Temperature and Humidity Monitoring System

Marcel Englmaier, Justin Koehler, and Jason Pearson

December 9, 2014

Contents

Background

Our client has many needs that have been unmet for various reasons. Existing problems make the current situation work improperly. 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. 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 25 °C or less, with automatic shut-off or critical shutdown temperature maximums of 35 °C. They must also be kept dry as any condensation 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 and to research done by many departments if it were damaged.

Our client informed us that there have been several incidents 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 paper rolls accrued a layer of surface rust, rendering it useless. It needed to be replaced, causing monetary damages and downtime.

Since these fateful incidents, WMU has had students implement several forms of reporting, and currently uses the a device called "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 useful, but unnecessary for our purposes. These devices also lack a very important feature: a central management system to view all the sensors. To alleviate this problem we proposed to create a web site that would communicate with a network of reliable, home brewed sensor units. This website was originally created by another WMU Computer Science Senior Design team and it currently is used to communicate with the Temperature@lert sensors.

We are unaware at this time of some specific information regarding the facilities such as the type and rating of their heating, ventilation, and air conditioning systems (HVAC for short), the dollar value of all 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. It is clear that WMU have a need for automated reporting, and our solution will be more than adequate regardless of this information. Our client has provided us with some basic information, which shows 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. We opted for a slower cycle to reduce server load and resource consumption. Our solution 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 $\pm 0.5\,^{\circ}$C. The maximum 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 ±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 specifications 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.

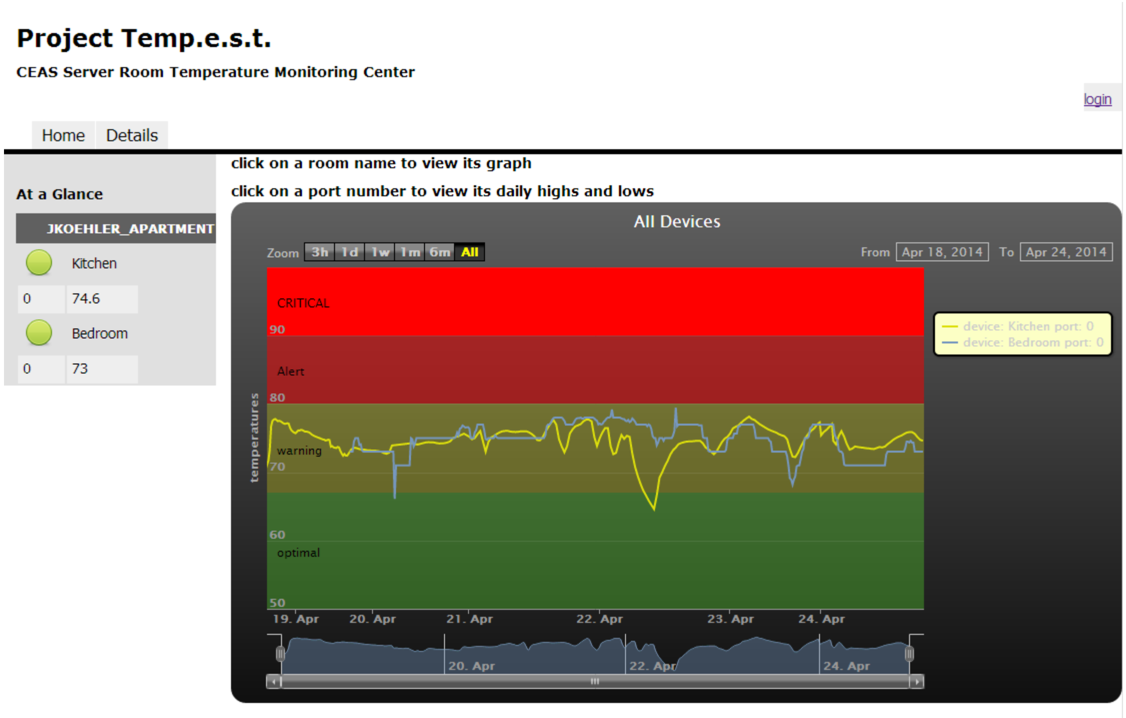

Figure 1: Initial View Of Site

The main page of the website that we inherited from the previous team is shown above. On this page there is a graph of all the temperatures of reported by all the sensor units that are currently being tracked. It does not currently display any humidity readings. We will be evolving the site to include this information in future releases. The existing framework alloys easy data viewing with little programming effort. We will use the existing framework and expand the code to include humidity. Additionally, we will be adding additional graphing functionality using the logged historical data.

The login process is very basic at this time. The existing web site uses lackluster security protocols, minimal, data sanitization, and no input verification. Our client has asked us to upgrade all security features and implement a system that will prevent session hijacking, network eavesdropping, cross site scripting, and brute force attacks.

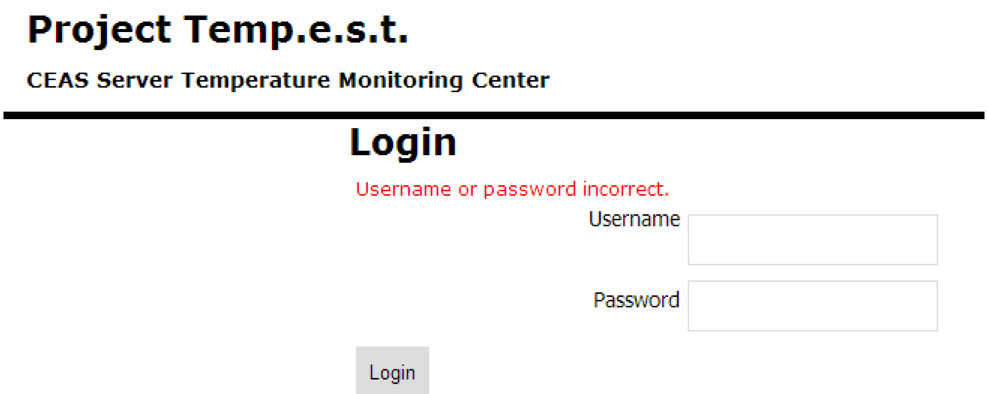

Figure 2: Login Page

As shown above, at this time the page has a simple Username and Password field along with a submit button. Our client has asked us to add additional features by adding a checkbox 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 their credentials, the framework will reference the data to see if it matches what is in the database. If it does, the framework will provide further access to the site. If the authentication fails, the framework will require the user to try again.

Our client asked us to create a single login page, but based on whether the user successfully authenticates as an administrator or a simple user provide 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.

