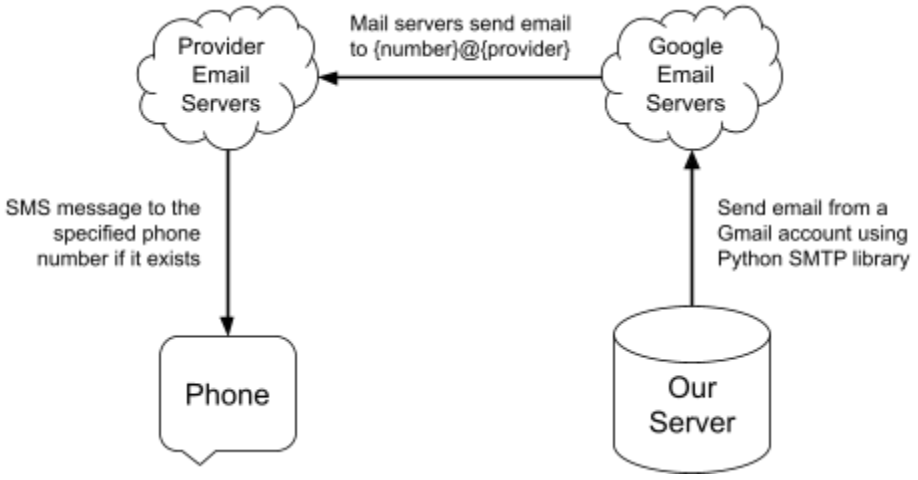



*Figure 1.12: Diagram of email-to-SMS gateway*

When the server receives an uploaded temperature value from the ESP32, it checks whether that value is above or below the user limits saved in **data.json**. Then, if the temperature exceeds the limits, the server sends out an SMS via email gateway to the user-specified phone number. The content of this SMS is determined by the saved text alerts, which can be modified by the user on the “Edit Alert Settings” webpage (**settings.html**).

Emails are sent from a dedicated Gmail account that we set up for this purpose, which can be accessed by our backend Python code using Python’s “smtplib” library. This library provides an API for Python developers to send emails by making a Simple Mail Transfer Protocol (SMTP) request. A table of providers and corresponding mail servers is shown below.

| **Provider** | **Address** |
|---|---|
| AT&T | txt.att.net |
| T-Mobile | tmomail.net |
| Verizon | vtext.com |
| Boost | sms.myboostmobile.com |
| Cricket | sms.cricketwireless.net |
| US Cellular | email.uscc.net |

