Ecole Supérieure
d'Informatique et du Numérique
COLLEGE OF ENGINEERING & ARCHITECTURE

Internet of Things

Project report

ING 4 Cloud computing & virtualisation

Developed by:

Ahmed Reda SIKILLI
Mohamed Ali OULGHAZI

College year: 2023/2024

Introduction

The project aimed to create an Arduino-based system utilizing a DHT11 sensor to monitor temperature and humidity. Additionally, the system employed a Bluetooth module for remote control, enabling adjustments to the monitoring settings.

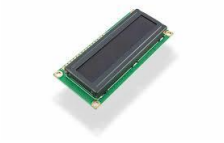
USED MATERIALS:

Arduino Uno Board: Central microcontroller managing sensor data and control functions.

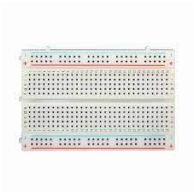
DHT11 Sensor: Acquired precise temperature and humidity data.

Bluetooth module: Enabled wireless communication for remote control.

LCD Display: Provided real-time display of temperature and humidity readings.



Breadboard, Jumper Wires, LED, and 220k Ohm Resistors: Essential for prototyping and electrical connections.




Objectives :

1. Develop an Arduino-based system to accurately monitor temperature and humidity.
2. Implement Bluetooth communication for remote control and adjustments to the monitoring system.
3. Integrate Node-RED to provide a user-friendly dashboard for remote control and visualization.

Methodology:

Arduino Setup:

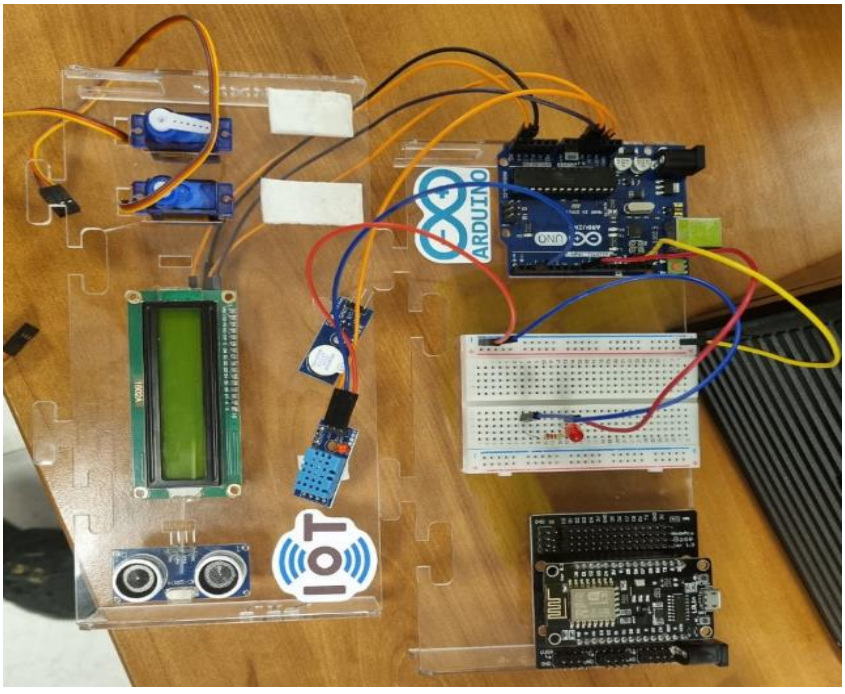
- Interfaced the DHT11 sensor to measure temperature and humidity accurately.
- Integrated a Bluetooth module for wireless remote control.
- Programmed LED indicator based on humidity thresholds.

Node-RED Integration:

- Configured Node-RED to establish communication with the Arduino via Bluetooth.
- Developed a dashboard in Node-RED for real-time visualization of sensor data.
- Implemented control nodes allowing users to adjust settings remotely.

Results:

Arduino:



In this Arduino setup, users will observe an LCD screen presenting real-time temperature and humidity readings. The display showcases information like "Temp=25.50C" and "Humidity=45.00%," providing immediate insights into the environmental conditions. Additionally, an LED indicator serves as a visual cue, responding to the humidity levels. When the humidity exceeds a predefined threshold, typically set at 55%, the LED activates, offering a clear and immediate alert regarding the humidity status.

