UNIVERSIDAD DE OVIEDO

TRABAJO FIN DE GRADO

Aplicación móvil para el control de la presión y temperatura de los neumáticos de un vehículo (TPMS) usando Bluetooth

Ignacio Bermejo Álvarez

Tutor del proyecto
José Ramón Arias García

Grado en Ingeniería Informática del Software
Febrero 2021

Agradecimientos

Quiero agradecer a mis docentes y en especial a mi tutor José Ramón por su ayuda y apoyo. Siempre atento y dispuesto cuando lo he necesitado.

También agradecer a mis padres por confiar en mí durante estos años de carrera. Su infinita paciencia es algo que nunca olvidaré. Gracias por todo.

Resumen

Bluetooth es una de las tecnologías de comunicación inalámbrica entre dispositivos más usada hoy en día y como en cualquier proceso de intercambio de información, la seguridad a la hora de transmitir el mensaje es primordial.

El principal objetivo de este proyecto es desafiar la seguridad de unos sensores de temperatura y presión para neumáticos que utilizan Bluetooth. La función de este producto consiste en proporcionar al usuario un sistema para monitorizar en tiempo real estos parámetros mediante el teléfono móvil. Para ello el fabricante proporciona una aplicación gratuita.

Una vez los datos han sido interceptados, se ha creado una aplicación móvil que sustituye a la proporcionada por el fabricante y añade ciertas funcionalidades. Entre ellas, cabe destacar, la posibilidad de aumentar el número de sensores monitorizados y una mejora en la interfaz que permite al usuario elegir entre diferentes tipos de vehículos.

BLE TMPS es el resultado de este proceso, el cual me ha permitido aprender a desarrollar en Android, sin conocimiento previo, y comprender el funcionamiento de Bluetooth para poder aplicarlo en futuros trabajos.

Palabras Clave

Bluetooth, BLE, TPMS, aplicación, automoción, vehículo, neumáticos, Android, monitorización en tiempo real, temperatura, presión.

Abstract

Bluetooth is one of the most common wireless technologies used nowadays to exchange data between devices, and, as it is the case with any other systems designed to exchange information, security is paramount.

The main objective of this project is to challenge the security of the Bluetooth operating sensors that measure temperature and pressure of tyres. The aim of this product is to offer the user a system to monitor these parameters in real time with a mobile phone. To this effect, the manufacturer provides a free application (App).

Once the data has been intercepted, a mobile application is created that replaces the one provided by the manufacturer, and it offers additional features. Among them, it is worth mentioning the possibility of increasing the number of sensors that are monitored and also the improvement in the interface that allows the user to choose between different vehicles.

BLE TMPS is, therefore, the result of this process, which has enabled me to learn about Android development, without having any previous knowledge, and to fully understand how Bluetooth works, in order to apply this knowledge in future projects.

Keywords

Bluetooth, BLE, TPMS, application, automotive, vehicle, tyre, Android, real-time monitoring, temperature, pressure.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1. Introducción

1.1 JUSTIFICACIÓN DEL PROYECTO

Bluetooth es una tecnología utilizada en todo el mundo. Hoy en día todos los dispositivos inteligentes lo integran. Su uso es tan sencillo -incluso a veces automático- que cualquier persona, en todos los rangos de edad, la puede utilizar.

Ahora bien, aunque aparentemente su función es tan simple como permitir escuchar música en unos altavoces o unos cascos, compartir una pantalla con un televisor, etc.… el usuario está transmitiendo y recibiendo datos por el aire, por lo que es una tecnología que no está libre de ataques. Siempre que se escucha la palabra "hackear", la gente lo relaciona con ataques a ordenadores, pero no siempre es así. Aunque no suele ser común encontrar casos de hackeos a móviles, esto está cambiando.

Otro concepto a tener en cuenta es que no es necesario estar conectado a internet para sufrir un ataque. La verdad es que casi cualquier dispositivo electrónico es susceptible de ser hackeado y Bluetooth se presenta como una de las puertas de entrada.

Aprovechando esta debilidad se diseña un proyecto que parte de la necesidad de hackear un dispositivo Bluetooth para interceptar la información transmitida.

El objetivo final que se persigue es crear una aplicación móvil que monitorice la presión y la temperatura de los neumáticos de un vehículo haciendo uso de un Sistema de monitorización de presión (TPMS) por Bluetooth facilitado por la Universidad. Este sistema está compuesto por unos sensores que se instalan en las válvulas de los neumáticos y una aplicación móvil que permite al usuario controlar el estado de estos.

En definitiva, lo que se pretende con este proyecto es cuestionar la calidad de un producto tanto desafiando la seguridad de su hardware como mejorando la interfaz y la funcionalidad de su software.

1.2 OBJETIVOS DEL PROYECTO

Los objetivos que persigue este proyecto son los siguientes:

1. Interceptar comunicaciones entre unos sensores de temperatura y presión para neumáticos que utilizan BLE (Bluetooth Low Energy o Bluetooth de baja energía) y una aplicación móvil.
2. Averiguar el proceso de emparejamiento entre los sensores y la aplicación.
3. Comprender el contenido del mensaje y descifrarlo en caso de estar encriptado.
4. Desarrollar una aplicación móvil que monitorice la presión y la temperatura proporcionadas por los sensores.
5. Si los datos transmitidos aportan otro tipo de información mostrársela al usuario.
6. Permitir al usuario utilizar los sensores en vehículos con distinto número ruedas (la aplicación del fabricante solo permite vehículos de 4 ruedas).
7. Permitir al usuario utilizar un mayor número de sensores simultáneamente (la aplicación del fabricante solo permite 4 sensores simultáneamente).

1.3 ESTUDIO DE LA SITUACIÓN ACTUAL

A la hora de desarrollar la aplicación, se ha estudiado como punto de partida la proporcionada por el fabricante del TPMS, con el fin de determinar las funcionalidades básicas y las posibles mejoras.

La oferta de aplicaciones de este tipo viene condicionada por el hardware, es decir, cada juego de sensores tiene su propia aplicación. Teniendo esto en cuenta, se ha realizado un estudio de las alternativas con el objetivo de encontrar distintas funcionalidades (siempre que no dependan del hardware) que se puedan englobar en una misma aplicación.

1.3.1 Aplicación del fabricante

TPMSII [1] es una aplicación gratuita para monitorización de temperatura y presión de neumáticos utilizando Bluetooth. Tiene soporte para Android (4.3 o superior) y iPhone (4s o superior) y requiere Bluetooth (4.0 o superior). Como todas las aplicaciones de este tipo, tiene una serie de funcionalidades básicas: monitorización en tiempo real de hasta cuatro sensores, control de los límites de advertencia de la temperatura y la presión, emparejamiento de sensores con las ruedas, intercambio de ruedas, gestión de distintos perfiles de vehículos y distintas unidades de medida de presión y temperatura.

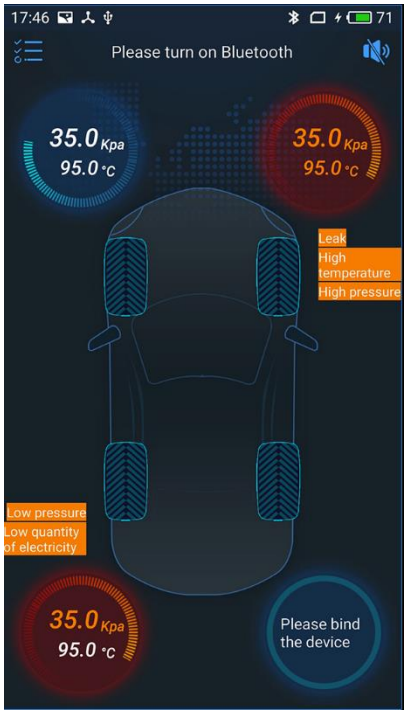


Figura 1.2: Home de la aplicación TPMSII

Figura 1.2: Menú de la aplicación TPMSII

Aparte de las funcionalidades básicas comunes en todas las aplicaciones de monitorización de TPMS, TPMSII ofrece lo siguiente:

- Monitorización en segundo plano: la aplicación sigue escaneando los sensores aun estando en segundo plano o la pantalla del móvil bloqueada.
- Mensajes de alerta por voz: la aplicación permite informar al usuario sobre posibles pérdidas de presión o valores altos de temperatura aun cuando está en segundo plano.
- Tres modos de emparejamiento: la aplicación ofrece los siguientes modos de emparejamiento:
    - Automático: el emparejamiento se realiza al conectar el sensor por primera vez cuando esta opción ha sido seleccionada.
    - Manual: emparejamiento ingresando el identificador del sensor.
    - Código QR: permite emparejar los cuatro sensores a la vez escaneando un código QR facilitado por el fabricante.

Tras haber probado la aplicación, se plantean las siguientes posibles mejoras:

- Permitir al usuario configurar distintos tipos de vehículos con distinto número de ruedas. El vehículo por defecto es de cuatro ruedas.
- Permitir vehículos de más de cuatro ruedas. Esto permitiría al usuario adquirir más de un juego de sensores y utilizarlos simultáneamente.

- Ampliar la información ofrecida al usuario. Los TPMS ofrecen más datos, además de la temperatura y la presión, como pueden ser: batería y calidad de señal.

1.3.2 Evaluación de aplicaciones alternativas

1.3.2.1 Multi Wheel BLE TPMS

[2]

Ventajas

- Configuración de distintos tipos de vehículos con distinto número de ruedas.
- Configuración de vehículos de más de cuatro ruedas, permitiendo hasta un total de 20.
- Personalización del perfil del vehículo. Permite añadir imagen, marca y modelo.
- Permite exportar los datos a un fichero.

Inconvenientes

- Pequeños fallos al usarla como bloqueo de ciertas vistas.
- Interfaz mejorable. Algunos elementos se superponen.
- Falta de un manual de instrucciones.

Figura 1.4: Home de la aplicación Multi Wheel

Figura 1.4: Menú de selección de vehículo de la aplicación Multi Wheel

1.3.2.2 StoreBao USB TPMS

[3]

Ventajas

- Interfaz limpia y sencilla.
- Ofrece más información al usuario: estado de la batería y mensaje de estado del neumático.

Inconvenientes

- Solo permite vehículos de cuatro ruedas.
- Orientación horizontal obligatoria. La mayoría de los soportes para móviles son en vertical.
- Pequeños fallos: notificación siempre activa, incluso al cerrar la aplicación.
- Un único modo de emparejamiento. La aplicación solo ofrece el emparejamiento automático.

Figura 1.5: Home de la aplicación StoreBao

Capítulo 2. Aspectos Teóricos

2.1 TPMS

"*Tire-Pressure Monitoring System*" o Sistema de monitorización de la presión de los neumáticos [4] es un sistema electrónico diseñado para monitorizar la presión del aire en los neumáticos en tiempo real. Su función principal es avisar al conductor en caso de pérdida de presión cuando alcanza un nivel establecido por el usuario.

Desde noviembre de 2014 este sistema es obligatorio en todo Europa para turismos, caravanas y vehículos con neumáticos antipinchazos. En Estados Unidos desde 2007 y en Asia desde 2014. Algunos de los requisitos que impone la legislación europea [5] a los TPMS son:

- Operar desde una velocidad de 40 km/h o menos, hasta la velocidad máxima del vehículo.
- Avisar mediante testigo luminoso como máximo a los 10 minutos desde la reducción de presión.
- Considerar reducción de presión cuando el neumático está a un 20% de la presión indicada por el fabricante o a una presión mínima de 1,5 bar.

Figura 2.1: Icono oficial de TPMS

Los TPMS se dividen en dos tipos:

- Directo (dTPMS): la medición se realiza mediante sensores colocados en la rueda, ya sea en el interior de la válvula o en el exterior. Existen dos tipos de sensores actualmente, clasificados según el medio de transmisión de datos. Estos son:

- Por radiofrecuencia: el sensor transmite a una centralita que muestra los datos al usuario.

Figura 2.2: TPMS por radiofrecuencia con sensores exteriores

- Por Bluetooth: el sensor transmite al móvil del usuario y mediante una aplicación proporcionada por el fabricante se monitorizan los datos. **Este será el modelo utilizado en este proyecto**.

Figura 2.3: TPMS por Bluetooth

- Indirecto (iTPMS): no emplea sensores físicos, sino que mide la presión a partir de la velocidad de giro de la rueda además de otros valores que se obtienen de forma externa. Suele estar integrado en la centralita del ABS y el ESP.

2.2 BLUETOOTH LOW-ENERGY (BLE)

Es un protocolo inalámbrico que nos permite comunicarnos entre dispositivos a un nivel muy bajo de consumo y a una velocidad de transferencia de datos muy baja. Aparece como un protocolo Bluetooth para dispositivos con una batería limitada, como pueden ser smartphones o dispositivos IoT (*Internet of Things*). La comunicación se realiza en la banda de frecuencia de 2.4GHz con tasas de transferencia de datos de 1Mbps.

2.2.1 Arquitectura

BLE [6] está formado por diferentes capas que interactúan entre sí. Cada una de ellas cumple una función y juntas forman los que se conoce como pila de protocolos. Esta se divide en tres partes:

- Aplicación: cubre determinados casos de uso y se encarga de manipular los datos. **La segunda parte de este proyecto consistirá en el desarrollo de esta capa**.

- Host: capas superiores de la pila.
    - Perfil genérico de acceso (GAP).
    - Perfil genérico de atributo (GATT).
    - Protocolo de atributo (ATT).
    - Protocolo de gestión de seguridad (SMP).
    - Protocolo de control lógico y adaptación de enlace (L2CAP).
    - Interfaz de control del host (HCI), en el lado del host.

- Controlador: capas inferiores de la pila.
    - Interfaz de control del host (HCI), en el lado del controlador.
    - Capa de enlace (LL).
    - Capa física (PHY).

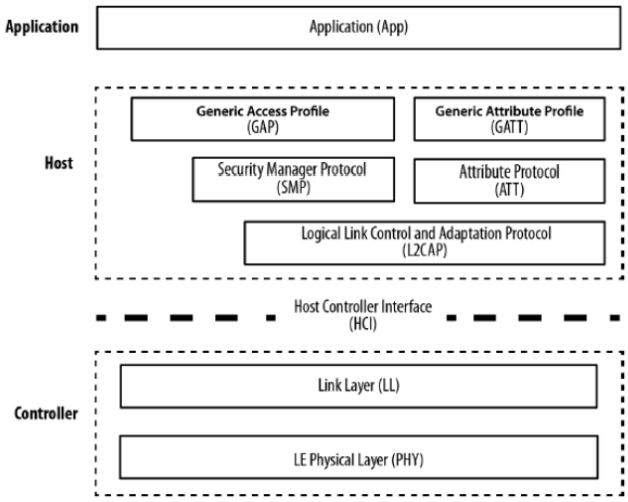


Figura 2.4: Pila de protocolos BLE

2.2.2 Canales

BLE utiliza 40 canales de radiofrecuencia con un espacio entre canales de 2MHz. Tres de ellos se denominan ADV o "*Advertisement*", es decir, canales que permiten a los dispositivos anunciarse para por ejemplo realizar una petición de conexión. Estos canales son el 37, 38 y 39. Los 37 restantes se utilizan para el envío de datos.

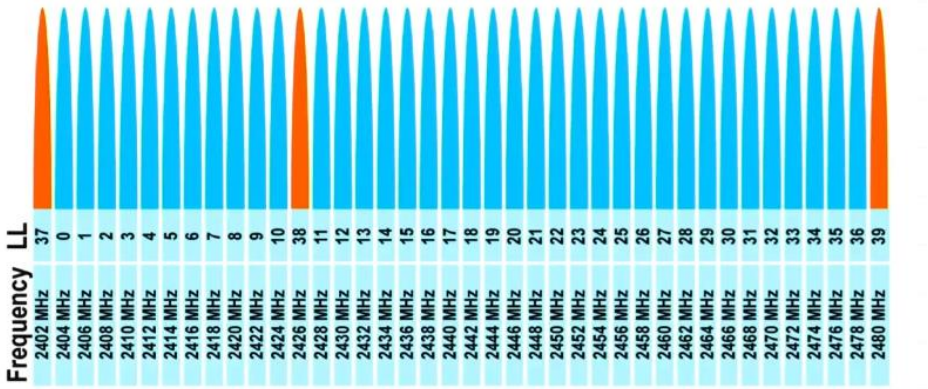


Figura 2.5: Canales en la frecuencia 2.4GHz en BLE

La transmisión de datos se realiza mediante saltos entre canales, es decir, se envían un número determinado de bytes por un canal, se cambia de canal, se envían más bytes, y así hasta que se completa la transmisión.

Para llevar a cabo este proceso se utiliza un patrón de intervalo de salto mediante el cual el emisor y el receptor acuerdan una serie de valores al establecer la conexión. Estos valores son:

- Intervalo de salto: tiempo en milisegundos que se permanece en un canal.
- Incremento de salto: número de canales que se van a saltar.

Otro elemento importante en la transmisión es el Mapa de Canales (*Channel Map*), el cual nos indica en una conexión cuales son los canales que se van a utilizar. Si un canal que se va a utilizar está bloqueado en el mapa de canales, el canal se reasigna a un canal disponible mediante un algoritmo de reasignación.

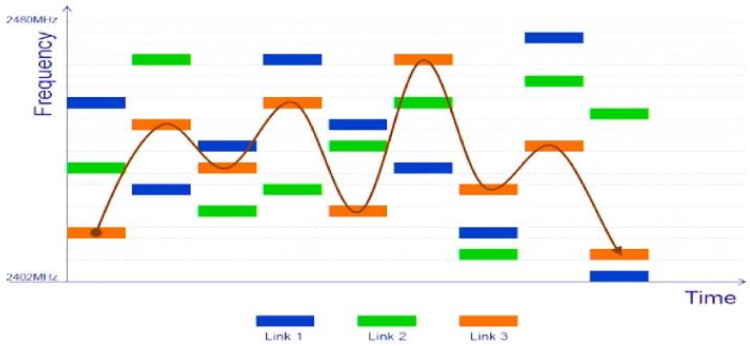


Figura 2.6: Mapa de canales en BLE

Finalmente, mediante un algoritmo de salto se calcula qué canal va a ser el siguiente. Actualmente existen 2 algoritmos:

- Selección de canal #1: los canales se seleccionan secuencialmente utilizando el incremento de salto. Utilizado en BLE versión 4.x y 5.

$$canal = (canal + incremento\ de\ salto)\ mod\ 37$$

- Selección de canal #2: se calcula a partir del contador de eventos y la dirección de acceso. Es semi aleatorio. Utilizado en BLE versión 5.

2.2.3 Roles

Tipos de roles que puede desempeñar un dispositivo. Es posible que operen en más de uno a la vez:

- Emisor (anunciante): dispositivo que envía mensajes de anuncio.
- Observador (escaneador): dispositivo que escanea paquetes de anuncio.

- Periférico (esclavo): dispositivo que espera a que alguien le pida una conexión y la acepta.
- Central (maestro): dispositivo que realiza el escaneo activo y manda el paquete de conexión al esclavo.

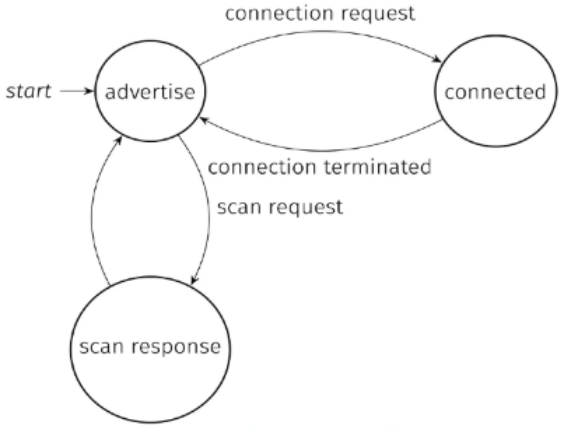


Figura 2.7: Rol esclavo en BLE

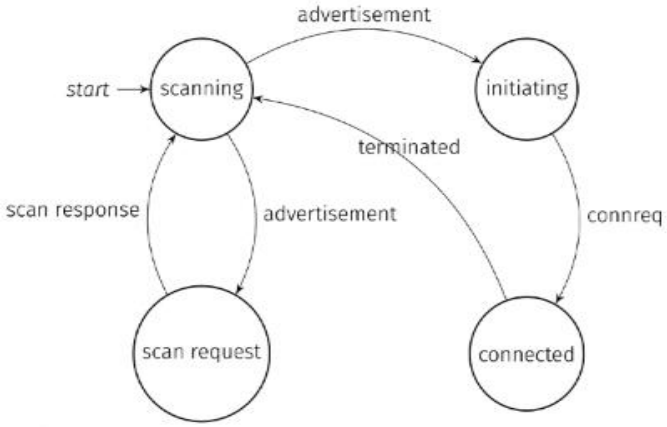


Figura 2.8: Rol maestro en BLE

2.2.4 Paquetes

El protocolo BLE tiene un único formato de paquete y se divide en dos tipos: de anuncio y de datos.

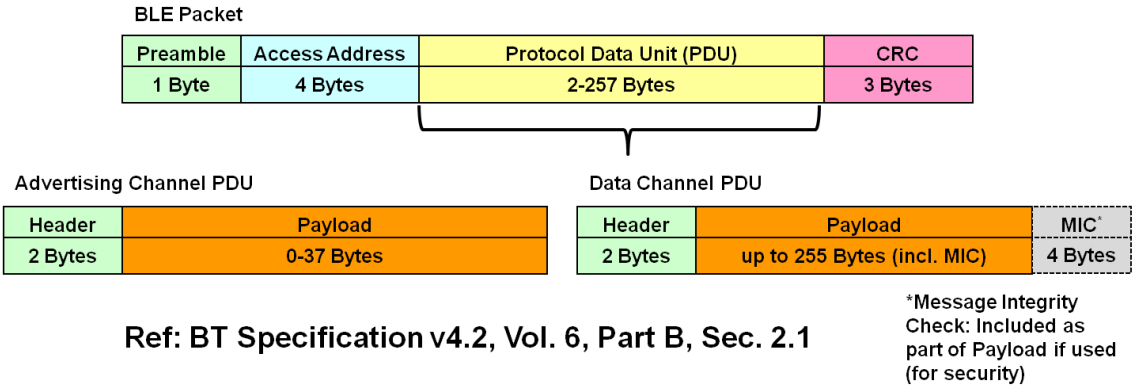


Figura 2.9: Paquetes BLE

Como se puede ver en la figura el paquete tiene la siguiente estructura:

- Preámbulo: 1 byte utilizado para la sincronización y la estimación de tiempo en el receptor.
- Dirección de acceso: 4 bytes utilizados como código de correlación por parte de los dispositivos conectados al canal físico. En los paquetes de anuncio es un valor fijo (0x8E89BED6) mientras que en los de datos es distinto para cada conexión.
- Unidad de datos de protocolo (PDU): de 2 a 257 bytes que dependen del tipo de paquete.
- Verificación de redundancia cíclica (CRC): código de detección de errores usado para detectar cambios accidentales en los datos.

2.2.4.1 De anuncio

Los PDUs de anuncios se utilizan para descubrir esclavos y conectarse a ellos o para transmitir datos a aplicaciones que no requieren una conexión completa. Se subdividen en tres tipos:

- De anuncio: según las características del receptor pueden ser:
    - Conectable: puede iniciar una conexión.
    - Escaneable: puede emitir una petición de escáner.
    - Directivo: está dirigido a un dispositivo en particular.

Tipo	Características
ADV_IND	Conectable, escaneable, no directivo
ADV_DIRECT_IND	Conectable, no escaneable, directivo
ADV_NONCONN_IND	No conectable, no escaneable, no directivo
ADV_SCAN_IND	No conectable, escaneable, no directivo

Tabla 1: Tipos de PDUs de anuncio

- De escaneo:
  - SCAN_REQ: respuesta a un ADV escaneable.
  - SCAN_RSP: respuesta a un SCAN_REQ.

- De inicio de conexión:
  - CONNECT_REQ: petición de conexión enviada a un dispositivo en estado de anuncio.

2.2.4.2 De datos

Los PDUs de datos se envían una vez se ha establecido la conexión entre el maestro y el esclavo. A continuación, se detalla su estructura:

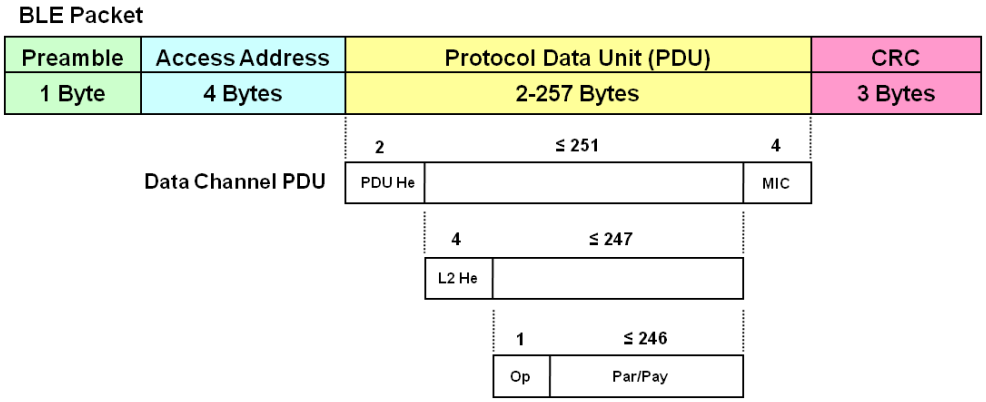


Figura 2.10: Estructura del Protocol Data Unit (PDU)

- PDU He (*Data PDU Header*): cabecera de paquetes de datos. 2 bytes.
- MIC (*Message Integrity Check*): 4 bytes opcionales para controlar integridad del mensaje.
- L2 HE (*L2CAP Header*): 4 bytes de cabecera del protocolo de control lógico y adaptación de enlace.
- Op (*ATT Operation Code*): 1 byte para el código de operación del protocolo de atributo (ATT)
- Par/Pay (*ATT Parameters & Payload*): hasta 246 bytes que contendrán los parámetros y la carga útil de protocolo ATT. **Esta parte es en la que se encuentran los datos a transmitir.**

Capítulo 3. Planificación del Proyecto y Resumen de Presupuestos

3.1 PLANIFICACIÓN

En este apartado se detalla la planificación final de este proyecto haciendo uso de la herramienta Microsoft Project (ver 5.3.2.5). Se muestran las actividades principales junto a su duración y un resumen. Para más información se recomienda abrir el archivo adjunto a este documento: *Planificación_TFG.mpp*.

Al inicio de este proyecto se realizó una planificación inicial con una estimación de la duración y las tareas que se iban a llevar a cabo. Se ha ido modificando a lo largo del proceso para ajustar los tiempos al resultado final.

El tiempo total de trabajo es de 389 horas, repartido entre principios de octubre de 2020 y principios de junio de 2021. La longitud y demora del proyecto se debe a cambios en la duración de las jornadas y una serie de interrupciones:

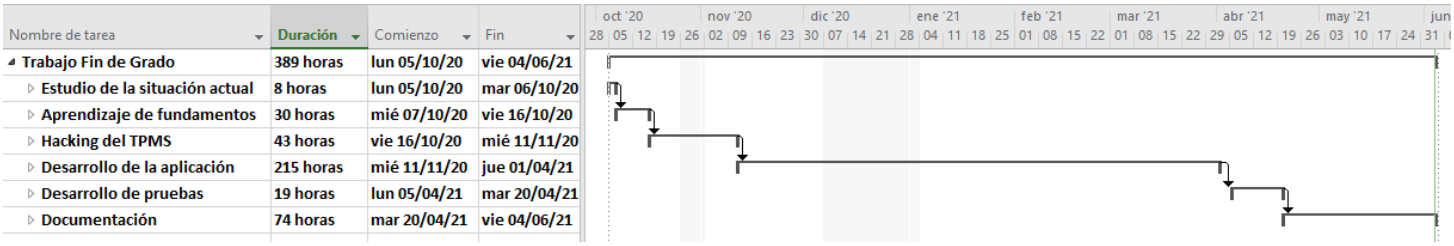
Jornadas laborales

- Jornada de 4 horas de lunes a viernes (16:00 a 20:00): desde el inicio hasta el 08/02/2021 y del 23/05/2021 hasta el final del proyecto.
- Jornada de 2 horas de lunes a jueves (18:00 a 20:00): desde 08/02/2021 a 23/05/2021.

Interrupciones

- Espera de 1 semana por la llegada de material para realizar las capturas de tráfico Bluetooth (Placa Micro:Bit): desde 23/10/2020 hasta 02/11/2020.
- Ausencia de 1 mes: desde 01/12/2020 hasta 04/01/2021.

En la siguiente figura se incluye el diagrama de Gantt sobre el desarrollo real del proyecto. Se muestran las tareas resumen con el objetivo de dar una visión sencilla del reparto del tiempo. Más adelante se muestran estas tareas desglosadas.



