part 2 – Alexa Skill Creation

I- Skill Design:

Before creating a skill, it is necessary to think about the purposes of the application: the actions to be executed, the smart devices that must be controlled and the requests that may be expressed by the user.

These elements will make it possible to determine if one or more intents must be used and for each intent, the set of phrases that a person might use to control the devices: The utterances. From, theses utterances, the slots can be identified and consequently the slot types.

1- **Invocation name:** "**room led lights**". Skill Invocation names in an Amazon Account should be different.

2- **Intent** : "**rgbled**"

2-1: **The slots:** In this example, 3 control instructions (on, off, toggle) and 4 object description (green, red, blue and all). Therefore, the Intent ***rgbled*** should contain 2 slots:

- One **slot** for the control instruction **rgbstate**: on, off and toggle are the possible slot values
- One **slot** for the led color **rgbcolor**: green, red, blue and all are the possible slot values.

2-2: **The utterances:**

Even if there are many control instructions and led colors, we will create utterances only for one led color (green) and one control instruction (on). Then, the utterances will be generalized to all possible slot values.

In Alexa developer console, the generalization is carried out by using the slot names between brackets

Utterances for green led/on instruction	Generalized utterances
on green	{rgbstate} {rgbcolor}
green on	{rgbcolor} {rgbstate}
on green led	{rgbstate} {rgbcolor} led
on the green led	{rgbstate} the {rgbcolor} led
green led on	{rgbcolor} led {rgbstate}
turn on the green led	turn {rgbstate} the {rgbcolor} led
turn the green led on	turn the {rgbcolor} led {rgbstate}

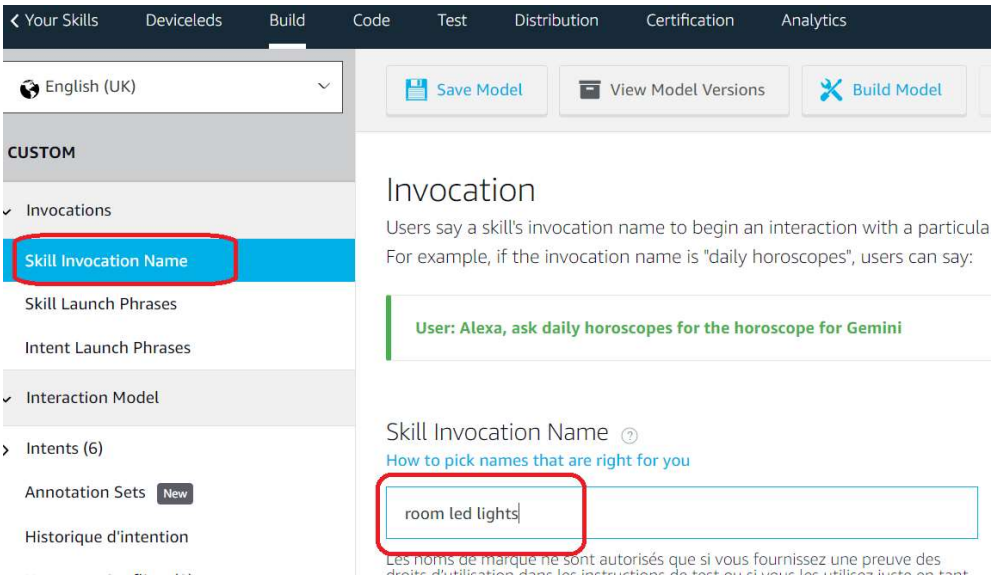
2-3 **Slot Types**: Finally, for each slot a type is defined that simply contains all possible values of the slot.

ctrlcolor : with the value set green, red, blue and all.

