THOMAS MORE

HAICO LUYCKX

Eco home assistant

2024 - 2025

Inhoud

1. PROJECTBESCHRIJVING 3

1.1. Welk probleem lost deze automatisering op? 3

1.2. Waarom is deze oplossing nuttig? 3

1.3. Schema ESP32 3

2. HARDWARE 4

2.1. Raspberry pi 5 4

2.2. ESP32 mini D1 DEV 5

2.3. SSD1315 - 0.96inch 5

2.4. BH1750 licht sensor 5

2.5. KY-016 RGB LED module 6

2.6. HC-SR04 ulrasoon sensor 6

2.7. DHT11 6

2.8. 1x witte LED (met 220Ω resistor) 7

3. INLEIDING VAN HET PROJECT 7

3.1. Gebruikte technologieën 7

3.2. Waarom deze technologieën 7

4. METHODOLOGIE 8

4.1. Installatie en configuratie van MQTT 8

4.1.1. Installeren van Mosquitto op Raspberry Pi 8

4.1.2. Configuratie van de broker-instellingen 8

4.1.3. Implementatie van wachtwoordbeveiliging 8

4.1.4. Testen van de broker-functionaliteit 9

4.1.4.1. Subscribe 9

4.1.4.2. Publish 9

4.2. Mijn eigen influxdb service 10

4.3. Test- en debuggingmethodes 13

5. TECHNISCHE DETAILS 13

5.1. ESP32-code met uitleg 13

5.2. Blynk app 21

5.3. Grafana dashboard configuratie 23

6. VERIFICATIEMETHODEN 24

6.1. Hoe heb je gecontroleerd dat MQTT-berichten correct worden verzonden? 24

6.2. Hoe heb je geverifieerd dat apparaten correct reageren op berichten? 24

6.3. Welke testscenario's heb je doorlopen? 24

7. VIDEO 24

1. Projectbeschrijving

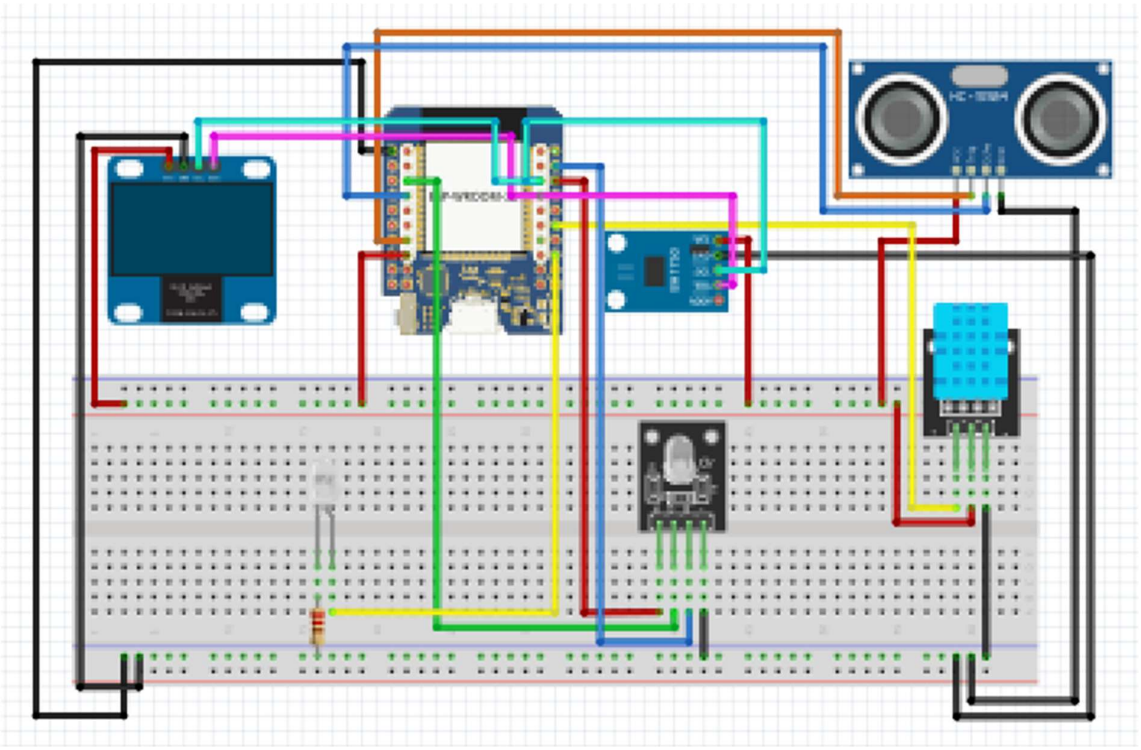
1.1. Welk probleem lost deze automatisering op?

