LockStars: Home Access System

Final Documentation

Team LockStars
Richard Fasani
Zach Imholte
Burton Judson
Jon Wiese

Table of Contents

Introduction

The LockStars “Internet of Doors” system was designed to allow a user to control aspects of their front door from their mobile device. This would allow the user to get into their house and see who is at their front door, just from their phone. It would allow for easier home security and diminish the need to carry around a house key throughout the day whenever the user left home, as they can get into their house with just their phone.

The way to do this was to add a solenoid and camera to the frame of the front door, each with an inexpensive device attached to allow them to communicate through the house’s WiFi network with the mobile application. A power converter was also added near the door to power the devices from house power. Also, a host was created to talk between the home server and the devices on the door. The server listens to the mobile app that the user controls. Whenever the user wants to see who is at their door or wants their door to open, they can interact with the mobile app, which sends a signal to the devices. The camera device also send the picture data it receives from the camera to the host to put on the server so the mobile app can access it. Both devices by the front door are also able to receive a signal if the doorbell is pressed and tell the camera to take a picture if someone presses the bell.

The design implemented does all of the things listed above. It works well as a prototype, but more coding could still be included to make the entire system more robust. Overall, the system works well together and all the pieces perform the functions they were designed to perform.

System Requirements

The first requirement for our design will be the required functionality of the entire system to solve the problem statement. The purpose of this project is to create a safety system for one’s home, and thus it should ensure and meet the requirement of acting as a door lock when the user is not home. The door strike should also be able to be remotely controlled via wifi from one of the registered devices, in order to allow the door to be opened for specific reasons whenever desired by the user. The other major requirement of our system will be the camera portion of the system, which will allow the homeowner to get a glimpse of who is visiting their home, even when no one is home. This will also have the possible capability of being triggered whenever the doorbell is rung. The camera will need to be triggered, capture a picture, and then pass the RGB values for the jpeg through a serial connection to the ESP chip or microcontroller. Once these values have been passed, they will be sent via wifi from the chip to a server where the values in an array will be assembled into the JPEG image that is desired and stored on that server. The user will then be able to use the mobile app to access this server, and thus the image of the person at the entrance of the home. From the intelligence side of things, the ESP or microcontroller must be able to trigger the camera to capture a photo from either a doorbell signal or from a user triggered command over the mobile application and wifi. Then, the picture data must be read serially to the chip and then sent to the server, where the picture will need to be assembled into a JPEG from the RGB values which will take more research and coding. The picture will then be stored in a database and be able to be accessed by any of the registered devices. The system will also need to be intelligent in the sense of triggering the door strike from a user sent command, and then also will pass a high signal to the door strike when the user’s device is close enough to jump onto the home wifi network. This will ensure that the user is within a close enough proximity to unlock the door.

The next requirement will to have this all interface over wifi and look at the following considerations. The biggest step will be connecting the two ESP chips to have the capabilities to talk and trigger each other, as well as correspond to a server that the user will control. This means that there will be two devices that will be required to be supported, the door strike ESP and the camera ESP. Then the server will need to interact with the various user devices via a mobile app, which will allow for the user interface that is required. The setup should also have the potential to be expanded to sufficiently handle more devices if more door strikes or cameras modules for various doors wanted to be added.

One further requirement for the system will be the size of the camera and electric door strike, as well as the connecting hardware and microcontroller board. The camera will need to be positioned above the perimeter of the door frame, facing the area directly in front of the entrance, and the electric door strike will take the place where most locks currently are by the handle for opening and closing. The microcontroller board will be mounted in the wall near the door, with a panel on the inside of the home to be able to access the system. The purpose of this will be to have the ability to tap into the home's power supply for our power constraints, as well as have panel access for maintenance and programming, in the case of emergencies. The power will need an AC-DC converter or transformer to step down the voltage to the 5 volts for the microcontroller or ESP module, as well as the 24 volts for the door strike. Another option will be to use a relay to control the door strike with a small output voltage from the ESP or microcontroller module. Since we are using the power from the home, power consumption and availability is not a large factor for our design. The system will also be lock-safe for situations when the power goes out, meaning the wifi system would be down as well as the power controlling the system. The door strike will need the high voltage signal to unlock, so without power the door will automatically lock. At this point, the manual key lock will still be functional to be able to enter and exit the home.There are no other size or weight constraints on our system.

System requirements In summary:

- act as door lock when user is not home
- camera should allow user to see who is at door
- camera to be triggered by doorbell or by app
- camera to send photo information over wifi
- photo information to be assembled into image
- image hosted on website
- image accessible by registered devices
- interface over wifi
- solution is extensible to "n" devices
- AC to DC converter capable of powering strike and ESP

Project Description

System Theory of Operation

The user should be able to communicate with the devices connected in the system to open their door lock and see who is at their door. There is a mobile application which acts as a user interface to allow the user to set up the devices connected to the network as well as control these devices. Once the user and devices are connected to the network, the user can tell the solenoid module to open and tell the camera module to take a picture through the mobile app. If the camera module is connected to the doorbell, it will also take a picture when the doorbell is pressed, and if the solenoid module is connected to the doorbell, it will send a signal through the WiFi network to the camera module to take a picture. The camera module takes the picture information from the camera and separates it into packets to send wirelessly to the host on the WiFi network. The host takes these packets of information and pieces them together into a picture, which it then sends to the server to publish. The mobile app is then able to look at the picture from the server and show it to the user. There is also a power board that converts house power to DC +12V, +5V, +3.3V, and GND. There are connectors from this power board to the solenoid and camera modules in order to provide enough power to the ESP8266 devices, camera, and solenoid. The power board, camera module, camera, solenoid module, solenoid, and doorbell connectors are all able to fit on or near a door frame and be powered by house power. The mobile application can be downloaded on an IOS device. The entire system can run on a single house WiFi network.

System Block Diagram

The system block diagram of the overall system is shown below in Figure 1.

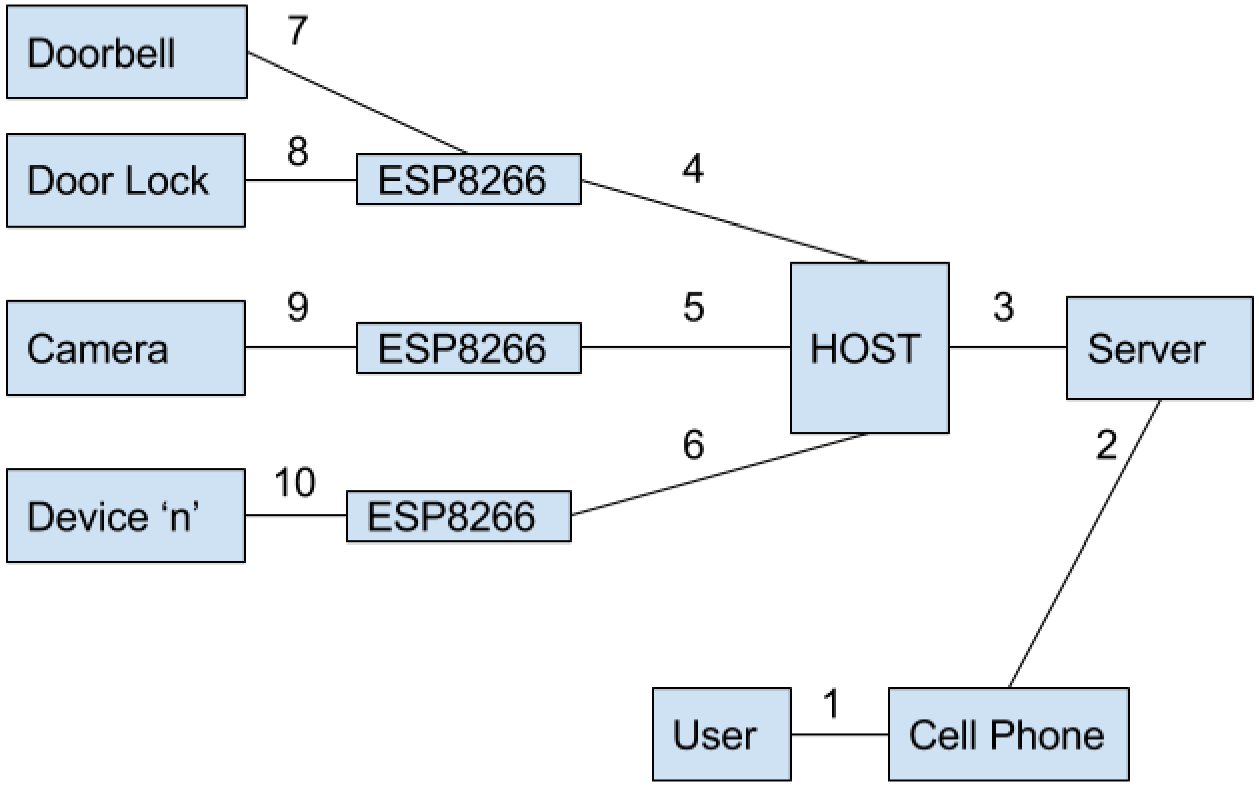

Figure 1: System Block Diagram

1. The interface between the user and the cell phone is through a mobile application. The mobile application allows the user to connect the devices to the home WiFi network, open the door lock, tell the camera to take a picture, and view the picture saved on the server. The cell phone usually has some security built in so the user has to log into the phone in order to access the mobile application.
2. The cell phone is able to connect to the server to retrieve an image saved to the server, as well as connect to the home WiFi network to send signals over the MQTT protocol.
3. The host sends the server information it has received and processed from the various ESP modules. This information is mostly the picture data received from the camera module.
4. A wifi connection, the interface from the host to the Door Lock ESP8266 receives a MQTT signal sent from the host to say how long the door lock should be open.

