Temperature Monitoring System

Mike Guzior, Jason Pearson, Marcel Englmaier and Justin Koehler

April 15, 2014

Contents

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). WMUs 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. Our client informed us that there had been an incident where the temperature of servers increased unhindered to the point that equipment was destroyed due to this lack of automated environment reporting. 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. It will increase autonomy, provide reporting, reduce cost, add better functionality, and provide a product that can be used by students and administrators a like.

The currently used Temperature @lert WiFi sensor has accuracy of $\pm.5\,^{\circ}$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. This uses a framework that allows easy data viewing and little coding. We will probably use the same framework to impliment this graphing processs. Once logged in there is much more functionallity for the graphing.

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 the 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.

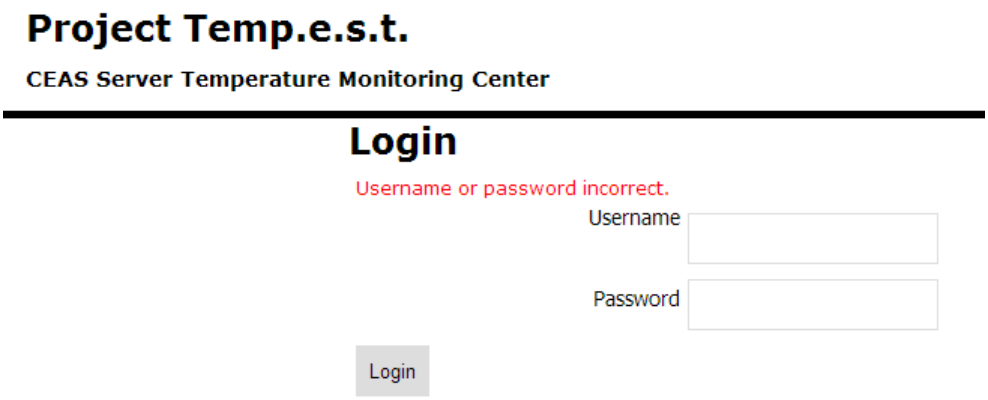

Figure 2: Login Page

The Add page will be basically the same for users, rooms, and devices. The page for users will have an empty form with fields appropriate for each user that includes UserID (in WMUs case, it would be the WIN ID) FirstName, LastName, UserName, EmailAddress, and PhoneNumber. The Email and Phone fields will have a checkbox next to them that dictates whether that is a means of communication for alerts. The password is not an editable field, as it either (to be decided later) to be either randomly generated by the server, and required to have the user change it upon first login, or there will be an email verification process that allows the user to create their own password. Similarly for rooms and devices, appropriate fields will be listed and upon verification, will be added to the database.

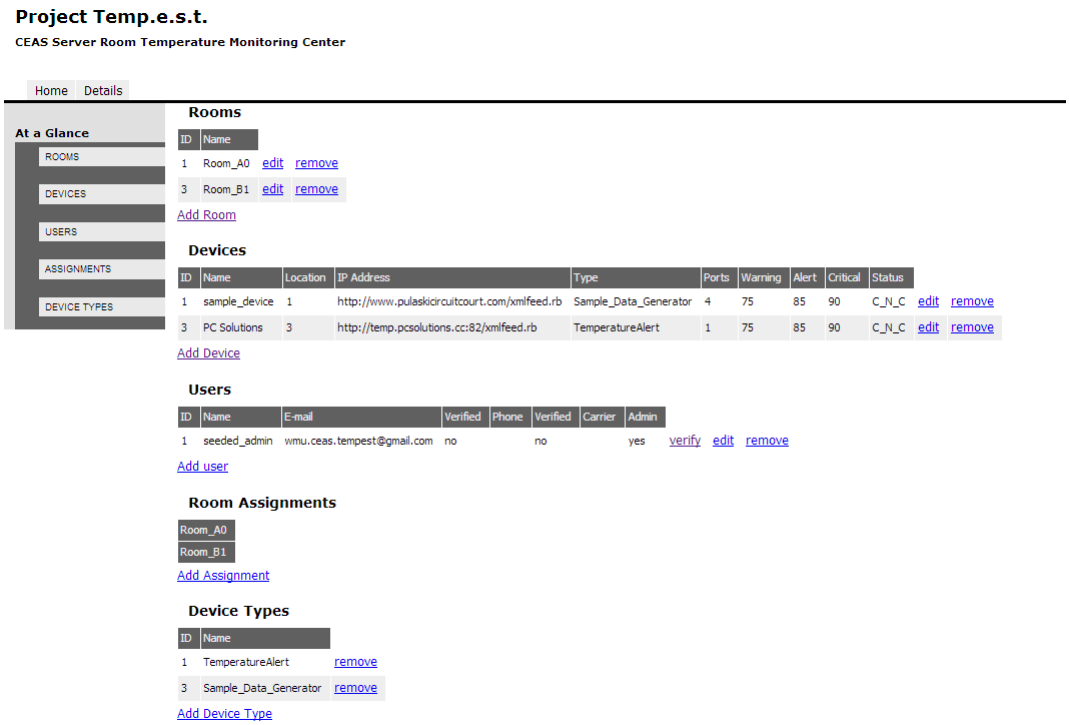

Figure 3: Add Page

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.

