Sistema de Gestión Polimórfica de Sensores para IoT (ESP32)

Reporte técnico

Alumno: Elias de Jesús Zúñiga de León
Materia: Programación Orientada a Objetos / Integración HW–SW
Fecha: 31 de octubre de 2025

Este documento describe el diseño, desarrollo, componentes y procedimiento de validación del caso de estudio: gestión polimórfica de sensores con listas enlazadas genéricas y ESP32.

Índice

1. Introducción

Este proyecto implementa un **sistema de bajo nivel** para registrar, almacenar y procesar lecturas de múltiples tipos de sensores (temperatura y presión) de forma unificada. Se atacan dos rigideces típicas: (1) *tipo de dato* (enteros vs. flotantes) y (2) *estructura* (número variable de lecturas).
La solución combina:

- **POO avanzada y polimorfismo:** jerarquía de sensores con una *clase base abstracta* que obliga la implementación de métodos esenciales.
- **Listas enlazadas genéricas (templates):** contenedores *hechos a mano* sin STL para gestionar lecturas de distinto tipo (`int`, `float`).
- **Integración HW–SW:** un ESP32 transmite lecturas simuladas por puerto serial; una aplicación C++ de escritorio las ingesta y procesa en una lista de gestión polimórfica.
- **Construcción reproducible:** firmware con PlatformIO/Arduino y aplicación nativa con CMake.

Objetivo. Demostrar una arquitectura que permita registrar cualquier tipo de sensor en una lista común, insertar lecturas genéricas y ejecutar *procesamiento polimórfico* (cada sensor implementa su propia lógica).

2. Manual técnico

2.1. Diseño y arquitectura

Jerarquía de clases

- `SensorBase` (abstracta): define interfaz común y expone `procesarLectura()`, `mostrarEstado()`, `obtenerNumLecturas()`. Mantiene un nombre (ID).
- `SensorTemperatura`: gestiona `ListaSensor<float>`; calcula promedio/mínimo/máximo.
- `SensorPresion`: gestiona `ListaSensor<int>`; calcula promedio/mínimo/máximo.
- `ListaSensor<T>`: lista enlazada simple genérica con operaciones `insertar`, `calcularPromedio`, `obtenerMaximo/Minimo`, `liberarMemoria`, etc.
- `ListaGestion`: lista polimórfica que guarda `SensorBase*` y permite `agregarSensor`, `buscarSensor`, `procesarTodos`, `mostrarTodos`, `liberarTodo`.

Diagrama (alto nivel)

