#include "Arduino.h"

#include <ESP8266WiFi.h>

#include "fauxmoESP.h"

#include "ESPAsyncWebServer.h"

#include "ESPAsyncTCP.h"

#include <Hash.h>

#define WIFI\_SSID "SmartShower" // Please Enter you Wifi name here

#define WIFI\_PASS "teamg" // Enter password here

#define SERIAL\_BAUDRATE 115200

fauxmoESP fauxmo;

#define RELAY\_PIN 5

const int buttonPin = 4; // the pin that the pushbutton is attached to

int buttonState = 0; // current state of the button

int lastButtonState = 0; // previous state of the button

// -----------------------------------------------------------------------------

// Wifi

// -----------------------------------------------------------------------------

void wifiSetup() {

// Set WIFI module to STA mode

WiFi.mode(WIFI\_STA);

// Connect

Serial.printf("[WIFI] Connecting to %s ", WIFI\_SSID);

WiFi.begin(WIFI\_SSID, WIFI\_PASS);

// Wait

while (WiFi.status() != WL\_CONNECTED) {

Serial.print(".");

delay(100);

}

Serial.println();

// Connected!

Serial.printf("[WIFI] STATION Mode, SSID: %s, IP address: %s\n", WiFi.SSID().c\_str(), WiFi.localIP().toString().c\_str());

}

void callback(unsigned char device\_id, const char \* device\_name, bool state){

Serial.print("Device "); Serial.print(device\_name);

Serial.print(" state: ");

if (state) {

Serial.println("ON");

digitalWrite(RELAY\_PIN, HIGH);

} else {

Serial.println("OFF");

digitalWrite(RELAY\_PIN, LOW);

}

}

void setup() {

pinMode(RELAY\_PIN, OUTPUT);

pinMode(buttonPin, INPUT\_PULLUP);

digitalWrite(RELAY\_PIN, LOW);

// Init serial port and clean garbage

Serial.begin(SERIAL\_BAUDRATE);

Serial.println("FauxMo demo sketch");

Serial.println("After connection, ask Alexa/Echo to 'turn <devicename> on' or 'off'");

// Wifi

wifiSetup();

// Fauxmo

fauxmo.addDevice("the light");

fauxmo.onMessage(callback);

}

void loop() {

fauxmo.handle();

// read the pushbutton input pin:

buttonState = digitalRead(buttonPin);

// compare the buttonState to its previous state

if (buttonState != lastButtonState) {

// if the state has changed, increment the counter

if (buttonState == LOW) {

Serial.println("on");

digitalWrite(RELAY\_PIN, HIGH);

}

else {

// if the current state is LOW then the button

// went from on to off:

Serial.println("off");

digitalWrite(RELAY\_PIN, LOW);

}

// Delay a little bit to avoid bouncing

delay(50);

}

// save the current state as the last state,

//for next time through the loop

lastButtonState = buttonState;

}

------------------------------------------------------------------------------------------------------------------------

/\*

Prototipo de Alexa Wise Shower

Code by Virgilio Aray Arteaga

February 2018

This skecht is an adaptation of the original code that can be found at the following address

https://github.com/odelot/aws-mqtt-websockets

\*/

//-----------------------------------------------

#include <Arduino.h>

#include <Stream.h>

#include <ESP8266WiFi.h>

#include <ESP8266WiFiMulti.h>

//AWS

#include "sha256.h"

#include "Utils.h"

//WEBSockets

#include <Hash.h>

#include <WebSocketsClient.h>

//MQTT PAHO

#include <SPI.h>

#include <IPStack.h>

#include <Countdown.h>

#include <MQTTClient.h>

//AWS MQTT Websocket

#include "Client.h"

#include "AWSWebSocketClient.h"

#include "CircularByteBuffer.h"

//AWS IOT config, change these:

char wifi\_ssid[] = "your network";

char wifi\_password[] = "your wifi password";

char aws\_endpoint[] = "?????????.iot.us-east-1.amazonaws.com";

char aws\_key[] = "???????????"; // your IAM credentials

char aws\_secret[] = "??????????????????"; // your IAM credentials

char aws\_region[] = "us-east-1";

char\* aws\_topic = "$aws/things/ducha/shadow/update";

int port = 443;

//MQTT config

//---------------------------------------//

const int maxMQTTpackageSize = 1024; // VERY IMPORTANT

//---------------------------------------//

const int maxMQTTMessageHandlers = 4; // VERY IMPORTANT

//---------------------------------------//

ESP8266WiFiMulti WiFiMulti;

AWSWebSocketClient awsWSclient(1000);

IPStack ipstack(awsWSclient);

MQTT::Client<IPStack, Countdown, maxMQTTpackageSize, maxMQTTMessageHandlers> \*client = NULL;

//# of connections

long connection = 0;

//generate random mqtt clientID

char\* generateClientID () {

char\* cID = new char[23]();

for (int i=0; i<22; i+=1)

cID[i]=(char)random(1, 256);

return cID;

}

//count messages arrived

int arrivedcount = 0;

char buf[500];

int ParImpar=1;

int OnOff=1;

String Respuesta ;

//

// Here the response of the AWS IOT

//

//callback to handle mqtt messages

//

//

void messageArrived(MQTT::MessageData& md)

{

MQTT::Message &message = md.message;

