

**Senior Design Project**

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

ENGE476 Senior Design Project I

Department of Engineering and Aviation Sciences

University of Maryland, Eastern Shore

Ndubuisi Iwuala

Kingsley Asonye

Project Advisor, Dr. Lei Zhang

Submitted

November 26, 2018

Special notice for formatting:

Don’t add any blank rows. Use spacing or page breaks.

Every figure or picture inserted in the text, must be accompanied with the caption and detailed description.

Keep the formatting consistent (learn and apply the MS ‘Styles’ in formatting).

The final report didn’t meet the basic formatting requirements won’t be accepted.

List of Contents

[List of Contents 3](#_Toc530835250)

[List of Figures 5](#_Toc530835251)

[List of Tables 6](#_Toc530835252)

[Abstract 7](#_Toc530835253)

[1. Introduction 8](#_Toc530835254)

[1.1. Backgound/Motivation 8](#_Toc530835255)

[1.2. Objective 10](#_Toc530835256)

[1.3. Design Requirements 10](#_Toc530835257)

[1.4. Design Constraints 10](#_Toc530835258)

[1.5. Design Methods 10](#_Toc530835259)

[2. Project Description 11](#_Toc530835260)

[2.1. System Description 11](#_Toc530835263)

[2.2. System Diagram 11](#_Toc530835264)

[2.3. System Functions 12](#_Toc530835265)

[3. Implementation Plan 13](#_Toc530835266)

[3.1. Tasks 13](#_Toc530835268)

[3.2. Team Organization 14](#_Toc530835269)

[3.2.1.Responsibility of Team Member 1. 14](#_Toc530835270)

[3.2.2Responsibility of Team Member 2. 14](#_Toc530835271)

[3.3. Timeline/Milestones/Delivery Plan 15](#_Toc530835272)

[4. Implementation 16](#_Toc530835273)

[4.4. Implementation of Task 1. 16](#_Toc530835278)

[4.5. Implementation of Task 2. 17](#_Toc530835279)

[4.6. Implementation of Task 2.1. 17](#_Toc530835280)

[4.7. Implementation of Task 2.2. 19](#_Toc530835281)

[4.8. Implemenatation of Task 2.3. 20](#_Toc530835282)

[4.9. Implemenatation of Task 3. 21](#_Toc530835283)

[4.10. Implementation of Task 4. 24](#_Toc530835284)

[4.11. Implementation of Task 4.1. 25](#_Toc530835285)

[4.12. Implementation of Task 4.2. 25](#_Toc530835286)

[4.13. Implementation of Task 4.3 26](#_Toc530835287)

[4.14. Implementation of Task 5.1. 27](#_Toc530835288)

[4.15. Implementation of Task 5.2. 28](#_Toc530835289)

[4.16. Implentation of Task 6. 29](#_Toc530835290)

[4.17. Implementation of Task 7. 29](#_Toc530835291)

[4.18. Implementation of Task 8. 29](#_Toc530835292)

[4.19. Implementation of Task 9. 29](#_Toc530835293)

[4.20. Implementation of Task 10. 29](#_Toc530835294)

[5. Conclusion. 29](#_Toc530835295)

[6. Acknowledgement 30](#_Toc530835296)

[7. Appendix 31](#_Toc530835297)

[A. Component Specs 31](#_Toc530835298)

[1. Specs of Arduino Due 31](#_Toc530835299)

[2. Specs of Raspberry Pi 31](#_Toc530835300)

[B. Source Code. 31](#_Toc530835301)

[1. Source Code of Graphic User Interface 31](#_Toc530835302)

[2. Source Code of Robotic Arm 31](#_Toc530835303)

[8. REFERENCES 32](#_Toc530835304)

List of Figures

[Figure 1. System Descrption 9](#_Toc453191241)

List of Tables

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.

1. Introduction

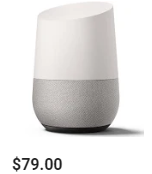
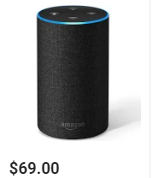
## Backgound/Motivation

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, is consisted 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. 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.



1. Here are images of the Amazon Echo 2nd generation (left) and the Google Home (right). Each provide smart home functionality and capability to create additional functions that can serve a greater purpose in individual.

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 our project, we will be using the Alexa services and IoT devices 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 between $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. Those appliances will be the television, light, and temperature.

## Design Requirements

1. Temperature differential or tolerance should be ±3°F
2. Will use display to show current & target temperature, heat, cool and fan for temperature control
3. Lights, television, and temperature will be controlled by voice
4. IR transmitter will be recognizable by the television
5. Thermostat components will be encased

## Design Constraints

1. Internet connectivity is required due to fact of the use of IoT devices in this project
2. $100 budget

## Design Methods

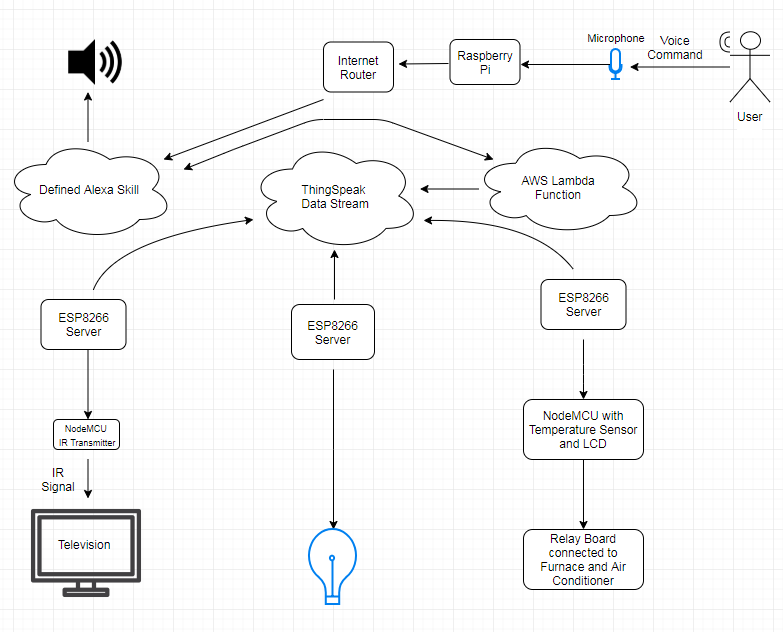
The first step is to implement Amazon Voice Services (AVS) Device SDK onto the raspberry pi. Once this is done, with a microphone connected to the raspberry pi, the pi can become an alternative to the Amazon Echo. The second step is for thevoice command to be sent through router and reach AWS via Wi-Fi. This is done in conjunction with the third step. The third step is for theAlexa Skills to take the voice command and based on our defined values by the user. That command will act as an enabler for the specific Alexa skill. The fourth step is to create an AWS Lambda function. This will serve as an output for the Alexa skill in JSON format. The fifth step is to create a ThingSpeak data channel in the cloud that will receive an output from the Lambda function. Therefore, the Lambda functions serves two purposes; send the response to the Alexa skill as well as send data to a data channel in the cloud. The sixth step is to retrieve the data that is in the ThingSpeak data channel. The result will be in JSON (JavaScript Object Notation) representation and will need to be parsed to receive the data we need. The final step is to execute the command by taking that data and matching it with a function.

1. Project Description

## System Description

This voice automated system is primarily controlled by the microprocessor Raspberry Pi. In our project the Raspberry takes the role of an Amazon Echo. The user will say 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. An Alexa skill is similar to an app. The skill will run depending on the command that is being spoken. The Alexa skill will call the Lambda function that it is linked to. This function will generate a voice 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. The NodeMCU (ESP8266) will pull the data from this channel and execute a command based on the data it reads.

## System Diagram



1. System Diagram

## C:\Users\sutemp609\Downloads\Industrialized State Graph (1).png System Functions

1. State graph representation of our system
2. Implementation Plan

## Tasks

* Task 1. Create AVS Prototype on Raspberry Pi
* Task 2. Program Alexa skills for each function
  + 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. Create a data channel for the television, temperature, and lights.
* Task 4. Program AWS Lambda Function
  + 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
  + Subtask 4.4. Test and troubleshoot
* Task 5. IR Emitter Design
  + Subtask 5.1. Create schematic for IR transmitter
  + Subtask 5.2. Circuit design simulation
  + Subtask 5.3. PCB design of IR transmitter
  + Subtask 5.4. Solder components to PCB
* Task 6. Temperature Control Design
  + Subtask 6.1. Create schematic for temperature control
  + Subtask 6.2. PCB design of temperature control
  + Subtask 6.3. Solder components to PCB
* Task 7. Light Control Design
  + Subtask 7.1. Create schematic for Light Control
  + Subtask 7.2. PCB design of Light Control
  + Subtask 7.3. Solder components to PCB
* Task 8. Determine efficient method to power each device
* Task 9. Program NodeMCU compatible IR emitter
  + Subtask 9.1. Receive codes from remote
  + Subtask 9.2. Program NodeMCU to read data from ThingSpeak and send appropriate IR signal.
  + Subtask 9.3. Test and Troubleshoot
* Task 10. Program NodeMCU as thermostat
  + Subtask 10.1. Program NodeMCU to read data from ThingSpeak and control temperature
  + Subtask 10.2. Test and Troubleshoot
* Task 11. Program NodeMCU for light control
  + Subtask 11.1. Program NodeMCU to read data from ThingSpeak and control relay switch
  + Subtask 11.2. Test and Troubleshoot
* Task 12. Design encasing for thermostat
  + Subtask 12.1. Verify dimensions
  + Subtask 12.2. CAD model of encasing
  + Subtask 12.3. 3-D print encasing

## Team Organization

Team Member 1: Ndubuisi Iwuala (Computer Engineering)

Team Member 2: Kingsley Asonye (Computer Engineering)

### 3.2.1.Responsibility of Team Member 1.

Task 1: Create AVS Prototype on Raspberry Pi

Task 2: Subtask 2.1. Program Alexa Skill for function light control

Task 2: Subtask 2.2. Program Alexa Skill for television remote control

Task 2: Subtask 2.3. Program Alexa Skill for thermostat control function

Task 3: Create a data channel for the television, temperature, and light

Task 4: Subtask 4.1. Program Lambda functions for light control

Task 4: Subtask 4.2. Program Lambda function for television control

Task 4: Subtask 4.3. Program Lambda function for temperature control

Task 4. Subtask 4.4. Test and troubleshoot

Task 4: Subtask 4.1. Create a data channel for the television, temperature, and light

