

**Senior Design Project**

**Design of Home Automation System for Appliance Control**

ENGE477 Senior Design Project II

Department of Engineering and Aviation Sciences

University of Maryland, Eastern Shore

Ndubuisi Iwuala

Kingsley Asonye

Project Advisor, Dr. Lei Zhang

Submitted

April 14, 2019

List of Contents

[List of Contents 2](#_Toc531958540)

[List of Figures 2](#_Toc531958541)

[List of Tables 4](#_Toc531958542)

[Abstract 5](#_Toc531958543)

[1. Introduction 6](#_Toc531958544)

[1.1. Backgound/Motivation 6](#_Toc531958545)

[1.2. Objective 8](#_Toc531958546)

[1.3. Design Requirements 8](#_Toc531958547)

[1.4. Design Constraints 8](#_Toc531958548)

[1.5. Design Methods 8](#_Toc531958551)

[2. Project Description 9](#_Toc531958552)

[2.1. System Description 9](#_Toc531958557)

[2.2. System Diagram 9](#_Toc531958558)

[2.3. System Functions 10](#_Toc531958559)

[3. Implementation Plan 11](#_Toc531958560)

[3.1. Tasks 11](#_Toc531958563)

[3.2. Team Organization 12](#_Toc531958565)

[3.2.1. Responsibility of Team Member 1. 12](#_Toc531958570)

[3.2.2. Responsibility of Team Member 2. 12](#_Toc531958571)

[3.3. Timeline/Milestones/Delivery Plan 13](#_Toc531958572)

[4. Implementation 14](#_Toc531958573)

[4.1. Implementation of Task 1. 14](#_Toc531958575)

[4.2. Implementation of Task 2. 16](#_Toc531958576)

[4.2.1. Implementation of Task 2.1. 16](#_Toc531958580)

[4.2.2. Implementation of Task 2.2. 18](#_Toc531958581)

[4.2.3. Implementation of Task 2.3. 19](#_Toc531958582)

[4.3. Implementation of Task 3. 20](#_Toc531958583)

[4.4. Implementation of Task 4. 24](#_Toc531958584)

[4.4.1. Implementation of Task 4.1. 24](#_Toc531958587)

[4.4.2. Implementation of Task 4.2. 25](#_Toc531958588)

[4.4.3. Implementation of Task 4.3 26](#_Toc531958589)

[4.5. Implementation of Task 5.1. 27](#_Toc531958590)

[4.5.1. Implementation of Task 5.1 (cont.) 28](#_Toc531958592)

[4.6 Implementation of Task 5.2. 29](#_Toc531958593)

[4.6.1. Implementation of Task 5.2 (cont.) 30](#_Toc531958595)

[4.6.2. Implementation of Task 5.2 (cont.) 31](#_Toc531958596)

[4.7. Implementation of Task 5.3 32](#_Toc531958597)

[4.8. Implementation of Task 6.1 33](#_Toc531958598)

[4.8.1. Implementation of Task 6.1. (cont.) 34](#_Toc531958601)

[4.9. Implementation of Task 6.2. 35](#_Toc531958602)

[4.10. Implementation of Task 7.1. 36](#_Toc531958603)

[4.10.1. Implementation of Task 7.1 (cont.) 37](#_Toc531958606)

[4.11. Implementation of Task 7.2. 38](#_Toc531958607)

[5. Conclusion 39](#_Toc531958608)

[6. Acknowledgement 40](#_Toc531958609)

[7. Appendix 41](#_Toc531958610)

[Component Specs 41](#_Toc531958611)

[A. Specs of NodeMCU V1.0 41](#_Toc531958612)

[B. Specs of Raspberry Pi 3 41](#_Toc531958613)

[Source Code. 41](#_Toc531958614)

[1. Source Code of Light Control Lambda Function 41](#_Toc531958615)

[2. Source Code of Television Control Lambda Function 45](#_Toc531958616)

[3. Source Code of Temperature Control Lambda Function 57](#_Toc531958619)

[8. REFERENCES 81](#_Toc531958620)

List of Figures

[Fig. 1 Amazon Echo 2nd generation (left) and the Google Home (right). 7](#_Toc531958512)

[Fig. 2 System Diagram 9](#_Toc531958513)

[Fig. 3 Flow Chart representation of the system 10](#_Toc531958514)

[Fig. 4 (a) Here is part of the registration process needed to prototype the Raspberry Pi. 14](#_Toc531958515)

[Fig. 5 (a) Stand-alone light invocation name. 16](#_Toc531958516)

[Fig. 6 (a) Television control invocation name. 18](#_Toc531958517)

[Fig. 7 (a) Temperature control invocaiton name. 19](#_Toc531958518)

[Fig. 8 Data Channel Credentials. 20](#_Toc531958519)

[Fig. 9 (a) Update Channel URL 21](#_Toc531958520)

[Fig. 10 (a) Light State ThingSpeak Data Channel. 21](#_Toc531958521)

[Fig. 11 Light Control Lambda Function Flow Chart 24](#_Toc531958522)

[Fig. 12 Television Control Lambda Function 25](#_Toc531958523)

[Fig. 13 Temperature Control Lambda Function. 26](#_Toc531958524)

[Fig. 14 IR Transmitter Schematic Design 27](#_Toc531958525)

[Fig. 15 IR Transmitter Breadboard Design. 28](#_Toc531958526)

[Fig. 16 Calculations for IR Transmitter Circuit 29](#_Toc531958527)

[Fig. 17 Circuit Simulation for IR Transmitter 30](#_Toc531958528)

[Fig. 18 Simulation Results 31](#_Toc531958529)

[Fig. 19 PCB Design of Transmitter 32](#_Toc531958530)

[Fig. 20 Trace Current and Width for PCB Design 32](#_Toc531958531)

[Fig. 21 Temperature Control Schematic 33](#_Toc531958532)

[Fig. 22 Breadboard for Temperature Control Circuit 34](#_Toc531958533)

[Fig. 23 PCB Design for Temperature Control Circuit 35](#_Toc531958534)

[Fig. 24 Trace Current & Width for PCB Design 35](#_Toc531958535)

[Fig. 25 Light Control Schematic 36](#_Toc531958536)

[Fig. 26 Breadboard Design for Light Control 37](#_Toc531958537)

[Fig. 27 PCB Design for Light Control 38](#_Toc531958538)

[Fig. 28 Trace Current & Width for PCB Design 38](#_Toc531958539)

List of Tables

[Table 1. Project Timeline and Delivery Plan 14](#_Toc531644504)

Abstract

By the end of the project, we should have a system where the user will be able to control their television, home temperature, and lights all by using voice commands. As well as a sensor that will detect when the door is open.

1. Introduction

## Backgound/Motivation

In an age where technological advancements are slowly becoming a norm in society, it can be fairy difficult for the average human being to identify “the next big thing.” With the internet being such a broad space and also a necessity in today’s society, it can be extremely beneficial for machines to communicate over the internet to minimize the amount of human interaction. This is where “Internet of Things” comes into play.

A “thing” is any device that is connected over the internet excluding laptops, cellphones, and personal computers. Internet of Things (IoT), first coined by Kevin Ashton in 1999 during a presentation for Procter & Gamble, is a network of physical devices from sensors to vehicles to home appliances that are connected to the internet that collect and exchange data. Though IoT was coined in 1999, the first IoT device can be traced back to the 1980s at Carnegie Melon University. A Coca Cola machine was connected to the internet and local programmers would connect by internet check whether drinks were available or if the drinks was cold before going to the vending machine. By 2013, IoT became an entire system comprising of sensors, microcontrollers, embedded systems and more which all send and receive data and execute functions. According to The Statistics Portal, in 2015 there was an estimated 15.41 billion connected IoT devices worldwide. Today in 2018 there is an estimated 23.14 billion IoT devices connected worldwide. By 2020 there will be an estimated 30.73 billion devices and by 2025 an estimated 75.44 billion connected IoT devices.

So how does the IoT actually work? As was previously stated, IoT is a network of physical devices all connected. Take a humidity sensor for example. Sensors take a physical input and output an electrical signal that can be converted to values by humans. Because the humidity sensor is connected to the internet, the data from the sensor can be sent to the cloud to be processed. That processing is done by code that is written in the cloud. The cloud, metaphor for the internet, consists of large data centers offering services such as storage, servers, computing, and much more. That data is sent to the cloud, processed, and depending on the system the data can be stored in the cloud or used to perform a specific function. That function could be to monitor humidity levels in a room. If the humidity gets too high indoors, that system could alert the user that there is a possibility that a window has been left open. In summary, this is how the internet of things works.

As we move further and further into the digital age, the home automation industry has been increasing year after year. Home automation refers to the automation of appliances using IoT devices. With devices such as the Amazon Echo and Google Home, users can configure and control many devices and appliances with the use of voice commands and/or app scheduling from a smartphone. Fig. 1 below shows these devices. According to Statista, in 2016 there were about 9.66 million energy management smart devices in homes across the United States. In 2022 there will be a projected 30.85 million energy management smart devices across homes in the United States. This does not include other devices such as security, comfort and lighting, home entertainment and many others. These numbers prove there is a huge market in the home automation business. Within the next couple of years, it is safe to say that nearly every home in the U.S. will have at least one smart device.

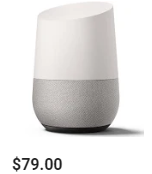
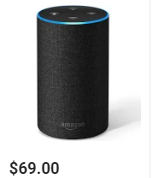


Fig. 1 Amazon Echo 2nd generation (left) and the Google Home (right).

Over recent years IoT devices have been one of the leading topics in technology. As studies show IoT devices are gaining more and more popularity as time goes on. We can safely assume IoT devices will become a norm in society. In November of 2014, Amazon announced the release of the Amazon Echo, the smart speaker with Amazon Alexa built in. Because the Echo has no screen, the device is solely reliant on speech to operate. This made Amazon to increase their experience in machine learning as well as artificial intelligence to make the product as efficient as possible. Fast forward to today, Amazon’s Alexa has become the primary interface for household smart devices across the nation. Amazon has built the necessary API’s (Application Program Interface) to allow those who are selling their own devices to connect their creations to the Alexa service. These can range from lighting, outlets, fans, thermostats, security, and more.

For this project, the use of Alexa Voice Services (AVS) and IoT devices will be used to develop a system for home automation using the voice recognition capability of Alexa. This would be extremely beneficial to those who are disabled still living at home as this creates an easier way of life. Small tasks that we do every day that require human interaction such as switching light on and off or having to change the channel on the television would be eliminated thanks to the capability of the Alexa. Those individuals with disabilities such as cerebral palsy or rheumatoid arthritis can feel more comfortable in their homes since the need of assistance for some everyday tasks can be reduced. Karin Willison, a woman with cerebral palsy, conducted a survey for disabled individuals and smart home devices. According to this survey, smart thermostats and lights were some devices among several devices listed that can potentially be an added benefit in the daily lives of those who have a disability. However, one down side to these devices is the cost. The nest smart thermostat can cost an upward of $249. Phillips smart light kit can cost around $69.99. Common household thermostats can range from $35-$60, which is almost a $190 difference. A 2-pack of 60-Watt bulbs can cost $2 at Home Depot. Though these smart products on the market come with a large number of added features, Digital Trends shows that only 10% of the people surveyed actually owned smart lightbulbs while 62% of them were previously aware of the technology. Cost can play a great role in the decision people make in purchasing new technology.

The purpose of this project is to develop a system that shares similar functionality as the current products on the market at a much lower cost as well as a system for those individuals who are disabled who can benefit from an inexpensive product. As previously stated, there are devices on the market that will accomplish similar tasks as our system. However, from a financial standpoint, these products will have a greater cost compared to the cost of our system.

## Objective

The objective of this project is to design a voice-controlled system that will allow a user to control appliances with spoken commands as well as an open/close detection system that will notify a user if the door, window, etc. is open. Those appliances will be the television, light, and temperature.

## Design Requirements

1. Will use display to show current & target temperature, heat, cool and fan for temperature control.
2. Stand-alone lights will be controlled by voice.
3. Television will be controlled by voice.
4. Temperature will be controlled by voice.
5. User will be notified if front door is opened or closed.

## Design Constraints

1. Internet connectivity is required due to fact of the use of IoT devices in this project
2. $250 budget
3. Will use AVS (Alexa Voice Services) for voice control capability.

## Design Methods

1. Project Description

## System Description

This voice automated system is primarily controlled by the microprocessor Raspberry Pi. For the project, the Raspberry takes the role of an Amazon Echo. The user will speak a voice command to the microphone that is connected to the Raspberry Pi. The Raspberry Pi must be connected to the internet. From there, the command is processed in the cloud. Depending on the command sent, there will be a defined Alexa skill that will match the command to a skill. The skill will run depending on the command that was spoken. The Alexa skill will call the Lambda function that it is linked to. This function will generate an output of confirmation for the user that is sent back to the Alexa skill to be spoken by Alexa through a speaker. The Lambda function will also send data to a ThingSpeak channel. That data is a representation of the function to be executed by the NodeMCU. Therefore, depending on the command, the Lambda function will send that data to the appropriate data channel. The NodeMCU will read the data from the channel and execute a command based on the data it reads.

The open/close portion is based on the proximity sensor that will be attached to the front door. This proximity sensor will be reed switch type sensor. A reed switch is magnetic switch that when closed, allows current flow. The magnetic portion of the sensor will be attached to the door and the sensor will be connected to the NodeMCU. When the user opens the front door, the user will receive a notification from their cell phone notifying the user that the door has been opened. However, this will only occur when set by the user. Through a web application, the user will be able to control whether notifications will be sent or not.

