Sentinel

By

Vinit Kumbharkar, Mitchell Meyer, Ryan Pecor

Submitted to
the Faculty of the School of Information Technology
in Partial Fulfillment of the Requirements for
the Degree of Bachelor of Science
in Information Technology

Vinit Kumbharkar	4/17/17 Date
Ryan Pecor	4/17/17 Date
Mitchell Meyer	4/17/17 Date
Russell McMahon, Faculty Advisor	4/17/2017 Date
Robin Carew, Faculty Advisor	4/17/17 Date

University of Cincinnati
College of
Education, Criminal Justice, and Human Services

April 2017

Contents

Acknowledgement

We would like to thank the University of Cincinnati and the School of Information Technology for this opportunity to work on a project of this size and scope. Spending two full semesters planning and developing Sentinel has been challenging, fun and rewarding. We would like to personally thank Professor James Scott, Professor Patrick Kumpf, Professor Robin Carew and Professor Russell McMahon. The guidance and support they gave to us and all the Senior Design groups was invaluable and lead to great results in our projects. We would also like to thank the numerous open source projects we utilized throughout the development of Sentinel. These resources helped us build a reliable and robust platform that fulfills many of our requirements.

Abstract

Homes and apartments are drastically more likely to be broken into without a security system. Even though it may seem small, a security installation sign is enough to deter criminals. However, some criminals are bold enough to attempt a break in; for that reason, we created Sentinel. Many existing home security systems are expensive, all encompassing, and require annual subscriptions for monitoring and support. Sentinel is a home security automation system that focuses on ease of installation, affordability and accessibility. Instead of having an all or nothing system, a user can integrate as many or as few of the supported components. Each component is seamlessly integrated into the web application allowing users to monitor and control their home entry points from anywhere. With localized voice integration, a simple phrase can lock down a user's home and control individual components.

Problem Statement

The standard lock and key is often not enough to stop a criminal from getting into someone's home. Houses and apartments are more likely to be broken into without a security system in place, and without this level of security to deter or catch the intruder, damages and lost property costs can add up. Existing security systems are costly to install, require monthly payments, and can be difficult to install in a temporary home or apartment. Our solution solves these issues by being inexpensive, modular and completely open source.

Project Description

We have created a home security automation system that can be controlled via voice or web application. The solution can control and monitor basic amenities such as lights, blinds, and motion sensors to detect movement in specific locations. By making the system voice controlled, we add another layer of simple accessibility from within the house. While away, a user can monitor and control their amenities in their home from anywhere in the world. The components are modular so that the user is only required to buy what they need. We have made it relatively easy to install for the average DIY user. It has a one-time cost for the supplies and software, and does not have a monthly subscription.

Technical Elements

Hardware

The system is built using at least one Arduino and a Raspberry Pi. The web application is accessible from any browser and has been built to be mobile friendly and compatible on all devices. A microphone is being used for voice input, a small motor for blinds control and motion sensors for door monitoring. In addition, we are using the standard breadboards, wires and required Arduino components.

Software

We built a web application that can run on any device with access to a web browser. The application is based in Node.js and we have written much of our front and back end in JavaScript. The MEAN stack, with React as opposed to Angular, is our full stack development environment. Mongo DB is used to store user information and component data, Express to display and handle

the UI and routes, React to organize the UI components, and Node to run as the server. This stack allows the most flexibility for us and allows us to maintain full control of the communication between devices. Github is used as our central repository where our code currently rests.

Network

Our network consists of a Raspberry Pi, Arduinos, and the Web Client that is being hosted on Heroku. The Raspberry Pi sits in a house connected via Wi-Fi. It talks with the website server over WebSockets to provide wireless real time bi-directional communication. When the Raspberry Pi receives a message over WebSockets, it executes code that talks to all connected clients updating the website, database, and Arduinos.

Security

Security is a major part of this project and there are many places that data could be stolen or controlled for malicious intent. One of the biggest threats that we faced was the ability to control Sentinel via voice from outside of the house. It wouldn't be hard for a criminal to loudly shout a malicious command. To combat this, the voice functionality does not control vital components. In addition, we have implemented a verbal password that allows the voice aspect of Sentinel to be locked or unlocked. When leaving the house, you can verbally lock down voice control. In future versions of Sentinel, we would like to have voice control work only when a recognized device is nearby. This would allow Sentinel to automatically lock itself down when an authorized user leaves the home.