It also transmits a MQTT signal if the doorbell has been pressed to tell the Camera ESP8266 to take a picture.

5. A wifi connection, the interface connecting the Camera ESP8266 and the host transmits a signal telling the ESP8266 to take and send a photo from the camera, and then transmits the photo information from the camera to the host in packets.
6. A wifi connection, this is a theoretical connection for the additional devices to be connected to the host. This is for future additions to the system.
7. The interface between the Doorbell and one ESP8266, this is a hard-wire connection that checks if the doorbell has been pressed
8. The interface between the Door Lock and the same ESP8266 as the doorbell, this is a hard-wire connection that provides 12V and 1A to a solenoid to open and close the door lock when the ESP8266 receives a specific signal.
9. The interface between the Camera and one ESP8266, this is a hard-wire connection that powers the camera from the ESP8266, and allows the ESP8266 to tell the camera to take a picture when it receives a specific signal, and then receive the picture from the camera in increments to send over MQTT.
10. For whatever devices are added later, they must be capable of interfacing with the server in some way, so they must communicate to the host through an additional ESP8266.

Design and Operation of Camera Module

The camera module must be able to connect to the home WiFi network, receive a signal to take a picture, take a picture, and send the picture data wirelessly over the home WiFi network. Figure 2 below shows a high level description of the code controlling the camera module. The complete code is included in the appendix of this document.

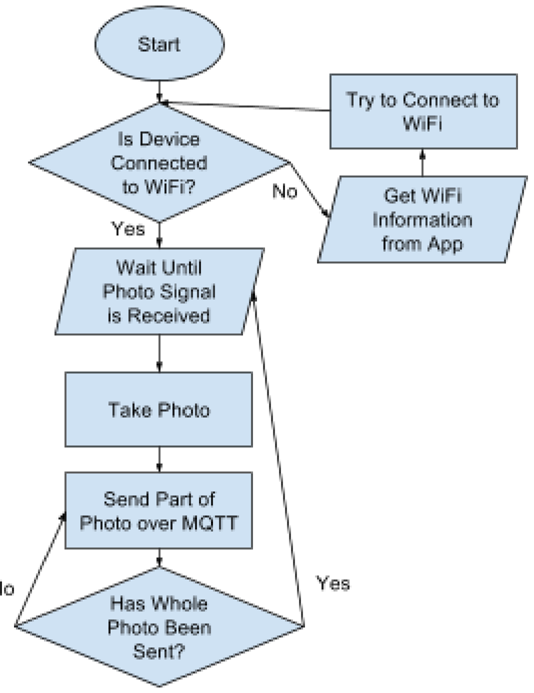
Start
Is Device Connected to WiFi?
No
Yes
Try to Connect to WiFi
Get WiFi Information from App
Wait Until Photo Signal is Received
Take Photo
Send Part of Photo over MQTT
Has Whole Photo Been Sent?
No
Yes

Figure 2: High Level Camera Module Code Flowchart

The flowchart is very high level and leaves out a lot of the intricacies of the signals being sent. When the device starts up, it tries to connect to a WiFi network. If it cannot connect, the ESP8266 becomes a server and listens until a website is searched on it. The mobile application takes information about the home network and sends it to the module as if it were searching for a website. The ESP8266 then uses the information sent to it to try and connect to the home network. If it does not connect, it becomes a server again, but if it connects, it moves on to waiting to be told to take a picture. The module subscribes to a topic and waits for a message telling it to take a picture. It also is programmed to listen to the doorbell if one is connected to it. If it receives either signal, it will tell an attached camera to take a picture. It then asks the camera to tell it a fixed amount of bytes from the picture and publishes those bytes to a topic over MQTT. It keeps doing this until it reads the end signal for the picture. At this point, the camera module returns to the state in which it waits for a signal to tell it to take and send another picture. The camera module was programmed in the arduino ide because it already included programming settings to program the device. The programming settings assigned a NodeMCU 0.9 board, serial uploading, 80MHz CPU frequency, 4M (3M SPIFFS) flash size, upload speed of 115200, and AVRISP mkII programmer.

The hardware schematic for the generic ESP module, as well as the board design are shown below in figures 3 and 4, which contain information for the camera module.