Nombre de tarea	Duración	Comienzo	Fin
Trabajo Fin de Grado	389 horas	lun 05/10/20	vie 04/06/21
Estudio de la situación actual	8 horas	lun 05/10/20	mar 06/10/20
Aprendizaje de fundamentos	30 horas	mié 07/10/20	vie 16/10/20
Hacking del TPMS	43 horas	vie 16/10/20	mié 11/11/20
Desarrollo de la aplicación	215 horas	mié 11/11/20	jue 01/04/21
Desarrollo de pruebas	19 horas	lun 05/04/21	mar 20/04/21
Documentación	74 horas	mar 20/04/21	vie 04/06/21

Figura 3.1: Diagrama de Gantt de la planificación

Aprendizaje de fundamentos	**30 horas**	**mié 07/10/20**	**vie 16/10/20**
Estudio de los TPMS	4 horas	mié 07/10/20	mié 07/10/20
Estudio de Buetooth Low Energy	16 horas	jue 08/10/20	mar 13/10/20
Estudio sobre desarrollo en Android	10 horas	mié 14/10/20	vie 16/10/20

Figura 3.2: Tareas realizadas en el estudio de la situación actual

Estudio de la situación actual	**8 horas**	**lun 05/10/20**	**mar 06/10/20**
Analisis de la aplicación del fabricante	2 horas	lun 05/10/20	lun 05/10/20
Análisis de aplicaciones alternativas	6 horas	lun 05/10/20	mar 06/10/20

Figura 3.3: Tareas realizadas en el aprendizaje de fundamentos

Hacking del TPMS	**43 horas**	**vie 16/10/20**	**mié 11/11/20**
Análisis del TPMS del fabricante	2 horas	vie 16/10/20	vie 16/10/20
Captura y análisis del tráfico	22 horas	lun 19/10/20	mié 04/11/20
Análisis del código de la aplicación del fabricante	10 horas	mié 04/11/20	vie 06/11/20
Retrospectiva y análisis conjunto	9 horas	lun 09/11/20	mié 11/11/20

Figura 3.4: Tareas realizadas en el hacking del TPMS

Desarrollo de la aplicación	**215 horas**	**mié 11/11/20**	**jue 01/04/21**
Análisis del sistema	**19 horas**	**mié 11/11/20**	**mar 17/11/20**
Identificación de requisitos del sistema	5 horas	mié 11/11/20	jue 12/11/20
Identificación de subsistemas	2 horas	jue 12/11/20	jue 12/11/20
Diseño de clases preliminar	2 horas	vie 13/11/20	vie 13/11/20
Análisis de casos de uso y escenarios	3 horas	vie 13/11/20	lun 16/11/20
Análisis de interfaces de usuario	5 horas	lun 16/11/20	mar 17/11/20
Especificación del plan de pruebas	2 horas	mar 17/11/20	mar 17/11/20
Diseño del sistema	**29 horas**	**mié 18/11/20**	**vie 27/11/20**
Diseño de la arquitectura	3 horas	mié 18/11/20	mié 18/11/20
Diseño de las clases	11 horas	mié 18/11/20	lun 23/11/20
Diseño de la base de datos	3 horas	lun 23/11/20	mar 24/11/20
Diseño de la interfaz	8 horas	mar 24/11/20	jue 26/11/20
Especificación técnica del plan de pruebas	4 horas	jue 26/11/20	vie 27/11/20
Implementación del sistema	**167 horas**	**vie 27/11/20**	**jue 01/04/21**
Implementación de estructura y navegación	10 horas	vie 27/11/20	mar 05/01/21
Implementación de la base de datos	15 horas	mar 05/01/21	lun 11/01/21
Implementación del servicio Bluetooth	20 horas	lun 11/01/21	lun 18/01/21
Implementación de módulos	**122 horas**	**lun 18/01/21**	**jue 01/04/21**
Módulo Home	60 horas	lun 18/01/21	lun 08/02/21
Módulo Pair	32 horas	mar 09/02/21	lun 08/03/21
Módulo Vehicles	30 horas	mar 09/03/21	jue 01/04/21

Figura 3.5: Tareas realizadas en el desarrollo de la aplicación

Desarrollo de pruebas	**19 horas**	**lun 05/04/21**	**mar 20/04/21**
Pruebas unitarias	4 horas	lun 05/04/21	mar 06/04/21
Pruebas de integración y del sistema	12 horas	mié 07/04/21	jue 15/04/21
Pruebas de usabilidad	3 horas	lun 19/04/21	mar 20/04/21

Figura 3.6: Tareas realizadas en el desarrollo de las pruebas

Documentación	74 horas	mar 20/04/21	vie 04/06/21
Documentación del estudio de la situación actual	4 horas	mar 20/04/21	jue 22/04/21
Documentación de los aspectos teóricos	8 horas	jue 22/04/21	jue 29/04/21
Documentación de la fase de hacking	16 horas	jue 29/04/21	jue 13/05/21
Documentación de la fase de análisis	6 horas	jue 13/05/21	mié 19/05/21
Documentación de la fase de diseño	8 horas	mié 19/05/21	mar 25/05/21
Documentación de la fase de implementación	4 horas	mar 25/05/21	mié 26/05/21
Documentación del desarrollo de las pruebas	4 horas	mié 26/05/21	jue 27/05/21
Manuales del sistema	4 horas	jue 27/05/21	vie 28/05/21
Presupuesto	6 horas	vie 28/05/21	lun 31/05/21
Conclusiones y ampliaciones	4 horas	lun 31/05/21	mar 01/06/21
Revisión de la documentación	10 horas	mar 01/06/21	vie 04/06/21

Figura 3.7: Tareas realizadas en la documentación

3.2 RESUMEN DEL PRESUPUESTO

Se ha realizado un presupuesto para este proyecto con el objetivo de estimar el coste real de desarrollo. Esta información se encuentra más detallada en el apartado Presupuesto, en el que se incluye: un presupuesto interno con los costes que le supondría a la empresa desarrollar el proyecto y un presupuesto para el cliente que añade tanto el IVA como el porcentaje de beneficio prorrateado.

A continuación, se muestra el presupuesto para el cliente.

PARTIDA	TOTAL
PARTIDA 1: Estudio de la situación actual	195,89 €
PARTIDA 2: Aprendizaje de fundamentos	734,60 €
PARTIDA 3: Hacking del TPMS	1.052,93 €
PARTIDA 4: Desarrollo de la aplicación	5.264,65 €
PARTIDA 5: Desarrollo de pruebas	465,25 €
PARTIDA 6: Documentación	1.812,02 €
TOTAL CLIENTE	**9.525,35 €**
IVA (21%)	2.381,34 €
TOTAL CLIENTE + IVA	**11.906,69 €**

Tabla 2:Resumen del presupuesto del cliente

Capítulo 4. *Hacking* del TPMS

En este capítulo se desarrollará todo el proceso de captura y descifrado de los datos enviados por los sensores a través de BLE. Los subapartados siguen un orden cronológico a excepción del último, en el cual se concluye con el objetivo de concretar el resultado, resumir el aprendizaje obtenido y analizar los fallos. Previo al subapartado 1, se realiza un estudio sobre la tecnología Bluetooth Low-Energy el cual está detallado en el apartado 2.2. A continuación, se describe de manera muy breve todo el procedimiento:

1. Análisis del TPMS: se descarga la aplicación y se prueban los sensores. Se determina como se realiza la conexión y que parámetros se pueden monitorizar.
2. Captura del tráfico: se realiza la captura del tráfico mediante dos procedimientos diferentes sin obtener resultados.
3. Análisis del código de la aplicación del fabricante: se estudia el código para ver cómo se establece conexión con los sensores. Tampoco se obtienen resultados ya que no parece que se realice la conexión.
4. Solución: tras días de búsqueda se encuentra información relevante (y todo lo anterior cobra sentido). Se identifican los paquetes de datos y se descifra el contenido.

4.1 HERRAMIENTAS UTILIZADAS

4.1.1 Hardware

4.1.1.1 Micro:Bit

Micro:Bit [7] es una pequeña placa programable diseñada por la BBC con fines educativos. Tanto el hardware como el software es de código abierto. Tiene un tamaño de 4 x 5 cm e incorpora varios sensores. Se puede usar tanto desde un ordenador como desde un móvil. Ofrece dos modalidades de programación:

- Programación visual o por bloques: mediante un entorno de programación gráfico propio: MakeCode de Microsoft.
- Programación por código: permite programar con JavaScript, Python o Scratch, entre otros.

La placa Micro:Bit dispone de:

- 25 LEDs programables individualmente
- 2 botones programables
- Pines de entrada y salida
- Sensor de luz y temperatura
- Sensores de movimiento (acelerómetro y brújula)
- Comunicación inalámbrica, vía Bluetooth y radio
- USB y conector para batería externa

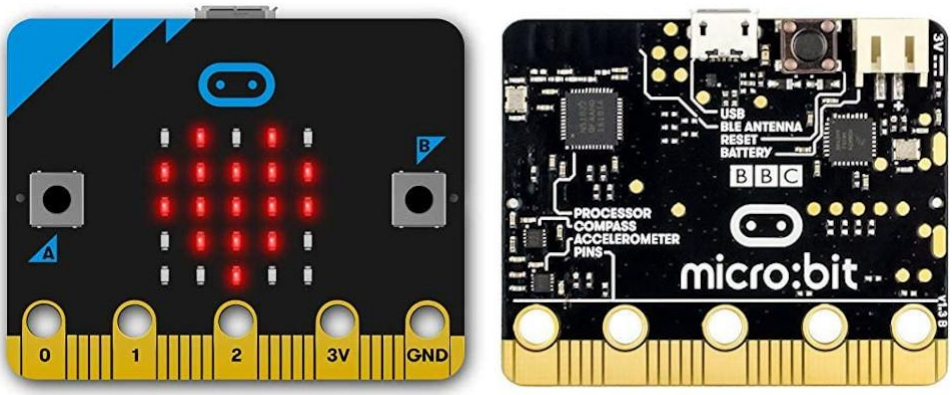


Figura 4.1: Placa Micro:Bit

4.1.2 Software

4.1.2.1 Wireshark

Wireshark [8] es un software libre y gratuito para análisis de protocolos y paquetes de datos creado por Gerald Combs. Esta herramienta permite interceptar el tráfico de red en tiempo real y mostrarlo de una manera más legible y organizada. Ofrece diversos tipos de filtrado que ayudan al usuario a encontrar el tráfico que quieren inspeccionar. También permite analizar tráfico previamente capturado y guardado en un fichero.

Es un software destinado tanto a administradores de sistemas y profesionales de seguridad, como al ámbito educativo.

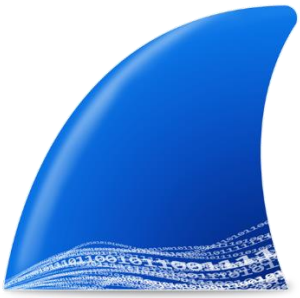


Figura 4.2: Logo de Wireshark

Wireshark tiene una licencia GPLv2, se puede ejecutar en más de veinte plataformas, es compatible con más de 480 protocolos de red y está basado en la librería pcap. Algunas de las funcionalidades más destacadas son: capturar tramas directamente desde la red, mostrarlas y filtrarlas, editar tramas y transmitirlas por la red, capturar tramas desde un ordenador remoto, exportar capturas de datos a diferentes formatos y realizar seguimientos de flujos o patrones de tráfico.

4.1.2.2 HCI snoop log

"*HCI snoop log"* es una herramienta del sistema operativo Android para depurar el tráfico Bluetooth mediante registros. Estos registros capturan los paquetes de la interfaz del controlador del host (HCI) y se almacenan en un fichero en la memoria interna. Este se encuentra generalmente en la ruta "data/misc/bluettooth/logs", aunque puede variar según la

versión del sistema operativo. El archivo generado se denomina *"btsnoop_hci.log"* y puede ser examinado con programas de análisis de protocolos como Wireshark.

Esta herramienta está disponible a partir de Android 4.4 y se encuentra en el menú de opciones de desarrollador.

4.1.2.3 BtleJack

BtleJack [9] es una herramienta de seguridad para realizar auditorías a dispositivos Bluetooth Low-Energy. Utilizando las placas Micro:Bit con firmware dedicado permite capturar tráfico de conexiones existentes y conexiones nuevas, hacer hijacking[1] y jamming[2] de conexiones existentes y exportar capturas de tráfico en formato PCAP y otros formatos.

Este proyecto se encuentra en un repositorio en GitHub bajo licencia MIT y se ha desarrollado en Python. Como explican sus creadores, para hacer un uso óptimo de la herramienta conviene utilizar tres Micro:Bit para que cada una escuche uno de los canales de anuncio de BLE y, de esta manera, no se escapen paquetes de datos durante la captura. Actualmente se está ampliando el soporte a otros tipos de hardware como pueden ser: Raspberry Pi, Adafruit's Bluefruit LE sniffer o nRF51822 Eval Kit.

[1] Ataque informático que consiste en secuestrar un elemento específico del entorno de Internet a través de rutas no autorizadas.

[2] Ataque informático que consiste en saturar los recursos de las máquinas destino para que sean incapaces de proporcionar los servicios normales.

4.2 ANÁLISIS DEL TPMS

El producto puesto a prueba es un Sistema de monitorización de la presión (*Tire-Pressure Monitoring System* o TPMS) facilitado por el tutor del proyecto. Como se describe en el apartado 2.1, se trata de un TPMS directo por Bluetooth con sensores externos.

Parece ser de procedencia china (caracteres chinos por todo el producto) y no se indica por ningún lado el fabricante o la marca. A continuación, se muestran unas imágenes de los sensores y el contenido de la caja.

Figura 4.3: Contenido del TPMS

Para llevar a cabo las pruebas (y el posterior desarrollo de la aplicación) se utilizará una rueda de bicicleta. Como se puede ver en las siguientes imágenes, el sensor se coloca en la válvula de inflado del neumático.

Figura 4.4: Rueda de prueba con el sensor

Para monitorizar los datos, el fabricante proporciona una aplicación móvil gratuita (ver apartado 1.3.1). Una vez descargada, te requieren obligatoriamente el permiso de localización y activar el Bluetooth (si no se concede, la aplicación se cierra). Siguiendo el manual se instala el sensor mediante uno de los cuatro métodos de emparejamiento y comienza la recepción de los datos. Se realizan tres comprobaciones:

1. Se comprueban los dispositivos Bluetooth enlazados con el móvil: el sensor no aparece.
2. Se extrae el sensor de la válvula: La aplicación lanza un mensaje de error en el neumático enlazado. Tanto por voz como en la interfaz.
3. Se traslada la aplicación a segundo plano: se sigue emitiendo el mensaje de error por voz.

En conclusión, podemos determinar varios comportamientos:

- La conexión se realiza sin emparejamiento con el sistema operativo, es decir, Android no detecta el sensor como un dispositivo Bluetooth.
- El móvil realiza un escaneo continuo ya que el error es lanzado pocos segundos después de retirar el sensor.
- La aplicación permite la monitorización en segundo plano, es decir, el escaneo no finaliza hasta que no se cierra la aplicación.

4.3 CAPTURA Y ANÁLISIS DEL TRÁFICO

Para determinar cómo se comunican los sensores con la aplicación, se procede a capturar el tráfico Bluetooth. Para ello se plantean dos puntos de partida diferentes:

1. Tengo una aplicación móvil en Android y un dispositivo BLE.
   Para este escenario, Android proporciona una herramienta que permite capturar todo el tráfico Bluetooth y depositarlo en un fichero en la memoria interna. Se puede encontrar en las opciones del desarrollador en ajustes.
2. No tengo aplicación o dispositivo BLE.
   En este caso se captura el tráfico a través del medio de compartición: el aire. Para ello hay que hacer uso de hardware que permita la escucha de canales de radiofrecuencia y software específico que procese los paquetes capturados.

Una vez obtenido el tráfico de datos se analizan los paquetes mediante un analizador de protocolos. Para este proyecto se ha utilizado Wireshark.

Aunque el escenario en este caso es claramente el punto 1, se terminan explorando las dos vías al no obtener ningún resultado válido con la primera opción. A continuación, se detallan los dos procedimientos.

4.3.1 HCI snoop log

4.3.1.1 Captura

Para capturar el tráfico mediante “HCI snoop log” se realiza el siguiente procedimiento:

1. Conectar y emparejar el sensor.
2. Activar las opciones del desarrollador en Android. Ir a Ajustes > Información del dispositivo y pulsar siete veces sobre “Número de compilación”
3. Activar el Bluetooth.
4. Activar la opción Ajustes > Opciones de desarrollador > Registro de búsqueda de Bluetooth.

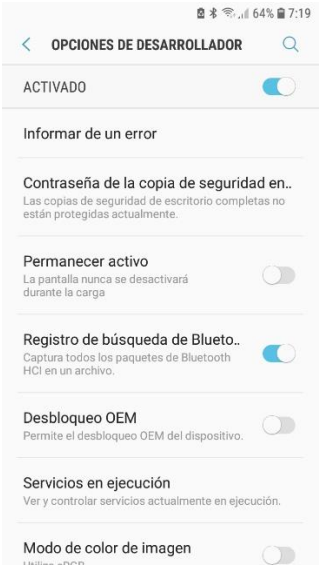


Figura 4.5: Menú de desarrollador en Android

5. Reiniciar el Bluetooth (Desactivar y volver a activar).
6. Abrir la aplicación del TPMS y esperar hasta que se reciban datos del sensor.
7. Volver a Ajustes > Opciones de desarrollador y clicar en "Informa de un error". Seleccionar "Informe completo".

Figura 4.6: Informe de un error en Android

8. Esperar hasta que aparezca la notificación "Informe de errores capturado" y abrirla.

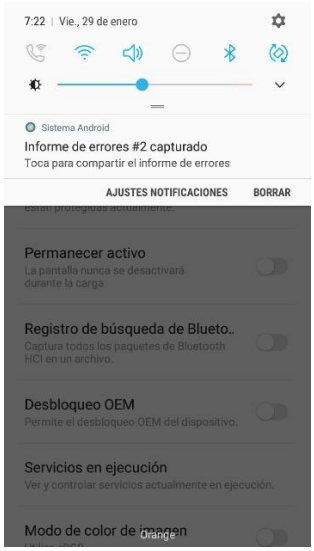


Figura 4.7: Notificación de informe capturado en Android

9. Compartir el informe.

Tras seguir estos pasos se obtiene una carpeta de nombre “bugreport <fecha y hora en la que se realizó el informe>”. Dentro de la carpeta, en la ruta “FS > data > log > bt” se encuentra el fichero “btsnoop_hci.log”.

4.3.1.2 Análisis

Utilizando Wireshark se analizan los paquetes capturados. Para ello primero hay que cambiar la extensión del fichero a “. pcap”.

Los paquetes que se buscan son los del protocolo ATT ya que son los que contienen los datos. Tras filtrar los paquetes en busca del protocolo ATT vemos que no se ha obtenido ninguno.

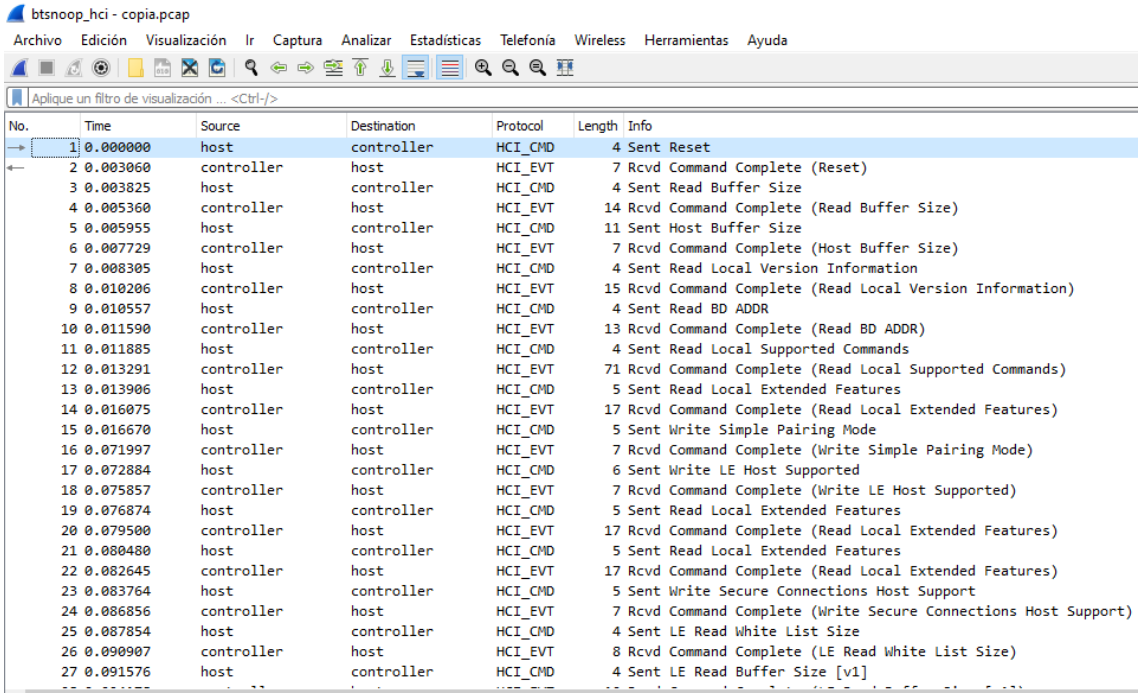


Figura 4.8: Tráfico del BLE en Wireshark

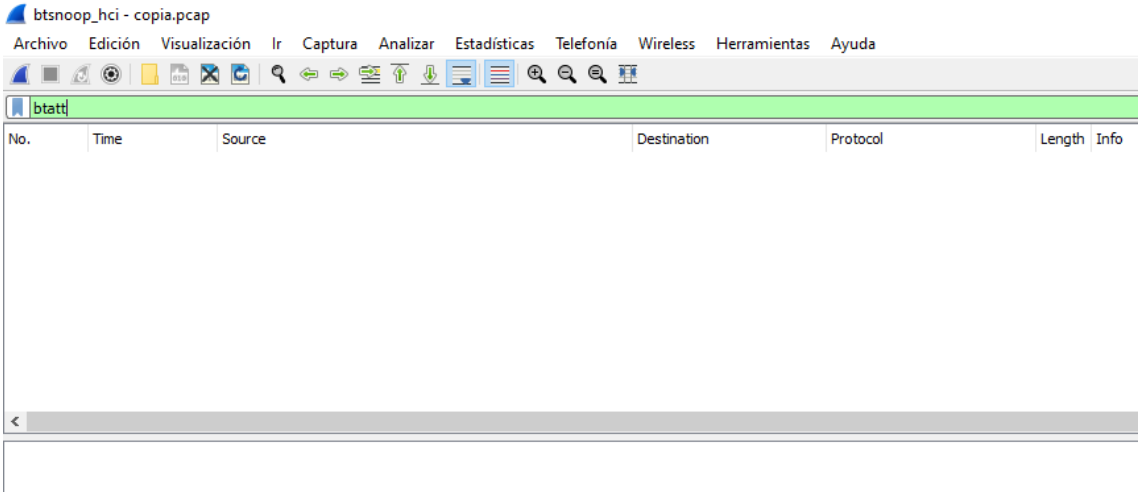


Figura 4.9: Filtrado sin resultados en Wireshark

Se repite el procedimiento varias veces para descartar que haya sido un fallo durante la captura, pero los resultados siguen siendo los mismos.

Se realiza una siguiente comprobación para descartar que la herramienta "HCI snoop log" sea el problema, es decir, que en realidad no capture paquetes ATT o que solo capture paquetes HCI. Para ello se repite el mismo procedimiento con un dispositivo diferente. En esta ocasión se utiliza un reloj inteligente que se conecta con una aplicación móvil para

volcar los últimos datos de salud recogidos. Como se puede ver en la siguiente imagen, esta vez sí se obtienen paquetes ATT, por lo que la herramienta de Android funciona correctamente.

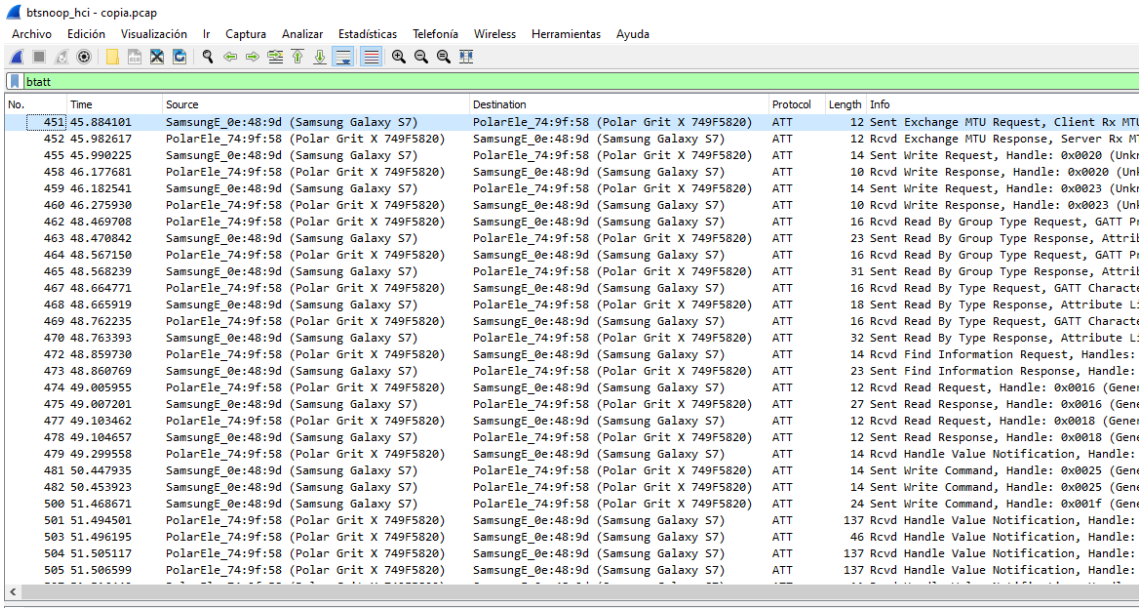


Figura 4.10: Tráfico reloj inteligente en Wiresahrk

Llegados a este punto, en el que no hay forma de obtener paquetes ATT, se empiezan a analizar uno a uno los paquetes en busca de algún tipo de información relevante o referente al sensor. Con esto se pretende descartar que en la captura no se estén obteniendo paquetes procedentes del sensor, ya que al menos se tendrían que estar enviando paquetes de anuncio.

En el paquete 836 se encuentran las primeras referencias al TPMS. Como se puede observar en la siguiente captura el nombre del dispositivo es “TPMS1_10539c”, el cual se corresponde con el identificador del sensor actualmente conectado. Dos líneas más arriba obtenemos la dirección MAC del sensor (80:ea:ca:10:53:9c).

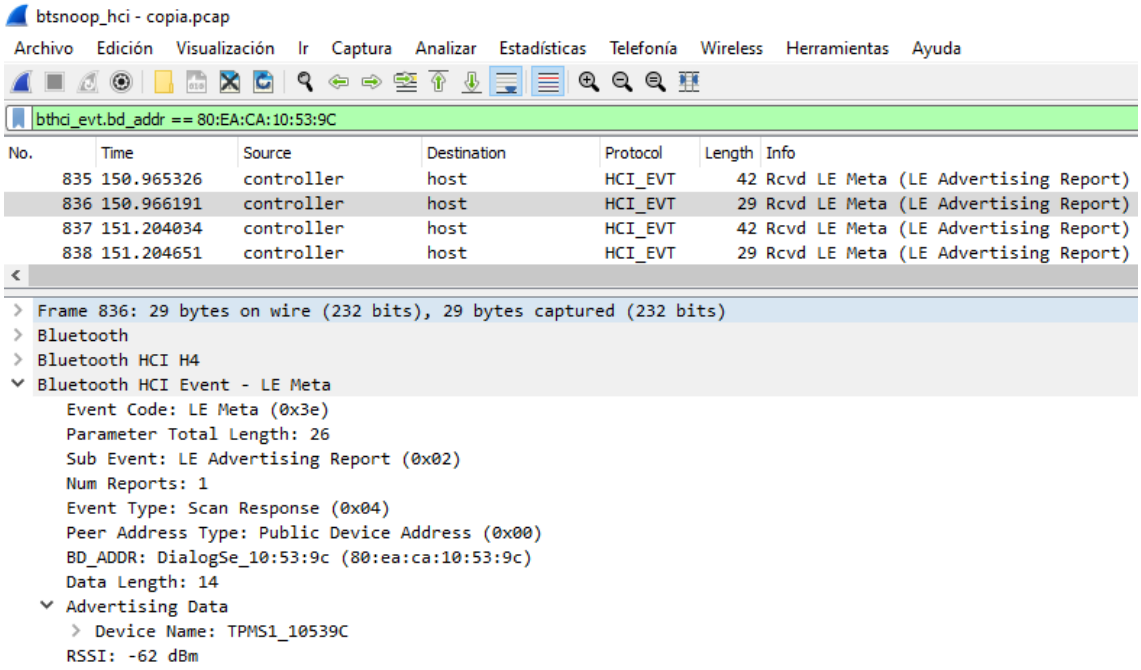


Figura 4.11: Paquete 836 del tráfico del BLE en Wireshark

Tras filtrar por la dirección MAC, podemos observar todos los paquetes relacionados con el sensor, de los cuales se pueden deducir las siguientes características:

- Todos los paquetes provienen del sensor, por lo que la aplicación no está enviando nada al dispositivo.
- Todos son paquetes de anuncio.
- Los paquetes vienen por pares, es decir, se envía uno de tipo "*Scannable Undirected Advertising*" y después uno de tipo "*Scan Response*".
- Los de tipo "*Scannable Undirected Advertising*" contienen información acerca del fabricante y un conjunto "Data" que siempre empieza por la dirección MAC.
- Los de tipo "*Scan Response*" contiene el nombre del dispositivo.

Aun teniendo todo esto en cuenta y aplicando la teoría estudiada no se llega a entender dónde está la información ni como está representada. Asumiendo que los datos llegan a través del protocolo ATT la búsqueda sigue enfocada en encontrar dichos paquetes.

En este punto, se barajan dos hipótesis; o bien el sensor no envía paquetes ATT o la herramienta no consigue capturarlos porque tiene algún tipo de cifrado.

La primera hipótesis queda descartada aplicando lo que se sabe hasta el momento: los datos tienen que estar encapsulados en el protocolo ATT.

Para poder descartar del todo la segunda hipótesis se procede a capturar siguiendo la segunda metodología: capturar todos los datos que viajan a través del aire con el objetivo de encontrar otro tipo de paquetes (paquetes aparentemente cifrados).

4.3.2 Placa Micro:Bit y BtleJack

Para lleva a cabo la captura de tráfico por el aire esta técnica plantea el siguiente escenario:

- Tres placas Micro:Bit, una para cada canal de anuncio (canales de *Advertisement*: 37, 38 y 39) con el fin de capturar los paquetes de conexión que viajen por cada canal.
- Flashear las placas con BtleJack para capturar tanto conexiones nuevas como conexiones existentes.

En este caso solo se va a utilizar una placa Micro:Bit (debido a que adquirir tres es una inversión demasiado grande) por lo que la probabilidad de no capturar el paquete de conexión es de un 66%. Suponiendo que se realizan varios intentos la captura debería producirse en algún momento. La preparación del entorno consiste en lo siguiente:

1. Conectar y emparejar el sensor.
2. Disponer de un sistema Linux para trabajar con BtleJack. En este caso se va a utilizar una máquina virtual con Ubuntu 20.04.
3. Descargar BtleJack de su repositorio en GitHub e instalar ejecutando el comando "python 3.x setup.py install".
4. Conectar la placa mediante un puerto USB y flashearla ejecutando el comando "btlejack -i".

