

Temperature Monitoring System

Mike Guzior, Jason Pearson, Marcel Englmaier and Justin Koehler

April 25, 2014

Contents

Abstract	2
Background	3
Design Decisions	16
Stories – Requirements	22
Stories – Functional	26
Spikes	27
Resources	31
Feasibility	32
References	33
Glossary	34
Ownership	35

Abstract

The goal of this project is to create an easy to use and low cost temperature monitoring system for anyone to use. The web end will allow the user to login in and view room statistics, as well as set warnings on thresholds. With the thresholds we would allow the user to select actions based on the threshold such as sending a text when it reaches a certain temperature. The room will contain a Raspberry Pi equipped with sensors which will use Ethernet to communicate to the web end and update information. The reason for using the Raspberry Pis is that we would be able to create a low cost sensor and be able to customize the code on the Pi as well.

Background

Our client has many needs that have been unmet for various reasons, with problems in their current situation, and our project provides solutions to those needs and provides resolutions to their problems.

Per Wikipedia, Western Michigan University's (WMU for short) Parkview campus was built in 2003 at a cost of \$72.5 million and is the home of the Western Michigan University College of Engineering and Applied Sciences (CEAS for short).

WMU's engineering website explains that WMU has state-of-the-art resources housed in a \$100 million high-tech facility. Sadly, our client has advised that this did not include any automated temperature or humidity sensors and reporting equipment in any of the rooms.

These are absolutely critical in rooms that maintain computer, technology, manufacturing, and scientific equipment to safeguard the investment and resources of the university. There are many risks that computer equipment face as they spend their entire life conducting electricity and being made of rust-prone metals. Standard servers are recommended to be kept at an average temperature of 22C or less, with automatic shut-off or critical shutdown temperature maximums of 35C. They must also be kept dry as any condensate will not only short any circuit boards it touches, but cause the servers themselves to rust, as well as the metal racks that support them. Aside from rusts and shorts, excess humidity is a cause of molding and mildewing which is unhealthy for

personnel and students, and also damages hardware and clogs air filters. There are many factors that provoke the need for this monitoring, from the downtime of a website, to the security networks that safeguard a campus, the need to have digital phones online, to safeguard data that would be lost in a failure, to the overall cost of the hardware. As an example, one server cluster with the moniker "Thor" has a hardware value of \$400,000 which would result in an excessive loss for the university if it were damaged.

Our client informed us that there have been several incidences where the temperature of servers increased unhindered to the point that equipment was destroyed due to this lack of automated environment reporting. One such incident where the temperature increased without staff knowing resulted in a \$500,000 loss. A previous loss due to humidity occurred when the humidity rose to the point of condensation and large steel papermaking rolls generated a layer of surface rust, rendering it unusable until it was repaired or replaced, causing monetary damages and downtime.

Since the fateful incident, WMU has had students implement several forms of reporting, and currently uses the Temperature @lert WiFi Edition to keep track of the temperature of rooms around campus. These sensors work very well but their major flaw is that they are very expensive. These sensors cost upwards of three hundred dollars per sensor and have many features that are neat, but unnecessary for our purposes. To alleviate this problem we proposed to create a server that would communicate with a network of home brewed, while reliable

sensor computers. The server was created by another Western Michigan Computer Science senior design team and it currently is used to communicate with the @lert sensors.

We are unaware at this time of some specific information regarding the facility such as the type and rating of their heating, ventilation, and air conditioning systems (HVAC for short), the dollar value of the equipment lost in the past, the British Thermal Units (BTUs for short) that are generated by this equipment, or how fast the temperature would increase in the rare event of an HVAC malfunction, but it is clear that their need for automated reporting, and our solution will be more than adequate regardless of this information. Our client has provided us with basic information that due to the thermal mass of the equipment in the rooms, a notification within several minutes would be more than adequate to prevent damage. As our prototype currently stands, there is roughly up to a 3 minute delay before a notification would be sent due to a sixty second temperature fetch cycle from the raspberry pi, a sixty second fetch cycle by the web server, and a sixty second processing cycle that generates the web pages, generates reports, processes data to the database, and would send an alert if the circumstances arose. This could easily be reduced to a total of sixty seconds for the whole process, and very well may be user-selectable on the web page at our client's request. It will increase autonomy, provide reporting, reduce cost, add better functionality, and provide a product that can be used by students and administrators alike.

The currently used Temperature @lert WiFi sensor

has accuracy of ± 0.5 °C. The max and minimum temperatures that the current sensor will calculate are -10 °C and +85 °C. The current sensor also gives humidity readings. This is not high priority, but if we are able to implement it that would be desirable. The humidity readings that the current sensors give are between 10% and 90% relative humidity. This relative humidity has $\pm 3\%$ relative humidity accuracy. One major feature of the sensor is the fact that it can be used over the network using wired or wireless connections. The wireless specs that it abides by are the 802.11b/g standards and allow for WPA/WEP security. These are the features that are used by the system that we need to implement on our client devices.

The website that we inherited from the previous team initial page looks like the following figure. On this page it shows a graph of all the temperatures that are currently being tracked.

It does not currently do anything with humidity. We will be evolving the site to include this information in future releases.

This uses a framework that allows easy data viewing and little coding.

We will probably use the same framework to implement this graphing process.

Once logged in there is much more functionality for the graphing.

Project Temp.e.s.t.