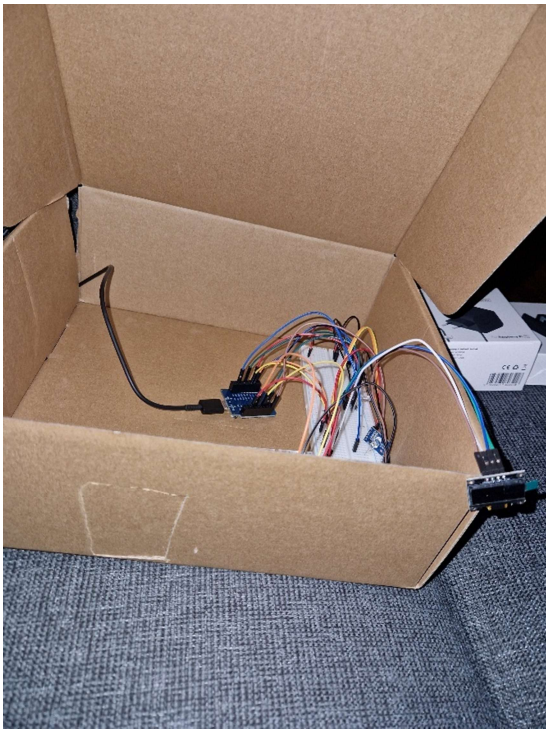
Het project is een simulatie van een ecologisch systeem voor een smart home assistant. Zo wil je bijvoorbeeld niet dat je verwarming aan staat als je de voordeur laat openstaan, of je wil de temperatuur van je verwarming aanpassen in een bepaalde kamer van het huis misschien zelfs het licht aan of uit laten gaan in een kamer als je dit zou vergeten zijn.

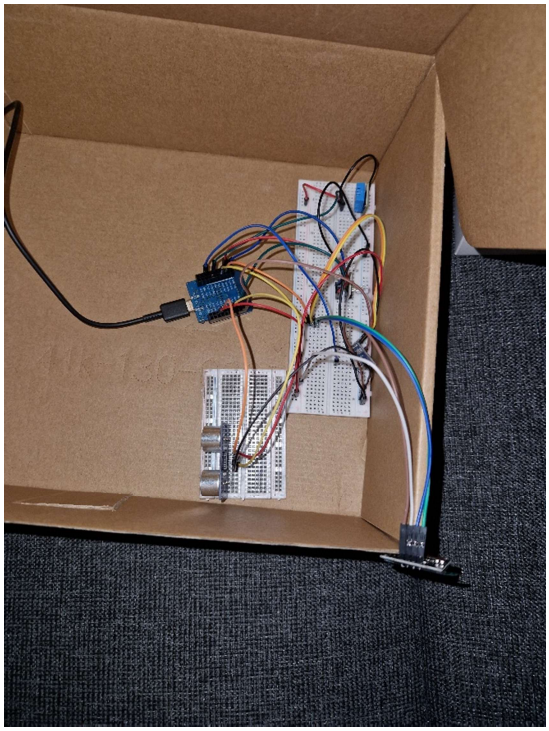
1.2. Waarom is deze oplossing nuttig?

Ik heb hier vooral voor gekozen omdat je zo makkelijker dingen van op afstand kan aansturen en je je niet meteen hoeft te verplaatsen om dit te doen. Ook als de deur meer dan een bepaalde tijd open staat gaat de verwarming uit (in de demo is dit 15s). Dit hebt bij het besparen van energie en warmte die anders verloren zou gaan.