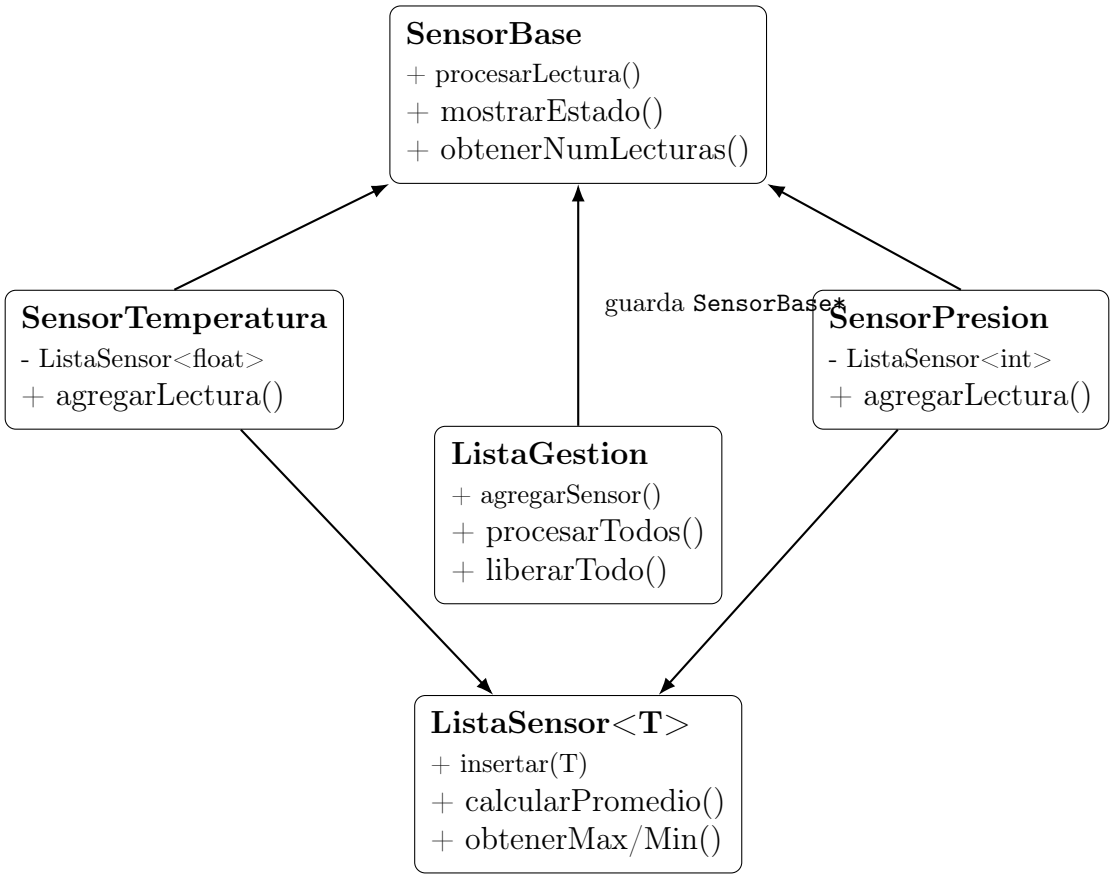

Figura 1: Relaciones principales de clases y listas.

2.2. Estructura del proyecto

Ruta	Contenido
`include/`	Cabeceras: `SensorBase.h`, `SensorTemperatura.h`, `SensorPres`
`src/main.cpp`	Firmware ESP32 (lecturas simuladas y envío por serial)
`arduino/esp32_sensor_simulator.ino`	Alternativa para Arduino IDE
`main.cpp`	Aplicación C++ de escritorio (menú, serial, polimorfismo)
`platformio.ini`	Configuración PlatformIO (baud 115200, tarjeta ESP32)
`CMakeLists.txt`	Proyecto CMake para compilar la app nativa

2.3. Componentes

Elemento	Modelo/Versión	Uso
ESP32 DevKit V1	ESP32-WROOM-32	Generación y transmisión de lecturas
Cable USB	Micro-USB/USB-C	Alimentación y comunicación
PC (Windows/Linux)	C++17 + CMake	Aplicación de gestión polimórfica
IDE/Tooling	PlatformIO/VSCode, Arduino IDE (opc.)	Compilar y cargar firmware
Documentación	Doxygen (opcional)	Generar documentación de cabeceras

2.4. Flujo de datos

1. El ESP32 envía cada `INTERVALO_LECTURA` ms un lote de valores en CSV: `T-001,23.5` y `P-001,1013`, seguidos de una línea separadora `--`.
2. La app de escritorio (`main.cpp`) abre el puerto (*Windows:* `COMx`; *Linux:* `/dev/ttyUSB0`), interpreta cada línea y crea/busca el sensor correspondiente en `ListaGestion`.
3. Cada sensor inserta su lectura en su `ListaSensor<T>` interna; las operaciones de proceso calculan promedio, mínimo y máximo.

3. Desarrollo

3.1. Firmware ESP32 (PlatformIO/Arduino)

Parámetros clave (`src/main.cpp`):

```
#define INTERVALO_LECTURA 2000
#define NUM_SENSORES_TEMP 3
#define NUM_SENSORES_PRES 2
#define TEMP_MIN 18.0
#define TEMP_MAX 30.0
#define PRESION_MIN 1000
#define PRESION_MAX 1020
```

El firmware imprime, por ejemplo:

```
T-001,24.7
T-002,19.3
T-003,28.1
```

```
P-001,1012
P-002,1004
---
```

3.2. Aplicación C++ nativa (polimorfismo + serial)

Menú principal. Las opciones incluyen crear sensores, registrar lecturas manuales, *leer desde ESP32*, procesar y mostrar estado.

```
========================================
  Sistema IoT - Monitoreo Polimorfico
========================================
1. Crear Sensor de Temperatura
2. Crear Sensor de Presion
3. Registrar Lectura Manual
4. Leer Datos desde ESP32 (Serial)
5. Procesar Todos los Sensores
6. Mostrar Estado de Sensores
7. Salir y Liberar Memoria
========================================
```

Lógica de ingesta. La función `leerDesdeSerial()` detecta el puerto por plataforma, lee líneas y:

1. Si llega `T-###,valor` crea/busca `SensorTemperatura` y agrega la lectura (`float`).
2. Si llega `P-###,valor` crea/busca `SensorPresion` y agrega la lectura (`int`).
3. Tras acumular lecturas, `procesarTodos()` imprime métricas por sensor.