Data transmission is another point of security concern. WebSockets handles communication between the local Node.js server, web client and Arduinos. We are also responsible for controlling each device's credentials and assigning them with unique identities that

match up to the users specific Raspberry Pi. This, along with user specific permissions and authentication help prevent other unauthorized outsiders from accessing and controlling Sentinel.

For the web application, as aforementioned, we have the user authenticated via site login. In future versions, we hope to add multifactor authentication which makes a password breach nearly impossible.

Objectives/Deliverables

The following section contains written deliverables that helped plan key development targets. **Figure 1** was made before the project began and **Figure 2** was made at the halfway point and only represents the second semester only.

Figure 1. Objectives/Deliverables

	Objective/Deliverable	**Date**
1	Finalize list of features	09/15/16
2	Research hardware/software to use	09/22/16
3	Define Initial idea of proof of concept	10/02/16
4	Buy appropriate hardware	10/11/16
5	Understand concepts behind Arduino, Raspberry Pi and underlying network	10/22/16
6	Build prototype	11/10/16
7	First presentation	11/21/16
8	Integrate various components	01/08/17
9	Add voice control features	01/22/17
10	Add mobile app	02/10/17
11	Testing and Debugging	03/02/17
12	Final Cleanup	03/20/17
13	Prepare for final presentation and present	TBD

Figure 2. Objectives/Deliverables Second Semester

• Wireless Communication Research o Researching various possibilities for Arduino and Raspberry Pi(s) to work wirelessly.	09/Jan/2017	23/Jan/2017
• Buying Necessary Wireless Hardware o Using data collected after initial research, buying wireless shield or newer version of wireless Arduino's with other necessary capabilities.	16/Jan/2017	22/Jan/2017

• Testing Wireless Hardware
  o Prototyping and creating web service/initial web server to control Arduino
  o Moving the web service/server to a remote location
  o Remote location testing the connection with Raspberry Pi
  o Using the Raspberry Pi to connect to Arduino
	23/Jan/2017	29/Jan/2017

• Planning and Building the Website
  o Planning initial layout and UX for the website.
  o Creating WireFrames
	30/Jan/2017	19/Mar/2017

• Web app Front-end o Coding the front-end of the website as per layouts from the last stage.	30/Jan/2017	12/Feb/2017
• Web app Back-end o Coding the back-end (web services, server side scripts).	13/Feb/2017	26/Feb/2017
• Final Website Clean-up o Final clean up and refactoring of the code.	06/Mar/2017	13/Mar/2017
• Implementing Electrical Hardware o Using amplifiers etc. to manage the electricity between Arduinos and home appliances.	06/Mar/2017	26/Mar/2017
• Buying Electrical Hardware o Buying necessary equipment (amplifiers, cords, cables etc.)	06/Mar/2017	12/Mar/2017
• Integrating and Testing Arduino/Raspberry Pi o Final integration of the entire ecosystem.	20/Mar/2017	26/Mar/2017
• Final Presentation Preparation	03/Apr/2017	10/Apr/2017
• IT Expo Preparation		

User Profile

Figure 3 represents our user profile and who may use our system and how they interact with it.

