DIY Home Security and Entertainment

CS499 University of Kentucky

December 2014

Personnel:

Cody Ortt

Tyler Shipp

Tyler Shrout

Disclaimer

Abstract

The Do-It-Yourself Security and Entertainment System project consists of a piece of existing code working with Ninja Blocks on a Raspberry Pi and some IP cameras and sensors. We were assigned the project for our 499 Senior Design course, working with Cirrusmio as our customer. Mike Dillion was our primary contact for the customer .Our goal was to come up with some new features and add to the extensibility of the existing system. We came up with two: Doorwatcher and ProximityPoll.

Doorwatcher uses a feed from an IP camera and compiles a 10 second .gif whenever someone rings the doorbell or opens the door, tripping one of the Ninja blocks sensors. The gif is then emailed to the company address.

ProximityPoll uses iBeacon to determine when someone with bluetooth on a mobile device enters a preset range. It keeps a running list of the UUIDs that are currently in range (inside the building) and can be used for such purposes as employee time keeping or custom messages and actions whenever a known user enters the range.

Introduction

Cirrusmio is a small, Lexington-based development house that focuses on products for non-profit organizations. They also do a fair share of tinkering with gizmos and gadgets, as evidenced by the overturned Makerbot Thing-O-Matic, the work-in-progress robotic perplexus solver, and the thinking chair/exercise ball one might notice upon first setting foot inside. I think I saw a Google Glass prototype laying around there as well. At Cirrusmio there's a sense that taking some cool new tech toy, and trying to make it do something cooler, is always a thing worth doing. It's that sentiment that drives our project, the Do-It-Yourself Security and Entertainment System, with Cirrusmio as our customer.

The Do-It-Youself Security and Entertainment System, set up at the Cirrusmio headquarters, is a Sinatra-Ruby based piece of code that works with NinjaBlocks on a Raspberry Pi and miscellaneous hardware. In its existing form, most interactions are handled through webhooks into the Ninja Blocks controller, with the system's only feature being the Cirrusmio project ainsleytwo. ainsleytwo is an application that if CURLed data will return a media player file that can played back over a announcement system. Think of it as a prototype robo-butler. Our job, tasked by our customer, was to think of a few additions to the existing system and give it a more robust feature-set.

As discussed, the features we decided to add are: Doorwatcher and Proximitypoll

Doorwatcher uses a feed from an IP camera and compiles a 10 second .gif whenever someone rings the doorbell or opens the door, tripping one of the Ninja blocks sensors. The gif is then emailed to the company address. Like ainsleytwo, it is written in Sinatra-Ruby and listens on a Thin web server for CURLs to the port. When the /door directory is accessed the method is called to begin capturing a gif. This method is called when one of the Ninja Blocks sensors is triggered, causing an event.

ProximityPoll uses iBeacon to determine when someone with Bluetooth on a mobile device enters a preset range. It keeps a running list of the UUIDs (Unique User Identification) that are currently in range (inside the building) and can be used for such purposes as employee time keeping, or custom messages /actions whenever a known user enters the range. Like ainsleytwo, it is written in Sinatra-Ruby and listens on a Thin web server for CURLs to port. When the /poll directory is accessed the method is called to begin capturing a gif. This method is called when a new device UUID enters the prescribed bluetooth proximity range.

The reason for adding these features is that they provide some additional functionality to the system already in place, and potentially could add to the “wow” or “cool”-factor when someone enters the Cirrusmio facility, helping to make a good impression. The features could also be modified to increase actual security and employee-tracking functionality, but we’ll discuss other possible future improvements later on in the report.

Product Specification

- Use of web-hooks via Ninja Blocks site.
- Access to webhooks via Ninja Blocks controller
- Sinatra-Ruby App listening on Thin web server

Customer requirements:

- Come up with 2 additional features that make use of the existing setup.
- Those features are doorwatcher and proximitypoll.
- For proximitypoll:
    - Detection of Residents/Visitors via Bluetooth
    - Relies on iBeacon technology
- For doorwatcher:
    - A .gif capture of the doorway when opened to show who entered, sent to the user
    - A .gif capture of who is at the door when the doorbell is rung, sent to the user

Product Planning

As for product planning for our applications long term applicability, they use fairly generic and industrial standard utilities to maintain portability. They operate by using a lightweight code system that doesn't require future optimizations and use SMTP and CURL to handle inputs and outputs. Due to the widespread use of SMTP and CURL our product will be able to function in the current market for a very long time. Although we designed it to work with the Raspberry Pi and Ninja Blocks, it can really work with any system that can send or receive CURL messages or emails.