# Design Process and Experimentation

## 2.1 - Design Process

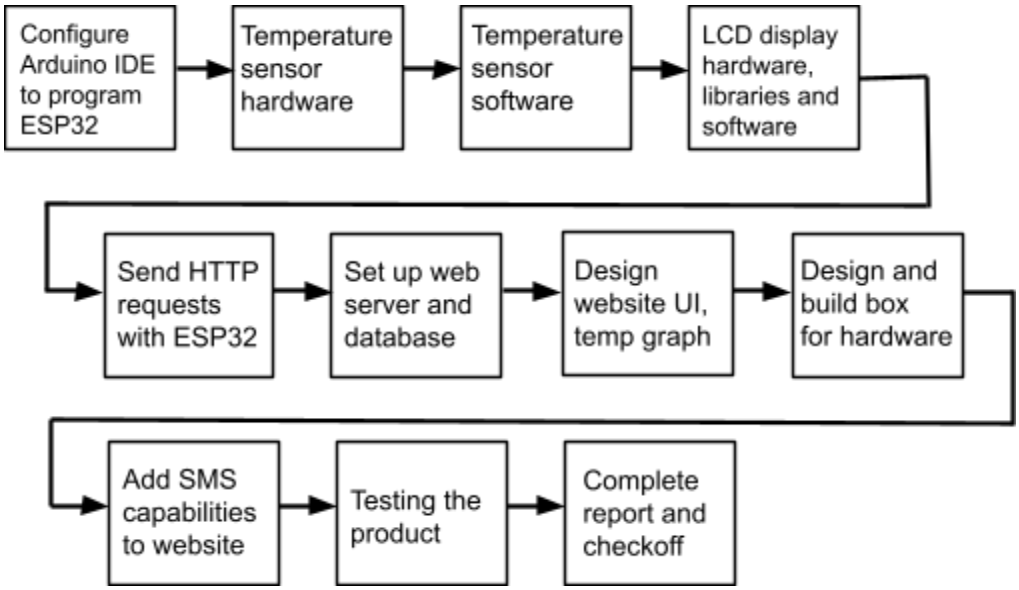



*Figure 2.1: Flowchart depicting the design process*

## 2.2 - Hardware Design Choices

The first major design choice we encountered during the planning stages was choosing the microprocessor. Because of our past and current experiences in embedded systems, we were first leaning towards using an Arduino Uno. However, once we realized that we would need to purchase a separate chip to add WiFi capabilities to the Arduino, we began to look at alternatives. We researched the price and features of the Raspberry Pi, ESP32, and ESP8266. In general, processors like the Raspberry Pi offer more features and computational power, at the cost of more setup required to program it. We were then able to identify that the ESP32 (see Fig. 2.2) would fit our needs and provide a good compromise between computational power and ease of use. The ESP32 offers Wifi capabilities, has hardware similar to the Arduino Uno, and can be programmed in a language that we know well (Arduino C++).

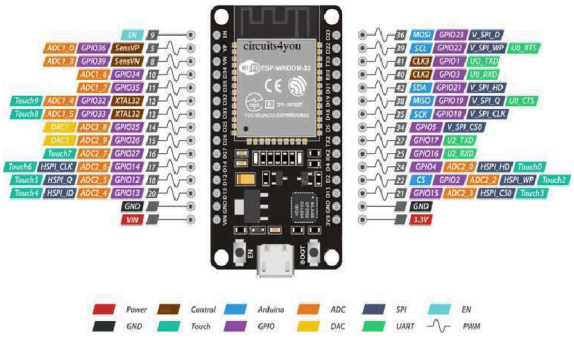



*Figure 2.2: ESP32 Pin Diagram*

The second hardware part we started to think about and plan was the temperature sensor. Our first idea was to build a thermocouple sensor due to Ming and Werner's previous knowledge from advanced electrical courses, but with the layout of the finished product we were unsure of how to execute this idea. Next, we thought of using a TMP36 sensor since Ming, Maneesh, and Kiana had already set up hardware and created code to read results in projects requiring the sensor in a previous course. However, because the TMP36 sensor was not waterproof and the cable was difficult to set up, we decided to keep looking.

Once we came across the DS18B20 sensor (see Fig. 2.3) it was clear that it met all of the specifications. It was already waterproof and could be purchased with an attached cable, and it was well-documented. One unclear detail was whether to run the sensor in normal mode or parasite mode. Normal mode would connect the sensor VCC line to a power supply, while parasite mode would allow the sensor to take power from the Data line signal without connecting the VCC to a power line. After some research, we learned that parasite mode is less reliable and can result in errors during operation. Thus, we decided to use normal mode.




*Figure 2.3: DS18B20 and cable*

One further detail we had to decide was the choice of resistor connected to the data line on the sensor. We chose 4.7 kΩ first based on the datasheet, and we believed it was a suitable resistance for both 3.3V and 5V DC input voltage.

However, as we were developing the website, the data sent from the sensor had a small time delay so that it was not forming a solid straight line on the website graph. We suspected that the sensor might not be getting enough current to operate correctly, so we checked the maximum operating current, which is 1.5 mA. We hypothesized that a 4.7 kΩ resistor is too large for 3.3V operation, and we changed it to 3.3 kΩ. However, this change did not solve the delay issue. We also tried using 4.7 kΩ and connected the sensor's VCC line directly to the 6V power supply. However, the issue still persisted. Finally, we discovered that the root cause of the delay in sensor readings was not in the hardware, but in the software. This is discussed further in Section 2.3. Because it was not a hardware issue, we switched back to using 3.3V and 4.7 kΩ as the datasheet recommended.

When first constructing the hardware, a button was obtained from a previous lab hardware kit. The functionality and wiring was working perfectly, but with the box construction we soon realized that we needed a taller button in order for it to be accessible by the user. Therefore we obtained a new longer button which used the same setup, was available at the engineering electronics shop, and worked with our design.



*Figure 2.4: Long push button*

We had bought a switch from the engineering shop initially when planning our design, but once we bought the battery pack we decided to incorporate the battery pack's built-in switch instead because it reduced the number of hardware components and wires, and the performance of the built-in switch already matched the user requirements.




