SKYLUX

Smartphone Controlled Skylight

California Polytechnic State University, San Luis Obispo

Computer Engineering Senior Project, Winter 2018

James Green (jgreen23@calpoly.edu)

Advisor

Richard Murray (rimurray@calpoly.edu)

ABSTRACT

There are numerous electric skylight openers available for purchase for home-use, but the majority of them are remote based, or operated by a wall-unit. Furthermore, these devices are in hard to reach places, so if one were to lose the remote on a remote operated system, the only option is to contact the manufacturer for a new device. As such, my senior project, in collaboration with Colton Sundstrom’s senior project, build upon our existing capstone project in order to allow operation of the Internet of Things (IoT) device over the internet. Our client, Richard Murray, was unsatisfied with the current state of his Velux skylight operator, so we wanted to create an end-product that could allow the operation to be controllable with any of his iOS devices, whether or not he was at home. Furthermore, the end-product should be secure, in that only authorized users could control the device. Since my client deemed that this project is very expansive, this project will also include a setup guide for new students to get the system running on their own environment for further improvements in areas such as scheduling, security, and more expansive features.

ACKNOWLEDGEMENT

A special thanks to Colton Sundstrom, a recent Cal Poly Computer Engineering graduate. His technical advice and direction of the web server modified and used in this project.

PROJECT OVERVIEW

This project covers the design and development of an iOS application for integration with a Raspberry Pi Zero-W controlling a Velux motorized skylight opener. The Velux operator normally depends on remote operation, so if the device is lost or broken, the only way to communicate with the device is to physically interact with it in a hard-to-reach location. My Capstone group in June 2017, called Skylux, created a prototype version of a device which was intended to be an Internet of Things (IoT) device. However, there were some shortcomings with the capstone project that, due to lack of specialization when working on the project, did not get enough dedicated development time. With this project, I have brought the software side of the Skylux system from an initial, alpha-level prototype stage to a ready-to-use beta application. The end product is an iOS application that utilizes a REST API in Swift 3 and Xcode 8 using key Swift functions such as URLSession And JSONSerialization. Through the project I worked with one of the Skylux members, Colton Sundstrom. His senior project involved designing the backend server, which my iOS device interacted with. I then took over his side of the project and integrated it with mine, and created a detailed user setup guide along with instructions for any future capstone group to take over the project.

The hardware consists of a Raspberry Pi Zero-W and motor driver that are hard-wired to control a Velux Skylight Operator (See Appendix A). The Pi is also running a Python backend, which is handled through the paho-mqtt python library by eclipse. When the server is brought online, it subscribes to mqtt topics on the server, which adds device numbers to the database and opens connections to that device topics. Once the operator receives the command from the topic it is listening to, then the device can be opened or closed.

The software consists of an iOS application acting as a MQTT client built using XCode 9 and Swift, interfaced with a server running a MQTT broker, RESTful api, and a SQLite database for the devices.

BACKGROUND

Project Goals

Originally, Skylux relied on a rudimentary web-view of a webserver hosted on the raspberry pi. As a result, the system must have been already running and set up beforehand, which required a lot of technical knowledge and dismantling of the system. The primary goal of my senior project was to move the system from this solution into a more integrated system. The goal will be to mimic the Amazon Alexa application, which has functionality when not connected to the home's WiFi network. The end goal of my project was to involve a server running on the Skylux operator while the iPhone application accesses this application when on a mobile network. This would have involved a total overhaul of the old solution, but will have used existing frameworks and strategies for operation.

Reflecting on the goals I had set out for myself at the beginning of the quarter, I would deem this project as mostly a success, with some minor drawbacks. The project can now be set up anywhere with an internet connection. As long as the user has access to a computer that can run Python, as well as administrator privileges on their network, the server can be setup to run the messaging portion of the system. On the hardware side, the current drawback is that the user must connect to the Pi with a display in order to configure the WiFi settings. Through discussion with my advisor, it was deemed that the most important aspects of the project were to create a detailed user guide with the current implementation, so that another team of students can take on the task of the WiFi handoff. This decision was based on the fact that this step could involve more hardware along with extensive knowledge of security topics, and I was unfamiliar with both topics.

