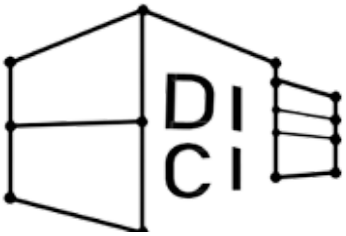
# Universidad de Tarapacá

# Facultad de Ingeniería

Departamento de Ingeniería en Computación e Informática




## Plan de Proyecto

## Detector de Ruidos Molestos “HUSH”


Autor(es): Iván Collao
Sebastian Eyraud
Guillermo Pino
Giorgio Rojas

Asignatura: Proyecto 2

Profesor: Diego Aracena

Fecha: 23 / 12 / 2025

**23 DE DICIEMBRE DE 2025**

## Historial de cambios

| Fecha | Versión | Descripción | Autor(es) |
|---|---|---|---|
| 07/10/2025 | 1.0 | Inicio del informe | Guillermo Pino |
| 14/10/2025 | 1.1 | Actualización del Informe | Iván Collao<br>Sebastian Eyraud<br>Guillermo Pino<br>Giorgio Rojas |
| 21/10/2025 | 1.2 | Finalización de la parte 1 del informe | Iván Collao<br>Sebastian Eyraud<br>Guillermo Pino<br>Giorgio Rojas |
| 28/10/2025 | 2.0 | Finalización de la parte 2 del informe | Iván Collao<br>Sebastian Eyraud<br>Guillermo Pino<br>Giorgio Rojas |
| 04/11/2025 | 2.1 | Inicio de la segunda fase de entrega del informe | Guillermo Pino<br>Giorgio Rojas |
| 18/11/2025 | 2.2 | Desarrollo de los casos de uso y diagramas de secuencia | Giorgio Rojas<br>Guillermo Pino |
| 25/11/2025 | 2.3 | Desarrollo de la arquitectura, herramientas y técnicas | Iván Collao<br>Sebastian Eyraud |
| 02/12/2025 | 2.4 | Desarrollo de los puntos de la última etapa del proyecto | Guillermo Pino |

# Índice

# Índice de tablas

# Panorama general

## Introducción

Este proyecto nace de la necesidad de ayudar a combatir la contaminación acústica, un problema creciente que hoy pone en riesgo el bienestar y el entorno. El objetivo es detectar y alertar sobre emisiones sonoras perjudiciales dentro de un cierto rango en el entorno o población. El sistema diseñado incorpora sensores de sonido, como micrófonos capaces de identificar niveles de decibelios superiores a los umbrales establecidos como aceptables. Una vez detectado el evento acústico, se activa un mecanismo que activará una “alarma” que alertará sobre algún dispositivo que haya sobrepasado un rango establecido.

## Propósito

El propósito del dispositivo es alertar al usuario en situaciones donde se excedan los límites de ruido tolerables. De este modo, se busca fomentar el respeto por el entorno sonoro compartido, promoviendo una convivencia más armoniosa en zonas habitadas en la ciudad.

## Alcance

El dispositivo utiliza sensores de sonido como un micrófono y un panel RGB además de la implementación de una Raspberry Pi 4 y el desarrollo de un sistema operativo de interfaz gráfica destinado al usuario.

## Objetivo

### Objetivo general

Desarrollar e implementar un dispositivo automatizado que detecta y alerta sobre ruidos molestos como música en excesivo volumen en áreas urbanas de alto índice de población.

### Objetivos específicos

- Estudiar Raspberry Pi y sensores de sonido y transceptores de RF (BLE).

- Diseñar y construir una maqueta funcional del dispositivo.
- Desarrollar un sistema de respuesta automatizada, que active la “alarma” sólo cuando se detecten niveles altos de decibeles (como música).
- Documentar el proceso de instalación, configuración y pruebas.
- Evaluar la efectividad del dispositivo en la reducción de la contaminación acústica.

## Suposiciones y restricciones

- Los materiales y herramientas de trabajo estarán disponibles dentro de la duración del proyecto.
- Tiempo: El proyecto se debe terminar antes de la fecha límite, la cual corresponde al 24 de diciembre de 2025.
- Hardware: El proyecto debe incluir a la placa Raspberry Pi 4 como su pieza de hardware principal para controlar actuadores y sensores.
- Modelo Virtual: El proyecto debe ser representado y mostrado a través de un dispositivo de realidad virtual (Meta Quest 3).
- Presupuesto: Los materiales para el proyecto incluyendo una tarjeta MicroSD y recursos para la impresión 3D de la carcasa los cuales no deben sobrepasar el presupuesto de 20.000$ CLP.

## Entregables del proyecto

Producto funcionando con Raspberry Pi 4 y software desarrollado.

Manual de usuario.

Bitácoras de cada semana (12 en total).

Informes de avance.

Presentación de proyecto.

Informe final.

# Organización del proyecto

## Personal y entidades internas

| Rol | Encargado | Involucrado(s) |
|---|---|---|
| Jefe de Proyecto | Giorgio Rojas | Giorgio Rojas |
| Programador | Sebastian Eyraud | Sebastian Eyraud<br>Giorgio Rojas<br>Ivan Collao<br>Guillermo Pino |
| Documentador | Guillermo Pino | Guillermo Pino<br>Sebastian Eyraud |
| Analista Programador | Giorgio Rojas | Giorgio Rojas |
| Diseñador | Iván Collao | Iván Collao |

Tabla 1: Personal y entidades internas

## Mecanismos de Comunicación

- Grupo de WhatsApp
- Grupo de Discord

# Planificación de los Procesos de Gestión

## Planificación de estimaciones

### Hardware

| Producto | Costo | Cantidad | Costo Total |
|---|---|---|---|
| Raspberry Pi 4 | $ 91.990 | 1 | $ 91.990 |
| Módulo Sensor de Sonido y Voz FC-04 | $ 2.890 | 1 | $ 2.890 |
| MicroSD | $ 12.000 | 1 | $ 12.000 |

| Producto | Costo | Cantidad | Costo Total |
|---|---|---|---|
| Monitor (LG) | $ 119.990 | 1 | $ 119.990 |
| Teclado | $ 8.990 | 1 | $ 8.990 |
| Mouse | $ 5.990 | 1 | $ 5.990 |
| PC1 (Acer) | $ 649.990 | 1 | $ 649.990 |
| PC2 (HP) | $ 700.000 | 1 | $ 700.000 |
| PC3 (Lenovo) | $ 549.990 | 1 | $ 549.990 |
| PC4 (Asus) | $ 499.990 | 1 | $ 499.990 |
| Total: | $ 2.641.820 | - | $ 2.641.820 |

Tabla 2: Hardware

**Software**

| Producto | Costo |
|---|---|
| Raspberry Pi OS | $ 0 |
| Google Docs | $ 0 |
| Visual Studio Code | $ 0 |
| Canva (free) | $ 0 |
| Python | $ 0 |
| Redmine | $ 0 |
| Arch Linux | $ 0 |
| Total: | $ 0 |