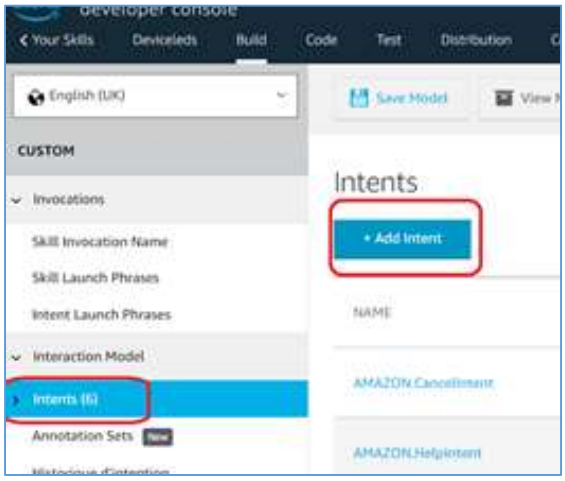
ctrlstate: with the value set on, off and toggle.

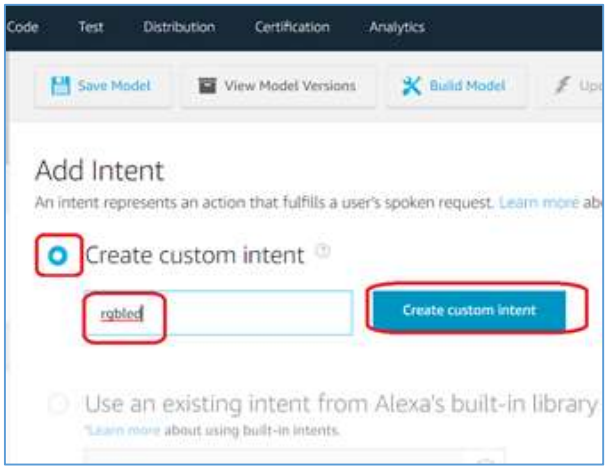
II- Skill Creation:

1- Skill Invocation



2- Access the Add Intent Window and Create a new Intent « **rgbled** »





3- Adding the utterances

Add the following utterances :

on green

green on

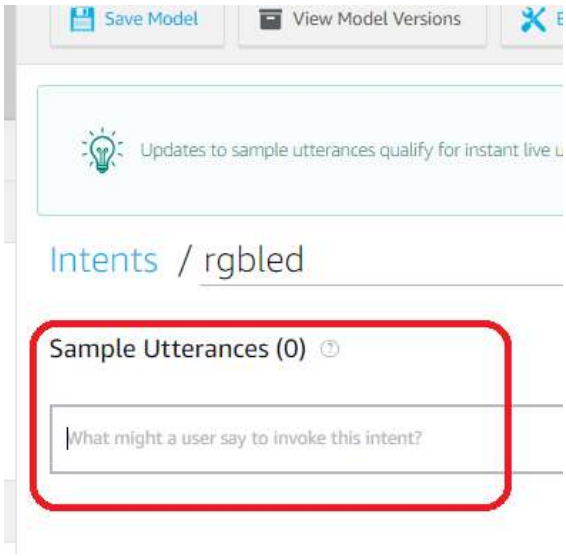
on green led

on the green led

green led on

turn on the green led

turn the green led on



Rq : In this example, the intent contains 2 slots. We only add the utterances for one value for each slot (green for the color slot and on for the action slot). The utterances will be generalized.

4- Utterances Generalization and slot identifying

For the first sentence (on green), right click on the text ”green” and create a slot “rgbcolor” and click on Add (Fig1).