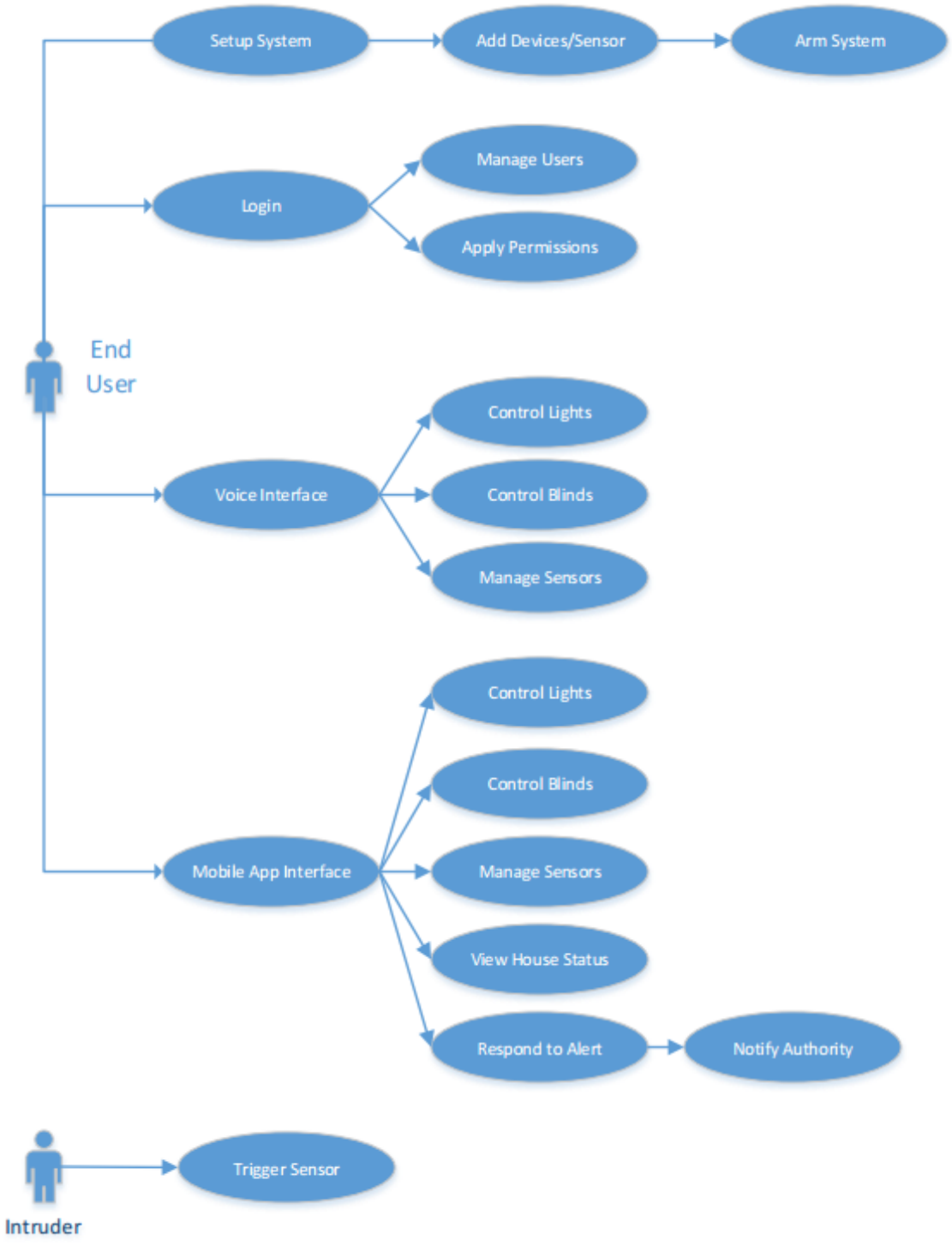
Figure 3. User Profile Diagram

User profile
Potential Users Home or apartment renters that want a modular security system on a budget.
Software and Interface Experience There are two major ways of interacting with the security system. Users can control the system with either their voice or via Sentinel's website.
Experience with Similar Applications Sentinel is like other voice and app controlled applications in terms of usability. Most of the controls are familiar to the average users who have experimented with speech recognition such as Google or Amazon Alexa, or who frequently surf the web.
Task Experience A user may have mid-level knowledge on using computers to be able to set up the system. Once setup, it is easy for anyone to control Sentinel via voice or the website.
Frequency of Use The system can be used at various times throughout the day. It is used most heavily after leaving and entering a building secured by the system. In addition, Sentinel sees more frequent use when a user is away from the home for extended periods of time.

Use Case Diagram

The following figure represents and shows the potential user interactions with Sentinel.

Figure 4. Use Case Diagram



Budget

Table 1 represents the materials that were required to complete this project. Most of the costs that were accumulated over the lifespan of this project were in the hardware that we needed to purchase such as the Arduinos and Raspberry Pi. We also incurred costs when purchasing lights and blinds to test functionality. When it comes to software, we primarily used free open-source libraries and developed our own solution. The rest of the software such as the Google Voice STT API were obtained through a free student license. The student license was somewhat limited compared to the full purchased versions, but for a project of our size, it worked out fine. If this continued into a full business after college, it would be easy to upgrade to the more robust paid versions. The development of the software took place on our own personal laptops, Arduinos, and on the Raspberry Pi itself.

Table 2 represents our projected labor costs that could have been paid out had we been compensated to complete this project. We came up with this number based on the difficulty of the project, similar pay rates in this field, and the average pay of our previous Co-op jobs.

