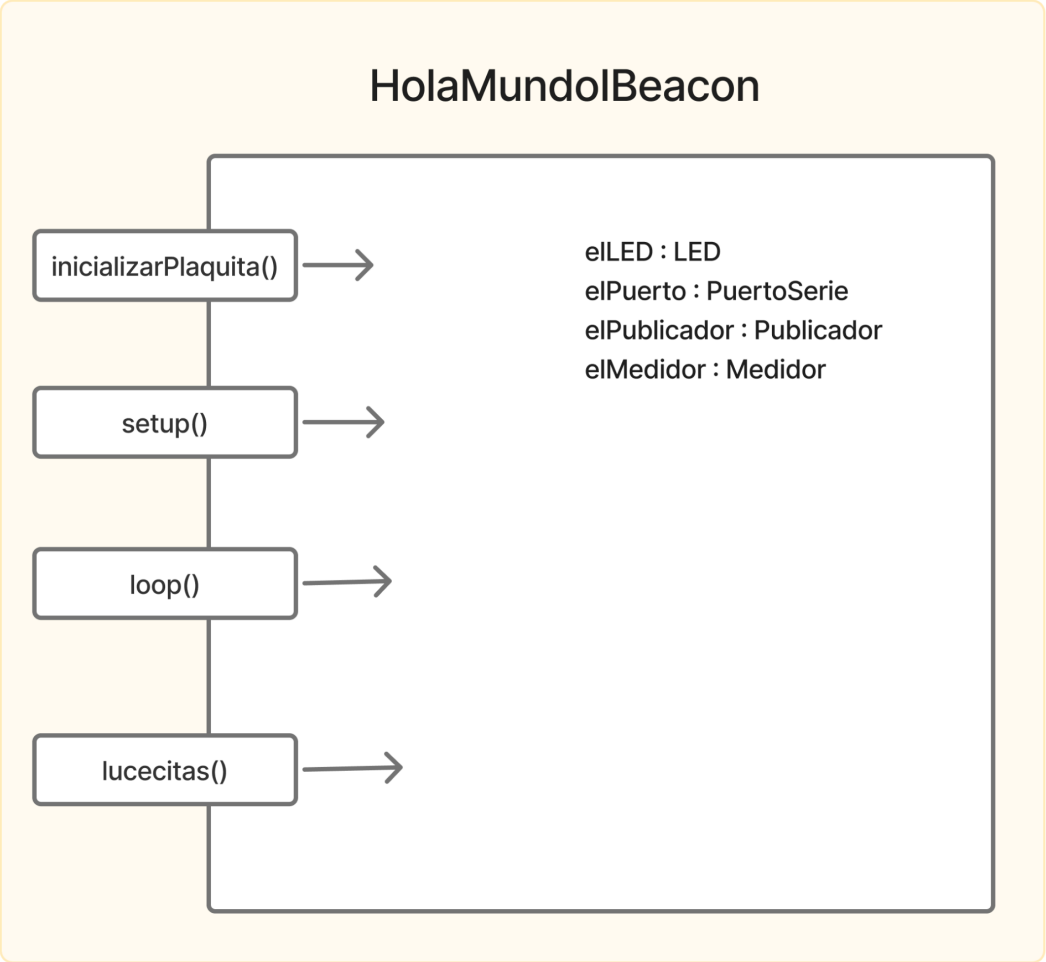
SPRINT 0 - Documento Diseños

Meryame Ait Boumlik

ARDUINO

1. HolaMundoIBeacon

Este programa es el principal de la placa. Inicializa los componentes (LED, puerto serie, emisor BLE) y utiliza las clases **Medidor** y **Publicador** para leer valores de sensores (CO_2, temperatura) y difundirlos mediante anuncios BLE (iBeacon). También incluye una secuencia de parpadeo del LED como prueba visual.



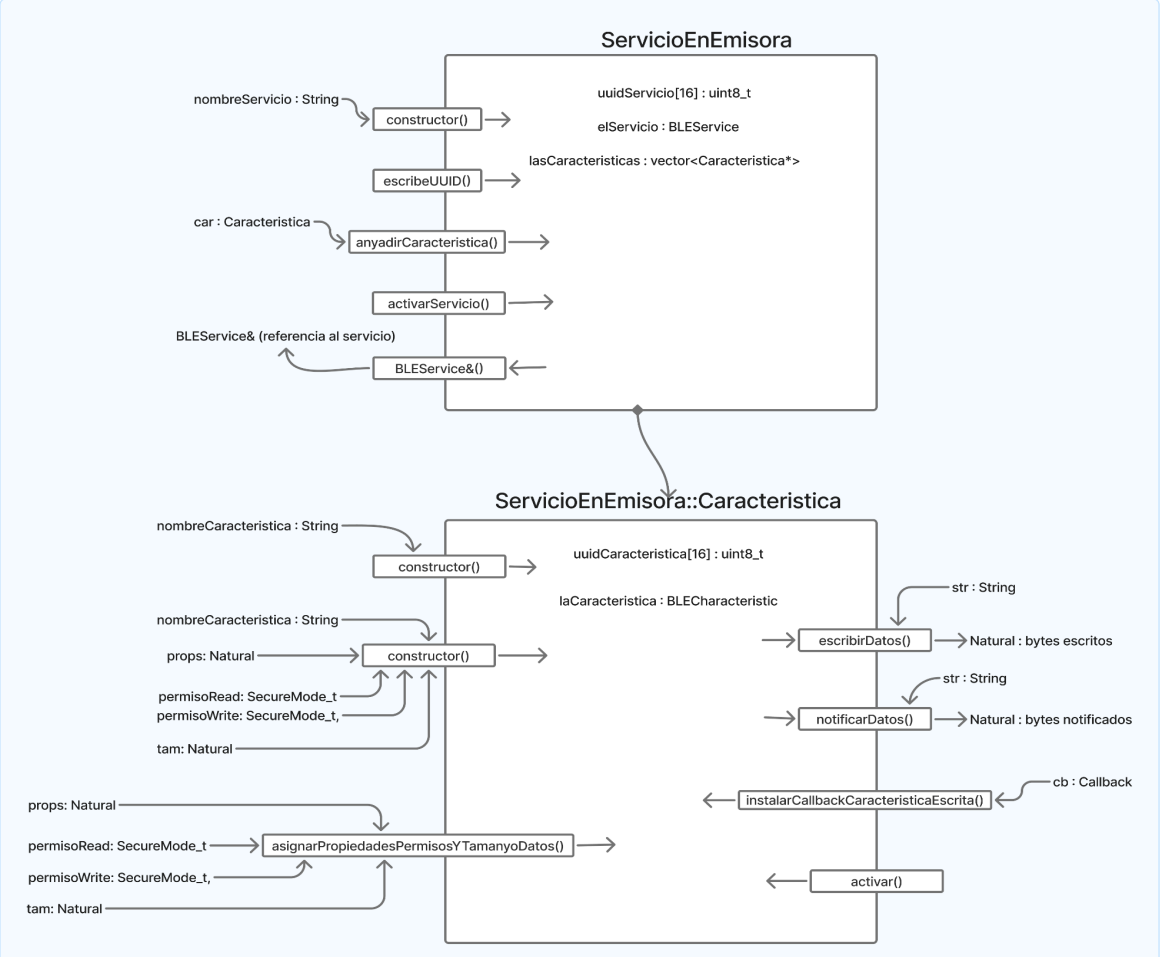
- **`inicializarPlaquita()`**: Inicializa la placa
- **`setup()`**: Configura el puerto serie, enciende la emisora BLE, inicia el medidor y muestra mensaje de inicio.

