Sistem de monitorizare a calitatii aerului in timp real

Butacu Laura-Diana
344C1
Github: https://github.com/dianabtc/AirQualityMonitoringIoT

Cuprins

1. Introducere

Proiectul “Air Quality Monitoring IoT” propune o solutie IoT care permite monitorizarea calitatii aerului in timp real utilizand senzori si actuatori integrati cu un microcontroler ESP32. Scopul principal al proiectului este de a colecta, analiza si afisa date relevante despre calitatea aerului, precum temperatura, umiditatea si nivelul de gaze nocive din aer, oferind utilizatorului posibilitatea de a fi informat si de a reactiona prompt in cazul in care conditiile sunt nesigure.

Sistemul foloseste urmatoarele componente:

- Microcontroler ESP32
- Senzor DHT11 pentru masurarea temperaturii si umiditatii
- Senzor MQ135 pentru detectarea nivelurilor de gaze precum dioxidul de carbon, amoniac si alti compusi organici volatili
- Ventilator controlat printr-un tranzistor pentru a imbunatati conditiile de aerisire in cazul in care sunt detectate niveluri periculoase de gaze
- LED RGB: pentru a indica vizual starea generala a calitatii aerului (verde pentru conditii bune, galben pentru conditii moderate si rosu pentru conditii periculoase)

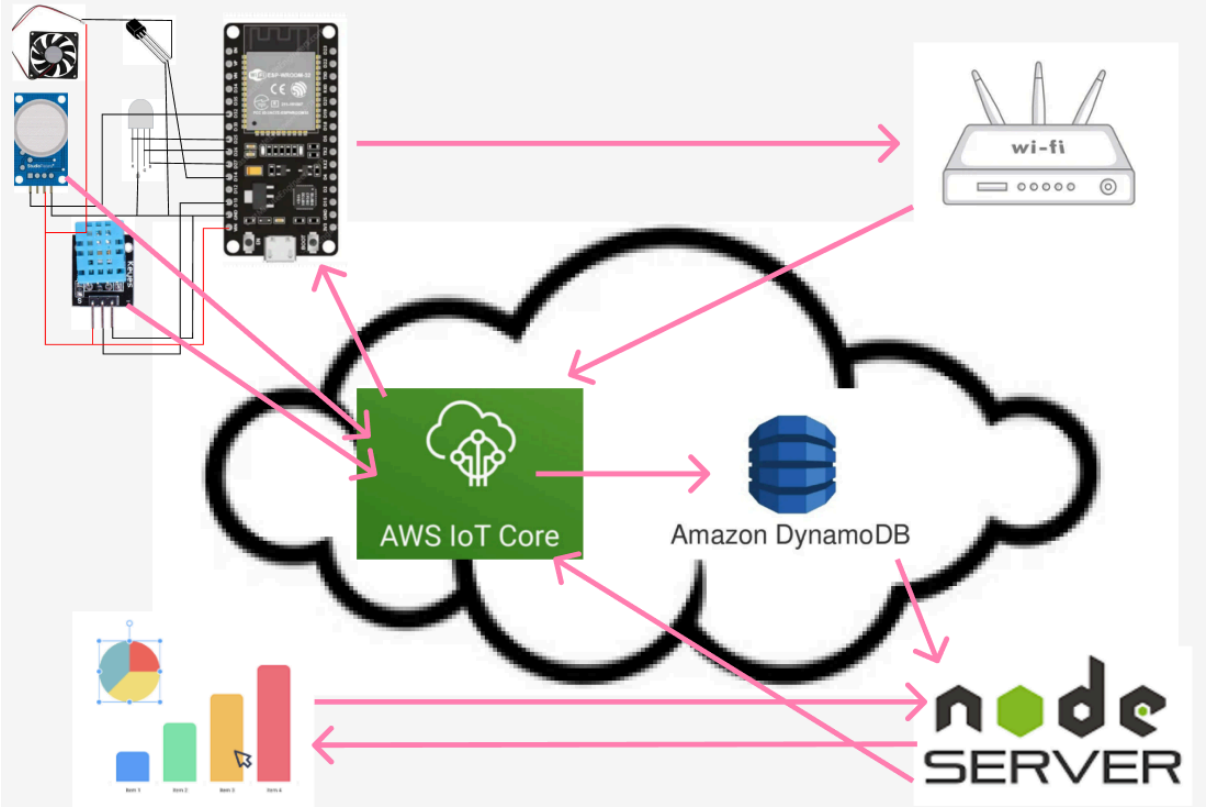
Proiectul are urmatoarele obiective principale:

- Integrarea in infrastructura IoT prin utilizarea tehnologiilor de cloud computing
- Monitorizarea datelor in timp real
    - Colectarea continua a datelor de la senzori si trimiterea lor in mod securizat catre un serviciu cloud (AWS IoT Core) pentru stocare
- Vizualizarea si analiza datelor in timp
    - Crearea unei interfete web care afiseaza informatiile colectate in diverse perioade de timp despre calitatea aerului, utilizand grafice
- Implementarea unui sistem de alerta atunci cand calitatea aerului scade suficient, pentru informare
- Controlul activ al mediului
    - Implementarea unui sistem de tip “turn on/off” al ventilatorului, oferind posibilitatea utilizatorului de a mentine calitatea aerului in parametri acceptabili
- Implementarea unui sistem vizual de alertare printr-un LED RGB

2. Arhitectura

Proiectul se bazeaza pe o arhitectura distribuita, unde componentele fizice (senzorii si actuatorii), microcontrolerul ESP32 si infrastructura cloud lucreaza impreuna pentru a colecta, analiza si afisa date despre calitatea aerului.

Diagrama topologiei retelei



Protocoale de comunicatie utilizate

1. Wi-Fi (802.11)
    - Asigura conectivitatea ESP32 la reteaua locala si permite comunicarea acestuia cu serviciile cloud AWS
    - ESP32 este configurat in modul STA (Station)
    - Rol: transmiterea datelor de la ESP32 catre AWS IoT Core si receptionarea comenzilor de control de la AWS IoT Core
2. MQTT
    - Asigura comunicatia intre ESP32 si AWS IoT Core

- Functioneaza pe portul 8883 pentru conexiuni securizate prin TLS/SSL
- Pentru publicarea datelor, utilizeaza topicurile: sensor/temperature, sensor/humidity, sensor/gas pentru datele de temperatura, umiditate colectate de la DHT11 si cele de concentratia gazelor colectate de la MQ135
- Pentru subscribe, ESP32 primeste comenzile de control ale ventilatorului (on/off) pe topicul commands/fan
- Flux de lucru: ESP32 publica datele colectate de la senzori pe topicurile respective. AWS IoT Core proceseaza mesajele si le stocheaza in DynamoDB. ESP32 se aboneaza la topicul commands/fan pentru a primi comenzi de activare sau dezactivare a ventilatorului

3. HTTP
   - Asigura comunicarea securizata intre aplicatia web si serverul local Node.js
   - Serverul local expune API-uri REST, cu endpoint-uri:
     i. GET /temperature: obtine ultimele 10 valori ale temperaturii
     ii. GET /humidity: obtine ultimele 10 valori ale umiditatii
     iii. GET /gas: obtine ultimele valori ale concentratiei gazelor
     iv. POST /send-command: trimite o comanda on/off catre ventilator

Alte protocoale:

1. NTP (Network Time Protocol)
   - Asigura sincronizarea ceasului intern al ESP32 cu ora globala
   - Asigura timestamp-uri sincronizate pentru datele colectate
   - ESP32 se conecteaza la serverul NTP pool.ntp.org pentru a obtine ora exacta

Descrierea componentelor utilizate

Senzori

1. Senzor DHT11
   - Masoara temperatura si umiditatea aerului
   - Datele sunt colectate cu o precizie de ±2°C in ceea ce priveste temperatura, iar umiditatea relativa cu o precizie de ±5%

- Pinul de date este conectat la D13, alimentare 5V si GND de pe ESP32

2. Senzor MQ135
    - Detecteaza concentratia gazelor (CO2, NH3, benzen) in aer
    - Intervalul de masurare este de 10-1000 ppm, timpul de incalzire este aproximativ 24 de ore pentru calibrare
    - Pinul analogic este conectat la D32, alimentare 5V si GND de pe ESP32

Actuatori

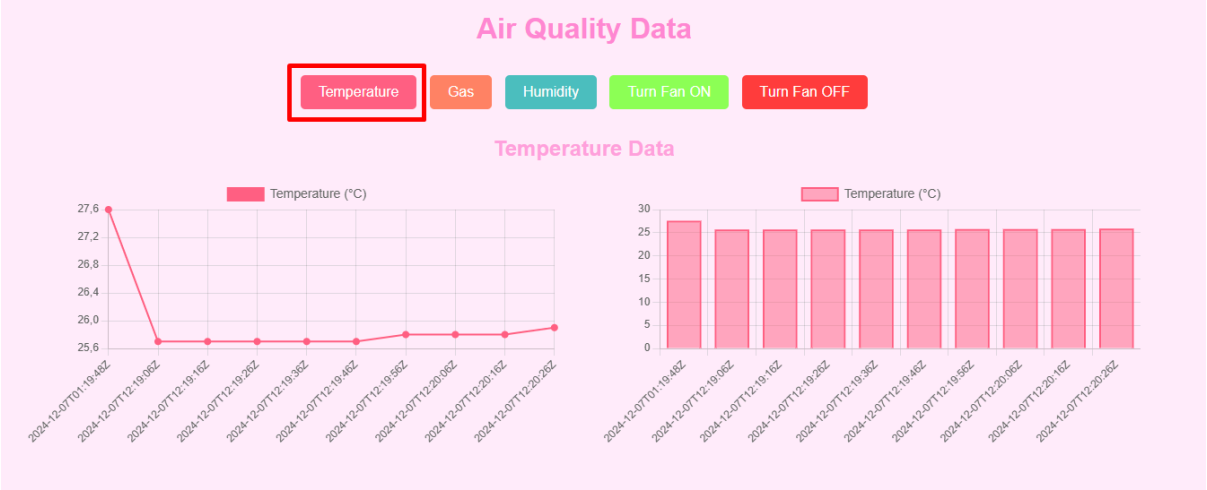
1. LED RGB:
    - Ofera un indicator vizual al calitatii aerului printr-un cod de culori (verde - aer de calitate buna, galben - aer de calitate moderata, rosu - aer de calitate slaba)
    - Canalul rosu este conectat la D25, cel verde la D26 si cel albastru la D27 prin rezistente de 220 ohmi
    - Culoarea este determinata direct in functie de valorile colectate
2. Ventilator
    - Raspunde comenzilor utilizatorului pentru a porni sau opri ventilarea aerului
    - Pinul de control este D14, conectat la baza unui tranzistor printr-o rezistenta de 220 ohmi
    - Alimentarea este de 5V, GND este conectat la colectorul tranzistorului

