Alex Powers, Ben Mitchinson, Meghan McLaughlin, Zain Khan
Group H (Team “Easy as Pi”)
ECE: 4880
October 7th 2019

Design Documentation

Problem Statement

Design a thermometer with an onboard display that is capable of relaying temperature data to a web interface in realtime. This web interface must also graph the incoming temperature data over a range of three-hundred seconds in respect to current time, as well as provide controls for operating the thermometer display, and for managing notification preferences for customizable text alerts. The thermometer must have physical controls to enable and disable the display, as well as turn the device on and off. The web interface must reflect the display / power state, as well as notify the user if the sensor has been disconnected.

Design Considerations / Trade offs

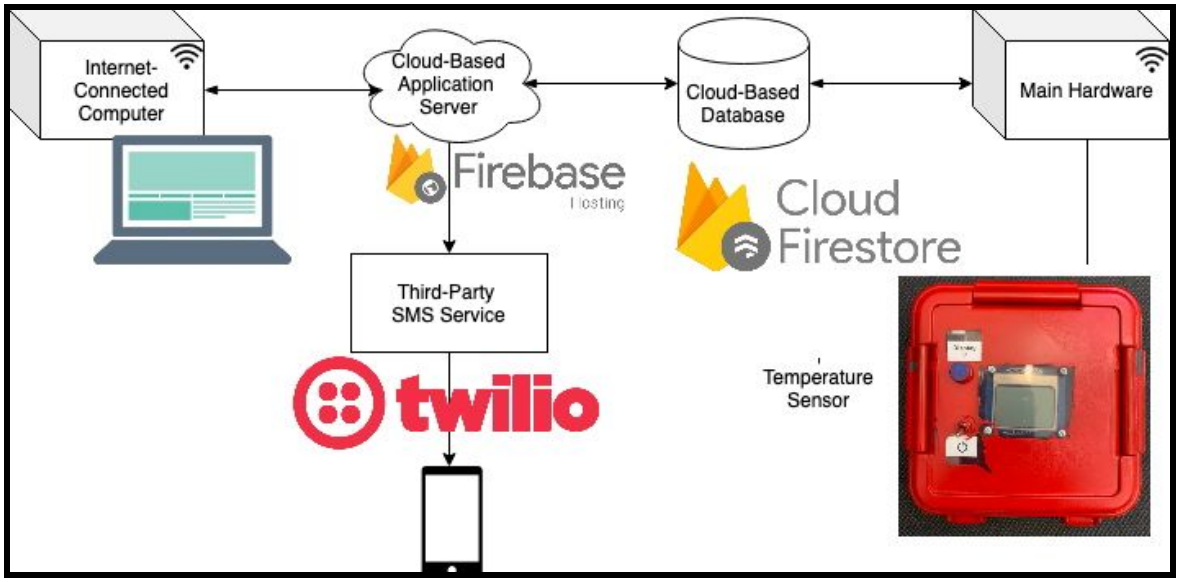
In order to accommodate the need for real time updates, utilize online hosting, and find a way to interact with a small scale database from multiple devices, we determined after research that Firebase would give us an all in one solution. Because our data set had no requirements for relations, Firebase’s “Firestore” NoSQL solution was perfect. They have an easy to use library in a wide variety of languages, so we were confident we’d be able to interact with our data easily, using similar logic on both the frontend to read data, and on the device itself for publishing sensor data. Additionally, Firebase provides methods to “subscribe” to new data on the web interface, so we were able to reflect new data rapidly, without making use of manual or pre-scheduled queries. We did have to spend money to use this solution, where as something like an SQL database in AWS RDS would have been free, but the total cost was less than a dollar.

In choosing between utilizing a Raspberry Pi, Arduino board, or ATMega device, deciding to use a Raspberry Pi was an easy choice. While using an Arduino would have allowed us to use a language like C, using a Raspberry Pi gave us the freedom to use a variety of languages, including python. The sensor we purchased had an existing open source library for interfacing in Python, as did our Nokia display which made development much more

streamlined, as we had an existing platform to work with for our hardware. Additionally our group already had possession of a Raspberry Pi, it has onboard Wifi, and it allowed use to utilize version control to manage our codebase. Anything that was contributed from a team member was directly pulled down the device, with no need to compile it using another computer.

