

## PROYECTO INTEGRADOR I –TST 2024

### Sprint Backlog - Semana 22: Implementación de Almacenamiento Local Temporal en ESP32

#### Descripción:

**Descripción:** Definir las funcionalidades necesarias para que el middleware gestione la comunicación entre los dispositivos Edge y la API.

#### Pasos:

1. Revisar las necesidades de comunicación entre los dispositivos Edge y la API.
2. Esquematizar el flujo de datos entre los dispositivos Edge y la API RESTful.
3. Definir los puntos clave de transformación de datos en el middleware.

Definir las funcionalidades necesarias para que el middleware gestione la comunicación entre los dispositivos Edge y la API. Pasos: Revisar las necesidades de comunicación entre los dispositivos Edge y la API. Esquematizar

el flujo de datos entre los dispositivos Edge y la API RESTful. Definir los puntos clave de transformación de datos en el middleware.

Para definir las funcionalidades necesarias del middleware que gestione la comunicación entre los dispositivos Edge y la API RESTful, es fundamental abordar los pasos que has mencionado de manera estructurada. Aquí te dejo una guía detallada para cada uno de los pasos:

#### 1. Revisar las necesidades de comunicación entre los dispositivos Edge y la API

Este paso implica identificar los requisitos de comunicación entre los dispositivos Edge (como el ESP32) y la API. Las necesidades clave a considerar incluyen:

- **Protocolo de Comunicación:** Identificar los protocolos más adecuados para la interacción, como HTTP, MQTT, o WebSockets, dependiendo de los requisitos de latencia, seguridad y volumen de datos.
- **Formato de los Datos:** Definir si los datos se intercambiarán en formato JSON, XML, o binario, y considerar la serialización/deserialización de datos para garantizar que el middleware pueda interpretar los datos correctamente.
- **Seguridad:** Identificar los mecanismos de autenticación y autorización, como el uso de tokens JWT para que los dispositivos Edge accedan a la API de manera segura.
- **Frecuencia de Comunicación:** Establecer la periodicidad o los eventos que desencadenan el envío de datos entre el Edge y la API.
- **Gestión de Conexión y Reintentos:** Implementar la lógica para gestionar la pérdida de conexión y los reintentos automáticos cuando los dispositivos Edge no puedan comunicarse temporalmente con la API.

## 2. Esquematizar el flujo de datos entre los dispositivos Edge y la API RESTful

El esquema debe mostrar cómo los datos fluyen desde los dispositivos Edge hacia la API a través del middleware. Los componentes clave serían:

- **Dispositivos Edge:** Recopilan datos de sensores y envían las lecturas hacia el middleware.
- **Middleware:** Procesa los datos recibidos desde los dispositivos Edge y los transforma (si es necesario) antes de enviarlos a la API. Las funcionalidades del middleware incluyen:
  - **Recolección de datos:** Recibe los datos del dispositivo Edge.
  - **Procesamiento:** Realiza validaciones o conversiones necesarias en los datos.
  - **Enrutamiento:** Decide a qué **endpoint** de la API RESTful deben enviarse los datos.
  - **Manejo de errores:** Detecta errores en la comunicación con la API o en los datos recibidos y los maneja (por ejemplo, reenviando o almacenando temporalmente).
- **API RESTful:** Recibe las solicitudes del middleware, procesa los datos y los almacena en la base de datos correspondiente, además de enviar respuestas al middleware.

## 3. Definir los puntos clave de transformación de datos en el middleware

La transformación de datos es una de las principales responsabilidades del middleware. Aquí se identifican las acciones que el middleware debe realizar para que los datos sean adecuados para la API:

- **Validación de datos:** Antes de enviar los datos a la API, el middleware debe validar si los datos recibidos desde los dispositivos Edge cumplen con los formatos requeridos (por ejemplo, rango de valores de los sensores, campos obligatorios).
- **Normalización de datos:** Los datos recopilados por diferentes dispositivos pueden tener formatos diferentes. El middleware debe estandarizar los datos a un formato común antes de enviarlos a la API (por ejemplo, convertir formatos de tiempo o unidades de medida).
- **Enriquecimiento de datos:** En algunos casos, puede ser necesario agregar metadatos o etiquetas a los datos, como incluir un sello de tiempo o la identificación del dispositivo que envió los datos.
- **Compresión o reducción de datos:** Si el ancho de banda es limitado o los datos son demasiado voluminosos, el middleware puede comprimir los datos o realizar un preprocesamiento para reducir su tamaño.
- **Transformación estructural:** Si el formato de los datos enviados por los dispositivos Edge no coincide con lo esperado por la API, el middleware debe transformar la estructura, por ejemplo, convirtiendo arrays en objetos JSON específicos.

