

Smart Home Platform

Submitted By:

Thorson Dai

Neil Surti

Austin Su

Imad-Uddin Siddiqui

Team Project Number: 36

Advisors:

Hana Godrich

Samuel Ramrajkar

May 10, 2018

**Electrical and Computer Engineering Department**

**Rutgers University, Piscataway, NJ 08854**

# **Project Abstract**

Many smart home devices provide limited functionality at cost. Take for instance, the Nest Thermostat, which retails at just over $200 at the time of this report. Without purchasing an additional sensor module, the Nest Thermostat makes a heating schedule for your home, essentially amounting to an expensive thermostat timer. Coupled with the tendency for companies to develop devices that only work with their own proprietary ecosystems means that making a home “smarter” is often a costly proposition.

Recognizing this, we sought to develop a more flexible and open platform for smart home devices to gather ambient data, communicate between themselves, and ultimately improve residential quality of life. As mentioned earlier, one primary constraint is that of cost, as it is typically the individual homeowner who has to pay out of pocket for a smart home system. For them, the benefits must clearly outweigh the associated costs. These higher-end smart home devices lack mass market appeal, fragmenting functionality into sleek but insular modules. Thus, for this capstone, cost considerations were a critical part of design.

In order to prototype our open smart home platform concept, we split the implementation into two parts. First, an ambient data collection node that has sensors for temperature, humidity, light levels, and motion detection. Second, a server for data processing and visualization that serves a web application for the smart home platform. At its core, our platform takes in data from a node, then processes it according to high and low comparator triggers set in software, updating the relevant control signal. This decoupling of data collection from smart home devices can not only reduce size footprint, but also allows for the sharing of sensors. As sensors generally account for a sizable part of the cost of smart devices, this can improve their affordability.

For future study and development, we are considering further improvement on data node functionality along with additional security and user-interface features for the web application, all without breaking the bank. Cost for prototyping a single data node was under $100, and we hope to be able to make these nodes even more affordable at around a $25 price point.

**Table of Contents**

[**Project Abstract**](#_3zon7843wwe8) **1**

[**Introduction**](#_92lcbnye4irx) **3**

[**Data Node (Hardware)**](#_4zq4jvtm103m) **4**

[**Platform Server (Software)**](#_k6pvp8u0crjp) **7**

[**Finishing Touches**](#_l40y94b1v61i) **13**

[**Cost and Sustainability Analysis**](#_kh2ck0ex7mm6) **13**

[**Summary and Future Work**](#_327plog326l2) **14**

[**Acknowledgements**](#_rhgm2upkmfel) **15**

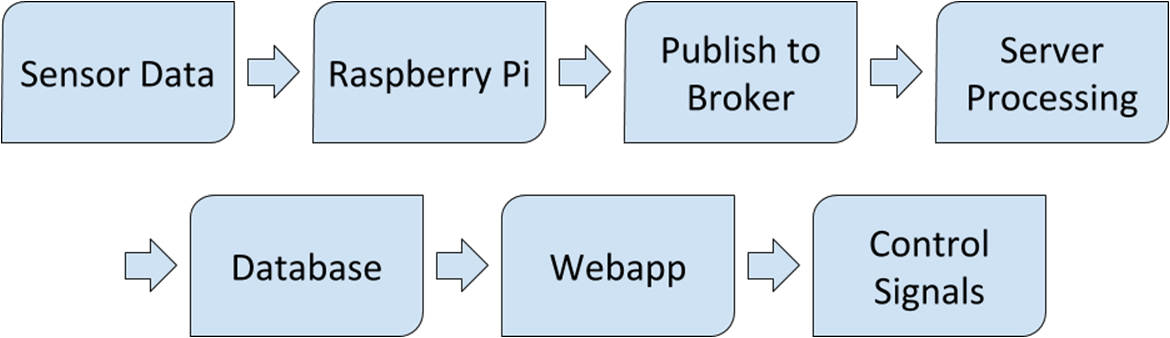
[**References**](#_iw580bi1bjqb) **16**

# **Introduction**

Many smart home devices provide limited functionality at cost. Take for instance, the Nest Thermostat, which retails at just over $200 at the time of this report. Without purchasing an additional sensor module, the Nest Thermostat makes a heating schedule for your home, essentially amounting to an expensive thermostat timer. Coupled with the tendency for companies to develop devices that only work with their own proprietary ecosystems means that making a home “smarter” is often a costly proposition.

Recognizing this, we sought to develop a more flexible and open platform for smart home devices to gather ambient data, communicate between themselves, and ultimately improve residential quality of life. As mentioned earlier, one primary constraint is that of cost, as it is typically the individual homeowner who has to pay out of pocket for a smart home system. Thus, for this capstone, cost considerations were a critical part of design.