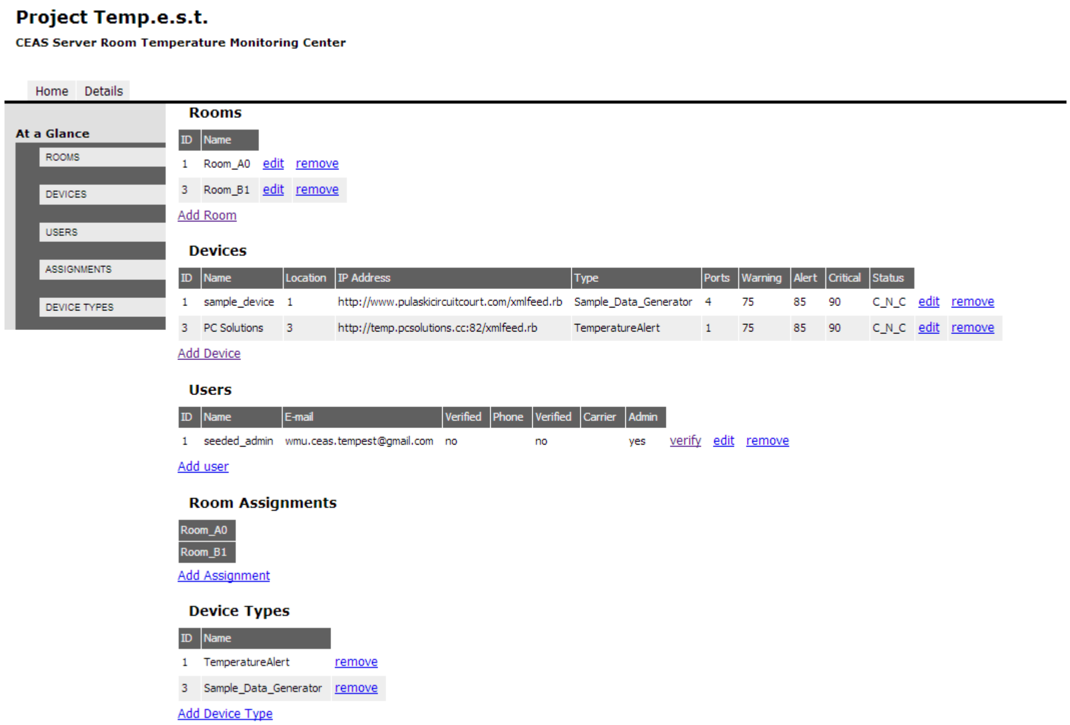

Figure 3: Add Page

Upon logging in as a system administrator this is what the admin sees. 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 redirects them to the login page so that all admin data isn't available to the public.

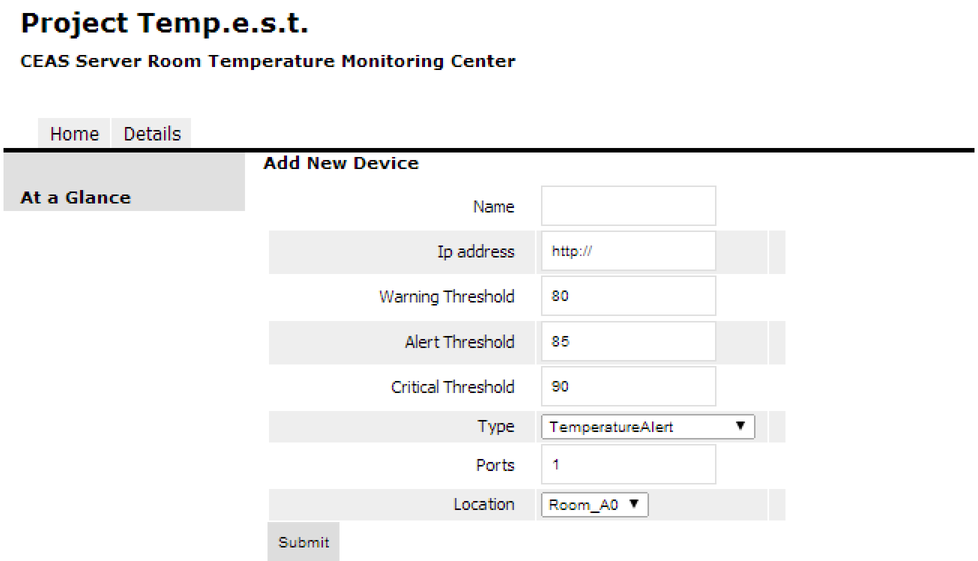

Figure 4: Add Device To Network Page

The above 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 data. The alert and critical thresholds are for setting the threshold for when to warn administrators for that room. The number of ports specifies simply the number of temperatures to expect coming from that device.

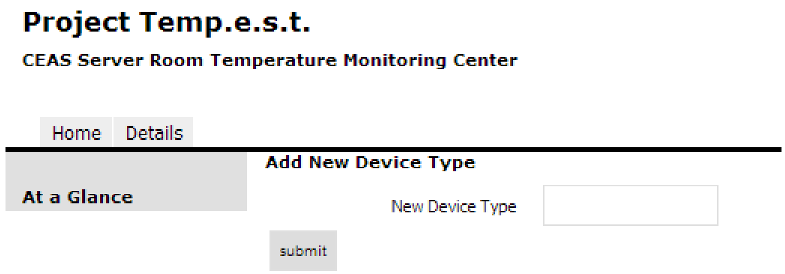

Figure 5: Add Another Device Type

The above 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.

The below figure shows the page where a new room can be added.

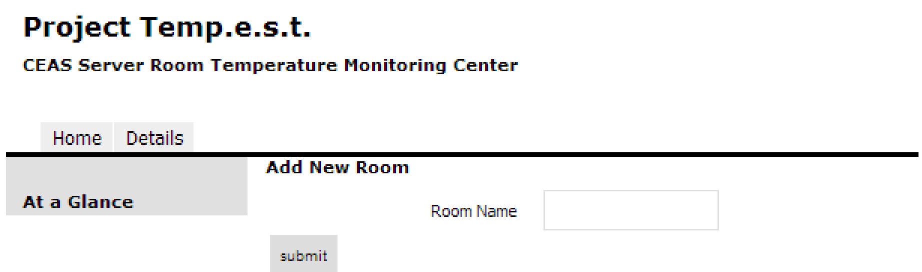

Figure 6: Adds a Room

There are three different levels of users. The highest level of user is the main system administrator, or "master admin". This administrator is in control of the whole system. Their permissions include all the powers of the normal room administrator, but are able make and demote normal room administrators.

The second tier of user is the normal room administrator. The room administrators are able see the stats of the rooms they have access and will receive alerts for those rooms. These administrators are able to add new users, add new devices, add new rooms, and change settings for these items.

The the third level of user is the regualar user. This user can see the graph of data on the front page and login. They also receive alerts, but are not able to change any settings.

There is a final type of user which requires no logging in. This option of user can only see the main page. It is beneficial to have it this way in case an administrator wants to quick check things and doesn't want to bother with logging in.

