



Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

Redes GR 20

**Proyecto Grupal Solución de IoT para Ciudades Inteligentes**

Documentación Técnica

**Estudiantes:**

Sebastián Bogantes Rodríguez - Carné: 2020028437

Fernando González Ramón - Carné: 2021075114

Pablo Sandí Sánchez - Carné: 2021120523

**Profesor:**

Rogelio González Quirós

Centro Académico de Alajuela

Semestre II | 2024

## ÍNDICE

<b>Solución Planteada.....</b>	<b>3</b>
<b>Arquitectura de IoT.....</b>	<b>4</b>
Componentes Principales.....	4
<b>Fuentes de Datos.....</b>	<b>5</b>
Tabla de Descripción Detallada.....	5
<b>Sensorización Utilizada.....</b>	<b>7</b>
Detalles del Sensor Simulado.....	7
Ventajas de la Simulación.....	7
<b>Herramientas de Visualización.....</b>	<b>9</b>
Node-RED Dashboard.....	9
Características del Dashboard.....	9
Telegram API para Notificaciones.....	9
Características de las Notificaciones.....	9
<b>Análisis de Resultados.....</b>	<b>11</b>
Hallazgos.....	11
Desempeño del Sistema.....	11
Impacto del Proyecto.....	16
Funcionamiento del sistema (código).....	17
<b>Conclusiones.....</b>	<b>19</b>
Logros Destacados.....	19
Dificultades y Solución.....	19
Posibles Mejoras Futuras.....	20
<b>Bibliografía.....</b>	<b>21</b>
<b>Anexos.....</b>	<b>22</b>

## **Solución Planteada**

En general, en Costa Rica y en diferentes lugares alrededor del mundo, es un hecho que el ruido ambiental se ha convertido en un problema significativo que afecta la calidad de vida de las personas. Los niveles elevados de ruido están asociados con problemas de salud, incluyendo estrés, trastornos del sueño y, a largo plazo, problemas auditivos. Como grupo de trabajo y ante esta situación, nos pareció que sería importante y generaría un gran aporte a la sociedad la implementación de sistemas de monitoreo de ruidos, los cuales permitan conocer en tiempo real los niveles de ruido en distintas áreas de las ciudades, y con esto facilitar la toma de decisiones en políticas de salud pública y ordenamiento urbano.

La solución que planteamos en este proyecto consiste en desarrollar un sistema de monitoreo de ruido basado en Internet de las Cosas (IoT), que utilice sensores de ruido distribuidos estratégicamente para captar datos en tiempo real (aspecto en el cual desarrollamos una simulación). Estos datos son procesados y visualizados mediante un dashboard accesible, lo que permite la identificación rápida de áreas problemáticas y mediante la ayuda de un bot, poder generar alertas cuando se superan niveles críticos. Esta herramienta de monitoreo no solo ayuda a visualizar la magnitud del problema, sino que también brinda un medio para evaluar la efectividad de las políticas implementadas por diferentes instituciones para reducir el ruido en las ciudades.

## Arquitectura de IoT

La arquitectura del sistema sigue una estructura distribuida y escalable que facilita la recopilación, el procesamiento y la visualización de datos en tiempo real. La solución consta de varios componentes clave que interactúan de manera coordinada para garantizar la captura eficiente de datos y su visualización adecuada.

### Componentes Principales

1. **Sensores de Ruido:** Dispositivos encargados de medir el nivel de ruido en decibelios (dB) en tiempo real. Cada sensor está configurado para enviar los datos capturados a través de un protocolo de comunicación MQTT. Como se mencionó anteriormente, esta parte del proyecto **se realiza mediante una simulación** de los sonidos, esto **debido a la falta de componentes físicos** para el mismo.
2. **Broker MQTT:** Componente que actúa como intermediario en la comunicación de los sensores con el servidor de procesamiento de datos. MQTT es un protocolo ligero y eficiente, ideal para aplicaciones IoT, ya que permite una comunicación confiable y rápida entre dispositivos de bajo consumo y el servidor.
3. **Servidor de Procesamiento (Node-RED):** Se utilizó Node-RED para recibir y procesar los datos de los sensores. Este componente nos permite configurar flujos de datos de forma visual, facilitando la integración con otros sistemas. Este servidor también incluye una función de análisis para detectar niveles críticos de ruido y activar alertas en el bot de Telegram cuando estos son superados.
4. **Dashboard de Visualización:** Utilizando la extensión Node-RED Dashboard, pudimos representar gráficamente los datos de ruido en tiempo real, esto para ofrecer una visión clara de las condiciones actuales en distintas áreas. Esto permite a los usuarios finales identificar patrones de ruido y zonas críticas de forma visual e intuitiva.
5. **Bot de Telegram para Alertas:** Como componente innovador desarrollamos un bot en Telegram para notificar acerca de estas alertas. Este se encarga de notificar a los usuarios cuando se detecta un nivel de ruido que supera el umbral predefinido. Este componente permite que los usuarios reciban alertas inmediatas en sus dispositivos móviles, facilitando una respuesta rápida.

