# IT Project: Smart Greenhouse

## Inleiding:

Dit is de documentatie van het project “Smart Greenhouse” van Robin De Voecht, student aan de Thomas More hogeschool in Katelijne.

Met het project Smart Greenhouse probeer ik een simpel kweekbakje volledig te automatiseren. Ik voorzie het plantje van water, voedingsstoffen, licht,… zonder ik zelf iets hoef te doen. Dit door verschillende sensoren toe te voegen. De data die ik ontvang van de sensoren zal worden bijgehouden en worden weergegeven in een grafiek op grafana doormiddel van “mqtt services”.

Deze IoT-technieken kunnen we gebruiken in veel verschillende projecten. Dit project is een goed voorbeeld van wat we op een groot schaal kunnen maken. Nu hebben we een klein kweekbakje volledig geautomatiseerd, maar stel je eens voor dat dit wordt toegepast bij een boerderij, of een teeltbedrijf, etc… Doormiddel van IoT kunnen we veel efficiënter en nauwkeuriger werken. Niet alleen dat, ook kunnen we vanop afstand onze teelt in het oog houden en indien nodig actie uitvoeren om de teelt de helpen.

## Gebruikte hardware:

Om dit project te realiseren hebben we natuurlijk verschillende componenten nodig. Deze componenten(microcontrollers, sensoren, actuatoren). Zorgen voor zowel dataopslag, communicatie en visualisatie. In het lijstje heb je van elk component een gedetailleerde uitleg.

### Algemene benodigdheden:

Hier is het lijstje van de algemene benodigdheden die ik heb gebruikt om mijn project te construeren. Dit is volledig je eigen keuze, het is dus mogelijk om andere materialen te gebruiken.

* **Kweekbakje**

Mijn kweekbakje zal in de houten doos staan. Het is net groot genoeg zodat er nog ruimte is voor de sensoren en bedrading.

* **Houten doos**

Ik gebruik een houten doos waarin in mijn kweekbakje ga in zetten. Een houten doos is handig voor gaten en vijzen in te boren om draden en sensoren vast te zetten.

* **Watertankje + waterleiding**

Aan de zijkant van mijn waterdoos zal een watertankje vasthangen, daarin zal de waterpomp en de waterlevel sensor in zitten (en water natuurlijk). Om water te kunnen transporteren gebruiken we een transparante waterleiding. Deze moet niet een bepaalde lengte hebben, het moet gewoon lang genoeg zijn om de afstand van het watertankje naar het kweekbakje te overbruggen.

* **Prototyping board**

Aan de prototyping board hangen verschillende componenten aan vast die nodig zijn voor de voeding. Ook de microcontroller en sensoren kunnen hier aan vast worden gesoldeerd.

* **5V adapter**

Dit is een adapter die er voor zorgt dat het hele project stroom krijgt. Aangezien al onze hardware maar maximum 5 volt nodig heeft is het niet nodig om een 12 volt adapter te gebruiken. De adapter zal van het stopcontact naar het prototyping board gaan en daar worden verdeeld in 3. Zodat de relay module en de microcontroller voorzien wordt van elektriciteit.

* **Elektrische bedrading + pin header connectors**

Rode en zwarte draden zodat je elektrische apparaten kunt verbinden met elkaar. Ook pin header connectors zijn nodig voor deze bedrading te koppelen.

* **Extra’s**

Met extra’s bedoel ik materialen die niet echt van belang zijn voor het project maar die wel nodig zijn om het te maken zoals een boormachine + nodige vijzen. Ook is er een soldeermachine nodig met het nodige soldeermateriaal.

### Microcontroller:

* **ESP32**

We gebruiken de esp32 om onze code op te uploaden. De GPIO pinnen gebruiken we voor het aansluiten van de actuatoren en sensoren.

De esp32 heeft ook ingebouwde wifi. Deze kunnen we gebruiken door het te verbinden met hetzelfde netwerk als onze laptop (of ander apparaat), zodat we de data kunnen vertonen op Grafana (hierover later meer uitleg).

Een extra pluspunt is dat het ook een zeer betaalbaar controller is met talrijke bibliotheken die we kunnen gebruiken voor het project.

### Sensoren:

* **PN532 RFID Reader module & RFID tags**

De pn532 kan worden ingezet om specifieke functies zoals identiteit en inventaris. Daarvoor heb je natuurlijk wel RFID-tags nodig. Het maakt gebruik van NFC (Near Field Communication), oftewel draadloze communicatie op korte afstand.

Wanneer we een RFID-tag tegen onze pn532 reader houden, zal de informatie van het plantje (die we op voorhand hebben ingesteld) worden weergegeven. Gebruiken we een andere RFID-tag, kunnen we info krijgen van een ander plantje.

* **Water level sensor**

Aangezien we het kweekbakjes voorzien van een waterpompje die automatisch water geeft aan het plantje, moeten we een water level sensor hebben die ervoor zorgt dat we weten wanneer het tankje met water bijna leeg is. Hij wordt geplaatst in het watertankje en is verbonden met een ledlampje. Wanneer het water bijna op is, zal het lampje branden en zien we dat het tankje moet bijgevuld worden.

#### Code waterlevel sensor:

We sluiten de sensor aan op pin 27, en de ledPin op pin 26. We hebben 2 tresholds, namelijk een waterlevel treshold en een lowleveltreshold. De waarde hiervan kan je zelf kiezen, ik heb ze op 800 en 500 gezet. Bij een hogere waarde zal de sensor sneller een melding geven.

const int sensorPin = 27;

const int waterLevelThreshold = 800;

const int lowLevelThreshold = 500;

const int ledPin = 14;

We gebruiken de functie ReadWaterLevel, deze wordt opgeroepen in de loop functie (net zoals alle andere functies van dit project. We slagen de waarde die we krijgen op in sensorValue. Indien deze waarde lager is dan onze lowLeveltreshold (dus wanneer er weinig water is) zetten we de waarde van onze ledpin op HIGH, zo zal het lampje aangaan.