Table 1. Materials Budget

Name	Amount	Unit Cost	Cost
Arduino	3	$25	$75
Raspberry Pi	1	$35	$35
Components for Arduinos and Raspberry Pi(s)	N/A	N/A	$100
Misc.	N/A	N/A	~$140
Total:			**$350**

Table 2. Labor Costs

Name	Hours	Pay Rate	Cost
Mitchell Meyer	80	$22/hr	$1,760
Ryan Pecor	80	$22/hr	$1,760
Vinit Kumbharkar	80	$22/hr	$1,760
Total:			**$5,280**

Test Plan

Overview

Below we discuss the methodologies and scope of our testing. The primary points of user testing were user input. These include the web application and voice control. Technical testing included confirmation of our use cases and the platform functionality. All developers doubled as QA analysts.

Scope

The scope of our testing included testing voice functionality, web client interaction, and hardware communication. It was important to make sure that the tests passed, not only in a desktop environment, but also on mobile phones and browsers. Tests were run by developers and a small group of early users. Developers created tests for the server functionality and client side functionality.

Objective

We confirmed that our application worked as expected. User input was received by both the voice control system and the web client and there were no conflicts with each other; the WebSocket connections worked as expected and everything was in sync. Speed, stability and security were the primary testing areas for both the server and client.

Testing Demographic

We had a few different demographics test Sentinel. An average user of any age could efficiently navigate and use either the web client or voice control to control components.

Ecosystem Testing

Sentinel was tested as an ecosystem in its entirety with all components active including Web Application, Arduino(s), Raspberry Pi, Voice Recognition Engine, and Remote Connectivity working together. Individual tests of components were limited to identifying defects, or testing component dependency.

Testing Procedures

As our product consists of hardware and software we used multiple layers of testing.

Steps for testing are mentioned below:

- Compile a list of Use Case Scenarios.
- Create test documentation for each Use Case Scenario
- Program necessary automated tests and documentation for the code base.
    - Choose a testing framework and library for Web Application testing.
    - Create Unit Tests for the Web Application
    - Build technical documentation using above unit tests.

Tests were performed as follows:

- Hardware Tests
    - These tests were performed to check functionality of all the hardware involved.
    - Example - Testing Arduino IDE and Code to turn a led on and off
- Software Tests
    - These involved testing of the Web Application and any other programming code involved.
    - Example - Unit Tests for the web app, and local Node.js server.
- Ecosystem Tests
    - These tests involved testing of connectivity between software, hardware and remote Node.js server.
    - Example - Turn a light on using voice command “Turn on Light”, or remotely control a function via web app.

Pass Fail Conditions

Sentinel was required to pass all written tests. Failed tests were documented and tracked for later fixing.

Team Member Testing Schedule

Tester	**Timeline**	**Frequency**
Developer	Feb. 6th - April 10th	Weekly Reviews
Developer (As QA)	Feb. 6th - April 10th	Bi-Weekly
Initial User	May 20th - April 5th	Four User Sessions