We knew that the web interface would need to update very quickly in order to appear responsive and keep up with Firebase. We choose to use “Preact”, a framework very similar to “React” but with a more streamlined virtual DOM, allowing for faster rendering. While at first using this framework was incredibly lightweight, responsive, and efficient with rendering, we realized that community component support was lacking. Since “Preact” has a smaller open source community than “React”, we had only one option for a graph component, “Preact-Charts”. This library was created with a specific use case in mind, so we had to develop patches to the library in order to fit our needs for things like set scaling and breaks in data. For later projects, we plan to use “React” in order to make use of it’s vibrant developer community and larger component packages.

System Overview



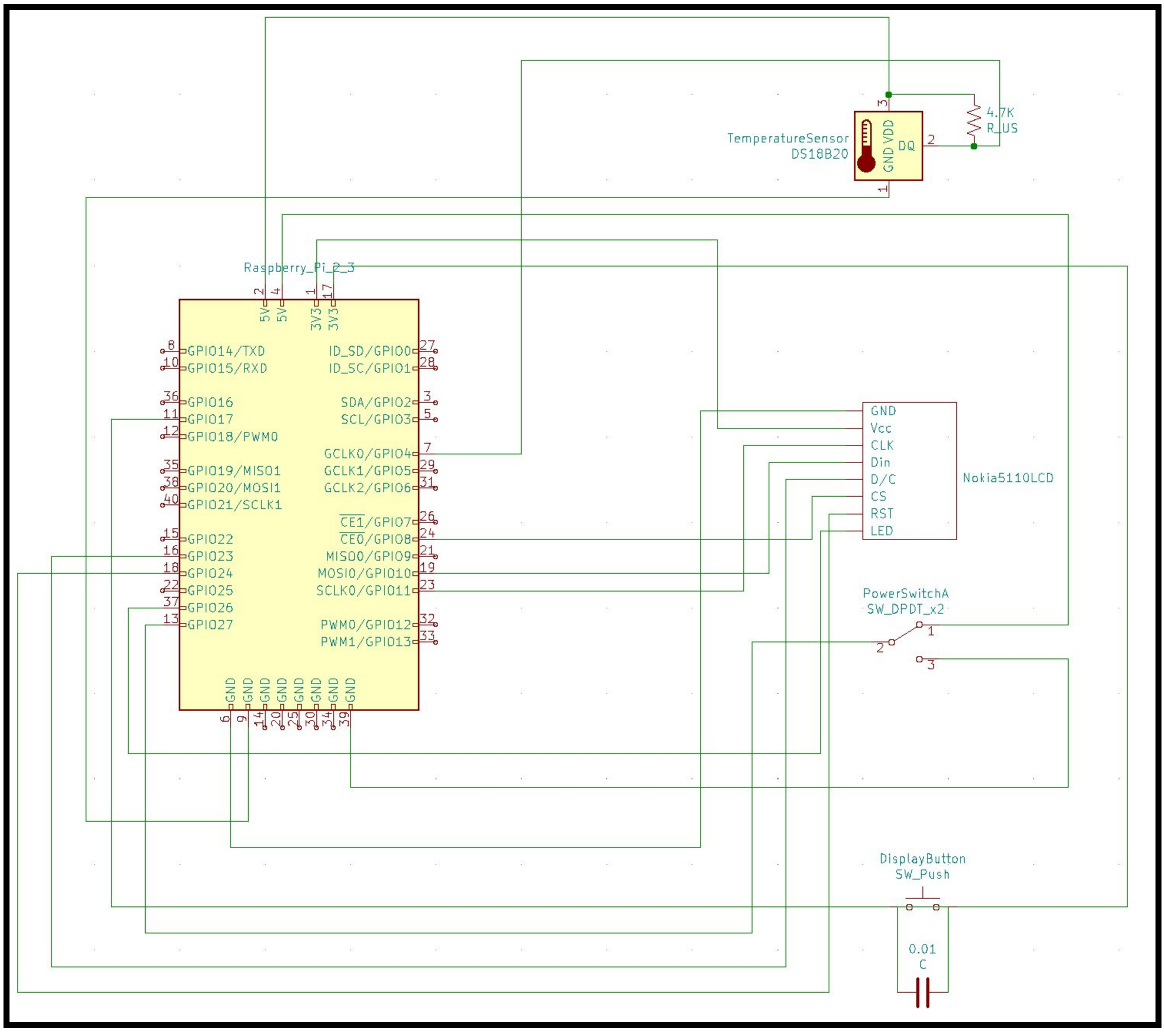
(Fig. 1: Overview of our system and its primary components)

The completed system consists of 4 major components: the main hardware, a cloud-based web application, an internet-connected computer, and a cell phone. The interaction of the components of our system is further explained below.