For the text “on” , create a slot “rgbstate”

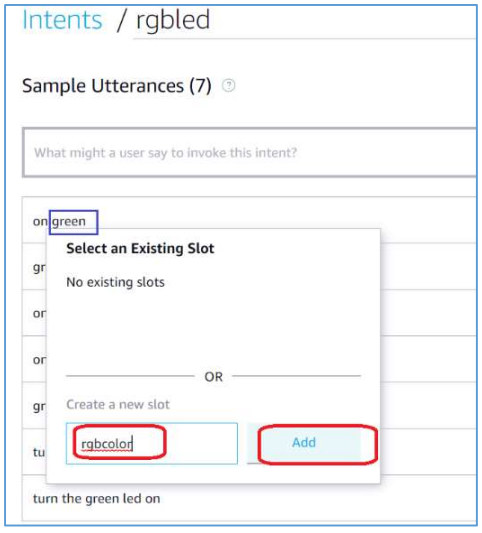


Fig1

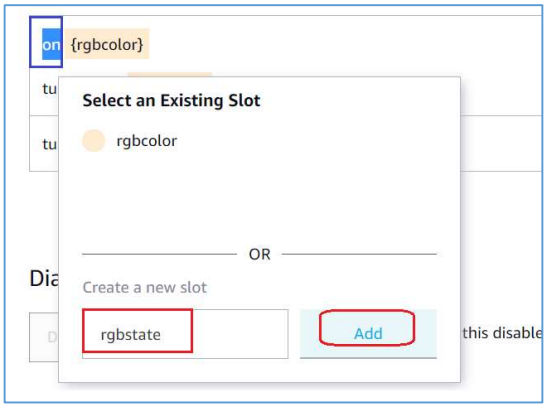
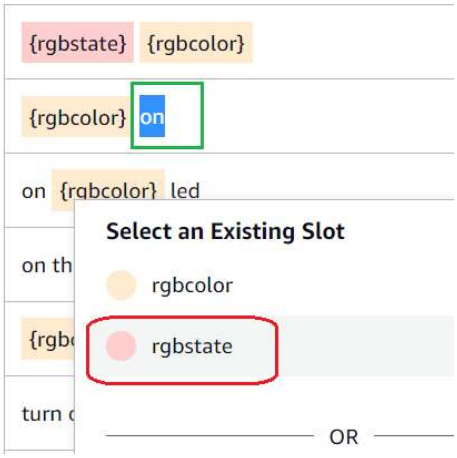
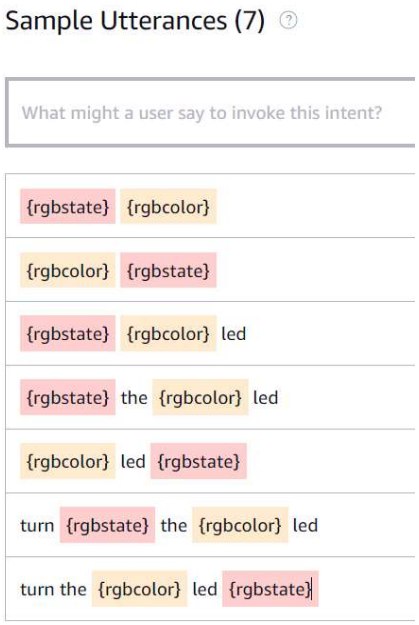


Fig2

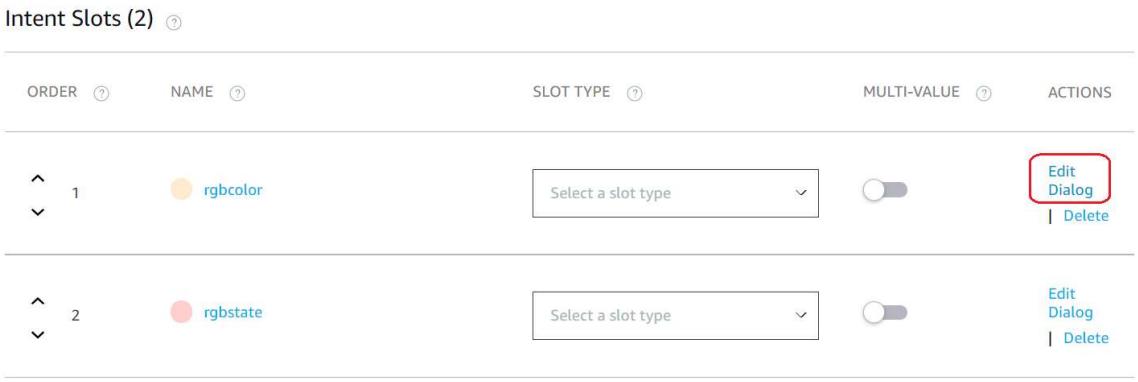
The same action are carried for all the remaining utterances