Effort and size estimation

We estimated sizing based on what we thought would become three modules. In the end we merged two of the modules as their solutions and implementations were identical. The doorbell and doorway capture were merged. As such we can say our initial estimate was 1700 lines of code for what would become Door Watcher and 1000 lines of code for what would become Proximity Poll.

Our actual number of lines of code for Door Watcher was 377 and 409 for Proximity Poll. This total was gotten by using the command *git ls-files |xargs wc -l* in GitShell which produced the total number of lines in each project. We then subtracted the length of the files that shouldn't be included.

Our estimates were off by about 3.5 times the actual values. This is primarily because none of us had worked with Ruby on Rails or Sinatra and we were not aware of how large scale of tasks you could accomplish with such concise code.

Scheduling and Milestones

Course Schedule	
Date	**Event**
Wed Aug 27	**Introduction, syllabus, assignment 1**
Fri Aug 29	**Software engineering/projects**
Mon Sep 1	**Holiday (Labor Day) no class**
Wed Sep 3	**Software engineering/projects**
Fri Sep 5	**Emily Dotson, Estudio, customer interaction**
Mon Sep 8	**Ilka Balk, College of Engineering Student Affairs Director employment search, resumes**
Wed Sep 10	**Customer presentations of their projects**
Fri Sep 12	**Carmen Calabro, Performance Technologies Group**

Mon Sep 15	**Projects assigned, teams formed**
Wed Sep 17	**Teams meet with customers**
Fri Sep 19	**Emily Dotson, Estudio, web page design**
Mon Sep 22	**Brent Seales, CS department chair, Google**
Wed Sep 24	**Tony Elam, Director of Strategic Initiatives, UK College of Engineering**
Fri Sep 26	**Derek Lane, College of Communication and Information, team projects**
Mon Sep 29	**Review of deliverables, web page**
Wed Oct 1	**Jeff Cole, HP/Extreme**
Fri Oct 3	**Ethics** **Project web pages available with requirements**
Mon Oct 6	**Dave Archer, Triton Corporation**
Wed Oct 8	**Project design lecture**
Fri Oct 10	**Project design lecture**
Mon Oct 13	**No regular classes October 13, 15, 17** **review midterm oral presentations with Estudio, schedule an appointment (Mon-Thurs 10AM-6PM): http://www.engr.uky.edu/estudio/** **Status meeting with instructor during class period (with each team, only have to attend class for your meeting)** **Monday October 13 status meeting with teams: 1, 2, 3**
Wed Oct 15	**Wednesday October 15 status meeting with teams: 4, 5, 7**
Fri Oct 17	**Friday October 17 status meeting with teams: 11, 12**
Mon Oct 20	**Design due on project web page** **team midterm presentations to class: teams 1, 2**
Wed Oct 22	**team midterm presentations to class: teams 3, 4**
Fri Oct 24	**team midterm presentations to class: teams 5, 7**
Mon Oct 27	**team midterm presentations to class: teams 11, 12**
Wed Oct 29	**Team review with instructor presentation/design**
Fri Oct 31	**Emily Dotson on technical writing**
Mon Nov 3	**Javaid Siddiqi, Perceptive Software (Lexmark)**

