Lab 6 – User Interface – Garage Door

Online Links

This lab is available on my personal website at: http://AaronNelson95.com/IT441Lab6.php
The code is also available on my GitHub repository at https://github.com/AaronNelson95/IT441

Objective

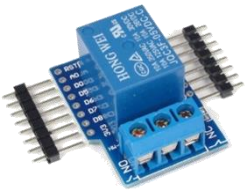
The lab will help one learn how to use an Arduino relay shield to interface with existing electronic hardware (such as a garage door opener button). When it is triggered, it will be able to control electricity flow to that external device. This relay can be activated with a push of a cell phone button or using a voice assistant, by using IFTTT and Adafruit IO services. The last time the relay was triggered can be seen in Home Assistant, and additional automations can be setup. This lab will help one learn:

- How to use a relay shield to control electricity flow
- How to connect Arduino to an online MQTT server such as Adafruit IO
- How to use two MQTT servers (one online and one local) in an Arduino project
- How to use IFTTT to push data to an Adafruit IO feed which is read by an Arduino device
- How to log data and perform other automations with Home Assistant.

Materials

To complete this lab, you will need the following materials:

- The three Arduino devices made in Lab 4
- The Raspberry Pi running Home Assistant set up in Lab 5
- A computer with the Arduino IDE
- A Wemos D1 mini microcontroller
- A Micro USB power cord
- A Breadboard
- A Wemos Relay Shield
- 3 Male-Male Jumper Wires (unless the soldering allows the shield to be placed on top of the board)



A Wemos Relay Shield

References

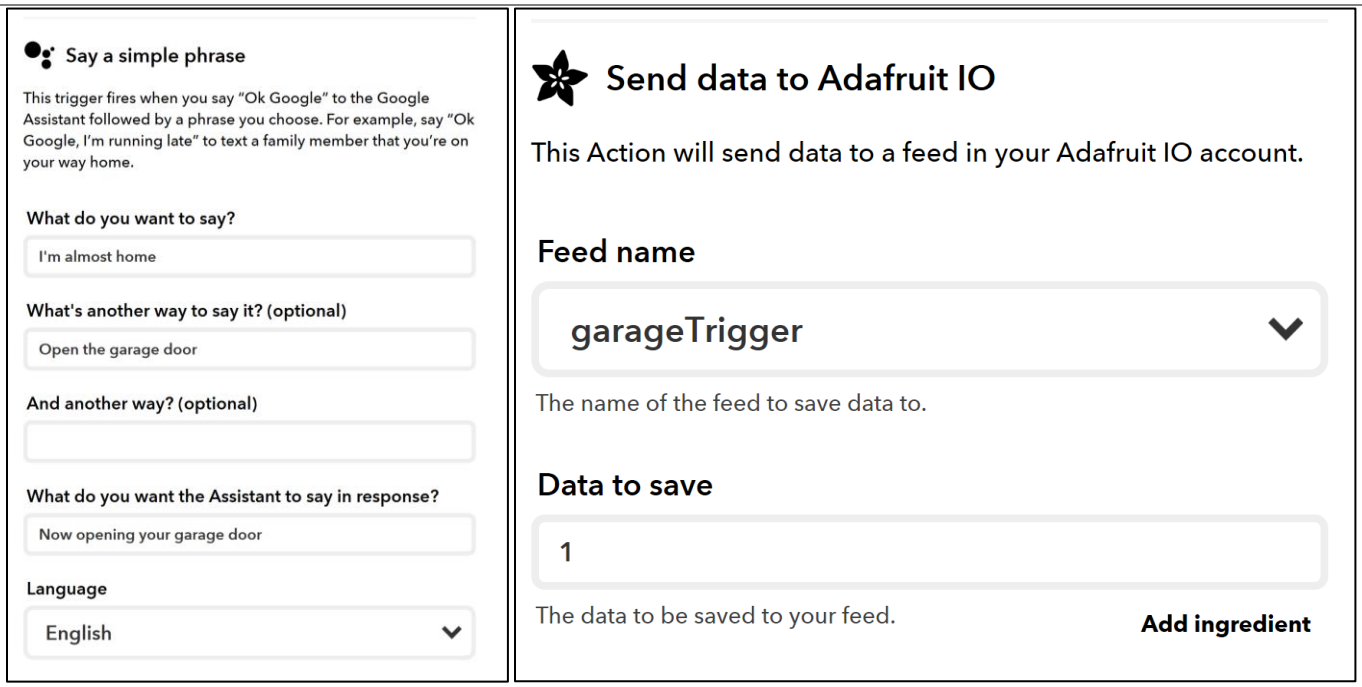
The following links may be helpful throughout this lab:

- http://aaronnelson95.com/IT441Lab4.php - Lab 4 instructions on how to set up the Mosquitto MQTT broker on a computer and how to develop 3 Arduino devices to communicate with MQTT
- http://aaronnelson95.com/IT441Lab5.php - Lab 5 instructions on how to set up Home Assistant on a Raspberry Pi and how to connect Wemos devices as sensors
- https://hackaday.io/project/4027-arduino-openhab-garage-door-control - An advanced tutorial on using a relay to control a garage door. It is more advanced than we use, but it helps to understand how a relay works
- https://www.instructables.com/id/Arduino-WiFi-Garage-Door-Opener/ - A simply written tutorial on actually connecting the relay to the garage door button (not done in this lab, but can be a future project)
- https://github.com/petehoffswell/garagedoor/blob/master/garagedoor/garagedoor.ino - Example code for a relay. Used to understand how the relay pulses to trigger the garage
- https://community.home-assistant.io/t/how-to-get-last-state-changed-from-switch/125656 - Template to show the last time a sensor was activated with Home Assistant
- https://ifttt.com/ - Used to combine services that will post a value to the Adafruit feed to trigger our relay

- https://io.adafruit.com/ - An online MQTT Broker that can work with IFTTT and Arduino devices
- https://arduinodiy.wordpress.com/2017/11/24/mqtt-for-beginners/ - Using Arduino with a local MQTT broker (such as Mosquitto)
- Arduino Example (after adding the Adafruit MQTT Library) at File -> Examples -> Adafruit MQTT Library -> mqtt_esp8266_callback – Used to have an Arduino device subscribe to an Adafruit IO feed.

