Compte-rendu

Travaux mini projet

Step1 : Affichage des données localement sur le serveur web

1.1-code Arduino

```
// Import required libraries
#ifdef ESP32
  #include <WiFi.h>
  #include <ESPAsyncWebServer.h>
  #include <SPIFFS.h>
#else
  #include <Arduino.h>
  #include <ESP8266WiFi.h>
  #include <Hash.h>
  #include <ESPAsyncTCP.h>
  #include <ESPAsyncWebServer.h>
  #include <FS.h>
#endif
#include <Wire.h>

/*#include <SPI.h>
#define BME_SCK 18
#define BME_MISO 19
#define BME_MOSI 23
#define BME_CS 5*/



//Adafruit_BME280 bme(BME_CS); // hardware SPI
```

```
//Adafruit_BME280 bme(BME_CS, BME_MOSI, BME_MISO, BME_SCK); // software SPI

// Replace with your network credentials
const char* ssid = "Galaxy A04s1A56";
const char* password = "ccqg5114";



// Create AsyncWebServer object on port 80
AsyncWebServer server(80);

String readBME280Temperature() {
  // Read temperature as Celsius (the default)
  float t = 10*random();
  // Convert temperature to Fahrenheit
  //t = 1.8 * t + 32;
  if (isnan(t)) {
    Serial.println("Failed to read from BME280 sensor!");
    return "";
  }
  else {
    Serial.println(t);
    return String(t);
  }
}

void setup(){
  // Serial port for debugging purposes
  Serial.begin(115200);
```

```
bool status;
// default settings
// (you can also pass in a Wire library object like &Wire2)

// Initialize SPIFFS
if(!SPIFFS.begin()){
  Serial.println("An Error has occurred while mounting SPIFFS");
  return;
}

// Connect to Wi-Fi
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.println("Connecting to WiFi..");
}

// Print ESP32 Local IP Address
Serial.println(WiFi.localIP());

// Route for root / web page
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send(SPIFFS, "/index.html");
});
server.on("/temperature", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send_P(200, "text/plain", readBME280Temperature().c_str());
});
// Start server
server.begin();
```