- **`lucecitas()`**: Hace parpadear el LED con distintos intervalos.

- **`loop()`**: Bucle principal. Incrementa un contador, muestra mensajes, hace parpadear el LED, mide CO_2 y temperatura, y publica los valores como anuncios BLE. Incluye una prueba de iBeacon con datos libres.

2. ServicioEnEmisora

Esta clase representa un **servicio BLE** que se puede anunciar desde la emisora. Permite configurar su UUID, añadir características y activar el servicio. Además, incluye la clase interna **Caracteristica** para gestionar las propiedades y el intercambio de datos.



❖ *Atributos (ServicioEnEmisora) :*

- **`uuidServicio[16]`** : Identificador único del servicio.
- **`elServicio`** **:** Objeto BLEService real.

- **`lasCaracteristicas`** : Lista de características asociadas.

❖ *Métodos principales (ServicioEnEmisora) :*

- **ServicioEnEmisora(nombreServicio):** Constructor, inicializa el UUID.
- **escribeUUID()**: Muestra el UUID en el puerto serie.
- **anyadirCaracteristica(car):** Añade una característica al servicio.
- **activarServicio()**: Activa el servicio y sus características.
- **operator BLEService&()**: Conversión implícita a BLEService.

❖ *Atributos (ServicioEnEmisora::Caracteristica) :*

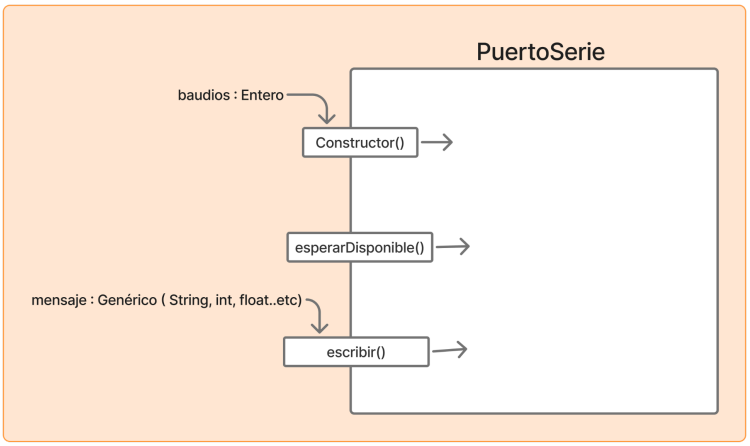
- **uuidCaracteristica[16] :** Identificador de la característica.
- **laCaracteristica :** Objeto BLECharacteristic real.

❖ *Métodos principales (ServicioEnEmisora::Caracteristica) :*

- **Constructores:** uno básico con nombre y otro extendido con propiedades, permisos y tamaño.
- **asignarPropiedadesPermisosYTamanyoDatos()**: Configura cómo se accede y qué tamaño de datos admite.
- **escribirDatos(str)**: Escribe datos en la característica.
- **notificarDatos(str)**: Notifica datos a los clientes conectados.
- **instalarCallbackCaracteristicaEscrita(cb)**: Instala un callback que se ejecuta cuando la característica recibe datos.
- **activar():** Inicia la característica en el servicio.

3.PuertoSerie

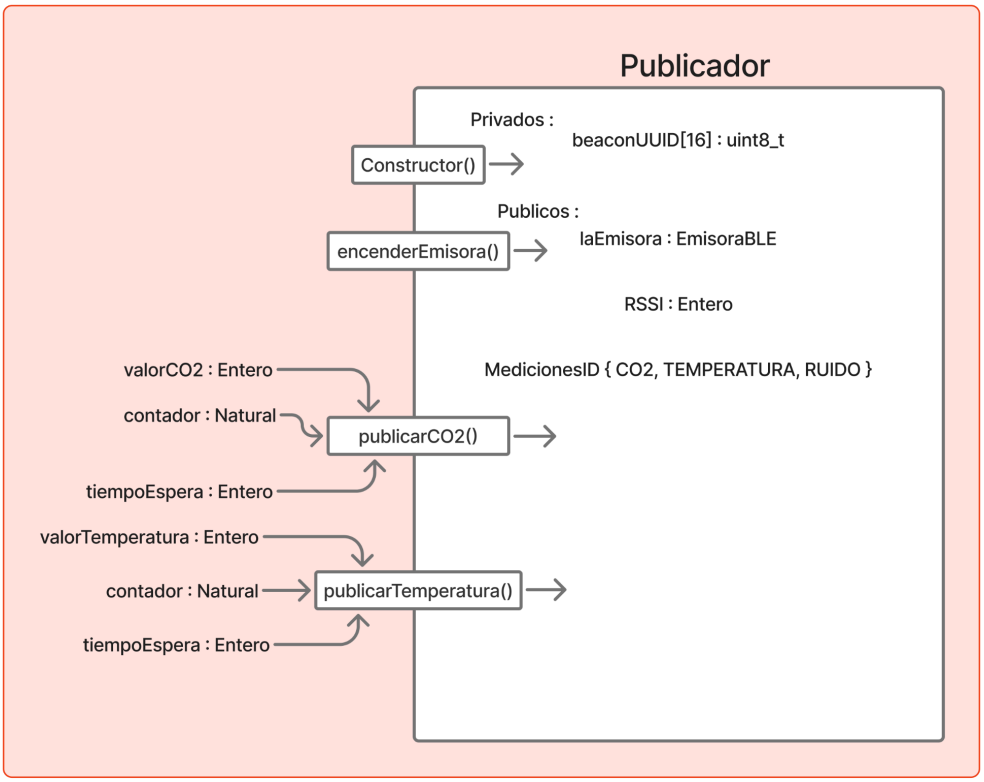
La clase `PuertoSerie` encapsula la comunicación serie de Arduino. Permite inicializar la conexión a un número específico de baudios, esperar a que el puerto esté disponible y enviar mensajes genéricos por el puerto serie.



