Grupo 4 - IoT

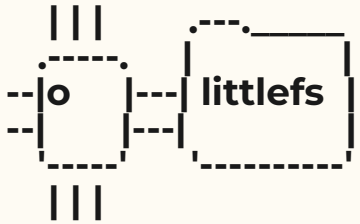
Sistema IoT para Monitoreo y Control de Contenedores Refrigerados (SIMCCR)

Cano Carbajo, Yeyson Samir
Correa De la Cruz, Ysaac Noe
De la Cruz Torres, Diego Alexander
Ramirez Gomero Bryan Anthony
Torre Presentación, Juan Alexis
Villar Arias, Angelo André

LittleFS

¿Por qué LittleFS?

- Robusto ante fallos de energía.
- Manejo eficiente de archivos con retención de 30 minutos.
- Soporte nativo en ESP-IDF y Arduino para ESP32.



Almacenamiento Local

- Se almacenan mediciones en /data.txt.
- Se eliminan registros mayores a 30 minutos.
- Se mantiene solo la data reciente para sincronización.

Sincronización con AWS IoT

- Si hay conexión, se envían datos en JSON.
- Si falla, los datos se almacenan y reintentan en la reconexión.

Flujo de Operación

1. Sensor registra temperatura y CO_2.
2. Se almacena en LittleFS.
3. Se intenta publicar en AWS IoT.
4. Si falla, se guarda localmente.
5. Al restablecer conexión, se reenvían los datos.

Sprint 1

Objetivo General:

- Establecer la infraestructura básica del sistema de monitoreo ambiental en ESP32.

Objetivos Específicos:

1. **Implementación de sensores:**
   - DS18B20 (Temperatura).
   - MQ-135 (CO_2).
2. **Almacenamiento de datos:**
   - Uso de LittleFS.
   - Retención de datos por 30 minutos.
3. **Conectividad con AWS IoT Core:**
   - Enviar datos en tiempo real.
   - Sincronización offline para evitar pérdida de información.

Resultados de Pruebas

1. Sensor de Temperatura DS18B20

- Conectado al **GPIO 4** del ESP32.
- Mediciones precisas y estabilidad en registros.
- Se activa **ALERT_TEMP** cuando supera los **8°C**.

2. Sensor MQ-135 (CO2)

- Conectado al **GPIO 34**.
- Umbral de **10,000 ppm** para activar alertas.
- **ALERT_CO2** y **ALERT_BOTH** cuando se superan límites.

3. Almacenamiento con LittleFS

- Uso de archivos CSV.
- Mecanismo de retención de **30 minutos**.
- Verificación de persistencia tras reinicios.

4. Sincronización con AWS

- Enlace con AWS IoT Core.
- Envió de datos automático.
- Reenvío de datos guardados cuando se restablece la conexión.

Sprint 2

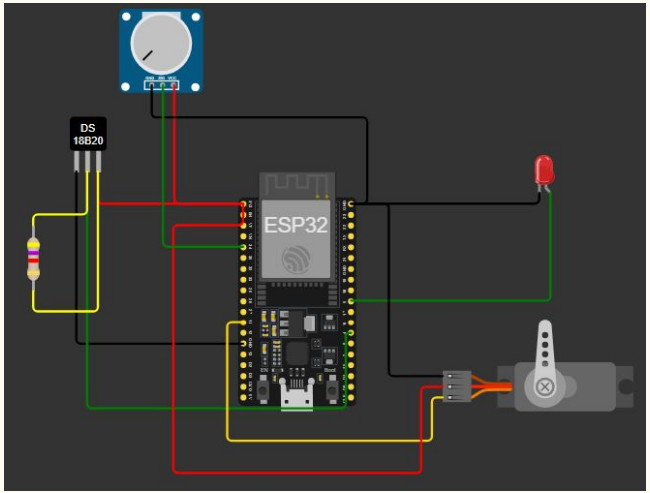
Objetivos:

1. Implementar y configurar los actuadores
2. Integrar la lógica de control basada en sensores
3. Optimizar el almacenamiento de eventos
4. Realizar pruebas de funcionamiento y validación
5. Documentar y entregar un sistema funcional

Sprint 2

Entregables al Final del Sprint

1. **Actuadores funcionando según valores de los sensores:**
   - Ventilador encendiéndose cuando la temperatura sube.
   - Servo abriendo/cerrando compuerta según CO_2.
2. **Código completamente funcional con la lógica de control.**
3. **Datos de activación correctamente registrados en el almacenamiento.**
4. **Pruebas documentadas y verificadas.**



Sprint 2

Resultados de las pruebas:

- **Control del Ventilador**: Se configuró un LED como indicador del ventilador, que se activa si la temperatura supera los 8 ° C.
  - **Resultado:** Funciona correctamente, encendiéndose y apagándose según el umbral definido.
- **Control del Servo (Compuerta)**: Se programó un servo para abrir la compuerta a 90° si el CO_2 supera los 10,000 ppm, y cerrarla a 0° si es menor.
  - **Resultado:** El servo responde adecuadamente a los niveles de CO_2 simulados.
- **Lógica de Activación por Umbrales**: Se establecieron los valores límite para activar ventilador y compuerta según temperatura y CO_2.
  - **Resultado:** Los actuadores reaccionan de forma precisa al superar los umbrales.
- **Integración Sensores-Actuadores**: Se conectaron las lecturas de temperatura y CO_2 con la activación automática de los actuadores.
  - **Resultado:** El sistema toma decisiones en tiempo real y acciona correctamente.
- **Registro de Eventos**: Se implementó el registro del estado de los actuadores y su publicación en AWS IoT.
  - **Resultado:** Los estados se reportan sin errores y se incluyen en los mensajes enviados.
- **Pruebas de Simulación**: Se simularon diversas condiciones ambientales para validar la lógica de control.
  - **Resultado:** Las pruebas fueron exitosas, confirmando el funcionamiento esperado.
- **Validación del Almacenamiento**: Se verificó la integridad y persistencia de los registros del ventilador y la compuerta.
  - **Resultado:** Los datos se almacenan correctamente y sin inconsistencias.

AWS

DynamoDB

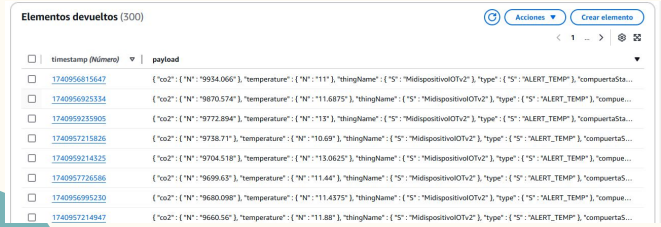
Almacenamiento de datos en dynamoDB

Justificación de la Elección

- NoSQL de alta disponibilidad
- Integración nativa con AWS IoT Core
- Bajo mantenimiento

AWS

Almacenamiento de datos en dynamoDB



Estructura de la tabla

- Clave de partición: thingName (String)
- Clave de ordenación: timestamp (Number)

Atributos adicionales

- temperature (Number)
- co2 (Number)
- type (String) – DATA, ALERT_TEMP, ALERT_CO2 o ALERT_BOTH
- fanState (Boolean)
- compuertaState (Boolean)

AWS

Almacenamiento de datos en dynamoDB

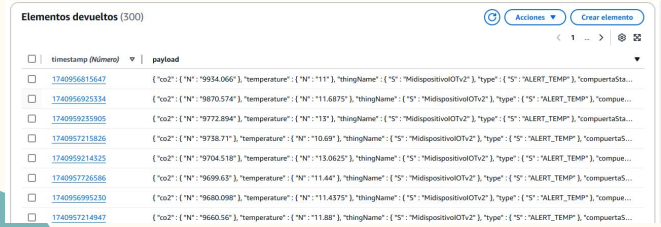
Integración con AWS IoT Core

- Publicación: tópico Salida/01
- Regla (Rule):

  *SELECT * FROM 'Salida/01'*
- Inserción: thingName como clave de partición y timestamp como clave de ordenación

AWS

Almacenamiento de datos en dynamoDB



Integración con AWS IoT Core

- Publicación: tópico Salida/01
- Regla (Rule):
  SELECT * FROM 'Salida/01'
- Inserción: thingName como clave de partición y timestamp como clave de ordenación

AWS

```python
import boto3
import json
import decimal
import time

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('SensorData')
```

Inicialización de DynamoDB

```python
def convert_decimals(obj):
    """Convierte Decimal a int o float para evitar errores de serialización."""
    if isinstance(obj, list):
        return [convert_decimals(x) for x in obj]
    elif isinstance(obj, dict):
        return {k: convert_decimals(v) for k, v in obj.items()}
    elif isinstance(obj, decimal.Decimal):
        return float(obj) if '.' in str(obj) else int(obj)
    else:
        return obj
```

Función para convertir valores Decimal

AWS

```python
def lambda_handler(event, context):
    # Obtener el número de días desde los parámetros de la URL
    query_params = event.get("queryStringParameters", {})
    dias = int(query_params.get("dias", 7))  # Por defecto, 7 días

    # Calcular el timestamp límite
    timestamp_limite = int(time.time() * 1000) - (dias * 24 * 60 * 60 * 1000)
```