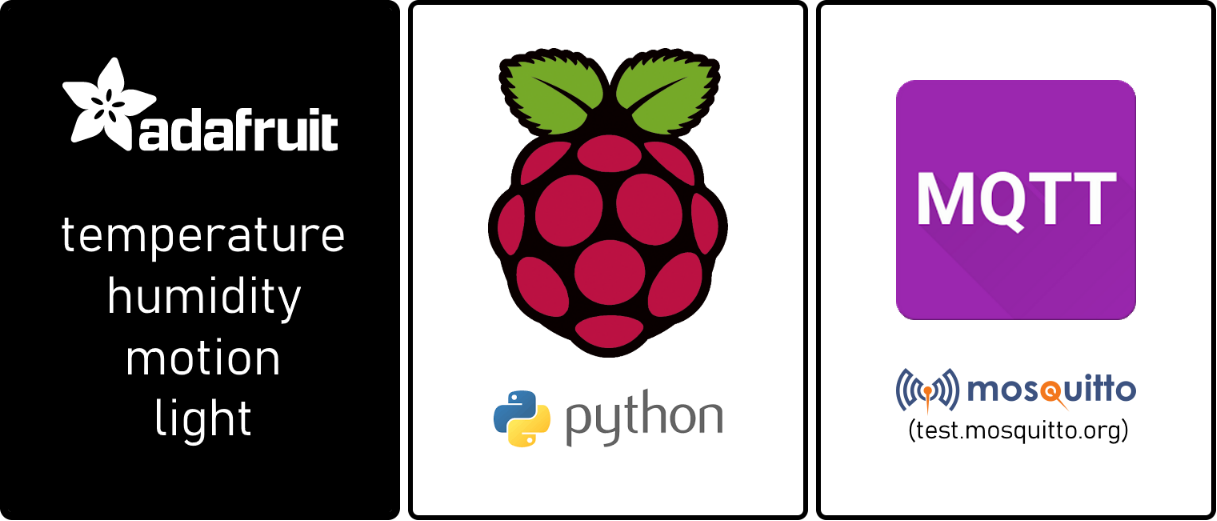
In order to prototype our open smart home platform concept, we split the implementation between hardware and software. First, an ambient data collection node that has sensors for temperature, humidity, light levels, and motion detection (hardware). Second, a server for data aggregation, processing, and visualization that serves a web application for the smart home platform (software). The following sections will detail their design and relevant considerations in decisions made.



*Figure 1: Flow Chart of Operation*

*The entire chain of operation of the implemented platform prototype*

# **Data Node (Hardware)**



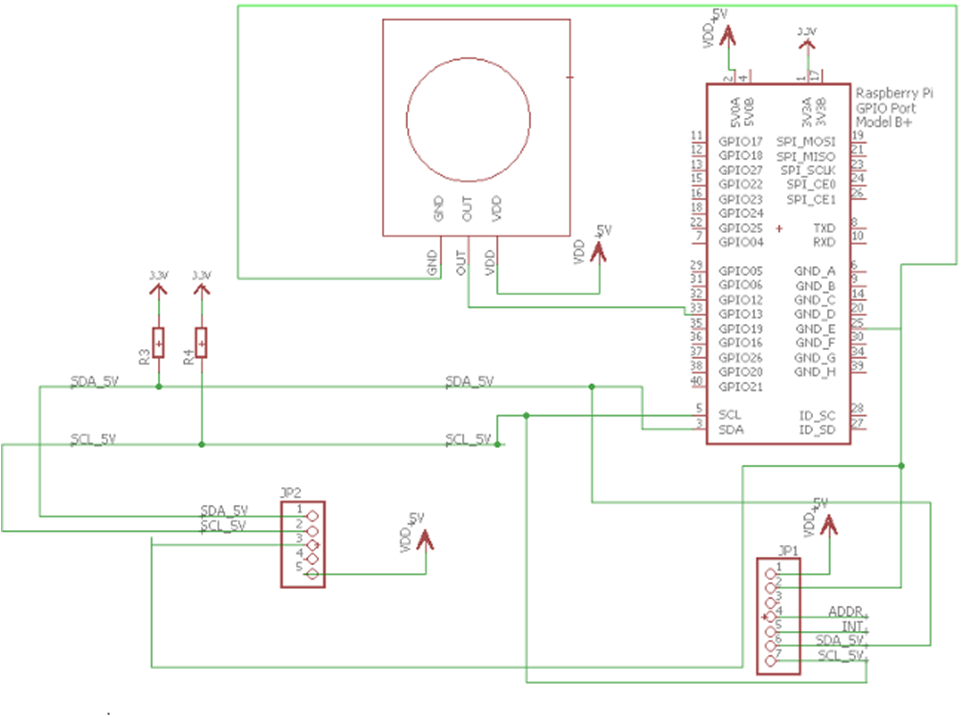
*Figure 2: Architecture of a Data Node*

*Adafruit sensors on a Raspberry Pi transmitting over MQTT IoT protocol*

For speed of development and ease of learning, Python was selected by members of the hardware team (Neil Surti, Imad-Uddin Siddiqui) as the language to write sensor drivers with. The sensors used in prototyping the data node were the Adafruit Si7021, TSL2561, PIR, which sense temperature / humidity, light levels, and motion respectively.

Our main motivation for using a full-sized Raspberry Pi over a microcontroller board was its built-in wifi module in addition to an ethernet cable port. With this, it was possible to test the MQTT transmission client code with a reliable connection (wired) and the robustness of the protocol under an unreliable connection (wifi). Another consideration was serial port access; the Raspberry Pi has an Inter-Integrated Circuit (I2C) bus. Interestingly, it can support up to 128 slaves, making the prototype easily extensible with more sensors and peripherals.

In order to run the sensors off the same bus, level translators on the sensors had to be desoldered and proper connections for serial data/clock were made to the I2C bus. Once the sensors were interfaced with appropriate drivers to call ambient values from registers, the next step was sending data.