Uiteindelijk sturen we de data door naar de database doormiddel van client.publish.

void ReadWaterLevel() {

  int sensorValue = analogRead(sensorPin);  //

  if (sensorValue >= waterLevelThreshold) {

    Serial.println("Water detected!");  //

    client.publish(waterLevel\_topic, String(waterLevelThreshold).c\_str());

    // ANDERE DINGENS

    digitalWrite(ledPin, LOW);

  } else if (sensorValue < lowLevelThreshold) {

    Serial.println("Warning: Low water level detected!");

    // LEDlamp aan wanneer water level laag is

    digitalWrite(ledPin, HIGH);

    client.publish(waterLevel\_topic, String(lowLevelThreshold).c\_str());

  }

  delay(1000);

}

* **Bme280 temperatuur & vochtigheidssensor**

De BME280 is een multifunctionele, nauwkeurige en goedkope sensor die we gebruiken en plaatsen op het prototyping board. Deze meet zowel de temperatuur en de vochtigheid rondom het plantje. Het kan ook dienen als luchtdruk sensor, maar dit hebben we niet nodig in het project. Hij staat ook in verbinding met de ventilator. Wanneer het te warm wordt, zal de ventilator beginnen draaien en het plantje afkoelen.

#### Code BME280:

We hebben 2 bibliotheken nodig voor de BME280.

We hebben een functie ReadBME280 aangemaakt die wordt opgeroepen in de loop functie. De temperatuur die we inlezen wordt geprint op het lcd scherm (ook de seriële monitor). Hij wordt ook tegelijkertijd gepubliceerd zodat hij wordt opgeslagen in de database.

Wanneer de temperatuur groter is dan een bepaalde treshold zal de ventilator pin (de relaypin waar de ventilator aan gekoppeld is) aangaan. Anders gaat die terug uit.

* #include <Adafruit\_BME280.h>
* #include <Adafruit\_Sensor.h>
* void ReadBME280() {
* Serial.print("Temp: ");
* Serial.println(bme.readTemperature());
* lcd.clear();
* lcd.setCursor(0, 0);
* lcd.print("Temp: ");
* lcd.print(bme.readTemperature());
* lcd.print(" \*C");
* lcd.setCursor(0, 1);
* lcd.print("Hum: ");
* lcd.print(bme.readHumidity());
* client.publish(Temp\_topic, String(bme.readTemperature()).c\_str());
* if(bme.readTemperature() > TemperatuurTreshold){
* digitalWrite(VentilatorPin, HIGH);
* } else{
* digitalWrite(VentilatorPin, LOW);
* }
* delay(1000);
* }
* **Ds18b20 temperatuursensor**

De DS18B20 is een betrouwbare en waterdichte temperatuursensor. Deze wordt geplaatst in de grond. Hij staat niet in verbinding met een actuator, dus wanneer de temperatuur laag / hoog wordt zal er niets gebeuren. Toch is het heel handig om deze te hebben in het project. Met enkel de data zijn we natuurlijk al heel veel. De data die hij verzameld zal worden opgeslagen en vertoond op Grafana. Zo weten we steeds wat de temperatuur is in de bodem en weten we ook of we iets moeten veranderen indien het te warm of te koud.

#### Code DS18B20:

Dit zijn de nodige bibliotheken om er voor te zorgen dat de sensor werkt. Ook verbinden we de sensor aan pin 5 van de esp32. We initialiseren OneWire en DallasTemperature om uiteindelijk metingen van de sensor te verkrijgen.

#include <OneWire.h>

#include <DallasTemperature.h>

const int oneWireBus = 5;

OneWire oneWire(oneWireBus);

DallasTemperature sensors(&oneWire);

In de setup gebruiken we sensors.begin(); om de sensor op te starten en de metingen te ontvangen.

  sensors.begin();

We maken de functie ReadDS18B20 aan en roepen ze op in de void loop. Als eerst vragen we de temperatuur metingen op en slagen we ze tijdelijk op in tempC. Indien de gemeten temperatuur -127 graden celcius is betekent het dat de metingen niet worden doorgegeven en dan zal er hoogstwaarschijnlijk een fout staan in de code. Als we een normale geloofwaardige temperatuur krijgen dan printen we deze af op het lcd scherm. In de code wordt het ook afgeprint op de seriële monitor, dit is enkel ter controle en mag weg gelaten worden. Uiteindelijk sturen we de succesvolle metingen ook door naar de database doormiddel van client.publish.

void ReadDS18B20() {

  sensors.requestTemperatures();

  float tempC = sensors.getTempCByIndex(0);

  if (tempC != -127.00) {

    Serial.print("Grond Temperatuur: ");

    Serial.print(tempC);

    Serial.print(" \*C");

    lcd.clear();

    lcd.setCursor(0, 0);

    lcd.print("Temp Grond:");

    lcd.setCursor(0, 1);

    lcd.print(tempC);

    lcd.print(" \*C");

      client.publish(groundTemp\_topic, String(tempC).c\_str());

  } else {

    Serial.print("Fout bij lezen van temp");

  }

  delay(1000);

}

* **Capacitieve bodemvochtsensor**

De capacitieve bodemvocht sensor, een moeilijke naam voor een eenvoudige sensor. Hij wordt net zoals de ds18b20 geplaatst in de grond. Hij staat in verbinding met het waterpompje. Als de grond te droog is, zal er water worden toegevoegd aan het bakje.

#### Code bodemvochtsensor:

De funtie ReadBodemVocht wordt opgeroepen in de loop functie. De waarde die wordt gelezen wordt afgeprint op het lcd scherm. Tegelijkertijd worden de waardes ook doorgestuurd naar de mqtt-broker. Indien de grond te droog is zal de waterpomppin van de relay module aan gaan en zal er water stromen.

* void ReadBodemVocht() {
* bodemValue = analogRead(sensorPin);
* // Print the sensor value to the serial monitor
* Serial.print("Soil Moisture Value: ");
* Serial.println(bodemValue);
* lcd.clear();
* lcd.setCursor(0, 0);
* lcd.print("Bodemvocht: ");
* lcd.setCursor(0, 1);
* lcd.print(bodemValue);
* client.publish(moisture\_topic, String(bodemValue).c\_str());
* if(bodemValue < BodemvochtTreshold){
* digitalWrite(WaterPompPin, HIGH);
* } else{
* digitalWrite(WaterPompPin, LOW);
* }
* // Wait 1 second before taking another reading
* delay(1000);
* }
* **Lichtafhankelijke weerstand (LDR) + ledstrip**

De LDR is een eenvoudige en nauwkeurige lichtsensor en wordt geplaatst in onze plantenbak. Hij meet voortdurend de lichtsterkte en zal er ook voor zorgen dat het plantje genoeg licht krijgt. Dit komt doordat hij is verbonden met een ledstrip. Wanneer er te weinig licht aanwezig is zal de ledstrip aanspringen.

