**Dwelling Security Interface:**

**A Narrative Thereunto**

Project created by Gereth Dittrick

The inspiration of this project comes from the combination of my own forgetfulness and resourcefulness. Since I typically walk to and from campus daily, I don’t need my keys to leave work or class, which also leads to me leaving them at work often. Typically leaving my keys at work leads to me being locked out of my apartment, which also leads to me picking the lock with paperclips, notebook reels or whatever else I had in my bag at the time. Since I had absolutely no experience picking locks when I did this for the first time, it was a bit unnerving.

The basic structure of the project consists of three major pieces, the base station, its sensors, and the main server which is mostly for storing information. The idea behind this setup was to push as much of the responsibility to the individual base stations so that it was possible for the project to scale if desired. The only thing the main server would contain are the backups of the base stations, the detailed information of the account holder (let’s pretend this was intended to be a marketed service) and their payment information, in addition to acting as a routing table for owners to access their base station’s website seamlessly.

The base station of the project also consists of three major parts being the database handle, the GUI, and the networking infrastructure. For each of these I opted to use the façade pattern, especially since when I initially wrote the code I hadn’t finished designing the specifics of how each of those sections would be implemented or what they would be implemented with.

The hardest part of this project was not only implementing the networking infrastructure, but doing it in an organized was that was easy to read and debug, rather than what you might see for some of Randy’s networking classes. What I ended up settling on was an implementation that attempted to mimic the inherent key-value pair functionality you see in JavaScript objects using the TreeMap class in java which resembles a red black tree. Using the TreeMap and a simple interface above, I boiled the entirety of the networking function of the base station down to the implementation of a single method call once the table was initialized as shown below.

package LIDS.Server.NetworkEvents;

import java.io.BufferedOutputStream;

public interface NetworkEvent{

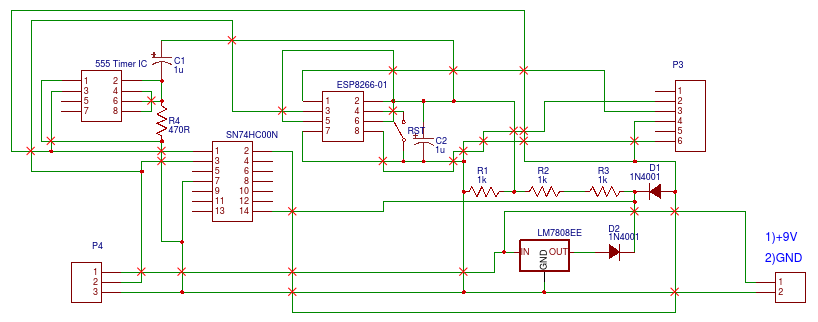
public void execute(String eventData, Object aFileNode, BufferedOutputStream aWriteStream);

public void execute(JsonObject aJsonObject, Sensor aSensor, BufferedOutputStream aWriteStream);

}

The next portion of the project was the sensors. The sensors themselves were quite plug and play assuming you plug the output into an interrupt capable pin. The sensor itself measures via infrared, and has a range of 7 meters or just under 23 feet for those who prefer imperial units, and a spread of 110 degrees. The unit also has an operating voltage range of 5V to 20V allowing for a lot of possibilities for what I might hook it up to. The downside to this module is that the sensors receive a lot of interference from direct sunlight and temperatures exceeding 32 degrees Celsius or just under 90 degrees Fahrenheit which is problematics if you live somewhere like Arizona, or really anywhere in the United States during the summer.

The wireless connection was the difficult part. Initially I was going to use general RF bands via the NRF24L01 chip. This would have been the easiest option had the chip had the chip a stable API to work with or documentation to go with it. In addition, flashing the firmware onto the chip was also more difficult than necessary since there are fifteen different models, in which three different websites will tell you three different ways of flashing the firmware onto the chip. Ultimately It was necessary to move on to something more stable.

My next choice was the ESP8266 which had a full system embedded within the chip, and it also had a more stable code base. Unfortunately, to use this and update it reliably, I would have had to create and print off an additional circuit board to support the ability to flash the firmware when the USB cable (adapter) was plugged in on boot. With time slipping away and better, faster, and more user-friendly options available, I abandoned the schematics below. 

My current choice communication of the sensor status is Arduino’s MKR1000 board sporting an ARM Cortex 0+ processor, this being by far the most powerful but efficient choice thus far. Another positive of this board is the Li-Po battery charger that is built in, which could easily allow for the use of a battery backup in the event of loss of the wired power. Unfortunately running off said battery would limit voltage to 3.7V out of the 5V output pin which is less than the required voltage for the sensor but it would still allow the board to report this, especially in the event of tampering.

Probably one of the most aggravating parts of this project was trying to get the sensors to work properly. Even in the final stages of the project, the sensors still do not work, and at this point I am unwilling to see if the Arduino mkr1000 I have is defective or not since they are 40$ before shipping. Currently I can at the very least trigger all the appropriate code for some very basic testing. The issue with the board I have is that the board does a reset after connecting to the base station for about 3-5 seconds. This issues causes messages to be received and then immediately deleted since the sensor is no longer connected and therefore removed.

The most important thing I learned very painstakingly is that sufficient design is necessary, otherwise I end up throwing away a lot of code. One of the large challenges of this project stems from the fact that I attempted to start this project very early on before learning proper design patterns and networking protocols, in which I ended up throwing out a lot of code over the course of the project due to deeply nested bugs that could be removed via the use of restructuring of the project code.   
 The problems aside the project did end up coming along nicely, although there are a few things I would do quite differently. The most important thing I would do differently would be to spend more time researching solutions rather than attempting to take the easy way out. An example of this was when I attempted to run a NodeJs server underneath my project, adding an extra communication barrier to the project making the project more clunky and resource hungry. In comparison, writing the code for implementing a webserver on top of the “built in” code for generating an SSL server socket, which it did end up being more code, but the removal of the extra communication layer made the project easier to debug and therefore better overall. Had I researched properly, I would have discovered the Javadoc’s for the SSL code and started with that I would have saved many hours worth of work.

Many of the hours put into the project were working with Java code since most of the project consists thereof. Other technologies were also used such as PHP for the main server, MySQL for the database, JavaScript to implement functionality in the web pages, and Arduino C++ to implement the Sensors for the project. The requirements to run the program for the base station are to be running a Debian Linux distribution that supports bash, and can run Java 8. Since the idea for the project was to be cheap to implement for the average user, the project was designed to be limited to the processing power of the Raspberry Pi 2B. The project can be run from windows as well but due to my lack of knowledge of the windows command prompt and writing .bat scripts, installation of its dependencies must be done by hand, installing MySQL, Java etc.

In the end, I can’t say that the project was any harder or easier than I initially predicted. I expected the project to be time consuming and end up have a comparatively large code base compared to the minimum requirements. Part of the reason I started the project so early was because I knew I would end up cutting features due to time. In the future I intend to continue to build upon this and continuously add features as eventually I would like to use the project as a basis for a moderate amount of home automation and interconnectivity, especially if I were capable of creating an AI to go with it.