Figura 1: Diagrama de Arquitectura IoT [Anexos]

### Fuentes de Datos

Las fuentes de datos del proyecto son los niveles de ruido en decibelios (dB) capturados por los sensores distribuidos en distintas áreas de la ciudad. Estos sensores recogen continuamente información sobre el ambiente (mediante una simulación), posteriormente envían los datos a través del protocolo MQTT hacia el servidor de procesamiento y el bot se encarga de realizar la alerta de los mismos.

---

Tabla de Descripción Detallada

Fuente de Datos	Descripción	Propósito	Frecuencia de Captura	Formato de Datos
<b>Sensor de Ruido</b> (Simulación en <i>simulacion_microfono.py</i> )	Este sensor simulado captura niveles de ruido en decibelios, generando datos que se envían a través de MQTT para representar la contaminación acústica.	Monitorear la contaminación acústica en tiempo real	Continuo	Número Flotante (dB)
<b>MQTT Broker</b>	Middleware para la transmisión de datos en tiempo real desde el sensor de ruido al sistema de procesamiento en Node-RED.	Facilitar la transmisión de datos	Continuo	JSON
<b>Node-RED</b> (Flujos)	Plataforma de	Centralizar y	Tiempo real	JSON

<i>flujo_niveles_ruido.json</i> y <i>flujo_niveles_ruido_2.json</i> )	desarrollo en la que se configuran los flujos de datos, incluyendo la recepción, procesamiento y envío de alertas.	procesar datos		
<b>Telegram API</b> (Código en <i>bot_code.js</i> )	API de Telegram utilizada para enviar alertas al usuario cuando se detectan niveles de ruido superiores al umbral establecido.	Notificar al usuario sobre niveles altos de ruido	Condicional (según umbral)	Texto (JSON)

## Sensorización Utilizada

Por otra parte, la sensorización utilizada en el proyecto se basa en un sensor simulado de ruido ambiental. Este sensor tiene la función de medir los niveles de ruido en el entorno, expresados en decibelios (dB). Al ser un sistema de simulación, se desarrolló el script *simulacion\_microfono.py*, el cual emula el comportamiento de un sensor físico capaz de capturar la intensidad de sonido en un espacio determinado.

La captura de los datos de ruido se realiza de forma continua, permitiendo así un monitoreo constante y en tiempo real de los niveles de sonido. Esta configuración es ideal para ser aplicada en entornos como los de las ciudades en el país, donde es fundamental identificar variaciones de ruido que puedan ser indicadores de condiciones anormales o potencialmente perjudiciales para las personas.

## Detalles del Sensor Simulado

- En lugar de utilizar un sensor físico de ruido, se ha implementado una simulación en Python (*simulacion\_microfono.py*) que genera valores aleatorios en una escala de decibelios realista para representar el nivel de ruido ambiental. Estos datos simulan las lecturas de un sensor de ruido convencional, permitiendo evaluar el sistema sin la necesidad de hardware especializado.
- La simulación genera datos de forma continua, replicando la captura de un sensor en tiempo real. Esto permite que los datos sean enviados en intervalos regulares al broker MQTT, lo cual garantiza un flujo constante de información hacia el sistema de procesamiento en Node-RED.

## Ventajas de la Simulación

- **Flexibilidad:** Permite probar el sistema completo sin necesidad de sensores físicos, lo cual facilita el desarrollo y prueba en entornos de situaciones donde no se dispone de un sensor real.
- **Reproducibilidad:** Los datos simulados se pueden ajustar para representar distintos niveles de ruido, lo cual es útil para realizar pruebas de umbrales y validar el correcto funcionamiento del sistema de alertas.

- **Ajustabilidad:** Es posible modificar la frecuencia de generación de datos y el rango de decibelios en la simulación, lo cual es útil para evaluar la escalabilidad del sistema y la eficiencia en el procesamiento de datos.
- **Escalabilidad:** Gracias a que el sistema del programa trabaja con un bot integrado, esto hace el proyecto escalable a que pueda trabajar con los micrófonos de los teléfonos, lo cual hace indiferente la necesidad de componentes físicos que tengan que ser previamente instalados. Además, con este aspecto se podrían realizar estudios referentes a los niveles de ruido en todas las zonas de una ciudad y visualizar la información de manera clara con el dashboard.



## Herramientas de Visualización

### Node-RED Dashboard

Node-RED Dashboard es la herramienta principal utilizada para la visualización en tiempo real de los datos. Este dashboard se ha configurado para proporcionar una interfaz clara y accesible, donde se pueden observar las variaciones en los niveles de ruido mediante gráficos interactivos.

### Características del Dashboard

#### 1. Gráfico de Niveles de Ruido:

- Representa los niveles de ruido en tiempo real en un gráfico lineal.
- El eje X muestra la progresión temporal, mientras que el eje Y representa los niveles de ruido en decibelios (dB).
- Configurado con un rango de 30 a 120 dB para reflejar condiciones realistas.

#### 2. Alertas Visuales:

- El dashboard está diseñado para resaltar visualmente los valores de ruido que superan el umbral crítico (100 dB), indicando la necesidad de intervención inmediata.

#### 3. Interfaz Modular:

- Diseñado de manera modular, lo que permite agregar más elementos visuales o modificar la configuración de los gráficos.