MQTT Protocol Research

The general MQTT terms include publish, subscribe, client, and subscriber. According to HiveMQ[1], "Publish/Subscribe decouples a client, which is sending a particular message (called publisher) from another client (or more clients), which is receiving the message (called subscriber)." In this case, the physical operator will be an instance of an MQTT client, defined as "any device from a micro controller up to a full fledged server, that has a MQTT library running and is connecting to an MQTT broker over any kind of network." So the client will be communicating to the server, the broker, which will filter the messages and send the commands to the Skylux operators.

MQTT is ideal for this environment because we are running on a device which is intended to be very passive, in that it will require little to no interaction from the user past the setup point. From its website, "MQTT is a machine-to-machine (M2M)/"Internet of Things" connectivity protocol.

[1] "MQTT Essentials: Client, Broker and Connection Establishment
" http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment. Accessed 6 Nov. 2017.

It was designed as an extremely lightweight publish/subscribe messaging transport. It is useful for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium."[2] In our case, this seemed like the logical next-step as an improvement from before, as our old solution required much more network operation and a larger code footprint. In this new environment, the code footprint is much smaller, and allows for streamlined communication from the iOS device in code to the operator via a simplified messaging service.

API REST Implementation

The Flask python library has an extension that allows for development of REST APIs. REST, or REpresentational State Transfer, is a method for creating standards between computer systems on the web, making it easier for different systems to interact with one another. Rest is defined by a system that is stateless, separate from the client, and has a uniform interface. The system contains commands such as GET, POST, PUT, DELETE, allowing data to be received and sent to the server. In my case, I took over the server from Mr. Sundstrom and moved the domain to `www.skyluxservices.xyz`. Once the code for the server is running, which will be detailed in the appendices, the server will be running the API on the local network over a dedicated port. From there, my iOS application, the client, will issue PUT and POST commands to login and operate the server, along with GET commands to check on the status of the device.

JWT Authentication: Logging In

JSON Web Token (JWT) is a lightweight and efficient way to securely transmit data as a JSON object. The data transmitted is signed, using an algorithm or a public and private key pair. This verifies the authenticity of the token, allowing it to be used for authentication. JWTs are primarily used for this purpose, authentication, or for information exchange. As the tokens can be verified after being sent, this serves as the method of logging into the service from the iOS device. Once the user associates a username and password to the device, that information is sent to the server and a JWT is associated with that information. Therefore, when the user logs into the system, they can provide a JWT, which will be saved on the device, in order to operate the skylight operator in a secure manner.

Swift: iOS Communication with REST Server

Since my app communicates with a web application, information returned from the server is formatted as JSON. I used the JSONSerialization[3] class to convert JSON into Swift data types in order to better utilize them. It was certainly a challenge collaborating with Colton to deserialize the JSON objects correctly, and required a lot of work on both of our ends. It was definitely a

[2] "MQTT." http://mqtt.org/. Accessed 19 Mar. 2018.
[3] https://developer.apple.com/reference/foundation/nsjsonserialization

learning experience seeing how the backend server was operating, as normally the REST APIs have been provided to me by a third party.

Utilizing the JSONSerialization class, one can translate a query method parameter into a corresponding request object and send the HTTP request to the web service. Once a URL is provided, a task can be created using the URLSession and passing the URL to the task. Then, the JSON received can be decoded into useful information, such as login information and operator status. Since the data received is raw JSON, the data must be parsed and converted into objects usable by the application.

DESIGN

Web Service API Endpoints

To interface with the server from the user perspective, Colton Sundstrom developed an API utilizing Flask, a python web framework, to enable iOS communication. The API utilizes POST and PUT REST commands to request and push data with the server. All commands return status code 200 upon successful completion, which is viewable in the XCode console if running the debugger, or an appropriate error in accordance to the HTML standard.