Written Tests

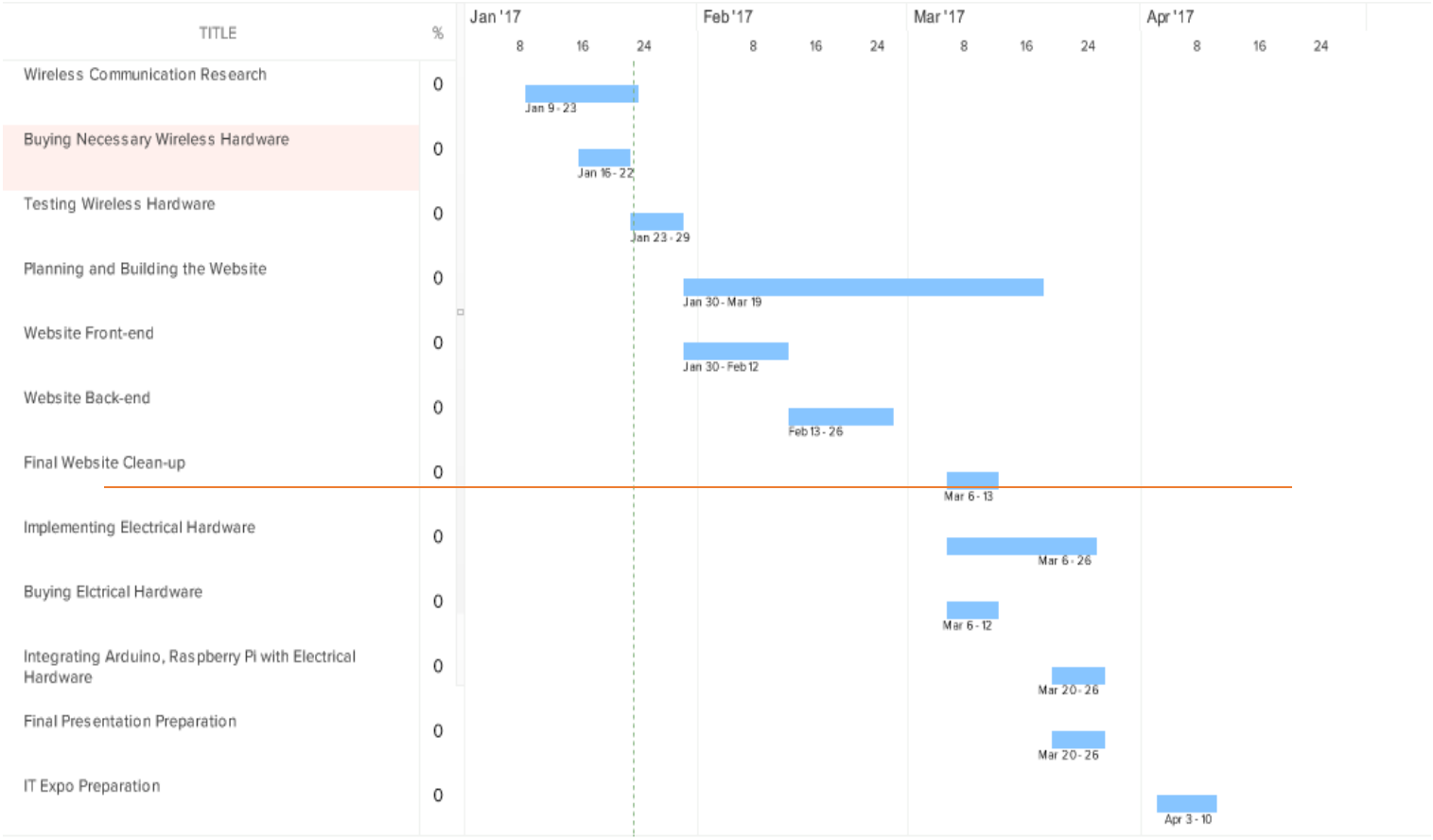
Tests	Steps	Expected Result	Pass/Fail
Control lights with website	1. User navigates to controls dashboard 2. User moves brightness sliders	Lights brighten and dim in accordance with website controls	Pass
Control lights via voice	1. User issues voice command 2. System responds acknowledging request	User issues command with specified light intensity and the light and website update accordingly	Pass
Control blinds with app	1. User navigates to controls dashboard 2. User moves blinds sliders	Blinds raise and lower in accordance with website controls	Pass
Control blinds with voice	1. User issues voice command 2. System responds acknowledging request	User issues command with blind height. Website and blinds update accordingly	Pass
Test door/windows sensors	1. Window or door is opened 2. System receives notification 3. User is notified	When door or window is opened, user is notified via website	Pass
See status of devices through app	1. User navigates to dashboard 2. Status of devices are shown	User can view the status of their devices on the website	Pass
Use voice to retrieve status of devices	1. User issues voice command 2. System responds with Sentinel device statistics	When asked, voice responds with status of currently connected Sentinel devices	Pass
User login	1. User navigates to login screen 2. User inputs their credentials 3. User hits log in button 4. If credentials are correct, they are directed to dashboard	When supplied with valid login credentials, user is logged into website	Pass
User log off	1. User clicks logout button 2. They are logged off and redirected to home page	User is logged off and redirected to home page.	Pass
Create new user	1. User clicks register 2. User is redirected to create user form 3. User fills in form and hits submit 4. User is redirected to dashboard	Verify no duplicates already in system and create user if form data is valid	Pass
Test wireless	1. Connect devices to internet over Wi-Fi 2. Use dashboard or voice to control devices	When devices are connected wirelessly, system should still function	Pass

Gantt Charts

1st Semester:

	Week1	Week2	Week3	Week4	Week5	Week6	Week7	Week8	Week9	Week10
Finalize Core Features										
Research Technologies										
Purchase Arduino/First Tests										
Software/hardware integration development										
Voice Control Development										
Proof of concept										
Finish Voice control Development										
Prepare Prototype										
Prepare Presentation										
Implement Feedback										

2nd Semester:



Problems Encountered

Hardware

As we began our research there were numerous challenges that we came across before we even began working. We needed to do a lot of research in order to buy parts for our project. One tough decision was which Arduino's to purchase. We had an Uno board initially, and it was great because it came with a detailed kit-- but it doesn't allow easy Wi-Fi connectivity. The issue was deciding whether we needed more components for our current board, or to buy different boards altogether. We decided to do more research and ended up with the Arduino MKR1000, which has built in Wi-Fi. We had already learned about the circuitry of the Arduino's and how the operating system worked making it an easy transition to the MKR1000.