4. Instalación y ejecución

4.1. Cargar firmware en ESP32 (PlatformIO)

1. Abrir la carpeta `sp32/` en VSCode con PlatformIO.
2. Conectar el ESP32 y verificar el puerto: sustituir `monitor_port` por su COM correcto en `platformio.ini`. Baud: `115200`.
3. Ejecutar **Upload** y luego **Monitor**. Deberá ver líneas CSV y la separadora `--` cada `INTERVALO_LECTURA` ms.

Captura 1 (obligatoria): *Monitor de PlatformIO mostrando las 5 líneas de un ciclo y* `--` (guardar como `img/monitor_platformio.png`).

4.2. Alternativa: Arduino IDE

1. Abrir `arduino/esp32_sensor_simulator/esp32_sensor_simulator.ino`.
2. Seleccionar tarjeta .ESP32 Dev Modulez el puerto correcto. Monitor Serial a 115200 baudios.

Captura 2 (opcional): *Arduino Serial Monitor con las líneas CSV* (`img/arduino_serial.png`).

4.3. Compilar la app nativa con CMake

1. Instalar CMake y un compilador C++17 (MSVC/MinGW en Windows o `g++` en Linux).
2. En una terminal en `sp32/` ejecutar:

```
mkdir build && cd build
cmake ..
cmake --build . --config Release
# Ejecutable generado: SistemaIoT (o SistemaIoT.exe)
```

Captura 3 (obligatoria): *Consola mostrando la configuración y compilación exitosa de CMake* (`img/cmake_build.png`).

4.4. Ejecución de la app

1. Conectar el ESP32. En Windows, verificar el puerto en .ªdministrador de dispositivos".
2. Abrir la app y elegir "4. Leer Datos desde ESP32 (Serial)". En el código por defecto se usa `COM9` (Windows) o `/dev/ttyUSB0` (Linux).
3. Tras recibir lecturas, usar "5. Procesar Todos los Sensoresz "6. Mostrar Estado de Sensores".

Captura 4 (obligatoria): *Pantalla del menú de la app* (`img/app_menu.png`).
Captura 5 (obligatoria): *Salida con promedios/mínimos/máximos tras procesar* (`img/app_processing.p`

5. Validación de la solución

5.1. Criterios

- Llegan lecturas en el formato esperado y con la periodicidad configurada.
- Para cada sensor existente, `obtenerNumLecturas()` coincide con el número de inserciones.
- `procesarLectura()` imprime promedio, mínimo y máximo correctos.
- Al salir, `liberarTodo()` libera memoria sin fugas (destructores invocados en cascada).