Endpoint (“/skylux/api/...”)	Method	Parameters
register	POST	Username, Password, MAC
login	POST	Username, Password
schedule	POST	Token, Command, Time
status	POST	Token

status	PUT	Token, Status, Active, MAC
device	POST	Token, Command

Table 1: www.skyluxservices.xyz API Endpoints

Table 1 above lists all of the endpoints that the application accesses with the URLSession tasks. Once the user registers a device, that user is associated with a JWT Token, which is then used for all further operation of the application. This ensures that not anybody can operate the Skylux operator, since only somebody with the username, password, and JWT token can issue commands that the server will accept.

iOS Application

The application attempts to closely mimic the design of the Amazon Alexa application, which was the original design choice . However, since the WiFi handoff portion was not implemented, the application starts off in a state where the operator is assumed to be online and connected to the internet. When the user launches the Skylux iOS application, they have the option to register, or log in.

Login and Registration.

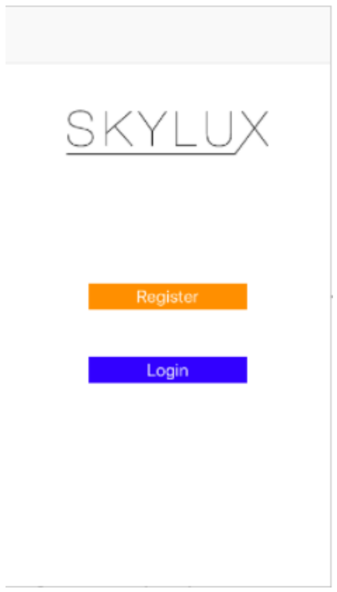


Figure 8.1: Skylux Entry Screen

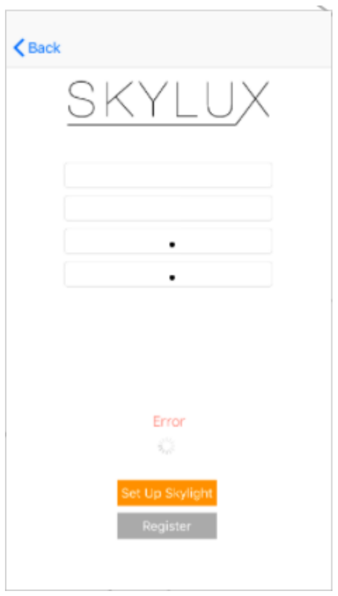


Figure 8.2: Skylux Registration Screen

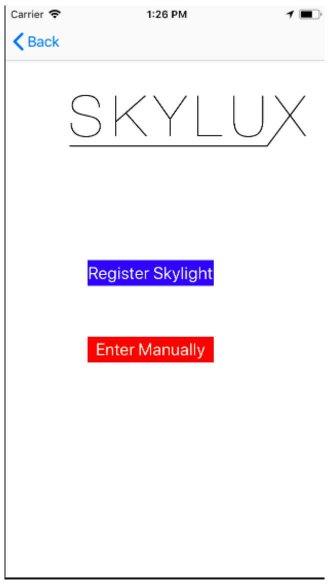


Figure 8.3: Skylight Registration Page

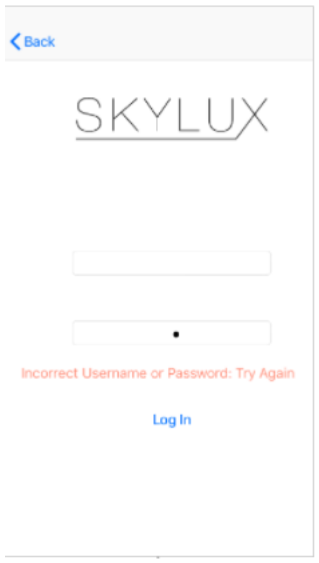


Figure 8.4: Skylux Login Screen

The figures above show the application screens for the registration and login process. If the user taps the registration button in figure 8.1, they will be brought to the screen shown in 8.2. From here, the "Set-up skylight" button is highlighted, and a tap on that button will bring the user to 8.3. From there, the user can register the skylight with a QR code or enter the information manually. The QR code contains the device information needed on the server, and will be sent to the server upon registration in a JSON POST request. After setting up the skylight, the user will be brought back to the registration screen, where they will choose and confirm a username and password. Once the correct information is provided, the username, password, and MAC address are all sent to the server via a POST, and if the information is valid, stored in the database on the server.

When the user navigates to 8.4, the username and password are then sent to the server in another POST request. If the server responds with a token, then the authentication process is complete, and the application is then moved to the main operation screen in 9.1.