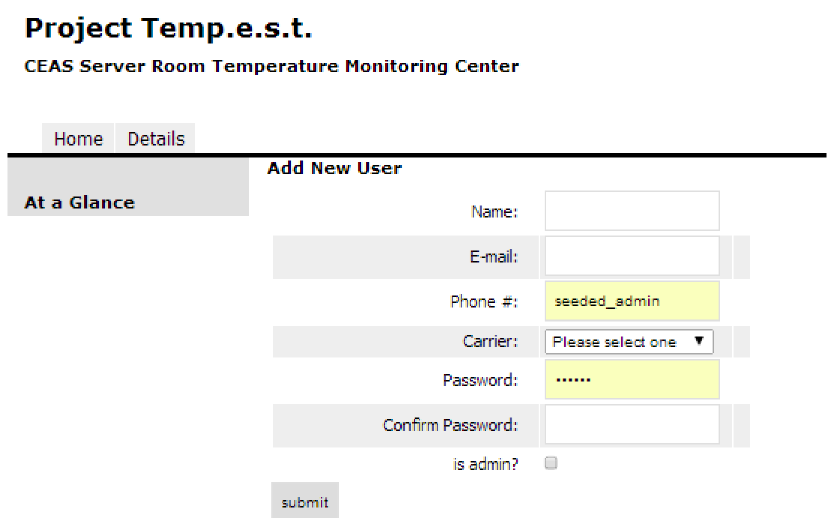

Figure 7: Page To Add Users

Above is the form for adding a new user. The required fields are name, email, and password. The password does have minimal requirements for good passwords. This is not very secure, and the email is also unverified.

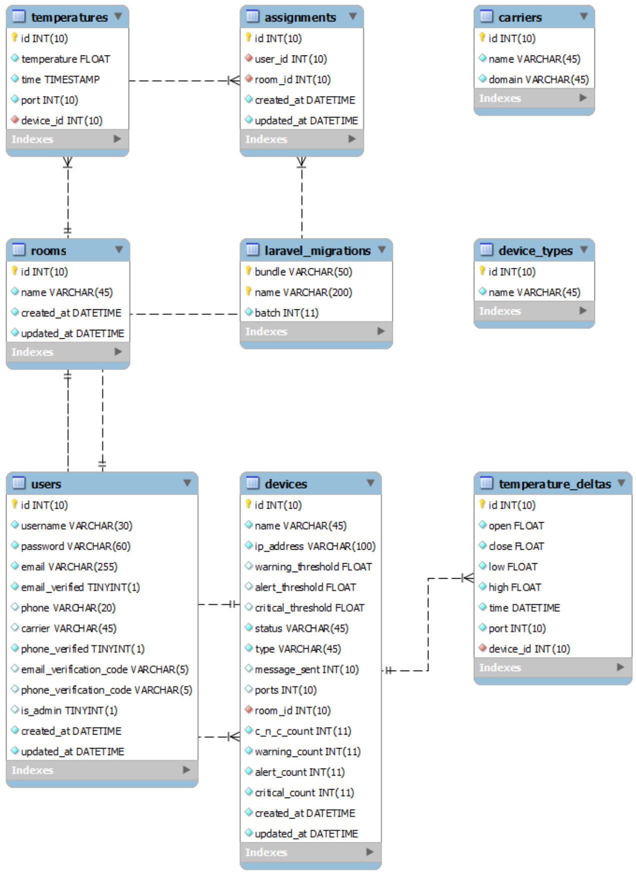

Figure 8: Entity Relationship Diagram

Above is the database diagram of the existing web site. We decided to not make minor changes because the database was designed in a manner that didn't allow humidity recording without adding another table

Design Decisions

Hardware, The Microprocessor

There were multiple options for microprocessors we could use. After some research, we narrowed down our list of possible units to these three: the RaspberryPi, the MSP430 Launchpad, and the Arduino Leonardo.

Figure 9: The RaspberryPi

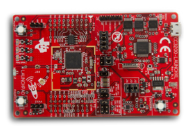

Figure 10: The MSP430 Launchpad

Figure 11: The Arduino Leonardo

Each of these devices had their own benefits. The MSP430 Launchpad was the cheapest option while the RaspberryPi was the most expensive option. A big problem with the MSP430 Launchpad is that is offers almost no features. It may be cheap, but the time, effort, and money we would need to put in to this device to connect it to the internet made it unusable to us.

The Arduino Leonardo was our second option. It came with many features. It was widely documented and widely used. But, the Arduino Leonardo face the same problem that the MSP430 Launchpad has: it isn't easily or cheaply connected to the internet.

Finally, the RaspberryPi. The RaspberryPi is heavily used and offers much documentation. In addition, it comes with multiple USB ports, an ethernet jack, and the ability to run a full Linux Operating System. This all comes in a $35 package. All we needed was to buy a $5 SD card to host the Operating System.

Hardware, The Sensors

Our client asked us to use both temperature and humidity sensors. We found many sensors:

- TMP36 Analog Temperature Sensor
- DS18B20 Digital Temperature Sensor
- AM2315 Digital Temperature Sensor
- DHT11 Sensor Unit
- DHT22 Sensor Unit

These sensor are all very good sensors in their own regard. We first looked at using two different sensors, and began by searching for temperature sensors.

The TMP36 was the first temperature sensor we looked at. It is cheap, and fast. It's also analog. Because it is an analog sensor, it requires calibration for each individual sensor. This is too much work and can cause reliablitiy issues, so we did not go with the TMP36.

The DS18B20 and AM2315 sensors were the next sensors we looked at. Both are digital, and require no calibration. We immediately removed the AM2315 from our list because it costs $30. We purchased a few DS18B20 sensors to do some preliminary testing. It is slow, requiring about a second to return proper data.

As we'd found our temperature sensor, we began looking for a humidity sensor. All of the ones we found received terrible reviews and we ignored them altogether. Then, we stumbled upon the DHT11 Sensor Unit. This sensor reads both temperature and humidity and is digital. It is about 3 times more expensive than a DS18B20 but is just as slow.

After testing for a few weeks, we began looking at the accuracy of our DHT11. As it turns out, the DHT11 is about 4 times worse in terms of accuracy than the existing $300 Temperature@lert units in place. We soon found out that the DHT11 has a big bother, the DHT22. It performs the same functions as the DHT11, but has a wider range of sensing, and is just as accurate as the existing Temperature@lert units. While the DHT22 is $5, it is very powerful and inscredibly accurate, and we decided to use the DHT22 for our final sensor.

Hardware, Enabling Internet Connectivity

We had two options for connecting our RaspberryPi sensors to the internet: a wireless or a wired connection. Both options we available to us by our Client.

Using the wired connection approach, we would be able to run the RaspberryPi using Power-over-Ethernet(PoE). All of WMU's ethernet cables in the CEAS building are PoE capable. The problem with this approach is the RaspberryPi doesn't allow PoE. We would need a PoE splitter attached to the RaspberryPi to remove the power from the ethernet cable and convert is to USB power which the RaspberryPi accepts. The benefits of PoE is that the RaspberryPi would not need an AC power unit.

Another drawback of using a wired connection would be the need to run an ethernet cable to wherever the RaspberryPi will be located. This is very inconvenient for moving the sensor units around and for cable cleanliness in general.

In contrast, using a wireless connection opens us up to a wide range of uses. While the USB wireless unit does cost about $10, this was a cost we were willing to incur. The wireless adapter we chose is a generic 802.11n adapter, but it is backwards compatible with the 802.11a/b/g protocols. Using a wireless adapter, we can move the sensor unit around to any location we require it to be, and because the 802.11n protocol is a long range protocol, we can go quite a distance from a wireless internet access point.