The main hardware gathers data from the temperature sensor when in the appropriate mode, i.e. the unit is on and the sensor is connected. These values temperature are written to a cloud-based database, which is possible as the main hardware also has an active internet connection. Our web application accesses the database, and displays the updated values in real time on the internet-connected computer.

In addition to displaying the temperature value, the web application also provides an interface for the user to enter their preferences for the system, including their phone number and threshold values for text alerts. Theses values are written to the same database that contains the temperature readings. The main hardware is able to read the values from this database, and will update its state accordingly. The main hardware is also able to connect to a third-party software to send a text message to the phone number stored in the database when the temperature crosses one of the threshold values.

Hardware Schematic



(Fig. 2: Our Hardware Schematic detailing the display, Pi, and sensor board)

Part List

- Raspberry Pi
- DS18B20 Temperature Sensor
- Nokia 5110 LCD
- SPDT Switch
- Push Button

Circuit operations functions as general microcontroller based applications do. The crux of the hardware is that the Raspberry Pi coordinates the data collection from the temperature sensor.

There are three connections for Vcc, GND, and data between the Pi and the sensor. The push button and switch are also fairly self-explanatory in that when the button is pressed, the GPIO pin that the button is connected to will read a different value as the circuit will complete with the press and a LOW value will be read as the default is HIGH (read the next paragraph for pull-up explanation). The switch works similarly as displayed by the circuit diagram above.

As pictured in the diagram, the non-IC hardware components feature pull up resistors. A 4.7k resistor was used for both the temperature sensor and the push button to connect to Vcc such that they have defined state at all times. The push button also featured a hardware debounce in the form of a 0.01 uF capacitor to prevent multiple button presses from being registered when it was pressed once. A software debounce was also implemented as the hardware one was not reliable enough.

Software on the Raspberry Pi

The Raspberry Pi utilized a python job scheduling framework known as Schedule. We decided to use this design paradigm because it allowed us to isolate the pieces of functionality into singular methods that were easier to debug, test and understand than the monolith style alternative.

The primary scheduled jobs are as follows:

- Reading the temperature
    - Determines if the sensor is plugged in or not and updates database
    - Reads temperature if connected and sends it to the database
- Sending texts
    - Determines if a text needs to be sent based on the current state of the database
    - Sends text if needed
- Syncing with firebase
    - Queries the database every second to get the current state and update local state
- Updating the display
    - If the display needs to be updated and is in a state where it should be displayed it will display either the current temperature or an unplugged message

In addition, there are two interrupt methods:

- Button press interrupt
    - Updates the local and database state to show that the button is pressed
- Switch flip interrupt
    - Updates switch state locally and in database
    - Turns off all processes until the switch is turned back on

Because all of the functionality has been so concisely defined up until this point, the main method is fairly simple. It simply runs all of the processes at their regularly scheduled intervals.

All code for the process management script is available here:
https://github.com/bmitchinson/ece4880/blob/master/Lab1/pi/main.py

