Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

CarAnalyzer:
Aplicación móvil para el diagnóstico de vehículos a través de la interfaz OBDII

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Autor: Navarrete Márquez, Julián

Tutor: Andrés Martínez, David de

2020-2021

Resumen

Por todos es sabido que la tecnología está presente en todos los aspectos de nuestra vida, desde un simple juguete para niños hasta un supercomputador para el procesamiento de datos. El sector del automóvil no es una excepción. En la actualidad todos los vehículos incorporan una interfaz y una pantalla para realizar la interacción persona-computador. Pero los fabricantes no siempre nos ofrecen toda la información que desearíamos, y aquí es donde entran en juego puertos como el OBDII.

Así pues, aprovechando las posibilidades que nos ofrecen estas tecnologías, en este trabajo se desarrollará una aplicación o APP para terminales Android la cual, mediante el protocolo de transmisión de voz y datos Bluetooth y el dispositivo de traducción de la interfaz OBD llamado ELM327, se substraerá la información que el usuario desee. Por ello, lo primero que se realizó fue una investigación sobre el modo de funcionamiento del puerto OBDII con el objetivo de averiguar qué tipo de información circula por él. Se observó que esta se identifica con PIDs y se puede acceder a ella mediante distintos modos de funcionamiento con distintos códigos. Una vez averiguado su forma de funcionamiento, se analizó el mercado en busca de aplicaciones similares para averiguar qué aspectos hay que mejorar y, una vez realizado eso, se pasó a la implementación de la aplicación.

Palabras clave: OBDII, Android, Bluetooth, ELM327, Diagnóstico, Vehículos

Abstract

It is common knowledge that technology is present in all aspects of our lives, from a simple children's toy to a supercomputer for data processing. The automotive sector is not an exception. Today, all vehicles incorporate an interface and a screen for human-computer interaction. But manufacturers do not always provide us with all the information we would like, and this is where ports such as the OBDII come into the scene.

So, taking advantage of the possibilities offered by these technologies, this work will develop an application or APP for Android terminals which, using the voice and data transmission protocol known as Bluetooth and the OBD interface translation device called ELM327, will subtract the information that the user wants. Therefore, the first thing that was carried out was an investigation into the mode of operation of the OBDII port with the aim of finding out what kind of information flows through it. It was observed that this information is identified with PIDs and can be accessed by different modes of operation with different codes. Once it was found out how it works, the market was analyzed in search of similar applications to find out what aspects need to be improved and, once that was done, the application was implemented.

Keywords: OBDII, Bluetooth, Android, ELM327, Diagnosis, Vehicles

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. Introducción

Desde hace un cierto tiempo, los vehículos de nueva fabricación deben llevar incluido un puerto denominado OBD (On Board Diagnosis). Este puerto es muy utilizado en las ITVs (Inspección Técnica de Vehículos) para analizar el estado del vehículo realizando un control sobre los sensores que presenta. Estos sensores son, principalmente, aquellos que detectan la emisión de gases contaminantes a la atmósfera producto de la combustión de combustibles fósiles. De este modo, en 1988 la California Air Resources Board [1] dictaminó que a partir de 1988 todos los vehículos motorizados dispusieran de un puerto OBD para el control de los gases contaminantes emitidos y el análisis del estado del vehículo.

Esta tecnología supuso una revolución, pero no era muy efectiva pues solo obtenía información de una serie de componentes específicos y no se podían calibrar los parámetros según las exigencias del territorio. Es por ello por lo que esto, unido a que posteriormente surgieron unas medidas más estrictas sobre las emisiones de gases, llevaron a la evolución del puerto al actual OBD2.

Con la llegada de la nueva versión, todo cambió, pues se podía acceder a otros niveles de información nunca vistos mediante la asignación de PIDs a los distintos elementos que circulan por el puerto y esto conllevó la aparición de todo tipo de herramientas para la interpretación de la información. Estas herramientas, son las interfaces de interpretación como la RS232 implementadas dentro de dispositivos como el mencionado ELM327 y el software especializado en interpretar la información transmitida por este último dispositivo.

En este trabajo se pretende realizar el desarrollo de una aplicación para dispositivos móviles Android denominada CarAnalyzer con la que obtener datos y estadísticas en tiempo real del vehículo mediante la integración de la dicha aplicación y la interfaz OBDII junto con otro dispositivo, el ELM327, del cual se hablará más adelante.

Así pues, la aplicación, que se comunica con el ELM327 mediante Bluetooth, obtendrá la información que se desea mediante el envío de mensajes con los PIDs y, cuando se reciba la respuesta, se interpretarán y convertirán a un formato estándar para que sean perfectamente comprensibles por los usuarios.

1.1. Motivación

Dado que la tecnología Android está muy presente en todos los aspectos de la vida es muy conveniente obtener más conocimiento sobre esta. Esto se puede realizar de muchas maneras y una de ellas es mediante la realización de una aplicación móvil. De este modo, se ahonda cómo funcionan internamente las aplicaciones de nuestros *smartphones* y los misterios que esconden.

El motivo por el que se decidió realizar este tipo de aplicación con esta funcionalidad es porque la cantidad de información que se proporciona por defecto en el propio vehículo puede llegar a ser limitada para determinados usuarios. Estos datos que se pueden obtener del vehículo son los utilizados en las ITVs de España para revisar el estado de los automóviles; por lo que, teniendo un conocimiento previo de lo que le ocurre a este antes de realizar la revisión, se pueden evitar los problemas que suponen una valoración desfavorable de la revisión.

1.2. Objetivos

El objetivo principal de este trabajo es la realización de una aplicación para dispositivos móviles Android con la que obtener información que va más allá de la proporcionada visualmente por el propio vehículo. De este modo, se pretende llegar a un tipo de público exigente y al que le gusta conocer todo lo que ocurre en su vehículo.

Mas allá de este objetivo principal, el trabajo tiene como objetivos:

- La elaboración de una aplicación sencilla y cuidada para el uso cuotidiano.
- Obtención de información del vehículo raramente visible.
- Mostrar errores que se hayan podido producir.
- Recordar la ubicación de dónde se ha aparcado el vehículo.

1.3. Estructura de la memoria

La memoria está organizada en distintos capítulos, sobre los cuales se aporta una breve descripción a continuación:

En el capítulo 2 se describe el estudio de mercado que se realizó para analizar las distintas aplicaciones que existen en el mercado actual de los dispositivos Android. De este modo se puede se puede obtener una ligera idea de los aspectos en los que debe mejorar la aplicación que se va a desarrollar y en los que se debe inspirar.

En el capítulo 3 se describen los aspectos relacionados con la interfaz OBDII. Entre estos se encuentran su historia y evolución y la forma de comunicación entre el dispositivo ELM327 y el software a desarrollar. Con esta comunicación entre las dos partes se puede obtener todo tipo de información que deseemos del vehículo, desde la que ya se proporciona por defecto en el propio vehículo hasta aquella que está oculta.

En el capítulo 4 se realiza la especificación de la aplicación que se va a desarrollar. En concreto se describirán las acciones que se pueden realizar en la aplicación (casos de uso), la estructuración y atributos propios de la información que se almacenará en la APP (diagrama de clases) y, además, se mostrará el diseño preliminar de la interfaz de la aplicación. Este diseño es en el que se deberá basar la versión final de la aplicación.

En el capítulo 5 se describe la arquitectura software que presenta la aplicación y, de esta manera, identificar claramente que acciones se realizarán internamente en la aplicación. Además, también se describirá la arquitectura total del sistema, es decir, el modo de interacción de todos los componentes presentes.

En el capítulo 6 se describen las diferentes funcionalidades que se han desarrollado en la APP. En concreto, se mostrarán diferentes fragmentos de código que sirven para la ejecución y el correcto funcionamiento de todas las funciones. También se realizarán unos comentarios sobre la estructuración de la APP y los servicios y bibliotecas que utiliza.

En el capítulo 7 se realiza un pequeño recorrido por la aplicación implementada, mostrando las diferentes pantallas existentes y describiendo que acciones se pueden realizar en ellas.

En el capítulo 8 se realiza un resumen de todo lo elaborado en el trabajo, los objetivos que se han conseguido y, además, una lista de posibles mejoras futuras que se le podrían añadir a la aplicación.

El apéndice A muestra los comandos AT o comandos de configuración para el dispositivo OBDII. Se muestran diferentes tablas con cada uno de los estándares existentes.

El apéndice B muestra la lista de PIDs interpretables por el OBDII divididos en distintos modos de funcionamiento. Todos los que se muestran, tienen relación con la APP desarrollada.

El apéndice C es un manual de usuario de la aplicación CarAnalyzer. En él se describen de forma clara y concisa cómo se realizan todas las acciones que se pueden realizar mostrando, además, imágenes para que sea más fácilmente interpretable por el usuario.

Por último, en el apéndice D se describe de manera más profunda la implementación de las funcionalidades de la APP mostrando código más interno de esta. En concreto, se describe el código relacionado con la gestión de los mensajes que se envían y reciben a través del Bluetooth.

2. Estudio de mercado

En el mercado actual existen diversas aplicaciones que proporcionan funcionalidades que van desde las más sencillas hasta las más complejas. Estas aplicaciones tienen interfaces diferentes y no todas son tan intuitivas para el uso de un usuario promedio. Es por ello por lo que se ha realizado un estudio de mercado para explorar cuáles son los puntos fuertes y débiles de las aplicaciones existentes y, de este modo, corregirlos.

2.1. Descripción del procedimiento realizado

En primer lugar, se exploró cuáles son las principales aplicaciones que implementan o que realizan las funciones que se desea en la aplicación a desarrollar. Por ello, se consultó, a través de la página Web Android Phobia [2], cuales eran las aplicaciones más recomendables y se encontró una lista con algunas de ellas. Una vez obtenidos los nombres de las aplicaciones, se seleccionaron los dispositivos para la prueba de estas; entre ellos tenemos el conector para la comunicación con el puerto OBDII (ELM327) y el *smartphone* en el que se instalarían las aplicaciones, un Xiaomi Mi 9T Pro con el Sistema Operativo (SO) Android 10. Hay que destacar que este teléfono móvil, posee pantalla completa, es decir, no presenta marcos, por lo que, si una aplicación no está bien optimizada para ese modo, se puede perder parte del contenido.

Así pues, se descargaron las aplicaciones a través de Google Play, que es el portal por defecto del SO para la descarga de aplicaciones de manera segura; y se realizaron las pruebas pertinentes para profundizar sobre la implementación de estas aplicaciones.

2.2. Análisis de la competencia

A continuación, se muestran unos pequeños análisis de las aplicaciones testeadas y su comparación.

2.2.1. OBD Auto Doctor

Esta aplicación ofrece una interfaz bastante agradable e intuitiva. En primer lugar, se deberá de activar el Bluetooth del dispositivo móvil para conectarse al dispositivo OBDII. Para ello se deberá pulsar en el icono de la parte superior derecha de la aplicación (Ver Figura 1a) y seleccionar el dispositivo deseado. Una vez realizado esto, se enciende el motor del coche y los datos comenzarán a representarse. Se podrá ver en tiempo real las revoluciones por minuto (rpm) del motor, la temperatura del aire que circula por el motor… Además, dispone de otras pestañas (Ver Figura 1b) para navegar entre otras funcionalidades como la lectura de códigos de error del coche, añadir manualmente sensores para observar otros datos, los ajustes de la aplicación, etc. Otra de sus ventajas es que dispone de la lectura de más de 18000 códigos de error.

Por el contrario, esta aplicación no dispone de todas las funciones posibles en esta versión gratuita. Para disponer de ellas, hay que suscribirse a un plan mensual o a un plan anual; lo cual puede no estar dentro de las posibilidades de muchas personas. Otro de sus inconvenientes es que no hay disponible una traducción de los textos a otros idiomas, lo que puede ser un impedimento muy grande para personas que deseen utilizar la aplicación y no tengan conocimientos de inglés.