/\*

//Serial.print("Message ");

//Serial.print(++arrivedcount);

//Serial.print(" arrived: qos ");

//Serial.print(message.qos);

//Serial.print(", retained ");

//Serial.print(message.retained);

//Serial.print(", dup ");

//Serial.print(message.dup);

//Serial.print(", packetid ");

//Serial.println(message.id);

//Serial.print("Payload ");

\*/

char\* msg = new char[message.payloadlen+1]();

memcpy (msg,message.payload,message.payloadlen);

// //Serial.println(msg);

Respuesta=msg; // "REspuesta" is a global var will be contain the paiload from aws iot

delete msg;

}

//

//

//

//connects to websocket layer and mqtt layer

bool connect () {

if (client == NULL) {

client = new MQTT::Client<IPStack, Countdown, maxMQTTpackageSize, maxMQTTMessageHandlers>(ipstack);

} else {

if (client->isConnected ()) {

client->disconnect ();

}

delete client;

client = new MQTT::Client<IPStack, Countdown, maxMQTTpackageSize, maxMQTTMessageHandlers>(ipstack);

}

//delay is not necessary... it just help us to get a "trustful" heap space value

delay (1000);

//Serial.print (millis ());

//Serial.print (" - conn: ");

//Serial.print (++connection);

//Serial.print (" - (");

//Serial.print (ESP.getFreeHeap ());

//Serial.println (")");

int rc = ipstack.connect(aws\_endpoint, port);

if (rc != 1)

{

//Serial.println("error connection to the websocket server");

return false;

} else {

//Serial.println("websocket layer connected");

}

//Serial.println("MQTT connecting");

MQTTPacket\_connectData data = MQTTPacket\_connectData\_initializer;

data.MQTTVersion = 3;

char\* clientID = generateClientID ();

data.clientID.cstring = clientID;

rc = client->connect(data);

delete[] clientID;

if (rc != 0)

{

//Serial.print("error connection to MQTT server");

//Serial.println(rc);

return false;

}

//Serial.println("MQTT connected");

return true;

}

//subscribe to a mqtt topic

void subscribe () {

//subscript to a topic

int rc = client->subscribe(aws\_topic, MQTT::QOS0, messageArrived);

if (rc != 0) {

//Serial.print("rc from MQTT subscribe is ");

//Serial.println(rc);

return;

}

//Serial.println("MQTT subscribed");

}

//send a message to a mqtt topic

void sendmessage () {

//send a message

MQTT::Message message;

//char buf[100];

//strcpy(buf, "{\"state\":{\"reported\":{\"on\": false}, \"desired\":{\"on\": false}}}");

//strcpy(buf, "{\"state\":{\"reported\":{\"interruptor\":\"on\"}}}");

message.qos = MQTT::QOS0;

message.retained = false;

message.dup = false;

message.payload = (void\*)buf;

message.payloadlen = strlen(buf)+1;

int rc = client->publish(aws\_topic, message);

//Serial.println(rc);

}

void setup() {

//

// Serial To comunicate with arduino mega

//

Serial.begin (19200,SERIAL\_8N1);

////delay (20000);

////Serial.setDebugOutput(1);

//fill with ssid and wifi password

WiFiMulti.addAP(wifi\_ssid, wifi\_password);

//Serial.println ("connecting to wifi");

while(WiFiMulti.run() != WL\_CONNECTED) {

delay(100);

//Serial.print (".");

}

//Serial.println ("\nconnected");

//fill AWS parameters

awsWSclient.setAWSRegion(aws\_region);

awsWSclient.setAWSDomain(aws\_endpoint);

awsWSclient.setAWSKeyID(aws\_key);

awsWSclient.setAWSSecretKey(aws\_secret);

awsWSclient.setUseSSL(true);

Respuesta = "nothing" ;// this global var will be contain the paiload from aws iot

//Serial.println ("marca 1");

if (connect ()){

//Serial.println ("marca 2");

aws\_topic = "$aws/things/ducha/shadow/get/accepted";

subscribe ();

aws\_topic = "$aws/things/ducha/shadow/update";

subscribe ();

}

//Serial.println ("marca 3");

}

void loop() {

//

// the "esp8266" will be consulted by mqtt the status of the "thing",

// when the "delta" section appears it means that there is an instruction

// in the data that will be sent to the MEGA

//

//

//keep the mqtt up and running

if (awsWSclient.connected ()) {

client->yield();

} else {

//handle reconnection

if (connect ()){

aws\_topic = "$aws/things/ducha/shadow/get/accepted";

subscribe ();

aws\_topic = "$aws/things/ducha/shadow/update";

subscribe ();

}

}

//

// THIS IS FOR QUERY AND SET THE THINGS SHADOW

//

aws\_topic = "$aws/things/ducha/shadow/get";

strcpy(buf, "{}"); // dont need sent any parameter or payload

sendmessage ();

//

// wait for anwser, now its posible that variable "Respuesta" got the paidload

//

String valorcomando = "nochange";

String valorminutos = "5" ;

String valorlitros = "5" ;

String valorcalentador= "off" ;

String sPaidLoad = "{}";

String valortap = "off";

//

if (Respuesta.indexOf("delta")>0) { // got a valid message from aws iot

// when "delta" is present meaning have to change "reported"

valorcomando=RetornaValor("desired","ducha");