#### 4. Acceso al Dashboard:

- Los usuarios pueden acceder al dashboard mediante un navegador web en la URL generada por Node-RED. El archivo *links\_node\_red.txt* contiene los enlaces directos para acceder tanto al flujo de Node-RED como al dashboard.

### Telegram API para Notificaciones

Además de la visualización directa, el sistema emplea un bot de Telegram para notificar a los usuarios cuando los niveles de ruido superan el umbral crítico. Las notificaciones incluyen información sobre el nivel detectado y permiten a los usuarios estar informados sin necesidad de supervisar constantemente el dashboard.

### Características de las Notificaciones

**1. Condicionalidad:**

- Las notificaciones se generan solo cuando el nivel de ruido supera los 100 dB, reduciendo las interrupciones innecesarias.

**2. Accesibilidad:**

- Al ser enviadas a través de Telegram, las notificaciones son fácilmente accesibles en dispositivos móviles, brindando información en tiempo real desde cualquier lugar.

**3. Formato:**

- *Figura 2: Formato Notificación Telegram Bot [Anexos]*

## **Análisis de Resultados**

A continuación, se detallan los principales hallazgos, el comportamiento del sistema y las implicaciones de los datos obtenidos.

### **Hallazgos**

#### **1. Distribución de Niveles de Ruido:**

- Los datos simulados reflejan un rango de niveles de ruido comprendido entre 30 dB y 120 dB, simulando escenarios reales como áreas residenciales tranquilas y zonas urbanas con tráfico o construcción.
- La mayor frecuencia de niveles de ruido se encuentra en el rango de 60 a 80 dB, lo cual representa escenarios urbanos moderados.

#### **2. Eventos de Ruido Crítico:**

- Se detectaron múltiples casos en los que los niveles de ruido superaron el umbral de 100 dB, activando las alertas del sistema. Esto refleja la capacidad del sistema para identificar condiciones críticas que requieren atención inmediata.

### **Desempeño del Sistema**

#### **1. Procesamiento en Tiempo Real:**

- La herramienta Node-RED ha sido fundamental en la recepción, procesamiento y visualización de los datos en tiempo real. La latencia entre la generación de datos y su visualización en el dashboard es clara e intuitiva, lo que asegura que los usuarios puedan monitorear los niveles de ruido sin retrasos significativos.
- Configuración de Componentes Node-RED: La respectiva configuración del nodo mqtt y el chart del dashboard.

**Editar nodo mqtt in**

Eliminar Cancelar Hecho

**Propiedades**

Servidor localhost:1883

Acción Suscribirse a un solo tema

Tema ciudad/ruido

CdS 2

Salida auto-detectar (objeto JSON, texto o buffer)

Nombre Nombre

☐ Habilitado

**Editar nodo chart**

Eliminar Cancelar Hecho

**Propiedades**

Group [Monitorización] Niveles de Rui

Size auto

Label Nivel de ruido en tiempo real

Type Line chart ☐ enlarge points

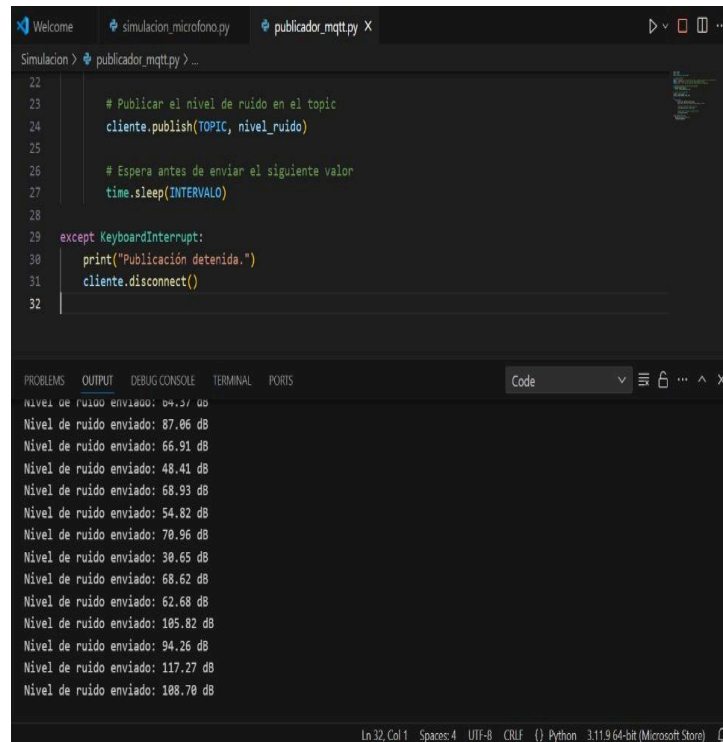
X-axis last 1 hours OR 1000 points

X-axis Label HH:mm:ss ☐ as UTC

Y-axis min 30 max 120

☐ Habilitado

- Ejecución del código de publicación mqtt y resultados en Node: Resultados del código de la publicación del mqtt en terminal y reflejados en la instancia de Node.