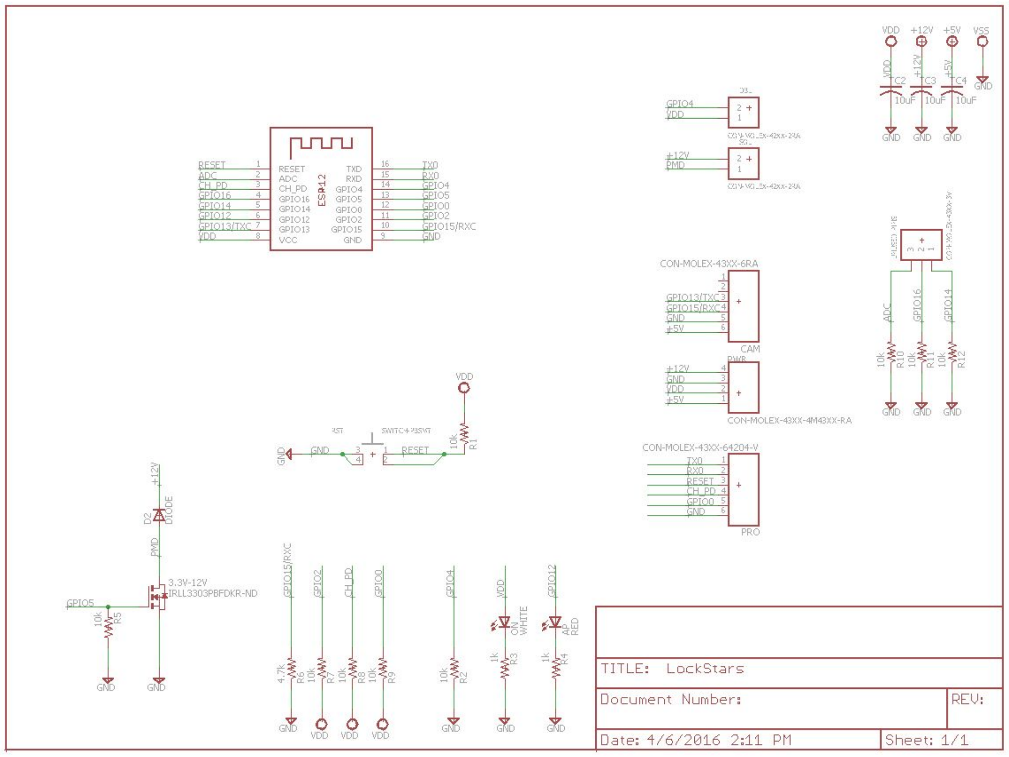

Figure 3: ESP Device Schematic with Camera Module

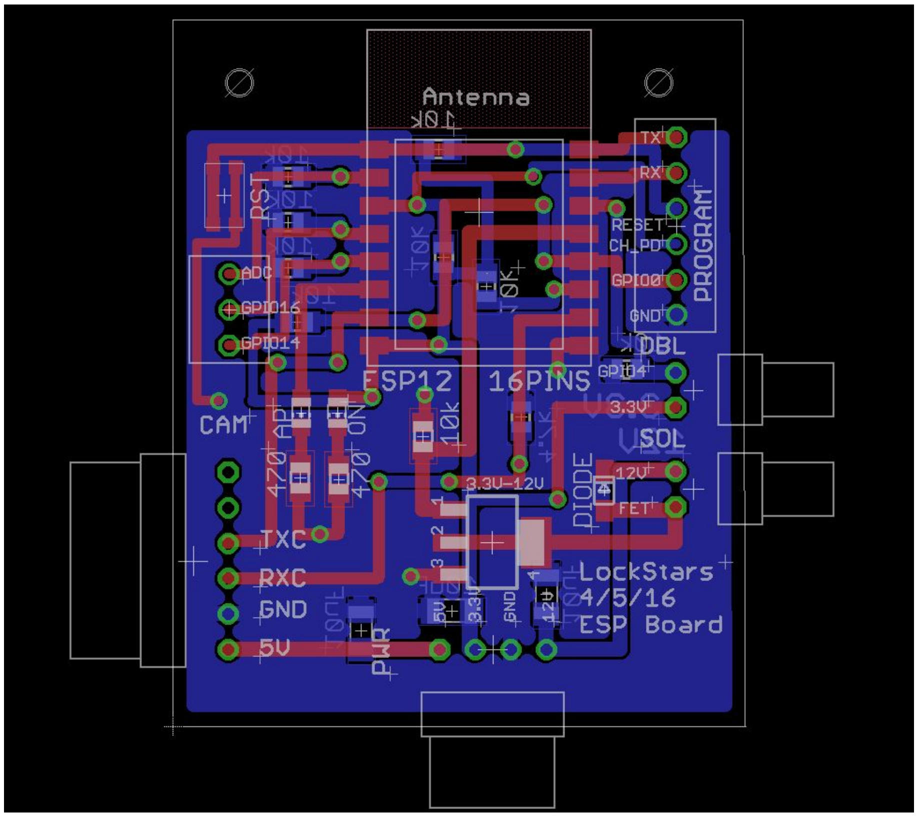

Figure 4: ESP Device Board with Camera Module

The camera module uses the ESP8266 as well as the connector that contained +5V, GND, RXC, and TXC. The connector on the device was used to communicate with a peripheral camera through a hardware serial interface using the ESP8266 pins GPIO13 and GPIO15. These pins acted as TX and RX with the camera to send and receive signals. The ESP8266 then took the signals received and published them over MQTT for the host to receive and process into a picture. The camera used was a TTL Serial Camera from adafruit and was used because it could be controlled with a serial connection.

The camera module was tested in parts. In the early stages, it was tested while attached to a logic analyzer just to ensure it was sending the correct signals, and the peripheral camera was responding in the desired way. After hardware communication had been verified, the ESP8266 was tested to see how large of packets it could receive and send from the peripheral camera and over MQTT. Initial tests tried having the entire picture sent at once, but it was too big for the flash memory of the ESP8266 to store, and its size changed according to how much was in the picture. Piecing the picture into 100 byte increments was tried next, but the peripheral camera did not seem to always send the correct data if 100 bytes of information was requested. The next step was to ask for 64 bytes and this worked, the peripheral camera sent the correct information in 64 byte packages and the ESP8266 could store 64 bytes in its flash memory without a problem. Once all the picture information could be sent accurately, small tests were run occasionally to ensure efficiency and optimization changes in the code did not detract from the module’s ability to perform its main function of receiving the picture from the peripheral camera and sending it over MQTT.

Design and Operation of Solenoid Module

The solenoid module must be able to connect to the home WiFi network, receive a signal to provide power to a solenoid, provide power to the solenoid, and send a signal over the home WiFi network to tell the camera module to take a picture if the doorbell is pressed. Figure 5 below shows a high level description of the code controlling the camera module. The complete code is included in the appendix of this document.

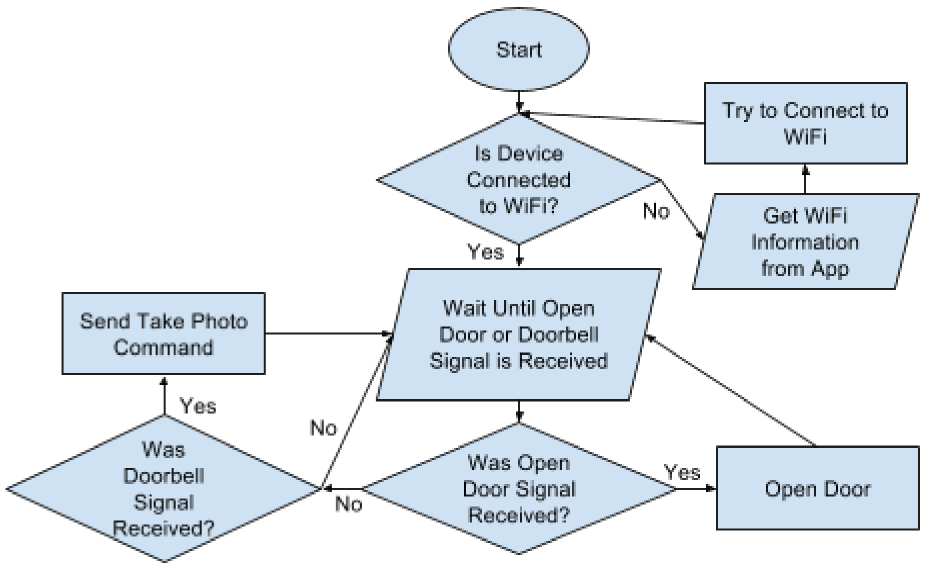
Start
Is Device Connected to WiFi?
Try to Connect to WiFi
No
Get WiFi Information from App
Yes
Send Take Photo Command
Wait Until Open Door or Doorbell Signal is Received
Yes
No
Was Doorbell Signal Received?
No
Was Open Door Signal Received?
Yes
Open Door

Figure 5: High Level Solenoid Module Code Flowchart

This flowchart is very high level, but it shows what is happening in the solenoid module program. When the device starts up, it tries to connect to a WiFi network. If it cannot connect, the ESP8266 becomes a server and listens until a website is searched on it. The mobile application takes information about the home network and sends it to the module as if it were searching for a website. The ESP8266 then uses the information sent to it to try and connect to the home network. If it does not connect, it becomes a server again, but if it connects, it moves on to waiting to be told to open the door by receiving a signal over MQTT, or publish a message to take a picture if it receives a signal that the doorbell was pressed. It listens to see if the doorbell is pressed by checking if GPIO5 goes high. The doorbell is just treated as a push button that shorts the pin to VDD if the button is pressed. When the button is not pressed, the pin is tied low to GND with a resistor. The ESP8266 is also subscribed to a topic to listen for a signal to open the door. The signal it receives comes from the mobile application and is a number that specifies how many seconds the solenoid should have power provided to it to open the door. The code does this by telling GPIO4 to go high for however many seconds is requested over MQTT. The solenoid module was programmed in the arduino ide because it already included programming settings to program the device. The programming settings assigned a NodeMCU 0.9 board, serial uploading, 80MHz CPU frequency, 4M (3M SPIFFS) flash size, upload speed of 115200, and AVRISP mkII programmer.

The hardware schematic for the generic ESP module, as well as the board design are shown below in figures 6 and 7, which contain information for the solenoid module.