1.3. Schema ESP32

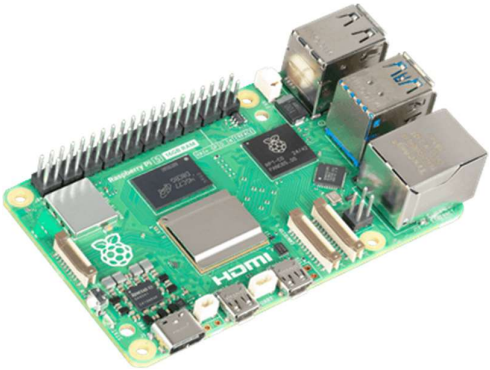




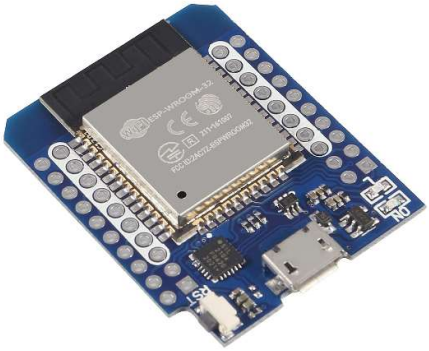


2. Hardware

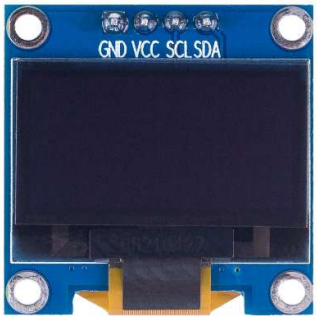
2.1. Raspberry pi 5



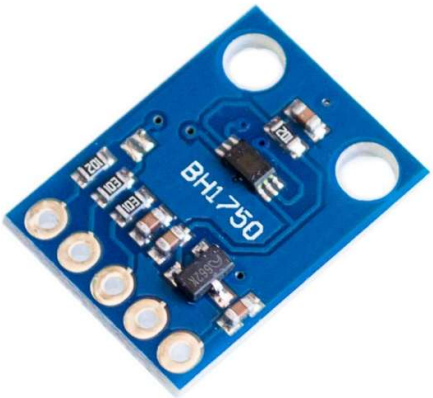
2.2. ESP32 mini D1 DEV



2.3. SSD1315 - 0.96inch

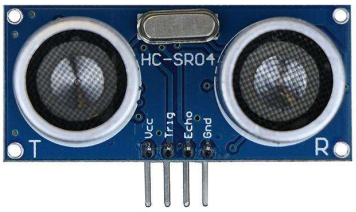


2.4. BH1750 licht sensor

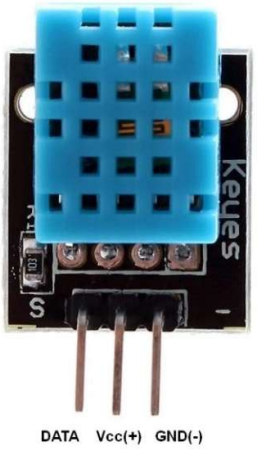


2.5. KY-016 RGB LED module

2.6. HC-SR04 ulrasoon sensor



2.7. DHT11



2.8. 1x witte LED (met 220Ω resistor)

3. Inleiding van het project

3.1. Gebruikte technologieën

Voor mijn project heb ik gebruik gemaakt van Moquitto (MQTT), InfluxDB, Grafana en Blynk.

Met MQTT kan je data versturen en ontvangen van en naar andere apparaten om deze hier te kunnen gebruiken en uit te lezen. Zo kan je 2 apparaten op verschillende plaatsten hebben en toch data kunnen delen tussen deze apparaten.

Influxdb wordt gebruikt om de gegevens op te slaan in een database die je doorkrijgt via MQTT. Deze data kan dan altijd bekeken worden van voorbije zendingen van informatie.

Met grafana kan je de data uit de database halen van influxdb en deze tonen op een manier naar jou keuze. Hier kan je een eigen dashboard maken en aanpassen naar jouw voorkeur om dit duidelijk weer te geven.

Via Blynk kan je alle info remote aanpassen via de app op je smart device. Zo kan je ook data die gemeten is uitlezen zonder in de buurt te zijn van je installatie en hier aanpassingen aan maken.

3.2. Waarom deze technologieën

We hebben voor deze technologieën gekozen omdat met combinatie van alle vier je een duidelijke representatie kan hebben van je uitgelezen data en of berichten. Verder geeft het zoals hiervoor gemeld je de mogelijkheid om data tussen apparaten uit te wisselen en hoeven ze niet rechtstreeks verbonden te zijn. Ook heb je hiermee de mogelijkheid om van op je smart device als een smartphone de waardes te veranderen en de data te bekijken.

4. Methodologie

4.1. Installatie en configuratie van MQTT

4.1.1. Installeren van Mosquitto op Raspberry Pi

Om te beginnen zorgen we ervoor dat onze Raspberry pi up to date is met:

```
sudo apt-get update
sudo apt-get upgrade
```

Vervolgens installeren we de mosquitto clients met het commando:

```
sudo apt-get install mosquitto mosquitto-clients
```

4.1.2. Configuratie van de broker-instellingen

Voor de configuratie passen we de file *mosquitto.conf* aan om ervoor te zorgen dat we de juiste poort gebruiken om mosquitto op te draaien en dat het de juiste files gebruikt voor onze credentials. Dit doen we door het volgende commando in te voeren:

```
sudo nano /etc/mosquitto/mosquitto.conf
```

In dit document checken we of de lijnen overeenkomen met de volgende tekst:

```
# Place your local configuration in /etc/mosquitto/conf.d/
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

pid_file /run/mosquitto/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

log_dest file /var/log/mosquitto/mosquitto.log

# include_dir /etc/mosquitto/conf.d

allow_anonymous false
password_file /etc/mosquitto/pwfile
bind_address 0.0.0.0
port 1883
```

4.1.3. Implementatie van wachtwoordbeveiliging

Voor het gebruik van mosquitto maken we een gebruiker aan met een wachtwoord om onze informatie die verzonden en ontvangen wordt geheim te houden. Dit doen we met volgend commando:

```
sudo mosquitto_passwd –c /etc/mosquitto/pwfile username
```

Hieronder nog eens de betekenis voor elk deel van het commando:

• Sudo : root user
• Mosquitto_passwd : make password
• -c : create new password file (more info on manual page (man mosquitto_passwd)
• /etc/... : location for the password file
• Username : free to choose name for the user

Vervolgens kan je ook het volgend commando gebruiken om mosquitto steeds automatisch te starten bij opstart van je raspberry pi:

sudo systemctl enable mosquitto

4.1.4. Testen van de broker-functionaliteit

Om te kijken of onze broker correct werkt kunnen we gebruik maken van een dummy message. Hiermee gaan we kunnen zien of het bericht correct doorkomt. Dit doen we door 2 SSH sessies op te zetten of 2 command windows open te doen als je op de GUI van je rapsberry werkt. Op de eerste gaan we subscriben en op de tweede gaan we een message publishen.

4.1.4.1. Subscribe

Hieronder het commando voor het subscriben om de verzonden info te kunnen ontvangen en de uitleg erover:

mosquitto_sub –v –t test/message –u username –P password

- **mosquitto_sub**

Dit is de Mosquitto MQTT-subscribe client. Dit betekent dat deze tool zich abonneert op een MQTT-topic en berichten ontvangt die daarop gepubliceerd worden.

- **-v** (verbose output)

Dit zorgt ervoor dat zowel het topic als de ontvangen berichten worden weergegeven in de console.