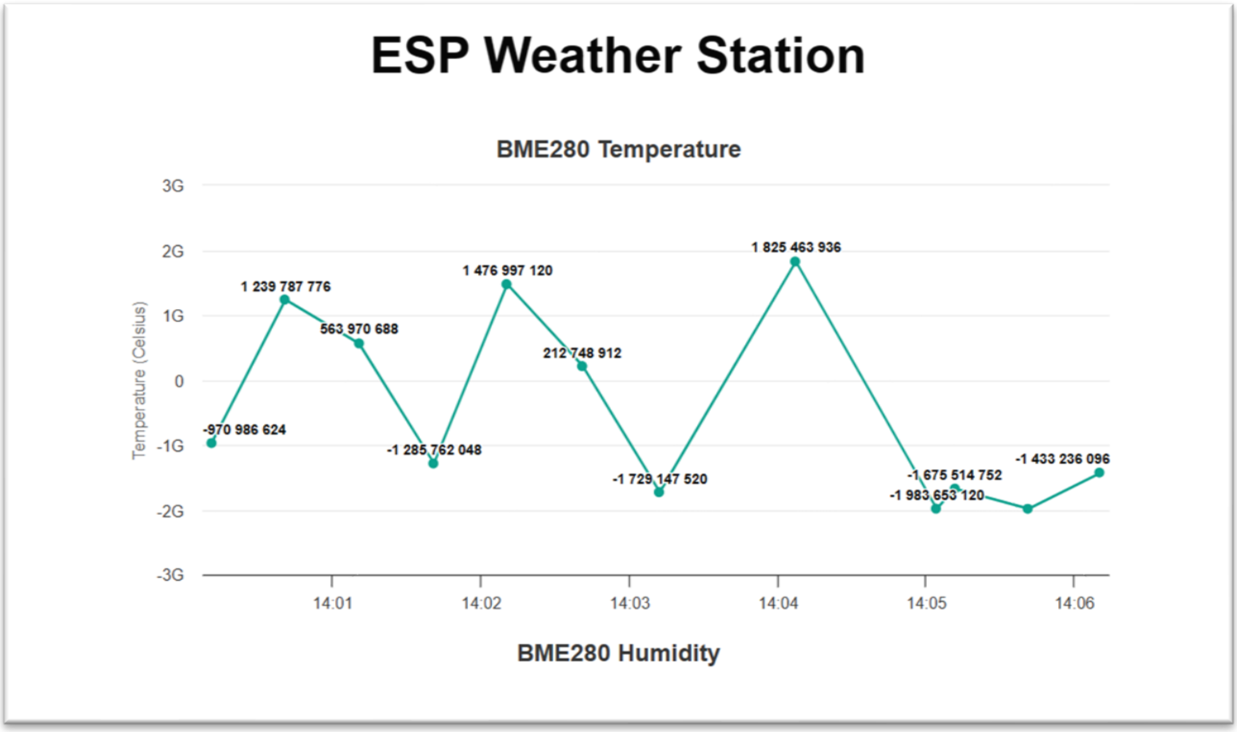
```
    } void loop(){

}
```

1.2-code index.html :

```
<!DOCTYPE html>
<html>
<head>
    <title>Temperature and Humidity Monitor</title>
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>
<body>
    <h1>Temperature and Humidity Monitor</h1>
    <canvas id="temperatureChart" width="400" height="200"></canvas>
    <canvas id="humidityChart" width="400" height="200"></canvas>

    <script>
        // Function to create a new chart
        function createChart(canvasId, label, data) {
            var ctx = document.getElementById(canvasId).getContext('2d');
            return new Chart(ctx, {
                type: 'line',
                data: {
                    labels: data.map(function (value, index) { return index; }),
                    datasets: [{
                        label: label,
                        data: data,
                        borderColor: 'rgb(75, 192, 192)',
                        tension: 0.1
                    }]
                },
                options: {
                    scales: {
                        y: {
                            beginAtZero: true
                        }
                    }
                }
            });
        }

        // Function to update chart data
        function updateChart(chart, newData) {
            chart.data.datasets[0].data.push(newData);
            chart.update();
        }

        // Create charts
        var temperatureData = [];
        var humidityData = [];
        var temperatureChart = createChart('temperatureChart', 'Temperature (°C)',
temperatureData);
        var humidityChart = createChart('humidityChart', 'Humidity (%)', humidityData);

        // WebSocket connection to ESP32 server
        var socket = new WebSocket('ws://' + window.location.hostname + ':81/');
        socket.onmessage = function (event) {
```

```
            var data = JSON.parse(event.data);
            updateChart(temperatureChart, data.temperature);
            updateChart(humidityChart, data.humidity);
        };
    </script>
</body>
</html>
```

1.3-capture



ESP Weather Station

BME280 Temperature

BME280 Humidity

Step 2: stockage et affichage des données dans le cloud influx Db

2.1-Code :