5.2. Evidencias (colocar capturas)

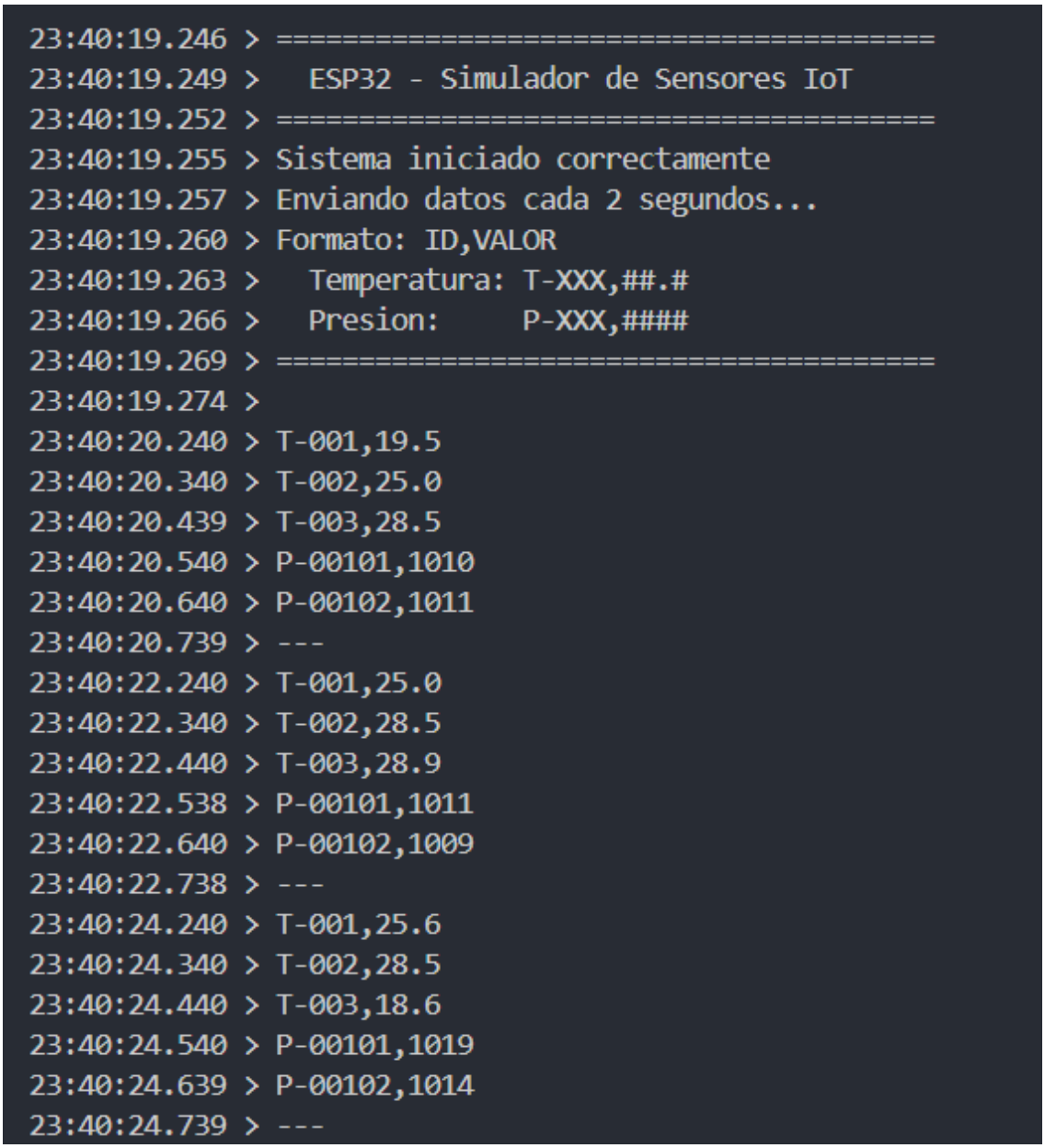

Figura 2: ESP32 transmitiendo lecturas simuladas (PlatformIO).

Figura 3: Compilación exitosa con CMake.