- **-t test/message** (topic selectie)

Dit geeft aan op welk topic je je abonneert. In dit geval luister je naar berichten die op het topic test/message worden gepubliceerd.

- **-u username** (gebruikersnaam)

Dit is de gebruikersnaam die je gebruikt voor authenticatie bij de MQTT-broker.

- **-P password** (wachtwoord)

Dit is het wachtwoord voor authenticatie bij de MQTT-broker.

4.1.4.2. Publish

Hieronder het commando voor het versturen van een bericht voor de test:

mosquitto_pub –h localhost –t test/message –m ‘hello world’ –u username –P password

- **mosquitto_pub**

Dit is de Mosquitto MQTT-publish client, waarmee je berichten naar een MQTT-topic kunt verzenden.

- **-h localhost** (hostnaam van de broker)

Dit geeft aan dat je verbindt met een MQTT-broker die draait op de **lokale machine** (localhost betekent hetzelfde als 127.0.0.1).

- **-t test/message** (topic selectie)

Het bericht wordt gepubliceerd op het **topic** test/message. Elk apparaat of client die zich hierop heeft geabonneerd (bijvoorbeeld met mosquitto_sub) zal het bericht ontvangen.

- **-m 'hello world'** (bericht inhoud)

Dit is het **bericht** dat wordt verstuurd. In dit geval wordt de tekst 'hello world' naar het topic test/message gestuurd.

- **-u username -P password** (authenticatie)

Dit zorgt ervoor dat je je met een gebruikersnaam en wachtwoord aanmeldt bij de MQTT-broker, als die authenticatie vereist.

4.2. Mijn eigen influxdb service

Om met de MQTT-server en de database te communiceren hebben we enkele libraries nodig die we eerste moeten installeren. Deze moeten we installeren

```
source ~/embedded/bin/activate
pip3 install paho-mqtt
sudo apt-install influxdb influxdb-client
```

Vervolgens maken we onze file aan en schrijven we hier de code om binnenkomende MQTT-berichten te verwerken en deze weg te schrijven naar onze influxdb database.

```
import sys
from typing import NamedTuple
import paho.mqtt.client as mqtt
from influxdb import InfluxDBClient

# InfluxDB-configuratie
INFLUXDB_ADDRESS = '192.168.129.71'
INFLUXDB_USER = 'haico'
INFLUXDB_PASSWORD =
INFLUXDB_DATABASE = 'eco_home_assistant'

# MQTT-configuratie
MQTT_ADDRESS = '192.168.129.71'
MQTT_USER = 'haico'
MQTT_PASSWORD =
MQTT_TOPICS = {
    "min_temp": "home/temp_min",
    "max_temp": "home/temp_max",
    "current_temp": "home/temp",
    "humidity": "home/humidity"
}
MQTT_CLIENT_ID = 'influxdb_bridge'
```

```python
# Verbinding met InfluxDB
influxdb_client = InfluxDBClient(INFLUXDB_ADDRESS, 8086, INFLUXDB_USER,
INFLUXDB_PASSWORD, None)

class SensorData(NamedTuple):
    measurement: str
    value: float

def init_influxdb():
    """Maakt de InfluxDB database aan als die nog niet bestaat."""
    databases = influxdb_client.get_list_database()
    if not any(db['name'] == INFLUXDB_DATABASE for db in databases):
        print(f"Aanmaken van nieuwe database: {INFLUXDB_DATABASE}")
        influxdb_client.create_database(INFLUXDB_DATABASE)
    influxdb_client.switch_database(INFLUXDB_DATABASE)

def on_connect(client, userdata, flags, rc, properties=None):
    """Wordt aangeroepen zodra de MQTT-client connectie maakt."""
    print(f"Verbonden met MQTT-broker (code {rc})")
    for topic in MQTT_TOPICS.values():
        client.subscribe(topic)
        print(f"Geabonneerd op {topic}")

def send_to_influxdb(sensor_data):
    """Stuurt sensor data naar InfluxDB."""
    json_body = [{
        "measurement": sensor_data.measurement,
        "fields": {"value": sensor_data.value}
    }]
    influxdb_client.write_points(json_body)
    print(f"Gegevens opgeslagen in InfluxDB: {sensor_data.measurement} =
{sensor_data.value}")

def on_message(client, userdata, msg):
    """Wordt aangeroepen wanneer een MQTT-bericht binnenkomt."""
    topic = msg.topic
    payload = msg.payload.decode("utf-8")

    try:
        value = float(payload)
    except ValueError:
        print(f"Ongeldige payload ontvangen op {topic}: {payload}")
        return

    measurement = [k for k, v in MQTT_TOPICS.items() if v == topic][0]
    sensor_data = SensorData(measurement, value)
    send_to_influxdb(sensor_data)
```

```
def main():
    init_influxdb()

    mqtt_client = mqtt.Client(client_id=MQTT_CLIENT_ID)
    mqtt_client.username_pw_set(MQTT_USER, MQTT_PASSWORD)
    mqtt_client.on_connect = on_connect
    mqtt_client.on_message = on_message

    try:
        mqtt_client.connect(MQTT_ADDRESS, 1883, 60)
    except Exception as e:
        print(f"Fout bij verbinden met MQTT-broker: {e}")
        sys.exit(1)

    mqtt_client.loop_forever()

if __name__ == "__main__":
    print("Starten van MQTT naar InfluxDB bridge...")
    main()
```

Deze code zorgt ervoor dat we de data die wordt verstuurd met MQTT kan worden weggeschreven in de InfluxDB database. Deze kan dan worden gebruikt om op grafana de data te tonen op een dashboard met een duidelijke layout.