*Figure 2.5: The battery pack*

## 2.3 - Software Design Choices

We also made several key design choices in software. Early on, we realized that we needed to host a website on the Internet in order to display temperature data. However, we also realized that we needed a way to upload data to the Internet and store it on a server. Therefore, we would need to find server space for our code, and then develop server-side software which was capable of accessing a database.

There are various options for server-side, or "backend," development. Among the most popular are LAMP stacks (a bundle of Linux, Apache HTTP server, MySQL database, and PHP). Hosting space can be found through various platforms as well, such as GoDaddy, AWS, and Google Cloud. One interesting option we found was to use Google Cloud for both the database and the hosting, as Google Cloud offers cutting-edge database services such as Firebase. However, the largest issue we were facing was that setting up a website is not a trivial

process. One must buy and register a domain name, find hosting space, and set up a suitable backend stack. In order to simplify this process, based on Maneesh's previous experience, we decided on using the PythonAnywhere platform by Anaconda. This platform provides free hosting space, as well as a backend stack consisting of Linux, Python, and MySQL. The only tradeoff is that a free account has caps on CPU usage. However, we decided to take this option, as upgrading the account would not be very expensive if needed.

In addition, we made the decision to use a JSON file for data storage instead of a relational SQL database, because JSON is very lightweight, while an SQL database would require more computational resources and potentially higher delays for reading or writing data. JSON is not designed for making complex queries about data, but the functionality we require is very simple and the ability to make complex queries is unnecessary for our purposes.

For the purpose of sending SMS messages, we considered two options. The first was Twilio, which is a paid API that allows developers to send text messages from a computer application. However, due to the pricing of this option, we decided to explore alternatives. The only alternative we found was a method known as SMS via email gateway. We decided to test this method by sending an email to a phone number. After verifying that it did indeed send an SMS message to that phone number, we decided to use this option because it was free.

As mentioned in Section 2.2, we encountered an issue during development in which the data displayed on the graph was not a solid line. There were gaps in the data every second or every few seconds at irregular but frequent intervals. After determining that the cause was not the hardware, we looked for issues in the software. We first tested the website frontend and backend, but we determined that they were definitely fast enough to handle updating once per second. We then looked for issues in the program on the ESP32. For example, we tested whether the delay function we were using was too long, or whether the serial monitor print statements we added for debugging were too slow. However, none of these appeared to be the cause of the delay. Thus we determined that the delay must come from the libraries we used to communicate with the temperature sensor.

After more research into the libraries, we learned that this was a known issue of the Dallas Temperature library. We identified and fixed the issue by switching to an older version of the library, as well as modifying one constant in the library code. This immediately fixed the delay issue, which allowed us to complete the project within our expected timeline.

# Test Report

## 3.1 - Test Specifications

The following table lists the user requirements that must be satisfied in order to verify the performance of the system. We have labeled each user requirement with an unique test ID number, a test name, and the details of the requirement.

| Test ID | Test | User Requirement |
|---|---|---|
| 1 | Cable Length | Thermometer sensor lies at the end of a 2.0+-0.1 meter cable. |
| 2 | Ice Water | Thermometer sensor should not be damaged when placed in ice water. |
| 3 | Box Features | Box contains at minimum of: a display, a button, a battery, and a power switch. |
| 4 | Battery Operated | The thermometer is functional off of just batteries and no connection to the computer or any other source. |
| 5 | Physically Robust Box | The box and its contents can withstand being dropped from the workbench to the floor. |
| 6 | Turn Upside Down | The box can be turned upside down with the circuit, connectors, and switches still working. |
| 7 | Connector | The temperature sensor should be easily connected and disconnected by the user. |
| 8 | Drop with Cable | When the cable is connected and the box is dropped onto the floor, the connectors and cables should not break but can become disconnected. |
| 9 | Sensor Plugged In | When the sensor is unplugged and plugged in again the LCD should continue operation with no user intervention |
| 10 | Switch Off | When the switch is off, the thermometer system cannot display temperatures and the data is not available on the internet |
| 11 | Switch On | When the switch is on, the correct temperature appears on the display when the button is pressed with no delay |
| 12 | Button Press | The button is “momentary contact” |
| 13 | Error Condition | If the temperature sensor is not plugged into the box, an error condition is shown to user |
| 14 | Unplugged Sensor | If the temperature sensor is not plugged into the box, a message stating "unplugged sensor" will appear on the computer |
| 15 | No Data Available | If the switch on the box is off, a "no data available" message will |