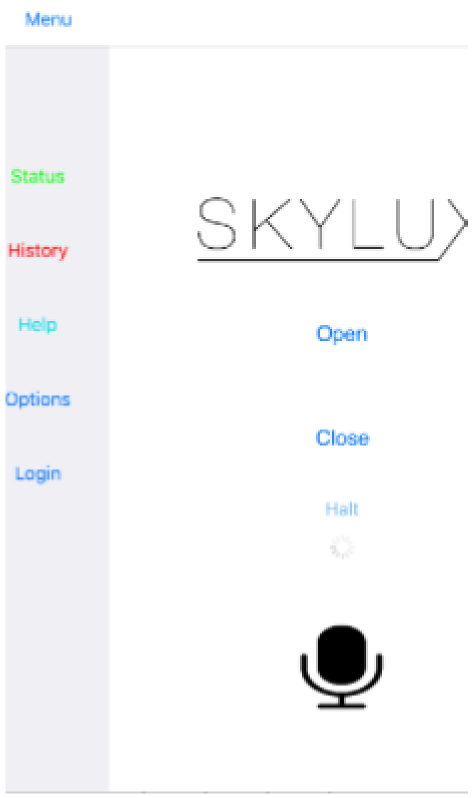


Figure 9.1: Skylux Main Screen with Menu Pulled Out

<u>Main Operation</u>

Now that the user is authenticated with the token, they can start issuing commands to the API. This is done with the OPEN/CLOSE buttons as seen in 8.1. The commands can also be issued with the microphone, which when tapped, will start listening for voice commands like “open”, “close”, or “status”. As previously stated, the command will be issued in a POST request, with the parameters being the authentication token and the desired command. The server then takes this command, decodes the token, then issues the command to the correct device. If the user wants to see the status of their device, they can tap the STATUS button as seen in 6.1. This will initiate a GET request with the parameters being the token and the device id which was provided upon registration. If the token is still valid, then the user will see the skylight status, which includes if it is open, and how open it is.

The user can also see where the last command was issued from and what it was by tapping the history button. This is currently all stored in the application, but hopefully with further development, this information can be stored on the server after more time is spent on developing this senior project. The other buttons lead to screens like the FAQ section and the login screen, along with a currently not implemented scheduling function.

SYSTEM TEST ANALYSIS

Through continuous development with my advisor, we were able to steer my project through a client-driven process to work out an end goal that was agreed upon by both parties. Since time was limited, it was agreed that the project be shifted to become more of a stepping point instead of a polished end-product. Through this report and the end setup process that will be included in the appendices, a new group of capstone students should be able to come in and understand the system in a way that the client may not have been able to adequately explain. Thus, I have focused on creating an intensive setup guide for the system, along with testing the system in multiple controlled environments to ensure system transportability and reliability.

The system has been shown to work under any network where the user had administrative privileges on their network and access directly to the microcontroller's display port on the board. However, if the user does not have those items, the system in its current state will be difficult to set up. The setup guide is created with this in mind, and will allow for the user to interface the Skylux iOS system with their router and network.

Since the server utilizes certain ports on the network, the user must have administrative privileges on the network, along with access to the router. If the ports are not open, then the services will not be visible over the internet, and the application will not run. However, the setup process has been set up on multiple networks where there is an internet connection and an accessible router, so the application seems to be very reliable and easy to set up though the guide I have created.

CONCLUSION

The original project met all of the client's initial goals when proposed back in the CPE 350 course in Winter 2017. While the first system worked adequately enough, it was not adequate enough for the client's end goal. This project took the idea and expanded on the initial implementation to create a more fully faceted system that is truly an IoT device. The system has been proved to work under a controlled environment, and there seems to be no issues preventing a theoretical setup in the client's home. However, the system is not commercially ready, since it is not developed with a security background in mind. There are also further improvements that could be made with the application, but the primary goal of the project was accomplished.

Areas that could be improved upon include:

1. Design

   This application is very simply designed. There are very well designed tools to create application designs, and could serve for a better user experience.

2. Security

   Currently, the information is stored in plaintext in both the phone and on the server. While the current design is functional, further encryption is needed to ensure security of personal information.