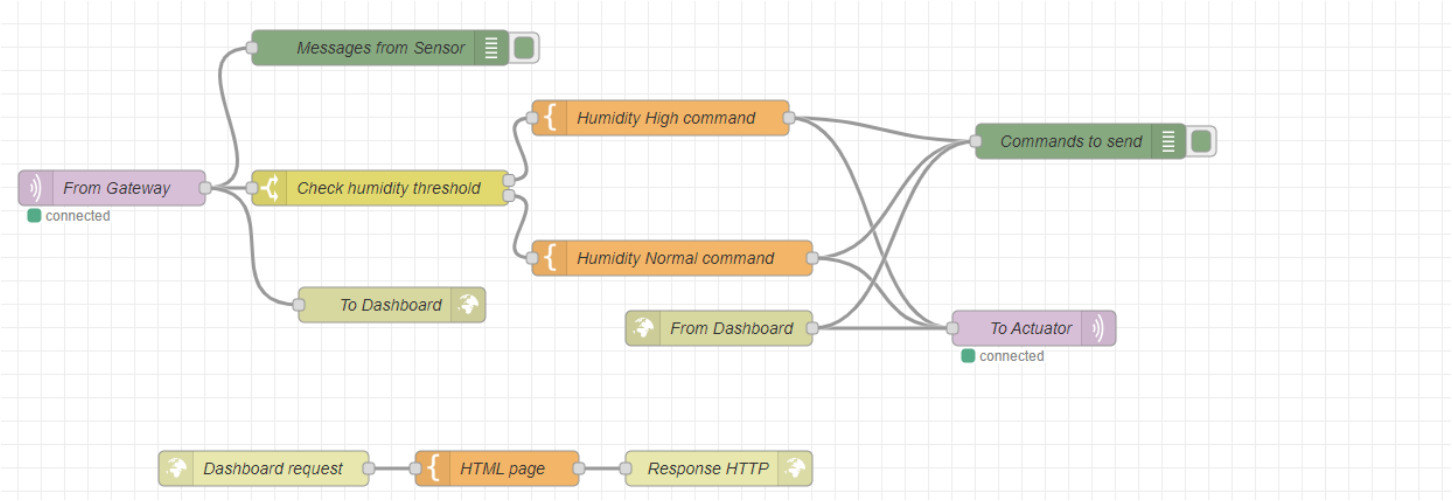
The code for this Setup:

```
#include <DHT.h>
#include <LiquidCrystal_I2C.h>
#include <SoftwareSerial.h>

#define DHTPIN 2
#define DHTTYPE DHT11

#define RX_PIN 11
#define TX_PIN 8
SoftwareSerial btSerial(RX_PIN, TX_PIN);
String btReceivedData;
#define MAX_BT_RECEIVE_DATA 16
int btSendFreq = 1000;
int lastTimeBTSend = 0;

DHT dhthamid(DHTPIN, DHTTYPE);
LiquidCrystal_I2C lcd(0x27, 20, 4);
#define LED_PIN 5
#define HUMIDITY_THRESHOLD 55.0

float adjustedHumidity = 0.0;
float adjustedTemperature = 0.0;

void setup() {
  Serial.begin(9600);
  // Bluetooth initialization
  btSerial.begin(9600);

  lcd.init();
  lcd.backlight();
  dhthamid.begin();
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  int currentTime = millis();
  if (currentTime - lastTimeBTSend >= btSendFreq) {
    // Read DHT sensor values and add the adjustments
    float temperature = dhthamid.readTemperature() + adjustedTemperature;
    float humidity = dhthamid.readHumidity() + adjustedHumidity;

    // Display sensor readings on LCD
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Temp=");
    lcd.print(temperature);
```

```
    lcd.print("C");

    lcd.setCursor(0, 1);
    lcd.print("Humidity=");
    lcd.print(humidity);
    lcd.print("%");

    String telemetry = "T:" + String(temperature, 2) + ",H:" + String(humidity, 2);

    btSerial.println(telemetry);
    lastTimeBTSend = currentTime;

    if ((humidity + adjustedHumidity) >= HUMIDITY_THRESHOLD) {
      digitalWrite(LED_PIN, HIGH);
    } else {
      digitalWrite(LED_PIN, LOW);
    }
  }

  // Receive part
  while (btSerial.available()) {
    char c = btSerial.read();
    if (c == '\n' || btReceivedData.length() == MAX_BT_RECEIVE_DATA) {
      // Adjust temperature and humidity based on received Bluetooth data
      if (btReceivedData.startsWith("temp+")) {
        float tempChange = btReceivedData.substring(5).toFloat();
        adjustedTemperature += tempChange;
      } else if (btReceivedData.startsWith("temp-")) {
        float tempChange = btReceivedData.substring(5).toFloat();
        adjustedTemperature -= tempChange;
      } else if (btReceivedData.startsWith("humidity+")) {
        float humChange = btReceivedData.substring(9).toFloat();
        adjustedHumidity += humChange;
      } else if (btReceivedData.startsWith("humidity-")) {
        float humChange = btReceivedData.substring(9).toFloat();
        adjustedHumidity -= humChange;
      }

      btReceivedData = "";
    } else if (c != '\r') {
      btReceivedData += c;
    }
  }

  delay(100);
}
```

