SISTEMA DE VENTILACIÓN POR DETECCIÓN DE C02

Memoria Proyecto DAD

6 DE JUNIO DE 2024

DESARROLLO DE APLICACIONES DISTRIBUIDAS

Jesús Carrascosa Carro

Introducción:

Las personas generan C02 como subproducto de la respiración de formas más o menos constante.

En lugares cerrados donde no suele haber una buena ventilación, el C02 tiende a acumularse. Esto junto a la facilidad de obtener las mediciones de C02 hace que sea uno de los indicadores más accesibles para la calidad de la ventilación.

Usos

Generalmente, los campos a los que se puede aplicar son tres:

- Prevención de enfermedades propagadas por el aire, como puede ser el covid19.
- Debido a que la mala ventilación perjudica a la concentración, este se puede aplicar en las aulas de los colegios y universidades, para garantizar un correcto rendimiento.
- Reducir los malos olores.

Componentes

Componentes Hardware:

- Dos ESP-WROOM-32 NODE MCU.
- SLIMBOOK ProX15 como servidor.
- Un sensor de calidad del aire MQ-135.
- Un Relé KY-019.
- Un buen puñado de cables DuPont.

Componentes Software

- Vert.X
- MQTT (utilizando a Mosquitto como bróker) y MQTT explorer.
- API REST.
- MySQL y Heidi SQL para la programación de la base de datos.
- Eclipse Enterprise IDE para la programación del backend.
- Platform.io para la programación de los ESP32.

Funcionamiento

Funcionamiento General.

El funcionamiento general, es el siguiente:

1. Los sensores de forma periódica envían mediciones al servidor.
2. El servidor comprueba al recibir los valores que las últimas mediciones de los sensores de un mismo grupo estén por debajo de un umbral
    a. Si lo supera, activa todos los relés(ventiladores) de todos los actuadores pertenecientes a un mismo grupo.
    b. En caso contrario, manda la orden de apagado a los actuadores.

Funcionamiento específico

Backend:

Aquí hemos definido la lógica de negoció mediante las API REST y MQTT , empleando Vert.X para garantizar la escalabilidad.

En el backend podemos modificar las siguientes cosas:

- El umbral de activación.
- Los puertos donde se ejecutan la aplicación web, la base de datos y el bróker MQTT.
- Las propias API REST.

Las API REST programadas (se acceden a ellas utilizando PostMan) son:

- `router.post("/api/sensor").handler(this::setSensor);:`
  Recibe un JSON con los datos de la medición y lo coloca en la base de datos. Además, comprueba si todas las últimas mediciones del los sensores de ese mismo grupo(groupID) son mayores o iguales que el umbral a través de la función checkCondition(GroupID). En caso afirmativo, envía un uno(encendido) mediante MQTT a todos los actuadores que tengan ese mismo groupID. En caso contrario envía un cero(apagado).
- `router.get("/api/sensor/:placaId/:id").handler(this::getSensor);`
  Obtiene la última medición de un sensor dado. Teóricamente los sensores y actuadores se identifican por el par Sensor /actuador id y el placa id, sin embargo, el placaId lo consideramos innecesario, por motivos que se detallarán más adelante.
- `router.get("/api/actuador/:placaId/:id").handler(this::getActuador);`
  Idem, pero con los actuadores.
- `router.post("/api/actuador").handler(this::setActuador);`
  Añade el estado de un actuador a la base de datos.

- `router.get("/api/lastactuadorGroupId/:groupId").handler(this::getLastActuadorGroupId);`
  Devuelve los últimos estados de todos los actuadores pertenecientes a un mismo groupID.
- `router.get("/api/lastsensorGroupId/:groupId").handler(this::getLastSensorGroupId);`
  Ídem, pero aplicado a las mediciones de los sensores.
- `router.get("/api/allactuadorGroupId/:groupId").handler(this::getAllActuadorGroupID);`
  Devuelve todos los estados de todos los actuadores.
- `router.get("/api/allsensorGroupId/:groupId").handler(this::getAllSensorGroupID);`
  Ídem, pero con los sensores.
- `router.get("/api/allactuador/:placaId/:id").handler(this::getAllActuador);`
  Devuelve todos los estados de un actuador.
- `router.get("/api/allsensor/:placaId/:id").handler(this::getAllSensor);`
  Íde, pero con los sensores.
- `router.get("/api/sensores/:placaId").handler(this::getAllSensores);`
  OBSOLETA Devuelve todas las mediciones de todos los sensores de una placaID.
- `router.get("/api/actuadores/:placaId").handler(this::getAllActuadores);`
  Igual, pero con actuadores.