Procedures

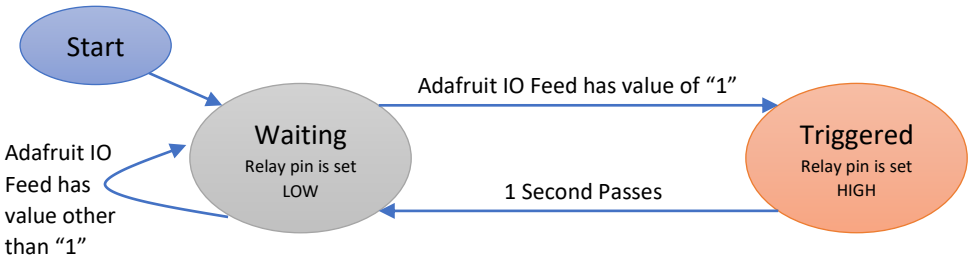
1. Setup the Door Opener, Echo Sensor, and Stoplight devices from Lab 4. In addition, you should have a Mosquitto MQTT server running locally. Verify that these devices work together and that they publish data to MQTT topics. The relay device will be used to also post information into the topic "garage/buttonRelay". Its values will change between "Triggered" and "Waiting".
2. Setup and configure Home Assistant with Hassbian from Lab 5. Sensors should show the status of Arduino devices. The IFTTT Webhook key should already be configured with Home Assistant through this lab. If it has not been, follow steps 8-10 in the Lab 5 report for an understanding of how automations work and how to connect different web services together.
3. Up to this point, each device only works locally. Home Assistant is able to *trigger* IFTTT applets through a webhook, but local devices cannot be triggered *by* IFTTT. Although Home Assistant provides this interactivity with IFTTT, it requires Hassbian to run on a public IP address. This may not always be feasible, so instead, an online MQTT broker (such as Adafruit IO) can be used to "send" information directly to an Arduino device. IFTTT can then publish information to the web-based Adafruit IO directly, and this can be read in by the relay device. Unfortunately, Home Assistant is also only allowed to run one MQTT connection, so in order to continue supporting and working with our previous devices (which ran with the local Mosquitto MQTT), our relay that is subscribed to the Adafruit IO feed should then post the feed results to the local Mosquitto server. Once in the local server, this can be setup as a sensor in Home Assistant. Basically, Adafruit IO acts as a web-based "middle-man" to connect IFTTT services to trigger an Arduino device, and Mosquitto acts as a local "middle-man" to connect the Arduino device to Home Assistant and the various automations there. Examine the "System Diagram" in the Appendix to understand how each device and service works together.
4. Setup a connection between IFTTT and Adafruit IO
    a. Create an account (if you haven't done so already) at https://io.adafruit.com. Once signed in, click the "IO" tab on the top of the page, then select the "Feeds" tab and click "view all"
    b. Click "Actions" and "Create a new Feed". This is the "topic" that the relay will subscribe to. Once created, view that feed. Notice how the URL to this topic goes "<your username>/feeds/<topic name>". Later in the Arduino code, you will need to specify both of these parameters to access your information.
    c. Click the AIO Key tab on the top right of the page. Copy the "Active Key" and save it for later. This will be used to give permission for your Arduino device to access the Adafruit IO feed.
    d. Sign into IFTTT and go to https://ifttt.com/adafruit. Connect your Adafruit service to your IFTTT account. Now you can create and experiment with various IFTTT triggers to post a value to your Adafruit IO feed. For increased security, you can come up with a specific value that would cause the relay to be activated (and all others will be ignored). The "That" portion of your applet will work with Adafruit IO and send data to the feed name (given in step 4b) and the activating value you would like to send (such as a simple "1").
        i. You can trigger this same action with multiple, separate triggers, for example, you can use an IFTTT button (which is activated by a phone widget) to do this action.
        ii. You can also link IFTTT with Google Assistant and specify phrases you would like to say to trigger the action such as "I'm almost home" or "Open the garage door".



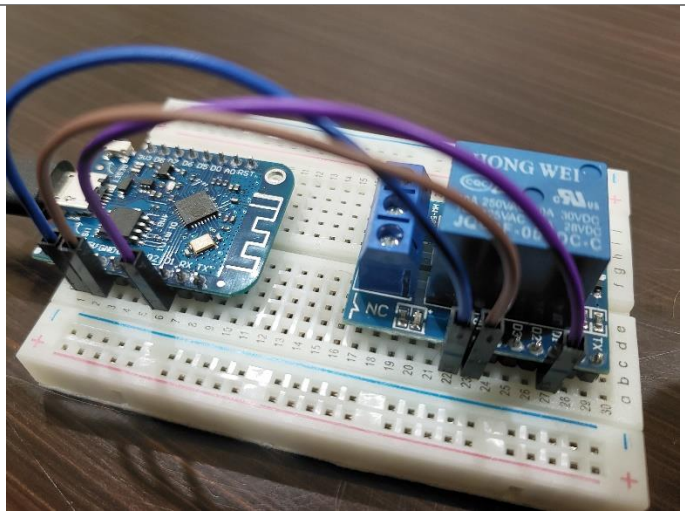
IFTTT trigger (on left) and action (on right) to send an Adafruit feed value when Google Assistant is spoken to

e. Test that your triggers work. When they are activated, you should see the table populate with the value in your Adafruit IO Feed page.

5. Develop the Arduino garage door relay device.
   a. First, design documents to help you understand the logical flow of the system.
      i. Develop a state diagram (if needed). There are two stages, the waiting stage and the triggered stage. When Adafruit IO sends the value you specify (such as "1"), move to the triggered stage, which will turn on the relay switch. Once in this stage, after one second, move to the waiting stage and turn off the relay switch. This simulates a "pulse" that "pushes" the garage door button.



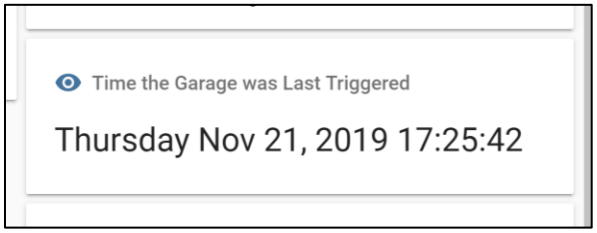
      ii. This device *only* triggers the garage door, which indirectly will trigger the magnetic reed switch. However, this device and the group of three devices from Lab 4 act independently of each other. Therefore, no change needs to be made to those devices, and their state diagram can still be found in the Lab 4 report (Procedures step 3).
      iii. This device will need to interact with both the web-based Adafruit IO MQTT broker (to subscribe to data) as well as the local Mosquitto MQTT broker (to post data). A system flowchart, like the one in the Appendix, may help with understanding how to program the Arduino board.
   b. Connect the relay shield to the Wemos board. If the board is not able to be placed directly on top (because of soldering), then connect the 5V, GND, and D1 pins. A circuit diagram of what should be connected is provided in the appendix.



The relay shield manually connected to the board using wires

c. Program the relay device to listen for Adafruit data, and post it to the local MQTT server.
   i. First, it may be helpful to test the relay pulse. This is simply done by setting pin D1 to output. Next set it to a HIGH voltage, wait 1 second (with a delay(1000)), then set it LOW again. When testing, you should hear a click sound and the relay board's light should go on and off.
   ii. Next, add the Adafruit MQTT Library and follow the example code located at File -> Examples -> Adafruit MQTT Library -> mqtt_esp8266_callback. Remove any lines that are posting to the "photocell" feed and instead modify the subscribe parts of the code to listen to your feed. Your username and AIO Key need to be defined in this code (from step 4b and 4c). Test that when you post the value in the feed, you can trigger the relay switch.
   iii. Finally, incorporate the local MQTT server by modifying the code at https://arduinodiy.wordpress.com/2017/11/24/mqtt-for-beginners/. Be careful of any places where variables may conflict with the Adafruit names (such as with the WiFiClient being named "client" and the term "MQTT" used frequently throughout). You do not need to have a callback function or subscribe to anything with Mosquitto, but you should connect to this server and then post "triggered" or "waiting" to the "garage/buttonRelay" topic only after Adafruit passes the proper value.

6. Working with Home Assistant Automations
   a. Home Assistant should already be setup from Lab 5. Following the same pattern as before, create a sensor to read data from the "garage/buttonRelay" topic. Just like that, you can now incorporate the relay device with the many features of Home Assistant. You can also design a card that shows the last time the device was triggered by using a template (https://community.home-assistant.io/t/how-to-get-last-state-changed-from-switch/125656).



