#include <ESP8266WiFi.h> //https://github.com/esp8266/Arduino

//needed for library

#include <DNSServer.h>

#include <ESP8266WebServer.h>

#include <WiFiManager.h> //https://github.com/tzapu/WiFiManager

#include <ArduinoJson.h>

#include <DHT.h>

#define DHTPIN D6

#define PUMPPIN D2

#define SEEDLING D3

#define GROWBED D4

#define TRIGGER D5

#define ECHO D0

#define BLUEPIN D1

#define REDPIN D7

#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);

ESP8266WebServer server(80);

const char\* host = "dev.crofters.in";

const char\* hoststring = "Host: http://dev.crofters.in";

String ssid;

String password;

String name;

boolean newSSID = false;

String deviceid = "test";

float humidity,temp\_f,distance; // Values read from sensor

float humiditythreshold=50,tempthreshold=30,distancethreshold=32.5;

int pump,filter,pumpstate;

int bluebrightness = 0; // how bright the LED is (0 = full, 512 = dim, 1023 = off)

int redbrightness = 0; // how bright the LED is (0 = full, 512 = dim, 1023 = off)

String webString=""; // String to display

// Generally, you should use "unsigned long" for variables that hold time

unsigned long dhtpreviousMillis = 0; // will store last temp was read

unsigned long switchpreviousMillis = 0;

unsigned long thresholdpreviousMillis = 0;

unsigned long dhtlatencypreviousMillis = 0;

unsigned long gcmpreviousMillis = 0;

unsigned long pumponpreviousMillis = 0;

unsigned long ledtimepreviousMillis = 0;

unsigned long statuspreviousMillis = 0;

int pumpon = 0;

unsigned long pumpoffpreviousMillis=0;

const long dhtinterval = 22000; // interval at which to send DHT sensor values

const long gcminterval = 60000\*10; // interval at which to send gcm notifications

const long dhtlatencyinterval = 2000;

const long switchinterval = 10000; // interval at which to ping for switch data

long pumpinterval = 60000\*20; // default 15 mins

const long pumpruntime = 60000\*3; // default 3 mins

const long thresholdinterval = 60000\*35; //it will check for change in threshold value every 30 mins

const long ledtimeinterval = 60000\*55;

void icroftsetup(void);

//Initial configuration for nodemcu

void icroftsetup(void){

char myIpString[24];

WiFiClient client;

IPAddress myIp = WiFi.localIP();

sprintf(myIpString, "%d.%d.%d.%d", myIp[0], myIp[1], myIp[2], myIp[3]);

String postdata = String("{\"deviceid\":\""+deviceid+"\",\"deviceip\":\"")+myIpString+"\"}";

client.connect(host,80);

delay(250);

if(client.connected()){

Serial.println("updating sensor data");

String url = "http://dev.crofters.in/api/updateIP?username=kevin";

url += "&ip=";

url += myIpString;

// This will send the request to the server

client.print(String("GET ") + url + " HTTP/1.1\r\n" +

"Host: " + host + "\r\n" +

"Connection: close\r\n\r\n");

String line = client.readStringUntil('\r');

while(client.available()){

char in = client.read();

//Serial.print(in);

}

while (client.connected() || client.available()) {

client.stop();

}

}else{

Serial.println("Not connected");

}

Serial.println("Before preparing switches");

// prepare all switches

pinMode(REDPIN, OUTPUT);

pinMode(BLUEPIN, OUTPUT);

// pinMode(FILTER, OUTPUT);

pinMode(SEEDLING, OUTPUT);

pinMode(PUMPPIN, OUTPUT);

pinMode(GROWBED, OUTPUT);

Serial.println("After preparing switches");

}

// to save wifi to device http://192.168.4.1/configure?

void setup() {

// put your setup code here, to run once:

Serial.begin(9600);

//WiFiManager

//Local intialization. Once its business is done, there is no need to keep it around

WiFiManager wifiManager;

wifiManager.setConfigPortalTimeout(180);

//reset settings - for testing

//wifiManager.resetSettings();

//fetches ssid and pass and tries to connect

//if it does not connect it starts an access point with the specified name

//here "AutoConnectAP"

//and goes into a blocking loop awaiting configuration

if(!wifiManager.autoConnect("icroft")) {

Serial.println("failed to connect and hit timeout");

//reset and try again, or maybe put it to deep sleep

ESP.reset();

delay(1000);

}

//if you get here you have connected to the WiFi

Serial.println("connected...yeey :)");

icroftsetup();

Serial.println("setup done :)");

}

//SETUP ENDS HERE

// custom function for accessing DHT11

void gettemperature() {

// Wait at least 2 seconds seconds between measurements.