CEAS Server Room Temperature Monitoring Center

[login](#)


[Home](#) [Details](#)

At a Glance

JKOEHLER_APARTMENT

 Kitchen

0 74.6

 Bedroom

0 73

click on a room name to view its graph

click on a port number to view its daily highs and lows

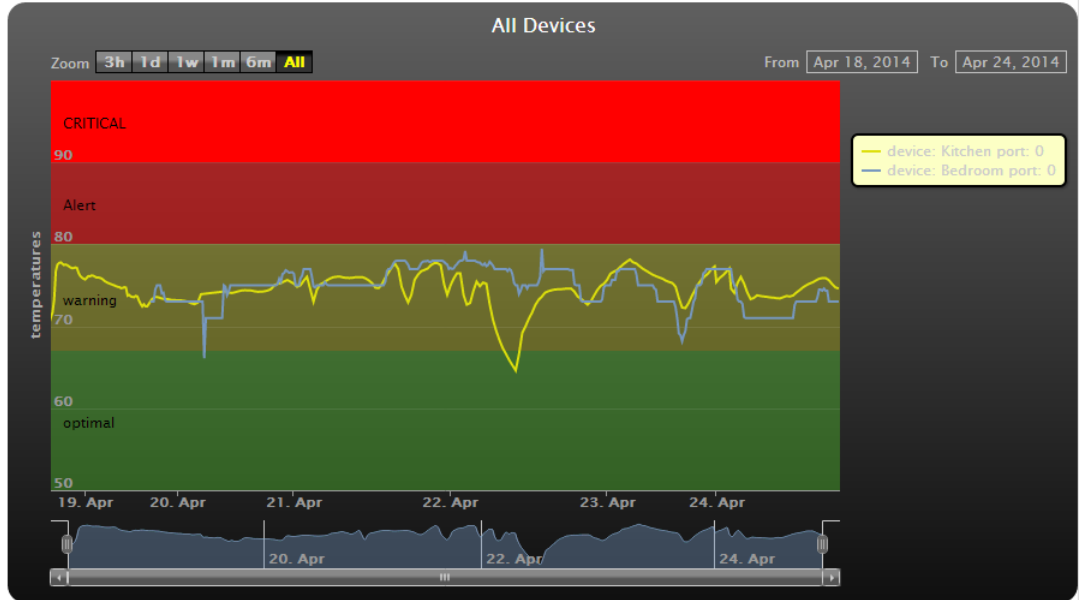


Figure 1: Initial View Of Site

The login process is very basic at this time but in future release will utilize stricter security, data sanitization, and input verification, and will prevent against session hijacking, network eavesdropping, cross site scripting, and brute force attacks.

At this time the page will simply have a Username and Password field (along with a submit button) which will have functionality added that disables the browser from remembering or saving this information.

Our first alpha release will be using a test database

with test users, test usernames, test passwords, and test data, so the security will not be an issue during this phase. When the user submits, the framework will reference the data to see if it matches what is in the database, and if it does, provide further access to the site, and if not, it will require the user to try again.

There will be only one login page, but based on whether the user successfully authenticates as an administrator or a simple user will determine the pages and views they have access to.

The admin will have access to the identical pages as the user, but will have an administrator functionality added to the pages, which will allow them to add new users, rooms, and devices, as well as delete or modify existing users, rooms, and devices.

A regular user will have a list of devices, whereas an administrator will see the same list but will have a button above said list that takes them to an add device page.

The list will have an edit and delete button next to each device for administrators as well. The edit and delete pages will be similar to the add page, and will be basically the same for users, devices, and rooms.

Project Temp.e.s.t.

CEAS Server Temperature Monitoring Center

Login

Username or password incorrect.

Username

Password

Login

Figure 2: Login Page

Upon logging in as a system administrator this is what the admin will see. This is the general hub for editing anything on the site. From here the admin can see rooms, users, room assignments and device types easily. The page looks just the same for a room administrator when logging into the site.

The only difference is that the room user won't have the option to edit any rooms, devices etc. If a non logged in user tries to access this page it will redirect them to the login page so that all admin data isn't available to the public.

Project Temp.e.s.t.

CEAS Server Room Temperature Monitoring Center

welcome, seeded_admin [logout](#)

[Home](#) [Details](#)

At a Glance

[ROOMS](#)

[DEVICES](#)

[USERS](#)

[ASSIGNMENTS](#)

[DEVICE TYPES](#)

Rooms

ID	Name	
7	JKoehler_Apartment	edit remove

[Add Room](#)

Devices

ID	Name	Location	IP Address	Type	Ports	Warning	Alert	Critical	Status	
7	Kitchen	7	http://team314.noip.me/xmlfeed.rb	TemperatureAlert	1	80	85	90	OK	edit remove
11	Bedroom	7	http://team314.noip.me:8080/xmlfeed.xml	TemperatureAlert	1	80	85	90	OK	edit remove

[Add Device](#)

Users

ID	Name	E-mail	Verified	Phone	Verified	Carrier	Admin	
1	seeded_admin	wmu.ceas.tempest@gmail.com	no		no		yes	verify edit remove
3	jason	jyb0577@wmich.edu	no		no		no	verify edit remove

[Add user](#)

Room Assignments

JKoehler_Apartment

[Add Assignment](#)

Device Types

ID	Name	
1	TemperatureAlert	remove
3	Sample_Data_Generator	remove
11	RaspberryPi	remove

[Add Device Type](#)

Figure 3: Add Page

The following figure is the form used when adding a new device to the network. The IP address is a major need in this form because it tells the server where to look for the xml. The alert and critical thresholds are for when to warn administrators for that room. The number of ports specifies simply the number of temperatures to

expect coming from that device.

Project Temp.e.s.t.

CEAS Server Room Temperature Monitoring Center

[Home](#) [Details](#)

At a Glance

Add New Device

Name	<input type="text"/>
Ip address	<input type="text" value="http://"/>
Warning Threshold	<input type="text" value="80"/>
Alert Threshold	<input type="text" value="85"/>
Critical Threshold	<input type="text" value="90"/>
Type	<input type="text" value="TemperatureAlert"/>
Ports	<input type="text" value="1"/>
Location	<input type="text" value="Room_A0"/>

Submit

Figure 4: Add Device To Network Page

Project Temp.e.s.t.

CEAS Server Room Temperature Monitoring Center

[Home](#) [Details](#)

At a Glance

Add New Device Type

New Device Type	<input type="text"/>
-----------------	----------------------

submit

Figure 5: Add Another Device Type

The following page is used to add a room to the monitoring system.