Vervolgens maken we een service file aan met volgende command:

sudo nano /lib/systemd/system/influxdb_eco_home_script.service

Hierin voegen we volgende code in toe om de file die we hiervoor hebben gemaakt te kunnen runnen als een service:

[Unit]
Description=Bridge to save data coming in through MQTT into InfluxDB
After=multi-user.target

[Service]
Type=idle
ExecStart=/home/haico/embedded/bin/python3 /home/haico/python/influxdb_eco_home.py

[Install]
WantedBy=multi-user.target

Opslaan en afsluiten met [CTRL]+X

Vervolgens zetten we de rechten goed van deze service file

sudo chmod 644 /lib/systemd/system/influxdb_eco_home_script.service

Nu gaan we ervoor zorgen dat de service die we net hebben gemaakt telkens automatisch opstart wanneer onze raspberry pi 5 opstart

```
sudo systemctl daemon-reload
sudo systemctl enable influxdb_eco_home_script
sudo systemctl start influxdb_eco_home_script
```

4.3. Test- en debuggingmethodes

Voor testcode heb ik standaard code gebruikt om eerst data uit te lezen en mijn led aan te sturen en hierna op verdergegaan.

Voor debugging heb ik code ingevoerd dat enkel data wordt doorgestuurd als er een verandering in komt om tegen te gaan dat er een enorme hoeveelheid aan data wordt verstuurd met MQTT. Vervolgens heb ik ook Serial prints toegevoegd om binnenkomende en uitgelezen data na te kijken.

5. Technische details

5.1. ESP32-code met uitleg

```
#define BLYNK_TEMPLATE_ID "user9"
#define BLYNK_TEMPLATE_NAME "user9@server.wyns.it"

#include <WiFi.h>
#include <PubSubClient.h>
#include <DHT.h>
#include <Wire.h>
#include <BH1750.h>
#include <Adafruit_SSD1306.h>
#include <BlynkSimpleEsp32.h>

// WiFi, MQTT en Blynk settings
const char* ssid = "Proximus-Home-089155";
const char* password = "";
const char* mqtt_server = "192.168.129.71";
const char* mqtt_user = "haico";
const char* mqtt_password = "";
const char* client_id = "esp32_home_assistant";
char auth[] = "8Wle5tD5B__VxYCnc7kVadJXSRUZWZEm";
```

We beginnen met het ingeven van de nodige info om te kunnen connecteren met de Blynk server, vervolgens voegen we ook de libraries toe die we willen gebruiken in onze code en tot slot de wifi en MQTT instellen die we nodig hebben om de connecties op te stellen en onze key om te verbinden met de Blynk pagina die we hebben gemaakt voor deze applicatie.

```
WiFiClient espClient;
PubSubClient client(espClient);

// DHT11 sensor
#define DHTPIN 4
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

// Witte LED
#define LED_PIN 2

// RGB LED
#define LED_RED 25
#define LED_GREEN 26
#define LED_BLUE 27

// HC-SR04 ultrasonic sensor
#define TRIG_PIN 5
#define ECHO_PIN 18

// OLED screen
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

// BH1750 lightsensor
BH1750 lightMeter;

// Blynk widgets
BlynkTimer timer;
WidgetLED ledLight(V0);
WidgetLED ketel_status(V2);
WidgetTerminal terminal(V10);
```

Vervolgens gaan we de pinnen toekennen voor de modules die we gebruiken om deze later te gebruiken in de setup. Ook kennen we de pinnen van onze Blynk widgets toe om ze zo later aan te kunnen sturen.

```
float min_temp = 18.0;
float max_temp = 22.0;
bool light_on = false;
float lastTemperature = -1000.0;
float lastHumidity = -1000.0;
float lastLux = 0.0;
bool lastLightOn = false;
bool doorOpen = false;
unsigned long doorOpenedTime = 0;
```