Task 5: Subtask 5.4. Solder components to PCB

Task 8: Determine efficient method to power each device

Task 9: Subtask 9.2. Program NodeMCU to read data from ThingSpeak and send appropriate IR signal

Task 10: Subtask 10.1. Program NodeMCU to read data from ThingSpeak and control temperature

Task 11: Subtask 11.1. Program NodeMCU to read data from ThingSpeak and control relay switch

Task 12: Subtask 12.1: Verify dimensions

### 3.2.2Responsibility of Team Member 2.

Task 5: Subtask 5.1. Create schematic for IR transmitter

Task 5: Subtask 5.2. Circuit design simulation

Task 5: Subtask 5.3. PCB design of IR Transmitter

Task 5: Subtask 5.4. Solder components to PCB

Task 6: Subtask 6.1. Create schematic for temperature control circuit

Task 6: Subtask 6.2. PCB design of temperature control circuit

Task 6: Subtask 6.3. Solder components to PCB

Task 7: Subtask 7.1. Create schematic for light control circuit

Task 7: Subtask 7.2. PCB design of light control circuit

Task 7: Subtask 7.3. Solder components to PCB

Task 8: Determine efficient method to power each device

Task 9: Subtask 7.1. Receive codes from remote

Task 9: Subtask 7.3. Solder NodeMCU and IR emitter to solder board

Task 9: Subtask 7.4. Test and Troubleshoot

Task 10: Subtask 8.2. Solder LCD display, sensor, and NodeMCU to breadboard

Task 10: Subtask 8.3. Test and Troubleshoot

Task 11: Subtask 9.2. Solder components on breadboard

Task 11: Subtask 9.3. Test and Troubleshoot

Task 12: Subtask 10.2: CAD Model

Task 12: Subtask 10.3: 3-D print encasing

## Timeline/Milestones/Delivery Plan

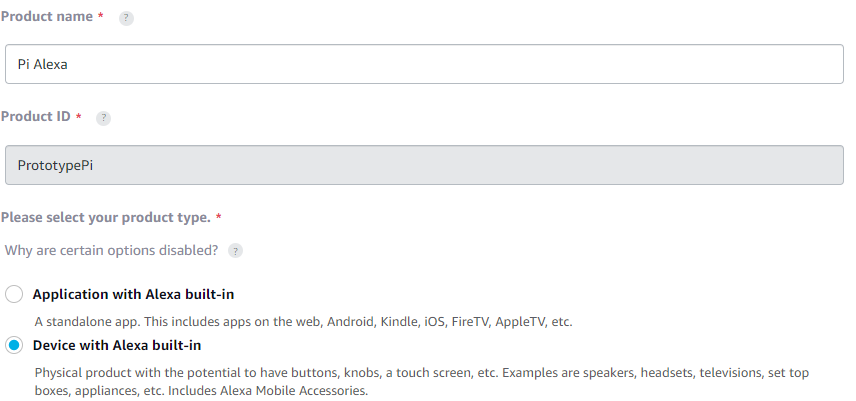
1. Project Timeline and Delivery Plan

|  |  |  |  |
| --- | --- | --- | --- |
| Time | Task | Comments | Responsible Personnel |
| 9 weeks | Project Planning | N/A | Ndubuisi/Kingsley |
| 1 week | Begin Task 1 | Implement SDK from AVS | Ndubuisi |
| 1.5 weeks | Begin Task 2 | Program Alexa Skill for each function | Ndubuisi |
| 3 days | Task 2.1 | Program Alexa Skill for light control | Ndubuisi |
| 4 days | Task 2.2 | Program Alexa Skill for television remote control function | Ndubuisi |
| 3 days | Task 2.3 | Program Alexa Skill for thermostat control function | Ndubuisi |
| 1 week | Begin Task 3 | Create a data channel for the television, temperature, and lights | Ndubuisi |
| 3.5 weeks | Begin Task 4 | Program AWS Lambda Function | Ndubuisi |
| 1.5 weeks | Task 4.1 | Program Lambda functions for light control | Ndubuisi |
| 1 week | Task 4.2 | Program Lambda function for television control | Ndubuisi |
| 1 week | Task 4.3 | Program Lambda function for temperature control | Ndubuisi |
| 4 weeks | Begin Task 5 | IR Emitter Design | Kingsley |
| 1 week | Task 5.1 | Create schematic for IR transmitter | Kingsley |
| 1 week | Task 5.2 | Circuit design simulation | Kingsley |
| 1 week | Task 5.3 | PCB design of IR transmitter circuit | Kingsley |
| 1 week | Task 5.4 | Solder components to PCB | Kingsley |
| 4 weeks | Begin Task 6 | Temperature Control Design | Kingsley |
| 1 week | Task 6.1 | Create schematic for Temperature Control circuit | Kingsley |
| 1 week | Task 6.2 | PCB design of Temperature Control circuit | Kingsley |
| 1 week | Task 6.3 | Solder components to PCB | Kingsley |
| 4 weeks | Begin Task 7 | Light Control Design | Kingsley |
| 1 week | Task 7.1 | Create schematic for Light Control circuit | Kingsley |
| 1 week | Task 7.2 | PCB design of Light Control circuit | Kingsley |
| 1 week | Task 7.3 | Solder components to PCB | Kingsley |
| 2 weeks | Begin Task 8 | NodeMCU Power Calculation | Ndubuisi/Kingsley |
| 3.5 weeks | Begin Task 9 | Program NodeMCU compatible IR emitter | Ndubuisi/Kingsley |
| 1 week | Task 9.1 | Receive codes from remote | Kingsley |
| 1 week | Task 9.2 | Program NodeMCU to read data from ThingSpeak and send appropriate IR signal | Ndubuisi |
| 1.5 week | Task 9.3 | Solder NodeMCU and IR emitter to PCB | Kingsley |
| 2.5 weeks | Begin Task 10 | Program NodeMCU as thermostat | Ndubuisi/Kingsley |
| 1 week | Task 10.1 | Program NodeMCU to read data from ThingSpeak and control temperature | Ndubuisi |
| 1.5 weeks | Task 10.2 | Solder LCD display, sensor, and NodeMCU to PCB | Kingsley |
| 2.5 weeks | Begin Task 11 | Program NodeMDU for light control | Ndubuisi/Kingsley |
| 1 week | Task 11.1 | Program NodeMCU to read data from ThingSpeak and control relay switch | Ndubuisi |
| 1.5 weeks | Task 11.2 | Solder components on PCB | Kingsley |
| 2.5 weeks | Begin Task 12 | Design encasing for thermostat | Ndubuisi/Kingsley |
| 1 week | Task 12.1 | Verify dimensions | Ndubuisi |
| 1 week | Task 12.2 | CAD model of encasing | Kingsley |
| 4 days | Task 12.3 | 3-D print encasing | Kingsley |

1. Implementation

## Implementation of Task 1.

To begin this project, we first had to install the appropriate image into the Raspberry Pi in order to turn it on. This involved us downloading and burning the Raspbian image called Raspbian Stretch onto a microSD card. Once this was complete, we began to implement the Alexa SDK. First, we started by registering the Raspberry pi as a new product on the Amazon Alexa Developer Console. Next, we followed the directions provided by Amazon Alexa GitHub. However, this posed a couple problems. Because the instructions were given on an image called Raspbian Jessie, an older version of the Raspbian series, parts of the installation failed. After some research, we were able to find a solution on acquiring Alexa onto the pi. We came across a man named Henry Mendez who managed to reduce the steps taken to run the SDK to scripts that needed to be entered into the pi. Using his GitHub and running the necessary scripts, we were able to install and run the Alexa SDK onto the pi.



1. Here is part of the registration process needed to prototype the Raspberry Pi. We must give our product a name and ID. This ID is used to identify our our pi during authorization. We also must specify the type of product we are authorizing. Since we want Alexa on our device, we select device with Alexa built-in.

A screenshot of a social media post

Description automatically generated

1. Here we must create a security profile for our device. This profile needs a name, description, and an Amazon generated security profile ID and client ID. The profile ID is what is identifies our security profile in Amazon services. Our client ID is what is used during the setup process for Alexa Voice Services (AVS) for authorization.

A screenshot of a computer

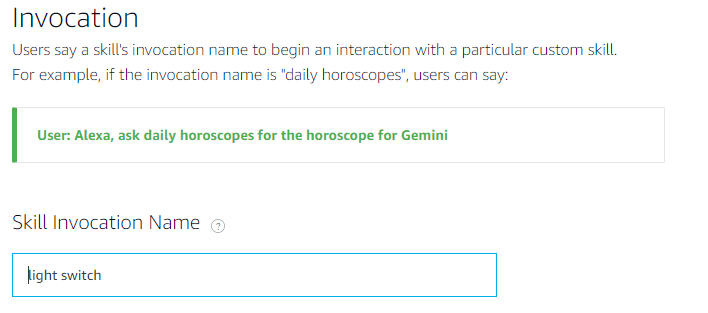
Description automatically generated

1. Here is what the AVS porgram looks like running on our Raspberry Pi. The application will connect to the service, authorize, and Alexa will then wait for the wake word.