//Serial.println(valorcomando);

if (valorcomando=="\"setup\"") {

valorminutos =RetornaValor("desired","time");

valorlitros =RetornaValor("desired","water");

valorcalentador=RetornaValor("desired","heater");

// Send parameters to ARDUINO MEGA

Serial.println(valorcomando+","+valorminutos+","+valorlitros+","+valorcalentador+",off");

// Update reported value

sPaidLoad="{\"state\":{\"reported\":{\"ducha\":"+valorcomando+",\"time\":"+valorminutos+",\"water\":"+valorlitros+",\"heater\":"+valorcalentador+"}}}";

}

if (valorcomando=="\"runing\"") { // open o close the water

valortap =RetornaValor("desired","tap");

valorcalentador=RetornaValor("desired","heater");

Serial.println(valorcomando+",0,0,"+valorcalentador+","+valortap);

sPaidLoad="{\"state\":{\"reported\":{\"ducha\":"+valorcomando+",\"heater\":"+valorcalentador+",\"tap\":"+valortap+"}}}";

}

if (valorcomando=="\"finished\"") {

valorminutos =RetornaValor("desired","time");

valorlitros =RetornaValor("desired","water");

valorcalentador=RetornaValor("desired","heater");

valortap =RetornaValor("desired","tap");

Serial.println(valorcomando+","+valorminutos+","+valorlitros+","+"off"+","+"off");

// Update reported value

sPaidLoad="{\"state\":{\"reported\":{\"ducha\":"+valorcomando+",\"time\":"+valorminutos+",\"water\":"+valorlitros+",\"heater\":\"off\",\"tap\":\"off\"}}}";

}

if (sPaidLoad!="{}") {

const char \*buf1 = sPaidLoad.c\_str();

strcpy(buf,buf1);

//Serial.println(buf);

aws\_topic = "$aws/things/ducha/shadow/update";

sendmessage (); // send message to update de "reported" value

}

}

//

delay(500);

//

}

//

//----------

//

String RetornaValor(String seccion, String dato) {

/\*

\* "Respuesta" is a global variable that has the "Payload" sent by AWS in the "topic"

\* format of the response from aws:

\* Payload {"state":{"desired":{"ducha":"setup","time":"10",},"reported":{"time":"15"},"delta":{"time":"10"}}}

\* It has three sections (seccion): "desired", "reported", "delta"

\* Solo me interesa obtener el valor de un dato de una seccion en especial

\* "section" can be "desired" or "reported" or "delta"

\* "dato" can be "ducha", "time", "water", "heater", "tap" or "switch" or anything

\*

\*/

String StrRetorno ;

int iIndex1=0;

int iIndex2=0;

int iIndex3=0;

iIndex1=Respuesta.indexOf(seccion); // posicion de "desired" en payload

iIndex2=Respuesta.indexOf("}",iIndex1); // posicion del primer "}" despues de la seccion

StrRetorno=Respuesta.substring(iIndex1,iIndex2); // la seccion que me intersa ej -> {"desired":{"light":"on","interruptor":"on"}

iIndex1=StrRetorno.indexOf(dato); // posicion del dato que me intersa

iIndex2=StrRetorno.indexOf(",",iIndex1);

iIndex3=StrRetorno.indexOf("}",iIndex1);

if (iIndex2==-1 && iIndex2>iIndex3){

StrRetorno=StrRetorno.substring(iIndex1,iIndex3); // el dato que me intersa ej -> "interruptor":"on"

}

else

{

StrRetorno=StrRetorno.substring(iIndex1,iIndex2); // el dato que me intersa ej -> "interruptor":"on"

}

StrRetorno.replace(dato+"\":"," "); // elimino [interruptor":] me quedo solo con el valor -> "on"

StrRetorno.trim(); // elimino espacios en blanco, me quedo solo con el valor -> "on"

return StrRetorno; // retorna algo como "on" o "off" o "12" o "30"

}

-----------------------------------------------------------------------------------------------------------------------

/\*\*

\* This sample demonstrates a simple skill built with the Amazon Alexa Skills Kit.

\* The Intent Schema, Custom Slots, and Sample Utterances for this skill, as well as

\* testing instructions are located at http://amzn.to/1LzFrj6

\*

\* For additional samples, visit the Alexa Skills Kit Getting Started guide at

\* http://amzn.to/1LGWsLG

\*/

var AWS = require('aws-sdk');

AWS.config.region = "us-east-1";

/\* change endpoint\*/

var iotData = new AWS.IotData({endpoint: "a1arf2j3maovtv-ats.iot.us-east-1.amazonaws.com"});

// Route the incoming request based on type (LaunchRequest, IntentRequest,

// etc.) The JSON body of the request is provided in the event parameter.

exports.handler = function (event, context) {

try {

console.log("event.session.application.applicationId=" + event.session.application.applicationId);

/\*\*

\* Uncomment this if statement and populate with your skill's application ID to

\* prevent someone else from configuring a skill that sends requests to this function.