The new board allowed us to connect to the server running on the Raspberry Pi. Working with WebSockets on this board was a challenge, but the Wi-Fi communications provided a fast and reliable connection. About half way through this project the raspberry pi zero wireless was released. This was another competitor to the Arduino since its size was equal to the Arduino and it too had Wi-Fi built in. Ultimately, the Arduino worked out best for our particular use case and is what we decided to go with.

Software

The software infrastructure behind our project has grown significantly since we began. We initially wanted to run our device connections on AWS IoT. This would allow us to focus on the application development. After getting started with AWS though we quickly realized we would lose too much control over our project and decided to work with our own Node.js servers and

Heroku. This lead to the issue of handling our own communications. We initially thought to build a REST API to handle the communications between our devices. This would have worked but we soon discovered a more intuitive solution that would work better for what we want, WebSockets. Websockets enables a real time, bi-directional communication between web clients and servers. Socket.io, which is a subset of WebSockets that is specifically used to interact with a website, ran on our Heroku server while Websockets ran on our Raspberry Pi. At this point we had total control over our front-end, back-end, user authentication and hardware communications. We realized we were building more than just an application; we were building a platform.

Another area where we had significant roadblocks was with our voice recognition implementation. Our first choice of voice recognition was Pocketsphinx. This was the recommended engine for voice recognition on the Raspberry Pi. We first ran into issues when required libraries that were supposed to be included in the installation were not. We needed to go searching on the internet for those files and manually install them. We eventually ran into another issue installing a few python libraries on one of our last dependencies. With problems continuing to pile up we turned to another voice recognition platform called Jasper. Installation on Raspbian was a little complicated, but the benefits after setup were substantial. Google STT is very well known and the installation and configuration with Jasper was much simpler than Pocketsphinx. The installation was smooth, as was generating our Google API code. The solution still wasn't perfect though. Jasper's instructions were out of date and we ended up having to do quite a bit of trouble-shooting and it was very slow. Since Jasper was entirely run on the Raspberry Pi enabling offline capabilities, it sacrificed speed. The platform also didn't recognize phrases very well, so we ended up repeating ourselves quite a bit. And finally, we only had 1000 calls to the Google STT platform per day—we burned through those very quickly each time we tested. We had to say

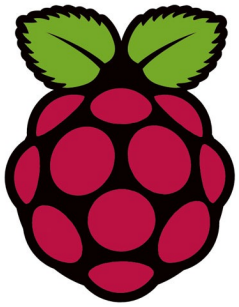
the phrase multiple times before it picked up what was being said, and once it finally matched the correct phrase, took about 10 seconds for the code to be executed. We got Jasper working, but the performance issues were a problem and we knew that this solution would not work for tech expo. Throughout the entirety of this project we considered Amazon's Alexa, but we didn't make a break through until later in the project. Once we finally figured out how to hook up Alexa with the platform that we were creating, we knew that this would be the best option. Since our team already had access to an Amazon Dot, there was no need to purchase extra materials. Alexa fixed all the previous problems that were stated above with Jasper and its flexibility. The ease of integration with users already existing Amazon Alexa devices made it a great solution.

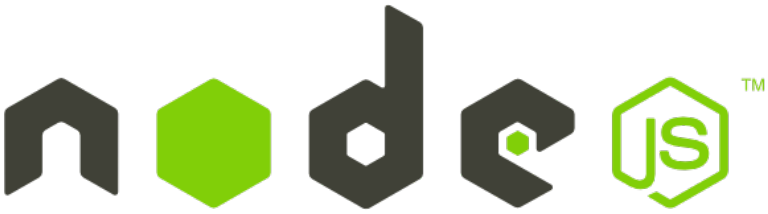
Conclusion

Home security continues to remain an important part of day to day life. Many options currently exist to support a wide range of customers, but Sentinel has a niche of its' own. Sentinel provides a cost-effective solution without compromising security by using inexpensive hardware, and open source software. Our platform addresses a share of the market that has been overlooked by large-scale security systems built for wealthier customers who plan on staying in one home for an extended period. Not only have we provided a unique solution, but working on this project has given us ample opportunity to explore technologies untouched by the traditional software development curriculum.