Tabla 3: Software

### Planificación de Recursos Humanos

| Integrantes | Rol | Hora Total | Sueldo/Hora | Sueldo Total |
|---|---|---|---|---|
| Giorgio Rojas | Jefe de Proyecto<br>Analista<br>Programador | 80 | $ 7.925 | $ 634.000 |
| Iván Collao | Diseñador | 80 | $ 7.360 | $ 588.800 |
| Sebastian Eyraud | Programador | 80 | $ 7.562 | $ 605.000 |
| Guillermo Pino | Documentador | 80 | $ 6.302 | $ 504.200 |
| Total: | | | | $ 2.332.000 |

Tabla 4: Planificación de Recursos Humanos

### Planificación de Costo Total

| Tipo de Costo | Costo |
|---|---|
| Costo de Hardware | $ 2.506.850 |
| Costo de Software | $ 0 |
| Costo de Recursos Humanos | $ 2.332.000 |
| Total: | $ 4.838.850 |

Tabla 5: Planificación de Costo Total

## Lista de actividades

### Carta Gantt (diciembre 2025)

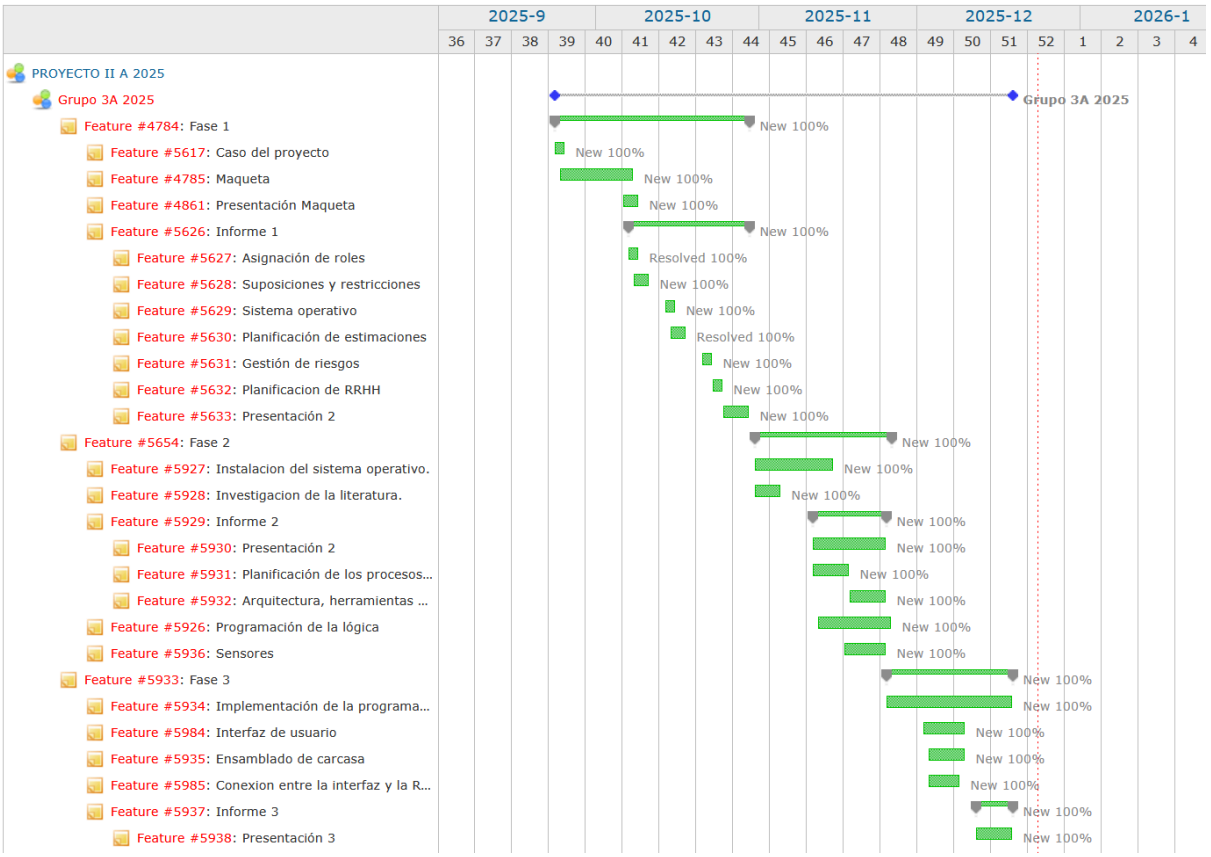



Tabla 6: Carta Gantt

### Actividades de trabajo

| Actividad | Descripción | Responsable |
|---|---|---|
| Bitacoras | Registro del avance de las actividades semanalmente. | Guillermo Pino |
| Organización | Planificación del proyecto, definición de roles y responsabilidades | Giorgio Rojas |
| Establecer Problemática | Identificación y definición clara del problema a | Sebastian Eyraud |

| Actividad | Descripción | Responsable |
|---|---|---|
| | resolver, análisis del contexto, establecimiento de objetivos | |
| Analizar distintas soluciones | Investigación y evaluación de alternativas disponibles | Sebastián Eyraud<br>Ivan Collao<br>Giorgio Rojas<br>Guillermo Pino |
| Creación del Modelo 3D | Diseño digital de la estructura y carcasa del prototipo utilizando software (Godot) | Sebastián Eyraud<br>Ivan Collao<br>Giorgio Rojas<br>Guillermo Pino |
| Presentación | Exposición formal del proyecto ante evaluadores, demostración del funcionamiento del prototipo | Sebastián Eyraud<br>Ivan Collao<br>Giorgio Rojas<br>Guillermo Pino |
| Informe 1 | Crear el informe para la fase 1 del proyecto | Sebastián Eyraud<br>Ivan Collao<br>Giorgio Rojas<br>Guillermo Pino |

Tabla 7: Actividades de trabajo

### Planificación de la gestión de riesgos

| Tipo de Riesgos | Descripción |
|---|---|
| Humanos | Bajo rendimiento, conflictos o alta rotación del equipo debido a baja moral, problemas de salud o malas relaciones. Dificultad para atraer o retener el talento necesario |
| Tecnológicos | Fallas, incompatibilidades o deficiencias en el hardware y software que provocan retrasos en el cronograma y las |

| Tipo de Riesgos | Descripción |
|---|---|
| | entregas. |
| Organización | Falta de dirección estratégica clara y comunicación deficiente por parte de la gerencia, lo que fomenta la incertidumbre y la desinformación en la organización. |
| Equipamiento | Equipo de trabajo inadecuado, obsoleto o insuficiente que merman la productividad y generan resistencia por parte del equipo. |
| Estimación | Subestimación del esfuerzo requerido para el desarrollo y la corrección de defectos, lo que resulta en el incumplimiento sistemático de los plazos acordados. |
| Requerimientos | Alta volatilidad o definición ambigua de los requerimientos, generando reprocesos constantes e insatisfacción del cliente. |

Tabla 8: Planificación de la gestión de riesgos

| Nivel de Impacto | Descripción |
|---|---|
| Inaceptable | Impacto determinante que puede amenazar la continuidad o el logro del proyecto. |
| Grave | Impacto significativo que necesita recursos extra para ser administrado, pero el proyecto tiene la posibilidad de seguir. |
| Leve | Efecto ligero que tiene la posibilidad de demorar algunos elementos del proyecto, pero no afecta de manera significativa los resultados. |
| Sin Importancia | Un impacto mínimo que no necesita una reacción rápida y que no tendrá un efecto significativo en el avance del proyecto. |