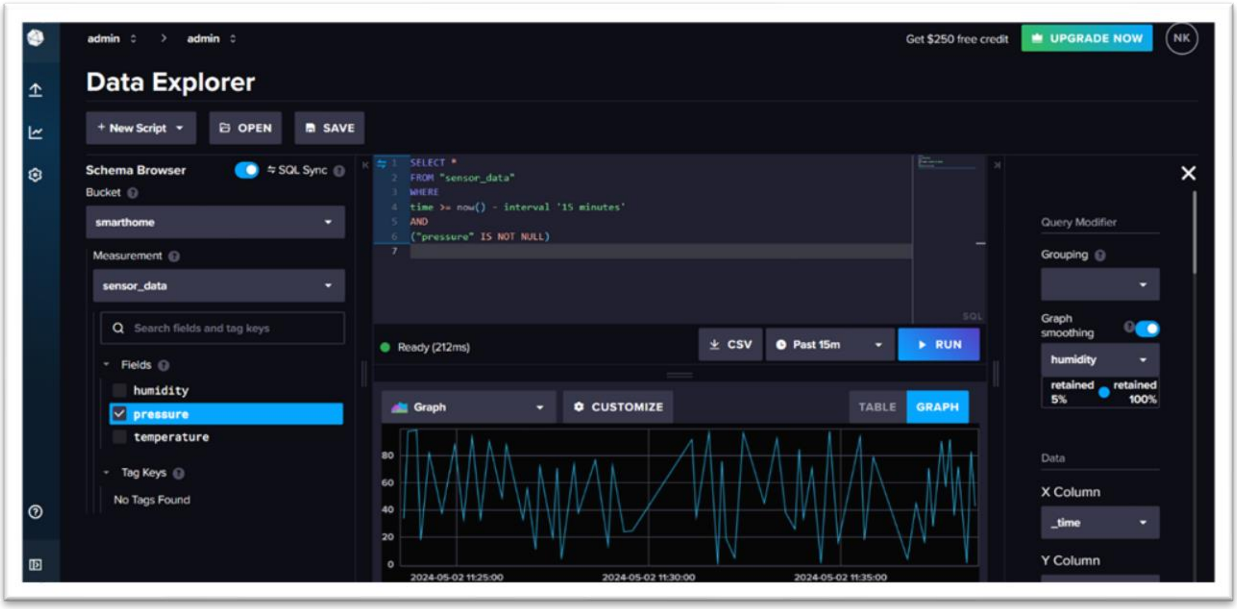
```
 #include <Arduino.h>
#include <WiFi.h>
#include <ESPAsyncWebServer.h>
#include <SPIFFS.h>
#include <Wire.h>
#include <InfluxDbClient.h>
```

```
#include <InfluxDbCloud.h>


// Définir les constantes pour les identifiants de connexion Wi-Fi et d'InfluxDB
const char* ssid = "Galaxy A04s1A56";
const char* password = "ccqg5114";
#define INFLUXDB_URL "https://us-east-1-1.aws.cloud2.influxdata.com"
#define INFLUXDB_TOKEN "XJ7q8IVIm5Po3yfRdRub9SCuERKdiFjMNdM9OzsPAvv3Xh_ZP83McX2EdQkt6HRnsHPGGtnkw8nB2UTkQWpxyw=="
#define INFLUXDB_ORG "34278df617e29c1c"
#define INFLUXDB_BUCKET "smarthome"
#define TZ_INFO "UTC1"
  float temperature ;
  float humidity;
  float pressure;
// Créer une instance du client InfluxDB
InfluxDBClient client(INFLUXDB_URL, INFLUXDB_ORG, INFLUXDB_BUCKET, INFLUXDB_TOKEN, InfluxDbCloud2CACert);


// Créer une instance du serveur web asynchrone sur le port 80
AsyncWebServer server(80);


// Fonction pour lire la température (valeur aléatoire pour l'exemple)
float readBME280Temperature() {
  return random(0, 100);
}


// Fonction pour lire l'humidité (valeur aléatoire pour l'exemple)
float readBME280Humidity() {
  return random(0, 100);
}
```

```
// Fonction pour lire la pression atmosphérique (valeur aléatoire pour l'exemple)
float readBME280Pressure() {
  return random(900, 1100);
}

// Fonction pour écrire les données dans InfluxDB
void writeToInfluxDB(float temperature, float humidity, float pressure) {
  Point dataPoint("sensor_data");
  dataPoint.addField("temperature", temperature);
  dataPoint.addField("humidity", humidity);
  dataPoint.addField("pressure", pressure);

  if (client.writePoint(dataPoint)) {
    Serial.println("Données écrites avec succès dans InfluxDB !");
  } else {
    Serial.println("Échec de l'écriture des données dans InfluxDB !");
    Serial.println(client.getLastErrorMessage());
  }
}

// Initialisation du programme
void setup() {
  Serial.begin(115200);

  // Connexion au réseau Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi..");
  }
   // Print ESP32 Local IP Address
  Serial.println(WiFi.localIP());


  // Initialisation du serveur web
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/index.html");
  });

  // Gestion des requêtes pour obtenir la température, l'humidité et la pression
  server.on("/temperature", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain", String(readBME280Temperature()).c_str());
  });
  server.on("/humidity", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain", String(readBME280Humidity()).c_str());
  });
  server.on("/pressure", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain", String(readBME280Pressure()).c_str());
  });

  // Démarrage du serveur web
  server.begin();

  // Vérification de la connexion à InfluxDB
  if (client.validateConnection()) {
    Serial.print("Connected to InfluxDB: ");
    Serial.println(client.getServerUrl());
  } else {
    Serial.print("InfluxDB connection failed: ");
    Serial.println(client.getLastErrorMessage());
  }
}
```

```
// Boucle principale du programme
void loop() {

  // Obtenir les valeurs de température, humidité et pression (valeurs aléatoires pour l'exemple)
 temperature = readBME280Temperature();
   humidity = readBME280Humidity();
   pressure = readBME280Pressure();
   Serial.println(temperature);
    Serial.println(humidity);
     Serial.println(pressure);



  // Écrire les données dans InfluxDB
  writeToInfluxDB(temperature, humidity, pressure);

  delay(5000); // Attendre 5 secondes avant la prochaine lecture
}
```

2.2-Captures :