As the project progressed we learned how to interface with low level hardware such as the Arduino and Raspberry Pi. We also had the opportunity to dive deep into voice recognition. Building the entire platform from the ground up using node.js has given us even more experience in large scale platform projects. Sentinel is at an outstanding functional place right now and could be easily expanded upon.

References

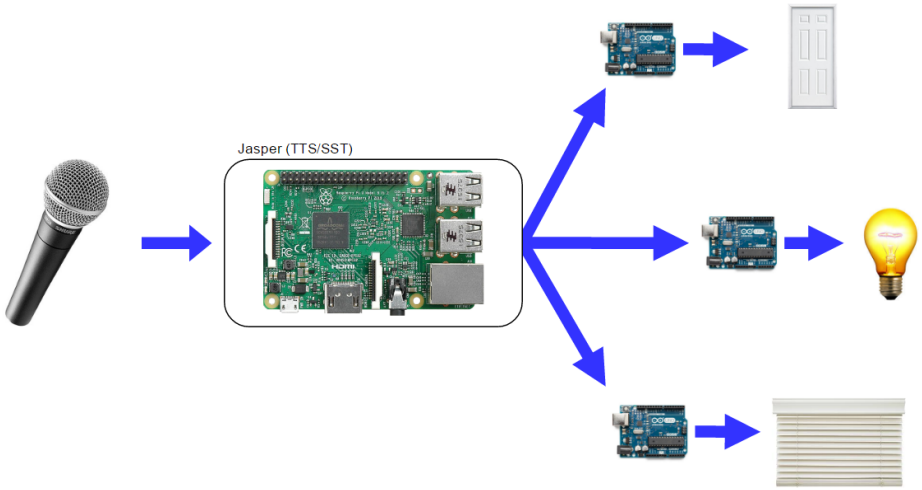




IoT Automation Group – West Chester, Ohio.

Appendix

Voice Functional Diagram



Code Snippets

This is our Python module for turning on lights from our voice command. This is handled within Jasper, after the speech engine has done its magic.

turnlighton.py

File Edit Search Options Help

```
# -*- coding: utf-8-*-
import random
import re
import subprocess

WORDS = ["TURN", "ON", "LIGHT"]


def handle(text, mic, profile):

    messages = ["Okay. Turning on light.",
                "Right away."]

    message = random.choice(messages)

    mic.say(message)

    subprocess.call("/home/pi/jasper/client/modules/picommands/lighton.sh", shell=True)


def isValid(text):

    return bool(re.search(r'\bturn on light\b', text, re.IGNORECASE))
```

This speech module runs this script (written and read in Python), when the string of words is recognized.