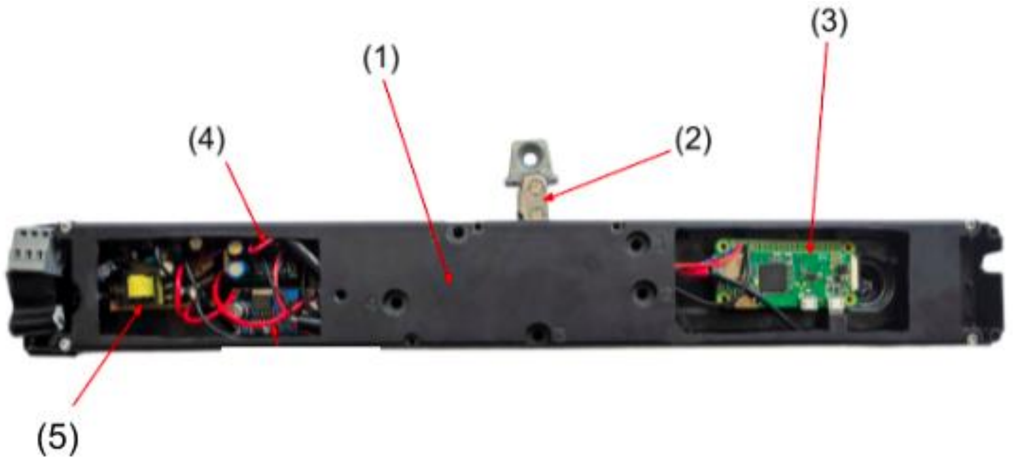
3. Features

   This project was started with more features in mind, such as weather data-based operation and scheduling. This project has lead me to research these topics and even deploy them in my own test projects, but these features did not make it into the application.

With that being said, this project will serve as the basis for many other projects in the future since it is so multi-faceted. So far, this has been the only project that has connected many different software and hardware components, and these IoT concepts are especially relevant considering the new and emerging technologies. This project has also taught me the intricacies of developing in the iOS environment, and many useful concepts such as JSON serialization and data parsing. Most of iOS development, if not an interactive application, is working on an application that gathers data from outside sources and provides it in a usable manner. This project has led me to research many of these topics I was previously unfamiliar with, and will prove to be very valuable knowledge when it comes to my future career choices.