Nu hebben we variabelen nodig om hier later in de code de ingelezen en binnenkomende waardes aan toe te kennen en deze te kunnen onthouden.

```
// once blynk is connected succesfully we want to turn on the RGB led to
// indicate that our heating is on. We also want to pull in the last written
// values to sync our system and not lose data. Lastly we want to clear our
// terminal for any leftover data and print the options.
BLYNK_CONNECTED() {
  ketel_status.on();
  Blynk.syncAll();

  terminal.clear();
  terminal.println("=== HOME ASSISTANT TERMINAL ===");
  terminal.println("Type 'DOOR' or 'LIGHT' to get status.");
  terminal.println("--------------------------------");
}
```

```
// this command is listening when something is written to V1
// V1 is a button in the app that will switch the led on and off.
BLYNK_WRITE(V1) {
  int pinValue = param.asInt();
  light_on = pinValue == 1;
  client.publish("home/light", light_on ? "on" : "off", true);
  digitalWrite(LED_PIN, light_on ? HIGH : LOW);
  if (light_on) {
    ledLight.on();
  } else {
    ledLight.off();
  }
}
```

```
// once a value is written to V5 (Step H input) we want to read this in
// and update our minimum temperature accordingly
BLYNK_WRITE(V5) {
  float new_temp = param.asFloat();
  if (new_temp != min_temp) {
    min_temp = new_temp;
    Serial.printf("Nieuwe min temp: %.1f\n", min_temp);
    client.publish("home/temp_min", String(min_temp).c_str(), true);
    Blynk.virtualWrite(V4, min_temp);
  }
}
```

```
// once a value is written to V7 (Step H input) we want to read this in
// and update our maximum temperature accordingly
BLYNK_WRITE(V7) {
  float new_temp = param.asFloat();
  if (new_temp != max_temp) {
```

```
    max_temp = new_temp;
    Serial.printf("Nieuwe max temp: %.1f\n", max_temp);
    client.publish("home/temp_max", String(max_temp).c_str(), true);
    Blynk.virtualWrite(V6, max_temp);
  }
}
```

```
// for our terminal we want to give access to the status of our door
// en the status of our light, this way you can check if your door was left
// open. More info can be added later if wanted or if more sensors are
connected.
BLYNK_WRITE(V10) {
  if (String("DOOR") == param.asStr()) {
    bool doorCurrentlyOpen = isDoorOpen();
    terminal.printf("The door is currently %s\n", doorCurrentlyOpen ? "OPEN" :
"CLOSED");
  } else if (String("LIGHT") == param.asStr()) {
    terminal.printf("The light is currently %s\n", light_on ? "ON" : "OFF");
  } else {
    terminal.println("Invalid command. Try 'DOOR' or 'LIGHT'.");
  }
  terminal.flush();
  delay(4000);
  terminal.clear();

  terminal.println("=== HOME ASSISTANT TERMINAL ===");
  terminal.println("Type 'DOOR' or 'LIGHT' to get status.");
  terminal.println("--------------------------------");
}
```

```
// in our setup we connect to serial monitor, wifi, MQTT client and Blynk
server
// we also start our needed hardware and software to make them ready for use
// our pin modes also need to be set to be able to control our leds and read
our values in
// lastly we want to run the readAndSendSensorData function that sends our
humidity,
// temperature and our light levels to blynk every 5 seconds
void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("WiFi Verbonden!");

  client.setServer(mqtt_server, 1883);
```

```
  connectMQTT();

  Wire.begin();
  lightMeter.begin();
  dht.begin();

  Blynk.begin(auth, ssid, password, "server.wyns.it", 8081);

  pinMode(LED_PIN, OUTPUT);
  pinMode(LED_RED, OUTPUT);
  pinMode(LED_GREEN, OUTPUT);
  pinMode(LED_BLUE, OUTPUT);
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);

  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println("OLED niet gevonden!");
    while (true)
      ;
  }

  timer.setInterval(5000L, readAndSendSensorData);
}
```

```
// we connect to the MQTT network with our credentials and subscribe
//to the topics to read the last written values in the database
void connectMQTT() {
  while (!client.connected()) {
    Serial.print("Verbinden met MQTT...");
    if (client.connect(client_id, mqtt_user, mqtt_password)) {
      Serial.println("Verbonden!");
      client.subscribe("home/temp_min");
      client.subscribe("home/temp_max");
      client.subscribe("home/light");
    } else {
      Serial.print("Fout, rc=");
      Serial.print(client.state());
      Serial.println(" Wachten op herverbinding...");
      delay(5000);
    }
  }
}
```

```
// we check if a connection was made with the MQTT server, if not
// we try to make the connection again. Then we read out the values
// of our sensors and check if the door is open
void loop() {
  if (!client.connected()) {
    connectMQTT();
  }
  client.loop();

  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();
   float lux = lightMeter.readLightLevel();
  bool doorCurrentlyOpen = isDoorOpen();
```

```
  // checking is the door is currently open, if this
  // is the case we set the soor open value to true and
  // start the timer for 15 seconds, after 15 seconds
  // we shut of the heating to save power
  if (doorCurrentlyOpen) {
    if (!doorOpen) {
      doorOpen = true;
      doorOpenedTime = millis();
    } else if (millis() - doorOpenedTime >= 15000) {
      setColor(0, 0, 0);
      ketel_status.setColor("#000000");
    }
  } else {
    doorOpen = false;
  }
```

```
  // controlling the RGB LED based on the minimim and macimum
  // set temperatures to set the color showing the status of
  // the heating
  if (!doorCurrentlyOpen) {
    if (temperature < min_temp) {
      setColor(255, 0, 0);
      ketel_status.setColor("#ff0000");
    } else if (temperature > max_temp) {
      setColor(0, 0, 255);
      ketel_status.setColor("#0000ff");
    } else {
      setColor(0, 255, 0);
      ketel_status.setColor("#00ff00");
    }
  }
```

```
  // display on the OLED screen
  displayInformation(temperature, humidity, doorCurrentlyOpen, light_on, lux);

  // we publish the updated status of our light, but only
  // when the value has changed to reduce data trafic
  if (light_on != lastLightOn) {
    client.publish("home/light_status", light_on ? "ON" : "OFF", true);
    lastLightOn = light_on;
  }

  // Debug info
  Serial.println("----Overzicht----");
  Serial.printf("Min: %.1f Max: %.1f\n", min_temp, max_temp);
  Serial.printf("Temp: %.1fC\n", temperature);
  Serial.printf("Vocht: %.1f%\n", humidity);
  Serial.printf("Deur: %s\n", doorCurrentlyOpen ? "OPEN" : "CLOSED");
  Serial.printf("Licht: %s\n", light_on ? "ON" : "OFF");
  Serial.printf("Helderheid: %.1flx\n", lux);
  Serial.println("");

  // we set our Blynk and the Blynk timer to run
  // making it possible to read and write from our app
  Blynk.run();
  timer.run();

  delay(5000);
}
```

```
// we read the values of our sensors again to be able to use the
// values and send them to our Blynk app. We also do checks to
// make sure the read values are actual numbers before we safe them.
// Then we publish it to MQTT if the value has changes and to Blynk
void readAndSendSensorData() {
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();
  float lux = lightMeter.readLightLevel();

  // Controle op NaN voor temperatuur en luchtvochtigheid
  if (isnan(temperature)) {
    Serial.println("Fout: Temperatuurmeting mislukt!");
    temperature = lastTemperature;
  }
  if (isnan(humidity)) {
    Serial.println("Fout: Vochtigheidsmeting mislukt!");
    humidity = lastHumidity;
  }
```

```
  if (isnan(lux)) {
    Serial.println("Fout: Lichtsterktemeting mislukt!");
    lux = lastLux;
  }

  if (temperature != lastTemperature) {
    client.publish("home/temp", String(temperature).c_str(), true);
    lastTemperature = temperature;
  }

  if (humidity != lastHumidity) {
    client.publish("home/humidity", String(humidity).c_str(), true);
    lastHumidity = humidity;
  }

  lastLux = lux;

  Blynk.virtualWrite(V3, temperature);
  Blynk.virtualWrite(V8, humidity);
  Blynk.virtualWrite(V9, lux);
}
```

