The Neighbor

A Subscription Based Equestrian monitoring and Treat Dispensing Service

Gavin Swiger

Valencia College

CIS 2910C IT Capstone

****

Table of Contents

[Objective 4](#_Toc45881541)

[Business and Profitability Synopsis 4](#_Toc45881542)

[Expected Project Timeline 5](#_Toc45881543)

[Schematic 6](#_Toc45881544)

[Mechanical Operations 7](#_Toc45881545)

[Components Break Down 8](#_Toc45881546)

[The Raspberry Pi 3 Model B /w Power Supply 8](#_Toc45881547)

[Raspberry Pi Camera Module V2-8 Megapixel,1080p 8](#_Toc45881548)

[Class 10 Micro SD Card 8](#_Toc45881549)

[SunFounder 9g Metal Servo 8](#_Toc45881550)

[Components Break Down Cont. 9](#_Toc45881551)

[Alfa Long-Rang WiFi Network Adapter 9](#_Toc45881552)

[Power Supply Extension cord 9](#_Toc45881553)

[Servo Wiring Diagram 10](#_Toc45881554)

[On Site Setup Guide: 11](#_Toc45881555)

[Current Limitations 12](#_Toc45881556)

[The Neighbor (On Startup): 13](#_Toc45881557)

[Scalability 14](#_Toc45881558)

[Code Walkthrough 15](#_Toc45881559)

[Feed.py 15](#_Toc45881560)

[Webstreaming.py 18](#_Toc45881561)

[SocketServer Current Limitations and Planned Authentication Update 21](#_Toc45881562)

[Basic network diagram. 23](#_Toc45881563)

[Security Benefits of the update 24](#_Toc45881564)

[One Time Use Link. 24](#_Toc45881565)

# Objective

The objective is to offer a third-party service which allows equestrians to monitor and interact with the horses they own or are currently leasing.

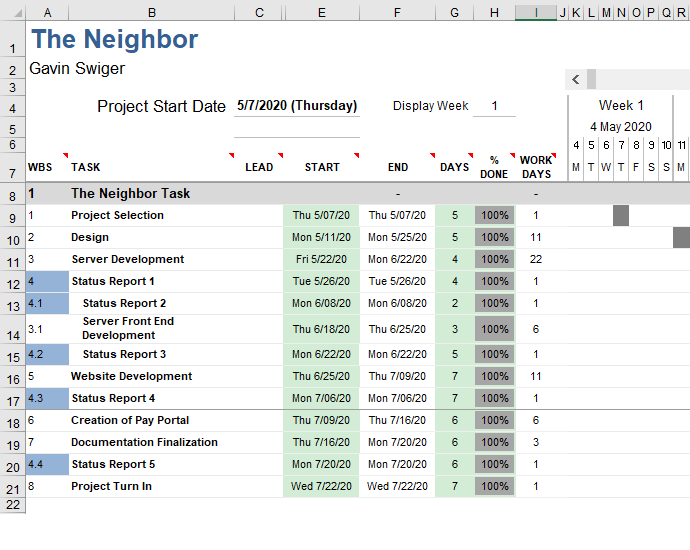
# Business and Profitability Synopsis

With the current model the goal is to not sell the units but lease them at a set price to the owners of equestrian centers for 6 to 12-month intervals. The target price is $19.99 a month. While the price of boarding a horse varies by location the cost of $19.99 would make up a tiny fraction of the total cost of board yet act as a major draw for potential clients[[1]](#footnote-1).

As it stands the current estimated cost of producing and installing a unit is roughly $190 dollars (Parts, Labor, and Installation). So, the first typical contract would pay for the unit. First year profitability would come from the secondary function of the Neighbor unit. This secondary function is the ability to feed owner provided treats on demand from anywhere for a small price. This price which will range from $.99-$2.99. The cost will in part be determined by what possible for the pay portal which has yet to be determined.

# Expected Project Timeline

Full excel sheet is located in the accompanying file.