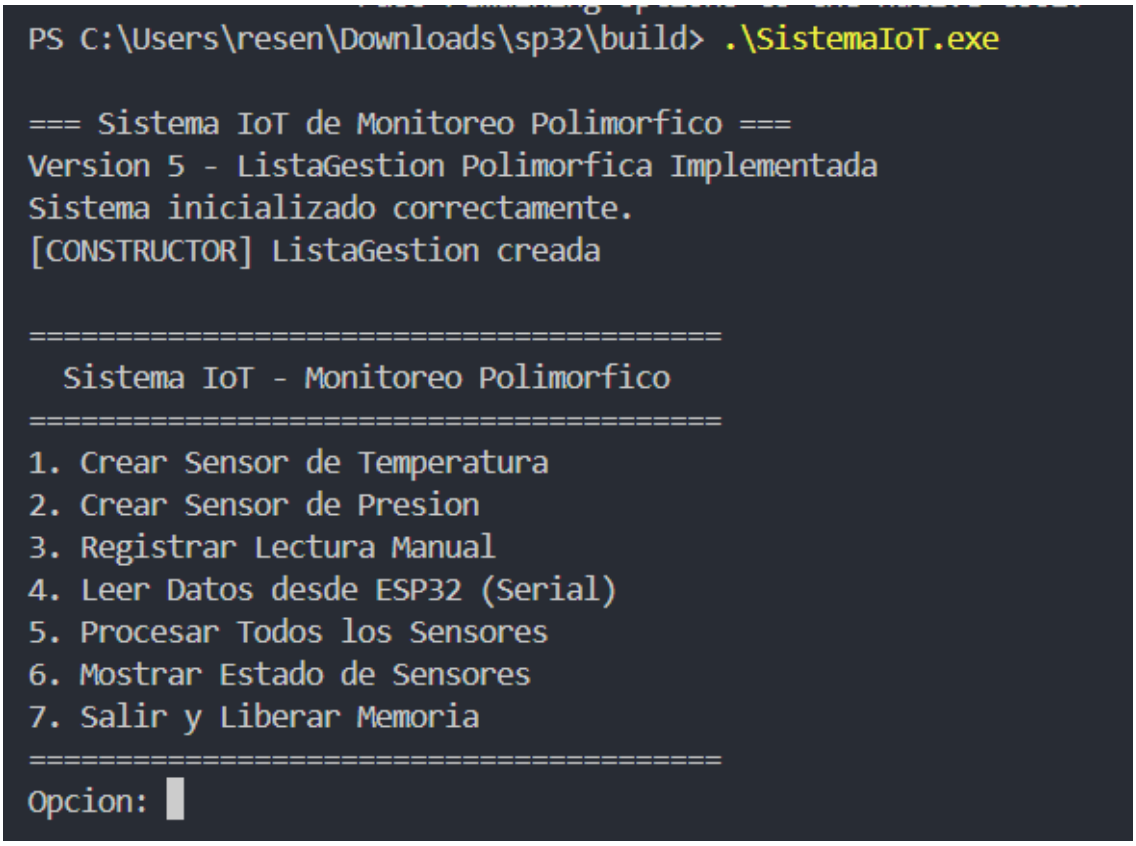

Figura 4: Aplicación nativa: menú principal.