La aplicación se puede obtener mediante el siguiente enlace:
https://play.google.com/store/apps/details?id=com.obdautodoctor&hl=es&gl=US

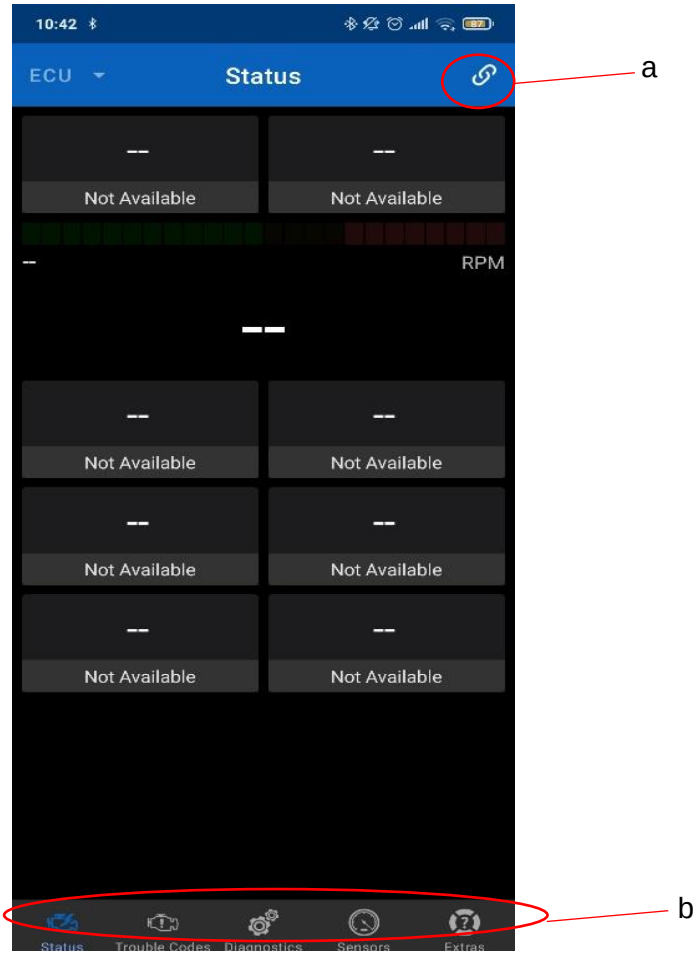


Figura 1: Interfaz OBD Auto Doctor

2.2.2. Torque Lite

En esta ocasión nos encontramos con una aplicación mucho más compleja y poco intuitiva. Nada más iniciar la aplicación, aparece un mensaje de que esta ha estado diseñada para una versión de Android anterior a la que posee el sistema de prueba, lo que podía ocasionar problemas. Y efectivamente, la interfaz no estaba bien adaptada para el dispositivo usado para la prueba y se perdía parte del contenido, tal y como se puede observar en la Figura 2. Al contrario que en la anterior, es una aplicación mucho menos intuitiva. También hay que destacar que añadir sensores era bastante más incómodo que en la anterior.

A favor de esta aplicación hay que decir que sí dispone de traducción a otros idiomas, aunque no todos los textos están traducidos y que se puede realizar una personalización más amplia de la interfaz, aunque no esté del todo pulida.

En general, no se obtiene una buena experiencia de usuario con esta aplicación.

Enlace para su obtención:
https://play.google.com/store/apps/details?id=org.prowl.torquefree&hl=es&gl=US

Figura 2: Interfaz Torque Lite

2.2.3. DashCommand

Nada más abrir la aplicación, aparece una serie de mensajes para la configuración. En concreto se pide seleccionar qué tipo de dispositivo se va a utilizar (ELM) y pide realizar un perfil del vehículo. Esto servirá en el caso de que se disponga de diversos coches y se desee que la información de cada uno esté diferenciada. Otro de los mensajes que aparece es si se quiere actualizar a la versión de pago para disponer de todas las funcionalidades.

Una vez ya dentro de la aplicación, encontramos con una interfaz muy bien organizada, con información distinta en cada pestaña y muchas funcionalidades a explorar. Se puede apreciar claramente en la Figura 3 como existen de diversas pestañas para acceder a distintos tipos de información. Uno de los puntos diferenciadores de esta aplicación respecto a otras es que dispone de una interfaz GPS en tiempo real y de un foro en el que se puede contactar con otros usuarios (sería requisito necesario disponer de una cuenta de usuario) e intercambiar opiniones.

Por el contrario, como se ha mencionado anteriormente, todas las funcionalidades están en la versión de pago; además de no disponer de traducción a otros idiomas.

Enlace a la aplicación:
https://play.google.com/store/apps/details?id=com.palmerperformance.DashCommand&hl=es&gl=ES

Figura 3: Interfaz DashCommand

2.2.4. Car Scanner ELM OBD2

Como en el caso de la aplicación DashCommand, antes de acceder a la APP en sí, se pide que se configure un perfil para el coche, se elija el idioma de la aplicación (de las pocas que dispone de esta opción y, además, presenta una gran variedad de traducciones disponibles) y el dispositivo a conectar. Una vez realizado esto, se entra de lleno a la aplicación en sí. Esta aplicación presenta una interfaz bastante clara y concisa con todo detallado y explicaciones sobre qué se puede observar en cada pestaña (Ver Figura 4). Además, existe un apartado en el que se le pueden realizar modificaciones de funcionamiento al coche, como puede ser que al pulsar el botón de cerrar las puertas si hay alguna ventanilla bajada se suba, plegar los espejos automáticamente…

Por el contrario, hay que destacar que esta es la versión gratuita y que, por lo tanto, no dispone de todas las funciones (aunque posee bastantes) y además presenta publicidad. Para eliminar este inconveniente, habría que comprar la versión de pago.

En definitiva, pese a sus pequeños inconvenientes, esta aplicación proporciona una experiencia de usuario bastante buena y es muy recomendable su uso.

La aplicación se puede obtener mediante la siguiente URL:
https://play.google.com/store/apps/details?id=com.ovz.carscanner&hl=es&gl=ES

Figura 4: Interfaz Car Scanner ELM OBDII

2.2.5. EODB Facile

En esta aplicación, como viene siendo habitual, aparecen mensajes de configuración nada más abrirla para crear el perfil del coche. La aplicación en sí no ofrece mucho, pues es la versión gratuita y dispone de la lectura justa de datos, pero la interfaz está bastante bien cuidada como se puede observar en la Figura 5. Además, está disponible en español, lo cual es de agradecer. Presenta una función que no ha sido incluida en ninguna otra aplicación, que es la de calcular el rendimiento del coche cuando está en marcha, por ejemplo, utilizando el tramo de 0-100Km/h. Desgraciadamente para esta funcionalidad hay que comprar una versión más completa.

Como se ha comentado, esta es la versión gratuita y está bastante limitada; es más, en este caso existen dos versiones de pago, una más básica y otra con todas las funcionalidades.

Obtención de la APP:
https://play.google.com/store/apps/details?id=org.eobdfacile.android&hl=es&gl=ES

Figura 5: Interfaz EODB Facile

2.3. Tabla de comparación

Una vez realizado un pequeño análisis de las aplicaciones, se ha decidido compararlas entre sí. Para ello se han seleccionado una serie de parámetros los cuáles son de vital importancia para una aplicación orientada al público general. Estos parámetros son los siguientes:

- Interfaz: Se refiere al diseño de la interfaz de usuario de la aplicación; es decir, si presenta un diseño cuidado y bien organizado que sea sencillo de comprender para cualquier tipo de usuario.
- Usabilidad: Es el nivel de facilidad de uso que presenta la aplicación. Un buen nivel de usabilidad indica que la aplicación es intuitiva y es menos probable que cause problemas a la hora de utilizarla por usuarios nuevos.
- Información suministrada: Este parámetro indica la cuántos datos de interés se pueden obtener del vehículo vía la aplicación. Dado que todas las aplicaciones testeadas son en su versión gratuita, no se ha tenido en cuenta si la versión de pago puede proporcionar más información.
- Nivel de personalización: Indica en qué medida la aplicación permite al usuario una personalización de la interfaz o de los datos a mostrar; es decir, en qué medida podemos elegir los datos que queremos que se muestren, en qué orden…
- Valoración general: Se refiere a la puntuación obtenida por la aplicación teniendo en cuenta todos los parámetros anteriores. Se puede considerar como la media de las puntuaciones.

Por último destacar que para valorar cada uno de los aspectos de la aplicación, se ha utilizado una escala del 1 al 5 (representada con el símbolo ★) , siendo 5 la valoración más alta y 1 la menor.

Tabla 1: Comparación de las aplicaciones testeadas

Aplicación	Interfaz	Usabilidad	Información suministrada	Nivel de personalización	Valoración general
OBD Auto Doctor	★★★	★★★★	★★★	★★★	★★★
Torque Lite	★	★	★	★★★★★	★
Dashcommand	★★★★	★★★★	★★★★★	★★★	★★★★
Car Scanner ELM OBD2	★★★★★	★★★★★	★★★★★	★★★	★★★★★
EOBD Facile	★★★★	★★★★	★	★★★	★★★

Como se puede observar en la tabla, hay diversas aplicaciones que destacan en varios aspectos, consiguiendo una valoración final alta. Este es el caso de Car Scanner ELM OBD2, la cual obtiene en la mayoría de los aspectos una valoración por encima de la media de las otras. El caso contrario es Torque Lite, que obtiene una puntuación pésima en todos los aspectos menos en el nivel de personalización, donde obtiene la puntuación más alta de la comparativa. El resto de las aplicaciones, se puede decir que son recomendables para el uso del usuario, pues presentan un nivel de equilibrio entre los atributos bastante bueno.

2.4. Conclusiones

Se ha observado que en el mercado existen diversas aplicaciones las cuales realizan una función similar a la que se quiere implementar. Las hay desde las más elaboradas y cuidadas hasta todo lo contrario. De este modo, se debe de tomar como punto de partida y de inspiración la aplicación Car Scanner ELM OBD2, pues es la que presenta mejor puntuación en todos los aspectos; pero también se deben de considerar los puntos fuertes del resto de aplicaciones con el objetivo de que la que se va a desarrollar sea lo más completa posible.

A modo de resumen, la aplicación a desarrollar debería de presentar una interfaz agradable, un nivel de personalización adecuado, información de interés para el usuario y un nivel de complejidad para el uso de usuarios noveles no muy elevado. Además, también sería de agradecer que implementara una función distintiva entre todas las aplicaciones analizadas como podría ser que fuera gratuita en su totalidad.

3. La interfaz OBDII

En la introducción se ha mencionado que en los vehículos actuales existe un puerto para obtener información más oculta sobre el vehículo. Se ha hablado de cómo apareció el puerto OBD y como evolucionó, pero ahora se realizará una explicación más detallada sobre esta interfaz, ahondando en detalles de su estructura y de la interfaz de acceso al puerto OBDII

3.1. Descripción general de la interfaz

La nueva generación del OBD, el OBD2, que fue recomendada por la SAE (Society of Automotive Engineers) [8], es capaz de obtener más información sobre el vehículo. Para conseguir esta funcionalidad, se guarda un registro de fallos y cada uno de estos tiene asignado un código; luego para obtener la información, la interfaz de acceso al puerto envía PIDs (Parameter ID, es la identificación de cada código para la comunicación) y entonces la centralita del coche, a través del puerto OBD2, envía los códigos de la información solicitada, los cuales son interpretables por la interfaz de acceso a través de un estándar y, seguidamente, un software presente en un dispositivo móvil u ordenador se encarga de obtener la información a través de la interfaz.

Generalmente, el dispositivo más comúnmente usado para conectarse al puerto OBD2 es el ELM327, del cual se hablará más adelante.

Desde 1996, todos los vehículos nuevos de los Estados Unidos y también los que se importaban a Europa debían de disponer de este puerto. En Europa, este puerto debía existir desde el año 2000 para los automóviles de gasolina, desde el 2003 para los diésel y desde 2005 para los camiones.

Según de donde proceda el vehículo, presentará un estándar diferente. En Europa es el EOBD, en Japón JOBD y en Estados Unidos OBD-2. Además, cada estándar presenta una norma que lo regula; de esto modo EODB es regulado por la norma ISO (International Organization for Standardization) [9] y el OBD-2 por la norma SAE.

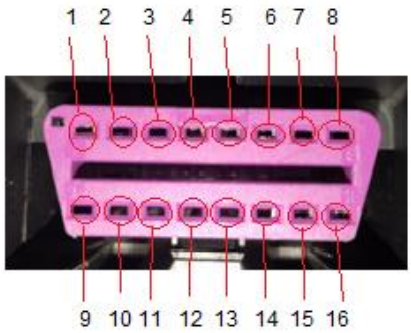
Figura 6: Comparativa conector OBD vs OBD2

3.2. El puerto

El puerto del OBD2, como se ha mencionado, es obligatorio en todos los vehículos motorizados. Este suele estar localizado debajo del volante del conductor y suele ser bastante reconocible dado su color y su forma, tal y como se puede visualizar en la Figura 7.

Figura 7: Localización del puerto

El puerto presenta una serie de pines, cada uno de los cuales tiene una funcionalidad distinta que va, desde la alimentación de la interfaz de acceso, hasta los distintos estándares para la obtención de información presente en el puerto.



1- Sin Uso
2- J1850 Bus Positivo
3- Sin Uso
4- Tierra del Vehículo
5- Tierra de la Señal
6- CAN High
7- ISO 9141-2 - Línea K
8- Sin Uso
9- Sin Uso
10- J1850 Bus Negativo
11- Sin Uso
12- Sin Uso
13- Tierra de la señal
14- CAN Low
15- ISO 9141-2 - Línea L
16 Batería - Positivo

Figura 8: Pines del puerto

En la Figura 8 se puede observar cómo hay determinados puertos que no tienen uso; esto es porque en el estándar global es de este modo, pero luego cada fabricante puede utilizar alguno de esos pines para mostrar la información que se desee. Por lo que, dependiendo de la marca del coche, hay algunos pines que pueden variar.