Manejo del evento y cálculo del rango de tiempo

AWS

Consulta a DynamoDB

AWS

Filtrado de datos por rango de tiempo

AWS

```
clean_data = []
for item in filtered_data:
    if 'payload' in item and isinstance(item['payload'], dict):
        new_item = item['payload'].copy()  # Copiar el contenido de payload
        clean_data.append(new_item)  # Agregar a la lista sin payload
```

Estructuración de los datos eliminando el campo payload

AWS

Ordenamiento de Datos

AWS

Retorno de Datos

Dashboard: Métricas hechas en Frontend

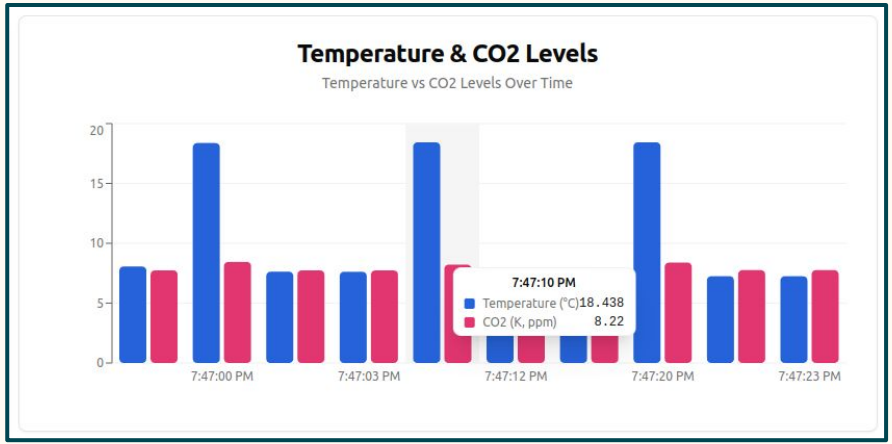
SensorCard

Este componente tiene como funcionalidad permitir visualizar la información específica de un sensor dentro de una tarjeta. Recibe datos provenientes de la API de AWS DynamoDB, tales como las mediciones de temperatura y concentración de CO2.

Dashboard: Métricas hechas en Frontend

Gráficas de Barras

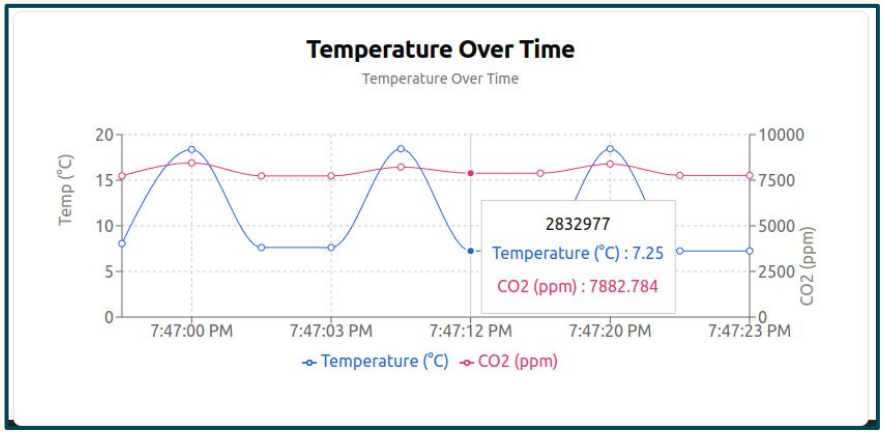
Representa los valores de las mediciones en formato de barras, permitiendo visualizar comparaciones entre diferentes registros. Este gráfico tiene la capacidad de clasificar los valores de CO2 en unidades por millón (ppm)



Dashboard: Métricas hechas en Frontend

Gráfico Lineal

Se muestra la evolución de las mediciones a lo largo del tiempo, facilitando el análisis de tendencias. En este gráfico, la clasificaciones de mediciones se realiza exclusivamente en parte por millón (ppm)