| | | |
|---|---|---|
| | | appear on the computer |
| 16 | Virtual Button Press | The temperature display can be turned on and off from the computer |
| 17 | Past Readings | Within ten seconds of starting the software on the computer, the past 300 seconds of data should be available |
| 18 | C/F Test | Graph is available in both degrees C and degrees F |
| 19 | Graph Format | Graph scrolls horizontally with the latest temperature on the right side of the screen. Once a second, a new value is uploaded and added to the graph. Older values scroll off the graph. The X-axis is labeled "Seconds ago" and the Y-axis is labeled "Temp" with the degree type. |
| 20 | Scalable | Physical size of graph is scalable with mouse |
| 21 | Missing Data | If data is missing, it should be obvious on the graph |
| 22 | Graph when Off | Graph continues to scroll even if temperature sensor is not plugged in or box is off |
| 23 | Connecting to Sensor | If the computer is on and the box is off, the data should appear within ten seconds of the box being turned on |
| 24 | Text Message | When both the computer and the box are on, if a temperature exceeds or dips lower than a given value, a text message is sent to a specified number |
| 25 | Hand Test | When the temperature sensor is held in someone's hand, the heat for their fingers should increase the readings on the temperature sensor |
| 26 | Temperature range | The design can handle temperatures below -10 C and above 63 C |
| 27 | Room Temperature | Verify that in the room temperature in the lab reads 22 degrees C +- 4 degrees C |
| 28 | Ice Mixture Temperature | When placed in water-ice mixture, the thermostat should read 0 degrees C +- 2 degrees C |

## 3.2 - Test Plan and Pass Criteria

The following table states the test ID, test name, and the passing criterion, respectively. The Passing Criterion column refers to the specific results that are considered acceptable to meet the corresponding requirement, as well as the error tolerance allowed for each test.

| Test ID | Test | Pass Criterion |
|---|---|---|
| 1 | Cable Length | 1.9 meter to 2.1 meter long cable |
| 2 | Ice Water | Thermometer sensor retains all function after fully submerged |
| 3 | Box features | Box contains display, button, battery, and power switch |
| 4 | Battery Operated | The thermometer can read accurate data and show it on the display with only battery power |
| 5 | Physically Robust Box | Once knocked off of the workbench, the temperature sensor, LCD, and WiFi should all work without issues |
| 6 | Turn Upside Down | All functionality works after turning the box upside down and wires all remain connected |
| 7 | Connector | The sensor can be connected and disconnected from the box without causing damage to the box and allowing full functionality when plugged in |
| 8 | Drop with Cable | All functionality remains within the cables is working after the drop |
| 9 | Sensor Plugged In | The LCD continues displaying the temperature on the LCD and on the website after being reconnected |
| 10 | Switch Off | When off, the LCD screen is black and the button doesn't work.<br>No data is being transferred from the internet or shown in the graph. |
| 11 | Switch On | The delay is less than 20 milliseconds and the temperature is tested and compared with another source and displayed in degrees C |
| 12 | Button Press | As soon as the button is pressed the display is on, and the display should go off with no delay when the button is released |
| 13 | Error Condition | Display tells user there's an error condition |
| 14 | Unplugged Sensor | An unplugged sensor message appears only when the sensor is unplugged |
| 15 | No Data Available | Turn the box on and off. Confirm that if the box is off, and only then, the "no data available" message appears on the screen |
| 16 | Virtual Button Press | Pressing a button on the computer turns the button on and off within 1 second |

| | | |
|---|---|---|
| 17 | Past Readings | After ten seconds we should be able to see and confirm the past 300 seconds are available |
| 18 | C/F Test | A button is available to toggle between both degrees C and degrees F on the website's graph |
| 19 | Graph Format | Every one second a new value is uploaded. Format looks as described in user requirements. Scrolls horizontally to past values on the left and most recent values on the right. |
| 20 | Scalable | Drag corner of the graph. Size of the graph should change accordingly. |
| 21 | Missing Data | Unplug the temperature sensor and plug it back in. Verify that there is a gap within the data graphed. Do the same for turning the switch off. |
| 22 | Graph when Off | Unplug the temperature sensor and verify that the graph continues to scroll and show data is missing. Verify this for turning the box off as well. |
| 23 | Connecting to Sensor | Have the computer running and turn on the box. Within ten seconds the data should appear in real time. |
| 24 | Text Message | Make the value on the temperature sensor go past the appropriate range and confirm a text message is sent to the number. |
| 25 | Hand Test | Verify that holding the sensor in your hands increases the temperature readings |
| 26 | Temperature range | If possible, verify sensor can read temperatures below -10C and above 63C |
| 27 | Room Temperature | In the lab, verify the sensor reads somewhere in the range of 18C and 26C |
| 28 | Ice Mixture Temperature | Verify the sensor reads in the range of -2C to 2C in ice-water mixture |