## System Diagram

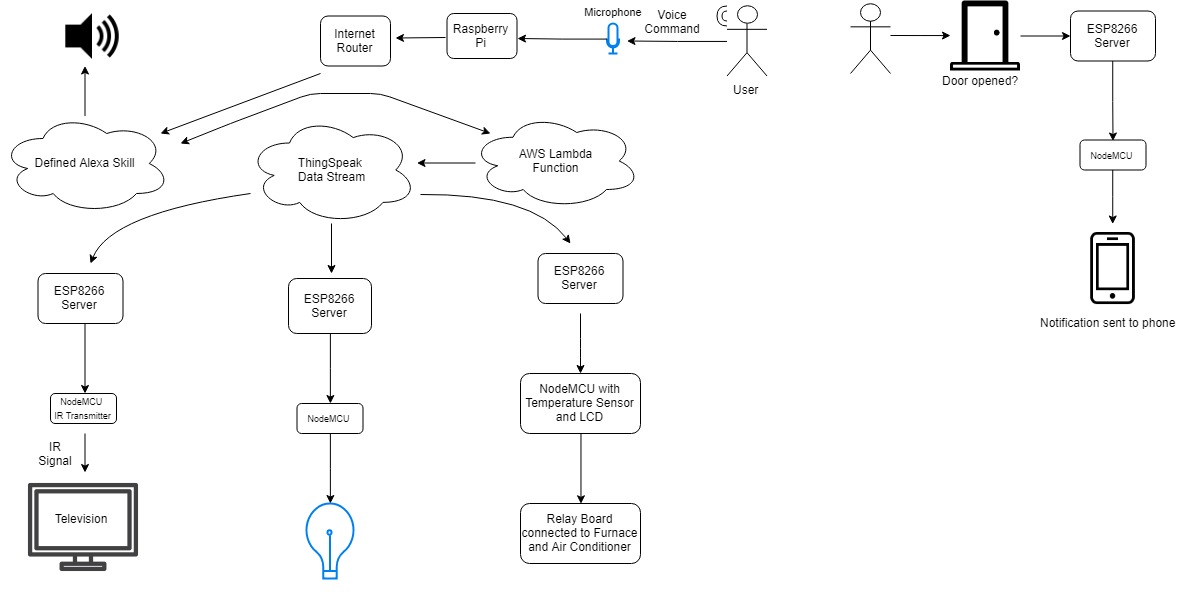


Fig. 2 System Diagram

## System Functions

Fig. 3 below is the flow chart representation of the system.

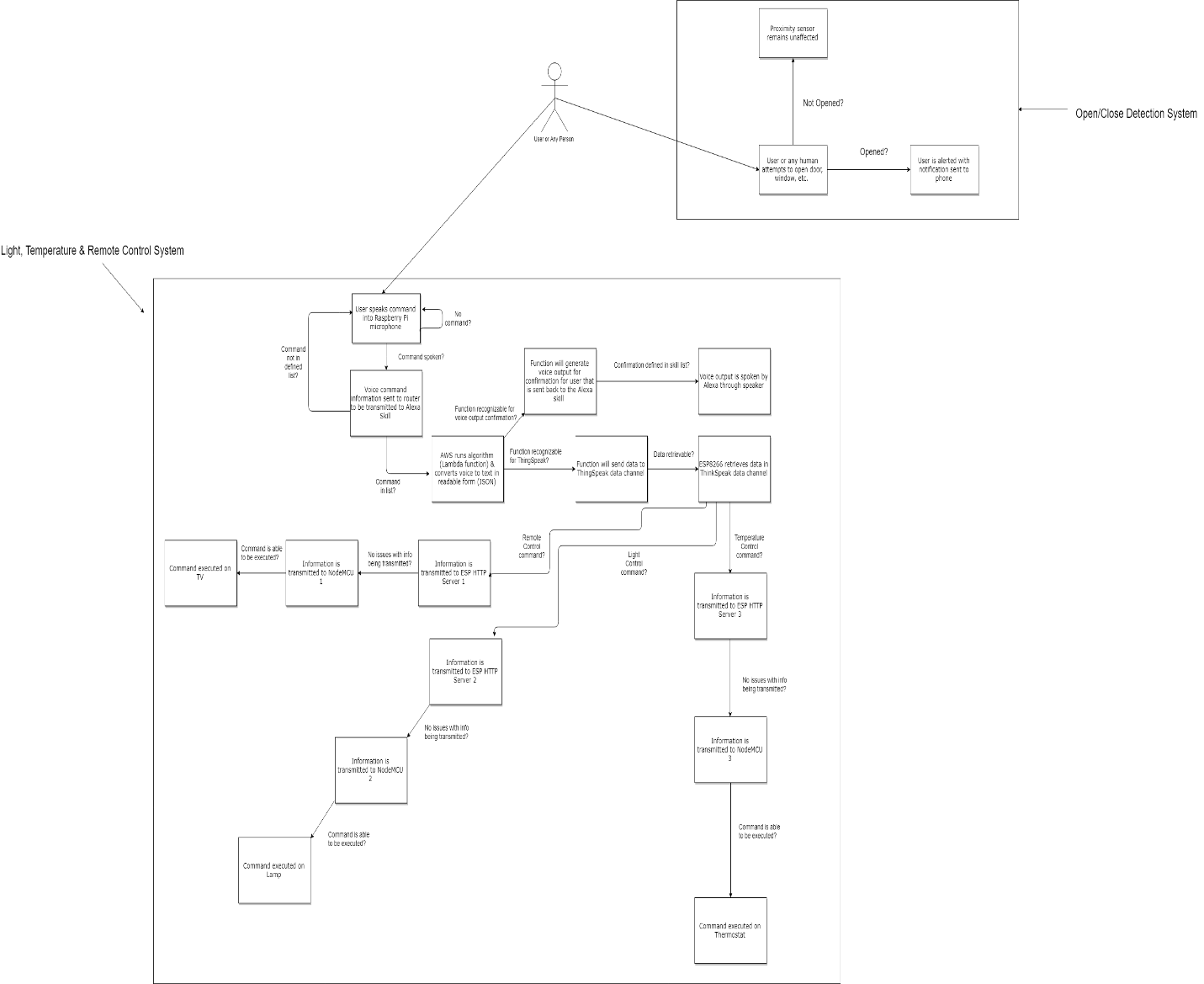


Fig. 3 Flow Chart representation of the system

1. Implementation Plan

In this section here, we will have our tasks, organization, and timeline.



## Tasks



* Task 1. Create AVS Prototype on Raspberry Pi
* Task 2. Alexa Skill Program
  + Subtask 2.1. Program Alexa Skill for light control
  + Subtask 2.2. Program Alexa Skill for television remote control function
  + Subtask 2.3. Program Alexa Skill for thermostat control function
* Task 3. Setup Data Channel for the Television, Temperature, and Lights.
* Task 4. AWS Lambda Program
  + Subtask 4.1. Program Lambda functions for light control
  + Subtask 4.2. Program Lambda function for television control
  + Subtask 4.3. Program Lambda function for temperature control
* Task 5. Remote Control Design
  + Subtask 5.1. Create schematic/breadboard design for IR transmitter
  + Subtask 5.2. Circuit design simulation (results & calculations)
  + Subtask 5.3. PCB design of IR transmitter
  + Subtask 5.4. Solder components to PCB
  + Subtask 5.5. Program NodeMCU
  + Subtask 5.6. Encasing Design
* Task 6. Temperature Control Design
  + Subtask 6.1. Create schematic/breadboard design for temperature control
  + Subtask 6.2. PCB design of temperature control
  + Subtask 6.3. Solder components to PCB
  + Subtask 6.4. Program NodeMCU
  + Subtask 6.5 Encasing Design
* Task 7. Stand-Alone Light Control Design
  + Subtask 7.1. Create schematic/breadboard design for light control
  + Subtask 7.2. PCB design of Light Control
  + Subtask 7.3. Solder components to PCB
  + Subtask 7.4. Program NodeMCU
* Task 8. Open/Close Sensing Design
  + Subtask 8.1. Create schematic
  + Subtask 8.2. PCB design
  + Subtask 8.3. Solder components
  + Subtask 8.4. Program NodeMCU
  + Subtask 8.5. Encasing Design
* Task 9. Power Calculation
* Task 10. Design Verification
  + Subtask 10.1. Assembly
  + Subtask 10.2. System testing and refinement

## Team Organization

Team Member 1: Ndubuisi Iwuala (Computer Engineering)

Team Member 2: Kingsley Asonye (Computer Engineering)



### Responsibility of Team Member 1.

Task 1: Create AVS Prototype on Raspberry Pi

Task 2: Alexa Skill Program

Task 3: Create a Data Channel for the Television, Temperature, and Lights.

Task 4: AWS Lambda Program

Task 5: IR Emitter Design

Task 6: Temperature Control Design

Task 7: Stand-Alone Light Control Design

Task 8: Open/Close Sensing Design

Task 9: Determine efficient method to power each device

Task 10: Design Encasing for Control Designs

### Responsibility of Team Member 2.

Task 5: IR Emitter Design

Task 6: Temperature Control Design

Task 7: Stand-Alone Light Control Design

Task 8: Open/Close Sensing Design

Task 9: Determine efficient method to power each device

Task 10: Design Encasing for Control Designs

## Timeline/Milestones/Delivery Plan

Table 1. Project Timeline and Delivery Plan

|  |  |  |  |
| --- | --- | --- | --- |
| Time | Task | Comments | Responsible Personnel |
| Week 5 | Begin Task 1 | Create AVS Prototype on Raspberry Pi | Ndubuisi |
| Week 6 | Begin Subtask 2.1, 2.2, 2.3 | Alexa Skill Program | Ndubuisi |
| Week 7 | Finish Task 2 & Begin Task 3 | Data Channel Setup | Ndubuisi |
| Week 8 | Finish Task 3 & Being Subtask 4.1, 4.2, 4.3 | 3 weeks needed to complete AWS Lambda Function for Light, Thermostat, and Television Control | Ndubuisi |
| Week 9 | Continue Subtask 4.1, 4.2, 4.3 | N/A | Ndubuisi |
| Week 10 | Begin Subtask 5.1, 5.2, 5.3 & Finish Subtask 4.1, 4.2, 4.3 | 3 weeks needed to complete Subtask 5.1, 5.2 5.3. Task 5 will carry over to next semester. | Ndubuisi/Kingsley |
| Week 11 | Continue Subtask 5.1, 5.2, 5.3 | N/A | Kingsley |
| Week 12 | Finish Subtask 5.1, 5.2, 5.3 & Begin Subtask 6.1 & 6.2 | 3 weeks needed to complete Subtask 6.1 & 6.2. Task 6 will carry over to next semester. | Kingsley |
| Week 13 | Continue Subtask 6.1, 6.2 | N/A | Kingsley |
| Week 14 | Finish Subtask 6.1, 6.2 & Begin Subtask 7.1, 7.2 | 3 weeks needed to complete Subtask 7.1, 7.2. Task 7 will carry over to next semester | Kingsley |
| Week 15 | Continue Subtask 7.1, 7.2 | N/A | Kingsley |
| Week 16 | Begin Task 8 | 4 weeks needed to complete all Subtasks for Task 8. | Ndubuisi/Kingsley |
| Week 17 | Begin Subtask 5.4, 5.5, 5.6 | 3 weeks needed to complete Subtask 5.4, 5.5, 5.6 | Ndubuisi/Kingsley |
| Week 18 | Begin Subtask 6.3, 6.4, 6.5 & Subtask 7.3, 7.4 | 3 weeks needed to complete Subtask 6.3, 6.4, 6.3 & Subtask 7.3, 7.4 | Ndubuisi/Kingsley |
| Week 19 | Finish Subtask 5.4, 5.5, 5.6 & Task 8 | Finish & Finalize remote and open/close designs | Ndubuisi/Kingsley |
| Week 20 | Finish Subtask 6.3, 6.4, 6.5 &7.3, 7.4 & Begin Task 9 | Finish & Finalize Temperature | Ndubuisi/Kingsley |
| Week 21 | Finish Task 9 | Complete Power Calculation for each Device | Ndubuisi/Kingsley |
| Week 22 | Begin Task 10 | Begin Design Verification | Ndubuisi/Kingsley |

1. Implementation

## Implementation of Task 1.

To begin this project, the first step is to install the appropriate image onto the Raspberry Pi in order for the system to operate. This involves downloading and burning the Raspbian image called Raspbian Stretch onto a microSD card. Once this is complete, next is to implement Amazon’s Alexa SDK. First step is registering the Raspberry pi as a new product on the Amazon Alexa Developer Console. Fig. 4(a) shows the screen that is displayed when registering the Raspberry Pi as an Alexa product. The product is given a name and ID. That ID is created by the user and must be used during the authorization process. Because the goal is to have Alexa on the Raspberry Pi, the option “Device with Alexa built-in” is selected. Fig. 4(b) shows the security profile that is required for the registration process as well. This profile needs a name, description, and an Amazon generated security profile ID and client ID. The profile ID is what identifies our security profile in Amazon services. The client ID is what is used during the setup process for Alexa Voice Services (AVS) for authorization. Next step is to follow the directions provided by Amazon Alexa GitHub. However, this posed a few problems. Because the instructions were given on an image called Raspbian Jessie, an older version of the Raspbian series, parts of the installation failed. As stated earlier, Raspbian Stretch is the image burned on the microSD card. After some research, a solution found involved a man named Henry Mendez, who managed to implement the Alexa SDK for use on the Raspbian Stretch image in use. Using his GitHub and running the necessary scripts, we were able to install and run the Alexa SDK onto the pi. Fig. 4(c) shows the Alexa SDK running on the Raspberry Pi. The application will connect to Alexa cloud, authorize the device, and wait for the wake word “Alexa”.

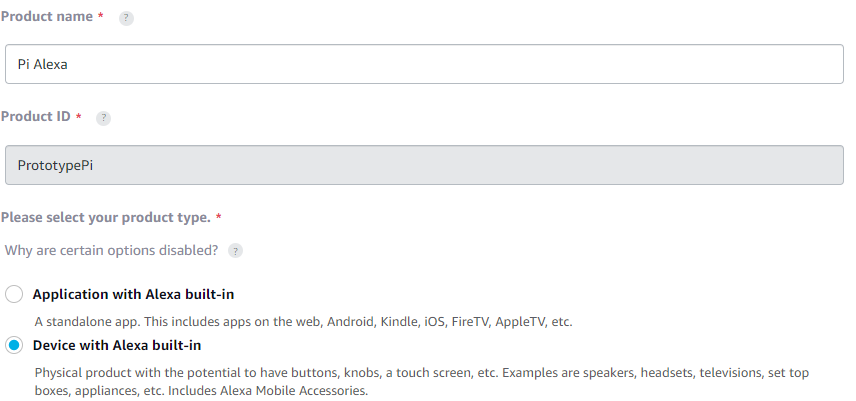


Fig. 4 (a) Here is part of the registration process needed to prototype the Raspberry Pi.

A screenshot of a social media post

Description automatically generated

Fig. 4 (b) Security Profile.

A screenshot of a computer

Description automatically generated

Fig. 4 (c) AVS program on Raspberry Pi.