Software, Web Framework

For our web framework, we had many choices. We had options ranging from Ruby on Rails, to Tomcat, to CakePHP, and even to code in raw PHP without a framework. In the end, we decided to use Laravel. We did extensive research, and found, according to the graph below, that Laravel has the most market share of all PHP web frameworks.

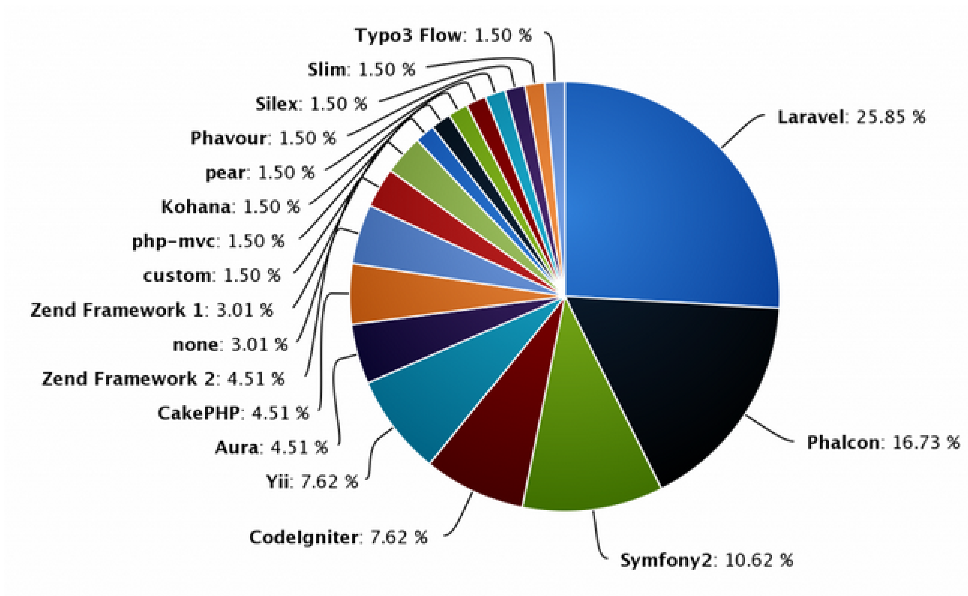

Figure 12: The Market Share of Web Frameworks

Having over a quarter market share over all other PHP web frameworks means that Laravel is doing something right. It also means that people trust Laravel, and because of this, there is quite a lot of documentation.

We did have a few more specific reasons for choosing Laravel over our other options. One of the biggest reasons we chose Laravel is because it has industry-standard security measures, that, when implemented properly, can lead to a very secure web site. In addition, Laravel is a little older, and has had time to mature and work out bugs. Therefore, Laravel provides an incredibly reliable Model-View-Controller system. As an added bonus, it's even open source!

Software, Sensor Unit Programming Language

Our RaspberryPi sensor unit requires some form of programming language. This decision should have been a rather simple decision, but during test, we had some issues. The manufacturer of our DHT22 sensing unit provides libraries for us to use to interface with the DHT22. During testing, we had some problems occur while trying to read data off of the DHT22. Originally, we used the C programming language to write the code which reads the DHT22. While C is incredibly fast, it requires compilation, which turns out wasn't the worst of our issues. While polling the DHT22 for measurements, it took the program anywhere from one to 15 tries to properly read the data. In our eyes, that isn't very reliable.

We decided to give Python a try. Python is a slow language but it doesn't require compilation. It is also extremely reliable, as the program we wrote reads the measurements properly the first time, every time. While it is slow, waiting about two seconds for an updated measurement is something our client was willing to live with, as the propagation of that data to the user-facing front end still takes time, and waiting an extra few seconds will not impact any hardware any more.

Server

For our server setup, we had to make a few decisions.

- What Operating System should we use?
- What Web Server should we use?
- What Database technology should we use?
- What Framework should we use?
- Where do we host the web server?

These questions are all equally important. We first had to pick the Operating System. We had the choice between Windows, a rather expensive option, and Linux, a very free option. We decided to go for the Linux system because of the cost and because it is open source.

Once we had our Operating System chosen, we made the decision to use a set of technologies that are very commonly used. It's called a LAMP stack, and stands for Linux, Apache, MySQL, and PHP. We had other options for each of these technologies. We could have used a web server called nginx, but the documentation and historical evidence of Apache working was already there. There were other database technologies that we could have used, but MySQL is free, powerful, and well documented. Finally, our reasoning behind PHP was explained in the Web Framework Design Decisions.

Finally, we had to choose where we wanted to host our website. We had options available to use like Microsoft Azure, Linode, or DigitalOcean. We opted for Amazon Web Services(AWS).

Our primary reason for picking AWS was that it is free for the first year, and then only $10 per month. In addition, we can easily migrate our virtual machine with great ease, and the uptime for any AWS service is more than 99.95%. As an added bonus, AWS gives us unlimited bandwidth at no extra charge. Even though our web site and server doesn't use much data at all, a heavy user load can add up quickly.

Web Site, Design and Styling

The existing web site that we took over from the previous team did not look nice at all. The buttons were ugly, the styling was poor, and the colors were mismatched. Furthermore, it was not mobile-compatibly at all. We had to find a solution.

We decided to tackle the graphs on the pages first. The existing web site used High Charts, which is a rather useful graphing library. As it turns out, the version that was being used was outdated and old. We updated it, and most of the problems that existed were gone. The updated version provided mobile support, scaled well, and looked much cleaner.

There were alternative for us to use. We could have scrapped High Charts altogether and gone with commonly used libraries like Google Charts, or Chart.js. We also thought about using the YUI, the Yahoo User Interface Library. All of these are used often across many different implementations, but we decided to stick with High Charts because it worked and provided all the functionality we required without a massive re-write of the main pages.

Along with an updated and cleaner charting library, we decided to scrap most of the existing styling on the web site. We wanted our web site to be accessible across all devices, and the existing site didn't provide that. We looked at using jQuery Mobile, which is slow, and Foundation, which is not insanely popular. We opted to go with a library called Bootstrap. Bootstrap is open source, widely used, and well documented. It's also incredibly easy of use and modify. It relies on a basic set of class tags which propogate and scale incredibly well. Once we finished implementing Bootstrap, the website became usable on any device and looked really nice.

Design

do your poops here

Implementation

Now how did this design turn out. Before all the old site vital areas were shown. Now after implementing BootStrap we have a completely different front end. Here is the initial view of the site once a user has logged in.