3.3. La interfaz de acceso. El ELM327

Como se ha comentado, todos los vehículos actuales poseen el puerto OBD2, pero la información que circula por este no se puede interpretar directamente. Es por ello por lo que existen las interfaces de acceso como el ELM327 (Ver Figura 9). Este dispositivo es un puente entre el puerto y el estándar de interfaces RS232, el cual se utiliza para la transmisión de la información de un dispositivo a otro. Sin embargo, en la actualidad se suele utilizar una conexión WiFi, Bluetooth o USB para comunicarse con el software deseado. Además, el dispositivo es capaz de interpretar hasta 9 protocolos ligados al puerto OBD y es completamente configurable, por lo que se puede adaptar al uso del usuario.

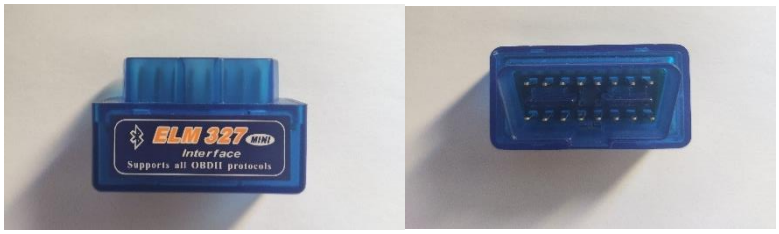


Figura 9: ELM327

3.3.1. Pines

El dispositivo ELM327 está formado por un chip que presenta 28 pines y cada uno de ellos presenta una funcionalidad distinta. Esto lo podemos observar en la Figura 10. Además, este chip está organizado en distintos bloques (Ver figura 11).

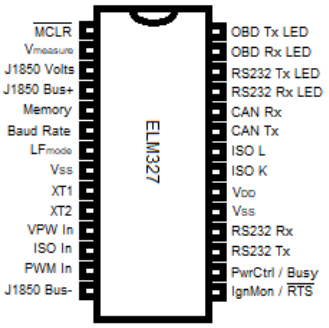


Figura 11: Diagrama de pines chip ELM327

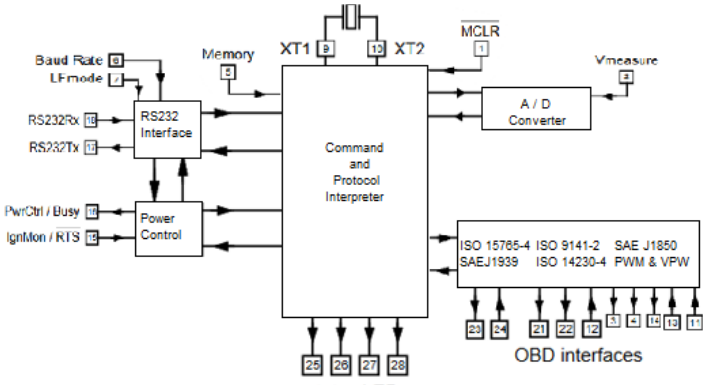


Figura 10: Diagrama de bloques ELM327

Realizando el análisis por bloques del chip se encuentra la funcionalidad y la conexión de cada uno de los pines:

Bloque RS232 Interface

- Baud Rate: Aplica la velocidad de transmisión de datos en baudios que se desea.
- LFmode: Si esta activo, los mensajes acabarán con un retorno de carro y un salto de línea. Si no lo está solo con el retorno de carro.
- RS232Tx: Pin para la transmisión de datos por RS232.
- RS232Rx: Pin para la recepción de información del RS232.

Bloque Power Control

- IgnMon / RTS: Según como se configure, este pin puede actuar para interrumpir el procesamiento del OBD para enviar un nuevo comando o como un monitor de encendido.
- PwrCtrl / Busy: Puede actuar como un indicador de que el dispositivo está procesando información o como un control para la potencia de salida.

Bloque Command and Control Interpreter

- RS232 Rx LED¸ RS232 Tx LED, OBD Rx LED y OBD Tx LED: Pines para los LEDs presentes en el dispositivo.
- XT1 y XT2: En estos pines está conectado un oscilador el cual también está conectado a Vss.
- Memory: Controla el estado por defecto de la memoria. Un nivel alto durante un inicio o reinicio significará que está habilitada por defecto; un nivel bajo estará desactivada.
- MCLR: Aplicar lógica negativa (0) a este pin hará que el ELM327 se reinicie.

Bloque A / D Converter

- Vmeasure: Se encarga de medir el voltaje que se aplica para que no supere los niveles que puedan dañar el dispositivo.

Bloque OBD Interfaces

- J1850 Volts: Controla el voltaje aplicado al bus J1850.
- J1850 Bus +: Se encarga de activar el bus J1850.
- J1850 Bus -: Si esta activo, se da preferencia a las aplicaciones que utilizan la interfaz J1850 PWM
- ISO K y ISO L: Sirve para proporcionar preferencia a los buses ISO9141 y ISO 14230.
- CAN Tx y CAN Rx: Pines que deben estar conectados a un transceptor CAN.
- VPW In: Pin de activación para el bus J1850 VPW.
- ISO In: Señal de activación para los datos de ISO9141 y ISO14230.
- PWM In: Pin de activación para la interfaz J1850 PWM.

Por último, se observan los pines Vss y V_{DD} que son para la conexión a tierra y la alimentación del chip respectivamente.

Además, se puede observar cómo cada bloque presenta una serie de conexiones con los demás. De este modo, el bloque RS232 presenta la comunicación con el control del dispositivo (Power Control) y con el intérprete de comandos (Command and Control Interpreter). Así mismo, este último también presenta conexiones con el control del dispositivo, además de con el conversor A / D (A / D Converter) y el bloque de las interfaces OBD (OBD

Interfaces). Todo esto es necesario para la correcta interpretación de los datos recibidos, de lo cual se hablará más adelante.

3.3.2. Configuración del dispositivo

Se ha mencionado que el ELM327 es completamente configurable y esto se realiza mediante comandos denominados AT. Existe una cantidad bastante grande de estos y cada uno puede ser para configurar una interfaz o un bus específico. A modo de resumen se han seleccionado como ejemplo unos comandos que se pueden utilizar para configurar el dispositivo según los gustos del usuario.

Tabla 2: Ejemplos de comandos AT y descripción

Comandos generales	Comandos OBD	Comandos ISO	Comandos J1939 CAN
<CR>: Repite el último comando	**AL**: Habilita mensajes largos	**KW**: Imprime palabras clave	**JE**: Utiliza el formato de datos ELM J1939
D: Ajustes por defecto	**AR**: Recepción automática	**IB 10**: Configura baudios de ISO a 10400	**JS**: Utiliza el formato de datos SAE J1939
Z: Reiniciar todo	**MA**: Monitorizar todo	**FI**: Establece inicio rápido	**DM1**: Monitoriza mensajes DM1
LP: Modo bajo consumo	**NL**: Mensajes de tamaño normal	**SW 00**: Para de enviar mensajes de *wakeup*	**JMT1**: Establece el multiplicador del temporizador a 1

En la Tabla 2 se muestran algunos comandos los cuales se le pueden enviar al ELM327 a través de un terminal o cualquier otra interfaz. Como se puede observar, algunos comandos son de configuración para alguno de los buses y otros muestran información que puede ser de interés. Para observar la lista completa de comandos AT disponibles, consultar el apéndice A.

3.3.3. Comunicación con el ELM327

Para la comunicación con el dispositivo, se puede realizar vía USB, Bluetooth o WiFi. De todos modos, la forma de comunicarse será la misma en todos los casos, pues el componente clave para la comunicación es la interfaz RS232. De este modo, se puede considerar a la interfaz RS232 como un simple puente para la comunicación entre el chip y la interfaz de comunicación que utilicemos (USB, Bluetooth o WiFi).

Cuando se conecte el ELM327 al puerto OBDII, los LEDS comenzarán a parpadear hasta que se queden estáticos, lo que indicará que está esperando a que se le envíe información. Acto seguido por medio de una interfaz para enviar mensajes como un terminal específico para la comunicación, se conectará y se recibirá un mensaje parecido al siguiente:

ELM327 v2.1

>

La primera línea indica la versión del dispositivo y la segunda línea es la línea de comandos para enviar los comandos que se deseen. Si las órdenes enviadas comienzan por *AT* se interpretarán como los comandos explicados anteriormente. Por el contrario, si no comienzan por estas dos letras se interpretarían como comandos OBD.

Los comandos OBD presentan una traducción de decimal a hexadecimal la cual sirve para interpretar la información y realizar los cálculos que correspondan. La equivalencia se muestra en la Tabla 3:

Tabla 3: Conversión decimal hexadecimal

Número Hexadecimal	Número Decimal
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	10
B	11
C	12
D	13
E	14
F	15

Para la comunicación mediante comandos OBD, en este caso, se utiliza el estándar SAE J1979. Este define 10 modos de funcionamiento los cuales se muestran en la Tabla 4.

Tabla 4: Modos de funcionamiento SAE J1979

Modo de funcionamiento	Descripción
01	Muestra la información actual
02	Muestra la información congelada
03	Muestra la información de códigos de error
04	Elimina los códigos de error y la información almacenada
05	Resultados test de sensores de oxígeno
06	Resultados test no monitorizados continuamente
07	Muestra los códigos de error pendientes
08	Modo de control especial
09	Solicita información del vehículo
0A	Solicita los códigos de error permanentes

Además de los modos de funcionamiento, se debe conocer qué PIDs soporta nuestro vehículo, es decir, qué información se puede obtener sobre el vehículo a través del puerto OBDII. Los PIDs son los identificadores de la información que se puede obtener. Cada vehículo de una marca puede soportar una serie de PIDs y no coincidir con los que se soportan en otro vehículo. Sin embargo, existe un PID que debe de ser soportado por todos sin

distinciones. Este es el 00; con el cual se detalla la lista de los PIDs soportados por un determinado modo de funcionamiento. Para clarificar un poco el proceso, si se envía al dispositivo el siguiente mensaje:

> 01 00

Se obtendrán los PIDs soportados en el vehículo para el modo de funcionamiento 01. La lista de estos puede ser muy extensa (Consultar apéndice B), pero algunos ejemplos de mensajes e información que se pueden obtener son los que se pueden observar en la Tabla 5.

Tabla 5: Ejemplos de mensajes y su interpretación

Modo de funcionamiento	PID	Descripción
01	05	Temperatura actual en ºC
01	0C	Revoluciones por Minuto del motor con incrementos de 1/4
02	0C	Revoluciones por Minuto del motor en un momento determinado con incrementos de ¼
02	04	Carga calculada del motor en un momento determinado

Los mensajes de respuesta vienen en un formato concreto formado por 4 bytes representados por las letras de la A hasta la D; siendo el byte A el *Most Significant Byte* (MSB) o byte más significativo, y el byte D el *Less Significant Byte* (LSB) o byte menos significativo. Así, el aspecto sería el que se puede ver en la Figura 12.

A								B								C								D							
A7	A6	A5	A4	A3	A2	A1	A0	B7	B6	B5	B4	B3	B2	B1	B0	C7	C6	C5	C4	C3	C2	C1	C0	D7	D6	D5	D4	D3	D2	D1	D0

Figura 12:Formato de las respuestas

De este modo, una vez enviado el mensaje y si la conexión con el ELM327 está correctamente realizada, se podría recibir una respuesta con el siguiente aspecto:

41 0C 1A F8

Analizando detenidamente el mensaje se obtiene el significado de cada par de caracteres. El 41 nos indica que es la respuesta a un mensaje del modo 01; en caso de que fuera una respuesta al modo 02 se recibiría un 42. El 0C es el PID de la información solicitada (Revoluciones por Minuto del motor). Y el resto de los caracteres, 1A F8, es la información obtenida en hexadecimal. Esto se debe de convertir a decimal del siguiente modo:

Resultado final: $1 * 16^3 + 10 * 16^2 + 15 * 16^1 + 8 * 16^0 = 6904$

Figura 13: Conversión de la información

Como se puede observar en la Figura 13, en primer lugar, se convierte a decimal mediante la equivalencia presentada en la Tabla 3 y luego se multiplica por la base (16) elevada la posición que ocupa el número. De este modo obtenemos que las revoluciones por minuto del motor (rpm) son de 6904; pero esto valor es muy elevado. Esto es debido a que, como se han indicado en la Tabla 5, se recibe con incrementos de 1/4; por lo que se debería de dividir el valor entre 4 para obtener un resultado de 1726 rpm, que es un valor mucho más coherente.

Al igual que ocurre al solicitar las rpm del motor, existen otros tipos de información las cuales también necesitan de una fórmula para la conversión a un valor más razonable. Por ejemplo, cuando se recibe el valor de la carga calculada del motor, el valor obtenido en decimal se debe de dividir entre 2.55 para obtener el valor normalizado. Esto ocurre en muchos más casos, por lo tanto, se debe de tener en cuenta a la hora de implementar el sistema. Para consultar en qué casos es necesario aplicar una fórmula para normalizar el resultado, consultar el apéndice B.

Por último, hay que destacar que, en casos muy puntuales, como se nos indica en el manual del dispositivo [10], el intérprete RS232 puede enviar datos nulos, es decir, enviar la cadena de caracteres 00. Es por ello por lo que, a la hora de diseñar la aplicación, se debe de dar soporte a esta situación, por lo que la aplicación debe ignorar este tipo de transmisión.