# Schematic

# Mechanical Operations

* The raspberrypi.org power supply does all the heavy lift and is fixed in place and attached to a 25ft 14 AWG outdoor rated extension cord.
* There are 4 stainless steel Cable Tie Mounts acting as the mounting brackets. Heavy duty zip ties (Up to ½ inch in width) are fed through them. This was done to allow for easy customer remounting in the case of the unit falling. This also allowing for a large variety of mounting scenarios.
* The hopper is loaded up with roughly 2 pounds of grain (preferably oat or flaxseed based without molasses) with a grain size of no more than .75inches by .25 inches. (This accommodates most horse grain but excludes treats where a single piece is the serving size.)
* The Sloped hopper reduces the weight on the thruway disk allowing it to be moved using less power.
* When the raspberry pi signals the servo motor the thruway disk (located just above the servo) is rotated roughly 1/16th of a rotation back and forth. This is done three times.
* The grain closest to the thruway disk grain to fall through a slot in the disk and down through the chute below.

# Components Break Down

## The Raspberry Pi 3 Model B /w Power Supply

* Cost per unit $52
* Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
* 1GB RAM
* CSI camera port for connecting a Raspberry Pi camera
* Micro USB power source handles up to 2.5A
* Raspian OS (https://www.raspberrypi.org/downloads/raspberry-pi-os/)

## Raspberry Pi Camera Module V2-8 Megapixel,1080p

* Cost per unit $22
* Maximum of 1080P30 and 8MP Stills in Raspberry Pi Board
* 2A Power Supply Highly Recommended
* Fixed Focus Lens

## Class 10 Micro SD Card

* Cost per unit ~$6
* 32 GB

## SunFounder 9g Metal Servo

* Cost per unit $7
* Operating Voltage Range is 4.8V~6.0V
* Operating travel: 180°±5°

# Components Break Down Cont.

## Alfa Long-Rang WiFi Network Adapter

* Cost per unit $29.99
* USB 2.0, Mini USB, type B female connector
* **IEEE Standards** 802.11a/b/g/n

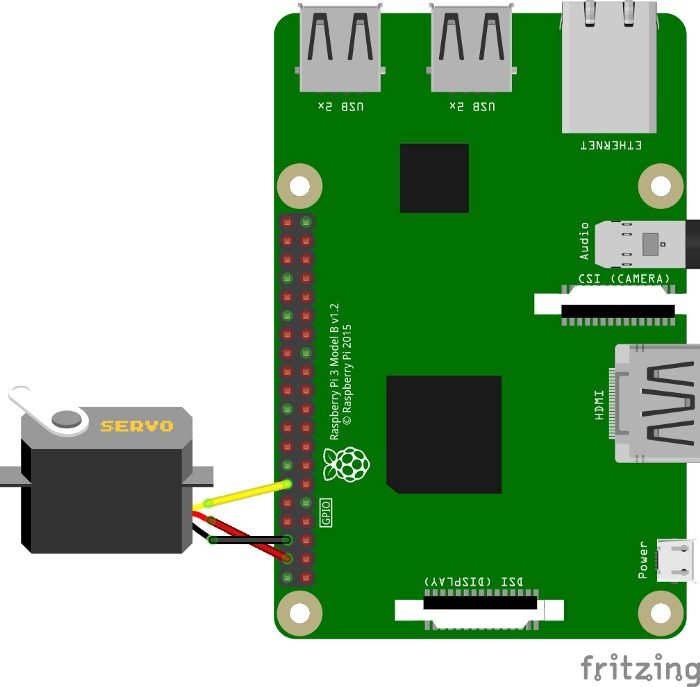
## Power Supply Extension cord

* Cost per unit $12
* 25ft long.
* 14 AWG

# Servo Wiring Diagram

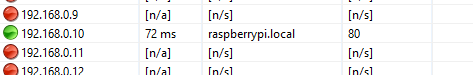
The servo is connected via 3 18inch 22-gauge wires.

The servo is power via the Pin#04, grounded through Pin#06, and signaled through Pin#12.