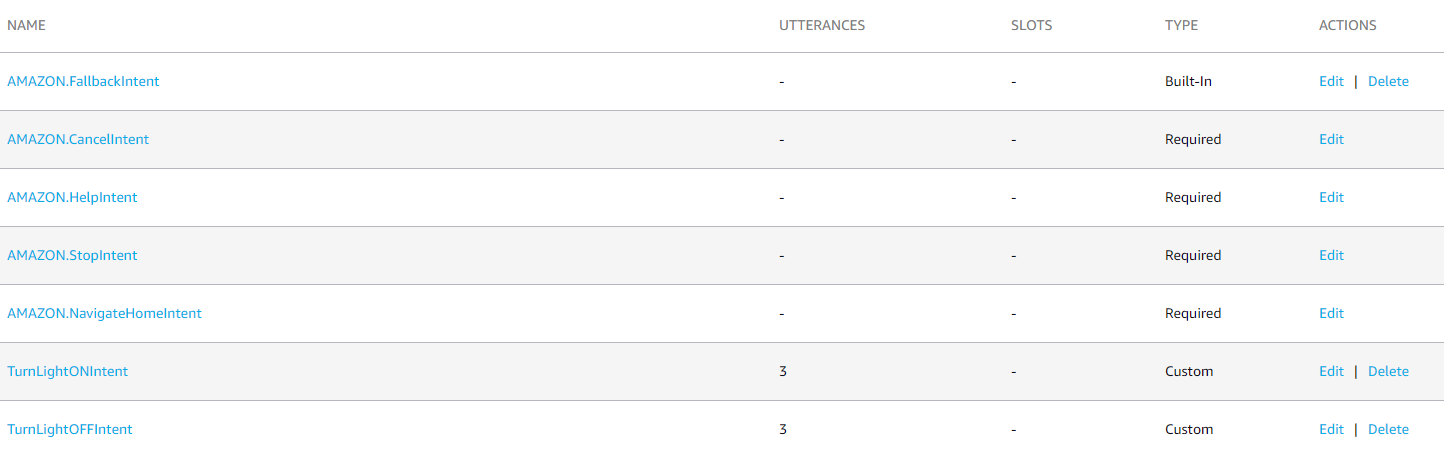
## Implementation of Task 2.

Within the Amazon Developer Console, we were able to create and save our custom-made Alexa skill. This can either be done by using the interaction model Amazon provides or writing an JSON schema code from scratch. We don’t have any prior knowledge in using JSON code, so we used the interaction model to form our code. In order for our skill to be built correctly with no errors, we first need to create an invocation name. This is the name that the user will call in order to call that Alexa skill. For example, if an invocation name was “light switch”, the user would have to tell Alexa to open light switch or do some action with light switch. This is how Alexa will know what skill is being called. Next, we created our intents. There is no limit to the amount of intents you can have for a particular skill. Intents are functions within our skill that our lambda function will call in order for an output to be generated for our skill. In other words, intents represent the action that fulfills a user’s spoken request. Next, we provide sample utterances for each intent. These utterances map to our intents and are the phrases the user must say to evoke each intent. These are the basics needed for your skill to be created. Now, the user will be able to say the invocation name as well as the sample utterance in order to start the skill. For example, the user can say “Alexa tell light switch to turn light on.” The invocation name is “light switch” and the utterance is “turn light on.” 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 response from the events generated from our Alexa skill.

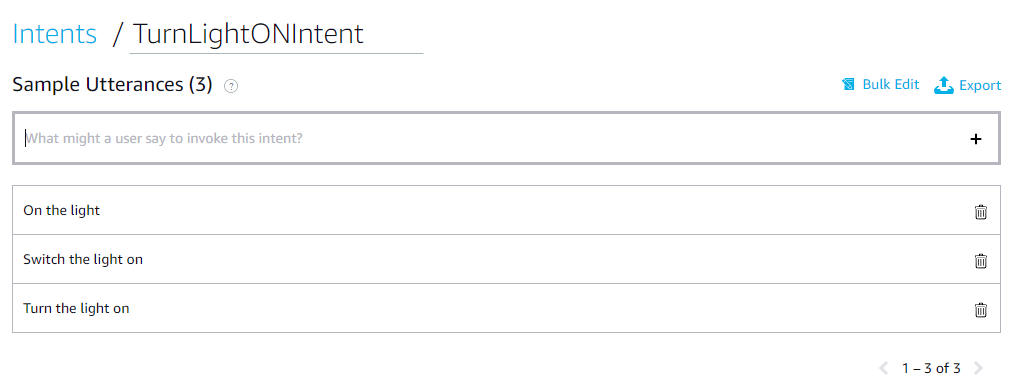
## Implementation of Task 2.1.



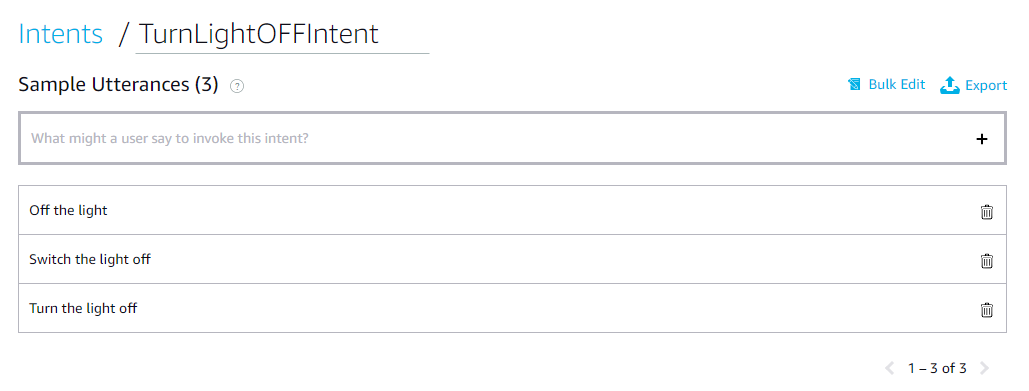
1. Here is where we define our skill invocation name. This is the light control portion.



1. Posted above are the list of intents. As you can see there are four required intents that are needed for the skill to operate. At the very bottom are our custom intents. TurnLightONIntent and TurnLightOFFIntent.

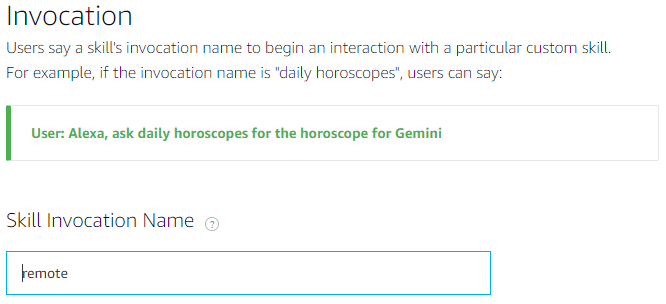


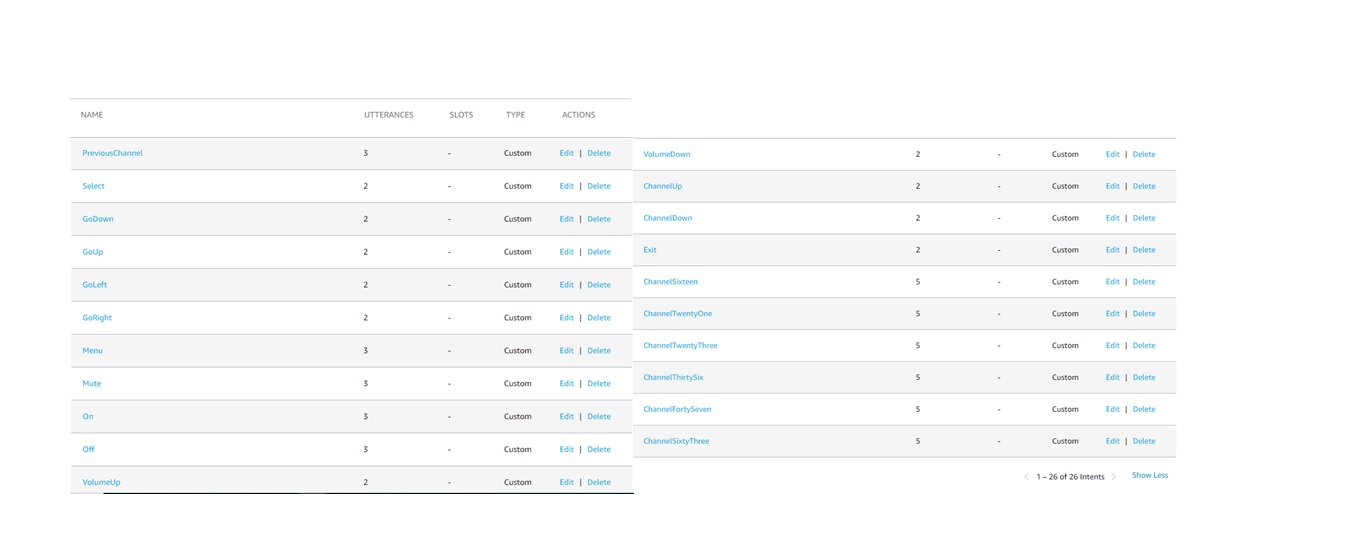
1. Utterances for turning on the light. These utterances are mapped directly to the TurnONLightIntent. If the user says these phrases in their voice command, this intent will be called. For example “Alexa, tell light switch to turn the light on.”



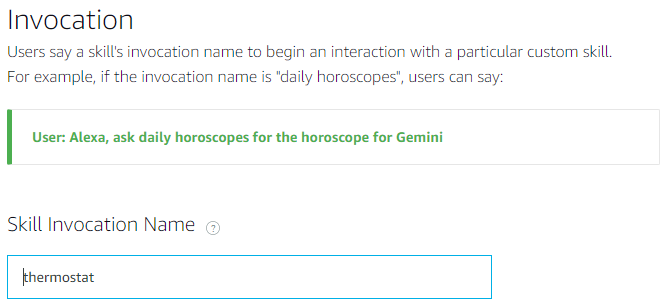
1. Utterances for turning off the lights. These utterances are mapped directly to the TurnOFFLightIntent. If the user says these phrases in their voice command, this intent will be called. For example “Alexa, tell light switch to turn the light off.

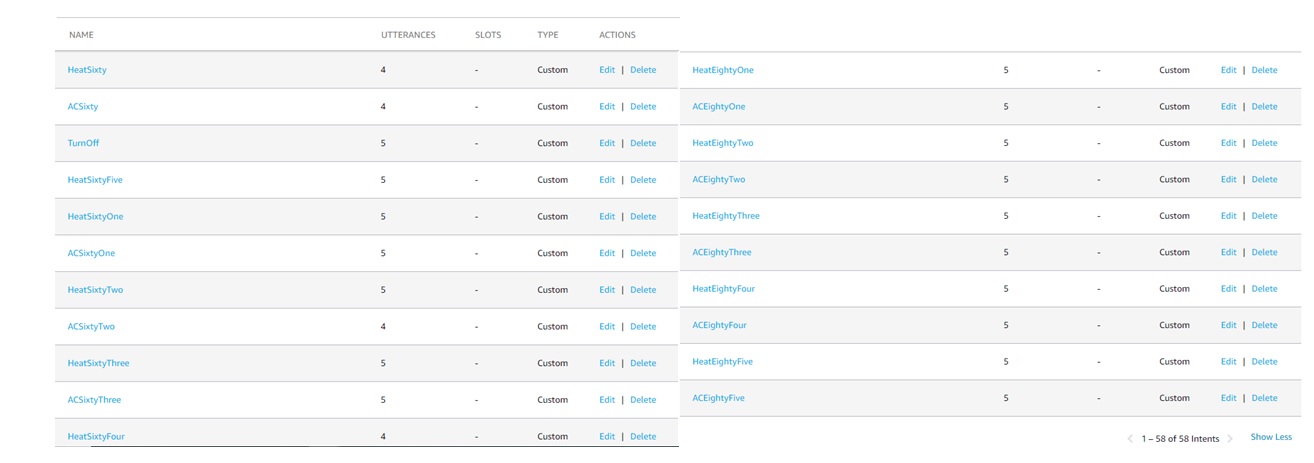
## Implementation of Task 2.2.



