



Documento de diseño

PROYECTO DE APLICACIONES DE BIOMETRÍA Y MEDIO AMBIENTE

Sprint #0

Autor: Santiago Fuenmayor Ruiz

26 de septiembre de 2025

Tabla de contenido

ARDUINO.....3

 INGENIERÍA INVERSA.....4

TELÉFONO7

 INGENIERIA INVERSA.....8

ARDUINO

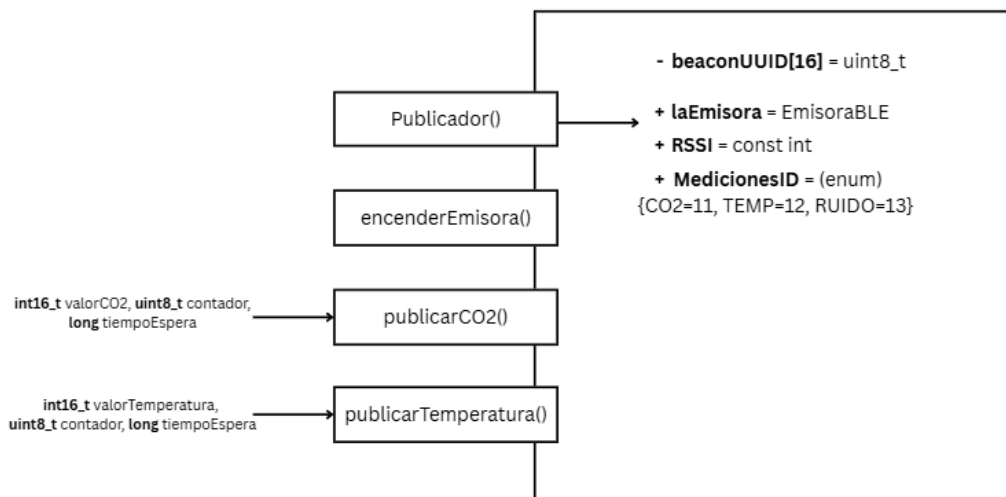
En la parte de Arduino, el código implementa el nodo sensor encargado de simular la lectura de parámetros ambientales y publicarlos mediante Bluetooth Low Energy (BLE) en formato iBeacon. Para ello se estructura en distintas clases con funciones bien definidas:

Medidor se encarga de obtener valores de CO₂ y temperatura (actualmente simulados), **Publicador** gestiona la creación de los paquetes iBeacon y su emisión, y **EmisoraBLE** abstrae la configuración del módulo BLE para encenderlo, detenerlo o iniciar la transmisión. Además, se incluyen clases de apoyo como **PuertoSerie**, utilizada para depuración a través del monitor serie, **LED**, que actúa como indicador visual del envío de datos, y **ServicioEnEmisora**, pensada para manejar servicios y características BLE en futuras ampliaciones.

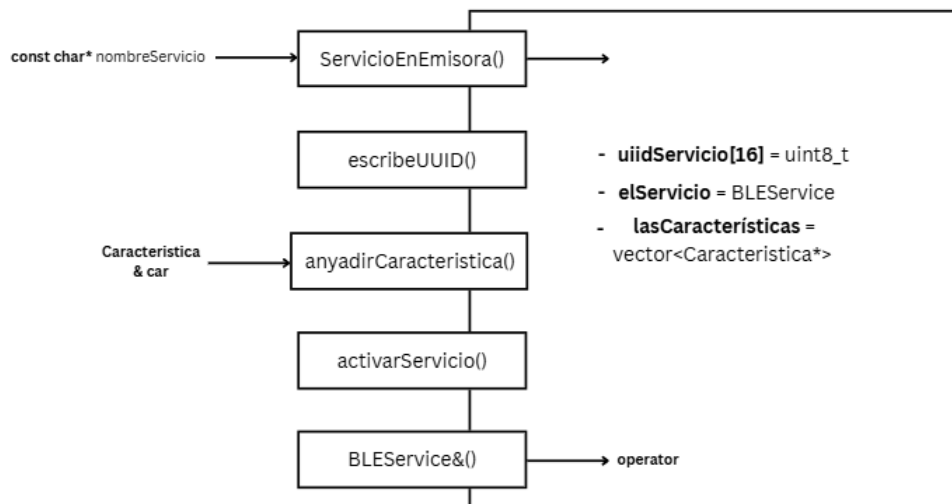
El flujo básico del sistema consiste en inicializar la emisora y el puerto serie, simular las medidas, empaquetarlas en un anuncio iBeacon y transmitirlos durante un tiempo definido, con la posibilidad de indicar el proceso mediante un parpadeo del LED. De este modo, el Arduino actúa como un beacon emisor de datos ambientales, que posteriormente son detectados por la aplicación móvil y enviados al servidor para su almacenamiento y consulta.