## 3.3 - Test Report

The following table is a completed report of our testing and verification phase. Each row contains information on one test: the test ID, the test name, the user requirement satisfied, the passing criterion, the actual results of the test, the pass or fail result, the date each test was completed, and any corrective action taken.

| Test ID | Test | User Requirement Addressed | Pass Criterion | Actual Result | Pass/ Fail | Date | Corrective Action |
|---|---|---|---|---|---|---|---|
| 1 | Cable Length | Thermometer sensor lies at the end of a 2.0+-0.1 meter cable | 1.9 meter to 2.1 meter long cable | 2.06m | Pass | 9/25/ 2022 | NA |
| 2 | Ice Water | Thermometer sensor should not be damaged when placed in ice water | Thermometer sensor retains all function after fully submerged | No damage to the sensor | Pass | 9/25/ 2022 | NA |
| 3 | Box features | Box contains at minimum of: a display, a button, a battery, and a power swtich. | Box contains display, button, battery, and power switch | The LCD is on the top of the box besides a push button. 6V batteries are placed in a battery pack with a switch and it's attached to the bottom of the tupperware. | Pass | 9/25/ 2022 | NA |
| 4 | Battery Operated | The thermometer is functional off of just batteries and no connection to the computer or any other source | The thermometer can read accurate data and show it on the display with only battery power | The thermometer reads in accurate data and data appears on the LCD with only battery connection | Pass | 9/25/ 2022 | NA |
| 5 | Physically Robust Box | The box and its contents can withstand being dropped from the workbench to the floor | Once knocked off of the workbench, the temperature sensor, LCD, and WiFi should all work without issues | Knocked off the workbench and the temperature sensor, LCD, and WiFi worked without any issues, but lid came off | Pass | 9/25/ 2022 | Taped the container lid to prevent it from opening |
| 6 | Turn Upside Down | The box can be turned upside down with the circuit, connectors, and switches still working | All functionality works after turning the box upside down | Turning upside down does not affect functionality and wires all | Pass | 9/25/ 2022 | NA |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | and wires all remain connected | remain connected | | | |
| 7 | Connector | The temperature sensor should be easily connected and disconnected by the user | The sensor can be connected and disconnected from the box without causing damage to the box and allowing full functionality when plugged in | The connector allows the sensor to be easily connected and disconnected from the box without causing damage to the box and full functionality work after the sensor is connected | Pass | 9/25/ 2022 | NA |
| 8 | Drop with Cable | When the cable is connected and the box is dropped onto the floor, the connectors and cables should not break but can become disconnected | All functionality remains within the cables is working after the drop | After the box is dropped from the bench to the floor, cables were not disconnected or broken | Pass | 9/25/ 2022 | NA |
| 9 | Sensor Plugged In | When the sensor is unplugged and plugged in again the LCD should continue operation with no user intervention | The LCD continues displaying the temperature on the LCD and on the website after being reconnected | After the sensor is unplugged and reconnected, the temperature is displayed on the LCD as well as on the website | Pass | 9/25/ 2022 | NA |
| 10 | Switch Off | When the switch is off, the thermometer system cannot display temperatures and the data is not available on the internet | When off, the LCD screen is black and the button doesn't work. No data is being transferred from the internet or shown in the graph. | When the switch is off, the battery is off. The thermometer system is turned off and LCD cannot display temperature and the internet doesn’t receive data anymore | Pass | 9/25/ 2022 | NA |
| 11 | Switch On | When the switch is on, the correct temperature appears on the display when the button is pressed with no delay | The delay is less than 20 milliseconds and the temperature is tested and compared with another source | When the switch is on, the delay is less than 20 milliseconds. The temperature is tested and displayed in degrees C | Pass | 9/25/ 2022 | NA |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | and displayed in degrees C | | | | |
| 12 | Button Press | The button is on momentary contact | As soon as the button is pressed the display is on, and the display should go off with no delay when the button is released | There is no delay between the button press and the display on the LCD. When the button is pressed, it displays the temperature in time. When off, the display goes off in time. | Pass | 9/25/ 2022 | NA |
| 13 | Error Condition | If the temperature sensor is not plugged into the box, an error condition is shown to user | Display tells user there's an error condition | The tells user there's an error condition, “sensor is not plugged in” | Pass | 9/25/ 2022 | NA |