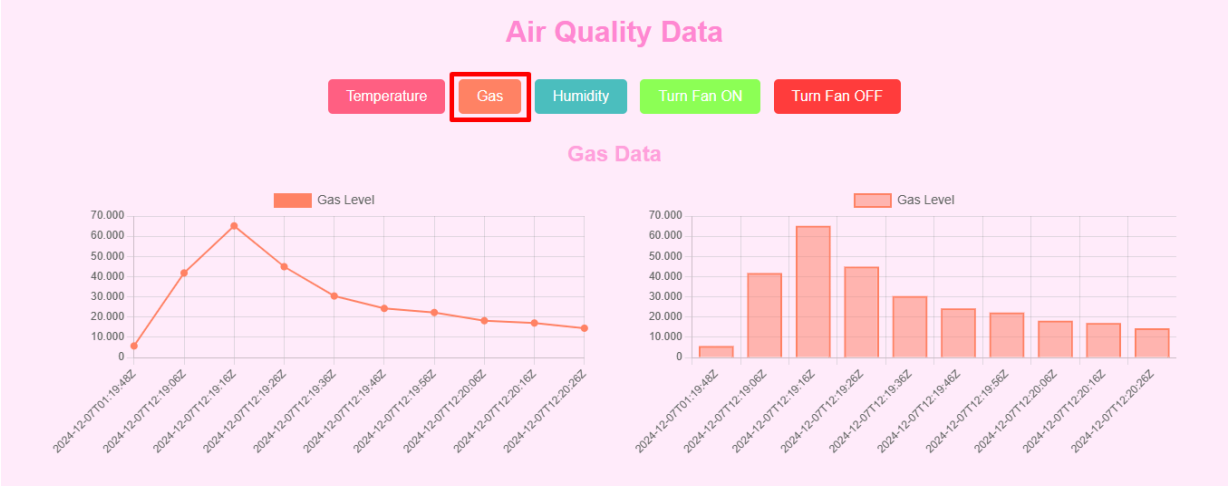
Aplicatia de control

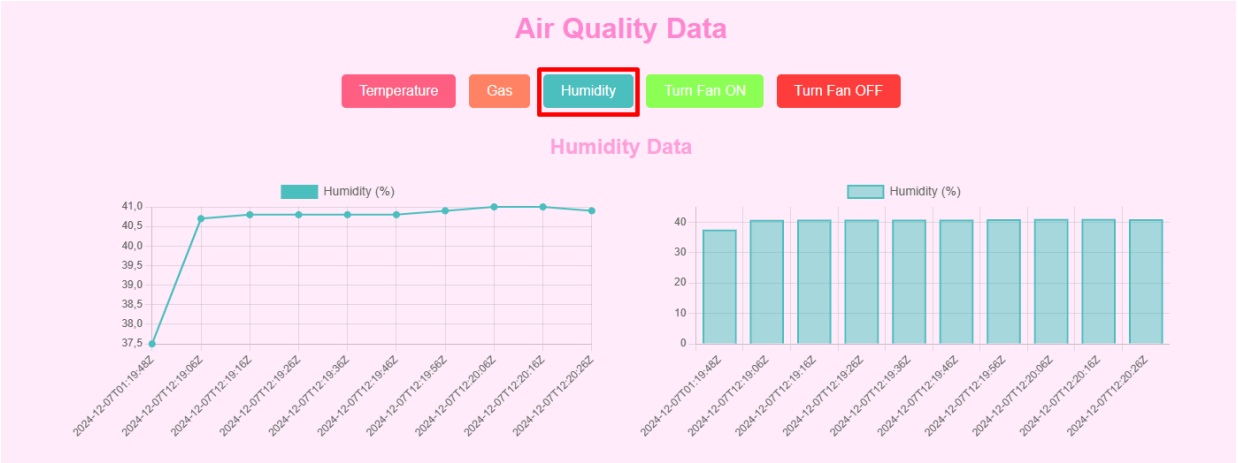
1. Baza de date DynamoDB
    - Stocheaza datele colectate intr-un tabel SensorData, cu urmatoarea structura:
        i. sensorType (partition key), care precizeaza tipul senzorului: temperature, humidity sau gas
        ii. timestamp (sort key), care arata ora la care a fost colectata valoarea
        iii. value, valoarea masurata
2. Server Node.js
    - Interfata intre baza de date si aplicatia web