Home Assistant Card shows last time the relay was triggered

   b. Just like in the past lab, Automations can be used to interact with your relay data in new ways. Here, we can use an IFTTT trigger to log the time the relay was activated into a Google Sheet. In Home Assistant, a

trigger can be made to watch a state change on the sensor.garage_button_trigger. When the value changes from "Triggered" to "Waiting", you can perform a simple action to call the ifttt.trigger service and send the service data "event: ButtonRelay". Now in IFTTT, you can create an applet that watches the "ButtonRelay" event in a web hook. You can then add a row to a spreadsheet, and this can be the same spreadsheet where the light colors are logged as well. There are many possibilities once you can link your devices with IFTTT.

7. Verify that your entire system works by triggering your device with IFTTT (such as through Google Assistant or a button). The relay should then "click" on and off. Check your Home Assistant card to see if this trigger was recognized locally and if an entry was added to your spreadsheet log. You can now use Arduino devices to trigger events in the outside world *and* to respond to outside triggers!

Thought Questions

1. What services did you consider to integrate into your project and why?

   I decided to incorporate both IFTTT and Adafruit IO into my project. It is difficult to directly send a trigger from IFTTT to perform an action on Home Assistant (such as to send a local MQTT value) without a public IP address. I saw this as a challenge that I wanted to solve. I eventually decided to use Adafruit as a "middle-man" to pass the IFTTT message along to my Arduino device. This worked just like Mosquitto, with having a topic (called a feed), but because it was hosted online, IFTTT directly worked with this service! It was as simple as selecting the feed I wanted to post to and the value of data to send. This solved my problem with sending messages to devices only connected to my local network. I also learned that Home Assistant could only work with one MQTT server- either Adafruit or Mosquitto. Rather than readjusting all of my other devices (which in hindsight sticking with Adafruit the entire time would have been beneficial), I wondered if it was possible to pass on a message through my relay device to the local Mosquitto. It worked and so I am able to use publicly-supported-only triggering services (through IFTTT) to cause actions on my local devices and private Home Assistant.

   Along with the overall web services of IFTTT and Adafruit, which I used to communicate with my device, I also utilized other services within IFTTT. I used some services to *act* on my device. Home Assistant uses automations to start an IFTTT webhook when an event occurs. I am always a fan of lists and numbers, so the first thing that came to my mind was logging the times when something happens with my devices. I added this data to a spreadsheet in the format I specified to achieve this. I also setup notifications (app and email) which would go off if I left the garage open for a long period of time. I also really enjoyed working with triggering services. Location and wifi-connection check services are always handy when building applets, but I was also able to explore some I never used before. I did an IFTTT Do Button that appears as a widget on my phone. When I click this, it can perform an action. This is super easy to control and can be done just as quickly as opening an app would be. I never used voice commands before, so I thought it would be neat to explore the Google Assistant service. With it, you can specify multiple messages to trigger the applet, and what the assistant would say in response. This would be useful in our system because you can activate the garage door completely hands free from your car by simply saying "OK Google, I'm almost home". This method worked very well and I was shocked by how fast the relay device responded to my command.

2. What services would you like to integrate in the future?

   Originally, I thought it would be great to have the garage button trigger when your location came in range of your home. However, I quickly realized how problematic this could be. You could be driving past a road on an errand and could activate the trigger even though you don't actually plan to be home yet! Your garage would then just be sitting open, unintended. I next thought that it could activate when you pick up your home

wifi, which would be a specific SSID. However, with this method also, you may be *walking* home and would cause the garage to open. Location and wifi connection would be fantastic when checking if you are home (such as to turn on your phone's ringtone), but it could cause problems or be dangerous if it opens your home to unwanted visitors. Instead, a great way I thought could work is to have IFTTT trigger if you are connected to your car's Bluetooth AND one of the previously mentioned events occurred. Unfortunately, I was disappointed to discover that IFTTT does not support multiple triggers. I believe other services, such as Tasker, does support this so I will have to look into that in the future and see if I could implement it with IFTTT. I would also like to explore other IFTTT services and see how they can be integrated. One feature I can add to my system is that when I leave the garage door open too long, it will automatically close. This will require some tweaks to my system (such as making sure nothing is blocking the door), but I can post to my Adafruit feed from IFTTT, which was activated from an Home Assistant automation (my relay is only set to listen to Adafruit values for security purposes). I also really liked playing with the voice commands service and I plan to explore how I can implement them with other applets now.

3. What was the biggest challenge you overcame in this lab?

   Some of my biggest challenges, as I mentioned, dealt with limitations of the services I was using. IFTTT can only use Home Assistant as an action if it has a public IP address, Home Assistant could only have 1 MQTT server connected normally, IFTTT cannot handle multiple triggers, etc. I saw each of these challenges as an opportunity to put the knowledge I gained in this class and other experience to the test. I tried to explore and consider other creative ways I could use tools and services to achieve my goals. Of course, I could have just changed everything to using Adafruit, or just set a public IP address, but I wanted to stretch myself and see how I could work with the system in the state it is currently in. It was fun and I enjoyed finding solutions to solving my challenges. Honestly, one of the most difficult things for me in this lab was trying to understand how the relay shield works. I had a hard time comprehending how it could operate with current electrical systems. I also had a hard time making the connection that it doesn't directly affect the other devices in this lab's system. Instead, as a result of activating the relay, the garage door may open or close (if it works correctly). Completely independent of the relay device, the magnetic reed switch (and other devices) are only activated by the garage door physically opening. Discussing the lab and relay with my teacher and other students helped me to understand what the relay should accomplish, which was my biggest holdup.

4. How is the overall system secured?

   Creating home automation devices without their system security in mind could become disastrous in a real life setting. Our previous devices worked more internally within the garage and only communicated with each other. This relay device, however, is intended to physically open your garage door. If an unintended party were able to discover how it communicates and works, they could learn how to open your garage while you are away, leading to potential theft or danger. Taking steps and precautions can help lower the risk of your devices being hacked and discovered. I purposely chose not to make Home Assistant publicly connectable on the internet. Instead, it can only be accessed within my home's wifi network. This requires someone to be near my home with knowledge of my password to be able to connect or even view Home Assistant. My relay is the only switch that has any connection to the internet outside of my local network. It connects to my private Adafruit IO feed. Even if someone knew my Adafruit username and feed location, they would need access to my account to obtain the AIO Key, without which, they can post nothing. I took an extra precaution with Adafruit as well in that my relay device only accepts a very specific value ("1" for lab testing purposes). Even if someone was able to post into my Adafruit IO feed, they would also need to know *what* to post. One of the largest potential security risks I have is my IFTTT services. If someone were to steal my phone, they would have access to my triggering button widget and my Google Assistant (which they would still need to know what to say). However, I keep my

phone on me at all times and it is protected and locked with an iris scanner and other biometrics. Keeping information as private as possible, and limiting the number of services you have connected to your home devices can help you greatly increase the security of your system.

5. Estimate the time you spent on this lab and report.

   The lab portion only took me a few hours (around 3-4). It was actually easier than I thought to come up with creative solutions to solve the challenges and limitations I faced. The report writeup took me around 8 hours.