// if the difference between the current time and last time you read

// the sensor is bigger than the interval you set, read the sensor

// Works better than delay for things happening elsewhere also

unsigned long currentMillis = millis();

if(currentMillis - dhtlatencypreviousMillis >= dhtlatencyinterval) {

// save the last time you read the sensor

dhtlatencypreviousMillis = currentMillis;

//prepare the DHT11 to sense.

dht.begin();

delay(500);

// Reading temperature for humidity takes about 250 milliseconds!

// Sensor readings may also be up to 2 seconds 'old' (it's a very slow sensor)

// humidity = (humidity+dht.readHumidity())/2; // Read humidity (percent)

humidity = dht.readHumidity() ;// Read humidity (percent)

temp\_f = (temp\_f+dht.readTemperature(true))/2; // Read temperature as Fahrenheit

Serial.println(humidity);

Serial.println(temp\_f);

// Check if any reads failed and exit early (to try again).

if (isnan(humidity) || isnan(temp\_f)) {

Serial.println("Failed to read from DHT sensor!");

//test test

temp\_f = 11;

humidity = 12;

return;

}

}

}

//update the switches (should map /)

void updateswitches() {

unsigned long currentMillis = millis();

String response;

bool begin = false;

WiFiClient client;

// Ping the node server once in 10 seconds to get the value of the switches

if(currentMillis - switchpreviousMillis >= switchinterval) {

// save the last time you read the sensor

switchpreviousMillis = currentMillis;

if(client.connect(host,80)){

Serial.println("updating switches data");

String url = "http://dev.crofters.in/api/updateValues?username=kevin";

// This will send the request to the server

client.print(String("GET ") + url + " HTTP/1.1\r\n" +

"Host: " + host + "\r\n" +

"Connection: close\r\n\r\n");

String line = client.readStringUntil('\r');

}

while(client.available()){

char in = client.read();

// Serial.print(in);

if (in == '{') {

begin = true;

}

if (begin) response += (in);

if (in == '}') {

break;

}

}

// Serial.print(response);

while (client.connected() || client.available()) {

client.stop();

}

Serial.println(response);

//Start parsing the JSON response obtained

int length = response.length()+1;

char json[length];

response.toCharArray(json,length);

StaticJsonBuffer<512> jsonBuffer;

JsonObject& root = jsonBuffer.parseObject(json);

// Test if parsing succeeds.

if (!root.success()) {

Serial.println("parseObject() update switches failed");

return;

}else{

int redlight = root["redlight"];

int bluelight = root["bluelight"];

int growbedlight = root["growbedlight"];

int seedlinglight = root["seedlinglight"];

int filter = root["filterswitch"];

int pump = root["pumpswitch"];

int test = root["relaytest"];

// seedling light

if (seedlinglight == 1) {

//write to the pins

digitalWrite(SEEDLING, HIGH);

Serial.println("seedlinglight on");

}

else {

digitalWrite(SEEDLING, LOW);

Serial.println("seedlinglight off");

}

if (growbedlight == 1) {

//write to the pins

digitalWrite(GROWBED, HIGH);

analogWrite(BLUEPIN,bluelight);

analogWrite(REDPIN,redlight);

Serial.println("growbedlight on");

}

else {

//write to the pins

digitalWrite(GROWBED, LOW);

Serial.println("growbedlight off");

}

/\*

if (filter == 1) {

//write to the pins

digitalWrite(FILTER, HIGH);

Serial.println("FILTER on");

}

else {

//write to the pins

digitalWrite(FILTER, LOW);

Serial.println("FILTER off");