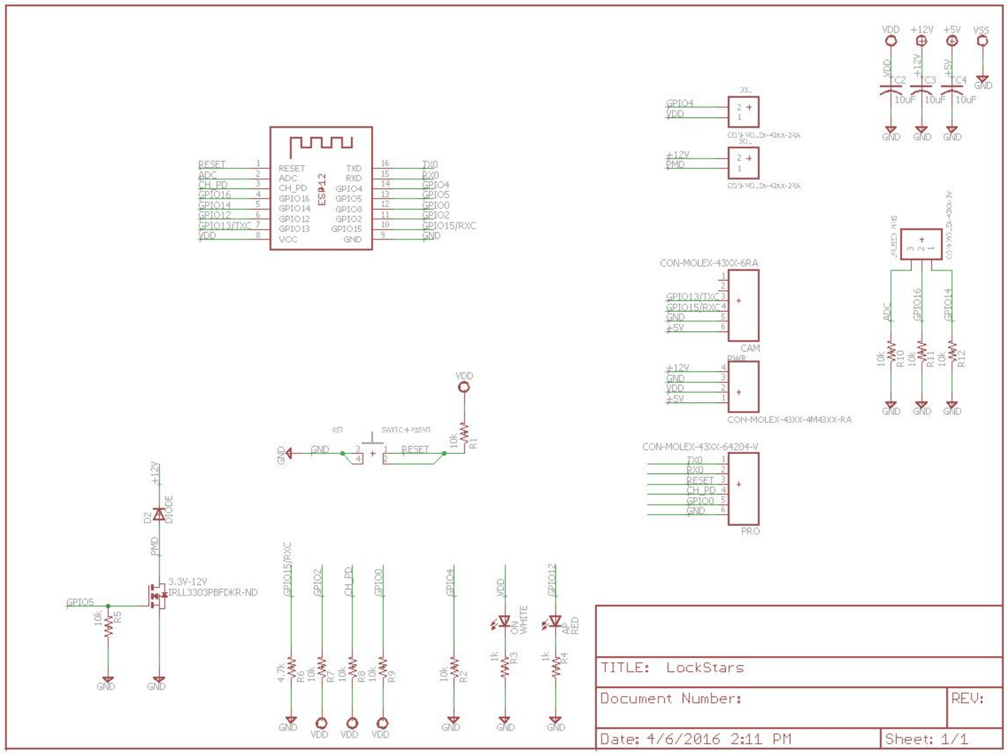

Figure 6: ESP Device Schematic with Solenoid Module

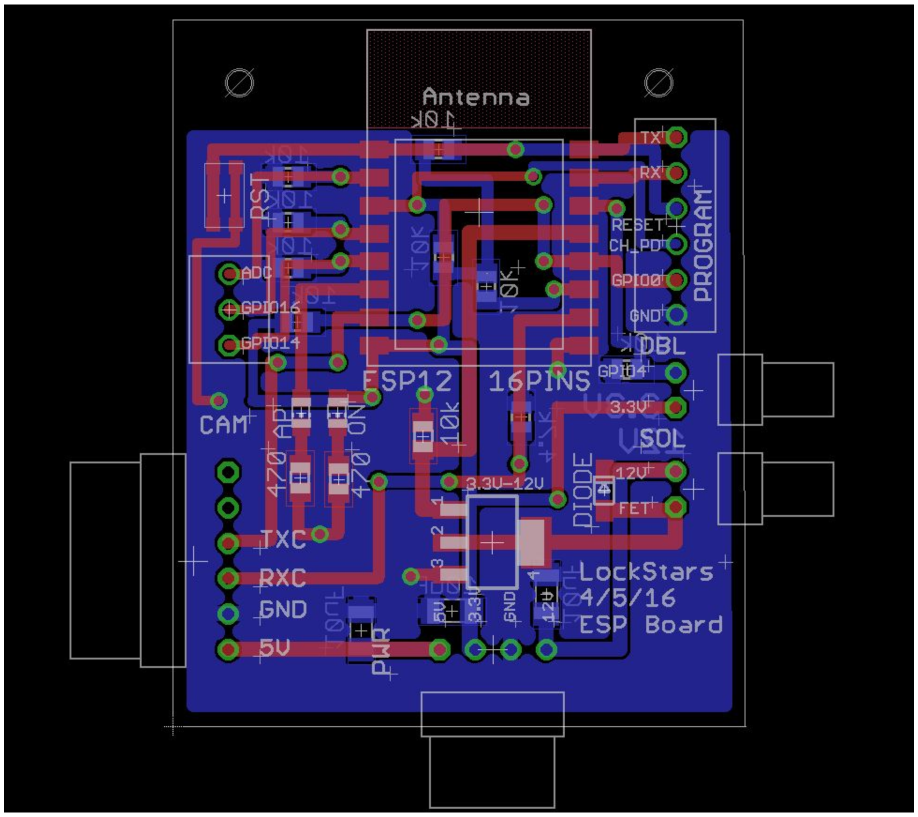

Figure 7: ESP Device Board with Solenoid Module

The solenoid module uses the ESP8266 as well as both two-pin connectors on the right hand side of the device and a power FET. The top two-pin connector is for the doorbell. A wire connection can be made from the board to a doorbell or push button to act as a doorbell. One pin of the connector is routed to VDD and the other is routed to GPIO5 and a pull-down resistor. When the doorbell or push button is pressed, the pins are shorted and GPIO5 reads a high signal, telling the ESP8266 that the doorbell was pressed. The bottom two-pin connector is for the solenoid. One output pin is set to +12V and the other pin is set to the drain of a power NMOS device rated for +12V and 1A from the drain to the source when +3.3V is provided to the gate. When GPIO4 goes high, it provides +3.3V to the gate, turning on the NMOS “switch” and allowing current to flow through the solenoid. Since it is a fail-secure solenoid, meaning it locks when there is no power provided, setting the GPIO4 signal high opens the solenoid. There is also a diode in parallel with the two connector pins to prevent the device from seeing large voltage spikes when power to the solenoid is switched from on to off.

This subsystem was tested as parts were acquired. Initially, the code was tested to make sure the pins could be set high, low, or as an input using the ESP8266, and that the output pins could change their value if certain signals were received. Once the solenoid was ordered and received, tests were performed to ensure it could run using a +12V power source, a power FET, and a diode. Once the power board was acquired, more tests were run to make sure the voltage received from the power board could power the solenoid. Smaller tests were also run whenever the code was changed to become more efficient or optimal in order to ensure the necessary functions of the ESP8266 would still work.

Design and Operation of Power Board

Each of the various devices have certain power requirements that need to be met. Rather than running off of a battery source, the system runs off of the home's power. This decision was made based on the fact that the system will be implemented in the wall surrounding the door, and thus the power will be accessible at this point. Also, being in the wall would make it difficult to switch batteries when the current battery dies. The other consideration was how to jump down the power from the traditional 120 volts AC to the requirements of the various devices. These values included 12 volts and 1 Amp for the solenoid lock, 5 volts and 75 milliamps for the camera module, and then 3.3 volts and 80 milliamps for the ESP device. The following is the board that was created to generate the power for our system:

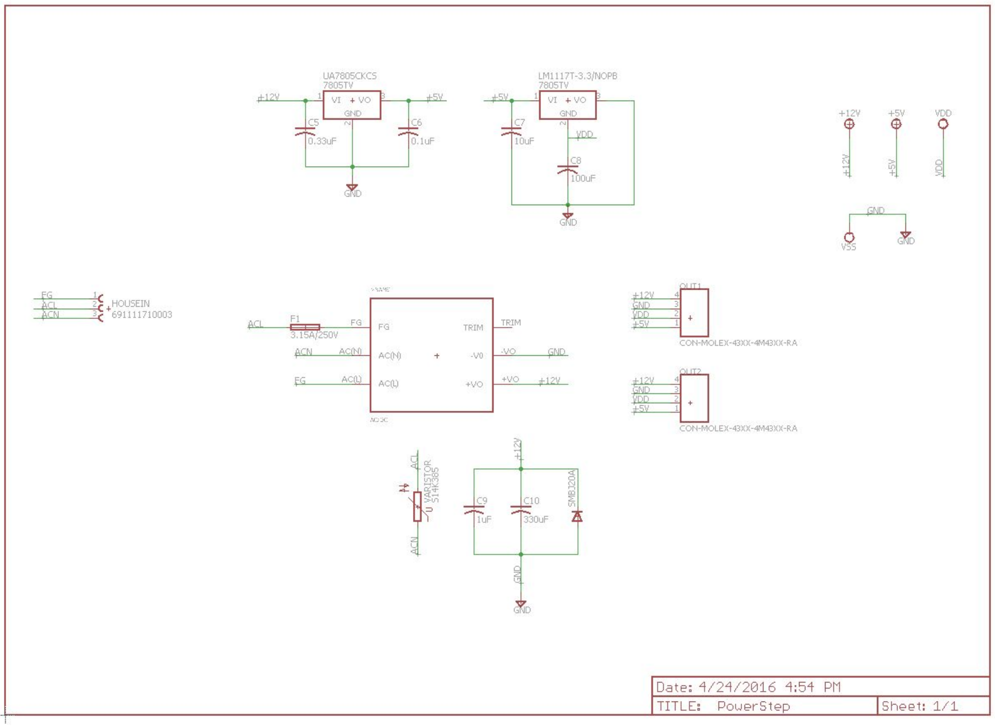

Figure 8: Power Board schematic to provide power for system

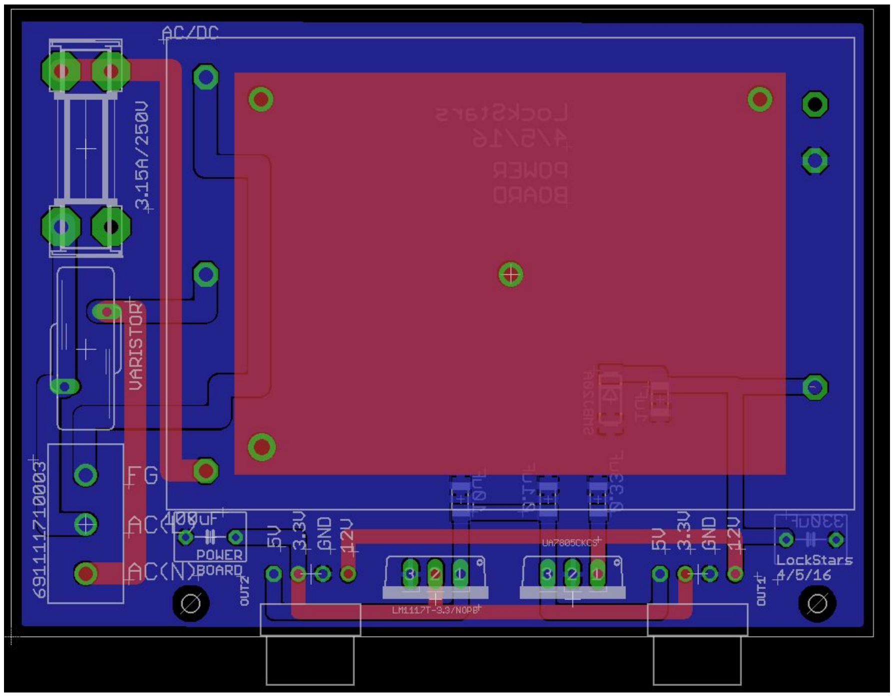

Figure 9: Power Board layout

Once the power specifications were identified, all that was required was to make a board that could provide the correct voltages and amount of power. This was achieved by ordering an AC to DC converter that would bring the 120 VAC power down to 12 VDC. The obstacle of doing the power through this method was that the ordered part did not have an eagle file to use in our schematic and board layout. Our group then researched and designed an eagle part with correct dimensions to use for the converter. Once this had been completed, our group was able to use a 5 volt and 3.3 volt regulator to bring the 12 volts required for the solenoid down to the levels required for the other devices in our system. Other issues that our group encountered was a mistake in the pin locations for the 3.3 volt regulator (switched the output and ground) and also the AC to DC converter. (switched the live input and ground) The first problem was remedied by creating a wire harness switching the pins to the necessary location that fit between the regulator and the holes in the board where the device should have been implemented. The second problem was a bit more challenging. This was accomplished by moving one side of the fuse holder off of the board, and then soldering a wire to this end that connected it into the screw terminal for field ground. Thus, the live power would run through the screw terminal, varistor, and fuse before being fed back into the screw terminal for field ground that was actually connected to the converter's pin for live power. Then an additional screw terminal was added to the position of where the fuse holder was moved from, and the field ground was fed into this terminal that connected with the converters pin for ground. Ultimately, the solutions worked and remedied these mistakes, and once this was identified our group went back and rerouted the boards to fix these for future purposes.

This system was tested by using a multimeter and ensuring all of the connections were correct before any of the parts were physically put onto the board. Once the issues were discovered, they were corrected and again tested using a multimeter to ensure nothing was shorted and the correct pins were all in the correct positions. After all of the parts had been mounted, power from the wall was added to ensure the converter worked correctly. Using a multimeter once more, the various voltages from the converter to the regulators were checked to ensure they were correct. The final step was to ensure that the power board was able to power the various components. This was completed by making the correct wire harnesses to connect all of the required boards. Our group then provided power to ensure that everything was functioning, and the devices all worked without any issues from a power standpoint.

Design and Operation of Server

Of the system requirements listed in “System Requirements”, the following are those which the server subsystem must sustain:

1) camera to send photo information over wifi: If the camera is going to send photo information over wifi, then the server must be able to receive the photo information over wifi. This was achieved by utilizing MQTT communications protocol and implemented in the Python scripting language using the paho-mqtt library for MQTT communications (https://pypi.python.org/pypi/paho-mqtt/1.1). This information is sent via the ESP device connected to the camera.
2) photo information to be assembled into image: this was achieved by encoding the binary information that was sent in packets over MQTT into the format that it was originally coded into: JPG. This creates (or overwrites) a new JPG file that is the most recent image to have been sent over wifi.
3) image hosted on website: then the machine running the Python script uploads that image to an Amazon Web Services (AWS) hosted website using the tinys3 library (https://www.smore.com/labs/tinys3/)
4) image accessible by registered devices: once the image is on the website, the mobile application on a registered device can access it using a web browser
5) interface over wifi: all of this is accomplished by leveraging an MQTT based publish/subscribe protocol hosted on a Raspberri Pi for development purposes. Eventually this will be moved to a web-based host.
6) solution is extensible to "n" devices: the Python server can subscribe to multiple topics for multiple cameras (or any other device whose data required processing in Python).

Before the server can begin normal operation, it must be initialized. This entails connecting to the MQTT broker, and subscribing to the topics of interest. After that, the server can proceed as depicted in the following flow chart:

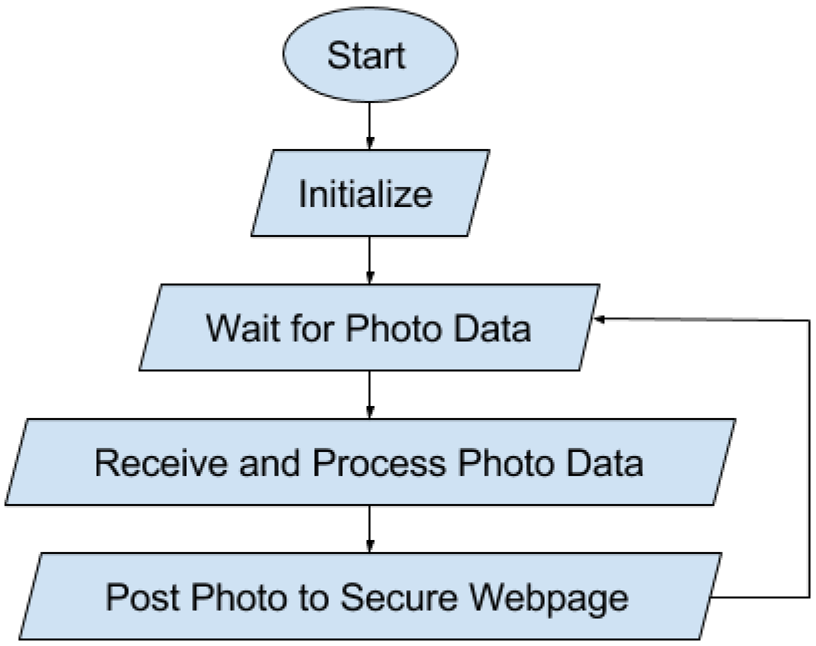

Figure 10: Server FlowChart

Python 2.7

The program used to implement the above described functionality is Python 2.7. This programming language was chosen due to its ubiquitous use across platforms, its open-source community, and its availability of great web programming tools. This enabled the program to utilize a convenient MQTT library Paho MQTT (http://www.eclipse.org/paho/), and will be good for interfacing with any web-based hosting.

Interrupt-Driven Program

The program itself is interrupt driven, and once the program has connected to the MQTT broker, will wait until a message comes over the “LSPic” topic which is the topic used to communicate the image data. This data is communicated in small segments to account for the small memory on the ESP8266. Every time a new message comes in, the program stores the data and looks for the end of the photo in a JPEG, which is “FFD9” in hex. After the program sees the FFD9 data from the photo, it knows that a complete photo has been transmitted, and then converts the hex data into a photo using the binascii library. Once the image is generated, it is stored somewhere that the mobile application can easily access it.

Subsystem Testing

This subsystem was tested by running the program and allowing it to wait for the image data to come through over the LSPic topic. The mobile app sent the “Take Photo” command to the ESP8266, and the ESP8266 took the photo. Then, the ESP8266 sent the data over the LSPic topic in segments 64 bytes long. The program automatically generated the new image, and saved it locally on the Python server. The Python script then uploads the image to a website hosted on AWS using the tinys3 library (https://www.smore.com/labs/tinys3/). This website is a static web page consisting only of Hyper-Text Markup Language (HTML). The final testing consisted of taking multiple images from the mobile application, one after another, to ensure that the most recent photo taken always shows up on the mobile application (after some <10 second delay).

The Python server was run using a Debian virtual machine. In reality, any machine capable of running Python could serve as the server. In fact, most of the unit testing was performed on a personal computer running Python 2.7.

Design and Operation of Network

The network consists of two main communications channels: the devices are all connected via an MQTT publish/subscribe protocol, and the image uploading and viewing occur over more traditional TCP/IP protocol.

The MQTT broker must be hosted as per the ISO/IEC PRF 20922 standards for message queuing telemetry transport (MQTT). This will eventually be a web-based system with login credentials. For development purposes and further prototyping, we are using the broker set up on a Raspberri Pi per the ISO standard above. Eventually, the same server will host the MQTT Protocol as well as the Server-Side webpage and data processing, instead of the disparate hosting utilized in the first design. The MQTT protocol is a publish/subscribe protocol, where the host keeps a log of multiple topics. In order for device A to communicate to device B, device A must publish a message to a topic that device B is subscribed to. So device B can subscribe to “test”, and device A can publish the message “Hello World” to the topic “test”, where the host will forward the message to all devices that are subscribed to “test” - in this case, device B.

In order for all of the devices to connect to the MQTT host, they must connect to it through a wifi router and declare the IP address of the host in order to connect.

Each ESP8266 device is connected to a functional device (camera, solenoid, etc) and this is a wired connection not discussed in this section. Please see the specific subsystem for wired connection information.

The TCP/IP is handled by tools such as the embedded web browser in mobile applications and the tinys3 Python library.

The above protocols were selected primarily for ease of use in addition to being able to fulfill our system requirements.