4. Especificación y diseño de la aplicación

A la hora de diseñar una aplicación hay que tener en cuenta diversos aspectos como pueden ser las acciones que va a poder realizar el usuario final en esta. También se debe poseer una idea de la estructuración de la APP y, por último, realizar una serie de diseños iniciales de cómo será la interfaz de usuario. Todos estos aspectos se consiguen mediante el diagrama de casos de uso, el diagrama de clases y los *mockups*. A continuación, se mostrará cada uno de estos aplicado a la aplicación en cuestión.

4.1. Diagrama de casos de uso

Un diagrama de casos de uso es una representación visual de cómo debería comportarse un sistema mediante la interacción con un usuario u otro sistema, es decir, un diagrama que muestra las relaciones entre los actores y los casos de uso de un sistema. Por lo tanto, es una herramienta de ayuda a la hora de planificar cualquier sistema.

En este caso, se ha diseñado un diagrama de casos de uso que tiene el aspecto que se puede visualizar en la Figura 14.

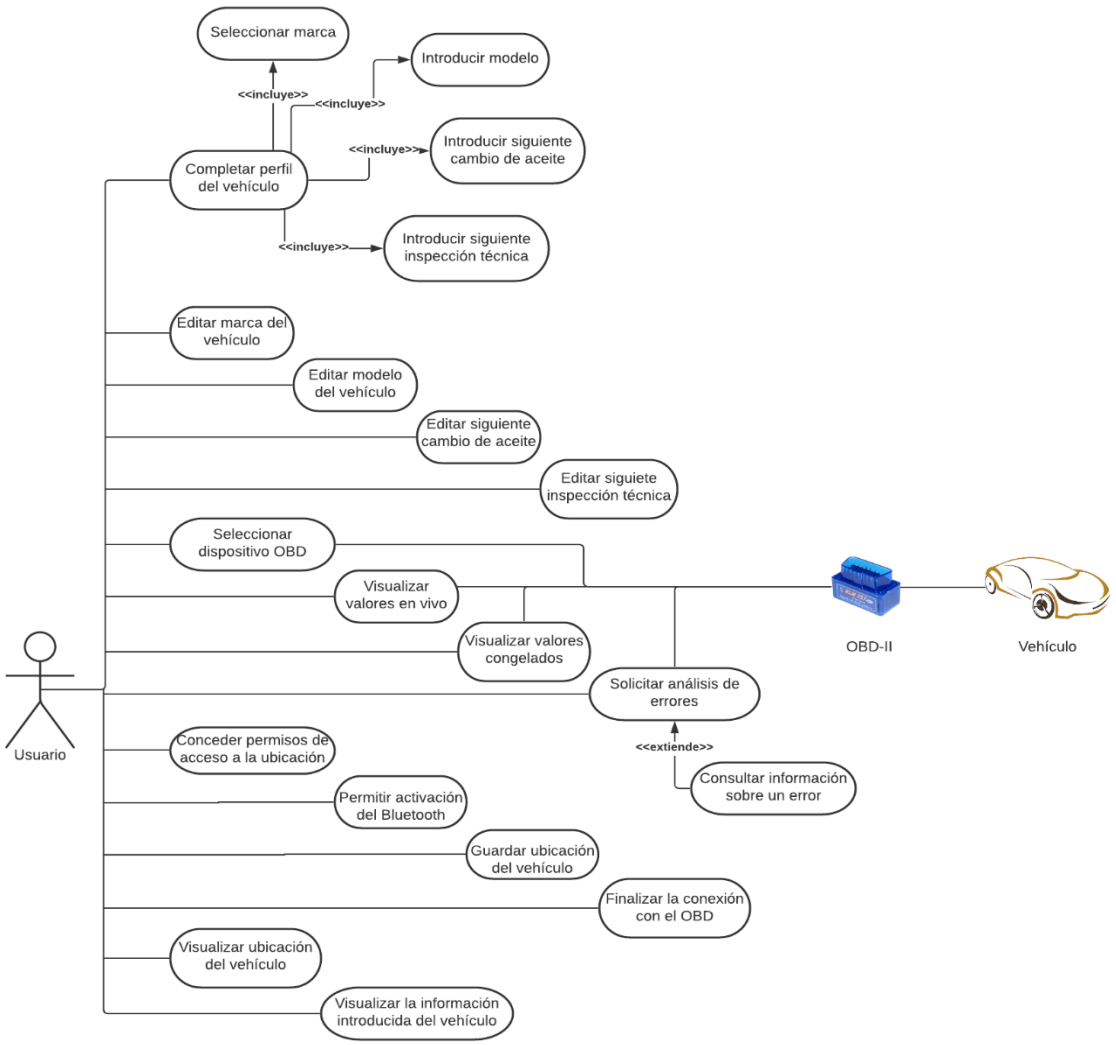


Figura 14: Diagrama de casos de uso

Tal y como se ha podido observar, en el diagrama se representan las acciones que puede realizar cada uno de los actores del sistema. El primero de los actores, el Usuario, es el que más acciones puede realizar, pudiendo completar un perfil del vehículo que utiliza y editarlo, solicitar información diversa y realizar otro tipo de acciones relacionadas con la APP y el teléfono móvil. El otro de los actores, el conector OBDII, simplemente se encarga de proporcionar la información que circula por el vehículo, siendo esta errores o valores de los sensores, por ejemplo.

Para proporcionar una visión más específica de cada uno de los casos de uso, a continuación, se van a mostrar las tablas de descripción de cada caso.

Tabla 6: Caso de uso "Completar perfil del vehículo"

Caso de uso	Completar perfil del vehículo
Actores	Usuario
Propósito	Crear un perfil del vehículo que se utiliza
Resumen	El usuario podrá rellenar una serie de datos sobre su vehículo, los cuales serán almacenados por la aplicación
Precondiciones	El usuario se encuentra en la pantalla de ajustes
Postcondiciones	Se almacena el perfil del vehículo
Incluye	Seleccionar marca, introducir modelo, introducir siguiente cambio de aceite e introducir siguiente inspección técnica
Extiende de	-
Hereda de	-
Flujo principal	1- El usuario se dirige a la pantalla de ajustes de la APP 2- El usuario completa todos los campos requeridos 3- El perfil del vehículo se ha creado y almacenado
Flujo alternativo	1- El usuario se dirige a la pantalla de ajustes de la APP 2- El usuario no completa el perfil o solo completa ciertos campos 3- El perfil no se guarda o lo hace parcialmente, según corresponda

Tabla 7: Caso de uso "Seleccionar marca"

Caso de uso	Seleccionar marca
Actores	Usuario
Propósito	Seleccionar cuál es la marca del vehículo que se utiliza
Resumen	El usuario podrá seleccionar entre las distintas marcas disponibles en la APP. Al finalizar se almacenará la información.
Precondiciones	El usuario se encuentra en la pantalla de ajustes
Postcondiciones	La marca seleccionada se ha almacenado
Incluye	-
Extiende de	-
Hereda de	-
Flujo principal	1- El usuario se encuentra en la pantalla de ajustes 2- El usuario ha pulsado sobre el campo marca 3- El usuario ha seleccionado una de las opciones 4- La marca seleccionada se ha almacenado
Flujo alternativo	1- El usuario se encuentra en la pantalla de ajustes 2- El usuario ha pulsado sobre el campo marca 3- El usuario no selecciona ninguna opción o cancela la acción 4- No se almacena la marca

Tabla 8: Caso de uso "Introducir modelo"

Caso de uso	Introducir modelo
Actores	Usuario
Propósito	Introducir cuál es el modelo del vehículo que se utiliza
Resumen	El usuario podrá introducir el modelo de su vehículo. Al finalizar se almacenará la información
Precondiciones	El usuario se encuentra en la pantalla de ajustes
Postcondiciones	El valor introducido se ha almacenado
Incluye	-
Extiende de	-
Hereda de	-
Flujo principal	1- El usuario se encuentra en la pantalla de ajustes 2- El usuario ha pulsado sobre el campo modelo 3- El usuario introduce el valor 4- El modelo se ha almacenado
Flujo alternativo	1- El usuario se encuentra en la pantalla de ajustes 2- El usuario ha pulsado sobre el campo modelo 3- El usuario no introduce ningún valor o cancela la acción 4- No se almacena la modelo

Tabla 9: Caso de uso "Introducir siguiente cambio de aceite"

Caso de uso	Introducir siguiente cambio de aceite
Actores	Usuario
Propósito	Introducir en cuántos kilómetros hay que realizar el siguiente cambio de aceite
Resumen	El usuario podrá rellenar en cuántos kilómetros debe realizar el siguiente cambio de aceite del vehículo que utiliza. Al finalizar se almacenará la información
Precondiciones	El usuario se encuentra en la pantalla de ajustes
Postcondiciones	El valor introducido se ha almacenado
Incluye	-
Extiende de	-
Hereda de	-
Flujo principal	1- El usuario se encuentra en la pantalla de ajustes 2- El usuario ha pulsado sobre el campo siguiente cambio de aceite 3- El usuario ha introducido un valor 4- La información introducida se almacena
Flujo alternativo	1- El usuario se encuentra en la pantalla de ajustes 2- El usuario ha pulsado sobre el campo siguiente cambio de aceite 3- El usuario no introduce ningún valor o cancela la acción 4- No se almacena la información

Tabla 10: Caso de uso "Introducir siguiente inspección técnica"

Caso de uso	Introducir siguiente inspección técnica
Actores	Usuario
Propósito	Introducir la fecha de la siguiente inspección
Resumen	El usuario podrá introducir la fecha de la siguiente inspección técnica del vehículo. Al finalizar se almacenará la información

Precondiciones	El usuario se encuentra en la pantalla de ajustes
Postcondiciones	El valor introducido se ha almacenado
Incluye	-
Extiende de	-
Hereda de	-
Flujo principal	1- El usuario se encuentra en la pantalla de ajustes 2- El usuario ha pulsado sobre el campo siguiente inspección técnica 3- El usuario ha introducido un valor 4- La información introducida se almacena
Flujo alternativo	1- El usuario se encuentra en la pantalla de ajustes 2- El usuario ha pulsado sobre el campo siguiente inspección técnica 3- El usuario no introduce ningún valor o cancela la acción 4- No se almacena la información

Tabla 11: Caso de uso "Editar marca del vehículo"

Caso de uso	Editar marca del vehículo
Actores	Usuario
Propósito	Modificar la marca del vehículo
Resumen	El usuario podrá modificar la marca del vehículo por una nueva. Al finalizar se almacenará la información
Precondiciones	El usuario se encuentra en la pantalla de ajustes
Postcondiciones	La información almacenada se ha actualizado
Incluye	-
Extiende de	-
Hereda de	-
Flujo principal	1- El usuario se encuentra en la pantalla de ajustes 2- El usuario ha pulsado sobre el campo marca 3- El usuario ha seleccionado una de las opciones 4- La marca almacenada se ha actualizado
Flujo alternativo	1- El usuario se encuentra en la pantalla de ajustes 2- El usuario ha pulsado sobre el campo marca 3- El usuario no selecciona ninguna opción o cancela la acción 4- No se actualiza la marca

Tabla 12: Caso de uso "Editar modelo del vehículo"

Caso de uso	Editar modelo del vehículo
Actores	Usuario
Propósito	Modificar el modelo del vehículo
Resumen	El usuario podrá modificar el modelo del vehículo que se introdujo por uno nuevo. Al finalizar se almacenará la información
Precondiciones	El usuario se encuentra en la pantalla de ajustes
Postcondiciones	La información almacenada se ha actualizado
Incluye	-
Extiende de	-
Hereda de	-
Flujo principal	1- El usuario se encuentra en la pantalla de ajustes 2- El usuario ha pulsado sobre el campo modelo 3- El usuario ha introducido un valor 4- La información almacenada se ha actualizado

Flujo alternativo	1- El usuario se encuentra en la pantalla de ajustes 2- El usuario ha pulsado sobre el campo modelo 3- El usuario no introduce ningún valor o cancela la acción 4- No se actualiza el modelo

Tabla 13: Caso de uso "Editar siguiente cambio de aceite"

Caso de uso	Editar siguiente cambio de aceite
Actores	Usuario
Propósito	Modificar los kilómetros del siguiente cambio de aceite
Resumen	El usuario podrá modificar en cuántos kilómetros se ha de realizar el siguiente cambio de aceite. Al finalizar se almacenará la información
Precondiciones	El usuario se encuentra en la pantalla de ajustes
Postcondiciones	La información almacenada se ha actualizado
Incluye	-
Extiende de	-
Hereda de	-
Flujo principal	1- El usuario se encuentra en la pantalla de ajustes 2- El usuario ha pulsado sobre el campo siguiente cambio de aceite 3- El usuario ha introducido un valor 4- La información almacenada se ha actualizado
Flujo alternativo	1- El usuario se encuentra en la pantalla de ajustes 2- El usuario ha pulsado sobre el campo siguiente cambio de aceite 3- El usuario no introduce ningún valor o cancela la acción 4- No se actualiza la información

Tabla 14: Caso de uso "Editar siguiente inspección técnica"