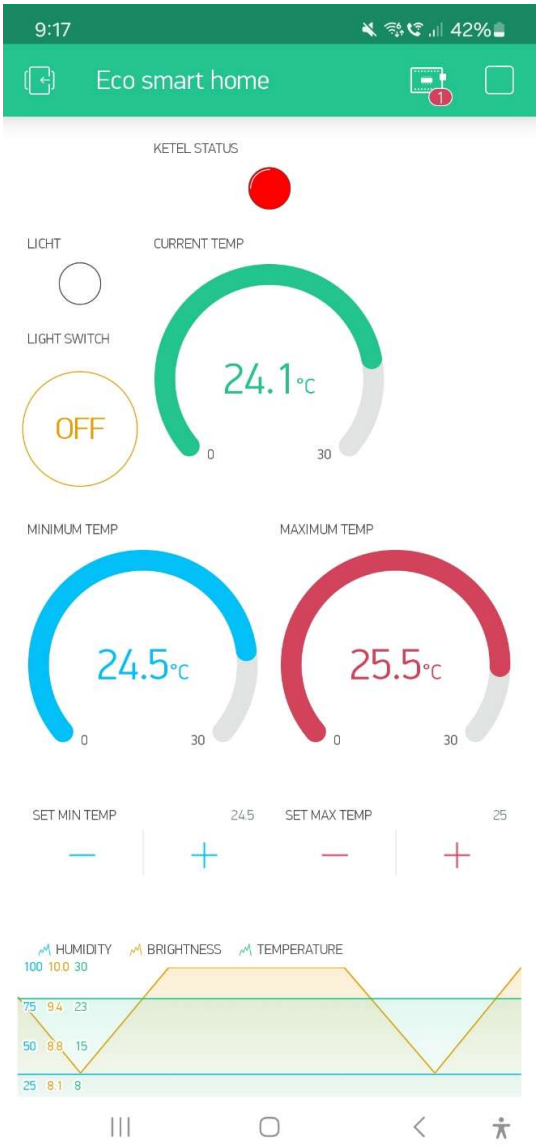
```
// we check if our door is open with the set distance
bool isDoorOpen() {
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);
  long duration = pulseIn(ECHO_PIN, HIGH);
  float distance = duration * 0.034 / 2;
  return (distance < 10.0);
}
```

```
// we set the needed values we want to display as the parameters
// to be able to use them
void displayInformation(float temperature, float humidity, bool doorOpen, bool
lightStatus, float lux) {
  display.clearDisplay();
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.printf("Min: %.1f Max: %.1f\n", min_temp, max_temp);
  display.printf("Temp: %.1fC\n", temperature);
  display.printf("Vocht: %.1f%\n", humidity);
  display.printf("Deur: %s\n", doorOpen ? "OPEN" : "CLOSED");
  display.printf("Licht: %s\n", lightStatus ? "ON" : "OFF");
  display.printf("Helderheid: %.1flx\n", lux);
```

```
  display.display();
}
```

