Diseños Java
SPRINT 0

PROYECTO APLICACIONES DE BIOMETRÍA Y MEDIO AMBIENTE

Nombre del Alumno: Alan Guevara Martínez

Enlace Git: https://github.com/ALANGMupv/ProyectoBiometria2025.git

Enlace trello:
https://trello.com/invite/b/68c9273bf08b76ab09c96593/ATTI19e0d5a059994ed24df6c6455b577bae3E1C4F40/gtiproyectobiometria

Curso: 2025/2026

ÍNDICE / TABLA DE CONTENIDOS

Introducción 3
MainActivity (archivo principal) 3
Clase Utilidades 4
Clase TramaIBeacon 5
LogicaFake (clase auxiliar creada por mí) 7

TABLA DE ILUSTRACIONES / DISEÑOS

Ilustración 1: Diseño MainAcitivity 4
Ilustración 2: Diseño Utilidades 5
Ilustración 3: Diseño TramaIBeacon 6
Ilustración 4: Diseño LogicaFake 7

Introducción

En este documento se presentan los **diseños de las clases en Java** que forman parte del proyecto Android con soporte **Bluetooth Low Energy (BLE)**.
Cada clase cumple una responsabilidad específica:

- **MainActivity**: punto de entrada de la app, gestiona permisos, Bluetooth y búsqueda de dispositivos.
- **Utilidades**: métodos auxiliares de conversión (bytes, UUID, strings, enteros).
- **TramaIBeacon**: representación de la trama iBeacon recibida al escanear dispositivos BLE (separa la trama).

Los diagramas completos de diseño pueden consultarse en el archivo de Figma correspondiente a **Ingeniería Inversa Java (Diseño)**: https://www.figma.com/board/5gclkZaVUzniiwrtNu5U6k/Ingenier%C3%ADa-Inversa-Java--Dise%C3%B1o-?node-id=0-1&t=RU2SPKPhdb7Djyhh-1

MainActivity (archivo principal)

Clase que actúa como controlador de la aplicación. Gestiona la inicialización del Bluetooth, el manejo de permisos, y coordina el escaneo de dispositivos BLE.

Funciones principales:

- **onCreate(Bundle savedInstanceState)**: inicializa la app y activa Bluetooth.
- **inicializarBlueTooth()**: obtiene el adaptador y escáner BLE, solicita permisos en función de la versión de Android.
- **buscarTodosLosDispositivosBTLE()**: inicia un escaneo general de dispositivos BLE.
- **buscarEsteDispositivoBTLE(String nombre):** inicia un escaneo filtrado por nombre, procesa la trama iBeacon del dispositivo al detectarlo, envía sus datos al servidor y muestra avisos de recepción y pérdidas.
- **mostrarInformacionDispositivoBTLE(ScanResult resultado)**: imprime en logs la información del dispositivo detectado, parseando la trama iBeacon.
- **detenerBusquedaDispositivosBTLE()**: detiene el escaneo en curso.
- **botonBuscarDispositivosBTLEPulsado(View v)**: vinculado al botón que inicia un escaneo general.
- **botonBuscarNuestroDispositivoBTLEPulsado(View v)**: vinculado al botón que busca un dispositivo concreto.
- **botonDetenerBusquedaDispositivosBTLEPulsado(View v)**: vinculado al botón que detiene la búsqueda.
- **onRequestPermissionsResult(...)**: gestiona la respuesta a las solicitudes de permisos.