}

\*/

if (pump == 1) {

//write to the pins

digitalWrite(PUMPPIN, HIGH);

Serial.println("PUMP on");

}

else {

//write to the pins

digitalWrite(PUMPPIN, LOW);

Serial.println("PUMP off");

}

if (test == 1) {

digitalWrite(D1, HIGH);

Serial.println("1on"); // turn the LED on (HIGH is the voltage level)

delay(1000);

digitalWrite(D2, HIGH);

Serial.println("2on"); // turn the LED on (HIGH is the voltage level)

delay(1000);

digitalWrite(D3, HIGH);

Serial.println("3on"); // turn the LED on (HIGH is the voltage level)

delay(1000);

digitalWrite(D4, HIGH);

Serial.println("4on"); // turn the LED on (HIGH is the voltage level)

delay(1000); // wait for a second

delay(5000);

digitalWrite(D1, LOW);

Serial.println("1OFF"); // turn the LED off by making the voltage LOW

delay(1000);

digitalWrite(D2, LOW);

Serial.println("2OFF"); // turn the LED off by making the voltage LOW

delay(1000);

digitalWrite(D3, LOW);

Serial.println("3OFF"); // turn the LED off by making the voltage LOW

delay(1000);

digitalWrite(D4, LOW); // turn the LED off by making the voltage LOW

Serial.println("4OFF");

delay(5000);

}

else {

Serial.println("no test ");

}

pumpinterval = 60000\*pump;

Serial.println("Switches updated");

}

}

}

//update the threshold for gcm

void updatethreshold() {

unsigned long currentMillis = millis();

String response;

bool begin = false;

WiFiClient client;

// Ping the node server once in 10 seconds to get the value of the switches

if(currentMillis - thresholdpreviousMillis >= thresholdinterval) {

// save the last time you read the sensor

thresholdpreviousMillis = currentMillis;

if(client.connect(host,80)){

Serial.println("updating threshold data");

client.println("GET /"+deviceid+"/updatethreshold HTTP/1.1");

client.println();

String line = client.readStringUntil('\r');

}

while(client.available()){

char in = client.read();

if (in == '{') {

begin = true;

}

if (begin) response += (in);

if (in == '}') {

break;

}

}

Serial.print(response);

while (client.connected() || client.available()) {

client.stop();

}

//Start parsing the JSON response obtained

int length = response.length()+1;

char json[length];

response.toCharArray(json,length);

StaticJsonBuffer<200> jsonBuffer;

JsonObject& root = jsonBuffer.parseObject(json);

// Test if parsing succeeds.

if (!root.success()) {

Serial.println("parseObject() update gcm failed");

return;

}else{

tempthreshold = root["tempthreshold"];

humiditythreshold = root["humiditythreshold"];

distancethreshold = root["distancethreshold"];

}

Serial.println("thresholds updated");

}

}

//Update the sensor values

void updatesensors() {

bool begin = false;

unsigned long currentMillis = millis();

WiFiClient client;

// Post to server once in 30 mins to update sensor value

if(currentMillis - dhtpreviousMillis >= dhtinterval) {

// save the last time you read the sensor

dhtpreviousMillis = currentMillis;

Serial.println("updating sensor data");

gettemperature(); //update the DHT11

Serial.println("before ultrasonic read");

long duration; // update ultrasonic sensor

pinMode(TRIGGER, OUTPUT);

digitalWrite(TRIGGER, LOW);

delayMicroseconds(5);

digitalWrite(TRIGGER, HIGH);

delayMicroseconds(10);

digitalWrite(TRIGGER, LOW);

pinMode(ECHO, INPUT);

duration = pulseIn(ECHO, HIGH);

distance = (duration/2) / 29.1;

String postdata = String("{\"temp\":")+temp\_f+",\"humidity\":"+humidity+",\"distance\":"+distance+"}";

Serial.println(postdata);

if(client.connect(host,80)){

Serial.println("updating sensor data");

String url = "http://dev.crofters.in/api/updateValues?username=kevin";

url += "&volume=";

url += distance;

url += "&temperature=";

url += temp\_f;

url += "&humidity=";

url += humidity;