Tabla 9: Niveles de impacto

| Probabilidad de Ocurrencia | Probabilidad de Ocurrencia |
|---|---|
| Alta | 71% - 100% |
| Media | 31% - 70% |
| Baja | 0% - 30% |

Tabla 10: Probabilidades de ocurrencia

| Riesgo | Tipo | Probabilidad de Ocurrencia | Nivel de Impacto | Acción Remedial |
|---|---|---|---|---|
| Integrante del equipo | Humano | 25% | Leve | Redistribuir tareas entre miembros disponibles |

| Riesgo | Tipo | Probabilidad de Ocurrencia | Nivel de Impacto | Acción Remedial |
|---|---|---|---|---|
| enfermo | | | | |
| Falla en microSD | Tecnológico | 30% | Grave | Implementar respaldos en múltiples tarjetas y verificar integridad periódicamente |
| Falta de respaldos | Tecnológico | 45% | Inaceptable | Realizar verificación semanal para verificar que se tienen múltiples respaldos |
| Componente defectuoso | Tecnológico | 35% | Leve | Mantener stock de componentes de repuesto, probar componentes al recibirlos e identificar proveedores alternativos |
| Falta de comunicación | Organización | 40% | Grave | Realizar reuniones diarias de sincronización; |
| Retrasos con los avances | Estimación | 35% | Leve | Revisiones semanales de progreso y hacer ajustes al plan |
| Falla en los sensores | Tecnológico | 30% | Leve | Adquirir sensores de respaldo |
| Recursos de Hardware Incompatibles | Equipamiento | 40% | Grave | Verificar compatibilidad antes de compra, mantener lista de hardware certificado |

| Riesgo | Tipo | Probabilidad de Ocurrencia | Nivel de Impacto | Acción Remedial |
|---|---|---|---|---|
| Errores en el código | Tecnológico | 50% | Leve | Implementar testing y revisiones de código |
| Dificultad para la obtención de sensores específicos | Tecnológico | 20% | Leve | Identificar proveedores alternativos y considerar sensores equivalentes |
| Cambio en los requerimientos | Requerimiento | 25% | Baja | Implementar control de cambios y evaluar impacto antes de aceptar. |
| Enchufes defectuosos | Tecnológico | 55% | Media | Usar fuentes de alimentación distintas |

Tabla 11: Riesgo y acción

## Planificación de los Procesos Técnicos

A continuación se describen el modelo de procesos, modelo de diseño, arquitectura y las herramientas y técnicas de desarrollo del proyecto.

### Modelo de procesos

Como modelo de procesos se describirán los requerimientos técnicos que el proyecto debe cumplir.

### Requerimientos

A continuación se definirán los requisitos funcionales y no funcionales del proyecto Hush que son cruciales para el buen funcionamiento del sistema desarrollado tanto de funcionalidad como de calidad esperada.

#### Requerimientos funcionales

- Reconocer sonidos en niveles altos de decibeles.
- Emitir una notificación al usuario en tiempo real (alertas).

- Permite al usuario una configuración de parámetros (establecer rangos de decibeles para la alerta de sonidos en la aplicación móvil).

**Requerimientos no funcionales**

- Interfaz intuitiva de fácil uso para el usuario.
- Fácil mantenibilidad para actualizar y corregir el sistema.
- Indicador del estado mediante un indicador visual (LED) que muestra el estado operativo.

## Modelo de diseño (caso de uso general)

Se ha modelado el siguiente diagrama para representar el funcionamiento del sistema:


El modelo contará con los siguientes actores para su correcto funcionamiento

- **Usuario:** Es quien puede gestionar desde la aplicación el límite de decibeles permitidos para usar.
- **Sensor de sonido:** Micrófono Grove el cual percibe las señales para analizar en el sistema.
- **Sensor Led:** enciende y apaga dependiendo del encendido o apagado del sistema.
- **Interfaz gráfica de usuario:** Aquí interactúa el usuario para ver la información y gestionar los parámetros.

### Casos de uso y diagrama de secuencia

#### Detección de ruido

| |
|---|
| **Nombre de caso de Uso**: Detección de ruido |
| **Resumen**:<br>El sistema recibirá información cada vez que el sensor de sonido registre un ruido que haya superado el límite predefinido. |
| **Actor(es)**: Sistema y sensor de sonido. |
| **Precondición**:<br>El sistema debe estar encendido y a la espera de que se produzca algún ruido. |
| **Descripción**:<br>1. El sensor de sonido detecta un valor de ruido ambiente.<br>2. El sistema compara el valor obtenido con el umbral definido por el usuario.<br>3. Si el nivel de ruido supera el umbral, el sensor entrega el valor al sistema.<br>4. El sistema recibe y valida la señal, confirmando que el límite de ruido fue excedido.<br>5. El sistema activa el LED indicador para señalar localmente la detección. |
| **Alternativa**:<br>3.1. Si el nivel de ruido detectado no supera el umbral, el sistema continúa en estado de espera sin activar el LED ni enviar notificaciones. |
| **Postcondición**:<br>El sistema queda listo para continuar monitoreando el nivel de ruido y activar nuevas alertas cuando corresponda. |