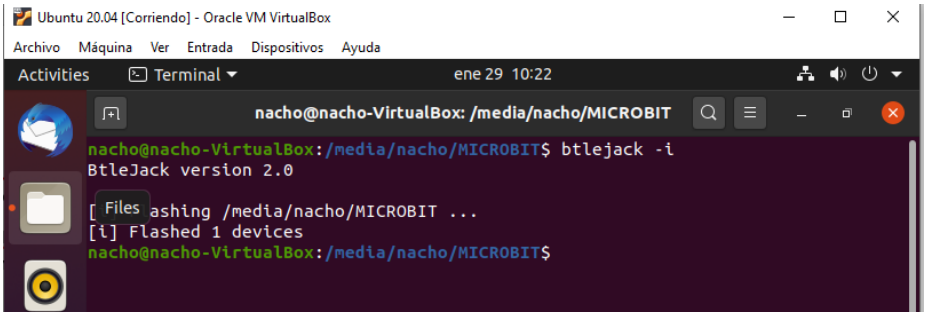


Figura 4.12: Flashing de BtleJack en la Micro:Bit en Ubuntu

BtleJack ofrece dos modos de escucha: conexiones nuevas o conexiones existentes. Se realizan las dos escuchas varias veces y no se detecta nada.

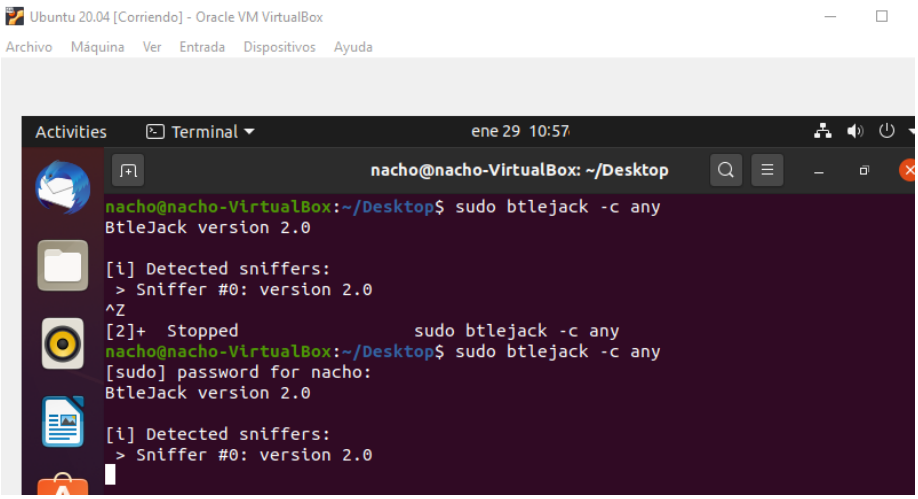


Figura 4.13: Escaneado de nuevas conexiones BLE con BtleJack

Dados estos resultados se investiga la herramienta BtleJack más a fondo hasta dar con el problema. Solo detecta dispositivos que se conectan, es decir, la búsqueda en ambos modos se limita a encontrar un paquete “CONNECT_REQ”. Por lo tanto, se puede concluir que los sensores realizan el intercambio de datos sin realizar una conexión con el móvil.

La hipótesis anteriormente descartada, ahora se confirma: el sensor no envía paquetes ATT.

4.4 ANÁLISIS DEL CÓDIGO DE LA APLICACIÓN DEL FABRICANTE

Como último recurso se procede a analizar el código de la aplicación del fabricante. De esta manera se pretende encontrar la solución desde el lado opuesto, es decir, en vez de analizar cómo envía los datos el sensor, intentar comprender cómo los recibe la aplicación.

Para llevar esto a cabo se necesita desempaquetar el fichero ".apk" y obtener una representación en código Java. Se realiza el siguiente procedimiento:

1. Descargar la APK
2. Desempaquetar la aplicación y obtener un fichero JAR que contenga los ficheros ".class" utilizando la herramienta dex2jar. Ejecutar el comando "d2j-dex2jar <fichero apk>".
3. Utilizar la herramienta grafica jd-gui para ver los archivos fuente de Java a partir de los archivos ".class".

El código obtenido se ve de la siguiente manera:

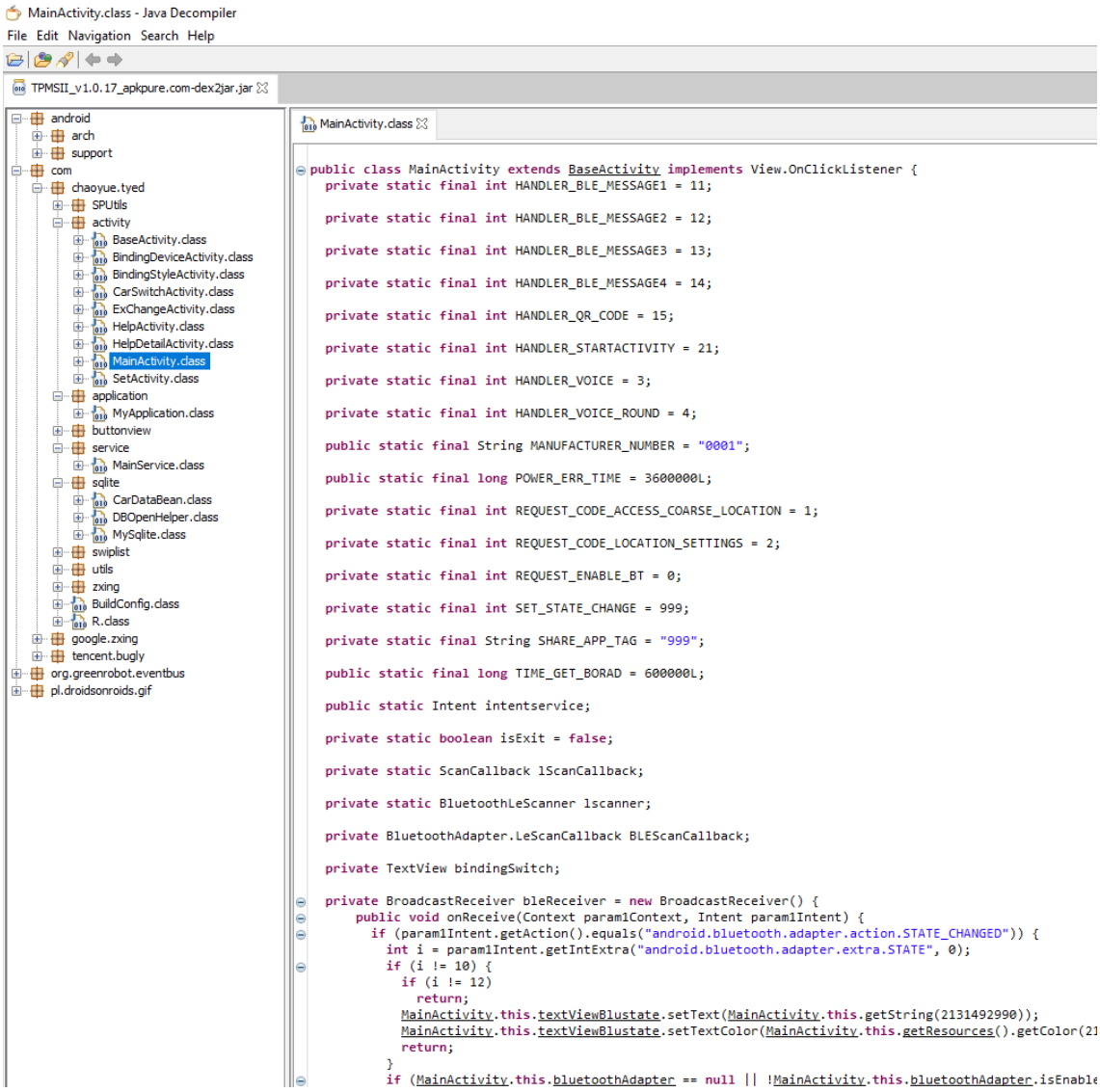


Figura 4.14: Código Java de la aplicación del fabricante

Pese a tener algunas pérdidas, el código es relativamente comprensible y se detectan rápidamente las clases principales de la aplicación. Sin embargo, la arquitectura y la forma de implementar dejan mucho que desear. A continuación, se muestran algunas de las partes más relevantes.

```
private void autoSearch() {
    BluetoothAdapter bluetoothAdapter = this.bluetoothAdapter;
    if (bluetoothAdapter == null || !bluetoothAdapter.isEnabled()) {
      startActivityForResult(new
Intent("android.bluetooth.adapter.action.REQUEST_ENABLE"), 0);
      return;
    }
    this.mStartTime = SystemClock.elapsedRealtime();
    this.handler.sendEmptyMessage(2);
    if (Build.VERSION.SDK_INT > 23)
      this.bluetoothAdapter.startLeScan(this.mLeScanCallback2);
    showSearchDialog();
}
     private void manualSearch(final int i) {
    SelfDialog selfDialog = new SelfDialog((Context)this);
    this.selfDialog = selfDialog;
    selfDialog.setTitle(getString(2131492986));
    this.selfDialog.setMessage("");
     ..............
     }

private void saomiaoSearch() {
    if (ContextCompat.checkSelfPermission((Context)this,
"android.permission.CAMERA") != 0) {
      ActivityCompat.requestPermissions((Activity)this, new String[] {
"android.permission.CAMERA" }, 11003);
      return;
    }
    startActivityForResult(new Intent((Context)this,
CaptureActivity.class), 11002);
}
```

En esta captura se pueden ver los métodos relativos a los modos de emparejamiento de los sensores con la aplicación. El primer método parece ser el que realiza el auto emparejado. Para ello inicia un escaneo de bluetooth. El segundo realiza el emparejado manual mediante un Dialogo de alerta. El tercero inicia la cámara, por lo que será el que realice el emparejamiento mediante escaneado de código QR.

```
private void checeBlueState() {
    BluetoothManager bluetoothManager =
(BluetoothManager)getSystemService("bluetooth");
    this.bluetoothManager = bluetoothManager;
    BluetoothAdapter bluetoothAdapter = bluetoothManager.getAdapter();
    this.bluetoothAdapter = bluetoothAdapter;
    if (bluetoothAdapter == null)
      showCustomToast(getString(2131492904), 4000);
    if (Build.VERSION.SDK_INT < 15)
      showCustomToast(getString(2131492905), 4000);
    bluetoothAdapter = this.bluetoothAdapter;
    if (bluetoothAdapter != null || bluetoothAdapter.isDiscovering())
      this.bluetoothAdapter.stopLeScan(this.BLEScanCallback);
    if (Build.VERSION.SDK_INT >= 21)
      lscanner = this.bluetoothAdapter.getBluetoothLeScanner();
    if (Build.VERSION.SDK_INT < 21) {
      this.lservice = new MainService();
      intentservice = new Intent((Context)this, MainService.class);
      MainService.startRun(this.BLEScanCallback, this.bluetoothManager,
this.bluetoothAdapter, this.handler);
      startService(intentservice);
    } else if (Build.VERSION.SDK_INT >= 21 && Build.VERSION.SDK_INT < 23) {
      this.lservice = new MainService();
      intentservice = new Intent((Context)this, MainService.class);
      MainService.startRun5(lscanner, lScanCallback, this.bluetoothManager,
this.bluetoothAdapter, this.handler);
      startService(intentservice);
    } else if (Build.VERSION.SDK_INT >= 23) {
      if (ContextCompat.checkSelfPermission((Context)this,
"android.permission.ACCESS_COARSE_LOCATION") != 0) {
        if
(ActivityCompat.shouldShowRequestPermissionRationale((Activity)this,
"android.permission.ACCESS_COARSE_LOCATION"))
          Toast.makeText((Context)this, getString(2131492970), 0).show();
        ActivityCompat.requestPermissions((Activity)this, new String[] {
"android.permission.ACCESS_COARSE_LOCATION" }, 1);
      }
      if (!Boolean.valueOf(isLocationEnable((Context)this)).booleanValue())
        setLocationService();
      this.lservice = new MainService();
      intentservice = new Intent((Context)this, MainService.class);
      this.lservice.startRunMore(this.BLEScanCallback,
this.bluetoothManager, this.bluetoothAdapter, this.handler);
      startService(intentservice);
    }
    IntentFilter intentFilter = new IntentFilter();

intentFilter.addAction("android.bluetooth.adapter.action.STATE_CHANGED");
    registerReceiver(this.bleReceiver, intentFilter);
     }
```

En esta captura se puede ver el método que controla el estado del Bluetooth. Pide permisos de localización y muestra un dialogo para activar el Bluetooth del dispositivo en caso de que se encuentre desactivado.

```
private void sendbleMessage(String paramString1, boolean paramBoolean,
String paramString2) {
    this.mMessage = handler.obtainMessage();
    if ("0001".equals(paramString1.substring(18, 22))) {
      StringBuilder stringBuilder1;
      if (this.mySqlite.fintDataCount() == 1) {
        new CarDataBean();
        CarDataBean carDataBean = this.mySqlite.findbyId();
        if (carDataBean.getTyreone().equals("0") &&
carDataBean.getTyretwo().equals("0") &&
carDataBean.getTyrethree().equals("0") &&
carDataBean.getTyrefour().equals("0")) {
          this.keytypeone = "";
          this.keytypetwo = "";
          this.keytypethree = "";
          this.keytypefour = "";
        } else {
          this.keytypeone = carDataBean.getTyreone();
          this.keytypetwo = carDataBean.getTyretwo();
          this.keytypethree = carDataBean.getTyrethree();
          this.keytypefour = carDataBean.getTyrefour();
        }
      }
        ……................................
     }
```

Este método es llamado directamente desde el ScanCallback, es decir, es el que recibe e interpreta los datos obtenidos en el escaneo. Parece ser que filtra la cadena de texto recibida. Si comienza por “0001” sigue procesando, en caso contrario, descarta el mensaje.

Esto podría indicar que la cadena enviada por los sensores debe contener esos números. Se realiza una comprobación en la cadena “Data”, obtenida en las capturas de tráfico del apartado 4.2.2, pero los caracteres “0001” no aparecen.

```
public static double getTemp(String paramString) {
  new BigInteger("0");
  String str1 = paramString.substring(42, 44);
  String str2 = paramString.substring(44, 46);
  String str3 = paramString.substring(46, 48);
  paramString = paramString.substring(48, 50);
  try {
    StringBuilder stringBuilder = new StringBuilder();
    stringBuilder.append(paramString);
    stringBuilder.append(str3);
    stringBuilder.append(str2);
    stringBuilder.append(str1);
    int i = Integer.parseInt(stringBuilder.toString(), 16) / 100;
    return i;
  } catch (NumberFormatException numberFormatException) {
    try {
      StringBuilder stringBuilder = new StringBuilder();
      stringBuilder.append(paramString);
      stringBuilder.append(str3);
      stringBuilder.append(str2);
      stringBuilder.append(str1);
      int i = (new BigInteger(stringBuilder.toString(), 16)).intValue();
      double d = i;
      Double.isNaN(d);
      return d / 100.0D;
    } catch (NumberFormatException numberFormatException1) {
      numberFormatException1.printStackTrace();
      return 0.0D;
    }
  }
}
```

```
public String getPress(String paramString) {
  String str1 = paramString.substring(34, 36);
  String str2 = paramString.substring(36, 38);
  String str3 = paramString.substring(38, 40);
  paramString = paramString.substring(40, 42);
  StringBuilder stringBuilder2 = new StringBuilder();
  stringBuilder2.append(paramString);
  stringBuilder2.append(str3);
  stringBuilder2.append(str2);
  stringBuilder2.append(str1);
  int i = Integer.parseInt(stringBuilder2.toString(), 16);
  StringBuilder stringBuilder1 = new StringBuilder();
  double d = i;
  Double.isNaN(d);
  stringBuilder1.append(d / 100000.0D);
  stringBuilder1.append("");
  return stringBuilder1.toString();
}
```

Estos métodos al parecer analizan una cadena de texto para obtener tanto la temperatura como la presión y lo primero que hacen es darles la vuelta a los valores recibidos por pares.

En conclusión, el análisis del código tampoco da lugar a una compresión de la comunicación entre sensores y aplicación. Si el código estuviera más ordenado y con nombres de variables y parámetros bien definidos, quizás se llegaría a sacar algo en claro.

4.5 SOLUCIÓN

Se han seguido tres procedimientos diferentes para intentar comprender el formato de transmisión de datos del TPMS, pero ninguno ha dado resultado. El proyecto queda en punto muerto y durante unos días se busca todo tipo de información, tanto sobre teoría de BLE como otras prácticas de depuración y captura.

Finalmente, se encuentra la solución: un repositorio en GitHub [10] en el que se explica cómo funciona un modelo concreto de BLE TPMS.

Una vez entendido el mecanismo, todo el trabajo realizado hasta el momento cobra sentido.

A continuación, se detalla cómo se lleva a cabo la comunicación y se realiza una nueva valoración de los resultados obtenidos en los apartados anteriores.

Como ya se había concluido anteriormente, los sensores no se conectan ni emparejan con el móvil y tampoco reciben datos de este. Todos los datos se transmiten como parte de los "Datos específicos del fabricante" incluidos en los paquetes de anuncio (ADV).

4.5.1 Datos específicos del fabricante

Son un conjunto de datos que permiten al fabricante añadir información en los paquetes de anuncio. Como se describe en el apartado 2.2.4.1 los paquetes de anuncio pueden ser de cuatro tipos, pero los Datos específicos del fabricante solo están presentes en los no directivos (ADV_IND, ADV_NONCONN_IND y ADV_SCAN_IND).

Entre estas tres opciones los sensores del TMPS se sitúan como ADV_SCAN_IND ya que, aparte de no directivos, son no conectables y escaneables. El formato del PDU de anuncio No directivo es el siguiente:

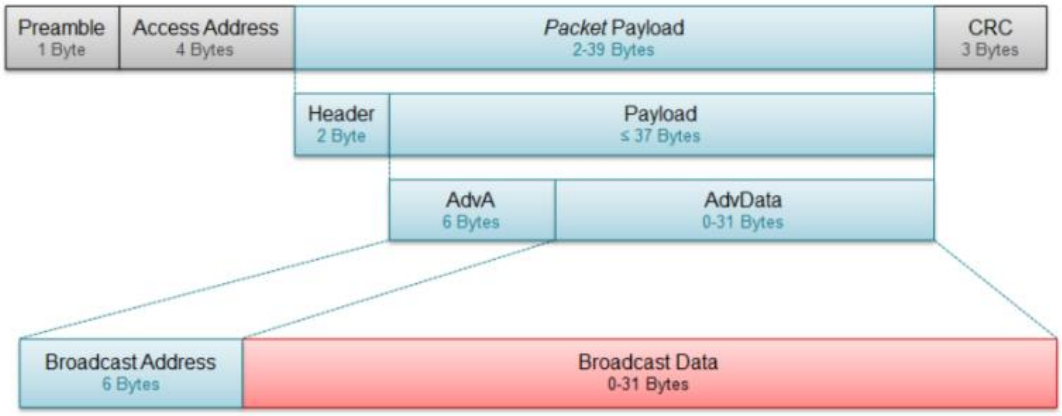


Figura 4.15: Paquete de anuncio no directivo

Como se verá más adelante al analizar la trama capturada, la cabecera (*Header*) contiene los bits que indican el tipo de PDU (ADV_SCAN_IND en este caso) y el tipo de dirección del dispositivo, que puede ser pública o aleatoria (pública en este caso). La dirección de dispositivo (AdvA o *Broadcast Address*) es la dirección MAC.

El contenido del bloque de datos (*AdvData o Broadcast Data*) puede ser elegido por el fabricante, siguiendo unos formatos específicos establecidos por la organización Bluetooth SIG. En la siguiente tabla se muestran los datos que se pueden incluir en este bloque.

Tipo de dato	**Valor**	**Descripción**
Flags	0x01	Modo de detectabilidad del dispositivo
Service UUID	0x02 – 0x07	Servicios GATT del dispositivo
Local Name	0x08 – 0x09	Nombre del dispositivo
TX Power Level	0x0A	Potencia de salida del dispositivo
Manufacturer Specific Data	0xFF	Datos especificados por el fabricante

Tabla 3: Contenido del bloque de datos del PDU de anuncio

En este caso, el TPMS incluye: Flags, Service UUID y Datos específicos del fabricante (*Manufacturer Specific Data*). La estructura de los Datos específicos del fabricante es la siguiente:

Longitud	Tipo de dato	ID de la compañía	Datos
1 byte	1 byte	2 bytes	≤ 25 bytes

Figura 4.16: Estructura de los Datos específicos del fabricante

El formato de los datos lo puede elegir el fabricante, aunque en la mayoría de los casos suelen seguir un estándar (ej.: iBeacons de Apple). En este caso no se ha podido determinar la procedencia del formato.

4.5.2 Formato de los datos

Los sensores del TPMS transmiten una cadena de datos de 17 bytes con la siguiente estructura:

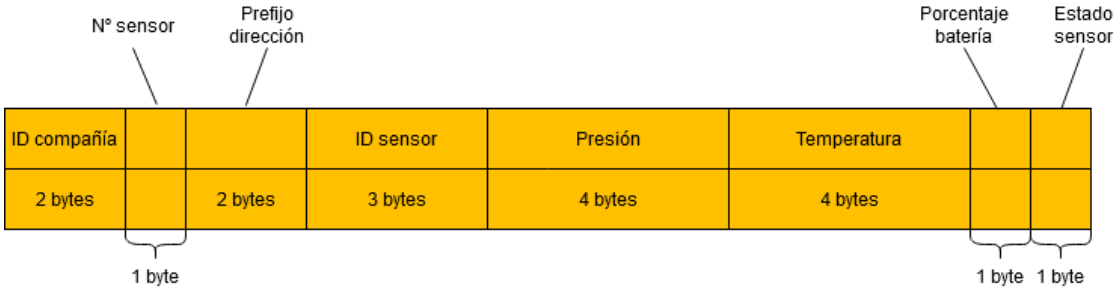


Figura 4.17: Estructura de los datos transmitidos por los sensores

ID de la compañía

El valor está codificado en formato little-endian[3], por lo que hay que pasarlo a decimal. Este identificador tiene que estar registrado en la organización Bluetooth SIG.

Número de sensor

El valor 80 se corresponde con el sensor 1, el 81 con el 2, el 82 con el 3…

Prefijo de la dirección

Es un valor fijo: *"EACA"*.

ID del sensor

Es el identificador único de cada sensor. Consta de 6 dígitos. El fabricante proporciona una tarjeta con los identificadores.

Presión

El valor está codificado en formato little-endian, por lo que hay que pasarlo a decimal. El valor obtenido divido por 1000 da la presión en kilopascal (kPa) y dividido por 100000 en bar.

Temperatura

El valor está codificado en formato little-endian, por lo que hay que pasarlo a decimal. El valor obtenido divido por 100 da la temperatura en Celsius.

[3] Formato adoptado por Intel para almacenar datos de más de 1 byte. (Ej.: "*0x4A3B2C1D"* en hexadecimal se corresponde con *{1D, 2C, 3B, 4A}* en little-endian*)*

Porcentaje de la batería

El valor está codificado en formato hexadecimal, por lo que hay que pasarlo a decimal. Los valores obtenidos pueden ser 25,50, 75 y 100.

Estado del sensor

Tiene dos valores posibles: *00* cuando el sensor detecta presión y *01* cuando no la detecta.

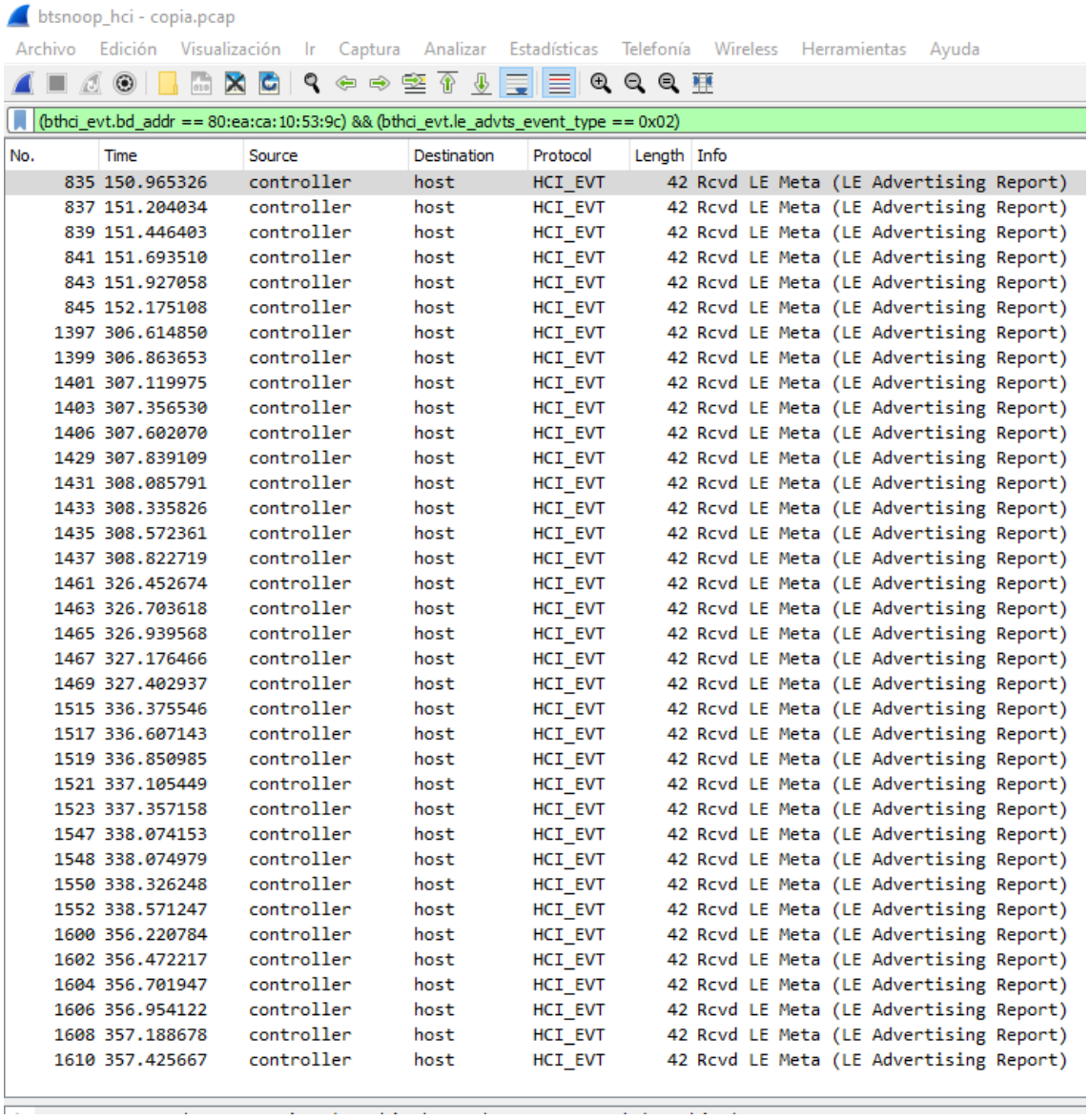
Por último, para obtener la **dirección MAC del dispositivo** es necesario componerla de la siguiente manera:
<nº sensor><prefijo de la dirección><identificador del sensor> separando los bytes con ":". (Ej.: 80:EA:CA:20:45:6C)

4.5.3 Análisis de la trama capturada

En este apartado se analiza una de las tramas capturadas anteriormente (apartado 4.2.1), a fin de concretar como se interpreta una transmisión de datos en un TPMS Bluetooth. Teniendo en cuenta todo lo aprendido hasta ahora, el proceso para obtener los paquetes de datos enviados por los sensores resulta bastante sencillo:

1. Capturar el tráfico utilizando la herramienta de Android: HCI snoop log (apartado 4.2.1.1).
2. Calcular la dirección MAC del dispositivo o dispositivos que se quieren "escuchar". El número de dispositivo se encuentra en la parte posterior del sensor, el prefijo tiene un valor conocido *"EACA"* y el identificador del sensor se encuentra en la tarjeta facilitada por el fabricante. En este caso se analiza el sensor 1 cuyo identificador es *"10539c"*. Por lo tanto, la MAC resultante es *"80:EA:CA:10:53:9C"*.
3. Utilizando Wireshark filtrar los paquetes por la dirección MAC y por el tipo de evento, que en este caso es "ADV_SCAN_IND". Por lo tanto, se aplica el siguiente filtro:
   "(bthci_evt.bd_addr == 80:ea:ca:10:53:9c) && (bthci_evt.le_advts_event_type == 0x02)".

El resultado es el siguiente:



No.	Time	Source	Destination	Protocol	Length	Info
835	150.965326	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
837	151.204034	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
839	151.446403	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
841	151.693510	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
843	151.927058	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
845	152.175108	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1397	306.614850	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1399	306.863653	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1401	307.119975	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1403	307.356530	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1406	307.602070	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1429	307.839109	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1431	308.085791	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1433	308.335826	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1435	308.572361	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1437	308.822719	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1461	326.452674	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1463	326.703618	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1465	326.939568	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1467	327.176466	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1469	327.402937	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1515	336.375546	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1517	336.607143	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1519	336.850985	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1521	337.105449	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1523	337.357158	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1547	338.074153	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1548	338.074979	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1550	338.326248	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1552	338.571247	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1600	356.220784	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1602	356.472217	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1604	356.701947	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1606	356.954122	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1608	357.188678	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)
1610	357.425667	controller	host	HCI_EVT	42	Rcvd LE Meta (LE Advertising Report)

Figura 4.18: Captura y filtrado de paquetes con datos relevantes

Como se puede observar, Wireshark está mostrando únicamente los paquetes que contienen Datos específicos del fabricante y que provienen del sensor puesto a prueba.