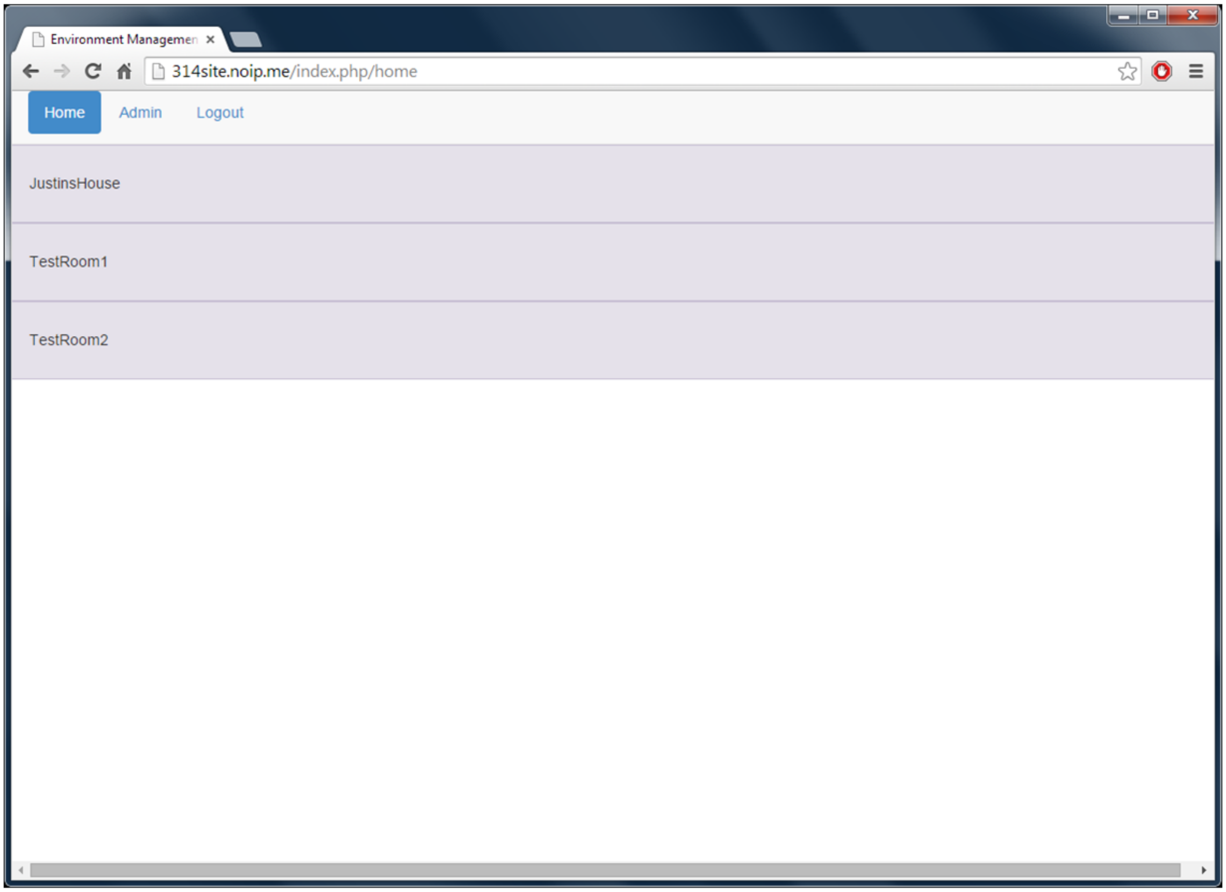

Figure 13: After Logging In

Now when a user clicks on the tab it will expand to show what the current temperature and humidity are. Next to that is the status of the machine i.e. okay or critical. Upon mousing over the graph it will give the temperature or humidity at that time in history. See the next figure.

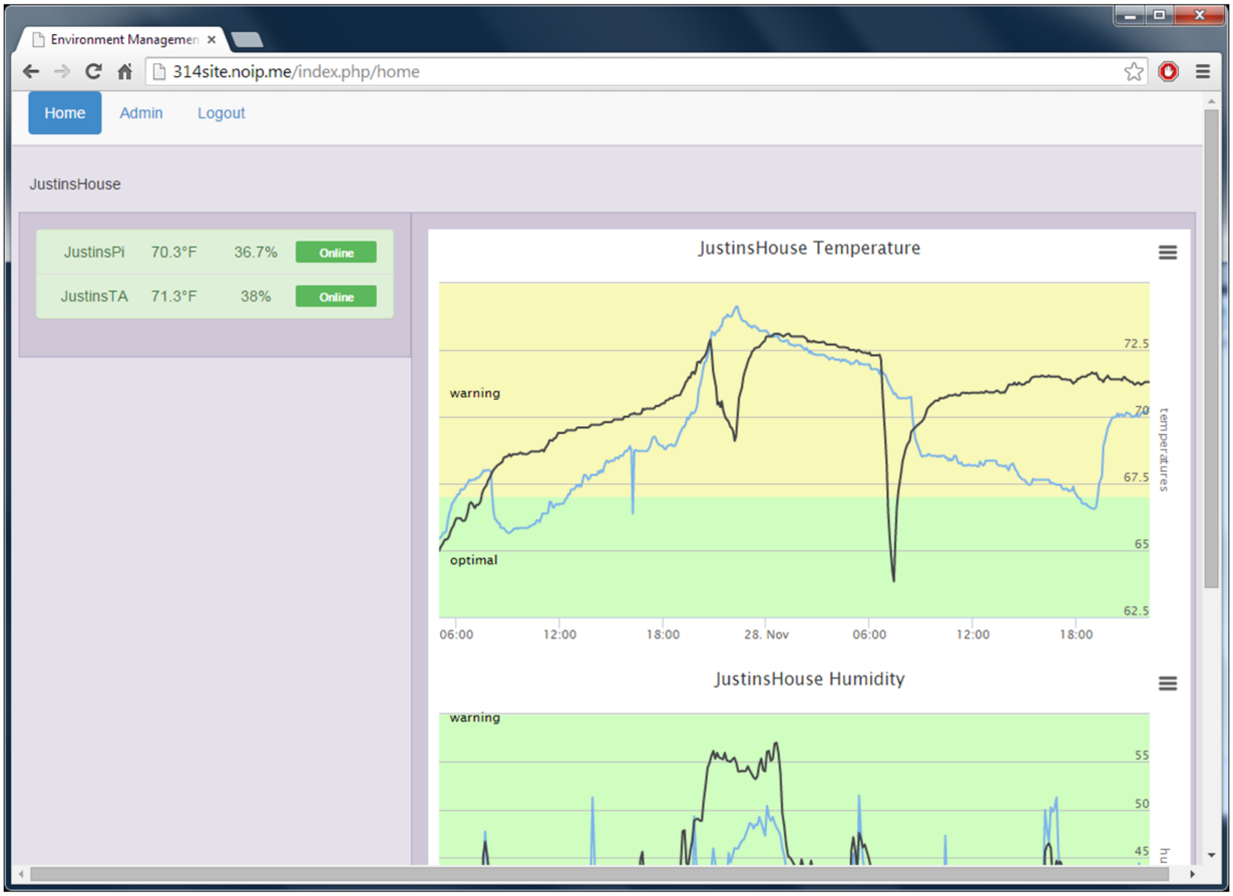

Figure 14: Open Tab and Data

These tabs are the same for each room and will show all appropriate data for each room. Next is the new and improved administration panel. On this panel there are five different tabs rooms, devices, users, room assignments and device types. All of these different devices the administrator is able to create, update and delete anything in those tabs.

