COMP 4905 Honours Project Report

Smart Home

Chris Ryan (100851198)

Supervisor: Prof. Louis D. Nel

Carleton University

School of Computer Science

April 12. 2016.

ABSTRACT

If a house were intelligent, in what ways would it provide services to its inhabitants? Would it be able to detect harmful levels of contaminants within the air or possibly adjust the temperature when people leave or enter the house? The goal of this project is to answer such questions through the development of 5 modular devices and the underlying systems that allow such devices to communicate with each other and users to provide a valuable service.

ACKNOWLEDGMENTS

I would like to thank my supervisor, Professor Louis Nel for guidance throughout the term. There are several libraries used throughout the project, without which it would not have been possible to complete within the timeframe, so I am grateful to all the authors who worked on these libraries and made them completely open to the public.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. **Introduction**

1.1. Motivation

The idea of a smart home is the networking of various devices to provide functionality directly through interfacing with said devices and indirectly through intelligent underlying systems that allow for communication between devices and utilization of collected data. Various companies are jumping on this smart home concept by developing their own internet enabled devices that deal with certain aspects of what a smart home could eventually become. Unfortunately there are several glaring problems with the approaches these companies are taking when trying to solidify their own stake in this relatively new market.

One such example is the philips hue, which is an ambient lighting system. Without warning philips issued an update that disabled certain bulbs that were previously considered compliant with the system, in an attempt to lock out competitors [1]. Another example is the revolv smart home hub, which will soon stop working completely (the devices will be bricked), simply because the parent company is moving on to different things and would rather entirely disable the device that customers paid for instead of leaving it as it currently stands [2].

These kinds of practices form distrust and hamper any kind of progress towards more useful systems. Users will not and should not purchase devices that could stop working entirely in the near future, especially when these devices are tasked with controlling important functionalities within a house, like managing temperature or lighting systems. Another problem is that with the litany of devices comes an interface that the provider implements, and since there is no full solution, users would need to deal with totally different interfaces for each system adopted.

Fortunately it is more feasible now than it even has been to make the home smarter yourself, as the availability and costs of hardware required to devise of such devices is relatively low. This project consists primarily of two sections, one being these devices (called modules) and the other being the hub. The hub is a node.js application and its purpose is to host a web app that will act as the interface for users and a broker that will act as an intermediary for all module communications.

1.2. Outline

The body of the report is primarily organized into topics that are of importance to the conceptualization of a smart home system. The following is a brief overview of each section.

Communications

There are essentially two paths in which communication can occur within this project, users will be communicating directly with the hub through a web page, and each individual module will communicate with users and/or other modules through the hub. It is clear that messages need to be sent and received in different ways with differing requirements so it is important to select the right application protocol for the job.

Modules (Smart Devices)

These are the physical devices of the project, they could be literally anything that provides some kind of functionality to the user. This chapter covers details pertaining to the 5 proposed devices that were built and how they interact with each other as well as the users.

Configuration

This chapter goes over libraries that were used and their purpose within the project. It also covers details regarding how the project is setup and the steps required to get everything working together properly.

Resources

Most of the actions that modules will perform are not particularly expensive, in terms of processing power, and so it is important to be aware of the resources required in each application to keep monetary and power consumption costs at an absolute minimum.

Security

For smart home systems, security is a hot topic because these devices operate within the privacy of your own home. There are many potential devices that you would not want to be accessible to strangers, such as a live feed camera, thermostat, or a baby monitor. So it is important to consider possible threats and methods to mitigate such threats.

User Interface

The user interface for this project is a single page web application. The goal was to create an interface that was easy to use and that presents all of the functionality that the project provides in a visually intuitive manner.

2. **Communications**

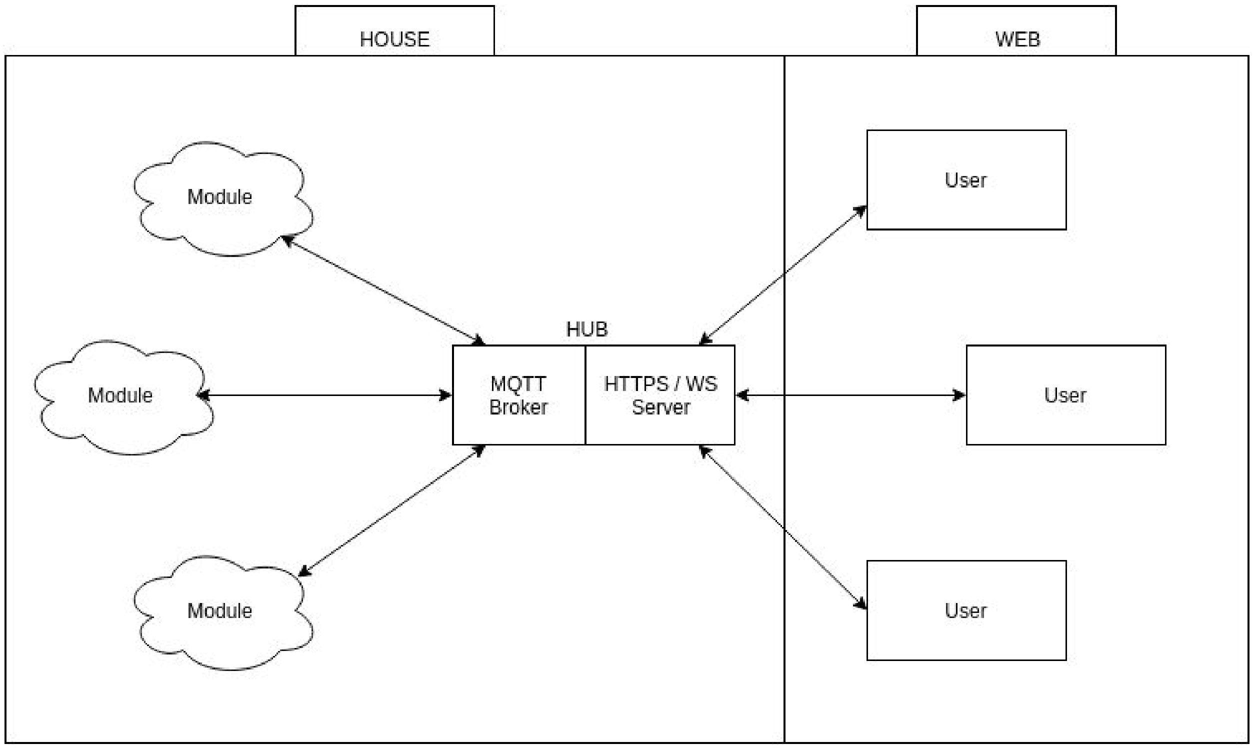


Figure 1: Communications Diagram

2.1. Web Users

From the web perspective, some static files (html, javascript, css) need to be served initially and then communication must be maintained between the user and the hub, because modules can send data at any point and the user should be able to interact with anything whenever they want to, so a combination of http and websockets are used to achieve this.