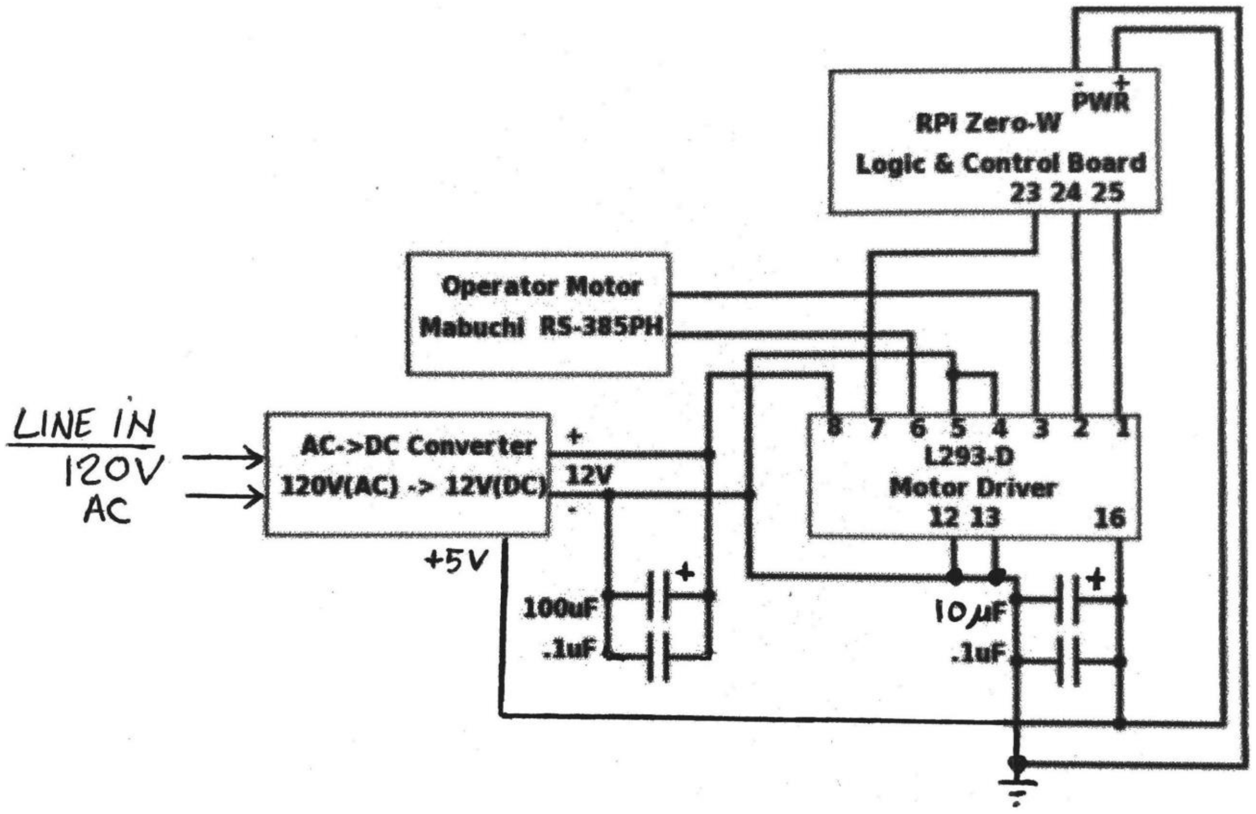
APPENDICES

A. Skylux Hardware



1. Velux Skylight Operator
2. Chain Actuator
3. Raspberry Pi-Zero-W
4. Motor Driver
5. AC-DC Converter

B. Wiring Diagram[4]



We improved to a new design by eliminating the LM2596 DC-DC converter and using a secondary +5V supply output from the AC-DC converter.

[4] Wiring Diagram by Richard Murray, 2017

SKYLUX SERVER SETUP GUIDE

Overview

Here are the steps needed in order to set up the Skylux system in a new environment. Keep in mind that this will not work at SecureMustangWireless at Cal Poly, and in its current state, requires display connections to the Raspberry Pi Zero-W in order to configure the network.

Note that if at any time the user gets denied with the error message: "Access Denied: Insufficient Privileges", then the user must type `>sudo su` and enter the administrator's information. Also, to keep the server running, you must keep the windows open for the terminal with the power constantly on for the server. To shut down the server cleanly, hit "CTRL+C".

Procedure

1. Log in to the GoDaddy Skylux website.
2. Obtain the IP address of your router
    a. This can be done at https://www.whatismyip.com, look for your public IPv4 address.
3. Navigate to the DNS management section of the website under My Products.
4. Under Records, edit the type A record.
    a. The "Host" is a placeholder name, replace the "@" with your desired name
    b. The "Points To" is where your local, static IP should be. Enter that IP here.
5. Log in to the server host machine with administrator privileges.
    a. If not already installed, install Python[5].
    b. Install Python modules sqlite3,Flask, Paho-MQTT, Homebrew, Mosquitto[6], JWT, and pyjwt
        i. Mosquitto can be installed from the homebrew project. See brew.sh and then use `> brew install mosquitto`
6. Assign your server a static IP address on your network
    a. This is usually done in network preferences. Be sure that the IP address you choose is in the valid range and not taken by another device.
7. Run the server code

[5] "Download - Python.org." https://www.python.org/downloads/. Accessed 6 Feb. 2018.
[6] "Installing MQTT Server on Mac OS – simplified. thinking.." 3 Oct. 2015, https://simplifiedthinking.co.uk/2015/10/03/install-mqtt-server/. Accessed 23 Mar. 2018.

a. In the command line, ensure you have admin privileges by typing
>sudo su

b. Run the code by typing

```
>python comp_mqtt.py
```

(in a new tab)>python comp_app.py

c. Now the server should be running, and you should see the port and other information in the terminal

8. Log in to the router. This procedure assumes an Apple Router is being used in the development environment, but the steps are essentially the same for any router.[7]
   a. Open AirPort Utility
   b. Reserve a DHCP-provided IP address for the host device.