Se selecciona uno de los paquetes y se analiza la información facilitada por el programa.

```
Bluetooth
    [Source: controller]
    [Destination: host]
Bluetooth HCI H4
Bluetooth HCI Event - LE Meta
    Event Code: LE Meta (0x3e)
    Parameter Total Length: 39
    Sub Event: LE Advertising Report (0x02)
    Num Reports: 1
    Event Type: Scannable Undirected Advertising (0x02)
    Peer Address Type: Public Device Address (0x00)
    BD_ADDR: DialogSe_10:53:9c (80:ea:ca:10:53:9c)
    Data Length: 27
  Advertising Data
    Flags
        Length: 2
        Type: Flags (0x01)
        000. .... = Reserved: 0x0
        ...0 .... = Simultaneous LE and BR/EDR to Same Device Capable (Host): false (0x0)
        .... 0... = Simultaneous LE and BR/EDR to Same Device Capable (Controller): false (0x0)
        .... .1.. = BR/EDR Not Supported: true (0x1)
        .... ..0. = LE General Discoverable Mode: false (0x0)
        .... ...1 = LE Limited Discoverable Mode: true (0x1)
    16-bit Service Class UUIDs
        Length: 3
        Type: 16-bit Service Class UUIDs (0x03)
        UUID 16: Unknown (0xfbb0)
    Manufacturer Specific
        Length: 19
        Type: Manufacturer Specific (0xff)
        Company ID: TomTom International BV (0x0100)
        Data: 80eaca10539c3ce40100e30800004b00
    RSSI: -62 dBm
```

Figura 4.19: Paquete de anuncio capturado

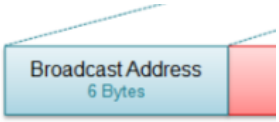
Podemos ver que Wireshark nos clasifica la trama y nos la presenta de forma sencilla. Esto se debe a que el programa conoce una gran cantidad de protocolos, entre ellos los de Bluetooth Low-Energy, por lo que puede interpretar todos los bytes recibidos excepto los específicos del sensor. A continuación, se identifican las partes del paquete de anuncio previamente estudiadas:

1. Cabecera del PDU de anuncio

Contiene el tipo de paquete de anuncio y el tipo de dirección de dispositivo

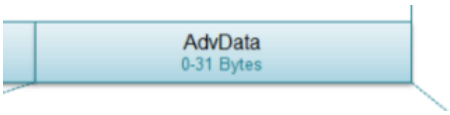
```
Event Type: Scannable Undirected Advertising (0x02)
Peer Address Type: Public Device Address (0x00)
```

2. Dirección del dispositivo




```
BD_ADDR: DialogSe_10:53:9c (80:ea:ca:10:53:9c)
```

3. Bloque de datos del paquete de anuncio




```
Data Length: 27
Advertising Data
```

4. Flags

```
Flags
    Length: 2
    Type: Flags (0x01)
    000. .... = Reserved: 0x0
    ...0 .... = Simultaneous LE and BR/EDR to Same Device Capable (Host): false (0x0)
    .... 0... = Simultaneous LE and BR/EDR to Same Device Capable (Controller): false (0x0)
    .... .1.. = BR/EDR Not Supported: true (0x1)
    .... ..0. = LE General Discoverable Mode: false (0x0)
    .... ...1 = LE Limited Discoverable Mode: true (0x1)
```

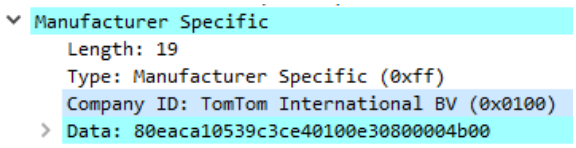
Contiene la longitud del bloque (*"Length: 2"*), el tipo de datos (*"Type: Flags (0x01)"*) y las flags que determinan el modo de detectabilidad y el procedimiento de descubrimiento. En este caso el dispositivo indica que no soporta BR/EDR (*Basic Rate/Enhanced Data Rate*) y que la detectabilidad y el procedimiento de descubrimiento son limitados, es decir, por tiempo limitado.

5. Service UUID

```
16-bit Service Class UUIDs
    Length: 3
    Type: 16-bit Service Class UUIDs (0x03)
    UUID 16: Unknown (0xfbb0)
```

Contiene la longitud del bloque (*"Length: 3"*), el tipo de datos (*"Type: 16-bit Service Class UUIDs (0x03)"*) y el UUID (*Universally unique identifier* o identificador único universal) que describe el servicio; que en este caso es desconocido por Wireshark.

6. Datos específicos del fabricante



Como se vio en el anterior apartado, este bloque también contiene la longitud y el tipo de datos. Wireshark interpreta los primeros dos bytes como identificador de la compañía y la encuentra en su base de datos. En este caso, los sensores pertenecen a *"TomTom International BV"*.

Por último, el bloque "Data" que contiene la cadena de bytes que nos aporta los datos relevantes. Aplicando el formato que se ha descrito en el apartado 4.5.2 se obtienen los siguientes valores:

Valor	Bytes	Conversión a decimal	Resultado
Nº DE SENSOR	80	-	**1**
PREFIJO DIRECCIÓN	eaca	-	**EA:CA**
ID DEL SENSOR	10539c	-	**10539c**
PRESIÓN	3ce40100	123964	**1,23964 BAR**
TEMPERATURA	e3080000	2275	**22,75ºC**
BATERÍA	4b	75	**75%**
ESTADO	00	0	**OK**

Tabla 4: Decodificación del bloque Data

Formato Little-endian Formato hexadecimal

4.6 CONCLUSIÓN

Esta fase del proyecto consta de tres etapas: comprender el funcionamiento de la tecnología Bluetooth Low-Energy, capturar el tráfico de datos entre la aplicación y los sensores y descifrar el formato de comunicación del TPMS.

La primera conclusión es que únicamente con conocimientos teóricos, probablemente no se habría llegado al resultado final, aunque si se hubieran ahorrado dos pasos: capturar con la placa Micro:Bit y analizar el código de la aplicación. También se habría evitado emplear tantas horas en buscar en la dirección equivocada.

La clave para conseguir los resultados finales fue el hallazgo de un repositorio en GitHub con una breve explicación que aclaraba el funcionamiento de este tipo de dispositivos Bluetooth. Este descubrimiento fue útil por un lado para dar con la solución final y por otro para ampliar los conocimientos teóricos de los que se debería haber partido.

A nivel personal considero que para la realización de un trabajo de investigación es muy importante conocer en profundidad los fundamentos teóricos acerca de la materia investigada. Por lo tanto, si desde un principio se hubiera conocido el tipo de comunicación propio de estos dispositivos, el trabajo realizado habría sido menos desconcertante y tedioso.

Haber investigado en profundidad acerca de la tecnología Bluetooth ha resultado muy interesante y un complemento muy importante para mi futuro profesional teniendo en cuenta que hoy en día el internet de las cosas (IoT) está en auge.

De aquí en adelante el proyecto se centrará en el desarrollo de una aplicación para Android que permita manejar los datos emitidos por los sensores, lo cual, plantea los siguientes desafíos: aprender a desarrollar una aplicación móvil y ofrecer una opción mejor a las que se presentan en el mercado.

Capítulo 5. Desarrollo de la Aplicación

5.1 ANÁLISIS DEL SISTEMA

5.1.1 Definición del Sistema

5.1.1.1 Determinación del Alcance del Sistema

El alcance de este sistema viene determinado por el hardware. El TPMS proporcionado por la Universidad es un modelo concreto que no se especifica en ninguna de las partes del dispositivo. El producto no contiene referencias a la marca o a la empresa que lo fabrica. Parece ser un modelo genérico de origen chino.

La única referencia al tipo de hardware que conforma los sensores se ha determinado a partir del repositorio encontrado en GitHub. Teniendo en cuenta que el formato de transmisión de los datos definido en el repositorio se ajusta al del modelo puesto a prueba en este proyecto, se puede concluir que la familia de los chips integrados en los sensores es la ESP32. Sin embargo, no se puede asegurar al 100% ya que puede haber otras familias de chips que utilicen ese mismo formato. Además, la aplicación móvil que se va a desarrollar implementará una interfaz con dos idiomas disponibles: español e inglés.

Teniendo en cuenta las dos características anteriormente definidas, el alcance de la aplicación se puede resumir en lo siguiente:

1. Usuarios que dispongan de un modelo de TPMS de la familia de chips ESP32.
2. Usuarios de habla inglesa o española.

5.1.2 Requisitos del Sistema

5.1.2.1 Identificación de Actores del Sistema

Teniendo en cuenta que la aplicación tendrá una base de datos local no será necesario un administrador que la gestione. El usuario será el único que tenga control sobre ella.

Por lo tanto, los actores del sistema serán los siguientes:

- Administrador: encargado del mantenimiento y soporte de la aplicación.
- Usuario

5.1.2.2 Obtención de los Requisitos del Sistema

5.1.2.2.1 Requisitos funcionales

Bluetooth

RF-1. El sistema debe comprobar si el Bluetooth del dispositivo está activado.

RF-1.1. Si se cumple RF-1, el sistema comprobará si el permiso de localización está concedido.

RF-1.1.1. Si no se cumple RF-1.1, el sistema solicitará al usuario el permiso de localización.

RF-1.1.1.1. Si el usuario no concede el permiso, el sistema lo volverá a solicitar.

RF-1.2. Si no se cumple RF-1, el sistema solicitará la activación del Bluetooth.

RF-1.2.1. Si el usuario no activa el Bluetooth, el sistema lo volverá a solicitar.

Monitorización

RF-2. El sistema debe mostrar la información enviada por los sensores que estén emparejados en el vehículo principal.

RF-2.1. El sistema debe mostrar los siguientes valores:

RF-2.1.1. Presión

RF-2.1.2. Temperatura

RF-2.1.3. Estado del sensor

RF-2.1.4. Batería del sensor

RF-2.2. El sistema debe mostrar los valores según las unidades de medida establecidas por el usuario

RF-2.2.1. El sistema debe permitir establecer las siguientes unidades de medida:

RF-2.2.1.1. Unidades de presión

RF-2.2.1.1.1. PSI

RF-2.2.1.1.2. BAR

RF-2.2.1.1.3. KPA

RF-2.2.1.2. Unidades de temperatura

RF-2.2.1.2.1. Celsius

RF-2.2.1.2.2. Fahrenheit

RF-3. El sistema debe notificar si los valores recibidos exceden los límites de advertencia establecidos por el usuario.

RF-3.1. El sistema debe permitir establecer los siguientes límites de advertencia:

RF-3.1.1. Presión

RF-3.1.1.1. Límite superior de advertencia

RF-3.1.1.2. Límite inferior de advertencia

RF-3.1.2. Temperatura

RF-3.1.2.1. Límite superior de advertencia

Emparejamiento

RF-4. El sistema debe mostrar los sensores emparejados en el vehículo principal.

RF-4.1. El sistema debe mostrar el identificador del sensor.

RF-5. El sistema debe permitir emparejar un sensor con una rueda.

RF-5.1. Emparejamiento manual

RF-5.1.1. El sistema debe permitir ingresar un máximo de 6 caracteres.

RF-5.1.2. El sistema debe comprobar si el número de caracteres ingresados es menor de 6.

RF-5.1.2.1. Si RF-4.1.2 se cumple, el sistema debe notificar que el número de caracteres mínimos es 6.

RF-5.2. Emparejamiento automático

RF-5.2.1. El sistema debe realizar un escaneo de nuevos sensores durante un tiempo limitado de 30 segundos.

RF-5.2.2. El sistema debe notificar si el escaneo no obtiene resultados.

RF-6. El sistema debe permitir desemparejar un sensor.

Gestión de vehículos

RF-7. El sistema debe permitir establecer un vehículo como principal.

RF-7.1. El máximo número de vehículos principales será 1.

RF-8. El sistema debe permitir añadir un vehículo.

RF-8.1. El sistema debe requerir un nombre de vehículo.

RF-8.1.1. El sistema debe comprobar si la cadena de texto está vacía.

RF-8.1.1.1. Si se cumple RF-7.1.1, el sistema debe notificar que es un campo obligatorio.

RF-8.2. El sistema debe requerir el tipo de vehículo.

RF-8.2.1. El tipo de vehículo debe contener el número de ruedas y la disposición de estas.

RF-9. El sistema debe permitir eliminar uno o varios vehículos.

RF-9.1. El sistema debe comprobar si entre los vehículos seleccionados se encuentra el vehículo principal.

RF-9.1.1. Si se cumple RF-8.1, el sistema debe notificar que no se puede eliminar el vehículo principal.

RF-10. El sistema debe permitir editar el nombre de un vehículo.

RF-10.1. El sistema debe comprobar si la cadena de texto está vacía.

RF-10.1.1. Si se cumple RF-7.1.1, el sistema debe notificar que es un campo obligatorio.

RF-11. El sistema debe mostrar los vehículos creados por el usuario.

Otros

RF-12. El sistema debe permitir establecer el tema de la aplicación.

RF-12.1. Tema del sistema

RF-12.2. Tema oscuro

RF-12.3. Tema claro

RF-13. El sistema debe mostrar la información acerca del desarrollador de la aplicación.

5.1.2.2.2 Requisitos no funcionales

RNF-1. La aplicación requerirá una versión de Android 5.0 o superior

RNF-2. La aplicación requerirá una versión de Bluetooth 4.0 o superior.

RNF-3. La aplicación tendrá una interfaz sencilla e intuitiva que permita su utilización a usuarios con conocimientos básicos en dispositivos móviles.

RNF-4. La aplicación será multilenguaje

RNF-4.1. Inglés

RNF-4.2. Español

5.1.2.3 Especificación de Casos de Uso

En este apartado se muestran tres diagramas de casos de uso seguidos de una descripción de cada uno de los casos. Cada diagrama representa una de las funciones generales de la aplicación:

1. Monitorización de sensores

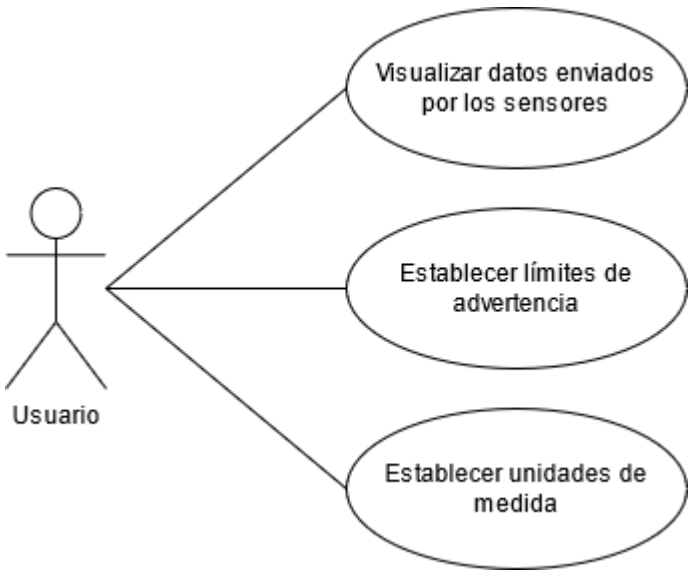


Figura 5.1: Diagrama de Casos de Uso de Monitorización de sensores

Nombre del Caso de Uso
Visualizar datos enviados por los sensores
Descripción
El usuario debe poder visualizar los datos enviados por los sensores en tiempo real y si estos están emparejados con las ruedas del vehículo principal.

Tabla 5: Especificación Caso de uso: Visualizar datos enviados por los sensores

Nombre del Caso de Uso
Establecer límites de advertencia
Descripción
El usuario debe poder establecer los límites de advertencia para los valores de temperatura y presión enviados por los sensores.

Tabla 6: Especificación Caso de uso: Establecer límites de advertencia

Nombre del Caso de Uso
Establecer unidades de medida
Descripción
El usuario debe poder establecer las unidades de medida para los valores de temperatura y presión enviados por los sensores.

Tabla 7: Especificación Caso de uso: Establecer unidades de medida

2. Emparejamiento de sensores

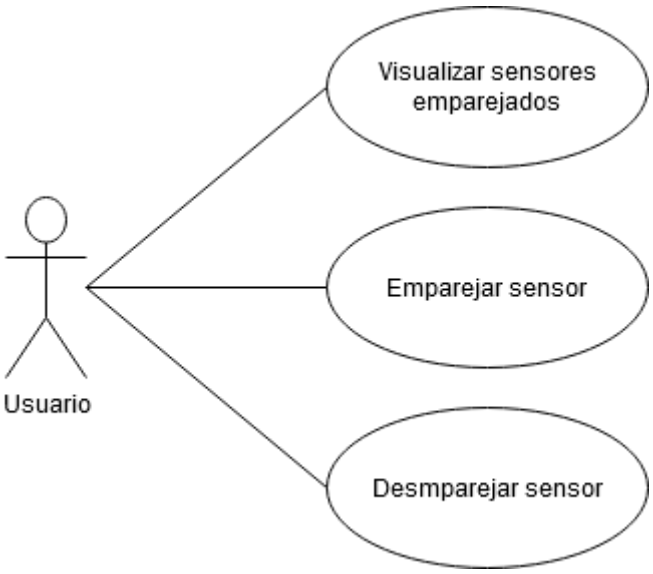


Figura 5.2: Diagrama de Casos de Uso de Emparejamiento de sensores

Nombre del Caso de Uso
Visualizar sensores emparejados
Descripción
El usuario debe poder visualizar los sensores están emparejado con las ruedas del vehículo principal.

Tabla 8: Especificación Caso de uso: Visualizar sensores emparejados

Nombre del Caso de Uso
Emparejar sensor
Descripción
El usuario debe poder emparejar un sensor con una rueda del vehículo principal. Un sensor se puede sobrescribir. El emparejamiento se puede realizar de dos modos diferentes: ingresando el identificador del sensor o escaneando nuevos sensores.

Tabla 9: Especificación Caso de uso: Emparejar sensor

Nombre del Caso de Uso
Desemparejar sensor
Descripción
El usuario debe poder desemparejar un sensor del vehículo principal.

Tabla 10: Especificación Caso de uso: Desemparejar sensor

3. Gestión de vehículos

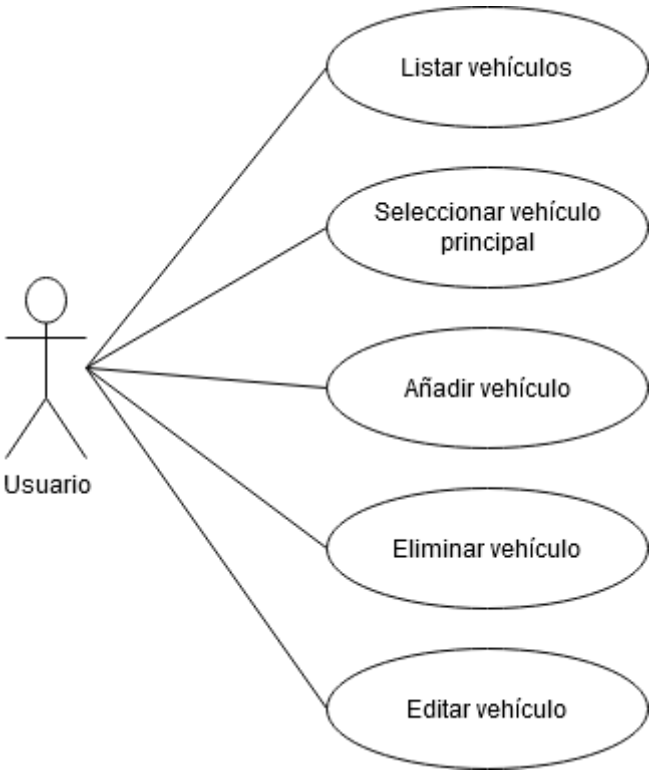


Figura 5.3: Diagrama de Casos de Uso de Gestión de vehículos

Nombre del Caso de Uso
Listar vehículos
Descripción
El usuario debe poder obtener la lista de vehículos guardados en la base de datos local. El usuario debe poder ver cuál de todos es el vehículo principal.

Tabla 11: Especificación Caso de uso: Listar vehículos

Nombre del Caso de Uso
Seleccionar vehículo principal
Descripción
El usuario debe poder seleccionar o cambiar el vehículo principal. Solo puede haber un vehículo principal.

Tabla 12: Especificación Caso de uso: Seleccionar vehículo principal

Nombre del Caso de Uso
Añadir vehículo
Descripción
El usuario debe poder añadir o crear un vehículo. El vehículo debe tener un nombre, un número determinado de ruedas y una disposición de las ruedas concreta.

Tabla 13: Especificación Caso de uso: Añadir vehículo

Nombre del Caso de Uso
Eliminar vehículo
Descripción
El usuario debe poder eliminar uno o más vehículos siempre que no sean el vehículo principal.

Tabla 14: Especificación Caso de uso: Eliminar vehículo

Nombre del Caso de Uso
Editar vehículo
Descripción
El usuario debe poder editar el nombre del vehículo.

Tabla 15: Especificación Caso de uso: Editar vehículo

5.1.3 Identificación de los Subsistemas en la Fase de Análisis

Se han identificado tres subsistemas en los que se dividirá la aplicación. La interacción entre ellos se muestra en el siguiente diagrama.

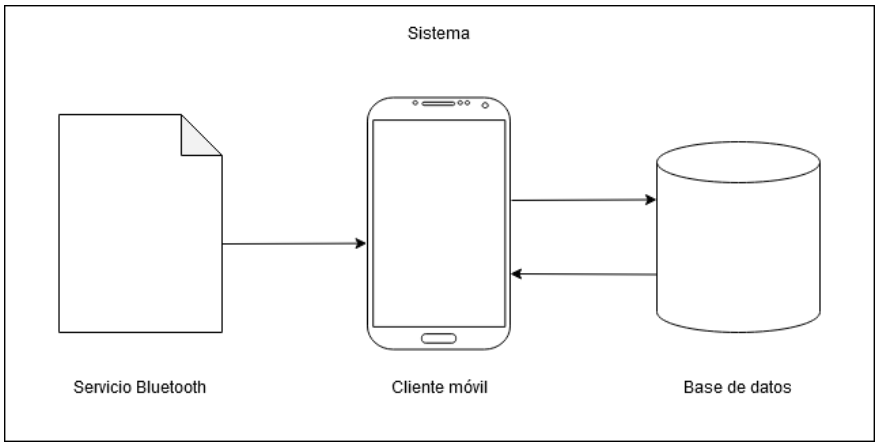


Figura 5.4: Diagrama de Subsistemas

5.1.3.1 Cliente móvil

El cliente móvil es el subsistema que contiene la mayor parte de la lógica y el que interactúa directamente con el usuario. Contiene todas las interfaces gráficas y además sirve de puente entre el servicio Bluetooth y la base de datos. El cliente móvil recibe los datos que escanea el servicio Bluetooth, los muestra al usuario y almacena parte de ellos en la base datos.

5.1.3.2 Servicio Bluetooth

El servicio Bluetooth es el subsistema que interactúa con el hardware, es decir, con los sensores que emiten datos. Contiene toda la lógica relativa al manejo del servicio Bluetooth de Android. Entre sus funciones principales se encuentran:

- Iniciar el servicio Bluetooth al iniciar la aplicación.
- Dar permisos de localización a la aplicación.
- Capturar los paquetes de datos mediante un escaneo.
- Buscar nuevos dispositivos que estén escaneando.
- Interpretar los datos recibidos.

- Enviar los datos al cliente móvil.

5.1.3.3 Base de datos

La base de datos es el subsistema encargado de almacenar las configuraciones realizadas por el usuario relativas a los vehículos. Es un base de datos local que consta de las siguientes partes:

- El servicio de bases de datos
- Las entidades
- Los objetos de acceso a datos (DAO)
- Los repositorios

5.1.4 Diagrama y Descripción de Clases

5.1.4.1 Diagrama de Clases

A continuación, se muestra el diagrama de clases preliminar el cual incluye las clases básicas necesarias para cada uno de los subsistemas anteriormente definidos.

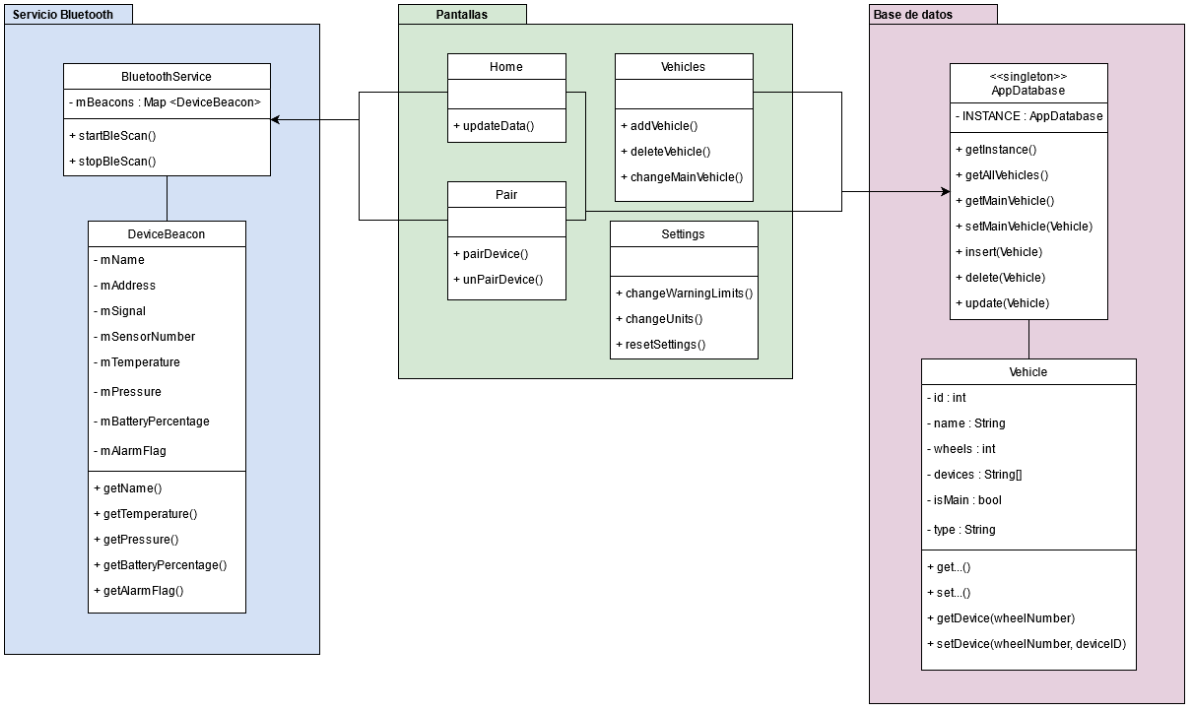


Figura 5.5: Diagrama de Clases Preliminar

5.1.4.2 Descripción de las Clases

En este apartado se realiza una breve descripción de las clases definidas en el diagrama y de los métodos y atributos propuestos.

5.1.4.2.1 Servicio Bluetooth

Nombre de la clase
BluetoothService
Descripción
Es la encargada de iniciar y controlar el servicio Bluetooth del dispositivo móvil. Realiza los escaneos y encapsula los datos capturados.
Atributos propuestos
mBeacons: es un Map que almacena las balizas de datos capturadas.
Métodos propuestos
+ startBleScan: inicia el escaneo. + stopBleScan: detiene el escaneo.

Tabla 16: Descripción de la clase BluetoothService

Nombre de la clase
DeviceBeacon
Descripción
Es la clase entidad de las balizas de datos. Se encarga de procesar y formatear los datos obtenidos en crudo en el escaneo.
Atributos propuestos
mName: nombre del dispositivo. mAddress: dirección del dispositivo. mSignal: potencia de la señal. mSensorNumber: número de identificación del dispositivo. mTemerature: valor de la temperatura. mPressure: valor de la presión. mBatteryPercentage: porcentaje de batería restante del dispositivo. mAlarmFlag: señal de alarma del dispositivo.
Métodos propuestos
+ getters: métodos “get” para todos los atributos.

Tabla 17: Descripción de la clase DeviceBeacon

5.1.4.2.2 Pantallas

Nombre de la clase
Home
Descripción
Es la encargada de mostrar los datos al usuario.
Atributos propuestos
-
Métodos propuestos
+ updateData: se encarga de actualizar la interfaz cuando llegan nuevos datos.

Tabla 18: Descripción de la clase Home

Nombre de la clase
Pair
Descripción
Es la encargada de emparejar los dispositivos con las ruedas del vehículo.
Atributos propuestos
-
Métodos propuestos
+ pairDevice: empareja el dispositivo indicado con la rueda seleccionada. +unPairDevice: desempareja el dispositivo en la rueda seleccionada.

Tabla 19: Descripción de la clase Pair

Nombre de la clase
Vehicles
Descripción
Es la encargada de añadir y eliminar los vehículos del usuario a la base de datos.
Atributos propuestos
-
Métodos propuestos
+ addVehicle: añade el vehículo a la base de datos. + deleteVehicle: elimina el vehículo de la base de datos. + changeMainVehicle: selecciona el vehículo principal.

Tabla 20: Descripción de la clase Vehicles

Nombre de la clase
Settings
Descripción
Es la encargada de establecer los ajustes del usuario.
Atributos propuestos
-
Métodos propuestos
+ changeWarningLimits: establece los límites de advertencia para la presión y temperatura. + changeUnits: establece la unidida de medida de presión y temperatura. + resetSettings: restablece los ajustes a los valores por defecto.

Tabla 21: Descripción de la clase Settings

5.1.4.2.3 Base de datos

Nombre de la clase
AppDatabase
Descripción
Es la encargada de iniciar la base de datos.
Atributos propuestos
INSTANCE: instancia única de la base de datos.
Métodos propuestos
+ getInstance: obtiene la instancia de la base de datos. + getAllVehicles: obtiene todos los vehículos. + getMainVehicle: obtiene el vehículo principal. + setMainVehicle: actualiza el vehículo marcado como principal. + insert, delete, update: operaciones básicas de escritura en la base de datos.

Tabla 22: Descripción de la clase AppDatabase

Nombre de la clase
Vehicle
Descripción
Es la entidad Vehículo.
Atributos propuestos
Id: identificador único del vehiculo. Name: nombre del vehículo. Wheels: número de rueda del vehículo. Devices: Array de dispositivos emparejados. isMain: valor booleano que indica si el vehículo está marcado como principal. Type: tipo de vehículo según la disposición de las ruedas.
Métodos propuestos
+ getters: métodos “get” para todos los atributos. + setters: métodos “set” para todos los atributos. + setDevice: establece el identificador del dispositivo con el número de rueda indicado. + getDevice: devuelve el identificador de dispositivo dado el número de rueda.