lighton.sh

File Edit Search Options Help

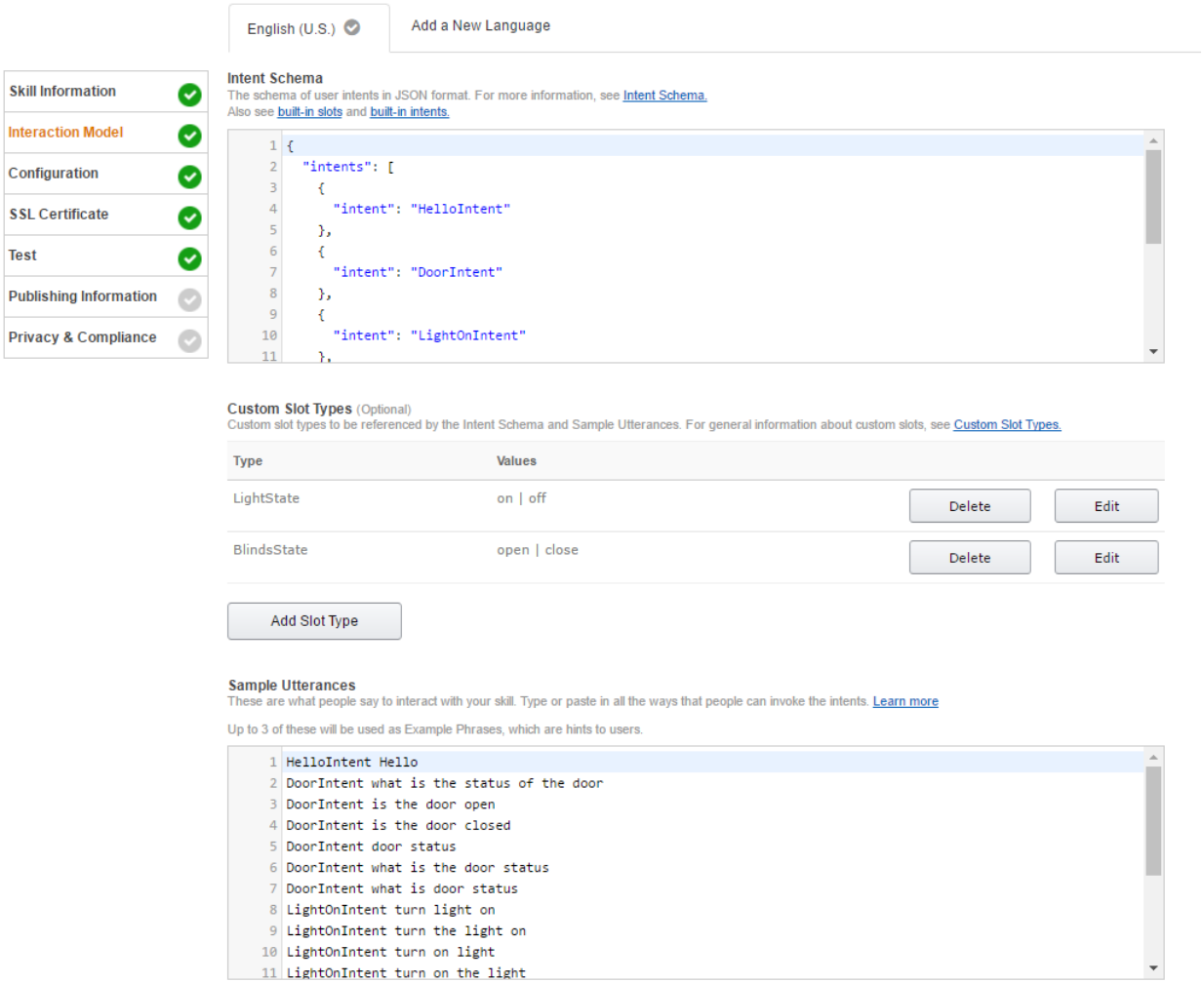
```
#!/usr/bin/env python

import serial
ser = serial.Serial('/dev/ttyACM0', 9600)
ser.isOpen()
print ser.portstr
ser.write('1')
ser.close()
```

Jasper running and waiting for voice input.

```
pi@raspberrypi ~ $ /home/pi/jasper/jasper.py
*******************************************************
*             JASPER - THE TALKING COMPUTER           *
* (c) 2015 Shubhro Saha, Charlie Marsh & Jan Holthuis *
*******************************************************
ALSA lib pcm.c:2217:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.rear
ALSA lib pcm.c:2217:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.center_lfe
ALSA lib pcm.c:2217:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.side
ALSA lib pcm.c:2217:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.hdmi
ALSA lib pcm.c:2217:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.hdmi
ALSA lib pcm.c:2217:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.modem
ALSA lib pcm.c:2217:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.modem
ALSA lib pcm.c:2217:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.phoneline
ALSA lib pcm.c:2217:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.phoneline
ALSA lib pcm_dmix.c:957:(snd_pcm_dmix_open) The dmix plugin supports only playback stream
Cannot connect to server socket err = No such file or directory
Cannot connect to server request channel
jack server is not running or cannot be started
WARNING:client.notifier:gmail_address or gmail_password not set in profile, Gmail notifier will not be used
No disturbance detected
No disturbance detected
WARNING:client.stt:Empty response: Nothing has been transcribed.
WARNING:client.stt:Empty response: Nothing has been transcribed.
```

Alexa's Interaction Model Interface



Alexa's back-end server code example

```
alexiaApp.intent('HelloIntent', 'Hello', () => {
  return 'Hello from Sentinel';
});
alexiaApp.intent('BlindsOpenIntent', 'Hello', () => {
  socket.emit("blind", "open");
  return 'Opening blinds';
});
alexiaApp.intent('BlindsCloseIntent', 'Hello', () => {
  socket.emit("blind", "close");
  return 'Closing Blinds';
});
alexiaApp.intent('LightOnIntent', 'Hello', () => {
  socket.emit("light", "on");
  return 'Turning light on';
});
```