Certification of Work

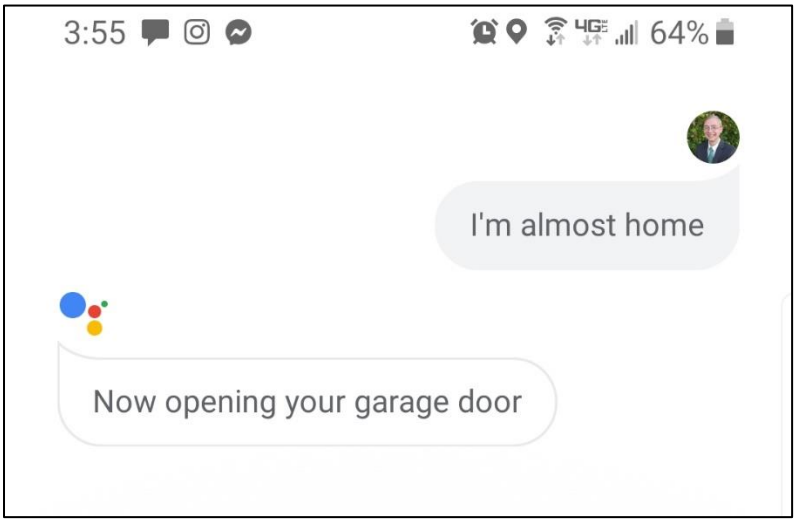
I certify that this lab represents my own work. Any configuration code that I used was basic public knowledge provided from the Home Assistant documentation. The code for the relay switch (working with Mosquitto and Adafruit IO) was built from examples provided along with their libraries. The examples I used are mentioned in the code comments.

– Aaron Nelson

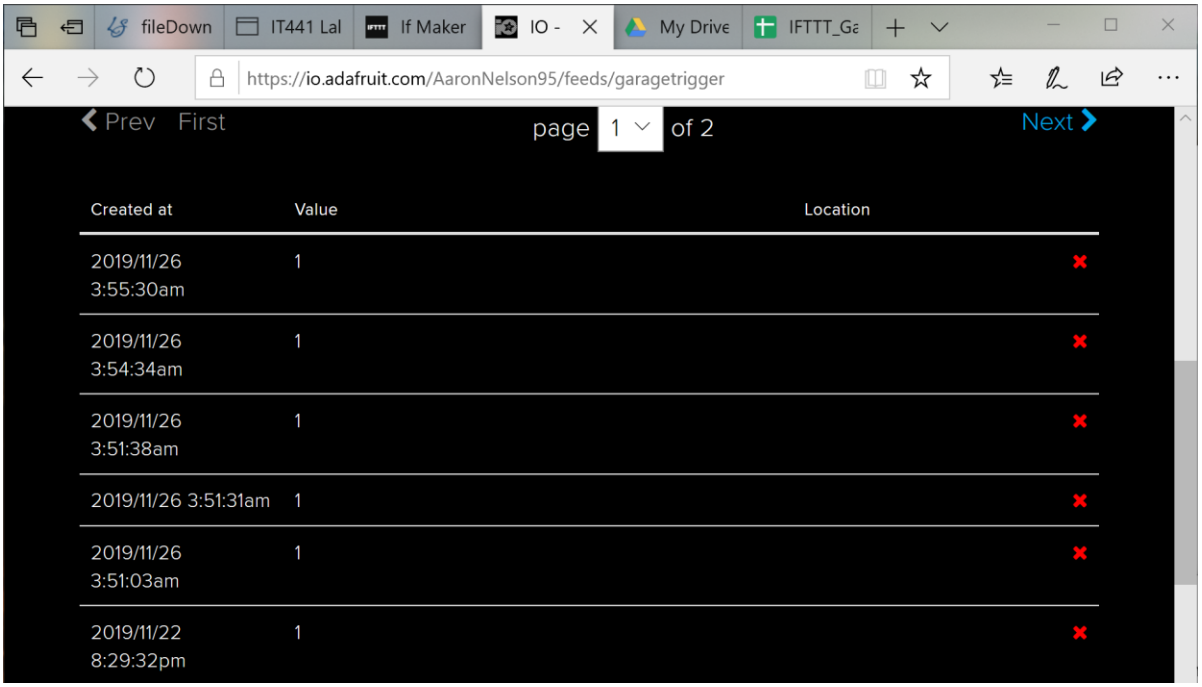
Appendix

Images of Final Product

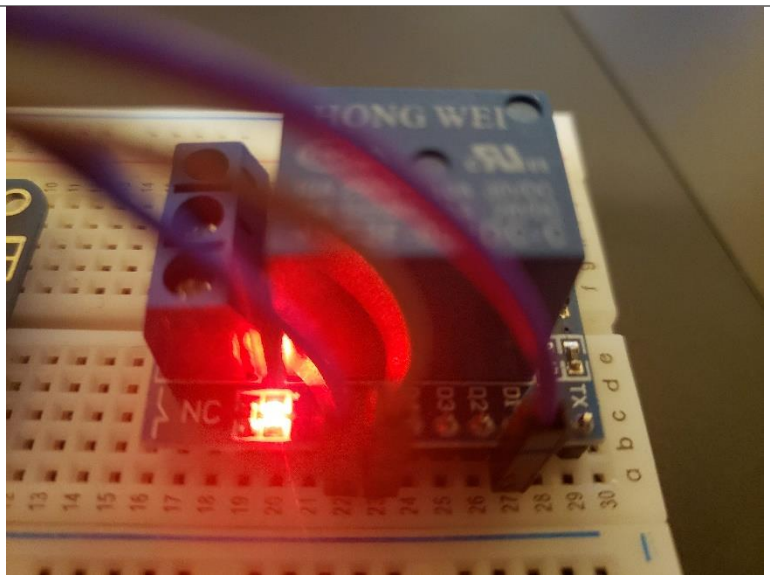
An IFTTT Button widget can be added to my phone to trigger the garage relay



In addition, I can talk to Google Assistant and this will also trigger a custom IFTTT Applet