Wed Nov 5	**Testing lecture**
Fri Nov 7	**Testing lecture**
Mon Nov 10	**Jeff Chartos, Galmont Consulting**
Wed Nov 12	**No regular class, testing review with instructor, teams 1,2, 3**
Fri Nov 14	**No regular class, testing review with instructor, teams 4, 5, 7**
Mon Nov 17	**No regular class, testing review with instructor, teams 11, 12**
Wed Nov 19	**No regular class, code review and project status with instructor, teams 1,2, 3**
Fri Nov 21	**No regular class, code review and project status with instructor, teams 4, 5 ,7**
Mon Nov 24	**No regular class, code review and project status with instructor, teams 11, 12**
Mon Dec 1	**Project presentation practice in Marksbury Auditorium teams 2, 5, 1**
Wed Dec 3	**Project presentation practice in Marksbury Auditorium teams 3, 4, 12**
Fri Dec 5	**Project presentation practice in Marksbury Auditorium teams 7, 11**
Mon Dec 8	**Project presentations in Marksbury Auditorium teams 2, 5, 1**
Wed Dec 10	**Project presentations in Marksbury Auditorium teams 3, 4, 12**
Fri Dec 12	**Project presentations in Marksbury Auditorium teams 7, 11**

Milestones	
Date	**Event**
Mon Sep 15	**Team 5 Assigned**
Wed Sep 17	**Meeting with Secondary Contact**
Wed Sep 24	**Meeting with Secondary Contact**
Fri Oct 3	**Developed Google Sites webpage with requirements**
Wed Oct 15	**Status Meeting**
Thurs Oct 16	**EStudio Appointment**
Mon Oct 20	**Design posted on webpage**
Thu Oct 23	**Meeting with Customer Success**
Fri Oct 24	**Midterm Presentation**
Fri Nov 14	**Testing Review**
Wed Nov 19	**Meeting with customer success + Gitter + significant progress in teams understanding of assignment**
Fri Nov 21	**Code Review**
Thu Nov 25	**Meeting with customer, work environment now provided**
Sun Nov 30	**Fleshed out working skeletons for Door Watcher and Proximity Poll**
Mon Dec 1	**Project presentation practice**
Thu Dec 4	**Pushed out Door Watcher (To ready for customer review)**
Mon Dec 8	**Project Presentation**
Fri Dec 12	**Pushed out Proximity Poll (To ready for customer review)**

Platforms, tools, languages

The program is written in Sinatrarb (Ruby) and uses Foreman to load the thin server via a procfile. Much of our features are based on the original code for ainsleytwo. Much like ainsleytwo, both of our features are hosted on the GitHub, which is useful for version control. Testing and running of the code was performed on an Ubuntu Amazon EC2 virtual machine, since the actual Raspberry Pi hardware was not forthcoming (a problem we'll discuss later in the Implementation section). The ruby gem dependencies are:

```
gem 'sinatra', '~> 1.4.4'

gem 'thin', '~> 1.6.2'

gem 'i18n', '~> 0.6.9'

gem 'activesupport', '~> 4.0.4'

gem 'rake', '~> 10.2.2'

gem 'foreman', '~> 0.75.0'

gem 'haml', '~> 4.0.5'

gem 'foreman-export-initscript', '~> 0.0.1'

gem 'mail', '~> 2.5.4'

gem 'rmagick', '~> 2.13.4'
```

Most of the code was written in notepad++ or using pico in the linux environment on the VMs.