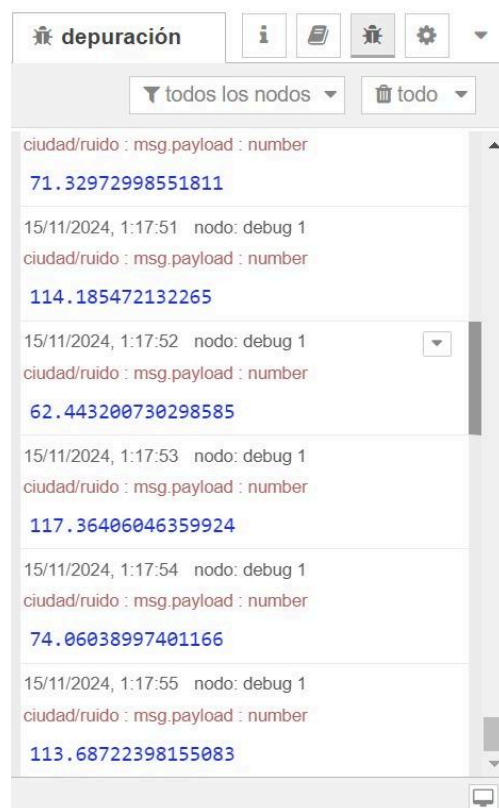


```
22
23 # Publicar el nivel de ruido en el topic
24 cliente.publish(TOPIC, nivel_ruido)
25
26 # Espera antes de enviar el siguiente valor
27 time.sleep(INTERVALO)
28
29 except KeyboardInterrupt:
30     print("Publicación detenida.")
31     cliente.disconnect()
32
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

nivel de ruido enviado: 64.37 dB  
Nivel de ruido enviado: 87.06 dB  
Nivel de ruido enviado: 66.91 dB  
Nivel de ruido enviado: 48.41 dB  
Nivel de ruido enviado: 68.93 dB  
Nivel de ruido enviado: 54.82 dB  
Nivel de ruido enviado: 70.96 dB  
Nivel de ruido enviado: 30.65 dB  
Nivel de ruido enviado: 68.62 dB  
Nivel de ruido enviado: 62.68 dB  
Nivel de ruido enviado: 105.82 dB  
Nivel de ruido enviado: 94.26 dB  
Nivel de ruido enviado: 117.27 dB  
Nivel de ruido enviado: 108.70 dB

Ln 32, Col 1 Spaces: 4 UTF-8 CRLF Python 3.11.9 64-bit (Microsoft Store)



## 2. Fiabilidad de las Alertas:

- El bot de Telegram ha enviado alertas precisas y oportunas en cada evento donde los niveles de ruido superaron el umbral crítico. Con lo cual, los

usuarios se aseguran el recibir información relevante sin sobrecargar el sistema con notificaciones innecesarias.

- Configuración de la función del Bot, conexión del http con la API de Telegram y creación y resultados reflejados del funcionamiento: Código respectivo del Bot, configuración de la solicitud http y proceso de creación del Bot junto a su prueba de funcionamiento.

**Editar nodo function**

Eliminar Cancelar Hecho

**Propiedades**

Nombre: Noise

Configuración Al inicio **En mensaje** Al final