- **PuertoSerie(baudios) (Constructor) :** Inicializa la comunicación serie a la velocidad indicada.
- **esperarDisponible() :** Espera hasta que el puerto serie esté listo para enviar/recibir datos.
- **escribir(mensaje) :**Envía un mensaje genérico (puede ser String, int, float, etc.) al puerto serie.

4. Publicador

La clase `Publicador` se encarga de **enviar las mediciones** de los sensores a través de la emisora BLE (`EmisoraBLE`) usando anuncios en formato iBeacon. Publica valores como CO2 o temperatura y controla el inicio y fin de las transmisiones.



❖ *Atributos*

- **beaconUUID** → Identificador del beacon.
- **laEmisora** → Emisora BLE.
- **RSSI** → Intensidad de señal.
- **MedicionesID** → Enum con tipos de medición.

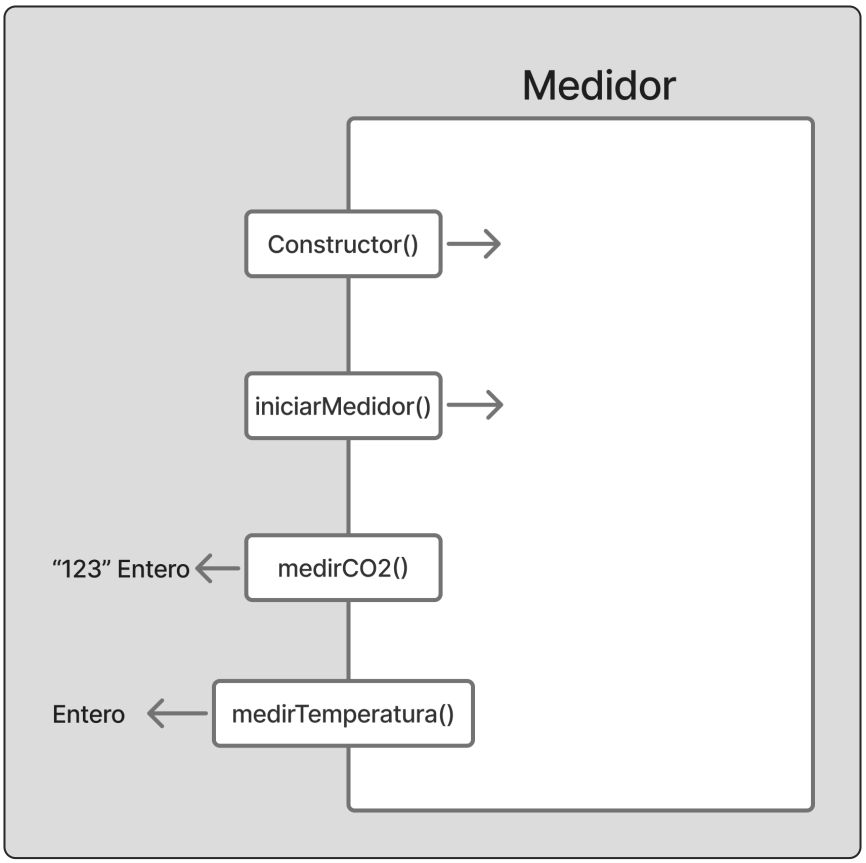
❖ *Métodos*

- **Publicador()** → Constructor.
- **encenderEmisora()** → Activa la emisora.
- **publicarCO2(valor, cont, espera)** → Envía valor de CO2.
- **publicarTemperatura(valor, cont, espera)** → Envía valor de temperatura.

➔ *Nota :* En el campo major:
- **byte alto** = ID de la medida (CO2, TEMPERATURA, RUIDO).
- **byte bajo** = contador de emisiones.