Bucket: smarthome

Measurement: sensor_data

Fields: ☑ pressure

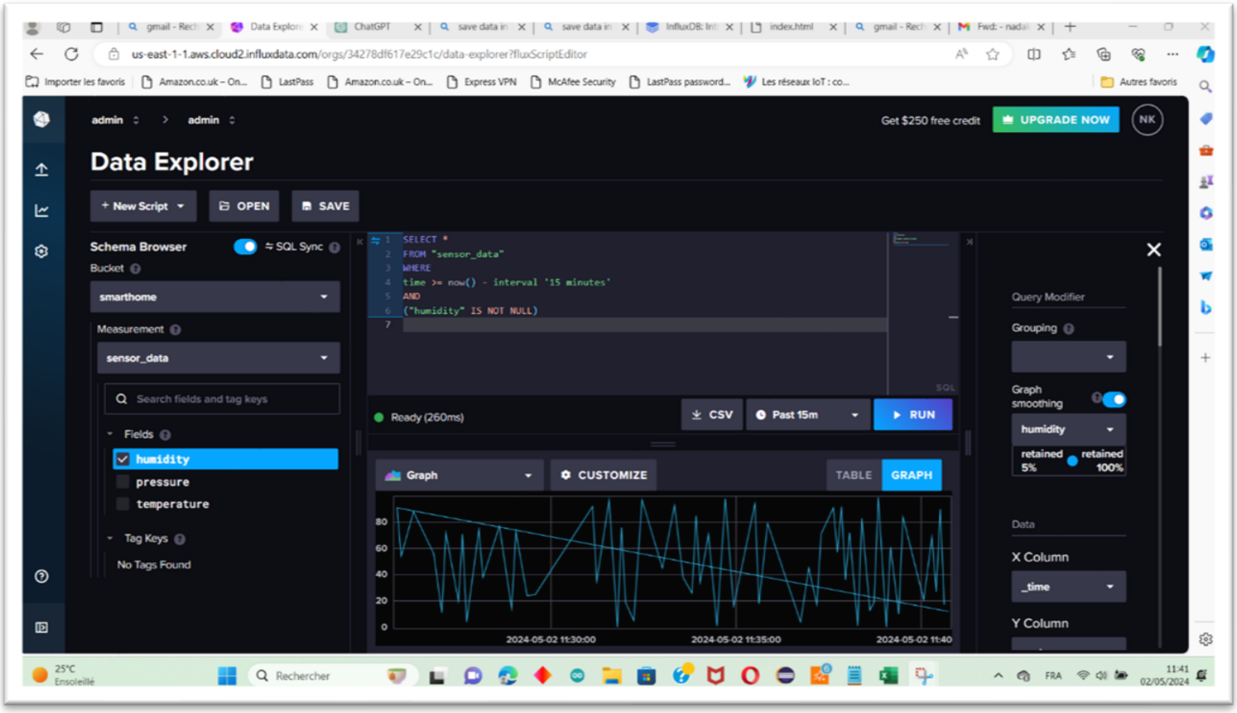
```
SELECT *
FROM "sensor_data"
WHERE
time >= now() - interval '15 minutes'
AND
("pressure" IS NOT NULL)
```

Ready (212ms)

Past 15m

Graph smoothing: humidity

X Column: _time



Bucket: smarthome

Measurement: sensor_data

Fields: ☑ humidity

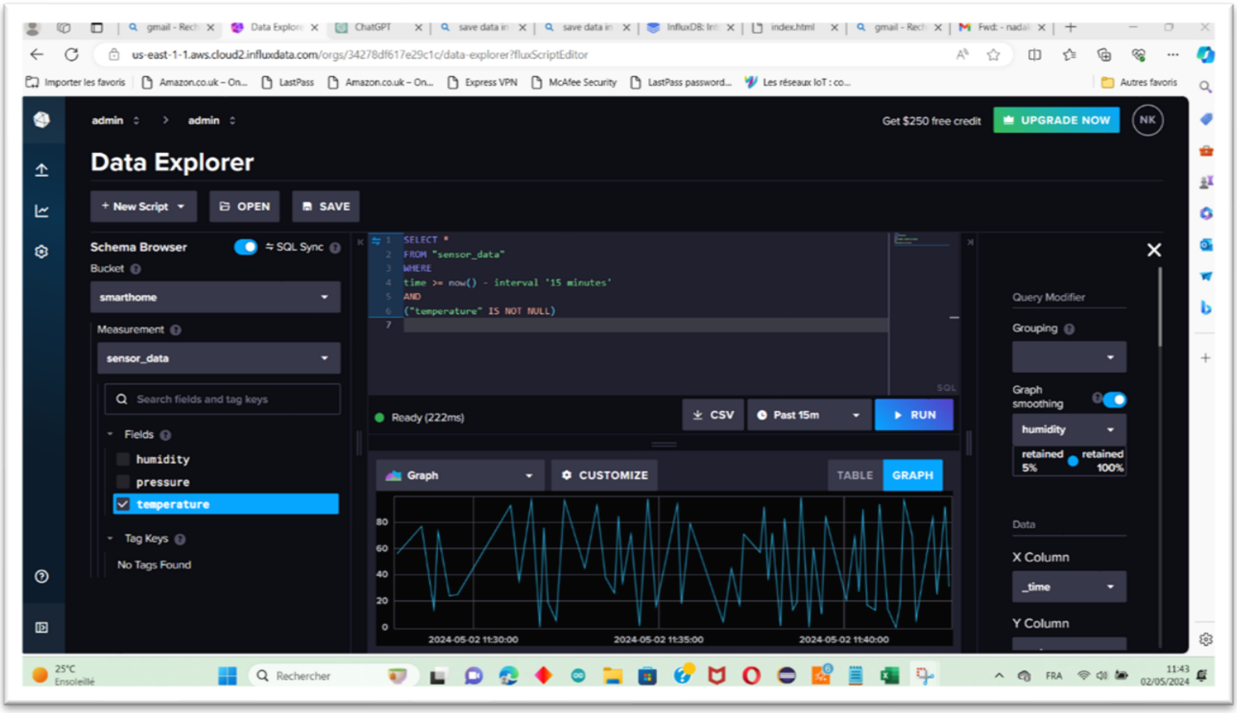
```
SELECT *
FROM "sensor_data"
WHERE
time >= now() - interval '15 minutes'
AND
("humidity" IS NOT NULL)
```