(Note this is the device that you want to access from a remote location.)
AirPort Utility > Select the base station > Edit > Network tab

   c. Setup port mapping on the base station. Click the Add + button under Port Settings. For the IP address, use the IP for the router. For the port numbers we want to open, use 1883 and 5000.
   d. After you click update, then the ports should be open. You can test this at http://canyouseeme.org.
9. Set up Skylux Operator
   a. METHOD 1
      i. If a wpa_supplicant.conf file is placed into the /boot/ directory, this will be moved to the /etc/wpa_supplicant/ directory the next time the system is booted, overwriting the network settings; this allows a Wifi configuration to be preloaded onto a card from a Windows or other machine that can only see the boot partition.
      ii. While unplugged, remove SD card from the operator.
      iii. Plug SD card into micro SD card reader and open its contents.
      iv. Create a wpa_supplicant.conf
         1. A typical wpa_supplicant.conf file is:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=US
network={
         ssid="«your_SSID»"
         psk="«your_PSK»"
         key_mgmt=WPA-PSK
}
```

[7] "Port Forwarding the Apple AirPort Extreme Router ... - RainMachine." https://www.rainmachine.com/support/portforwarding/Port-Forwarding-Apple-AirPortExtreme-Router-for-HTTPS.pdf. Accessed 13 Mar. 2018.

v. Plug SD card back into operator.
vi. Plug in operator, the operator should now connect to home network and listen for server calls.

b. METHOD 2
   i. Plug in operator
   ii. Plug in mini HDMI converter and keyboard via converter to RPi
   iii. Log in using username: pi password: thx1138
   iv. Edit the file /etc/wpa_supplicant/wpa_supplicant.conf
      1. Replace existing network information with your own
   v. `>sudo reboot`
   vi. The operator should now connect to home network and listen for server calls.

ANALYSIS OF SENIOR PROJECT DESIGN

Summary of Functional Requirements

Describe the overall capabilities of functions of your project or design. Describe what your project does. (Do not describe how you designed it.)

This project:

- Must be operational over the internet
- App must run on any current iOS device with version 9.0 or above
- Must show status information of the skylight in app
- Must be operational on a typical Local Area WiFi Network
- Must control the Skylux operator designed in Capstone 350/450 2017 with client Murray
- Must be secure and reliable

Primary Constraints

Describe significant challenges or difficulties associated with your project or implementation. For example, what were limiting factors or other issues that impacted your approach? What made your project difficult? What parameters or specifications limited your options or directed your approach?

For a large portion of time, I did not have a computer that was able to develop iOS applications. It turns out that in order to develop for the iOS environment, you must have an Apple device, and one that is quite current in order to run the latest version of XCode. Since I was working nearly full time outside of school to put myself through school, it was tough enough to afford a workstation to tackle the project I had committed to unknowing of the requirements. I obtained my development machine very far into my first quarter of development, and since I was already limited to developing in what little free time I had, I was very inhibited with the progress of this project. I would have liked to flesh out the project more on the application side, but that required more hardware knowledge for the Raspberry Pi, of which I was quite unfamiliar. With that in mind, my advisor directed me to having a solid basis for the project, and encouraged me along to the end product.

Economic

- *Original estimated cost of component parts (as of the start of your project)*
- *Actual final cost of component parts (at the end of your project)*
- *Attach a final bill of materials for all components*
- *Additional equipment costs (any equipment needed for development?)*

- *Original estimated development time (as of the start of your project)*
- *Actual development time (at the end of your project)*

Since my project was an extension of my Capstone group project, I will use that for my bill of materials.

Skylux Original Estimated Cost of Component Parts

Quantity	Product	Manufacturer	Price per Unit	Purpose
2	Raspberry Pi 3	Raspberry Pi	$35	Main Board to control the operator
2	16GB Ultra Plus Micro-SD CL10	San Disk	$15	Memory for the Raspberry Pi Boards
1	3D Printing Funds	Local	$100	Casing
4	Additional Development Boards		$20	Possible future iterations
1	Apple Dev License	Apple	$100	Access to Apple App Store Publication
1	Circuit Components	Sparkfun/Digikey	$50	Circuit Elements
1	Electric Motor Driver	Texas Instruments	$5	Driver for electric motor
1	Electric Motor	Mabuchi	$1	Prototype
		Total	$526	

Actual Final Cost of Components to Build Each System