The main purpose of a room is to group sensors together and make it easier to distribute work among the administrators.

Project Temp.e.s.t.

CEAS Server Room Temperature Monitoring Center

The screenshot shows a web interface for 'Project Temp.e.s.t. CEAS Server Room Temperature Monitoring Center'. At the top, there are two navigation tabs: 'Home' and 'Details'. Below these is a horizontal line. On the left side, there is a sidebar with a button labeled 'At a Glance'. The main content area is titled 'Add New Room'. It contains a label 'Room Name' followed by a text input field. Below the input field is a button labeled 'submit'.

Figure 6: Adds a Room

There are three different levels of users.

The highest level of user is the main system administrator. This administrator is in control of the whole system. Their permissions include but are not limited to adding new users, adding new devices, adding new rooms etc.

The second tier of user are the normal room administrators. The room administrators are able to see the stats of the rooms they have access to and will receive alerts for those rooms.

The last layer of user is the non-registered user. This user can see the graph of data on the front page and login. It is beneficial to have it this way in case an administrator wants to quickly check things and doesn't want to bother

with logging in.

Below shows the form for adding a new user. The required fields are name, email, and password. The password does have minimal requirements for good passwords.

Project Temp.e.s.t.
CEAS Server Room Temperature Monitoring Center

Home Details

At a Glance

Add New User

Name:

E-mail:

Phone #: seeded_admin

Carrier: Please select one ▼

Password:

Confirm Password:

is admin? ☐

submit

Figure 7: Page To Add Users

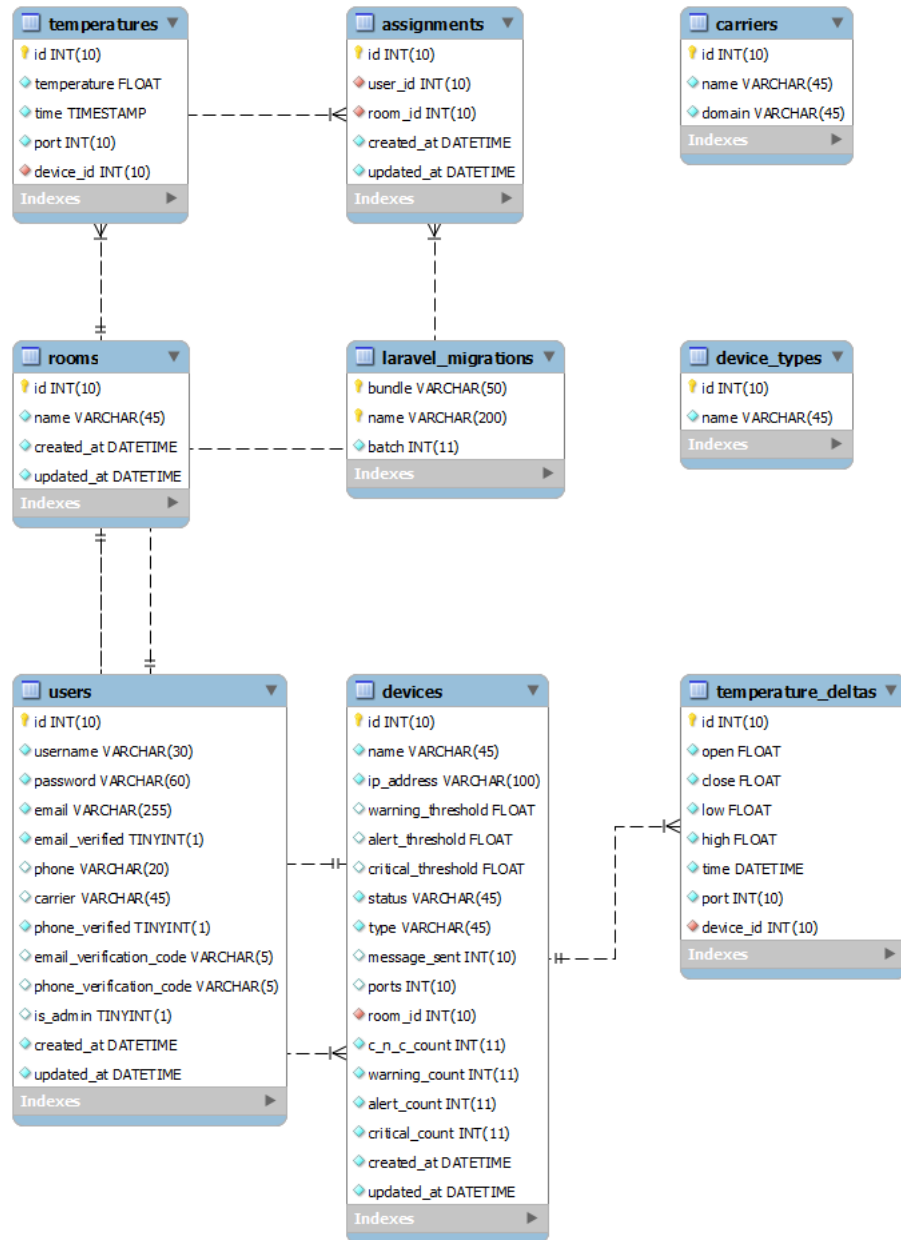


Figure 8: Entity Relationship Diagram

Design Decisions

- Raspberry Pi: We decided to use the to connect to the temperature and humidity sensor because it is cost efficient and is able to send and receive data from the monitoring server. Other alternatives we looked at are the Arduino and the MSP430, but we ended up deciding on the raspberry pi because there was more documentation and had all the required hardware in one bundle. Also Raspberry Pis require less hardware configuration allowing a end user with less hardware skills to still be able to utilize our project.
- Git: Team members were more familiar with the workings of Git and could be used with a GUI.
- Sensor: For our case we will be using a humidity and temperature sensor that we found on Adafruit's website. The sensor is called the DHT11 basic temperature-humidity sensor. For easy reference it is currently product number 386. Although this sensor is cheap it has enough power to determine the temperature and humidity of a room easily. Several limitations that the DHT11 has is it can only poll the temperature every 3-5 seconds and has a resolution of 1°C with an accuracy of $\pm 2^{\circ}\text{C}$. Figure below is the DHT11.