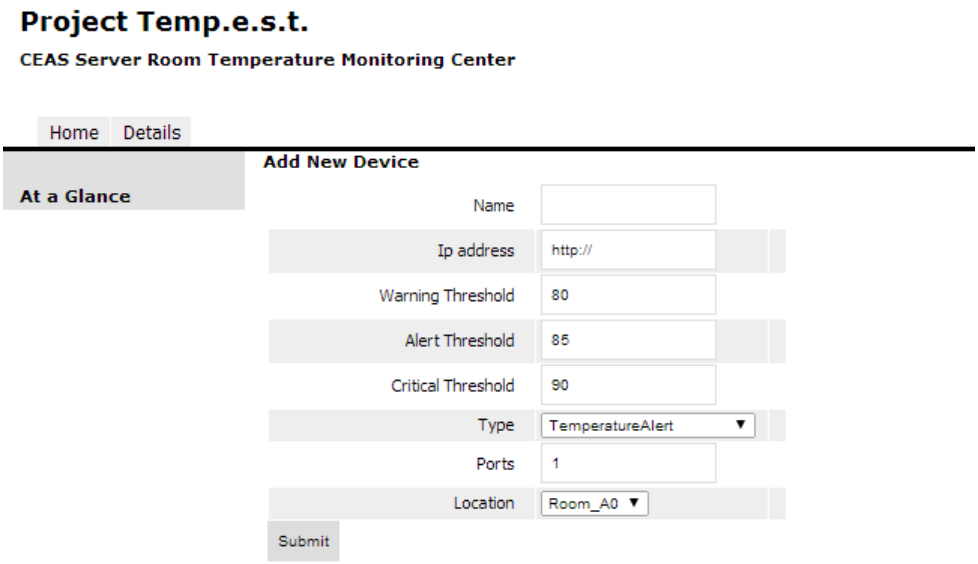

Figure 4: Add Device To Network Page

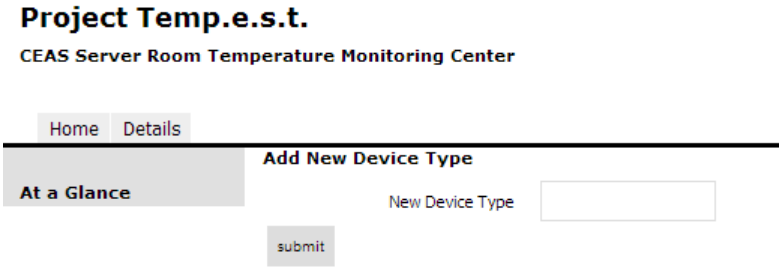

Figure 5: Add Another Device Type

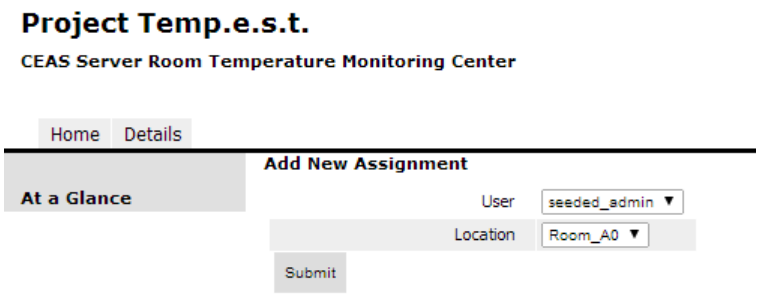

Figure 6: Page to Assign Users to Rooms

Add an admin to oversee a server room

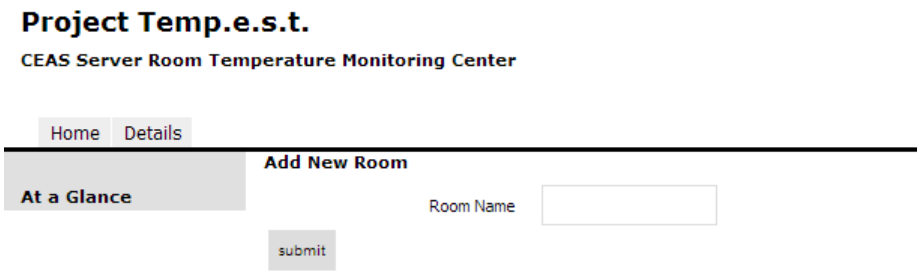

Figure 7: Adds a Room

Add a server room to the list of rooms to monitor

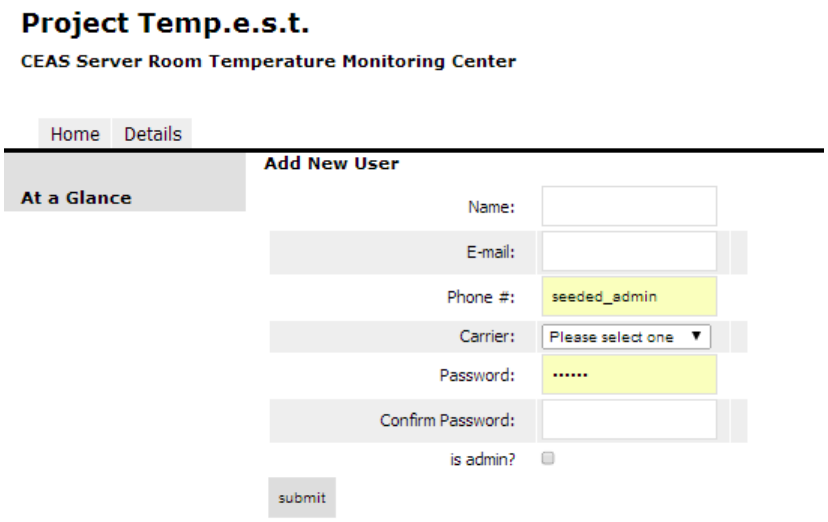

Figure 8: Page To Add Users

The page that will be displayed when adding a new user to the list of authorized admins
There are three different levels of users. A main admin will be in cotrol of the whole system. Can add and remove users and middle admins, can see the entire system.
The middle admins will be able see the stats of the rooms they have access and will receive alerts for those rooms. Users are only able to see the stats of each room.