### Resumen de funcionalidades del middleware:

1. **Recepción de datos desde Edge:** Escuchar y recibir datos desde dispositivos Edge.
2. **Transformación de datos:** Validar, normalizar, enriquecer y, si es necesario, comprimir los datos antes de enviarlos a la API.
3. **Comunicación con API RESTful:** Realizar llamadas HTTP a los endpoints de la API RESTful.
4. **Gestión de errores:** Implementar reintentos automáticos y almacenamiento temporal en caso de fallo en la conexión.

5. **Seguridad:** Autenticar y autorizar las comunicaciones entre Edge y API, posiblemente usando tokens JWT.
6. **Manejo de desconexiones:** Almacenar localmente los datos si se pierde la conexión y reenviarlos cuando se restablezca la conectividad.

Este enfoque debería proporcionarte una base sólida para gestionar la comunicación y procesamiento entre los dispositivos Edge y la API mediante un middleware robusto y eficiente.

Para ilustrar cómo se implementaría un middleware que gestione la comunicación entre los dispositivos Edge (como un ESP32) y una API RESTful, te mostraré un ejemplo paso a paso utilizando un enfoque práctico con tecnologías comunes.

## Escenario:

Supongamos que tienes un **ESP32** conectado a sensores de temperatura y humedad, y deseas enviar estas lecturas a una API RESTful que maneja los datos en un servidor. El middleware se encargará de recibir, procesar y reenviar esos datos.

### 1. Revisión de las necesidades de comunicación

- **Protocolo de Comunicación:** Utilizaremos **HTTP** para enviar las lecturas de los sensores desde el ESP32 al middleware.
- **Formato de los Datos:** Los datos se enviarán en formato **JSON**.
- **Seguridad:** Implementaremos autenticación mediante **tokens JWT**.
- **Frecuencia de Comunicación:** El ESP32 enviará datos cada minuto.
- **Gestión de Conexión:** El middleware almacenará temporalmente los datos si la conexión a la API falla, y los reenviará cuando se restablezca la conectividad.

### 2. Esquematización del flujo de datos

- **ESP32:** Toma lecturas de los sensores.
- **Middleware:** Recibe los datos del ESP32, los valida y los envía a la API RESTful.
- **API RESTful:** Recibe los datos procesados y los almacena en una base de datos.

El flujo sería algo así:


1. **ESP32** → Envía datos en formato JSON → **Middleware**
2. **Middleware** → **Procesa y valida datos** → API RESTful
3. **API RESTful** → Recibe y almacena datos en la base de datos

### 3. Ejemplo de código

#### Middleware usando Flask (Python)

El middleware aquí actúa como un servidor que recibe datos del ESP32, los valida y luego los envía a la API RESTful.

python

 Copiar código

```
from flask import Flask, request, jsonify
import requests
import time

app = Flask(__name__)

# API RESTful URL y autenticación JWT
API_URL = "https://mi-api.com/datos"
JWT_TOKEN = "eyJhbGciOiJIUzI1NiIsInR...".

# Función para enviar datos a la API RESTful
def enviar_a_api(datos):
    headers = {
        'Authorization': f'Bearer {JWT_TOKEN}',
        'Content-Type': 'application/json'
    }
    try:
        respuesta = requests.post(API_URL, json=datos, headers=headers)
        if respuesta.status_code == 200:
            return True
        else:
            return False
    except Exception as e:
        print(f"Error al conectar con la API: {e}")
        return False

# RUTA: Recibe los datos desde el ESP32
@app.route('/middleware', methods=['POST'])
def recibir_datos():
    try:
```

//...