With http once a user makes a request, the data is transferred and then the connection is closed, which makes it perfect for serving static files as they can be requested once and then cached. Although there is no way to have real time bidirectional communication without constantly polling, resulting in an unnecessary overhead from connecting and disconnecting [3]. This type of communication is achieved using websockets, which work over a single socket and remain open for as long as needed [3]. So once the javascript files are received by a user via http, they are loaded by the browser, which sets up a websocket connection that all further communication between the server and the user will go through.

2.2. Module Clients

For module communications, the protocol needed to be lightweight to be fast and energy efficient. Http could have worked but as mentioned previously, the overhead of polling frequently on such low power devices would be too costly (in terms of both speed and power consumption) as it is necessary to maintain a connection for any module that takes an input from the user. Fortunately there is a protocol that is suited to such a scenario, which is called mqtt (formerly mq telemetry transport). Below is the abstract from the mqtt specification.

“MQTT is a Client Server publish/subscribe messaging transport protocol. It is light weight, open, simple, and designed so as to be easy to implement. These characteristics make it ideal for use in many situations including constrained environments such as for communication in Machine to Machine (M2M) and Internet of Things (IoT) contexts where a small code footprint is required and/or network bandwidth is at a premium. The Protocol runs over TCP/IP, or over other network protocols that provide ordered, lossless, bi-directional connections.” [4]

As is usual for publish / subscribe protocols, a broker must be used, which essentially act as an intermediary to all connected clients. It receives all messages that are published and pushes them to clients who are subscribed to that "topic" of message. Each message has one of three quality of services associated with it. Qos0 ("at most once") means that the message is delivered according to the best efforts of the operating environment, and message loss can occur. Qos1 ("at least once") ensures that the message will arrive but duplicates can occur. Qos2 ("exactly once") ensures that the message will arrive exactly once. [5]

As the quality of service level increases so does the overhead associated with the message so it is important to select the level that applies to the scenario [5]. For example the weather station would use qos0 since it sends messages frequently enough that another one will go through soon after, so message loss would not be a big problem. The registration message that is initially sent from all modules on startup would use qos1 because otherwise the module may begin publishing data when it is not actually registered.

2.3. Compatibility

Using mqtt for modules and websockets for users required some extra work to get them to operate together correctly, which is essentially adapter code for any functionality that needed to be exposed. The broker does not know that the web user exists (at least not in a client capacity), so when the server starts, the broker subscribes to the outgoing topics of every module and then when a message is published by a module, it is relayed through websockets to all users who have registered a message handler of the same name as the topic.

Another example of this adapter code would be for message retention, which is implemented by the broker. The idea is that any message published can be flagged for retention and then the most recent flagged message for a given topic is stored in memory so that when a client subscribes to that topic, the retained message is sent immediately [6]. This is useful because otherwise users would not see messages unless they are published while they are actively viewing the web page. Since users do not actually subscribe to topics through the broker, a work around was necessary, which is achieved by creating a websocket message handler so that when a user wants to subscribe, the topic name is sent to the server and the retained message for that topic is retrieved through the mqtt broker (if one exists) and emitted back to that user as though it had just occurred.

2.3. Modifications

It would have been much cleaner to just use mqtt directly in the browser but that is not possible at the moment as most browsers do not provide a method to open a raw tcp connection [7]. In the time being a better solution would be to use mqtt over websockets as that would remove all the unecessary adapter code, allowing users to act directly as mqtt clients [7]. This will obviously come with new concerns, especially with respect to security since allowing users to publish and subscribe with the freedom of a module could be problematic.

Mqtt is used exclusively on the module side, but it is important to note that it is not appropriate for all applications, it was not designed for continuously streaming large amounts of binary data (such as video streaming). One solution would be to also incorporate a different protocol along with streaming techniques for when a module needs to stream binary data and continue using mqtt for everything else (sensor data, controls, etc).

3. **Modules**

3.1. Hardware

When selecting hardware for the project, emphasis was generally on keeping the costs as low as possible, which comes with issues, such as very long delivery times and the possibility of knockoff or doa units. See Table 1 for a listing of all purchased hardware. Various components are used in the project but when formulating a module, a single piece of hardware is chosen as the “brain”, which in the case of this project was either a raspberry pi or esp8266.

Item	Quantity	Price per unit	Seller
Magnetic Reed Switch	2	1.15	Ebay
ESP8266 ESP-12E	4	3.3	Ebay
ESP-12E Adapter Plate	4	0.66	Ebay
FT232RL FTDI	1	3.18	Ebay
Photosensitive Resistance Sensor	1	1.44	Ebay
Soil Moisture Sensor	1	1.50	Ebay
Raspberry Pi 2 B+	1	N/A	N/A
4W Chassis & Speed Encoder	1	29.52	Ebay
LED	4	N/A	Ebay
4.7K Resistor	1	0.01	Ebay
L29D3 Motor Driver	1	1.30	Ebay
AMS117 Voltage Stepper	4	0.55	Ebay
DS18B20 Temperature Sensor	1	2.08	Ebay
Servo	1	N/A	Hobbyking

Table 1: Hardware Listing

The raspberry pi is a full blown computer (microprocessor, usb, hdmi), and would generally be used in modules that require more resources, such as image processing or heavy mathematics. It does have several gpio pins but since it is running an operating system (usually linux), another process could get priority over the cpu and cause it jitter. Also the raspberry pi does not have any wifi capability built in so a usb wifi dongle must be used. The esp8266 on the other hand is a microcontroller, which are ideal for interfacing with devices via gpio pins such as servos or sensors. It contains a wifi module (including antenna), which is unheard of at such a low price point.

There is a much larger emphasis on the esp8266, and modules build around it, throughout the project as the unit itself epitomizes the ideals of this project, whereas the raspberry pi certainly has useful applications, most do not require that amount of power. Throughout the rest of the report abbreviations will be used for the esp8266 (esp) and raspberry pi (rpi).

3.2. Overview

All of the physical modules are in a prototype configuration (breadboards, cable ties, etc), so they are very bulky and could probably fit in a much smaller space. The reason for this is that throughout the entire term tweaks were made to modules as they were needed, so it simply would not have been feasible to make anything permanent and so most modules are not practical for real world usage at the moment. The following is a brief description of each module that was created along with an image of the module in context.

Reed (esp): Attaches to anything that can be opened or closed and publishes a message each time the reed switches change state.

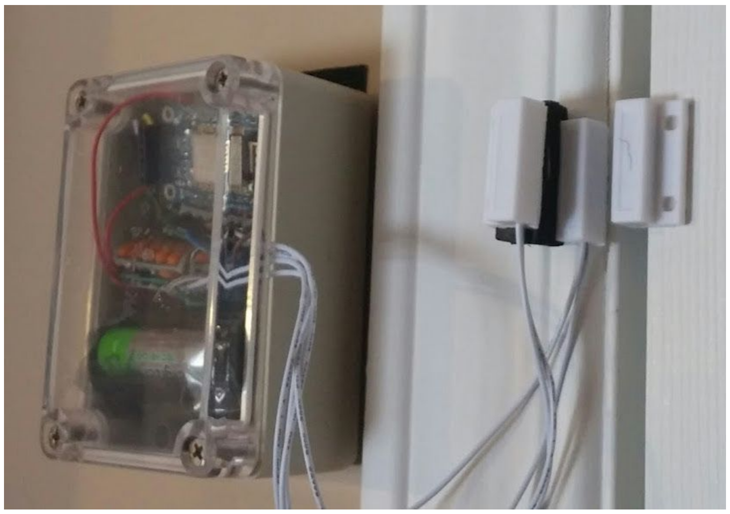


Figure 2: Reed Module

Weather station (esp): Periodically reads and publishes the temperature from a DS18B20 digital thermometer.