****

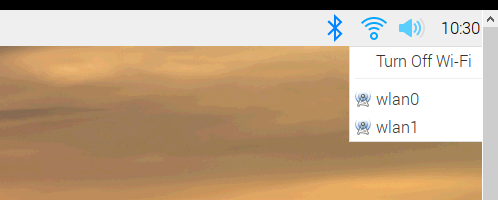
# On Site Setup Guide:

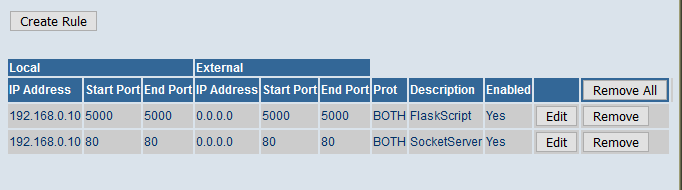
**Step 1:** Connect The Neighbor manually through the ethernet port located on the back of the device.

**Step 2:** Navigate to the router devices list and locate the local address of The Neighbor. (Alternatively use the Angry IP Scanner Tool) 

**Step 3:** Using the local address Remote Desktop into The Neighbor

**Step 4:** Enter the Wireless Credentials of the Customer’s network into the wlan1 (Alfa High Gain Long-Rang WiFi Network Adapter)



**Step 5:** Using the Port Fowarding Settings on the router (Typically labeled under “Advance”) create two rules for each of the local IP addresses corresponding with the local ip address of the neighbor unit.

**Step 6: (Pending development)** Using the customers log in information and ensure that you are able to connect to each unit.

## Current Limitations

* Customer must have accessible wifi (Although the Neighbor comes with a large antenna)
* Customer has port forwarding functionality on router.
* Each neighbor unit must have wifi credentials entered in manually (Currently).
* There is a plan to incorporate backdoor functionality later on, but none exist at this moment. Once developed there will need to be an additional rule added here.

# The Neighbor (On Startup):

Commands have been added to the end of the Raspberry Pi’s Boot Sequence to allow for the device to continue running in the event of the device reboot after initial setup aka a loss of power.

The method I used to do this was through the editing of the rc.local file located at **/etc/rc.local**

**/usr/local/bin/noip2**

This sends local and external ip address to the noIP service.

**sudo python3 /home/pi/Desktop/webstreaming.py &**

Using python 3 this launches the webstreaming python script.

**export FLASK\_APP=/home/pi/Desktop/flaskfeedserver.py &**

Assigns the flask application to the default flask web application variable

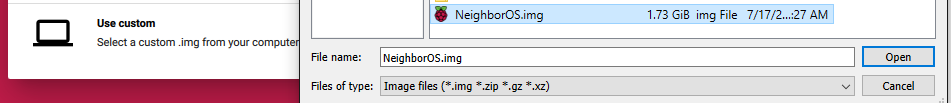
**flask run --host=0.0.0.0 &**

Launches Flask at whatever the specific Neighbor’s local address is.

# Scalability

The ability to have multiple units at a single location is a critical aspect of the neighbor design, the availability of parts is also important hence all parts ultimately being able to be purchased online at a moments notice. Since the physical unit is a matter of assembly, I am going to provide a brief walk through of setting up the nieghbor OS in it’s current state.

**1.** Using the Raspberry Pi Imager v1.2 selected “Use Custom”

**2.** Select the NeighborOS file.

**3.** Choose the SD card designated for use in the next neighbor Unit.

**4.** Press “Write” and wait for the image to finish writing to the SD card.

**5.** Insert the SD card into the neighbor unit and power it on.

**6.** Open the terminal and use the command “sudo noip2 -C”

**7.** Enter the log in credentials associated with the account which is currently gavinswiger@yahoo.com and my password.