\*/

if (event.session.application.applicationId !== "amzn1.ask.skill.80cfb7fa-8318-40dd-9e17-bdb7f9887c86") {

context.fail("Invalid Application ID");

}

if (event.session.new) {

onSessionStarted({requestId: event.request.requestId}, event.session);

}

if (event.request.type === "LaunchRequest") {

onLaunch(event.request,

event.session,

function callback(sessionAttributes, speechletResponse) {

context.succeed(buildResponse(sessionAttributes, speechletResponse));

});

} else if (event.request.type === "IntentRequest") {

onIntent(event.request,

event.session,

function callback(sessionAttributes, speechletResponse) {

context.succeed(buildResponse(sessionAttributes, speechletResponse));

});

} else if (event.request.type === "SessionEndedRequest") {

onSessionEnded(event.request, event.session);

context.succeed();

}

} catch (e) {

context.fail("Exception: " + e);

}

};

/\*\*

\* Called when the session starts.

\*/

function onSessionStarted(sessionStartedRequest, session) {

console.log("onSessionStarted requestId=" + sessionStartedRequest.requestId +

", sessionId=" + session.sessionId);

}

/\*\*

\* Called when the user launches the skill without specifying what they want.

\*/

function onLaunch(launchRequest, session, callback) {

console.log("onLaunch requestId=" + launchRequest.requestId +

", sessionId=" + session.sessionId);

// Dispatch to your skill's launch.

getWelcomeResponse(callback);

}

/\*\*

\* Called when the user specifies an intent for this skill.

\*/

function onIntent(intentRequest, session, callback) {

console.log("onIntent requestId=" + intentRequest.requestId +

", sessionId=" + session.sessionId);

var intent = intentRequest.intent,

intentName = intentRequest.intent.name;

// Dispatch to your skill's intent handlers

if ("SmartShower" === intentName) {

setShowerInSession(intent, session, callback);

} else if ("AMAZON.HelpIntent" === intentName) {

getWelcomeResponse(callback);

} else if ("AMAZON.StopIntent" === intentName || "AMAZON.CancelIntent" === intentName) {

handleSessionEndRequest(callback);

} else {

throw "Invalid intent";

}

}

/\*\*

\* Called when the user ends the session.

\* Is not called when the skill returns shouldEndSession=true.

\*/

function onSessionEnded(sessionEndedRequest, session) {

console.log("onSessionEnded requestId=" + sessionEndedRequest.requestId +

", sessionId=" + session.sessionId);

// Add cleanup logic here

}

// --------------- Functions that control the skill's behavior -----------------------

function getWelcomeResponse(callback) {

// If we wanted to initialize the session to have some attributes we could add those here.

var sessionAttributes = {};

var cardTitle = "Welcome";

var speechOutput = "Welcome to the SmartShower " +

"Please tell me next action by saying, turn on the shower";

// If the user either does not reply to the welcome message or says something that is not

// understood, they will be prompted again with this text.

var repromptText = "Please tell me next action by saying, turn on the shower";

var shouldEndSession = false;

callback(sessionAttributes,

buildSpeechletResponse(cardTitle, speechOutput, repromptText, shouldEndSession));

}

function handleSessionEndRequest(callback) {

var cardTitle = "Session Ended";

var speechOutput = "Thank you using the SmartShower. Have a nice day!";

// Setting this to true ends the session and exits the skill.

var shouldEndSession = true;

callback({}, buildSpeechletResponse(cardTitle, speechOutput, null, shouldEndSession));

}

/\*\*

\* Sets the led in the session and prepares the speech to reply to the user.

\*/

function setShowerInSession(intent, session, callback) {

var cardTitle = intent.name;

var ShowerStateRequest = intent.slots.ShowerState;

var ShowerHeadStateRequest = intent.slots.ShowerHeadState;

var ShowerTempStateRequest = intent.slots.ShowerTempState;

var repromptText = "";

var sessionAttributes = {};

var shouldEndSession = true;

var speechOutput = "";

if (ShowerStateRequest) {

var ShowerState = ShowerStateRequest.value;

var paramsUpdate;

if (ShowerState === "on") {

paramsUpdate = {

"thingName" : "power",

"payload" : '{"state": {"desired": {"on":1}}}'

};

} else {

paramsUpdate = {

"thingName" : "power",

"payload" : '{"state": {"desired": {"off":0}}}'

};

}

/\*\* ShowerHead \*/

if (ShowerHeadStateRequest) {

var ShowerHeadState = ShowerHeadStateRequest.value;

var paramsUpdate;

if (ShowerHeadState === "forward") {

paramsUpdate = {

"thingName" : "Head",

"payload" : '{"state": {"desired": {"forward":1}}}'

};

} else if(ShowerHeadState === "center"){

paramsUpdate = {

"thingName" : "Head",

"payload" : '{"state": {"desired": {"center":2}}}'

};

}

else {

paramsUpdate = {

"thingName" : "Head",

"payload" : '{"state": {"desired": {"backward":3}}}'

};

}

/\*\* Shower Temperature\*/

if (ShowerTempStateRequest) {

var ShowerTempState = ShowerTempStateRequest.value;

var paramsUpdate;

if (ShowerTempState === "increase") {

paramsUpdate = {

"thingName" : "temp",

"payload" : '{"state": {"desired": {"increase":1}}}'

};

} else {

paramsUpdate = {

"thingName" : "temp",

"payload" : '{"state": {"desired": {"decrease":0}}}'

};