---

Cell Phone Alerts

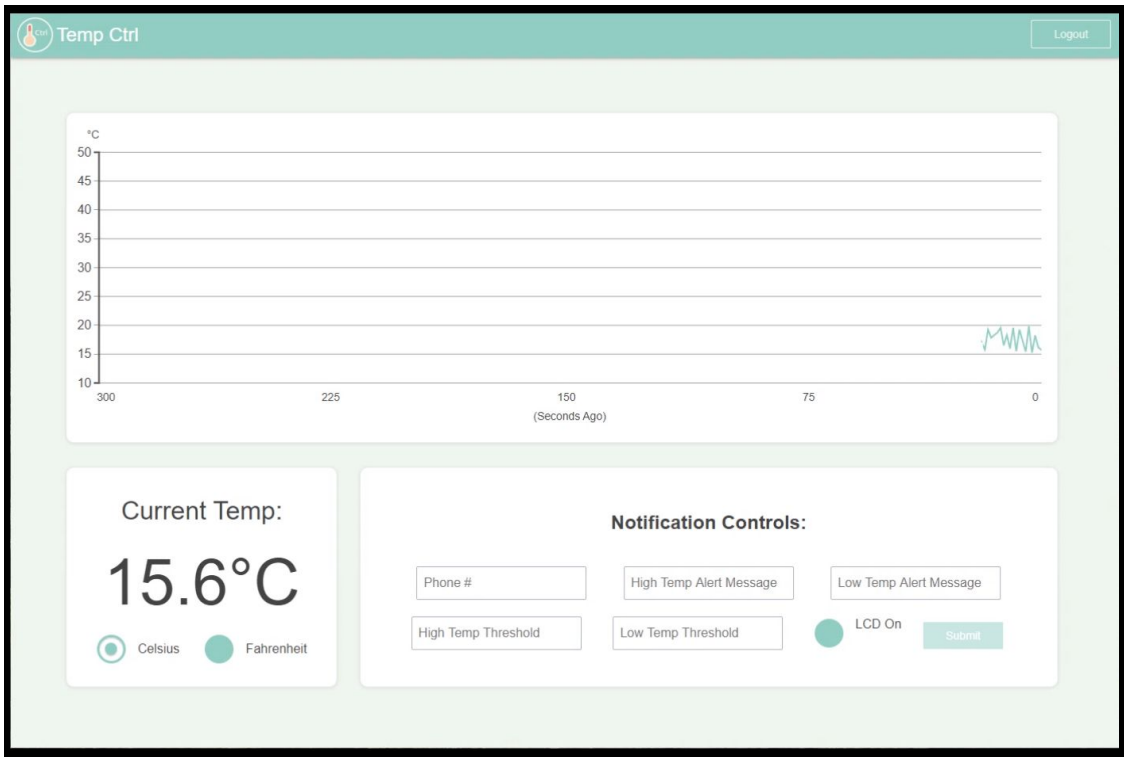
To send text messages when the temperature sensor detected values over and under defined thresholds was to use a third-party Python library from Twilio. Twilio allows users on their free trial program to send any texts from the account specified phone number to any other phone number that is verified by the account (done through an acknowledgement text pattern between the trial number and a user's phone number). The main texting method took in two inputs, that being the message content (as this can be changed from the desktop) and the recipient phone number. Rate limiting was a concern of ours, as we did not want to bombard users with texts when the temperature remained beyond either threshold, so we simply added a delay of 20 seconds in between texts.

The general pseudocode steps of operation are as follows:

- User sets high and low temperature thresholds to receive texts alongside a recipient phone number
- User sets messages to receive in either threshold pass
- User defined values on the computer are sent to the box such that both components have updated values
- Box detects that the temperature is beyond either threshold
- Box schedules to send text to defined phone number
- Text is sent and window of 20 seconds between texts starts
    - This last step repeats as necessary until it is back within normal operating range (or new thresholds are defined and the temperature is no longer outside that range)

Computer Application

The frontend of our application was constructed using Preact for fast virtual DOM render speeds, and Typescript for stricter javascript typing and debugging advantages. It is hosted publicly so that each member of the team could test their changes to the hardware while developing, and also includes added authentication so that our database remains protected while exposed online. It has a fully responsive design that scales to any desktop or mobile resolution, and is subscribed to realtime Firestore updates, so that there's almost no delay between when the temperature is measured, and when it is relayed to the user.