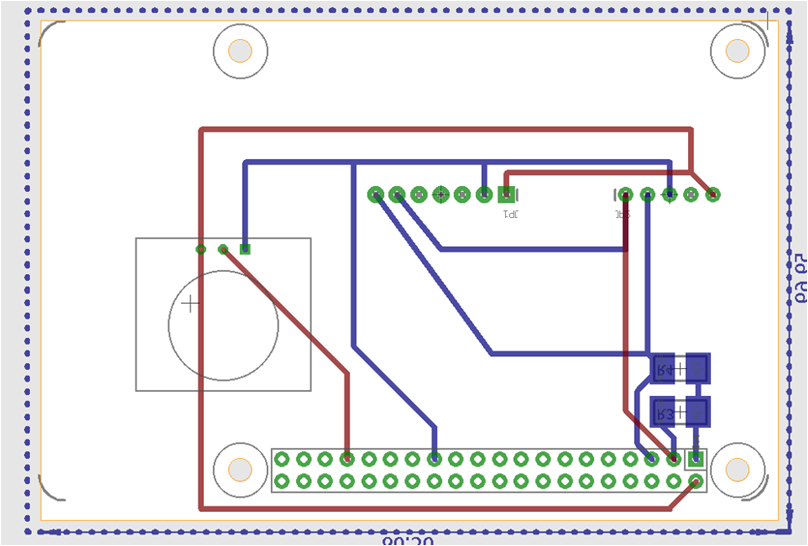
*Figure 3: Data Node Schematic*

*Wiring schematic of data node (Raspberry Pi and sensors)*

With the help of Thorson Dai of the software team, a Python script was written that sampled values from each of the sensors, creating a per-minute average and writing to a named pipe. It did take the hardware team several days to figure out how pipes work and how to use them.

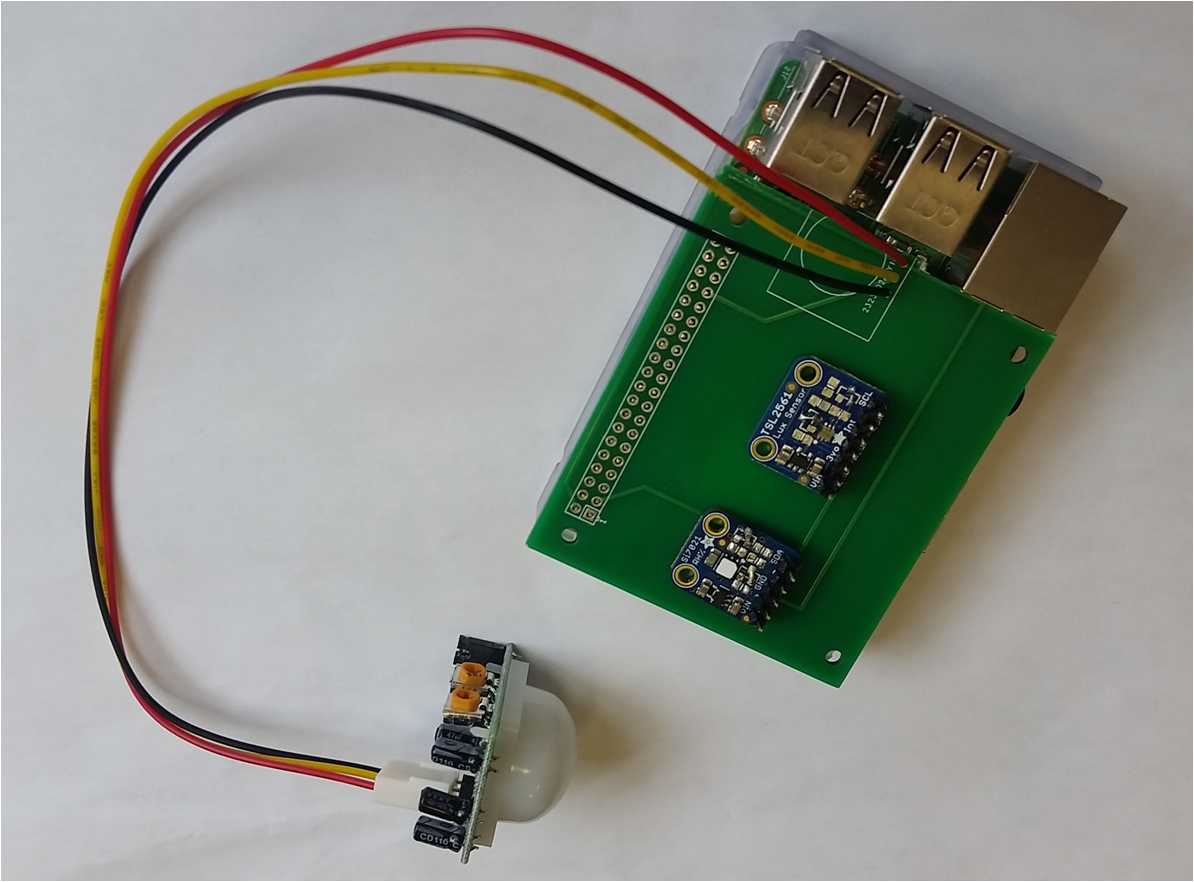
Reading from the other end of the named pipe was a Node.js (Javascript) process that was a commented snippet taken from the platform server This was done for the sake of time, as the snippet had already been tested and known to be functional. Doing so made it also made it easier for us to determine an connection issue as the cause for message loss on integration.

Once connection configuration issues were addressed, data was successfully sent to the platform server over the internet. With most of the hardware done, Neil Surti laid out the custom PCB board for printing.