## Implementation of Task 2.

Within the Amazon Developer Console, custom Alexa skills can be developed. This can either be done by using the interaction model Amazon provides or writing a JSON schema code from scratch. Due to underling factors, the interaction model deemed the best fit in the development of the JSON code. For the skill to be built correctly with no errors, these required parameters are to be fulfilled. Those parameters are invocation name, intents, and sample utterances. The invocation name is the name given to begin an interaction with the custom skill. For example, if an invocation name was “space facts”, the user can say “Alexa, ask space facts for a fact”. This is how Alexa will know what skill is being called. Next, is the development of the custom intents. There is no limit to the amount of intents you can have for a particular skill. An intent represents an action that fulfills a user’s spoken request. Next, is the development of the sample utterances for each intent. These utterances map to the intents and are the phrases the user must say to evoke each intent. These are the basics needed for the Alexa skill to be created. The user will now be able to say the invocation name as well as the sample utterance in order to call the intent. For example, the user can say “Alexa ask space facts for a fact.” The invocation name is “space facts” and the utterance is “a fact”. Once all these parameters have been fulfilled, the last step is to specify an endpoint. This endpoint will be linked to a Lambda function. That function will run in when the intent is called from the Alexa skill. That function is where the output of the skill is located.



### Implementation of Task 2.1.

(a) below shows the invocation name of the stand-alone light control Alexa skill. The invocation name is named “light switch” due to the fact that this is a very relatable name the user can say for this functional. (b) shows the list of intents for the Alexa skill. Two intents that are custom named TurnLightONIntent and TurnLightOFFIntent respectively. The other intents can be observed as required intents due to the functionality of the Alexa skill. Due to the uniqueness of this skill, the only intents needed are these two. (c) and (d) shows the utterances that will be required to evoke the intents. As previously stated, these utterances are directly mapped to each intent so only one intent will run depending on the user’s command. Similar to intents, there is no limit to the amount of utterances that can be mapped to an intent.

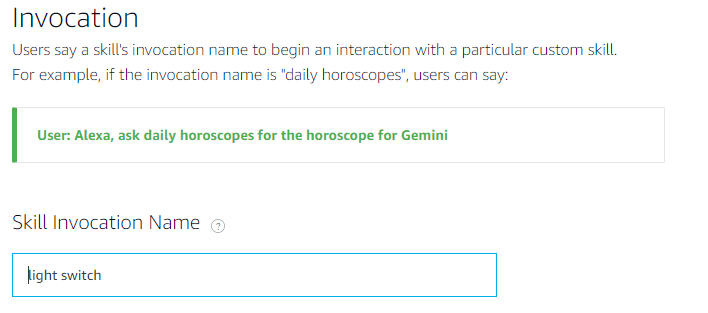


Fig. 5 (a) Stand-alone light invocation name.

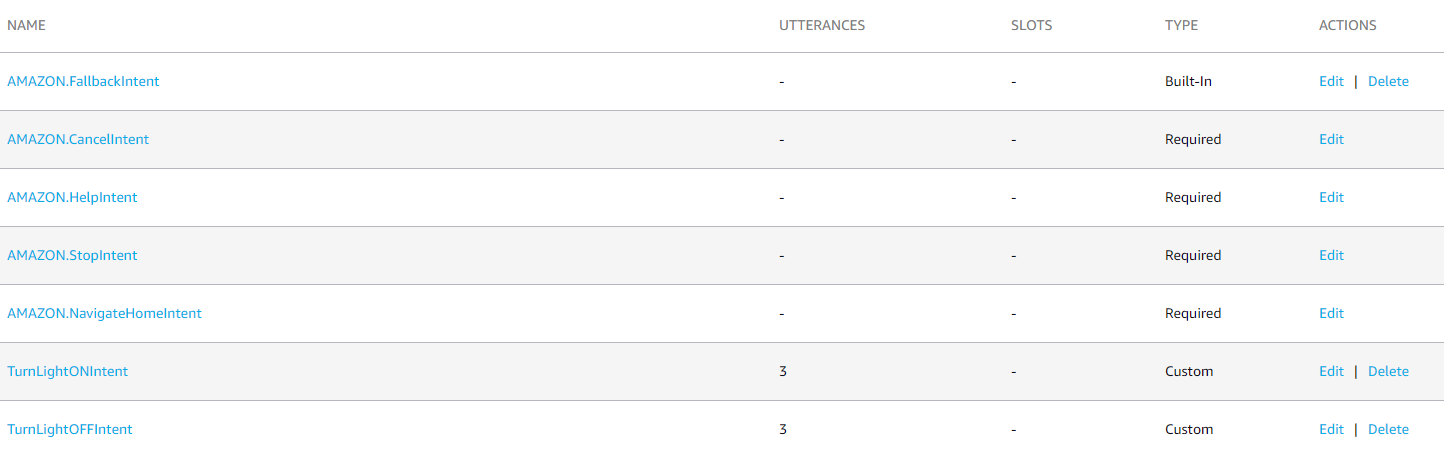


Fig. 5 (b) Intent list for light control Alexa skill.

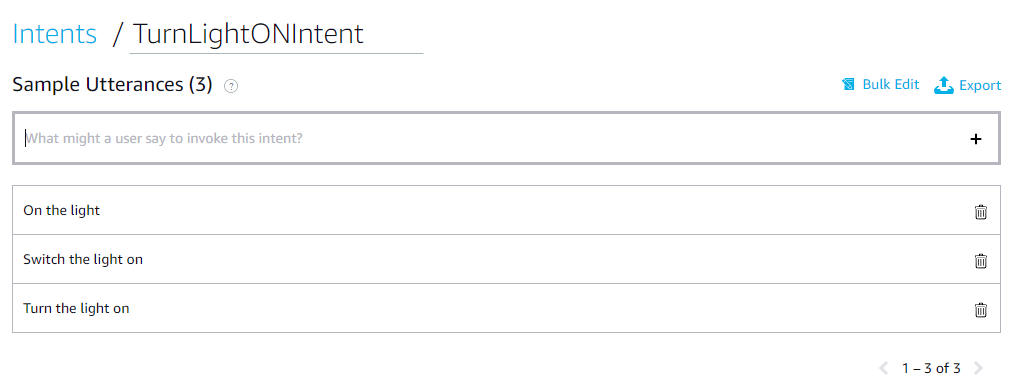


Fig. 5 (c) Utterances for TurnLightONIntent.

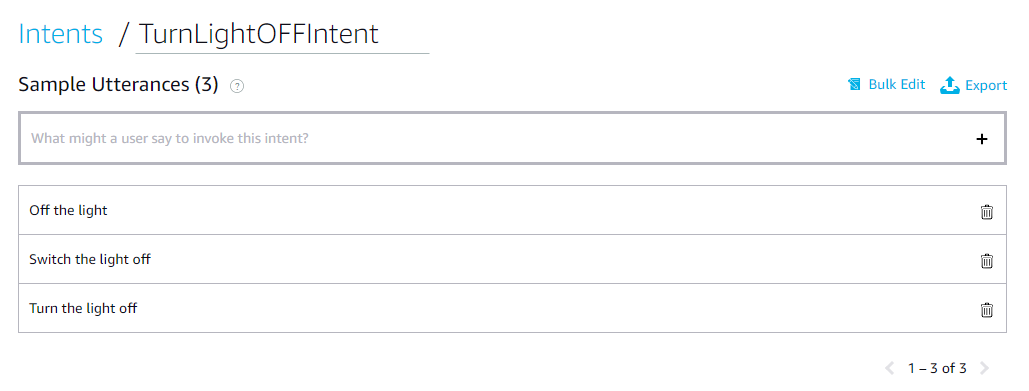


Fig. 5 (d) Utterances for TurnLightOFFIntent.

### Implementation of Task 2.2.

(a) below shows the invocation name for the television control Alexa skill. The invocation name is “remote” due to the fact that this is a very practical name for the functionality of the system. The user will use the name remote to call this specific skill. (b) shows list of intents needed for this skill. Due to the number of different functions a remote has, the number of intents increases. Very much like the skill previously displayed, this skill also has utterances that are mapped in which each intent. The user has wide range of phrases to use that Alexa will recognize.

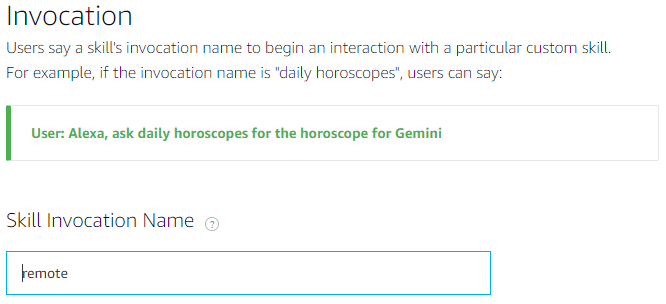


Fig. 6 (a) Television control invocation name.

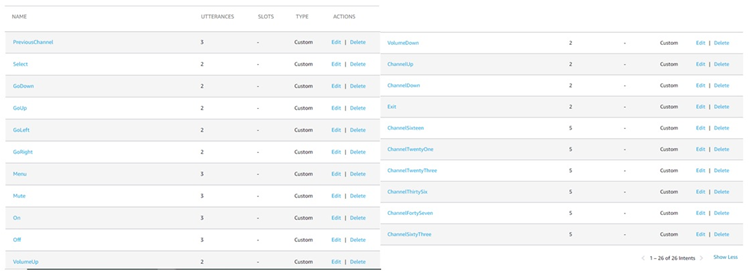


Fig. 6 (b) Intents for television control.

### Implementation of Task 2.3.

(a) below shows the invocation name for the temperature control Alexa skill. The invocation name is “thermostat” due to the fact that this is a very practical name for the functionality of the system. The user will use the name thermostat to call this specific skill. (b) shows the list of custom intents for the temperature control skill. This particular skill has several intents due to the fact that there are several different options the user can have when changing temperature. Again, this skill also has utterances that are mapped to each intent. That gives the Alexa a wide range of phrases to recognize.

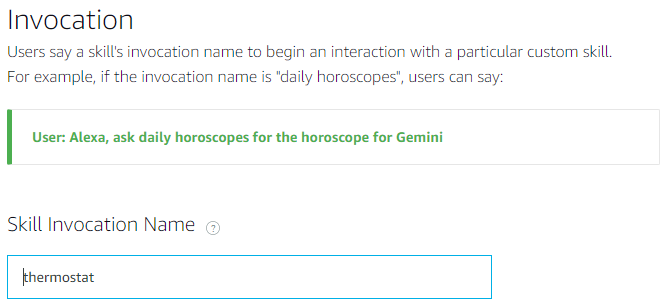


Fig. 7 (a) Temperature control invocaiton name.

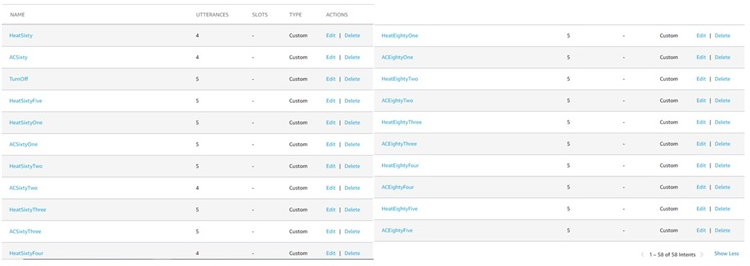


Fig. 7 (b) Intent list for temperature control.

## Implementation of Task 3.

Task 3 involved developing data channels on ThingSpeak to store the state of each output on the cloud. The idea behind this is to use HTTP GET requests to allow the NodeMCU to retrieve the data in this data channel. HTTP GET request allows the NodeMCU to retrieve data from a given server. That is the only function of a GET request. Using this method will not alter the data in anyway. There are other ways to retrieve information from the data channel. One common way is to use an MQTT broker. MQTT is Messaging Queuing Telemetry Transport. It is a lightweight publish and subscribe system where the client can publish and receive messages. This sounds like the perfect solution for IoT communication. However, this requires the router to be set to use port forwarding. Most routers have the ports closed, meaning the user would have to open the required ports manually. Since the objective is to only read data and not send data, HTTP GET requests are ideal for this project.

The first step in creating a ThingSpeak data channel by creating a ThingSpeak account. Next, is to create a new data channel and enter the required credentials. below shows what those required credentials are in order to create a new data channel. The channel must be given a name and description, followed by the required fields needed for the channel. The field represents where data will be stored within the channel. For example, in the case for a temperature and humidity sensor, Field 1 can be for temperature reading and Field 2 can be for humidity readings. API keys to send data to a specific field which (a) shows how the data will be written into each field. Using the URL, any number can be written into the field by changing the value at the end of the URL. (b) is the URL used to read the data from the field. Similar to the URL in (a), the URL can be changed slightly to read the results needed. The number of fields read, and the number of results read can be changed directly from the URL parameters. (a), (b), (c), (d), and (e) show how the values will be displayed going into the channels for the light, temperature, and television respectively.

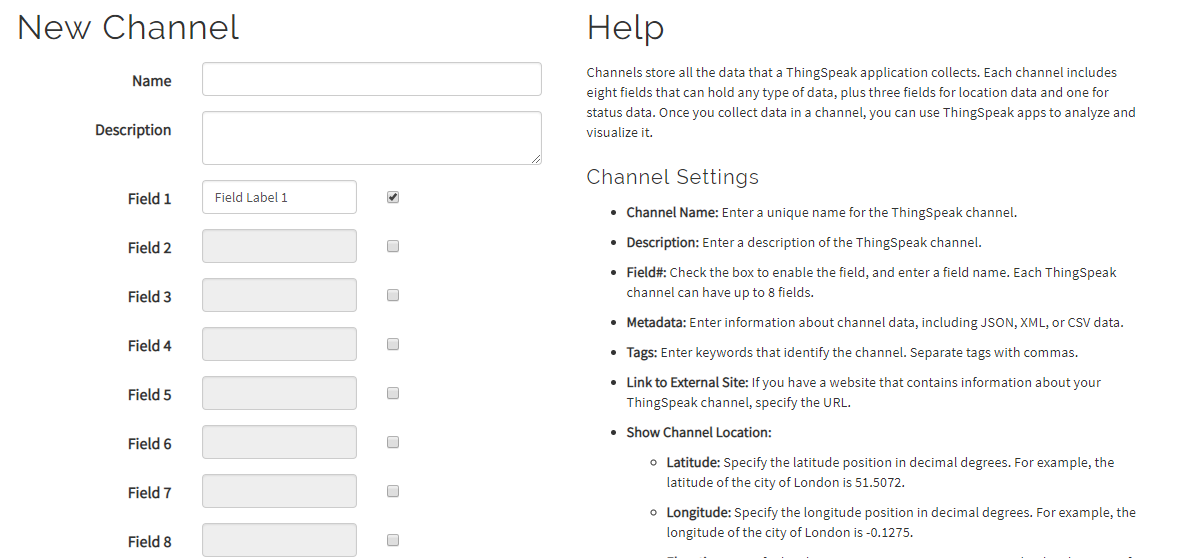


Fig. 8 Data Channel Credentials.

