Facultad
de
Ciencias

APP para la monitorización en tiempo real del estado del vehículo
(Realtime vehicle status tracking mobile APP)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: David Berbil Ruiz

Directora: Marta Elena Zorrilla Pantaleón

Febrero - 2018

Índice de contenido

Índice de tablas

Índice de imágenes

Agradecimientos

Sirvan estas primeras líneas para agradecer a mi familia lo que han hecho por mi durante mi vida, el acompañarme durante este camino y ayudarme en todo lo necesario.

De igual manera, agradecer a mis amigos y compañeros de clase el apoyo que me han dado y esa sensación de sentirme como en casa con todos ellos, durante todos estos años.

Finalmente, agradecer a todos los profesores con los que he compartido aula, por todos los conocimientos que me han transmitido, al igual que las buenas prácticas y formas de trabajar inducidas. En especial, este agradecimiento va dirigido a la profesora Marta Zorrilla, quién además ha dirigido este proyecto.

Resumen

Los avances tecnológicos producidos durante los últimos años en el ámbito de las comunicaciones móviles en el sector del automóvil han tenido como consecuencia un aumento en la capacidad de gestión y control por parte de los fabricantes y de los clientes sobre sus vehículos, además de facilitar la interacción entre ellos. Sin embargo, en su mayoría, estas nuevas funcionalidades se ven ligadas a las propias marcas, teniendo que recurrir a aplicaciones oficiales y a los talleres oficiales para hacer uso de ellas.

Por ello, CEINOR, ha decidido desarrollar una aplicación independiente de los fabricantes que ofrezca funcionalidades equivalentes a las proporcionadas por estos. Entre ellas, siendo esta el objetivo del proyecto, está el desarrollo de un módulo que permita conocer en tiempo real información del vehículo, bien sean parámetros su funcionamiento, como posibles averías o fallos en el mismo.

Esta aplicación denominada *Smart Monitoring* se ha desarrollado para dispositivos con sistema operativo Android utilizando Xamarin, tecnología que permite el desarrollo de aplicaciones móviles multiplataforma.

Palabras clave: automoción conectada, diagnóstico a bordo, estándar OBDII, aplicación móvil.

Abstract

Technological developments produced during the last few years in the mobile communications arena applied to the automotive sector have led to an increase in the management and control of the vehicles by manufacturers and customers. However, most of these new features are provide by car brands, being necessary to buy official applications and go to official garage to take advantage of them.

For this reason, CEINOR decided to develop an application that offers features equivalent to those provided by official brands. One of these functionalities, and the main goal of this project, is the development of a module that allows users to know vehicle information, such as internal parameters or possible troubles of the car in real time.

This application is available for Android devices. This was developed with Xamarin, a multiplatform suite.

Keywords: car-connected, diagnostic on board, mobile application, OBDII standard.

Acrónimos

Abreviatura	Significado
ECU	Engine Control Unit (Unidad de Control de Motor)
OBD	On Board Diagnostics
ACID	Atomicy, Consistency, Isolation, Durability
TFS	Team Foundation Server
SAE	Society of Automotive Engineers
PID	Parameter ID
DTC	Diagnostic Trouble Code
MVVM	Model-View-ViewModel (Modelo-Vista-Modelo de Vista)
UUID	Universally Unique Identifier
MVC	Modelo-Vista-Controlador

Tabla 1. Acrónimos presentes en el documento

1.- Introducción

En este capítulo, se expone de forma breve el contexto bajo el cual se ha desarrollado este trabajo de fin de grado, sus objetivos y la metodología y plan de trabajos seguidos.

1.1.- Antecedentes y objetivos.

La industria del automóvil ha sido, durante los últimos años, un referente en innovación tecnológica y en la incorporación de sus avances en el equipamiento de vehículos. Aspectos como la seguridad y la eficiencia de los vehículos se han visto incrementados de forma notoria y, en gran medida, esto se ha debido a la incorporación y la evolución de la tecnología en nuestros coches.

Otro de los aspectos fundamentales que ha surgido y evolucionado a raíz de este desarrollo es el de la monitorización y diagnóstico de vehículos. Años atrás los datos que se podían conocer del estado del vehículo no iban más allá de los proporcionados por el propio cuadro de instrumentos, o los indicados por un mecánico después subir el coche a un elevador y revisar pieza a pieza su estado.

Hoy en día, los vehículos disponen de ordenadores de abordo que ofrecen una gran cantidad de información en tiempo real, permitiendo al usuario beneficiarse ampliamente de ello.

El proyecto desarrollado tiene como objetivo facilitar al usuario información de distinta índole del vehículo a través de su Smartphone.

Este proyecto se enmarca en el desarrollo de la aplicación *CarEConnect*, aplicación móvil lanzada por CEINOR, una empresa de desarrollo software orientada a al sector de la automoción.

Esta aplicación permite interconectar los vehículos de los usuarios con sus talleres, ofreciendo una alternativa a los talleres no oficiales, que no disponen del software oficial de las marcas. Esta aplicación dispone, entre sus diversas funcionalidades, un módulo denominado *Smart Monitoring*, que permite a los usuarios en tiempo real obtener datos de funcionamiento del vehículo, así como analizar posibles fallos mecánicos que sufra el mismo. La comunicación se realiza mediante tecnología Bluetooth [1], que permite a los usuarios, a través de los móviles, conectarse a la "Unidad de Control de Motor" (ECU) [2], para disponer de todos estos datos. Esta conexión se realiza mediante la implantación de un sistema estándar de diagnóstico de a bordo en vehículos (OBDII) [3].

Por tanto, para el uso de este módulo, además de la propia aplicación, se requiere de un Smartphone con tecnología Bluetooth, así como de un dispositivo OBDII. Este módulo, dispone de una base de datos interna, implementada en SQLite [4], para el almacenamiento de la información, y, para ciertas funcionalidades, se apoya en

servicios web desarrollados por la empresa, fuera del ámbito de este proyecto de trabajo de fin de grado.

1.2.- Metodología

La metodología utilizada para el desarrollo de este proyecto se ha visto influenciada por la propia metodología de trabajo utilizada en CEINOR.

Así, se puede describir como una metodología en cascada [5], que cuenta con algunos aspectos provenientes de metodologías ágiles, como son unos requisitos dinámicos y rectificables a lo largo del proyecto y la implicación por parte de los interesados (en este caso los directivos de CEINOR) en el desarrollo del proyecto, pero siguiendo las fases básicas de la metodología en cascada al igual que su secuencia.

De esta manera, el proyecto comenzó con una captura inicial de requisitos en una serie de reuniones entre el equipo de desarrollo y los accionistas y directivos de CEINOR. Tras esto, se especificaron los modelos software iniciales y se inició el proceso de implementación. Se realizaban reuniones diarias (similar a *Daily SCRUM Meeting*) que permitían dar a conocer los distintos avances y complicaciones durante todo el proceso, así como posibles modificaciones tanto en requisitos como en la implementación. Una vez se completó una solución estable con las funcionalidades deseadas, comenzó una etapa de pruebas a la par que el propio refinamiento de la implementación. Se debe señalar también que, durante el inicio del proyecto, se realizó un proceso de familiarización tanto de la terminología propia del sector como de las herramientas y tecnologías utilizadas para su desarrollo.

1.3.- Plan de trabajo

El plan de trabajo seguido para este proyecto ha dependido directamente de las fechas marcadas por la propia empresa para la finalización de este, ya que el proyecto de la aplicación móvil finalizaba durante el mes de diciembre. De esta manera, el proyecto se desarrolló en 69 días, iniciándose este módulo el 4 de septiembre y finalizándose el 7 de diciembre.

Tal y cómo se describía en el apartado anterior, la metodología de trabajo estaba basada principalmente en una metodología en cascada, recogiendo el plan de trabajo (ver Figura 1) resultante una primera toma de contacto con el proyecto y la empresa, la captura de requisitos y el diseño de la aplicación, su implementación, subdividida en los principales módulos de esta fase de implementación, y una última fase de pruebas.

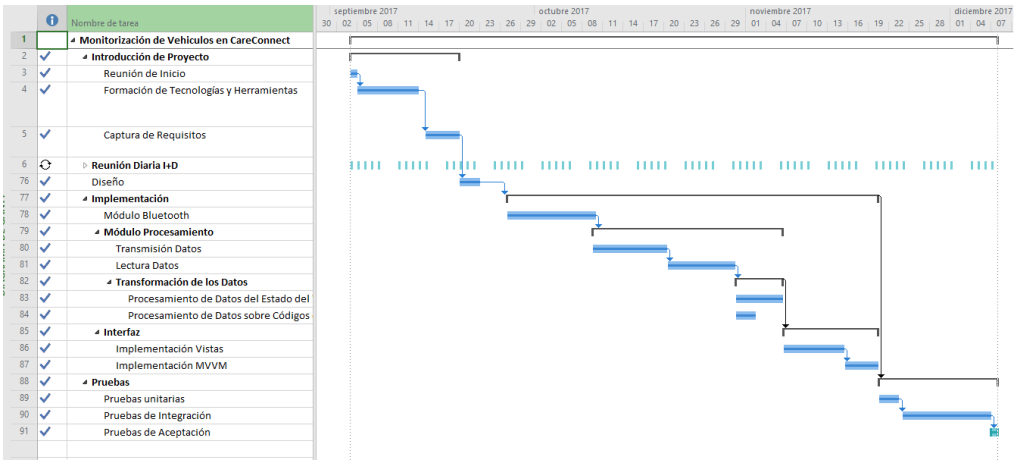

Figura 1. Diagrama de Gantt del plan de trabajo

2.- Tecnologías y herramientas

En este apartado se describen aquellas tecnologías y herramientas que se han utilizado durante el desarrollo del proyecto.

Tecnologías

Android

- Android [6] es un sistema operativo orientado a dispositivos móviles de pantalla táctil basado en el núcleo de Linux, siendo además multiplataforma.