Tabla 23: Descripción de la clase Vehicle

5.1.5 Análisis de Casos de Uso y Escenarios

5.1.5.1 Caso de Uso 1: Visualizar datos enviados por los sensores

Visualizar datos enviados por los sensores	
Precondiciones	Permiso de localización concedido. Bluetooth encendido. Al menos un vehículo registrado. Al menos un sensor emparejado. Los sensores emparejados tienen que estar correctamente instalados y dentro del rango de distancia de BLE 4.0 (hasta 100 metros) respecto al dispositivo móvil.
Postcondiciones	-
Actores	Usuario
Descripción	El usuario accederá a la vista “Inicio” y el escaneo empezará automáticamente. Cuando el dispositivo reciba los datos los mostrará en pantalla identificando la rueda que los envía. Los datos mostrados serán: Temperatura Presión Además, se mostrarán al usuario las siguientes señales de advertencia: La señal es estable o inestable. La temperatura excede los límites de advertencia. La presión excede los límites de advertencia. La batería restante del sensor es del 25% o menor.
Escenarios secundarios	Se desactiva el Bluetooth del dispositivo móvil: el escaneo se detiene y se obliga al usuario a activarlo de nuevo. Se desinstala el sensor: se indica al usuario mediante la señal de estabilidad de señal.

Tabla 24: Caso de Uso 1: Visualizar datos enviados por los sensores

5.1.5.2 Caso de Uso 2: Establecer límites de advertencia

Establecer límites de advertencia	
Precondiciones	-
Postcondiciones	-
Actores	Usuario
Descripción	El usuario modifica los valores por defecto para los límites de advertencia. Se podrán establecer límites superiores de temperatura y los limites superiores e inferiores de presión. Cuando los dateos recibidos excedan dichos límites se informará al usuario.
Escenarios secundarios	Se modifican los límites de advertencia durante la recepción de datos: se actualiza la última baliza de datos recibida.

Tabla 25: Caso de Uso 2: Establecer límites de advertencia

5.1.5.3 Caso de Uso 3: Establecer unidades de medida

Establecer unidades de medida	
Precondiciones	-
Postcondiciones	-
Actores	Usuario
Descripción	El usuario modifica los valores por defecto para las unidades de medida. Se podrán establecer las unidades de medida para la temperatura y para la presión.
Escenarios secundarios	Se modifican las unidades de medida durante la recepción de datos: se actualiza la última baliza de datos recibida.

Tabla 26: Caso de Uso 3: Establecer unidades de medida

5.1.5.4 Caso de Uso 4: Visualizar sensores emparejados

Visualizar sensores emparejados	
Precondiciones	Al menos un vehículo registrado. Al menos un sensor emparejado.
Postcondiciones	-
Actores	Usuario
Descripción	El usuario podrá visualizar los sensores emparejados a cada rueda del vehículo.
Escenarios secundarios	-

Tabla 27: Caso de Uso 4: Visualizar sensores emparejados

5.1.5.5 Caso de Uso 5: Emparejar sensor con una rueda

Emparejar sensor con una rueda	
Precondiciones	Al menos un vehículo registrado.
Postcondiciones	Las ruedas con sensores emparejados deberán mostrar los datos en la pantalla “Inicio” siempre y cuando el sensor esté emitiendo datos.
Actores	Usuario
Descripción	El usuario podrá emparejar un sensor a cada rueda de un vehículo. El emparejamiento se realiza mediante el identificador del sensor y la posición de la rueda en la que se instalará. El usuario podrá elegir dos métodos para realizar el emparejamiento: Manual Automático
Escenarios secundarios	El usuario elige el emparejamiento manual: se requerirá introducir el identificador del sensor a emparejar. El usuario elige el emparejamiento automático: comenzará un escaneo en busca de sensores que estén emitiendo sin estar actualmente emparejados. Se requerirá activar el Bluetooth en caso de no estar activado.

Tabla 28: Caso de Uso 5: Emparejar sensor con una rueda

5.1.5.6 Caso de Uso 6: Desemparejar sensor

Desemparejar sensor	
Precondiciones	Al menos un vehículo registrado. Al menos un sensor emparejado.
Postcondiciones	Las ruedas desemparejadas dejarán de mostrar datos en la pantalla “Inicio”.
Actores	Usuario
Descripción	El usuario podrá desemparejar el sensor de una rueda.
Escenarios secundarios	-

Tabla 29: Caso de Uso 6: Desemparejar sensor

5.1.5.7 Caso de Uso 7: Visualizar vehículos

Visualizar vehículos	
Precondiciones	-
Postcondiciones	-
Actores	Usuario
Descripción	El usuario podrá visualizar los vehículos que ha creado. Se mostrará el nombre y si el vehículo está designado como principal. Solo podrá haber un vehículo principal.
Escenarios secundarios	-

Tabla 30: Caso de Uso 7: Visualizar vehículos

5.1.5.8 Caso de Uso 8: Añadir, editar y eliminar vehículos

Añadir, editar y eliminar vehículos	
Precondiciones	Para editar o eliminar, debe haber al menos un vehículo. Para eliminar un vehículo, este no podrá ser el vehículo principal
Postcondiciones	Al añadir, editar o eliminar la lista se actualizará inmediatamente. Al añadir un vehículo, este pasará a ser el vehículo principal.
Actores	Usuario
Descripción	Añadir: el usuario podrá registrar un nuevo vehículo, indicando su nombre y el tipo de vehículo según el número de ruedas y su disposición. Editar: el usuario podrá modificar el nombre del vehículo. Eliminar: el usuario podrá eliminar uno o varios vehículos a la vez.
Escenarios secundarios	Al añadir se mostrará un mensaje de error si el campo del nombre está vacío o el tipo de vehículo no ha sido seleccionado. Al editar se mostrará un mensaje de error si el campo del nombre está vacío. Al eliminar se mostrará un mensaje de error si entre los seleccionados se encuentra el vehículo principal.

Tabla 31: Caso de Uso 8: Añadir, editar y eliminar vehículos

5.1.5.9 Caso de Uso 9: Seleccionar vehículo principal

Seleccionar vehículo principal	
Precondiciones	Al menos dos vehículos registrados. Uno principal y uno secundario.
Postcondiciones	Al seleccionar el nuevo vehículo principal la lista se actualizará inmediatamente.
Actores	Usuario
Descripción	El usuario podrá cambiar el vehículo principal. De todos los vehículos creados siempre habrá uno designado como principal. Solo puede haber un vehículo principal.
Escenarios secundarios	El vehículo designado como principal será el que se muestre en las demás pantallas de la aplicación.

Tabla 32: Caso de Uso 9: Seleccionar vehículo principal

5.1.6 Análisis de Interfaces de Usuario

5.1.6.1 Descripción de la Interfaz

5.1.6.1.1 Principal

La pantalla principal de la aplicación mostrará al usuario los datos enviados por los sensores mediante una pequeña ventana para cada rueda del vehículo. La disposición dependerá del vehículo seleccionado como principal, es decir, dependiendo del número de ruedas y la disposición de estas se mostrarán más o menos ventanas y de mayor o menor tamaño. En el centro se mostrará un dibujo o imagen del tipo de vehículo seleccionado.

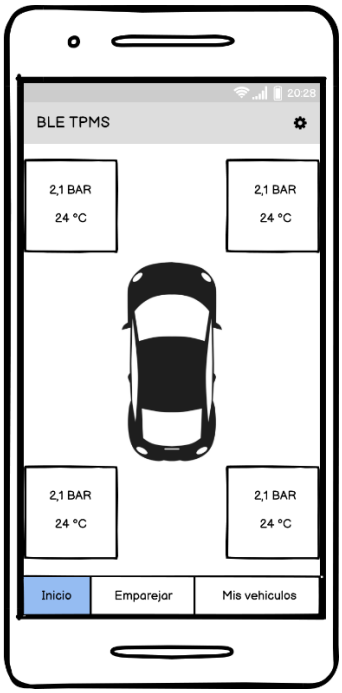


Figura 5.6: Diseño inicial de la pantalla Principal

5.1.6.1.2 Emparejar

Esta pantalla mostrará una estructura y disposición similar a la principal. En cambio, las ventanas pasarán a mostrar el identificador del sensor, en caso de estar emparejado, y funcionarán como botones que iniciarán ventanas modales en las que realizar las operaciones de emparejamiento.

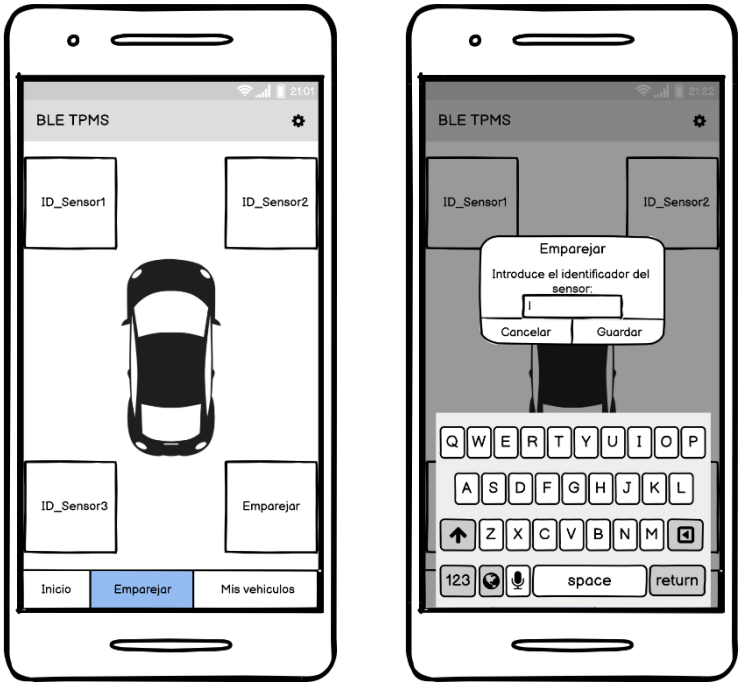


Figura 5.7: Diseño inicial de la pantalla Emparejar

5.1.6.1.3 Vehículos

Esta pantalla consistirá en un listado de vehículos que muestren el nombre y un botón para establecer el vehículo principal. En un principio el manejo de la lista será el habitual en Android, es decir, clic prolongado para seleccionar un ítem y múltiple selección permitida.

En la parte inferior derecha se incluirá un botón que despliegue una ventana modal con la que se permita al usuario añadir nuevos vehículos.

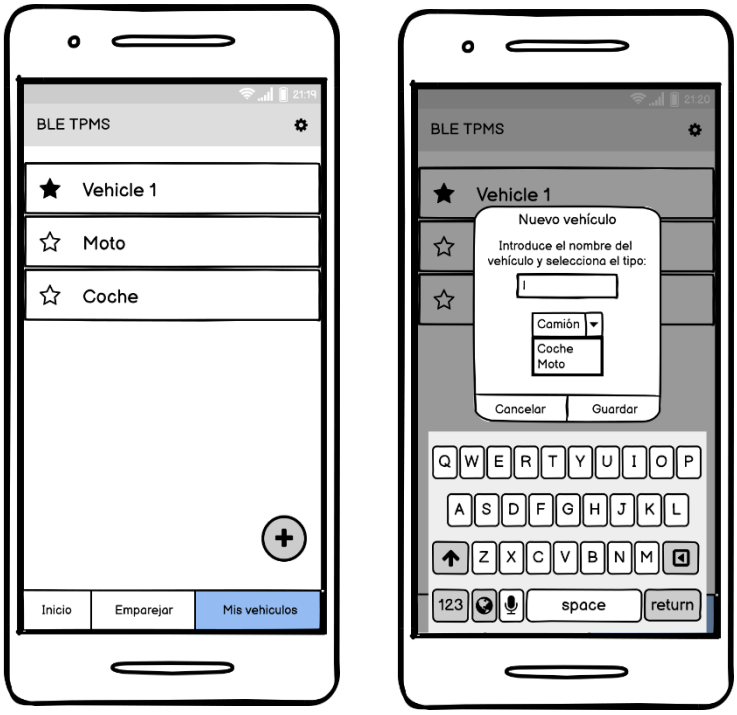


Figura 5.8: Diseño inicial de la pantalla Vehículos

5.1.6.1.4 Ajustes

Figura 5.9: Diseño inicial de la pantalla Ajustes

5.1.6.2 Descripción del Comportamiento de la Interfaz

Como se ha descrito en el apartado anterior, el diseño inicial de la aplicación consta de cuatro pantallas principales. Para acceder a ellas se dispondrá de una barra fija en la parte inferior a modo de menú con la que el usuario se podrá desplazar fácilmente entre las tres ventanas que facilitan las funcionalidades principales de la aplicación. Para acceder a la cuarta ventana, la de ajustes de la aplicación, se dispondrá de un botón en la parte superior derecha.

El comportamiento de la interfaz es muy sencillo e intuitivo ya que la funcionalidad de la aplicación es muy simple. Una pantalla para monitorizar datos, otra para emparejar dispositivos ya sea introduciendo un identificador o de forma automática mediante escaneo Bluetooth, otra pantalla que hace la función de gestora de vehículos (añadir, editar y eliminar), y por último una de ajustes.

Se añadirá información extra mediante mensajes de error, claros y concisos, que indiquen al usuario instrucciones relevantes en los procedimientos de introducción de texto o borrado de vehículos, así como múltiples ventanas de confirmación para dar al usuario una segunda oportunidad a la hora de ejecutar acciones importantes.

5.1.6.3 Diagrama de Navegabilidad

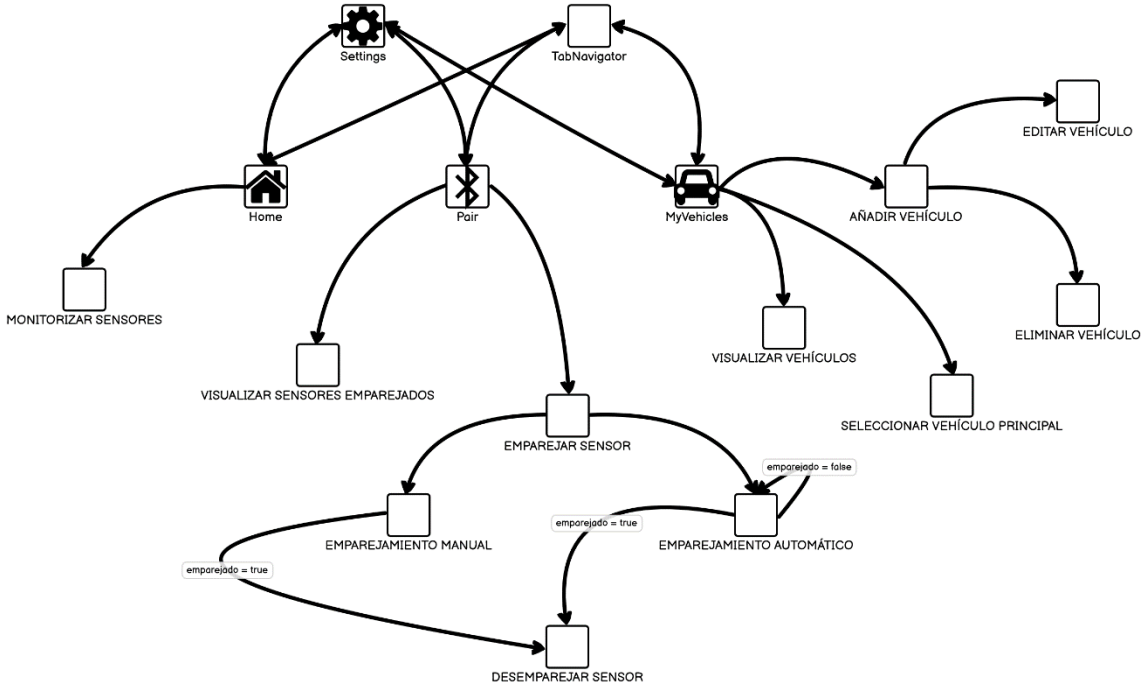


Figura 5.10: Diagrama de navegabilidad

5.1.7 Especificación del Plan de Pruebas

5.1.7.1 Pruebas Unitarias

Las pruebas unitarias estarán enfocadas a las partes del código de uso común, es decir, a las clases y funciones públicas utilizadas por diferentes partes de la aplicación.

Se desarrollarán pruebas unitarias para asegurar el correcto funcionamiento de la configuración del servicio de Bluetooth. De esta manera se comprobará la estabilidad de este en diferentes versiones de Android.

También se comprobará la solidez de la clase que formatee los datos obtenidos en crudo enviados por los sensores. Esta clase se encargará de transformar una cadena de bytes en los diferentes valores que se conoce que aportan los sensores, por lo tanto, será necesario comprobar que los tipos y los cambios de unidades de medida son correctos.

5.1.7.2 Pruebas de Integración

Las pruebas de integración se encargarán de comprobar que los diferentes módulos de la aplicación funcionan correctamente en conjunto, de manera que, los casos de uso descritos en el punto 5.1.5 reproduzcan los resultados esperados.

Gran parte de la aplicación se basa en la interacción del usuario por lo que se realizarán pruebas de interfaz con la intención de simular todo tipo de acciones posibles. Además de verificar que la interfaz se comporta de la manera esperada se tendrán en cuenta los resultados de las acciones relativas a la base de datos, comprobando si se muestran los registros pertinentes y si se actualiza inmediatamente y de forma correcta.

5.1.7.3 Pruebas del Sistema

Las pruebas del sistema serán las encargadas de comprobar que la conexión de la aplicación con la base de datos es correcta evitando así errores en tiempo de ejecución y mala persistencia en los datos.

Además, se comprobará que el servicio Bluetooth de Android se inicia de forma correcta y sin fallos, algo fundamental, teniendo en cuenta que es el pilar básico de esta aplicación.

5.1.7.4 Pruebas de Usabilidad

Para las pruebas de usabilidad se realizará un estudio en un entorno real donde usuarios de edades comprendidas entre los 18 y 75 años utilicen la aplicación durante unos días para finalmente dar un feedback mediante unos sencillos formularios.

El objetivo de estos será conocer lo intuitiva que es la aplicación y que tipo de mejoras en la interfaz se pueden llevar a cabo en un futuro.

5.2 DISEÑO DEL SISTEMA

5.2.1 Arquitectura del Sistema

5.2.1.1 Diagrama de Paquetes

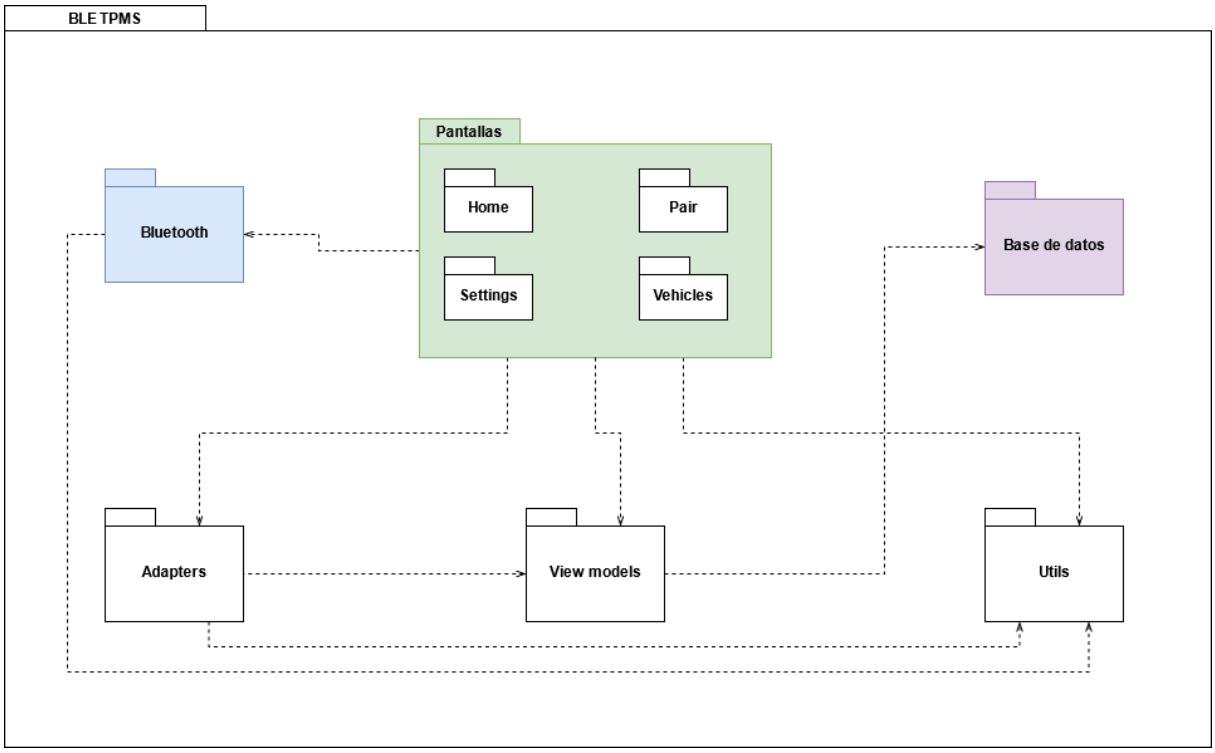


Figura 5.11: Diagrama de paquetes

El diagrama anterior representa el diseño inicial de la arquitectura de la aplicación según los paquetes y subpaquetes en los que se dividirá el código. A continuación, se describen cada uno de ellos:

- Pantallas: es el paquete correspondiente a la interfaz de usuario. Se divide en 4 subpaquetes, cada uno correspondiente a una de las pantallas principales de la aplicación. Interactúa directamente con el resto de los paquetes a excepción de la base de datos.
- Base de datos: contiene toda la lógica de la base de datos: Constructor, repositorio, dao, entidades… Se comunica con la interfaz a través de los "view models" o modelos-vista.

- View models: o modelos-vista, clases encargadas de almacenar y administrar los datos relacionados con la interfaz de usuario de manera que se conserven cuando se realicen cambios en esta.
- Bluetooth: contiene toda la lógica del servicio Bluetooth: inicio y configuración del servicio, escaneos, formateo de datos obtenidos...
- Adapters: contiene las clases encargadas de actualizar los elementos de las vistas que trabajen con la base de datos.
- Utils: contiene clases con código auxiliar de uso común en el resto de los paquetes.

5.2.1.2 Diagrama de Despliegue

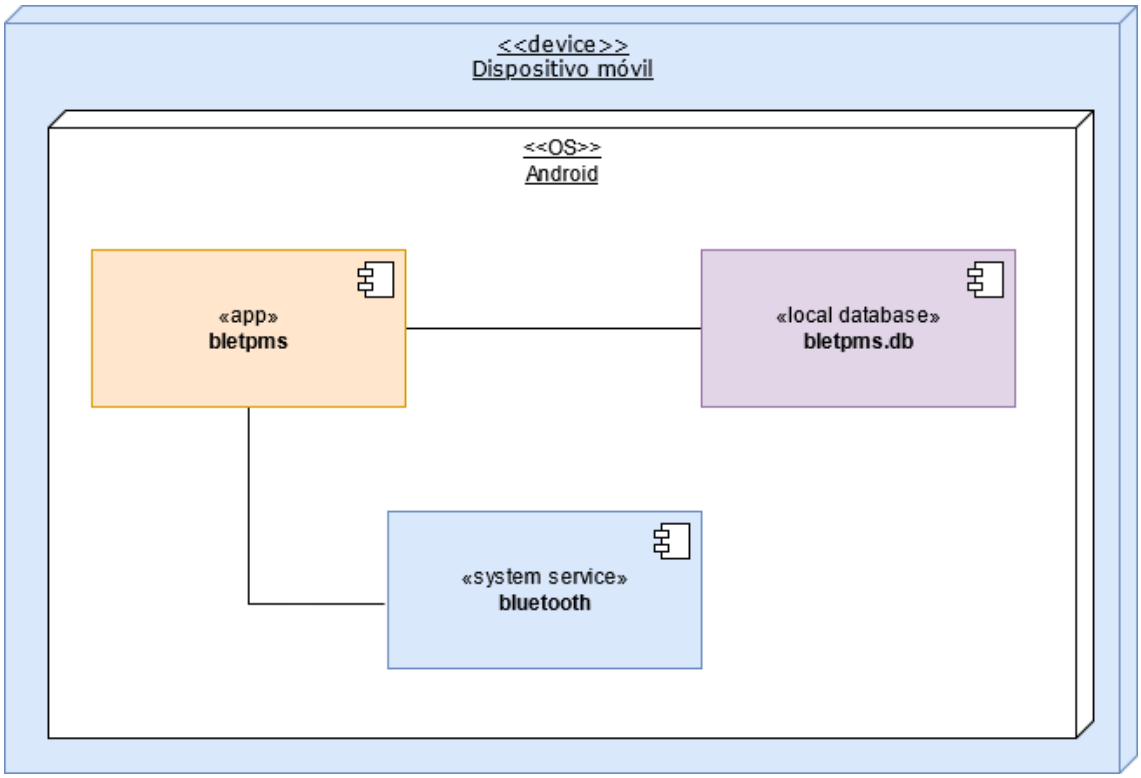


Figura 5.12: Diagrama de despliegue

5.2.2 Diseño de Clases

5.2.2.1 Diagrama de Clases

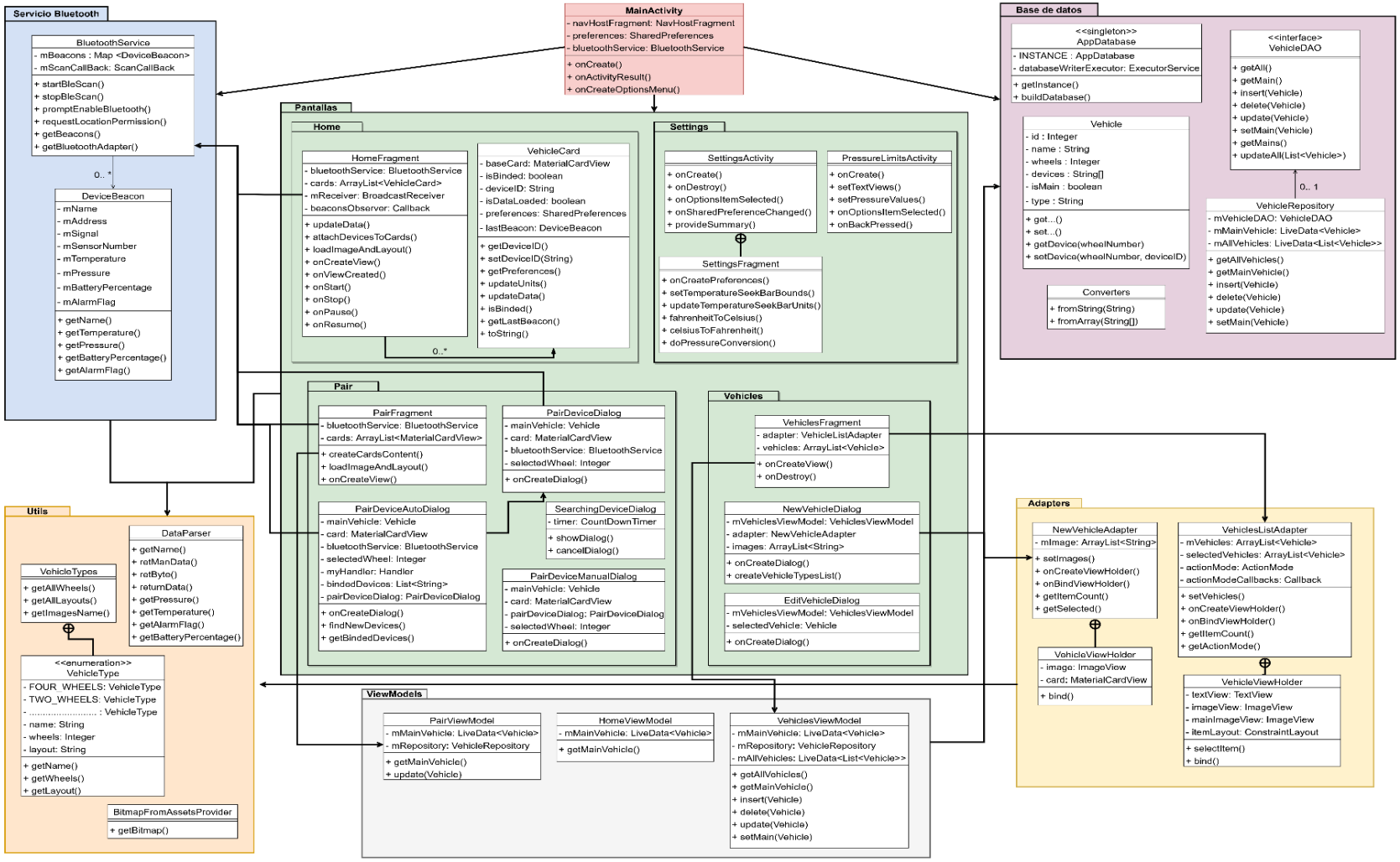


Figura 5.13: Diagrama de clases

5.2.2.2 Descripción de las Clases

- **MainActivity**: inicia la aplicación. Crea el navegador entre pantallas (NavHostFragment), el archivo de preferencias/ajustes (SharedPreferences) e inicia el servicio Bluetooth (BluetoothService).

5.2.2.2.1 Pantallas

- Home
  - **HomeFragment**: clase de la pantalla "Inicio". Muestra al usuario los datos enviados por los sensores. Se encarga de cargar el layout correspondiente al vehículo seleccionado como principal, el cual obtiene observando el campo "mainVehicle" en la base de datos. Para cada rueda del vehículo crea un objeto VehicleCard que extiende a MaterialCardView. Finalmente es la encargada de inicia el escaneo Bluetooth a través del servicio Bluetooth y de observar el ArrayMap de balizas de datos (DeviceBeacon) que se va actualizando con cada resultado del escaneo. De esta manera cuando el ArrayMap cambia la IU se actualiza.
  - **VehicleCard**: clase que extiende a MaterialCardView. Se encarga de la lógica de cada rueda del vehículo. Recibe la baliza de datos (DeviceBeacon) correspondiente y actualiza los valores en la IU.