Tabla 12: Detección de ruido

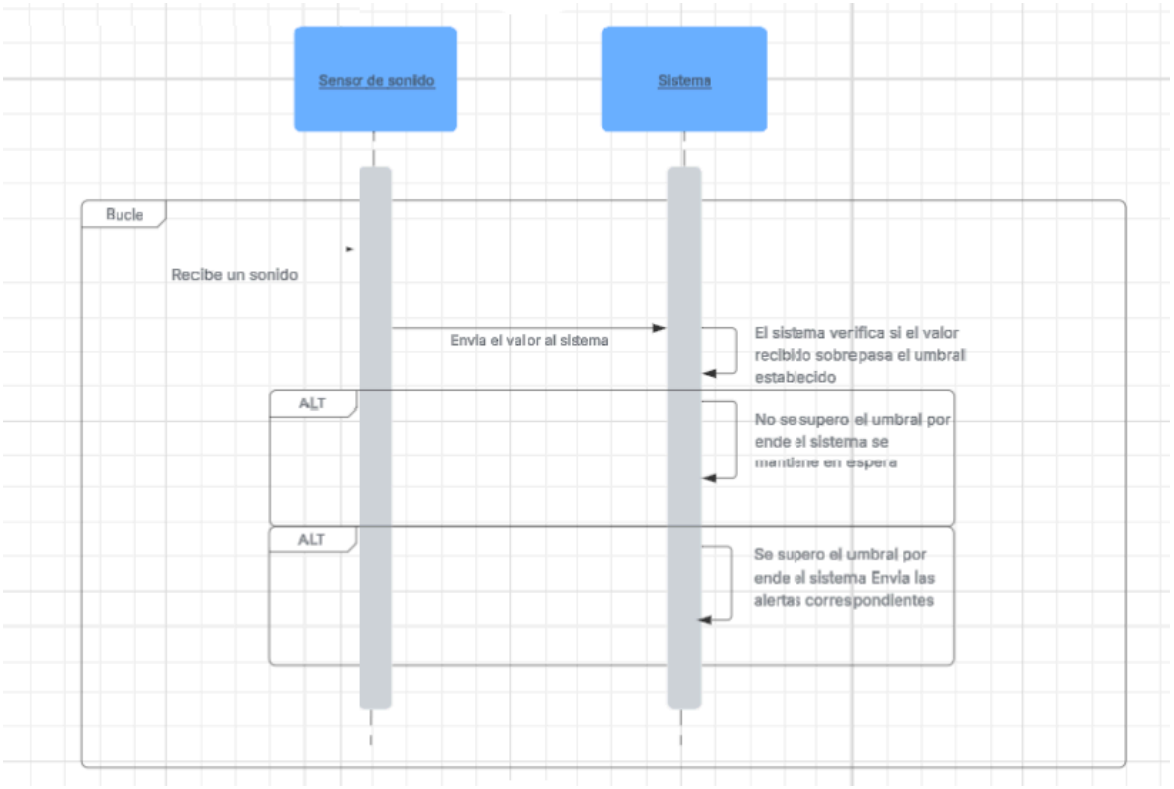



**Diagrama 1.** Diagrama de secuencia de la detección de ruido.

**Notificar al usuario**

| |
|---|
| **Nombre de caso de Uso**: Notificar al usuario |
| **Resumen:**<br>El sistema, utilizando la Raspberry Pi 4B y un micrófono, detecta un nivel de sonido que excede un umbral establecido y envía una notificación de alerta al usuario por su canal preferido (e.g., email, mensaje push, Telegram). |
| **Actor(es)**: Sistema, usuario, sensor LED, Teléfono |
| **Precondición**:<br>El sistema debe estar encendido y ejecutando el software, el micrófono debe estar conectado y funcionando correctamente. El umbral de ruido (dB) ha sido configurado. La configuración de notificación del usuario (canal y destino) es válida y accesible |
| **Descripción**:<br>1. Al detectar un nivel de dB que supera el umbral definido y recibido el valor entregado por el sensor se inicia el evento de notificar al usuario. |

| |
|---|
| 2. El sistema genera un mensaje de alerta y envía una notificación al usuario y muestra un mensaje en el sensor led. |
| **Alternativa**:<br>2.1. Falla al enviar la notificación. Si el servicio de notificación no responde o falla el envío. |
| **Postcondición**: El usuario ha recibido una notificación de la alerta de ruido |

Tabla 13: Notificar al usuario

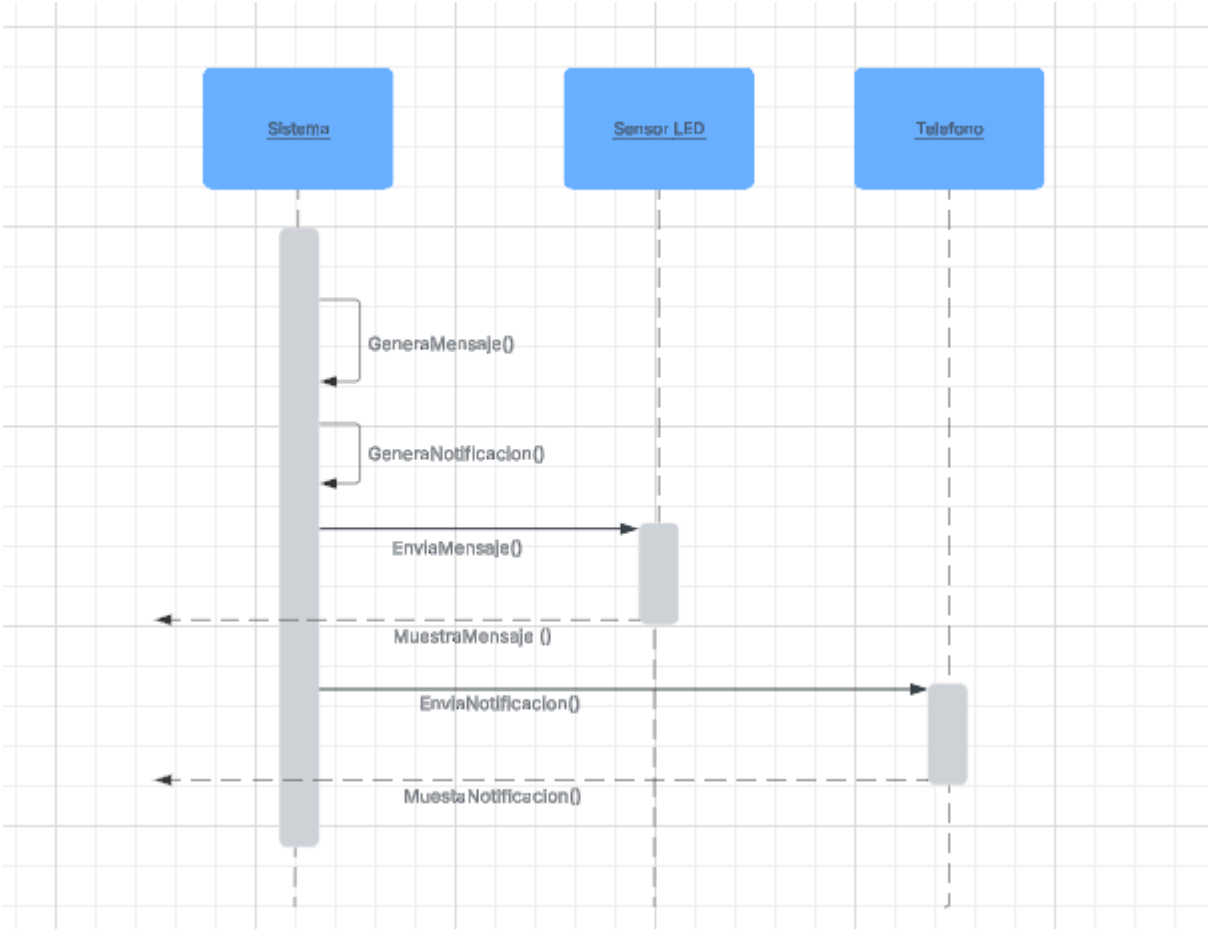



**Diagrama 2.** Diagrama de secuencia de la notificación al usuario.

**Encender y apagar el sistema**

| |
|---|
| **Nombre de caso de Uso**: Encender y apagar el sistema |
| **Resumen:** El usuario puede encender o apagar el sistema mediante un botón físico o interruptor conectado a la Raspberry Pi. El encendido inicializa los componentes necesarios para el monitoreo del sensor de sonido, mientras que el apagado detiene todos los procesos y coloca al sistema en un estado seguro |
| **Actor(es)**: Sistema, usuario. |
| **Precondición**:<br>El sistema debe contar con una batería o pila instalada y con suficiente carga.<br>El usuario debe interactuar con los botones de la interfaz gráfica. |
| **Descripción**:<br>1. El usuario presiona el botón físico de encendido/apagado del dispositivo.<br>2. El sistema detecta la señal manual enviada por el usuario.<br>3. Si la señal corresponde a encender, el sistema inicializa los módulos principales, incluyendo el sensor de sonido, la interfaz de notificaciones y el LED indicador.<br>4. Si la señal corresponde a apagar, el sistema detiene de forma segura todas las funciones activas, apaga el LED e ingresa en modo de bajo consumo para preservar la batería.<br>5. El sistema verifica que cada componente haya terminado su proceso de inicialización o apagado correctamente. |
| **Alternativa**:<br>**2.1** Si el usuario presiona el botón cuando la batería se encuentra con carga insuficiente, el sistema no inicia e informa la condición mediante un parpadeo breve del LED. |
| **Postcondición**: El usuario ha recibido una notificación de la alerta de ruido |