#### Code LDR:

Initialiseren van de LDR en de ledstrip, we geven ook het aantal leds door zodat de lampjes juist kunnen worden afgesteld. Om de ledstrip te laten werken is er ook een extra librarie voor nodig, namelijk de adafruit\_neopixel librarie.

#include <Adafruit\_NeoPixel.h>

const int ldrPin = 34;

//ledstrip

#define pin 18

#define NUM\_leds 30

Adafruit\_NeoPixel strip = Adafruit\_NeoPixel(NUM\_leds, pin, NEO\_GRB + NEO\_KHZ800);

We moeten de ledstrip dan ook initialiseren in de setup.

strip.begin();

strip.show();

Met de functie void ReadLDR, die wordt opgeroepen in de loop, zorgen we ervoor dat de metingen van de ldr worden weergegeven op het lcd scherm en ook op de seriële monitor.

Indien de opgemeten waarde hoger is lager is dan 3000, dan zal de ledstrip aanspringen. Je kan een kleur meegeven door 3 getallen (lager dan 355) in te vullen na strip.color. Ikzelf heb gekozen voor een warme kamerkleur.

void ReadLDR() {

  int ldrValue = analogRead(ldrPin);

  lcd.clear();

  Serial.print("Lichtintensiteit: ");

  Serial.println(ldrValue);

  lcd.setCursor(0, 0);

  lcd.print("Licht: ");

  lcd.print(ldrValue);

  client.publish(Light\_topic, String(ldrValue).c\_str());

  if (ldrValue < 3000) {

    for (int i = 0; i < strip.numPixels(); i++) {

      strip.setPixelColor(i, strip.Color(246, 205, 139));

    }

    strip.show();

  } else {

    for (int i = 0; i < strip.numPixels(); i++) {

      strip.setPixelColor(i, strip.Color(0, 0, 0));  // Zet alle LED's uit

    }

    strip.show();

  }

  delay(1000);

}

### Actuatoren:

* **Waterpomp 12V**

Om er voor te zorgen dat het water van de tank naar het kweekbakje komt gebruiken we een waterpomp die 12 volt aankan. Maar wanneer ik hier 12 volt op aansluit, is het water zeer krachtig, daarom geven we het maximaal 5 volt. We willen het plantje natuurlijk niet beschadigen of te veel water geven. Het water gaat van de tank naar het kweekbakje via een transparante waterleiding.

* **Ventilator**

Indien het te warm is, wordt de ventilator ingeschakeld om de omgeving af te koelen. Deze ventilator kan ook aangesloten worden op 12 volt. Maar omdat alle andere componenten maximaal 5 volt aankunnen, laten we de ventilator ook op 5 volt draaien zodat we niet extra converters moeten aansluiten. De ventilator wordt vastgemaakt aan de zijkant van de houten doos. Er wordt een gat geboord zodat de lucht naar binnen kan worden geblazen .

* **Relay module, 5V, 4 channels.**

Een relay module functioneert zoals een schakelaar. In onze code kunnen we ervoor zorgen dat channel 1 (bijvoorbeeld) aan gaat. Zo zal er elektriciteit kunnen doorstromen en zal het apparaat dat aan dat channel vast hangt, aanspringen. Aan channel 1 van onze relay hangt de waterpomp aan vast. De ventilator is verbonden met channel 2. De andere 2 channels hebben we niet nodig. Hier is een voorbeeldje van hoe de code voor de relay er uit ziet:

#### Code Relay module:

Veel code is er niet van de relay module. We moeten eigenlijk enkel de pinnen waarop onze apparaten aan zijn aangesloten initialiseren. De ventilator is aangesloten met pin 17. De Waterpomppin met pin 16. Ik maak ook 2 tresholds aan, wanneer de sensorwaardes boven of onder deze treshold gaan, zal het bijhorend kanaal aangaan en gaat het apparaat dat er aan vast hangt ook aan.

//relay

const int VentilatorPin = 25;

const int WaterPompPin = 26;

const int BodemvochtTreshold = 1400;

const int TemperatuurTreshold = 20;

Een voorbeeld van de relaycode bij een sensor:

if(bodemValue < BodemvochtTreshold){

    digitalWrite(WaterPompPin, HIGH);

  } else{

    digitalWrite(WaterPompPin, LOW);

  }

### Communicatie:

* **LCD L2c display**

Op het lcd scherm kunnen we alle data lezen die van belang zijn om het plantje te doen groeien. Van temperatuur tot hoeveelheid vocht. Het zal worden vastgemaakt vanonder binnenin de houten doos.

#### Code LCD:

Voor het lcd scherm te laten werken hebben we de liquidcrystal\_I2C library nodig. Door de l2c hebben we maar 2 pins die we aan een GPIO pin moeten aansluiten. Deze zijn pin 22 en pin 21.

#include <LiquidCrystal\_I2C.h>

#define LCD\_ADDRESS 0x27

#define LCD\_COLUMNS 16

#define LCD\_ROWS 2

LiquidCrystal\_I2C lcd(LCD\_ADDRESS, LCD\_COLUMNS, LCD\_ROWS);

Lcd.init en lcd.backlight zijn nodig om het apparaat op te starten. De clear functie is om er voor te zorgen dat er geen tekst op voorhand op het scherm komt te staan.

lcd.clear();

lcd.init();

lcd.backlight();

* **Raspberry pi**

Om ervoor te zorgen dat we op onze laptop, op Grafana de data die we verzamelen kunnen bekijken, hebben we een raspberry pi nodig. Dat is een soort van mini computer die we net zoals de esp32 verbinden met het internet. Met enkele stukken code kunnen we de data naar de raspberry pi sturen en ze verwerken op Grafana.