- Pair
  - **PairFragment**: clase de la pantalla "Emparejar". Muestra al usuario las ruedas del vehículo con sensores emparejados y permite emparejar o desemparejar. Carga el layout de la misma forma que HomeFragment. Las vistas MaterialCardView correspondientes a cada rueda añaden funcionalidad a modo de botones que inicia ventanas de diálogo para emparejar o desemparejar.
  - **PairDeviceDialog**: ventana de diálogo que muestra las opciones de emparejamiento (Manual o Automático) y la opción "Desemparejar" en caso de la rueda seleccionada ya se encuentre emparejada con un sensor.

  - **PairDeviceManualDialog**: ventana de diálogo que permite ingresar el identificador del sensor a emparejar o editar el que ya se encuentra emparejado.
  - **PairDeviceAutoDialog**: ventana de diálogo que muestra un mensaje con instrucciones para realizar el escaneo en busca de nuevos sensores. Esta clase se encarga de realizar un escaneo mediante el servicio Bluetooth y de actualizar la base de datos y la IU cuando se encuentre un nuevo dispositivo.
  - **SearchingDeviceDialog**: ventana de diálogo que muestra una cuenta atrás durante el escaneo de nuevos sensores. Mediante un Handler controla la duración del escaneo de manera que finaliza el escaneo al terminar un tiempo determinado por el campo "SEARCHING_TIME_MS".
- Vehicles
  - **VehiclesFragment**: clase de la pantalla "Mis vehículos". Permite al usuario crear, editar y eliminar vehículos. También permite seleccionar el vehículo principal. Consta de una RecyclerView que se actualiza automáticamente cuando la lista de vehículos cambia. Utiliza VehicleListAdapter para mostrar y controlar los ítems de la lista. Implementa un menú ActionMode para las acciones de editar y eliminar vehículos, y un botón que despliega una ventana de diálogo para crear vehículos.
  - **NewVehicleDialog**: ventana de diálogo para crear vehículos. Implementa el adapter "NewVehicleAdapter" para mostrar los diferentes tipos de vehículos seleccionables.
  - **EditVehicleDialog**: ventana de dialogo para editar el nombre de los vehículos.
- Settings
  - **SettingsActivity**: clase que implementa la base de la pantalla "Ajustes".
  - **SettingsFragment**: clase interna de SettingsActivity que implementa el contenido de la pantalla "Ajustes". Es la encargada de actualizar el archivo de preferencias (SharedPreferences).
  - **PressureLimitsActivity**: clase que implementa la pantalla para establecer los límites de advertencia de presión. Se accede a través de una de las opciones de SettingsFragment.

5.2.2.2.2 Bluetooth

- **BluetoothService**: clase que implementa la lógica Bluetooth. Contiene métodos para pedir permiso de localización al usuario, solicitar la activación del Bluetooth del dispositivo, comenzar y detener el escaneo. También define un ObservableArrayMap de tipo DeviceBeacon al cual se van añadiendo los resultados del escaneo.
- **DeviceBeacon**: clase que se encarga de extraer los valores de cada resultado del escaneo, es decir, transforma la cadena de bytes obtenida en valores de temperatura, presión, etc.

5.2.2.2.3 Base de datos

- **AppDatabase**: clase abstracta y singleton que define la base de datos Room. Contiene el método para obtener la instancia y el DAO de vehículos además de una instancia de un servicio de ejecución en un solo hilo el cual se encargará de las operaciones de escritura.
- **Vehicle**: clase de la entidad Vehículo.
- **VehicleDAO**: clase que contiene las consultas SQL.
- **VehicleRepository**: clase que maneja las operaciones de la base de datos.
- **Converters**: clase que contiene métodos auxiliares de conversión de tipos para la base de datos.

5.2.2.2.4 ViewModels

Las clases **ViewModel** o ModeloVista se encargan de gestionar los datos que mostrarán las clases de la IU. Permite que se conserven los datos después de cambios de configuración, como las rotaciones de pantallas.

Los tres ViewModels de esta aplicación contienen métodos que operan con la base de datos.

5.2.2.2.5 Adapters

Las clases Adapter se encargan de enlazar los datos con las vistas de tipo RecyclerView. Esta aplicación dispondrá de dos:

- **VehicleListAdapter**: adaptador para la lista que muestra los vehículos registrados por el usuario. También se encarga de la selección múltiple y del

menú de acción (ActionMode) que muestra las acciones de editar y eliminar, y de la lógica de cambio de vehículo principal.

- **NewVehicleAdapter**: adaptador para la lista que muestra los tipos de vehículos seleccionables al crear un nuevo vehículo.

5.2.2.2.6 Utils

- **VehicleTypes**: clase que gestiona los tipos de vehículos. Los métodos de esta clase devuelven colecciones con la información de cada tipo de vehículo según el campo que se requiera: layout, número de ruedas, imagen.
- **VehicleType**: enumerador con la información de cada tipo de vehículo.
- **DataParser**: clase que analiza la cadena de bytes obtenida en el resultado del escaneo y devuelve los valores de temperatura, presión, etc.
- **UnitConverter**: clase auxiliar para conversiones de presión y temperatura.
- **BitmapFromAssetsProvider**: clase que implementa un método que devuelve una imagen localizada en la carpeta “assets” dado el nombre.

5.2.3 Diseño de la Base de Datos

5.2.3.1 Descripción del SGBD Usado

El SGBD empleado en esta aplicación es SQLite a través de la biblioteca de persistencia Room. En la fase de análisis del sistema, en el apartado de requisitos, se determina la necesidad de utilizar una base de datos local para guardar los vehículos creados por el usuario.

Este SGBD se escoge siguiendo las recomendaciones de desarrollo de Android, sobre como guardar contenido en una base de datos local.

5.2.3.2 Integración del SGBD en Nuestro Sistema

EL SGBD se encuentra implementado en el paquete "database" que contiene las siguientes clases:

- AppDatabase: clase abstracta que extiende de RoomDatabase y sirve como punto de acceso para la conexión subyacente a los datos persistentes.
- Vehicle: es la entidad que define la única tabla de la base de datos.
- VehicleDao: es la interfaz que define los métodos utilizados para acceder a la base de datos.
- VehicleRepository: es el repositorio con el que se administran los datos.
- Converters: clase que convierte los tipos primitivos en tipos complejos y viceversa. Esto se debe a que Room no acepta tipos complejos como Arraylist.

Previamente a la implementación de estas clases es necesario instalar las dependencias de Room.

5.2.3.3 Diagrama E-R

Esta base de datos solo contiene una tabla, por lo que el diagrama entidad-relación no existe como tal. Además, Room no permite referencias entre clases de entidad. Es necesario crear una clase intermedia que haga la relación. Este fue el motivo por el que se decidió crear

una sola tabla con unos de sus campos de tipo Array en vez de dos tablas relacionadas. A continuación, se muestra la tabla Vehículo:

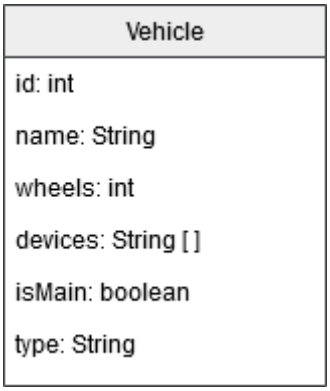


Figura 5.14:Tabla Vehicle de la base de datos

5.2.4 Diseño de la Interfaz

5.2.4.1 Principal

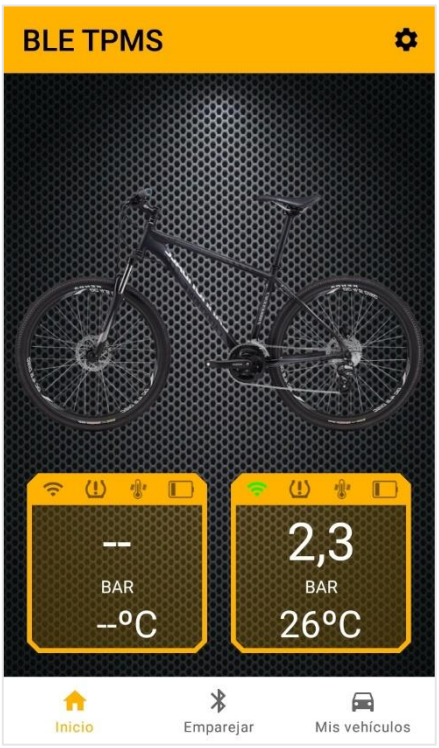


Figura 5.15: Diseño de la pantalla Principal

5.2.4.2 Emparejar

Figura 5.16: Diseño de la pantalla Emparejar

Figura 5.17: Diseño de la pantalla Emparejar [opciones]

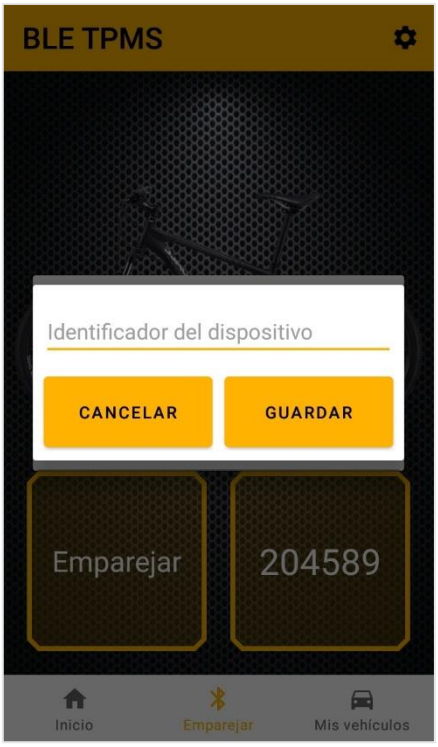


Figura 5.18: Diseño de la pantalla Emparejar [manual]

Figura 5.19: Diseño de la pantalla Emparejar [automático]

Figura 5.20: Diseño de la pantalla Emparejar [automático II]

5.2.4.3 Vehículos

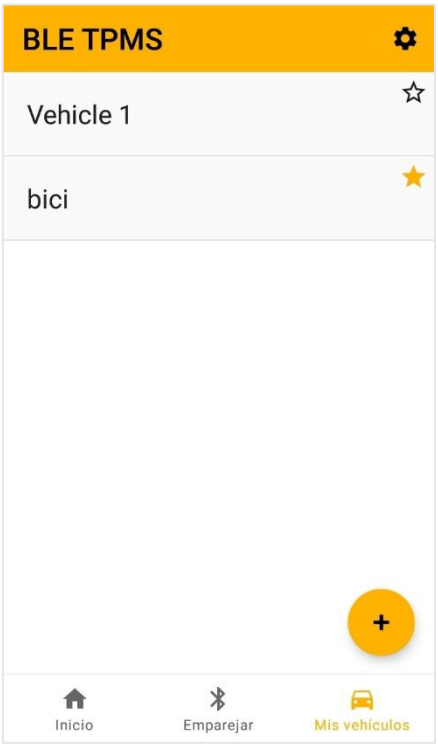


Figura 5.21: Diseño de la pantalla Vehículos

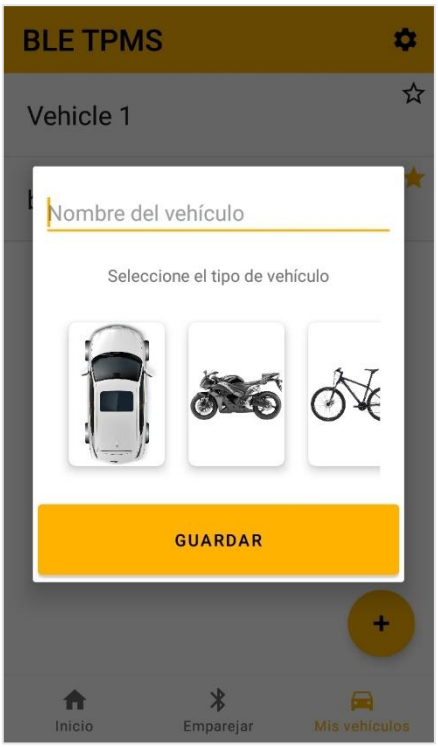


Figura 5.22: Diseño de la pantalla Vehículos [Diálogo crear vehículo]

5.2.4.4 Ajustes

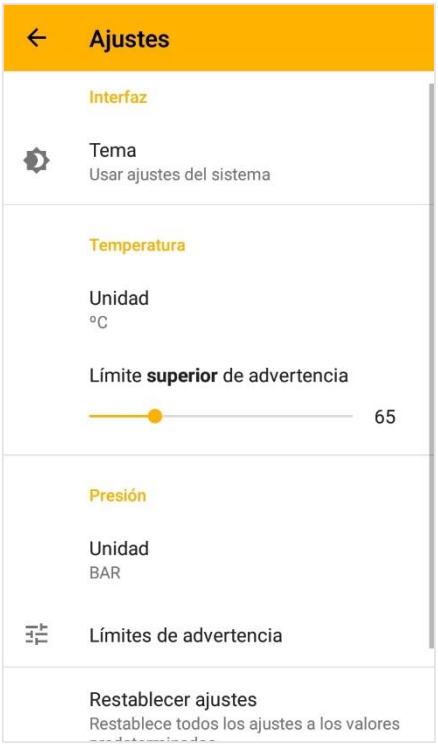


Figura 5.23: Diseño de la pantalla Ajustes

5.2.5 Especificación Técnica del Plan de Pruebas

5.2.5.1 Pruebas Unitarias

5.2.5.1.1 BluetoothServiceTest

Nombre de la prueba
startScaning()
Descripción
Comprueba que el inicio de un escaneo Bluetooth se realiza correctamente para las versiones de Android desde 5.0 hasta 10.0.

Tabla 33: Descripción prueba unitaria: startScaning()

Nombre de la prueba
stopScaning()
Descripción
Comprueba que tras el inicio de un escaneo Bluetooth se detiene correctamente para las versiones de Android desde 5.0 hasta 10.0.

Tabla 34: Descripción prueba unitaria: stopScaning()

Nombre de la prueba
stopScanning_neverStarted()
Descripción
Comprueba que al detener un escaneo Bluetooth que no se ha iniciado la respuesta es la esperada para las versiones de Android desde 5.0 hasta 10.0.

Tabla 35: Descripción prueba unitaria: stopScanning_neverStarted()

5.2.5.1.2 DataParserTest

Nombre de la prueba
retManData_correctInput_returnOK()
Descripción
Comprueba que el método retManData() devuelve un String de longitud 36 al pasar por parámetro una cadena bytes con el formato correcto.

Tabla 36: Descripción prueba unitaria: retManData_correctInput_returnOK()

Nombre de la prueba
retManData_nullInput_returnNull()
Descripción
Comprueba que el método retManData() devuelve null al pasar por parámetro null.

Tabla 37: Descripción prueba unitaria: retManData_nullInput_returnNull()

Nombre de la prueba
getPressureKPA()
Descripción
Comprueba que el método getPressureKPA devuelve un valor de tipo “double”.

Tabla 38: Descripción prueba unitaria: getPressureKPA()

Nombre de la prueba
getPressureBAR()
Descripción
Comprueba que el método getPressureBAR devuelve un valor de tipo “double”.

Tabla 39: Descripción prueba unitaria: getPressureBAR()

Nombre de la prueba
getPressurePSI()
Descripción
Comprueba que el método getPressurePSI devuelve un valor de tipo “double”.

Tabla 40: Descripción prueba unitaria: getPressurePSI()

Nombre de la prueba
getTemperatureCelsius()
Descripción
Comprueba que el método getTemperatureCelsius devuelve un valor de tipo “double”.

Tabla 41: Descripción prueba unitaria: getTemperatureCelsius()

Nombre de la prueba
getTemperatureFahrenheit()
Descripción
Comprueba que el método getTemperatureFahrenheit devuelve un valor de tipo “double”.

Tabla 42: Descripción prueba unitaria: getTemperatureFahrenheit()

Nombre de la prueba
getSensorNumber()
Descripción
Comprueba que el método getSensorNumber devuelve un valor de tipo “Integer” y comprendido entre 1, 2, 3 y 4.

Tabla 43: Descripción prueba unitaria: getSensorNumber()

Nombre de la prueba
getSensorAddress()
Descripción
Comprueba que el método getSensorAddress devuelve un “String” de longitud 17

Tabla 44: Descripción prueba unitaria: getSensorAddress()

Nombre de la prueba
getBatteryPercentage()
Descripción
Comprueba que el método getBatteryPercentage devuelve un valor de tipo “Integer” y comprendido entre 0, 25, 75 y 100.

Tabla 45: Descripción prueba unitaria: getBatteryPercentage()

Nombre de la prueba
getAlarmFlag()
Descripción
Comprueba que el método getAlarmFlag devuelve un valor de tipo “Integer” y comprendido entre 0 y 1.

Tabla 46: Descripción prueba unitaria: getAlarmFlag()

5.2.5.1.3 UnitConverterTest

Nombre de la prueba
fahrenheitToCelsius()
Descripción
Comprueba que el método fahrenheitToCelsius realiza correctamente la conversión de temperatura de Fahrenheit a Celsius.

Tabla 47: Descripción prueba unitaria: fahrenheitToCelsius()

Nombre de la prueba
celsiusToFahrenheit()
Descripción
Comprueba que el método celsiusToFahrenheit realiza correctamente la conversión de temperatura de Celsius a Fahrenheit.

Tabla 48: Descripción prueba unitaria: celsiusToFahrenheit ()

Nombre de la prueba
pressureConversions_floatValueInput()
Descripción
Comprueba que el método doPressureConversion realiza correctamente la conversión de presión de BAR a KPA y PSI dado un valor de entrada de tipo *float*.

Tabla 49: Descripción prueba unitaria: pressureConversions_floatValueInput()

Nombre de la prueba
pressureConversions_integerValueInput()
Descripción
Comprueba que el método doPressureConversion realiza correctamente la conversión de presión de BAR a KPA y PSI dado un valor de entrada de tipo *integer*.

Tabla 50: Descripción prueba unitaria: pressureConversions_integerValueInput()

Nombre de la prueba
pressureConversions_wrongUnitInput_returnZero()
Descripción
Comprueba que el método doPressureConversion devuelve 0 dada una unidad de entrada que no sea BAR, KPA o PSI.

Tabla 51: Descripción prueba unitaria: pressureConversions_wrongUnitInput_returnZero()

5.2.5.1.4 VehicleTypesTest

Nombre de la prueba
getAllWheels_returnsOK()
Descripción
Comprueba que el método getAllWheels_returnsOK devuelve un objeto de tipo “Map” que contiene los valores esperados.

Tabla 52: Descripción prueba unitaria: getAllWheels_returnsOK()

Nombre de la prueba
getAllLayouts_returnsOK()
Descripción
Comprueba que el método getAllLayouts_returnsOK devuelve un objeto de tipo “Map” que contiene los valores esperados.

Tabla 53: Descripción prueba unitaria: getAllLayouts_returnsOK()

Nombre de la prueba
getImagesName_returnsOK()
Descripción
Comprueba que el método getImagesName_returnsOK devuelve un objeto de tipo “Map” que contiene los valores esperados.

Tabla 54: Descripción prueba unitaria: getImagesName_returnsOK()

Nombre de la prueba
getAllVehicleTypeInfo_existingName_returnsOK()
Descripción
Comprueba que todas las colecciones de datos devuelven los valores esperados pasando por parámetro una clave existente.

Tabla 55: Descripción prueba unitaria: getAllVehicleTypeInfo_existingName_returnsOK()

Nombre de la prueba
getAllVehicleTypeInfo_noExistingName_returnsNull()
Descripción
Comprueba que todas las colecciones de datos devuelven null pasando por parámetro una clave no existente.

Tabla 56: Descripción prueba unitaria: getAllVehicleTypeInfo_noExistingName_returnsNull()

5.2.5.2 Pruebas de Integración y del Sistema

5.2.5.2.1 DatabaseTest

Nombre de la prueba
insertAndReadList()
Descripción
Añadir un coche de 4 ruedas con el nombre “test vehicle”.
Resultado esperado
La lista de vehículos se actualiza y muestra el nuevo vehículo.

Tabla 57: Descripción prueba de integración: insertAndReadList()

Nombre de la prueba
insertDeleteAndReadList()
Descripción
Añadir un coche de 4 ruedas con el nombre “test vehicle” y eliminarlo.
Resultado esperado
La lista muestra el vehículo y tras eliminarlo queda vacía.

Tabla 58: Descripción prueba de integración: insertDeleteAndReadList()

Nombre de la prueba
insertUpdateAndReadList()
Descripción
Añadir un coche de 4 ruedas con el nombre “test vehicle” y cambiarlo a moto de 2 ruedas.
Resultado esperado
La lista muestra el vehículo y en la base de datos queda registrado como una moto de 2 ruedas.

Tabla 59: Descripción prueba de integración: insertUpdateAndReadList()

Nombre de la prueba
insertUpdateAllAndReadList()
Descripción
Añadir 2 coches de 4 ruedas de nombre “test vehicle” y “test vehicle 2” y cambiarlos a motos de dos ruedas
Resultado esperado
La lista muestra los vehículos y en la base de datos quedan registrados como motos de 2 ruedas.

Tabla 60: Descripción prueba de integración: insertUpdateAllAndReadList()

Nombre de la prueba
insertAndGetMainVehicle()
Descripción
Añadir 2 coches de 4 ruedas de nombre “test vehicle” y “test vehicle 2”. El primero con el campo “isMain” a true y el segundo a false.
Resultado esperado
Obtener el vehículo principal de la base de datos y que el nombre sea “test vehicle”.

Tabla 61: Descripción prueba de integración: insertAndGetMainVehicle()

5.2.5.2.2 HomeFragmentTest

Nombre de la prueba
checkBaseElements()
Descripción
Añadir un coche de 4 ruedas con el nombre “test vehicle” y acceder a la ventana “Inicio”.
Resultado esperado
Se muestran los elementos principales: Fondo Imagen del vehículo 4 VehicleCard El texto “BAR” y “ºC”

Tabla 62: Descripción prueba de integración: checkBaseElements() [Home]

Nombre de la prueba
checkWaitingForDataState()
Descripción
Añadir un coche de 4 ruedas con el nombre “test vehicle” y acceder a la ventana “Emparejar” para añadir un sensor con el identificador “000000”. Después volver a la ventana “Inicio”.
Resultado esperado
En la VehicleCard correspondiente aparece un indicador circular de progreso.

Tabla 63: Descripción prueba de integración: checkWaitingForDataState()

5.2.5.2.3 PairFragmentTest

Nombre de la prueba
checkBaseElements()
Descripción
Añadir un coche de 4 ruedas con el nombre “test vehicle” y acceder a la ventana “Emparejar”.
Resultado esperado
Se muestran los elementos principales: Fondo Imagen del vehículo 4 VehicleCard Texto “Emparejar”

Tabla 64: Descripción prueba de integración: checkBaseElements() [Pair]

Nombre de la prueba
checkManualBind()
Descripción
Añadir un coche de 4 ruedas con el nombre “test vehicle” y acceder a la ventana “Emparejar” para añadir un sensor con el identificador “000000”.
Resultado esperado
Se muestra el identificador en la MaterialCardView correspondiente.

Tabla 65: Descripción prueba de integración: checkManualBind()

Nombre de la prueba
checkAutoBind()
Descripción
Añadir un coche de 4 ruedas con el nombre “test vehicle” y acceder a la ventana “Emparejar”. Clicar en una de las MaterialCarView y seleccionar “Automático”
Resultado esperado
Se muestra la ventana de diálogo con las instrucciones de emparejamiento automático.

Tabla 66: Descripción prueba de integración: checkAutoBind()

Nombre de la prueba
checkEditAndUnbind()
Descripción
Añadir un coche de 4 ruedas con el nombre “test vehicle” y acceder a la ventana “Emparejar” para añadir un sensor con el identificador “000000”. Después editar el identificador y cambiarlo a “000001”. Por último, desvincular ese sensor.
Resultado esperado
No se muestra ningún identificador.

Tabla 67: Descripción prueba de integración: checkEditAndUnbind()

5.2.5.2.4 VehicleFragmentTest

Nombre de la prueba
checkBaseElements()
Descripción
Acceder a la ventana “Mis vehículos” y añadir un coche de 4 ruedas con el nombre “test vehicle”. Después clicar en el botón para añadir vehículo.
Resultado esperado
Se muestra la lista con 1 vehículo y la ventana de diálogo para añadir vehículos que contiene una lista con los tipos de vehículos seleccionables.

Tabla 68: Descripción prueba de integración: checkBaseElements() [Vehicle]

Nombre de la prueba
addVehicles()
Descripción
Acceder a la ventana “Mis vehículos” y añadir un coche de 4 ruedas con el nombre “test vehicle”, un coche de 4 ruedas con el nombre “Four Wheels” y una moto con el nombre “Motorbike”.
Resultado esperado
Se muestran los 3 vehículos en la lista.

Tabla 69: Descripción prueba de integración: addVehicles()

Nombre de la prueba
addVehicle_wrongInput_showToast()
Descripción
Acceder a la ventana “Mis vehículos” y añadir un vehículo sin ingresar nombre ni seleccionar el tipo.
Resultado esperado
Se muestran los mensajes de error correspondientes.

Tabla 70: Descripción prueba de integración: addVehicle_wrongInput_showToast()

Nombre de la prueba
deleteOneVehicleAndMultipleVehicles()
Descripción
Acceder a la ventana “Mis vehículos” y añadir 2 coches de 4 ruedas con el nombre “test vehicle” y “First delete”. Eliminar el vehiculo “First delete”. Añadir 2 coches de 4 ruedas con el nombre “Second delete 1” y “Second delete 2”. Eliminar los dos vehículos.
Resultado esperado
La lista muestra únicamente el vehículo con nombre “test vehicle”.

Tabla 71: Descripción prueba de integración: deleteOneVehicleAndMultipleVehicles()

Nombre de la prueba
deleteOneVehicleAndMultipleVehicles_mainSelected_showErrorMessage()
Descripción
Acceder a la ventana “Mis vehículos” y añadir 2 coches de 4 ruedas con el nombre “test vehicle” y “test vehicle 2”. Eliminar el vehículo de nombre “test vehicle 2”
Resultado esperado
Se muestra el mensaje de error correspondiente a tratar de eliminar un vehículo principal.

Tabla 72: Descripción prueba de integración: deleteOneVehicleAndMultipleVehicles_mainSelected_showErrorMessage()

Nombre de la prueba
editVehicleName()
Descripción
Acceder a la ventana “Mis vehículos” y añadir un coche de 4 ruedas con el nombre “test vehicle”. Cambiar el nombre a “NEW test vehicle”.
Resultado esperado
La lista muestra un vehículo con nombre “NEW test vehicle”.

Tabla 73: Descripción prueba de integración: editVehicleName()

Nombre de la prueba
editVehicleName_wrongInput_showToast()
Descripción
Acceder a la ventana “Mis vehículos” y añadir un coche de 4 ruedas con el nombre “test vehicle”. Editar el vehículo y eliminar el nombre.
Resultado esperado
Se muestra el mensaje de error correspondiente.

Tabla 74: Descripción prueba de integración: editVehicleName_wrongInput_showToast()

Nombre de la prueba
changeMainVehicle_itemOptionAndMenuOption()
Descripción
Acceder a la ventana “Mis vehículos” y añadir 2 coches de 4 ruedas con el nombre “test vehicle” y “New main”. Cambiar el vehículo principal de “New main” a “test vehicle”.
Resultado esperado
La lista muestra los vehículos e indica que el principal es “test vehicle”.

Tabla 75: Descripción prueba de integración: changeMainVehicle_itemOptionAndMenuOption()

5.2.5.2.5 SettingsActivityTest

Nombre de la prueba
checkThemePreference()
Descripción
Acceder a “Ajustes” y cambiar el tema de la aplicación de “Tema del sistema” a “Claro” y de “Claro” a “Oscuro”
Resultado esperado
La aplicación cambia de tema.

Tabla 76: Descripción prueba de integración: checkThemePreference()

Nombre de la prueba
checkTemperaturePreferences()
Descripción
Acceder a “Ajustes” y cambiar las unidades de temperatura de Celsius a Fahrenheit.
Resultado esperado
El valor de la barra de límite de advertencia de temperatura cambia de 65 a 149.

Tabla 77: Descripción prueba de integración: checkTemperaturePreferences()

Nombre de la prueba
checkPressurePreferences()
Descripción
Acceder a “Ajustes” y cambiar las unidades de presión de BAR a KPA y de KPA a PSI.
Resultado esperado
Los valores de la barra de límite de advertencia de presión cambian de [2.0, 3.0] a [200.0, 300.0] y a [29.0, 43.5].

Tabla 78: Descripción prueba de integración: checkPressurePreferences()

Nombre de la prueba
checkAboutPreference()
Descripción
Acceder a “Ajustes” y abrir la ventana de diálogo de “Acerca de”.
Resultado esperado
Se muestra la ventana de diálogo “Acerca de”.

Tabla 79: Descripción prueba de integración: checkAboutPreference()

Nombre de la prueba
checkResetSettingsPreference()
Descripción
Acceder a “Ajustes” y cambiar las unidades de medida de temperatura y presión de Celsius a Fahrenheit y de BAR a KPA. Clicar en la opción “Restablecer ajustes”.
Resultado esperado
Las barras de límites de advertencia muestran los valores por defecto.

Tabla 80: Descripción prueba de integración: checkResetSettingsPreference()

5.2.5.3 Pruebas de Usabilidad

Las pruebas de usabilidad tendrán como objetivo determinar si el diseño de la aplicación es sencillo e intuitivo. Para ello, 5 usuarios de edades comprendidas entre los 18 y 75 años probarán la aplicación y darán una opinión a través de un cuestionario con el formato que se muestra a continuación.