Tabla 14: Encender y apagar el sistema

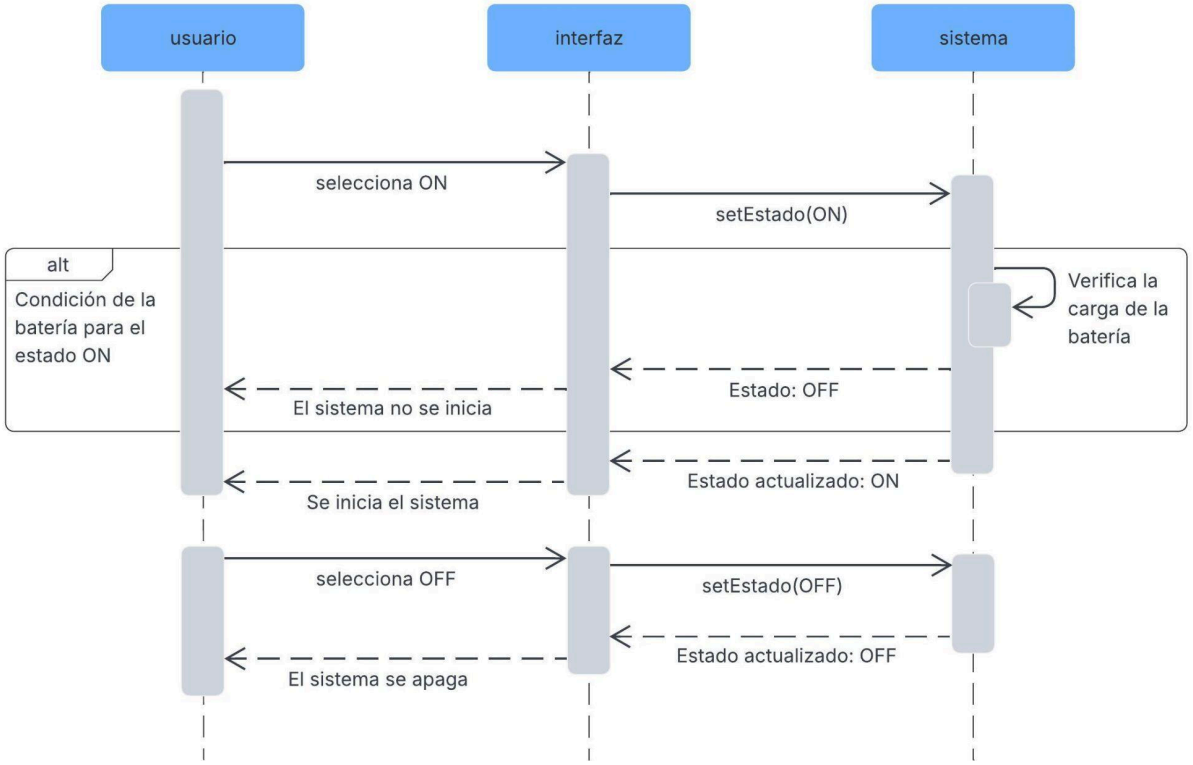



**Diagrama 3.** Diagrama de secuencia de encender y apagar el sistema.

## Descripción de la arquitectura

La arquitectura del sistema se basa en un modelo cliente-servidor embebido, donde la Raspberry Pi 4 actúa como servidor central encargado del procesamiento, coordinación y ejecución de las funciones críticas del proyecto. Los sensores y actuadores conectados funcionan como clientes que envían datos o ejecutan acciones según las instrucciones del sistema.

El servidor (Raspberry Pi 4) recibe continuamente la información entregada por el sensor de sonido mediante comunicación GPIO o I2C, interpreta los niveles de ruido y determina si se supera el umbral establecido. Tras el análisis, el servidor coordina la respuesta mediante la activación de un LED indicador o el envío de una notificación al usuario a través de los canales previamente configurados (como mensajería instantánea o correo electrónico).

La arquitectura cuenta con tres capas principales:

1. **Capa de adquisición de datos**
    - Compuesta por el sensor de sonido FC-04. (Sound Sensor Groove v1.6)

- Captura niveles de ruido en tiempo real y envía los datos a la Raspberry Pi mediante señales digitales o analógicas procesadas por librerías de Python.

2. **Capa de procesamiento y lógica**

   - Ejecutada en la Raspberry Pi 4 usando Python. Se encarga de:
     - Interpretar valores del sensor.
     - Comparar con umbrales definidos.
     - Gestionar el encendido y apagado del sistema.
     - Controlar el LED indicador.
     - Preparar y enviar notificaciones al usuario.

3. **Capa de notificación e interacción**

   - Informa al usuario mediante:
     - LED indicador conectado a GPIO.
     - Notificaciones enviadas por la red (alertas push).

   - Permite que el usuario configure los parámetros del sistema (umbrales de ruido y ajustes de sensibilidad).

Esta arquitectura modular facilita la escalabilidad, permitiendo incorporar sensores adicionales o nuevas acciones de respuesta sin modificar la estructura principal del sistema.

## Herramientas y técnicas

### Herramientas de Software

- **Raspberry Pi OS:** Sistema operativo principal donde se ejecuta el programa de monitoreo y control.
- **Python 3:** Lenguaje principal de programación para la comunicación con sensores, manejo de GPIO y envío de notificaciones.
- **Librerías de Python:**
  - gpiozero/rpi.GPIO: Control de pines GPIO para el LED y el botón.
  - smbus/I2C: Comunicación con componentes conectados mediante buses digitales.

- **Thonny IDE:** Entorno de desarrollo utilizado para programar, depurar y ejecutar el código en la Raspberry Pi.
- **Visual Studio Code:** Utilizado para la edición del código y manejo de archivos del proyecto.
- **Redmine / Google Docs:** Herramientas de planificación, colaboración y registro del progreso semanal.