INGENIERÍA INVERSA

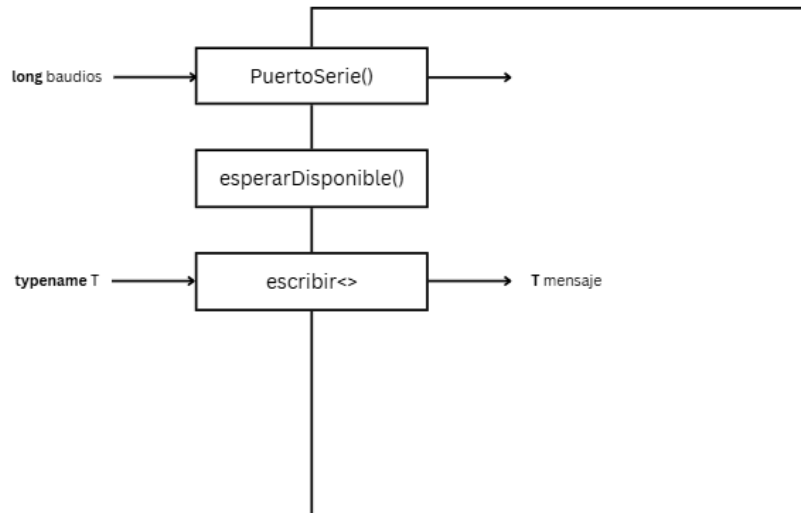
Publicador



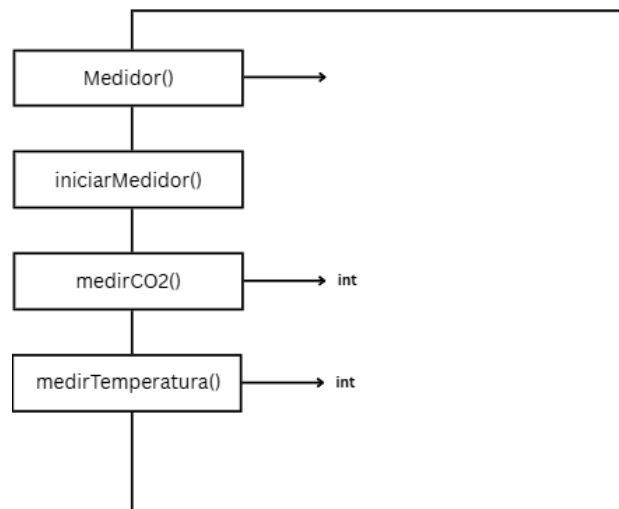
ServicioEnEmisora



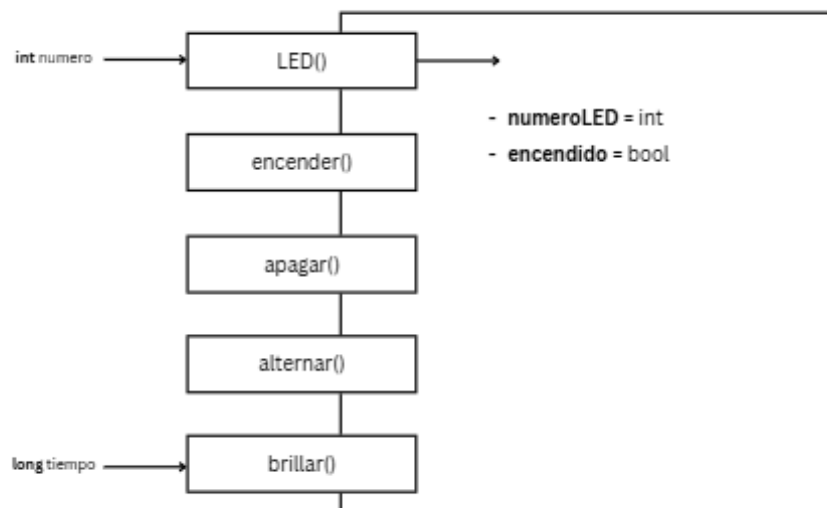
PuertoSerie



Medidor



LED



```

graph TD
    subgraph BibliotecaComunicacionBLE [Biblioteca de comunicación BLE]
        EmisoraBLE()
        encenderEmisora1(encenderEmisora())
        encenderEmisora2(encenderEmisora())
        detenerAnuncio()
        estaAnunciando()
        emitirAnuncionBeacon()
        emitirAnuncioBeaconLibre()
    end

    subgraph BibliotecaComunicacionBLE2 [Biblioteca de comunicación BLE]
        anyadirServicio()
        anyadirServicioConSusCaracteristicas()
        anyadirServicioConSusCaracteristicasYActivar()
        instalarCallbackConexionEstablecida()
        instalarCallbackConexionTerminada()
        getConexion()
    end

    EmisoraBLE() --> anyadirServicio()
    EmisoraBLE() --> anyadirServicioConSusCaracteristicas()
    EmisoraBLE() --> anyadirServicioConSusCaracteristicasYActivar()
    EmisoraBLE() --> instalarCallbackConexionEstablecida()
    EmisoraBLE() --> instalarCallbackConexionTerminada()
    EmisoraBLE() --> getConexion()

    anyadirServicio() --> anyadirServicioConSusCaracteristicas()
    anyadirServicioConSusCaracteristicas() --> anyadirServicioConSusCaracteristicasYActivar()
    anyadirServicioConSusCaracteristicasYActivar() --> instalarCallbackConexionEstablecida()
    instalarCallbackConexionEstablecida() --> instalarCallbackConexionTerminada()
    instalarCallbackConexionTerminada() --> getConexion()

    EmisoraBLE() --> encenderEmisora1
    encenderEmisora1 --> encenderEmisora2
    encenderEmisora2 --> detenerAnuncio()
    detenerAnuncio() --> estaAnunciando()
    estaAnunciando() --> emitirAnuncionBeacon()
    emitirAnuncionBeacon() --> emitirAnuncioBeaconLibre()