Preguntas	Nota
Funcionalidad	
¿La información mostrada en la pantalla "Inicio" le resulta comprensible?	
¿Le resulta sencillo cambiar las unidades de medida de temperatura y presión?	
¿Le resulta sencillo cambiar los límites de advertencia de temperatura y presión?	
¿Le resulta sencillo emparejar los sensores?	
¿Le resulta sencillo desemparejar los sensores?	
¿Le resulta sencillo añadir un nuevo vehículo?	
¿Le resulta sencillo editar el nombre de un vehículo?	
¿Le resulta sencillo eliminar uno o más vehículos?	
¿Le resulta sencillo cambiar el vehículo principal?	
Interfaz	
¿El tipo y tamaño de letra es el adecuado?	
¿Los iconos e imágenes son las adecuadas?	
¿Los colores empleados son adecuados?	
¿El diseño de las pantallas es claro y atractivo?	
General	
¿Le resulta fácil de usar?	
¿Cree que la aplicación ofrece información suficiente?	
¿Recomendaría esta aplicación?	

Tabla 81: Cuestionario para pruebas de usabilidad

5.3 IMPLEMENTACIÓN DEL SISTEMA

5.3.1 Lenguajes de Programación

5.3.1.1 Java

Java [11] es un lenguaje de programación orientada a objetos diseñado por la empresa Sun Microsystems, la cual fue adquirida posteriormente por Oracle. Salió al mercado en 1995 y actualmente es el segundo lenguaje más usado en el mundo. Permite la ejecución de un mismo programa en diferentes sistemas operativos, tiene soporte para trabajo en red y está diseñado para ejecutar código en sistemas remotos de forma segura.

Figura 5.24: Logo de Java

5.3.1.2 SQL

SQL (Structured Query Language o Lenguaje de Consulta Estructurada) [12] es un lenguaje de dominio específico diseñado para almacenar, manipular y recuperar datos en bases de datos relacionales. Fue diseñado por IBM y comercializado por Oracle en 1979.

Figura 5.25: Logo de SQL

5.3.1.3 XML

XML (eXtensible Markup Language o Lenguaje de Marcado Extensible) [13] es un metalenguaje que permite definir lenguajes de marcado para almacenar datos de forma legible, jerarquizada y estructurada.

Figura 5.26: Logo de XML

5.3.2 Herramientas y Programas Usados para el Desarrollo

5.3.2.1 Android Studio

Android Studio [14] es un entorno de desarrollo integrado (IDE) para aplicaciones del sistema operativo Android. Es un software gratuito y multiplataforma desarrollado por Google. Admite distintos lenguajes de programación como pueden ser Java, C++ o Kotlin. Ofrece herramientas tanto para generar la lógica de la aplicación como para diseñar su interfaz. Está basado en IDE de JetBrains, IntelliJ IDEA e incluye un compilador basado en Gradle.

Figura 5.27: Logo de Android Studio

5.3.2.2 SQLite

SQLite [15] es un sistema de gestión de bases de datos relacional de dominio público cuya principal característica es que se integra con la aplicación, en lugar de ser un proceso

independiente como los SGBD cliente-servidor. Tiene un tamaño muy reducido (aprox. 275Kb) por lo que su uso es muy conveniente en sistemas integrados como Android o iOS. Algunos ejemplos de aplicaciones muy conocidas que utilizan SQLite son: Mozilla Firefox, Adobe Photoshop Elements o Skype.

Figura 5.28: Logo de SQLite

5.3.2.3 Room

Room [16] es una librería que proporciona una capa de abstracción sobre SQLite para el manejo de bases de datos en Android. Funciona con una arquitectura cuyas clases se marcan con anotaciones y verifica las consultas en tiempo de compilación. Esto le permite acceder a todas las funcionalidades de SQLite mientras mantiene la seguridad de tipos proporcionada por los constructores de consultas de SQL de Java.

5.3.2.4 Microsoft Word

Microsoft Word [17] es un procesador de textos creado por Microsoft en 1983 que forma parte del paquete ofimático Microsoft Office. Ofrece multitud de herramientas para la redacción de documentos de texto.

Figura 5.29: Logo de Microsoft Word

5.3.2.5 Microsoft Project

Microsoft Project [18] es un software de administración de proyectos desarrollado por Microsoft. Ofrece multitud de herramientas para planificar y evaluar tareas, gestionar recursos, controlar horarios y calcular presupuestos.

Figura 5.30: Logo de Microsoft Project

5.3.2.6 Draw.io

Draw.io [19] es una herramienta online gratuita para la elaboración de diagramas y gráficas. Ofrece una amplia variedad de figuras, así como una interfaz sencilla que facilita la unión de los elementos gráficos empleados.

Figura 5.31: Logo de Draw.io

5.3.3 Creación del Sistema

5.3.3.1 Problemas Encontrados

5.3.3.1.1 *Layouts*[4] dinámicos y adaptables

Uno de los mayores problemas a la hora de desarrollar la interfaz de los vehículos fue la disposición de las ruedas en la pantalla. Como se muestra en los diseños preliminares el objetivo era mostrar un número determinado de elementos según el número de ruedas que tuviera el vehículo seleccionado. Al haber tantas opciones (ocho tipos de vehículos) el número de layouts que habría que implementar era muy grande y obligaba a diseñar un layout específico para cada vehículo.

Para reducir el número de layouts la solución que se planteó fue agrupar por número de ruedas, es decir, en caso de tener tres tipos de vehículos de 4 ruedas, en lugar de cargar un layout para cada uno, se diseñó un layout común a todos los vehículos de 4 ruedas. Esto redujo considerablemente el número de ficheros xml y facilita el añadir nuevos tipos de vehículos en el futuro.

Para implementar esta característica hubo que diseñar la manera de cargar el layout correspondiente en función del número de ruedas, es decir, que los layouts fueran dinámicos. Esto se consiguió implementando una clase enumeradora en la que se le asigna a cada tipo de vehículo el nombre del fichero xml que le corresponde. De esta manera, al iniciar la vista, la aplicación consulta en la base de datos el tipo de vehículo seleccionado y en la clase enumeradora el nombre del fichero correspondiente.

Otra de las razones que propiciaron este diseño fue el tamaño de los elementos que muestran la información de cada rueda. Para conseguir un diseño adaptativo había que ajustarlo según el número de ruedas, es decir, aprovechar la pantalla cuando solo se muestran dos o tres ruedas (elementos grandes) y repartir la pantalla cuando se muestran ocho ruedas (elementos pequeños).

[4] Diseño o disposición de los elementos en pantalla. En desarrollo de interfaces hace referencia a archivos XML que contienen un esquema de distribución de los elementos que se muestran.

5.3.3.1.2 Base de datos

El SGBD se eligió siguiendo las recomendaciones de desarrollo de Android. Una vez implementada, empezaron a surgir complicaciones ya que no se conocía realmente la biblioteca Room y sus limitaciones.

El primer problema surgió al querer relacionar las únicas dos tablas que en un principio iban a conformar la base datos. Room no permite las referencias entre clases de entidades y te obliga a crear una clase intermedia. Se plantearon tres soluciones:

1. Cambiar de SGBD: en vez de Room utilizar SQLlite directamente, la cual si permite hacer relaciones. Esta opción se descartó por dos motivos: primero por no desechar todo el trabajo realizado hasta el momento (estudio e implementación de Room) y segundo porque Android recomendaba Room encarecidamente por una serie de ventajas. Mas adelante se vio que estas ventajas no marcarían la diferencia en bases de datos tan pequeñas como la que se necesitaba.
2. Establecer las relaciones de la manera que indicaba Room. Esta opción se descartó debido a la complejidad que conllevaba implementarla teniendo en cuenta que la base de datos que se necesitaba era muy simple.
3. Crear una única tabla con un campo de tipo Array. Se escogió esta opción porque aparentemente era la más sencilla.

Una vez implementada la tabla surgió el segundo problema. Room no permite tipos complejos como Array, por lo que al compilar daba un error. Para solucionarlo se siguieron las instrucciones de la guía de Room en las que indican que se puede hacer una conversión de tipo complejo a tipo primitivo y viceversa mediante métodos con la anotación "TypeConverter". Se implementó la clase "Converters" con dos funciones que convierten de tipo String a Array y viceversa.

Otro problema al usar Room fue la gestión de las operaciones con la base de datos. Estas no se pueden ejecutar en el hilo principal por lo que te obliga a usar el contenedor LiveData o a ejecutar un hilo en segundo plano en cada operación de acceso.

5.4 DESARROLLO DE LAS PRUEBAS

5.4.1 Pruebas Unitarias

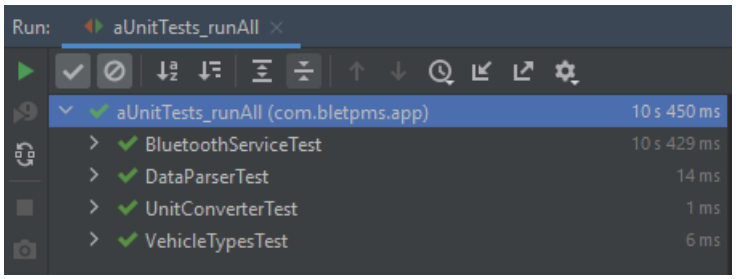


Figura 5.32: Ejecución de todas las pruebas unitarias

5.4.1.1 BluetoothServiceTest

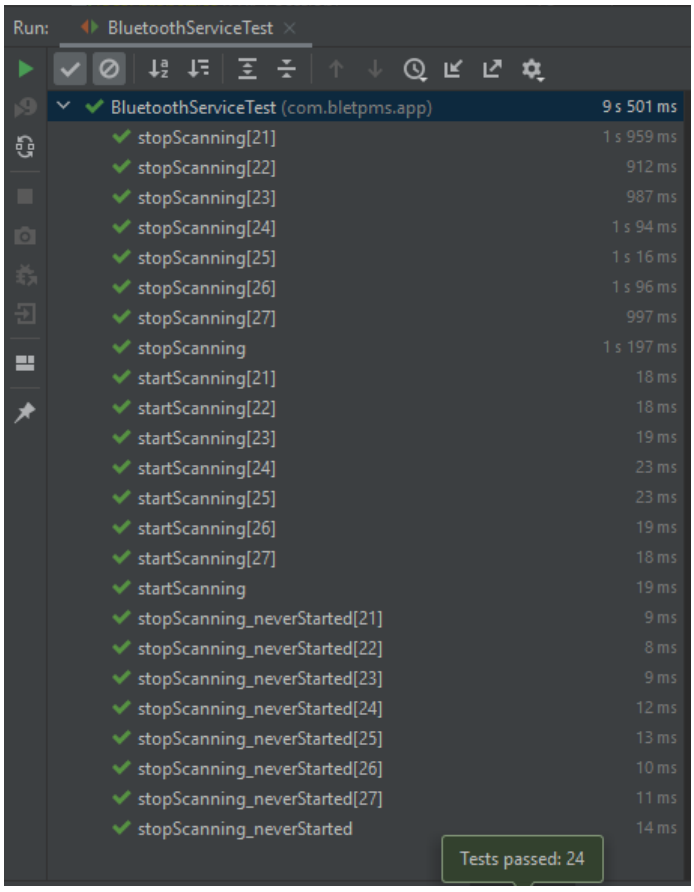


Figura 5.33: Ejecución de BluetoothServiceTest

5.4.1.2 DataParserTest

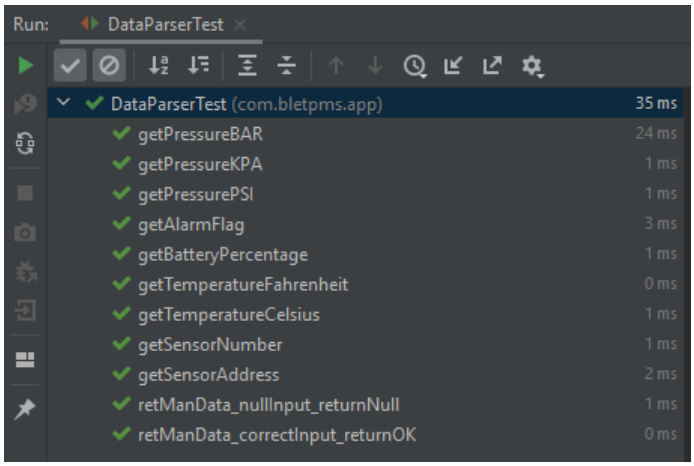


Figura 5.34: Ejecución de DataParserTest

5.4.1.3 UnitConverterTest

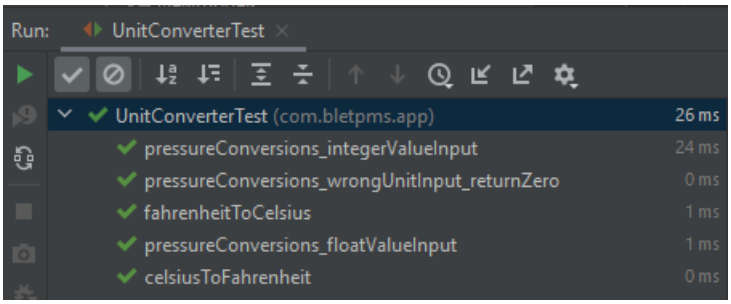


Figura 5.35: Ejecución de UnitConverterTest

5.4.1.4 VehicleTypesTest

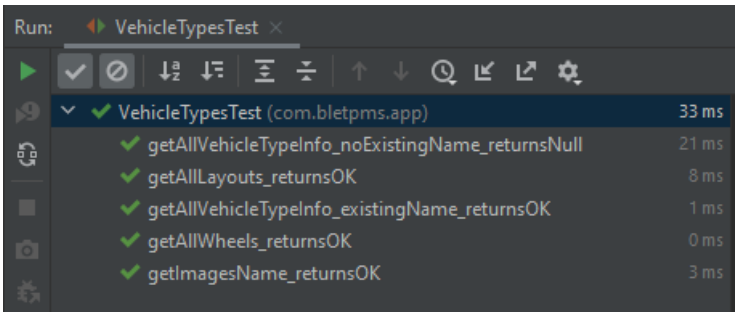


Figura 5.36: Ejecución de VehicleTypesTest

5.4.2 Pruebas de Integración y del Sistema

En este apartado se muestran los resultados de la ejecución de las pruebas de integración y del sistema descritas en el apartado 5.2.5.2. Como se puede ver a continuación, los resultados obtenidos se corresponden con los esperados.

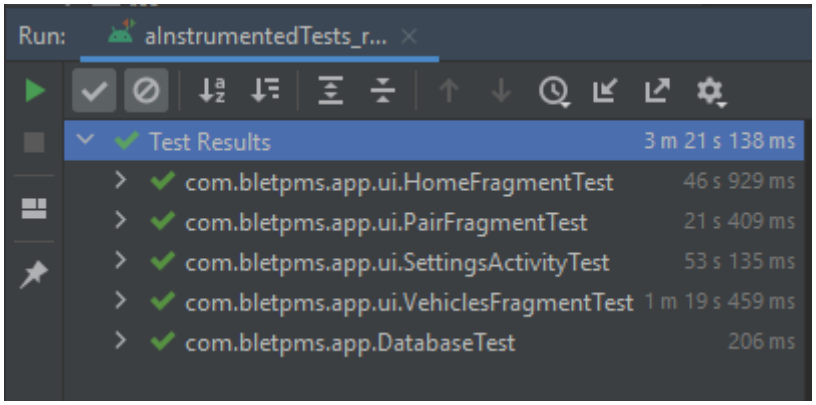


Figura 5.37: Ejecución de todos los test de Integración y Sistema

5.4.2.1 DatabaseTest

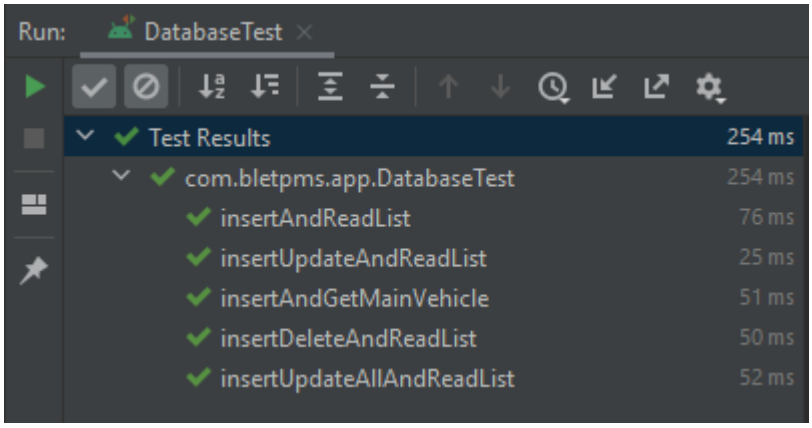


Figura 5.38: Ejecución de DatabaseTest

5.4.2.2 HomeFragmentTest

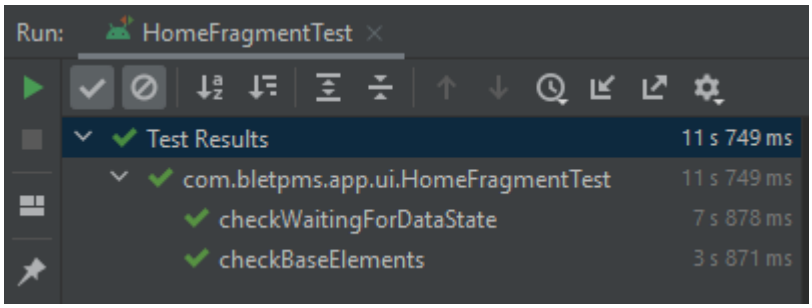


Figura 5.39: Ejecución de HomeFragmentTest

5.4.2.3 PairFragmentTest

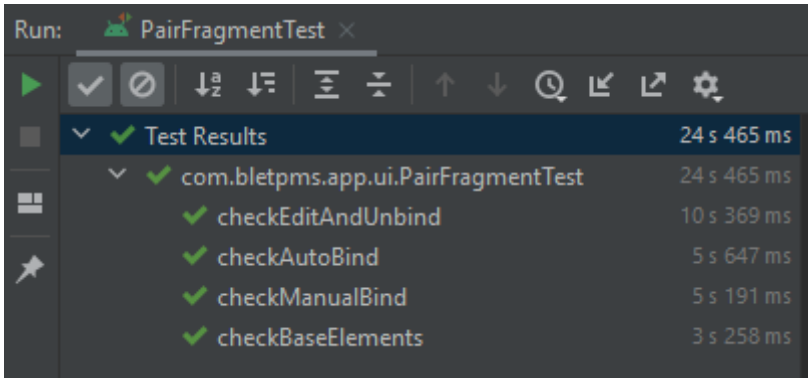


Figura 5.40: Ejecución de PairFragmentTest

5.4.2.4 SettingsActivityTest

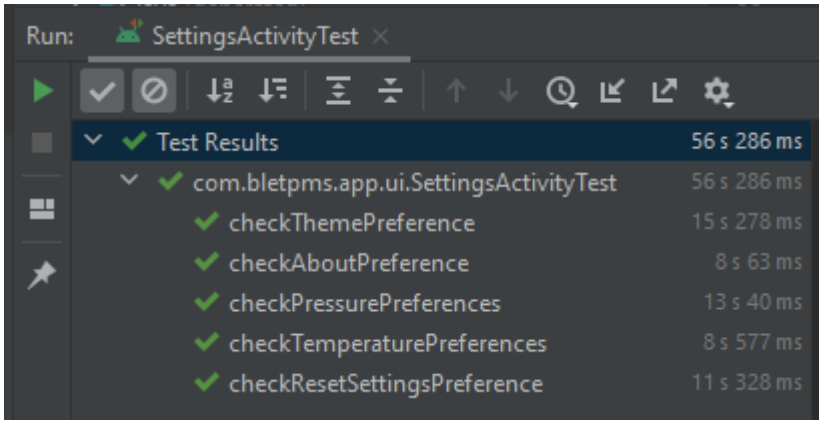


Figura 5.41: Ejecución de SettingsActivityTest

5.4.2.5 VehiclesFragmentTest

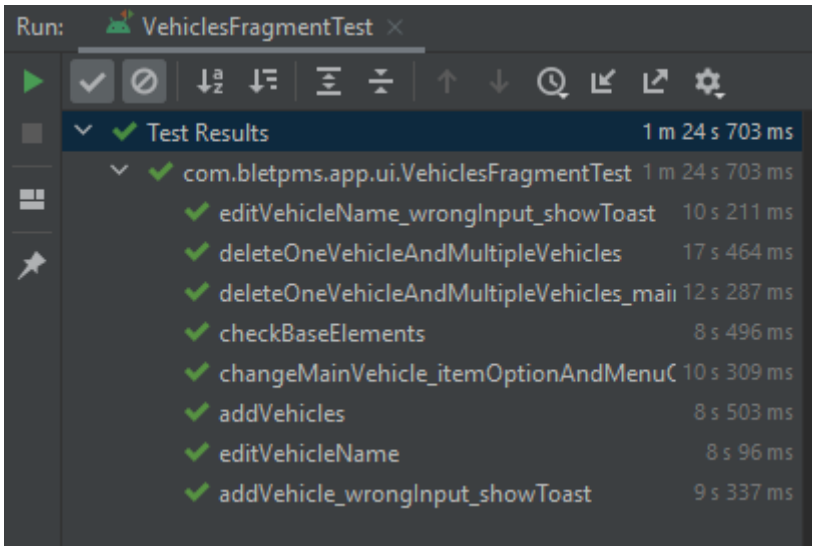


Figura 5.42: Ejecución de VehiclesFragmentTest

5.4.3 Pruebas de Usabilidad y Accesibilidad

Para realizar las pruebas se han escogido 5 usuarios con los siguientes perfiles (edad y nivel de informática):

- Usuario 1: 18 años. Avanzado.
- Usuario 2: 25 años. Avanzado.
- Usuario 3: 31 años. Experto.
- Usuario 4: 58 años. Básico.
- Usuario 5: 65 años. Básico.

Se les han encomendado las siguientes tareas con el objetivo de que prueben todas las funcionalidades de la aplicación.

1. Crear 3 vehículos con nombres a su elección y en el orden especificado: 2 coches de 4 ruedas, 1 moto de 2 ruedas y 2 bicicletas.
2. Eliminar la moto.
3. Eliminar los 2 coches a la vez.
4. Cambiar el nombre de las bicicletas a “Bici prueba 1” y “Bici prueba 2”.
5. Seleccionar “Bici prueba 1” como vehículo principal.
6. Emparejar dos sensores manualmente con los identificadores “000000” y “000001” a “Bici prueba 1”.
7. Desemparejar los dos sensores.
8. Seleccionar “Bici prueba 2” como vehículo principal.
9. Emparejar dos sensores automáticamente a “Bici prueba 2” dados los sensores reales.
10. Mostrar los datos enviados por los sensores en tiempo real.
11. Cambiar las unidades de medida de temperatura y presión.
12. Modificar los límites de advertencia de temperatura y presión.

5.4.3.1 Resultados del cuestionario

Una vez completadas, se les ha trasladado el formulario descrito en el apartado 5.2.5.3. A continuación, se muestran los resultados ponderados.

Preguntas	Nota
Funcionalidad	
¿La información mostrada en la pantalla "Inicio" le resulta comprensible?	4,6
¿Le resulta sencillo cambiar las unidades de medida de temperatura y presión?	5
¿Le resulta sencillo cambiar los límites de advertencia de temperatura y presión?	5
¿Le resulta sencillo emparejar los sensores?	3,4
¿Le resulta sencillo desemparejar los sensores?	5
¿Le resulta sencillo añadir un nuevo vehículo?	5
¿Le resulta sencillo editar el nombre de un vehículo?	3
¿Le resulta sencillo eliminar uno o más vehículos?	3
¿Le resulta sencillo cambiar el vehículo principal?	4,6
Interfaz	
¿El tipo y tamaño de letra es el adecuado?	5
¿Los iconos e imágenes son las adecuadas?	5
¿Los colores empleados son adecuados?	5
¿El diseño de las pantallas es claro y atractivo?	4,6
General	
¿Le resulta fácil de usar?	4,6
¿Cree que la aplicación ofrece información suficiente?	4
¿Recomendaría esta aplicación?	5

Tabla 82: Resultado cuestionario de pruebas de usabilidad

5.4.3.2 Conclusiones

En general la aplicación recibe una buena nota en todos los apartados menos en 3 de ellos que tienen una nota por debajo de 4. A continuación se resumen los motivos y las posibles soluciones.

Emparejar sensores

Los usuarios notifican que el proceso de emparejamiento automático a veces falla o se necesitan realizar dos intentos o más.

En este caso el problema no es la aplicación, son los sensores. Su sistema de envío de balizas tiene un intervalo de espera bastante largo cuando no nota cambio de presión o temperatura. Teniendo en cuenta que las pruebas se realizaron instalando y desinstalando los sensores en una bicicleta sin realizar desplazamientos, esta sería la razón por la que el

proceso de emparejamiento a veces falla. Simplemente los sensores tardan en enviar los datos.

Eliminar y editar vehículos

Los usuarios de mayor edad y con nivel de informática básico encontraron dificultades a la hora de eliminar o editar vehículos. Esto se debe a que estas dos opciones se encuentran en un menú contextual que aparece en la barra superior cuando se realiza un clic prolongado sobre uno de los vehículos.

El resto de los usuarios encontraron rápidamente el menú ya que el clic prolongado es una acción a la que están más acostumbrados.

Por lo tanto, la solución que se propone es añadir a los ítems de la lista de vehículos la funcionalidad de eliminar o editar mediante deslizamiento lateral.

5.5 MANUALES DEL SISTEMA

5.5.1 Manual de Usuario

5.5.1.1 Primeros pasos

Activar Bluetooth

La primera vez que el usuario ejecute la aplicación se le requerirá que active el Bluetooth del dispositivo en caso de no estar activado.

Figura 5.43: Manual de usuario: Pantalla "Activar Bluetooth"

Permiso de localización

Una vez activado tendrá de conceder los permisos de localización necesarios para que funcione correctamente el servicio Bluetooth. Como se puede ver a continuación, primero se le mostrará un mensaje de aviso y una vez aceptado se desplegará el cuadro de dialogo de Android para conceder el permiso.

Figura 5.44: Manual de usuario: Pantalla "Permiso de localización"

Vehículo por defecto

Una vez concedido el permiso de localización el usuario se encontrará en la pantalla "Inicio" en la cual verá un coche de 4 ruedas con un recuadro para cada una de ellas. Este vehículo se crea por defecto. A partir de aquí se plantean dos opciones:

1. Si el vehículo coincide con el que el usuario quiere monitorizar podrá cambiarle el nombre si lo desea y continuar con el siguiente paso: Emparejar sensor.
2. Si el vehículo no coincide o si el usuario lo desea, puede Crear un nuevo vehículo.

Figura 5.45: Manual de usuario: Pantalla "Vehículo por defecto"

5.5.1.2 Pantalla “Inicio”

El usuario debe acceder a la ventana “Inicio” mediante la barra de navegación situada en la parte inferior. Esta es la pantalla principal de la aplicación cuya funcionalidad es mostrar en tiempo real los datos enviados por los sensores. Como se puede ver en la siguiente imagen se muestra el vehículo principal y un recuadro para cada rueda.

Figura 5.46: Manual de usuario: Pantalla "Inicio"

Los recuadros constan de una parte central que muestra la presión y la temperatura, y una barra en la parte superior con unos iconos que aportan información extra mediante colores. De izquierda a derecha se describen a continuación:

1. Calidad de señal: En verde si la señal del sensor es correcta. En rojo si el sensor tiene algún problema de conexión.
2. Advertencia de presión: sin color si la presión se encuentra dentro del rango establecido. En rojo en caso contrario.
3. Advertencia de temperatura: sin color si la temperatura se encuentra dentro del rango establecido. En rojo en caso contrario.
4. Porcentaje de batería: sin color si queda un 25% o más de batería restante. Rojo en caso contrario.

Además, en caso de excederse los límites de temperatura o presión se mostrarán en rojo los valores de la parte central del recuadro.

Para modificar los límites de advertencia o las unidades de medida ver el apartado Ajustes.

5.5.1.3 Pantalla “Emparejar”

El usuario debe acceder a la ventana “Emparejar” mediante la barra de navegación situada en la parte inferior. En esta pantalla podrá ver el vehículo actualmente seleccionado como “Vehículo principal” (ver Seleccionar vehículo principal) y los recuadros correspondientes a cada rueda en los cuales puede verse el identificador del sensor en caso de estar emparejado o bien la palabra “Emparejar” en caso contrario.

Figura 5.47: Manual de usuario: Pantalla "Emparejar"

5.5.1.3.1 Emparejar sensor

Clicar en el recuadro de la rueda que se quiera emparejar y seleccionar una de las opciones de emparejamiento.

Figura 5.48: Manual de usuario: Opciones de emparejamiento

Emparejamiento automático

Una vez seleccionada la opción de emparejamiento automático se mostrará una ventana con las instrucciones. El procedimiento consiste en tener preparado el sensor e inmediatamente después de clicar en "Buscar" conectarlo a la válvula del neumático. El sensor se activará al notar la presión y comenzará a enviar los datos. De esta manera la aplicación detectará el nuevo dispositivo y lo emparejará. Para este proceso es necesario tener el Bluetooth activado. En caso de no encontrar el nuevo sensor el usuario deberá repetir el proceso.

Figura 5.49: Manual de usuario: Emparejamiento automático

Emparejamiento manual

Ingresar el identificador del sensor y clicar en "Guardar". Los identificadores se encuentran en una tarjeta que viene dentro de la caja del producto como se puede ver a continuación.

Figura 5.50: Manual de usuario: Tarjeta con identificadores

5.5.1.3.2 Desemparejar sensor

Clicar en el recuadro de la rueda que se quiere desemparejar y seleccionar "Desemparejar".

Figura 5.51: Manual de usuario: Desemparejar sensor

5.5.1.4 Pantalla “Mis vehículos”

El usuario debe acceder a la ventana “Mis vehículos” mediante la barra de navegación situada en la parte inferior. Esta pantalla es la que permite gestionar los vehículos. Se muestra una lista con los vehículos registrados por el usuario. El icono ★ indica que el vehículo es el principal y por lo tanto será el que se muestre en el resto de las pantallas de la aplicación.