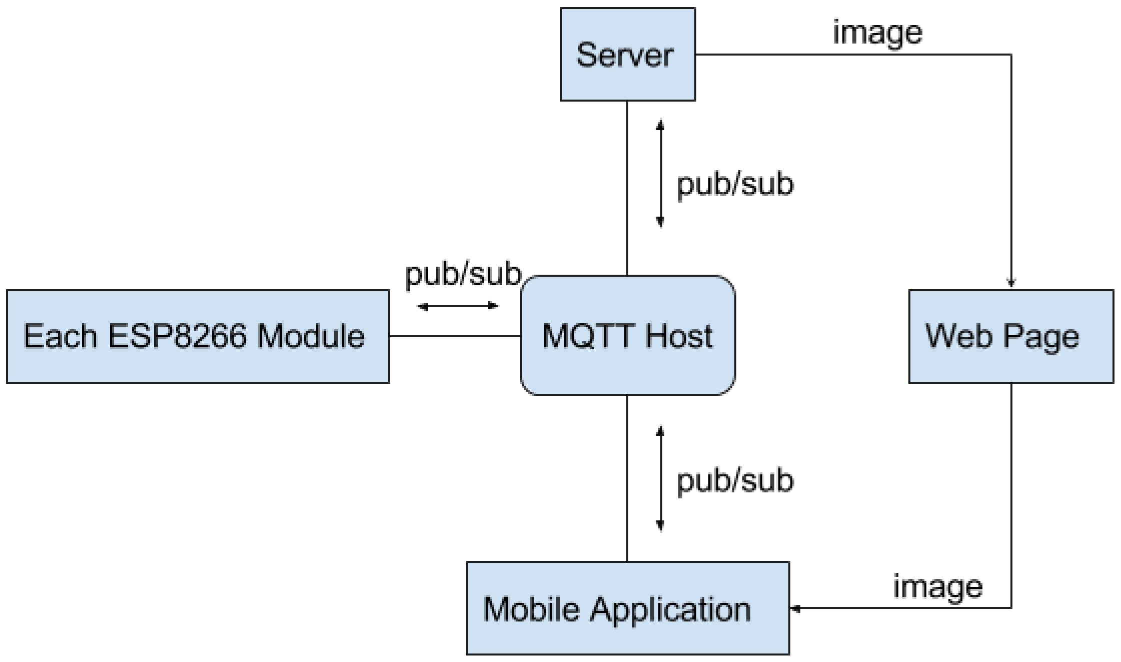

Figure 11: Wireless Network Diagram

*In the above figure, the wifi router/internet backbone connecting the devices to the host is implicit, and not explicitly shown.

The testing of this system is implicit in the testing of the communications of all the other subsystems.

Design and Operation of Mobile Application

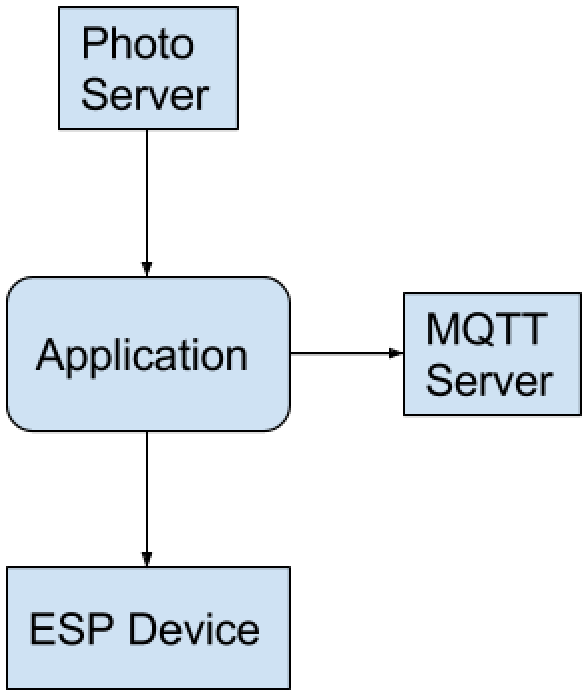

Figure 12: High Level Mobile Application Flowchart

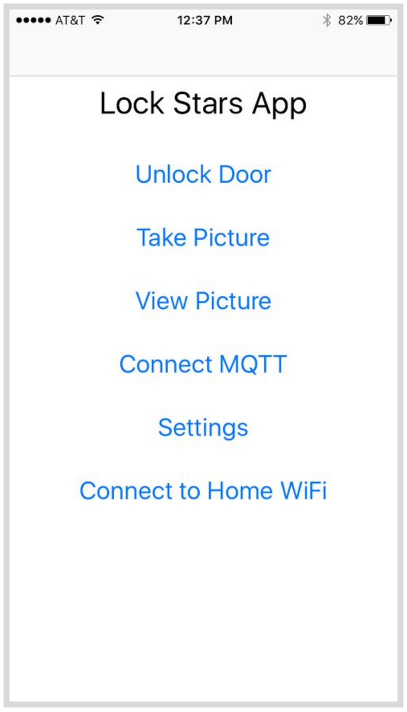

Figure 13: Mobile Application Screenshot

The mobile application is used as an interface a user to control the camera and solenoid and to view pictures taken with the camera. It is able to publish MQTT messages so that the strike opens the door for a user specified amount of time and so the camera takes a picture when desired. Once pictures are taken with the camera it is be able to access pictures taken from the camera on a web server. In addition to this, the application is able to connect to the ESP device and pass along home network information in order to connect to the network wirelessly. The ESP connection and home network information is stored locally in the settings for the app and can be updated at any time. The application is made for iOS devices and coded using the Swift language and CocoaMQTT client library. The operating system was chosen because of the widespread use of iOS devices, ease of integration with the other subsystems, and familiarity with app development in iOS. The CocoaMQTT client library was chosen because of it allows the application to easily use the MQTT protocol. During development of the app, it was tested using the iOS simulator built-in to the Xcode development program

The code used for the iOS application is included in the Appendix.

User Interface

During operation, the user interfaces with the system almost exclusively using the mobile application as described in the above section. In addition to the functionality of the application, the system will also respond to the doorbell being pressed by taking a picture in the same manner as if selected on the application.

These methods of user interface are described in greater detail in the Design and Operation of Camera Module and Design and Operation of Mobile Application.

System Integration Testing

System Testing

<u>Connecting Subsystems to Home WiFi</u>
The first thing to be tested on the integrated set of subsystems was the ability to connect the Camera Module and Lock Module to the home wireless network using the mobile application. This is done by connecting to the wireless network produced by the module on the mobile device, selecting the “Connect to Home WiFi” button on the application, following the instructions given, and repeating for all lock and camera modules.

<u>Taking and Viewing a Picture</u>
Taking a picture with the integrated systems is tested in 3 ways:

1. Taking a picture from the mobile application main menu
2. Taking a picture from the mobile application view picture view
3. Taking a picture by ringing the doorbell

To verify that the picture was taken and delivered to the web server, after each step press the “View Picture” button in order to access the view picture view.

<u>Unlocking the Door</u>
Unlocking the door with the integrated system is tested from 2 views in the mobile application:

1. The main menu
2. The view picture view

The functionality of unlocking the door is further tested by changing the desired time the door is left unlocked in the application settings and repeating the steps above.

Design Requirements Met

This section will detail how the system testing demonstrates that each of the design requirements were met, in the order they were originally introduced in the “System Requirements” section

- act as door lock when user is not home: the door lock was verified to remain locked unless the user unlocked it for a specific amount of time, after which the lock would return to being locked. This allows the system to act as a door lock when the user is not at home.
- camera should allow user to see who is at door: by interfacing the mobile app with the web hosting of the image taken by the camera, and the camera being also controlled by the interface, the testing verified that the user can indeed take a photo of whomever is at the door before deciding whether or not to unlock the door.
- camera to be triggered by doorbell or by app: during system testing, both the doorbell pushbutton and the mobile application were verified to be able to trigger the taking and updating of the photo.
- camera to send photo information over wifi: this was verified during system testing when the camera and accompanying ESP device were successfully able to upload an image to a web server that was viewable from multiple web browsers.
- photo information to be assembled into image: this was verified when the image data was verified to have been reconstructed into a file locally on the python server, and also verified to be repeatable and able to overwrite the previous file.
- image hosted on website: this was verified during the user testing when the image was accessible from multiple web pages.
- image accessible by registered devices: this was verified by having multiple devices (laptop, iPhone) be able to access the image.
- interface over wifi: this was ensured by use of MQTT protocols as well as traditional web browsing protocols
- solution is extensible to "n" devices: this is ensured by the modularity of development, where any additional device would only need its own device name, ESP device, and topics to publish/subscribe to. Everything else will support it. This was verified by interfacing multiple ESP devices controlling disparate functionality (the doorbell/lock and the camera).
- AC to DC converter capable of powering strike and ESP: this was verified after the construction of the boards was complete, and the functionality of both the camera and the solenoid were verified to meet the above described

specifications. They would not have been able to meet the above described if they had not been powered properly, and the power was stepped down from 120VAC house power.

User and Installation Manual

Installation

<u>Camera Installation</u>
Drill a hole above the door frame to place the camera in, facing outwards. You might consider protecting the camera with some thick plastic or plexiglass. Wire as described by schematics shown above in Design and Operation of Camera Module. Once the rest of the system is installed, test the system to ensure that the camera is capturing the right area, adjust if necessary.