*Figure 4: PCB Board Layout*

*Finalized PCB board layout that was submitted for printing*



*Figure 5: Printed PCB Board*

*Actual printed PCB board with proper connections and fit*

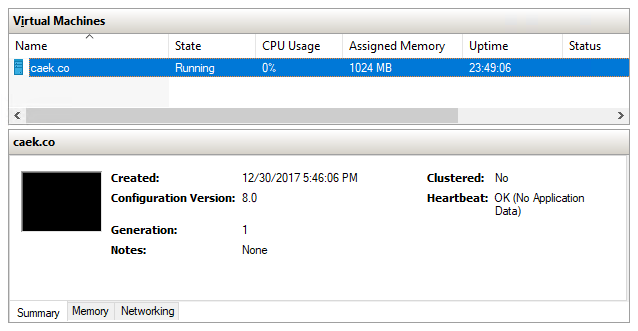
# **Platform Server (Software)**



*Figure 6: Architecture of a Platform Server*

*A simplified card display of the prototype platform server architecture*

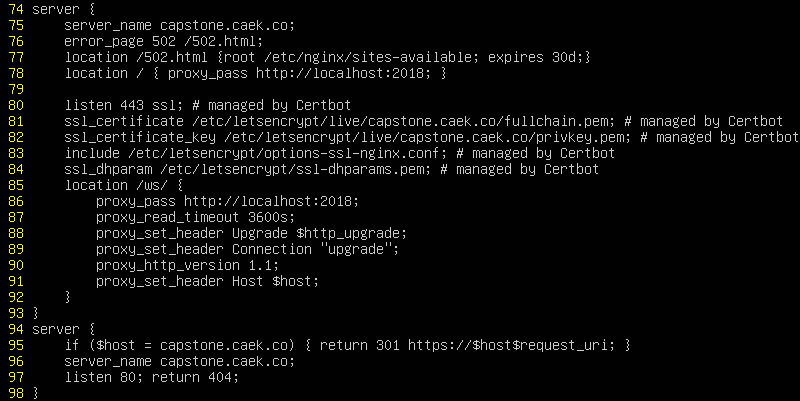
With the data being sent to the platform server, we now describe the core of our Smart Home Platform project. Backend and frontend work was handled by Thorson Dai, with additional frontend research performed by Austin Su.



*Figure 7: Hyper-V Dashboard*

*A cropped screenshot of the Hyper-V hypervisor dashboard*

Shown in *Figure 7*, the Ubuntu platform server was virtualized using Hyper-V virtualization on actual physical hardware. Although underutilized, it can be used in an implementation of load balancing which will be discussed in a later section. For a more straightforward network configuration, the virtual machine was assigned a IPv4 address on the local network separate from the host machine. Thus, port forwarding of ports 80 and 443 (standard HTTP and HTTPS ports) points to this IPv4 address of the (virtualized) platform server.



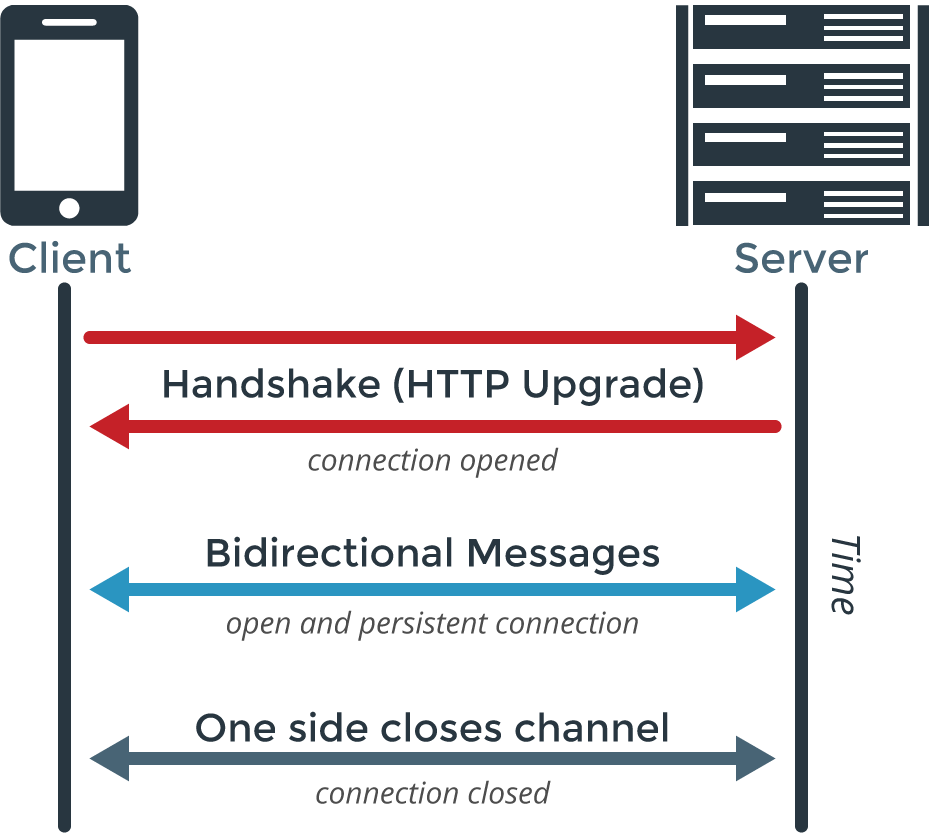
*Figure 8: /etc/nginx/sites-available/default*

*The NGINX configuration for capstone.caek.co*

Because we are using a subdomain of caek.co for the project, *capstone.caek.co*, we selectively direct requests to that specific subdomain to the process listening in on the local port 2018 using NGINX directives. NGINX serves as a reverse proxy - simply put, it can receive internet traffic and redirect it to an appropriate server or service location. It can also apply encryption (SSL).