Ready (260ms)

Past 15m

Graph smoothing: humidity

X Column: _time



Bucket: smarthome

Measurement: sensor_data

Fields: humidity, pressure, ☑ temperature

Tag Keys: No Tags Found

```
SELECT *
FROM "sensor_data"
WHERE
time >= now() - interval '15 minutes'
AND
("temperature" IS NOT NULL)
```

Ready (222ms)

Grouping

Graph smoothing: humidity, retained 5% – retained 100%

X Column: _time

Y Column

Step3 : visualisation des données sur le cloud influx DB et sur le serveur web local

3.1-sketch Arduino :

```
#ifdef ESP32
  #include <WiFi.h>
  #include <ESPAsyncWebServer.h>
  #include <SPIFFS.h>
#else
  #include <Arduino.h>
  #include <ESP8266WiFi.h>
  #include <Hash.h>
  #include <ESPAsyncTCP.h>
  #include <ESPAsyncWebServer.h>
  #include <FS.h>
#endif
#include <Wire.h>
#include <InfluxDbClient.h>
#include <InfluxDbCloud.h>

const char* ssid = "Galaxy A04s1A56";
const char* password = "ccqg5114";
#define INFLUXDB_URL "https://us-east-1-1.aws.cloud2.influxdata.com"
#define INFLUXDB_TOKEN
"XJ7q8IVIm5Po3yfRdRub9SCuERKdiFjMNdM9OzsPAvv3Xh_ZP83McX2EdQkt6HRnsHPGGtnk
w8nB2UTkQWpxyw=="
#define INFLUXDB_ORG "34278df617e29c1c"
#define INFLUXDB_BUCKET "smarthome"
#define TZ_INFO "UTC1"

AsyncWebServer server(80);
InfluxDBClient client(INFLUXDB_URL, INFLUXDB_ORG, INFLUXDB_BUCKET,
INFLUXDB_TOKEN, InfluxDbCloud2CACert);

String readBME280Temperature() {
  float t = 10 * random();
  if (isnan(t)) {
    Serial.println("Failed to read from BME280 sensor!");
    return "";
  } else {
    Serial.println(t);
    return String(t);
  }
}

String readBME280Humidity() {
  float h = random(0, 100);
  Serial.println(h);
  return String(h);
}

String readBME280Pressure() {
  float p = random(900, 1100);
  Serial.println(p);
  return String(p);
}

void writeToInfluxDB(float temperature, float humidity, float pressure) {
  Point dataPoint("sensor_data");
  dataPoint.addField("temperature", temperature);
  dataPoint.addField("humidity", humidity);
  dataPoint.addField("pressure", pressure);

  if (client.writePoint(dataPoint)) {
    Serial.println("Data written successfully to InfluxDB!");
  } else {
    Serial.println("Failed to write data to InfluxDB!");
    Serial.println(client.getLastErrorMessage());
  }
```