C#

- C# o C Sharp [7] es un lenguaje de programación orientado a objetos desarrollado por Microsoft dentro de su plataforma .NET. Deriva de C, compartiendo grandes similitudes con Java.

SQLite

- Sistema de gestión de bases de datos relacional basado en las propiedades ACID, que se caracteriza por trabajar de forma embebida, siendo ideal para sistemas móviles.

Sockets

- Un Socket [8] se define como un canal de comunicación que permite el intercambio de datos. En este caso, se usarán BluetoothSocket, similar a un Socket de tipo TCP, que realiza el intercambio de datos mediante InputStream y OutputStream.

Bluetooth

- Especificación industrial que posibilita la transmisión de datos entre dispositivos a través de un enlace por radiofrecuencia.

On Board Diagnostics

- Sistema de diagnóstico a bordo, que ofrece la posibilidad de monitorizar el motor y otros sistemas del vehículo.

Herramientas

Visual Studio

- Visual Studio [9] es un entorno de desarrollo integrado para sistemas operativos Windows, que trabaja con la tecnología .NET.

Xamarin

- Xamarin [10] es una plataforma destinada al desarrollo de aplicaciones móviles multiplataforma en C# mediante código compartido.

Microsoft Project

- Microsoft Project [11] es un software dirigido a la administración de proyectos.

Team Foundation Server

- Team Foundation Server (TFS) [12], es un conjunto de tecnologías de colaboración que permiten coordinar el trabajo de cara al desarrollo de un producto, a través de, entre otras funciones, el control de código fuente.

MagicDraw UML

- MagicDraw UML [13] es una herramienta CASE desarrollado por No Magic destinada al modelado de datos.

ELM327

- El dispositivo ELM327 [14] es un controlador producido por ELM Electronics destinado al diagnóstico a bordo de vehículos, basándose en el estándar OBDII.

SonarQube

- SonarQube [15] es una herramienta para el análisis y la evaluación de código fuente.

2.1.- On Board Diagnostics

OBD corresponde con las siglas de *On Board Diagnostics*, y representa el sistema de diagnóstico a bordo en vehículos.

Esta tecnología surge a finales de los años 70 de la mano de la marca alemana Volkswagen, tomando ya unas dimensiones considerables a finales de los 80, cuando *California Air Resources Board*, promovió que todos los vehículos estadounidenses a gasolina debían contar con un sistema capaz de controlar las emisiones de los vehículos de cara a reducir la contaminación del aire. Paralelamente, la *Society of Automotive Engineers* (SAE) promovería la primera estandarización de este tipo de tecnología, creando el protocolo denominado OBD I. Años más tarde, se renovaría esta estandarización dando lugar al protocolo OBDII. Esta segunda reglamentación, incluía una mejor regulación de los parámetros, así como la gestión sobre un mayor número de parámetros, o nuevas funcionalidades como la detección y almacenamiento de fallos.

Los estándares OBD llegarían a Europa en el año 2001, bajo la denominación de EOBD, una adaptación del OBDII americano. En el año 2003, EOBD sería de obligatorio cumplimiento para todo vehículo diésel desarrollado en la Unión Europea, mientras que en 2008 Estados Unidos implantaría la misma obligación para todos los vehículos.

De esta manera, los vehículos disponen de un puerto OBD a través del cual se puede recoger toda esta información de la ECU. Así, si bien el propio sistema de

diagnóstico estaba regulado por OBDII/EOBD, la forma de comunicación no cuenta con esa regulación, dependiendo de cada fabricante, y dando lugar a distintos estándares de comunicación.

- SAE J1850 PWM
- SAE J1850 VPW
- ISO9141-2
- ISO14230-4 KWP
- ISO 15765-4 CAN
- SAE J1939 CAN**.** ***Para vehículos pesados***

Los fabricantes como norma general sitúan este puerto OBD bajo el volante del vehículo, o bien en la caja de fusibles, como se puede comprobar en la Figura 2. Sin embargo, pueden darse casos en los que este puerto se encuentre en zonas como puede ser junto al cenicero o incluso en el asiento del copiloto.

Figura 2. Puerto OBD

Este puerto dispone de 16 pines de conexión, a través de los cuáles se realiza el proceso de comunicación con la ECU del vehículo mediante el dispositivo OBDII que se conecte en dicho puerto. Entre estos 16 pines, cada fabricante utiliza por defecto unos pocos para habilitar la conexión, de tal manera que, en función de los protocolos anteriormente mencionados, ligados a cada fabricante, estarán habilitados unos pines u otros. Se puede comprobar qué pines están habilitados para la comunicación por ejemplo visualizando el puerto y comprobando cuáles de ellos disponen de cobre en su toma para hacer contacto.

Igualmente, existe una serie de dispositivos que implementan OBDII y permiten sacar provecho de esta funcionalidad. Estos dispositivos serán los que se conecten al puerto OBD, pudiendo acceder a estos de diversas maneras, como a través de puerto serie (RS232), USB, WiFi o Bluetooth.

En la actualidad, la mayoría de los dispositivos que podemos encontrar en el mercado, están basados en el chip ELM327 desarrollado por ELM Electronics, capaz de atender a todos los protocolos de comunicación anteriormente enumerados, y con distintas velocidades de transmisión, ofreciendo incluso la posibilidad de utilizar protocolos configurables por el usuario. Sin embargo, muchas de estas unidades son meras copias de dispositivos basados en este chip, dando lugar a que, con su uso, los resultados no se ajusten de manera adecuada a la realidad, o en casos más graves, dañando incluso la ECU de los vehículos. En la Figura 3 se puede ver uno de estos dispositivos, utilizados para el desarrollo del proyecto.

Figura 3. Dispositivo OBDII basado en el chip ELM327

La comunicación con el ELM327 se basa en el envío de comandos y la lectura de las respuestas recogidas por el OBDII. Las instrucciones a las que responde el ELM327 responden a dos categorías:

- Comandos AT

Los comandos AT, conocidos como comandos *Hayes* [16], es un lenguaje desarrollado por *Hayes Communications*, cuyo fin original era configurar módems, estandarizándose finalmente a un gran listado de dispositivos de comunicación.

De cara a los dispositivos OBD, además, existen una serie de comandos AT específicos que permiten que la capacidad de configuración sobre este dispositivo sea total.

- Comandos OBD

Los comandos OBD se consideran como los comandos de instrucción utilizados sobre el ELM327 para la consulta de datos o la ejecución de instrucciones sobre el ordenador de a bordo de nuestro vehículo. Todo aquel comando que se envíe al ELM327 que no comience con las siglas AT, será considerado OBD, sin embargo, existen unos comandos predefinidos que actúan sobre nuestro vehículo, los cuáles se describen en la Tabla 2.

Comando	Función
01	Muestra los valores de los parámetros disponibles en tiempo real.
02	Muestra el "*Diagnostic Trouble Code"* (DTC), o código de fallo, relacionado a un evento producido en el vehículo.
03	Muestra los DTC almacenados.
04	Borra los datos almacenados, incluidos DTC.
05	Muestra los resultados de las pruebas sobre los sensores de oxígeno, para vehículos sin comunicación CAN-bus.
06	Muestra los resultados de las pruebas sobre los sensores de oxígeno, para vehículos con comunicación CAN-bus.
07	Muestra los DTC generados durante el último ciclo de conducción o actual.
08	Operación de control sobre componentes del sistema de a bordo.
09	Información general del vehículo.
0A	Muestra DTCs permanentes.

Tabla 2. Instrucciones OBD reconocidas por ELM327

Así mismo, existen los códigos denominados "*Parameter ID"* (PID) [17], que corresponden con códigos en formato hexadecimal que hacen referencia a cierto valor o componente del vehículo. Estos PIDs son utilizados en los modos de consulta para poder conocer, bajo distintas circunstancias, el valor de un parámetro deseado.

Respecto a estos PIDs, cabe destacar que no es de obligatorio cumplimiento que los fabricantes den cobertura a todos los disponibles, de tal manera que un fabricante puede tener disponible ciertos PIDs, y dándose el caso que el PID que esté operativo en un vehículo de una marca, puede no estarlo en otra.

De igual manera, podemos encontrar los códigos DTC. Estos códigos corresponden con códigos en respuesta a los diagnósticos sobre el estado del vehículo respecto a posibles fallos. Estos códigos, representan un fallo detectado en el vehículo, bajo un formato de un carácter seguido de cuatro dígitos.

El primer carácter representa el sistema del vehículo en el que se ha detectado el fallo, contemplando las opciones de la Tabla 3.

Carácter	Sistema del vehículo afectado
"P"	Sistema de propulsión
"B"	Carrocería o habitáculo
"C"	Tren de rodaje, sistema de frenos o suspensión
"U"	Red de comunicaciones

Tabla 3. Sistema afectado por código DTC

El primer dígito responde a la naturaleza del propio código de fallo, Estos códigos, pueden ser genéricos o propios de cada fabricante, representando este dígito de que tipo es. Los valores, mostrados en las Tablas 4 y 5, varían en función a qué sistema del vehículo responde.

Dígito	Origen del código
0	Definido por el estándar SAE
1	Definido por el fabricante
2	Definido por el estándar SAE
3	Definido bajo acuerdo del SAE y el fabricante

Tabla 4. Origen del código DTC para códigos tipo P

Dígito	Origen del código
0	Definido por el estándar SAE
1	Definido por el fabricante
2	Definido por el fabricante
3	Reservado

Tabla 5. Origen del código DTC para códigos tipo B, C y U

Los tres dígitos restantes hasta completar el código, forman el resto del identificador del mismo.

3.- Especificación de requisitos

En este apartado se describen los requisitos funcionales y no funcionales del sistema a desarrollar.

3.1.- Captura de requisitos

La captura de requisitos del sistema se realizó mediante entrevistas con los propios directivos de la empresa y el jefe del departamento de desarrollo. Tras esto, durante las reuniones diarias y muestras de los avances, se fueron matizando e incluyendo nuevos requisitos. Durante los primeros días del proyecto, se realizó un *mock-up* de la aplicación que serviría de apoyo a la comunicación con los *stakeholders*.