| 14 | Unplugged Sensor | If the temperature sensor is not plugged into the box, a message stating "unplugged sensor" will appear on the computer | An unplugged sensor message appears only when the sensor is unplugged | When the sensor is unplugged, “unplugged sensor” shows up on LCD | Pass | 9/25/ 2022 | NA |
| 15 | No Data Available | If the switch on the box is off, a "no data available" message will appear on the computer | Turn the box on and off. Confirm that if the box is off, and only then, the "no data available" message appears on the screen | If the box is turned off, the "no data available" message appears on the LCD screen | Pass | 9/25/ 2022 | NA |
| 16 | Virtual Button Press | The temperature display can be turned on and off from the computer | Pressing a button on the computer turns the button on and off within 1 second | Pressing a virtual button on the computer can turn on the LCD screen within 1 second delay | Pass | 9/25/ 2022 | NA |
| 17 | Past Readings | Within ten seconds of starting the software on the computer, the past 300 seconds of data should be available | After ten seconds we should be able to see and confirm the past 300 seconds are available | The data history in the past 300 seconds are available on the user interface | Pass | 9/25/ 2022 | NA |
| 18 | C/F Test | Graph is available in both degrees C and | A button is available to | Below the website's graph, a | Pass | 9/25/ 2022 | NA |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | degrees F | toggle between both degrees C and degrees F on the website's graph | button can be used to toggle between both degrees C and degrees F | | | |
| 19 | Graph Format | Graph scrolls horizontally with the latest temperature on the right side of the screen. Once a second, a new value is uploaded and added to the graph. Older values scroll off the graph. The X-axis is labeled "Seconds ago" and the Y-axis is labeled "Temp" with the degree type. | Every one second a new value is uploaded. Format looks as described in user requirements. Scrolls horizontally to past values on the left and most recent values on the right. | Every one second a new value is uploaded. The graph on the website is able to scroll horizontally to pass values to the left and most recent values appear on the right. | Pass | 9/25/ 2022 | NA |
| 20 | Scalable | Physical size of graph is scalable with mouse | Drag corner of the graph. Size of the graph should change accordingly. | The size of graph can be adjusted and resized. | Pass | 9/25/ 2022 | NA |
| 21 | Missing Data | If data is missing, it should be obvious on the graph | Unplug the temperature sensor and plug it back in. Verify that there is a gap within the data graphed. Do the same for turning the switch off. | When unplug the temperature sensor and plug it back in there is a gap within the data graph. We had the same behavior when the switch turned off. | Pass | 9/25/ 2022 | NA |
| 22 | Graph when Off | Graph continues to scroll even if temperature sensor is not plugged in or box is off | Unplug the temperature sensor and verify that the graph continues to scroll and show data is missing. Verify this for turning the box off as well. | When the sensor is unplugged and/or box is turned off, the graph continues to scroll and show data is missing. | Pass | 9/25/ 2022 | NA |
| 23 | Connecting to Sensor | If the computer is on and the box is off, the data should appear within ten seconds of the box being | Have the computer running and turn on the box. | When the computer is running and the box is turned on, | Pass | 9/25/ 2022 | NA |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | turned on | Within ten seconds the data should appear in real time. | data appears within ten seconds. | | | |
| 24 | Text Message | When both the computer and the box are on, if a temperature exceeds or dips lower than a given value, a text message is sent to a specified number | Make the value on the temperature sensor go past the appropriate range and confirm a text message is sent to the number. | When temperature sensor values go past the appropriate range a text message is sent to the number. | Pass | 9/25/ 2022 | NA |
| 25 | Hand Test | When the temperature sensor is held in someone's hand, the heat for their fingers should increase the readings on the temperature sensor | Verify that holding the sensor in your hands increases the temperature readings | Holding the sensor in hands increases the temperature readings | Pass | 9/25/ 2022 | NA |
| 26 | Temperature range | The design can handle temperatures below -10 C and above 63 C | If possible, verify sensor can read temperatures below -10 C and above 63C | Because of limited resources, we were able to verify that our sensor down to -3 C to up to 50 C | Pass | 9/25/ 2022 | NA |
| 27 | Room Temperature | Verify that in the room temperature in the lab reads 22 degrees C +- 4 degrees C | In the lab, verify the sensor reads somewhere in the range of 18C and 26C | The sensor reads values in the range of 18 degrees C and 26 degrees C | Pass | 9/25/ 2022 | NA |
| 28 | Ice Mixture Temperature | When placed in water-ice mixture, the thermostat should read 0 degrees C ± 2 degrees C | Verify the sensor reads in the range of -2C to 2C in ice-water mixture | Sensor reads values in the range of -2 degrees C to 2 degrees C in ice-water mixture | Pass | 9/25/ 2022 | NA |