1. Here is the invocation name assicated with the television control portion.
2. Here are the custom intents associated with television control.

## Implementation of Task 2.3.

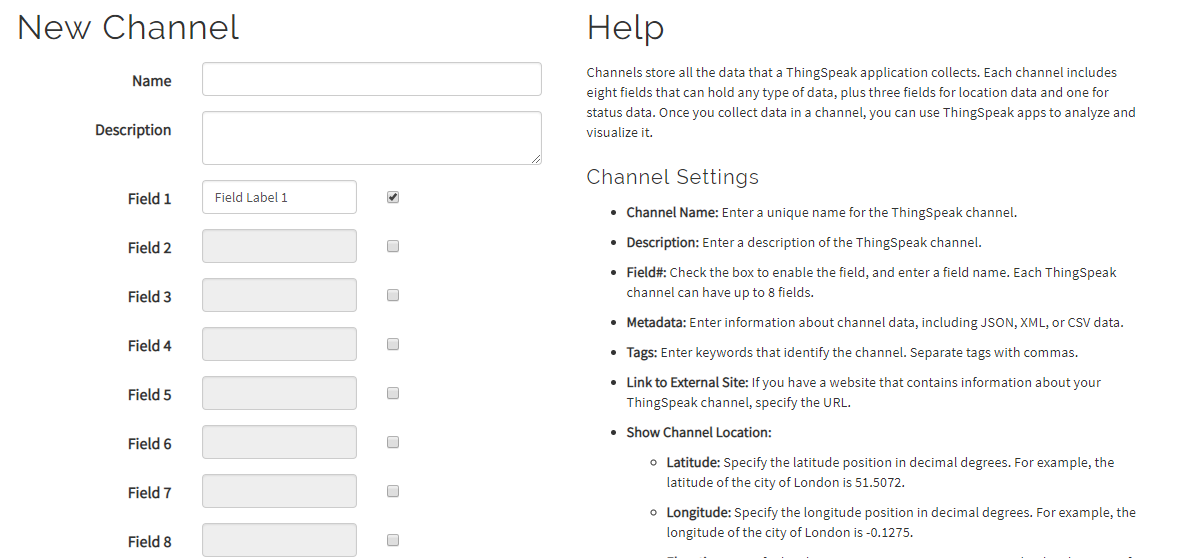


1. Here is the invocation name for the temperature control skill
2. Here is a list of the custom intents for the temperature control skill. There are intents for turning the AC on or the heat on from 60-85 degrees

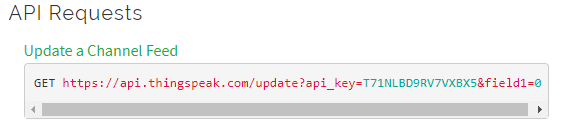
## Implementation of Task 3.

For task 3, we decided to create a data channel on ThingSpeak to send the state of each output to be stored on the cloud. That way we can use HTTP GET requests to allow our NodeMCU to retrieve the data in this data channel. HTTP GET request allows our NodeMCU to retrieve data from a given server. That is the only function of a GET request. Using this method will not alter our data in anyway. There are other ways to retrieve information from the data channel. We could have used an MQTT broker. MQTT is Messaging Queuing Telemetry Transport. It is a lightweight publish and subscribe system where you can publish and receive messages as a client. This sounds like the perfect solution for IoT communication. However, because this requires the router to be set to use port forwarding. Most firewalls have the ports closed so that means the user would have to open each port manually. Since we only want to read data and not send data, HTTP GET requests are ideal for our project.

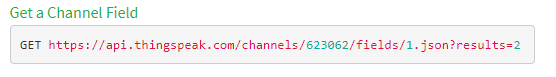
In order to create the ThingSpeak data channel we first started by creating a ThingSpeak account. Next, we created a new data channel and entered the required credentials. Once the channel was created, we used the API keys to send data to a specific field.



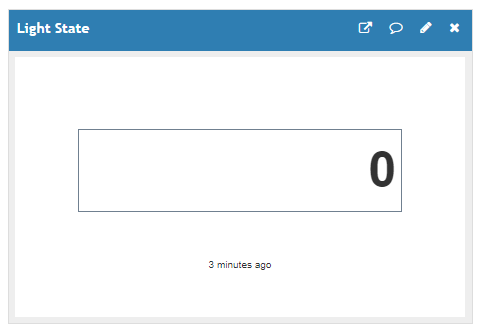
1. Here is where we created the data channel for temperature, television, and light control. The field represents the type of data added. For example, in the case for a temperature and humitdiy sensor, field 1 can be for temperature readings and field 2 can be for humidity readings.



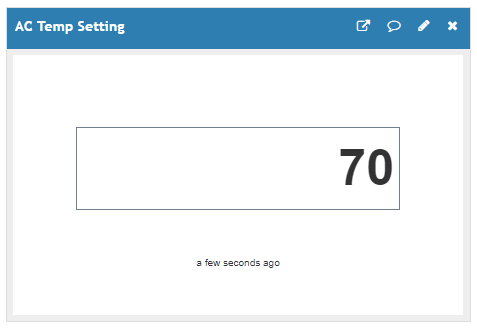
1. Here is what we used to send data to the channel. The api\_key=T71… refers to the API key given to each channel. This is used as a layer of authentication so that if someone would like to add data to the channel, they would need the API key. There are two API keys that are given, a read and write API key. This is the API key to write data to the channel. More specifically field 1 of that channel. If the channel has multiple fields, and we wanted to write data to field 2, the end of the URL would be &field1=&field2=<value>. The same would be true if the field in interest was field 3 and so on.



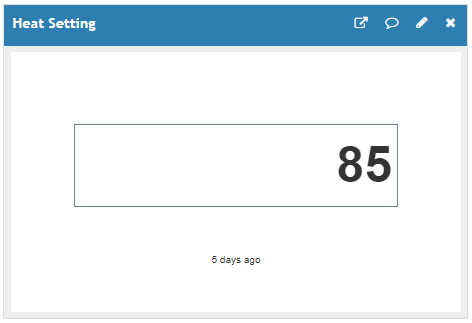
1. Here is the URL used for the GET request. Similar to the channel update URL, we edit certain parameters in order to get the data we are looking for. We can edit the channel ID to specify the channel we are looking for. We can edit the field to specify the specific field we are looking for. We can also edit the results to retrieve the last x number of results. The format the data will be in is also specified in the URL. This shows the format will be JSON.



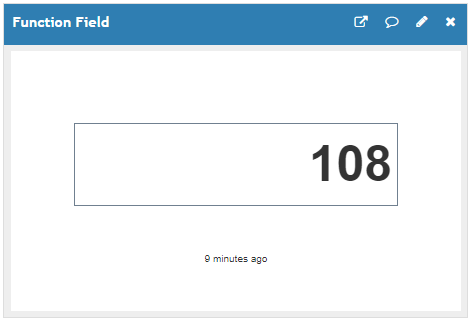
1. This is a numeric representation of what is being retrevied by the data channel. This channel is for the light state. 0 represents the light being off. 1 represents the light being on. This is the value that will be read using GET requests.



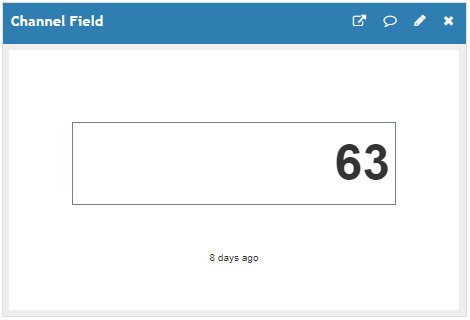
1. Here is the field for the AC temperature setting. This is the value that will be read using GET requests.



1. Here is the field for the heat temperature setting. This is the value that will be read using GET requests.



1. Here is the field for the television function. The value in the data channel determines the function that will be executed. Here 108 is showing. This is representing the “Previous Channel” button on the remote control.



1. Here is the field for the television channel. The value in the data channel determines what channel the television will be set on. Here 63 means channel 63.

## Implementation of Task 4.

Task 4 involves creating our 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 our code as background tasks. The Lambda function will run only when triggered by our Alexa skill. AWS Lambda offers several different runtimes and languages we can use to create our function from Python, C#, and Node.js. We decided to use Node.js as our runtime, essentially JavaScript.

## Implementation of Task 4.1.

A close up of text on a black background

Description automatically generated

Here is the flow chart for how the Lambda function will run for the light control function. The Alexa skill will map the intent type with the utterance the user spoke. If the utterance matches one of the inents, 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.

## Implementation of Task 4.2.

A close up of a map

Description automatically generated

1. Here is the flow chart for how the Lambda function will run for the television control function. The Alexa skill will map the intent type with the utterance the user spoke. If the utterance matches one of the inents, 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 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 spearate data fields within the channel.