De este proceso, los requisitos funcionales y no funcionales definidos fueron los que se relacionan en los apartados 3.2 y 3.3.

3.2.- Requisitos funcionales

Los requisitos funcionales son propiedades asociadas a funciones del sistema, describiendo el comportamiento de estas.

Los requisitos obtenidos para el desarrollo de este proyecto pueden categorizarse en dos puntos principalmente. El primero de ellos en referencia al uso de la tecnología Bluetooth para establecer la comunicación con el dispositivo OBDII, siendo el segundo referente a la propia comunicación con el dispositivo, el procesamiento y presentación de los datos. Estos se recogen en la Tabla 6.

Identificador	Descripción
Requisitos relacionados con la conexión Bluetooth	
RF-001	El sistema debe ser capaz de comprobar el estado actual del Bluetooth en el *Smartphone*, y activarlo siempre que sea necesario para el uso del módulo a desarrollar.
RF-002	El sistema debe ser capaz de lanzar un descubrimiento para localizar los dispositivos OBDII próximos al terminal.
RF-003	El sistema debe ser capaz de establecer la comunicación con un dispositivo OBDII seleccionado por el usuario.
RF-004	El sistema debe ser capaz de poner fin a dicha conexión entre terminal y dispositivo OBDII.
Requisitos relacionados con la comunicación con el dispositivo OBDII	
RF-005	El sistema debe ser capaz de inicializar el dispositivo OBDII, estableciendo una configuración de transmisión y lectura de datos estándar independientemente del vehículo en el que se esté utilizando.
RF-006	El sistema ofrecerá dos modos de consulta de información; información en tiempo real sobre el estado del vehículo, e información sobre los posibles fallos detectados por la centralita.
RF-007	El sistema debe ser capaz de detectar cualquier error que se pueda producir durante la comunicación con el OBDII y en caso de producirse, detener las consultas en ejecución.
RF-008	El sistema debe de ser capaz de solicitar parámetros sobre el funcionamiento del vehículo en tiempo real, procesarlos y mostrarlos al usuario.
RT-009	El sistema debe permitir al usuario seleccionar los parámetros que desea observar en cada momento.
RF-010	La configuración sobre la visibilidad de los parámetros debe mantenerse pese a que la aplicación se cierre.
RF-011	El sistema debe permitir al usuario almacenar los datos que está visualizando en una base de datos interna.
RF-012	El sistema debe ser capaz de solicitar los DTC registrados en el vehículo al OBDII, procesarlos, y presentar los resultados al usuario.
RF-013	El sistema debe permitir enviar los diagnósticos realizados vía email a los talleres de confianza del usuario.

Tabla 6. Requisitos funcionales de la aplicación

3.3.- Requisitos no funcionales

Los requisitos no funcionales, recogidos en la Tabla 7, establecen una serie de líneas a seguir sobre el rendimiento, la disponibilidad y la seguridad.

Identificador	Descripción
RNF-001	El sistema debe ser fácilmente mantenible.
RNF-002	Las interfaces de usuario deben ser intuitivas y de fácil aprendizaje para los usuarios.

Tabla 7. Requisitos no funcionales de la aplicación

Respecto a la seguridad, se ha considerado oportuno que, para usar la aplicación, sea necesario estar registrado en el sistema *CarEConnect* y autentificarse mediante una combinación de correo electrónico y contraseña, pero este requisito es ajeno a este proyecto de trabajo de fin de grado.

4.- Diseño del sistema

En base a los requisitos del sistema, se diseñan los modelos software que forjan su implementación.

4.1.- Diseño arquitectónico

El diseño arquitectónico hace referencia a la arquitectura que se va a establecer para desarrollar el sistema, indicando además a los distintos patrones que se utilizarán en la implementación del mismo, en busca de satisfacer los requisitos establecidos además de facilitar la comprensión del código de la aplicación y favorecer la mantenibilidad.

En este caso, se ha optado por un diseño en tres capas, estructurado en capa de presentación, negocio y datos.

- Presentación: presenta las distintas interfaces a través de las cuáles el usuario interactúa con la aplicación.

- Negocio: capa que actúa de intermediario entre la capa de Presentación y la capa de Acceso a Datos. En esta se encuentra la lógica que implementa la funcionalidad de la aplicación, gestionando así todos los procesos que la componen, haciendo uso, cuando es requerido, de los datos obtenidos en la capa de acceso a datos.

- Acceso a Datos: capa encargada de obtener y transferir los datos, en este caso con los elementos externos que interactúan con el sistema y en los que se basa el mismo. Para ser concretos, ejecuta las instrucciones necesarias para inicializar el dispositivo OBDII y solicitar y recoger los distintos datos requeridos, así como trabajar con la base de datos para almacenar los datos cuando sea indicado.

Para este proyecto, además, se ha utilizado el patrón Modelo-Vista-Modelo de Vista (MVVM) [18], un patrón de arquitectura software que promueve el desacoplar la lógica de la aplicación de la propia interfaz. Se compone (ver Figura 4) de:

Modelo:

- Referente a la lógica de la aplicación.

Vista:

- Compone la parte gráfica de la aplicación, haciendo referencia a la o las interfaces de usuario en las que se va a aplicar dicho patrón.

Modelo de Vista:

- Componente intermedio entre el modelo y la vista. Contiene la lógica de presentación.

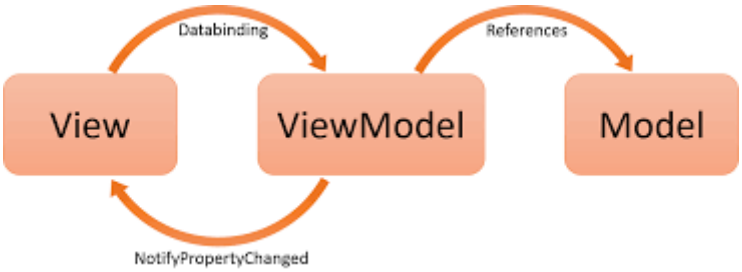

Figura 4. Esquema patrón Modelo-Vista-Modelo de Vista

Además, MVVM contempla distintos modos de funcionamiento, ofreciendo así distintos modos de comunicación entre el modelo y la vista:

- *OneWay*: modo por defecto, con enlace únicamente de lectura; los cambios en el *ViewModel* (VM) se ven reflejados en la Vista, pero los cambios realizados en la Vista no se propagarán al VM.

- *OneWayToSource*: el enlace es únicamente de escritura, los cambios en el VM no se ven reflejados en la Vista, pero los cambios realizados en la Vista si se propagarán al VM.

- *TwoWay*: propone un enlace bidireccional, combinando así el comportamiento de los dos primeros modos.

La aplicación de este patrón da como resultado el desacoplamiento de la aplicación en cuanto a su lógica y su interfaz, siendo el elemento que relaciona a ambos el objeto que compone el Modelo de Vista.

Este patrón puede recordarnos al patrón Modelo-Vista-Controlador (MVC), arquitectura utilizada frecuentemente para separar los datos de una aplicación, la lógica y la interfaz de usuario en tres componentes distintos. Una de las principales diferencias entre ambos, es el modo en el que se gestionan los cambios. En caso del MVC, los cambios realizados en la lógica deben ser aplicados a través de una gestión previa del controlador, mientras que en MVVM estos cambios son aplicados de forma automática mediante la el Modelo de Vista.

En este proyecto, MVVM se ha utilizado para presentar los datos en tiempo real, utilizando este patrón para registrar los cambios en los valores de los parámetros visualizados y actualizarlos en la interfaz.

4.2.- Diseño detallado

El diseño detallado de este proyecto (ver Figura 5) se compone de:

- El módulo que hace referencia a los dispositivos móviles, en los cuáles se instalará la *apk*, incluyendo la base de datos.

- El dispositivo OBDII, que se encarga de solicitar y recoger datos del vehículo. Pueden almacenarse los datos en una pequeña memoria interna de la que dispone, pero en esta ocasión se utiliza como mero transmisor.

- El servidor de CEINOR dónde se encuentran los *Web Services* a los que accede la aplicación móvil. Este módulo no ha sido desarrollado dentro de este trabajo de fin de grado.

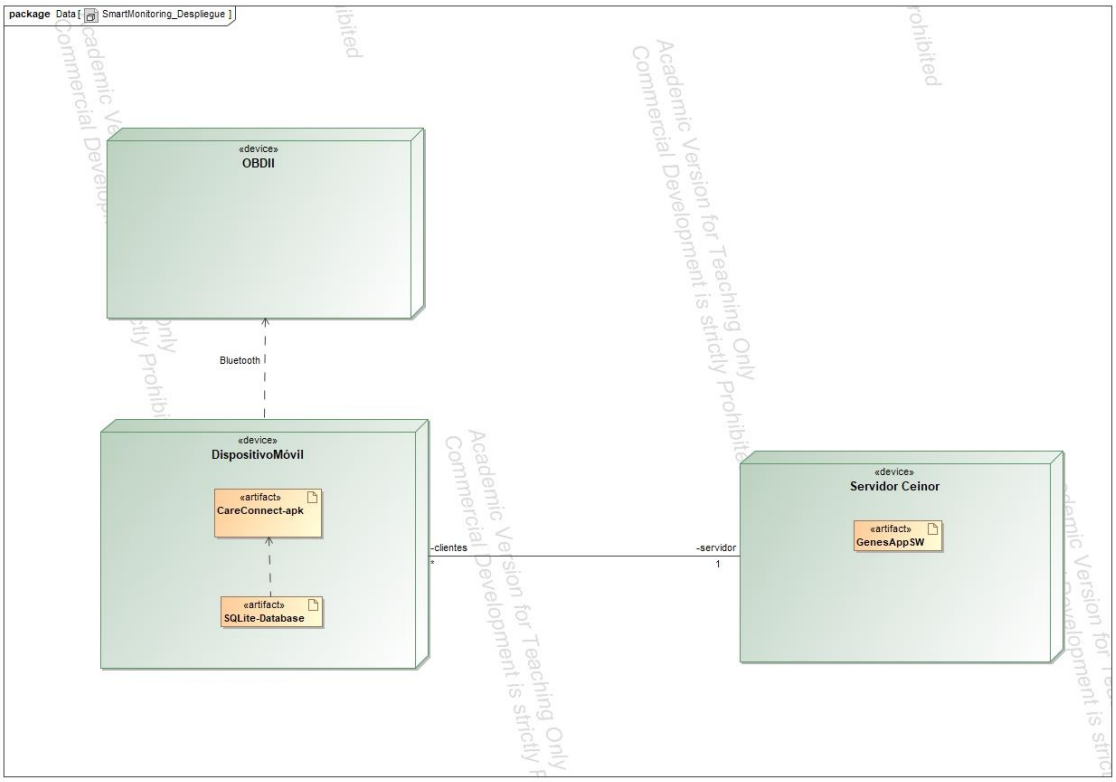

Figura 5. Diagrama de despliegue

5.- Implementación

La fase de implementación hace referencia a la realización de la especificación técnica, modelada en la fase previa de diseño y con el punto de partida de los requisitos del sistema.