```

Diagrama de flujo de la implementación de la interfaz de la biblioteca de comunicación BLE. El diagrama muestra una columna de funciones de la biblioteca de comunicación BLE a la izquierda y una columna de funciones de la biblioteca de comunicación BLE a la derecha. Las funciones de la biblioteca de comunicación BLE a la izquierda son: EmisoraBLE(), encenderEmisora(), encenderEmisora(), detenerAnuncio(), estaAnunciando(), emitirAnuncionBeacon(), and emitirAnuncioBeaconLibre(). Las funciones de la biblioteca de comunicación BLE a la derecha son: anyadirServicio(), anyadirServicioConSusCaracteristicas(), anyadirServicioConSusCaracteristicasYActivar(), instalarCallbackConexionEstablecida(), instalarCallbackConexionTerminada(), and getConexion(). Las flechas indican las dependencias entre las funciones. Las flechas azules indican las dependencias de la biblioteca de comunicación BLE a la izquierda. Las flechas verdes indican las dependencias de la biblioteca de comunicación BLE a la derecha. Las flechas amarillas indican las dependencias de la biblioteca de comunicación BLE a la izquierda y la biblioteca de comunicación BLE a la derecha.

```

graph TD
    activar() --> Carácteristica1[Carácteristica()]
    Carácteristica1 --> Carácteristica2[Carácteristica()]
    Carácteristica2 --> asignarPropiedades[asignarPropiedadesPermisosYTamañoDatos()]
    asignarPropiedades --> escribirDatos[escribirDatos()]