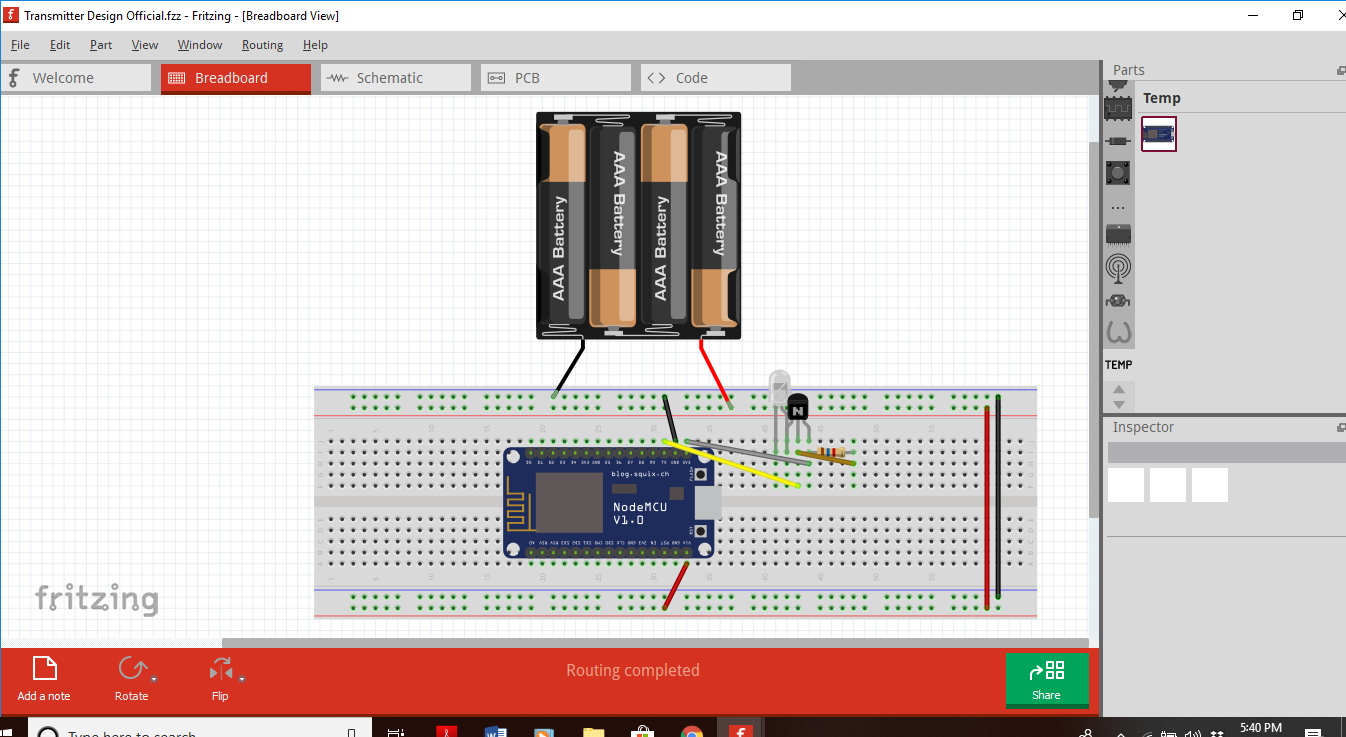
## Implementation of Task 4.3

A close up of a map

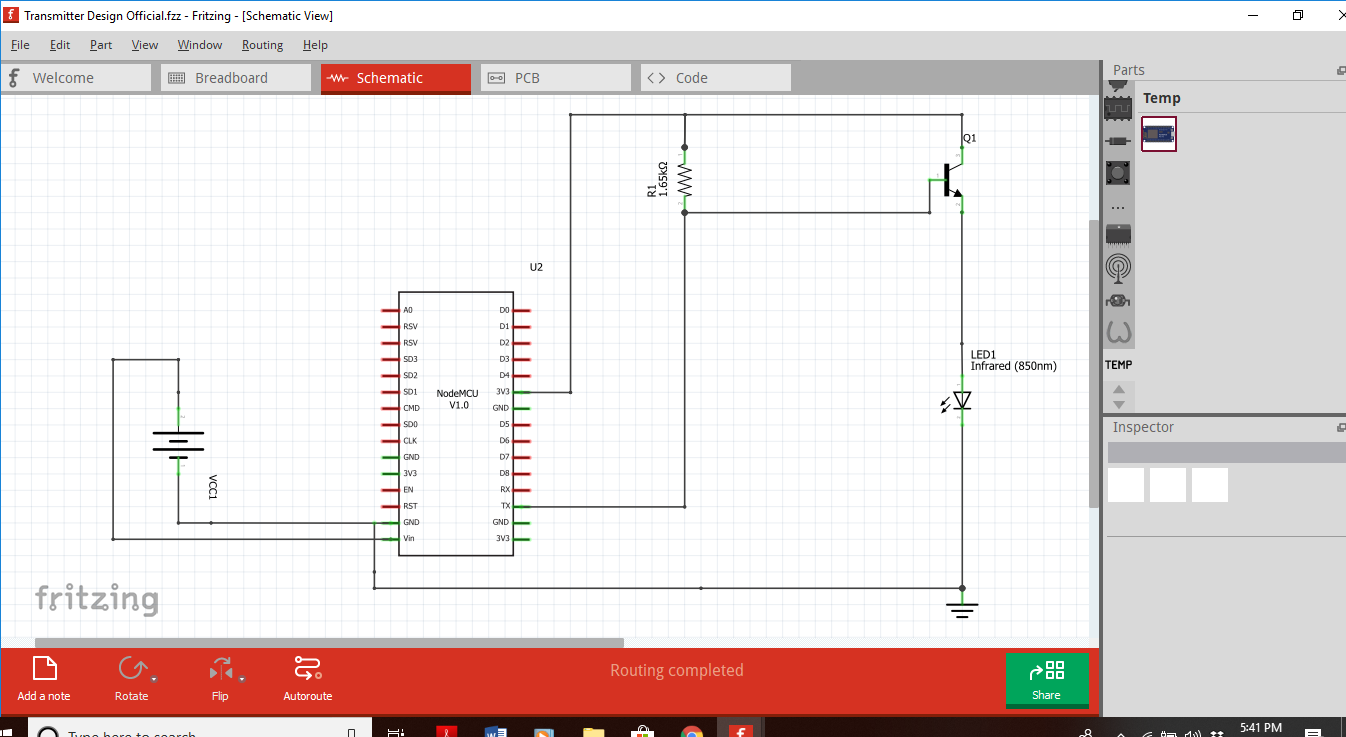
Description automatically generated

1. Here is the flow chart for how the Lambda function will run for the temperature control function. The Alexa skill will map the intent type with the utterance the user spoke. If the utterance matches one of the inents, 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 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 spearate 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.

## Implementation of Task 5.1.

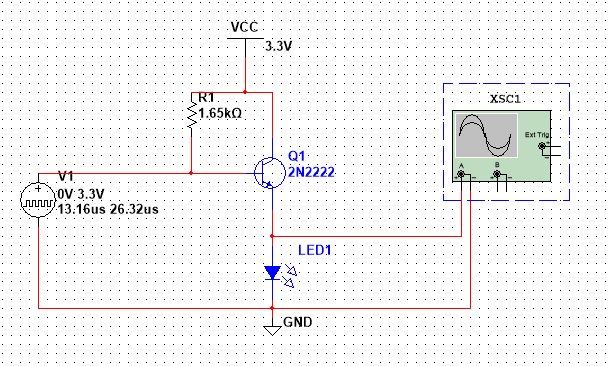


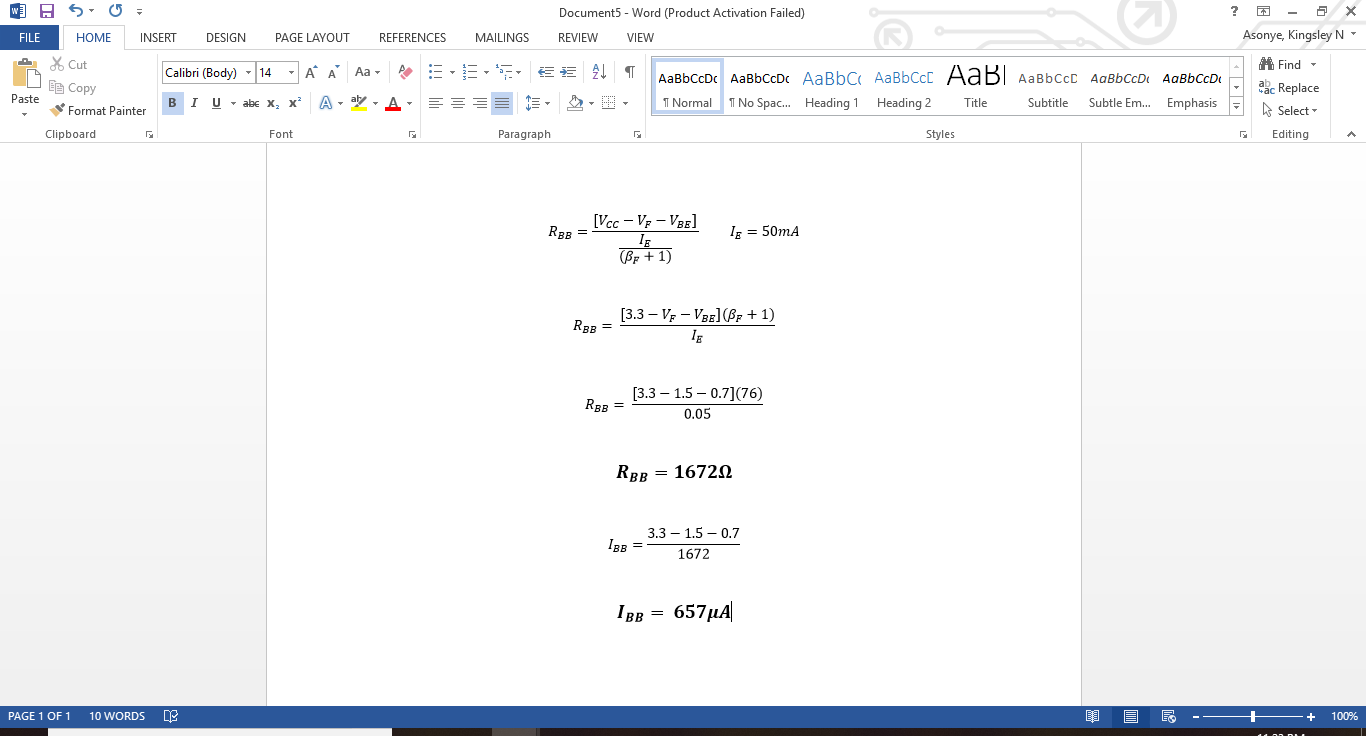
1. This figure displays the breadboard design of the IR transmitter we will be using to send codes to the IR receiver. In this breadboard design, components used to complete the design are: NodeMCU V1.0, Infrared LED (850nm), NPN transistor and a 5V battery. For schematic purposes, we used a 5V battery instead of using a 9V wall wart because the wall wart component along with the power jack was not available in the Fritzing library.