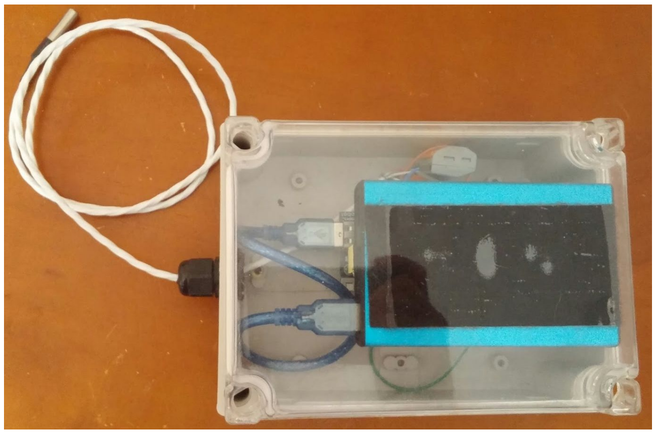


Figure 3: Weather Station Module

Hydroponics (esp): Potted plant that periodically publishes messages on the humidity levels of the soil and the strength of sunlight from a photoresistor. It also uses a water reservoir and tubing piped through a servo which acts as a valve that can be opened for a set interval of time.

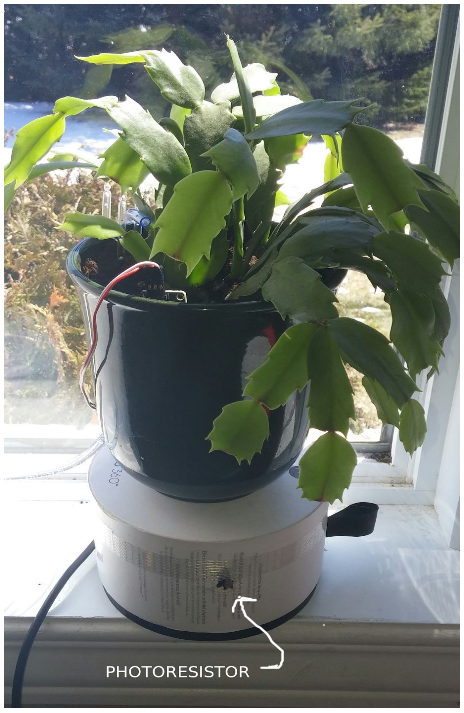


Figure 4: Hydroponics Module

Lighting (esp): Simple LED light that can be turned on or off.

Figure 5: Lighting Module

Sentry (rpi): Tri wheel driving platform with pi camera. The two rear motors are driven by a single L29D3 motor driver, which is controlled over gpio.

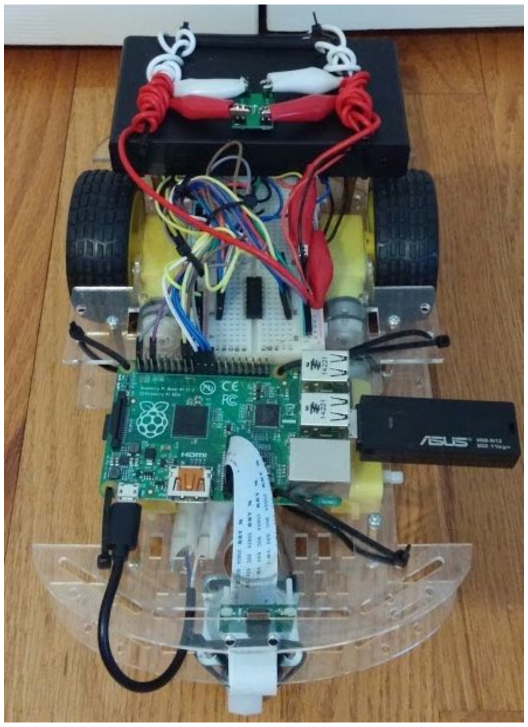


Figure 6: Sentry Module

Internally modules are treated as generically as is possible because they can be so many different things. At the most basic level, a module consists of incoming and outgoing message topics. Every module needs to be setup when it boots, which generally consists of the following steps.

- Read id stored on the module if it exists, otherwise generate the id (24 character hexadecimal string) and store it.
- Connect to wifi (if not already).
- Establish connection with broker.
- Publish registration message with module details (id, outgoing/incoming topics, type).
- Subscribe to all incoming topics (appended with id to target this module only).
- Execute module specific code.

Each module publishes messages to its outgoing topics in different ways depending on the application, for example, the sentry will publish camera images constantly, the reed will only publish a message when the door changes state, and the weather station will publish the temperature periodically (~ every 5 mins). Incoming topics are message topics to which a module subscribes, where the module will perform some action when a message is received, such as turning a light on or off.

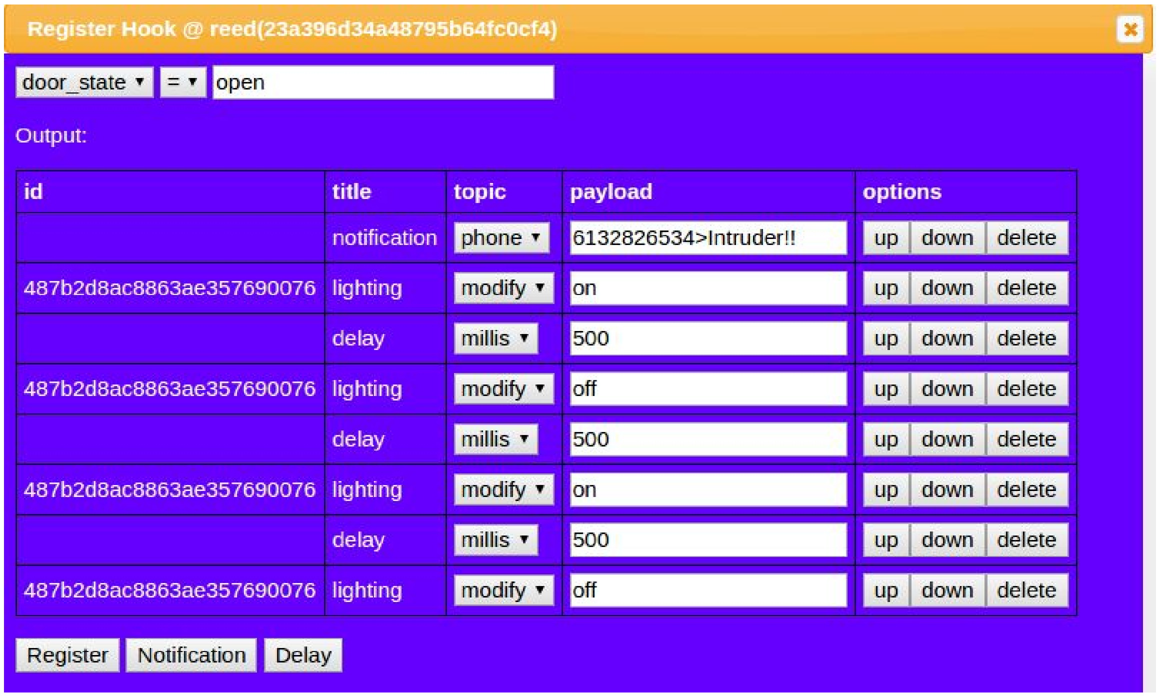
Currently all modules connect to a single router but as more modules are developed, this will become problematic because modules can be spread around the entire house and possibly even outside. One solution would be to use multiple routers around the house and then modules would simply connect to the router with the strongest signal. Another more interesting solution would be to use the modules themselves to build their own wireless infrastructure, where each non sleeping module could act as a repeater node since the esp can connect to and act as an access point for other modules.