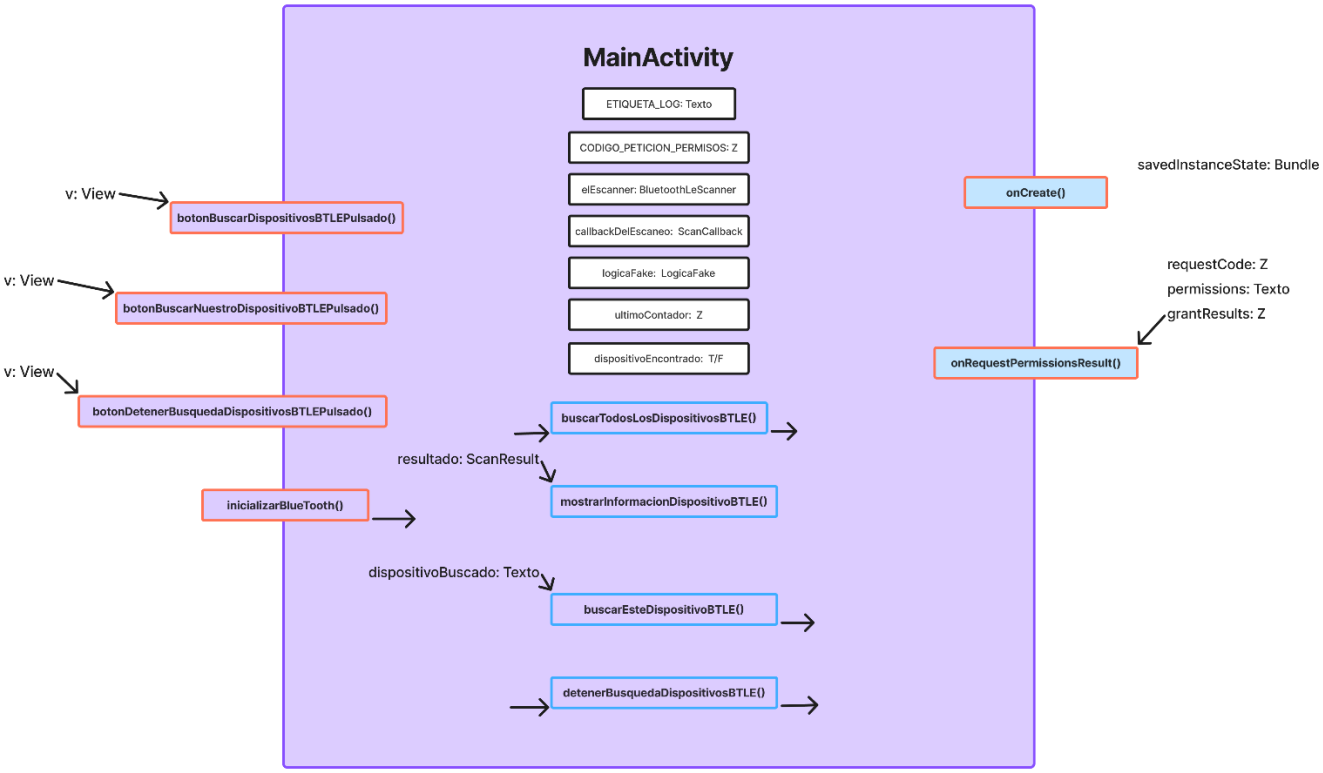

Ilustración 1: Diseño MainAcitivity

Clase Utilidades

Contiene métodos estáticos de conversión y ayuda para tratar cadenas, bytes y UUID. Facilita la interpretación de datos provenientes de tramas BLE.

Funciones principales:

- **stringToBytes(String texto)**: convierte un texto en array de bytes.
- **stringToUUID(String uuid)**: convierte un string de 16 caracteres en un UUID.
- **uuidToString(UUID uuid)**: convierte un UUID a String.
- **uuidToHexString(UUID uuid)**: convierte un UUID a representación hexadecimal.
- **bytesToString(byte[] bytes)**: convierte un array de bytes a texto.
- **dosLongToBytes(long msb, long lsb)**: combina dos long en un array de 16 bytes.
- **bytesToInt(byte[] bytes)**: convierte un array de bytes a entero.
- **bytesToLong(byte[] bytes)**: convierte un array de bytes a long.
- **bytesToIntOK(byte[] bytes)**: convierte manualmente bytes a entero controlando desbordamientos.
- **bytesToHexString(byte[] bytes)**: convierte bytes en una cadena hexadecimal legible.