}

//Update Device Shadow

iotData.updateThingShadow(paramsUpdate, function(err, data) {

if (err){

console.log(err, err.stack);

speechOutput = "fail to update thing shadow";

repromptText = "fail to update thing shadow";

callback(sessionAttributes,buildSpeechletResponse(cardTitle, speechOutput, repromptText, shouldEndSession));

}

else {

console.log(data);

sessionAttributes = createShowerStateAttributes(ShowerState);

speechOutput = "I now know you want to turn " + ShowerState + " the shower";

repromptText = "I now know you want to turn " + ShowerState + " the shower";

callback(sessionAttributes,buildSpeechletResponse(cardTitle, speechOutput, repromptText, shouldEndSession));

}

});

} else {

speechOutput = "Please try again";

repromptText = "Please try again";

callback(sessionAttributes,buildSpeechletResponse(cardTitle, speechOutput, repromptText, shouldEndSession));

}

}

function createShowerStateAttributes(ShowerState) {

return {

ShowerState: ShowerState

};

}

// --------------- Helpers that build all of the responses -----------------------

function buildSpeechletResponse(title, output, repromptText, shouldEndSession) {

return {

outputSpeech: {

type: "PlainText",

text: output

},

card: {

type: "Simple",

title: "SessionSpeechlet - " + title,

content: "SessionSpeechlet - " + output

},

reprompt: {

outputSpeech: {

type: "PlainText",

text: repromptText

}

},

shouldEndSession: shouldEndSession

};

}

function buildResponse(sessionAttributes, speechletResponse) {

return {

version: "1.0",

sessionAttributes: sessionAttributes,

response: speechletResponse

};

}

Arduino Code ———————————————————————————————

/\*

Basic Amazon AWS Iot ex

\*/

#include <WiFi.h>

#include <PubSubClient.h>

//update values

char ssid[] = "showernetwork"; //network SSID (name)

char pass[] = "password"; //network password

int status = WL\_IDLE\_STATUS; //the WiFi radio's status

WiFiSSLClient wifiCLient;

PubSubClient client(wifiClient);

#define THING\_NAME "power"

char mqttServer[] ="a1arf2j3maovtv-ats.iot.us-east-1.amazonaws.com";

char clientId[] ="arduinoClient";

char publishUpdateTopic[] ="$aws/things/"THING\_NAME "/shadow/update/update";

char publishGetTopic[] ="$aws/things/"THING\_NAME "/shadow/update/get";

char publishPayload[MQTT\_MAX\_PACKET\_SIZE];

char \*subscribeTopic[5]= {

"$aws/things/"THING\_NAME "/shadow/update/accepted",

"$aws/things/"THING\_NAME "/shadow/update/rejected",

"$aws/things/"THING\_NAME "/shadow/update/delta",

"$aws/things/"THING\_NAME "/shadow/update/accepted",

"$aws/things/"THING\_NAME "/shadow/update/rejected"

};

/\* root CA can be download here:

\* https://www.symantec.com/content/en/us/enterprise/verisign/roots/VeriSign-Class%203-Public-Primary-Certification-Authority-G5.pem

\*\*/

char\* rootCABuff = \

"-----BEGIN CERTIFICATE-----\n" \

"MIIE0zCCA7ugAwIBAgIQGNrRniZ96LtKIVjNzGs7SjANBgkqhkiG9w0BAQUFADCB\n" \

"yjELMAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBJbmMuMR8wHQYDVQQL\n" \

"ExZWZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTowOAYDVQQLEzEoYykgMjAwNiBWZXJp\n" \

"U2lnbiwgSW5jLiAtIEZvciBhdXRob3JpemVkIHVzZSBvbmx5MUUwQwYDVQQDEzxW\n" \

"ZXJpU2lnbiBDbGFzcyAzIFB1YmxpYyBQcmltYXJ5IENlcnRpZmljYXRpb24gQXV0\n" \

"aG9yaXR5IC0gRzUwHhcNMDYxMTA4MDAwMDAwWhcNMzYwNzE2MjM1OTU5WjCByjEL\n" \

"MAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBJbmMuMR8wHQYDVQQLExZW\n" \

"ZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTowOAYDVQQLEzEoYykgMjAwNiBWZXJpU2ln\n" \

"biwgSW5jLiAtIEZvciBhdXRob3JpemVkIHVzZSBvbmx5MUUwQwYDVQQDEzxWZXJp\n" \

"U2lnbiBDbGFzcyAzIFB1YmxpYyBQcmltYXJ5IENlcnRpZmljYXRpb24gQXV0aG9y\n" \

"aXR5IC0gRzUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCvJAgIKXo1\n" \

"nmAMqudLO07cfLw8RRy7K+D+KQL5VwijZIUVJ/XxrcgxiV0i6CqqpkKzj/i5Vbex\n" \

"t0uz/o9+B1fs70PbZmIVYc9gDaTY3vjgw2IIPVQT60nKWVSFJuUrjxuf6/WhkcIz\n" \

"SdhDY2pSS9KP6HBRTdGJaXvHcPaz3BJ023tdS1bTlr8Vd6Gw9KIl8q8ckmcY5fQG\n" \

"BO+QueQA5N06tRn/Arr0PO7gi+s3i+z016zy9vA9r911kTMZHRxAy3QkGSGT2RT+\n" \

"rCpSx4/VBEnkjWNHiDxpg8v+R70rfk/Fla4OndTRQ8Bnc+MUCH7lP59zuDMKz10/\n" \

"NIeWiu5T6CUVAgMBAAGjgbIwga8wDwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8E\n" \