**8.** When you get to the part of the wizard that allows you to enter the hostname enter the corresponding hostname. It will be something like “neighbor002.ddns.net”

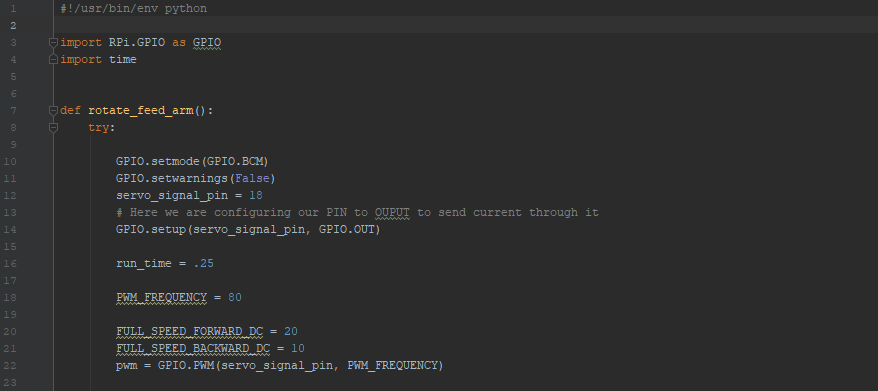
**9.** Once complete reboot the Raspberry Pi and open the terminal once again.

**10.** Enter command “sudo noip2 -S” to insure that configuration information is inputted correctly.

# Code Walkthrough

I will attempt to keep this section as succinct as possible. I will also not mention what a section of code does if I have already covered it from a different file. For a development environment I used JetBrain’s PyCharm Community Edition. I chose this IDE simply because I had used it before with game development.

## Feed.py

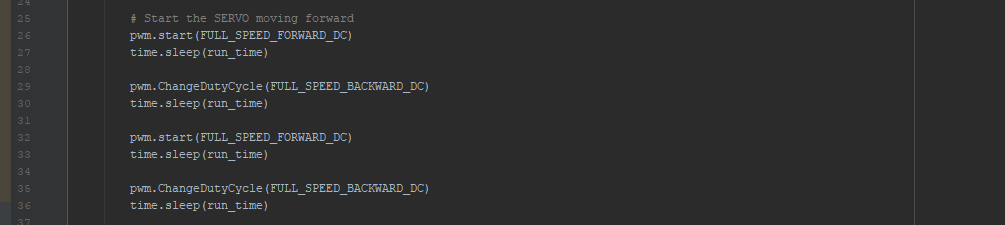
****

At the top there is Shebang pointing to the Rpi’s python interpreter. This is following by importing the GPIO and Time library. This is followed by the rotate\_Feed\_arm() method. At the top of the method we have some setup required to use the GPIO pins.

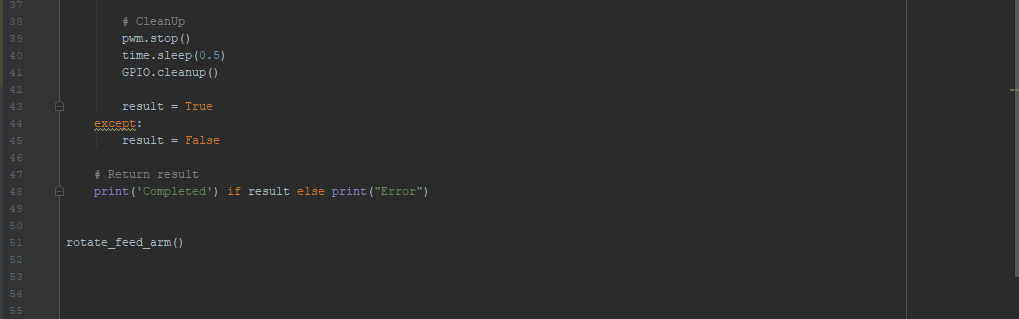
I designated pin 18 as the signal pin this can be seen via the wiring diagram. The PWM\_FREQUENCY carriable (Pulse Width Modulation) operates effectively at any number between 80-100.

The Variable “run\_time” is currently at .25 but will be edited later on after more testing with a completed assembled system. As it stands this run time turns a 4inch diameter disk via the servo motor the desired amount.