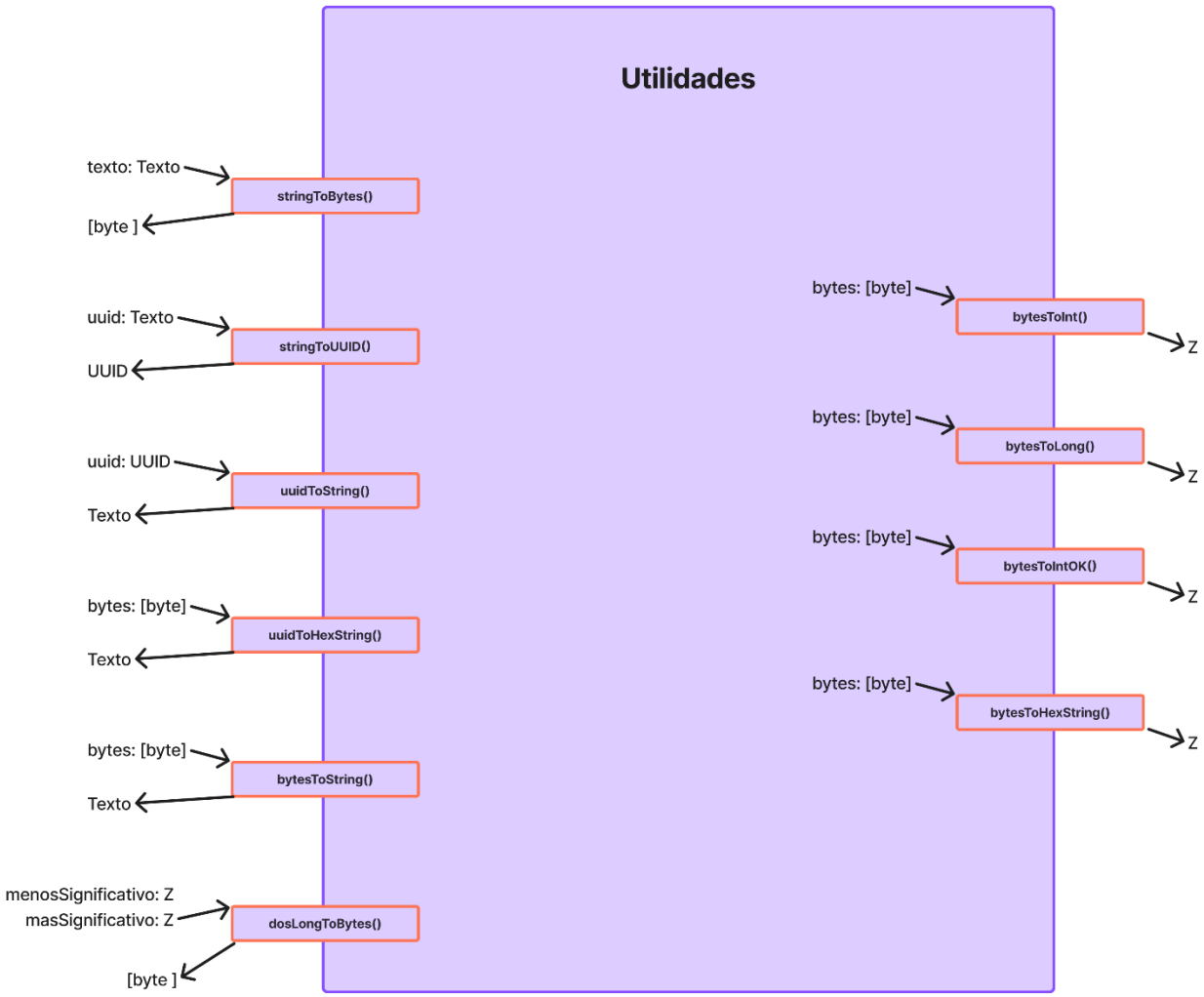

Ilustración 2: Diseño Utilidades

Clase TramaIBeacon

Representa y parsea los bytes de una **trama iBeacon**, separando sus diferentes campos. Se crea a partir de los bytes recibidos en un anuncio BLE.

Atributos principales:

- prefijo (9 bytes)
- uuid (16 bytes)
- major (2 bytes)
- minor (2 bytes)
- txPower (1 byte)
- advFlags (3 bytes)
- advHeader (2 bytes)
- companyID (2 bytes)
- iBeaconType (1 byte)

- iBeaconLength (1 byte)
- losBytes (array completo recibido)

Funciones principales (getters):

- **getPrefijo()**
- **getUUID()**
- **getMajor()**
- **getMinor()**
- **getTxPower()**
- **getLosBytes()**
- **getAdvFlags()**
- **getAdvHeader()**
- **getCompanyID()**
- **getiBeaconType()**
- **getiBeaconLength()**

Constructor:

- **TramaIBeacon(byte[] bytes)**: recibe los bytes de la trama y extrae todos los campos.