Caso de uso	Editar siguiente inspección técnica
Actores	Usuario
Propósito	Modificar la fecha de la siguiente inspección
Resumen	El usuario podrá modificar la fecha en la que se ha de realizar la siguiente inspección del vehículo. Al finalizar se almacenará la información
Precondiciones	El usuario se encuentra en la pantalla de ajustes
Postcondiciones	La información almacenada se ha actualizado
Incluye	-
Extiende de	-
Hereda de	-
Flujo principal	1- El usuario se encuentra en la pantalla de ajustes 2- El usuario ha pulsado sobre el campo siguiente inspección técnica 3- El usuario ha introducido un valor 4- La información almacenada se ha actualizado
Flujo alternativo	1- El usuario se encuentra en la pantalla de ajustes 2- El usuario ha pulsado sobre el campo siguiente inspección técnica 3- El usuario no introduce ningún valor o cancela la acción 4- No se actualiza la información

Tabla 15: Caso de uso "Solicitar análisis de errores"

Caso de uso	Solicitar análisis de errores
Actores	Usuario
Propósito	Visualizar los errores presentes en el vehículo
Resumen	El usuario podrá realizar la petición de los errores que presenta el vehículo. Cuando finalice la acción, estos, si es que los hay, se podrán visualizar sobre la interfaz
Precondiciones	El Bluetooth está activado en el dispositivo móvil y la aplicación está conectada al dispositivo OBD
Postcondiciones	Se muestran los errores presentes o se informa de que no los hay, según corresponda
Incluye	-
Extiende de	-
Hereda de	-
Flujo principal	1- El usuario se encuentra en la pantalla de errores 2- El usuario ha pulsado sobre el botón de búsqueda 3- Se muestra la información relativa a los errores según corresponda
Flujo alternativo	-

Tabla 16: Caso de uso "Visualizar valores en vivo"

Caso de uso	Visualizar valores en vivo
Actores	Usuario
Propósito	Observar la evolución de determinados valores
Resumen	El usuario podrá realizar la petición de visualización en vivo de determinados valores del vehículo. Los cambios serán reflejados en la interfaz
Precondiciones	El Bluetooth está activado en el dispositivo móvil y la aplicación está conectada al dispositivo OBD
Postcondiciones	Se muestra la información correspondiente en la pantalla
Incluye	-
Extiende de	-
Hereda de	-
Flujo principal	1- El usuario se encuentra en la pantalla de valores en vivo 2- El usuario ha pulsado sobre el botón de comenzar análisis 3- Se muestra la información en la pantalla
Flujo alternativo	-

Tabla 17: Caso de uso "Visualizar valores congelados"

Caso de uso	Visualizar valores congelados
Actores	Usuario
Propósito	Modificar el modelo del vehículo
Resumen	El usuario podrá realizar la petición de visualización en un momento determinado de algunos valores del vehículo. Los cambios serán reflejados en la interfaz
Precondiciones	El Bluetooth está activado en el dispositivo móvil y la aplicación está conectada al dispositivo OBD
Postcondiciones	Se muestra la información correspondiente en la pantalla
Incluye	-
Extiende de	-

Hereda de	-
Flujo principal	1- El usuario se encuentra en la pantalla de valores congelados 2- Se muestra la información en la pantalla
Flujo alternativo	-

Tabla 18: Caso de uso "Conceder permisos de acceso a la ubicación"

Caso de uso	Conceder permisos de acceso a la ubicación
Actores	Usuario
Propósito	Proporcionar a la aplicación el poder de acceder la ubicación del teléfono móvil
Resumen	El usuario podrá conceder permiso a la aplicación para que esta pueda utilizar los servicios de ubicación para determinadas acciones
Precondiciones	La aplicación no tiene permisos para acceder a la ubicación
Postcondiciones	Se puede acceder a la ubicación del dispositivo
Incluye	-
Extiende de	-
Hereda de	-
Flujo principal	1- El usuario inicia la aplicación por primera vez 2- El usuario concede los permisos 3- Se almacena la configuración
Flujo alternativo	1- El usuario inicia la aplicación por primera vez 2- El usuario no concede los permisos 3- Se almacena la configuración

Tabla 19: Caso de uso "Permitir la activación del Bluetooth"

Caso de uso	Permitir la activación del Bluetooth
Actores	Usuario
Propósito	Permitir que la aplicación active el Bluetooth
Resumen	El usuario podrá conceder permiso a la aplicación que active el Bluetooth. Al finalizar se activará el Bluetooth
Precondiciones	El Bluetooth está desactivado
Postcondiciones	Se ha activado el Bluetooth
Incluye	-
Extiende de	-
Hereda de	-
Flujo principal	1- El usuario pulsa sobre el botón para conectarse por Bluetooth al OBD 2- Se cuestiona al usuario si desea que la aplicación active el Bluetooth 3- El usuario da su autorización 4- El Bluetooth se activa
Flujo alternativo	1- El usuario pulsa sobre el botón para conectarse por Bluetooth al OBD 2- Se cuestiona al usuario si desea que la aplicación active el Bluetooth 3- El usuario no da su autorización 4- El Bluetooth no se activa

Tabla 20: Caso de uso "Guardar ubicación del vehículo"

Caso de uso	Guardar ubicación del vehículo
Actores	Usuario
Propósito	Almacenar dónde se encuentra el vehículo
Resumen	El usuario podrá almacenar en la aplicación el lugar donde ha estacionado su vehículo. Al finalizar se representará la ubicación en la interfaz
Precondiciones	La aplicación tiene permisos para acceder a la ubicación y el servicio está activo
Postcondiciones	La ubicación se ha almacenado y representado en la pantalla
Incluye	-
Extiende de	-
Hereda de	-
Flujo principal	1- El usuario se encuentra en la pantalla de inicio 2- El usuario ha pulsado sobre el botón de guardar la ubicación 3- Se almacena el valor y se representa en la pantalla
Flujo alternativo	-

Tabla 21: Caso de uso "Finalizar la conexión con el OBD"

Caso de uso	Finalizar la conexión con el OBD
Actores	Usuario
Propósito	Desconectarse del dispositivo OBD al que está conectado la APP
Resumen	El usuario podrá finalizar la conexión con el dispositivo al que esté conectado de manera manual
Precondiciones	Se está conectado a un dispositivo
Postcondiciones	La conexión con el OBD queda finalizada
Incluye	-
Extiende de	-
Hereda de	-
Flujo principal	1- El usuario pulsa sobre el botón de finalizar conexión Bluetooth 2- La conexión finaliza
Flujo alternativo	-

Tabla 22: Caso de uso "Visualizar ubicación del vehículo"

Caso de uso	Visualizar ubicación del vehículo
Actores	Usuario
Propósito	Conocer la ubicación del vehículo en el mapa
Resumen	El usuario podrá consultar dónde se encuentra su vehículo por medio de la aplicación de mapas
Precondiciones	El usuario ha guardado la ubicación de su vehículo previamente y se dispone de la aplicación de mapas
Postcondiciones	Se muestra la ubicación en el mapa
Incluye	-
Extiende de	-
Hereda de	-
Flujo principal	1- El usuario se encuentra en la pantalla de inicio 2- El usuario pulsa sobre el botón de visualizar ubicación 3- Se representa la ubicación en la aplicación de mapas
Flujo alternativo	-

Tabla 23: Caso de uso "Consultar información sobre un error"

Caso de uso	Consultar información sobre un error
Actores	Usuario
Propósito	Obtener más información sobre un error
Resumen	El usuario podrá conocer más información y saber en qué consiste un error que se encuentre en su vehículo. Se abrirá una página en el navegador para obtener esa información
Precondiciones	Se ha realizado el análisis de errores y se ha encontrado alguno
Postcondiciones	Se muestra una página web con información sobre ese error
Incluye	-
Extiende de	-
Hereda de	-
Flujo principal	1- El usuario se encuentra en la pantalla de errores 2- El usuario pulsa sobre uno de los errores 3- Se muestra la página web específica de ese error
Flujo alternativo	-

Tabla 24: Caso de uso "Visualizar la información introducida del vehículo"

Caso de uso	Visualizar la información introducida del vehículo
Actores	Usuario
Propósito	Permite ver los datos introducidos al completar el perfil del vehículo
Resumen	El usuario podrá visualizar en la interfaz de la aplicación todo lo que haya introducido cuando se ha completado el perfil del vehículo
Precondiciones	Se ha completado algunos o todos los campos del perfil del vehículo
Postcondiciones	-
Incluye	-
Extiende de	-
Hereda de	-
Flujo principal	-
Flujo alternativo	-

Con todo esto, se obtiene una representación clara de las funciones que se realizarán en el sistema diseñado y, en concreto, de las acciones que va a poder realizar el usuario en la aplicación.

4.2. Diagrama de clases

Un diagrama de clases es una representación gráfica que sirve para comunicar el diseño de un programa orientado a objetos. Representa las relaciones entre las distintas clases existentes en el sistema proporcionando, por tanto, una visión más general y clarificadora de este.

En el caso de la aplicación que incumbe este trabajo, se ha desarrollado un diagrama de clases con el que visualizar la estructura de dicha aplicación además de los atributos y métodos que presenta cada clase. Como resultado, se ha obtenido el diagrama que se puede observar en la Figura 15.

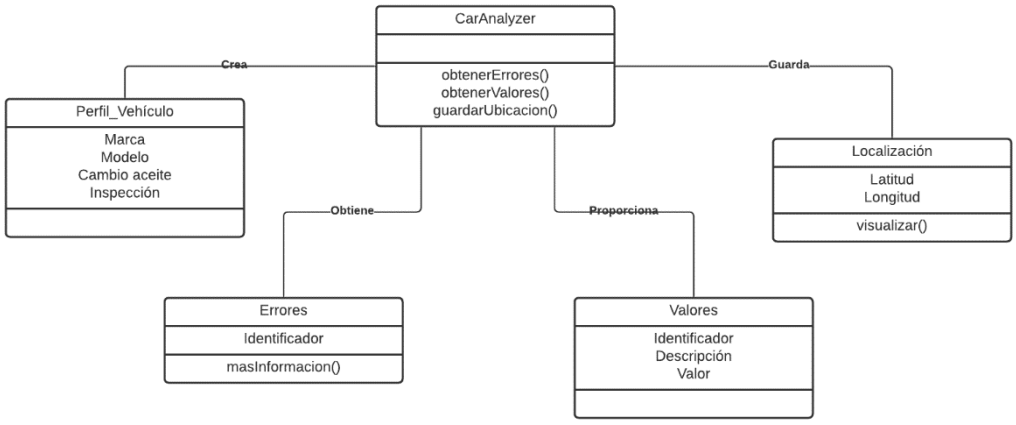


Figura 15: Diagrama de clases

Así pues, se puede ver como la aplicación obtiene y almacena datos de diferentes tipos. Se puede identificar que la aplicación proporcionará los errores y valores que estén presentes en el vehículo, teniendo de estos como principal atributo el identificador. También, en la aplicación se creará un perfil del vehículo con distintos atributos. Por último, hay que comentar que en la APP también se podrá almacenar la ubicación del vehículo mediante los atributos de la latitud y la longitud.

4.3. Diseño de la interfaz y esquema de navegación

Antes de abordar el diseño de la APP, se realizaron una serie de *mockups*, los cuales sirvieron de modelo e inspiración para la elaboración final de la interfaz que presenta la aplicación. Para ello se utilizó la herramienta NinjaMock [11] con la que se realizaron diseños hasta encontrar un interfaz que se adecuara a los requisitos. De tal manera, los *mockups* que se elaboraron fueron los siguientes:

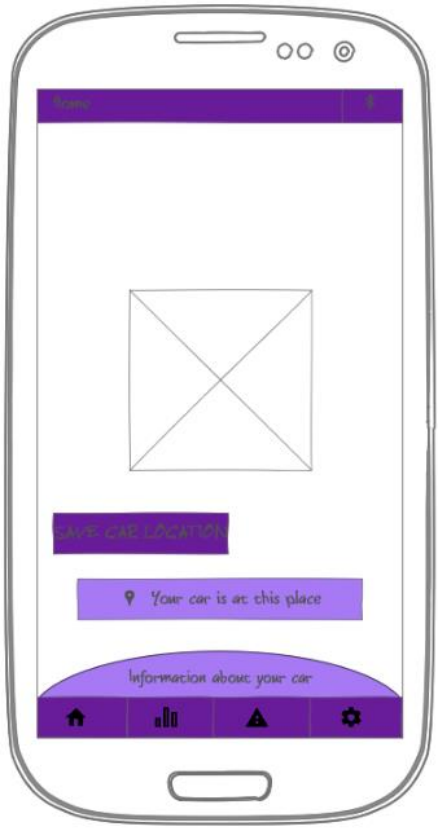


Figura 19: Diseño pantalla de inicio

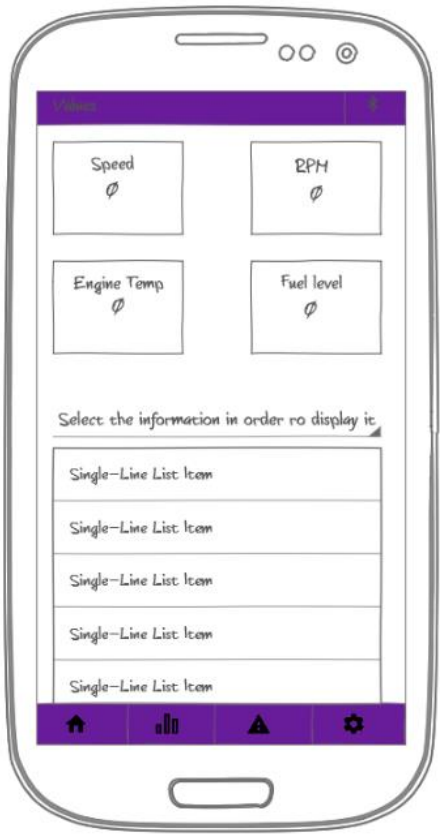


Figura 18: Diseño pantalla de valores

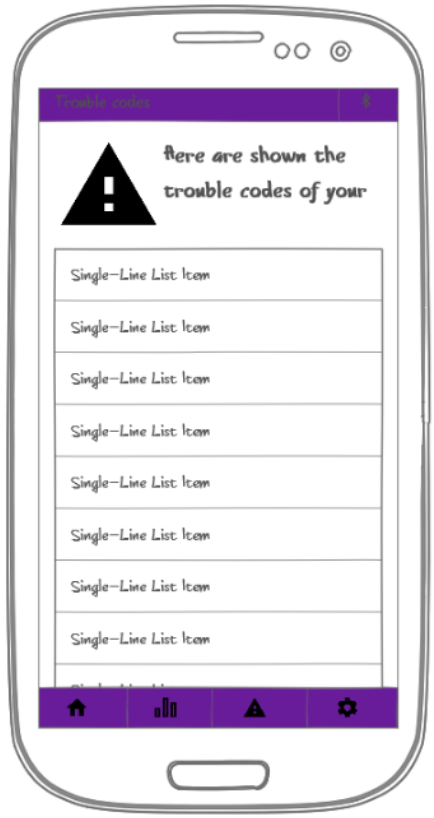


Figura 17: Diseño pantalla de errores

Figura 16: Diseño pantalla de ajustes

Estos diseños corresponden a las cuatro pantallas de la aplicación, en las que se pueden realizar funcionalidades distintas y hay que destacar que, aunque en ellos aparece un tipo de letra, este no será el de la APP final, el cual será del tipo *sans-serif-medium* o *sans-serif* de terminación redonda. Este tipo de letra es de los más comunes entre las APPs del mercado, pues es un tipo de letra especialmente diseñado para facilitar la lectura y que, además, proporciona sensación de sobriedad, modernidad, alegría y seguridad. También se especificó la paleta de colores a utilizar, en la que se definen diversos colores tanto para los modos claro y oscuro de la aplicación. De este modo, los colores seleccionados son los que se pueden visualizar en la Figura 20.

Figura 20: Paleta de colores

La paleta de colores que se ha elegido no es muy común entre las aplicaciones móviles y, en concreto, no lo es en las aplicaciones que poseen la misma temática que la que se desarrolla. Por tanto, con esta elección se consiguen unos colores identificativos y con los cuales los usuarios pueden diferenciar esta APP de la competencia.

Además del diseño de las pantallas principales de la APP, también se diseñó el aspecto que tendrían los diferentes diálogos que aparecerían en la aplicación, siendo estos los que se pueden observar en las imágenes siguientes:

Figura 24: Diseño diálogo concesión de permisos de ubicación

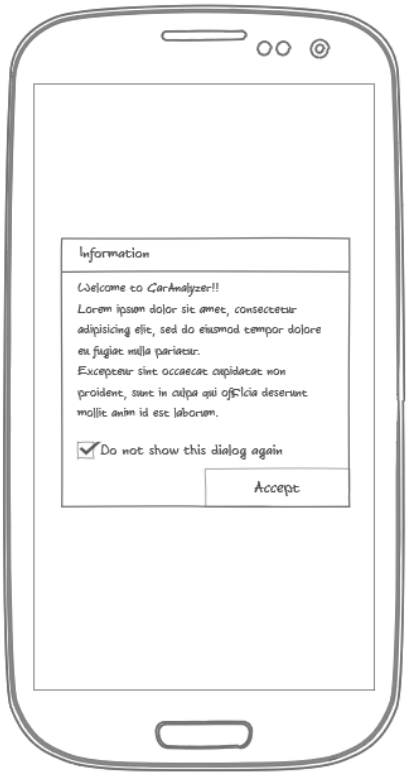


Figura 23: Diseño diálogo inicial

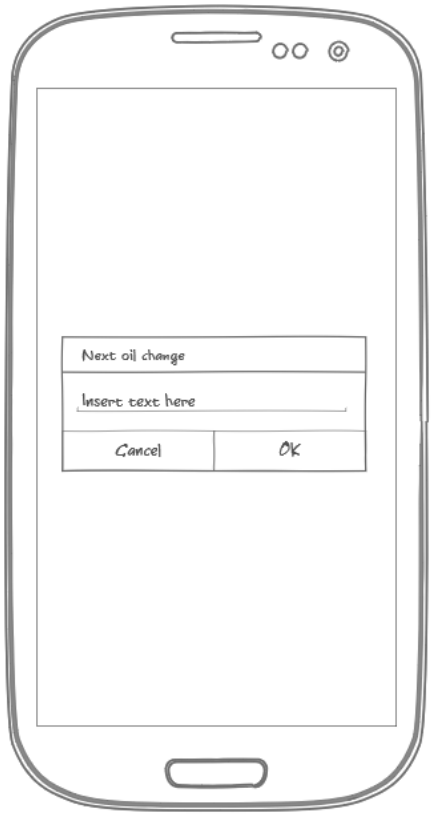


Figura 21: Diseño diálogo introducción de siguiente cambio de aceite

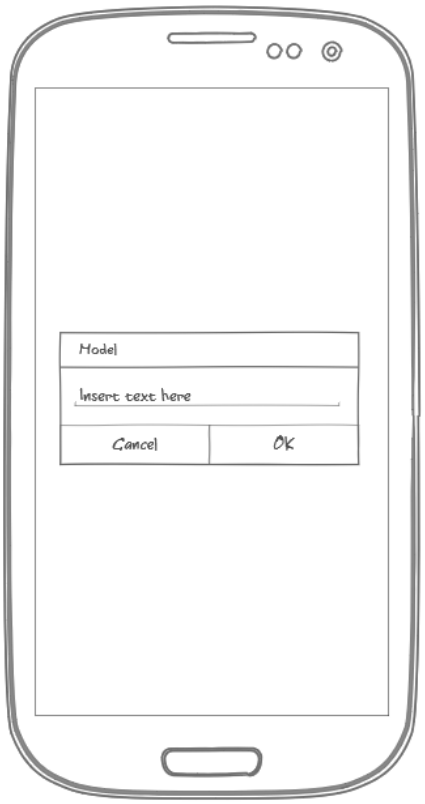


Figura 22: Diseño diálogo introducción de modelo

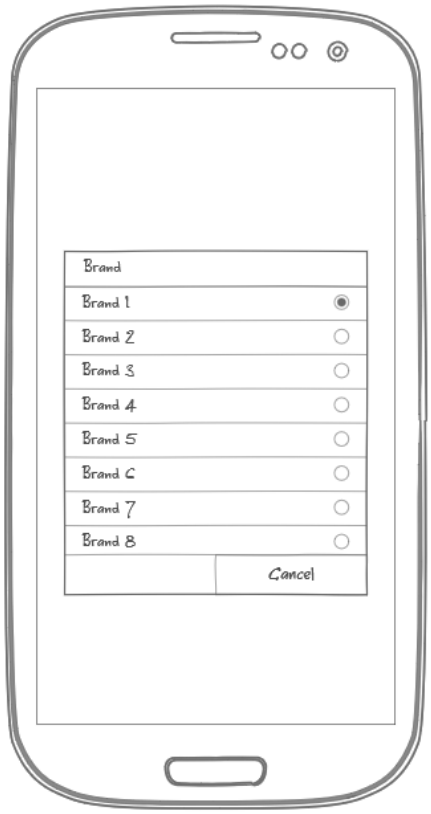


Figura 27: Diseño diálogo selección de marca

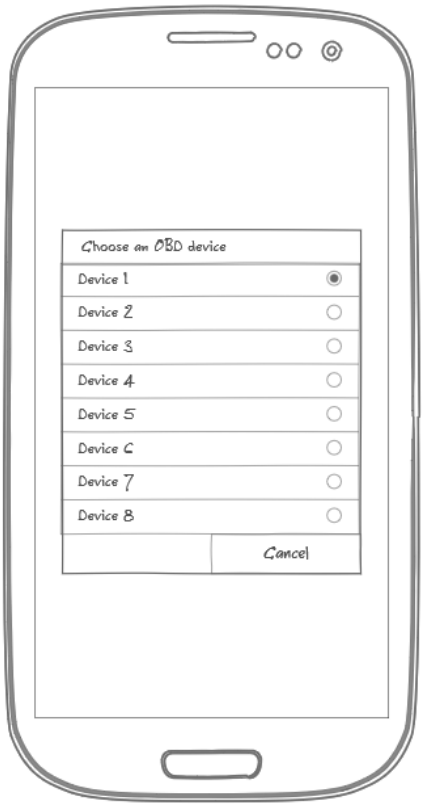


Figura 26: Diseño diálogo selección de OBD

Figura 25: Diseño diálogo introducción de siguiente inspección técnica

Vistos los diseños que se desarrollaron para la implementación de la APP, hay que comentar el esquema de navegación de esta. El patrón de navegación que presenta es el de una barra de navegación inferior con la que se puede fluctuar entre las distintas pantallas de la aplicación. Además, existen diversos diálogos los cuales cargan realizando determinadas acciones. Por ejemplo, si se desea completar el perfil del vehículo y, en concreto, se desea introducir el modelo de este, se mostrará el diálogo que se ha podido ver en la Figura 24. De este modo, el esquema de navegación es el siguiente:

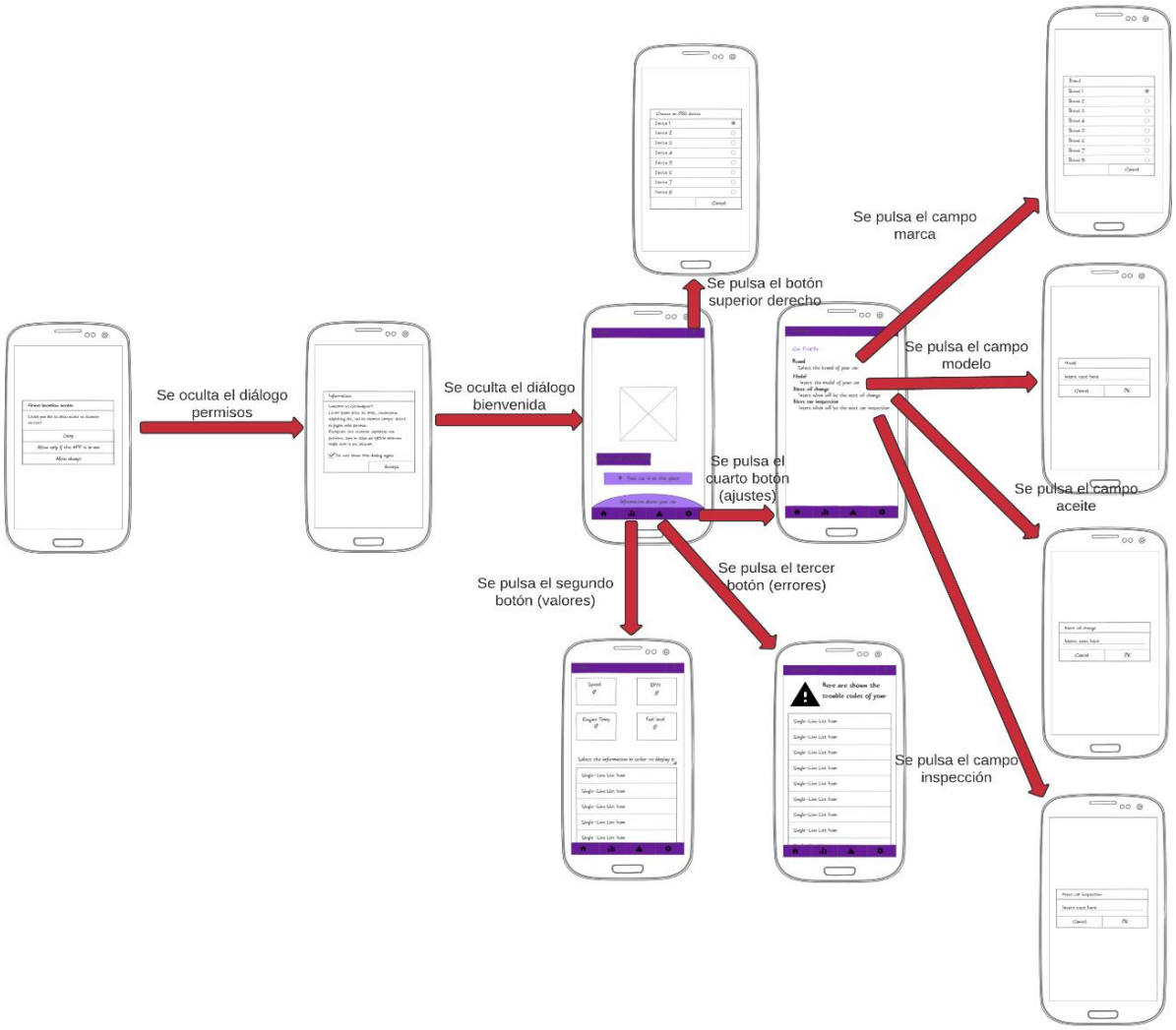


Figura 28: Esquema de navegación en la aplicación

Hay que comentar que, aunque en el esquema anterior no se represente, se puede navegar desde cualquiera de las cuatro pantallas principales a las otras tres y, además, también se puede llegar a mostrar el diálogo de selección del dispositivo OBD si se pulsa sobre el botón desde cualquier pantalla principal.