## MQTT Verbinding:

Een mqtt verbinding, oftewel een Message Queuing Telemetry Transport is een manier van communicatie waarmee je berichten tussen apparaten kan verzenden. Het maakt gebruik van publiceren en abonneren. Hiervoor heb je een MQTT-broker nodig. In het geval van mijn project is dit een raspberry pi. De raspberry pi is een centrale server die de berichten ontvangt en doorstuurt naar apparaten die erop zijn geabonneerd. Hij zorgt er ook voor dat de data automatisch wordt opgeslagen in een database. Hierdoor krijgen we 2 verschillende code voor een succesvolle mqtt-verbinding. Een stukje code staat geschreven op de esp32, de andere code staat geschreven op de Raspberry Pi.

### Code esp32:

Allereerst heb je 2 bibliotheken nodig alvorens je met mqtt kan beginnen. We gebruiken de wifi en de pubsubclient bibliotheek. Om alles juist te configureren, moeten we de juiste internet gegevens in onze code zetten. Op school is het netwerk ssid “embed” en het wachtwoord “weareincontrol”. Bij mij thuis zal dit dus anders zijn. Ook het ip-adres, poort, gebruikersnaam, wachtwoord en clientID van de mqtt-broker moeten we in onze code zetten. Ten slotte moeten we ook nog onze topics doorgeven waarop we met de mqtt-broker kunnen abonneren. Ik heb voor elke sensor een aparte topic aangemaakt.

#include <WiFi.h>

#include <PubSubClient.h>

const char\* ssid = "embed";               //Verander

const char\* password = "weareincontrol";  //verander

const char\* mqttServer = "192.168.1.108";  //verander

const int mqttPort = 1883;

const char\* mqttUser = "\*\*\*\*\*";

const char\* mqttPassword = "\*\*\*\*\*\*\*";

const char\* clientID = "client\_livingroom";  // MQTT client ID

const char\* moisture\_topic = "home/livingroom/moisture";

const char\* groundTemp\_topic = "home/livingroom/bodemTemp";

const char\* Temp\_topic = "home/livingroom/Temp";

const char\* Light\_topic = "home/livingroom/Light";

const char\* waterLevel\_topic = "home/livingroom/waterLevel";

WiFiClient espClient;

PubSubClient client(espClient);

We maken 2 functies aan namelijk setup\_wifi() en reconnect(). Deze functies zorgen ervoor dat je verbonden bent met het internet en dat je ook een succesvolle verbinding hebt met de mqtt-broker. Indien er iets niet klopt, zal er een foutmelding worden gegeven in onze seriële monitor.

void setup\_wifi() {

  delay(10);

  Serial.println();

  Serial.print("Connecting to ");

  Serial.println(ssid);

  WiFi.begin(ssid, password);

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

    delay(500);

    Serial.print(".");

  }

  Serial.println("");

  Serial.println("WiFi connected");

  Serial.println("IP address: ");

  Serial.println(WiFi.localIP());

}

void reconnect() {

  while (!client.connected()) {

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

    if (client.connect(clientID, mqttUser, mqttPassword)) {

      Serial.println("connected");

    } else {

      Serial.print("failed, rc=");

      Serial.print(client.state());

      delay(2000);

    }

  }

}

In de setup starten we de wifi en de server van de broker met volgende code:

setup\_wifi();

client.setServer(mqttServer, mqttPort);

In de loop functie zorgen we dat er steeds een verbinding is, indien die er niet is, zal de functie reconnect steeds worden opgeroepen tot er wel een verbinding is.

if (!client.connected()) {

    reconnect();

  }

  client.loop();

Al die code heb je nodig om een stabiele verbinding te verkrijgen tussen de mqtt-broker en de esp32. Wanneer je geen foutmeldingen hebt kan je de data van de sensoren publiceren met client.publish(topic, String(data).c\_str());

Zorg ervoor dat de topic hetzelfde is als de topic die je meegeeft bij de configuratie. Ook is het handig dat je de juiste sensordata geeft bij de juiste topic. Enkele voorbeeldjes hiervan zijn:

client.publish(Light\_topic, String(ldrValue).c\_str());

client.publish(waterLevel\_topic, String(lowLevelThreshold).c\_str());

client.publish(Temp\_topic, String(bme.readTemperature()).c\_str());

client.publish(groundTemp\_topic, String(tempC).c\_str());

### Code MQTT-Broker

Bij de mqtt-broker moeten we ook juist configureren zodat er een goede verbinding is met de esp32. We importeren eerst 3 bibliotheken, NamedTuple, paho.mqqt.client en influxdb. De NamedTuple bibliotheel zorgt ervoor dat je bepaalde objecten, waardes, elementen een benaming kunt geven waardoor je makkelijker en overzichtelijker kan werken. Paho.mqqt.client zorgt voor de mqtt-verbinding met de microcontroller. En influxDB zorgt ervoor dat we de data die we ontvangen, kunnen opslagen en weergeven op Grafana.

import re

from typing import NamedTuple

import paho.mqtt.client as mqtt

from influxdb import InfluxDBClient

Nadat je een account hebt gemaakt op influxdb, zet je de gebruikersnaam en wachtwoord mee in de code. Geef een naam aan je database, in mijn geval is het “weather\_stations”. Het influxdb\_address is bij mij localhost, dat komt omdat ik op dezelfde host als de broker werk. Indien dat een ander is, moet je het juiste adres geven.

Ook moeten we gegevens ingeven die overeenkomen met de gegevens bij de esp32 voor de mqtt-verbining. Gebruik hetzelfde address, gebruikersnaam en wachtwoord als hetgeen dat je eerder hebt gebruikt.

Maak dan een nieuwe influxdb\_client aan.

INFLUXDB\_ADDRESS = 'localhost'

INFLUXDB\_USER = '\*\*\*\*\*'

INFLUXDB\_PASSWORD = '\*\*\*\*\*\*\*'

INFLUXDB\_DATABASE = 'weather\_stations'

MQTT\_ADDRESS = '192.168.1.108'

MQTT\_USER = '\*\*\*\*\*'

MQTT\_PASSWORD = '\*\*\*\*\*'

MQTT\_TOPIC = 'home/+/+'

MQTT\_REGEX = 'home/([^/]+)/([^/]+)'

MQTT\_CLIENT\_ID = 'MQTTInfluxDBBridge'

influxdb\_client = InfluxDBClient(INFLUXDB\_ADDRESS, 8086, INFLUXDB\_USER,

INFLUXDB\_PASSWORD, None)

Dit zorgt voor een overzichtelijke datastructuur waarin je de sensordata kan opslaan.