    escribirDatos --> notificarDatos[notificarDatos()]
    notificarDatos --> instalarCallback[instalarCallbackCarácteristicaEscrita()]
    instalarCallback --> activar()

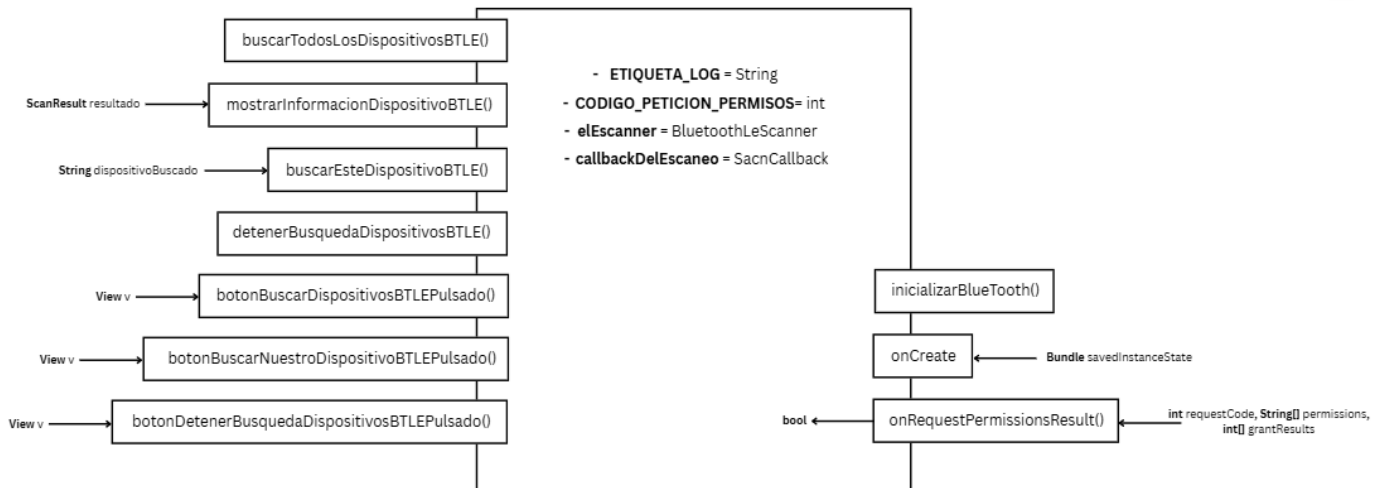
    nombreCarácteristica --> Carácteristica1
    Carácteristica1 --> uuid["- uuidCarácteristica[16] = uint8_t"]
    Carácteristica1 --> laCarácteristica["- laCarácteristica = BLECharacteristic"]
    props["const char*, uint8_t props, SecureMode_t read, SecureMode_t write, uint8_t tam"] --> Carácteristica2
    props --> asignarPropiedades
    str1["const char* str"] --> escribirDatos
    str1 --> notificarDatos
    cb["CallbackCarácteristicaEscrita cb"] --> instalarCallback
  
```

TELÉFONO

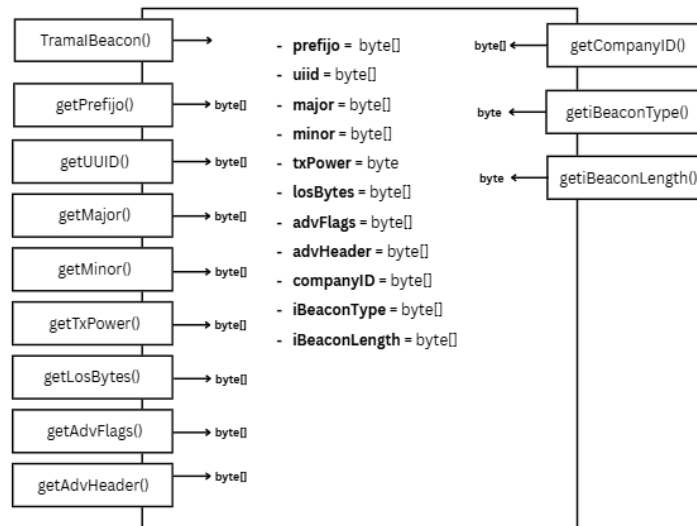
En la parte de **Android**, la aplicación funciona como receptor de los anuncios iBeacon enviados por el Arduino, interpretando las tramas y mostrando los valores medidos. Se apoya en las clases MainActivity para la gestión de la interfaz y detección, TramaBeacon para decodificar los paquetes recibidos y Utilidades para funciones auxiliares, mientras que el AndroidManifest.xml define los permisos de Bluetooth y localización. En este Sprint 0 su funcionamiento es básico o simulado, mostrando cómo se prepararían los datos para su posterior envío al servidor.

INGENIERIA INVERSA

MainActivity



TramalBeacon



Utilidades