5. Medidor

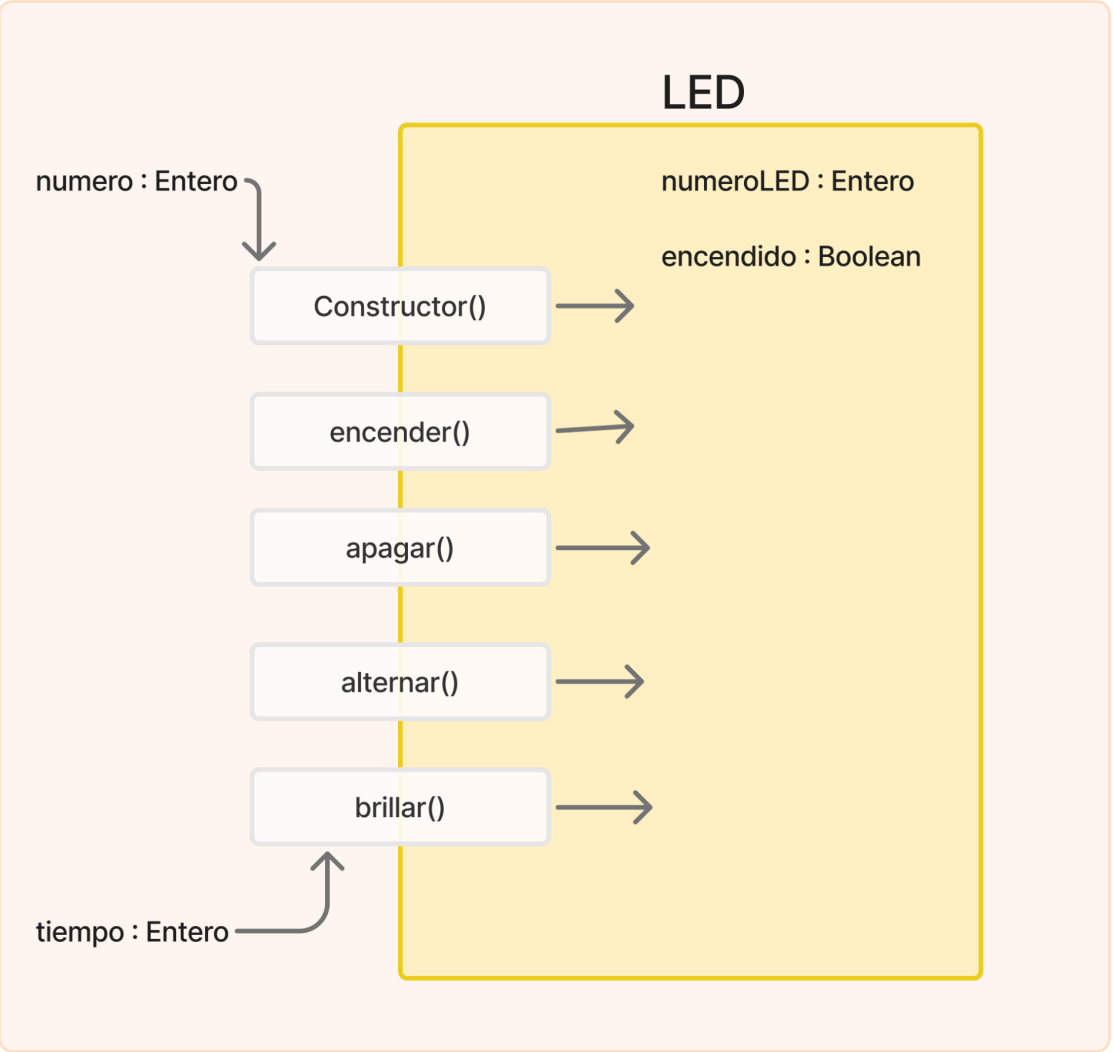
Clase que simula las lecturas de sensores. Devuelve valores de ejemplo para **CO2** y **temperatura**



- **Medidor()** → Constructor vacío.
- **iniciarMedidor()** → Inicializa el medidor (actualmente no hace nada).
- **medirCO2()** → Devuelve un valor entero de CO2 (ejemplo: 235).
- **medirTemperatura()** → Devuelve un valor entero de temperatura (ejemplo: -12).

6. LED

Clase para controlar un LED conectado a la placa. Permite encender, apagar, alternar el estado y hacerlo brillar durante un tiempo determinado.