// This will send the request to the server

client.print(String("GET ") + url + " HTTP/1.1\r\n" +

"Host: " + host + "\r\n" +

"Connection: close\r\n\r\n");

String line = client.readStringUntil('\r');

while(client.available()){

char in = client.read();

//Serial.print(in);

}

while (client.connected() || client.available()) {

client.stop();

}

}

}

}

//send

//Update the sensor values and send corresponding gcm message

void tempforgcm() {

unsigned long currentMillis = millis();

WiFiClient client;

// Post to server once in 30 mins to update sensor value

if(currentMillis - dhtpreviousMillis >= gcminterval) {

// save the last time you read the sensor

gcmpreviousMillis = currentMillis;

if(temp\_f > tempthreshold) {

String postdata = String("{\"temp\":")+temp\_f+"}";

Serial.println(postdata);

if(client.connect(host,80)){

Serial.println("updating switches data");

String url = "http://dev.crofters.in/api/updateValues?username=kevin&temperature=";

url += temp\_f;

// This will send the request to the server

client.print(String("GET ") + url + " HTTP/1.1\r\n" +

"Host: " + host + "\r\n" +

"Connection: close\r\n\r\n");

String line = client.readStringUntil('\r');

}

}

}

}

//humidity for gcm

void humidityforgcm() {

unsigned long currentMillis = millis();

WiFiClient client;

// Post to server once in 30 mins to update sensor value

if(currentMillis - gcmpreviousMillis >= gcminterval) {

// save the last time you read the sensor

gcmpreviousMillis = currentMillis;

if(humidity > humiditythreshold) {

String postdata = String("{\"humidity\":")+humidity+"}";

Serial.println(postdata);

if(client.connect(host,3000)){

Serial.println("updating switches data");

String url = "http://dev.crofters.in/api/updateValues?username=kevin&humidity=";

url += humidity;

// This will send the request to the server

client.print(String("GET ") + url + " HTTP/1.1\r\n" +

"Host: " + host + "\r\n" +

"Connection: close\r\n\r\n");

String line = client.readStringUntil('\r');

}

}

}

}

//waterlevelforgcm

void waterlevelforgcm() {

unsigned long currentMillis = millis();

WiFiClient client;

// Post to server once in 30 mins to update sensor value

if(currentMillis - gcmpreviousMillis >= gcminterval) {

// save the last time you read the sensor

gcmpreviousMillis = currentMillis;

if(distance > distancethreshold) {

String postdata = String("{\"distance\":")+distance+"}";

Serial.println(postdata);

if(client.connect(host,80)){

Serial.println("updating switches data");

String url = "http://dev.crofters.in/api/updateValues?username=kevin&volume=";

url += distance;

// This will send the request to the server

client.print(String("GET ") + url + " HTTP/1.1\r\n" +

"Host: " + host + "\r\n" +

"Connection: close\r\n\r\n");

String line = client.readStringUntil('\r');

}

}

}

}

//method to run pump

void runpump(){

if(pumpon==1)

{

unsigned long currentMillis = millis();

if(currentMillis - pumpoffpreviousMillis >= pumpruntime) {

pumpoffpreviousMillis = currentMillis;

digitalWrite(PUMPPIN,0);

Serial.println("Pump Stopped");

pumpon=0;

pumponpreviousMillis=millis();

}

}

else {

unsigned long currentMillis = millis();

if(currentMillis - pumponpreviousMillis >= pumpinterval) {

// save the last time you read the sensor

pumponpreviousMillis = currentMillis;

digitalWrite(PUMPPIN,1);

Serial.println("Pump Started");

pumpon = 1;

pumpoffpreviousMillis=millis();

}

}

}

void loop()

{

Serial.println(ssid);

delay(500);

if(WiFi.status() == WL\_DISCONNECTED || WiFi.status() == WL\_CONNECTION\_LOST){

Serial.println("Resetting loop");

ESP.reset();

}else{

runpump();

delay(250);

updateswitches();

delay(250);

updatesensors();

delay(250);

waterlevelforgcm();

delay(250);

humidityforgcm();

delay(250);

tempforgcm();

delay(250);

}

}