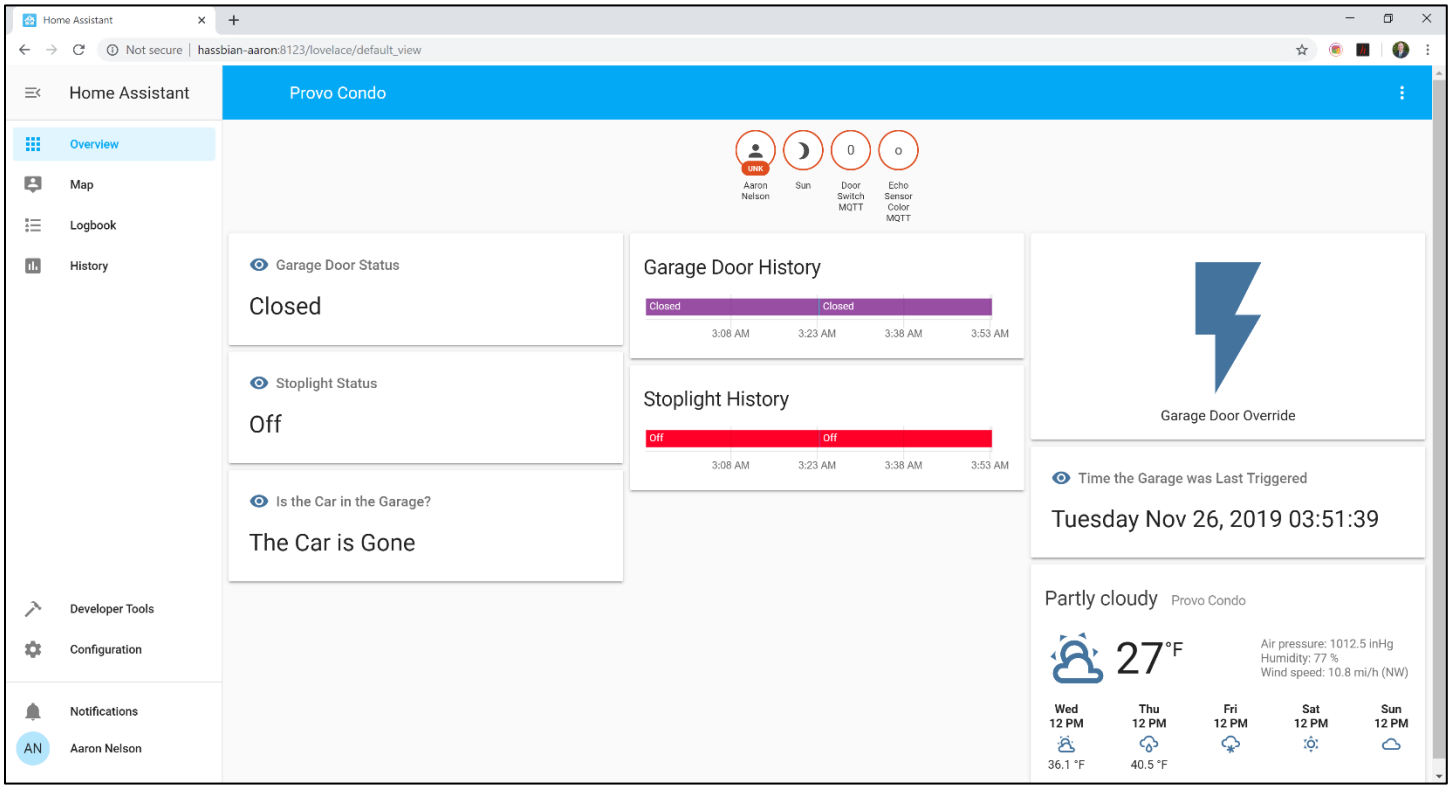
Adafruit IO posts the value I setup with IFTTT each time my applet is activated



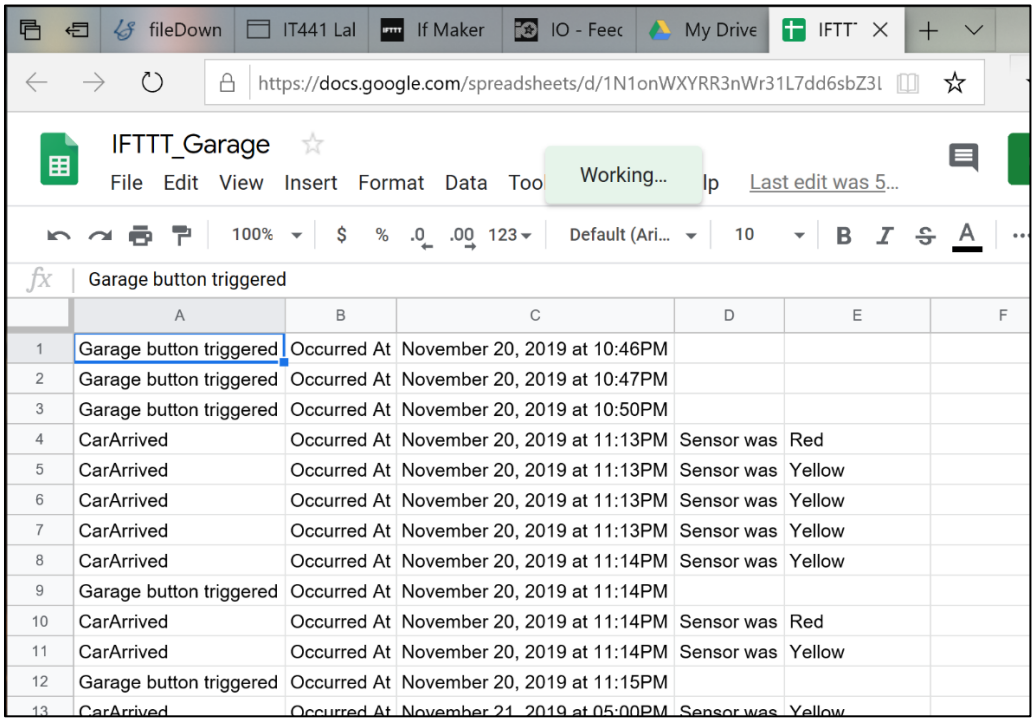
When the value of "1" is received from Adafruit, a clicking sound is made and the light turns on for one second

```
aaron@Aaron-Laptop: ~
aaron@Aaron-Laptop:~$ mosquitto_sub -i "Windows Listener" -t "#" -v
garage/buttonRelay Triggered
garage/buttonRelay Waiting
garage/buttonRelay Triggered
garage/buttonRelay Waiting
garage/buttonRelay Triggered
garage/buttonRelay Waiting
```

The Mosquitto Server is still properly setup from Lab 4. It now listens to the relay locally and shows when it is triggered



Home Assistant now displays a card that interacts with the relay switch. It will show the time the device was last activated



IFTTT_Garage

File Edit View Insert Format Data Too Working... lp Last edit was 5...