The final result will be as follows:

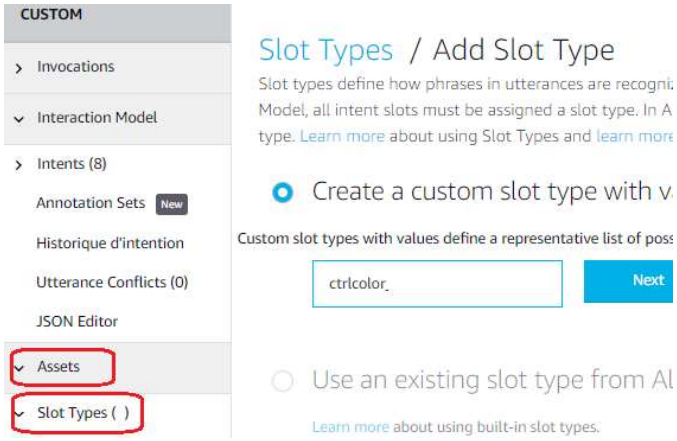


The created slots will be displayed in the intents interface.

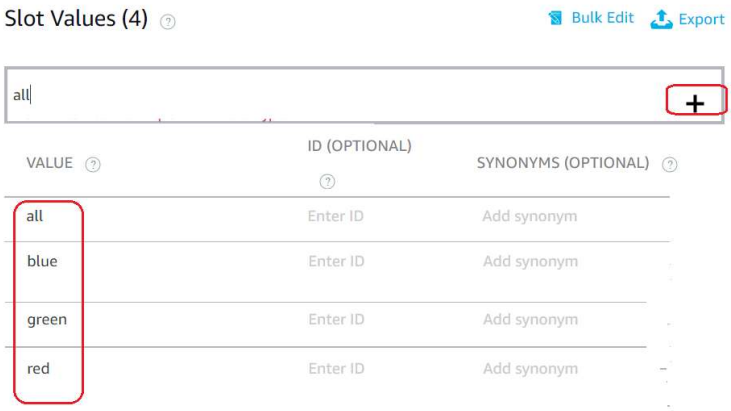


5- Adding Slot Types

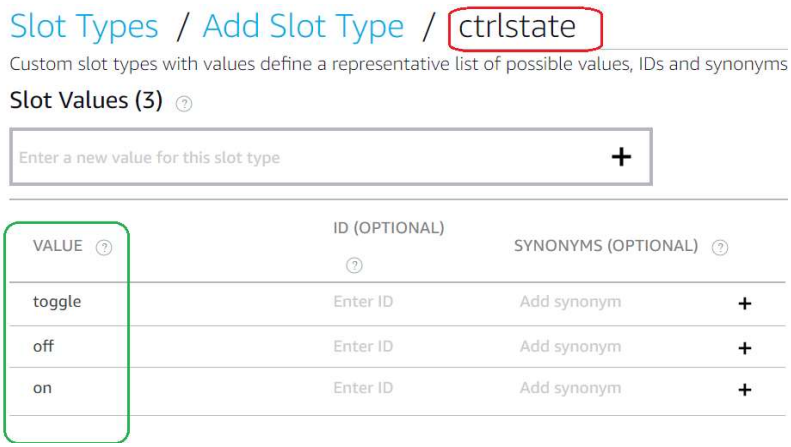
In the Assets -> Slot Types , create a "**Slot Type**" = "**ctrlcolor**"



Add the following slot values.