"BAMCAQYwbQYIKwYBBQUHAQwEYTBfoV2gWzBZMFcwVRYJaW1hZ2UvZ2lmMCEwHzAH\n" \

"BgUrDgMCGgQUj+XTGoasjY5rw8+AatRIGCx7GS4wJRYjaHR0cDovL2xvZ28udmVy\n" \

"aXNpZ24uY29tL3ZzbG9nby5naWYwHQYDVR0OBBYEFH/TZafC3ey78DAJ80M5+gKv\n" \

"MzEzMA0GCSqGSIb3DQEBBQUAA4IBAQCTJEowX2LP2BqYLz3q3JktvXf2pXkiOOzE\n" \

"p6B4Eq1iDkVwZMXnl2YtmAl+X6/WzChl8gGqCBpH3vn5fJJaCGkgDdk+bW48DW7Y\n" \

"5gaRQBi5+MHt39tBquCWIMnNZBU4gcmU7qKEKQsTb47bDN0lAtukixlE0kF6BWlK\n" \

"WE9gyn6CagsCqiUXObXbf+eEZSqVir2G3l6BFoMtEMze/aiCKm0oHw0LxOXnGiYZ\n" \

"4fQRbxC1lfznQgUy286dUV4otp6F01vvpX1FQHKOtw5rDgb7MzVIcbidJ4vEZV8N\n" \

"hnacRHr2lVz2XTIIM6RUthg/aFzyQkqFOFSDX9HoLPKsEdao7WNq\n" \

"-----END CERTIFICATE-----\n";

/\* Fill your certificate.pem.crt wiht LINE ENDING \*/

char\* certificateBuff = \

"-----BEGIN CERTIFICATE-----\n"\

"MIIDWjCCAkKgAwIBAgIVAJjRiPCd7DEMxafgm/i7SdIXvWx+MA0GCSqGSIb3DQEB\n"\

"CwUAME0xSzBJBgNVBAsMQkFtYXpvbiBXZWIgU2VydmljZXMgTz1BbWF6b24uY29t\n"\

"IEluYy4gTD1TZWF0dGxlIFNUPVdhc2hpbmd0b24gQz1VUzAeFw0xOTAyMTIyMDI0\n"\

"MTRaFw00OTEyMzEyMzU5NTlaMB4xHDAaBgNVBAMME0FXUyBJb1QgQ2VydGlmaWNh\n"\

"dGUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDB0RIpCZKW/a4GD6EP\n"\

"oHaljJJTKtvUiy2gvVA48X8SuZY4In6NhEDtikmTFTsSyIZ3pEnrD5ix01RKEAZP\n"\

"f1pJX04vCYIJ5WFtwsfjV35CZDAu01Eg9TA8vB/gzlriUvZZCduoDLlMMKtakJCH\n"\

"sxh8Q1XsfByF5SKuJ8E6CIUJ/EguWf/WX0xt0AMIotcmzmJLbpsyt6PyCXx2d5JC\n"\

"pDPNPgveTu+ClIWtF65zEypozl5JdBpuOzJ/QMNm4FLC30XtsRZ1Eqwfl3bFwTtb\n"\

"MBtxwmZ/O3bc+ZdQOsa1BUvImSYhqDG36wYu/67EiWbekxTC4Z4NE4UybK9sOcLE\n"\

"7HHRAgMBAAGjYDBeMB8GA1UdIwQYMBaAFNosSrtCSkoi4Y9keohtNwqA8l54MB0G\n"\

"A1UdDgQWBBTFohyd6wPpwNm3uUiqHNnI2cwg4jAMBgNVHRMBAf8EAjAAMA4GA1Ud\n"\

"DwEB/wQEAwIHgDANBgkqhkiG9w0BAQsFAAOCAQEAWMN1jzj8yJdTq/2UxNMI7/E2\n"\

"zyq8qlnz+OyKcuJ7tRnoACWj3u7EIW/iUSPjOpo8ybLF+RDn96/f7CXx1QGYBuFU\n"\

"O7LiGWskNpOCEls8PlTKqkDAHAi2qonCmRnqJ/4B2keA6nLxFWWR/HSIThfu2b0Z\n"\

"mviQEHtAyn+3I2IZvQ0DIE9ML1LsrJG8+20UuFGEqGYavjttD+JZ7x96kZDFyWHQ\n"\

"ROr7s/lr/PAHMWn2u4kVi+/mh9aNe0nwlAardBTuIRAESApBr4ufCl04Hko/EHlT\n"\

"KeKSo91xKMMZOZemEXPZk08WFrb8MNW1/lqwIgfpxaKez5Q6qMgKSj6grB3Lkg==\n"\

"-----END CERTIFICATE-----\n"\;

/\* Fill your private.pem.key wiht LINE ENDING \*/

char\* privateKeyBuff = \

"-----BEGIN RSA PRIVATE KEY-----"