**Herramientas de Hardware**

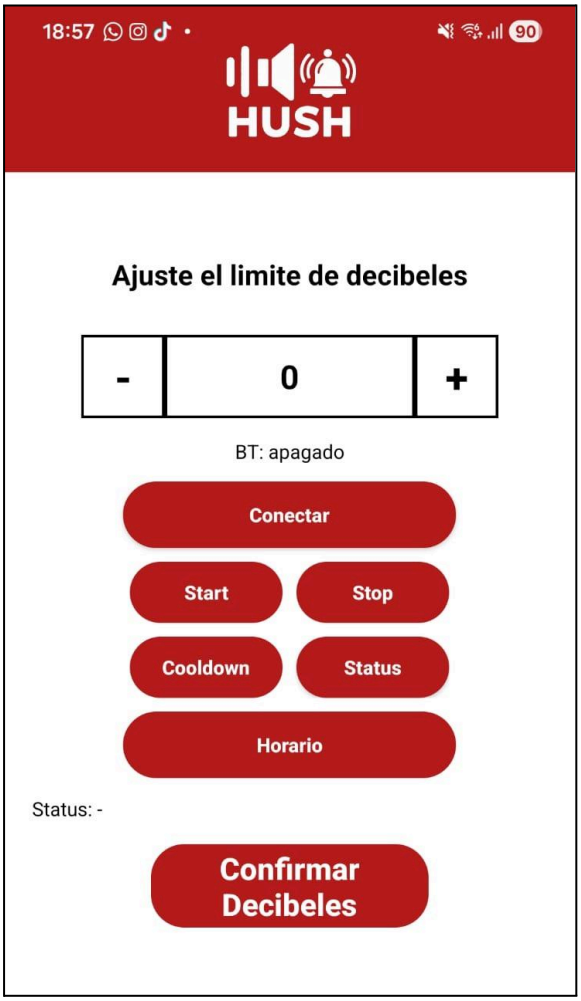
- **Raspberry Pi 4:** Unidad principal de procesamiento y comunicación con sensores.
- **Sensor de sonido FC-04:** Responsable de medir niveles de ruido en dB.
- **LED indicador:** Elemento de señalización visual cuando se detectan niveles de ruido superiores al umbral.
- **Grove button ON/OFF:** Permite al usuario encender o apagar el sistema manualmente.
- **Tarjeta MicroSD:** Almacenamiento del sistema operativo y del software del proyecto.

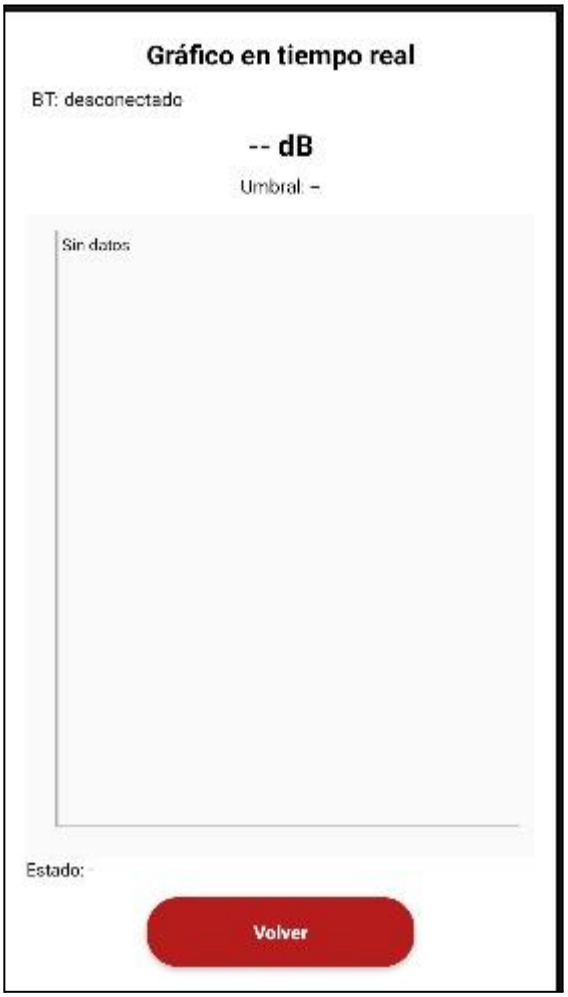
**Técnicas utilizadas**

- **Programación modular en Python:** Separación de las funciones del sistema para mejorar la mantenibilidad
- **Pruebas incrementales:** Evaluación por etapas del funcionamiento del sensor, LED, notificaciones y la lógica de control.
- **Metodología incremental:** Desarrollo por iteraciones, avanzando cada semana según bitácoras del proyecto.
- **Modelado en VR (Meta Quest 3):** Construcción de una representación virtual del prototipo para validar el diseño.

## Interfaz de usuario

La ventana principal del sistema Hush es intuitiva y simple de usar, permite ajustar el límite de decibeles percibidos, tiene dos botones, el botón más '+' y menos '-' ambos permiten aumentar o disminuir el límite de decibeles respectivamente, una vez al aceptar los cambios desde el botón 'Confirmar' el sistema lo reconoce y desde ese nivel establecido se generan automáticamente alertas destinada al usuario.



# Planificación de procesos de soporte

## Documentación

### Manual de usuario

Documento dirigido al usuario, le permite saber cómo usar el software y los cuidados del producto, también previene los errores y solución ante las fallas que puedan surgir como también contactos de soporte.

### Documentación del código

Este documento es útil para informar, demostrar y explicar las funcionalidades del sistema creado, permiten conocer las implementaciones usadas como también ayuda a posteriores actualizaciones en el futuro.

### Wiki del proyecto

Se utilizó una plataforma web diseñada para centralizar la documentación y el seguimiento del desarrollo del proyecto en tiempo real. En ella se registran y presentan los avances de forma continua, abarcando todo el proceso desde la fase inicial hasta su conclusión.

# Problemas encontrados

## Sistema operativo y librería Grove Pi

Durante el proceso de instalación del sistema operativo Linux en el equipo Raspberry Pi se presentaron diversas dificultades relacionadas con la compatibilidad de controladores y la configuración inicial del entorno. Estos inconvenientes retrasaron el proceso de desarrollo del sistema, ya que fue necesario realizar cambios del sistema operativo.

### Solución

Se instaló el sistema operativo nativo de la Raspberry Pi 4B, conocido como Raspberry OS. Esta plataforma ofrece una mejor compatibilidad con la librería GrovePi, permitiendo una integración más estable con los sensores Grove.

### Encendido de la Raspberry

El dispositivo Raspberry Pi 4B presentó fallas al intentar encenderse debido a problemas con el cargador utilizado. El mal funcionamiento del cargador ocasionó daños en una de las entradas del puerto USB, lo que impidió el arranque normal del equipo y detuvo el avance del proyecto.

#### Solución

Se trabajó directamente con otro dispositivo Raspberry Pi 4b.

## Implementación

Consiste en la fase de ejecución e integración del software y hardware del proyecto, en ella se describen los módulos programados que son esenciales para que el sistema cumpla con las funcionalidades y los requisitos técnicos que abordan la problemática propuesta.

### Plan de integración

Ofrece un sistema con una interfaz de usuario intuitiva y simple de usar, además el sistema cuenta con la capacidad de emitir notificaciones para las alertas de ruidos y permite configurar el nivel de ruido percibido como molesto. El proyecto contempla un dispositivo integrado por una Raspberry Pi, sensor de sonido, software desarrollado, manual de usuario y documentación técnica.