- Expune API-uri REST pentru vizualizarea datelor si controlul ventilatorului

3. Aplicatia web
   - Utilizatorul poate vizualiza datele in grafice interactive, realizate cu framework-ul Chart.js, putand accesa o sectiune pentru fiecare tip de date: temperatura, umiditate, concentratie de gaze







Utilizatorul poate controla ventilatorul prin apasarea butoanelor “Turn fan ON/OFF”. In acest caz, se trimite comanda on/off la serverul de backend, care o va publica pe topicul commands/fan la care este abonat microcontrolerul.

3. Implementare

a. Configurare hardware:

Componente utilizate:

- Microcontroler ESP32
- Senzor DHT11 pentru masurarea temperaturii si umiditatii
- Senzor MQ135 pentru detectarea gazelor
- LED RGB pentru indicatorul vizual
- Ventilator controlat prin tranzistor

Conectarea senzorilor:

- DHT11:
  - Pinul de date conectat la D13
  - Alimentare 5V si GND conectate la ESP32

- MQ135:
    - Pinul analogic conectat la D32
    - Alimentare 5V si GND conectate la ESP32

Conectarea actuatorilor:

- LED RGB:
    - Pinul rosu la D25, verde la D26, albastru la D27
    - Fiecare pin este conectat printr-o rezistenta de 220 ohmi
- Ventilator:
    - Pinul de control conectat la D14 printr-un tranzistor NPN
    - Alimentarea ventilatorului la 5V, iar GND la colectorul tranzistorului.

Alimentare: ESP32 conectat la o sursa de alimentare USB.

b. Configurare software:

Platforme utilizate:

- Arduino IDE: pentru incarcarea codului pe ESP32
    - Se instaleaza driver-ul CP210x pentru ESP32 https://www.silabs.com/developer-tools/usb-to-uart-bridge-vcp-drivers?tab=downloads
    - Se instaleaza librariile necesare: WiFi, PubSubClient, ArduinoJSON, DHT si MQ135 pentru senzori
    - Se adauga ESP32 boards: File -> Preferences: https://dl.espressif.com/dl/package_esp32_index.json si apoi se instaleaza esp32 din Board Manager
- AWS Management Console: pentru configurarea si gestionarea serviciilor AWS, inclusiv DynamoDB si SNS

Configurarea serviciului cloud AWS:

- IoT Core
    - Se creeaza un Thing (ex: ESP32AirQuality)
    - Se descarca certificatele, cheia privata si CA pentru criptarea datelor
    - Se creeaza topicurile pentru publicarea datelor de la senzori: sensor/temperature, sensor/humidity, sensor/gas si pentru primirea comenzilor de pornire sau oprire a ventilatorului commands/fan si se seteaza regulile din Message Routing, de tipul:

```
SELECT sensorType, value, timestamp FROM 'sensor/gas'
WHERE sensorType = 'gas'
SELECT sensorType, value, timestamp FROM
'sensor/humidity' WHERE sensorType = 'humidity'
SELECT sensorType, value, timestamp FROM
'sensor/temperature' WHERE sensorType = 'temperature'
```

- DynamoDB: pentru stocarea datelor de la senzori
    - Tabelul SensorData cu Partition Key tipul senzorului (temperature/gas/humidity), Sort Key timestamp-ul si valoarea
- IAM (Identity and Access Management):
    - Se creeaza un user (ex: IoTWebApp)
    - Se creeaza politici pentru a permite ESP32 sa acceseze resursele necesare: iot:Connect, iot:Publish, iot:Subscribe, si pentru a permite scrierea datelor colectate in tabelul DynamoDB: dynamodb:PutItem
    - Se creeaza o politica pentru a permite aplicatiei sa trimita notificari prin Amazon SNS, sns:Publish
- SNS (Simple Notification Service)
    - Se creeaza un topic SNS (AirQualityAlerts), se retine ARN-ul (Amazon Resource Name) si este adaugat email-ul ca endpoint

Configurarea pentru server si interfata web:

- Se instaleaza Node.js si npm
- Se instaleaza dependentele npm install aws-sdk express cors pentru interactiunea cu AWS DynamoDB si SNS
- Express.js si CORS: pentru gestionarea cererilor HTTP
- Pentru frontend, framework-ul utilizat este Chart.js pentru reprezentarea grafica a datelor

Organizarea codului:

- AirQualityMonitoring.ino: logica principala a ESP32 care se conecteaza la WiFi, initializeaza senzorii, LED-ul, se sincronizeaza cu NTP si se conecteaza la AWS IoT Core. Citeste datele de la senzori si le publica pe topicurile specifice si le trimite catre cloud. Se actualizeaza starea LED-ului pentru a reflecta calitatea aerului.
- sensors/dht11.cpp si sensors/mq125.cpp: Functii pentru initializarea si citirea datelor de la senzori.

- aws/aws_utils.cpp: Functii de conectare la serviciul cloud: **connectAWS()**, de publicare a datelor prin construirea unui JSON cu “sensorType”, “value” si timestamp: **publishSensorData()**, si de subscribe la topicul commands/fan si in functie de comanda primita in JSON, “on”/”off”, se aprinde/stinge ventilatorul: **callback**.
- aws/aws_certificates.h: contine certificatele pentru conexiune securizata
- web_app/server.js: backend care gestioneaza cererile pentru preluarea datelor din baza de date: GET /temperature, GET/humidity, GET/gas, precum si trimiterea comenzilor catre ESP32 POST/send-command. In plus, la un interval de 10 secunde, verifica daca ultima valoare introdusa in baza de date depaseste pragul de conditii periculoase ale aerului; daca da, va trimite o notificare.
- web_app/index.html: interfata utilizatorului pentru vizualizarea datelor in 3 sectiuni diferite si sub forma de 2 grafice, de tip line si bar: “temperature”, “humidity” si “gas” si pentru controlul ventilatorului prin cele doua butoane: “Turn Fan ON”/”Turn Fan OFF”. CSS si JavaScript sunt integrate pentru design si functionalitate dinamica, pentru a afisa alerta de conditii periculoase a aerului.

Fluxul implementarii:

- ESP32:
  1. Se conecteaza la WiFi si sincronizeaza timpul prin NTP.
  2. Citeste datele de la senzori.
  3. Trimite periodic datele catre AWS IoT Core folosind MQTT.
- Backend:
  1. Interogheaza baza de date DynamoDB pentru cele mai recente 10 valori, luate dupa timestamp.
  2. Trimite notificari prin SNS atunci cand valoarea concentratiei gazului depaseste 1500.
  3. Raspunde cererilor frontend-ului de schimbare a modului ventilatorului (“on”/”off”) si publica comanda respectiva pe topicul “commands/fan”

c. Configurare sistem de alertare si notificare

Sistem de notificare - backend:
Pentru a trimite notificari in cazul in care concentratia gazelor din aer depaseste pragul critic (1500ppm), am utilizat Amazon SNS (Simple Notification Service). Configurarea a presupus:

1. Crearea unui topic SNS
   1.1. Am creat topicul SNS AirQualityAlerts si am copiat ARN-ul corespunzator: arn:aws:sns:eu-central-1:463470938276:AirQualityAlerts
   1.2. A fost configurat endpoint-ul de notificare cu email-ul pentru primirea mesajului.
2. Configurarea politicii IAM pentru SNS:
   2.1. Politica a fost creata pentru a permite sns:Publish asupra topicului SNS.
3. In backend, se verifica periodic (la 10 secunde) datele cele mai recente din tabela SensorData pentru gas si in cazul in care ultima valoare depaseste pragul critic, se trimite notificarea.

Sistem de alertare vizuala - frontend:

1. Se iau datele de la backend
   1.1. Aplicatia web trimite cereri catre endpoint-ul backend /gas pentru a prelua ultimele date ale senzorilor, folosind metoda fetch() din JavaScript, in **checkAlerts()**.
2. Daca valoarea depaseste pragul critic de 1500, se afiseaza un mesaj de alerta sub forma de popup (utilizand un container HTML).
3. Verificarea alertelor se face automat, la fiecare 10 secunde, folosind setInterval().
4. Cand pragul este depasit, se trimite si catre backend comanda de pornire automata a ventilatorului.

4. Vizualizare si Procesare de Date

Datele colectate de la senzori sunt afisate intr-o interfata web, creata folosind HTML, CSS si JavaScript. Aceste date sunt prelucrate si afisate sub forma de

doua grafice, de tip line si de ip bar, create utilizand framework-ul Chart.js. Interfata permite vizualizarea informatiilor despre temperatura, concentratia gazelor din mediu si umiditate, avand acces la ultimele 10 valori colectate si ordonate dupa timestamp.