The following defined “SPEED” s were values are provided by sunFounder who makes the servo motor.



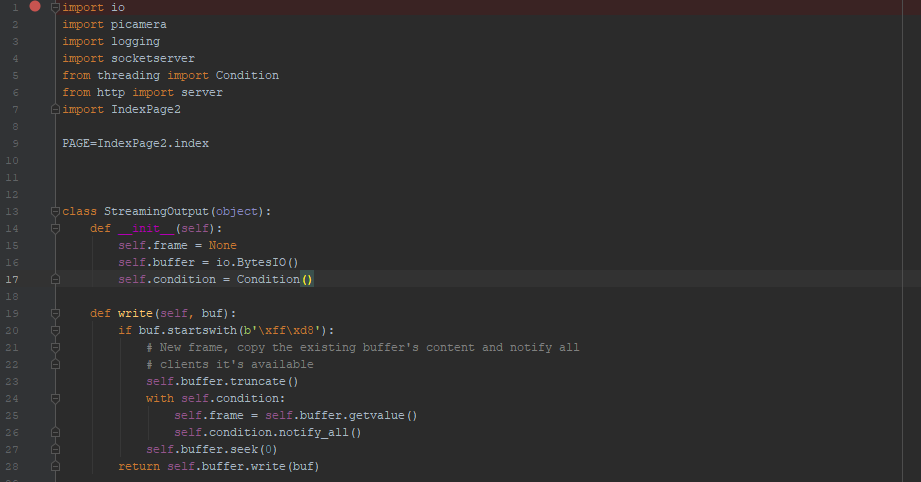
Lines 25-36 are telling the Rpi to move the motor at the defined speed forward, backward, forward and backward again. Doing a short shimmy allows for the flow of the treat grain to be more easily stopped. The amount of times the disk must move forward and backward may be changed in the future depending on testing after full assembly.



Lines 37-50 are just cleanup and some bits of code for testing. The rotate\_feed\_arm() call has been commented out as the function is actually called by the flask server script.

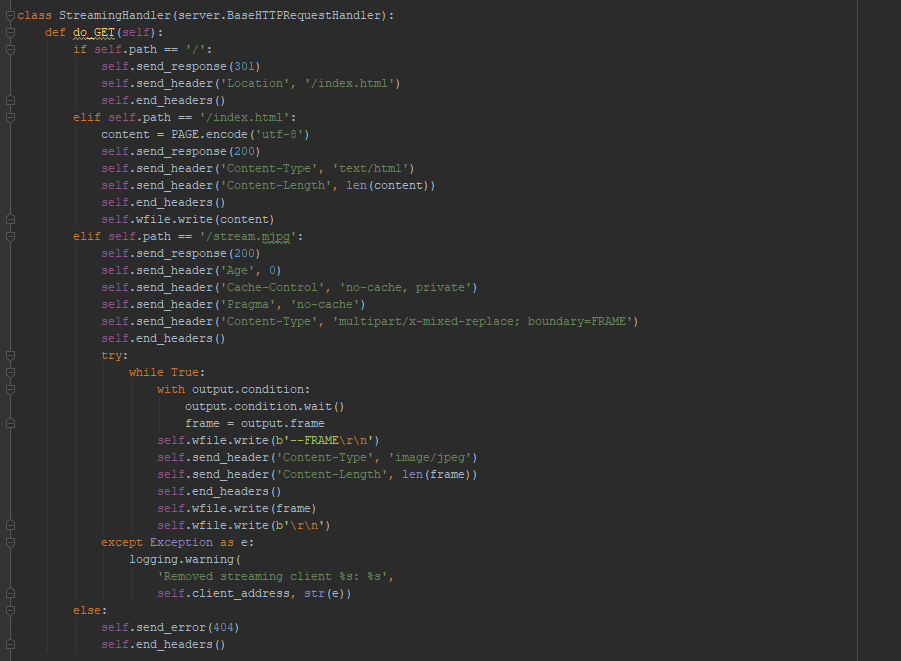
## Webstreaming.py

The current script used to stream video through the camera module is an altered version on the one located at <https://picamera.readthedocs.io/en/release-1.13/recipes2.html>. At this moment it is appropriately licensed and attributed.