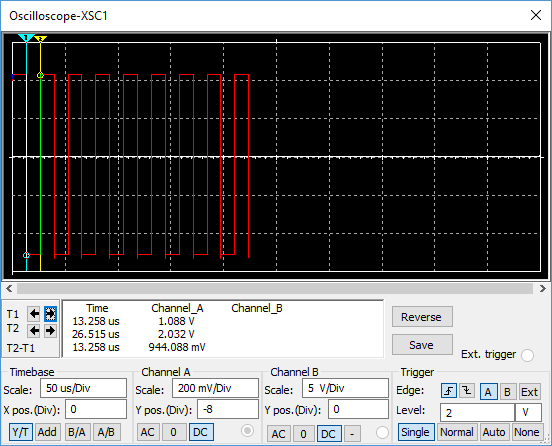


1. This figure display the schematic design for the IR transmitter. The voltage source will be powering this circuit with its positive side to the Vin input on the NodeMCU and the negative side to ground. The NodeMCU has a built in regulator to drop the voltage to 3.3V needed for the circuit. We used an 850nm IR LED because that is the wavelength that the LED operates and which is visible to the human eye. The NPN transistor is a standard using in remote control devices for its ability to produce a current flow from the collector to the emitter, which is what we need for the schematic.

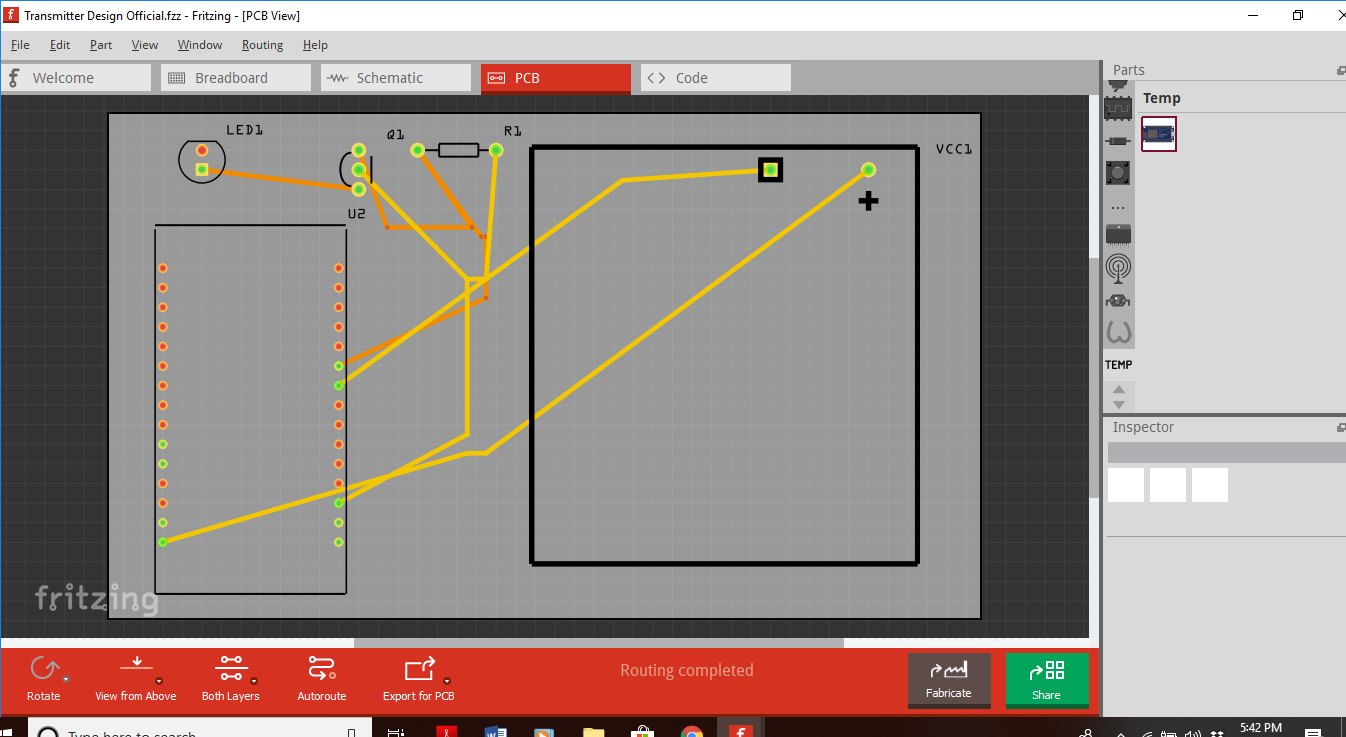
## Implementation of Task 5.2.







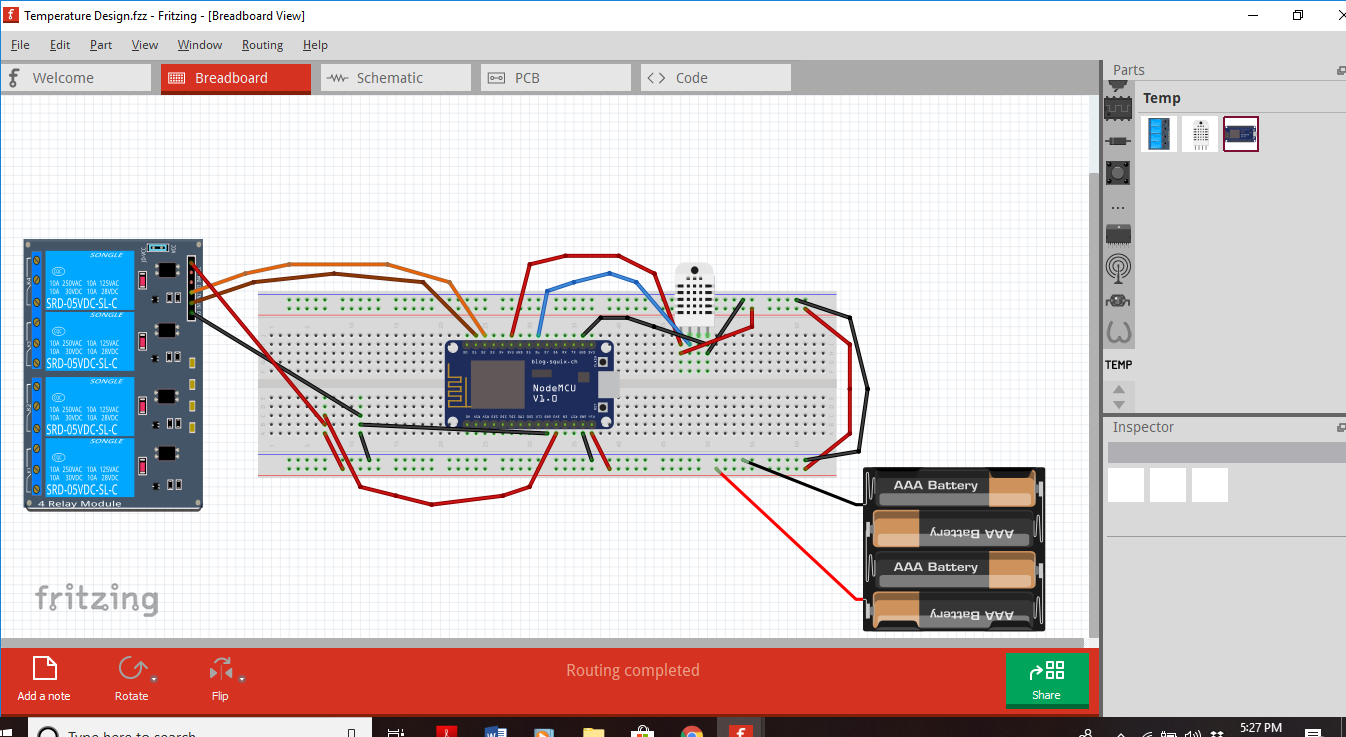
## Implementation of Task 5.3.

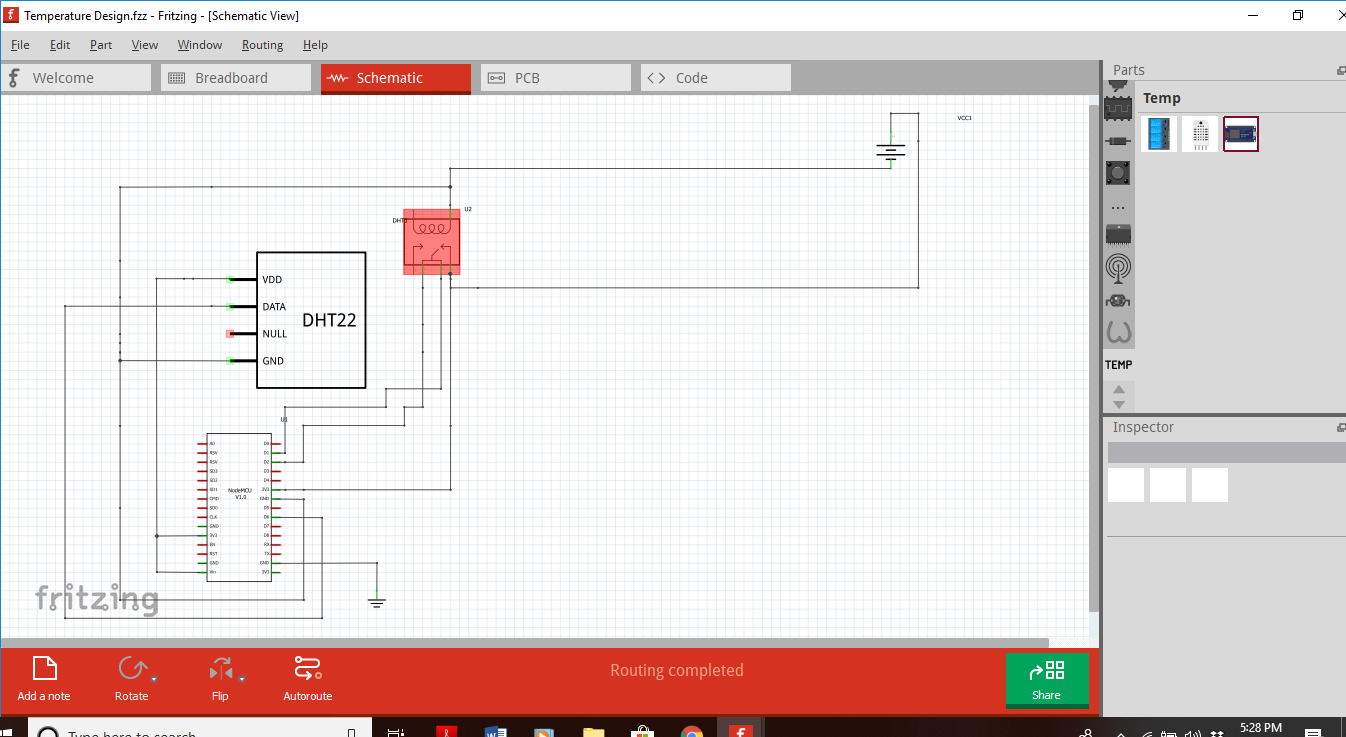


1. This figure shows the printed circuit board (PCB) design for the IR transmitter circuit. This PCB Design is based on the breadboard design followed by the schematic design. From there that is how the PCB Design is made