So what is listening on port 2018? It is the Node.js (Javascript) process that receives and processes data node information, serves web application content, and visualizes data. This program interfaces with an MQTT broker to receive data node information, MySQL for data storage and processing, and opens WebSocket connections for live data updates. Particular for this purpose, uWebSockets was chosen for its low memory overhead, low-level simplicity, and speed over its other library counterparts, such as socket.io and ws.

Note that much like any other website, the web application served by the platform server is sent over HTTPS protocol. Because HTTPS is a half-duplex protocol, it relies on a request-response paradigm. This means that a client must submit a request for the server to respond to. For a live data feed, however, it makes more sense for the server to initiate communications, sending data over a persistent connection as it arrives and is processed from an ambient node, which is what the usage of WebSocket provides.



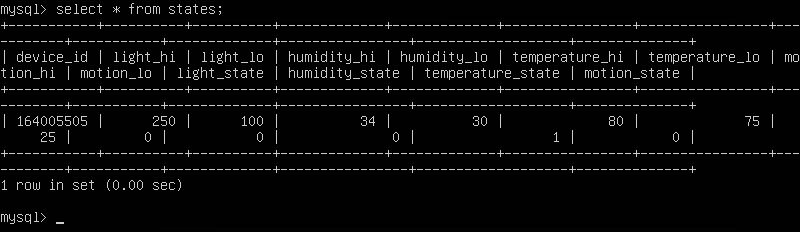
*Figure 9: WebSockets vs REST*

*A comparison of half-duplex HTTP and WebSockets (Source: PubNub)*

After the initial request from the client for the web application, using the protocol upgrade mechanism provided by HTTPS, a WebSocket connection is initialized. From there, either the client or server can send information across, reusing the same connection. This saves some bandwidth, as handshake headers need not be resent each time as in HTTPS, but for the most part the performance different is negligible. However, in terms of system design, it makes the system far more flexible to use a stream-based protocol (WebSocket) over a message-based protocol (HTTPS). While for some sensors it may be appropriate to gather data points over the course of a minute and average them, other more critical node peripherals and smart home devices may require a faster response and update time.

In terms of data sending and storage, MQTT and MySQL were chosen as part of the server architecture for their robustness and relational nature, respectively. The former was specifically designed for IoT (Internet of Things) applications, which is the general category that smart home devices fall in. Specifically, the MQTT protocol handles dropped connections with reconnects and data resends. Of course, much of this is handled by an intermediate broker. Due to time constraints we decided against integrating a broker into the platform server process. Instead, we opted to use one of several online free MQTT broker test services for prototyping.

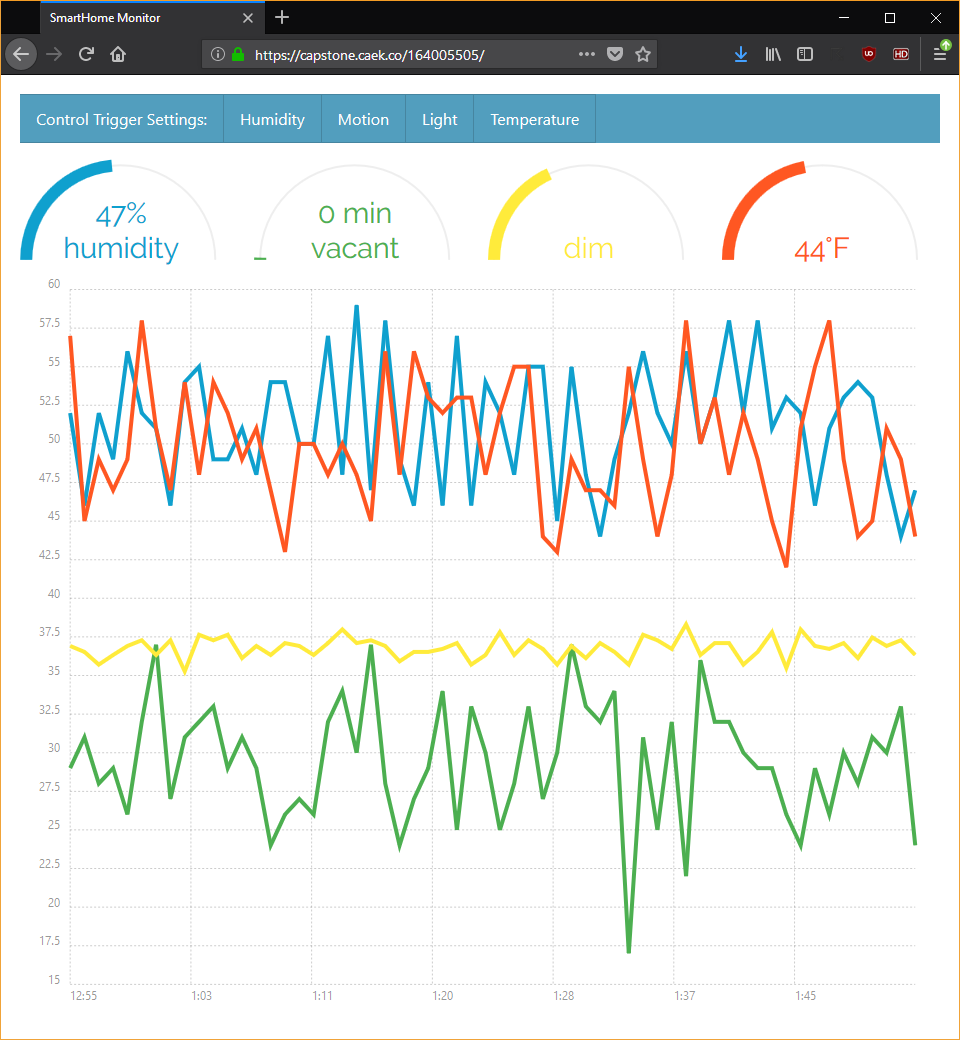
Meanwhile, the latter (MySQL) was selected for storage due to its relational database model. As data would be associated with particular data nodes, which are associated with particular sensors, a relational data model seemed most appropriate. This does not discount NoSQL databases like MongoDB and Redis, however, which offer greater flexibility in entry definition due to their schemaless nature. Regardless, this design decision will definitely require reevaluation in the future, considering our goal of a more flexible platform.