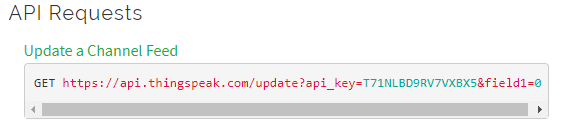


Fig. 9 (a) Update Channel URL

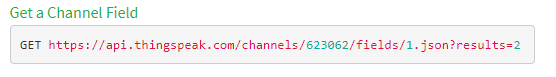


Fig. 9 (b) Read Channel Field URL.

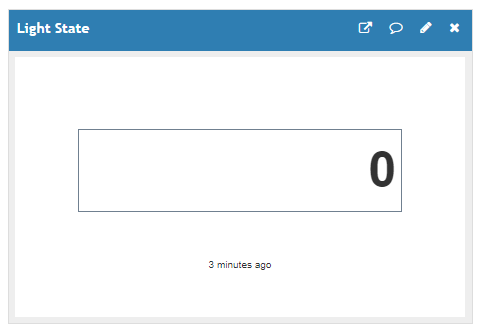


Fig. 10 (a) Light State ThingSpeak Data Channel.

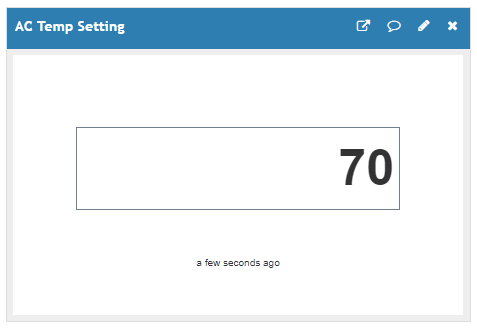


Fig. 10 (b) Thermostat ThingSpeak Data Channel (AC).

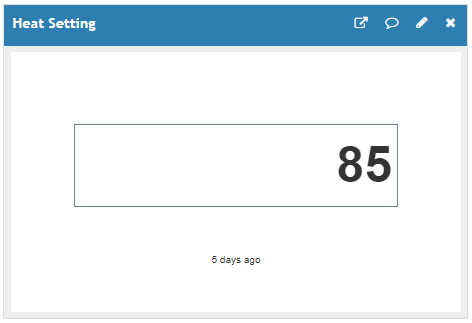


Fig. 10 (c) Thermostat ThingSpeak Data Channel (Heat).

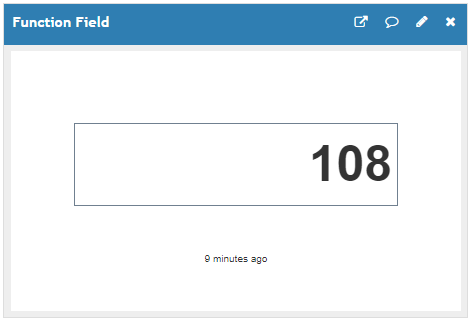


Fig. 10 (d) Television Control ThingSpeak Data Channel (TV Function).

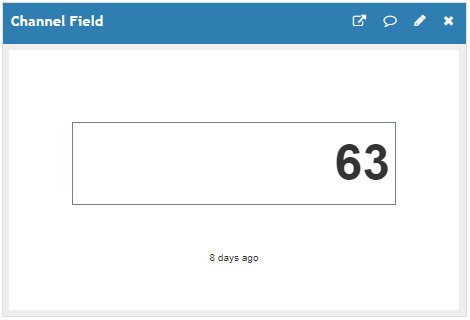


Fig. 10 (e) Television Control ThingSpeak Data Channel (TV Channel).

## Implementation of Task 4.

Task 4 involves programming the function codes for our Alexa skills. These codes are called AWS Lambda functions. An AWS Lambda function is a serverless compute service that is running is the code as background tasks. The Lambda function will run only when triggered by the Alexa skill. The Lambda function gives the output for the Alexa skill as well as the output for the ThingSpeak data channel. AWS Lambda offers several different runtimes and languages we can use to program the function. The languages vary from Python, C#, and Node.js



### Implementation of Task 4.1.

Fig. 11 below shows the program flow of the Lambda function for the light control function. Because the utterance for Alexa skill is mapped to the intent type, if the utterance matches one of the intents, that intent is valid, and a confirmation response will be sent back to the user. If the utterance does not match an intent, an error message will be sent back to the user. If the intent was to turn the light off, the value 0 will be sent to the ThingSpeak channel. If the intent was to turn the light on, the value 1 will be sent to the ThingSpeak data channel.

A close up of text on a black background

Description automatically generated

Fig. 11 Light Control Lambda Function Flow Chart

### Implementation of Task 4.2.

Fig. 12 below shows the program flow of the Lambda function for the television control Lambda function. Similar in structure to the light control Lambda function, there are two intent types. Because the utterance for Alexa skill is mapped to the intent type, if the utterance matches one of the intents, that intent is valid, and a confirmation response will be sent back to the user. If the intent was to change the television to a specific channel, such as channel 16 or 63, the value of that channel will be sent to the ThingSpeak data channel. If the intent was to command the television to do some function, such as turn on or turn the volume up or down, a value corresponding to that function will be sent to the ThingSpeak data channel. These two values will be sent to separate data fields within the channel.

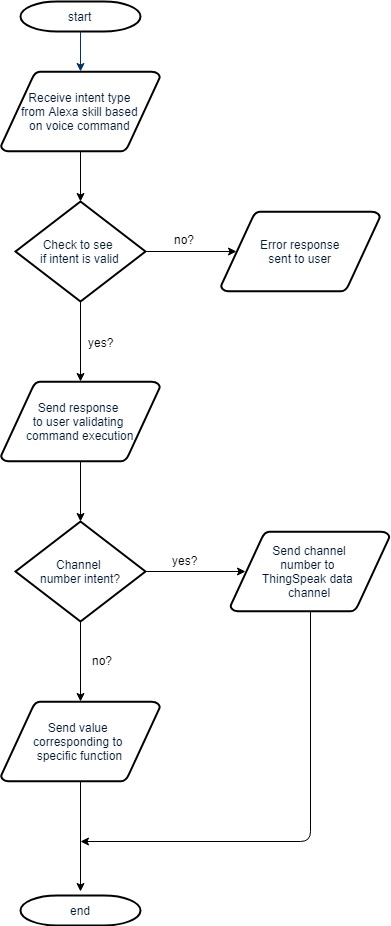


Fig. 12 Television Control Lambda Function

### Implementation of Task 4.3

Fig. 13 below shows the program flow of the temperature control Lambda function. The Alexa skill will map the intent with the utterance the user spoke. If the utterance matches one of the intents, that intent is valid, and a confirmation response will be sent back to the user. If the utterance does not match an intent, an error message will be sent back to the user. Again, the program from is similar to the light control function and television control function with one difference. If the intent was to change the AC to a specific value, such as 65 degrees or 70 degrees, the degree value will be sent to the ThingSpeak data channel. If the intent was to change the heat to a specific value, such as 70 degrees or 75 degrees, that value will be sent to the ThingSpeak data channel; similar to the AC. These two values will be sent to separate data fields within the channel, so the heat and AC values do not get mixed up. If the intent was neither to change the heat or AC setting, then the only other valid intent is to turn the unit off. This will send a value of 0 to the ThingSpeak data channel

A close up of a map

Description automatically generated

Fig. 13 Temperature Control Lambda Function.

## Implementation of Task 5.1.

Fig. 14 display the schematic design for the IR transmitter. The voltage source from the power jack supplying 5V will be powering this circuit with its power side to the Vin input on the NodeMCU and the ground side to reference ground. The NodeMCU has a built-in regulator to drop the voltage to 3.3V needed for the circuit. We used an 850nm IR LED because that is the wavelength that the LED operates. The NPN transistor is used in this project is because of that NPN transistors have slight greater conductivity than PNP transistors and for its ability to produce a current flow from the collector to the emitter, which is what we need for the schematic.

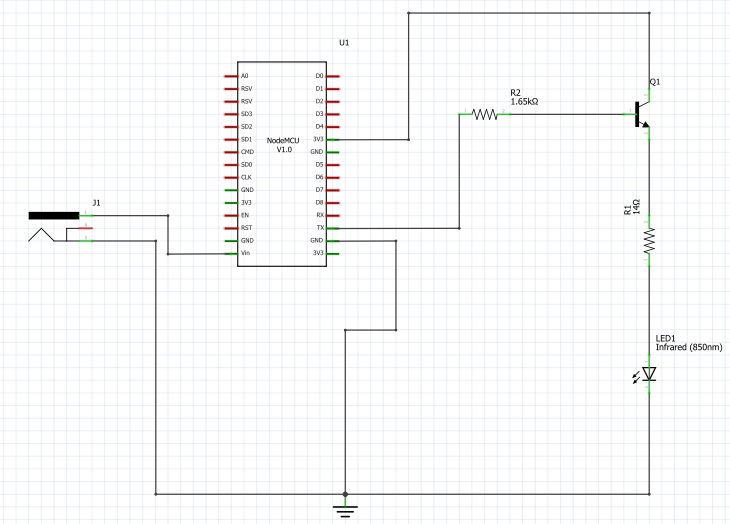


Fig. 14 IR Transmitter Schematic Design



### Implementation of Task 5.1 (cont.)

Fig. 15 displays the breadboard design for the IR transmitter. The voltage source from the power jack supplying 5V will be powering this circuit with its power side to the VIN input on the NodeMCU and the ground side to reference ground. The NodeMCU has a built-in voltage regulator to drop the voltage to 3.3V needed for the circuit. We used an 850nm IR LED because that is the wavelength that the LED operates. The NPN transistor is used in this project is because NPN transistors have slight greater conductivity than PNP transistors and for its ability to produce a current flow from the collector to the emitter, which is what we need for the breadboard design.

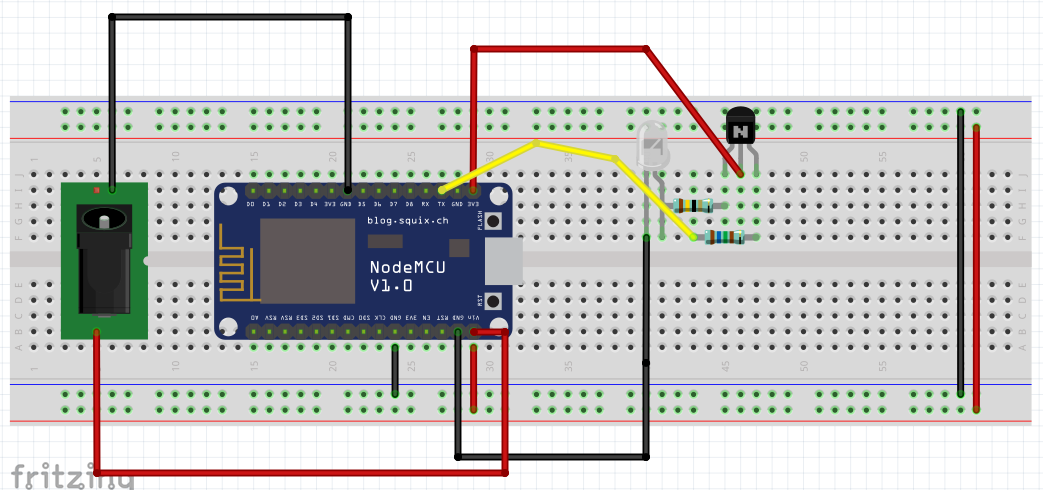


Fig. 15 IR Transmitter Breadboard Design.

## Implementation of Task 5.2.

Fig. 16 displays the calculations for the IR transmitter circuit. Including in these calculations are the voltage across each resistor, voltage across each junction on the transistor, current across each resistor, pulse current source voltage and voltage loop equation to verify the circuit loop equals to 0V. These calculations were necessary to ensure that the resistor value allow 50mA to through the IR LED for maximum efficiency.

Fig. 16 Calculations for IR Transmitter Circuit



### Implementation of Task 5.2 (cont.)

Fig. 17 shows the schematic of the IR Emitter. This schematic was created to simulate the output created by the IR emitter circuit. Using a pulse current of 15mA, which is the output from the NodeMCU pin and the 38kHz which is the frequency an IR transmitter is modulated at and a 1650Ω to stabilize the current at about 7.5mA. There is a current 42.124mA coming from the collector side of the transistor. The emitter current is the collector and base current fused together which will output a value of about 50mA which is showed below. The output must be at 50mA to ensure that the IR LED is operating at the maximum efficiency.

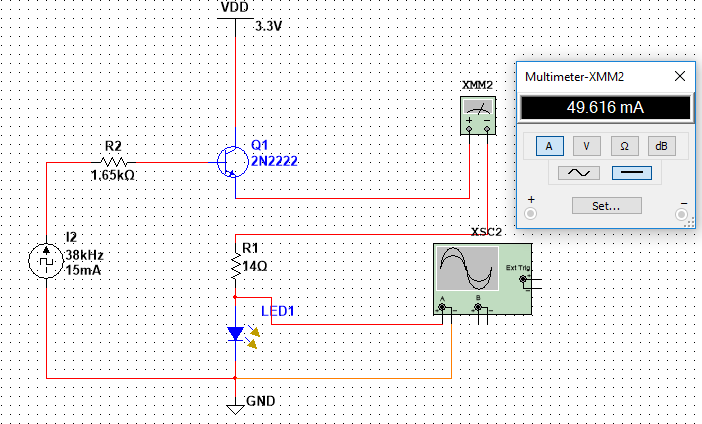


Fig. 17 Circuit Simulation for IR Transmitter

### Implementation of Task 5.2 (cont.)

Fig. 18 displays the output of the IR transmitter circuit. The view is from an oscilloscope display in the wave pattern due to the pulse current used in the circuit. Timebase and Channel A was adjusted to 50μs/Div and 200mV/Div respectively for a better view of the waveform. The modulated frequency of an IR transmitter is 38kHz. Based on the period of the wave which is 26.515μs, and using the equation f = 1/T, the simulated frequency comes out to be 37.7kHz which is approximately 38kHz.