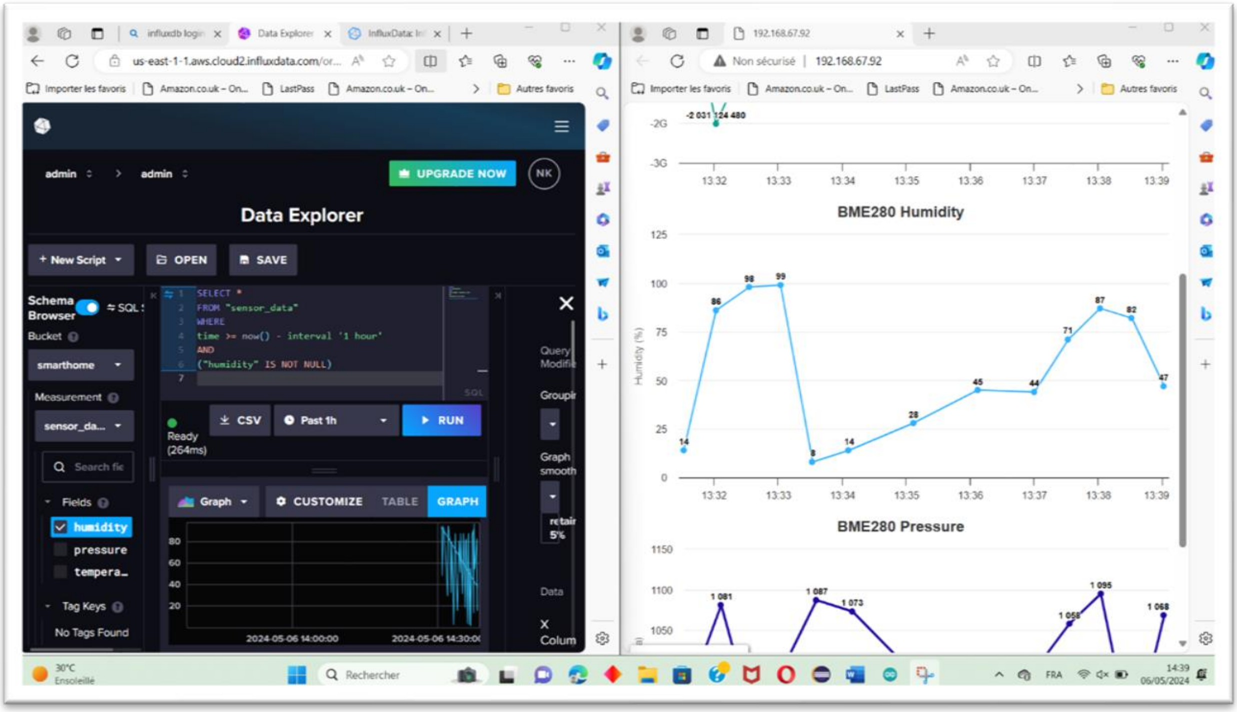
```
}

void setup() {
 Serial.begin(115200);

 bool status;

 if (!SPIFFS.begin()) {
  Serial.println("An Error has occurred while mounting SPIFFS");
  return;
 }

 WiFi.begin(ssid, password);
 while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.println("Connecting to WiFi..");
 }

 Serial.println(WiFi.localIP());

 server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send(SPIFFS, "/index.html");
 });

 server.on("/temperature", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send_P(200, "text/plain", readBME280Temperature().c_str());
 });

 server.on("/humidity", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain", readBME280Humidity().c_str());
  });

  server.on("/pressure", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain", readBME280Pressure().c_str());
  });

  server.begin();

  timeSync(TZ_INFO, "pool.ntp.org", "time.nis.gov");

  if (client.validateConnection()) {
    Serial.print("Connected to InfluxDB: ");
    Serial.println(client.getServerUrl());
  } else {
    Serial.print("InfluxDB connection failed: ");
    Serial.println(client.getLastErrorMessage());
  }
}

void loop() {
  float temperature = readBME280Temperature().toFloat();
  float humidity = readBME280Humidity().toFloat();
  float pressure = readBME280Pressure().toFloat();

  writeToInfluxDB(temperature, humidity, pressure);

  delay(5000);
}
```

3.2-Code index.html :

