Tecnicatura Superior en Telecomunicaciones

Espacio Curricular: Programación

DISPOSITIVO IoT PARA CONTROL DE CONCENTRACIÓN DE CO2 EN AMBIENTES CERRADOS

Entrega Nº 3

Alumnos:
Ares, Diego Ezequiel
Gimenez Coria, Fernando
Roldan, Patricio Leandro
Profesor: Lanfranco Lisandro

29/10/2024

Resumen

Este documento describe un sistema de monitoreo ambiental desarrollado para gestionar datos de calidad de aire en tiempo real. El sistema utiliza un microcontrolador ESP32, que captura datos de CO2, temperatura, humedad y nivel de batería, y los envía a un servidor Flask para su procesamiento y almacenamiento en una base de datos MySQL. Además, cuenta con un extractor de aire que se activa automáticamente cuando el nivel de CO2 supera un umbral crítico, proporcionando una respuesta inmediata ante condiciones ambientales adversas.

Índice

SISTEMA DE MONITOREO AMBIENTAL CON ESP32 Y SERVIDOR FLASK

Este sistema de monitoreo ambiental está diseñado para captar y gestionar datos de calidad de aire (CO2, temperatura, humedad y nivel de batería) usando un ESP32. El ESP32 envía datos a un servidor central, que los procesa y los almacena en una base de datos. Además, el sistema incluye un control de ventilación mediante un extractor, el cual se activa automáticamente cuando los niveles de CO2 superan un umbral crítico.

Función

Captura datos de sensores y envía las lecturas al servidor Flask

Configuración

El archivo configura los pines del sensor MQ135 para medir el CO2, el AHT25 para temperatura y humedad, y un pin para monitorear el nivel de batería.

Componentes Principales

Dispositivo ESP32

El ESP32 captura datos ambientales mediante sensores y envía lecturas a través de WiFi a un servidor Flask para su almacenamiento y análisis.

Servidor Flask

El servidor Flask actúa como receptor de datos enviados por el ESP32. Este servidor almacena las lecturas en una base de datos y emite alertas según los valores recibidos.

Base de Datos MySQL

La base de datos almacena las lecturas y alertas generadas por el servidor Flask. Esto permite tener un historial de lecturas y generar reportes en caso necesario.

Descripción de Archivos

Código ESP32

Este archivo configura el ESP32 para:

- Captura de Datos: Usa sensores MQ135 (para CO2), AHT25 (para temperatura y humedad) y una entrada analógica para leer el nivel de batería.
- Control del Extractor: Cuando los niveles de CO2 son críticos, activa el extractor de aire.
- Envía los Datos: Cada 60 segundos, envía los datos capturados al servidor Flask

Servidor Flask

Este archivo configura el servidor Flask para escuchar en el puerto 5000 y gestionar los datos que el ESP32 envía.

Server.py

```
from server import app
def iniciar_servidor():
 app.run(debug=True, host='0.0.0.0', port=5000)
if __name__ == "__main__":
 iniciar_servidor()
```

Conexión a la Base de Datos

Este archivo establece la conexión a la base de datos MySQL y define métodos para insertar lecturas, alertas y estados de batería.

API de Lecturas

Define el endpoint /api/lecturas en Flask, que recibe los datos de sensores y los valida antes
de insertarlos en la base de datos.

Instalación y Configuración

Requisitos

- ESP32
- Servidor Flask con Python 3.8+
- Base de Datos MySQL
- Librerías Python: Flask, pymysql

Configuración

- Configurar Credenciales WiFi en el ESP32.
- Configurar la Base de Datos MySQL y crear tablas necesarias.
- Configurar el Servidor Flask y asegurar que el puerto esté abierto para conexiones
- externas.

Ejecución

- Ejecutar el Servidor Flask usando python server.py.
- Subir el código al ESP32.
- Verificar la Base de Datos para revisar las lecturas y alertas generadas.

Resultados Obtenidos

1. **Captura y Envío de Datos**: Se implementó con éxito la captura de datos de CO2, temperatura, humedad y nivel de batería mediante el ESP32, enviándo estos datos al servidor Flask.
2. **Activación de Extractor de Aire**: Se comprobó que el extractor de aire se activa correctamente cuando los niveles de CO2 superan el umbral de ppm, garantizando así una respuesta inmediata a condiciones de aire peligrosas.
3. **Almacenamiento de Datos y Alertas**: Los datos de cada sensor se registran en la base de datos MySQL junto con alertas, lo que permite un monitoreo continuo y acceso al historial de mediciones.
4. **Alertas de Nivel de Batería**: Se detecta y almacena una alerta cuando el nivel de batería baja de 3.5 V, permitiendo así la anticipación de recargas o cambios de batería para asegurar la operatividad del dispositivo.

Conclusión

El sistema de monitoreo ambiental basado en ESP32 y servidor Flask implementado ha logrado captar, gestionar y almacenar de manera efectiva datos de CO2, temperatura, humedad y nivel de batería en tiempo real. Al enviar estos datos a un servidor Flask central, el sistema permite un análisis detallado y la activación automática de un extractor de aire cuando los niveles de CO2 superan un umbral crítico, lo cual aporta una solución eficiente para prevenir condiciones peligrosas en espacios cerrados. La integración con una base de datos MySQL asegura un historial robusto para el análisis de tendencias y la generación de reportes.