class SensorData(NamedTuple):

    location: str

    measurement: str

    value: float

Deze functie wordt aangeroepen wanneer de client verbinding maakt met de broker. Het abonneert ook op de topics.

def on\_connect(client, userdata, flags, rc):

    """ The callback for when the client receives a CONNACK response from the server."""

    print('Connected with result code ' + str(rc))

    client.subscribe(MQTT\_TOPIC)

De parser voor de mqtt berichten zal none retourneren wanneer er geen data wordt doorgestuurd, anders zal hij deze opslagen met de juiste locatie, measurement en payload.

def \_parse\_mqtt\_message(topic, payload):

    match = re.match(MQTT\_REGEX, topic)

    if match:

        location = match.group(1)

        measurement = match.group(2)

        if measurement == 'status':

            return None

        return SensorData(location, measurement, float(payload))

    else:

        return None

In deze funtie worden de gegevens van de sensor doorgestuurd naar influxDB.

def \_send\_sensor\_data\_to\_influxdb(sensor\_data):

    json\_body = [

        {

            'measurement': sensor\_data.measurement,

            'tags': {

                'location': sensor\_data.location

            },

            'fields': {

                'value': sensor\_data.value

            }

        }

    ]

    influxdb\_client.write\_points(json\_body)

On message zal worden opgeroepen wanneer er een bericht wordt ontvangen op een geabonneerd topic. Het zorgt ervoor dat de gegevens wordt geanalyseerd en geschreven op een manier zodat de computer deze makkelijk kan verwerken.

def on\_message(client, userdata, msg):

    """The callback for when a PUBLISH message is received from the server."""

    print(msg.topic + ' ' + str(msg.payload))

    sensor\_data = \_parse\_mqtt\_message(msg.topic, msg.payload.decode('utf-8'))

    if sensor\_data is not None:

        \_send\_sensor\_data\_to\_influxdb(sensor\_data)

Deze functie controleert of er een influxDB-database bestaat met de naam die je hebt ingegeven (in mijn geval weather\_stations). Indien deze niet bestaat zal hij er een extra aanmaken. Vervolgens wordt er overgeschakeld naar de juiste database.

def \_init\_influxdb\_database():

    databases = influxdb\_client.get\_list\_database()

    if len(list(filter(lambda x: x['name'] == INFLUXDB\_DATABASE, databases))) == 0:

        influxdb\_client.create\_database(INFLUXDB\_DATABASE)

    influxdb\_client.switch\_database(INFLUXDB\_DATABASE)

In de mainfunctie wordt de influxDB-database geïnitialiseerd en geconfifureerd met de inloggegevens en de callbackfunctie. Vervolgens maakt hij verbinding met de broker en begint hij de gegevens te verwerken in een oneindige lus.

def main():

    \_init\_influxdb\_database()

    mqtt\_client = mqtt.Client(MQTT\_CLIENT\_ID)

    mqtt\_client.username\_pw\_set(MQTT\_USER, MQTT\_PASSWORD)

    mqtt\_client.on\_connect = on\_connect

    mqtt\_client.on\_message = on\_message

    mqtt\_client.connect(MQTT\_ADDRESS, 1883)

    mqtt\_client.loop\_forever()

if \_\_name\_\_ == '\_\_main\_\_':

    print('MQTT to InfluxDB bridge')

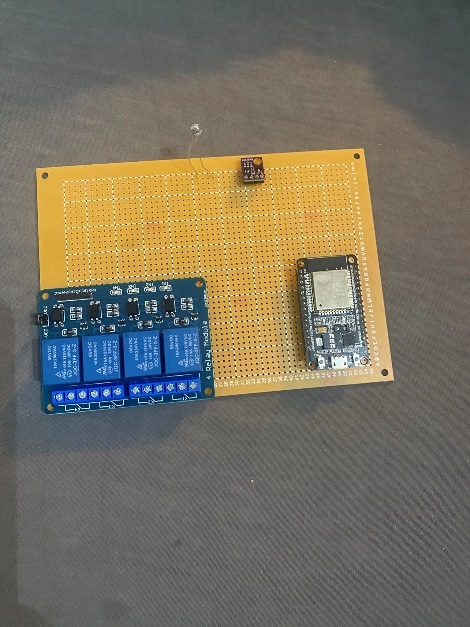
    main()

## Implementatiedetails

### Opzet hardware:

Al de code die ik tot nu toe heb uitgelegd is natuurlijk niet het enige dat moest worden gemaakt. Ook is het belangrijk dat al je bedrading vast zit en dat je een mooie stevige constructie hebt waar je kweekbakje in past.

Zoals reeds vermeld heb ik een houten doos. Deze heb ik gekozen omdat het makkelijk is om gaten er in te boren en met vijzen de componenten vast te zetten. Hier is een uitleg over hoe mijn hardware in elkaar zit.

Ik heb een vrij grote houten doos, deze was eigenlijk eerst een doos waar 2 wijnflessen in stonden, nu dient het voor mij als greenhouse. Je kan het deurtje er langs boven uitschuiven om zo in de binnenkant te geraken. Het eerste wat ik heb gedaan is de lay-out van het prototyping board. Het is handig om deze zo te plaatsen zodat je geen draden naar de andere kant van het plaatje hoeft te trekken. Aan de onderkant van het plaatje worden lijnen getrokken naar de juiste componenten. De voeding zal worden verdeeld in 3 delen. 2 draden gaan naar de relay, en 1 draad naar de esp32. Indien alle componenten op het prototyping board vastgemaakt zijn. Is het tijd voor het vastmaken van de componenten die aan de buitenkant hangen.

Voor we alle draden kunnen verbinden met de componenten moeten we ervoor zorgen dat er gaten worden geboord waardoor we de draden kunnen trekken.

Aan de achterkant wordt er rechtsboven een gaatje geboord waar de voeding doorloopt. Ook de draden van de waterlevelsensor, waterpomp en de transparante waterleiding worden hierdoor getrokken. De sensor zal verbonden worden met de esp32 en de waterpomp met de relay module.

De linkerkant van de houten doos bevat een watertankje. Deze is vastgemaakt met vijzen. Hierin zal de waterlevel sensor hangen samen met de waterpomp. De transparante leiding gaat van de pomp naar binnen.