3.3. Hooks

On their own modules are generally not intended to be particularly "smart" but when connected to each other quickly become more useful, which was the driving idea behind hooks. The hooks system allows users to take the output of a specific module (weather station->temperature, reed->door state, etc) and register any number of events that will trigger when a published message to that output topic is greater than, less than, or equal to some target value.

Currently there are 3 options for events, it can be a message published to the incoming topic of any module, a delay (in milliseconds), or a notification (email or sms). When the user registers a hook, all the details are stored in mongodb, then whenever a module publishes a message, any registered hooks associated with that module are retrieved, then if the value of the new message triggers a hook, all the hook's events are evaluated in order.

Through chaining of events it is easy to envision various automation tasks, which greatly simplifies creating new modules because a lot of the basic functionality that would be implemented on the module side can be easily generated in a user friendly way using these hooks. For example the hydroponics module allows users to turn the water on for a set interval of time and it also publishes the soil moisture levels, so a simple hook could check if the published moisture is below some value and if so turn the water on. Another example would be an intruder warning system, where whenever a door opens somewhere within the house an sms notification could be sent to your phone and the light would flash on and off. Fig. 7 shows how such a hook registration would look from the user interface.


Register Hook @ reed(23a396d34a48795b64fc0cf4)

door_state = open

Output:

id	title	topic	payload	options
	notification	phone	6132826534>Intruder!!	up down delete
487b2d8ac8863ae357690076	lighting	modify	on	up down delete
	delay	millis	500	up down delete
487b2d8ac8863ae357690076	lighting	modify	off	up down delete
	delay	millis	500	up down delete
487b2d8ac8863ae357690076	lighting	modify	on	up down delete
	delay	millis	500	up down delete
487b2d8ac8863ae357690076	lighting	modify	off	up down delete

Figure 7: Hooks Registration Dialog

The hooks system is relatively simplistic and could benefit from features in the future. One such feature would be to incorporate the date and time where the house is located, so that a hook could trigger at a time on a specified day of the week. Alternatively a modifier could be used where a hook will only execute if it is triggered at a certain time during the week, such as only during the day on weekdays for example.

4. **Configuration**

4.1. Libraries

Aedes (server js)

Implementation of an mqtt broker. It is bare bones but provided all the broker functionality that was necessary for this project. [8]

Esp8266 Core For Arduino (c++)

Allows you to program in arduino and upload sketches to the esp the same way that you would with an actual arduino (through the ide). Normally you would need to acquire and flash the firmware yourself and then upload lua code which would get executed by an interpreter on the esp, so this method is more efficient and familiar. This library can be installed directly from the arduino ide using the boards manager option. [9]

Pubsubclient (c++) / Paho - Mqtt (python)

Client libraries for interacting with the mqtt broker. [10][11]

Jquery (client js)

Used primarily for simplifying HTML document manipulation and event handling. [12]

Jquery UI (client js)

Set of user interface interactions, effects, widgets, and themes built on top of the jquery library. [13]

Mongoose (server js)

Mongodb object modeling tool. [14]

Arduino Temperature Control Library (c++)

Used to interface with the DS18B20 one wire digital thermometer used in the weather station module. [15]

Twilio - Node / Nodemail (server js)

Used to send notifications via sms and email respectively. [16][17]

Socket.io (server & client js)

Used to create websockets for real time bidirectional communication with users. [18]

Node - Minify (server js)

Compresses javascript files into a single file, stripping unnecessary information and reducing content where possible. Having a single compressed file is much faster because only one request needs to be made to the server. Minification is also useful because it obfuscates the client javascript so that users might have a harder time determining functionality. [19]

RPi.GPIO (python)

Provides access to the gpio pins of the rpi which is used to communicate with the motor driver in the sentry module. [20]

Picamera (python)

Provides an interface to the rpi camera module, which is used to capture picamera images in the sentry module. [21]

ArduinoJson (c++)

JSON library for usage in embedded systems. [22]

Babel Core / Babel Preset ES2015 (server js)

Many updates were made to javascript with ES6, but support for it is still very much underway. This library is used to allow javascript to be written in ES6 and then transpiled to ES5 at launch. [23]

Snap.svg (client js)

Used to manipulate scalable vector graphics (svg), in particular the layout image for the house. [24]

4.2. Setup

All work on this project was done on linux machines so the setup will probably be slightly different if another operating system is used. On the server machine, you will need nodejs and mongodb installed. After copying the node.js project files ("Hub" directory) to that machine, run the command "npm install", which will install all libraries listed in the package.json file. Running the command "node app.js" will generate code (transpile & minification) and then start the servers. Note that the transpile step is destructive (original client javascript files are replaced).

Uploading code to modules is a bit tedious at the moment. For the rpi you will first need to setup wifi (using a usb dongle) and then start using ssh to wirelessly access the rpi. You will need to transfer the python files and then you can setup a script that runs the module boot script each time the rpi starts up. All python libraries can be installed via pip (package manager).

For the esp you need to connect specific pins to a usb to serial interface (such as the FT232RL) and then you can upload the code directly via the arduino IDE. Newer esp models have a built in interface (to micro usb) and voltage steppers so those are obviously easier to work with. Arduino libraries need to be installed by hand, placed in the libraries folder (default is /home/Arduino/libraries). The entire Pubsubclient library ("Libraries/HubClient") was included in the project submission because modifications were made to the original library, namely creation of the HubClient class which extends PubSubClient, so that folder must be placed in the arduino libraries directory as well.

The web page will be accessible from the local ip address of the device that is running the server while connected to your local network (ex. "https://192.168.0.131:1337") or from your public ip while outside of your network (ex. "https://99.240.210.44:1337"). Also you will need to port forward (1337) in your router's settings. Lastly a generated client certificate ("Hub/ssl/personal-crt.p12") must be installed in your browser to make a secure connection to the server. At the moment some credentials are stored as literals in code so a few changes will be necessary to get everything working properly.