(Fig. 3: The single page application web interface)

The general pseudocode of operation is as follows:

- On page load render 3 components: "Graph", "Temp", and "Controls"
- Graph:
    - The graph component fetches the last 300 seconds of available temperature from Firestore

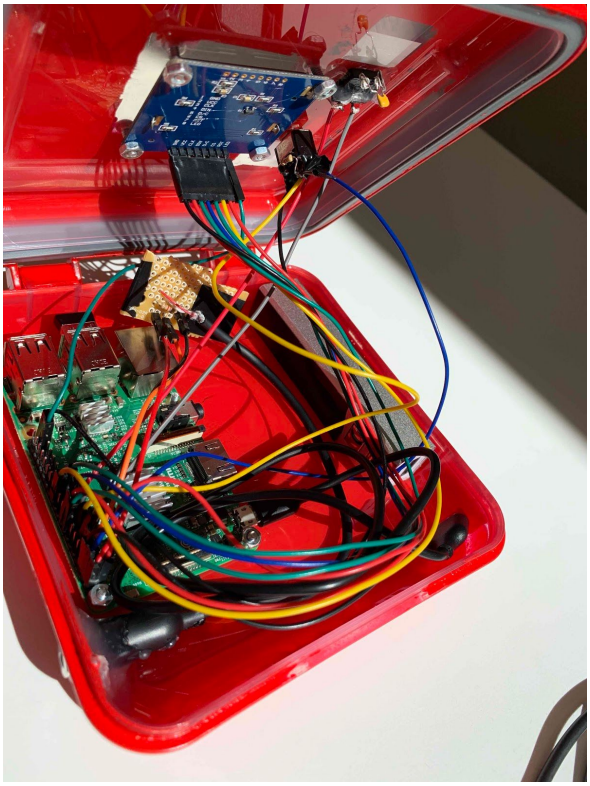
    - Graph component interprets gaps in this data, indicating a disconnect or power off, and removes gaps in order to visualize proper breaks in time
    - The graph then uses a provided firestore method to subscribe to new data, and adds it to the component state whenever it's received.
    - This dataset is handed to our modified version of "Preact-Charts", which plots all points as needed onto the dom by intelligently "diffing" it between updates with the help of preact.
    - Every second, if firebase has not delivered any recent data, we plot a break in data in order to keep the graph "scrolling" to the left side of the screen as required.
- Temp:
    - Upon load, Temp fetches the most recent temperature from Firestore, as well as the button, switch, and connected status variables to determine if the temperature should be displayed or not.
    - If any of the status variables are enabled, the component updates it's text to display the proper message to the user indicating the devices state.
    - Temp subscribes to changes in any of the 3 status variables, as well as the collection of temperatures in order to always be up to date.
    - Temp also displays a "vice-versa" button set that modifies the render to display the C scaled instead to F if enabled.
- Controls:
    - Upon load, the Controls component hides the form elements from being displayed to the user, unless they are logged in to a pre authorized google account.
    - One the user is logged in, the form fields are automatically populated with the contents that currently exist in Firestore for notification preferences and display toggling.
    - The component then subscribes to changes from Firebase so that it's form fields and display toggle accurately reflect what's taking place on the device.

All code for these three primary components is available here:
https://github.com/bmitchinson/ece4880/tree/master/Lab1/frontend/preact/components

---

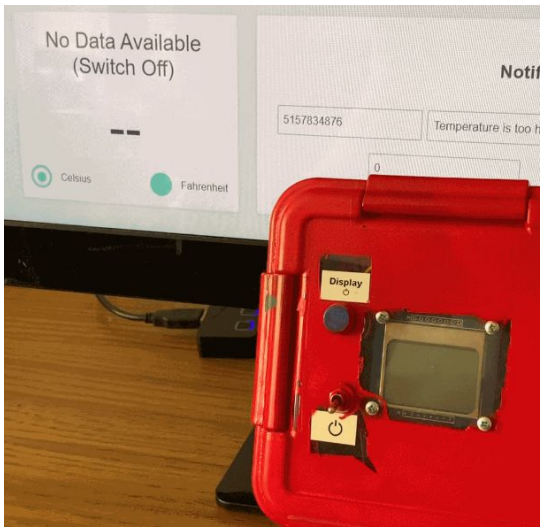
Pictures/Diagrams of System

Our headphone jack wired sensor to fulfill the requirement of a “terminal connector”



The internals of our “Third Box”, the Pi, phone battery, and a look at the mounted display and controls

The front of our “Third Box”



Our web interface reflecting switch state

Project Retrospective

Team Member Contributions

Team Member	Completed/Assigned Tasks

Alexander Powers	- Hardware:
	  - Spray painted the final product
	  - Assisted with debugging and problem-solving
	- Software:
	  - Scheduling of tasks on the Raspberry Pi
	  - Networking components of the Raspberry Pi
	  - Software integration of LCD and temperature sensor
	- Documentation