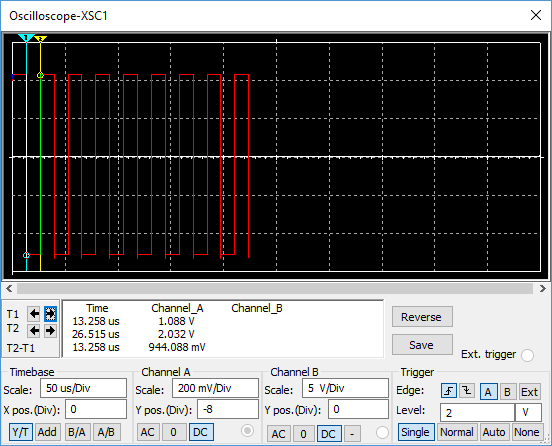


Fig. 18 Simulation Results

## Implementation of Task 5.3.

Fig. 19 displays the PCB design for the IR transmitter. The voltage source from the power jack supplying 5V will be powering this circuit with its power side to the VIN input on the NodeMCU and the ground side to reference ground. The NodeMCU has a built-in regulator to drop the voltage to 3.3V needed for the circuit. We used an 850nm IR LED because that is the wavelength that the LED operates. The NPN transistor is used in this project is because of that NPN transistors have slight greater conductivity than PNP transistors and for its ability to produce a current flow from the collector to the emitter, which is what we need for the PCB design. It is imperative that the tracings on the top layer are not intersecting as well as the bottom layer to ensure that it does not short circuit. Creating the PCB Board will require you to have a ground fill on the bottom layer for heat dissipation so the components do not overheat also the trace width should be equivalent to the amount of current flowing through that wire. The small wires (12 mil) indicate that there is a small current going them and the larger wire (24 mil) indicate there is a larger current going through that area. Fig. 20 displays the trace currents of wires and the required trace width of the wires for the PCB Design.

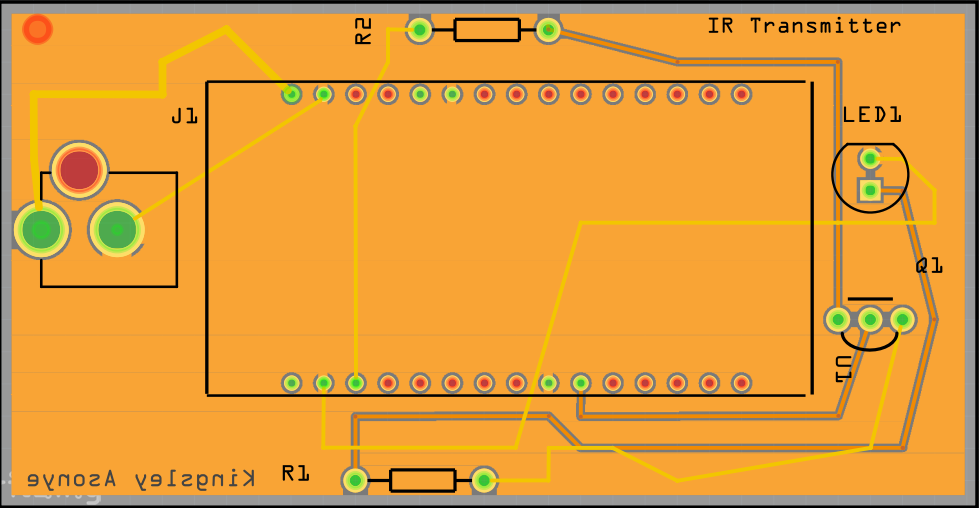


Fig. 19 PCB Design of Transmitter

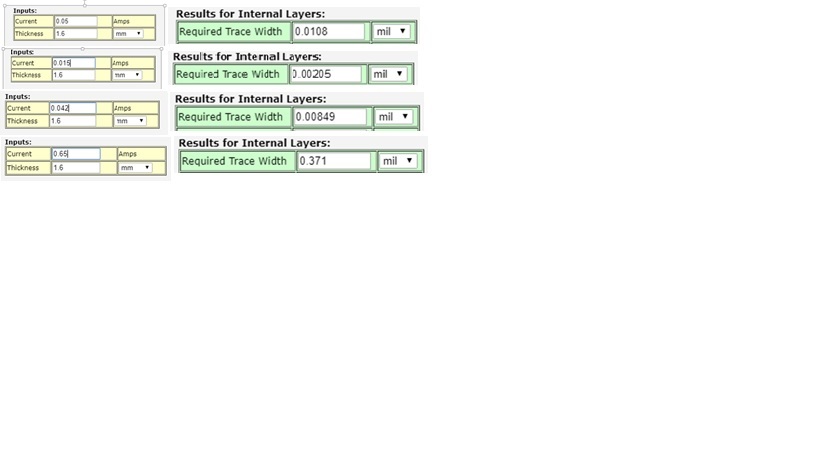


Fig. 20 Trace Current and Width for PCB Design

## Implementation of Task 5.4.

## Implementation of Task 5.5.

## Implementation of Task 5.6.

## Implementation of Task 6.1

Fig. 21 displays the schematic design for temperature control. The voltage source from the power jack supplying 9V 650mA will be powering this circuit with its positive side (PWR) to the Vcc input on the relay board and it’s also connected to the Vin input on the NodeMCU since the NodeMCU is built with a voltage regulator and the negative side (GND) to reference ground. Inputs IN1, IN2 and IN3 which will be controlling heat, A/C and fan both connected to digital inputs D1, D2 and D3 on the NodeMCU respectively outputting 15Ma on each pin. DATA and VDD input from the temperature sensor is connected to the digital input D6 and 3v3 on the NodeMCU respectively and of course the circuit has to be grounded.

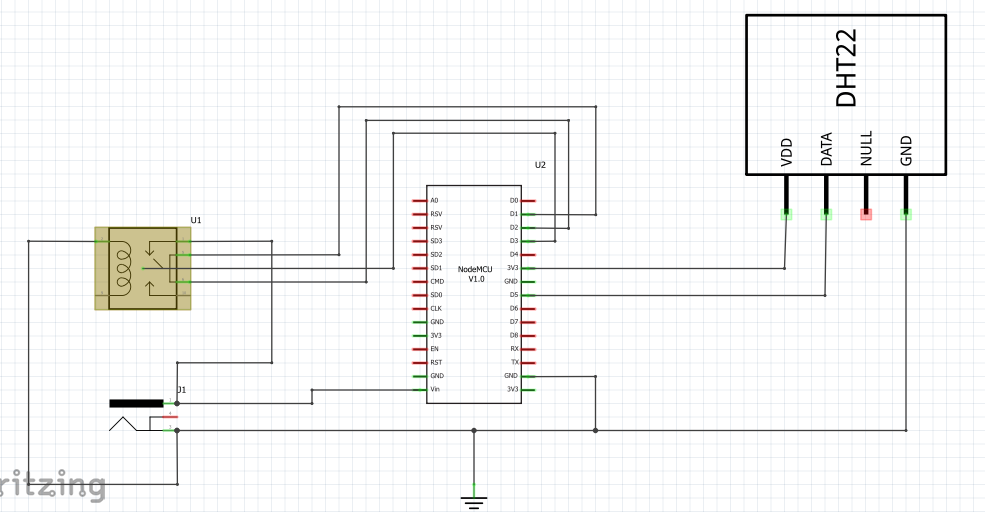


Fig. 21 Temperature Control Schematic



### Implementation of Task 6.1. (cont.)

Fig. 22 displays the schematic design for temperature control. The voltage source from the power jack supplying 9V 650mA will be powering this circuit with its positive side (PWR) to the Vcc input on the relay board and it’s also connected to the Vin input on the NodeMCU since the NodeMCU is built with a voltage regulator and the negative side (GND) to reference ground. Inputs IN1, IN2 and IN3 which will be controlling heat, A/C and fan both connected to digital inputs D1, D2 and D3 on the NodeMCU respectively outputting 15Ma on each pin. DATA and VDD input from the temperature sensor is connected to the digital input D6 and 3v3 on the NodeMCU respectively and of course the circuit has to be grounded.

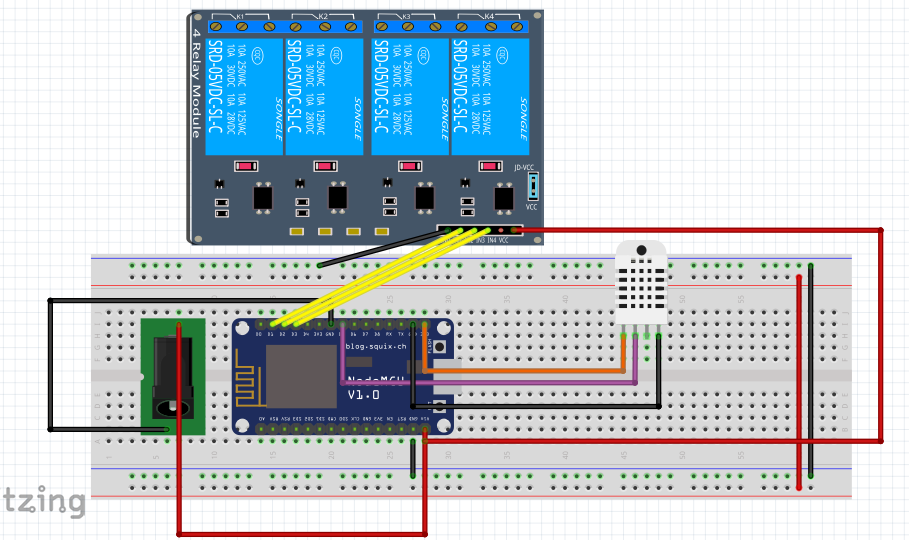


Fig. 22 Breadboard for Temperature Control Circuit

## Implementation of Task 6.2.

Fig. 23 displays the schematic design for temperature control. The voltage source from the power jack supplying 9V 650mA will be powering this circuit with its positive side (PWR) to the Vcc input on the relay board and it’s also connected to the Vin input on the NodeMCU since the NodeMCU is built with a voltage regulator and the negative side (GND) to reference ground. Inputs IN1, IN2 and IN3 which will be controlling heat, A/C and fan both connected to digital inputs D1, D2 and D3 on the NodeMCU respectively outputting 15Ma on each pin. DATA and VDD input from the temperature sensor is connected to the digital input D6 and 3v3 on the NodeMCU respectively and of course the circuit has to be grounded. It is imperative that the tracings on the top layer are not intersecting as well as the bottom layer to ensure that it does not short circuit. Creating the PCB Board will require you to have a ground fill on the bottom layer for heat dissipation so the components do not overheat also the trace width should be equivalent to the amount of current flowing through that wire. The small wires (12 mil) indicate that there is a small current going them and the larger wire (24 mil) indicate there is a larger current going through that area. Fig. 24 displays the trace currents of wires and the required trace width of the wires for the PCB Design.

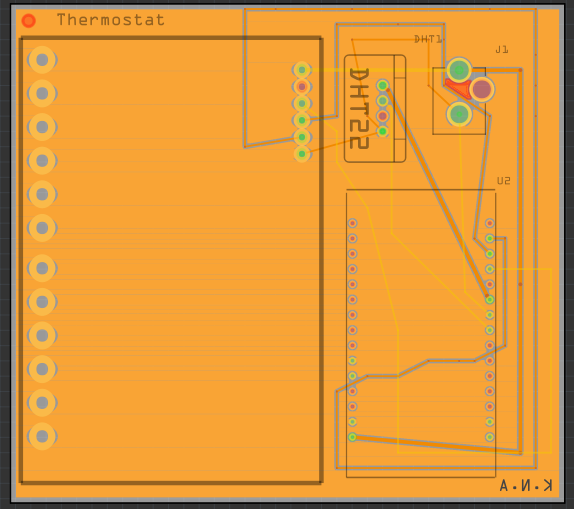


Fig. 23 PCB Design for Temperature Control Circuit

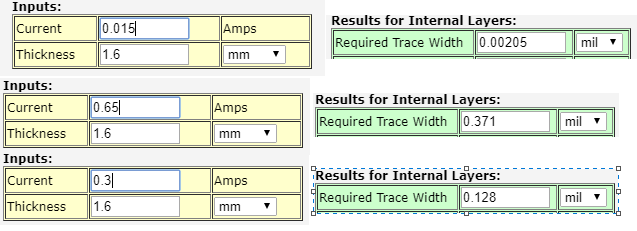


Fig. 24 Trace Current & Width for PCB Design

## Implementation of Task 6.3.

## Implementation of Task 6.4.

below shows the program flowchart for how the NodeMCU will be programmed for the temperature control. Starting off, the NodeMCU will attempt to connect to the Wi-Fi based on the credentials given such as the SSID and password. If the NodeMCU has not connected to the Wi-Fi, there will be another attempt, and this will repeat until the NodeMCU is connected. Once connected, the NodeMCU will make a HTTP GET request to the ThingSpeak data stream. The last result saved in the data stream is what will be saved as a string. The temperature sensor will read the current temperature in real time and depending on the result saved, the NodeMCU will know whether the air conditioner or heat is being requested. The LCD will also display the target temperature, current temperature, and thermostat mode. If the air conditioner was requested and the target temperature requested is lower than the current temperature being read, then the relay controlling the air conditioner and circulating unit will be switched on. Otherwise, they will be switched off. If the heat is being requested and the target temperature requested is higher that the current temperature being read, then the relay controlling the heating unit and circulating unit will be turned on. Otherwise, they will be switched off. If neither the air conditioner nor heat is being requested, then the only valid request is to turn the unit off. Therefore, the relay controlling the air conditioner, heat and circulating unit will be turned off.

A close up of a map

Description automatically generated

Fig. 25 NodeMCU Program Flowchart (Temperature Control)

## Implementation of Task 6.5.

(a) displays the CAD model of the temperature control unit. Nearing the top of the model is a rectangular cut out. This space was designed to be where the wires for the air conditioner, heating, and circulating unit will be connected to the relay board. Directly under the hole cut out for the wires are four platforms that the relay board will be mounted to. Each screw hole is dimensioned precisely for the relay board to allow all four screws to be screwed into these screw holes. Each hole is 3mm in diameter allowing a #5 screw to fit into each screw hole. Likewise, the same process was put into action for the mounting of the PCB design. The PCB was dimensioned precisely allowing all four corners to be screwed into the respective screw holes. Each screw hole has a diameter of 2.2mm allowing for a #3 screw to fit into each screw hole. On all 4 corners of the encasing are screw holes that will allow the encasing to be mounted to the wall using standard #6 (3.3mm) dry wall screws. On the left side of the encasing is where the hole for the power plug and ventilation holes are placed. The larger hole is for the power plug and will provide an opening for the PCB to be powered. This hole has a diameter of 9mm and was dimensioned precisely to ensure that the plug will connect directly to the PCB without any issue. The 12 smaller holes to the left are the holes to provide ventilation. Each hole has a diameter of 1.5mm. At the top of (a) are ridges that will allow for the lid of the encasing to slide onto to close. Fig. 27 (b) shows the lid for the encasing. The lid is designed to be slid onto the body to close and slid off to open. The cut out on the lid is dimensioned where the LCD will be placed.