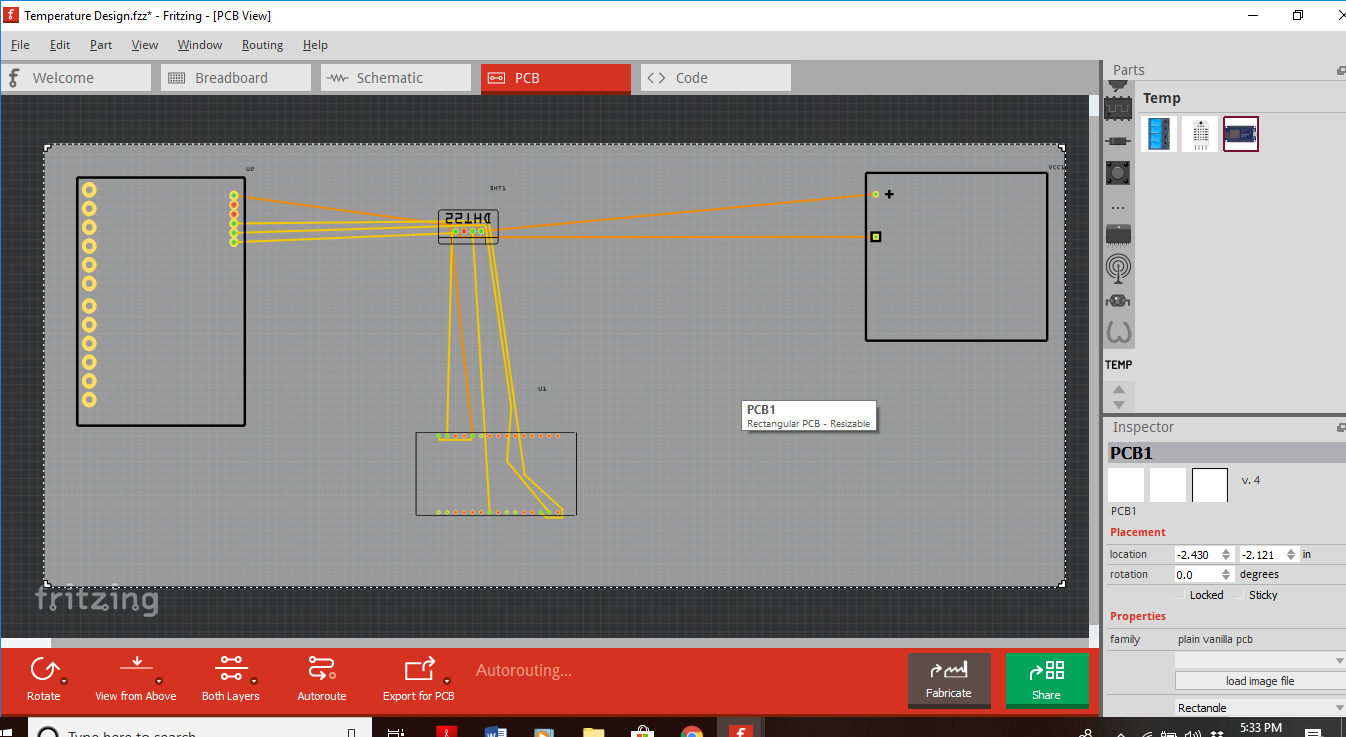
## Implementation of Task 5.4.

## Implementation of Task 6.1



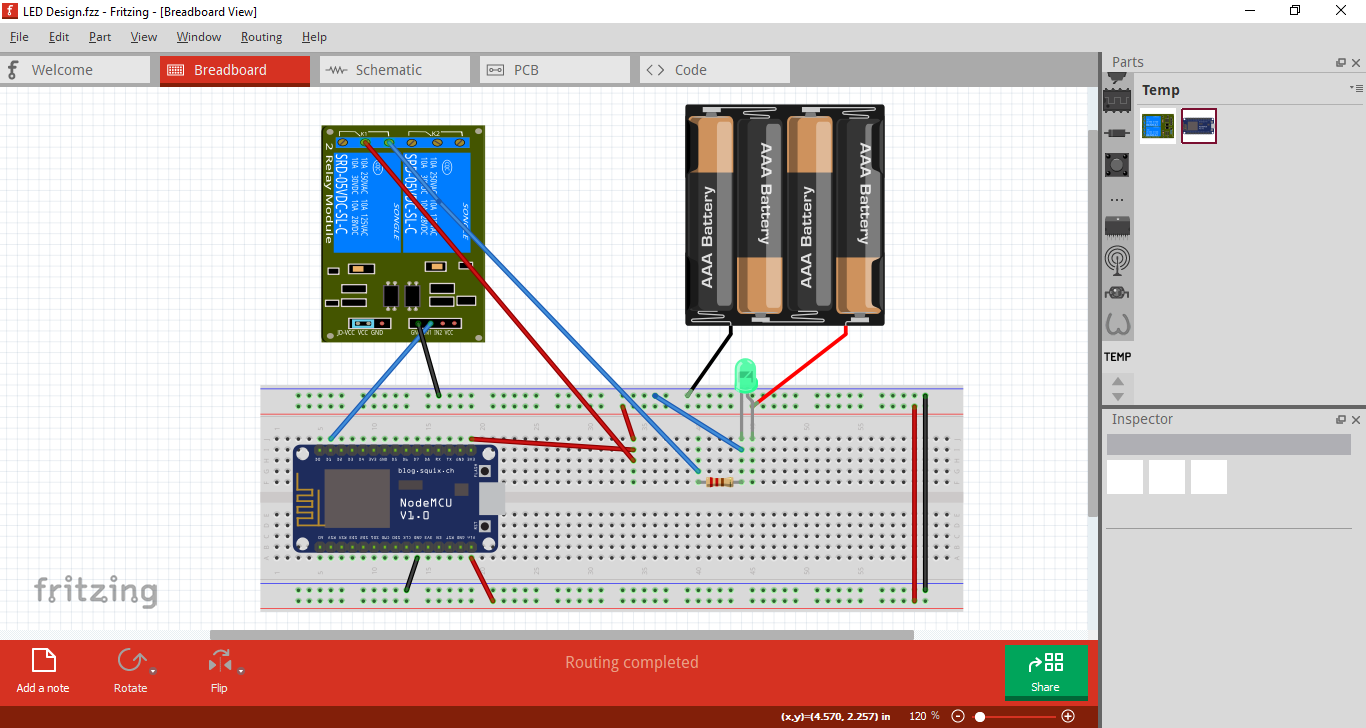


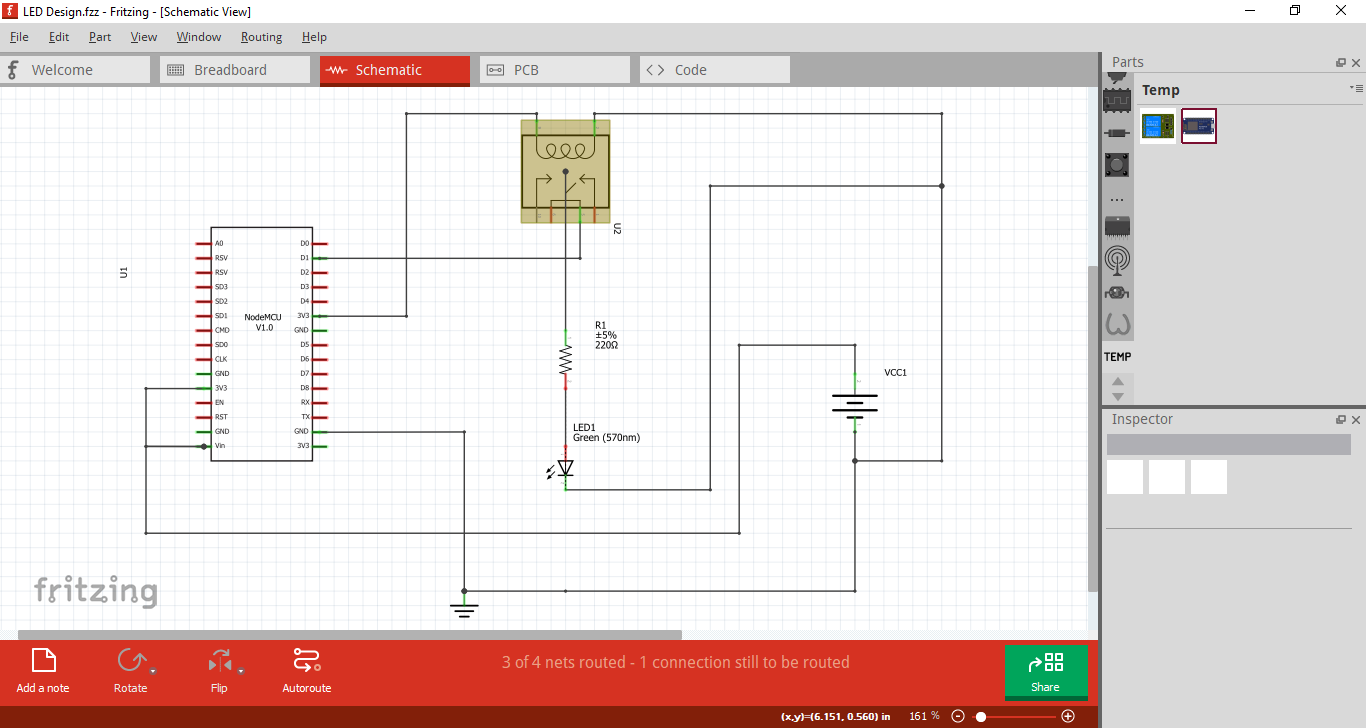
## Implementation of Task 6.2.