1. **Structura frontend-ului**
   Interfata este construita folosind:
   - HTML pentru structura, cu sectiuni dedicate fiecarui tip de senzor
   - CSS pentru stilizare, utilizand o schema de culori specifica
   - JavaScript pentru logica de afisare si manipulare a datelor

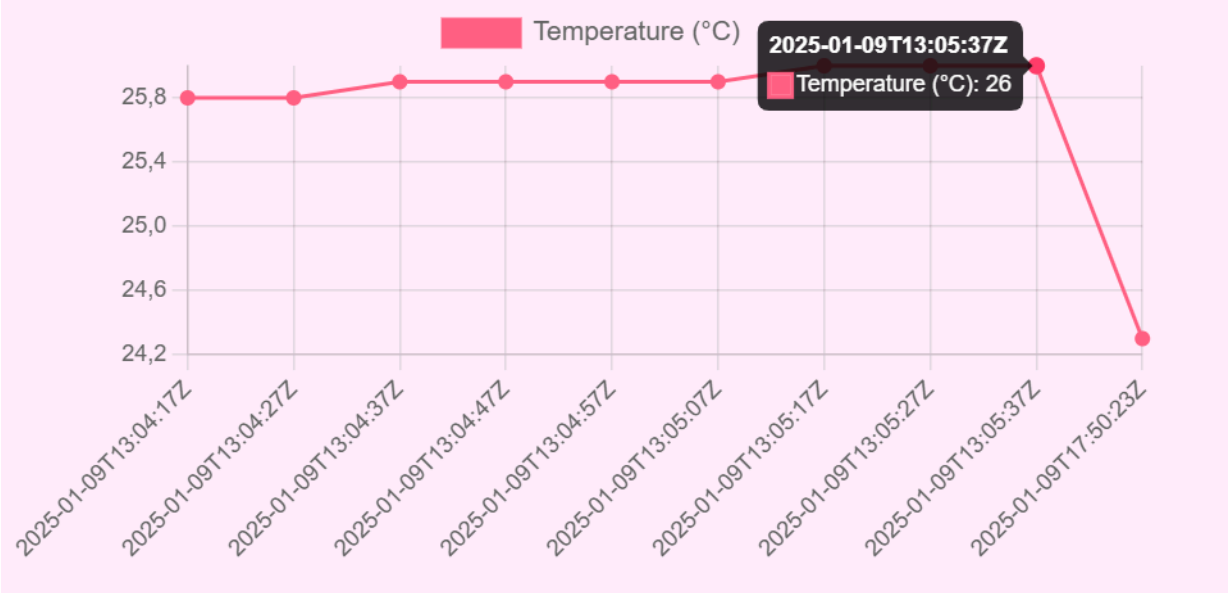
   Elemente principale:
   - Cele trei sectiuni dedicate fiecarui tip de senzor: temperatura, gaz, umiditate
   - Butoane de navigare, care permit trecerea rapida intre sectiuni
   - Butoane de activare/dezactivare a ventilatorului
   - Containerele grafice cu Chart.js, ce contin graficele de tip line si bar pentru reprezentarea datelor

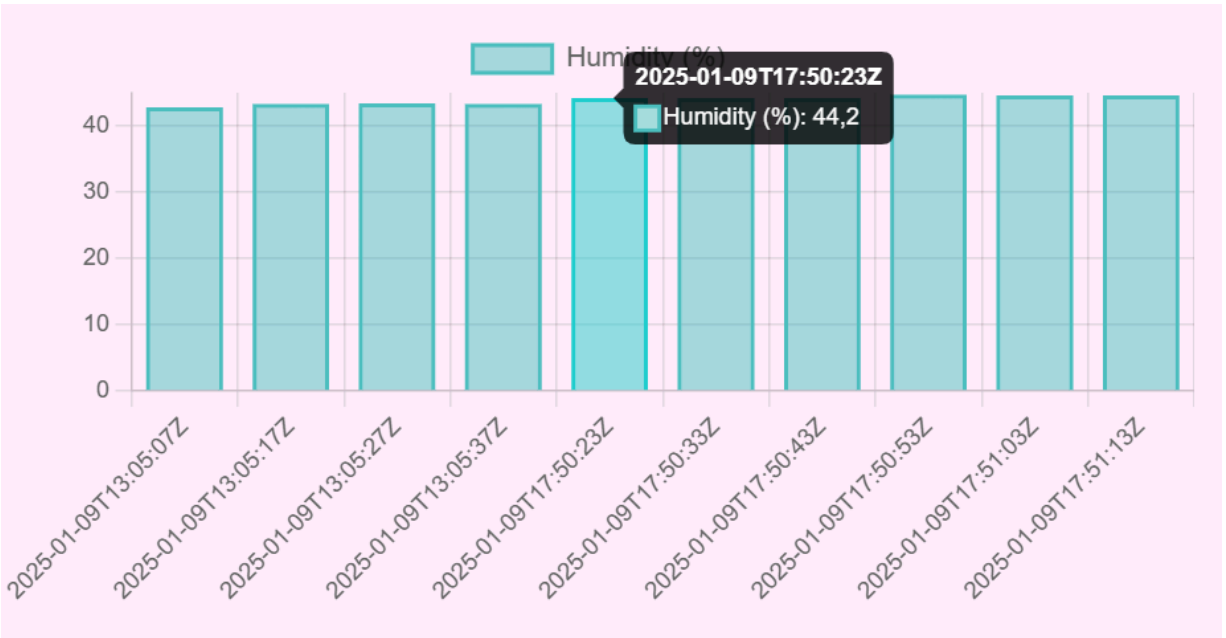
2. **Framework-ul Chart.js**
   Datele sunt afisate in timp real in interfata utilizand Chart.js, care este integrat in aplicatie prin intermediul:
   ```
   <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
   ```
   Aceasta permite accesarea functionalitatilor oferite de Chart.js pentru a crea cele doua tipuri de grafice folosite.