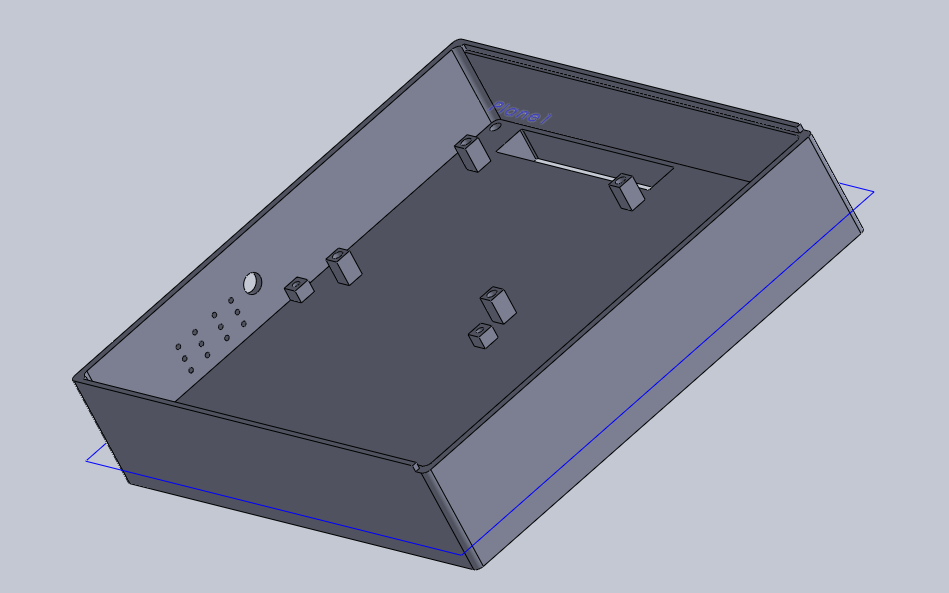


Fig. 26 (a) Temperature control unit encasing (Body)

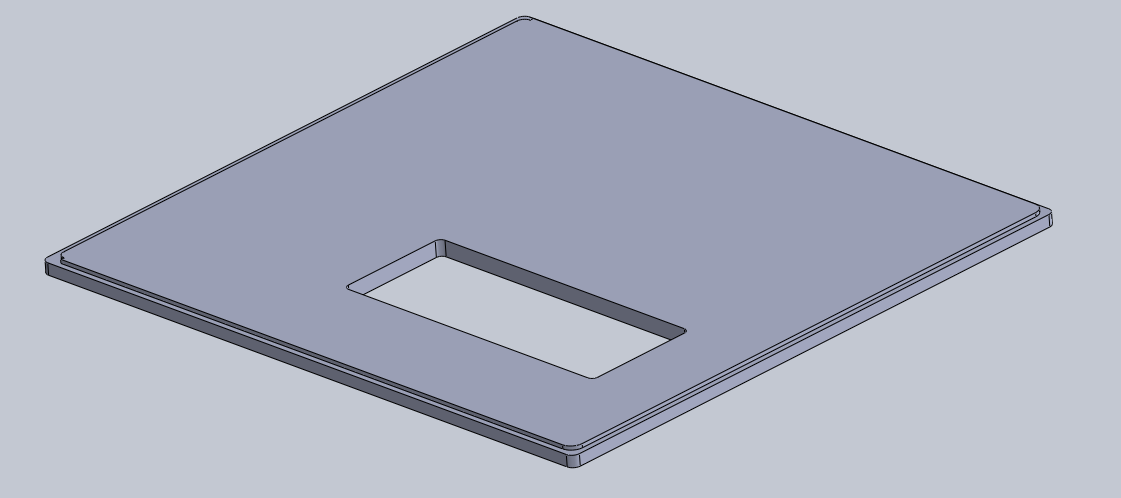


Fig. 26 (b) Temperature control unit encasing (Lid)

## Implementation of Task 7.1.

Fig. 25 displays the schematic for the light control circuit. Components used in this circuit are a 5V battery with its postive side connected to the Vin and 3v3 input on the NodeMCU to power the node but also bring down the voltage to 3.3V to power the circuit and its negative connected directly to the ground input on the relay board, the cathode end of the LED and reference ground. IN1 input on the relay board is connected to the digital input D1 on the NodeMCU. COM on the relay board is connected to the 3.3V on the NodeMCU. Normally Open or NO input on the relay board is connected to the 220Ω resistor. A resistor of that value is used is to reduce the current that is flowing through the LED so it wont burn out.

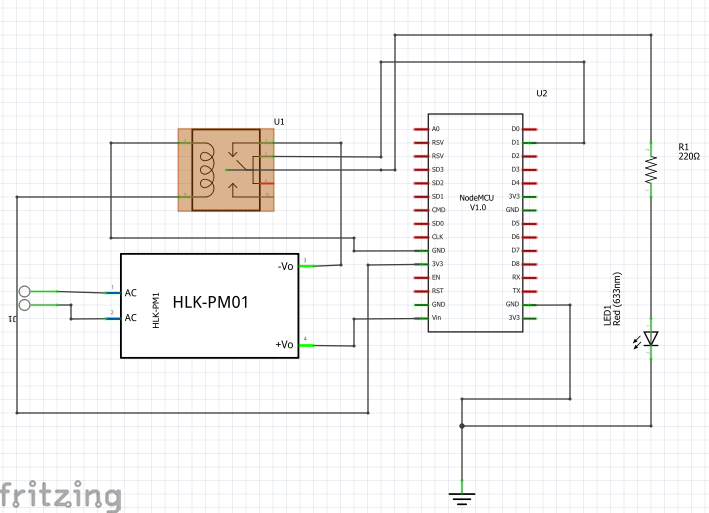


Fig. 27 Light Control Schematic



### Implementation of Task 7.1 (cont.)

Fig. 26 displays the schematic for the light control circuit. Components used in this circuit are a 5V battery with its postive side connected to the Vin and 3v3 input on the NodeMCU to power the node but also bring down the voltage to 3.3V to power the circuit and its negative connected directly to the ground input on the relay board, the cathode end of the LED and reference ground. IN1 input on the relay board is connected to the digital input D1 on the NodeMCU. COM on the relay board is connected to the 3.3V on the NodeMCU. Normally Open or NO input on the relay board is connected to the 220Ω resistor. A resistor of that value is used is to reduce the current that is flowing through the LED so it wont burn out.

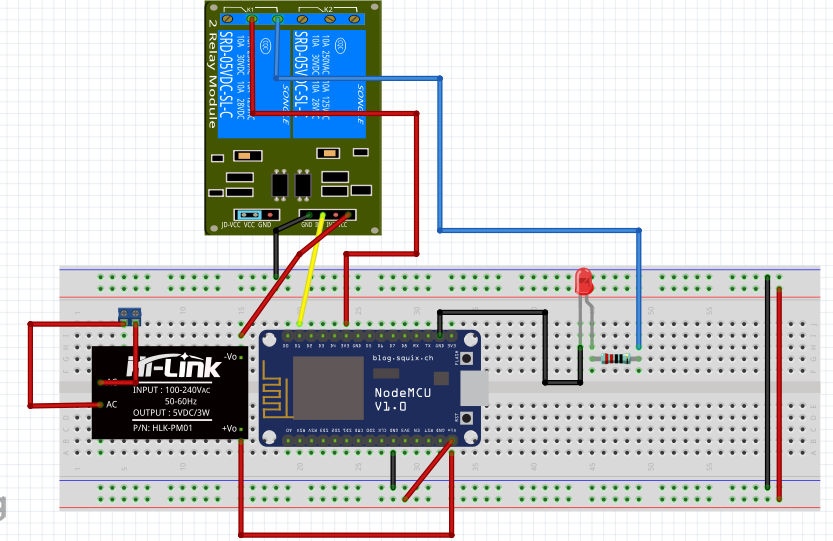


Fig. 28 Breadboard Design for Light Control

## Implementation of Task 7.2.

Fig. 27 displays the schematic design for light control. The voltage source from the 120VAC to 5VDC Converter 600mA will be powering this circuit with its negative Vout pin to the Vcc input on the relay board and its positive Vout pin to the VIN input on the NodeMCU since the NodeMCU is built with a voltage regulator. On the AC inputs which is coming from the wall, there is a screw terminal indicating that there is input power going into the circuit. Input IN1 on the relay board will be controlling the lights turning it on and off, connected to digital input D1 on the NodeMCU outputting 15mA on the pin. There will be a limiting resistor used to reduce to the current flowing to the light so it doesn’t burn out quickly. It is imperative that the tracings on the top layer are not intersecting as well as the bottom layer to ensure that it does not short circuit. Creating the PCB Board will require you to have a ground fill on the bottom layer for heat dissipation so the components do not overheat also the trace width should be equivalent to the amount of current flowing through that wire. The small wires (12 mil) indicate that there is a small current going them and the larger wire (24 mil) indicate there is a larger current going through that area. Fig. 28 displays the trace currents of wires and the required trace width of the wires for the PCB Design.

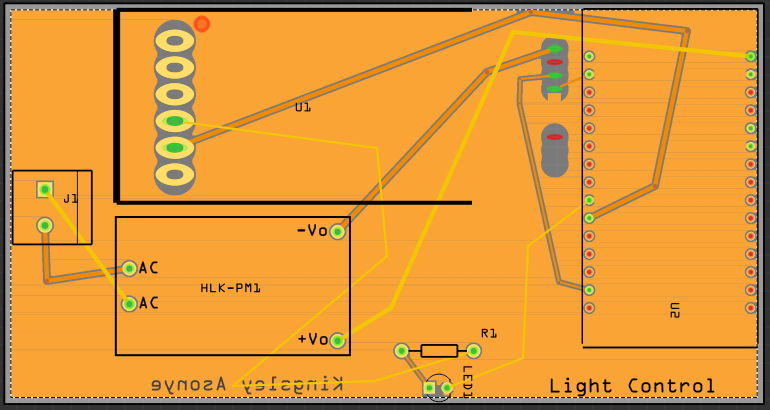


Fig. 29 PCB Design for Light Control

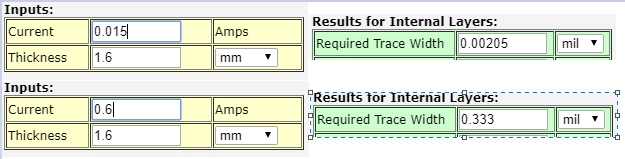


Fig. 30 Trace Current & Width for PCB Design

## Implementation of Task 7.3.

## Implementation of Task 7.4.

Fig. 31 below shows the program flowchart for how the NodeMCU will be programmed for the stand-alone light control. Starting off, the NodeMCU will attempt to connect to the Wi-Fi based on the credentials given such as the SSID and password. If the NodeMCU has not connected to the Wi-Fi, there will be another attempt, and this will repeat until the NodeMCU is connected. Once connected, the NodeMCU will make a HTTP GET request to the ThingSpeak data stream. The last result saved in the data stream is what will be saved as a string value. If that value is “1”, then the NodeMCU will close the relay switch allowing for the outlet to have power. If that value is “0”, then the NodeMCU will open the relay switch, cutting off power to the outlet.

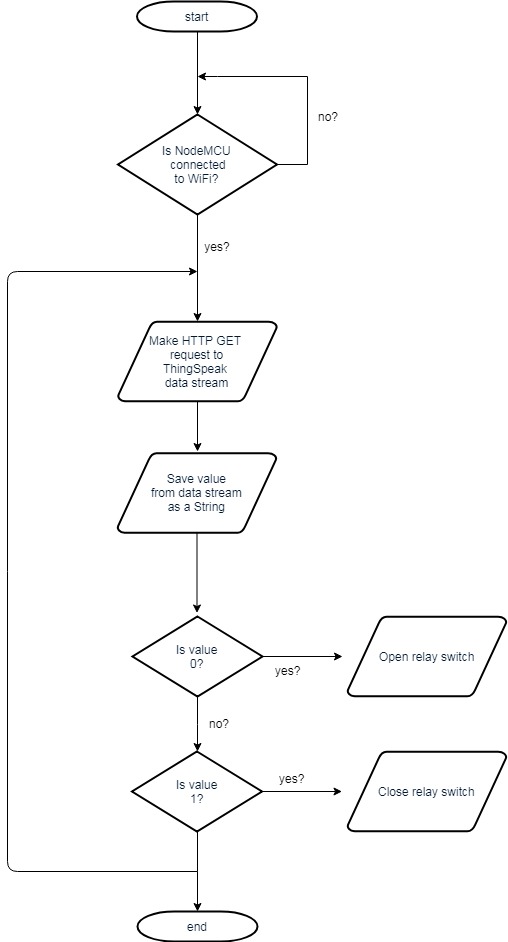


Fig. 31 NodeMCU Program Flowchart (Stand Alone Light)

## Implementation of Task 8.1.

## Implementation of Task 8.2.

## Implementation of Task 8.3.

## Implementation of Task 8.4.

Fig. 32 below shows the program flowchart for how the NodeMCU will be programmed for the stand-alone light control. Starting off, the NodeMCU will attempt to connect to the Wi-Fi based on the credentials given such as the SSID and password. If the NodeMCU has not connected to the Wi-Fi, there will be another attempt, and this will repeat until the NodeMCU is connected.

A close up of text on a black background

Description automatically generated

Fig. 32 NodeMCU Program Flowchart (Door Sensor)

## Implementation of Task 8.5.

## Implementation of Task 9.

## Implementation of Task 10.1.

## Implementation of Task 10.2.

1. Conclusion

During the first semester in senior design, we have made several strides in achieving the goal of developing a voice automated home automation system. By the end of senior design, our goal is to have a fully functioning system that will be not only control only stand-alone lights, but other general purpose on/off appliances, a remote that controls the television, and a thermostat that controls home temperature, all using voice commands. Throughout this project, we have had to overcome several challenges in order to make substantial progress and achieve success. The biggest problem to overcome was identifying a project suitable for this course. Due to this issue, we started our project during the midpoint of the semester. Due to this time lose, we had less time to work on the project. However, with good planning, diligence, and dedication, we were able to make progress with our project. To make up for additional ground that was lost, we plan to work diligently during winter break.