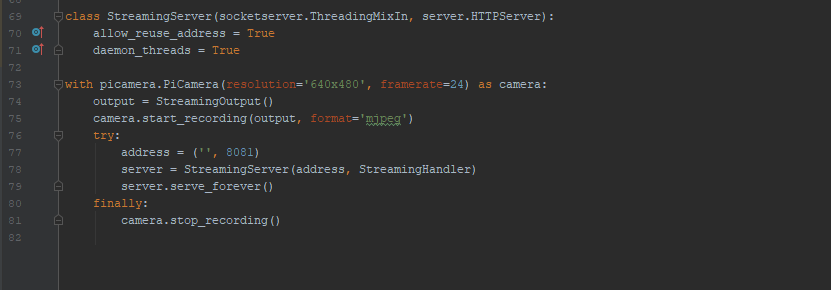
After imports there is an variable referenced called “index” from the IndexPage2.py file. The index variable holds the html and style code used to dress up the video stream. It will be referenced later in this document.

The StreamingOutput Object allows the users computer to access the video stream data continuously and in an order which makes sense for the User. There is currently a bug which may be a result of this bit of code as the live stream can be requested and duplicated indefinitely. I plan on working to fix this.



The StreamingHandler class determines what happens when the user sends an HTTP request (Effectively when they access the address.). It starts by pointing the users browser to address/index then encodes the PAGE String type variable into a standardized UTF-8 encoded which can be read and interpreted by browsers. Following this it generates the /stream.mjpg path than attempts to continuously check for a new image than updates the image being displayed at /stream.mjpg. These images are kept in order thanks to the StreamingOutput class previously created.

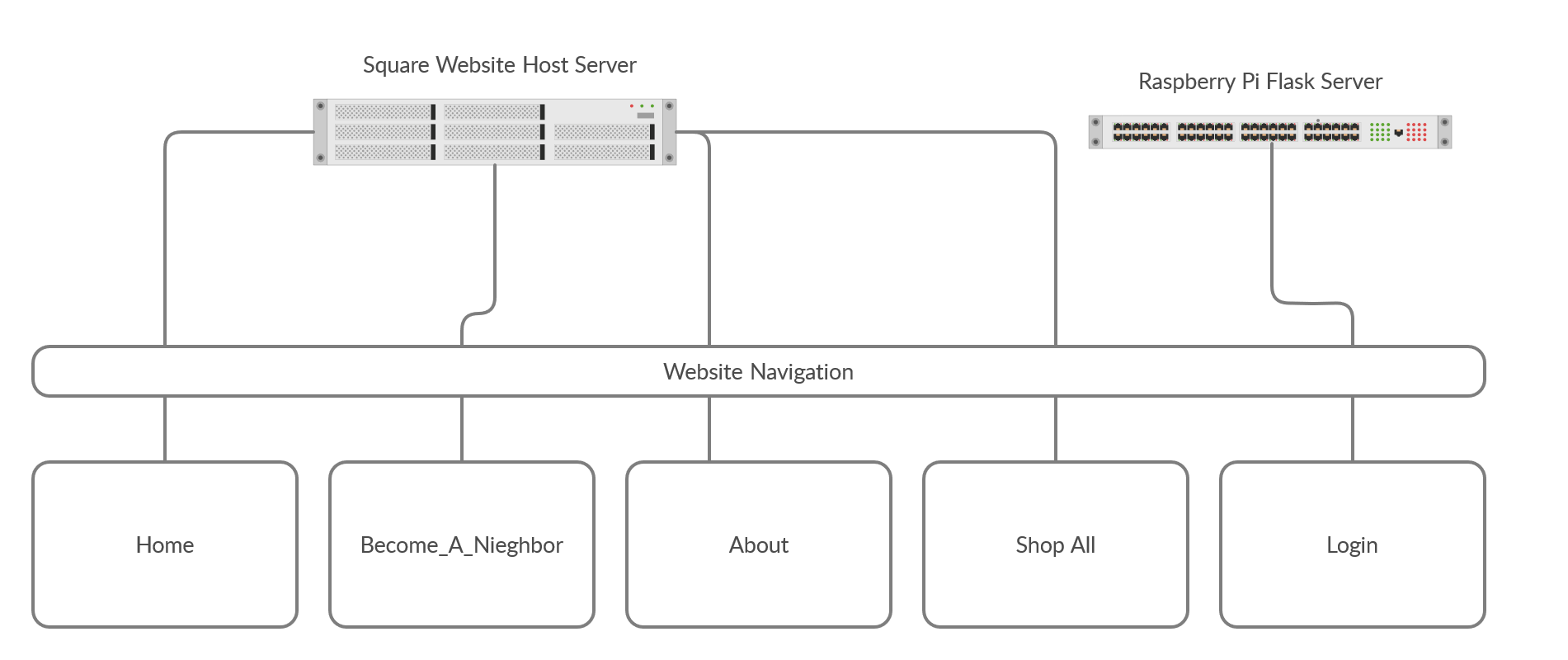
There is currently logging functionality when running the server through an IDE on the Rpi but I plan on outputting the logs to a file which is accessible offsite at a later date.