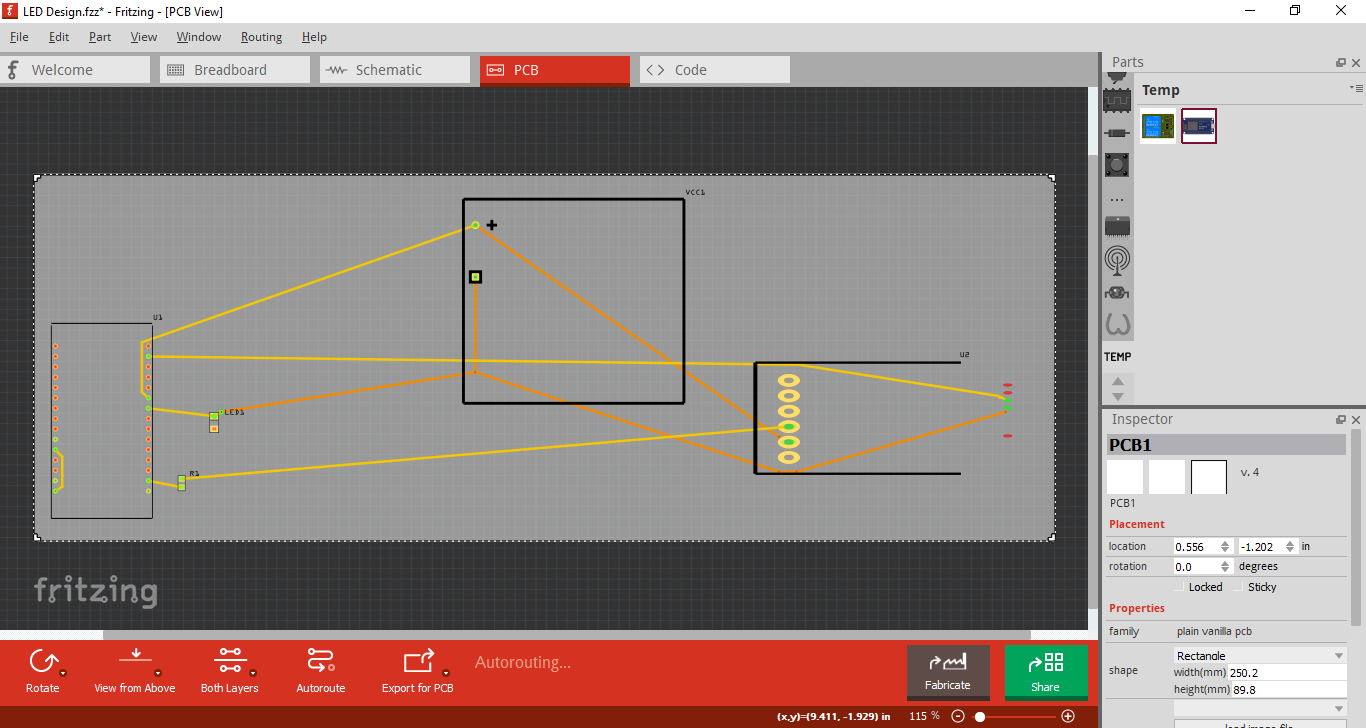
## Implementation of Task 6.3.

## Implementation of Task 7.1.





## Implementation of Task 7.2



## Implementation of Task 7.3.

## Implementation of Task 8.

## Implementation of Task 9.

## Implementation of Task 10.

## Implementation of Task 11.

## Implementation of Task 12.

1. Conclusion.

By the end of the project, conclude the project and your learning experience.

1. Acknowledgement

If you get help or support from someone else (besides the team member and the advisor) and want to show your appreciation, put here (**do not include the advisor**).

We would like to thank Dr. Alvernon Walker, Justin, and Nathan Bane for assisting us in our project.

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.

1. Component Specs
2. Specs of Arduino Due

...

1. Specs of Raspberry Pi

…

1. Source Code.
2. 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 "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

};

};

};

1. 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] D. Vantrease, R. Schreiber, M. Monchiero, M. McLaren, N. P. Jouppi, M. Fiorentino*, et al.*, "Corona: System Implications of Emerging Nanophotonic Technology," in *Computer Architecture, 2008. ISCA '08. 35th International Symposium on*, 2008, pp. 153-164.

[2] X. Zhang and A. Louri, "A multilayer nanophotonic interconnection network for on-chip many-core communications," in *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, 2010, pp. 156-161.

[3] C. Batten, A. Joshi, J. Orcutt, A. Khilo, B. Moss, C. Holzwarth*, et al.*, "Building manycore processor-to-DRAM networks with monolithic silicon photonics," in *High Performance Interconnects, 2008. HOTI '08. 16th IEEE Symposium on*, 2008, pp. 21-30.

[4] Y. Pan, P. Kumar, J. Kim, G. Memik, Y. Zhang, and A. Choudhary, "Firefly: illuminating future network-on-chip with nanophotonics," in *IEEE/ACM Intl. Symp. on Computer Architecture (ISCA)*, 2009, pp. 429-440.

[5] N. Kirman, M. Kirman, R. K. Dokania, J. F. Martinez, A. B. Apsel, M. A. Watkins*, et al.*, "Leveraging Optical Technology in Future Bus-based Chip Multiprocessors," in *Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on*, 2006, pp. 492-503.

[6] J. M. Cianchetti, C. J. Kerekes, and H. D. Albonesi, "Phastlane: a rapid transit optical routing network," in *Proceeding of: 36th International Symposium on Computer Architecture (ISCA)*, 2009, pp. 441-450.

[7] A. Shacham, K. Bergman, and L. P. Carloni, "Photonic Networks-on-Chip for Future Generations of Chip Multiprocessors," *Computers, IEEE Transactions on,* vol. 57, pp. 1246-1260, 2008.

[8] A. Shacham, K. Bergman, and L. P. Carloni, "On the Design of a Photonic Network-on-Chip," in *First International Symposium on Networks-on-Chip, 2007. NOCS 2007*, 2007, pp. 53-64.

[9] M. Kwai Hung, Y. Yaoyao, W. Xiaowen, Z. Wei, L. Weichen, and X. Jiang, "A Hierarchical Hybrid Optical-Electronic Network-on-Chip," in *VLSI (ISVLSI), 2010 IEEE Computer Society Annual Symposium on*, 2010, pp. 327-332.

[10] D. Ding and D. Z. Pan, "OIL: a nano-photonics optical interconnect library for a new photonic networks-on-chip architecture," presented at the Proceedings of the 11th international workshop on System level interconnect prediction, San Francisco, CA, USA, 2009.

[11] A. Joshi, C. Batten, K. Yong-Jin, S. Beamer, I. Shamim, K. Asanovic*, et al.*, "Silicon-photonic clos networks for global on-chip communication," in *Networks-on-Chip, 2009. NoCS 2009. 3rd ACM/IEEE International Symposium on*, 2009, pp. 124-133.

[12] D. Vantrease, R. Schreiber, M. Monchiero, M. McLaren, N. P. Jouppi, M. Fiorentino*, et al.*, "Corona: system implications of emerging nanophotonic technology," in *Proc. 35th IEEE/ACM Int'l Symp. Computer Architecture (ISCA)*, 2008, pp. 153-164.

[13] L. Zhang, E. Regentova, and X. Tan, "A 2D-Torus Based Packet Switching Optical Network-on-Chip Architecture," presented at the *IEEE International Symposium on Photonics and Optoelectronics* (SOPO 2011), Wuhan, China, 2011.

[14] L. Zhang, E. E. Regentova, and X. Tan, "Packet switching optical network-on-chip architectures," *Comput. Electr. Eng.,* vol. 39, pp. 697-714, 2013.

[15] G. Huaxi, X. Jiang, and W. Zheng, "A novel optical mesh network-on-chip for gigascale systems-on-chip," in *Circuits and Systems, 2008. APCCAS 2008. IEEE Asia Pacific Conference on*, 2008, pp. 1728-1731.

[16] G. Huaxi, X. Jiang, and Z. Wei, "A low-power fat tree-based optical Network-On-Chip for multiprocessor system-on-chip," in *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE '09.*, 2009, pp. 3-8.

[17] Y. Yaoyao, X. Jiang, H. Baihan, W. Xiaowen, Z. Wei, W. Xuan*, et al.*, "3-D Mesh-Based Optical Network-on-Chip for Multiprocessor System-on-Chip," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on,* vol. 32, pp. 584-596, 2013.

[18] A. Shacham, K. Bergman, and L. P. Carloni, "Photonic networks-on-chip for future generations of chip multiprocessors," *IEEE Trans. Computers,* vol. 57, pp. 1246-1260, 2008.

[19] A. W. Poon, F. X. Xu, and X. Luo, "Cascaded active silicon microresonator array cross-connect circuits for WDM networks-on-chip," in *Proc. SPIE*, 2008, p. 689812.

[20] M. Lipson, "Compact Electro-Optic Modulators on a Silicon Chip," *IEEE Journal of Selected Topics in Quantum Electronics,* vol. 12, pp. 1520-1526, 2006.

[21] M. Lipson, "Guiding, modulating, and emitting light on Silicon-challenges and opportunities," *Lightwave Technology, Journal of,* vol. 23, pp. 4222-4238, 2005.

[22] T. Xianfang, Y. Mei, Z. Lei, J. Yingtao, and Y. Jianyi, "Wavelength-routed optical networks-on-chip built with comb switches," in *Photonics Conference (IPC), 2013 IEEE*, 2013, pp. 46-47.

This page was intentionally left blank.