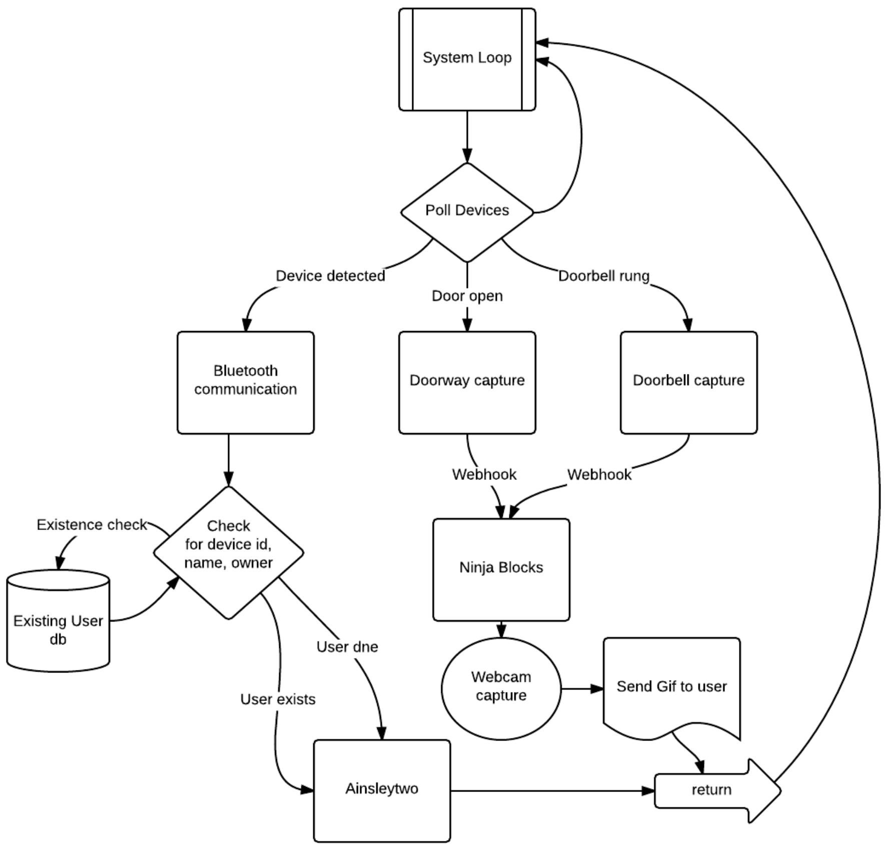
Design

Apps are written in Ruby to run on our DIY system which will use Ninja Blocks and Raspberry Pi's to communicate and send data to a webhook Ninja Blocks site. Two apps have been specifically chosen and others have been listed as possible additional features. The two apps that have been chosen are the Bluetooth communication app (proximitypoll), and the doorway capture app (doorwatcher) .

Environment

- Deamonized Thin web server
- Foreman to export init script
- Running on Raspberry Pi (Simulated via a VM during vevelopment)
- Ninja Blocks Controller
- Written in Ruby, with dependency on Sinatra Web app library
- Interacts with existing ainsleytwo code including: app.rb, say.rb,

Module Descriptions and Data Flow



Module 1: Bluetooth communication app

The purpose of this app is to collect information from a user's Bluetooth device (smart phones, tablets, etc) and use that information to track who is entering or leaving the premises. We will use an existing system, iBeacon, to collect this information based on physical proximity. The app itself will only decide what to do with that information.

The app will take the information (a data string containing a UUID, time stamp and key) and check that against a database of known users in the form of a data file. If the user exists we can enact commands that curl data to the other apps (doorwatcher or ainsleytwo), such as have a specialized greeting or record a gif of the door when they arrive.

Pseudocode for Module 1:

```
while system online:
    poll for devices
    if device found:
        check for device id
        check for device name
        check for device owner
        if exists in db:
            perform user-specified action
```

Module 2: Doorway Capture (doorwatcher)