```
1 const umbral = 100; // Define el umbral en decibelios
2 if (msg.payload > umbral) {
3   msg.payload = {
4     chat_id: "1589031873", // tu chat ID
5     text: "¡Alerta! Nivel de ruido alto detectado: " + msg.;
6   };
7   return msg;
8 } else {
9   return null;
10 }
11
```

Habilitado

**Editar nodo http request**

Eliminar Cancelar Hecho

**Propiedades**

Método: POST

URL: https://api.telegram.org/bot7579588832:AAHGUB:

☐ Habilitar conexión segura (SSL/TLS)

☐ Usar autenticación

☐ Habilitar conexión activa (keep-alive)

☐ Usar proxy

☐ Enviar solo respuestas que no sean 2xx al nodo Catch

☐ Deshabilitar el análisis estricto de HTTP

Devolver: una cadena UTF-8

Habilitado

← Devolver

una cadena UTF-8

▼

BotFather

2 144 053 usuarios al mes

Good, now let's choose a username for your bot. It must end in 'bot'. Like this, for example: TetrisBot or tetris\_bot.

01:34

noise\_bot

01:35

Sorry, this username is already taken. Please try something different.

01:35

noise\_bot\_TEC

01:35

Sorry, the username must end in 'bot'. E.g. 'Tetris\_bot' or 'Tetrisbot'

01:35

noiseTEC\_bot

01:35

Done! Congratulations on your new bot. You will find it at t.me/noiseTEC\_bot. You can now add a description, about section and profile picture for your bot, see /help for a list of commands. By the way, when you've finished creating your cool bot, ping our Bot Support if you want a better username for it. Just make sure the bot is fully operational before you do this.

Use this token to access the HTTP API:  
7579588832:AAHGUBaq5I\_wHbsUg2nKcYdaJpwyxNI18iA  
Keep your token secure and store it safely, it can be used by anyone to control your bot.

For a description of the Bot API, see this page:  
<https://core.telegram.org/bots/api>

01:35

Info. del bot

BotFather

2 144 053 usuarios al mes

BotFather is the one bot to rule them all. Use it to create new bot accounts and manage your existing bots.

Descripción

@BotFather

Nombre de usuario

Notificaciones

2 enlaces

Política de privacidad del bot

Reportar

Detener y bloquear el bot

15/11/2024, 2:07:28 nodo: debug 1

ciudad/ruido : msg.payload : number

58.66092009055896

15/11/2024, 2:07:29 nodo: debug 1

ciudad/ruido : msg.payload : number

62.99949089724383

15/11/2024, 2:07:30 nodo: debug 1

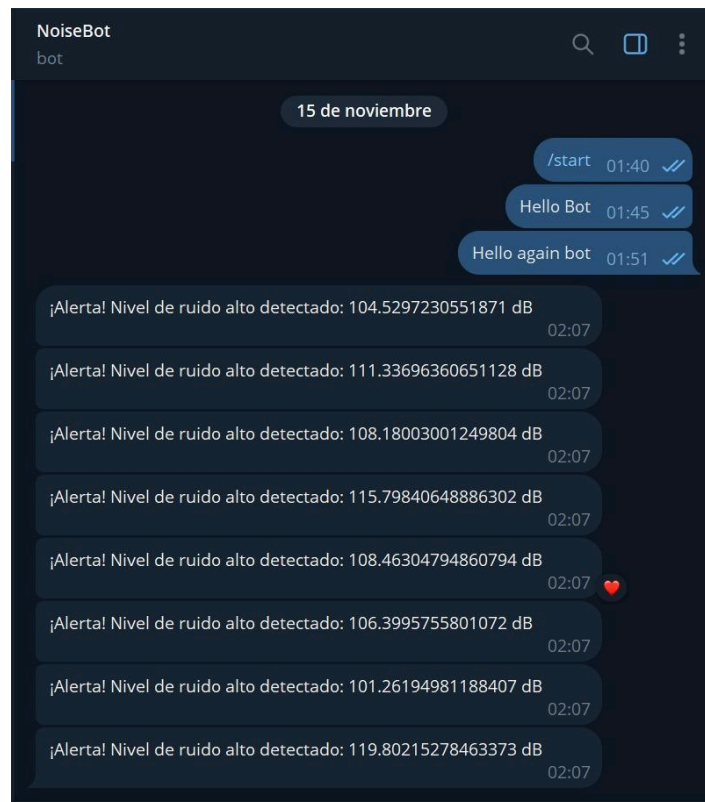
ciudad/ruido : msg.payload : number

55.25658384180301

NoiseBot

¡Alerta! Nivel de ruido detectado: 111.35

RESPONDER



## Impacto del Proyecto

### 1. Identificación de Puntos Críticos:

- A través de los datos visualizados en el dashboard y las alertas de Telegram, es posible identificar patrones recurrentes en los niveles de ruido, lo que facilita la priorización de áreas para la implementación de medidas correctivas.
- Datos del Dashboard: Vista del Dashboard en funcionamiento.



### 2. Escalabilidad del Sistema:

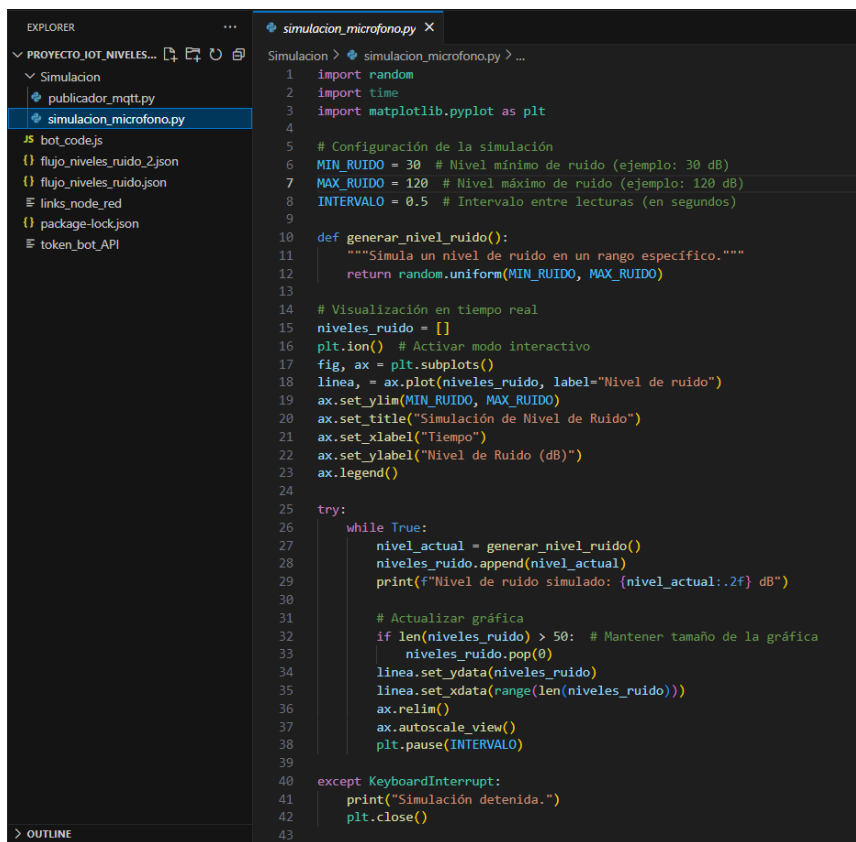


- El sistema está diseñado para ser escalable, permitiendo la incorporación de múltiples puntos de monitoreo con mínima configuración adicional, lo que amplía su potencial para analizar grandes áreas urbanas.

## Funcionamiento del sistema (código)

- Código de Simulación de Micrófono:

Este código simula la captación de niveles de ruido ambiental en tiempo real, emulando el funcionamiento de un micrófono. Utiliza la biblioteca *random* para generar valores aleatorios de decibelios dentro de un rango realista (30 a 120 dB), representando distintas condiciones acústicas. Estos valores se generan en intervalos definidos de 0.5 segundos, configurados mediante la constante INTERVALO. El script también emplea *matplotlib.pyplot* para visualizar los datos en un gráfico que se actualiza dinámicamente, permitiendo observar las fluctuaciones de los niveles de ruido de manera continua. Gracias a esto se consigue herramienta efectiva para probar y analizar sistemas de monitoreo de ruido sin necesidad de hardware físico, facilitando el desarrollo y la validación de aplicaciones IoT en entornos controlados.