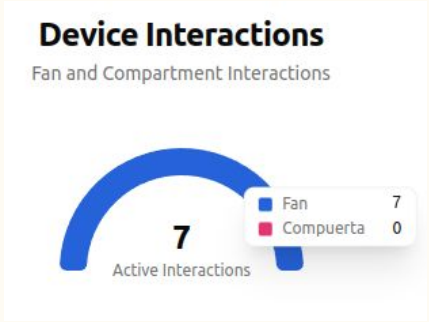
Dashboard: Métricas hechas en Frontend

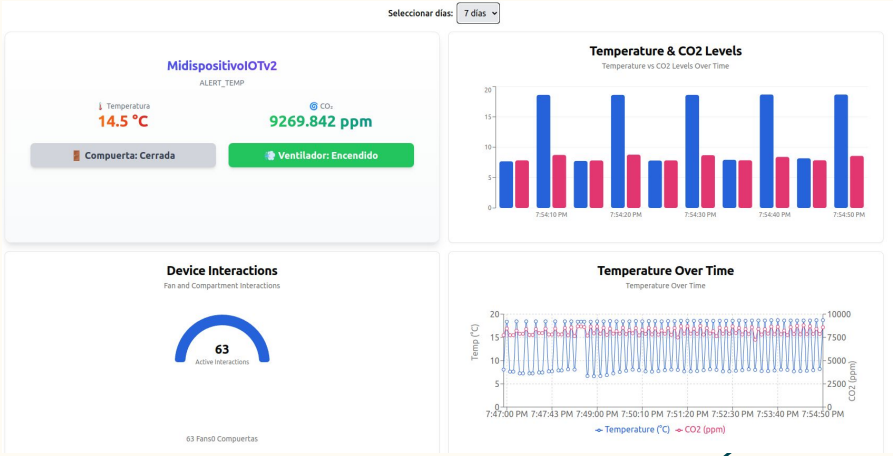
Gráfico Radial

Presenta los datos en una estructura circular, brindando una visualización alternativa de los valores registrados.

Filtrado de datos por días

El sistema permite filtrar las mediciones de sensores según rangos de tiempo específicos, proporcionando opciones para visualizar datos de los últimos 1, 3 o 7 días.





Análisis descriptivo de los datos
Evaluación y Tendencias

Metodología

- Datos capturados durante 1 semana por ESP32
- Almacenamiento en DynamoDB y procesamiento con Python

Análisis de Temperatura

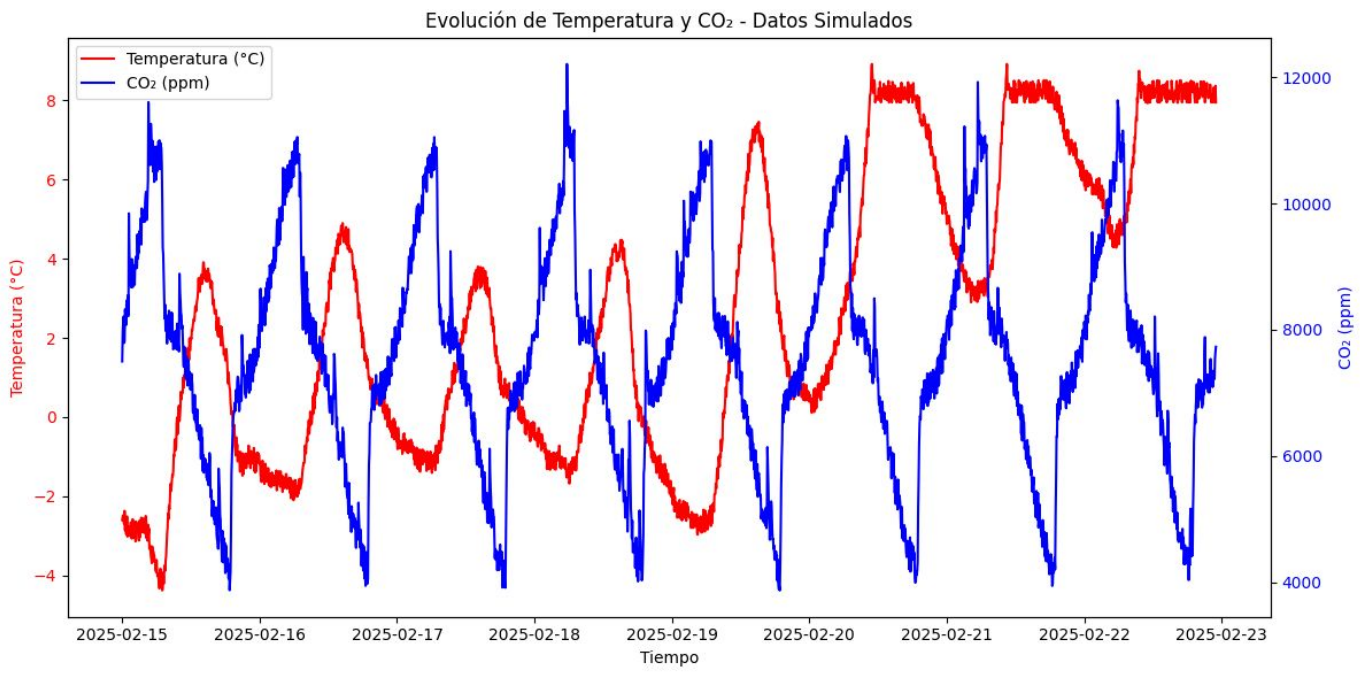
- Rango: -4.38°C a 8.92°C | Promedio: 2.65°C
- Picos máximos: ~2 PM | Mínimos: ~5 AM
- Activación de ventilador al acercarse a 8°C