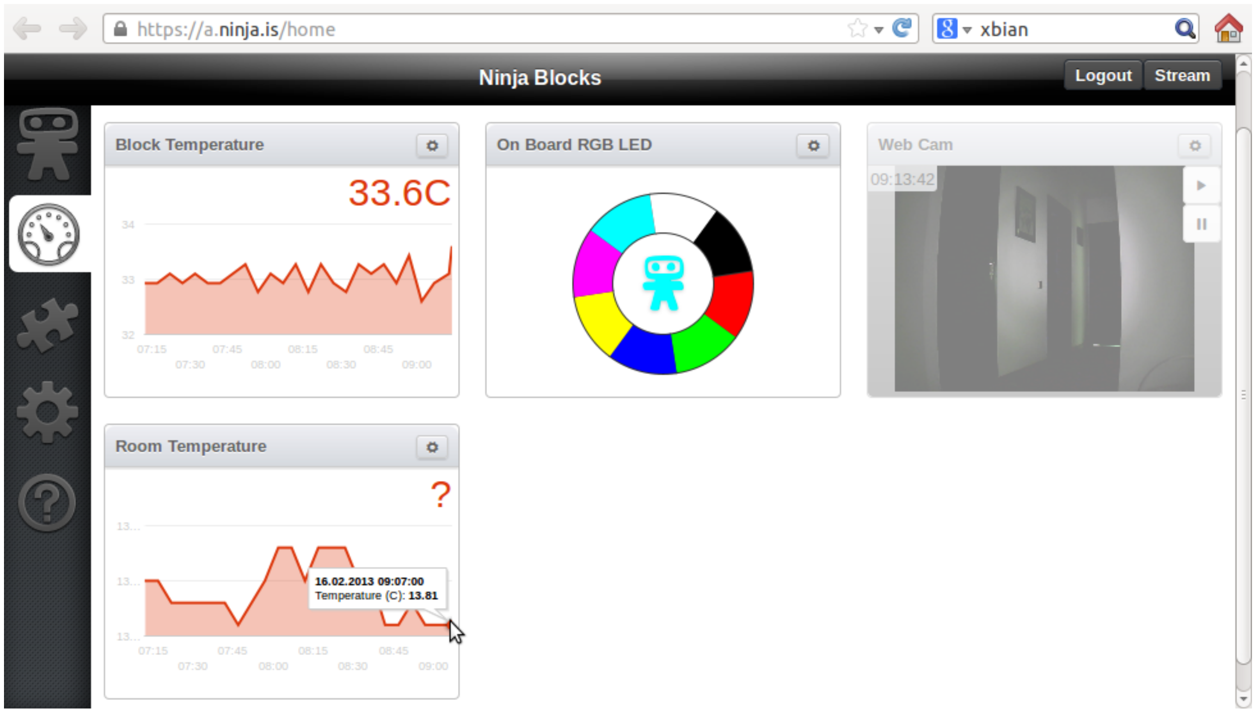
This module will take a .gif capture from a webcam when the doorway is opened and send it to the user. This will allow a user to view who actually enters their home even when they are not there. This will be done by using the Ninja Blocks door sensor and doorbell sensor which will transmit a message to the Ninja Block controller, then call a doorwatcher method via webhooks. The /door method will then compile a gif from the IP cam and send it to a specified email address.

Pseudocode for Module 2:

```
While system online
    Check sensors' status
        If status of doorway is open or doorbell rang
             (RF Sent to Ninja Block from door sensor)
            Send status to Ninja Block site webhook
            Ninja Block site call doorwatcher method via webhook
            Send .gif to user
```

User Screens

There are no user screens for our apps since they reside on a port and wait for something else to call them. For doorwatcher, you can look at the webhook rules and the camera feed on the Ninja Blocks dashboard, shown below:



User Scenarios (Use Cases)

1. When a person enters the proximity range of the Bluetooth beacon, the app will check if they are a pre-existing user. When the owner of the home enters his home, he hears his own personal theme music played over a speaker system. When a new visitor, let's say his name is John, enters they will hear the speaker system say "Welcome, John" or "John is at the door" if the app can cull their name from the smart device data.
2. When the owner is at work, he sees an email notification that someone has opened the door at home. The email includes a gif of the person entering the home.

Implementation

Problems During Implementation:

Over the course of the project we had to overcome several issues that presented themselves early on. Firstly, our primary contact for our customer, Mike Dillion, was not present for the first two scheduled meetings, so we instead met with other Cirrusmio employees who tried to fill in as proxy contacts. The problem with this is that we came away with some initial requirements that led us in the wrong direction. We thought that we were supposed to redesign the underlying system, making it more modular based on IFTTT design philosophies. We didn't know that the Ninja blocks controller would handle that for us. so we spent the first month researching modular designs and trying to decide on a coding language to best fit that requirement.

It wasn't until about a month into the process that we actually met with Mike and got a clear direction for the design. We were to design our apps to interact with the existing system, so we could use the existing ainsleytwo code as a base. Once we had that it was up to us to decide which new features would be added.

Limitations:

We also encountered problems when trying to set up a test environment. We could write code but we had no way to run it or test it without access to a Raspberry Pi. At one point we asked for one of the Ninja Blocks-provisioned Raspberry Pis, and even though the customer agreed to provide us with one it never materialized. The solution for this was to set up a Virtual Machine that would simulate a linux environment on a Raspberry Pi. Mike created an Amazon EC2 instance that we could access from powershell via an ssh command. Using that machine, we cloned our git repositories and were able to compile and run our code in the virtual environment.