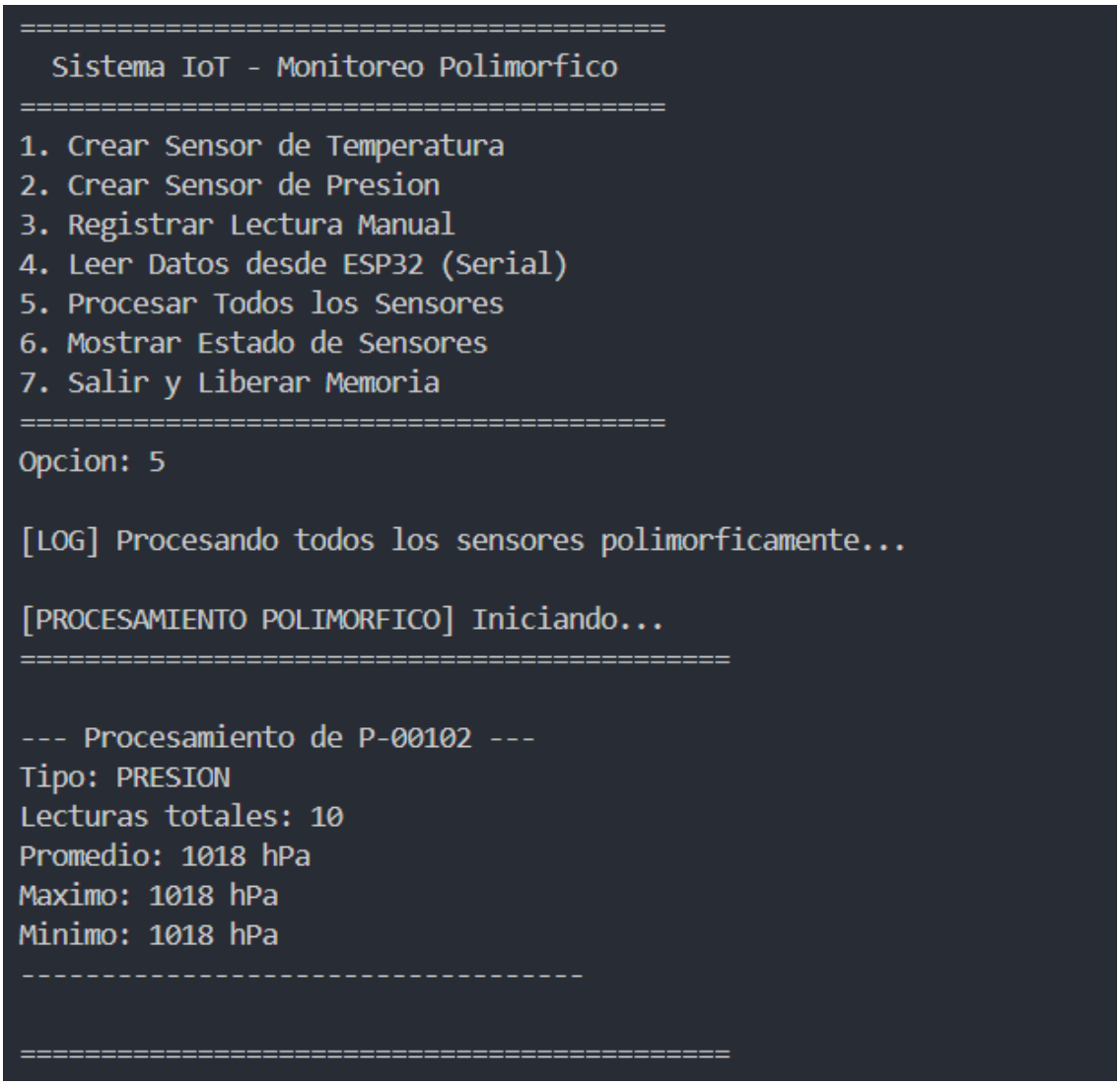

Figura 5: Procesamiento polimórfico: métricas por sensor.

6. Resultados de consola (ejemplo)

```
=== Sistema IoT de Monitoreo Polimorfico ===
[SERIAL] Esperando datos del ESP32...
[RECV] T-001,24.7
[RECV] T-002,19.3
[RECV] T-003,28.1
[RECV] P-001,1012
[RECV] P-002,1004
[SUCCESS] Se recibieron 5 lecturas del ESP32
--- Procesando Sensores ---
[T-001] Lecturas: 1 Prom: 24.7 Min: 24.7 Max: 24.7
[T-002] Lecturas: 1 Prom: 19.3 Min: 19.3 Max: 19.3
[T-003] Lecturas: 1 Prom: 28.1 Min: 28.1 Max: 28.1
[P-001] Lecturas: 1 Prom: 1012 Min: 1012 Max: 1012
[P-002] Lecturas: 1 Prom: 1004 Min: 1004 Max: 1004
```

7. Conclusiones

Se logró un sistema capaz de integrar lecturas heterogéneas mediante polimorfismo y listas genéricas sin depender de STL. La separación HW–SW permite escalar tipos de sensores y estrategias de procesamiento con bajo acoplamiento. El uso de CMake y PlatformIO facilita la reproducibilidad del flujo de construcción.

A. Funciones principales por clase

Clase	Funciones destacadas
`SensorBase`	`procesarLectura()`, `mostrarEstado()`, `obtenerNumLecturas()`
`SensorTemperatura`	`agregarLectura(float)`, `procesarLectura()`
`SensorPresion`	`agregarLectura(int)`, `procesarLectura()`
`ListaSensor<T>`	`insertar(T)`, `calcularPromedio()`, `obtenerMaximo()`, etc.
`ListaGestion`	`agregarSensor()`, `buscarSensor()`, `procesarTodos()`, etc.

B. Parámetros configurables

- `INTERVALO_LECTURA` (ms): periodo de transmisión.
- `NUM_SENSORES_TEMP`, `NUM_SENSORES_PRES`: cantidad de IDs simulados.
- Rangos `TEMP_MIN/MAX`, `PRESION_MIN/MAX`.
- Puerto serial por plataforma en la app (`COMx` o `/dev/ttyUSB0`).

C. Generar documentación con Doxygen (opcional)

1. Instalar Doxygen. En `sp32/`: ejecutar `doxygen -g` para crear `Doxyfile`.

2. Editar `INPUT = include` y habilitar HTML. Ejecutar `doxygen`.
3. Abrir `html/index.html`.

Captura 6 (opcional): *Página principal de Doxygen* (`img/doxygen_index.png`).

Nota: Para una entrega formal, coloque todas las imágenes dentro de la carpeta `img/` referenciadas en este documento.