Server's Local IP Address

- HubClient.cpp (line 4)
- boot.py (line 9)

Wifi Username & Password

- HubClient.h (lines 27-28)

4.3. Code Structure

The server side code is divided up into several node.js modules (see Hub/lib directory) that essentially deal with different functionalities (minification, notifications, schemas, etc). On the client side, a namespace is created initially to avoid polluting the global namespace and then features are attached to that as needed (see Hub/client directory). Module dialogs are generated through prototypal inheritance so that basic modules will not require any extra client side code whatsoever, which was the case for the weather station and reed modules.

For relatively simplistic modules only a single source file needs to be written because all the setup details and shared functionality have been abstracted away to some degree as it should be exactly the same for all modules. For modules written in python, there is a boot script and for modules written in c++, the client class from the pubsub library was extended to include these details. This allows the module code to be very compact, for example the main source file that is flashed to the reed module is less than 26 lines of c++ code.

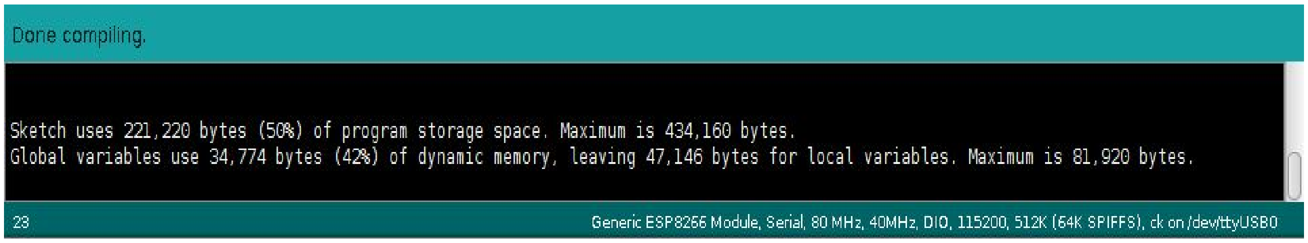
5. **Resources**

5.1. Memory

Generally with respect to modules, the only storage necessary will be some basic configuration details, as when larger storage needs do arise for a particular application, the server should be modified to account for those details. Storage on the rpi is relatively simple as it has a filesystem so a json file is used.

The esp uses eeprom (electrically erasable programmable read-only memory), where you need to read and write specific ranges of bytes. Since the only piece of data that needs to be stored in config at the moment is the mongodb id, it is simple to just reserve the first few bytes of the eeprom to account for that. In the future, when more space is needed for other data, such as credentials, there will need to be a detailed mapping of how the eeprom is split up into byte ranges.

The esp has a very limited eeprom size but you also have to be very conservative when writing code as well, ensuring that dynamic memory is not wasted because it can quickly run out. Fig. 8 is a screenshot of the output for simple arduino sketch uploaded to the esp which effectively illustrates this point.



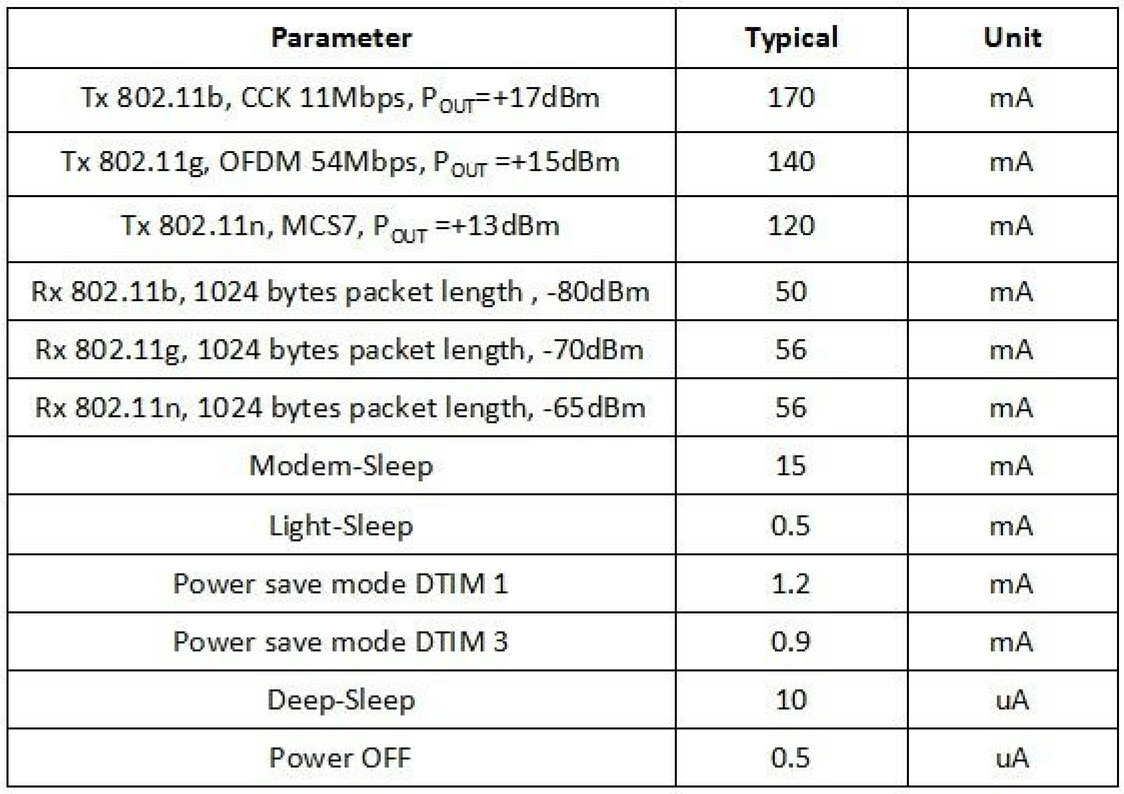
```
Done compiling.

Sketch uses 221,220 bytes (50%) of program storage space. Maximum is 434,160 bytes.
Global variables use 34,774 bytes (42%) of dynamic memory, leaving 47,146 bytes for local variables. Maximum is 81,920 bytes.

23                    Generic ESP8266 Module, Serial, 80 MHz, 40MHz, DIO, 115200, 512K (64K SPIFFS), ck on /dev/ttyUSB0
```

Figure 8: Screenshot of Compiled Arduino Sketch

5.2 Power Consumption

When it comes to the rpi, power consumption is complex as it is an entire computer, so it would be necessary to individually consider everything hooked up to the unit (gpio, usb devices, cameras) and so its power consumption was not much of a consideration for this project. Although it would certainly be beneficial to be aware of the cost of connected components when determining a suitable power source and to disable anything that is not necessary for a given module as newer rpi models can draw upwards of an amp under stress. [25]

For the esp there needed to be a large emphasis on conserving power because most applications that would make use of the esp will be using batteries (as little as 2 AA batteries) and need to function over long periods of time without having to be recharged or have the batteries replaced. See Table 2 for test data on esp power consumption.