ANEXO 1 Codigo esp32

main.cpp

```
#include <Arduino.h>
#include <Wire.h>
#include "MQ135Sensor.h"
#include "AHT25Sensor.h"
#include "SensorDataAPI.h"
#include "Extractor.h"
#include "time.h"
// Configuración de Pines
#define MQ135_PIN 36
#define BATERIA 34
#define EXTRACTOR 12
const char* ssid = "Fibertel WiFi032 2.4GHz";
const char* password = "vuelalto67";
const char* apiEndpoint = "http://localhost:5000/api/lecturas";
const char* apiKey = "adamantium";
const int id_disp = 2;
// Configuración de IP
IPAddress ip(192, 168, 0, 100);
IPAddress gateway(192, 168, 0, 1);
IPAddress subnet(255, 255, 255, 0);
// Inicialización de Sensores y Periféricos
MQ135Sensor mq135(MQ135_PIN);
AHT25Sensor aht25;
Extractor extractor(EXTRACTOR);
int triggerAPI = 0;
SensorDataAPI sensorAPI(ssid, password, apiEndpoint, apiKey, extractor, ip,
gateway,
subnet);
void setup() {
 Serial.begin(115200);
 sensorAPI.conectarWiFi();
 Wire.begin();
 mq135.begin();
 aht25.begin();
}
```

```
void loop() {
 float mq135Data = mq135.readFilteredData();
 float ahtTemp, ahtHumidity;
 aht25.readData(ahtTemp, ahtHumidity);
 float nivel_bat = (analogRead(BATERIA) / 4095.0) * 3.3 * 2;
 if (triggerAPI == 60){
 triggerAPI = 0;
 sensorAPI.enviarLecturas(ahtTemp, ahtHumidity, mq135Data, nivel_bat, id_disp);
 }
 sensorAPI.manejarComandosExtractor();
 delay(1000);
 triggerAPI++;
}
```

ANEXO 2 Conexión a la Base de Datos

metodos_db.py

```
import pymysql

class ConexionBD:
 def __init__(self, host, usuario, contraseña, base_datos):
        self.host = host
        self.usuario = usuario
        self.contraseña = contraseña
        self.base_datos = base_datos
        self.conexion = None
        self.cursor = None
 def conectar(self):
 try:
 self.conexion = pymysql.connect(
 host=self.host,
 user=self.usuario,
 password=self.contraseña,
 database=self.base_datos
 )
 self.cursor = self.conexion.cursor()
 print("Conexión a la base de datos exitosa")
 except Exception as e:
 print(f"Error al conectar a la base de datos: {e}")
 def insertar_lectura(self, dispositivo_id, co2_ppm, temperatura, humedad, nivel_bateria):
 query = f"""
 INSERT INTO lecturas (dispositivo_id, co2_ppm, temperatura, humedad,
nivel_bateria)
 VALUES ({dispositivo_id}, {co2_ppm}, {temperatura}, {humedad},
{nivel_bateria});
 """
 try:
 self.cursor.execute(query)
 self.conexion.commit()
 print("Lectura insertada correctamente")
 except Exception as e:
```

```
print(f"Error al insertar la lectura: {e}")
def insertar_alerta(self, dispositivo_id, tipo_alerta, nivel_co2, descripcion):
query = f"""
INSERT INTO alertas (dispositivo_id, tipo_alerta, nivel_co2, descripcion)
VALUES ({dispositivo_id}, '{tipo_alerta}', {nivel_co2}, '{descripcion}');
"""
try:
self.cursor.execute(query)
self.conexion.commit()
print("Alerta insertada correctamente")
except Exception as e:
print(f"Error al insertar la alerta: {e}")
```

ANEXO 3 - API DE LECTURA

api_lecturas.py

```
from flask import Flask, request, jsonify
from metodos_db import ConexionBD
app = Flask(__name__)
bd = ConexionBD('localhost', 'root', 'root', 'control_ambiental')
bd.conectar()
@app.route('/api/lecturas', methods=['POST'])
def recibir_lecturas():
 clave_api = request.headers.get('X-API-KEY')
 if clave_api not in ['silver', 'gold', 'adamantium']:
 return jsonify({'error': 'Clave API no válida'}), 401
 data = request.json
 dispositivo_id = data.get('dispositivo_id')
 co2_ppm = data.get('co2_ppm')
 temperatura = data.get('temperatura')
 humedad = data.get('humedad')
 nivel_bateria = data.get('nivel_bateria')
 if None in [dispositivo_id, co2_ppm, temperatura, humedad, nivel_bateria]:
 return jsonify({'error': 'Datos incompletos'}), 400
 try:
 bd.insertar_lectura(dispositivo_id, co2_ppm, temperatura, humedad, nivel_bateria)
 if co2_ppm > 1000:
 bd.insertar_alerta(dispositivo_id, "Encendido extractor", co2_ppm, "CO2 excede
los 1000 ppm")
 else:
 bd.insertar_alerta(dispositivo_id, "Apagado extractor", co2_ppm, "CO2 en nivel
aceptable")
 if nivel_bateria < 3.5:
 bd.insertar_alerta(dispositivo_id, "Batería baja", nivel_bateria, "Nivel de batería
bajo")
 bd.insertar_estado_bateria(dispositivo_id, nivel_bateria)
 return jsonify({'status': 'Datos recibidos e insertados correctamente'}), 200
 except Exception as e:
 return jsonify({'error': str(e)}), 500
```