Profesor: Gonzalo Vera

Rutas del middleware para interactuar con la API

Introducción

El **middleware** desempeña un papel crucial en los sistemas de Internet de las Cosas (IoT), actuando como un puente de comunicación entre los dispositivos IoT y una API centralizada. En el contexto de este proyecto de **monitoreo de la calidad del aire**, el middleware se encarga de recibir datos de los sensores instalados en los dispositivos edge (como los valores de temperatura, humedad y otros parámetros ambientales), validarlos y reenviarlos a la API correspondiente para su procesamiento y almacenamiento en la nube.

El principal objetivo del middleware es **intermediar la comunicación** entre los dispositivos IoT y la API, garantizando que los datos sean transmitidos de forma eficiente y segura. Esto implica la gestión de las solicitudes **POST** enviadas por los dispositivos edge, que contienen los datos recopilados, y las solicitudes **GET**, que permiten la recuperación de datos procesados o almacenados en la API. Además, el middleware se encarga de validar los datos antes de reenviarlos a la API para asegurar su integridad, lo que es esencial para la precisión de los sistemas de monitoreo en tiempo real.

Este componente intermedio permite una arquitectura más flexible y escalable, ya que puede adaptarse a diferentes tipos de dispositivos y sensores IoT. Asimismo, centraliza la lógica de negocio y facilita la interoperabilidad entre distintos dispositivos y servicios, al actuar como un punto de control para las comunicaciones y el flujo de información entre la capa de borde (edge) y la nube.

Rutas Definidas

En el desarrollo del middleware para el proyecto de **monitoreo de la calidad del aire**, se han definido dos rutas clave para interactuar con la **API RESTful**: una ruta **GET** y una ruta **POST**. Estas rutas permiten la comunicación bidireccional entre los dispositivos IoT (como el **ESP32**) y la API, facilitando tanto la recuperación de información como el envío de datos desde los sensores.

Ruta GET

La ruta **GET** está diseñada para obtener información desde la API. Esta solicitud permite a los dispositivos o usuarios consultar datos específicos, como el estado actual de los sensores, registros históricos o resultados procesados que han sido almacenados previamente en la base de datos de la API.

1. **Función principal:** La ruta GET se utiliza para **solicitar información**. El middleware recibe la solicitud desde los dispositivos edge o desde una interfaz de usuario, consulta la API y devuelve la respuesta con los datos solicitados.

2. **Flujo de trabajo:**

    - El dispositivo IoT o el cliente envía una solicitud GET al middleware.
    - El middleware procesa la solicitud y se comunica con la API utilizando la URL correspondiente.
    - La API responde con los datos requeridos, que pueden incluir información de los sensores, datos procesados, o registros históricos.
    - El middleware devuelve esta respuesta al dispositivo que realizó la solicitud.

3. **Ejemplo:** Si el ESP32 solicita el historial de los niveles de temperatura y humedad, el middleware se conecta con la API y devuelve los datos al ESP32 o a un dispositivo cliente.

Ruta POST

La ruta **POST** es utilizada para recibir datos generados por los dispositivos IoT, como los valores de temperatura, humedad y otros parámetros de la calidad del aire. Estos datos se validan dentro del middleware y, si cumplen con los criterios predefinidos, se reenvían a la API para su procesamiento y almacenamiento.

1. **Función principal:** La ruta POST se utiliza para **enviar información** desde los dispositivos IoT (como el ESP32) hacia la API. Este flujo es comúnmente empleado para transmitir los datos capturados por los sensores en tiempo real.

2. **Flujo de trabajo:**

    - El ESP32 recopila datos de los sensores, como la temperatura y la humedad.
    - Envía estos datos al middleware a través de una solicitud POST.
    - El middleware recibe esta solicitud y valida los datos: verifica si los valores de los sensores están dentro de un rango aceptable o si hay algún error en la transmisión.
    - Si los datos son válidos, el middleware los reenviará a la API mediante otra solicitud POST. En caso de que no pasen la validación, se devuelve una respuesta de error.
    - La API recibe los datos, los procesa y los almacena en la base de datos.
    - El middleware puede devolver una respuesta al ESP32 confirmando que los datos fueron enviados exitosamente.

3. **Validación:** El middleware se asegura de que los datos recibidos sean coherentes y estén en el formato correcto. Por ejemplo, si los valores de temperatura y humedad son inválidos (como valores fuera de rango o datos corruptos), el middleware no reenviará la solicitud a la API, y en su lugar, devolverá un mensaje de error al ESP32.