Ben Michinson	- Frontend:
	  - Created the site using Preact and Typescript
	  - Managed hosting / live deployment
	- Software:
	  - Initialized the Firebase Firestore
	  - Software integration of publishing to Firestore

Meghan McLaughlin	- Hardware:
	  - Wiring LCD and Temperature sensor to Raspberry Pi
	  - Soldering hardware components, adding 3.5 mm audio cable connection to the temperature sensor/pi interface
	- Software
	  - Assistance with Raspberry Pi program logic
	  - Assistance with debugging and problem-solving
	- Documentation

Zain Khan	- Hardware:
	  - Responsible for obtaining parts
	  - Wiring/soldering of peripherals (buttons/sensor)
	- Software:
	  - Text message functionality through Twilio
	- Documentation

Degree of Success

We consider our implementation of this project and fulfillment of the project spec greatly successful. All of the required design features were met, as referenced in our Test Report, and our check off experience also reflected this sentiment. There is no part of the project we thought was lackluster, and both hardware and software components of this project were well executed despite having a team of only CSE's. As discussed in the next section, the only portion of the project we felt was not held at as high of a standard as the rest was the time measurement between pressing the button and having the screen show the temperature. While we felt this wasn't lacking, as the display reacted quickly, we did not sit down and comprehensively make 20 millisecond tests as we pressed the button.

A concern that our group had before checkoff was that we did not have the flexibility to put in any phone number as a recipient to our alert texts. The only phone numbers that were applicable were those that our Twilio trial account had verified. Although we could upgrade our account to a premium version to have this feature in our project, this was expensive and we would have rather avoided it. Thankfully, we were informed that not all phone numbers were necessary to be contacted and that our existing design worked as intended.

Less than Successful Factors

Robust timing tests were not implemented in regards to the responsiveness of the display. The 20 millisecond threshold was generalized into seeing if the display would show up noticeably fast enough after the button was pressed. During check off this was not mentioned to be a problem, and all of our group members agreed that the display responsiveness was apt, however it was the one area where we were not strict in our adherence to the requirements. For future projects, our team intends to be more deliberate about our timing by making measurements.

Test Report

Requirement	Test Process	Test Result	Test Outcome
Physical Requirements	See if necessary components are there (computer for UI, phone capable of receiving texts, box containing display, button, battery, and	All items are included as expected, connections are terminating, and connections to the box are easy to plug in and remove.	Pass

	power switch, and a thermometer sensor) and that all connections to the box have terminating connections mounted to the box. The connections should be easy to connect/disconnect.		
Sensor Length	Used measuring tape to measure the tip of the sensor to the end of the headphone jack connector.	1.95 meters long. Within 2.0 +- 0.1 m range.	Pass
Ice Water Capabilities	Placed sensor in ice water mixture for ~30 seconds and tested functionality during/after.	Temperature sensor worked as intended during/after. The temperature sensor boasts an acceptable range from -55 C to 125 C so this was not a concern.	Pass
Box is Robust/Enclosed	Dropped from table height (~1 m off the ground) and held box upside down to see if functionality is still there after.	Parts maintained their connection with one another and no damage was sustained as the enclosure was sturdy and parts were mounted to it appropriately. When held upside down, the box functioned normally as well (all switches and buttons worked as expected).	Pass
Plug in/Remove Sensor and see if Box Resumes Normal Operation	Removed sensor and plugged it back in repeatedly and monitored box functionality after transitioning states.	The box resumed functionality without the need for human intervention to start it again (as the graph resumed plotting data and the temperature sensor collected	Pass