Parameter	Typical	Unit
Tx 802.11b, CCK 11Mbps, P_{OUT}=+17dBm	170	mA
Tx 802.11g, OFDM 54Mbps, P_{OUT} =+15dBm	140	mA
Tx 802.11n, MCS7, P_{OUT} =+13dBm	120	mA
Rx 802.11b, 1024 bytes packet length , -80dBm	50	mA
Rx 802.11g, 1024 bytes packet length, -70dBm	56	mA
Rx 802.11n, 1024 bytes packet length, -65dBm	56	mA
Modem-Sleep	15	mA
Light-Sleep	0.5	mA
Power save mode DTIM 1	1.2	mA
Power save mode DTIM 3	0.9	mA
Deep-Sleep	10	uA
Power OFF	0.5	uA

Table 2: [26] Test Data of Power Consumption of ESP8266

Power consumption proved to be an issue early on, and while working on the reed module, a model for handling such requirements quickly became necessary, which was then applicable to all battery powered modules. Initially, without any modifications to the esp, the reed module was constantly publishing the state of the reed switch to the hub, and so it was very responsive but it burned through two AA batteries (~1300mAh each) within 24 hours, which is clearly unacceptable.

Maintaining a wifi connection is expensive, so you only want to actually connect to wifi when there is new data to publish or after a certain interval of time has passed (depending on the application). In the case of the reed module this is when the door is opened or closed, but by setting up wifi and connecting to the broker for every message means that there is a delay from when the reed switch changes state to when the message is actually published, which is roughly a few seconds.

Originally to determine if the reed switch has changed the gpio pin was constantly polled, but this is bad because it changes so rarely, but if you poll less frequently then the module becomes significantly less responsive. Fortunately the esp has sleep modes that make these modules viable on battery power, these 3 modes are modem, light, and deep sleep [26].

While in deep sleep mode, the esp draws a mere 10 uA, which means that on just 2 AA batteries the unit will remain operational (without ever waking up mind you) for several years. From deep sleep the esp can be awoken by providing a pulse to the reset pin, or after a set interval of time, which requires connecting a specific gpio pin (model dependent) to the reset pin so the pulse can be generated internally after the interval. The weather station uses the latter method, since weather changes slowly, the sleep interval can be quite large (~5 mins). [26]

The reed module proved to be more challenging as the pulse needed to be generated each time the door opened or closed. Fortunately a reed switch itself can be used to generate this pulse because when a magnet is over the reed switch it is a closed connection and when the magnet is removed the connection is opened, see Fig. 9 for illustration. By attaching one end of the reed switch to ground and the other to the reset pin, the module will wake every time a magnet passes over the reed switch.

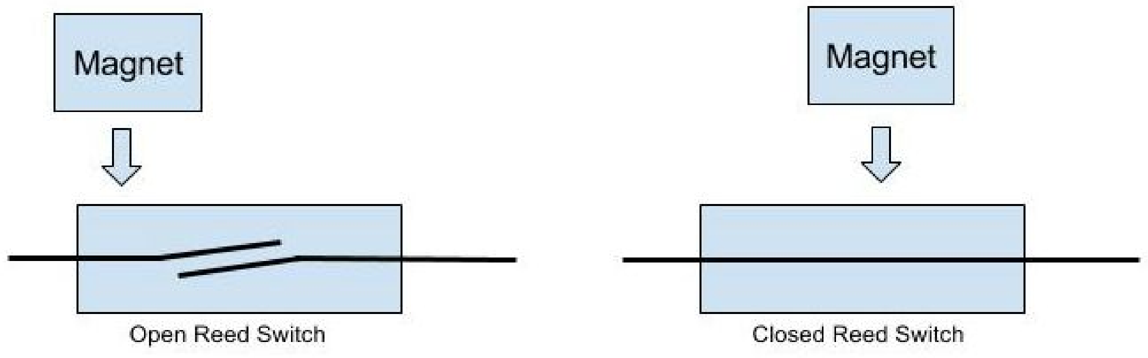


Figure 9: Reed Switch

Now unfortunately this will not tell you if the door is being opened or closed, just that the state has changed. So to solve this problem two reed switches are stacked ontop of eachother, where the one further away from the door acts as the module reset switch and the one closer to the door is checked when the module wakes to determine if the door is closed or open.

Deep sleep is usable for most modules that do not subscribe to any incoming message topics because otherwise they would need to remain connected to the broker to stay subscribed to those topics. However there are situations where responsiveness is so important that the module cannot sleep even though incoming messages are of no concern, such as the detection of dangerous gases or other hazard warning devices.

6. **Security**

6.1. Overview

Https and secure websockets are used ensure that all web communication is authorized and encrypted. The authentication is through usage of client certificates, where the server challenges the client with a certificate request during the TLS handshake. The certificates are generated and self signed using openssl with generated components (keys, certificate authority, etc). Also a certificate revocation list is used so that if a particular certificate is being used improperly it can be revoked at any point. [27]

On the module side, the focus was simplicity and speed over security, as there is always an overhead with security and when dealing with very power limited devices like the esp it becomes problematic. With that said, even though this setup would not be secure enough for real world usage, it is beneficial to quickly go over why it is insecure and detail some of the possible changes that could be made to improve security.

6.2. Methods of Attack

Now to consider the ways in which an attacker may try to attack the system, there are these wifi enabled devices that communicate directly with the mqtt broker which occurs entirely over a single wifi router, so the first step would be to ensure that the router is secure (WPA2, firewall, etc). Furthermore the router and operating system of the machine running the servers should be configured to suit the needs of the system that is being built to help minimize the ways in which it can be attacked. For example, mqtt works entirely over tcp so there is no point in allowing udp packets whatsoever. Another example would to block access to all ports that the mqtt broker is not using. [28]

If someone were able to gain access to the router, they would be able to do quite a bit of damage as they would have direct access, since there is no mqtt client authentication taking place as of yet, so they could connect to the broker as an mqtt client. This means that they could not only subscribe to all messages that are published but also act as a module themselves publishing whatever they want. Since modules communicate in a very predictable manner it would be possible to determine if a message did not originate from a module. So one improvement would be to implement abnormal message detection, emitting a notification in such a situation.

Another issue is where the attacker gains physical access to any module, which is a very real concern because some modules would operate outside of the house, such as the weather station.

Since the rpi is running an operating system (namely linux) it can essentially be as secure as it needs to be. Unfortunately for the esp, physical access is problematic because the esp provides no code reading protection. With access to the code the attacker has access to any authentication details that could be used and with that the entire system is exposed.

Various methods could be employed to obscure the credentials on the esp, such as storing them encrypted in eeprom rather than as a literal in the code. Unfortunately it will always be possible to reverse engineer any method(s) that are used since the module itself needs to operate entirely on its own. The obvious solution here would be to simply use a different microcontroller that does provide more protection and use the esp solely as a wifi access point.

7. **User Interface**

7.1. Layout

The background of the web page consists of a layout for the house. To generate the layout an android application called MagicPlan was used, which makes it easy to map out the entire house simply by scanning from room to room. The format exported from the app is an svg (scalable vector graphic), which is an xml media type used in applications which require scalable static or interactive graphic capability [29].

Unfortunately there is very little descriptive information stored within the svg, so it is not possible to ascertain which elements form components, such as windows or doors, within the layout. The svg file is comprised of many path elements which are used to represent the outline of shapes within the image [30].