En la ruta **POST** del middleware, la validación de los datos enviados por los dispositivos IoT (como el **ESP32**) es crucial para asegurar que la información recibida sea precisa y esté en el formato correcto antes de reenviarla a la **API**.

Pasos de Validación en la Ruta POST

1. **Recepción de los Datos del ESP32**

    - El middleware recibe una solicitud **POST** del ESP32 con los datos que se van a enviar, como la temperatura y la humedad, en formato **JSON**.

Comprobación de la Estructura de los Datos

- El middleware analiza la estructura de la solicitud **JSON** para asegurarse de que contiene todas las claves necesarias, como temperatura y humedad. Esto garantiza que los datos enviados por el ESP32 sean completos.
- Si faltan campos obligatorios (como la temperatura o la humedad), el middleware devolverá un mensaje de error indicando que la solicitud está incompleta.

Ejemplo de validación

```
if 'temperatura' not in data or 'humedad' not in data:
    return {"error": "Datos incompletos: falta temperatura o humedad"}, 400
```

Validación del Formato de los Datos

- Además de verificar la estructura, el middleware también valida que los valores recibidos tengan el **tipo de dato** correcto. Por ejemplo, se asegura de que la temperatura y la humedad sean valores numéricos (tipo float o int).
- Si alguno de los valores no es numérico, la solicitud es rechazada, y se devuelve un mensaje de error indicando que los datos son inválidos.
- **Ejemplo de validación**

```
if not isinstance(data['temperatura'], (float, int)) or not isinstance(data['humed
    return {"error": "Formato de datos incorrecto: la temperatura y la humedad deb
```

if not isinstance(data['temperatura'], (float, int)) or not isinstance(data['humedad'], (float, int)):

return {"error": "Formato de datos incorrecto: la temperatura y la humedad deben ser numéricos"}, 400

Verificación de Rangos de Valores

- Para asegurarse de que los datos sean coherentes, el middleware puede incluir una verificación de **rangos aceptables**. Por ejemplo, la temperatura debe estar dentro de un rango razonable (por ejemplo, entre -50°C y 60°C), y lo mismo para la humedad (entre 0% y 100%).
- Si los valores están fuera de estos rangos, el middleware considerará que los datos son inválidos y devolverá un error.
- **Ejemplo de validación**

```
if not (-50 <= data['temperatura'] <= 60) or not (0 <= data['humedad'] <= 100):
    return {"error": "Valores fuera de rango: la temperatura debe estar entre -50
```

if not (-50 <= data['temperatura'] <= 60) or not (0 <= data['humedad'] <= 100):

return {"error": "Valores fuera de rango: la temperatura debe estar entre -50 y 60, y la humedad entre 0 y 100"}, 400

Respuesta de Validación

- ✓ Si los datos pasan todas las verificaciones, el middleware los considera válidos y los **reenvía a la API**.
- ✓ Si los datos no son válidos, el middleware devuelve una respuesta de error con el código de estado **400** (Bad Request) y un mensaje detallado explicando el motivo de la falla.

Pruebas con Herramientas como Postman

Postman es una herramienta útil para realizar pruebas de APIs y verificar que las rutas **POST** y **GET** de tu middleware funcionen correctamente.

1. Instalación de Postman

- ✓ Si no tienes Postman instalado, puedes descargarlo desde su sitio web oficial.
- ✓ Una vez instalado, abre la aplicación.

2. Pruebas de la Ruta GET

- ✓ **Paso 1:** Selecciona el método **GET** en la interfaz de Postman.
- ✓ **Paso 2:** Ingresa la **URL** de tu middleware para la ruta GET. Por ejemplo:

```
http://localhost:5000/api/obtener-datos
```

- ✓ **Paso 3:** Haz clic en el botón **Send** para enviar la solicitud. El middleware debería devolver una respuesta con datos o un mensaje indicando que la solicitud se ha procesado.
- ✓ **Resultado esperado:** Deberías ver en la sección de respuesta de Postman los datos obtenidos desde la API o un mensaje indicando el estado de la conexión.
- ✓ **Ejemplo de respuesta:**

```
{
  "temperatura": 24.5,
  "humedad": 60.2
}
```

3. Pruebas de la Ruta POST

- **Paso 1:** Selecciona el método **POST** en Postman.
- **Paso 2:** Ingresa la **URL** de tu middleware para la ruta POST. Por ejemplo:

```
http://localhost:5000/api/enviar-datos
```

- **Paso 3:** Dirígete a la pestaña **Body** en Postman y selecciona **raw** como el tipo de datos. Luego selecciona **JSON** en el menú desplegable que está al lado de la opción **raw**.
- **Paso 4:** Ingresa los datos de ejemplo en formato **JSON** que quieras enviar al middleware, por ejemplo:

```
{
  "temperatura": 24.5,
  "humedad": 60.2
}
```

- **Paso 5:** Haz clic en el botón **Send** para enviar la solicitud POST.