Because of this limitation we were unable to test the physical calls coming from the Ninja Blocks or iBeacon, but as long as the function calls work similarly to ainsleytwo then we can assume that the webhooks coming from the Ninja Blocks will be able to access them in the same way. Mike was okay with this and said not to worry about it. This also had an effect on our original test plans because it meant we'd have to simulate the trigger pull that would call the webhooks to our apps.

Testing Plan and Results:

Many of our initial test cases were written under the assumption that the systems would be in place for physical testing. As such we had to revise our test plans to focus more on unit testing under the virtual machine constraints. Here is an example of a unit test plan for each app:

proximitypoll

1. Add fake user data to userlist.txt
2. Pass the data into the polling method
3. Check for say() output based on data
4. Clear user data from temp database
5. Check that your user data is added correctly
6. modify the data except primary keys (restricted)
7. wait 1 minute, call the polling method again
8. Verify that your modified user data was not reset

Result: Pass

Doorbell Sensor

1. send call to /door method, noting the exact time
2. check that video is recorded and stored in temp folder
3. check that correct length of video (10s) is sent to email
4. verify video quality
5. verify time signature lines up with time doorbell was rang

Result: Pass

For function testing, we only ended up having two main functions, poll and door:

/poll		
Input	Expected	Result
UUID found in datalist	Perform action in user preferences	Pass
UUID not found in datalist	Sends email to default address. Notifying that an unrecognized UUID was scanned.	Pass
Gets bad UUID data	Sends email to default address. Notifying that an unrecognized UUID was scanned.	Pass
Gets malformed data	split operation fails to execute, will get an index error.	Pass, the format of the inbound data should always be the same.

/door		
Input	Expected	Result
called with data	Ignores data. gif generated and emailed.	Pass
called without data	gif generated and emailed.	Pass
called twice in less than 10 seconds	calls will queue, second gif will be delayed by the second remaining from the first call. No gif truncation.	Pass

Future Enhancements/Maintenance

There are really a limitless number of future enhancements that could be made for Proximity Poll. DoorWatcher is kind of limited in that it does what it needs to do and it doesn't need to do anything else. Due to the fact that Proximity Poll can call any custom webhooks specified in the user data, you can hook it up to any system that supports webhooks. This would allow integration with other systems, both new and existing. Proximity Poll is also configured to allow matching of variable names to variable values from user list files and although we only took advantage of this to implement custom emails, you could use it to store any variables you wish in the user list file.

The technical understanding requirement was an issue raised by users in product testing. Manipulating the files to configure a different settings is unforgiving of mistakes and is not always very informative about what went wrong.

Maintenance of the system should be easy to keep up with since the Ruby Gemfile specifies which versions it requires to run. This way future updates to those libraries will not affect our system. The only real maintenance that will be required is maintaining the user list file and the email accounts associated with the SMTP. There may be some maintenance associated with the hardware being implemented into the system but the code itself shouldn’t require much if any maintenance.

Conclusions

We learned a lot about the importance of communication and keeping up with deadlines. The worst thing you can do when communication fails is give up. Both parties have to keep attempting to make communication work until it does. If we could do this project differently we would have wanted to meet our customer sooner as it would have given us time to expand into many more additional features. Our team is talented and great at pushing things through but we also had busy schedules and after losing a lot of our time it was hard to recover from it. Our customer is thrilled with what our product does but he said he wanted us to conform to a different coding style even though we already wrote the whole project in one. We haven’t done our final product demonstration to our customer as we are supposed to meet him when this paper is due. We will have to reserve any further comments about the issue until then.

References

Cirrusmio, Inc, et al, ainsleytwo, (2014), GitHub repository, https://github.com/CirrusMio/ainsleytwo

Maiza, Moncef, rmagick, (2009), GitHub repository, https://github.com/rmagick/rmagick

Mizerany, Blake, sinatra, (2012), Documentation site: http://www.sinatrarb.com/

User Manuals and Installation Guides

The READMEs are displayed on the respective github repositories as they are depicted in the following pages.

doorwatcher

Running Server

Deamonized Thin

```
sudo thin -a 10.0.1.101 -p 80 -R config.ru -d --pid thin.pid start
```

Foreman

Development

```
foreman start -f Procfile.dev
```

Production

Foreman can export an init script via the following: bluepill, inittab, runit, upstart