Aan de rechterkant wordt er een gat even groot als de ventilator gemaakt. Zo kan de ventilator met 4 vijzen vastgemaakt worden en de lucht naar binnen blazen. De draden kunnen we langs de binnenkant verbinden met de relay module.

### Testing:

Doorheen de code zie je verschillende keren de command serial.print(“…”). Dit is voor mij de meest betrouwbare manier om mijn code te testen. Zo ben ik ook te werk gegaan met dit project. Na het ontvangen van mijn componenten ben ik ze gaan testen. Dit deed ik door op het internet op te zoeken hoe de sensors werkten en hoe ik ze moest aansluiten. Ik heb voor elke sensor een aparte sketch gemaakt en ervoor gezorgd dat de waardes die gemeten worden tevoorschijn komen in de seriële monitor.

Wanneer al mijn sensoren apart zijn getest, heb ik ze in 1 groot bestand gezet. Wanneer alle waardes die ik nodig had nog steeds tevoorschijn kwamen in de seriële monitor, kon ik ze laten afprinten op het lcd scherm.

## Grafana:

Bij de uitleg van de mqtt-verbinding heb ik uitgelegd hoe de waardes ontvangen, verwerkt en terug doorgestuurd werden door de mqtt-broker (raspberry pi).

Hier zijn de panelen waarin ik de waardes van mijn sensor weergeef. Let op dat dit nog niet mooi in detail is gemaakt. Ook is er bij de bodemvocht een 0 te zien. Dit komt omdat de sensor eventjes defect is en geen waarde geeft.

Afbeelding met schermopname

Automatisch gegenereerde beschrijvingAfbeelding met schermopname, tekst

Automatisch gegenereerde beschrijving

Afbeelding met schermopname

Automatisch gegenereerde beschrijving

Afbeelding met schermopname, tekst, Perceel

Automatisch gegenereerde beschrijving

## PCB Design:

Afbeelding met tekst, diagram, Plan, Technische tekening

Automatisch gegenereerde beschrijvingHet PCB Design ziet er zeer ingewikkeld uit maar is uiteindelijk vrij eenvoudig. Hier worden alle sensoren verbonden met de juiste nummer van de esp32. Elke sensor gaat naar de 3.3volt, ground en naar het signaal.

## Innovatie en aanpassingsvermogen:

### Modulair ontwerp + RFID

Aangezien het project een vrij modulair ontwerp is, is het zeer eenvoudig om verschillende sensoren er aan toe te voegen. De esp32 is een microcontroller dat voor dit project meer als genoeg poorten heeft om extra sensoren of actuatoren te ondersteunen. Je kan de esp32 ook uit het houdertje halen en er een andere code op uploaden.

Dat is niet het enigste, momenteel staat er een kweekbakje met een muntplantje in

Maar doormiddel van de rfid integratie is het mogelijk om het systeem aan te passen naar een ander plantje. Een basilicum plantje moet bijvoorbeeld altijd warm staan, indien we de rfid-tag van dat soort plantje instellen, zullen de waardes automatisch aan worden gepast naar de juist ingestelde waarde. De ventilator zal dus veel minder vaak draaien dan bij een andere plant.

### Automatisering:

De bedoeling van dit project is om alles automatisch te laten verlopen. Dit heb ik (in mijn ogen) ook gehaald. Wanneer de in het stopcontact word gestoken zal het systeem direct beginnen werken. Er zal automatisch water worden gedispenseerd, het licht zal automatisch aangaan. De ventilator zal automatisch beginnen met draaien indien nodig en de waardes van de sensoren zullen worden vertoond op het lcd scherm en worden doorgestuurd naar Grafana. Het enigste wat je zelf in het oog moet houden is de watertank. Wanneer het rode lampje brand, zal je water moeten toevoegen in het bakje aan de zijkant van de doos. Zeer eenvoudig dus!

## Projectevaluatie

Dit project is natuurlijk op een zeer kleine schaal gemaakt. Het kan ten slotte maar 1 plantje onderhouden. Maar als we dit even als grootschalig project voorstellen in een teeltbedrijf, boerderij, etc. dan kunnen we vaststellen dat dit wel een grote impact kan zijn op de landbouwpraktijken. De boeren die in deze technologie investeren zullen alles digitaal kunnen monitoren en controleren. Ze hoeven enkel te binnen schieten wanneer de gewassen volgroeit zijn of wanneer er iets fout is gelopen. Dit is dus een veel effectievere manier van teelten.

Een echte aanrader dus voor alle boeren en teeltbedrijven. In de toekomst kan het natuurlijk nog steeds verbeteren. Hier zijn enkele persoonlijke ideeën waarmee je het systeem nog kan verbeteren:

* **Integratie van weerapp**

Er zou een weer app kunnen worden geïntegreerd in het systeem die ervoor zorgt dat er mogelijke maatregelen kunnen worden genomen naargelang de weersvoorspellingen van de weer app. Stel dat het gaat stormen op de plaats waar je teelt groeit, dan krijgt de gebruiker hiervoor een melding (dagen op voorhand), zodat die eventuele maatregelen kan nemen die een sensor / actuator niet kan om zijn teelt te beschermen.

* **AI functie**

AI kan altijd wel helpen bij eender welk project. Stel je voor dat je net wil beginnen met het planten van een bepaald gewas, maar je weet niet goed welke soort kruid, groente, … het best kan groeien in je grond. Dan kan men bepaalde kenmerken beschrijven aan ai over je grond, deze zal dan een uitleg / voorspelling geven wat de beste teelt is.

Ook is het een mogelijkheid om AI de planten te laten scannen zodat hij eventueel wat info / opmerkingen kan geven over de plant die aan het groeien is.

* **Extra sensoren / actuatoren**

Dit kan altijd handig zijn. Het is ook handig om een verwarmingselement er aan toe te voegen die ervoor zorgt dat de ruimte warmer wordt indien het te koud is. Of eventueel een soort van gordijn, dat automatisch naar beneden rolt om de hoeveelheid licht te verminderen. Verder zullen er nog talrijke sensoren en actuatoren bestaan die je kan helpen.

Net zoals de waterpomp kunnen we het systeem voorzien van automatische bemesting, of andere middelen die de plant nodig heeft om te groeien.