# Project Retrospective

## 4.1 - Roles of Team Members

Leader: Kiana Erickson
Hardware responsible: Werner Bayas and Ming He
Software responsible: Maneesh John
Mechanical responsible: Kiana Erickson
Time management: Kiana Erickson
Budget: Ming He
Buyer: Werner Bayas

## 4.2 - Gantt Chart

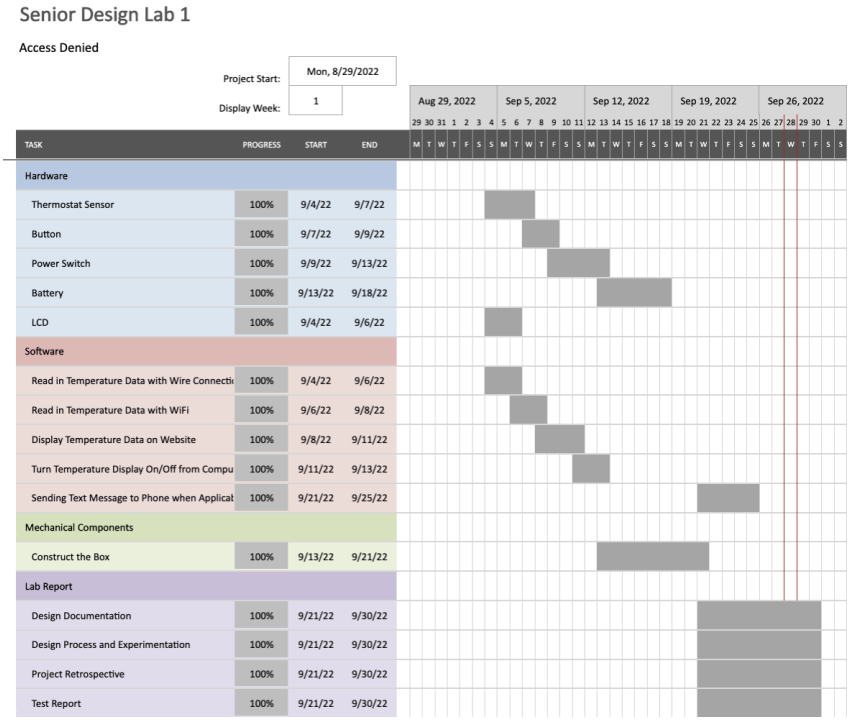



*Figure 3.1: Gantt Chart Illustrating the Timeline of the Lab*

## 4.3 - Project Management

For this project, our team decided to conduct it using agile development project concepts. Our success is based on continuous evolutionary development and improvement. Our design process method allowed us to identify some failures at an early stage and fix them right away. We set a goal for each week and we tried to accomplish them as well as possible. Eventually, we ended up with a working product earlier than expected.

## 4.4 - Workload Distribution

We decided to divide the lab into 3 different parts: the hardware, software, and the implementation. For the hardware, Ming and Werner worked on creating a prototype of a functional system using the devices already mentioned. Maneesh was in charge of creating the software to control the inputs and output of our microcontroller and to connect the ESP32 to the wifi. He also created the code to operate the website and the warning messages that will be sent to the operator. Kiana helped to transform the prototype into a final product, gave support through all the different parts, wrote the tests and their criterions, and kept the team organized.