		information again, we verified this was true).	
On/Off Switch Functionality	Turned off the switch to see if the systems functionality turns off as desired.	Thermometer system cannot display the temperature on the box and the computer display also does not show the temperature/continue plotting data.	Pass
Button Press Activating Display	When the button is pressed, the correct temperature should appear on the display (in C) with no appreciable delay.	Button triggers the display to turn on as expected. There were no issues mentioned during check off about delay in display activation.	Pass
Display Readability	Display should be readable when it is turned on and the temperature being displayed should be correct within the range of operation of the device.	The display was able to be read when the button was pressed in normal lighting conditions. The temperature being displayed was also correct (as verified by other temperature measuring devices).	Pass
Momentary Button Contact Triggering Display	Test if button triggers display quickly. When button is let go, display should turn dark with no noticeable delay.	Works as intended and the display is both turned on and off with button pressure	Pass
Sensor Not Plugged In Display Reaction	When sensor is not plugged in, the display should notify the user that this is an issue.	Display reads a “Disconnected” error when the sensor is not connected.	Pass
User Interface Degree Requirements	Real time temperature should update when temperature changes (once a second), and the degrees unit should be interchangeable between F and C.	Temperature readings on the UI updates once a second when sensor readings change and the degree units are interchangeable on the UI (see diagram above	Pass

		for an explanation as to how this works) successfully.	
Sensor Not Plugged In UI Reaction	When sensor is not plugged in, the UI should inform the user that the sensor is not plugged in instead of displaying the temperature.	When the sensor is disconnected, a message informing the user that the sensor is disconnected appears in place of the usual temperature.	Pass
No Data Available UI Reaction	When the box is turned off, a new message stating that no data is available should show up in place of where the temperature would normally appear.	In place of where the temperature is reported, a new message stating that no data is available shows up when the box was turned off. When tested with the above condition (sensor was not plugged in), we made it so this takes precedence and that the no data available message appears when the box is both turned off and the sensor is unplugged.	Pass
Turn On/Off Box Display from Computer	Pressing the on/off button on the UI.	We achieve the same functionality as physically turning the switch on and off with the virtual button on the UI (display turns off and the temperature area on the UI mentions that data is not being collected). The response is within one second.	Pass
Graph Scale Requirements	Having the sensor plugged in with the box “on” should display a graph with specific features.	Graph shows data collected from the past 300 seconds, turns on within 10 seconds of starting the website	Pass

		from the computer, starts in degrees C, has a range of 50 C and 10 C, and remains in C despite the toggling of the units of temperature (C or F). The x-axis displays the time since the temperature value was recorded and is labeled as “Seconds ago From the Current Time”.	
Graph Reaction To Missing Data	Collect data for 10 seconds and then remove the sensor for the next 10 seconds then resume collecting data.	When the data was not being collected by the sensor, there was a gap for the missing data on the graph display. The graph continued to scroll during this time. When the sensor was connected again, the real time display of data continued.	Pass
Graph Scrollability	Collect data for a period of time (> 30 seconds) and see how the plotting changes.	When data is collected for this long, the graph scrolls horizontally from right to left. When the time exceeds 300 seconds, older values scroll off of the graph.	Pass
Graph Scalability	Manipulating the browser size with the mouse.	All UI components (including graph) rearrange and scale to fit on the resized page.	Pass
Graph Response when Box is Turned On	Turning the box on after the box has been off and collecting data for a short period of time.	Graph responds to the data collection when the box is turned on near instantaneously (within 2 seconds). This falls in range of the 10 second window.	Pass

Text Message Triggered by Set Thresholds	Make the temperature exceed or go below user defined thresholds on the UI (reference the image above for how this can be toggled).	A text message is sent to the specified phone number in both situations (exceeding and going below the thresholds) with the respective exceeding or going below message.	Pass
Text Message Content is Adjustable	Changing the exceed threshold message on the UI while setting the max temp to a few degrees C above room temperature and holding the probe in hand.	When the temperature exceeds the threshold, as expected, the new message that was input into the UI is delivered to the recipient's phone number.	Pass
Holding Sensor in Hand Raises Temperature	Turn on the box and computer and see what happens to the displayed temperature as the sensor is held in hand.	The temperature listed increases as expected within a few seconds. Testing with a soldering iron achieved the same result (also with a more intense temperature rise).	Pass
Lab Temperature Readings Follow Expected Guidelines	Turning on the temperature sensor/box in the senior design lab.	The measured temperature fell within the expected range of 22 +- 4 C (it was roughly 24 C).	Pass
Ice Water Temperature Readings Follow Expected Guidelines	Turning on the temperature sensor/box and placing the sensor in a cup of ice water.	The measured temperature fell within the expected range of 0 +- 2 C (it wavered between 1 and 2 C).	Pass