..//

```

        'Authorization': f'Bearer {JWT_TOKEN}',
        'Content-Type': 'application/json'
    }
    try:
        respuesta = requests.post(API_URL, json=datos, headers=headers)
        if respuesta.status_code == 200:
            return True
        else:
            return False
    except Exception as e:
        print(f"Error al conectar con la API: {e}")
        return False

# RUTA: Recibe los datos desde el ESP32
@app.route('/middleware', methods=['POST'])
def recibir_datos():
    try:
    except Exception as e:
        return jsonify({"error": str(e)}), 500

# Función para reenviar datos almacenados temporalmente cuando la conexión se restablezca
def reenviar_datos_pendientes():
    try:
        with open('datos_pendientes.txt', 'r') as f:
            for linea in f:
                datos = eval(linea.strip())
                exito = enviar_a_api(datos)
                if exito:
                    print("Datos reenviados correctamente.")
                else:
                    print("Error al reenviar los datos.")
    except FileNotFoundError:
        print("No hay datos pendientes por reenviar.")
    except Exception as e:
        print(f"Error al procesar datos pendientes: {e}")

if __name__ == "__main__":
    # Reenvía datos pendientes al iniciar el middleware
    reenviar_datos_pendientes()
    # Ejecuta el servidor Flask
    app.run(host='0.0.0.0', port=5000)

```



## Descripción del código:

1. **Recepción de datos:** El middleware expone una ruta (/middleware) que recibe los datos del ESP32 en formato JSON.
2. **Validación de datos:** Valida que el JSON contenga los campos temperatura y humedad. Si faltan, devuelve un error.
3. **Transformación de datos:** En este ejemplo, no es necesaria ninguna transformación adicional de datos. Sin embargo, puedes añadirla aquí (por ejemplo, convertir valores a una escala común).
4. **Comunicación con la API RESTful:** Usa la función `enviar_a_api` para hacer una solicitud POST a la API con un **token JWT** para la autenticación.
5. **Gestión de errores:** Si falla la conexión con la API, los datos se guardan temporalmente en un archivo `datos_pendientes.txt`.
6. **Reintento de envío:** Al iniciar el middleware, se llama a la función `reenviar_datos_pendientes`, que intenta reenviar cualquier dato almacenado temporalmente si la conexión falla previamente.

## Código del ESP32 (Arduino)

```
cpp Copiar código

#include <WiFi.h>
#include <HttpClient.h>

const char* ssid = "TU_SSID";
const char* password = "TU_PASSWORD";
const char* middleware_url = "http://IP_DEL_SERVIDOR:5000/middleware";

void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
  }
}
```

//..

..//

```
// Cuerpo de la solicitud
String json_data = "{\"temperatura\":" + String(temperatura) + ",\"humedad\":" + String(

// Enviar datos al middleware
http.begin(middleware_url);
http.addHeader("Content-Type", "application/json");
int httpResponseCode = http.POST(json_data);

if (httpResponseCode > 0) {
    String respuesta = http.getString();
    Serial.println(httpResponseCode);
    Serial.println(respuesta);
} else {
    Serial.println("Error en la solicitud POST");
}

http.end();
Serial.println("No conectado a WiFi");
}

// Esperar 60 segundos antes de la siguiente lectura
delay(60000);
}
```

### Explicación:

- El ESP32 toma lecturas simuladas de temperatura y humedad y las envía en formato JSON al middleware usando HTTP.

### Funcionalidades del Middleware para el Proyecto AQI

Este documento detalla las funcionalidades esenciales que debe tener el middleware encargado de gestionar la comunicación entre los dispositivos Edge (como el ESP32) y una API RESTful que almacena los datos de sensores en una base de datos remota. Este middleware está diseñado para procesar varios datos de sensores en una única solicitud y desglosarlos para ser almacenados en la base de datos, todo mientras maneja la autenticación de usuarios y dispositivos.

### Necesidades del Sistema

Los dispositivos Edge envían regularmente datos de sensores que incluyen múltiples mediciones (por ejemplo, temperatura, humedad, presión, etc.). Sin embargo, la base de datos remota donde se almacenan estos datos está configurada para aceptar solo un dato por fila. Además, el sistema debe asegurar que tanto

los usuarios como los dispositivos estén autenticados correctamente para evitar el acceso no autorizado.

## Pasos para Definir las Funcionalidades del Middleware

### 1. Revisar las Necesidades de Comunicación entre los Dispositivos Edge y la API

Para garantizar una comunicación fluida y segura entre los dispositivos Edge y la API remota, es necesario:

- **Formato de los datos enviados:** Los dispositivos Edge envían un conjunto de datos en un solo paquete (por ejemplo, cuatro mediciones distintas). El middleware debe desglosar este paquete en múltiples solicitudes, de manera que cada medición se registre como una fila individual en la base de datos remota.
- **Autenticación de dispositivos y usuarios:** Cada dispositivo y usuario debe autenticarse con el middleware antes de enviar o manipular datos. El middleware debe verificar tanto las credenciales del usuario como del dispositivo mediante JWT o algún otro método de autenticación.
- **Gestión de errores:** Si alguno de los datos no puede almacenarse, el middleware debe decidir si se deben reintentar los envíos, ignorar el error o detener el procesamiento para evitar inconsistencias en los datos.