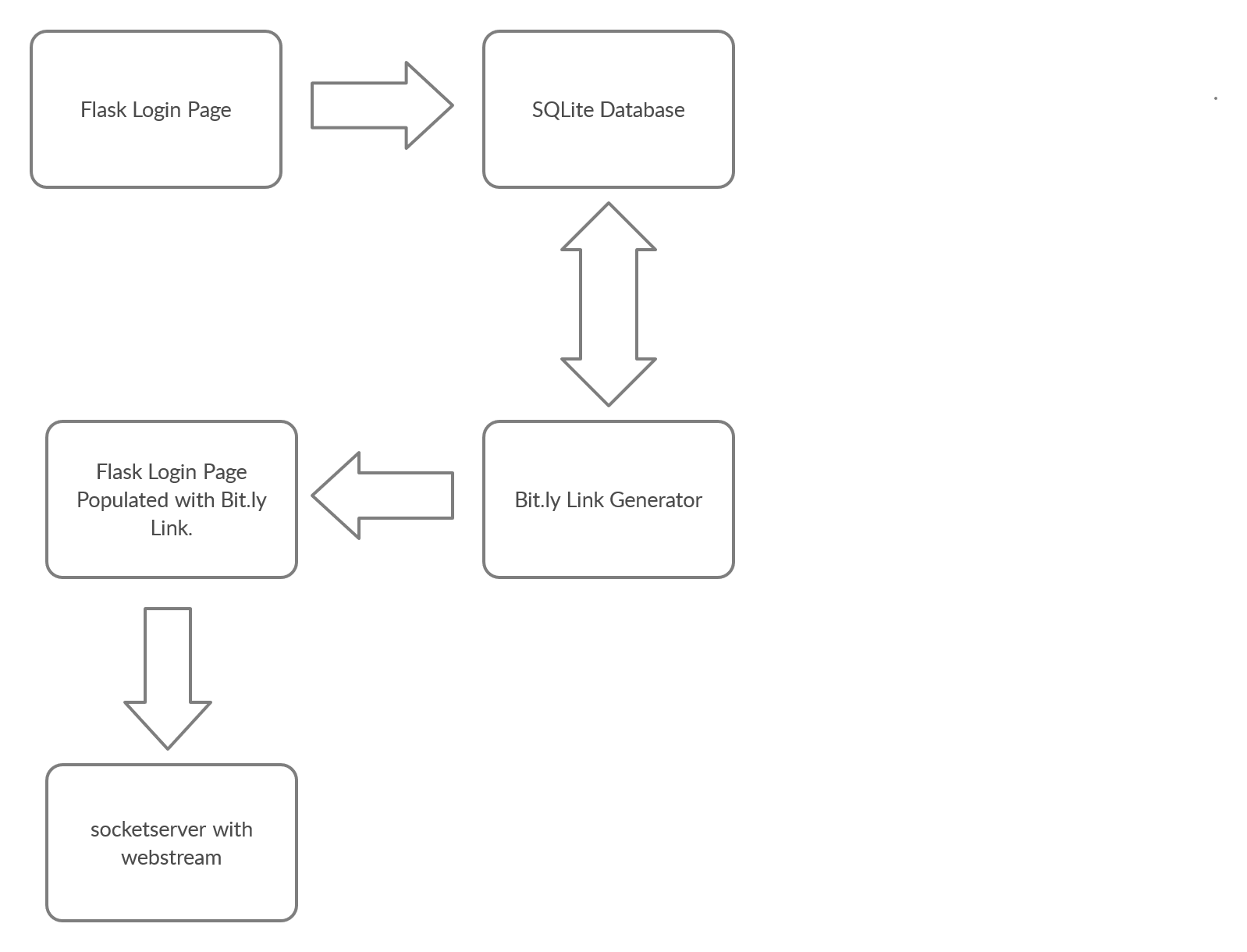
The StreamingServer Class according to the documentation is what allows for the server to be terminated even if all the created threads have not exited. [[2]](#footnote-2) Next, we have the actual camera module being referenced with a number of alterable settings. At this point in the code I could rotate the video feed if that ends up being need when the project is fully assembled. \***This has since been updated to port 80.**