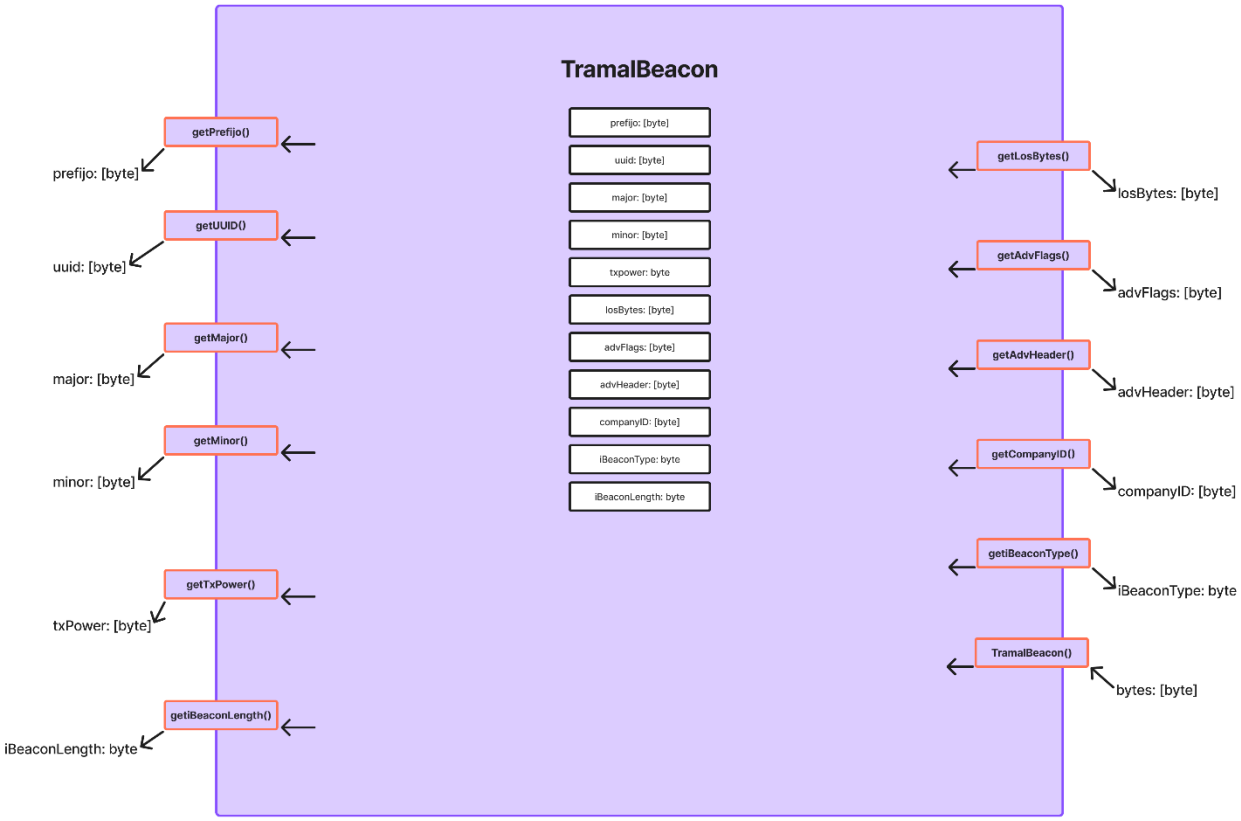

Ilustración 3: Diseño TramaIBeacon

LogicaFake (clase auxiliar creada por mí)

Clase que actúa como capa de lógica para el envío de datos al servidor. Se encarga de construir un objeto JSON con la información recibida y realizar una petición HTTP POST a la API.
Se ejecuta en un hilo independiente para no bloquear el hilo principal de la app (UI thread).
Funciones principales:

- **guardarMedicion(String uuid, int gas, int valor, int contador):** genera un JSON con los datos de la medición (uuid, gas, valor, contador), lo envía por POST a la API definida en API_URL, muestra logs de depuración y maneja posibles errores en la conexión.

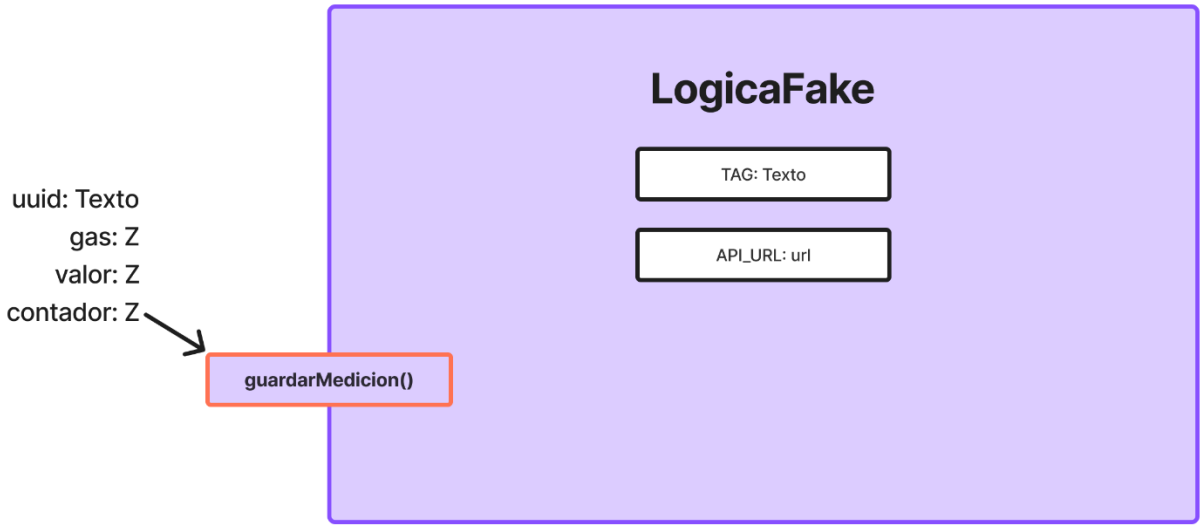

Ilustración 4: Diseño LogicaFake