*Figure 10: States Table*

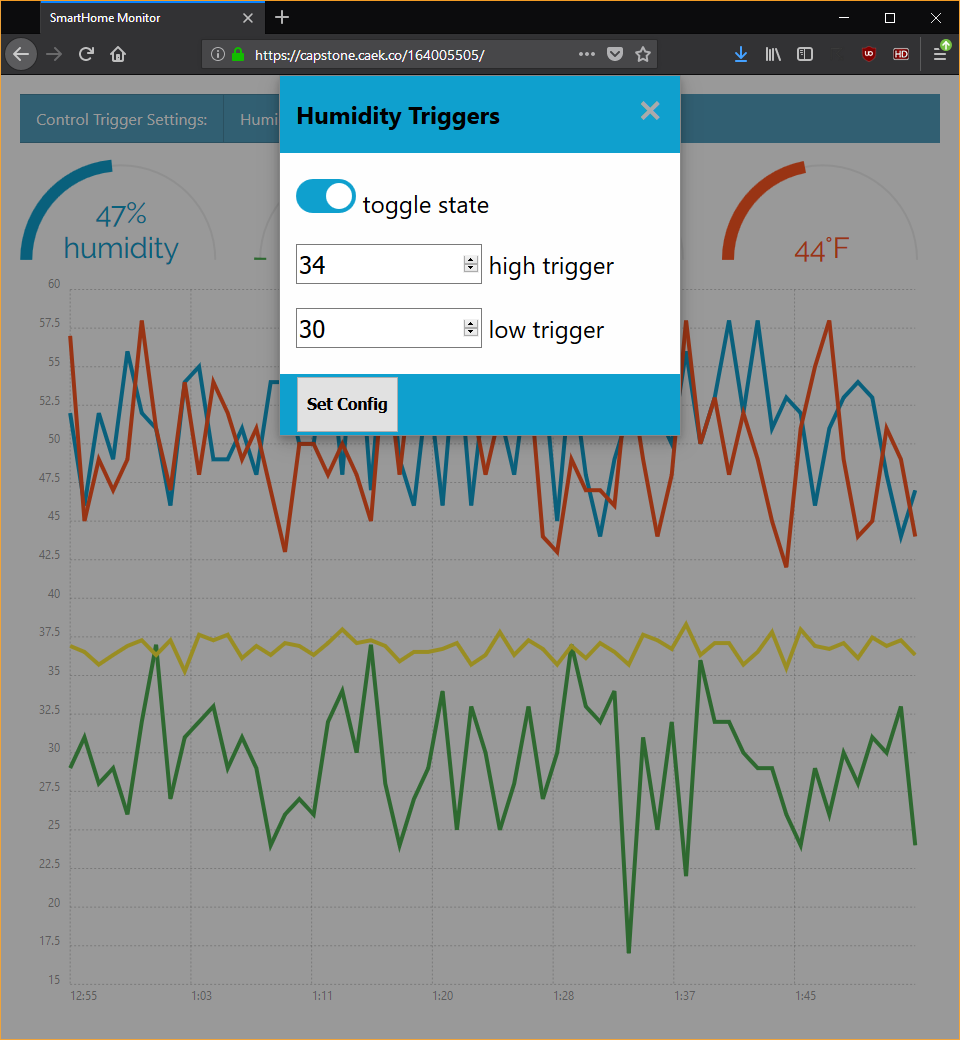
*The states table of the platform server process after application testing*

Finally, in terms of processing, the platform server process evaluates each set of ambient data it receives via a software-defined Schmitt trigger comparator. A pair of user-defined thresholds (high, low) for each ambient data value (temperature, humidity, light level, motion events) are used in this comparator. When an ambient value exceeds the high threshold, the state value for that particular ambient data type is set high and vice versa when the ambient value falls below the low threshold. These trigger-related values are stored in the aforementioned MySQL database, which operates alongside the platform server process on the same virtual machine.