Por último, se elaboró el diseño del icono de la aplicación que, después de varios diseños iniciales, se optó por un icono que siguiera la estética de la APP y que fuera representativo de esta. El icono es el que se puede ver en la Figura 29.

Figura 29: Icono de la aplicación

5. Arquitecturas desarrolladas

5.1. Arquitectura de la aplicación

Toda aplicación o software que exista en la actualidad presenta una arquitectura en la cual esta se organiza y define el funcionamiento y la interacción entre las distintas partes. Existen diversos modelos de arquitectura como puede ser el *Model View Presenter* (MVP) [12], el *Model View Controller* (MVC) [13] o el *Model View ViewModel* (MVVM) [14] entre otros. Cada uno de ellos representa una arquitectura distinta, pero generalmente, en el ámbito de las aplicaciones para dispositivos móviles, los más utilizados son el MVP y el MVVM, siendo este último el caso que se refleja en la aplicación desarrollada.

El modelo MVVM es un modelo que desacopla, en el mayor grado posible, la interfaz de usuario de la lógica de la aplicación. Una representación gráfica de este modelo es la que se puede visualizar en la Figura 30.

Figura 30: Diagrama arquitectura MVVM

Tal y como se puede observar hay diferentes transmisiones de información entre las distintas entidades de la aplicación. La vista (*View*) es la interfaz de la aplicación; es decir, el lugar en donde se van a ver reflejados todos los datos que se soliciten o todos los cambios que se realicen en la APP. A la hora del desarrollo de la aplicación, todas las clases relacionadas con la interfaz deberán ser fácilmente identificables a simple vista.

La vista lleva asociada de manera indirecta la información presente en el modelo (*Model*). El modelo es el lugar dónde se localizan los datos y la lógica de la aplicación. Nunca realizará ninguna acción en el sistema a desarrollar y aunque, como se ha dicho, tiene relación indirecta con la vista, no existirá nunca una dependencia como tal con esta. Por lo tanto, es el modelo de vista (*ViewModel*) el que actúa como intermediario entre los otros dos componentes. En este caso, en el modelo de vista sí que se encuentra toda la lógica de la aplicación y actúa como una abstracción de la interfaz.

Por lo tanto, en el desarrollo de la APP, se debe de poder identificar de manera clara qué clases de las que se desarrollarán se asociarán con cada uno de los elementos descritos. Todo esto se verá en apartados siguientes.

5.2. Arquitectura general del sistema

Una vez vista la arquitectura de la aplicación, se puede visualizar la arquitectura del sistema en su totalidad. Hay que destacar que en el sistema con el que se ha trabajado hay tres actores principales los cuales son la aplicación desarrollada, el dispositivo OBD que actúa

como puente y el vehículo utilizado. La representación de esta arquitectura se puede visualizar en la imagen siguiente:

Figura 31: Esquema general del sistema

En la imagen se observa cómo la aplicación solicita un PID al dispositivo OBD y este realiza la petición al vehículo el cual le contesta con la información disponible. Por último, el OBD realiza las transformaciones necesarias y envía la respuesta a la APP. De esta manera, se podría decir que la aplicación es la interfaz en la que los datos son representados y el OBD actúa como puente entre las otras dos entidades.

Así pues, para la correcta comunicación entre todas las partes es necesario un hardware específico (el OBD) y que el cliente, en este caso la aplicación, implemente un estándar para la comunicación con el OBD y este, tal y como se describió en apartados anteriores, es el estándar SAEJ1979.

6. Implementación de las funcionalidades de la aplicación

Una vez realizado el recorrido sobre la aplicación se va a pasar a comentar la estructura de esta, es decir, sus clases y métodos implementados para el correcto funcionamiento.

La APP CarAnalyzer se ha elaborado en el entorno de desarrollo Android Studio [15] y con el lenguaje de programación Java [16]. La aplicación está estructurada en diversas capas, separando los elementos de la interfaz de usuario del código interno, tal y como se ha comentado en apartados anteriores. De esta manera, la estructura de la aplicación es la que se puede visualizar en la Figura 40.

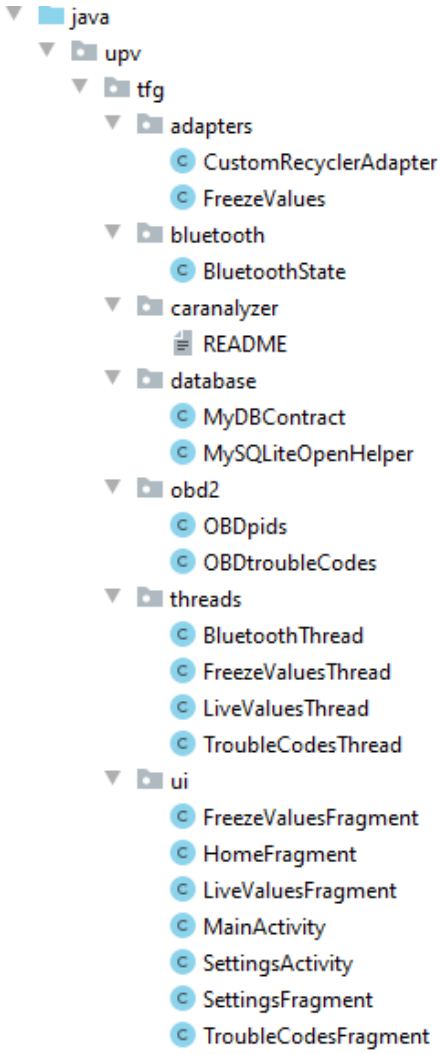


Figura 32: Estructuración de la aplicación

Así pues, tal y como se representa en la imagen, se aplica la separación en capas correspondiente al modelo MVVM mencionado anteriormente. Se puede ver como todas las clases relacionadas con la interfaz de usuario o vista, se encuentran en un mismo paquete denominado “ui”, el modelo se localiza en el paquete “obd2” y los modelos de vista se encuentran en el paquete “threads”. Hay que destacar también la existencia de otros paquetes como el que contiene las clases relacionadas con la base de datos (database), el que mantiene la información del Bluetooth del dispositivo (bluetooth) y aquel que almacena las interfaces definidas para adaptar distintos elementos de la interfaz de usuario (adapters). De este modo, al seguir esta separación de cada uno de los elementos de la aplicación, se

consigue un mayor grado de independencia de la APP entre los distintos elementos, resultando más sencillo para el programador depurar cualquier error que pueda aparecer durante la ejecución.

A la hora de implementar cualquier aplicación en un dispositivo móvil, hay que tener en cuenta que todas tienen un hilo principal que es el que gestiona la interfaz de la aplicación. Dicho hilo es de vital importancia para el correcto funcionamiento de la APP y en él no se pueden realizar tareas que bloqueen su ejecución y que puedan provocar que la aplicación deje de funcionar, dando a entender al usuario que está mal desarrollada. Es por ello por lo que se han creado los distintos hilos secundarios (localizados en el paquete “threads”, tal y como se ha mencionado) los cuales se encargan de la ejecución de las tareas en segundo plano. De esta manera, si ocurre un error en alguno de estos, la aplicación podrá seguir ejecutándose por el usuario sin ningún tipo de problema, ya que el hilo principal, el de la interfaz, es independiente del resto.

También hay que destacar que en la APP se han utilizado distintas bibliotecas externas. Estas van desde la biblioteca de servicio de Google sobre la localización [17], la biblioteca para almacenar las preferencias [18] y todas aquellas relativas al diseño de la interfaz de usuario como la biblioteca Material Design [19]. Todas ellas aportan a la aplicación las distintas funcionalidades y el aspecto que la caracteriza.

Unas de las mayores problemáticas a la hora de desarrollar la aplicación, fueron el establecimiento de la conexión por Bluetooth y la realización de la correcta gestión e interpretación de la información recibida desde el dispositivo OBD, además de la implementación de las otras funcionalidades que se pueden realizar. Es por ello por lo que, para proporcionar una visión de esta problemática, se va a comentar la implementación de cada una de las funcionalidades de la APP.

6.1. Realización de la conexión Bluetooth con el dispositivo OBD

Una de las principales tareas que debe de poder realizar la aplicación, es la conexión Bluetooth con el dispositivo deseado. En primer lugar, se realizó un plan de actuación para abordar esta función y por ello se diseñó un diagrama de flujo el cual se puede ver en la Figura 41.

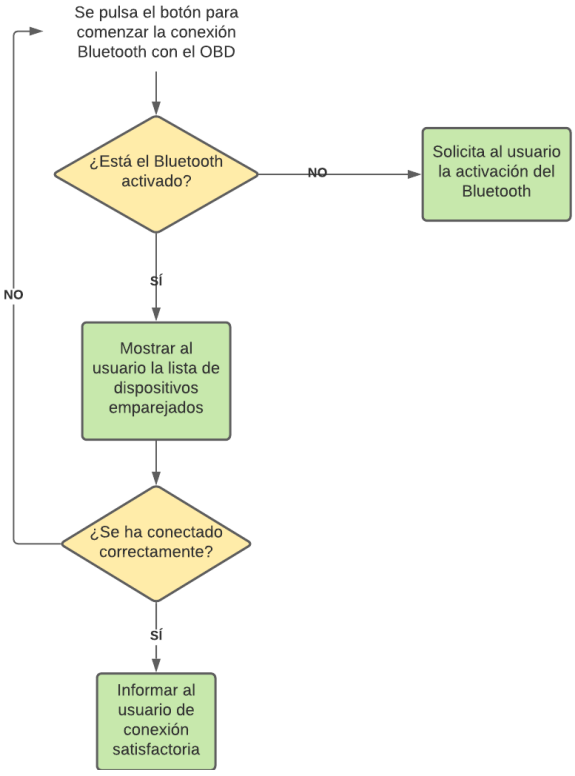


Figura 33: Diagrama de flujo "Conexión con dispositivo OBD"

Una vez definido el plan, se pasó al desarrollo. Para ello, se creó la clase *BluetoothState* en la que se almacena de manera permanente mientras la aplicación está en ejecución, el estado de la conexión. De este modo, lo primero que se realiza es la obtención de una instancia de la clase mencionada. Para ello se creó el método que se puede visualizar a continuación:

```
  /** Method that gets an instance of the BluetoothState */
private void getInstanceBluetoothState(){ this.bluetoothState = new BluetoothState(); }
```

Obtenida la instancia, hay que configurarla, de modo que se creó otro método en el que se realiza todo lo necesario para poner en marcha la conexión.

```
  /** Method that sets up the Bluetooth connection
 * @return boolean
 */
private boolean setUpBluetoothConnection(){
    /* Gets the Bluetooth Adapter */
    BluetoothAdapter bluetoothAdapter = this.bluetoothState.getBluetoothAdapter();
    /* Checks if the Bluetooth of the phone is enabled. If not, requests it */
    if (!bluetoothAdapter.isEnabled()) {
        Intent intent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        startActivity(intent);
    }
    else {
        /* Gets a set of paired devices from the phone */
        Set<BluetoothDevice> pairedDevices = bluetoothAdapter.getBondedDevices();
        /* Array of the names from the paired devices */
        ArrayList<String> deviceNamesArray = new ArrayList<>();
        /* Fills the array with the names */
        for (BluetoothDevice device : pairedDevices) {
            String deviceName = device.getName();
            deviceNamesArray.add(deviceName);
```

```
        }
        /* Stops the discovering of new Bluetooth devices */
        bluetoothAdapter.cancelDiscovery();
        /* Sets up the AlertDialog */
        AlertDialog.Builder selectDeviceAlertDialog = new AlertDialog.Builder(this);
        selectDeviceAlertDialog.setTitle(R.string.bluetoothAlertDialogTitle);
        /* List of the devices that can be choice */
        final ArrayAdapter arrayAdapter = new ArrayAdapter(this,
                android.R.layout.select_dialog_singlechoice);
        /* Fills the list with de device names */
        for (int i = 0; i < deviceNamesArray.size(); i++) {
            arrayAdapter.add(deviceNamesArray.get(i));
        }
        /* Attaches a negative button to the dialog */
        selectDeviceAlertDialog.setNegativeButton(R.string.bluetoothAlertDialogCancel,
                new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                /* Closes the AlertDialog */
                dialog.dismiss();
            }
        });
        /* Puts the list of device names into the dialog */
        selectDeviceAlertDialog.setAdapter(arrayAdapter, new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int _item) {
                /* The name and the MAC address of the selected device */
                String bluetoothDeviceName = "", bluetoothDeviceMAC = "";
                bluetoothDeviceName = deviceNamesArray.get(_item);
                /* Iterates the list of paired devices */
                for (BluetoothDevice device : pairedDevices) {
                    /* Checks if the name of one of them is the name selected on the dialog */
                    if (device.getName().equals(bluetoothDeviceName)) {
                        /* Gets the MAC address of the device */
                        bluetoothDeviceMAC = device.getAddress();
                        /* Sets it to the BluetoothState */
                        bluetoothState.setAddress(bluetoothDeviceMAC);
                        /* Checks if the connection is successful. If it is, shows a message
                            and starts the thread */
                        if (bluetoothState.getConnection()) {
                            Toast.makeText(MainActivity.this,
                                    R.string.successfullyConnected, Toast.LENGTH_LONG).show();
                            bluetoothThread = new BluetoothThread(bluetoothState);
                            bluetoothThread.start();
                            bluetoothThread.setIfStarted(true);
                            miBluetooth.setIcon(R.drawable.ic_bluetooth_on);
                        }
                    }
                }
            }
        });
        /* Shows the dialog */
        selectDeviceAlertDialog.show();
        return true;
    }
    return false;
}
```