```

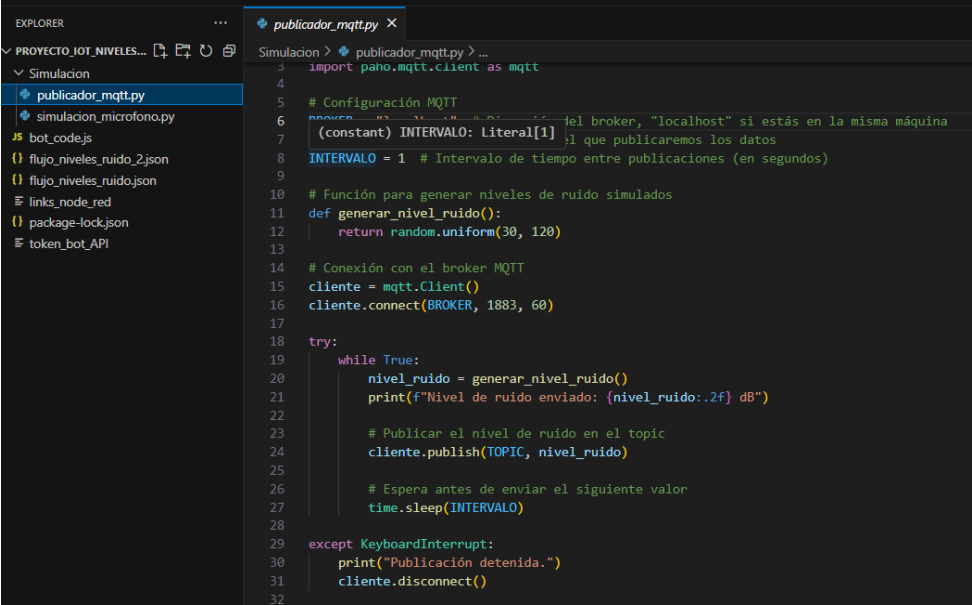
1  import random
2  import time
3  import matplotlib.pyplot as plt
4
5  # Configuración de la simulación
6  MIN_RUIDO = 30 # Nivel mínimo de ruido (ejemplo: 30 dB)
7  MAX_RUIDO = 120 # Nivel máximo de ruido (ejemplo: 120 dB)
8  INTERVALO = 0.5 # Intervalo entre lecturas (en segundos)
9
10 def generar_nivel_ruido():
11     """Simula un nivel de ruido en un rango específico."""
12     return random.uniform(MIN_RUIDO, MAX_RUIDO)
13
14 # Visualización en tiempo real
15 niveles_ruido = []
16 plt.ion() # Activar modo interactivo
17 fig, ax = plt.subplots()
18 linea, = ax.plot(niveles_ruido, label="Nivel de ruido")
19 ax.set_ylim(MIN_RUIDO, MAX_RUIDO)
20 ax.set_title("Simulación de Nivel de Ruido")
21 ax.set_xlabel("Tiempo")
22 ax.set_ylabel("Nivel de Ruido (dB)")
23 ax.legend()
24
25 try:
26     while True:
27         nivel_actual = generar_nivel_ruido()
28         niveles_ruido.append(nivel_actual)
29         print(f"Nivel de ruido simulado: {nivel_actual:.2f} dB")
30
31         # Actualizar gráfica
32         if len(niveles_ruido) > 50: # Mantener tamaño de la gráfica
33             niveles_ruido.pop(0)
34         linea.set_ydata(niveles_ruido)
35         linea.set_xdata(range(len(niveles_ruido)))
36         ax.relim()
37         ax.autoscale_view()
38         plt.pause(INTERVALO)
39
40 except KeyboardInterrupt:
41     print("Simulación detenida.")
42     plt.close()
43

```

- Código de Publicación en Node RED mqtt:

Por otra parte, el publicador lo que hace es simular un sensor de ruido ambiental utilizando la

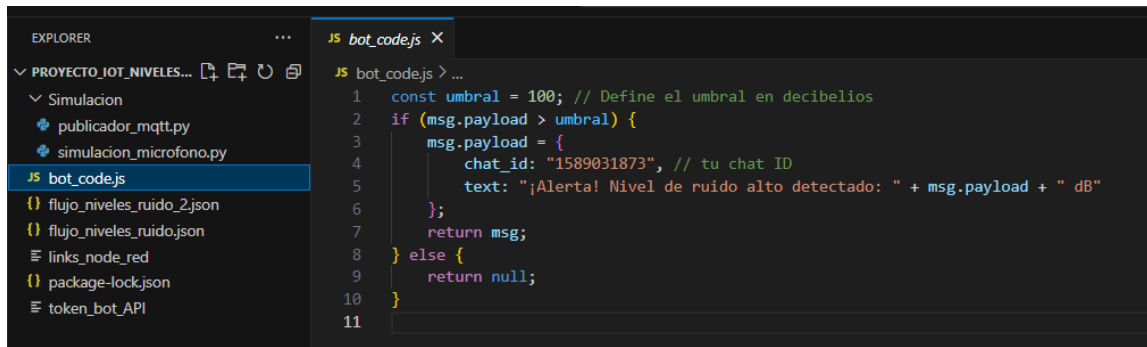
biblioteca *paho.mqtt.client* para publicar valores aleatorios de niveles de ruido en un broker MQTT local. Configura el broker, el puerto y el topic para la transmisión, definiendo un intervalo que regula la frecuencia de publicación de los datos. Mediante un bucle continuo, genera y publica los valores en el topic, mientras los imprime en la consola para monitoreo. En caso de interrupción, el programa finaliza limpiamente desconectándose del broker. Este es parte clave para probar sistemas IoT relacionados con niveles acústicos, eliminando una vez más la necesidad de hardware físico y permitiendo la integración con dashboards y otras aplicaciones de visualización.