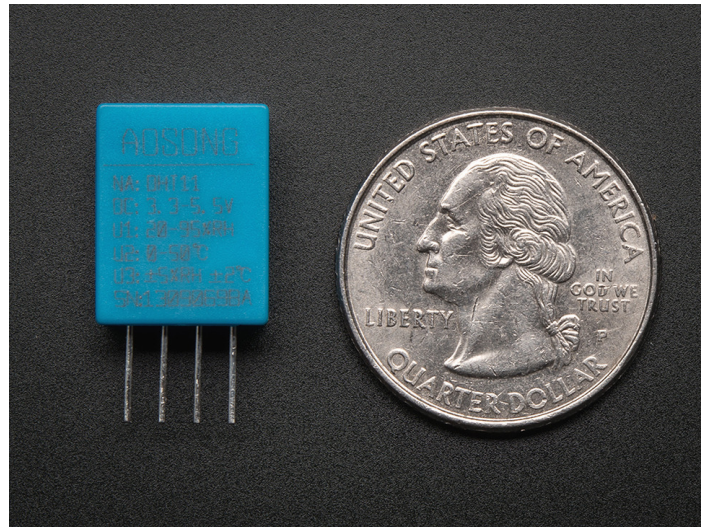


Figure 9: Temperature Sensing DHT11 Unit.

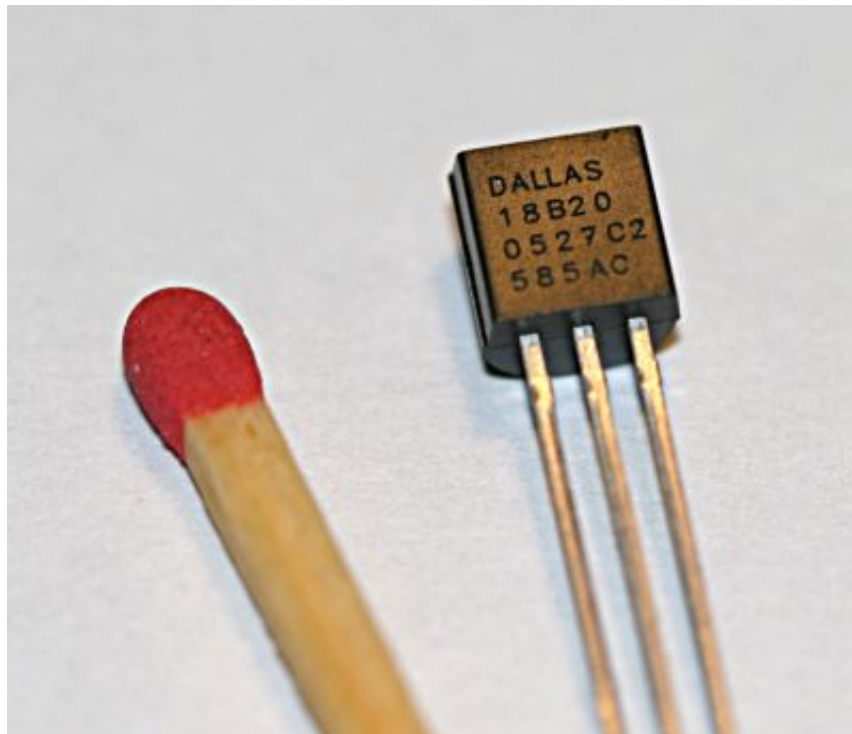


Figure 10: A DS18B20 sensor for size comparison.

With the limitations previously mentioned, we will be exploring other sensor alternatives. There are many others that vary in price, accuracy, range, and resolution, and those we have looked into are in the chart below. The sensor that the Temperature @lert device that WMU provided us uses the DS18B20 sensor, which has a comparable cost and much greater resolution. Our team is currently working on modifying an existing driver or building our own for use with our Raspberry Pi. The DS18B20 has a low cost to begin with, but can be bought in bulk for further reduced prices. Due to their small size, it would be extremely quick and easy to bind/wrap either the DHT11, DHT22, or DS18B20 into a cable that would have an RJ-45 connector on the end, whereas it would be just as easy to have our custom 3d-printed RaspberryPi case have an RJ-45 jack so there would be no direct wiring/soldering to the RaspberryPi, and the sensors could be removed on a whim for storage, transport, or replacement should a single sensor go bad. The DHT11 and DHT22 have one great advantage over the DS18B20 in that the latter does not contain a humidity sensor, hence the smaller package.

Name	Lowest Cost per unit	Maximum Resolution	Maximum Accuracy	Range	Analog or Digital	Notes
DS18B20	\$1.60	0.0625°C	±0.5°C	-55°C to +125°C	Digital	
DHT11	\$1.01	1.00°C	±0.5°C	0°C to +50°C	Digital	Needs pullup resistor
DHT22	\$6.80	0.10°C	±0.5°C	-40°C to +80°C	Digital	Needs pullup resistor
AM2302	\$15.00	0.10°C	±0.5°C	-40°C to +80°C	Digital	Built-in resistor
TMP36	\$1.50	0.50°C	±1.0°C	-40°C to +125°C	Analog	
MLX90614	\$15.95	0.50°C	±0.5°C	-40°C to +125°C	Analog	
MCP9808	\$4.95	0.0625°C	±0.25°C	-40°C to +125°C	Digital	
MAX31855	\$14.95	0.25°C	±2.0°C	-200°C to +1350°C	Analog	Thermocouple

Figure 11: Alternative Sensors to DHT11

- Laravel: Although our team lacks experience with the Laravel system we decided that using the old team's code would be worth the steep learning curve of the Laravel system.
- Cake PHP: Looking at Cake PHP we determined that though this is a very great framework the existing code should be used.