For the upcoming semester, we have mapped out a plan that will ensure this project is completed in totality. The next steps are to complete the remote-control design, temperature control design, and stand-alone light design as we have not completed all the subtasks for these designs. Additionally, we will be adding open/closing detecting functionality and that design will need to be done, as well as the power calculation and design verification. This experience has shown us the work required for real engineering design. Through this experience we have grown mentally and now understand what is required of engineers in the real world. We are eager to continue working on this project and complete the steps required to complete the project in totality.

1. Acknowledgement

Dr. Alvernon Walker – Engineering Professor

Justin Derickson – Electrical Engineering Student

Nathan Bane – Electrical Engineering Student

Christopher Blanks – Electrical Engineering Student

Joshua Orebiyi – Mechanical Engineering Student

Tochi Chukwu – Mechanical Engineering Student

1. Appendix

You can put reference info here, including: i) specs of components used in the system, ii) source code (must be here but not in the body text), iii) CAD figures, etc.

Component Specs

1. Specs of NodeMCU V1.0

* 3.3V Operated (can be USB Powered)
* Built-in ESP8266 for Wi-Fi capabilities.
* PWM, I2C capability
* Arduino IDE compatible
* 12-200mA working current
* 15mA output current per GPIO pin

1. Specs of Raspberry Pi 3

* CPU: 4x ARM Cortex-A53, 1.2GHz
* GPU: Broadcom VideoCore IV
* RAM: 1 GB LPDDR2 (900MHz)
* Bluetooth 4.1 Included
* MicroSD Card Storage
* 40 GPIO pins
* HDMI, 3.5mm audio jack, 4x USB 2.0, Ethernet, Camera Serial Interface, and Display Serial Interface

Source Code.

1. Source Code of Light Control Lambda Function

var https = require('https'); //we must include the https library to make HTTP GET requests

exports.handler = (event, context) => {

try {

if (event.session.new) {

// New Session

console.log("NEW SESSION");

}

switch (event.request.type) {

case "LaunchRequest":

// Launch Request

console.log(`LAUNCH REQUEST`);

context.succeed(

generateResponse(

buildSpeechletResponse("Welcome to the Light Control Skill, say turn light on or turn light off", true), //This is what Alexa will say without any intent phrase

{}

)

);

break;

case "IntentRequest": // This will begin the intent requests

// Intent Request

console.log(`INTENT REQUEST`);

switch(event.request.intent.name) {

case "TurnLightONIntent":

var endpoint = "https://api.thingspeak.com/update?api\_key=T71NLBD9RV7VXBX5&field1=1";

// Value "1" sent to thingspeak

https.get(endpoint, function (result) { // This is the get request to upload the data to ThingSpeak

console.log('Success, with: ' + result.statusCode);

context.succeed(generateResponse(buildSpeechletResponse("The light has been turned on.", true), // This is what Alexa will say if light is turned on

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "TurnLightOFFIntent": //the turn light off intent

var endpoint\_2 = "https://api.thingspeak.com/update?api\_key=T71NLBD9RV7VXBX5&field1=0"; /\* Value "0" sent to thingspeak \*/

https.get(endpoint\_2, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("The light has been turned off.", true), //This is what Alexa will say if light is turned off

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "SessionEndedRequest":

// Session Ended Request

console.log(`SESSION ENDED REQUEST`);

break;

default:

context.fail(`INVALID REQUEST TYPE: ${event.request.type}`);

}

}

}

catch(error) {context.fail(`Exception: ${error}`) }

// This is what builds the Alexa response

buildSpeechletResponse = (outputText, shouldEndSession) => {

return {

outputSpeech: {

type: "PlainText",

text: outputText

},

shouldEndSession: shouldEndSession

};

};

//This is what plays the Alexa response that was build

generateResponse = (speechletResponse, sessionAttributes) => {

return {

version: "1.0",

sessionAttributes: sessionAttributes,

response: speechletResponse

};

};

};

1. Source Code of Television Control Lambda Function

var https = require('https');

exports.handler = (event, context) => {

try {

if (event.session.new) {

// New Session

console.log("NEW SESSION"); //log this for debugging

}

switch (event.request.type) {

case "LaunchRequest":

// Launch Request

console.log(`LAUNCH REQUEST`);

context.succeed(

generateResponse(

buildSpeechletResponse("Welcome to the Remote Control skill. Tell me a command pertaining to your television.", true), //This is what Alexa will say without any intent phrase

)

);

break;

case "IntentRequest":

// Intent Request

console.log(`INTENT REQUEST`);

switch(event.request.intent.name) {

case "VolumeUp": //Volume up intent

var endpoint = "https://api.thingspeak.com/update?api\_key=XGCXWVZVO8YFWF8J&field1=100";

// Value "100" sent to thingspeak

// Update ThingSpeak data stream

https.get(endpoint, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(generateResponse(buildSpeechletResponse("Turning volume up.", true), // This is what Alexa will say when the volume is being turned up.

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "VolumeDown": //Volume down intent

var endpoint\_2 = "https://api.thingspeak.com/update?api\_key=XGCXWVZVO8YFWF8J&field1=101"; /\* Value "101" sent to thingspeak \*/

https.get(endpoint\_2, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("Turning volume down.", true), //This is what Alexa will say when the volume is being turned downg

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ChannelUp":

var endpoint\_3 = "https://api.thingspeak.com/update?api\_key=XGCXWVZVO8YFWF8J&field1=102"; /\* Value "102" sent to thingspeak \*/

https.get(endpoint\_3, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("Going one channel up.", true), //This is what Alexa will say when the channel will go up by one

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ChannelDown":

var endpoint\_4 = "https://api.thingspeak.com/update?api\_key=XGCXWVZVO8YFWF8J&field1=103"; /\* Value "103" sent to thingspeak \*/

https.get(endpoint\_4, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("Going one channel down.", true), //This is what Alexa will say when the channel will go down by one

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "Mute":

var endpoint\_5 = "https://api.thingspeak.com/update?api\_key=XGCXWVZVO8YFWF8J&field1=104"; /\* Value "104" sent to thingspeak \*/

https.get(endpoint\_5, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("Muting TV.", true), //This is what Alexa will say if television is being muted

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "On":

var endpoint\_6 = "https://api.thingspeak.com/update?api\_key=XGCXWVZVO8YFWF8J&field1=105"; /\* Value "105" sent to thingspeak \*/

https.get(endpoint\_6, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("Turning TV on.", true), //This is what Alexa will say if television is being turned on

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "Off":

var endpoint\_7 = "https://api.thingspeak.com/update?api\_key=XGCXWVZVO8YFWF8J&field1=106"; /\* Value "106" sent to thingspeak \*/

https.get(endpoint\_7, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("Turning TV off.", true), //This is what Alexa will say if television is turned off

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "Menu":

var endpoint\_8 = "https://api.thingspeak.com/update?api\_key=XGCXWVZVO8YFWF8J&field1=107"; /\* Value "107" sent to thingspeak \*/

https.get(endpoint\_8, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("Opening TV menu.", true), //This is what Alexa will say if the television menu is opened

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "PreviousChannel":

var endpoint\_9 = "https://api.thingspeak.com/update?api\_key=XGCXWVZVO8YFWF8J&field1=108"; /\* Value "108" sent to thingspeak \*/

https.get(endpoint\_9, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("Switching to previous channel", true), //This is what Alexa will say if the previous channel is being requested

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "Select":

var endpoint\_10 = "https://api.thingspeak.com/update?api\_key=XGCXWVZVO8YFWF8J&field1=109"; /\* Value "109" sent to thingspeak \*/

https.get(endpoint\_10, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("Selecting option.", true), //This is what Alexa will say if option is selected

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "Exit":

var endpoint\_11 = "https://api.thingspeak.com/update?api\_key=XGCXWVZVO8YFWF8J&field1=110"; /\* Value "110" sent to thingspeak \*/

https.get(endpoint\_11, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("Okay.", true), //This is what Alexa will say the command is to exit

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "GoDown":

var endpoint\_12 = "https://api.thingspeak.com/update?api\_key=XGCXWVZVO8YFWF8J&field1=111"; /\* Value "111" sent to thingspeak \*/

https.get(endpoint\_12, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("Going down.", true), //This is what Alexa will say if the option is to go down

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "GoUp":

var endpoint\_13 = "https://api.thingspeak.com/update?api\_key=XGCXWVZVO8YFWF8J&field1=112"; /\* Value "112" sent to thingspeak \*/

https.get(endpoint\_13, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("Going up.", true), //This is what Alexa will say if the option is to go up

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "GoLeft":

var endpoint\_14 = "https://api.thingspeak.com/update?api\_key=XGCXWVZVO8YFWF8J&field1=113"; /\* Value "113" sent to thingspeak \*/

https.get(endpoint\_14, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("Going left.", true), //This is what Alexa will say if the option is to go left

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "GoRight":

var endpoint\_15 = "https://api.thingspeak.com/update?api\_key=XGCXWVZVO8YFWF8J&field1=114"; /\* Value "114" sent to thingspeak \*/

https.get(endpoint\_15, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("Going right.", true), //This is what Alexa will say if the option is to go right

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ChannelSixteen": //Channel 16 intent

var endpoint\_16 = "https://api.thingspeak.com/update?api\_key=XGCXWVZVO8YFWF8J&field1=&field2=16"; /\* Value "16" sent to thingspeak \*/

https.get(endpoint\_16, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("Going to channel sixteen.", true), //This is what Alexa will say if channel is switched to 16.

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ChannelFour": //Channel 4 intent

var endpoint\_22 = "https://api.thingspeak.com/update?api\_key=XGCXWVZVO8YFWF8J&field1=&field2=4"; /\* Value "4" sent to thingspeak \*/

https.get(endpoint\_22, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("Going to channel four.", true), //This is what Alexa will say if channel is changed to 4.

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ChannelTwentyOne": //Channel 21 intent

var endpoint\_17 = "https://api.thingspeak.com/update?api\_key=XGCXWVZVO8YFWF8J&field1=&field2=21"; /\* Value "21" sent to thingspeak \*/

https.get(endpoint\_17, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("Going to channel twenty one.", true), //This is what Alexa will say if channel is switched to 21

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ChannelTwentyThree": //Channel 23 intent

var endpoint\_18 = "https://api.thingspeak.com/update?api\_key=XGCXWVZVO8YFWF8J&field1=&field2=23"; /\* Value "23" sent to thingspeak \*/

https.get(endpoint\_18, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("Going to channel twenty three.", true), //This is what Alexa will say if channel is changed to 23.

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ChannelThirtySix": //Channel 36 intent

var endpoint\_19 = "https://api.thingspeak.com/update?api\_key=XGCXWVZVO8YFWF8J&field1=&field2=36"; /\* Value "36" sent to thingspeak \*/

https.get(endpoint\_19, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("Going to channel thirty six.", true), //This is what Alexa will say if channel is changed to 36.

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ChannelFortySeven": //Channel 47 intent

var endpoint\_20 = "https://api.thingspeak.com/update?api\_key=XGCXWVZVO8YFWF8J&field1=&field2=47"; /\* Value "47" sent to thingspeak \*/

https.get(endpoint\_20, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("Going to channel forty seven.", true), //This is what Alexa will say if channel is changed to 47.

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ChannelSixtyThree": //Channel 63 intent

var endpoint\_21 = "https://api.thingspeak.com/update?api\_key=XGCXWVZVO8YFWF8J&field1=&field2=63"; /\* Value "63" sent to thingspeak \*/

https.get(endpoint\_21, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("Going to channel sixty three.", true), //This is what Alexa will say if channel is changed to 63.

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "SessionEndedRequest":

// Session Ended Request

console.log(`SESSION ENDED REQUEST`);

break;

default:

context.fail(`INVALID REQUEST TYPE: ${event.request.type}`);

}

}

}

catch(error) {context.fail(`Exception: ${error}`) }

// This is what will build the Alexa response

buildSpeechletResponse = (outputText, shouldEndSession) => {

return {

outputSpeech: {

type: "PlainText",

text: outputText

},

shouldEndSession: shouldEndSession

};

};

//This is what plays the Alexa response

generateResponse = (speechletResponse, sessionAttributes) => {

return {

version: "1.0",

sessionAttributes: sessionAttributes,

response: speechletResponse

};

};

};

3. Source Code of Temperature Control Lambda Function

var https = require('https');

exports.handler = (event, context) => {

try {

if (event.session.new) {

// New Session

console.log("NEW SESSION"); //log this for debugging

}

switch (event.request.type) {

case "LaunchRequest":

// Launch Request

console.log(`LAUNCH REQUEST`);

context.succeed(

generateResponse(

buildSpeechletResponse("Welcome to the Temperature Control Skill. Say Alexa tell thermostat to set heat or AC to a specific temperature.", true), //This is what Alexa will say without any intent phrase

{}

)

);

break;

case "IntentRequest":

// Intent Request

console.log(`INTENT REQUEST`);

switch(event.request.intent.name) {

case "HeatSixty": //Start the intent to set heat to 60 degrees

var endpoint = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=&field2=60";

https.get(endpoint, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(generateResponse(buildSpeechletResponse("The heat has been set to 60", true),

{}

)

);

})

.on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "HeatSixtyOne": //Start the intent to set heat to 61 degrees

var endpoint\_2 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=&field2=61";

https.get(endpoint\_2, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(generateResponse(buildSpeechletResponse("The heat has been set to 61", true),

{}

)

);

})

.on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "HeatSixtyTwo": //Start the intent to set heat to 62 degrees

var endpoint\_3 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=&field2=62";

https.get(endpoint\_3, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(generateResponse(buildSpeechletResponse("The heat has been set to 62", true),

{}

)

);

})

.on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "HeatSixtyThree": //Start the intent to set heat to 63 degrees

var endpoint\_4 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=&field2=63";

https.get(endpoint\_4, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(generateResponse(buildSpeechletResponse("The heat has been set to 63", true),

{}

)

);

})

.on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "HeatSixtyFour": //Start the intent to set heat to 64 degrees

var endpoint\_5 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=&field2=64";

https.get(endpoint\_5, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(generateResponse(buildSpeechletResponse("The heat has been set to 64", true),

{}

)

);

})

.on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "HeatSixtyFive": //Start the intent to set heat to 65 degrees

var endpoint\_6 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=&field2=65";

https.get(endpoint\_6, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(generateResponse(buildSpeechletResponse("The heat has been set to 65", true),