5.1.- Estructura del proyecto

Como se ha explicado en el capítulo anterior, se ha aplicado una arquitectura de tres capas, acompañada además del patrón MVVM. Junto a esto, de cara a la implementación, se ha de tener en cuenta el cómo se desarrolla un proyecto multiplataforma en Xamarin. Un proyecto de esta categoría se desarrolla bajo la configuración Xamarin.Forms, en la cual se encuentra un proyecto común dónde se aloja todo el código que no depende de un lenguaje nativo al que se pueda destinar el proyecto (Android, iOS o Windows), y junto a este, los proyectos propios de cada plataforma, dónde se aloja el código dependiente de cada las plataformas.

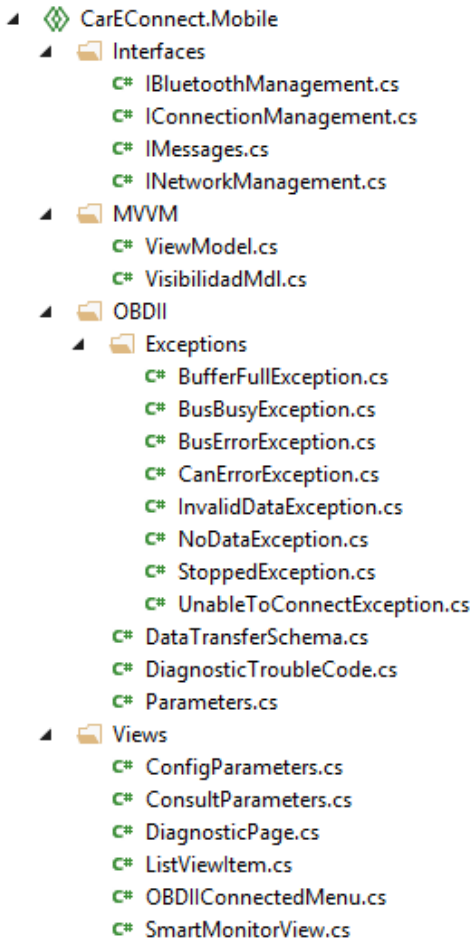

Figura 6. Organización de paquetes del proyecto Xamarin común

Así, en la Figura 6, se muestra la disposición del proyecto CarEConnect.Mobile, dónde se encuentra todo el código compartido, es decir, no contiene ninguna dependencia con ninguna API en concreto.

En este primer proyecto se encuentran los siguientes elementos:

- Views: en esta carpeta se alojan todas las vistas de la aplicación, lo que formaría la capa de presentación.

- Interfaces: en esta carpeta se alojan las interfaces de la capa de negocio, cuyas implementaciones son dependientes de cada plataforma y, por tanto, se encuentran en los proyectos específicos.

- MVVM: En esta carpeta se encuentra la implementación del objeto que cumple la función de "Modelo de vista" explicado en el capítulo de diseño. También se incluye la clase *VisibilidadMdl* en la que se almacenan las variables que indican qué parámetros están visibles y cuáles no.

- OBDII: En esta carpeta se encuentran los objetos de dominio utilizados para la interacción con el dispositivo OBDII, como pueden ser, los parámetros enumerados que se van a utilizar, el formato en el que se muestran los códigos de falla del modo 03 (DTC), el formato de transferencia de información, o las excepciones generadas por el OBDII.

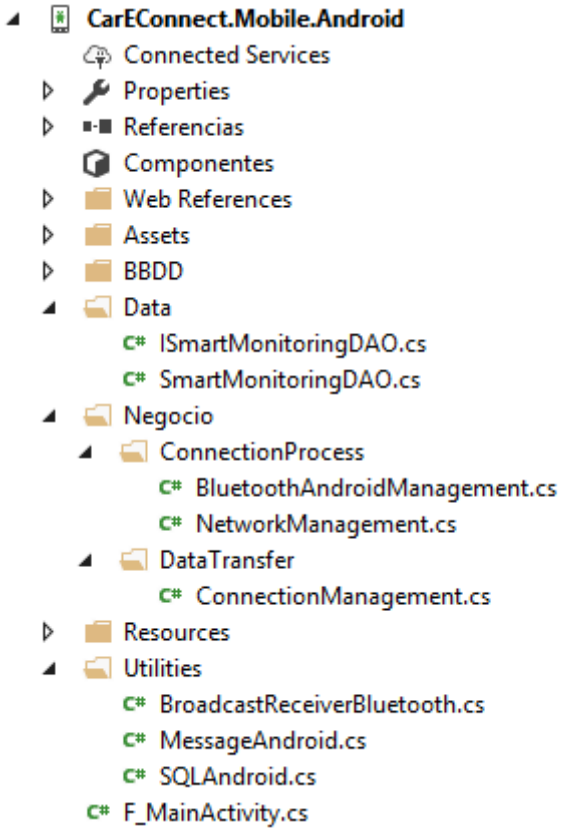

Figura 7. Organización de paquetes del proyecto Xamarin Android

En la Figura 7 se ofrece la disposición del proyecto específico para la plataforma Android. En este proyecto se encuentra:

- Negocio: en esta carpeta se encuentran las implementaciones de las interfaces de negocio, las cuales se categorizan en *ConnectionProcess* dónde se gestiona todo aquello relacionado con la conexión a través de la tecnología Bluetooth y funcionalidades específicas como el uso de la conexión a internet, y una segunda

carpeta *DataTransfer* donde se gestiona la solicitud y procesamiento de información recibida desde la ECU a través del OBDII.

- Data: interfaz e implementación del acceso a datos.

- BBDD: interfaz de uso de la base de datos y el mapeado de los parámetros con los que se trabaja en forma de tablas (ver Figura 8).

- Utilities: En esta carpeta se almacena la clase *BroadCastReceiverBluetooth*. Esta clase implementa un componente propio de Android como es el *BroadcastReceiver*. Esto, permite registrar eventos, mediante la notificación del propio sistema de Android. En este caso, registramos eventos como puede ser el inicio o fin del proceso de descubrimiento, el descubrir un dispositivo, o el intento de emparejamiento entre dispositivos.

Cabe destacar que, al contrario que el resto de las interfaces de negocio, la interfaz *ISQLite*, que ofrece los métodos necesarios para gestionar la creación y la conexión a la base de datos SQLite que se utiliza en esta aplicación, se encuentra en el proyecto nativo de Android en lugar de en el proyecto común. Esto se debe a que la configuración establecida para el proyecto global *CarEConnect* no permite añadir referencias en el proyecto común, lo que imposibilita el añadir las librerías necesarias de SQLite a dicho proyecto, obligando a trasladar esa interfaz al proyecto de Android junto a su implementación, la clase *SQLAndroid*.

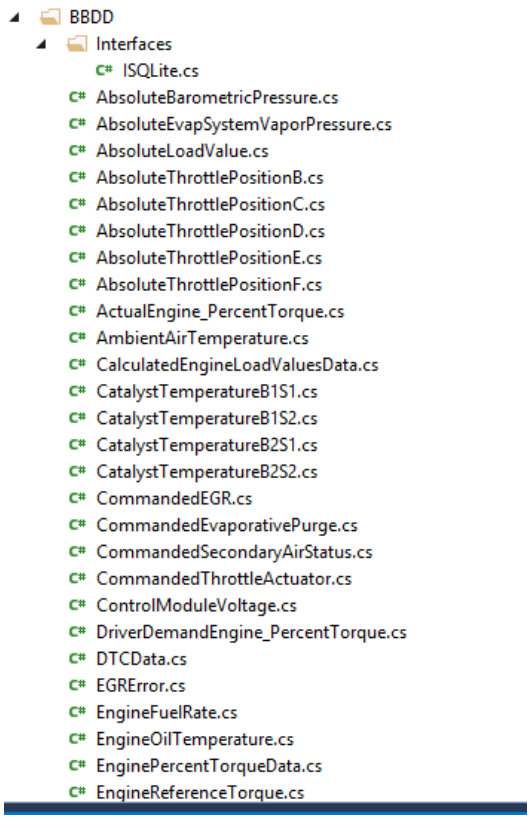

Figura 8. Organización de carpeta de Base de Datos de Xamarin Android

5.2.- Funcionamiento de la aplicación.

En este apartado se comentan de forma breve los aspectos esenciales en los que se basa este desarrollo. Si bien todos los detalles son importantes, se desea al menos contextualizar los pilares básicos de la aplicación.

5.2.1.- Interacción Bluetooth

Una vez se active el módulo *Smart Monitoring*, lo primero de lo que se debe encargar la aplicación es que el Bluetooth esté operativo para que posteriormente pueda ser utilizado de cara a descubrir dispositivos y una posible conexión.