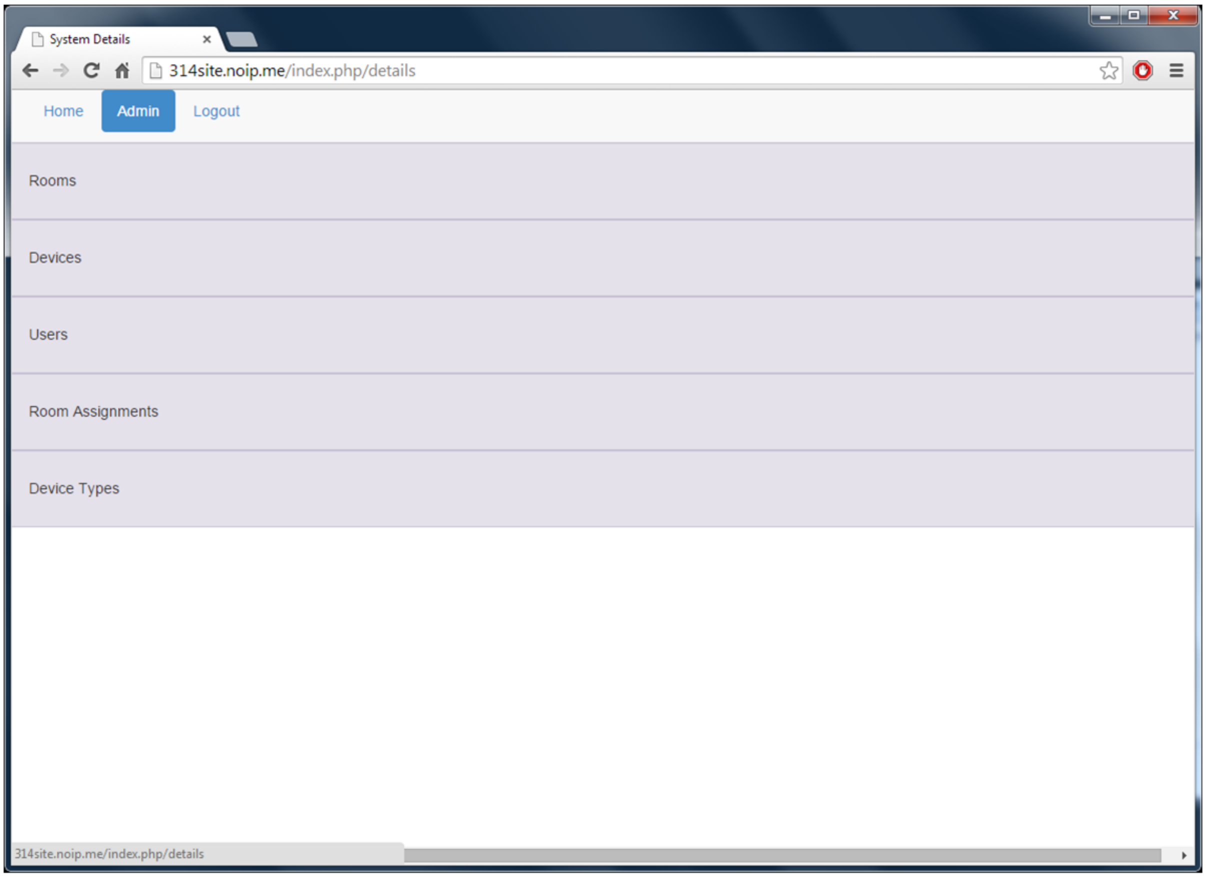

Figure 15: First View of Administration Panel

The rooms tab has just those options and only has room name as its only parameter. See the next figure for the rooms user interface.

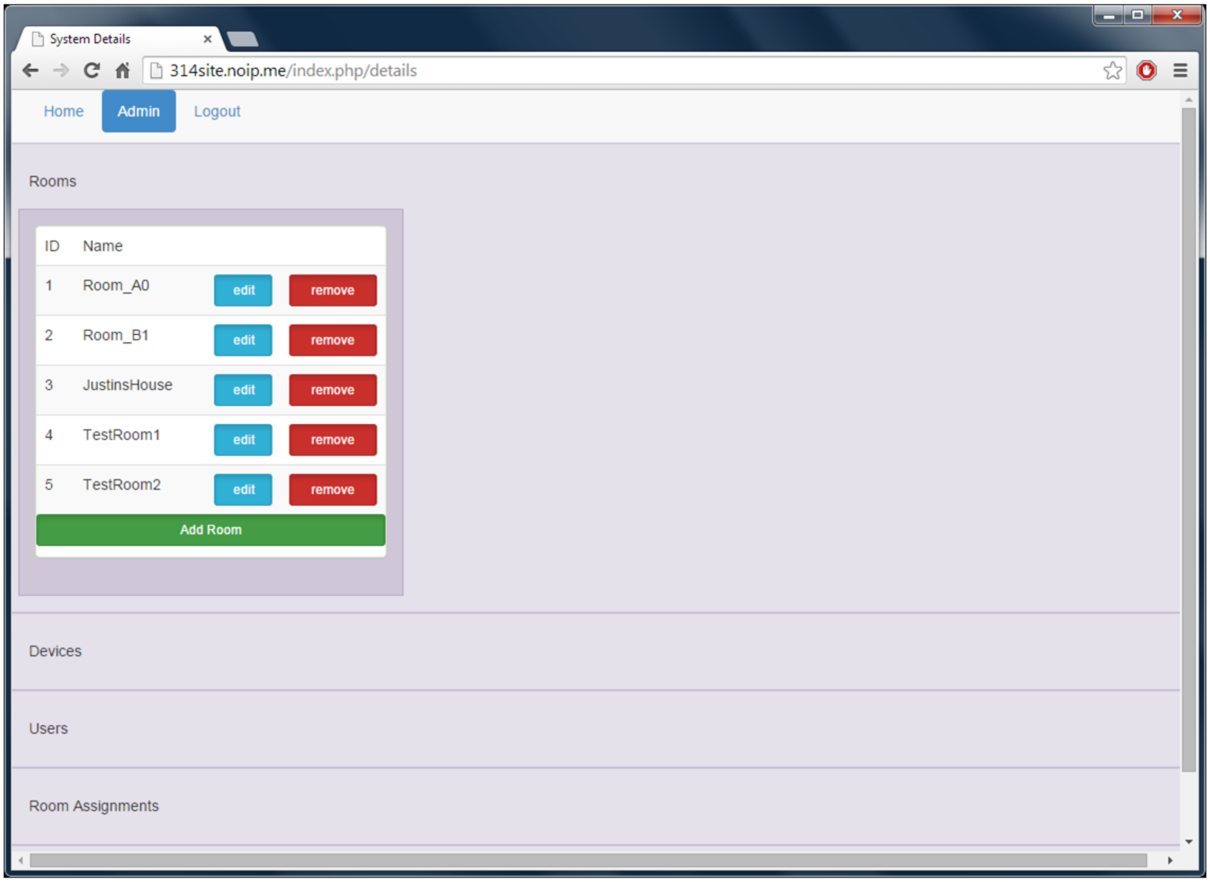

Figure 16: View of the Rooms Tab

The next tab devices tab. This tab shows the device id, name, location, ip address, type of device, number of ports on the device, warning, alert and critical temperatures, and lastly status. The administrator can add, remove and update devices.