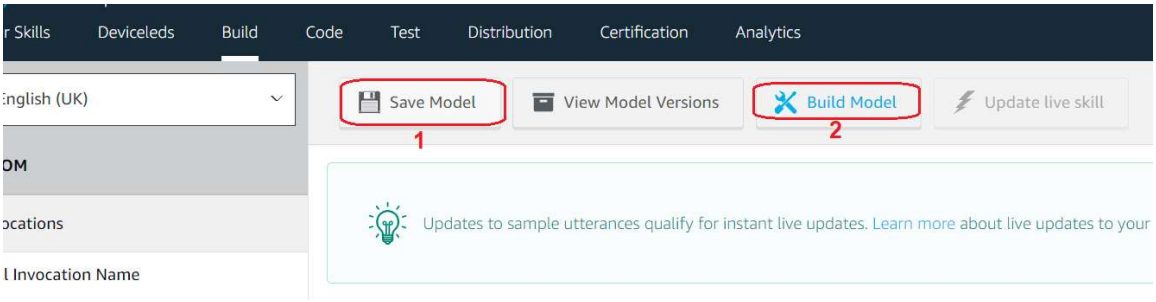
Add the slottype « **ctrlstate** » with the values on, off and toggle.



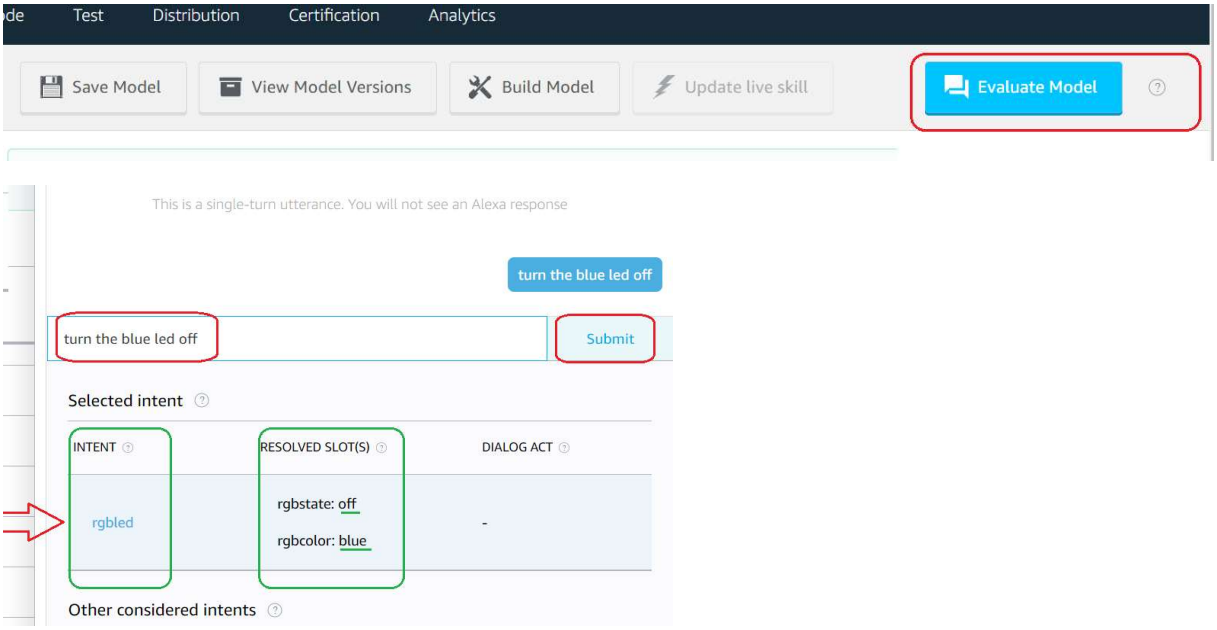
6- Assigning Slot types

Finally, the created slot types (ctrlcolor and ctrlstate) must be assigned to the defined slots (rgbcolor and rgbstate) :