  Resultado esperado:

- Si los datos son válidos, deberías recibir una respuesta con el código de éxito (por ejemplo, **200 OK**) y un mensaje indicando que los datos han sido enviados correctamente a la API.
- Si los datos son inválidos (por ejemplo, si falta algún campo o los valores están fuera de rango), deberías recibir una respuesta con un código de error (como **400 Bad Request**) y un mensaje explicando el error.

Ejemplo de respuesta de éxito:

```
{
  "mensaje": "Datos recibidos correctamente y reenviados a la API"
}
```

Ejemplo de respuesta de error:

```
{
  "error": "Datos incompletos: falta temperatura o humedad"
}
```

4. **<u>Pruebas de Casos de Error</u>**

- Para asegurarte de que el middleware está manejando adecuadamente los errores, puedes realizar pruebas enviando datos incorrectos o incompletos.
- **Ejemplo de datos incorrectos (falta el campo de humedad):**

```
{
  "temperatura": 24.5
}
```

Resultado esperado: El middleware debería devolver una respuesta de error con un mensaje explicando que el campo de humedad falta:

```
{
  "error": "Datos incompletos: falta temperatura o humedad"
}
```

5. **Monitoreo de las Solicitudes**

   - ✓ En la consola de **Postman**, puedes ver todos los detalles de las solicitudes enviadas y sus respuestas, lo que te permite identificar posibles problemas de configuración o errores en la lógica del middleware.

Realizar pruebas con Postman te permite asegurarte de que las rutas **GET** y **POST** de tu middleware funcionan correctamente antes de probar con dispositivos físicos como el ESP32. Las pruebas permiten simular datos y escenarios que los dispositivos IoT enviarán o solicitarán, y te ayudan a identificar cualquier error en la validación o en la lógica de la interacción con la API.

Este enfoque es fundamental para verificar que el middleware maneja adecuadamente tanto los datos como las posibles excepciones antes de ser implementado en un entorno de producción.

Resumen de las rutas

- **GET**: Sirve para **obtener** información desde la API. Los dispositivos IoT o los usuarios pueden consultar información a través de esta ruta.
- **POST**: Se utiliza para **enviar** datos generados por los sensores desde los dispositivos IoT (como el ESP32). El middleware valida los datos antes de reenviarlos a la API para su procesamiento.

Configuración del middleware en Flask

Para **configurar el middleware en Flask** y asegurar que interactúe correctamente con el ESP32 y la API RESTful. Este middleware servirá como intermediario para gestionar solicitudes **POST** y **GET**, y también realizar la validación de datos.

Objetivo:

- Configurar las rutas **POST** y **GET** en un servidor middleware para recibir los datos del ESP32 y enviarlos a la API **https://api.gonaiot.com/plata/datos_dispositivos**.
- Implementar validación de datos para asegurar que los valores recibidos (temperatura y humedad) sean correctos.
- Devolver respuestas adecuadas para los dispositivos Edge.

Recapitulando las necesidades del middleware:

- **Rutas para interactuar con la API**:
  - **GET**: Para obtener datos desde la API.
  - **POST**: Para enviar datos a la API.
- **Validación de datos** en las solicitudes.
- Asegurarse de que las respuestas sean adecuadas para los dispositivos Edge (como el ESP32).

Paso 1: Instalar las dependencias

Antes de empezar a codificar el middleware, asegúrate de tener **Flask** y **requests** instalados. Si aún no las tienes instaladas, puedes hacerlo con el siguiente comando:

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

app = Flask(__name__)
```

2. Variables de configuración para la API

Declaramos la URL del servidor de la API y, opcionalmente, una clave de autenticación que la API pueda requerir. En tu caso, puedes necesitarla dependiendo de cómo esté configurada la API.