*Figure 11: Data Visualization*

*Random values from data node to platform server process to test sending*

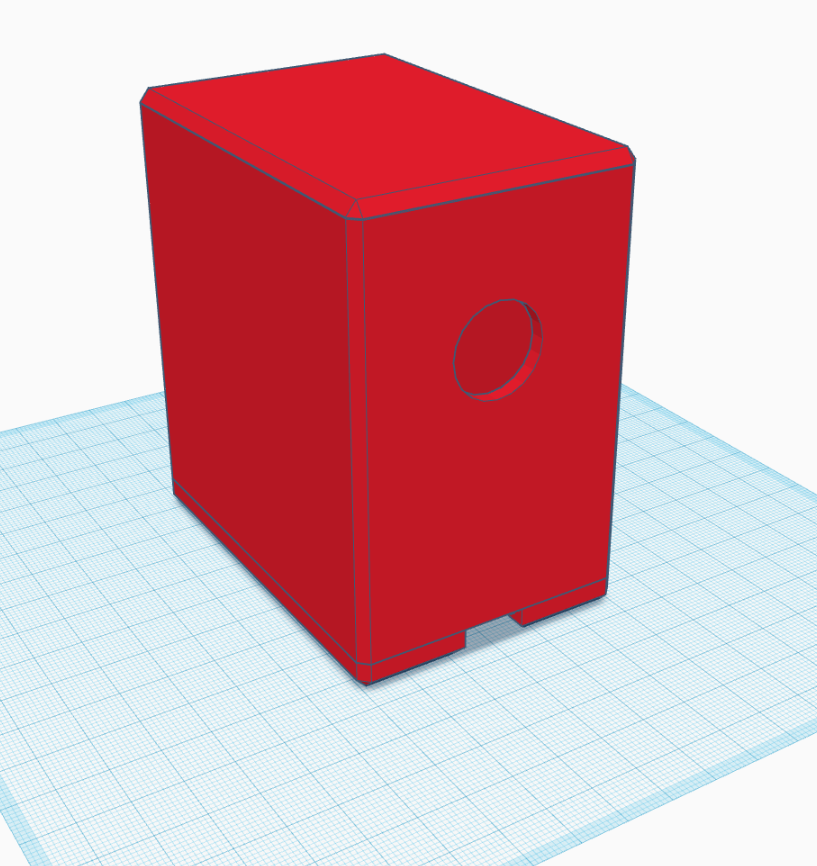


*Figure 12: Schmitt Trigger*

*UI for viewing and setting state, alongside high and low trigger values*

# **Finishing Touches**

As preparation for the Capstone Expo day, Thorson Dai with the assistance of Austin Su modeled the data node enclosure in TinkerCAD for 3D printing.



*Figure 13: Realization of 3D Model*

*The modeled enclosure in TinkerCAD; after printing and installation*

# **Cost and Sustainability Analysis**

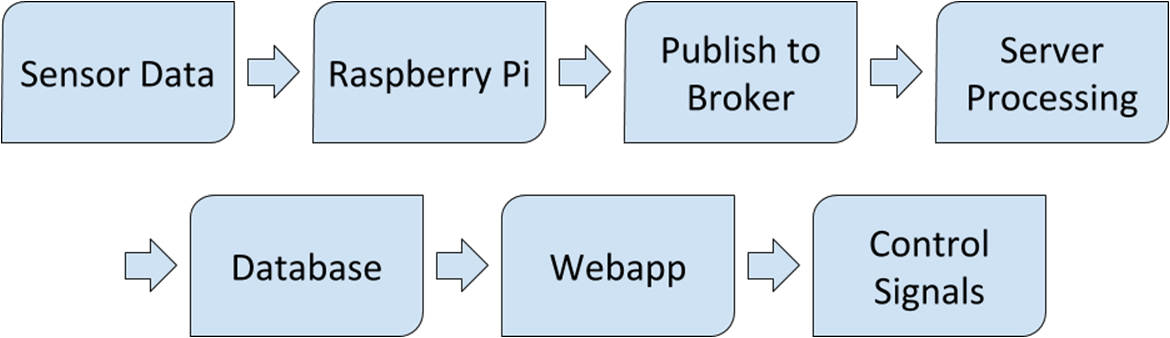
The cost of our prototype was roughly $85 in hardware parts, much of it coming from the Raspberry Pi 3 kit we purchased. This does not include the cost of the custom enclosure, which brings the total cost around $100. Adapting our design to use more specialized components and can greatly decrease costs, in addition to economies of scale offered by mass production.

Server overhead from the MySQL database, NGINX, and the platform server process was minimal when loaded with a single data node. The process takes significantly less than 5.0% memory and idles at under 1.0% CPU for a 1 GB single-core Ubuntu instance. Preliminary load testing shows 10.0% CPU utilization with memory holding constant. While large scale testing with metrics were not performed at this stage of the project, current scalability is acceptable. As of this report, a virtual server of this size would cost $5/month.

# **Summary and Future Work**

Recognizing the consumer costs associated with a smart home system, we sought to develop a more flexible and open platform for smart home devices to gather ambient data, communicate between themselves, and ultimately improve residential quality of life. Thus, for this capstone, cost considerations were a critical part of design.

In order to prototype our open smart home platform concept, we split the implementation between hardware and software.