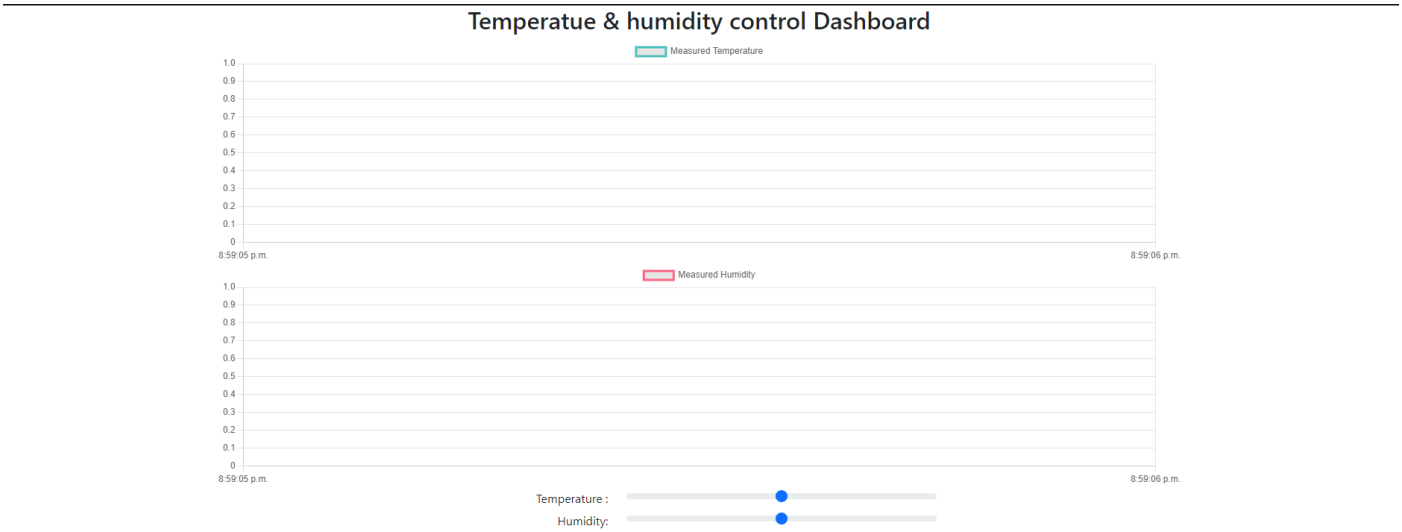
This Arduino code orchestrates a comprehensive system that enables users to monitor environmental conditions and make adjustments remotely. It employs a DHT11 sensor to capture temperature and humidity data, which is then displayed in real-time on an LCD screen, providing users with immediate environmental insights. The code facilitates Bluetooth communication, allowing users to interact with the system wirelessly. Through a connected device, users can remotely adjust temperature and humidity offsets, influencing the sensor's readings. Moreover, the system incorporates an LED indicator that responds to humidity levels, illuminating when the humidity surpasses a preset threshold. Overall, this code amalgamates sensor readings, a user-friendly display, wireless communication, and adjustable thresholds, empowering users to monitor and control environmental conditions conveniently and efficiently, whether for personal use or specific applications.

Node-Red Structure:



This interface configuration shows a system to manage temperature and humidity control via MQTT, WebSocket connections, and a graphical dashboard. It includes nodes to receive telemetry data from a specific MQTT topic, evaluate humidity thresholds, generate commands based on the readings, and send these commands to an actuator via MQTT. The dashboard, accessible via an HTTP request, offers real-time visualization of temperature and humidity data through charts. It also provides interactive sliders enabling users to adjust these parameters. WebSocket connections facilitate bidirectional communication between the dashboard and the system, allowing seamless updates and control functionalities. Overall, this interface integrates various nodes to create a comprehensive system for monitoring and controlling temperature and humidity levels.