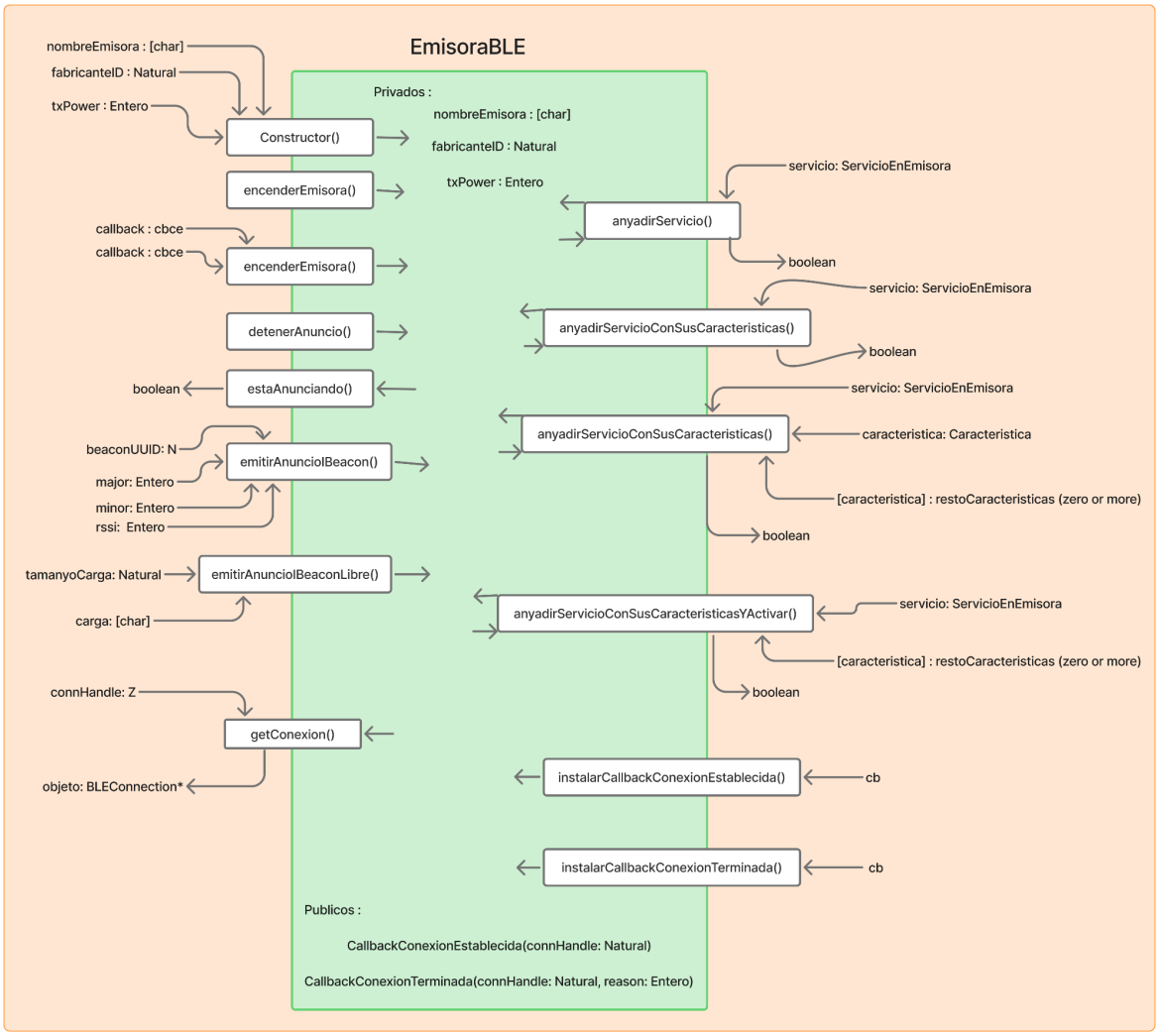


- **LED(numero)** → Constructor, inicializa el LED en el pin indicado.
- **encender()** → Enciende el LED.
- **apagar()** → Apaga el LED.
- **alternar()** → Cambia el estado (si está encendido lo apaga, si está apagado lo enciende).
- **brillar(tiempo)** → Hace parpadear el LED durante el tiempo dado (en ms).

Internamente guarda el número de pin (numeroLED) y el estado (encendido).

7. EmisoraBLE

Clase que simplifica el uso de la librería **Adafruit Bluefruit BLE**. Permite encender y detener la emisora, emitir anuncios en formato **iBeacon** o personalizados, añadir servicios y características, y gestionar callbacks de conexión/desconexión.



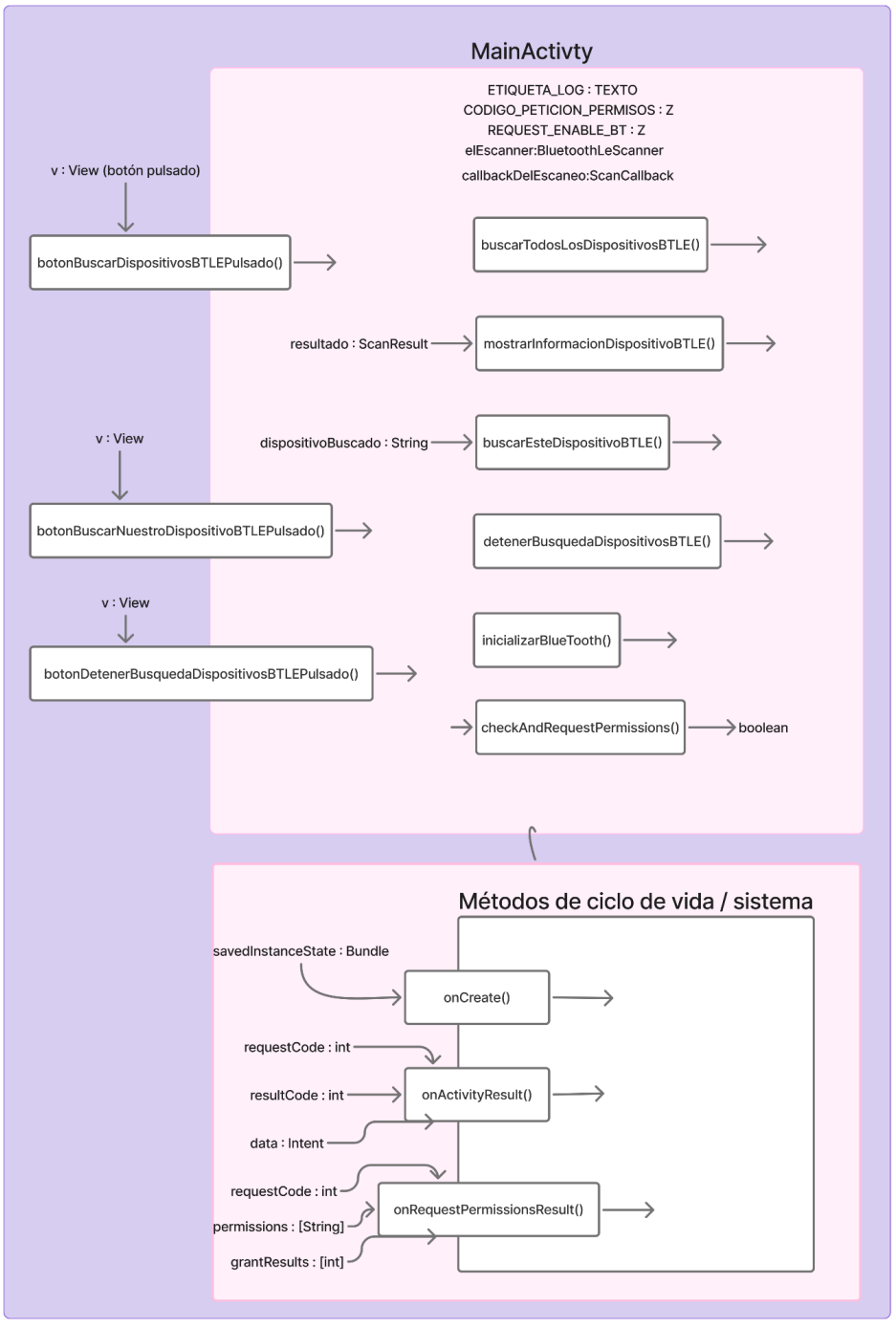
- **EmisoraBLE(nombre, fabricanteID, txPower)** → Constructor que configura la emisora.
- **encenderEmisora()** → Enciende la emisora y detiene anuncios previos.
- **encenderEmisora(cbEstablecida, cbTerminada)** → Igual que el anterior, pero instalando callbacks de conexión/desconexión.
- **detenerAnuncio()** → Detiene el anuncio si está en curso.
- **estaAnunciando()** → Devuelve si la emisora está anunciando.
- **emitirAnuncioIBeacon(uuid, major, minor, rssi)** → Emite un anuncio iBeacon estándar.
- **emitirAnuncioIBeaconLibre(carga, tamanyo)** → Emite un iBeacon con datos libres en la carga de 21 bytes.
- **anyadirServicio(servicio)** → Añade un servicio BLE.
- **anyadirServicioConSusCaracteristicas(...)** → Añade un servicio junto a sus características.
- **anyadirServicioConSusCaracteristicasYActivar(...)** → Añade el servicio, lo configura y lo activa.
- **instalarCallbackConexionEstablecida(cb)** → Instala un callback al establecerse una conexión.