We have also decided to separate the project into 4 main functional pieces which each need their required spikes. The flow of the normal communication is as displayed as below.

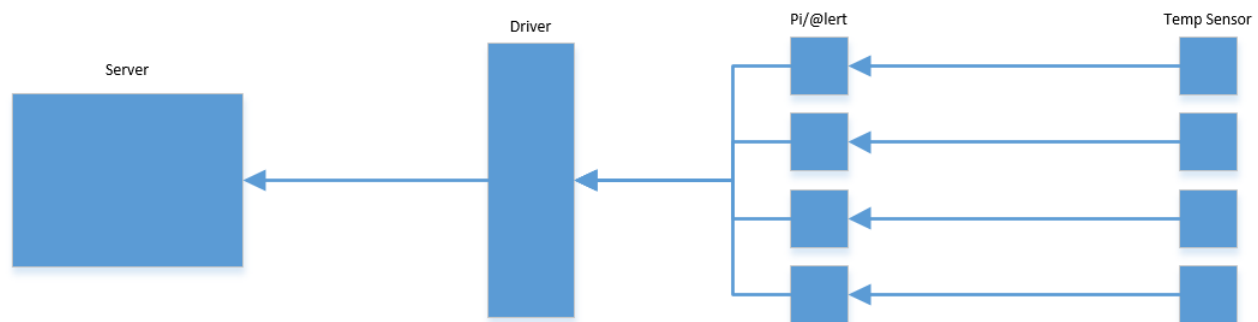


Figure 12: How the Data Goes Through The System

The first part in the process is getting the temperature from the device. For the WiFi @lert system this has been already done by the manufacture. As for the Pi we have a program that will read the temperature and output it to standard output. We have two options we can either create our own custom version of the code the company supplied, or we can create a program that uses their code to get the temperature. Both of these options are viable and both equally easy to do. However, both will need their respective spikes.

The next step is to create a way for the devices to communicate with the server.

Currently the @lert system does this by hosting an xml document that the server retrieves.

Our current prototype of the raspberry pi has duplicated this functionality and as a result does not actually require a separate parser.

The two options that we are looking at for enabling the hosting of the xml document are either apache or Nginx.

Another option is to not use a hosting program and to use a program that will give data to any requesting process.

We will have spikes for all options and determine which will be the best for the low memory that a raspberry pi has.

The third step is for the server to decode what data was received. This done by the parser factory PHP task. What the parser factory does is it takes in a type and an IP address. Then it does a switch statement based on the type and returns an object that contains the correct

methods for that reader type.

For our Raspberry Pi module we will need to make an object to parse its data and an entry in the ParserFactory to detect the new object. The object for the Raspberry Pi module will need to use the ITemperature Parser PHP interface in order to be used correctly.

In theory this can all be done using the GUI that was created by the previous team, but both the GUI and the manual method should be tried.

The final piece in the data retrieval process is the server parsing and using the data. If we follow the guidelines that were established by the previous groups temperature sensing unit the server should be able to treat them both the same with their respective data. We will be using the other groups code for graphing, mygraph.js. In this it reads in all the data for the devices. Our device will simply be put in with the rest of the devices and report itself with the other devices.

Stories – Requirements

This project requires a server running Linux, and in our case we will be using Ubuntu on this system. The project will also require a web server that runs any 2.x version of Apache, which according to W3Techs is used by 58.70% of all websites as of March 30th, 2014. The choice to utilize Apache 2.x-compliant technologies was made with the idea that the majority of web servers utilize this technology, so it will be accessible to the majority. We also require a temperature sensor of your choice. We will be using the DHT11 and our tutorials will be using this part. We will be interfacing with two devices at first, beginning with the Temperature @lert WiFi edition TM-WIFI220, seen here:



Figure 13: @lert WiFi edition TM-WIFI220

which has an initial cost of US \$300.00 +tax/shipping, a size footprint of H: 1.25, W: 4.00, L: 6.00, an overall

volume (excluding antennae) of 30 cubic inches, supports PoE, has a yearly estimated power usage of \$7.77 (based on 24 hour use @ \$0.1348 per kilowatt hour, and as is not able to be physically upgraded or added on to outside of its two available temperature and humidity sensors.

We are also designing our own sensing device that utilizes Raspberry Pis running Raspbian, with a generic case it would appear as the following:



Figure 14: Example of Raspberry Pi

which has an initial cost of US \$61.47, a size footprint of H: 1.00, W: 2.80, L: 3.80, an overall volume (excluding antennae) of 10.64 cubic inches, could potentially support Power over Ethernet (PoE), has a yearly estimated power usage of \$5.89 (based on 24 hour use @

\$0.1348 per kilowatt hour, and as is fully upgradeable in any capacity. Some example upgrades could be light sensors (for knowing if a windowless room has its lights on or off), motion sensors (to indicate movement in the room), video/still camera (for security monitoring), or even a touch screen (for things like giving the unit the ability to edit settings without connecting to a computer, or using as an employee time clock as an example). The possibilities are endless. If you would like Wi-Fi connectivity for your Raspberry Pi like we are going to, we will use a wireless receiver. With the wireless functionality, this requires the unit to simply be plugged in to a power outlet as no physical network connection is required. The Wi-Fi at CEAS has extremely poor signal in places due to several factors including metal anti-theft grates that inadvertently shield them, as well as the very thick concrete walls. If the client desired, a future release of the unit could include an optional network port so that it could be wireless and wired for network connectivity redundancy in the cases that the Wi-Fi signal is poor as the unit is placed in a signal dead zone, or that the Wi-Fi goes out but the LAN would still be active. Either way the system would alert the designated users if the units lost connectivity.