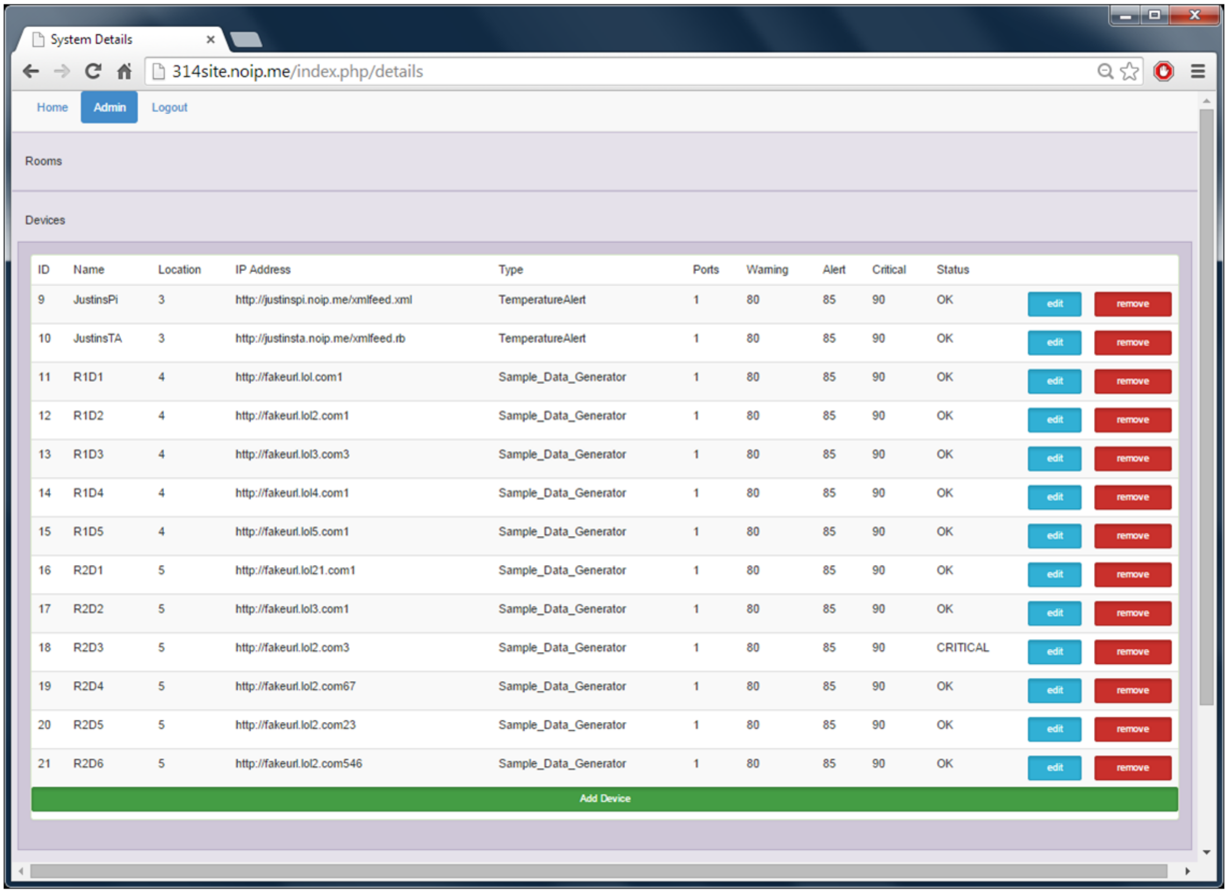

Figure 17: Device Administration

The users tab allows new users and administrators to be created as well as updating and deleting users. This tab also allows verification of the users email and phone alerts. For a verification the user receives a text or email with a verification code that that the user types in the code and this allows the system to verify that the text or email went through.

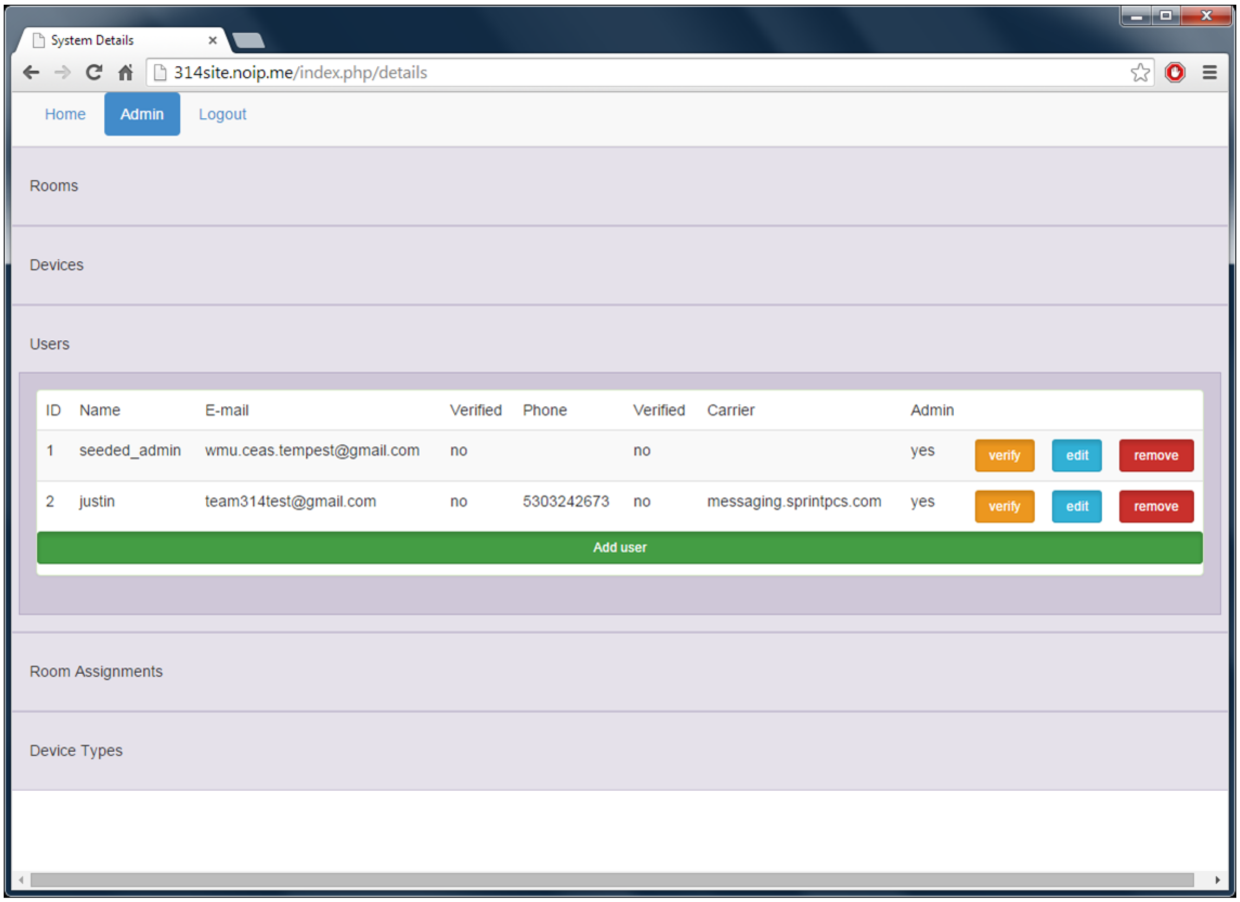

Figure 18: User Administration

The room assignments tab is very important. This is where administrators assign users to rooms. When a user is assigned to a room they will receive alerts when temperature benchmarks are reached.