7- Save and build the created skill



8- Evaluate the created model to test if the defined utterances can trigger the intent and if the slot values can be detected



part 3 – Lambda Service

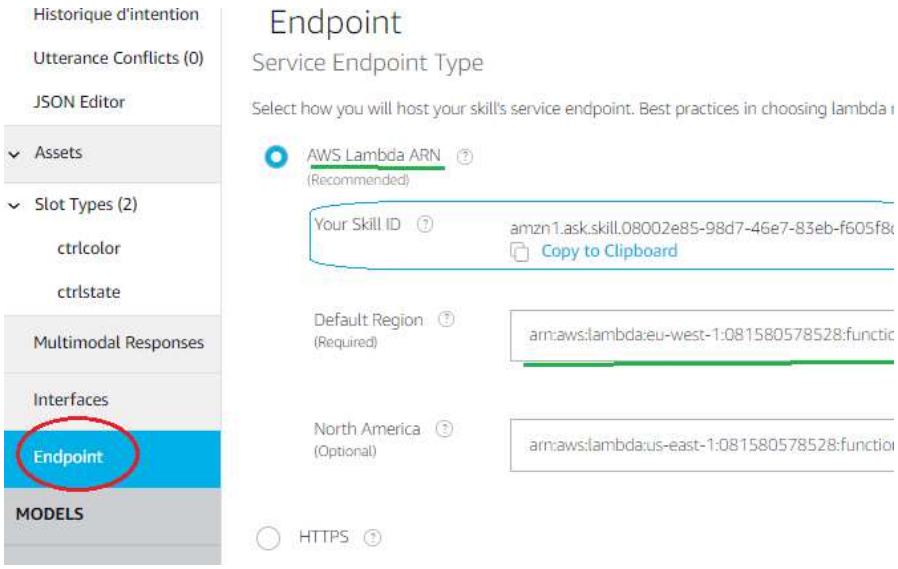
AWS Lambda is a cloud serverless compute service, i.e it is possible to run code without provisioning or managing infrastructure. Simply write and upload code as a .zip file or container image.

The user has not to pay servers or clusters, he pays when the code is running depending only on the execution time.

The code executed on the lambda run time environment is called a lambda function. Each Lambda code (Function) has a unique ID and can be called from anywhere and can be triggered from any cloud service (AWS).

Generally, to use the Lambda service, an AWS account must be created (credit card provided), but for the Alexa applications, the Lambda service is free and integrated within the alexa service interface.

When creating the Alexa Service, it is possible to choose the programming language to be used (principally node js and python) and the two services are bound together (see following Figure): the Lambda Function is called each time the Invocation name or an intent is detected.



It is however possible to specify a HTTPS URL used to process the alexa triggered events

I- Lambda Code for Alexa Voice Service (Node JS)

When an alexa skill is created, a default index.js file is added in the lambda environment and it contains node js code to process the invocation name detection (Welcome message) and the default intents (HELP, CANCEL, ETC…,). Each event processing code is contained in one Handler.

The entry point (first code to be executed) when a lambda function is triggered by an Alexa Voice Service (AVS) event, is identified by the instruction **export.handler**

```
// This handler acts as the entry point for your skill, routing all request and response
// payloads to the handlers above. Make sure any new handlers or interceptors you've
// defined are included below. The order matters - they're processed top to bottom.
exports.handler = Alexa.SkillBuilders.custom()
    .addRequestHandlers(
        LaunchRequestHandler,
        HelloWorldIntentHandler,
        HelpIntentHandler,
        CancelAndStopIntentHandler,
        SessionEndedRequestHandler,
        IntentReflectorHandler)
    .addErrorHandlers(
        ErrorHandler)
    .lambda();
```

This entry point, indicates what Handlers to execute (in order) to process an Alexa. When one Handler is able to process the event, the remaining Handlers are no more executed.