   Pasi:
   - Integrarea in aplicatie.
   - Crearea graficelor cu new Chart(), ce primeste ca parametri elementul canvas unde va fi afisat graficul si configuratia acestuia: tipul, datele si optiunile de personalizare.
   - Configurarea datelor
     - Datele sunt obtine prin functia asincrona fetchData de la endpoint-urile specifice (ex: http://localhost:3000/gas). Dupa preluarea acestora, valorile sunt prelucrate pentru a construi graficul
       - **labels**: timestamp-urile folosite pe axa X
       - **datasets**: valorile senzorilor folosite pe axa Y

- Personalizarea graficelor, specificand backgroundColor, borderColor, legendele, scales si textul
- Actualizarea dinamica a graficelor, utilizand functia destroy() inainte de a genera un grafic nou, pentru a evita suprapunerea acestora

Framework-ul permite, de asemenea, vizualizarea valorilor specifice de la anumite puncte.





3. Butoanele aplicatiei: navigare si control

Interfata include butoane pentru navigarea intre diferite sectiuni (temperatura, umiditate si concentratia gazelor) si controlul ventilatorului.

Aceste butoane sunt configurate pentru a imbunatati experienta utilizatorului si pentru a lega actiunile din interfata cu backend-ul aplicatiei.

Navigarea intre sectiuni:

- Fiecare buton afiseaza functia showSection(), care controleaza vizibilitatea sectiunilor

```
<button class="btn-temperature" onclick="showSection('temperature')">Temperature</button>
```

Aceasta functie ascunde toate sectiunile existente, activeaza doar sectiunea corespunzatoare butonului apasat si afiseaza graficele corespunzatoare utilizand drawChart().

Controlul ventilatorului:

- La apasarea butoanelor, se apeleaza functia de sendCommand('on'/'off'), care face un POST catre backend si utilizeaza fetch() pentru a comunica uc serverul la ruta /send-command.

```
<button class="btn-fan-on" onclick="sendCommand('on')">Turn Fan ON</button>
```

- Serverul backend comunica cu dispozitivul IoT prin MQTT. Se specifica topicul commands/fan la care este abonat dispozitivul, payload-ul este un obiect JSON ce contine comanda ("command":"on"}, si se specifica si qos 0.

```
const params = {
      topic: 'commands/fan',
      payload: JSON.stringify({ command: command }),
      qos: 0,
};
```

- Este folosit SDK-ul AWS pentru a publica mesajul catre AWS IoT Core.

```
await iotData.publish(params).promise();
```

Procesarea datelor - sistemul de alerta

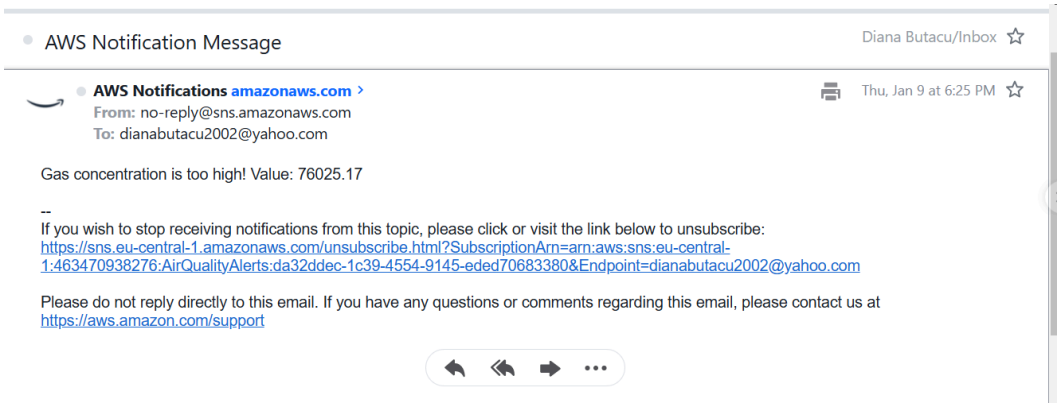
Procesarea datelor are scopul de a analiza in timp real valorile concentratiei de gaze din aer si de a identifica situatia periculoasa in care aceasta valoare a depasit pragul critic, de 1500ppm. In acest caz, in interfata se afiseaza un popup

in partea de sus a ecranului cu un mesaj corespunzator: “Gas concentration is too high! Value: {value}”, si se porneste automat si ventilatorul, iar backend-ul trimite o notificare prin email, utilizand serviciul Amazon SNS.