Quantity	Product	Manufacturer	Price per Unit	Purpose
1	Raspberry Pi Zero W	Raspberry Pi	$10.00	WiFi CPU to control the operator
1	16GB Ultra Plus Micro-SD CL10	San Disk	$9.26	Memory for the Raspberry Pi Board
1	Driver PCB	Custom	$2.00	Control motor driver
1	Mabuchi RS-385PH	Mabuchi	$2.15	Replacement Motors for operator
1	Skylight Operator	Velux	$341.25	Manufacture Product Integration
-	Tax		$32.81	
		Total	$397.47	Or $56.22 using an existing donor unit

Labor

Category	Estimate (hrs)	Actual (hrs)
MQTT Communication	40	10
REST API Interaction	0	50
iOS Design	30	50
JSON Parsing	10	30
RPi Integration	40	0
TOTAL	120	140

Environmental

Describe any environmental impact associated with manufacturing or use.

Ideally, this project was aimed to improve the environmental impact of the user. This application is aimed to keep the internal climate of the home regulated, and with further improvements on the project, it could be more environmentally impactful. Since the home climate would be regulated with a skylight that is easy to operate, the user would need to rely less on appliances to regulate home climate, which all use electricity, which has a negative environmental impact, and costs money for the user.

Manufacturability

Describe any issues or challenges associated with manufacturing
Installation of the iOS application is straight forward. Hardware implementation leverages an off-the-shelf Raspberry Pi and a small custom PCB for the driver circuitry using a standard 2-layer process. The most significant labor is retrofitting an existing Velux operator, which requires careful disassembly and removal of the original Velux RF controller and replacement with the new Skylux components.

Sustainability

- *Describe any issues or challenges associated with maintaining the completed device or system.*
  The issue currently maintaining the device is that it is cumbersome to transport from one network environment to another. Currently the user must hook up to the operator in order to reset the network information, and that is not sustainable. Maintenance is easy, however, since the server must only be left running and the iOS device should be able to contact it at any time.
- *Describe how the project impacts the sustainable use of resources.*
  The project leverages re-use of an existing skylight operator with minimal new components and software control with an existing smartphone which makes this project very sustainable in resource usage.
- *Describe any upgrades that would improve the design of the project.*
  Areas that could be improved upon include:

  Design

  This application is very simply designed. There are very well designed tools to create application designs, and could serve for a better user experience.

  Security

  Currently, the information is stored in plaintext in both the phone and on the server. While the current design is functional, further encryption is needed to ensure security of personal information.

Features

This project was started with more features in mind, such as weather data-based operation, scheduling, and a WiFi handoff . This project has lead me to research these topics and even deploy them in my own test projects, but these features did not make it into the application.

- *Describe any issues or challenges associated with upgrading the design.*
  WiFi handoff implementation may require extra hardware, and soldering to the small Pi Cobbler is quite difficult. There were also not many security measures taken when developing, so upgrading the system to be more encrypted will take a team of effort.

Ethical

Describe ethical implications relating to the design, manufacture, use or misuse of the project.

Device does not seem to have any adverse ethical implications.

Health and Safety

Describe any health and safety concerns associated with design, manufacture or use.

This system was designed assuming a typical operational environment. Situations such as power failures or load imbalances were not taken into account in actual operation of the device. The device may fail and cause issues with usability, but user safety is a minimal risk.

The safety of the user's personal information is at stake as well. If a malicious actor were to gain access into the user's system, all of the information is stored in plaintext, so the user is opening themselves to a cyber-attack.

Social and Political

Describe any social and political concerns associated with design, manufacture or use.

Device has minimal social or political associations.

Development

I learned nearly all of the aspects of this project outside of Cal Poly courses. Subjects such as REST API interactions and MQTT messaging were all new to me, as were interacting with the iOS environment. All of my previous experience coding was either in a Linux environment or a controlled system. Never before was I able to interact with the iOS library, or any of its tools such as XCode or Swift, both of which required extensive research outside of class.