<u>Solenoid Installation</u>
Make sure the door latch is flush against the wall when contracted, otherwise opening the door will be difficult or impossible. Wire as described by schematics shown above in Design and Operation of Solenoid Module.

<u>Python Server Installation</u>
The server should be centrally hosted so that the installation and setup just need to link to the proper server. However, if you want to install your own server, follow these instructions: First, download and install Python 2.7 to the machine you want to run the Python program on. Then download and install the accompanying libraries (tinys3, paho-mqtt - see Appendix A) to the same machine. You will also need to have a website to host the images from - instructions for using AWS are also linked to in Appendix A, or you can choose your own website (in which case you would not need tinys3). Then download the Python program listed in our Code section, update the AWS security keys in the code, and you should be able to run the program from a terminal.

<u>Mobile Application Installation</u>
Download the mobile application from the Apple Store (if this version of the app is not yet released, you will need to download the code from our Code section and install using XCode from an Apple laptop). An android version of the application will be released in a future update.

<u>ESP Device Installation</u>
The ESP Devices should be installed in proximity to their associated power boards or functional devices, and wires as described above in Project Description. They can be programmed with the codes on our Code section, and the IP address of the MQTT broker will need to be updated.

Set-Up

One the product is installed, the set-up can be done entirely over the mobile application. Once the application is downloaded onto a smartphone, the user should select the “Settings” option on the home screen of the app. This will take the user to lockStarsApp settings in the Setting app. The user should then put in the home WiFi information pertaining to the server the devices will be connected to. In the “MQTT” section, the user should input the same server address as in the home WiFi section, input port 1883, and then input a custom Client ID of the customer’s choice, but each phone should have a unique Client ID. The user can also select how long the door remains unlocked for in this menu.

After the information is filled, the user should search for close-by WiFi networks. Two networks should appear: LockStarsCamera #### and LockStarsLock ####. Connect to one of these networks with the password “LockStars”. Once connected to the network, open the lockStarsApp and click on the “Connect to Home WiFi” option, and then press connect. Repeat these steps with every LockStars_____ #### network until they no longer appear or no longer connect in settings.

Finally once all the devices are connected to the network, open the lockStarsApp and choose the “Connect to MQTT” option. Once a dialogue box appears to say it has connected, the user can unlock the door, take a picture, and view a picture all within the mobile application.

To unlock the door, select the “Unlock Door” option in the app. In order to change how long the door stays open, edit the amount of time in the lockStarsApp settings.

To take and view a picture, choose the “Take Picture” option, or choose the “View Picture” option followed by the “Take Picture” option. In order to view the last photo taken, and not take a new photo, just choose the “View Photo” option.

Working Verification

The product is working if the solenoid opens when the “Unlock Door” function is pressed in the mobile application, and a picture is taken when the “Take Picture” option is pressed. The camera and door lock modules should also have a green and red LED turned on when initially started up or whenever reset. By following the directions in the Set-Up section, the red light should turn off when the devices connect to the home network, leaving only the green LED on whenever the device has power.

If the solenoid and camera are not responding to the mobile application, read through the Troubleshooting section to find potential ways to fix the problems.

Troubleshooting

If a device is not working, first try to go through the steps listed in Set-Up and ensure the installation is correct. If the device can be seen as a network when searching for local WiFi networks, connect to the device with the password “LockStars” and follow the directions in Set-Up to connect the device to the home WiFi network. If the device still shows up as a network, check the information in the settings in the mobile app to make sure it is correct. If the device still does not work, make sure the green LED is on to make sure it is getting power. If none of the above fixes the issue, restart the device by pushing the button on it and then follow the steps listed in Set-Up.

If the mobile application is not working, make sure it is connected to MQTT by pressing the “Connect to MQTT” option on the home screen and wait for a message to come up saying it has connected. If it does not give the message, make sure the phone is connected to the home WiFi network and try again. If the problem persists or the app freezes, kill and restart the app or check for updates through the app store.

Conclusions

This section covers those changes which would be made to the prototype herein described in order to optimize the product for market launch.

Camera Changes
A nicer camera would likely be purchasing with a wider view lens than the one used in the prototype. Additionally, the camera used in the prototype came with a PCB with a lot of unnecessary functionality that we would not need, so by designing our own camera board to interface with the ESP board, a lot of efficiency for cost of production could be achieved.

Solenoid Changes
The actual lock could be a bit larger, and some people might like to have different options for purchasing. Additionally, a physical system that would allow someone inside the house to manually unlock the door in case of a power outage would be necessary before moving to market.

Power Changes
Rather than buying a part off of the shelve, our group could design our own method of AC/DC conversion to save money in the manufacturing of the device.

Server Changes
The python server and web hosting could all be made central and in the cloud. This product would have to be supported by its own servers rather than Notre Dame's and AWS. Additionally, the process could potentially be sped up with better code written for efficiency by writing our own MQTT routines.

Mobile Application Changes
The mobile application would undergo thorough usability testing to determine how best to optimize the layout by having users attempt to do the things they want to do with it. This will help make the app more intuitive and robust before product launch.

ESP Device Changes
The ESP Devices themselves would likely not be changed significantly. Some optimization of board layout for spatial efficiency might be needed to make the product cheaper, but that is the primary concern. From a software side, developments could be made to be able to update the firmware of WiFi.

Competition
Currently, the leading competitor in this market is the Ring Video Doorbell that currently goes for $199 per device. The advantages to this product is the capability for video and

two-way communication between the phone and the doorbell. However, this product does not allow for the user to actually control the lock on the door. Our product, with some modifications at minor costs, would be able to add video (which is already possible with the current camera) and two-way communication. Along with adding these features, we would be able to cut down our current cost (around $150) by optimizing certain parts and through mass production. Thus, our product would have higher functionality while also keeping a lower cost per device.

Appendix

Python 2.7

https://www.python.org/downloads/

tinys3

https://www.smore.com/labs/tinys3/

paho-mqtt

http://www.eclipse.org/paho/

Static Web Page Hosting on Amazon Web Services

http://docs.aws.amazon.com/gettingstarted/latest/swh/website-hosting-intro.html

Camera and Solenoid Modules (in Arduino ide) - Anyone at Notre Dame can View

https://drive.google.com/a/nd.edu/folderview?id=0B7jU7Q5jsHMbU21pcVpDendJdTg&usp=sharing

Mobile Application Code (in X-Code) - Anyone at Notre Dame can View

https://drive.google.com/a/nd.edu/folderview?id=0B7jU7Q5jsHMbUU9QdkpLZ0tLbDA&usp=sharing

Host and Server Code (in Python) - Anyone at Notre Dame can View

https://drive.google.com/a/nd.edu/folderview?id=0B7ptp35mHBa6WXc5a0lvMWtvZW8&usp=sharing

Eagle Files - Anyone at Notre Dame can View

https://drive.google.com/a/nd.edu/folderview?id=0B7jU7Q5jsHMbQVBTVUI0OXFQR2M&usp=sharing