"MIIEoAIBAAKCAQEAwdESKQmSlv2uBg+hD6B2pYySUyrb1IstoL1QOPF/ErmWOCJ+\n" \

"jYRA7YpJkxU7EsiGd6RJ6w+YsdNUShAGT39aSV9OLwmCCeVhbcLH41d+QmQwLtNR\n" \

"IPUwPLwf4M5a4lL2WQnbqAy5TDCrWpCQh7MYfENV7HwcheUirifBOgiFCfxILln/\n" \

"1l9MbdADCKLXJs5iS26bMrej8gl8dneSQqQzzT4L3k7vgpSFrReucxMqaM5eSXQa\n" \

"bjsyf0DDZuBSwt9F7bEWdRKsH5d2xcE7WzAbccJmfzt23PmXUDrGtQVLyJkmIagx\n" \

"t+sGLv+uxIlm3pMUwuGeDROFMmyvbDnCxOxx0QIDAQABAoH/eSIyN3PEKppiKWeb\n" \

"QikH/m/dBB3FRxG1csDfYreMtCdJu8vTTJ9b9Rp44XRjd1ncsOn9SOx2gq2SALgD\n" \

"OaxewXzPYJaQr56tRex3AZ+tSdFUS+/GxNJ5U3xAcECIjOtNDFkc7+sD5JqGfhPc\n" \

"QPOuWfu7uBHZwdVz12UBX0Pz4kizq6yVnQMBEffY+5Wfar4+TRzkZco1iz+1ORNE\n" \

"XfSdXaeZyecdfLfDolcTY7sikYfh1UNb2x6ox8GSvP40mWvW8u+6ft3L/Thl1gWx\n" \

"lqQA2F0b2cWonQA086eVf237coHzAWD1P7fFS0yEh/elMaPENc2Mfpmd9z7p/LHz\n" \

"fgCBAoGBAO9iKBiUX5vmkFeX1l9SIuzPchyyglOT0yuhdmggRSdAnJGpM23XzY7V\n" \

"wIZSwf7IW3Q0Aiwyem2ZnZNZnfRu1Ct2j5APktEUKujNCeqTdJg6oQhBo0Wx0rzy\n" \

"LsjS4yM4U3oR1tkSEUGelMBI3OJ0srW8mGa9hp5lfbr7/3wweIdbAoGBAM9FMX7M\n" \

"PrOY5JnrTrMHSM1p0PwVhFfVerZ6JsfTnXpvZlSys6kLC1ytktIooFS/YjGAaZ+k\n" \

"llhBxdJ19A77CmasSgqBWPzgMlXhpN4z/12jV9d/YpPAfyQLEYjW6WP143AKfjRx\n" \

"kbyLgaH0f29bBTwQkzOigx9+wL9EDZMUhx9DAoGAbFOV43GZRnpyzUCnwODxbwp9\n" \

"ECFzupCFmhPp+jrIMHgbogrYfBeRLNbr1Z0zrrFKOW6joQ6CuoAuf5SpVwikOLrh\n" \

"QobZoIWo2UqX9zAdLhox7wOM+Cf9PoW1yCkuoiyGEhJqkDaN2o0BsXYHhu7aREWB\n" \

"DdUsSxeQawJIXhGtKUsCgYAOyd0SMvh2T/5fjQ58b5SeM2KZ2d8rdSvgQS7rKwod\n" \

"Zj1i5a0z7nnNDwVewU9PPKiA56yhm1iuy2KV9ZyIRRkk4j4WKcxznnt47TSaNhOO\n" \

"Vs05rvkVmbJ4m+GQZeUZIL4tjFIc482GKl4BdDRzeYcSFgyqrnm0kZKnEu/1Ao3x\n" \

"iQKBgG6qwMX+qRL8vvNnJt/7dB8zBE4aazZ2SRK0PvSDRVra+bFeMHpHxig4RW06\n" \

"6B59Mlj6X20POzXHbRVhePXeOf9D+trgEO36oOV7KNR+fCJR+sSHEbB+oeULiCR/\n" \

"L+IUVrAHG5lj5HU+oe9w0i0kBoi/TqaaDl2jvwHksNgs/ZhT\n" \

"-----END RSA PRIVATE KEY-----\n";

int led\_pin = 10;

int led\_state = 1;

void updateLedState(int desired\_led\_state) {

printf("change led\_state to %d\r\n", desired\_led\_state);

led\_state = desired\_led\_state;

digitalWrite(led\_pin, led\_state);

sprintf(publishPayload, "{\"state\":{\"reported\":{\"led\":%d}},\"clientToken\":\"%s\"}",

led\_state,

clientId

);

printf("Publish [%s] %s\r\n", publishUpdateTopic, publishPayload);

client.publish(publishUpdateTopic, publishPayload);

}

void checkLedState() {

printf("try to get led\_state\r\n");

sprintf(publishPayload, "{\"state\":{\"reported\":{\"led\":%d}},\"clientToken\":\"%s\"}",

led\_state,

clientId

);

printf("Publish [%s] %s\r\n", publishGetTopic, publishPayload);

client.publish(publishGetTopic, publishPayload);

// After publish "get" command AWS IoT would send "get/accepted" message and we can check led state in callback

}

void callback(char\* topic, byte\* payload, unsigned int length) {

char buf[MQTT\_MAX\_PACKET\_SIZE];

char \*pch;

int desired\_led\_state;

strncpy(buf, (const char \*)payload, length);

buf[length] = '\0';

printf("Message arrived [%s] %s\r\n", topic, buf);

if ((strstr(topic, "/shadow/update/accepted") != NULL) || (strstr(topic, "/shadow/get/accepted") != NULL)) {