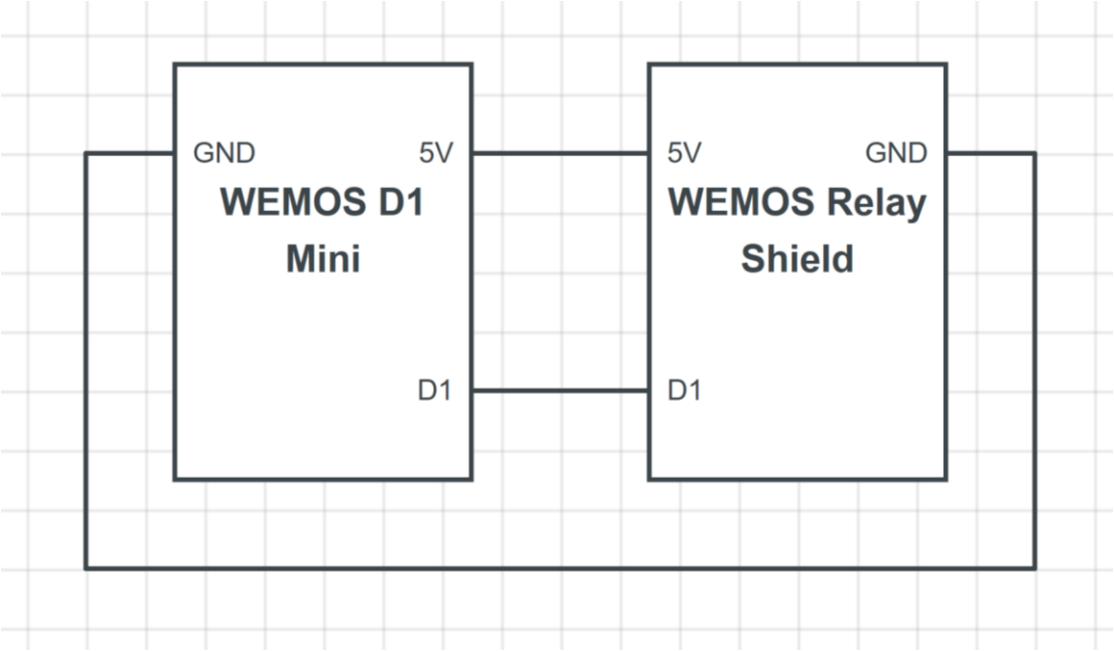
100% $ % .0 .00 123 Default (Ari... 10

fx Garage button triggered

	A	B	C	D	E	F
1	Garage button triggered	Occurred At	November 20, 2019 at 10:46PM			
2	Garage button triggered	Occurred At	November 20, 2019 at 10:47PM			
3	Garage button triggered	Occurred At	November 20, 2019 at 10:50PM			
4	CarArrived	Occurred At	November 20, 2019 at 11:13PM	Sensor was	Red	
5	CarArrived	Occurred At	November 20, 2019 at 11:13PM	Sensor was	Yellow	
6	CarArrived	Occurred At	November 20, 2019 at 11:13PM	Sensor was	Yellow	
7	CarArrived	Occurred At	November 20, 2019 at 11:13PM	Sensor was	Yellow	
8	CarArrived	Occurred At	November 20, 2019 at 11:14PM	Sensor was	Yellow	
9	Garage button triggered	Occurred At	November 20, 2019 at 11:14PM			
10	CarArrived	Occurred At	November 20, 2019 at 11:14PM	Sensor was	Red	
11	CarArrived	Occurred At	November 20, 2019 at 11:14PM	Sensor was	Yellow	
12	Garage button triggered	Occurred At	November 20, 2019 at 11:15PM			
13	CarArrived	Occurred At	November 21, 2019 at 05:00PM	Sensor was	Yellow	

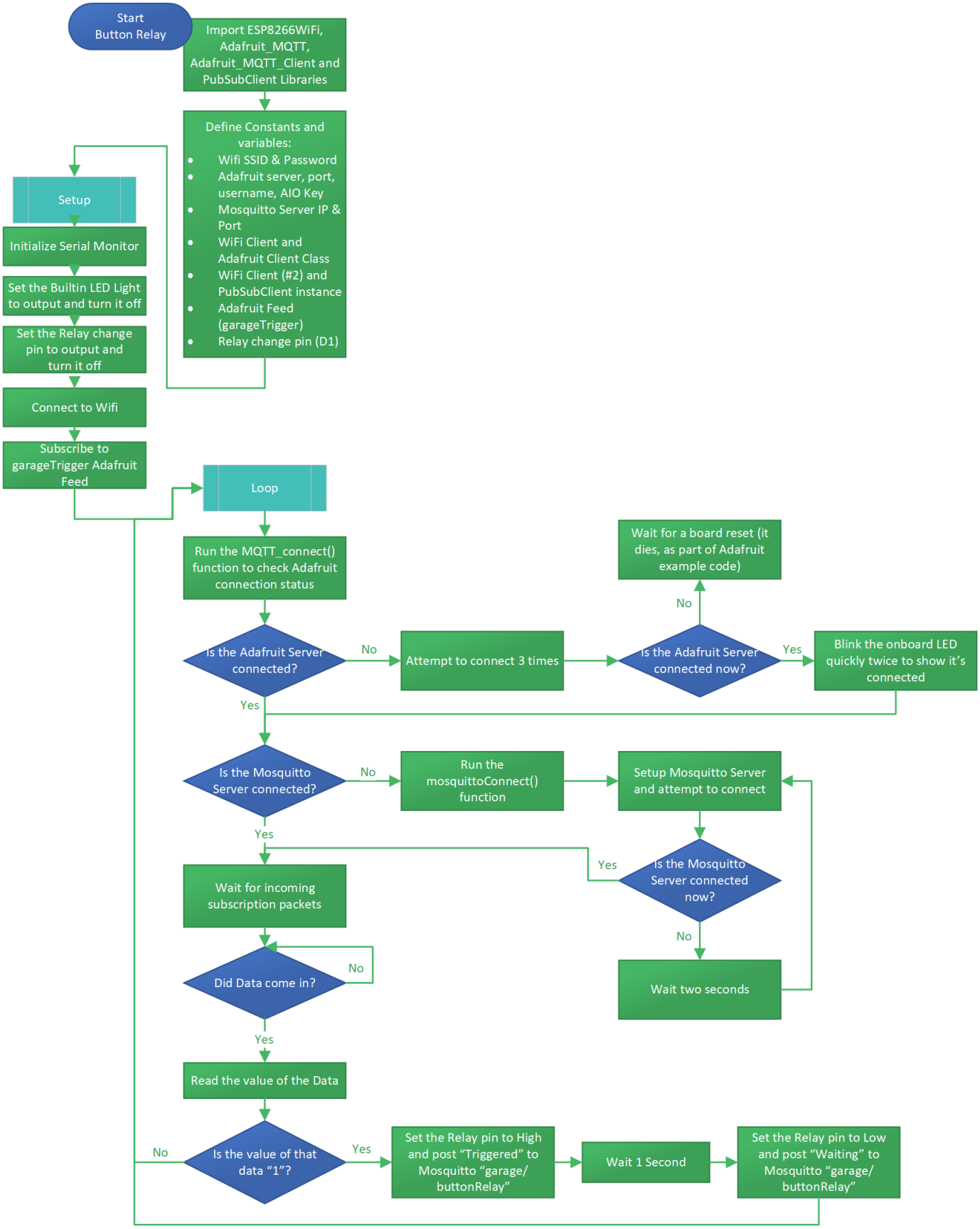
And a log of when the relay is triggered is displayed alongside when the car arrives in a Google Sheet