As the Temp @lert is \$300 per unit, and our RaspberryPi is \$60, that is a cost savings of 80%. When compared to other companies products, it is an even greater cost savings as seen here:

Company	Product	Cost per unit	Requires Base?	Base Cost / Ports	Wireless/Wired
Serverscheck.com	SensorGateway	\$235	Yes	\$225 / 8 Ports	Wired
Team314	RaspberryPi	\$60	No	-	Wireless
TemperatureAlert.com	TM-WiFi330	\$300	No	-	Wireless
Avtech.com	Room Alert 3E	\$145	No	-	Wired
ITWatchDogs.com	WatchDog 15	\$189	No	-	Wired
LaCrosseTechnology.com	La Crosse Alerts	\$100	Yes - Included	-	Wireless
TheClimate.co.uk	CM-2	\$740	No	-	Wired
TempGenius.com	TDS	\$400	Yes - Included	-	Wireless

Figure 15: A comparison of prosumer-grade sensor devices with the Team 3.14 RaspberryPi.

Our research indicates that at a very minimum, a one-rack server room would require five sensors: one at the front-bottom of the rack (or top if it was top-cooled) to monitor the intake, one at the back top of the rack to monitor the outtake, one in the center of the room, one at the end of the room furthest from the cooling units, and one next to the cooling unit. With the current temperature sensors that WMU uses, this would be a \$1500 investment, with one unit costing \$300. With our units, it would be a total cost of \$300, so you could go the route of putting ten sensors in the room to cover all possible hot zones for the cost of two of the current sensors. The possibilities are endless with the reduced cost and easy availability of the units.

In future releases, we also will implement mobile and tablet versions of the site. We have not yet begun to research any frameworks or API's for this yet but we are aware of some such as Foundation 5 which will automatically make your site look appropriate based on design standards for the different platforms.

Stories – Functional

- Use a temperature sensor to monitor the temperature of the servers room located on the CEAS campus.
- Check the humidity of the server rooms located on the CEAS campus.
- Graph the humidity and temperatures of the rooms.
- Send the data to a server to be checked.
- Use timers to make sure that communication with the servers are not lost
- Send text messages and emails to the server administrators to alert them of loss of server connection and temperatures that are higher than the recommended level.
- Establish permissions for the various administrators that will be monitoring the system.
- Create documentation so that the next group of users can use the software

Spikes

To do these stories we will be creating spikes to show that critical sections are actually plausible. After we do this we combine the spikes and add minor code to complete the system. Some of our spikes include:

Connecting the temperature sensor:

Below is our development version of the temperature sensor setup.

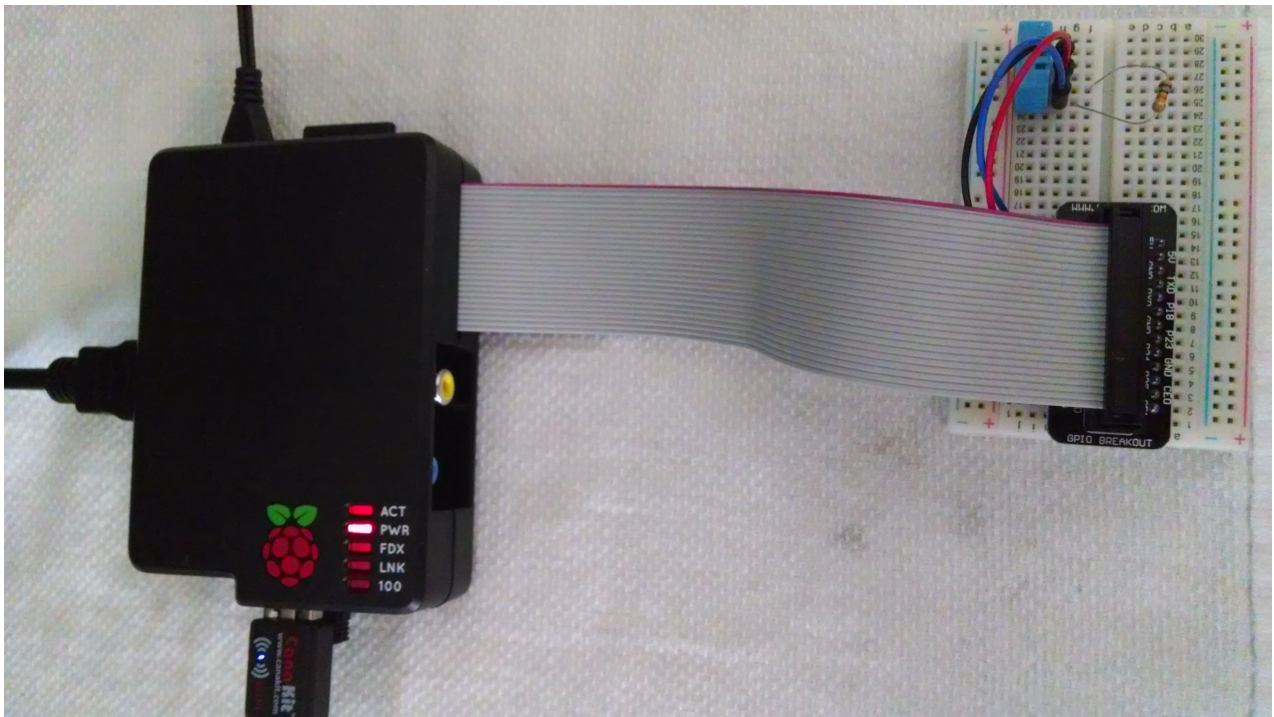


Figure 16: Current Setup For Temperature Sensor

In the future when the project leaves the development process we would like to create a much easier way to connect the sensor and make the design more sleek.