- **instalarCallbackConexionTerminada(cb)** → Instala un callback al terminarse una conexión.
- **getConexion(connHandle)** → Devuelve el objeto de conexión BLE asociado al identificador.

JAVA

1. MainActivity

Es la actividad principal de la app Android. Controla la inicialización de Bluetooth, la búsqueda de dispositivos BLE/iBeacon y la gestión de permisos.



❖ *Atributos principales*

- **elEscanner : BluetoothLeScanner** → escáner BLE.
- **callbackDelEscaneo : ScanCallback** → recibe resultados de escaneo.
- **Constantes** para logs, permisos y request codes.

❖ *Métodos de botones*

- **botonBuscarDispositivosBTLEPulsado(View v)** → inicia la búsqueda de todos los dispositivos BLE.

- **botonBuscarNuestroDispositivoBTLEPulsado(View v)** → inicia la búsqueda de un dispositivo específico (por UUID o nombre).

- **botonDetenerBusquedaDispositivosBTLEPulsado(View v)** → detiene la búsqueda BLE.

❖ *Métodos auxiliares*

- **buscarTodosLosDispositivosBTLE()** → inicia escaneo genérico.

- **buscarEsteDispositivoBTLE(String dispositivoBuscado)** → inicia escaneo filtrado.

- **detenerBusquedaDispositivosBTLE()** → para el escaneo.

- **mostrarInformacionDispositivoBTLE(ScanResult resultado**) → muestra nombre, dirección, RSSI y datos crudos del anuncio; si es iBeacon, extrae uuid, major, minor y txPower.

- **inicializarBlueTooth()** → inicializa el adaptador, habilita Bluetooth y obtiene el escáner BLE.

- **checkAndRequestPermissions()** → comprueba y pide permisos necesarios (scan, connect, location).

❖ *Métodos de ciclo de vida / sistema*

- **onCreate(Bundle savedInstanceState)** → inicializa la actividad y Bluetooth.

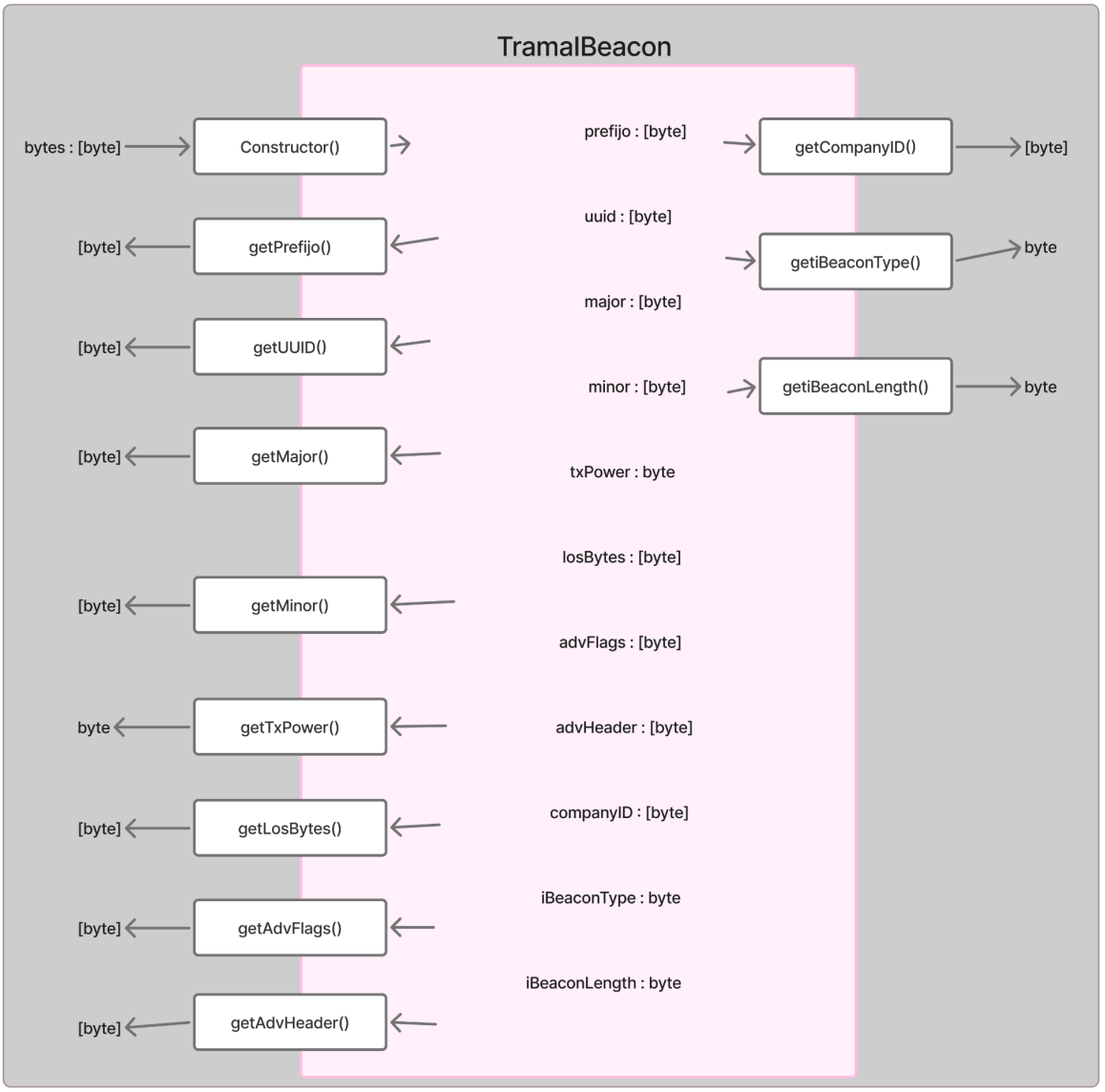
- **onActivityResult(...)** → maneja el resultado de habilitar Bluetooth.