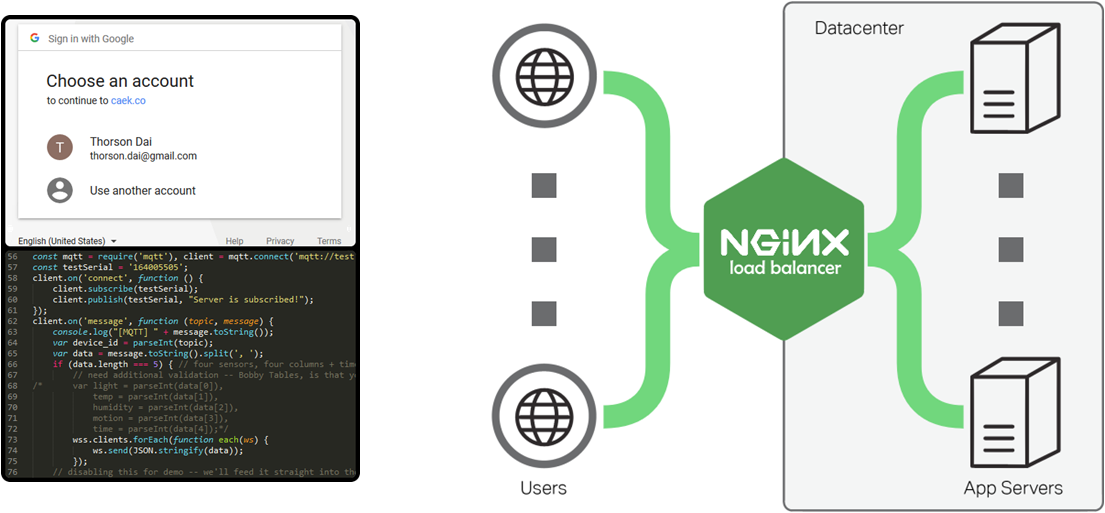


*Figure 14: Flow Chart of Operation*

*The entire chain of operation of the implemented platform prototype*

For hardware, we developed an ambient data collection node that has sensors for temperature, humidity, light levels, and motion detection. Calling on our drivers written in Python, the obtained values would be written to a named pipe. A Node.JS process would then read from the pipe and publish the data to a broker; in this case, one of many free MQTT test brokers available. The hardware cost of the design is under $100, and we are confident in being able to bring it down to around $25 per node in the future.

On the software side, processing was performed on the aggregated data obtained from the server which was subscribed to MQTT publish events. This involved running each of the ambient data types through their respective Schmitt trigger comparator through software. Through this, the state of various signals would be set high or low, thereby providing an automated sensor-dependent control scheme for smart home devices, in addition to a web application for settings and data visualization.

****

*Figure 15: Future Vision*

*Authentication through social login and load balancing through NGINX*

Future work for this capstone would most likely involve a redesign to reduce device footprint and cost. Additionally, it may be worth investigating an implementation of propagating control signals locally from data nodes.

Moreover, Security is a big concern in terms of IoT devices, and smart home devices are no different. If the Smart Home Platform were to be a viable commercial product, it would have a vital role in ensuring the secure storage and transmission of data, along with some form of device registration. One avenue we have examined but not yet integrated is using social logins for authentication, such as Google sign-in.

Lastly, in order to secure future scalability, we return to the current architecture using NGINX as our reverse proxy. This can be used effectively to distribute Smart Home Platform traffic across *multiple* servers. Our Hyper-V virtualization setup also makes it possible to create an environment to implement and test this distributed load balancing concept. This feature, although seemingly ancillary, is vital to the sustainability of this project.

# **Acknowledgements**

We would like to not only thank Professor Hana Godrich for her support but also Samuel Ramrajakar for his patient guidance in advising us.

# **References**

“15.1. os - Miscellaneous operating system interfaces¶,” *Python 2.7.14 documentation*. [Online]. Available: https://docs.python.org/2/library/os.html. [Accessed: 10-Mar-2018].

Adafruit Industries, “Adafruit Si7021 Temperature & Humidity Sensor Breakout Board,” *Adafruit*. [Online]. Available: https://www.adafruit.com/product/3251. [Accessed: 10-Mar-2018].

Adafruit Industries, “Adafruit TSL2561 Digital Luminosity/Lux/Light Sensor Breakout,” *Adafruit*. [Online]. Available: https://www.adafruit.com/product/439. [Accessed: 10-Mar-2018].

Adafruit Industries, “PIR (motion) sensor,” *Adafruit*. [Online]. Available: https://www.adafruit.com/product/189. [Accessed: 10-Mar-2018].

J. Boorstin, “Humans hooked on 21 billion of these by 2020,” *CNBC*, 01-Feb-2016. [Online]. Available: https://www.cnbc.com/2016/02/01/an-internet-of-things-that-will-number-ten-billions.html. [Accessed: 10-Mar-2018].

“Chartist.js,” *Chartist - Simple responsive charts*. [Online]. Available: https://gionkunz.github.io/chartist-js/. [Accessed: 10-Mar-2018].

C. Coyier, Preethi, G. Graham, R. Rendle, and G. Mainardi, “CSS-Tricks,” *CSS-Tricks*. [Online]. Available: https://css-tricks.com/. [Accessed: 10-Mar-2018].

“Raspberry Pi 3 Model B,” *Raspberry Pi*. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. [Accessed: 10-Mar-2018].

“Raspberry Pi Zero W,” *Raspberry Pi*. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-zero-w/. [Accessed: 10-Mar-2018].

“Using FIFOs in Python,” *timmurphy.org*. [Online]. Available: http://timmurphy.org/2013/11/11/using-fifos-in-python/. [Accessed: 10-Mar-2018].

“WebSockets,” *MDN Web Docs*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets\_API. [Accessed: 10-Apr-2018].

“WebSockets vs REST: Understanding the Difference,” *PubNub*, 06-Jan-2015. [Online]. Available: https://www.pubnub.com/blog/2015-01-05-websockets-vs-rest-api-understanding-the-difference/. [Accessed: 10-Apr-2018].