Esto, se realiza a través del objeto *BluetoothAdapter* (ver Figura 9), que representa el dispositivo Bluetooth local correspondiente con el propio Smartphone.

```
static BluetoothAdapter bluetoothAdapter;
0 referencias | Practicas2, Hace 23 días | 1 autor, 1 cambio
public BluetoothAndroidManagement()
{
    bluetoothAdapter = BluetoothAdapter.DefaultAdapter;
}
```

Figura 9. Manejo de la tecnología Bluetooth local del dispositivo móvil

Será este objeto el que nos proporcione los métodos necesarios para realizar todas las acciones necesarias en referencia a la tecnología Bluetooth, siendo capaz de:

- Indicar si el Bluetooth está activado o desactivado.
- Activar o desactivar el Bluetooth.
- Iniciar y finalizar el descubrimiento de dispositivos Bluetooth.
- Indicar si se está realizando un proceso de descubrimiento.

Por tanto, en primer lugar, se verifica que el Bluetooth está activo en el dispositivo (en caso contrario se activa), para dar lugar a un proceso de descubrimiento.

En este descubrimiento, se detectan dispositivos Bluetooth, en forma de *BluetoothDevice*. Esta detección se realiza a través de la clase *BroadCastReceiverBluetooth* mencionada en el apartado 5.1. En esta clase, se registran las diversas acciones que puede producir el Bluetooth, como puede ser el cambio en el estado del Bluetooth, el inicio o fin del descubrimiento, la detección de un dispositivo, o el intento de emparejamiento con un dispositivo, entre otros.

De esta manera, para cada una de las acciones registradas, se puede implementar el proceso a seguir para cada una de ellas.

Así, durante el proceso de descubrimiento, a través de este objeto se van registrando dispositivos Bluetooth, verificando que son dispositivos OBDII y no se encuentran direcciones MAC (identificadores del dispositivo) duplicadas. Así, se almacenan en una lista para mostrarlas al usuario.

Una vez finalizado el descubrimiento, y sean mostrados los dispositivos al usuario, podrá seleccionar uno de ellos para dar paso a la conexión.

El proceso de conexión puede venir precedido de un proceso de emparejamiento o vinculación con el dispositivo Bluetooth, que se realiza a través del propio dispositivo (ver Figura 10).

```
/**
* Método para vincular el dispositivo deseado
* **/
1 referencia | Practicas2, Hace 17 días | 1 autor, 6 cambios
public async Task bondedDevice(BluetoothDevice device)
{
    if (bluetoothAdapter.IsDiscovering)
    {
        CancelDiscovery();
    }
    bool res = device.CreateBond();
    await Task.Delay(BONDED_TIME);
}
```

Figura 10. Proceso de emparejamiento con el dispositivo Bluetooth

Una vez el dispositivo móvil se encuentra asociado con un dispositivo OBDII, se genera un *BluetoothSocket* para establecer la conexión entre ambos.

El Bluetooth Socket es una interfaz muy similar a los sockets de tipo TCP. De forma general, pueden encontrarse dos tipos, de tipo servidor (*BluetoothServerSocket*) y el tipo cliente, o *BluetoothSocket*.

El tipo más común de Bluetooth Socket es el RFCOMM [19], el cual es soportado por Android. RFCOMM es un conjunto de protocolos de transporte que emulan una conexión de puerto seria RS-232.

La conexión se realiza en un proceso de dos pasos (ver Figura 11). En primer lugar, se crea un socket RFCOMM mediante el denominado UUID (Universally Unique IDentifier), código de identificación de los que disponen los dispositivos Bluetooth. Una vez se crea dicho Socket, se procede a conectarlo. En caso de que ambos procesos se realicen de forma exitosa, ambos dispositivos estarán ya conectados.

```
try
{
   BluetoothSocket tryConnect = device.CreateRfcommSocketToServiceRecord(uuid.Uuid);

    tryConnect.Connect();
    connected = true;
}
catch (IOException ex)
{
    throw ex;
}
```

Figura 11. Representación de cómo crear y conectar un BluetoothSocket

5.2.2.- Comunicación con OBDII

Establecida la conexión con el OBDII, la comunicación se realiza a través del propio *BluetoothSocket* generado en el proceso de conexión.

Así, el *BluetoothSocket* dispone de dos objetos, *OutputStream* e *InputStream* que se utilizan como modo de transmisión. En caso del *OutputStream*, sirve como objeto de escritura (ver Figura 12), siendo el *InputStream* el objeto destinado a la lectura de cara a recibir datos. Para esta implementación en concreto, entre el proceso de escritura mediante el *OutputStream* y el proceso de lectura de la respuesta a través del *InputStream*, se han establecido unos tiempos de espera que concedan al OBDII el margen temporal suficiente para gestionar toda la transmisión.

Además, en la Figura 13 se puede comprobar que la lectura se realice byte a byte, dándose por concluida en el momento que se reciba el carácter '>'. Esto se debe a que este carácter es el utilizado por el ELM327 para determinar el fin de sus respuestas.

```
byte[] cmd;

    cmd = Encoding.ASCII.GetBytes(command + CARRIAGE_RETURN);

i_Socket.OutputStream.Write(cmd, 0, cmd.Length);
```

Figura 12. Representación formato envío de datos por BluetoothSocket

```
string data = "";
 try

    {

        System.Text.StringBuilder builder = new System.Text.StringBuilder();
        char c;
        byte a;

        char[] chr = new char[100];
        byte[] bytes = new byte[20];

        while (true)
        {

            a = (byte)i_Socket.InputStream.ReadByte();

            c = (char)a;
            if (c.Equals('>'))
            {

                break;

            }
            builder.Append(c);
        }


        data = builder.ToString();

        i_Socket.InputStream.Flush();


    }
    catch (System.Exception e)
    {
        throw e;

    }
```

Figura 13. Representación formato lectura de datos de respuesta por BluetoothSocket

5.2.3.- Modelo-Vista-Modelo de Vista

Como se mencionaba en el capítulo de Diseño, el patrón MVVM ha sido utilizado de cara a refrescar de forma automática la interfaz que presenta los parámetros del vehículo en tiempo real.

Para satisfacer dicho comportamiento, se requiere el uso de diversos elementos que faciliten el flujo de información en los enlaces entre el Modelo-Modelo de Vista y Modelo de Vista-Vista. Si bien entre el Modelo y el Modelo de Vista mediante referencias directas ya se establece el nivel de "comunicación" necesario (ver Figura 15), para que el Modelo de Vista y la Vista interactúen, debe recurrirse al uso de los llamados "*DataBinding*" y "*NotifyPropertyChanged*".

El primero de ellos es un concepto utilizado en la vista, cuya función es enlazar las propiedades de dos objetos distintos, de tal manera que, al aplicarlo en una propiedad sobre otra de otro objeto diferente, la primera de ellas asumirá el valor de la

segunda. Cabe destacar que, para hacer uso de estos elementos, hay que indicar también mediante la propiedad *BindingContext* sobre qué objeto se van a establecer los mismos (ver Figura 14).

```
BindingContext = i_ViewModel;
i_RPM = new Label();
i_RPM.Text = "RPM";
i_RPMResult = new Label();
i_RPMResult.SetBinding(Label.TextProperty, "Rpm");
i_RPMUds = new Label();
i_RPMUds.Text = "rpm";
```

Figura 14. Representación de cómo establecer un DataBinding.

```
try
{
    DataTransferSchema dr = await i_SmartMonitoringDAO.ConsultParametersAsync(Parameters.PID.RPM);
    string value = i_SmartMonitoringDAO.readRPM(dr);
    i_ViewModel.Rpm = value;
}
catch (InvalidDataException)
{
    i_ViewModel.Rpm = NO_DATA;
}
```

Figura 15. Ejemplo de vinculación de los datos desde la capa de gestión al objeto intermediario

En caso de *NotifyPropertyChanged*, es un concepto basado en eventos que permite notificar el cambio sobre una variable de forma automática. Este concepto, se desarrolla a través de la implementación por parte de una clase de la interfaz *INotifyPropertyChanged*, e incluyendo un objeto en concepto de evento *PropertyChangedEventHandler* (ver Figura 16). Así, podremos "configurar" nuestras variables de tal manera que cuando se modifiquen, se lance este evento, notificando así su cambio (ver Figura 17). Esto, unido al *DataBinding* anteriormente explicado, permite que aquella propiedad que esté asociada a nuestra variable del "Modelo de Vista", reciba su valor de forma automática cada vez que ésta sea modificada.

```
public event PropertyChangedEventHandler PropertyChanged;

99+ references | Practicas2, Hace 23 días | 1 autor, 1 cambio
private void OnPropertyChanged(string propertyName)
{
    PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}
```

Figura 16. Representación de la gestión del cambio de valor en propiedades mediante eventos

```
2 referencias | Practicas2, Hace 18 días | 1 autor, 2 cambios
public string Rpm
{
    get
    {
        return i_RPM;
    }

    set
    {
        i_RPM = value;
        OnPropertyChanged("Rpm");
    }
}
```

Figura 17. Modo de uso del registro de cambio de eventos

5.3.- Interfaces

En base a la implementación desarrollada, se ofrece a continuación una muestra de las principales vistas que componen la aplicación.

Ajeno al desarrollo de este trabajo de fin de grado, en primer lugar, tenemos la interfaz principal de *CarEConnect* (ver Figura 18), dónde, entre sus diversas opciones, encontramos *Smart Monitoring*.