# SocketServer Current Limitations and Planned Authentication Update

Currently a major security concern is the lack of ability to authenticate the user due to how the python library socketserver presents the html code / webserver. Socketserver at the moment loads the entire html as a string variable which makes calling local scripts or data bases from the webpage virtually impossible. After consulting with a friend who works in the web development space, I have drafted a solution and plan on implementing this ASAP. The solution is as follows.



This is currently a very similar to the current navigation, the only difference is Login currently will land you onto the socketserver located at neighbor001.ddns.net:80. [[3]](#footnote-3) The Proposed fix will instead land you onto a flask server at the above address.



At the new login page, the user will enter their login credentials (provided by the barn they board at.). The new flask login page located at neighbor001.ddns.net:80 will validate the login information via an SQLite Database. Once validated the flask server will take the live stream URL which matches the given user’s login information and use it to generate a onetime use Bit.ly link. Once the Bit.ly link is generated the page will then redirect the user to the address which is the address to the socketserver. The socketserver address will be the page that is displaying the live stream.

## Basic network diagram.

User navigate to <https://www.theneighborflorida.com/>. From this point they select “Login” from the nav bar. The page brings them to the flask server which then validates their credentials, from there they are directed to the URL which holds the page hosting the live stream. This page is physically hosted from the Neighbor unit onsite connected to the internet through barn’s wireless router.

## Security Benefits of the update

This design provides two potentially 3 layers of security, one being basic login authentication, the second being the complete masking on the address which holds the livestream. The third will be determined by the functionality of flask and the Bitly API, the third is it being a onetime use link.

## One Time Use Link.

I believe there are two ways of currently doing this, both have yet to be fully explored. Ideally the Bit.ly API will offer a tool which generates the link as a “one-time link”. The other slightly more complicated method would be when the user attempts to connect to the socketserver the socketserver will then be able to read the URL. From this point it would compare it against URLs previously used on the SQLite database. If it had been used before, reject the connection.

1. ***The Economic Aspects of a Small Equine Boarding Operation in North Florida*** *https://ufdcimages.uflib.ufl.edu/IR/00/00/19/82/00001/FE42800.pdf* [↑](#footnote-ref-1)
2. https://docs.python.org/2/library/socketserver.html [↑](#footnote-ref-2)
3. *This is a pseudo login page as anyone could essentially “view source” and see the username and password and the link is to the prototype livestream is currently protected.)* [↑](#footnote-ref-3)