For more info on Foreman export formats see: http://ddollar.github.io/foreman/#EXPORT-FORMATS

Getting Started

Have Door Watcher record a .gif from the webcam and send it to the specified e-mail.

```
curl http://127.0.0.1:80/door
```

Raspberry Pi setup

```
sudo apt-get update
```

Install git:

```
sudo apt-get install git
```

Install ruby:

```
sudo apt-get install ruby #=> ruby 1.9.3, ruby 2 would be better
```

```
sudo apt-get install ruby-dev #=> provides required libraries
```

Install bundler:

```
sudo gem install bundler
```

Clone DoorWatcher:

```
git clone https://github.com/CirrusMio/doorwatcher.git
```

Bundle gems:

```
cd /path/to/doorwatcher && sudo bundle install
```

How to Use

Once you have everything installed, there are two things you want to configure or at least take note of inside handlers/door_action.

1. Edit line 22 to contain the URL of your still frame captures from your camera.
2. Line 53-59 for email input and 67-72 for email output.

Explanation of Process

Capture frames -> Add frames to .gif -> attach .gif to email -> send email

Fun ideas!

In theory you can use any trigger to curl to doorwatcher to get a .gif. You could use it to captures .gifs of anything you want with a webcam. No need for a door!

(Much borrowed from AinsleyTwo)

Proximitypoll

Contributing

Running Server

Deamonized Thin

```
sudo thin -a 10.0.1.101 -p 80 -R config.ru -d --pid thin.pid start
```

Foreman

Development

```
foreman start -f Procfile.dev
```

Production

Foreman can export an init script via the following: bluepill, inittab, runit, upstart

For more info on Foreman export formats see: http://ddollar.github.io/foreman/#EXPORT-FORMATS

Getting Started

Have Proximity Poll record data and call other webhooks from userlist file.

```
curl --data "msg=uuid@DDMMYYYY@key" http://127.0.0.1:80/poll
```

Raspberry Pi setup

```
sudo apt-get update
```

Install git:

```
sudo apt-get install git
```

Install ruby:

```
sudo apt-get install ruby #=> ruby 1.9.3, ruby 2 would be better
```

```
sudo apt-get install ruby-dev #=> provides required libraries
```

Install bundler:

```
sudo gem install bundler
```

Clone ProximityPoll:

```
git clone https://github.com/CirrusMio/proximitypoll.git
```

Bundle gems:

```
cd /path/to/proximitypoll && sudo bundle install
```

How to Use

Once you have everything installed, there are four things you want to configure or at least take note of.

1. Edit / Record the IP/Port Proximity Poll will be listening on. The Procfile.dev will have "-p WXYZ" where WXYZ is the port it will listen on.
2. The default action handler. In handlers/in_proximity.rb you will want to set lines116 and 117 to your preferences. By default it will send an email to itself containing the unknown UUID that triggered it.

3. In the same file as the one above look for lines 127-142. These lines contain the sender smtp information. You may customize these if you wish.
4. The userlist.txt file contains custom e-mail notification settings as well as calls to other devices using curl.

userlist.txt

This file is so important it needs its own section! There are samples in the file in the repo as well. There are a few conventions to keep in mind while using this file. A user is separated by:

```
[{uuid}
...
settings
...
]
```

Any comments or notes may be left outside of the [] and it will not affect the ability to read the file.

Variables within a block of user information are formed as follows:

```
@variableName:variableValue
```

You may specify a custom curl message when the user's UUID is detected by:

```
>--data "messages or data goes here" http://localhost:4565/door
```

The above is a sample that will trigger our related Door Watcher app when the UUID is read in.

A full sample:

```
[{10101}
 >http://localhost:4565/door
 @name:GuyMan
 @email:guysboss@gmail.com
 >--data "words=Hello+Guy" http://localhost:4567/say
 ]
```

This sample would call Door Watcher when GuyMan (AKA: 10101) enters the range and then send an
email to guysboss@gmail.com and greet him by calling AinsleyTwo.

Fun ideas!

Use this to track employees, children, hotel residents and more!

Use it for custom greetings!

With iBeacon distance ranges you can have it even change the music for each room you enter.

Custom entertainment settings for each user.

(Much borrowed from AinsleyTwo)