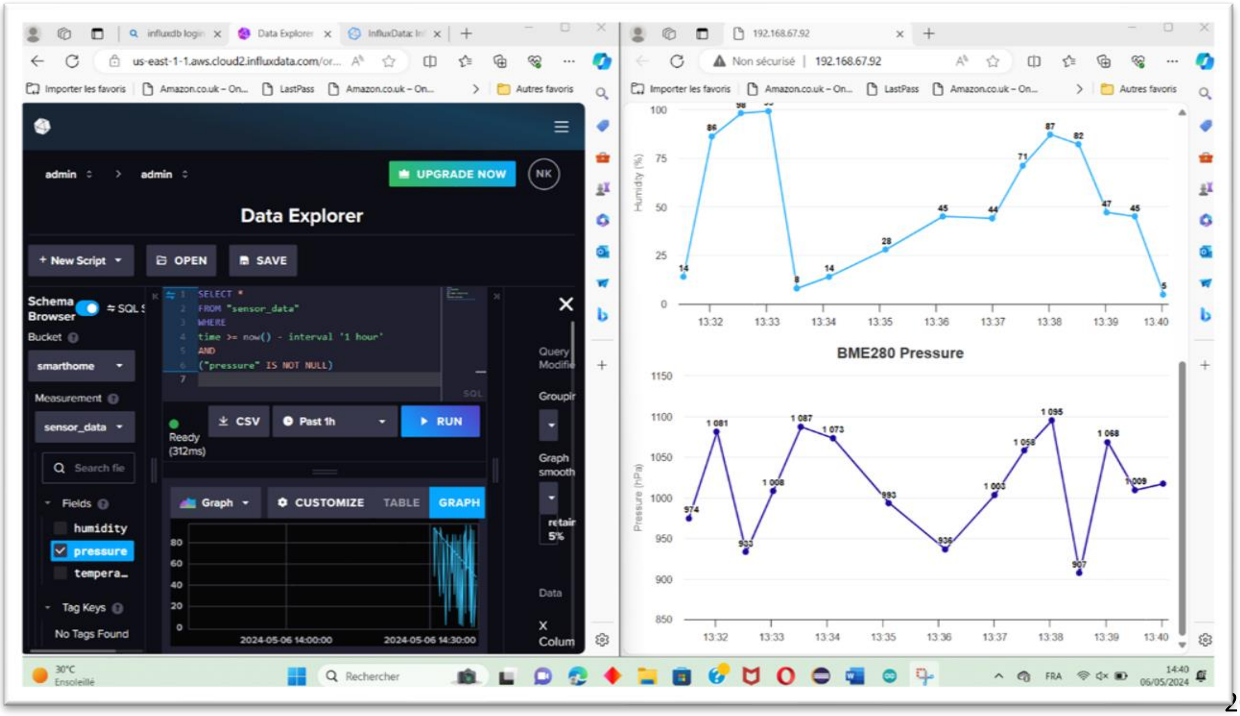
```
<!DOCTYPE html>
<html>
<head>
  <title>Temperature, Humidity and Pressure Monitor</title>
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>
<body>
  <h1>Temperature, Humidity and Pressure Monitor</h1>
  <canvas id="temperatureChart" width="400" height="200"></canvas>
  <canvas id="humidityChart" width="400" height="200"></canvas>
  <canvas id="pressureGauge" width="400" height="200"></canvas>

  <script>
    function createChart(canvasId, label, data) {
      var ctx = document.getElementById(canvasId).getContext('2d');
      return new Chart(ctx, {
        type: 'line',
        data: {
          labels: data.map(function (value, index) { return index; }),
          datasets: [{
            label: label,
            data: data,
            borderColor: 'rgb(75, 192, 192)',
            tension: 0.1
          }]
        },
        options: {
          scales: {
                y: {
                    beginAtZero: true
                }
            }
        }
    });
}

function createGauge(canvasId, label, value) {
    var ctx = document.getElementById(canvasId).getContext('2d');
    return new Chart(ctx, {
        type: 'doughnut',
        data: {
            datasets: [{
                data: [value, 100 - value],
                backgroundColor: [
                    'rgba(75, 192, 192, 1)',
                    'rgba(255, 255, 255, 0.2)'
                ]
            }],
            labels: [label, '']
        },
        options: {
            cutout: '90%',
            rotation: 1 * Math.PI,
            circumference: 1 * Math.PI,
            plugins: {
                legend: {
                    display: false
        }
      }
    }
  });
}

var temperatureData = [];
var humidityData = [];
var pressureGauge = createGauge('pressureGauge', 'Pressure (hPa)', 50);

var temperatureChart = createChart('temperatureChart', 'Temperature (°C)', temperatureData);
var humidityChart = createChart('humidityChart', 'Humidity (%)', humidityData);

var socket = new WebSocket('ws://' + window.location.hostname + ':81/');
socket.onmessage = function (event) {
  var data = JSON.parse(event.data);
  updateChart(temperatureChart, data.temperature);
  updateChart(humidityChart, data.humidity);
  updateGauge(pressureGauge, data.pressure);
};

function updateChart(chart, newData) {
  chart.data.datasets[0].data.push(newData);
  chart.update();
}

function updateGauge(chart, newValue) {
  chart.data.datasets[0].data[0] = newValue;
  chart.data.datasets[0].data[1] = 100 - newValue;
```

```
        chart.update();
      }
   </script>
</body>
</html>
```

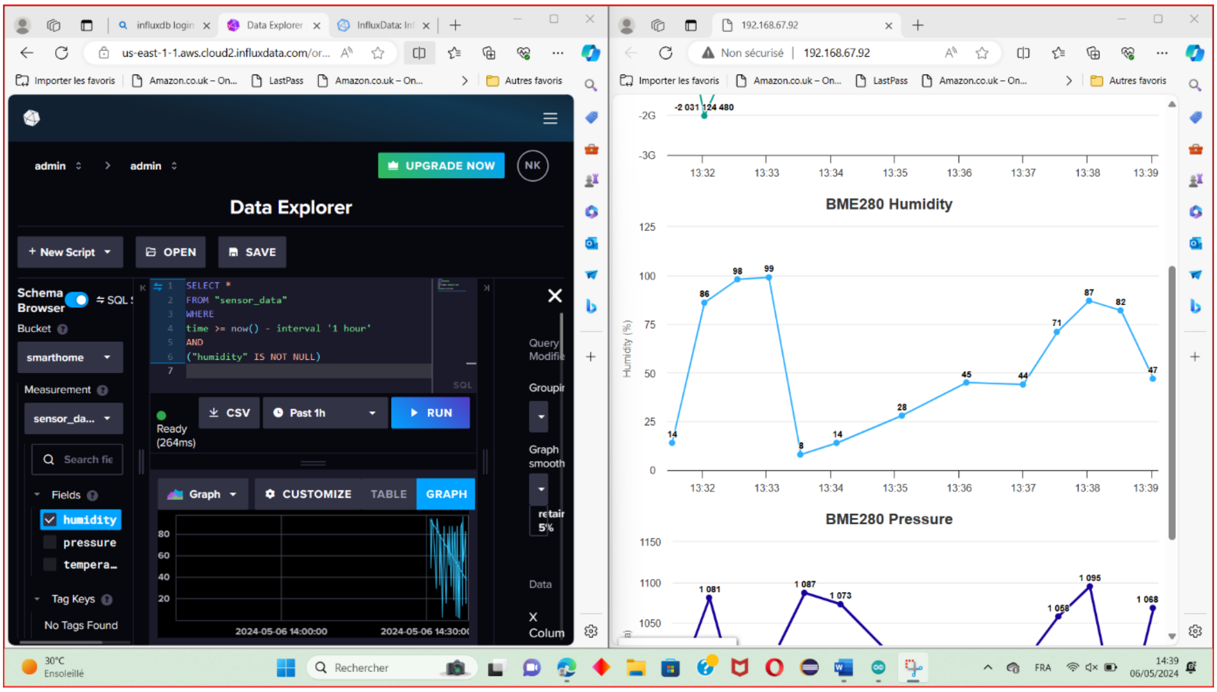
3.3-Captures



```
SELECT *
FROM "sensor_data"
WHERE
time >= now() - interval '1 hour'
AND
("humidity" IS NOT NULL)
```