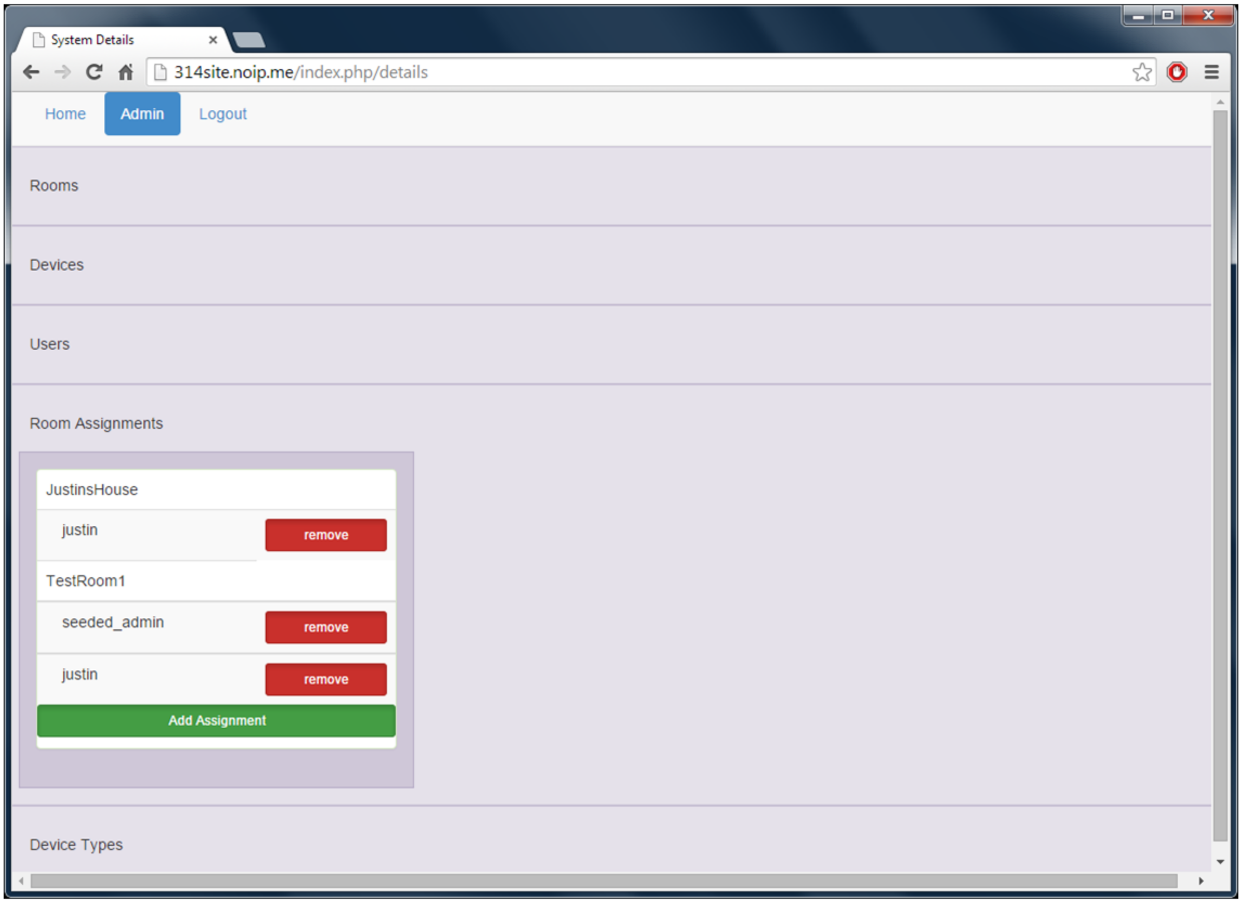

Figure 19: User Room Assignments

Testing

There are two types of testing that we used in this project unit testing and functional testing. Unit Testing is a must because it will allow the user to test every feature when a new patch comes out and allows after each upgrade to check if the features still work. Unit tests cannot be used to check everything so there must also be functional tests that show the site is reacting correctly and can handle loads properly.

Unit tests for this project are ran through the command line and use laravel's testing framework.

There were many different functional tests that we ran to test full functionality of our project. A few tests that we ran include

- E-mail Alerts
- Text Message Alerts
- Virtual Sensors to test the web site's reporting and usability

Security

Maintenance

After the completion of this project the current senior design team will offer no maintenance. All modifications will be done by the staff at Western Michigan University and anyone else who uses the project. This being said there are modifications that we had planned but didn't have time to get to that could be added to increase the user experience for the project. These include

- Adding a secure layer between the pi and the server,
- Adding historical data logging to the Raspberry Pi,
- Creating a way to retrieve historical data from the device if it was down for a period,
- Upgrading the user control area to have group administration,
- Making a program that would poll servers for their data and add it to a graph.

We plan on working on implementing these features, and modifying the project with our own ideas in the coming future, gut we do not guarantee any finished products. We will, however, offer our work to the client upon completion of a stable version and successful testing.

In addition, we have created multiple guides and how-to manuals for installing and building the sensor unit, and for putting the server together.

Resources

- Raspberry Pi
- Temperature Sensor(s)
- Humidity Sensor(s)
- Web Server, preferably externally hosted
- Soldering Equipment
- SD card(s) loaded with the Raspbian Operating System
- Power Connector(s) for Raspberry Pi
- WiFi usb dongle for Raspberry Pi

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 related these are the sites we use

- http://laravel.com/docs/quick
- http://www.w3schools.com/
- http://www.noip.com
- http://httpd.apache.org/
- http://www.w3.org
- http://aws.amazon.com/

Glossary

GUI
: Graphical User Interface. The windows a user interacts with.

MSP430 Launchpad
: A 16-bit microcontroller platform made by Texas Instruments.

RaspberryPi
: 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 is 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 for our project is open source and will be hosted for all to access and modify as they desire on GitHub.com.

Intellectual Property (IP)

As this project is being developed as a Senior Design project for Western Michigan University (WMU) under 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.com, 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.

Warranty

A maintenance document has been created for the project. Other than that document and this document no other outside assistance is required by the members of this project, and no warranty is given or implied.