- **onRequestPermissionsResult(...)** → maneja la respuesta del usuario al pedir permisos.

➔ *Nota* : En **mostrarInformacionDispositivoBTLE**, cuando detecta un iBeacon, interpreta la trama con **TramaIBeacon**, mostrando uuid/major/minor/txPower. Esto conecta directamente con lo que publica la placa Arduino (Publicador).

2. TramaIBeacon

La clase `TramaIBeacon` representa y descompone una trama iBeacon recibida en un escaneo Bluetooth LE. Permite acceder a cada campo de la trama (prefijo, UUID, major, minor, txPower, etc.) de forma sencilla.



- **constructor** → recibe la trama bruta del anuncio BLE y la parsea en campos: asigna losBytes, extrae prefijo (0–8), uuid (9–24), major (25–26), minor (27–28) y txPower (29). descompone el prefijo en advFlags (0–2), advHeader (3–4), companyID (5–6), iBeaconType (7) e iBeaconLength (8).
  - ➔ *Nota*: se asume longitud mínima ≈30 bytes; en producción conviene validar antes de parsear.
- **getPrefijo()** → devuelve los primeros 9 bytes del anuncio.

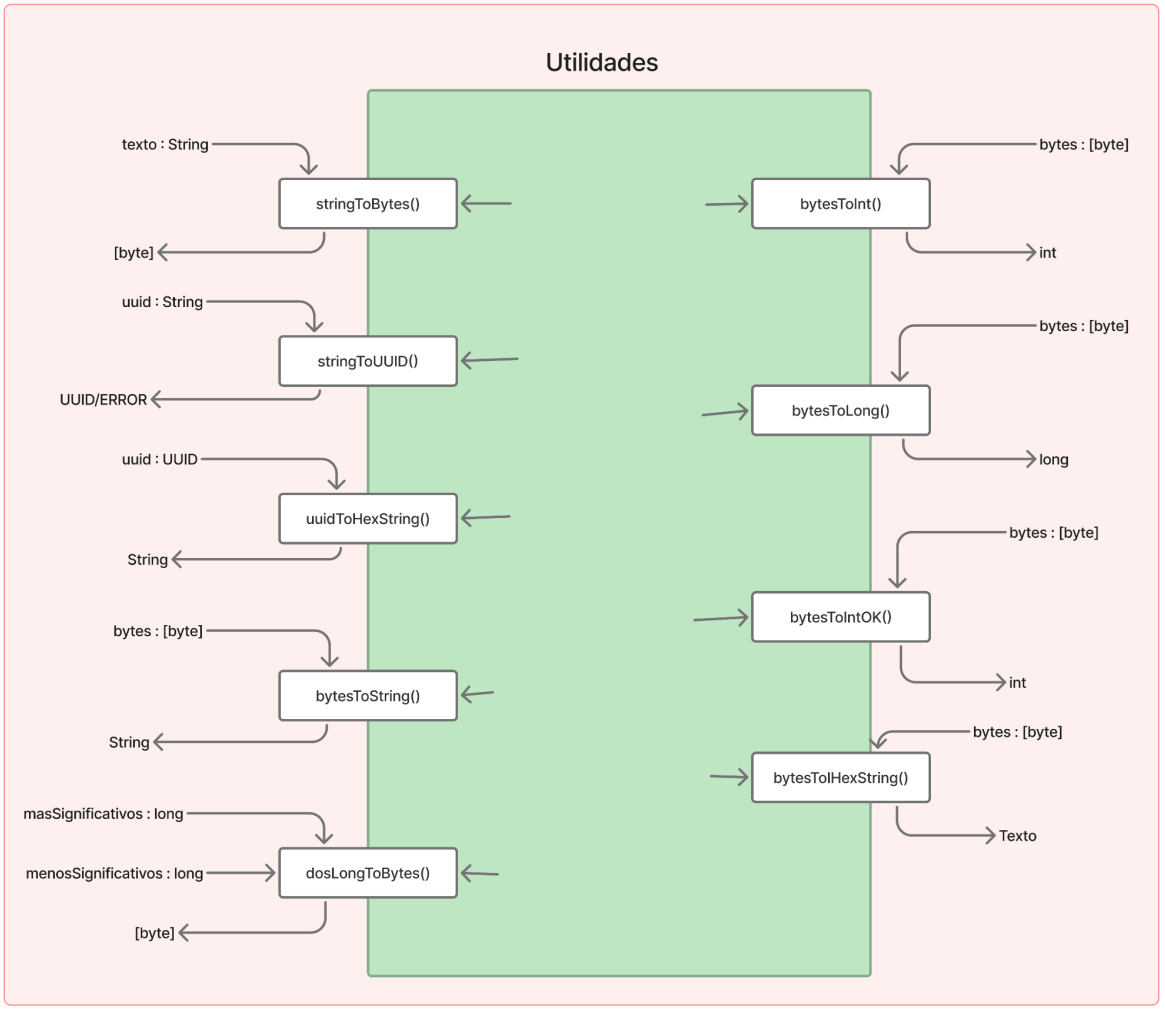
- **getUUID()** → identificador único (16 bytes).

- **getMajor() / getMinor()** → identifican grupo/dispositivo dentro del UUID.
- **getTxPower()** → potencia de transmisión estimada.
- **getLosBytes()** → trama completa en bruto.
- **getAdvFlags(), getAdvHeader(), getCompanyID(), getiBeaconType(), getiBeaconLength()** → subcampos del prefijo según iBeacon.

➔ *Nota* : La clase no interpreta los valores (p.ej. no convierte `major` en entero), sino que devuelve los arrays de bytes directamente.

3. Utilidades

Clase de utilidades estática que facilita conversiones entre distintos formatos de datos usados en el tratamiento de tramas iBeacon. No tiene atributos ni constructor, ya que todos sus métodos son estáticos.



- **stringToBytes(String texto)** → convierte un texto en un array de bytes.