Circuit Diagram

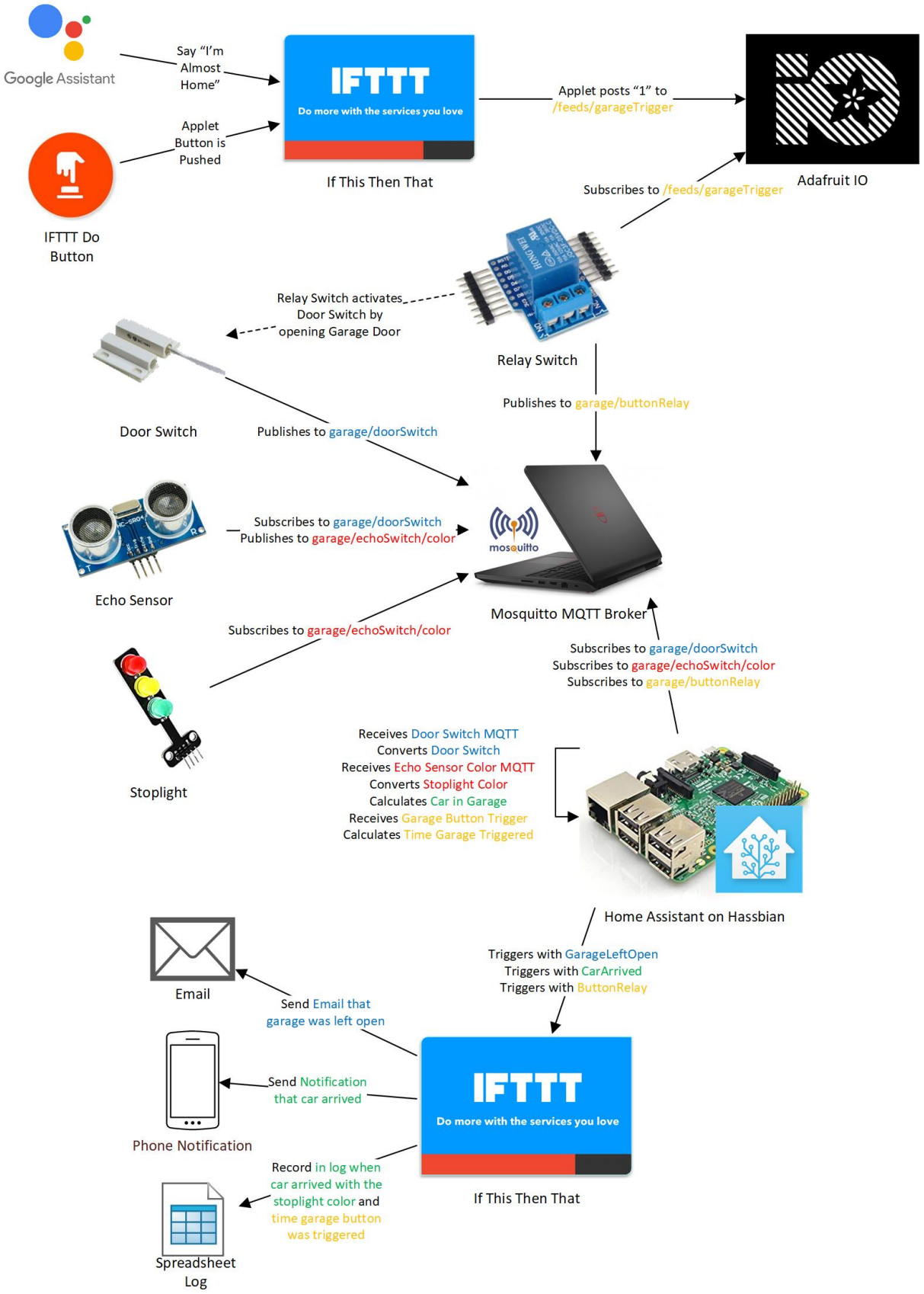


Necessary connections of the Relay Shield (if the shield is not placed directly on the board)

System Flowchart



System Diagram



Code (this, lab 4 and lab 5 code is available at https://github.com/AaronNelson95/IT441)

Configuration Files

configuration.yaml (stored in /home/homeassistant/.homeassistant)

The following was added to the Lab 5 YAML to create sensors for the new relay device

```
sensor:
    # Directly returns the value of garage/buttonRelay (Triggered or Waiting)
  - platform: mqtt
    name: "Garage Button Trigger"
    state_topic: "garage/buttonRelay"

    # This template shows the last time the relay button was triggered
  - platform: template
    sensors:
      time_garage_triggered:
        entity_id: sensor.garage_button_trigger
        friendly_name: "Time Garage Triggered"
        value_template: "{{as_timestamp(states.sensor.garage_button_trigger.
                        last_changed) | timestamp_custom('%A %b %d, %Y %H:%M:%S')}}"
```

automations.yaml (stored in /home/homeassistant/.homeassistant)

The last automation was added to watch for when the relay is triggered. The time of this is logged to a Google Sheet through IFTTT

```
- id: '1573538073059'
  alias: Garage Left Open Email
  description: Send an email to myself after the garage door was left open
  trigger:
    # Happens 1 minute after the door_switch value changes to "Opened"
  - entity_id: sensor.door_switch
    for: 00:01:00
    from: Closed
    platform: state
    to: Opened
  condition: []
  action:
    # Sends a simple IFTTT trigger with an event name
  - data:
      event: GarageLeftOpen
    service: ifttt.trigger
- id: '1573540765244'
  alias: Car Just Arrived Home
  description: Send a notification and record a log entry when I arrive home with the
                car
  trigger:
    # Happens when the car just arrives at home in the car_in_garage state
  - entity_id: sensor.car_in_garage
    platform: state
    to: The Car is Currently at Home
  condition: []
  action:
    # Sends an IFTTT trigger that also contains the value the stoplight sensor was set
      to
  - data_template: {"event":"CarArrived", "value1":"{{
                                          states.sensor.stoplight_color.state }}"}
    service: ifttt.trigger
- id: '1574312280000'
  alias: Garage Door Button Was Pushed
  description: Add time of occurrence to IFTTT spreadsheet
```

```
  trigger:
    # Happens when the relay button is triggered
  - entity_id: sensor.garage_button_trigger
    from: Triggered
    platform: state
    to: Waiting
  condition: []
  action:
    # Sends an IFTTT trigger that will log the time this occurs into a Google Sheet
      (Activated with the event "ButtonRelay")
  - alias: ''
    data:
      event: ButtonRelay
    service: ifttt.trigger
```