```python
API_URL = "https://api.gonaiot.com/plata/datos_dispositivos"
API_KEY = "plata"
```

3. Ruta GET para obtener datos de la API

Aquí creamos una ruta para que el middleware haga una solicitud **GET** a la API y reciba los datos desde el servidor. Estos datos serán enviados de vuelta a cualquier dispositivo que haga la solicitud.

- **Ruta**: /obtener_datos
- **Método HTTP**: GET

```python
@app.route('/obtener_datos', methods=['GET'])
def obtener_datos():
    try:
        # Hacer la solicitud GET a la API
        headers = {
            'Authorization': f'Bearer {API_KEY}'
        }
        response = requests.get(API_URL, headers=headers)

        if response.status_code == 200:
            data = response.json()
            return jsonify(data), 200
        else:
            return f"Error al obtener los datos: {response.status_code}", response.status_

    except Exception as e:
        return f"Error: {str(e)}", 500
```

3. Ruta GET para obtener datos de la API

Aquí creamos una ruta para que el middleware haga una solicitud **GET** a la API y reciba los datos desde el servidor. Estos datos serán enviados de vuelta a cualquier dispositivo que haga la solicitud.

- **Ruta**: /obtener_datos
- **Método HTTP**: GET

```python
@app.route('/enviar_datos', methods=['POST'])
def enviar_datos():
    try:
        # Obtener los datos enviados por el ESP32 en formato JSON
        datos = request.json

        # Validación básica de los datos
        if 'temperatura' not in datos or 'humedad' not in datos:
            return "Faltan los campos 'temperatura' o 'humedad' en los datos", 400

        # Preparar el encabezado de la solicitud
        headers = {
            'Authorization': f'Bearer {API_KEY}',
            'Content-Type': 'application/json'
        }

        # Hacer la solicitud POST a la API con los datos del ESP32
        response = requests.post(API_URL, headers=headers, json=datos)

        if response.status_code == 201:
            return jsonify({"mensaje": "Datos enviados con éxito a la API"}), 201
        else:
            return f"Error al enviar los datos: {response.status_code}", response.status_

    except Exception as e:
        return f"Error: {str(e)}", 500
```

- **Validación de datos**:
    - Se verifica que los campos temperatura y humedad estén presentes en los datos enviados por el ESP32.
    - Si no están, se devuelve un error con código **400** (Bad Request).
- **Envío de datos a la API**:
    - Los datos del ESP32 se envían a la API utilizando el método **POST** y se incluyen en el cuerpo de la solicitud.
    - Si la API responde con un código **201**, significa que los datos se han enviado correctamente.

5. Ejecución de la aplicación

Finalmente, iniciamos la aplicación Flask para que escuche las solicitudes entrantes.

```python
if __name__ == '__main__':
    app.run(debug=True)
```

Paso 4: Pruebas del Middleware

1. **Prueba de la ruta GET**:
    - Puedes hacer una solicitud GET a http://127.0.0.1:5000/obtener_datos desde tu navegador o usando una herramienta como Postman para ver los datos que devuelve la API.
2. **Prueba de la ruta POST**:
    - Puedes enviar datos JSON a la ruta http://127.0.0.1:5000/enviar_datos desde Postman. Por ejemplo:

Verifica que el middleware reenvíe los datos a la API y devuelva la respuesta adecuada.

Paso 5: Código

```
from flask import Flask, request, jsonify
import requests
app = Flask(__name__)
# Variables de configuración de la API
API_URL = "https://api.gonaiot.com/plata/datos_dispositivos"
API_KEY = "plata"  # Si la API requiere autenticación, puedes agregarla aquí

# Ruta GET para obtener los datos desde la API
@app.route('/obtener_datos', methods=['GET'])
def obtener_datos():
  try:
    # Hacer la solicitud GET a la API
    headers = {
      'Authorization': f'Bearer {API_KEY}'
    }
    response = requests.get(API_URL, headers=headers)

    if response.status_code == 200:
      data = response.json()
      return jsonify(data), 200
```

```
        else:
            return f"Error al obtener los datos: {response.status_code}", response.status_code
    except Exception as e:
        return f"Error: {str(e)}", 500

# Ruta POST para recibir datos del ESP32 y enviarlos a la API
@app.route('/enviar_datos', methods=['POST'])
def enviar_datos():
    try:
        # Obtener los datos enviados por el ESP32 en formato JSON
        datos = request.json

        # Validación básica de los datos
        if 'temperatura' not in datos or 'humedad' not in datos:
            return "Faltan los campos 'temperatura' o 'humedad' en los datos", 400

        # Preparar el encabezado de la solicitud
        headers = {
            'Authorization': f'Bearer {API_KEY}',
            'Content-Type': 'application/json'
        }
        # Hacer la solicitud POST a la API con los datos del ESP32
        response = requests.post(API_URL, headers=headers, json=datos)

        if response.status_code == 201:
            return jsonify({"mensaje": "Datos enviados con éxito a la API"}), 201
        else:
            return f"Error al enviar los datos: {response.status_code}", response.status_code
    except Exception as e:
        return f"Error: {str(e)}", 500
# Ejecutar la aplicación Flask
if __name__ == '__main__':
    app.run(debug=True)
```