The first Handler to be executed is the LaunchRequest Handler (Invocation Name detected). It is possible to customize the response

```
/**
 * Called when the user launches the skill without specifying what they want.
 */
const LaunchRequestHandler = {
    canHandle(handlerInput) {
        return Alexa.getRequestType(handlerInput.requestEnvelope) === 'LaunchRequest';
    },
    handle(handlerInput) {
        const speakOutput = 'Welcome, This Application helps turning on , off, or toggling RGB Leds?';

        return handlerInput.responseBuilder
            .speak(speakOutput)
            .reprompt(speakOutput)
            .getResponse();
    }
};
```

Obviously, there is no Handler for the added Intent. So, a Handler call must be added to the **exports.Handler** list

```
exports.handler = Alexa.SkillBuilders.custom()
    .addRequestHandlers(
        LaunchRequestHandler,

        RgbLedIntentHandler,

        HelloWorldIntentHandler,
        HelpIntentHandler,
        CancelAndStopIntentHandler,
        FallbackIntentHandler,
        SessionEndedRequestHandler,
        IntentReflectorHandler)
    .addErrorHandlers(
        ErrorHandler)
```

The chosen Handler name is RgbLedIntentHandler

The Handler code will contain two functions:

1- **canHandle**: to test if the sent message (Intent from AVS) can be processed by the handle.
2- **handle**: is executed if the canHandle returns true.

```
const RgbLedIntentHandler = {

    canHandle(handlerInput) {
        return Alexa.getRequestType(handlerInput.requestEnvelope) === 'IntentRequest'
            && Alexa.getIntentName(handlerInput.requestEnvelope) === 'rgbled';
    },

    handle(handlerInput) {

        const slots = handlerInput.requestEnvelope.request.intent.slots
        //const colorRequest = rgbled.slots.color.value;
        const ledcolor = slots['rgbcolor'].value
        const ledcommand = slots['rgbstate'].value

        const [mqttspeech, mqttmsg] = buildresponse(ledcolor,ledcommand);

        PublushMqttMsg(mqttmsg);

        const speakOutput = mqttspeech;

        return handlerInput.responseBuilder
            .speak(speakOutput)
            .reprompt(speakOutput)
            .getResponse();
    }
};
```

The handle function gets the JSON message as input (handlerinput), extracts the slot values, builds a speech response (mqttspeech) and publishes a message (mqttmsg)

Two other functions are added:

- **buildresponse** : to build the speech returned to the user and the mqtt message to be published.

```
function buildresponse (ledcolor,ledcommand) {

    var mqttspeech;
    var mqttmsg;

    switch (ledcolor) {
case 'blue':
    mqttspeech = "Blue Led is";
    mqttmsg = "BLU";
    break;
case 'green':
    mqttspeech = "Green Led is";
    mqttmsg = "GRN";
    break;
case 'red':
    mqttspeech = "Red Led is";
    mqttmsg = "RED";
    break;
case 'all':
    mqttspeech = "All Leds are";
    mqttmsg = "ALL";
    break;
default:
    mqttspeech = "NULL";
    }

    switch (ledcommand) {
case 'on':
    mqttspeech = mqttspeech+" on";
    mqttmsg += "ON";
    break;
case 'off':
    mqttspeech = mqttspeech+" off";
    mqttmsg += "OF";
    break;
case 'toggle':
    mqttspeech = mqttspeech+" toggled";
    mqttmsg += "TO";
    break;
default:
    mqttspeech = "NULL";
    }

return [mqttspeech, mqttmsg];
```

- **PublishMqttMsg**: to publish the mqtt message to the mqtt broker.

```
function PublushMqttMsg (mqttmsg)
{
    var mqttpromise = new Promise( function(resolve,reject){

    var client = mqtt.connect({port:15281,host:'z96e4d99.us-east-1.emqx.cloud',usernam

    client.on('connect', function() { // When connected
        // publish a message to  mqtt topic
        client.publish('iotdevice/action', mqttmsg);
        client.end();
```