Fluxul de procesare a datelor:

- Colectarea datelor din baza de date
  - Severul backend acceseaza baza de date AWS DynamoDB SensorData, care stocheaza valorile trimise de catre senzori
  - La fiecare 10 secunde, serverul executa o interogare pentru a obtine ultimele valori inregistrate
- Procesarea datelor pentru alertare
  - Dupa preluarea datelor, se verifica daca ultima valoare trimisa depaseste pragul critic de 1500
- Alertare in frontend si backend daca conditia este indeplinita:
  - **Notificarea utilizatorului prin Amazon SNS in backend**: Un mesaj de alerta este trimis la emailul configurat pe endpoint-ul topic-ului SNS
  - Se utilizeaza ARN-ul topic-ului configurat, AirQuality Alerts

```
const notificationParams = {
    Message: `Gas concentration is too high! Value: ${latestData.value}`,
     TopicArn: 'arn:aws:sns:eu-central-1:463470938276:AirQualityAlerts',
 };
const snsResponse = await sns.publish(notificationParams).promise();
```



- In **frontend, un mesaj de alerta este afisat intr-un popup colorat** deasupra ecranului.
- Alerta vizuala este definita in HTML folosind un container care contine un mesaj pentru textul alertei si un buton de inchidere.

```
    <div id="alert-container" style="position: fixed; top: 0; left: 0; width: 100%; background-color: rgb(255, 0, 170); color: white; display: none; text-align: center; padding: 10px; display: flex; justify-content: space-between; align-items: center; box-sizing: border-box;">
        <span id="alert-message" style="flex-grow: 1; text-align: center;"></span>
        <span class="closebtn" onclick="document.getElementById('alert-container').style.display='none';" style="cursor: pointer; font-weight: bold; margin-left: 10px; padding: 5px;">&times;</span>
    </div>
```

- Utilizatorul poate inchide manual alerta apasand pe simbolul “x”.

Functia de procesare a datelor ruleaza periodic folosind un interval setat la 10 secunde, similar cu intervalul de timp de publicare a datelor de catre ESP32.
In plus, in frontend, la identificarea conditiilor periculoase, se trimite catre backend comanda de activare automata a ventilatorului. Mesajul va fi trimis pe topicul commands/fan unde este abonat ESP32.

5. Securitate

Masuri de autentificare si criptare

1. Autentificare si autorizare in AWS
   Pentru a restrictiona accesul doar la resursele necesare, s-au configurat politici IAM:
   - ESP32 este autorizat sa se conecteze la AWS IoT Core si sa execute urmatoarele actiuni:
     - iot:Connect pentru conectarea la brokerul MQTT
     - iot:Publish si iot:Subscribe pentru publicarea si abonarea la topicuri
   - Serverul are politici care permit accesul la:
     - dynamodb:putItem pentru scrierea datelor colectate de la senzori in tabelul SensorData
     - sns:Publish pentru trimiterea notificarilor prin Amazon SNS

2. Utilizarea certificatelor pentru AWS IoT Core: ESP32 utilizeaza certificate generate la crearea Thing-ului in platforma pentru autentificarea cu brokerul MQTT:
   - Certificat Root CA: pentru a valida conexiunea la endpoint-ul AWS
     ```
     net.setCACert(AWS_CERT_CA);
     ```
   - Certificat dispozitiv: pentru dispozitivul ESP32 pentru a demonstra identitatea acestuia
     ```
     net.setCertificate(AWS_CERT_CRT);
     ```
   - Cheie privata
     ```
     net.setPrivateKey(AWS_CERT_PRIVATE);
     ```

3. Conexiune securizata intre ESP32 si AWS IoT Core
   - Protocolul MQTT peste TLS cu portul 8883, destinat comunicatiilor MQTT criptate
     ```
     client.setServer(AWS_IOT_ENDPOINT, 8883);
     ```

4. Utilizarea bibliotecilor securizate pentru conectivitate
   - **WiFiClientSecure**: pentru a securiza conexiunea ESP32 la Internet prin protocolul TLS.

- **PubSubClient**: utilizata pentru implementarea protocolului MQTT pe dispozitivul ESP32. Integrarea acestei biblioteci cu WiFiClientSecure permite trimitere si primirea mesaje MQTT criptate.

```
WiFiClientSecure net = WiFiClientSecure();
PubSubClient client(net);
```