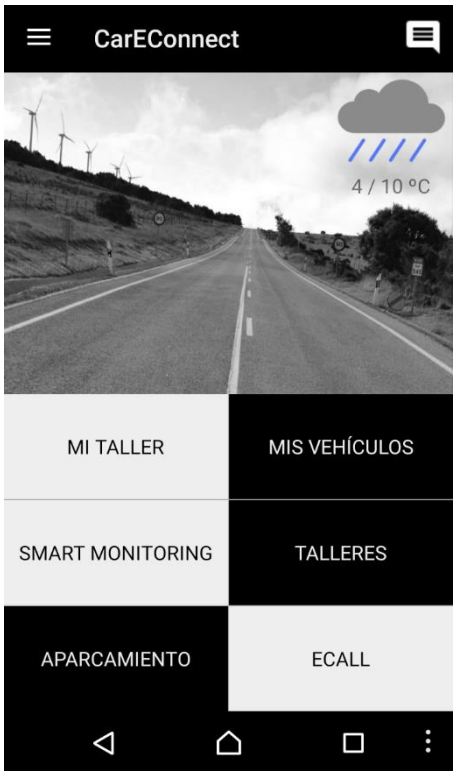

Figura 18. Menú CarEConnect

Al seleccionar esta opción en el menú, pueden darse tres casos:

- El dispositivo no tiene Bluetooth, se notifica al usuario y la aplicación regresa a esta vista.
- El dispositivo tiene el Bluetooth desactivado, activándose de forma automática y notificando de ello al usuario, para después comenzar la búsqueda (descubrimiento) de dispositivos Bluetooth.
- El dispositivo tiene el Bluetooth activado, dando paso al descubrimiento.

Tras finalizar el descubrimiento, pueden darse dos situaciones:

- Situación favorable, en la que se encuentren uno o más dispositivos OBDII (ver Figura 19), mostrándoselos al usuario en forma de lista.
- Situación desfavorable, en la que no se encuentre ningún dispositivo OBDII, notificándoselo al usuario y dándole la posibilidad de reintentar la búsqueda.

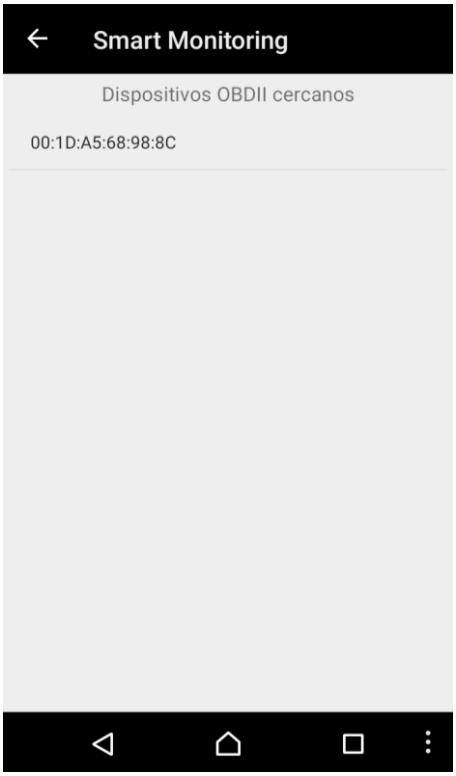

Figura 19. Vista del listado con dispositivos OBDII

En caso favorable, es decir, que se encuentren dispositivos, el usuario puede seleccionar cualquiera de ellos para establecer una conexión. De esta manera, una vez seleccionado se iniciará el proceso de conexión, incluyendo el emparejamiento Bluetooth previo si fuera necesario. En esta ocasión, pueden darse tres situaciones:

- La conexión se realiza con éxito, dando paso a la Figura 20.
- La conexión se realiza con éxito, pero el proceso de inicialización del OBDII posterior no completa de forma exitosa. Este proceso depende del estado del

vehículo; si no está al menos el contacto dado, no se completa. Esto se notifica al usuario, indicando que para continuar con el proceso debe tener al menos el contacto activado y se le ofrece la opción de reintentar el proceso de inicialización.

- La conexión no se establece, notificándoselo al usuario.

Figura 20. Menú con opciones de consulta disponibles

Una vez conectados al OBDII, la aplicación muestra un menú con los modos de utilización que se ofrecen al usuario, para interactuar con su vehículo a través de dicho OBDII.

- Consultar en Tiempo Real (ver Figura 21)
- Diagnóstico de Fallos (ver Figuras 23 y 24)

Si seleccionamos la opción “Consultar en Tiempo Real”, tras mostrar durante un breve periodo de tiempo una pantalla de carga, se muestran los parámetros en tiempo real del estado del vehículo.

Cada uno de estos datos está inmerso en un *BoxView* en el que se incluye el nombre del parámetro, sus unidades, y el valor de dicho parámetro en tiempo real. Estos parámetros vienen dados por los PIDs que se mencionaban en el apartado 2.1, y tal y cómo se comentaba, puede darse la situación de que el parámetro deseado no pueda ser proporcionado por el vehículo. En ese caso, se mostrará para dicho parámetro “NO DATA”.

Figura 21. Vista de la función Consultar en Tiempo Real

La aplicación, permite configurar qué parámetros se desean visualizar en cada momento. Para ello, en esta vista se dispone dentro de la *toolbar* de la opción "Configurar Parámetros", que nos lleva a la vista correspondiente a la Figura 22.

De igual manera, también ofrece una opción que permite almacenar los datos en visualización en la base de datos destinada a esta función, notificando la activación o desactivación de este almacenamiento al usuario.

Figura 22. Vista de la configuración de los parámetros visibles

En esta vista, se pone a disposición del usuario todos los parámetros que la aplicación es capaz de mostrar. Cabe destacar que no se encuentran todos los parámetros que la ECU tiene a su disposición, sino que se realizó un sondeo con los parámetros que podrían ser de mayor interés para los usuarios.

De esta manera, acompañado cada parámetro de un *switch*, se permite la activación o desactivación de cada uno de ellos. Una vez los parámetros estén a gusto del usuario, simplemente retrocediendo a la vista anterior, la configuración será guardada (notificándose al usuario de ello), mostrándose así la vista de consulta con la nueva selección de parámetros.

Si, por el contrario, el usuario selecciona la opción “Diagnóstico de Fallos” en la Figura 20, dará paso al diagnóstico de los códigos de fallo que la ECU detecte en el vehículo.

En este diagnóstico pueden darse dos situaciones, una situación favorable en la que no se encuentre ningún código de fallo, notificándoselo al usuario, o, por el contrario, que se encuentren uno o más códigos de fallo, mostrándoselos al usuario en un listado.

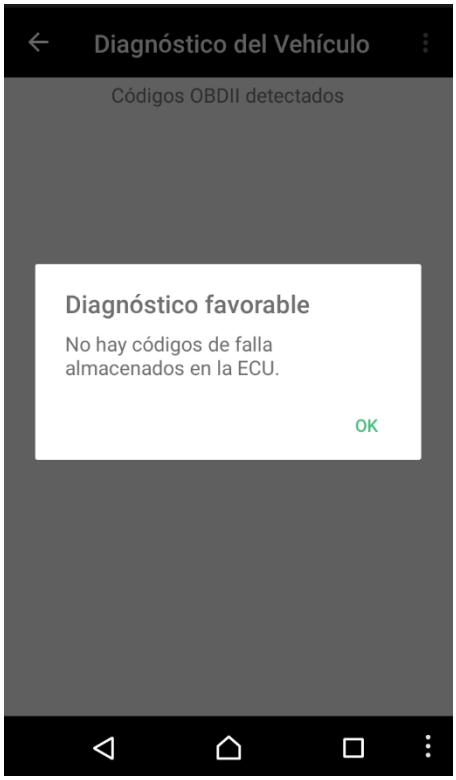

Figura 23. Vista de la función Diagnóstico del Vehículo en caso de no encontrar códigos activos.

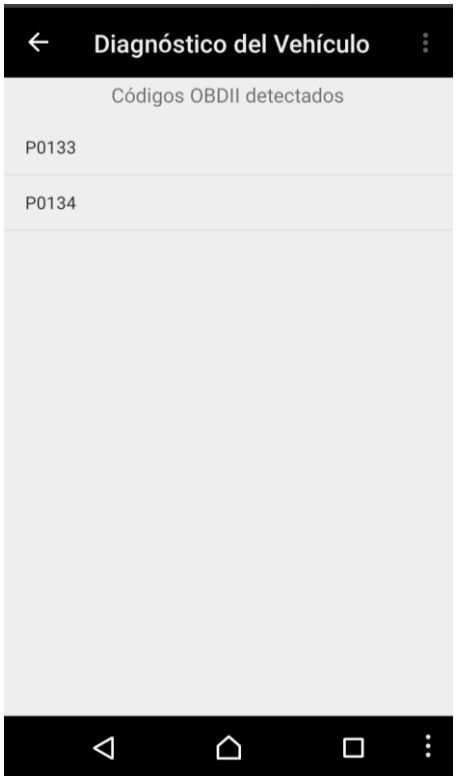

Figura 24. Vista de la función Diagnóstico del Vehículo con lista de códigos de fallos

Esta vista dispone también de diversas opciones en la barra de herramientas:

- Enviar el diagnóstico a un taller registrado en la aplicación, a través de su aplicación de correo predeterminada, generando un correo estándar con el listado de fallos.

- Una breve explicación de la codificación de los códigos de falla y su significado.

Por último, añadir que ambas opciones están accesibles, pudiendo pasar de una a otra sin ningún tipo de problema si el usuario lo desea.

Para el diseño de las interfaces que componen el módulo, se ha procurado cumplir con distintos principios de diseño, cómo puede ser la consistencia de las pantallas o la navegación intuitiva entre las propias vistas.

El mantener una consistencia en el módulo se ha perseguido a través de implantar una apariencia común, tanto en la forma de presentación de la información, como en la disposición y el formato de los botones con los que se puede interactuar durante su uso.

De igual manera, se ofrece una navegación intuitiva al usuario utilizando botones de instrucciones con títulos claros y descriptivos de la acción que realizan, dando al usuario la posibilidad de prever de forma sencilla lo que implica cada uno de ellos, indicando además en cada una de las interfaces en qué estado se encuentra la aplicación en ese momento.

5.4.- Gestión de la configuración

La gestión de la configuración es una de las actividades más importantes a la hora de desarrollar un proyecto software hoy en día.

Se puede entender como gestión de la configuración al conjunto de actividades y técnicas utilizadas para gestionar un proyecto software a lo largo de su proceso de desarrollo, de cara a controlar todos los cambios que en él se producen.