After determining the subset of paths within the layout that represented the components, it was possible to simply merge all intersecting paths (by looking at their bounding boxes), as a single component can be composed of many paths. Then handlers are registered so that when a component is clicked, the merged bounding box representing that component is highlighted. Fig. 10 shows the selection of a door within the layout.

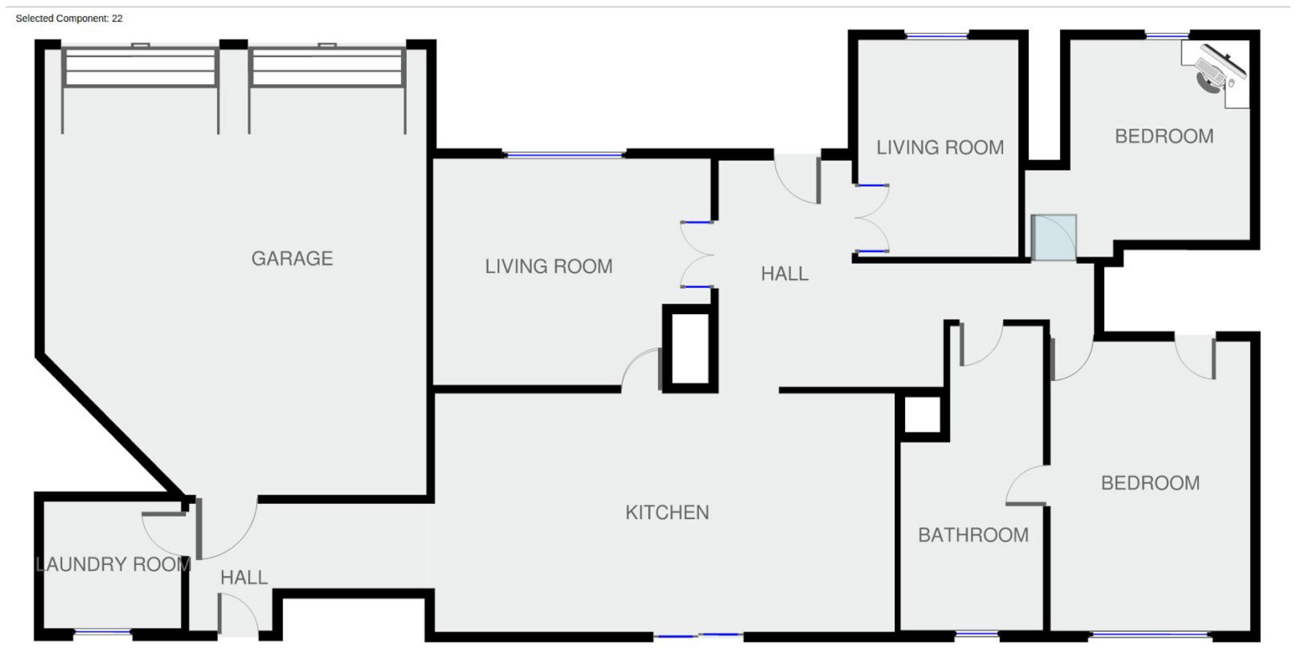


Figure 10: House Layout

Now this method of parsing is very specific to the MagicPlan app so to support other applications or formats, a different method of parsing would need to be considered. Also there are certain components in the app that will not parse correctly, such as stairs, since they do not contain intersecting components (they are segmented).

7.2. Dialogs

Jquery UI dialogs are used for most interaction within the web page. There is the modules dialog that contains a table with rows for each registered module, another for hook registration, and a dialog for each individual module. Using dialogs made the interface relatively clean because the user can open exactly what they want to see while keeping everything else hidden.

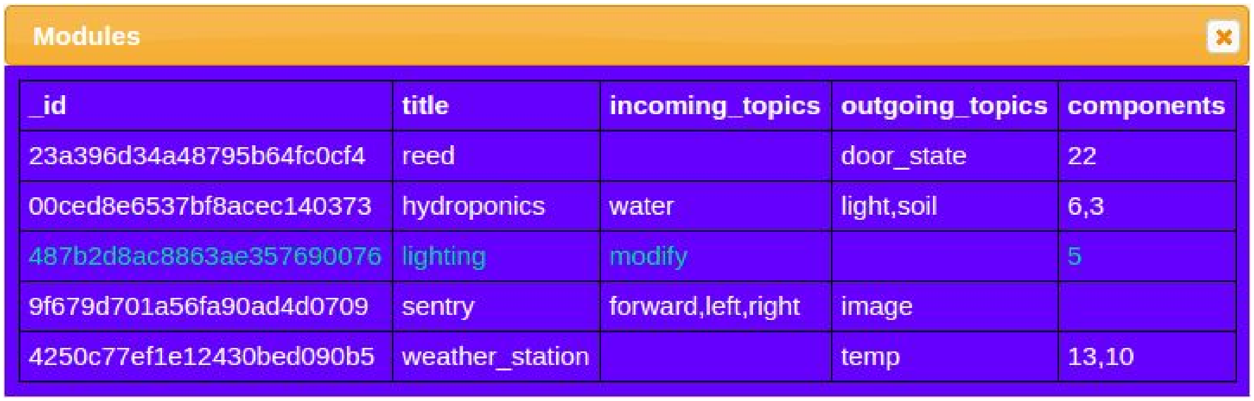
The module table dialog is used to open each individual module dialog, where the user can select the row for a module dialog they want to open or close. One important note is that by selecting a module row, the user is subscribing or unsubscribing to that modules outgoing topics. This was done so that the user does not receive all module messages as there could be many modules which could create a lot of traffic, especially with modules that send large amounts of data quickly, such as the sentry module.

Each module dialog contains a “hook in” and possibly “hook out” (if the module has incoming topics) button. The “hook in” button opens the hooks registration dialog and populates a dropdown option with the outgoing topics of the selected module. The “hook out” button adds a row to the registration dialog’s event table, which represents a single payload sent to a specified topic for that module. See to Fig. 7 for illustration.

7.3. Location

Knowing where a module is located within the house is necessary as many modules will depend on it. For example, if you cannot know which door a reed module is associated with, it loses almost all purpose. Keeping track of where a module is located is entirely handled through the user interface, meaning that each module has no inherent knowledge of where it located within the house, so it is up to the user to specify its location.

Each module dialog contains a place and remove button, which will add or remove the currently selected component within the layout to the module’s list of components. Then when a component is selected, each row for any module that was placed in that component is highlighted. See Fig. 11 for illustration. Now this works fine for modules that are static but would not for modules that move around the house, such as the sentry module. The location of these types of modules was not a consideration for this project but one way this could have been achieved would be through GPS, although that may be too imprecise for indoors usage.


Modules

_id	title	incoming_topics	outgoing_topics	components
23a396d34a48795b64fc0cf4	reed		door_state	22
00ced8e6537bf8acec140373	hydroponics	water	light,soil	6,3
487b2d8ac8863ae357690076	lighting	modify		5
9f679d701a56fa90ad4d0709	sentry	forward,left,right	image	
4250c77ef1e12430bed090b5	weather_station		temp	13,10

Figure 11: Module Table Dialog

8. **Conclusion**

8.1. Analysis

Most of the features that I originally set out to complete are integrated now at the end of the project. I believe that the project demonstrates that building a home smart is not only possible but can be achieved at a reasonably low cost. There are obviously many modifications and improvements to be made but that is generally the case for projects of large scope. Below are a few features that I had hoped to get working but turned out to be more complex than I had originally expected.