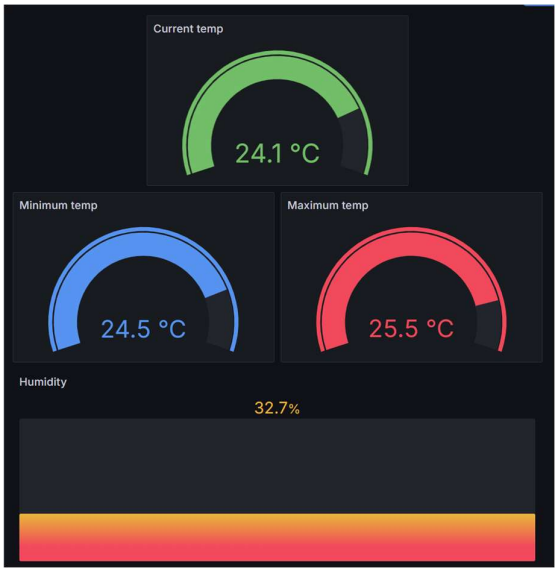
```
// a basic function to change the color of our RGB LED
void setColor(int R, int G, int B) {
  analogWrite(LED_RED, R);
  analogWrite(LED_GREEN, G);
  analogWrite(LED_BLUE, B);
}
```

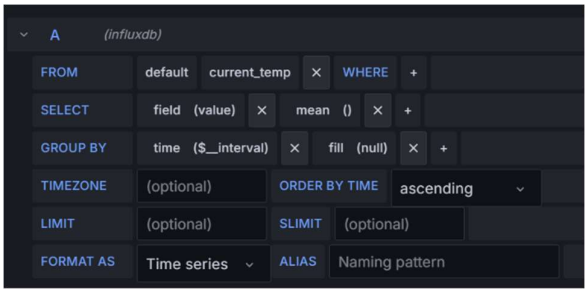
5.2. Blynk app



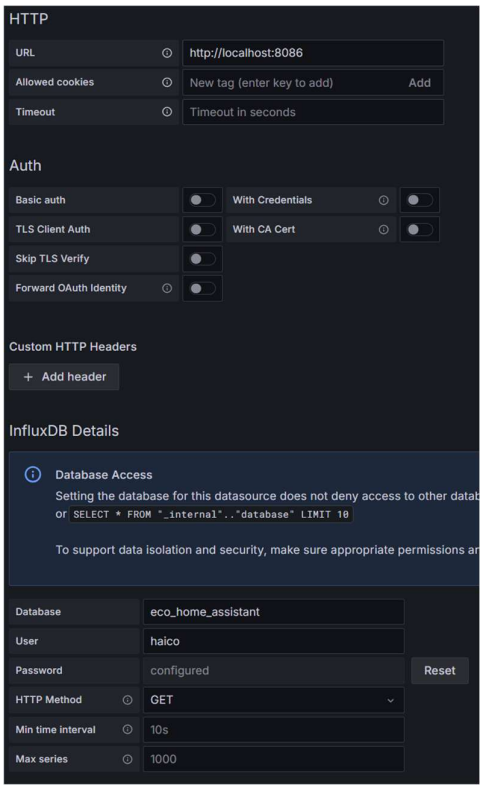

9:18
42%
Eco smart home
OFF
0
30
MINIMUM TEMP
MAXIMUM TEMP
24.5°C
25.5°C
0
30
0
30
SET MIN TEMP
24.5
SET MAX TEMP
25
HUMIDITY
BRIGHTNESS
TEMPERATURE
100 10.0 30
75 9.4 23
50 8.8 15
25 8.1 8
0 7.5 0
Live
15m
30m
1h
1d
TERMINAL
=== HOME ASSISTANT TERMINAL ===
Type 'DOOR' or 'LIGHT' to get status.

5.3. Grafana dashboard configuratie





Voor het visuele gedeelte heb ik gekozen om 3 gauges te gebruiken voor de temperaturen weer te geven en een bar chart om de vochtigheid weer te geven. Hier heb ik minimumwaardes aan gegeven (0-30 voor temp en 0-100 voor vochtigheid) en de kleuren verandert voor een duidelijkere weergave.
Dit allemaal met een gewone weergave van de laatst ingelezen waardes zoals in het voorbeeld hierboven.



Hier heb ik mijn database aan gekoppeld die voor dit project is aangemaakt samen met de user om de database te kunnen gebruiken.

6. Verificatiemethoden

6.1. Hoe heb je gecontroleerd dat MQTT-berichten correct worden verzonden?

6.2. Hoe heb je geverifieerd dat apparaten correct reageren op berichten?

Voor beide vragen heb ik prints gedaan naar de seriële monitor van de data die binnenkomt en de data die wordt verstuurd. Verder zijn er ook visuele veranderingen die gebeuren met de data die toekomt en verstuurd wordt.

Ik heb checks toegevoegd die nagaan of de ingelezen data van de DHT11 een nummer is of niet.

Een print die zegt wat de fout is bij het connecteren met MQTT en vervolgens de connectie herstart bij de ESP32.

En een check die zegt als de OLED niet gevonden kan worden.

6.3. Welke testscenario's heb je doorlopen?

Ik heb vooral getest op de scenario’s waaraan ik heb gedacht dat fout zouden kunnen lopen.

Zo heb ik ook gaandeweg error checks toegevoegd wanneer deze zich voordeden.

7. Video

Ik heb ervoor gekozen om mijn video online te zetten om tijd te besparen met het downloaden van de file en hem meteen beschikbaar te maken.

https://youtu.be/i7wYbE6Hbeo

THOMAS
MORE