* **Robots**

Als we echt naar de toekomst kijken, kunnen we robots toevoegen aan het systeem. Deze robots worden geactiveerd wanneer een teelt klaar is en kunnen de oogsten automatisch ophalen. Ze zouden ook, indien goed ontwikkeld, alle voedingstoffen automatisch voorzien (van water, meststoffen, onkruidbestrijding, …).

* **Energiebeheer**

Om er voor te zorgen dat dit systeem ook goed blijft voor het milieu, kunnen we het aansluiten aan zonnepanelen. Zonne-energie is namelijk veel milieuvriendelijker.

## Volledige code:

### Code esp32:

#include <Wire.h>

#include <LiquidCrystal\_I2C.h>

#include <Adafruit\_NeoPixel.h>

#include <Adafruit\_BME280.h>

#include <Adafruit\_Sensor.h>

#include <OneWire.h>

#include <DallasTemperature.h>

#include <WiFi.h>

#include <PubSubClient.h>

const char\* ssid = "telenet-34090"; //embed    //Verander

const char\* password = "BfBv60WUspH3";  // weareincontrol //verander

const char\* mqttServer = "192.168.0.235";  //verander

const int mqttPort = 1883;

const char\* mqttUser = "robinpi";

const char\* mqttPassword = "robindevoecht";

const char\* clientID = "client\_livingroom";  // MQTT client ID

const char\* moisture\_topic = "home/livingroom/moisture";

const char\* groundTemp\_topic = "home/livingroom/bodemTemp";

const char\* Temp\_topic = "home/livingroom/Temp";

const char\* Light\_topic = "home/livingroom/Light";

const char\* waterLevel\_topic = "home/livingroom/waterLevel";

WiFiClient espClient;

PubSubClient client(espClient);

// ds18b20

const int oneWireBus = 5;

OneWire oneWire(oneWireBus);

DallasTemperature sensors(&oneWire);

//BME280

#define BME\_SCK 12

#define BME\_MISO 19

#define BME\_MOSI 23

#define BME\_CS 5

Adafruit\_BME280 bme;

// Led Strip

#define pin 18

#define NUM\_leds 30

Adafruit\_NeoPixel strip = Adafruit\_NeoPixel(NUM\_leds, pin, NEO\_GRB + NEO\_KHZ800);

//LCD

#define LCD\_ADDRESS 0x27

#define LCD\_COLUMNS 16

#define LCD\_ROWS 2

LiquidCrystal\_I2C lcd(LCD\_ADDRESS, LCD\_COLUMNS, LCD\_ROWS);

//WaterLevelSensor

const int sensorPin = 27;

const int waterLevelThreshold = 800;

const int lowLevelThreshold = 500;

const int ledPin = 14;

//LDR

const int ldrPin = 34;

// Bodemvochtsensor

const int Bodemvocht = 35;

int bodemValue = 0;

//relay

const int VentilatorPin = 25;

const int WaterPompPin = 26;

const int BodemvochtTreshold = 1400;

const int TemperatuurTreshold = 21;

void setup\_wifi() {

  delay(10);

  Serial.println();

  Serial.print("Connecting to ");

  Serial.println(ssid);

  WiFi.begin(ssid, password);

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

    delay(500);

    Serial.print(".");

  }

  Serial.println("");

  Serial.println("WiFi connected");

  Serial.println("IP address: ");

  Serial.println(WiFi.localIP());

}

void reconnect() {

  while (!client.connected()) {

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

    if (client.connect(clientID, mqttUser, mqttPassword)) {

      Serial.println("connected");

    } else {

      Serial.print("failed, rc=");

      Serial.print(client.state());

      delay(2000);

    }

  }

}

void setup() {

  Serial.begin(115200);

  setup\_wifi();

  client.setServer(mqttServer, mqttPort);

  Serial.println("Initializing BME280...");

  Wire.begin();

  bool status;

  status = bme.begin(0x76);

  if (!status) {

    Serial.println("could not find BME280");

    while (1)

      ;

  }

  Serial.println("Default test");

  Serial.println();

  //Initialisatie

  pinMode(sensorPin, INPUT);

  pinMode(ledPin, OUTPUT);

  pinMode(ldrPin, INPUT);

  //LCD initialiseren

  lcd.clear();

  lcd.init();

  lcd.backlight();

  //LED initialisatie

  strip.begin();

  strip.show();

  //ds18b20

  sensors.begin();

  // Relay

  pinMode(VentilatorPin, OUTPUT);

  pinMode(WaterPompPin, OUTPUT);

  digitalWrite(VentilatorPin, LOW);

  digitalWrite(WaterPompPin, LOW);

}

void loop() {

  if (!client.connected()) {

    reconnect();

  }

  //Call all functions

  client.loop();

  delay(1000);

  ReadWaterLevel();

  ReadLDR();

  delay(2000);

  ReadBME280();

  delay(2000);

  ReadDS18B20();

  delay(2000);

  ReadBodemVocht();

  delay(2000);

}

void ReadWaterLevel() {

  int sensorValue = analogRead(sensorPin);  //

  if (sensorValue >= waterLevelThreshold) {

    Serial.println("Water detected!");  //

    //stuur waarde door naar database

    client.publish(waterLevel\_topic, String(waterLevelThreshold).c\_str());

    // Water wordt getedecteer en ledlampje zal uitgaan

    digitalWrite(ledPin, LOW);

  } else if (sensorValue < lowLevelThreshold) {

    Serial.println("Warning: Low water level detected!");

    // LEDlamp aan wanneer water level laag is

    digitalWrite(ledPin, HIGH);

    //stuur waarde door naar database

    client.publish(waterLevel\_topic, String(lowLevelThreshold).c\_str());

  }

  delay(1000);

}

void ReadLDR() {

  // Waarde wordt opgemeten en geprint op lcd scherm.

  int ldrValue = analogRead(ldrPin);

  lcd.clear();

  Serial.print("Lichtintensiteit: ");

  Serial.println(ldrValue);

  lcd.setCursor(0, 0);

  lcd.print("Licht: ");

  lcd.print(ldrValue);

  //Waarde wordt doorgestuurd naar database

  client.publish(Light\_topic, String(ldrValue).c\_str());

//Wanneer er te weinig licht is, zal de ledstrip aan springen

  if (ldrValue < 1500) {

    for (int i = 0; i < strip.numPixels(); i++) {

      strip.setPixelColor(i, strip.Color(246, 205, 139));  // stel een bepaalde kleur in.