```
1 import paho.mqtt.client as mqtt
2
3 # Configuración MQTT
4
5 # Constantes para configurar el broker, "localhost" si estás en la misma máquina
6 (constant) INTERVALO: Literal[1] # Intervalo de tiempo entre publicaciones (en segundos)
7
8 INTERVALO = 1 # Intervalo de tiempo entre publicaciones (en segundos)
9
10 # Función para generar niveles de ruido simulados
11 def generar_nivel_ruido():
12     return random.uniform(30, 120)
13
14 # Conexión con el broker MQTT
15 cliente = mqtt.Client()
16 cliente.connect(BROKER, 1883, 60)
17
18 try:
19     while True:
20         nivel_ruido = generar_nivel_ruido()
21         print(f"Nivel de ruido enviado: {nivel_ruido:.2f} dB")
22
23         # Publicar el nivel de ruido en el topic
24         cliente.publish(TOPIC, nivel_ruido)
25
26         # Espera antes de enviar el siguiente valor
27         time.sleep(INTERVALO)
28
29 except KeyboardInterrupt:
30     print("Publicación detenida.")
31     cliente.disconnect()
32
```

- Código Funcionamiento del Telegram Bot:

Este es el código que se encuentra dentro de la función de Node-RED y posteriormente se conecta con la solicitud http de la API de Telegram, permitiendo al bot enviar las notificaciones. En el código, se establece un umbral de 100 decibelios y, mediante una condicional, se evalúa si el nivel de ruido recibido (*msg.payload*) excede este valor. Si es así, se genera un mensaje de alerta y se incluye el ID del chat donde se enviará la notificación. En caso contrario, la función no devuelve nada. Este script es una parte esencial del sistema, ya que implementa la funcionalidad de alertas en tiempo real, permitiendo una interacción directa y rápida a través de Telegram para informar a los usuarios sobre condiciones acústicas que podrían requerir atención inmediata.



The image shows a screenshot of the Visual Studio Code editor. On the left, the Explorer sidebar displays a project structure for 'PROYECTO\_IOT\_NIVELES...'. It includes a 'Simulacion' folder containing 'publicador\_mqtt.py' and 'simulacion\_microfono.py', and a 'JS' folder with 'bot\_code.js' (which is selected). Other files in the 'JS' folder include 'flujo\_niveles\_ruido\_2.json', 'flujo\_niveles\_ruido.json', 'links\_node\_red', 'package-lock.json', and 'token\_bot\_API'. The main editor window shows the content of 'bot\_code.js', which contains a JavaScript function that checks if a message's payload exceeds a threshold of 100 decibels. If it does, it sends a chat message with a specific ID and a text alert.