Arduino Code

Lab_6_Button_Relay.ino

```
/*
 * Created by Aaron Nelson
 * Lab 6 - User Interface - Garage Door
 * 11-26-19
 * This script is used with the Wemos Relay Shield. It is meant to trigger the opening
      of a garage door. It listens for a value (of '1') to be posted in an Adafruit IO
      feed. Then it triggers the relay for one second. It posts in the local MQTT server
      that this completed.
 *
 * The MQTT portion of this script was obtained from
      https://arduinodiy.wordpress.com/2017/11/24/mqtt-for-beginners/
 *
 * The Adafruit IO portion of this script was obtained from the Arduino example (after
      adding the Adafruit Library) at File -> Examples -> Adafruit MQTT Library ->
      mqtt_esp8266_callback
 *
 * The Relay triggering portion was obtained from
      https://github.com/petehoffswell/garagedoor/blob/master/garagedoor/garagedoor.ino
 *
    Relay Shield Pins (if soldering doesn't allow shield to be placed on board
      directly):
        5V: +5VDC
        D1 (triggering pin): D1
        G: GND
 */

#include <ESP8266WiFi.h>            // This contains the libraries that allows the board
                                        to connect to wifi
#include "Adafruit_MQTT.h"          // Library to access data from an Adafruit IO online
                                        feed
#include "Adafruit_MQTT_Client.h"   // Used to subscribe to an Adafruit IO online feed
#include <PubSubClient.h>           // This is used for communication with the local
                                        MQTT Server (Mosquitto)

/************************ WiFi Access Point *********************************/

#define WLAN_SSID       "***YOUR WIFI HERE***"        // Specify the name of your wifi
#define WLAN_PASS       "***YOUR PASSWORD HERE***"    // Specify the password for your
                                                          wifi
```

```
/************************* Adafruit IO Setup *********************************/

#define AIO_SERVER      "io.adafruit.com"       // Pulling data from the Adafruit site
#define AIO_SERVERPORT  1883                    // use 8883 for SSL
#define AIO_USERNAME    "***YOUR ADAFRUIT NAME***" // Username for Adafruit (goes before
                                                      the /feed/# MQTT feed)
#define AIO_KEY         "***YOUR ADAFRUIT KEY***"  // Obtained by going to
                                    io.adafruit.com and clicking AIO Key link in top
                                    right corner. Copy the "Active Key" here

/************************* Mosquitto MQTT Setup *********************************/

const char* mqttServer = "192.168.137.1";       // The location of your local MQTT
                                                      Server
const int mqttPort = 1883;                      // The port number your MQTT Server is
                                              running on (1883 is the default for Mosquito)

WiFiClient buttonRelayClient;                   // Name for our MQTT Wifi connection
                                                      client
PubSubClient relayClient(buttonRelayClient);    // Creates a partially initialised
                                                      client instance


/************ Global State (you don't need to change this!) ******************/

// Create an ESP8266 WiFiClient class to connect to the MQTT server.
WiFiClient client;

// Setup the MQTT client class by passing in the WiFi client and MQTT server and login
      details.
Adafruit_MQTT_Client mqtt(&client, AIO_SERVER, AIO_SERVERPORT, AIO_USERNAME, AIO_KEY);

/****************************** Feeds ***************************************/

// Notice MQTT paths for AIO follow the form: <username>/feeds/<feedname>
// Watch a feed called 'garageTrigger' and subscribe to changes.
Adafruit_MQTT_Subscribe garageTriggerbutton = Adafruit_MQTT_Subscribe(&mqtt,
AIO_USERNAME "/feeds/garageTrigger");

/************************* Sketch Code ********************************/

const int relayPin = D1;            // Defines the pin that controls the relay (Digital
                                          pin 1)
int relayState = LOW;               // Initially, the relay should not be on

// Bug workaround for Arduino 1.6.6, it seems to need a function declaration
// for some reason (only affects ESP8266, likely an arduino-builder bug).
void MQTT_connect();

void setup() {                      // This runs when the board is first turned on
  Serial.begin(115200);             // This allows serial information when connected to
                                          a computer (in here, this shows MQTT passed
                                          information)

  pinMode(relayPin, OUTPUT);        // The relay pin should be set as an output (it
                                          tells it when to be triggered)
  digitalWrite(relayPin, LOW);      // The relay pin should be set to low (it should not
                                          be on)
```

```
  pinMode(LED_BUILTIN, OUTPUT);      // Prepares the builtin light pin for output (to
                                     notify when the board is connected to MQTT servers)
  digitalWrite(LED_BUILTIN, HIGH);   // Initially, turn off the builtin LED light

  /* Connect to WiFi */
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");    // This is shown on the serial if connected to a
                                        computer
  Serial.println(WLAN_SSID);         // It displays the wifi it is trying to connect to

  WiFi.mode(WIFI_STA);               // It sets the wifi mode to "Station" (rather than
                                        "AP" which is the Access Point mode)
  WiFi.hostname("ButtonRelay");      // Hostname to uniquely identify our device
  WiFi.begin(WLAN_SSID, WLAN_PASS);          // Attempts to connect to wifi using
                                                provided credentials

  while (WiFi.status() != WL_CONNECTED) {   // While the wifi is not connected yet:
    delay(500);                      // every half second,
    Serial.print(".");               // it prints a dot on the Serial Monitor to show it
                                        is still trying to connect
  }
  Serial.println("");
  Serial.println("WiFi connected"); // When it does connect, show that there was success


  // Setup the Adafruit MQTT subscription for the garageTrigger feed.
  mqtt.subscribe(&garageTriggerbutton);
}




void loop() {                        // This constantly cycles through, ensuring both
                                        MQTT servers are connected and watching if
                                        information comes across Adafruit IO
  // Ensure the connection to the MQTT server is alive (this will make the first
  // connection and automatically reconnect when disconnected).  See the MQTT_connect
  // function definition further below.
  MQTT_connect();
  if (!relayClient.connected()){     // If Adafruit IO is no longer connected
    mosquittoConnect();              // Attempt to reconnect
  }

  // this is our 'wait for incoming subscription packets' busy subloop
  Adafruit_MQTT_Subscribe *subscription;
  // Adafruit will listen to the feed we subscribed to
  while ((subscription = mqtt.readSubscription(5000))) {
    Serial.println("in while loop");
    if (subscription == &garageTriggerbutton) {
      // Something was picked up in the feed (a value was posted to Adafruit)
      Serial.print(F("Got: "));
      String information = "";       // A string to insert the characters read
      char* ch = (char *)garageTriggerbutton.lastread;
      information.concat(ch);        // Add the read character to the information string
      Serial.println(information);

      if (information == "1") {      // A triggering value of "1" was obtained
```

```
        Serial.println("Starting the trigger");   // Show it was received in the Serial
                                                     Monitor
        relayClient.publish("garage/buttonRelay", "Triggered");   // Show it is being
                                                  triggered in the Local Mosquitto Server
        digitalWrite(relayPin, HIGH);               // Activate the relay
        delay(1000);                                // Wait one second (adjust according
                                                     to your garage door button)
        digitalWrite(relayPin, LOW);                // Deactivate the relay (to act as a
                                                     pulse)
        relayClient.publish("garage/buttonRelay", "Waiting");     // Show job completed
                                                                  in Local Mosquitto server
        information = "0";          // Done otherwise it would keep the last value (1)
                                       and continue to "read" triggered commands
      }
    }
  }
}




// Function to connect and reconnect as necessary to the Adafruit IO MQTT server.
// Should be called in the loop function and it will take care if connecting.
void MQTT_connect() {
  int8_t ret;

  // Stop if already connected.
  if (mqtt.connected()) {
    return;
  }

  Serial.print("Connecting to Adafruit... ");

  uint8_t retries = 3;                  // It will attempt to reconnect 3 times before
                                          quitting entirely
  while ((ret = mqtt.connect()) != 0) { // connect will return 0 for connected
       Serial.println(mqtt.connectErrorString(ret));
       Serial.println("Retrying Adafruit connection in 5 seconds...");
       mqtt.disconnect();
       delay(5000);  // wait 5 seconds
       retries--;    // Count against one of the retries because there was no success
       if (retries == 0) {
         // basically die and wait for WDT to reset me
         while (1);
       }
  }
  Serial.println("Adafruit Connected!");  // Connection was successful!

  // Blink LED quickly to show that it is connected to wifi and the Adafruit server
  digitalWrite(LED_BUILTIN, LOW);   // light turns on
  delay(100);
  digitalWrite(LED_BUILTIN, HIGH);  // light turns off
  delay(100);
  digitalWrite(LED_BUILTIN, LOW);   // light turns on
  delay(100);
  digitalWrite(LED_BUILTIN, HIGH);  // light turns off
}
```

```
// Function to connect and reconnect as necessary to the local Mosquitto MQTT server.
void mosquittoConnect() {               // When first connecting to MQTT or when it
                                           becomes disconnected, run this script
  relayClient.setServer(mqttServer, mqttPort);     // Setup the server connection with
                                                      information the user provided
  while(!relayClient.connected()) {     // If the MQTT server is not connected or becomes
                                           disconnected, try again
    Serial.print("Connecting to Mosquitto... ");
    if (relayClient.connect("ButtonRelay")) {      // Try to connect with this name
      Serial.println("Mosquitto Connected!");
    } else {
      Serial.print("failed... Re");   // Repeatedly says "failed... Reconnecting" as
                                           this loop cycles through
      // Serial.print(client.state());             // Show in the Serial Monitor what
                                           the current state of the server connection is
      delay(2000);                     // Wait two seconds before trying to connect to
                                           the server again
    }
  }
}
```