## 4.5 - Timeline/Plan

The first couple of days, we focused on obtaining all the equipment necessary to complete the project. We chose a day and time that works for all of us to meet each week. Werner and Ming, then started to put all the hardware pieces together so we can test the software from an early stage. In the meantime, Maneesh was working on software to be able to connect the ESP32 to a WiFi network. Once accomplished, we focused on getting the temperature sensor subsystem working. We also needed to have the LCD working to be able to test our data reading from the sensor. We then made sure that our readings were correct and started to work on sending those data readings to the server through WiFi. The next challenges were, on the hardware side, making the push button control the LCD lighting and finding the right power supply for our system; on the software side, we needed to collect the data and transform it into a graph and we needed to send warning text messages to the user. Our final phase was the integration of all the hardware into a compact device that Kiana was creating and modifying throughout the hardware and software phases. After all phases were successfully completed, we were able to finish our documentation and test report.

## 4.6 - Critical Assessment

We finished our lab feeling accomplished and successful. We were able to have our software and hardware working and communicating in between and we had a box robust enough to handle all the testing. All team members maintained clear communication, allowing us to meet our expectations in each phase of development until the end, with no delays.

Most of our design choices turned out to be appropriate or excellent. For example, the ESP32 was a good decision for the microcontroller because it was easy to program, powerful enough for our purposes, and could easily connect to WiFi. However, we ended up having problems when we were trying to connect to WPA2 Enterprise networks, such as eduroam, due to additional security procedures. While the processor can connect to such networks, we had trouble understanding how to implement it, and we did not consider this issue when we bought it. A Raspberry Pi might have been easier to connect to eduroam. However, despite this minor issue, the ESP32 was an excellent choice for all our requirements.

Perhaps because of a lack of product research, we did not make optimal decisions when ordering components. The sensor we ordered had a cable one meter long and cost around $9.

Later, we found that we could have obtained a similar sensor that was three meters long and $5 from the Engineering Shop at the College of Engineering. Similarly, we ordered a switch but later decided to use the battery pack's built-in switch, and we ordered a button but later decided to use a longer button. Some orders could have been avoided with more careful planning and research of components before implementing designs.

Time was another limitation. Because we had limited time, we could not use a printed circuit board (PCB) instead of breadboards. A PCB would have brought more mechanical stability to our final product and would have allowed us to reduce its size.

On the software side, our design choices all proved to be sufficient for our purposes. Our free PythonAnywhere account provided hosting space, a database, and the ability to run server-side Python code, and we never exceeded the CPU limits of the free account. Our website frontend was clean and effective due to the Google Charts and jQuery libraries.

The only software design choice that led to obstacles was the use of the Dallas Temperature and One Wire libraries in Arduino C++. These libraries led to a bug in logic that was very difficult to identify, and also difficult to correct. This could possibly have been avoided by doing more research into these libraries before deciding to use them. However, ultimately they did provide the functionality we required, so the end result was satisfactory.

One of the main concerns we can make improvements on will be paying more attention to small design details for our finished product. We realized that the way that our product looked might not be the most appealing or user-friendly to everyone. Considering not only functionality but also user-friendliness is a big aspect of product design, and we will make sure to add that to our overall design goals for the next project.

# Appendix & References

## 5.1 - Hardware Datasheets

"16x2 LCD Display Module." Compotents101.com, 2021. [Online] Available: https://components101.com/displays/16x2-lcd-pinout-datasheet

"ESP32-DEVKITM-1." docs.espressif.com. [Online] Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/user-guide-devkitm-1.html

"DS18B20 - Programmable Resolution 1-Wire Digital Thermometer." Digi-Key.com. [Online] Available: https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf

## 5.2 - ESP32 Libraries

"Dallas Temperature Library." [Online] Available: https://www.arduino.cc/reference/en/libraries/dallastemperature/

"One Wire Library." [Online] Available: https://www.arduino.cc/reference/en/libraries/onewire/

## 5.3 - Backend Development

"PythonAnywhere." [Online] Available: https://www.pythonanywhere.com/

"Flask Microframework." [Online] Available: https://flask.palletsprojects.com/en/2.2.x/

## 5.4 - Front-End Development

"Google Charts API." [Online] Available: https://developers.google.com/chart

"Bootstrap 3." [Online] Available: https://getbootstrap.com/docs/3.3/

"JQuery." [Online] Available: https://jquery.com/