```
JS bot_code.js X
JS bot_code.js > ...
1  const umbral = 100; // Define el umbral en decibelios
2  if (msg.payload > umbral) {
3      msg.payload = {
4          chat_id: "1589031873", // tu chat ID
5          text: "¡Alerta! Nivel de ruido alto detectado: " + msg.payload + " dB"
6      };
7      return msg;
8  } else {
9      return null;
10 }
11
```

## **Conclusiones**

En general, el proyecto ha permitido desarrollar un sistema funcional y escalable, capaz de recopilar, procesar y analizar datos acústicos en tiempo real. A través de la implementación de diferentes tecnologías, se logró cumplir con los objetivos propuestos y ofrecer una solución efectiva para monitorear la contaminación acústica en entornos tanto urbanos como rurales del país. A continuación se detallan las conclusiones del proyecto.

## **Logros Destacados**

### **1. Integración de Componentes IoT:**

- El uso de una arquitectura basada en Node-RED, MQTT y un bot de Telegram permitió integrar la captura de datos, su procesamiento y la notificación en tiempo real de eventos críticos, demostrando la capacidad de las tecnologías IoT para abordar problemas de la vida cotidiana de manera eficiente.

### **2. Simulación de Sensores Realistas:**

- La simulación de sensores de ruido en Python nos permitió analizar condiciones realistas de niveles acústicos en el país, con lo cual se pudo realizar pruebas exhaustivas del sistema sin necesidad de hardware físico.

### **3. Visualización y Notificaciones Efectivas:**

- La configuración del dashboard de Node-RED proporcionó una herramienta clara y visualmente intuitiva para monitorear los niveles de ruido en tiempo real. Asimismo, el bot de Telegram garantizó la entrega inmediata de alertas cuando los niveles superaban el umbral crítico.

### **4. Eficiencia del Sistema:**

- El tiempo de respuesta entre la detección de eventos críticos y la generación de notificaciones fue mínimo, demostrando la eficacia de la solución planteada para aplicaciones en tiempo real.

## **Dificultades y Solución**

### **1. Limitación de Hardware:**

- La ausencia de sensores físicos fue resuelta mediante la simulación de niveles de ruido en Python, lo que permitió desarrollar y validar el sistema en su totalidad.

### **2. Configuración de Node-RED y MQTT:**

- Inicialmente, se presentaron desafíos en la configuración de las herramientas. Sin embargo, mediante documentación adecuada y pruebas iterativas, se logró establecer una configuración estable y funcional.

## **Posibles Mejoras Futuras**

### **1. Integración de Sensores Físicos:**

- Implementar sensores reales de ruido en futuras iteraciones para validar el sistema en un entorno práctico. Esto también se podría llegar a trabajar con los micrófonos de los dispositivos móviles de los usuarios.

### **2. Almacenamiento de Datos:**

- Añadir una base de datos para almacenar datos históricos de ruido y permitir análisis más detallados de patrones a largo plazo.

### **3. Optimización del Dashboard:**

- Incluir nuevas métricas y visualizaciones para mejorar la usabilidad y la capacidad de interpretación de datos por parte de los usuarios.

## **Bibliografía**

Eclipse Foundation. (n.d.). *MQTT - The Standard for IoT Messaging*. Recuperado de <https://mqtt.org>

Node-RED. (n.d.). *Node-RED: Flow-based programming for the Internet of Things*. Recuperado de <https://nodered.org>

Mosquitto. (n.d.). *Eclipse Mosquitto: An open source MQTT broker*. Recuperado de <https://mosquitto.org>

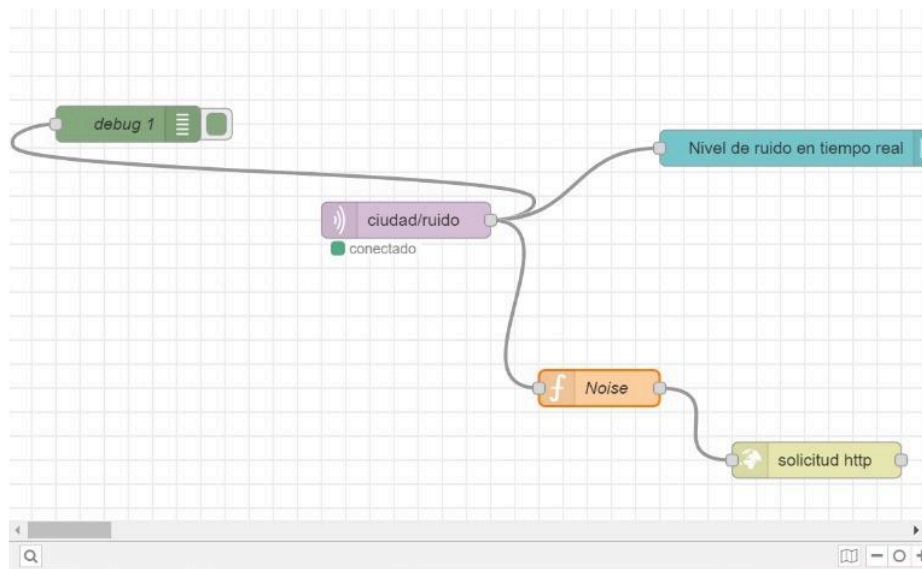
Telegram. (n.d.). *Telegram Bot API*. Recuperado de <https://core.telegram.org/bots/api>

Dashboards in Node-RED. (n.d.). *Node-RED Dashboard Documentation*. Recuperado de <https://flows.nodered.org/node/node-red-dashboard>

## Anexos

### NOTA:

Aclaración que en la carpeta del código viene un readme incluido para poder replicar el proceso del proyecto. Se encuentran los links a las diferentes herramientas de trabajo de Node, el archivo de flujo de niveles de ruido, el cual debe ser importado en el Node-RED, el código que debe ir dentro de la función del bot (denominada “Noise”), y también el token del bot para hacer la solicitud http en caso de ser necesaria.



*Figura 1: Diagrama de Arquitectura IoT*

```
text: "¡Alerta! Nivel de ruido alto detectado: "
```

*Figura 2: Formato Notificación Telegram Bot*