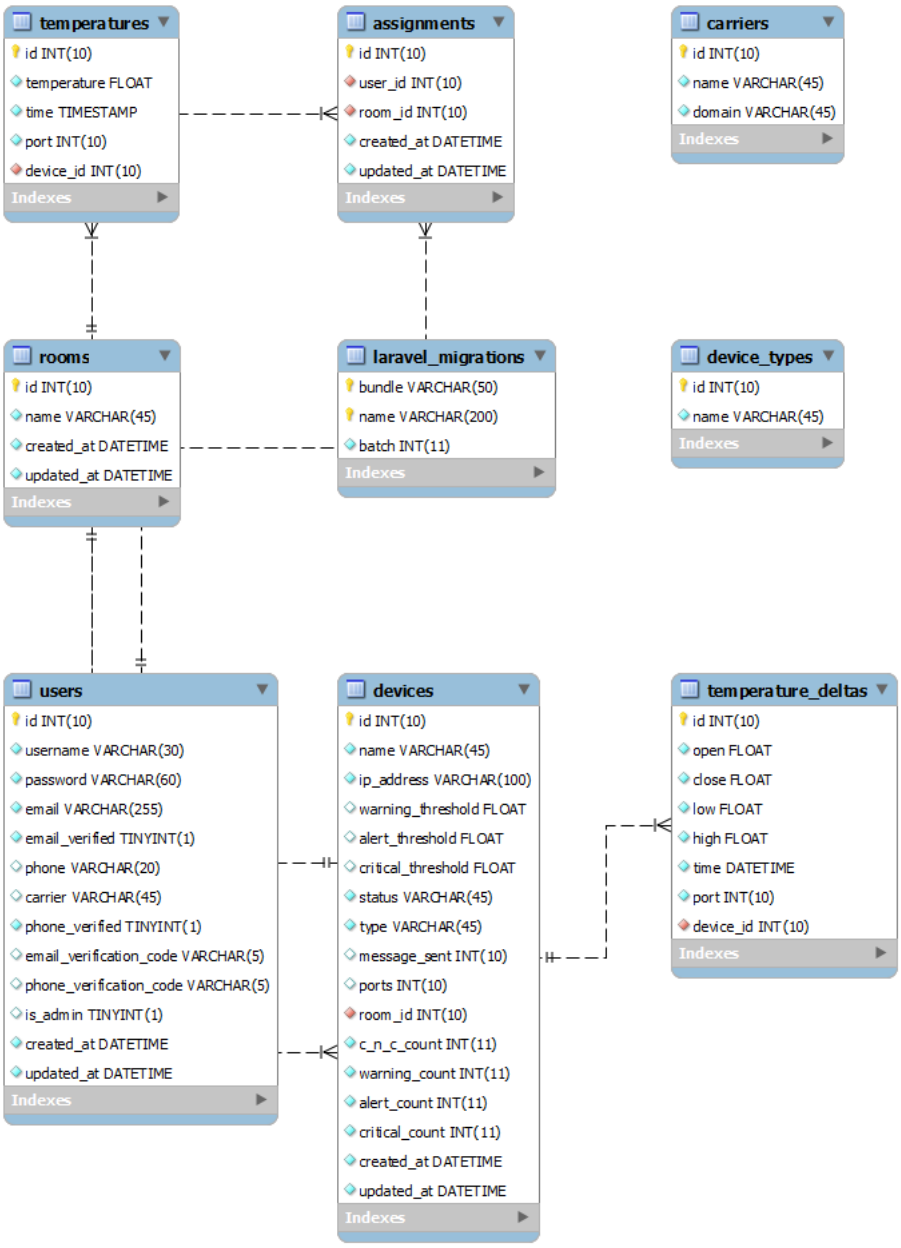

Figure 9: Entity Relationship Diagram

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. One limitation that the DHT11 has is it can only poll the temperature every two seconds. Figure below is the DHT11.

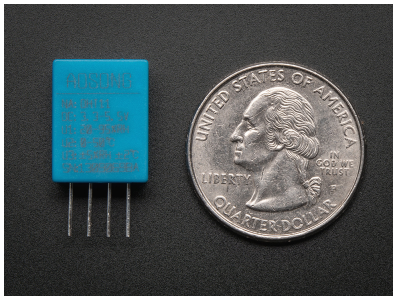

Figure 10: Temperature Sensing DHT11 Unit

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

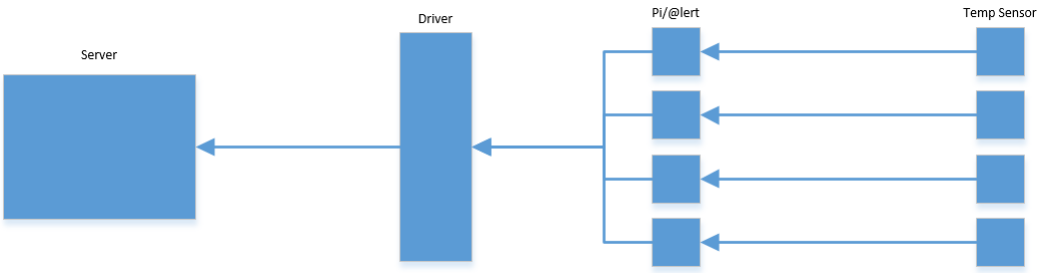

Figure 11: How the Data Goes Through The System

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. This is easily possible on the raspberry pi. The two options that we are looking at for enablig the hosting of the xml document are either apache or nginex. 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 recieved. This done by the parser factory php task. What the parser factory does is it takes in a type and an ip. 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 temeperature 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 deveices 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 interfacing with two devices at first, beginning with the Temperature @lert WiFi edition TM-WIFI220, seen here:

Figure 12: @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 rasberian, with a generic case it would appear as the following:

Figure 13: 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, supports 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 touchscreen (for things like giving the unit the ability to edit settings without connecting to a computer, or using as an employee timeclock as an example). The possibilities are endless. If you would like WiFi connectivity for your Raspberry Pi like we are going to, we will use a wireless receiver.

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.
- 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 admins 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:

- Raspberry pi temperature retrieval code
- Raspberry temperature to xml
- Sending emails to admins
- Sending text messages to admins
- Using Apache for hosting the xml document
- Using nginex for hosting the xml document
- Creating a driver for the laravell code for the pi
- Connecting a raspberry pi to a wireless network

Resources

- Raspberry Pi
- Temperature Sensors
- Humidity Sensors
- Web Server
- Soldering Equipment
- Operating System loaded SD Cards with Rasperian
- 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 teamwhere 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 laravell php package. However, this will be overcame due to good documentation on their site and contacts with laravell experience. Being software oriented scientists the raspberry pi at first seemed like a daunting task. However, Marcel has experience with hardware and this combined with good documentation will allow for us to overcome the hardware experience issues.

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/

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.

Rasberian
: 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 Licensea. 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.