    }

    strip.show();

  } else {

    for (int i = 0; i < strip.numPixels(); i++) {

      strip.setPixelColor(i, strip.Color(0, 0, 0));  // Zet alle LED's uit

    }

    strip.show();

  }

  delay(1000);

}

void ReadBME280() {

  //lees de waarde van de bme280, print deze op het lcd scherm.

  Serial.print("Temp: ");

  Serial.println(bme.readTemperature());

  lcd.clear();

  lcd.setCursor(0, 0);

  lcd.print("Temp: ");

  lcd.print(bme.readTemperature());

  lcd.print(" \*C");

  lcd.setCursor(0, 1);

  lcd.print("Hum: ");

  lcd.print(bme.readHumidity());

  // Stuur waarde door naar database

  client.publish(Temp\_topic, String(bme.readTemperature()).c\_str());

// Indien de opgemeten temperatuur te hoog staat, zal de ventilator aan springen. Dit gebeurd via de relay

  if (bme.readTemperature() < TemperatuurTreshold) {

    digitalWrite(VentilatorPin, HIGH);

  } else {

    digitalWrite(VentilatorPin, LOW);

  }

  delay(1000);

}

void ReadDS18B20() {

  // Print de opgemeten waarde naar het lcd scherm

  sensors.requestTemperatures();

  float tempC = sensors.getTempCByIndex(0);

  if (tempC != -127.00) {

    Serial.print("Grond Temperatuur: ");

    Serial.print(tempC);

    Serial.print(" \*C");

    lcd.clear();

    lcd.setCursor(0, 0);

    lcd.print("Temp Grond:");

    lcd.setCursor(0, 1);

    lcd.print(tempC);

    lcd.print(" \*C");

    // Waarde wordt doorgestuurd naar grafana.

    client.publish(groundTemp\_topic, String(tempC).c\_str());

  } else {

    Serial.print("Fout bij lezen van temp");

  }

  delay(1000);

}

void ReadBodemVocht() {

  // print de opgemeten waarde op het lcd scherm

  bodemValue = analogRead(Bodemvocht);

  Serial.print("Soil Moisture Value: ");

  Serial.println(bodemValue);

  lcd.clear();

  lcd.setCursor(0, 0);

  lcd.print("Bodemvocht: ");

  lcd.setCursor(0, 1);

  lcd.print(bodemValue);

  // Waarde wordt doorgestuurd naar database

  client.publish(moisture\_topic, String(bodemValue).c\_str());

// Indien de gemeten waarde van de bodemvochtsensor lager is dan de meegegeven treshold (1400).

// dan zal de relay pin, waar de waterpomp aan vast hangt, voor 2 seconden aan gaan.

  if (bodemValue > BodemvochtTreshold) {

    digitalWrite(WaterPompPin, LOW);

    delay(2000);

    digitalWrite(WaterPompPin, HIGH);

  } else {

    digitalWrite(WaterPompPin, HIGH);

  }

  // Wait 1 second before taking another reading

  delay(1000);

}

### Code raspberry pi (mqtt-broker):

INFLUXDB\_ADDRESS = 'localhost'

INFLUXDB\_USER = '\*\*\*\*\*'

INFLUXDB\_PASSWORD = '\*\*\*\*\*\*\*'

INFLUXDB\_DATABASE = 'weather\_stations'

MQTT\_ADDRESS = '192.168.1.108'

MQTT\_USER = '\*\*\*\*\*'

MQTT\_PASSWORD = '\*\*\*\*\*'

MQTT\_TOPIC = 'home/+/+'

MQTT\_REGEX = 'home/([^/]+)/([^/]+)'

MQTT\_CLIENT\_ID = 'MQTTInfluxDBBridge'

influxdb\_client = InfluxDBClient(INFLUXDB\_ADDRESS, 8086, INFLUXDB\_USER,

INFLUXDB\_PASSWORD, None)

class SensorData(NamedTuple):

    location: str

    measurement: str

    value: float

def on\_connect(client, userdata, flags, rc):

    """ The callback for when the client receives a CONNACK response from the server."""

    print('Connected with result code ' + str(rc))

    client.subscribe(MQTT\_TOPIC)

def \_parse\_mqtt\_message(topic, payload):

    match = re.match(MQTT\_REGEX, topic)

    if

match:

        location = match.group(1)

        measurement = match.group(2)

        if measurement == 'status':

            return None

        return SensorData(location, measurement, float(payload))

    else:

        return None

def \_send\_sensor\_data\_to\_influxdb(sensor\_data):

    json\_body = [

        {

            'measurement': sensor\_data.measurement,

            'tags': {

                'location': sensor\_data.location

            },

            'fields': {

                'value': sensor\_data.value

            }

        }

    ]

    influxdb\_client.write\_points(json\_body)

def on\_message(client, userdata, msg):

    """The callback for when a PUBLISH message is received from the server."""

    print(msg.topic + ' ' + str(msg.payload))

    sensor\_data = \_parse\_mqtt\_message(msg.topic, msg.payload.decode('utf-8'))

    if sensor\_data is not None:

        \_send\_sensor\_data\_to\_influxdb(sensor\_data)

def \_init\_influxdb\_database():

    databases = influxdb\_client.get\_list\_database()

    if len(list(filter(lambda x: x['name'] == INFLUXDB\_DATABASE, databases))) == 0:

        influxdb\_client.create\_database(INFLUXDB\_DATABASE)

    influxdb\_client.switch\_database(INFLUXDB\_DATABASE)

def main():

    \_init\_influxdb\_database()

    mqtt\_client = mqtt.Client(MQTT\_CLIENT\_ID)

    mqtt\_client.username\_pw\_set(MQTT\_USER, MQTT\_PASSWORD)

    mqtt\_client.on\_connect = on\_connect

    mqtt\_client.on\_message = on\_message

    mqtt\_client.connect(MQTT\_ADDRESS, 1883)

    mqtt\_client.loop\_forever()

if \_\_name\_\_ == '\_\_main\_\_':

    print('MQTT to InfluxDB bridge')

    main()

## Bibliografie:

* Randomnerdtutorials.com
* Aliexpress / amazon (bestellen van componenten)
* Github
* Adafruit.com
* Arduino.com
* Instructables.com
* https://www.youtube.com/watch?v=AyjRRPSmyZg&ab\_channel=Dial