Esta gestión es de gran importancia, sobre todo cuando se desarrolla un proyecto de manera colaborativa, lo que permite a los distintos desarrolladores el compartir código y controlar siempre el código y sus cambios, manteniendo versiones estables del sistema.

En este caso en concreto, el control de código se ha llevado a cabo de forma conjunta con el resto del proyecto *CarEConnect*, mediante la herramienta Team Foundation Server, a través de la sincronización con Visual Studio (ver Figuras 25 y 26)

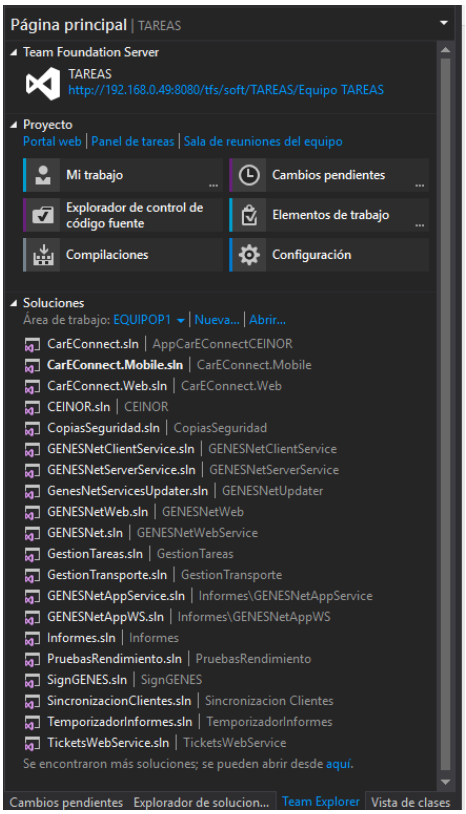

Figura 25. Gestión del proyecto y soluciones con Team Foundation Server

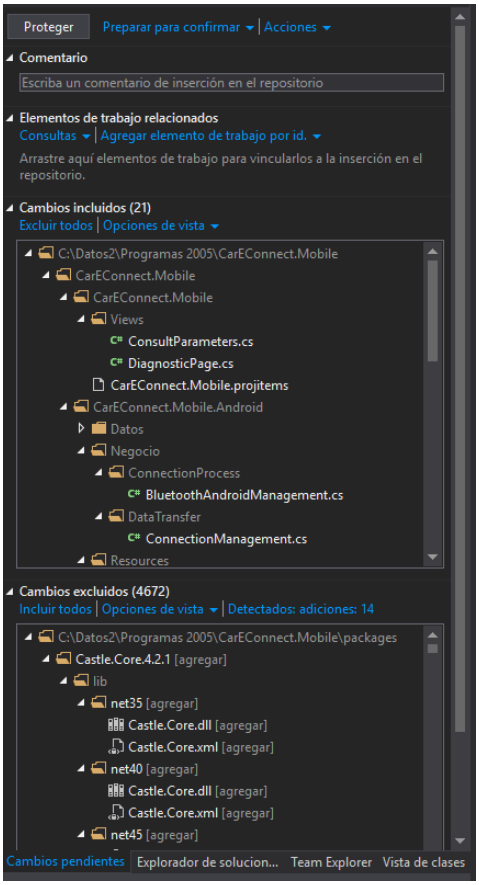

Figura 26. Gestión de los cambios mediante Team Foundation Server

6.- Pruebas y calidad

En este capítulo se describe de forma breve las pruebas que se han realizado para el sistema.

6.1.- Pruebas unitarias

Las pruebas unitarias consisten en probar módulos aislados de código para verificar su comportamiento aislado, sin la interacción con otros módulos.

La idea principal de este módulo era probar los métodos de negocio referidos a la gestión del Bluetooth y la gestión de la comunicación con el OBDII, así como los métodos de la capa de datos correspondientes al procesamiento de la información.

Sin embargo, estas pruebas se han debido aplazar y realizar de forma conjunta como subsistemas dentro de las pruebas de integración. Este aplazamiento se ha debido a que, para aislar el comportamiento de cada módulo, se debían simular componentes como la base de datos o los objetos BluetoothAdapter y BluetoothSocket, pilares fundamentales en los procesos a probar. Esta simulación se realizaría mediante el uso de mocks, objetos destinados a la simulación, pero ha resultado imposible generar mocks de estos objetos en C#.

Dentro de las pruebas unitarias, sí que se han realizado pruebas sobre el objeto *ViewModel*, verificando el comportamiento del patrón MVVM en cuanto a la modificación de sus parámetros y la generación de las correspondientes notificaciones en base a eventos.

6.2.- Pruebas de Integración

Las pruebas de integración consisten en probar el comportamiento de los distintos módulos unitarios en forma de sistemas o subsistemas.

Así, las pruebas que no se pudieron realizar de forma unitaria comentadas en el punto anterior, se han desarrollado dentro de este punto, ya con la interacción real del Bluetooth y los Bluetooth Sockets, y comunicaciones reales con el OBDII.

De esta manera, se ha verificado que:

- El módulo de conexión Bluetooth funciona de forma adecuada, lo que incluye:
    - Detección del estado del Bluetooth.
    - Proceso de descubrimiento de dispositivos.
    - Proceso de conexión con un dispositivo OBDII.

- El módulo de comunicación con el OBDII funciona de la forma esperada, verificando así que:
    - Los parámetros enviados al OBDII se adecúan al comportamiento esperado, tanto a la hora de inicializar el dispositivo como posteriormente durante la petición de datos.
    - Las respuestas del OBDII se adecúan al comportamiento esperado, lo que incluye posibles excepciones y que los datos del estado en tiempo real se encuentran en los rangos de valores establecidos para cada uno de ellos.
    - Los parámetros solicitados en el modo de consulta se adecúan a la configuración de visibilidad establecida en cada momento.

- La base de datos almacena los datos de forma correcta.

- El comportamiento del MVVM verificado mediante pruebas unitarias se comporta de manera correcta integrándose con el resto de la aplicación.

6.3.- Pruebas de Aceptación

Las pruebas de aceptación consisten en la validación por parte del cliente que solicita el desarrollo, del comportamiento del sistema, en base a directrices que el propio cliente establece.

Por tanto, estas pruebas se han realizado con los directivos de la empresa y compañeros del equipo de desarrollo, probando en distintos vehículos el funcionamiento de la aplicación.

6.4.- Medición de calidad

Uno de los procesos que se deben considerar a la hora de desarrollar un software, en sus últimas fases, es el análisis de su calidad. La calidad depende de distintas variables y métricas, y una de estas variables corresponde a la propia calidad del código desarrollado.

Para este proyecto, utilizaremos la herramienta Sonar para el análisis de la calidad del código desarrollado. Esta herramienta nos permite analizar el código desde distintas perspectivas.

Analizando la primera versión del código, obtenemos las estadísticas mostradas en las Figuras 27 y 28.

Figura 27. Información general del proyecto obtenida por Sonar

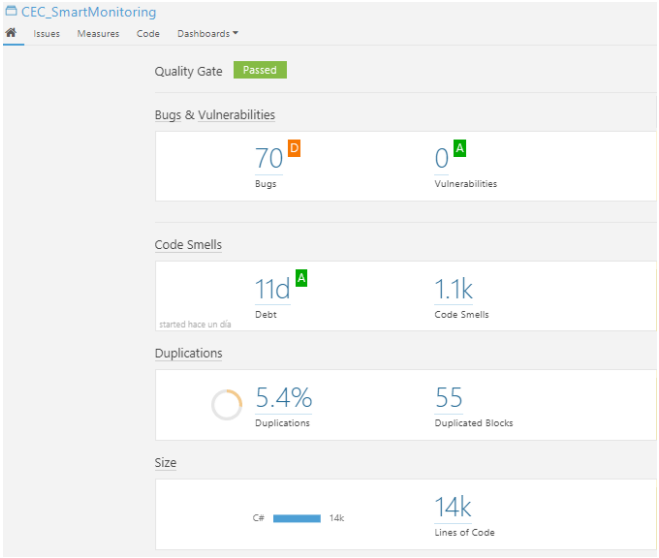

Figura 28. Resultados obtenidos por Sonar tras el análisis

Como se puede ver, la herramienta tras analizar el código nos ofrece una primera valoración como es el "*Quality Gate*", que en este caso está aprobada. Sin embargo, Sonar nos ofrece distintas propiedades y cualidades sobre el código desarrollado; entre estas, por ejemplo, la confiabilidad, la facilidad de mantener dicho código, la seguridad que ofrece, la complejidad de los ficheros, los fragmentos de código duplicados, líneas documentadas, o problemas generales. Todas estas mediciones persiguen el mismo propósito, analizar el código y facilitarte su mejora de calidad.

En base a los resultados mostrados anteriormente, el aspecto en el que más nos debemos preocupar en principio es la confiabilidad del código, marcada por los bugs y las vulnerabilidades. Si bien no tenemos vulnerabilidades, tenemos 70 bugs que nos otorgan una calificación de "D", ya que además alguno de ellos es de carácter crítico.

Una vez analizados estos bugs, en su mayoría se reducen a aspectos como el tratamiento adecuado de operaciones con distintos tipos de unidades, el manejo adecuado de variables estáticas y el uso correcto de métodos de carácter asíncrono.

Otro de los aspectos a mejorar en este caso corresponde con la mantenibilidad del código. Si bien obtenemos una valoración inicial de "A", al no superar un porcentaje del 0,05% de ratio de deuda técnica, su valoración es de 11 días de deuda, siendo un aspecto que es interesante mejorar.

Nuevamente, una vez analizado el código y los resultados en este apartado, llegamos a la conclusión de que gran parte de esta deuda se acumula en aspectos como el no seguir ningún estándar a la hora de nombrar variables y clases, la existencia de código que no se estaba utilizando en ese momento, así como código comentado, o la utilización incorrecta de variables en cuanto a su declaración global requiriéndose para momentos puntuales, o el no ser declaradas de "sólo lectura" cuando así lo requieren. Es decir, en su mayoría, son problemas de gran acumulación pero que tienen una solución muy sencilla y rápida, lo que fácilmente nos permite reducir esta deuda con su corrección.