### 2. Esquematizar el Flujo de Datos entre los Dispositivos Edge y la API RESTful

El flujo de datos general entre los dispositivos Edge y la API remota es el siguiente:

1. **Autenticación:**
  - Los dispositivos Edge envían una solicitud de autenticación al middleware. El middleware valida las credenciales y genera un token JWT para autenticar las siguientes solicitudes.
  - Si el dispositivo ya tiene un token, lo envía con la solicitud de datos, y el middleware lo valida antes de procesar los datos.
2. **Recepción de datos:**
  - El dispositivo Edge envía un JSON que contiene varias mediciones de sensores (por ejemplo, temperatura, humedad, presión).
  - El middleware recibe el JSON y lo desglosa en mediciones individuales.
3. **Transformación de datos:**
  - Cada medición se normaliza o transforma si es necesario (por ejemplo, convertir grados Celsius a Kelvin).
4. **Envío a la API remota:**
  - El middleware envía las mediciones desglosadas como solicitudes individuales a la API RESTful de la base de datos.
  - La API almacena cada medición en una fila separada.
5. **Manejo de errores:**
  - Si una solicitud a la API remota falla, el middleware debe manejar el error según la configuración (reintentar, detener el flujo o ignorar el error).

### 3. Definir los Puntos Clave de Transformación de Datos en el Middleware

El middleware debe incluir las siguientes funcionalidades clave para procesar los datos de los sensores antes de enviarlos a la base de datos:

1. **Recepción de múltiples mediciones en una sola solicitud:**
  - El middleware recibe un paquete JSON desde el dispositivo Edge que contiene múltiples mediciones.
  - Por ejemplo:



```

json
Copiar código
{
  "id_dispositivo": 1,
  "fecha_recoleccion": "2024-10-01 12:00:00",
  "mediciones": [
    {"valor": 25.5, "unidad": "C"},
    {"valor": 65.2, "unidad": "%"},
    {"valor": 1013.25, "unidad": "hPa"},
    {"valor": 15.4, "unidad": "µg/m³"}
  ]
}

```

## 2. Desglose de datos:

- El middleware toma las mediciones del paquete y las desglosa en múltiples solicitudes. Cada solicitud incluirá solo una medición y se enviará de forma individual a la API RESTful.
- El middleware procesará algo como:

```

json
Copiar código
{
  "id_dispositivo": 1,
  "fecha_recoleccion": "2024-10-01 12:00:00",
  "medición": {"valor": 25.5, "unidad": "C"}
}

```

- Esto se repetirá para cada una de las mediciones en el JSON original.

## 3. Normalización de datos:

- Algunos datos necesitan ser transformados antes de ser enviados. Por ejemplo, si una medición de temperatura está en grados Celsius, el middleware puede convertirla a Kelvin para asegurar consistencia:

```

python
Copiar código
@app.before_request
def normalize_data():
    if request.method == "POST" and request.json:
        mediciones = request.json.get("mediciones", [])
        for medicion in mediciones:
            if medicion['unidad'] == "C":
                medicion['valor'] += 273.15
                medicion['unidad'] = "K"

```

## 4. Autenticación de solicitudes:

- El middleware asegura que todas las solicitudes estén autenticadas usando JWT. Esto es esencial para garantizar que solo dispositivos autorizados puedan enviar datos.
- Al recibir una solicitud en la ruta de datos (/data), el middleware verifica si el token JWT es válido:

```

python
Copiar código
@jwt_required()
@app.route('/data', methods=['POST'])
def handle_data():
    # Procesar la solicitud de datos aquí
    pass

```

## 5. Envío de datos desglosados a la API:

- Tras el desglose y normalización, el middleware realiza solicitudes POST individuales a la API remota para almacenar cada medición en la base de datos.

#### 6. Gestión de errores:

- Si alguna solicitud a la API falla, el middleware debe manejar el error. Las estrategias de gestión de errores pueden incluir:
  - **Reintentar el envío:** Si la conexión falla temporalmente.
  - **Ignorar el error:** Si se decide no procesar ese dato.
  - **Registrar el error:** Para su análisis posterior sin interrumpir el flujo.

**Archivo Markdown:** `funcMiddleware.md`

Voy a generar todo este contenido en un archivo Markdown con el nombre `funcMiddleware.md`, que incluirá todos los detalles descritos aquí.