Raspberry pi temperature retrieval code:

We have completed this spike by using code from the Adafruit website. After examining this code we wrote our own driver for the temperature monitor that better fits our needs. See the figure after the wireless network spike for proof of concept for getting the temperature. The figure shows the results of running the Adafruit driver, which has a success rate of actually getting the temperature of one out of six tries. This was unacceptable to us, so our custom driver has a 100% success rate, which is not shown as it just works perfectly.

Raspberry temperature to xml:

Our code for retrieving the code off of the raspberry pi immediately writes the data to the xml code that is needed to transmit the temperature over the network.

Using Apache for hosting the xml document:

A lot of people use raspberry pi's to host websites using apache so hosting a simple XML document was very easy to accomplish by following one of the numerous guides around the internet.

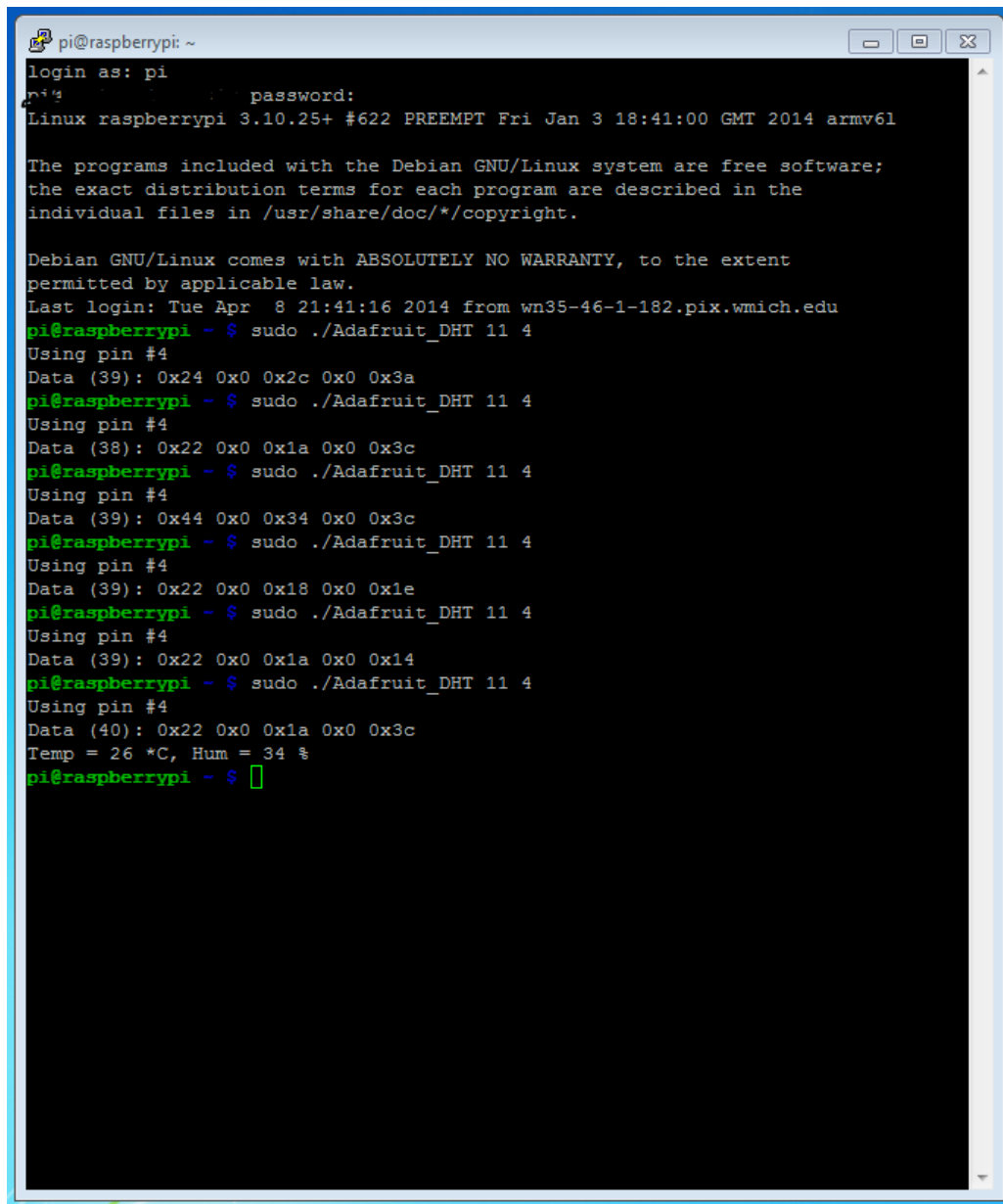
Creating a driver for the Laravel code for the pi:

Based on the Laravel code that was handed down to us it was easy to create another module for the raspberry pi.

Connecting a raspberry pi to a wireless network:

Once we plugged in the wireless adapter it seemed to work right off the bat. This was expected because it was

provided by Adafruit who makes the raspberry pi's. We also tested with other brand wireless connectors which worked equally well.

A screenshot of a terminal window titled 'pi@raspberrypi: ~'. The terminal shows a login sequence for user 'pi' with a password prompt. After login, it displays the Linux version '3.10.25+ #622 PREEMPT Fri Jan 3 18:41:00 GMT 2014 armv6l' and a message about Debian GNU/Linux being free software. It then shows the last login time and IP address. The user runs a command to run a script 'Adafruit_DHT' with arguments '11 4'. The script outputs sensor data for pin #4, including temperature and humidity, and then prints the raw data bytes for each reading. The user runs the script multiple times, and the output shows varying data values. The terminal window has a blue title bar and standard window controls (minimize, maximize, close) in the top right corner.

```
pi@raspberrypi: ~
login as: pi
pi's password:
Linux raspberrypi 3.10.25+ #622 PREEMPT Fri Jan 3 18:41:00 GMT 2014 armv6l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Apr  8 21:41:16 2014 from wn35-46-1-182.pix.wmich.edu
pi@raspberrypi ~$ sudo ./Adafruit_DHT 11 4
Using pin #4
Data (39): 0x24 0x0 0x2c 0x0 0x3a
pi@raspberrypi ~$ sudo ./Adafruit_DHT 11 4
Using pin #4
Data (38): 0x22 0x0 0x1a 0x0 0x3c
pi@raspberrypi ~$ sudo ./Adafruit_DHT 11 4
Using pin #4
Data (39): 0x44 0x0 0x34 0x0 0x3c
pi@raspberrypi ~$ sudo ./Adafruit_DHT 11 4
Using pin #4
Data (39): 0x22 0x0 0x18 0x0 0x1e
pi@raspberrypi ~$ sudo ./Adafruit_DHT 11 4
Using pin #4
Data (39): 0x22 0x0 0x1a 0x0 0x14
pi@raspberrypi ~$ sudo ./Adafruit_DHT 11 4
Using pin #4
Data (40): 0x22 0x0 0x1a 0x0 0x3c
Temp = 26 *C, Hum = 34 %
pi@raspberrypi ~$
```