{}

)

);

})

.on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "HeatSixtySix": //Start the intent to set heat to 66 degrees

var endpoint\_7 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=&field2=66";

https.get(endpoint\_7, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(generateResponse(buildSpeechletResponse("The heat has been set to 66", true),

{}

)

);

})

.on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "HeatSixtySeven": //Start the intent to set heat to 67 degrees

var endpoint\_8 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=&field2=67";

https.get(endpoint\_8, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(generateResponse(buildSpeechletResponse("The heat has been set to 67", true),

{}

)

);

})

.on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "HeatSixtyEight": //Start the intent to set heat to 68 degrees

var endpoint\_9 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=&field2=68";

https.get(endpoint\_9, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(generateResponse(buildSpeechletResponse("The heat has been set to 68", true),

{}

)

);

})

.on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "HeatSixtyNine": //Start the intent to set heat to 69 degrees

var endpoint\_10 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=&field2=69";

https.get(endpoint\_10, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(generateResponse(buildSpeechletResponse("The heat has been set to 69", true),

{}

)

);

})

.on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "HeatSeventy": //Start the intent to set heat to 70 degrees

var endpoint\_11 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=&field2=70";

https.get(endpoint\_11, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(generateResponse(buildSpeechletResponse("The heat has been set to 70", true),

{}

)

);

})

.on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "HeatSeventyOne": //Start the intent to set heat to 71 degrees

var endpoint\_12 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=&field2=71";

https.get(endpoint\_12, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(generateResponse(buildSpeechletResponse("The heat has been set to 71", true),

{}

)

);

})

.on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "HeatSeventyTwo": //Start the intent to set heat to 72 degrees

var endpoint\_13 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=&field2=72";

https.get(endpoint\_13, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(generateResponse(buildSpeechletResponse("The heat has been set to 72", true),

{}

)

);

})

.on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "HeatSeventyThree": //Start the intent to set heat to 73 degrees

var endpoint\_14 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=&field2=73";

https.get(endpoint\_14, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(generateResponse(buildSpeechletResponse("The heat has been set to 73", true),

{}

)

);

})

.on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "HeatSeventyFour": //Start the intent to set heat to 74 degrees

var endpoint\_15 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=&field2=74";

https.get(endpoint\_15, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(generateResponse(buildSpeechletResponse("The heat has been set to 74", true),

{}

)

);

})

.on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "HeatSeventyFive": //Start the intent to set heat to 75 degrees

var endpoint\_16 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=&field2=75";

https.get(endpoint\_16, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(generateResponse(buildSpeechletResponse("The heat has been set to 75", true),

{}

)

);

})

.on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "HeatSeventySix": //Start the intent to set heat to 76 degrees

var endpoint\_17 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=&field2=76";

https.get(endpoint\_17, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(generateResponse(buildSpeechletResponse("The heat has been set to 76", true),

{}

)

);

})

.on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "HeatSeventySeven": //Start the intent to set heat to 77 degrees

var endpoint\_18 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=&field2=77";

https.get(endpoint\_18, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(generateResponse(buildSpeechletResponse("The heat has been set to 77", true),

{}

)

);

})

.on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "HeatSeventyEight": //Start the intent to set heat to 78 degrees

var endpoint\_19 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=&field2=78";

https.get(endpoint\_19, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(generateResponse(buildSpeechletResponse("The heat has been set to 78", true),

{}

)

);

})

.on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "HeatSeventyNine": //Start the intent to set heat to 79 degrees

var endpoint\_20 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=&field2=79";

https.get(endpoint\_20, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(generateResponse(buildSpeechletResponse("The heat has been set to 79", true),

{}

)

);

})

.on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "HeatEighty": //Start the intent to set heat to 80 degrees

var endpoint\_21 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=&field2=80";

https.get(endpoint\_21, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(generateResponse(buildSpeechletResponse("The heat has been set to 80", true),

{}

)

);

})

.on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "HeatEightyOne": //Start the intent to set heat to 81 degrees

var endpoint\_22 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=&field2=81";

https.get(endpoint\_22, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(generateResponse(buildSpeechletResponse("The heat has been set to 81", true),

{}

)

);

})

.on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "HeatEightyTwo": //Start the intent to set heat to 82 degrees

var endpoint\_23 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=&field2=82";

https.get(endpoint\_23, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(generateResponse(buildSpeechletResponse("The heat has been set to 82", true),

{}

)

);

})

.on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "HeatEightyThree": //Start the intent to set heat to 83 degrees

var endpoint\_24 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=&field2=83";

https.get(endpoint\_24, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(generateResponse(buildSpeechletResponse("The heat has been set to 83", true),

{}

)

);

})

.on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "HeatEightyFour": //Start the intent to set heat to 84 degrees

var endpoint\_25 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=&field2=84";

https.get(endpoint\_25, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(generateResponse(buildSpeechletResponse("The heat has been set to 84", true),

{}

)

);

})

.on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "HeatEightyFive": //Start the intent to set heat to 85 degrees

var endpoint\_26 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=&field2=85";

https.get(endpoint\_26, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(generateResponse(buildSpeechletResponse("The heat has been set to 85", true),

{}

)

);

})

.on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ACSixty": //Start the intent to set AC to 60 degrees

var endpoint\_27 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=60";

// The temperature value is sent to ThingSpeak

https.get(endpoint\_27, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("The AC has been set to 60", true), //This is what Alexa will say if value is set

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ACSixtyOne": //Start the intent to set AC to 61 degrees

var endpoint\_28 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=61";

// The temperature value is sent to ThingSpeak

https.get(endpoint\_28, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("The AC has been set to 61", true), //This is what Alexa will say if value is set

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ACSixtyTwo": //Start the intent to set AC to 62 degrees

var endpoint\_29 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=62";

// The temperature value is sent to ThingSpeak

https.get(endpoint\_29, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("The AC has been set to 62", true), //This is what Alexa will say if value is set

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ACSixtyThree": //Start the intent to set AC to 63 degrees

var endpoint\_30 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=63";

// The temperature value is sent to ThingSpeak

https.get(endpoint\_30, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("The AC has been set to 63", true), //This is what Alexa will say if value is set

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ACSixtyFour": //Start the intent to set AC to 64 degrees

var endpoint\_31 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=64";

// The temperature value is sent to ThingSpeak

https.get(endpoint\_31, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("The AC has been set to 64", true), //This is what Alexa will say if value is set

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ACSixtyFive": //Start the intent to set AC to 65 degrees

var endpoint\_32 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=65";

// The temperature value is sent to ThingSpeak

https.get(endpoint\_32, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("The AC has been set to 65", true), //This is what Alexa will say if value is set

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ACSixtySix": //Start the intent to set AC to 66 degrees

var endpoint\_33 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=66";

// The temperature value is sent to ThingSpeak

https.get(endpoint\_33, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("The AC has been set to 66", true), //This is what Alexa will say if value is set

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ACSixtySeven": //Start the intent to set AC to 67 degrees

var endpoint\_34 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=67";

// The temperature value is sent to ThingSpeak

https.get(endpoint\_34, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("The AC has been set to 67", true), //This is what Alexa will say if value is set

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ACSixtyEight": //Start the intent to set AC to 68 degrees

var endpoint\_35 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=68";

// The temperature value is sent to ThingSpeak

https.get(endpoint\_35, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("The AC has been set to 68", true), //This is what Alexa will say if value is set

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ACSixtyNine": //Start the intent to set AC to 69 degrees

var endpoint\_36 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=69";

// The temperature value is sent to ThingSpeak

https.get(endpoint\_36, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("The AC has been set to 69", true), //This is what Alexa will say if value is set

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ACSeventy": //Start the intent to set AC to 70 degrees

var endpoint\_37 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=70";

// The temperature value is sent to ThingSpeak

https.get(endpoint\_37, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("The AC has been set to 70", true), //This is what Alexa will say if value is set

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ACSeventyOne": //Start the intent to set AC to 71 degrees

var endpoint\_38 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=71";

// The temperature value is sent to ThingSpeak

https.get(endpoint\_38, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("The AC has been set to 71", true), //This is what Alexa will say if value is set

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ACSeventyTwo": //Start the intent to set AC to 72 degrees

var endpoint\_39 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=72";

// The temperature value is sent to ThingSpeak

https.get(endpoint\_39, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("The AC has been set to 72", true), //This is what Alexa will say if value is set

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ACSeventyThree": //Start the intent to set AC to 73 degrees

var endpoint\_40 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=73";

// The temperature value is sent to ThingSpeak

https.get(endpoint\_40, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("The AC has been set to 71", true), //This is what Alexa will say if value is set

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ACSeventyFour": //Start the intent to set AC to 74 degrees

var endpoint\_41 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=74";

// The temperature value is sent to ThingSpeak

https.get(endpoint\_41, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("The AC has been set to 74", true), //This is what Alexa will say if value is set

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ACSeventyFive": //Start the intent to set AC to 75 degrees

var endpoint\_42 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=75";

// The temperature value is sent to ThingSpeak

https.get(endpoint\_42, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("The AC has been set to 75", true), //This is what Alexa will say if value is set

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ACSeventySix": //Start the intent to set AC to 76 degrees

var endpoint\_43 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=76";

// The temperature value is sent to ThingSpeak

https.get(endpoint\_43, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("The AC has been set to 76", true), //This is what Alexa will say if value is set

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ACSeventySeven": //Start the intent to set AC to 77 degrees

var endpoint\_44 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=77";

// The temperature value is sent to ThingSpeak

https.get(endpoint\_44, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("The AC has been set to 77", true), //This is what Alexa will say if value is set

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ACSeventyEight": //Start the intent to set AC to 78 degrees

var endpoint\_45 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=78";

// The temperature value is sent to ThingSpeak

https.get(endpoint\_45, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("The AC has been set to 78", true), //This is what Alexa will say if value is set

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ACSeventyNine": //Start the intent to set AC to 79 degrees

var endpoint\_46 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=79";

// The temperature value is sent to ThingSpeak

https.get(endpoint\_46, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("The AC has been set to 79", true), //This is what Alexa will say if value is set

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ACEighty": //Start the intent to set AC to 80 degrees

var endpoint\_47 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=80";

// The temperature value is sent to ThingSpeak

https.get(endpoint\_47, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("The AC has been set to 80", true), //This is what Alexa will say if value is set

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ACEightyOne": //Start the intent to set AC to 81 degrees

var endpoint\_48 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=81";

// The temperature value is sent to ThingSpeak

https.get(endpoint\_48, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("The AC has been set to 81", true), //This is what Alexa will say if value is set

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ACEightyTwo": //Start the intent to set AC to 82 degrees

var endpoint\_49 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=82";

// The temperature value is sent to ThingSpeak

https.get(endpoint\_49, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("The AC has been set to 82", true), //This is what Alexa will say if value is set

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ACEightyThree": //Start the intent to set AC to 83 degrees

var endpoint\_50 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=83";

// The temperature value is sent to ThingSpeak

https.get(endpoint\_50, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("The AC has been set to 83", true), //This is what Alexa will say if value is set

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ACEightyFour": //Start the intent to set AC to 84 degrees

var endpoint\_51 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=84";

// The temperature value is sent to ThingSpeak

https.get(endpoint\_51, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("The AC has been set to 84", true), //This is what Alexa will say if value is set

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "ACEightyFive": //Start the intent to set AC to 85 degrees

var endpoint\_52 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=85";

// The temperature value is sent to ThingSpeak

https.get(endpoint\_52, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("The AC has been set to 85", true), //This is what Alexa will say if value is set

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "TurnOff": //Turn unit off intent

var endpoint\_53 = "https://api.thingspeak.com/update?api\_key=CI62VYHJJMC0PTTQ&field1=&field2=&field3=0";

// The value 0 is sent to ThingSpeak to represent unit is off

https.get(endpoint\_53, function (result) {

console.log('Success, with: ' + result.statusCode);

context.succeed(

generateResponse(buildSpeechletResponse("The thermostat unit has been turned off.", true), //This is what Alexa will say if value is set

{}

)

);

}).on('error', function (err) {

console.log('Error, with: ' + err.message);

context.done("Failed");

});

break;

case "SessionEndedRequest":

// Session Ended Request

console.log(`SESSION ENDED REQUEST`);

break;

default:

context.fail(`INVALID REQUEST TYPE: ${event.request.type}`);

}

}

}

catch(error) {context.fail(`Exception: ${error}`) }

// This is what builds the responce for Alexa

buildSpeechletResponse = (outputText, shouldEndSession) => {

return {

outputSpeech: {

type: "PlainText",

text: outputText

},

shouldEndSession: shouldEndSession

};

};

//This is what plays the responce for Alexa

generateResponse = (speechletResponse, sessionAttributes) => {

return {

version: "1.0",

sessionAttributes: sessionAttributes,

response: speechletResponse

};

};

};

1. REFERENCES

[1] "Smart Home - United States | Statista Market Forecast.” *Statista*, www.statista.com/outlook/279/109/smart-home/united-states#market-users.

[2] “What Is Home Automation?” *Smart Home Energy Saving Products | Smarthome*, www.smarthome.com/sc-what-is-home-automation.

[3] Willison, Karin. “7 Helpful Smart Home Devices for People With Disabilities.” *The Mighty*, 29 Aug. 2017, themighty.com/2017/08/smart-home-devices-for-people-with-disabilities/.

[4] “6 Ways Smart Home Technology Is Benefitting People with Disability.” *The Tipping Foundation*, 5 June 2018, www.tipping.org.au/6-ways-smart-home-technology-is-benefitting-people-with-disability/.

[5] Castle, Steve. “Survey: Majority Have Purchased LEDs, Want Smart Lighting.” *Digital Trends*, Digital Trends, 16 June 2015, 11:19, www.digitaltrends.com/home/survey-majority-have-purchased-led-light-bulbs/.

[6] McCracken, Harry. “Echo And Alexa Are Two Years Old. Here's What Amazon Has Learned So Far.” *Fast Company*, Fast Company, 7 Nov. 2017, www.fastcompany.com/3065179/echo-and-alexa-are-two-years-old-heres-what-amazon-has-learned-so-far.

[7] “What Is MQTT and How It Works.” *Random Nerd Tutorials*, randomnerdtutorials.com/what-is-mqtt-and-how-it-works/.

[8] “Raspberry Pi 3: Specs, Benchmarks & Testing.” Rotate Display 90º? - Raspberry Pi Forums, 15 Nov. 2018, www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/.