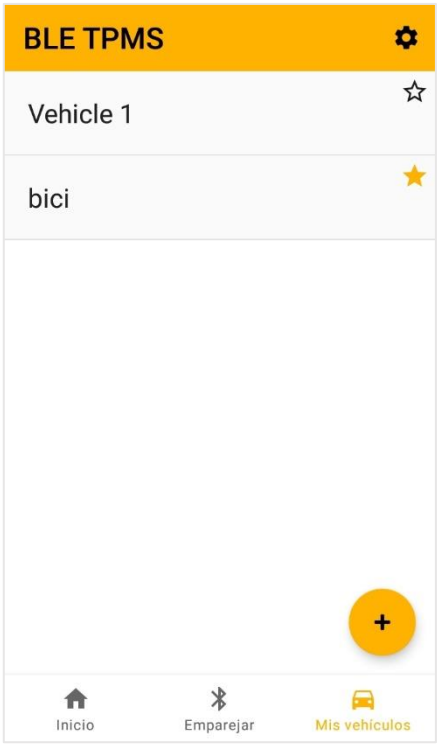


Figura 5.52: Manual de usuario: Pantalla "Mis vehículos"

5.5.1.4.1 Crear vehículo

Clicar en el botón “+” Situado en la parte inferior derecha de la pantalla. Se desplegará una ventana en la que habrá que ingresar el nombre y seleccionar el tipo de vehículo. Una vez creado, el nuevo vehículo pasará automáticamente a vehículo principal.

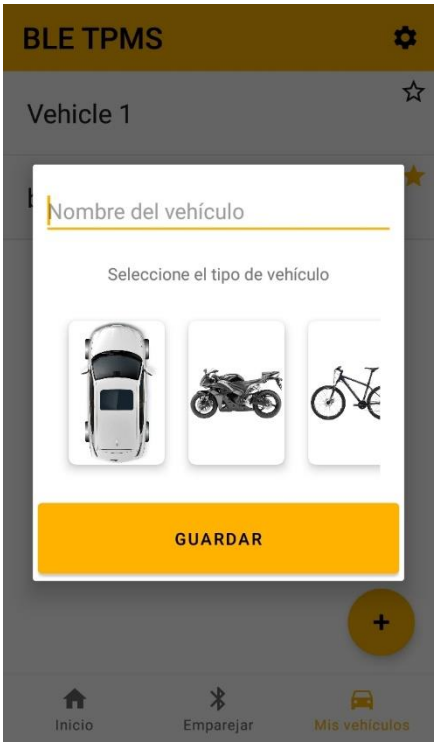


Figura 5.53: Manual de usuario: Crear vehículo

5.5.1.4.2 Editar o eliminar vehículo

Editar vehículo

Realizar un clic prolongado sobre el vehículo que se quiera editar hasta que se despliegue un menú contextual en la barra superior. Al clicar sobre el icono ✎ se abrirá una ventana en la que se podrá modificar el nombre del vehículo.

Eliminar vehículo

Realizar un clic prolongado sobre el vehículo que se quiera editar hasta que se despliegue un menú contextual en la barra superior. Al clicar sobre el icono 🗑 se el vehículo quedará eliminado.

Para eliminar múltiples vehículos a la vez realizar un clic prolongado sobre uno de los vehículos a eliminar y añadir a la selección el resto de los vehículos mediante un clic. Después clicar sobre el icono 🗑 y los vehículos quedarán eliminados.

Para poder eliminar un vehículo, este no debe estar seleccionado como vehículo principal. Por lo tanto, es necesario cambiar el vehículo principal a otro que se desee

mantener o bien crear un nuevo vehículo. De esta manera la aplicación obliga a tener al menos un vehículo creado.

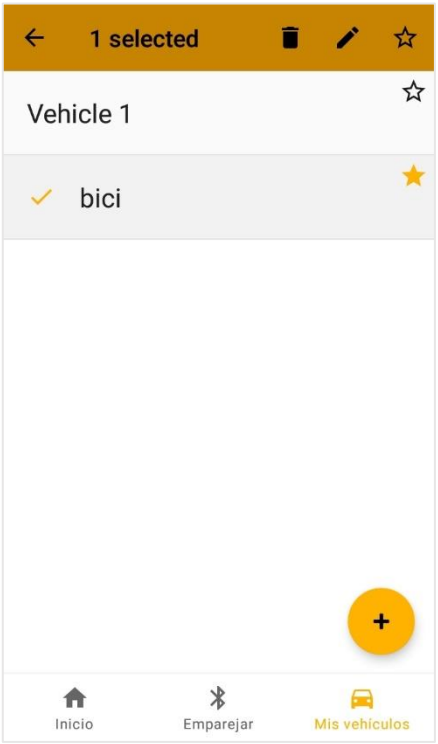


Figura 5.54: Manual de usuario: Editar o eliminar vehículo

5.5.1.4.3 Seleccionar vehículo principal

Para cambiar el vehículo principal clicar sobre el icono ☆ o bien realizar un clic prolongado sobre el vehículo y clicar sobre el mismo icono en el menú contextual de la barra superior.

5.5.1.5 Pantalla "Ajustes"

Para acceder a la ventana "Ajustes" clicar sobre el icono ⚙ situado en la parte superior derecha en cualquiera de las pantallas de la aplicación.

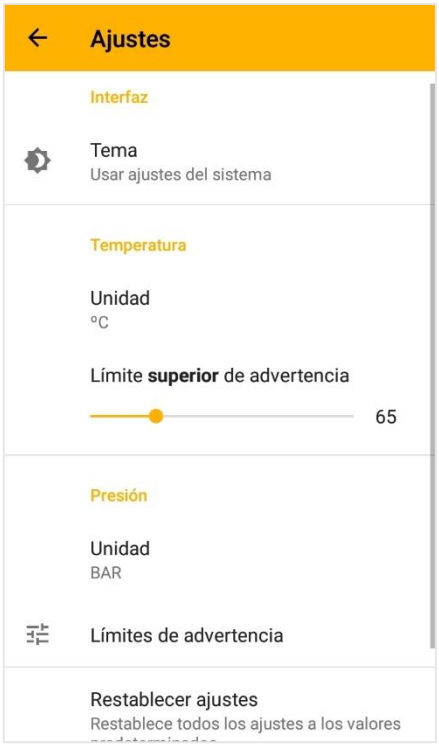


Figura 5.55: Manual de usuario: Pantalla "Ajustes"

Para cambiar las unidades de medida de temperatura o presión clicar sobre “Unidad” y seleccionar una de las opciones que se ofrecen. Automáticamente se actualizarán los valores de las barras de límites de advertencia.

Para modificar los límites de advertencia de temperatura utilice la barra “Límite superior de advertencia”. La temperatura solo dispondrá de límite superior.

Para modificar los límites de advertencia de presión clicar en “Límites de advertencia” en el apartado “Presión” para acceder a la siguiente ventana. La presión dispondrá de límite de advertencia superior e inferior.

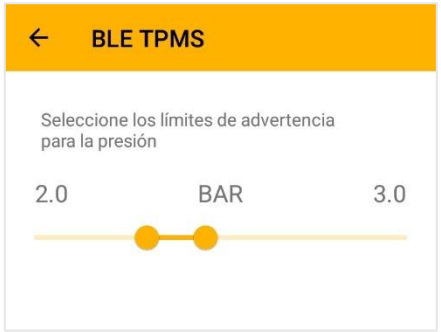


Figura 5.56: Manual de usuario: Límites de advertencia para la presión

5.5.2 Manual de Instalación

1. Obtener el archivo .apk mediante una de las siguientes opciones:
    - Archivo ble_tpms.apk adjunto a este documento (ver 9.1)
    - Descargar desde el siguiente enlace: https://unioviedo-my.sharepoint.com/:u:/g/personal/uo240279_uniovi_es/EeqAwGMmNUtKio7vmTynPN8BV7RID1bZyMabmmiu_FLirA?e=2YhXuc

2. Ejecutar ble_tpms.apk desde un dispositivo Android. Para ello el archivo debe estar en la memoria del dispositivo, ya sea transfiriéndolo desde un ordenador o bien descargándolo mediante el enlace.

3. Conceder permisos para instalar aplicaciones desconocidas. Pare ello siga los siguientes pasos:

    - Seleccione Ajustes

Figura 5.57: Permisos de instalación desde fuentes desconocidas I

- Active “Permitir desde esta fuente”

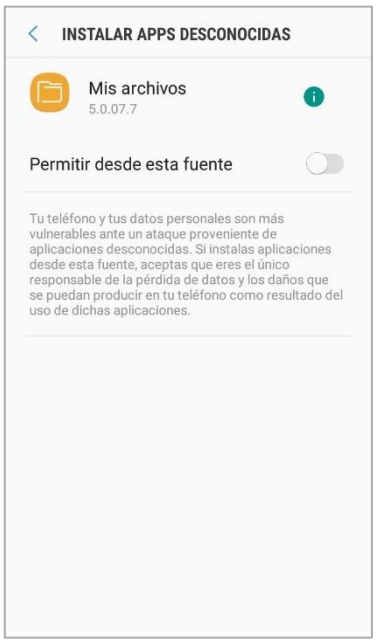


Figura 5.58:Permisos de instalación desde fuentes desconocidas II

- Regrese a la ventana anterior y seleccione “Instalar”

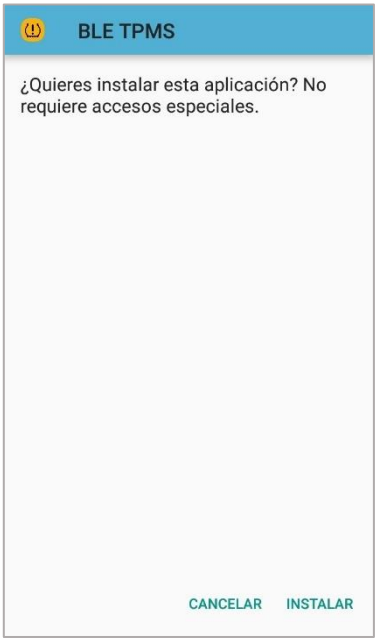


Figura 5.59: Permisos de instalación desde fuentes desconocidas III

5.5.3 Manual del Programador

De cara a futuras ampliaciones o por mera curiosidad se describe a continuación el entorno de desarrollo y algunas de las partes más importantes del código de la aplicación.

La aplicación se ha desarrollado en Java utilizando el entorno Android Studio, por lo tanto, se recomienda su instalación. Para obtener el código habrá que descargar el siguiente repositorio de GitHub: https://github.com/ignaciobermejo/BLE_TPMS_APP.git
Una vez creado el proyecto se podrá observar la siguiente estructura:

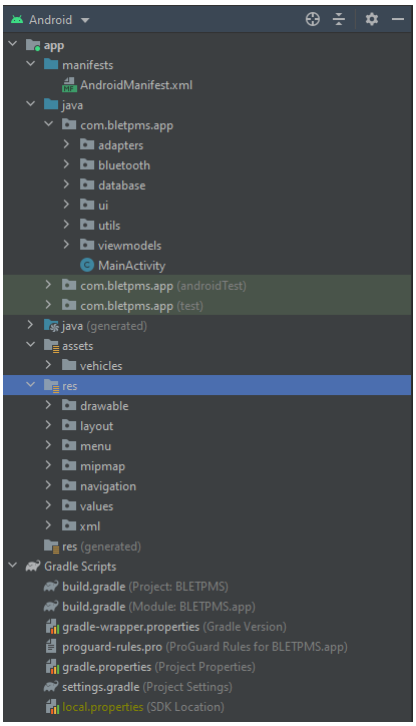


Figura 5.60: Manual del programador: Estructura del código

Dentro de la carpeta “java” se encuentra los paquetes de las clases descritos en apartados anteriores:

- com.bletpms.app: 5.2.2 - Diseño de Clases
- com.bletpms.app (androidTest): 5.2.5.2 - Pruebas de Integración y del Sistema
- com.bletpms.app (test): 5.2.5.1 - Pruebas Unitarias

En la carpeta “assets.vehicles” se encuentran las imágenes de los vehículos en formato PNG y nombradas según su número de ruedas y distribución. En caso de querer añadir más tipos de vehículos se recomienda guardar las nuevas imágenes en esta carpeta y utilizar la

clase auxiliar BitmapFromAssetsProvider del paquete “utils” para obtener el bitmap en el código.

En la carpeta “res” se encuentran todos los recursos utilizados por la aplicación (a excepción de las imágenes de los vehículos). El paquete más importante es “layout”, el cual contiene los archivos XML con los diseños de las vistas.

En la ruta “res.values.themes” se encuentran los XML que contienen los colores de la aplicación.

5.5.3.1 Añadir tipos de vehículos

En este apartado se describirá cómo añadir nuevos tipos de vehículos. Cualquier usuario con conocimientos mínimos de desarrollo en Android podría ampliar esta funcionalidad si se da el caso de que su vehículo no se encuentra entre los propuestos por la aplicación, o bien, si desea cambiar cualquiera de las imágenes de los vehículos que se ofrecen actualmente. Los pasos a seguir son:

1. Guardar la imagen en formato PNG y con una resolución semejante a las demás en la ruta “app.assets.vehicles”. Se recomienda nombrar el archivo de forma simple a modo de identificador (Ej.: “6+6rem”: vehículo de 6 ruedas con un remolque de 6 ruedas).
2. En la ruta “app.res.layout” escoger uno de los XML cuyo nombre empiece por “vehicle_layout” que más se ajuste al tipo de vehículo que se quiere añadir, o bien, diseñar uno nuevo. Estos archivos son los correspondientes a los diseños de las vistas donde se muestra el vehículo con los recuadros de cada rueda.
3. En la clase “VehicleTypes” (app.java.com.bletpms.app.utils) añadir al enumerador “VehicleType” el nuevo tipo de vehículo siguiendo la estructura: NOMBRE(“nombre_del_png”, nº de ruedas, nombre_del_layout).

Capítulo 6. Conclusiones y Ampliaciones

6.1 CONCLUSIONES

El objetivo de este trabajo fin de grado era cuestionar la calidad de un producto tanto desafiando la seguridad de su hardware como mejorando la interfaz y la funcionalidad de su software. Los sistemas de monitorización de presión y temperatura (TPMS) son un elemento de uso común en la automoción y debido a que habitualmente vienen integrados en los vehículos la gente no repara en su importancia y mucho menos en su funcionamiento.

Este proyecto se ha centrado en los TPMS por Bluetooth que es un producto que permite dotar a los vehículos antiguos de este sistema haciendo uso de los dispositivos móviles.

Tras realizar el estudio de mercado se ha podido concluir que los TPMS de calidad son escasos y tienen precios muy elevados. Recientemente se han empezado a producir a precios asequibles y la mayoría presenta algún tipo de defecto o una funcionalidad incompleta.

Al final de este trabajo se ha podido determinar que la producción de TPMS de calidad y a buen precio no requiere un esfuerzo desmesurado, lo que lleva a pensar que aquí hay un posible nicho de mercado.

Entrando un poco más en detalle se puede dividir este producto en dos partes: hardware y software. En cuanto al hardware y debido a la falta de conocimientos suficientes en electrónica no se profundizará. Basta con decir que son aparatos realmente sencillos: una pequeñísima placa con una pila del mismo tamaño y en el caso del TPMS que aquí se analiza, un ensamblaje y un contenedor (de plástico) muy pobres. Por lo tanto, bastaría con prestar un poco más de atención a esta parte sin que ello conlleve un incremento de recursos muy alto.

En cuanto al software también se ha dividido en dos partes: el sistema Bluetooth y la aplicación móvil.

El primer desafío de este proyecto ha sido comprender la tecnología Bluetooth para más adelante centrarse en la modalidad que utiliza este TPMS. En este punto del proyecto se cometió un grave error que sin duda hizo perder mucho tiempo. Se enfocó la búsqueda de los datos en el tipo de comunicación que no era, probablemente por no profundizar lo suficiente. Tras muchas horas de estudio en la dirección incorrecta, se encontró un documento con una descripción muy breve pero suficiente para entender qué tipo de comunicación Bluetooth realizaban los sensores y en qué formato se enviaban los datos.

En un principio se creía que los TPMS tendrían algún tipo de sistema de seguridad, lo que obligó a utilizar herramientas de hacking sin ningún resultado, pero finalmente resultó ser algo muy sencillo. Los sensores se comunican de una manera muy simple. En cuanto detectan cambios de presión o temperatura comienzan a emitir los datos abiertamente y sin establecer ninguna conexión con el dispositivo receptor. Llegados a este punto, concluía la primera parte del trabajo.

La segunda parte del proyecto consistió en desarrollar una aplicación Android sin tener conocimiento previo. Ha sido un trabajo tedioso y en cierta manera autodidacta, pero ha demostrado algo muy importante de lo que no se había tomado conciencia: los conocimientos adquiridos en esta carrera no son tan genéricos y poco útiles como se pensaba. La experiencia obtenida a lo largo de estos años permite desarrollar en cualquier entorno o lenguaje de software desconocido y llegar a desenvolverse con relativa soltura en poco tiempo.

Tras finalizar este proyecto se resume el aprendizaje en tres puntos: ciertos conocimientos en la tecnología Bluetooth y más concretamente en Bluetooth Low Energy (BLE), en auge debido al internet de las cosas (IoT). Conocimientos básicos de hacking, como captura de datos, descifrado de paquetes y lectura de tramas. Y, por último, programación básica en Android, lo que permitirá desarrollar aplicaciones sencillas o realizar pequeñas aportaciones en proyectos más grandes.

6.2 AMPLIACIONES

A continuación, se describen las posibles ampliaciones para llevar a cabo en la aplicación. Debido a la falta de tiempo y la complejidad de estas, no se han podido desarrollar.

6.2.1 Intercambiar ruedas

Esta funcionalidad consiste en añadir una pantalla que permita al usuario intercambiar las ruedas del vehículo de manera que conserve el sensor que tiene actualmente emparejado. Esto se debe a que es muy habitual que al cambiar los neumáticos de un vehículo se instalen los traseros en el lugar de los delanteros y viceversa, o bien, que se intercambien directamente uno por otro.

Para llevar a cabo esta ampliación habría que desarrollar una nueva interfaz que permita seleccionar las ruedas que se va a intercambiar. Se podría utilizar una interfaz semejante a la de las pantallas “Inicio” y “Emparejar” en las que se añada a las vistas de las ruedas una funcionalidad semejante a la de un elemento seleccionable, de manera que cuando se encuentren las dos ruedas seleccionadas se intercambien la rueda 1 por la 2.

6.2.2 Vehículo de más de 8 ruedas

Esta característica se ha encontrado en otra aplicación semejante, vista en el apartado 1.3.2.1. Permitiría al usuario utilizar más de 8 sensores simultáneamente, lo cual sería muy útil cuando se tienen vehículos grandes como camiones de transporte con gran carga.

Inicialmente se descartó incluir esta funcionalidad por tema de diseño ya que no es posible mostrar más de 8 vistas de ruedas simultáneamente en la aplicación de forma que el usuario pueda ver la información de cada rueda a un tamaño legible en la misma pantalla y sin tener que deslizar hacia abajo. Uno de los requisitos iniciales de esta aplicación fue poder utilizarla sin interacción del usuario, es decir, a modo de monitor instalado en el vehículo. Una vez que el vehículo iniciase la marcha, el usuario podría utilizar el dispositivo móvil para extender la información de la centralita.

Por lo tanto se propone una interfaz especial para vehículos de más de 8 ruedas en la cual se muestra la información de manera más concisa y menos visual mediante texto plano. Cada rueda se representa mediante una línea que muestra los datos resaltados con colores.

6.2.3 Avisos con sonido

Esta ampliación propone dotar a la aplicación de avisos de sonido cuando se excedan los límites de advertencia de presión y temperatura. De esta manera el usuario sería informado, aunque no esté mirando el dispositivo móvil, lo cual resultaría muy útil ya que la mayor parte del tiempo estaría con la vista fijada en la carretera.

Otra ampliación que se podría llevar a cabo a raíz de ésta sería convertir la aplicación en un servicio en segundo plano. Actualmente está diseñada para funcionar únicamente cuando se encuentre abierta y en primer plano. Esto permitiría al usuario utilizar la aplicación con el dispositivo bloqueado o guardado en el bolsillo de manera que solo requiriese de su atención cuando un aviso de sonido le indique que algo va mal.

6.2.4 Imágenes de vehículos personalizadas

Esta funcionalidad permitiría al usuario elegir la imagen del su vehículo cargándola desde el dispositivo móvil. Se requeriría un formato y un tamaño determinado.

Para implementarlo se propone cambiar la vista de añadir vehículo a una pantalla nueva en vez de un cuadro de dialogo. Esto permitiría tener más espacio para incluir las opciones por defecto y la funcionalidad de cargar imagen desde el dispositivo, incluso se podría enseñar una vista previa del resultado.

Capítulo 7. Presupuesto

Este apartado tiene como objetivo simular un presupuesto real del proyecto que se ha llevado a cabo. Al tratarse de un trabajo de fin grado, el alumno no ha recibido ningún tipo de remuneración y el único coste asumido ha sido la compra de un dispositivo (por decisión propia de alumno) necesario para la parte de hacking (Placa Micro:Bit).

El presupuesto que se detallada a continuación se divide en dos partes: un presupuesto interno con los costes que le supondría a la empresa desarrollar el proyecto y un presupuesto para el cliente que añade tanto el IVA como el porcentaje de beneficio prorrateado.

El proyecto ha sido realizado por una única persona que de cara al presupuesto figurará como analista programador y se ha determinado que el salario bruto es de 15 €/hora.

7.1 PRESUPUESTO INTERNO

7.1.1 Recursos de la empresa

RECURSOS DE TIPO TRABAJO	PRECIO/H
Analista programador	15,00 €

Tabla 83: Costes de recursos de tipo trabajo

7.1.2 Presupuesto de costes

En la siguiente tabla se muestran los costes directos divididos por partidas. Se opta por no realizar un desglose de éstas ya que todas las subtareas se realizan por la misma persona. Es resultado son 389 horas de trabajo a un coste de **5.835,00 €.**

COSTES DIRECTOS			
PARTIDA	**HORAS**	**PRECIO/HORA**	**TOTAL**
Estudio de la situación actual	8	15,00 €	120,00 €
Aprendizaje de fundamentos	30	15,00 €	450,00 €
Hacking del TPMS	43	15,00 €	645,00 €
Desarrollo de la aplicación	215	15,00 €	3.225,00 €
Desarrollo de pruebas	19	15,00 €	285,00 €
Documentación	74	15,00 €	1.110,00 €
		TOTAL	**5.835,00 €**

Tabla 84: Costes directos

En la siguiente tabla se muestran los costes indirectos durante los 7 meses de desarrollo del proyecto. Se ha calculado su tarifa mensual mediante los siguientes ajustes.

Los dos primeros cargos se corresponden al hardware que la empresa facilita al programador pero que no han sido comprados específicamente para este proyecto. Son recursos que la empresa va a amortizar durante un periodo mínimo de 3 años. El ordenador tiene un precio de 900 euros y el dispositivo Android de 360 euros. El tercer cargo se corresponde a un dispositivo de captura de tráfico Bluetooth que se ha comprado específicamente para este proyecto. Su precio mensual se calcula dividiendo el coste total de 21 euros entre los 7 meses que dura el proyecto con el objetivo de amortizarlo a lo largo de este. Para las licencias de software se incluye el precio actual de mercado y para el alquiler de oficina, gasto de electricidad y conexión a internet se tiene en cuenta que la empresa dispone de instalaciones en las que se llevan a cabo otros proyectos. El precio es la parte proporcional correspondiente a este proyecto. Por último, se consideran 50 euros de gastos de transporte, ya sea para combustible o para transporte público, de los que el trabajador dispone para los desplazamientos desde al domicilio hasta las oficinas.

COSTES INDIRECTOS			
CONCEPTO	**MES**	**PRECIO/MES**	**TOTAL**
Ordenador	7	25,00 €	175,00 €
Dispositivo Android	7	10,00 €	70,00 €
Placa Micro:Bit	7	3,00 €	21,00 €
Licencia Microsoft Office	7	10,50 €	73,50 €
Licencia Microsoft Project	7	25,30 €	177,10 €
Alquiler de oficina	7	150,00 €	1.050,00 €
Internet	7	30,00 €	210,00 €
Electricidad	7	15,00 €	105,00 €
Transporte	7	50,00 €	350,00 €
		TOTAL	**2.231,60 €**

Tabla 85: Costes indirectos

7.1.3 Agregación final del presupuesto de costes

En la siguiente tabla se muestra el coste total de realización del proyecto mediante la agregación de costes directos e indirectos.

CONCEPTO	PRECIO
Costes directos	5.835,00 €
Costes indirectos	2.231,60 €
TOTAL	**8.066,60 €**

Tabla 86: Agregación de costes

En el siguiente apartado se muestra el presupuesto para el cliente en cual se han prorrateado los costes indirectos y un beneficio del 25% entre las diferentes partidas. Finalmente, se le ha aplicado un 21% de IVA.

7.2 PRESUPUESTO DE CLIENTE

PARTIDA	**TOTAL**
PARTIDA 1: Estudio de la situación actual	195,89 €
PARTIDA 2: Aprendizaje de fundamentos	734,60 €
PARTIDA 3: Hacking del TPMS	1.052,93 €
PARTIDA 4: Desarrollo de la aplicación	5.264,65 €
PARTIDA 5: Desarrollo de pruebas	465,25 €
PARTIDA 6: Documentación	1.812,02 €
TOTAL CLIENTE	**9.525,35 €**
IVA (21%)	2.381,34 €
TOTAL CLIENTE + IVA	**11.906,69 €**

Tabla 87: Presupuesto de cliente

Capítulo 8. Referencias Bibliográficas

[1] TPMSII - Google Play Store, [En línea]. Available: https://play.google.com/store/apps/details?id=com.chaoyue.tyed. [Último acceso: 05 Octubre 2020].

[2] Multi Wheel BLE TPMS - Google Play Store, [En línea]. Available: https://play.google.com/store/apps/details?id=com.sysgration.tpms.app. [Último acceso: 05 Octubre 2020].

[3] StoreBao USB TPMS - Google Play Store, [En línea]. Available: https://play.google.com/store/apps/details?id=com.tpms3. [Último acceso: 05 Octubre 2020].

[4] HELLA España, Webinar Completo - Sensor de Control de la Presión de los Neumáticos (TPMS), 2020. [En línea]. Available: https://www.youtube.com/watch?v=jj5Dhsi_rhE. [Último acceso: 07 Octubre 2020].

[5] Parlamento Europeo y del Consejo, «Agencia Estatal Boletín Oficial del Estado,» REGLAMENTO (UE) Nº 523/2012 DE LA COMISIÓN, 20 junio 2012. [En línea]. Available: https://boe.es/doue/2012/160/L00008-00012.pdf. [Último acceso: 07 octubre 2020].

[6] Microchip Developer, Introduction to Bluetooth® Low Energy, [En línea]. Available: https://microchipdeveloper.com/wireless:ble-introduction. [Último acceso: 13 octubre 2020].

[7] Micro:Bit, [En línea]. Available: https://microbit.org/. [Último acceso: 29 abril 2021].

[8] Wireshark, [En línea]. Available: https://www.wireshark.org/. [Último acceso: 1 mayo 2021].

[9] BtleJack, «Github/virtualabs,» 2019. [En línea]. Available: https://github.com/virtualabs/btlejack. [Último acceso: 10 mayo 2021].

[10] BLE-TPMS, «Github/ra6070,» [En línea]. Available: https://github.com/ra6070/BLE-TPMS. [Último acceso: 18 enero 2021].

[11] Documentación de Java, [En línea]. Available: https://docs.oracle.com/en/java/. [Último acceso: 26 mayo 2021].

[12] Documentación de SQL, [En línea]. Available: https://dev.mysql.com/doc/. [Último acceso: 25 mayo 2021].

[13] Documentación de XML, [En línea]. Available: https://www.w3schools.com/xml/el_documentation.asp. [Último acceso: 25 mayo 2021].

[14] Documentación de Android Studio, [En línea]. Available: https://developer.android.com/studio/intro. [Último acceso: 26 mayo 2021].

[15] Documentación de SQLite, [En línea]. Available: https://www.sqlite.org/docs.html. [Último acceso: 26 mayo 2021].

[16] Documentación de Room, [En línea]. Available: https://developer.android.com/training/data-storage/room. [Último acceso: 26 mayo 2021].

[17] Microsoft Word, [En línea]. Available: https://www.microsoft.com/es-es/microsoft-365/word. [Último acceso: 26 mayo 2021].

[18] Microsoft Project, [En línea]. Available: https://www.microsoft.com/es-es/microsoft-365/project/project-management-software. [Último acceso: 26 mayo 2021].

[19] Draw.io, [En línea]. Available: https://app.diagrams.net/. [Último acceso: 26 mayo 2021].

[20] C. Y. Ong, «Punch Through,» The Ultimate Guide to Android Bluetooth Low Energy, 15 mayo 2020. [En línea]. Available: https://punchthrough.com/android-ble-guide/. [Último acceso: 18 enero 2021].

[21] J. Lindh, «Texas Instruments,» Bluetooth®low energy Beacons, 2015-2016. [En línea]. Available: https://www.ti.com/lit/an/swra475a/swra475a.pdf. [Último acceso: 18 enero 2021].

[22] Room Guide, «Codepath,» [En línea]. Available: https://guides.codepath.com/android/Room-Guide. [Último acceso: 11 enero 2021].

[23] GitHub, [En línea]. Available: https://github.com/.

[24] Stack Overflow, [En línea]. Available: https://stackoverflow.com/.

[25] Medium, [En línea]. Available: https://medium.com/.

[26] Youtube, [En línea]. Available: https://www.youtube.com/.

Capítulo 9. Apéndices

9.1 CONTENIDO ENTREGADO

Además de este documento, se hace entrega de un fichero comprimido en ZIP que contiene lo siguiente:

- **README.txt**: fichero de texto con la descripción del contenido de la carpeta.
- **ble_tpms.apk**: archivo de instalación de la aplicación para dispositivos Android.
- **ble_tpms.zip**: fichero que contiene el código de la aplicación. Incluye los mismos archivos que el repositorio de GitHub:
  https://github.com/ignaciobermejo/BLE_TPMS_APP.
- **Presupuesto_TFG.xlsx**: archivo de Microsoft Excel que contiene el presupuesto del proyecto.
- **Planificación_TFG.mpp**: archivo de Microsoft Project que contiene la planificación del proyecto.
- **Diagrama_clases.png**: imagen del diagrama de clases de la aplicación.