Schematic and Board Design for Camera and Solenoid Modules

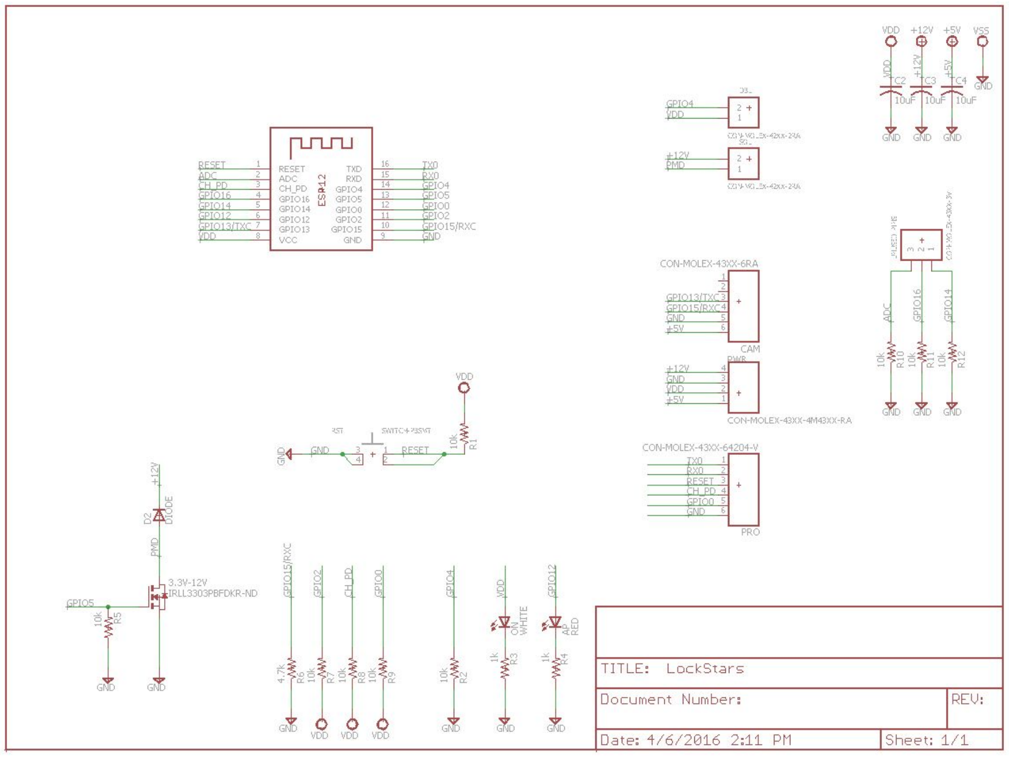

Figure 13: ESP Device Schematic

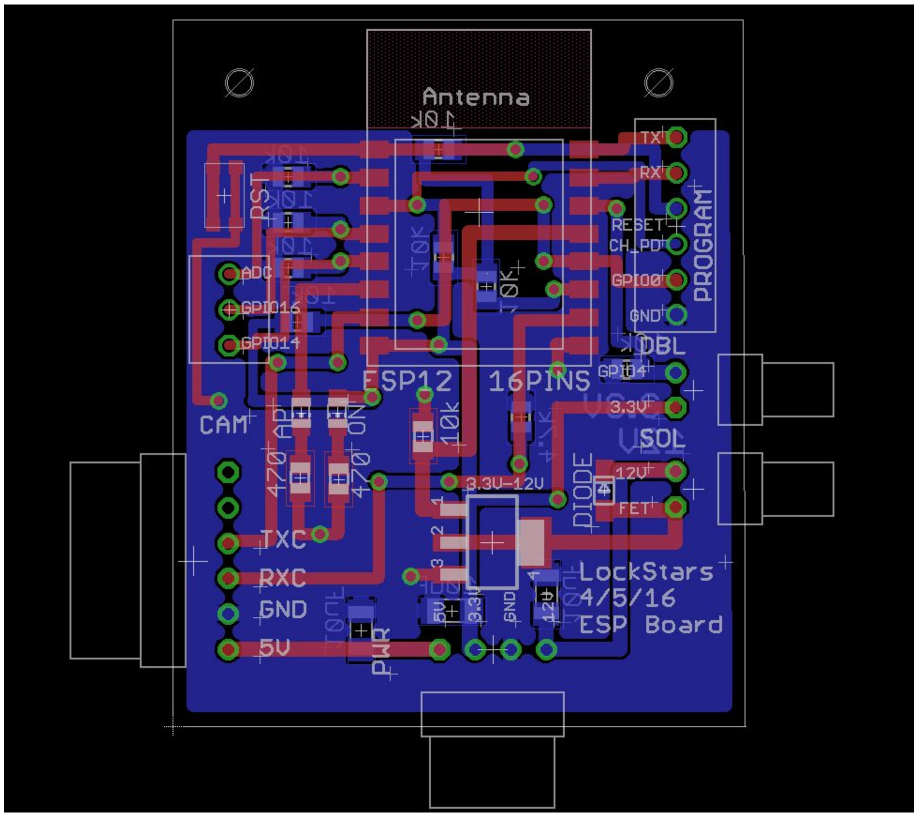

Figure 14: ESP Device Board

Schematic and Board Design for Power Board

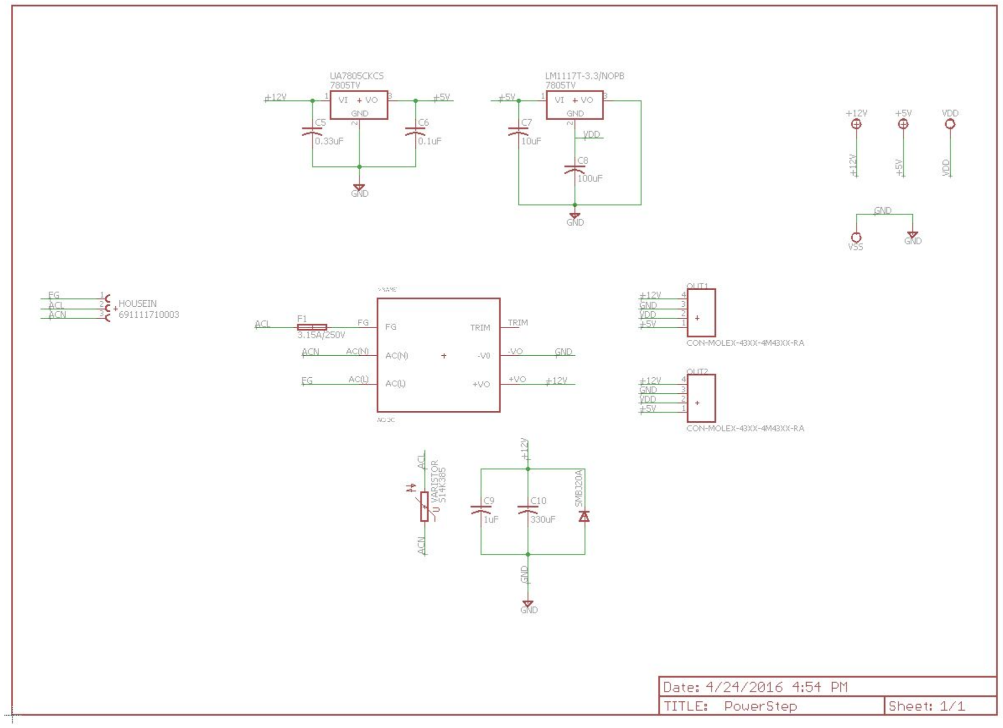

Figure 15: Power Board Device Schematic

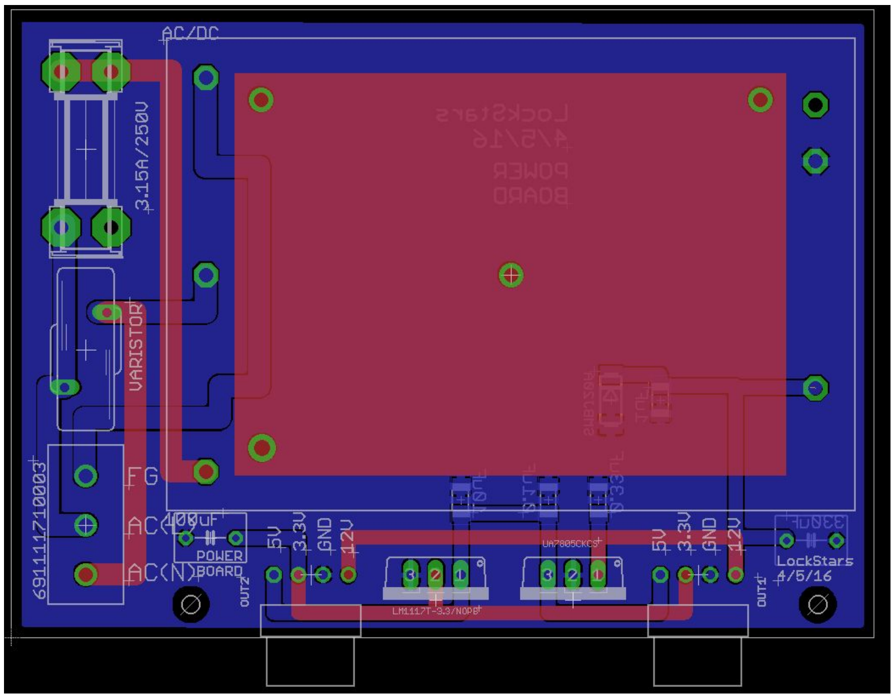

Figure 16: Power Board Device Board

Device Datasheets

ESP Datasheet:
https://cdn-shop.adafruit.com/product-files/2471/0A-ESP8266__Datasheet__EN_v4.3.pdf

AC/DC Converter Datasheet: http://www.cui.com/product/resource/vsk-s25.pdf

Terminal Screw Block Datasheet:
http://media.digikey.com/pdf/Data%20Sheets/Wurth%20Electronics%20PDFs/Terminal%20Blocks%20WR-TBL.pdf

MOSFET Datasheet: http://www.irf.com/product-info/datasheets/data/irll3303pbf.pdf

5 Volt Regulator Datasheet: http://www.ti.com/lit/ds/symlink/ua7805.pdf

3.3 Volt Regulator Datasheet: http://www.ti.com/lit/ds/symlink/lm1117.pdf

Varistor Datasheet: http://en.tdk.eu/inf/70/ds/SIOV_S14KE2K1_AdvanceD_MP.pdf

TVS Datasheet: https://www.fairchildsemi.com/datasheets/SM/SMBJ100A.pdf

Fuse Datasheet:
http://www.littelfuse.com/~/media/electronics/datasheets/fuses/littelfuse_fuse_218_datasheet.pdf.pdf

Fuse Holder Datasheet:
http://www.littelfuse.com/~/media/electronics/datasheets/fuse_blocks/littelfuse_fuse_block_656_datasheet.pdf.pdf