The web api polling section of the hooks system was not integrated. Initially I believed that there would be a relatively generic way to deal with retrieving data from various open apis but this proved to be more difficult. The problem is that most apis require keys to interact with them, which means that config details need to be considered as well, which would be a bit much to ask of an end user and would be challenging to make user friendly. It would probably be more appropriate to simply add support for various apis to the project itself, possibly treating apis as virtual modules.

Being able to reset a module would have been useful, particularly during development. Resetting would result in the mongodb document being removed for that module and then the module itself would be reset, erasing any stored config details and then triggering a registration. This would be easy for modules that are always connected to the broker, like the lighting or sentry, but modules that only have outgoing topics, like the reed or weather station, only connect to the broker to publish messages then immediately sleep.

One possible hardware solution would be to integrate a physical reset button into each module. A software solution that I contemplated was the idea of a message queue. Messages sent to particular modules could queue up in the server and then when a module wakes to publish a message, it could wait for the queue of messages to be delivered before going back to sleep. This technique would also have the benefit of allowing all modules to be configurable by the user. For example the weather station wakes on an interval which is hard coded but with this users would be able to set that interval, storing the details in eeprom rather than in code.

Lastly a recommendation that I would make to anyone attempting a similar project is to try to stick with languages and tech that you are familiar with if possible, especially if you have a relatively short timeframe. This project was spread over many libraries and languages, so working in contexts in which I was unfamiliar or rusty certainly took a toll. Although I am pleased with the choices I made because the project has been and continues to be a great learning experience.

8.2. Future Work

The modules of this project are totally open ended and the possibilities for them are literally endless. Future work would see improvements and modifications to existing modules and development of entirely new modules. The modules in this project are relatively simple, usually focusing on the data provided from a single sensor or performing some basic functionality. This was necessary to get enough of them finished to effectively test the system being built but there is no reason that modules cannot be so much more. For example the weather station could be extended to deal with sensors to detect rainfall, wind speed, humidity, etc.

One intriguing idea for a module was a mobile application, that could act as a module and an interface simultaneously. As a module it could be a way of determining who is within the house (one of many possible functions) and as a user interface it could provide a mobile friendly experience through gestures and possibly even voice recognition.

Improvements to the UI could be made to be more interactive which would provide a more user friendly experience. This could be handled through usage of the snap svg library as there are many capabilities with respect to svg animations. For example, once a reed module is assigned to a door, and it publishes state, the component within the layout svg of the door could actually animate open or closed.

Lastly a great future consideration would be to deal more with the data being provided by the modules, which was something that originally intrigued me. Even simply generating graphs to show how a module performed throughout the day would be incredibly useful for developers and users as a visualization tool.

REFERENCES

[1] Bruce Schneier, “How the Internet of Things Limits Consumer Choice”, http://www.theatlantic.com/technology/archive/2015/12/internet-of-things-philips-hue-lightbulbs/421884/, Published Dec 24. 2015.

[2] Rob Price, “Google's parent company is deliberately disabling some of its customers' old smart-home devices”, http://uk.businessinsider.com/googles-nest-closing-smart-home-company-revolv-bricking-devices-2016-4, Published Apr 4. 2016.

[3] Peter Lubbers & Frank Greco, “HTML5 WebSocket: A Quantum Leap in Scalability for the web”, Websocket.org, http://www.websocket.org/quantum.html.

[4] Andrew Banks & Rahul Gupta, “MQTT Version 3.1.1 Plus Errata 01”, https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html, Published Dec 10. 2015.

[5] Andrew Banks & Rahul Gupta, “MQTT Version 3.1.1 Plus Errata 01: Quality of service levels and protocol flows”, https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.html#_Toc442180912, Published Dec 10. 2015.

[6] Andrew Banks & Rahul Gupta, “MQTT Version 3.1.1 Plus Errata 01: MQTT Control Packets (3.3.1.3 RETAIN)”, https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.html#_Toc442180851, Published Dec 10. 2015.

[7] HiveMQ, “MQTT Over Websockets”, http://www.hivemq.com/blog/mqtt-essentials-special-mqtt-over-websockets.

[8] Matteo Collina, Aedes MQTT Broker (Version 0.12.4), https://github.com/mcollina/aedes.

[9] ESP8266, ESP8266 Core For Arduino (Version 2.1.0), https://github.com/esp8266/Arduino.

[10] Ian Tester, Pubsubclient (Version 1.99.2), https://github.com/Imroy/pubsubclient.

[11] Eclipse Foundation, Paho - Mqtt (Version 1.1), https://pypi.python.org/pypi/paho-mqtt/1.1.

[12] jQuery Foundation, jQuery (Version 1.11.3), https://jquery.com/.

[13] jQuery Foundation, jQuery UI (Version 1.11.4), https://jqueryui.com/.

[14] Automatic, Mongoose (Version 4.3.7), https://github.com/Automattic/mongoose.

[15] Miles Burton, Arduino Temperature Control Library (Version 3.7.6), https://github.com/milesburton/Arduino-Temperature-Control-Library.

[16] Twilio, Twilio - Node (Version 2.8.0), https://github.com/twilio/twilio-node.

[17] Nodemailer, Nodemailer (Version 2.0.0), https://github.com/nodemailer.

[18] Socket.IO, Socket.IO (Version 1.4.4), https://github.com/socketio/socket.io.

[19] Rodolphe Stoclin, Node - Minify (Version 1.3.7), https://github.com/srod/node-minify.

[20] Ben Croston, RPi.GPIO (Version 0.6.2), https://pypi.python.org/pypi/RPi.GPIO.

[21] Dave Jones, Picamera (Version 1.10), https://github.com/waveform80/picamera.

[22] Benoit Blanchon, ArduinoJson (Version 5.1.1), https://github.com/bblanchon/ArduinoJson.

[23] Babel, Babel (Version 6.7.4), https://github.com/babel/babel.

[24] Snap.svg, Snap.svg (Version 0.4.1), http://snapsvg.io/.

[25] Raspberry Pi, "FAQS: Power", https://www.raspberrypi.org/help/faqs/#power.

[26] Espressif (Kelly), "ESP8266 Power Consumption", http://bbs.espressif.com/viewtopic.php?t=133, Published Jan 12. 2015.

[27] Anders Brownworth, "HTTPS Authorized Certs with Node.js", http://engineering.circle.com/https-authorized-certs-with-node-js/, Published Feb 03. 2015.

[28] HiveMQ, “Securing MQTT Systems”,
http://www.hivemq.com/blog/mqtt-security-fundamentals-securing-mqtt-systems.

[29] World Wide Web Consortium, “Appendix P: Media Type Registration for image/svg+xml”,
https://www.w3.org/TR/SVG/mimereg.html, Published Aug 16. 2011.

[30] World Wide Web Consortium, “SVG Paths”, https://www.w3.org/TR/SVG/paths.html,
Published Aug 16. 2011.