En cuanto a la base de datos el esquema que se ha utilizado es el siguiente:

```
DROP DATABASE IF EXISTS Proyecto_DAD;
CREATE DATABASE IF NOT EXISTS Proyecto_DAD;
USE Proyecto_DAD;

DROP TABLE IF EXISTS  mediciones;
DROP TABLE  IF EXISTS actuadores;
DROP TABLE IF EXISTS placas;

CREATE TABLE  `placas`(
`placaIdDB`  INT NOT NULL AUTO_INCREMENT,
`placaId`  INT NOT NULL,
PRIMARY KEY (`placaIdDB`));
CREATE INDEX `placaId_idx` ON `placas` (`placaId
CREATE TABLE  `mediciones`(
`valueId` INT NOT NULL AUTO_INCREMEnT,
`medicionId`  INT NOT NULL,
`placaId`  INT NOT NULL,
`concentracion`  DOUBLE NOT NULL,
 `fecha` BIGINT  NOT NULL,
`groupId` INT NOT NULL,
PRIMARY KEY(`valueId`),
FOREIGN KEY(`placaId`)REFERENCES `placas`(`placaId`));

CREATE TABLE  `actuadores`(
`statusId`  INT NOT NULL AUTO_INCREMENT,
```

```
`actuadorId`  INT NOT NULL ,
`placaId`  INT NOT NULL,
`statusValue`  BOOL NOT NULL,
`fecha` BIGINT  NOT NULL,
`groupId` INT NOT NULL,
PRIMARY KEY(`statusId`),
FOREIGN KEY(`placaId`)REFERENCES `placas`(`placaId`) );

INSERT INTO placas(placaId) VALUES (0),(1),(2);
-- INSERT INTO Proyecto_DAD.actuadores(actuadorId, placaId,
statusValue, fecha, groupId) VALUES
-- (0,0,true,UNIX_TIMESTAMP(NOW()),0),
-- (0,0,true,UNIX_TIMESTAMP(NOW())+1,0),
-- (1,1,true,UNIX_TIMESTAMP(NOW()),1),
-- (1,1,true,UNIX_TIMESTAMP(NOW())+1,1),
-- (2,2,true,UNIX_TIMESTAMP(NOW()),2),
-- (2,2,true,UNIX_TIMESTAMP(NOW())+1,2);
-- INSERT INTO Proyecto_DAD.mediciones(medicionId, placaId,
concentracion, fecha, groupId) VALUES
-- (0,0,FLOOR(RAND() * 1000),UNIX_TIMESTAMP(NOW()),0),
-- (0,0,FLOOR(RAND() * 1000),UNIX_TIMESTAMP(NOW())+1,0),
-- (0,0,FLOOR(RAND() * 1000),UNIX_TIMESTAMP(NOW())+3,0),
-- (0,0,FLOOR(RAND() * 1000),UNIX_TIMESTAMP(NOW())+50,0),
-- (20,0,FLOOR(RAND() * 1000),UNIX_TIMESTAMP(NOW())+70,0),
-- (20,0,FLOOR(RAND() * 1000),UNIX_TIMESTAMP(NOW()),0),
-- (1,1,FLOOR(RAND() * 1000),UNIX_TIMESTAMP(NOW()),1),
-- (1,1,FLOOR(RAND() * 1000),UNIX_TIMESTAMP(NOW())+1,1),
-- (2,2,FLOOR(RAND() * 1000),UNIX_TIMESTAMP(NOW()),2),
-- (2,2,FLOOR(RAND() * 1000),UNIX_TIMESTAMP(NOW())+1,2);

DELIMITER //

CREATE TRIGGER AddPlacaOnMeasurementInsertBefore
BEFORE INSERT ON mediciones
FOR EACH ROW
BEGIN
  IF NOT EXISTS (SELECT * FROM placas WHERE placaId = NEW.placaId)
THEN
    INSERT INTO placas (placaId) VALUES (NEW.placaId);
  END IF;
END //

DELIMITER ;
DELIMITER //
CREATE TRIGGER AddPlacaOnActuadorInsertBefore
BEFORE INSERT ON actuadores
FOR EACH ROW
BEGIN
  IF NOT EXISTS (SELECT * FROM placas WHERE placaId = NEW.placaId)
THEN
    INSERT INTO placas (placaId) VALUES (NEW.placaId);
  END IF;
END //

DELIMITER ;
```