Análisis de CO_2

- Rango: 3,872 a 12,207 ppm | Promedio: 7,561 ppm
- Incremento notable durante la noche
- Apertura de compuerta al superar 10,000 ppm

Análisis descriptivo de los datos

Evaluación y Tendencias



Análisis descriptivo de los datos
Patrón y Comportamiento de Actuadores

Patrones Identificados

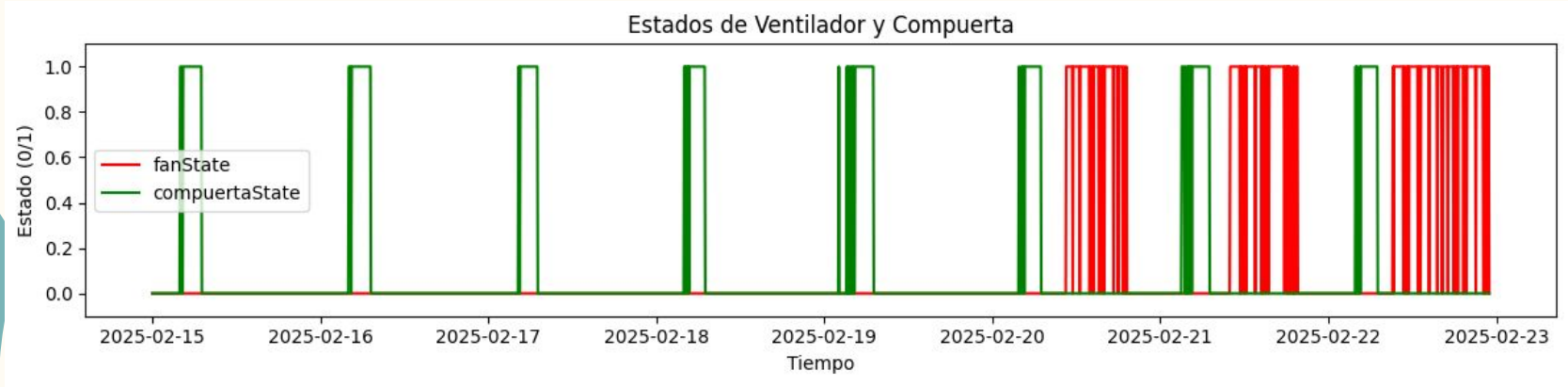
- Temperatura: Ciclo diario bien definido con incremento en horas de tarde
- CO_2: Acumulación durante la noche y madrugada
- Correlación inversa entre ambas variables en ciertos períodos

Componentes Actuadores

- Ventilador: Mayor activación en período diurno
- Compuerta: Mayor actividad nocturna
- Eficacia demostrada en control de parámetros ambientales

Análisis descriptivo de los datos

Patrón y Comportamiento de Actuadores

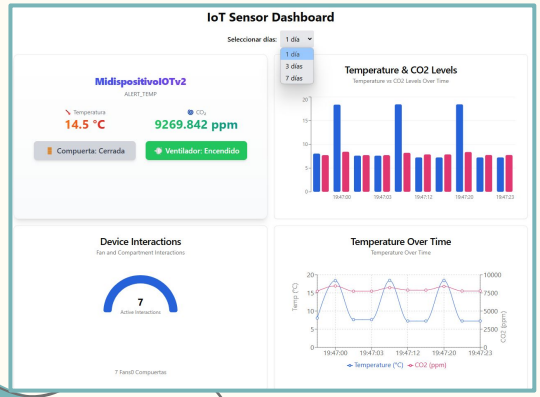


Análisis descriptivo de los datos

Visualización en Tiempo Real

Panel de monitoreo

- Visualización de estados actuales y alertas
- Gráficos comparativos con filtrado temporal (1, 3 o 7 días)



Análisis descriptivo de los datos

Visualización en Tiempo Real