#### Módulo de Telegram

El código implementa comandos que operan en segundo plano para procesar mensajes de forma remota. Se filtran los mensajes para aceptar únicamente aquellos provenientes de un usuario autorizado. Cuando se recibe un comando válido, se envía al sistema y es procesado, si existen cambios de parámetros como el umbral establecido, se guardan los nuevos parámetros y se devuelve una respuesta de confirmación al usuario a través del chat de Telegram.

```
# Telegram listener (comandos) - long polling
# ============================================================
def run_telegram_commands_in_thread(core: NoiseMonitorCore):
    def _worker():
        allowed = str(CHAT_ID).strip()  # solo aceptar comandos desde este chat
        offset = 0

        while True:
            if BOT_TOKEN == "PON_TU_TOKEN_AQUI" or not BOT_TOKEN.strip():
                time.sleep(5)
                continue

            try:
                params = urllib.parse.urlencode({"timeout": 30, "offset": offset})
                url = f"https://api.telegram.org/bot{BOT_TOKEN}/getUpdates?{params}"

                with urllib.request.urlopen(url, timeout=35) as resp:
                    data = json.loads(resp.read().decode("utf-8"))

                if not data.get("ok"):
                    time.sleep(2)
                    continue

                for upd in data.get("result", []):
                    offset = max(offset, upd.get("update_id", 0) + 1)

                    msg = upd.get("message") or upd.get("edited_message")
                    if not msg:
                        continue

                    chat_id_in = str((msg.get("chat") or {}).get("id", ""))
                    if allowed and chat_id_in != allowed:
                        continue

                    text = msg.get("text", "")
                    reply = core.process_command(text)
                    if reply:
                        if text.strip().startswith(("/set_threshold", "/set_cooldown")):
                            save_runtime_config(core)
                        send_telegram_message(reply, BOT_TOKEN, CHAT_ID)

            except Exception as e:
                print(f"⚠️ Telegram polling error: {e}")
                time.sleep(2)

    t = threading.Thread(target=_worker, daemon=True)
    t.start()
    return t
```

**Módulo de conexión Bluetooth**

Aquí se establece un servidor Bluetooth. Permite controlar el sistema de monitoreo de ruido desde el dispositivo móvil. El módulo funciona independiente para no interrumpir el programa principal, quedando a la espera de conexiones entrantes para procesar comandos remotos para iniciar o detener el monitoreo y ajustar umbrales.

```
# Bluetooth SPP server (Android -> Raspberry)
# Protocolo: 1 JSON por línea (terminada en \n), respuesta JSON por línea.
# Ejemplos:
# {"cmd":"get_status"}
# {"cmd":"apply","threshold_db":70,"notification_cooldown":120}
# {"cmd":"start"}
# {"cmd":"stop"}
# {"cmd":"command","text":"/set_threshold 70"}
# ============================================================
def run_bluetooth_server_in_thread(core: NoiseMonitorCore):
    def _worker():
        try:
            import bluetooth  # python3-bluez / pybluez
        except Exception as e:
            print("⚠️ Bluetooth Python no disponible. Instala python3-bluez (o PyBluez). Error:", e)
            return

        server_sock = None
        try:
            server_sock = bluetooth.BluetoothSocket(bluetooth.RFCOMM)
            server_sock.bind(("", 1 ))
            server_sock.listen(1)
            port = server_sock.getsockname()[1]


            print(f"✓ Bluetooth SPP listo: servicio '{BT_SERVICE_NAME}', RFCOMM port {port}")

            while True:
                try:
                    client_sock, client_info = server_sock.accept()
                    print("✓ Bluetooth conectado:", client_info)

                    buf = b""
                    while True:
                        data = client_sock.recv(1024)
                        if not data:
                            break
                        buf += data
                        while b"\n" in buf:
                            line, buf = buf.split(b"\n", 1)
                            line = line.strip()
                            if not line:
                                continue

                            resp = handle_bt_line(core, line.decode("utf-8", errors="replace"))
                            out = (json.dumps(resp, ensure_ascii=False) + "\n").encode("utf-8")
                            client_sock.send(out)
```

```
                except Exception as e:
                    print("⚠️ Bluetooth client error:", e)
                finally:
                    try:
                        client_sock.close()
                    except Exception:
                        pass
                    print("ℹ️ Bluetooth desconectado")

        except Exception as e:
            print("⚠️ Bluetooth server error:", e)
        finally:
            try:
                if server_sock:
                    server_sock.close()
            except Exception:
                pass

    t = threading.Thread(target=_worker, daemon=True)
    t.start()
    return t


def handle_bt_line(core: NoiseMonitorCore, text: str) -> dict:
    # 1) Permitir enviar comandos tipo Telegram: "/set_threshold 70"
    if text.strip().startswith("/"):
        reply = core.process_command(text.strip())
        if reply:
            if text.strip().startswith(("/set_threshold", "/set_cooldown")):
                save_runtime_config(core)
            return {"ok": True, "type": "command", "reply": reply, "status": core.get_status()}
        return {"ok": False, "error": "unknown command"}

    # 2) JSON por línea
    try:
        req = json.loads(text)
    except Exception:
        return {"ok": False, "error": "invalid json"}

    cmd = str(req.get("cmd", "")).strip()

    if cmd == "ping":
        return {"ok": True, "pong": True}

    if cmd == "get_status":
        return {"ok": True, "status": core.get_status()}

    if cmd == "get_history":
```

```
if cmd == "get_history":
    limit = req.get("limit", 100)
    return {"ok": True, "history": core.get_history(limit)}

if cmd in ("apply", "set_config"):
    core.set_config(
        threshold_db=req.get("threshold_db"),
        notification_cooldown=req.get("notification_cooldown"),
        max_points=req.get("max_points"),
    )
    save_runtime_config(core)
    return {"ok": True, "status": core.get_status()}

if cmd == "start":
    core.start()
    return {"ok": True, "status": core.get_status()}

if cmd == "stop":
    core.stop()
    return {"ok": True, "status": core.get_status()}

if cmd == "mute":
    minutes = req.get("minutes", 0)
    core.set_mute_minutes(int(minutes))
    return {"ok": True, "status": core.get_status()}

if cmd == "unmute":
    core.set_mute_minutes(0)
    return {"ok": True, "status": core.get_status()}

if cmd == "test_telegram":
    core.test_telegram()
    return {"ok": True, "status": core.get_status()}

if cmd == "command":
    line = str(req.get("text", ""))
    reply = core.process_command(line)
    if reply:
        if line.strip().startswith(("/set_threshold", "/set_cooldown")):
            save_runtime_config(core)
        return {"ok": True, "reply": reply, "status": core.get_status()}
    return {"ok": False, "error": "unknown command"}

return {"ok": False, "error": "unknown cmd"}
```