La razón por la cual se ha mantenido la columna placaID, es por continuidad con las anteriores entregas, donde se identificaba a un sensor o actuador específico por el par sensor/actuador ID y placa ID, sin embargo, como cada placa tiene un sensor o actuador único, esto ha hecho que placaID sea funcionalmente “irrelevante”. es decir, no tiene ninguna utilidad real. Por esta razón se añade automáticamente a la base de datos al añadir un estado o medición y en la implementación hardware, toma el mismo valor que el id de actuador o sensor.

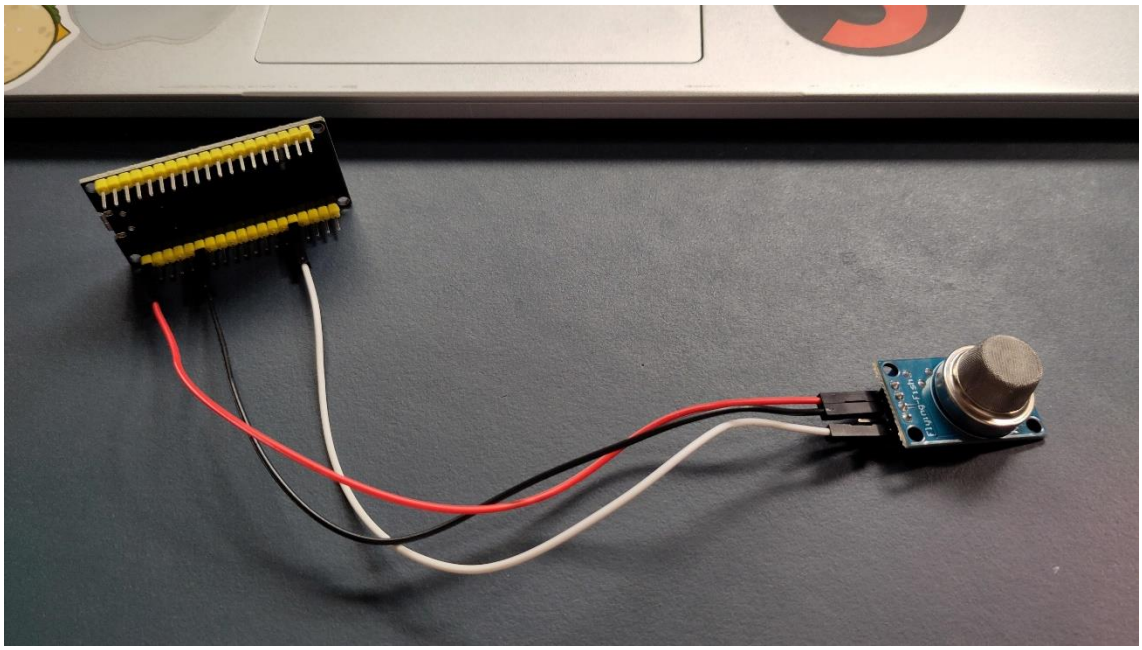
Todos los sensores de un mismo group ID afectan a todos los actuadores que posean el mismo groupID. Es decir, el groupID conecta o enlaza a los actuadores y a los sensores.

Hardware:

Se divide en dos partes:

Sensor:

Envía cada x tiempo (cinco segundos en nuestro caso) un post al servidor de backend que contienen la concentración medida en ese instante.



Como se puede observar, hemos conectado directamente la salida analógica del MQ135 (5 Voltios) a la entrada del esp32 (3.3 Voltios). En teoría, primero se debería hacer una conversión de las tensiones mediante un circuito divisor de tensión, pero al no disponer de las resistencias necesarias, se ha optado a conectarlo directamente, por lo que los valores medidos pueden no corresponderse a la realidad.

Actuador:

Cada vez que recibe un evento MQTT(enviado por el servidor) cambia su estado y envía su estado al servidor mediante un post.