Dashboard:



This webpage is designed to show real-time temperature and humidity data through two line graphs. Users can interact with the page using sliders to set their preferred temperature and humidity levels. It's like a control panel where you can see how hot or humid it is and adjust settings to reach the desired conditions. The sliders act like controls, allowing you to change the settings and potentially influence the environment the system is monitoring. The graphs visually represent how the conditions change over time, giving users a clear view of the trends.

Gateway code:

```
import serial
import paho.mqtt.client as mqtt
import json
import time

# Serial init
serial_port = '/dev/ttyUSB0'  # Replace with your Arduino's serial port
baud_rate = 9600
btSerial = serial.Serial(serial_port, baud_rate)
btSerial.timeout = 1

# MQTT topics
client_id = 'ad4b0184-b950-4b55-b8d5-3f0278174859'
telemetry_topic = client_id + '/telemetry'
commands_topic = client_id + '/commands'

# MQTT init
mqtt_client = mqtt.Client(client_id)
mqtt_client.connect('test.mosquitto.org')
mqtt_client.loop_start()
```

```python
# Commands handling part
def handle_commands(client, userdata, message):
    command = json.loads(message.payload.decode())
    print("Command received:", command)
    # Parse commands
    if "humidity+" in command:
        humidity_plus = f"humidity+{command['humidity+']}\n"
        btSerial.write(humidity_plus.encode())
    elif "humidity-" in command:
        humidity_minus = f"humidity-{command['humidity-']}\n"
        btSerial.write(humidity_minus.encode())
    elif "temp+" in command:
        temp_plus = f"temp+{command['temp+']}\n"
        btSerial.write(temp_plus.encode())
    elif "temp-" in command:
        temp_minus = f"temp-{command['temp-']}\n"
        btSerial.write(temp_minus.encode())

mqtt_client.subscribe(commands_topic)
mqtt_client.on_message = handle_commands

# Telemetry handling part
while True:
    # Receive the telemetry from BT
    btTelemetry = btSerial.readline().decode('ascii').strip()
    if btTelemetry != '':
        print('Received BT telemetry:', btTelemetry)
        # Create the telemetry JSON
        telemetryValues = btTelemetry.split(',')
        if len(telemetryValues) != 2:
            print('Incorrect number of values received')
        else:
            try:
                # Parse values from the device telemetry string
                humidity = float(telemetryValues[0])
                temperature = float(telemetryValues[1])
                telemetryJSON = {'humidity': humidity, 'temperature': temperature}

                # Add timestamp in milliseconds
                telemetryJSON['timestamp'] = round(time.time() * 1000)
                telemetryJSONStr = json.dumps(telemetryJSON)

                # Send to Cloud
                print("Sending telemetry:", telemetryJSONStr)
                mqtt_client.publish(telemetry_topic, telemetryJSONStr, qos=1)
            except ValueError:
                print('Wrong telemetry format')
```

This Python code operates as a vital intermediary, connecting an Arduino device to an MQTT broker. It orchestrates bidirectional communication between these components. It establishes a serial link with the Arduino for data exchange and configures MQTT connections for message transmission. Upon receiving commands via MQTT topics, it interprets and translates these commands into specific serial instructions to adjust the behavior of the connected Arduino. Simultaneously, it constantly retrieves telemetry data from the Arduino, formats it into JSON structures containing humidity, temperature, and timestamps, and publishes this data to the MQTT broker. This gateway enables external systems to remotely control the Arduino's operations by sending commands and also retrieve real-time telemetry data via MQTT, facilitating a seamless interaction with the Arduino from external applications or devices connected to the broker.

Conclusion

This project orchestrates a comprehensive system that connects an Arduino device, an MQTT broker, a Node-RED flow, and a web-based dashboard. The Arduino collects telemetry data related to temperature and humidity, communicates this data via serial communication to a Python gateway, which then publishes this data to an MQTT broker. Simultaneously, it listens for commands from the MQTT broker to adjust Arduino settings.

Node-RED, a flow-based development tool, processes telemetry data received from the MQTT broker, allowing for real-time visualization, decision-making based on thresholds, and sending commands back to the Arduino through the broker.

The web-based dashboard leverages Chart.js to visualize telemetry data and incorporates sliders for manual control. WebSocket connections facilitate real-time updates between the dashboard and the Arduino.

This holistic integration demonstrates a sophisticated IoT ecosystem, enabling remote monitoring and control over Arduino devices while providing a user-friendly interface for data visualization and manual adjustments, contributing to an efficient and interactive system for managing temperature and humidity.