Sin embargo, Sonar considera el hecho de que gran cantidad de métodos, y variables, no son utilizados o nunca cambian su valor, por el hecho de que estas acciones se realizan a partir del patrón MVVM o desde otras clases, considerando estos métodos y variables sobrantes en el código cuando esto realmente es incorrecto.

Con todo esto, aplicados las correcciones necesarias de cara a las propiedades comentadas anteriormente como son la confiabilidad y la mantenibilidad, obtenemos los resultados mostrados en la Figura 29.

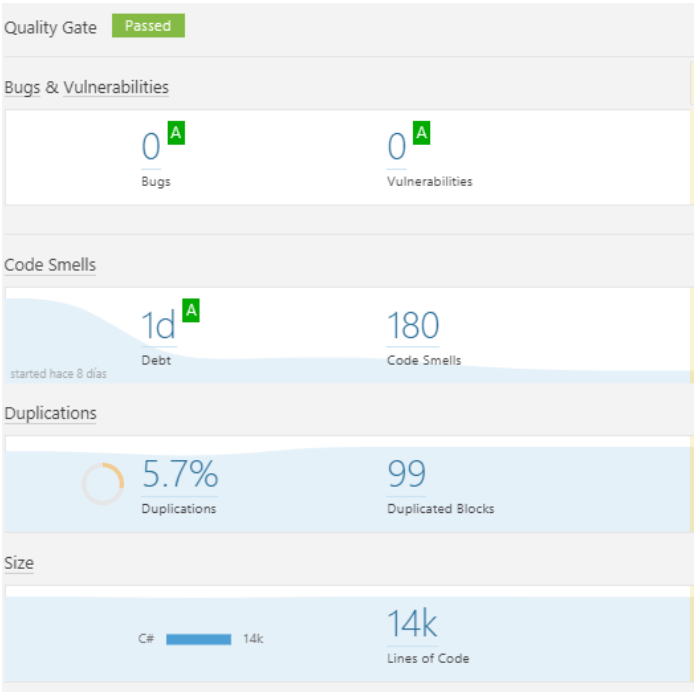

Figura 29. Estadísticas de Sonar tras aplicar los distintos cambios

Como se puede ver, se consiguen corregir el 100% de los bugs de cara a la confiabilidad del código, reduciendo además la deuda técnica a un día. En consecuencia, se incrementa el código duplicado en un 0,3%, lo que no se considera un incremento preocupante.

Siendo los aspectos anteriores los más relevantes a la hora de analizar el código, cabe también destacar la posibilidad de analizar la complejidad del código resultante (ver Figura 30). Se procura que la complejidad (analizada en este caso por clases) sea la menor posible.

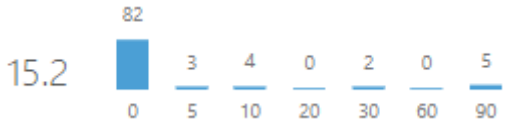

Figura 30. Categorización de la complejidad de las clases del proyecto

Tras analizar los resultados de la complejidad, se puede comprobar que la mayoría de las clases del código tienen una complejidad muy baja o nula, concentrándose en solo unas pocas de ellas un alto grado de complejidad. Al no considerarse una situación atípica en la que el que el grado de complejidad sea creciente en el análisis global no se han tomado medidas, pero podría realizarse una refactorización de estas clases para mejorar estos resultados.

7.- Conclusiones y trabajos futuros

En este capítulo se describen las conclusiones alcanzadas durante el desarrollo de este proyecto, al igual que distintas ideas y tareas que se podrían realizar como extensión a este proyecto base en un futuro.

7.1.- Conclusiones

El presente proyecto tiene como finalidad el otorgar a los usuarios de la aplicación de *CarEConnect* la posibilidad de monitorizar el estado de su vehículo en tiempo real. El objetivo final de este módulo es que el usuario pueda detectar comportamientos anómalos o fallos en su vehículo, o simplemente verificar su comportamiento, y transmitir estos comportamientos a cualquiera de sus talleres para poder solucionarlo.

El proyecto se ha desarrollado de forma satisfactoria, cumpliendo todos los requisitos especificados, a excepción de la réplica del sistema para iOS, debido a la imposibilidad de trabajar con la tecnología Bluetooth en dispositivos con este sistema operativo.

Sin embargo, no ha sido fácil; este proyecto ha requerido por mi parte una familiarización con distintas tecnologías y herramientas que no conocía, siendo las más destacadas Xamarin como entorno de desarrollo, y la tecnología OBDII junto a la

tecnología Bluetooth que me han requerido una fase de aprendizaje importante. De igual manera, la complejidad que supone el hecho de tener que adaptarse a tantos protocolos de comunicación, fabricantes, e incluso dispositivos OBDII, han hecho de este proyecto, sobre todo en la fase de pruebas, todo un reto.

En cuanto a mi propia vivencia durante el desarrollo de este proyecto, su realización me ha abierto los ojos sobre cómo es desarrollar un proyecto software en la vida real, incluyendo la experiencia en una empresa, lo que me ha dado pie a verificar la importancia del trabajo colaborativo, de una definición concisa de los requisitos, o los plazos de entrega, entre otras tantas cosas que hay que considerar para realizar un proyecto de esta envergadura (costes, clientes potenciales, etc.).

7.2.- Trabajos futuros

Los requisitos que se establecieron para este proyecto como se decía anteriormente se han cubierto. De esta manera, y con la especificación del proyecto global de *CarEConnect* acabada, en principio no se debería proseguir con el desarrollo en un futuro, pero, si así fuese, existen distintos conceptos o ideas a implementar para lo que envuelve a este módulo:

- Replicar el sistema para que funcione con tecnología WiFi y dar cobertura tanto a sistemas Android como iOS.

- Incluir más parámetros de monitorización en tiempo real.

- Implementar un sistema automatizado que permita comparar los valores en tiempo real respecto a valores medios obtenidos en base a registros almacenados para poder comprobar el comportamiento normal del vehículo y posibles variaciones.

- Implementar más modos de actuación que ofrece la tecnología OBDII.

- Incluir distintos formatos de presentación de los datos como pudieran ser gráficas o elementos que simulen el cuadro de mandos de un vehículo.

- En función de la posible expansión de la aplicación a nivel internacional, incluir su traducción a varios idiomas.

Bibliografía

[1] Bluetooth. Wikipedia: la enciclopedia libre. 30 agosto 2017, 17:30 [consulta: septiembre 2017] Disponible en: https://es.wikipedia.org/wiki/Bluetooth

[2] Unidad de control motor. Wikipedia: la enciclopedia libre. 26 junio 2017, 22:23 [consulta: septiembre 2017] Disponible en: https://es.wikipedia.org/wiki/Unidad_de_control_de_motor

[3] OBD. Wikipedia: la enciclopedia libre. 25 junio 2017, 23:46 [consulta: septiembre 2017] Disponible en: https://es.wikipedia.org/wiki/OBD

[4] SQLite. SQLite. (s.f.) [consulta: octubre 2017] Disponible en: https://www.sqlite.org/

[5] Desarrollo en cascada. Wikipedia: la enciclopedia libre. 24 agosto 2017, 14:46 [consulta: septiembre 2017] Disponible en: https://es.wikipedia.org/wiki/Desarrollo_en_cascada

[6] Android. Android. (s.f.)[consulta: septiembre 2017] Disponible en: https://www.android.com/

[7] C Sharp. Microsoft. (s.f.) [consulta: septiembre 2017] Disponible en: https://docs.microsoft.com/es-es/dotnet/csharp

[8] Socket de Internet. Wikipedia: la enciclopedia libre. 9 agosto 2017, 12:50 [consulta: octubre 2017] Disponible en: https://es.wikipedia.org/wiki/Socket_de_Internet

[9] Visual Studio. Visual Studio. (s.f.) [consulta: septiembre 2017] Disponible en: https://www.visualstudio.com/es

[10] Xamarin. Xamarin. (s.f.) [consulta: septiembre 2017] Disponible en: https://www.xamarin.com

[11] Microsoft Project. Wikipedia: la enciclopedia libre. 8 julio 2017, 18:18 [consulta: septiembre 2017] Disponible en: https://es.wikipedia.org/wiki/Microsoft_Project

[12] Team Foundation Server. Team Foundation Server. (s.f.) [consulta: septiembre 2017] Disponible en: https://www.visualstudio.com/es/tfs

[13] MagicDraw. No Magic. (s.f.) [consulta: septiembre 2017] Disponible en: https://www.nomagic.com/products/magicdraw

[14] ELM327 OBDII to RS232 Interpreter. ELM Electronics. (s.f.) [consulta: octubre 2017] Disponible en: https://www.elmelectronics.com/wp-content/uploads/2016/07/ELM327DS.pdf

[15] SonarQube. SonarQube. (s.f.) [consulta: septiembre 2017] Disponible en: https://www.sonarqube.org

[16] Conjunto de comandos Hayes. Wikipedia: la enciclopedia libre. 19 mayo 2017, 15:09 [consulta: octubre 2017] Disponible en: https://es.wikipedia.org/wiki/Conjunto_de_comandos_Hayes

[17] OBD-II PID. Wikipedia: la enciclopedia libre. 18 abril 2017, 18:12 [consulta: octubre 2017] Disponible en: https://es.wikipedia.org/wiki/OBD-II_PID

[18] The MVVM Pattern. Xamarin Developer (s.f.) [consulta: noviembre 2017] Disponible en: https://developer.xamarin.com/guides/xamarin-forms/enterprise-application-patterns/mvvm

[19] Protocolos Bluetooth. Wikipedia: la enciclopedia libre. 4 julio 2016, 09:24 [consulta: octubre 2017] Disponible en: https://es.wikipedia.org/wiki/Protocolos_Bluetooth