Figure 17: SSH Over Wireless Network And Sensor Data Retrieval

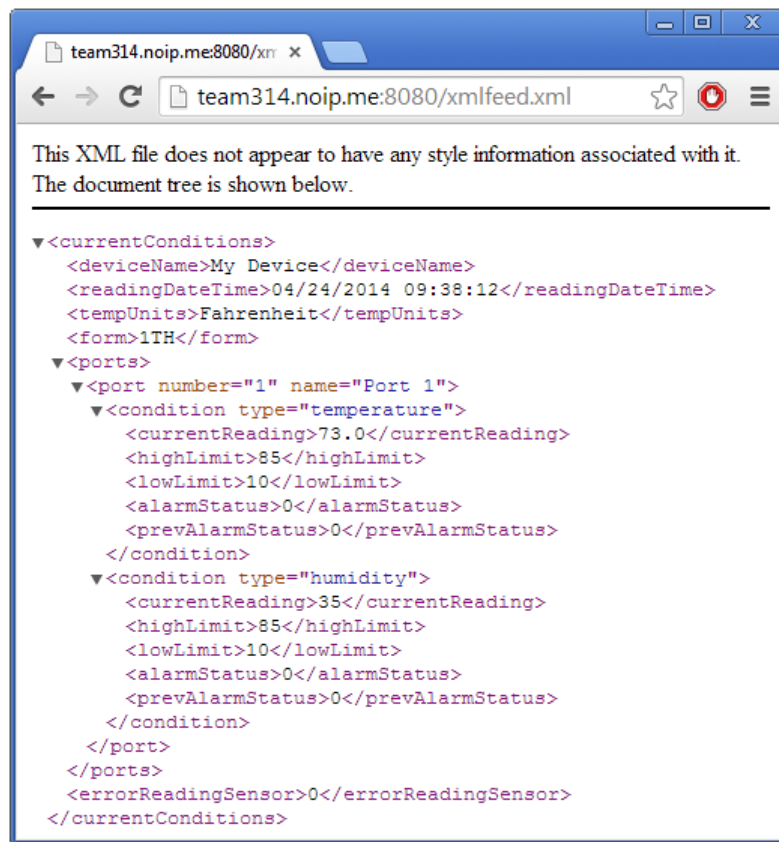


Figure 18: XML hosted on the RaspberryPi via Team 3.14's NoIP address.

Resources

- Raspberry Pi
- Temperature Sensors
- Humidity Sensors
- Web Server
- Soldering Equipment
- Operating System loaded SD Cards with Raspbian
- Power connectors for Raspberry pi
- WiFi Connectors for Raspberry pi

Feasibility

Based the cost of the hardware and the previous source code the feasibility of this project is high with many of the features being of low to mid risk. We had a very informative meeting with a member from the previous senior design team where we saw how everything works and its limitations.

Based on this meeting all our goals seem very reasonable and extra features can be implemented very easily for future releases.

The biggest problem that we will encounter is the steep learning curve with the Laravel PHP package. However, this will be overcome due to good documentation on their site and contacts with Laravel experience. Being software oriented scientists the raspberry pi at first seemed like a daunting task.

References

For everything raspberry pi we use these sites

- <http://www.raspberrypi.org/>
- <http://www.raspbian.org/>
- C Programming 2nd Edition
- <http://www.adafruit.com/>

For everything web server these are the sites we use

- <http://laravel.com/docs/quick>
- <http://www.w3schools.com/>
- <http://www.noip.com>
- <http://apache.org>
- <http://www.w3.org>

Glossary

GUI

Graphical User Interface. The windows a user interacts with.

MSP430

A 16-bit microcontroller platform made by Texas Instruments.

Raspberry Pi

A credit-card-sized single-board computer developed by the Raspberry Pi Foundation.

Arduino

A series of microcontrollers that are very commonly used for computer to real world communications.

Raspbian

A Debian based operating system that we will use for our Raspberry Pis

CEAS

College of Engineering and Applied Sciences at Western Michigan University.

PHP

A recursive acronym for PHP Hypertext Preprocessor - the web programming language being used.

Ownership

Licenses

Our project will be under several licenses. PHP is a free open source software released under the PHP License. Laravel is licensed under the MIT license and per the agreement we are free to modify, distribute and re-publish the source code on the condition that the copyright notices are left intact. In the event that our project was used to generate revenue, or be sold as a standalone software package, the license permits us to incorporate Laravel into any commercial or closed source application. The GNU license and will be open source and will be hosted for all to access and modify as they desire on GitHub.

Intellectual Property (IP)

As this project is being developed as a Senior Design project for Western Michigan University (WMU) at the direction of Dr. John Kapenga, WMU will retain the intellectual rights to the software.

Non-Disclosure Agreement (NDA)

No non-disclosure agreement is being used at this time. The project is maintained on GitHub, which is freely and openly accessible to anyone who wishes to view it, and is thus tracked by search engines such as Google, where it is able to be searched for by anyone on the planet.