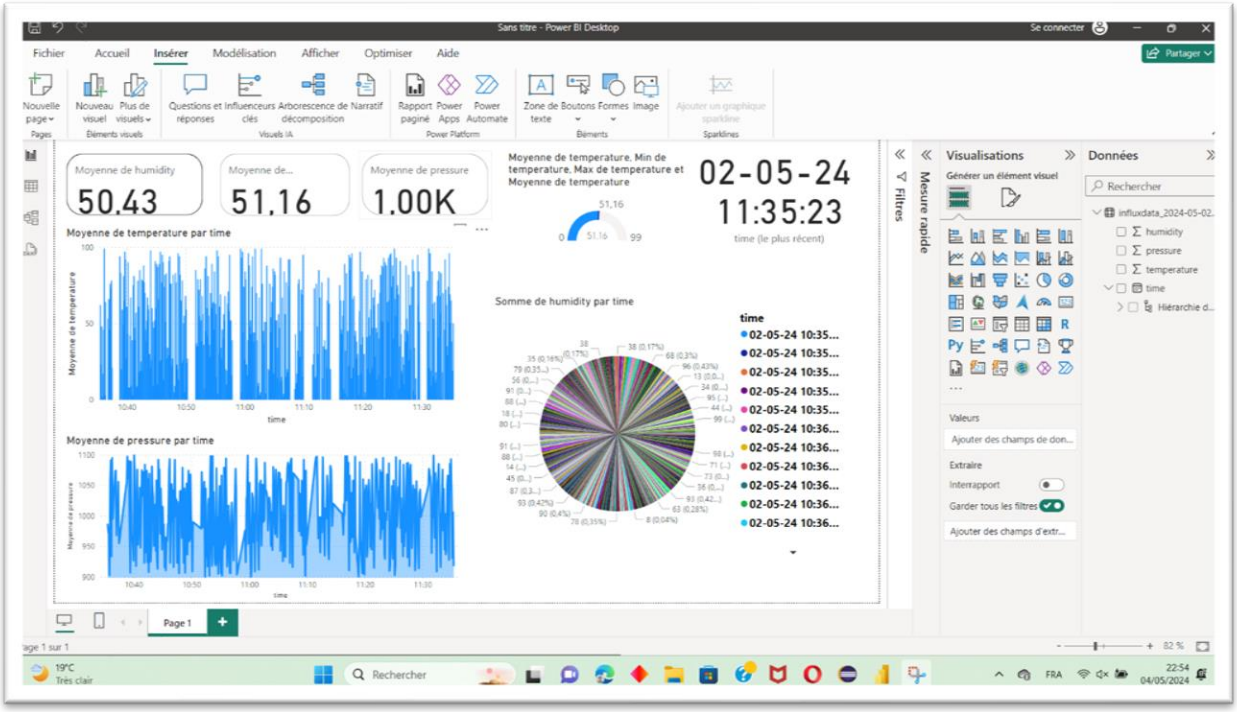
```
SELECT *
FROM "sensor_data"
WHERE
time >= now() - interval '1 hour'
AND
("pressure" IS NOT NULL)
```



```sql
SELECT *
FROM "sensor_data"
WHERE
time >= now() - interval '1 hour'
AND
("humidity" IS NOT NULL)
```

Step 4 : affichage des données stockées dans influx DB sur power BI :

On a téléchargé un fichier Excel qui contient les données collectées pendant une période du temps puis on a affiché la figure suivante :



Sans titre - Power BI Desktop
Fichier
Accueil
Insérer
Modélisation
Afficher
Optimiser
Aide
Moyenne de humidity
50.43
Moyenne de...
51.16
Moyenne de pressure
1.00K
Moyenne de temperature, Min de temperature, Max de temperature et Moyenne de temperature
02-05-24 11:35:23
time (le plus récent)
Moyenne de temperature par time
Moyenne de pressure par time
Somme de humidity par time
Visualisations
Données
influxdata_2024-05-02...
humidity
pressure
temperature
time
Page 1