**Módulo de la interfaz gráfica**

Aquí se encuentra definida la interfaz gráfica de la aplicación móvil. Se usa la librería Tkinter y la función principal es iniciar la ventana que muestra en tiempo real los decibelios detectados, el estado del sistema y la conexión con Telegram, integrando además un gráfico dinámico para visualizar el historial de ruido y una línea roja que marca el umbral de alerta. Para interactuar con el usuario, incluye botones de control para iniciar o detener el monitoreo y realizar pruebas de envío de mensajes por Telegram.

```
# Tkinter UI
# ============================================================
class NoiseMonitorApp:
    def __init__(self, root: tk.Tk, core: NoiseMonitorCore):
        self.root = root
        self.core = core

        root.title("Monitor de Decibeles - GrovePi + Telegram + Bluetooth")
        root.geometry("720x520")
        root.resizable(False, False)

        tk.Label(root, text="Monitor de Ruido", font=("Arial", 20, "bold")).pack(pady=10)

        self.db_label = tk.Label(root, text="Decibeles: -- dB", font=("Arial", 18))
        self.db_label.pack()

        self.status_label = tk.Label(root, text="Estado: Inactivo", font=("Arial", 14))
        self.status_label.pack(pady=5)

        self.telegram_label = tk.Label(root, text="Telegram: Listo", font=("Arial", 12), fg="green")
        self.telegram_label.pack()

        button_frame = tk.Frame(root)
        button_frame.pack(pady=10)

        self.start_btn = ttk.Button(button_frame, text="Iniciar", command=self.on_start)
        self.start_btn.grid(row=0, column=0, padx=10)

        self.stop_btn = ttk.Button(button_frame, text="Detener", command=self.on_stop)
        self.stop_btn.grid(row=0, column=1, padx=10)

        self.test_btn = ttk.Button(button_frame, text="Test Telegram", command=self.on_test_telegram)
        self.test_btn.grid(row=0, column=2, padx=10)

        self.fig, self.ax = plt.subplots(figsize=(7, 3))
        self.ax.set_title("Historial de Decibeles")
        self.ax.set_xlabel("Tiempo (puntos)")
        self.ax.set_ylabel("dB")

        initial = self.core.get_history(self.core.max_points)
        self.line, = self.ax.plot(range(len(initial)), initial, linewidth=2)

        self.threshold_line = self.ax.axhline(
            y=self.core.threshold_db, color="r", linestyle="--",
            label=f"Umbral ({self.core.threshold_db} dB)"
        )
        self.ax.legend()
```

```
        self.ax.legend()

        self.canvas = FigureCanvasTkAgg(self.fig, master=root)
        self.canvas.get_tk_widget().pack()

        self._ui_refresh()
        root.protocol("WM_DELETE_WINDOW", self.on_close)

    def on_start(self):
        self.core.start()

    def on_stop(self):
        self.core.stop()

    def on_test_telegram(self):
        self.telegram_label.config(text="Telegram: Enviando prueba...", fg="orange")
        self.core.test_telegram()

    def _ui_refresh(self):
        st = self.core.get_status()
        hist = self.core.get_history(self.core.max_points)

        self.status_label.config(text="Estado: Monitoreando" if st["running"] else "Estado: Detenido")
        self.db_label.config(text=f"Decibeles: {st['last_db']} dB")

        tmsg = st["telegram_status"]
        if "Error" in tmsg:
            self.telegram_label.config(text=f"Telegram: {tmsg}", fg="red")
        elif "Alerta" in tmsg:
            self.telegram_label.config(text=f"Telegram: {tmsg}", fg="red")
        elif "Test OK" in tmsg:
            self.telegram_label.config(text=f"Telegram: {tmsg}", fg="green")
        elif "Silenciado" in tmsg:
            self.telegram_label.config(text=f"Telegram: {tmsg}", fg="orange")
        else:
            self.telegram_label.config(text=f"Telegram: {tmsg}", fg="green")

        self.line.set_data(range(len(hist)), hist)
        self.ax.set_xlim(0, max(10, len(hist)))

        if hist:
            mn, mx = min(hist), max(hist)
            self.ax.set_ylim(mn - 5, mx + 5)

        thr = st["threshold_db"]
        self.threshold_line.set_ydata([thr, thr])
        self.threshold_line.set_label(f"Umbral ({thr} dB)")
        self.ax.legend()
```

**Módulo de núcleo compartido**

Aquí se define la lógica del sistema, se encarga de las gestiones de configuración de los sensores y el procesamiento de datos en segundo plano. El propósito es monitorear continuamente los niveles de ruido, compararlos con un umbral de decibelios predefinido y mantener un historial de datos en una cola circular para su visualización.

```
# Núcleo (motor) compartido: sensor + lógica + estado
# ============================================================
class NoiseMonitorCore:
    def __init__(self,
                 sound_sensor_port: int = DEFAULT_SOUND_SENSOR,
                 threshold_db: float = DEFAULT_THRESHOLD_DB,
                 max_points: int = DEFAULT_MAX_POINTS,
                 notification_cooldown: int = DEFAULT_COOLDOWN):

        self.sound_sensor_port = sound_sensor_port
        self.threshold_db = float(threshold_db)
        self.max_points = int(max_points)
        self.notification_cooldown = int(notification_cooldown)

        self._lock = threading.Lock()
        self._running = False
        self._thread = None

        self.last_raw = 0
        self.last_db = 0.0
        self.alert_active = False
        self.last_notification_time = 0.0

        self.history = deque([0.0] * self.max_points, maxlen=self.max_points)

        self.telegram_status = "Listo"
        self.telegram_last_ok = None  # None/True/False

        self.muted_until = 0.0  # para /mute

    def start(self):
        with self._lock:
            if self._running:
                return
            self._running = True

        self._thread = threading.Thread(target=self._loop, daemon=True)
        self._thread.start()

    def stop(self):
        with self._lock:
            self._running = False
            self.alert_active = False

        if HAY_GROVE:
            try:
                setRGB(0, 0, 0)
                setText_norefresh("")
```

# Conclusión

El proyecto logró desarrollar un dispositivo funcional automatizado integrando la tecnología IoT, una Raspberry Pi 4b y sensores. Este dispositivo es capaz de medir parámetros ambientales de sonido en tiempo real junto a los sensores y una interfaz de usuario intuitiva. La correcta definición del objetivo como solución al problema propuesto, restricciones, supuestos y planificación de los procesos técnicos permitió mantener el enfoque en la usabilidad y confiabilidad del sistema, garantizando resultados consistentes.

# Trabajo futuro

## Optimización del sensor de micrófono

Mejoras en la optimización de los sonidos percibidos por el sensor de micrófono como también aumentar la cantidad de sensores de micrófono integrados para un resultado más preciso.

## Interfaz de usuario

Mejoras en la interfaz de usuario en cuanto a la comodidad visual y nuevas funcionalidades que se puedan desarrollar en un futuro a partir del uso cotidiano que se le de.

## Diseño mejorado y compacto del dispositivo

Idealmente se puede trabajar en las mejoras del producto final en cuanto a su compactibilidad y protección del acabado de la carcasa.

## Referencias

Python (Lenguaje de programación)

Van Rossum, G., & Drake, F. L. Jr. (2009). Python 3 reference manual. CreateSpace. https://ir.cwi.nl/pub/5008 citebay.com

Arch Linux (Distribución del sistema operativo)

Arch Linux Wiki. (n.d.). ArchWiki: Reading. https://wiki.archlinux.org/title/Help%3AReading wiki.archlinux.org

Raspberry Pi OS (Sistema operativo)

Raspberry Pi Foundation. (n.d.). Raspberry Pi documentation. https://www.raspberrypi.com/documentation/raspberrypi.com