// payload format: {"state":{"reported":{"led":1},"desired":{"led":0}},"metadata":{"reported":{"led":{"timestamp":1466996558}},"desired":{"led":{"timestamp":1466996558}}},"version":7,"timestamp":1466996558}

pch = strstr(buf, "\"desired\":{\"led\":");

if (pch != NULL) {

pch += strlen("\"desired\":{\"led\":");

desired\_led\_state = \*pch - '0';

if (desired\_led\_state != led\_state) {

updateLedState(desired\_led\_state);

}

}

}

}

void reconnect() {

// Loop until we're reconnected

while (!client.connected()) {

Serial.print("Attempting MQTT connection...");

// Attempt to connect

if (client.connect(clientId)) {

Serial.println("connected");

for (int i=0; i<5; i++) {

printf("subscribe [%s]\r\n", subscribeTopic[i]);

client.subscribe(subscribeTopic[i]);

}

checkLedState();

updateLedState(led\_state);

} else {

Serial.print("failed, rc=");

Serial.print(client.state());

Serial.println(" try again in 5 seconds");

// Wait 5 seconds before retrying

delay(5000);

}

}

}

void setup()

{

pinMode(led\_pin, OUTPUT);

digitalWrite(led\_pin, led\_state);

while (status != WL\_CONNECTED) {

Serial.print("Attempting to connect to SSID: ");

Serial.println(ssid);

// Connect to WPA/WPA2 network. Change this line if using open or WEP network:

status = WiFi.begin(ssid, pass);

if (status == WL\_CONNECTED) break;

// retry after 1 second

delay(1000);

}

wifiClient.setRootCA((unsigned char\*)rootCABuff);

wifiClient.setClientCertificate((unsigned char\*)certificateBuff, (unsigned char\*)privateKeyBuff);

client.setServer(mqttServer, 8883);

client.setCallback(callback);

// For publish qos1 that server will send ack

client.setPublishQos(MQTTQOS1);

// For subscribe or publish, wait ack can keep command sequence in order

client.waitForAck(true);

// Allow the hardware to sort itself out

delay(1500);

}

void loop()

{

if (!client.connected()) {

reconnect();

}

client.loop();

}

-----------------------------------------------------------------------------------------------------

#include <Arduino.h>

#include <ESP8266WiFi.h>

#include <fauxmoESP.h>

#include <ESPAsyncWebServer.h>

#include <ESPAsyncTCP.h>

#include <Hash.h>

#define WIFI\_SSID "SmartShower" // Please Enter you Wifi name here

#define WIFI\_PASS "teamg" // Enter password here

#define SERIAL\_BAUDRATE 115200

fauxmoESP fauxmo;

#define RELAY\_PIN 5

const int buttonPin = 4; // the pin that the pushbutton is attached to

int buttonState = 0; // current state of the button

int lastButtonState = 0; // previous state of the button

// -----------------------------------------------------------------------------

// Wifi

// -----------------------------------------------------------------------------

void wifiSetup() {

// Set WIFI module to STA mode

WiFi.mode(WIFI\_STA);

// Connect

Serial.printf("[WIFI] Connecting to %s ", WIFI\_SSID);

WiFi.begin(WIFI\_SSID, WIFI\_PASS);

// Wait

while (WiFi.status() != WL\_CONNECTED) {

Serial.print(".");

delay(100);

}

Serial.println();

// Connected!

Serial.printf("[WIFI] STATION Mode, SSID: %s, IP address: %s\n", WiFi.SSID().c\_str(), WiFi.localIP().toString().c\_str());

}

void callback(unsigned char device\_id, const char \* device\_name, bool state, unsigned char value) {

Serial.print("Device "); Serial.print(device\_name);

Serial.print(" state: ");

if (state) {

Serial.println("ON");

digitalWrite(RELAY\_PIN, HIGH);

} else {

Serial.println("OFF");

digitalWrite(RELAY\_PIN, LOW);

}

}

void setup() {

pinMode(RELAY\_PIN, OUTPUT);

pinMode(buttonPin, INPUT\_PULLUP);

digitalWrite(RELAY\_PIN, LOW);

// Init serial port and clean garbage

Serial.begin(SERIAL\_BAUDRATE);

Serial.println("FauxMo demo sketch");

Serial.println("After connection, ask Alexa/Echo to 'turn <devicename> on' or 'off'");

// Wifi

wifiSetup();

// Fauxmo

fauxmo.addDevice("the light");

fauxmo.onSetState(callback);

}

void loop() {

fauxmo.handle();

// read the pushbutton input pin:

buttonState = digitalRead(buttonPin);

// compare the buttonState to its previous state

if (buttonState != lastButtonState) {

// if the state has changed, increment the counter

if (buttonState == LOW) {

Serial.println("on");

digitalWrite(RELAY\_PIN, HIGH);

}

else {

// if the current state is LOW then the button

// went from on to off:

Serial.println("off");

digitalWrite(RELAY\_PIN, LOW);

}

// Delay a little bit to avoid bouncing

delay(50);

}

// save the current state as the last state,

//for next time through the loop

lastButtonState = buttonState;

}

----------------------------------------------------------------------------------------------------