- **stringToUUID(String uuid)** → transforma un string de 16 caracteres en un objeto UUID.
- **uuidToString(UUID uuid)** → convierte un UUID en texto plano.
- **uuidToHexString(UUID uuid)** → convierte un UUID en su representación hexadecimal.
- **bytesToString(byte[] bytes)** → transforma bytes en caracteres legibles.
- **dosLongToBytes(long msb, long lsb)** → empaqueta dos números largos en un array de 16 bytes.
- **bytesToInt(byte[] bytes)** → interpreta los bytes como un número entero.
- **bytesToLong(byte[] bytes)** → interpreta los bytes como un número largo (long).
- **bytesToIntOK(byte[] bytes)** → versión segura de conversión a entero (máx. 4 bytes, con signo).
- **bytesToHexString(byte[] bytes)** → devuelve los bytes en formato hexadecimal legible, separados por :.

➔ *Nota* : Esta clase se usa como herramienta de apoyo en otras clases para interpretar y mostrar los datos de los beacons.

BBDD

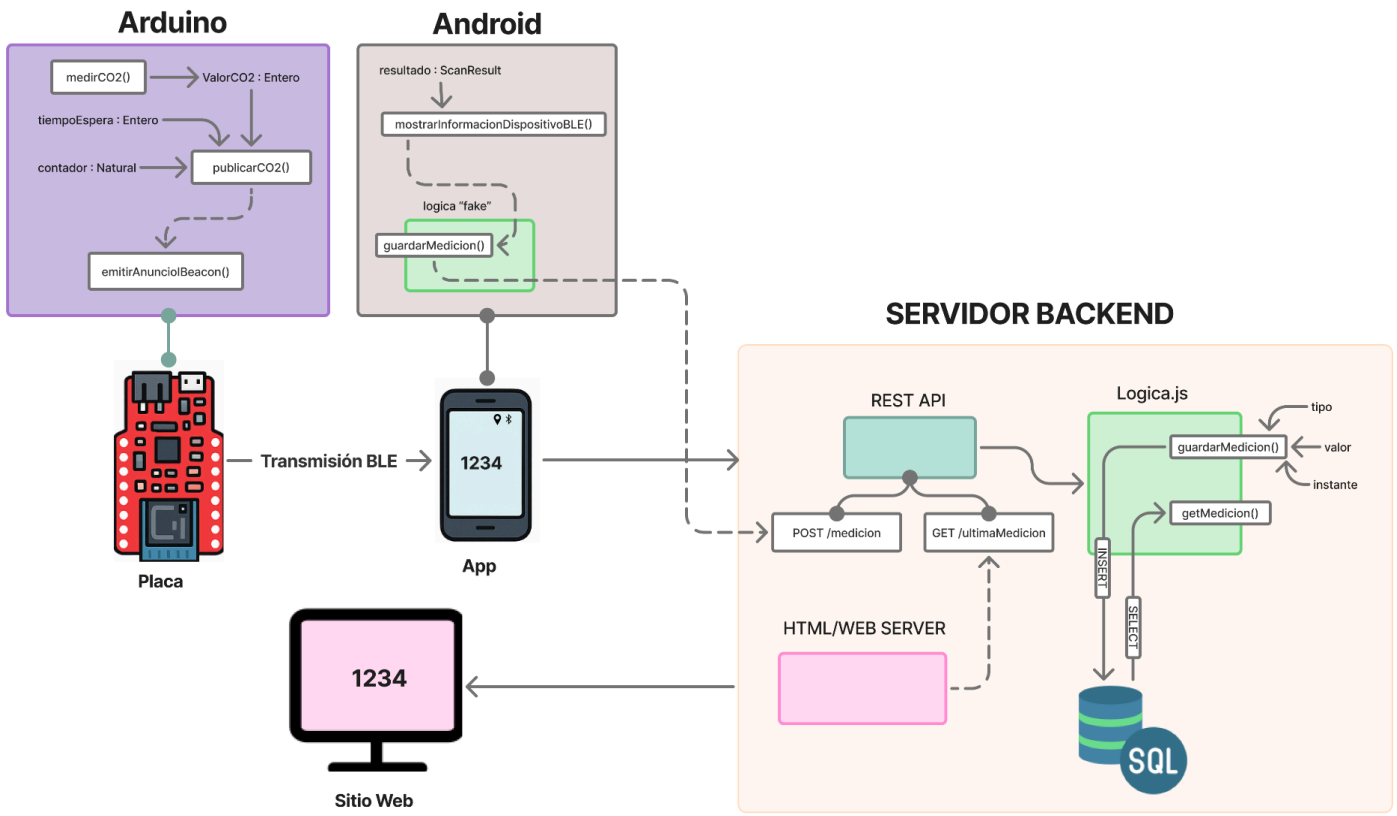
La base de datos almacena todas las mediciones recibidas desde los dispositivos BLE (Arduino + App). Cada registro representa un dato de tipo ambiental (CO_2, temperatura, ruido) con su valor y el instante en que fue capturado.

Mediciones

***id** (entero)	**tipo** (texto)	**valor** (real)	**instante** (fecha en ISO)
1	CO2	450	2025-02-21T15:30:00
3	RUIDO	40	2025-02-21T15:30:10
2	TEMPERATURA	22	2025-02-21T15:30:05

Logica Del Negocio

La lógica del negocio se implementa en el servidor, en el módulo **Logica.js**.
Su función es recibir los datos enviados desde la app Android a través de la **REST API** y gestionarlos en la base de datos.



❖ *Funciones principales:*

- **guardarMedicion(tipo, valor, instante)**

Inserta en la tabla Mediciones un nuevo registro con el tipo de sensor (CO_2, temperatura, ruido), el valor numérico medido y el instante de captura.

- **getMedicion()**

Recupera de la base de datos la última medición registrada. Esta función se usa tanto en la app como en la interfaz web para mostrar el valor más reciente.

❖ *Flujo de trabajo:*

1) La app Android detecta un valor BLE y llama a `guardarMedicion()` enviando los datos al servidor.
2) El servidor guarda la medición en SQL mediante la lógica del negocio.

3) Cuando se consulta el endpoint /`ultimaMedicion`, se invoca `getMedicion()`, que devuelve los datos más recientes en formato JSON.