Tal y como se puede observar, primeramente, se obtiene el adaptador Bluetooth presente en el dispositivo móvil que se está utilizando y, seguidamente, se comprueba si el Bluetooth está activado. En caso de que no lo esté se le preguntará al usuario si desea que la aplicación active el mencionado servicio. Una vez realizada esta comprobación, se obtienen los dispositivos emparejados del *smartphone* de los cuales se almacena su nombre en un

array. Seguidamente se cancela la búsqueda de nuevos dispositivos Bluetooth (esto es necesario para que el teléfono no siga buscando nuevos dispositivos a los que conectarse mientras se establece la conexión con el dispositivo deseado, pues podría ocasionar un error además de la carga que supondría realizar las dos acciones a la vez) y se configura un diálogo que se mostrará al usuario en el que se podrá seleccionar el dispositivo al que desea conectarse. El usuario seleccionará uno de estos, se comprobará que este está entre los dispositivos emparejados y, si lo está, se obtendrá la dirección MAC. Esta dirección se le pasará a la clase *BluetoothState* para que mantenga el estado y para finalizar, se informará al usuario por medio de un mensaje flotante que la conexión se ha realizado correctamente y se lanzará el hilo que gestiona los mensajes que se envían y reciben, que pertenece a la clase *BluetoothThread*.

6.2. Solicitud de los valores presentes en el vehículo

Una vez establecida la conexión con el dispositivo, ya se puede solicitar la información que se desee visualizar sobre el vehículo. Para ello, existen diferentes clases las cuales actúan como puente entre la interfaz de usuario y los mensajes recibidos en el gestor presente en la clase *BluetoothThread*. Estas clases de las que se habla son hilos de ejecución independientes al hilo de ejecución principal de la aplicación y encontramos tres tipos: el *LiveValuesThread*, el *FreezeValuesThread* y el *TroubleCodesThread*.

Antes de comenzar el desarrollo, igual que en la ocasión anterior, se definió un diagrama de flujo para clarificar la forma de actuar delante del problema de la gestión de los menajes. El diagrama es el que se puede visualizar a continuación:

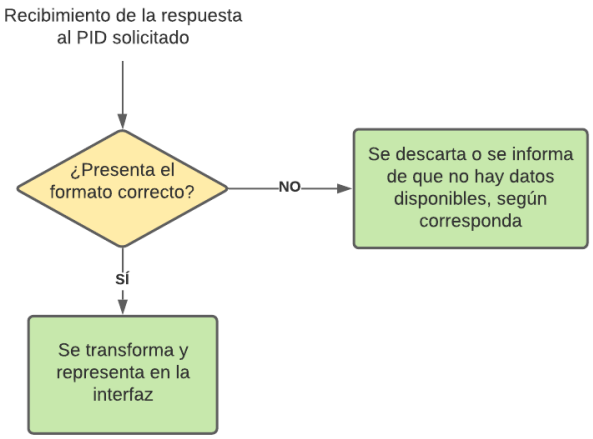


Figura 34: Diagrama de flujo "Gestión de los mensajes"

Teniendo claro el plan de actuación, se pasó a la implementación. En primer lugar, el hilo denominado *LiveValuesThread* es el que se encarga de gestionar la información que es necesaria representar en la pantalla de valores en vivo de la aplicación. Este hilo tiene sus propias variables privadas con las que gestiona todo lo relacionado a los PIDs que se envían y la información que se recibe. Posee un atributo en el que se almacena la lista de PIDs que debe de enviar, la cual la obtiene de la clase *OBDpids* (esta clase es la que presenta todos los PIDs que se pueden enviar al dispositivo OBD). De este modo, se va iterando sobre la lista de manera cíclica para ir solicitando la información correcta en cada instante. El hilo realiza una consulta del estado del mensaje cada 100 ms y, cuando este está listo, la información se

obtiene y es transformada a un valor decimal (dado que se recibe en hexadecimal, tal y como se describió en un apartado anterior) por medio de un método presente en la clase *OBDpids* mencionada y del cual se hablará más adelante. Una vez transformada la información, simplemente hay que depositarla en la interfaz de usuario. Por tal de clarificar, el código de ejecución del hilo es el siguiente:

```
   /** The instructions to be executed by the thread */
@Override
public void run() {
    while (this.start){
        try {
            /* Send the message */
            sendMessage();
            /* Stores the pid sent */
            String pid = this.pid;
            /* Holds until the message is ready */
            waitMessageStateChange();
            /* Gets the message */
            String decData = getMessage(this.bluetoothThread.getLastMessage());
            /* Gets if the thread has to stop or continue */
            this.start = this.reference.get().hasToStop();
            /* If the user is not in the fragment or decided not to continue,
            stops the thread */
            if(!this.start || this.reference.get().getActivity() == null){
                break;
            }
            /* Calls the method in order to update the UI */
            this.reference.get().getActivity().runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    reference.get().updateUI(decData, pid);
                }
            });
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

En segundo lugar, el hilo denominado *FreezeValuesThread* se encarga de la gestión de la información que se va a proyectar en la pantalla valores congelados. Este hilo realizaría la misma función que el anterior, pero utiliza otros PIDs para la obtención de información distinta. El código que ejecuta es el que se puede visualizar a continuación:

```
   /** The instructions to be executed by the thread */
@Override
public void run() {
    while (this.start){
        try {
            /* Send the message */
            sendMessage();
            /* Holds until the message is ready */
            waitMessageStateChange();
            /* Gets the message */
            String decData = getMessage(this.bluetoothThread.getLastMessage());
            /* Gets if the thread has to stop or continue */
            this.start = this.reference.get().hasToStop();
            /* If the user is not in the fragment or decided not to continue,
            stops the thread */
            if(!this.start || this.reference.get().getActivity() == null){
```

```
                break;
            }
            this.reference.get().getActivity().runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    reference.get().updateUI(decData);
                }
            });
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Se puede observar que el código es casi idéntico al del hilo anterior, por lo que se podría haber unificado todo en un mismo hilo simplemente modificando el nombre de un par de métodos e introduciendo unas pequeñas modificaciones. Pero la razón de realizar una implementación independiente es que, de cara a la vista del programador, es mucho más sencillo de depurar algún posible error que pueda ocurrir en la aplicación si se realiza esta implementación. Además, dado que cada hilo gestiona una pantalla de la aplicación distinta, tiene sentido que exista un hilo para cada una de ellas.

Por último, el hilo denominado *TroubleCodesThread*, se encarga de la gestión de la pantalla de códigos de error. En este caso el plan para abordar la problemática es distinto, dado que la información que proporciona el OBD también lo es. Para ello se realizó el siguiente diagrama:

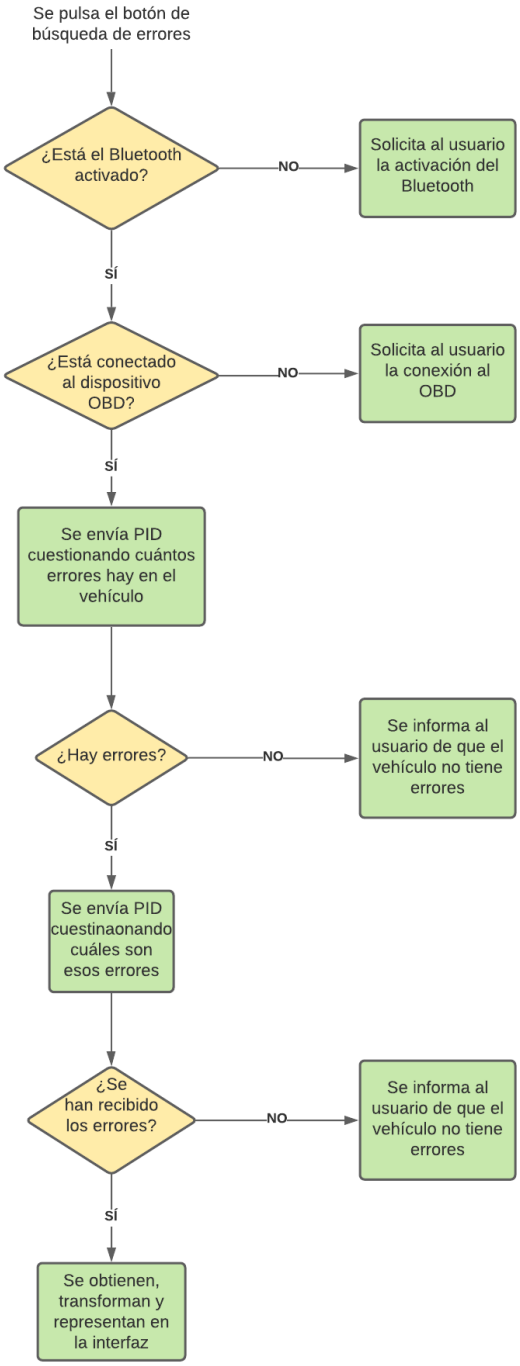


Figura 35: Diagrama de flujo "Funcionamiento de la búsqueda de errores"

Por lo tanto, en este hilo se pregunta en primer lugar si existe algún tipo de error en el vehículo. En caso negativo, no realizará ninguna acción más; pero si existe algún error solicitará los identificadores de estos. Una vez obtenido alguno, se transformará de manera que pueda ser interpretable para los usuarios por medio de un método presente en la clase *OBDtroubleCodes*, que se describirá más adelante, e introducirá la información en la interfaz para que pueda ser visualizada por el usuario. Por tanto, el código de ejecución de este hilo es el que sigue:

```
   /** The instructions to be executed by the thread */
@Override
public void run() {
    while(this.start) {
        try {
            /* Send the message */
            sendMessage();
            /* Holds until the message is ready */
            waitMessageStateChange();
            /* Gets the message */
            getMessage(bluetoothThread.getLastMessage());
        } catch (IOException | InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Estos códigos son simplemente la ejecución de los algoritmos implementados. Para visualizar la gestión del envío y la recepción de un mensaje se puede consultar el apéndice D.

6.3. Conversión de la información recibida

Tal y como se ha comentado anteriormente, una vez se ha recibido el mensaje con la información solicitada, esta se debe de convertir a un valor decimal o un valor interpretable por el usuario. Para ello existen diversos métodos o funciones en distintas clases que cumplen con lo requerido. En la clase *OBDpids* encontramos el método siguiente:

```
   /** Method that receives de last sent PID and the hexadecimal data that the device has returned,
    converts it to a decimal value and applies the corresponding equation
 * @param pid String: The last sent pid
 * @param byteA String: The hexadecimal data received from the device (byte A)
 * @param byteB String: The hexadecimal data received from the device (byte B)
 * @return String */
public String convertHexToDec(String pid, String byteA, String byteB){
    /* Deletes any white space present on the data */
    String localByteA = byteA.trim(), localByteB = byteB.trim();
    /* Auxiliary variables to store the decimal value */
    int decDataA = 0, decDataB = 0;
    /* Converts the hexadecimal data to decimal data */
    if(!localByteA.equals("")){
        decDataA = Integer.parseInt(localByteA, 16);
    }
    else{
        return "";
    }
    if(!localByteB.equals("")){
        decDataB = Integer.parseInt(localByteB, 16);
    }
    /* Search the corresponding equation to apply */
    switch (pid){
        case "01 04":
        case "01 2F":
            return Math.round((decDataA / 2.55)) + "";
        case "01 05":
        case "01 0F":
            return (decDataA - 40) + "";
        case "01 06":
        case "01 07":
        case "01 08":
```

```
        case "01 09":
            return ((decDataA / 1.28) - 100) + "";
        case "01 0A":
            return 3 * decDataA + "";
        case "01 0D":
        case "01 0B":
            return decDataA + "";
        case "01 0C":
            return  (((256 * decDataA) + decDataB) / 4) + "";
        case "01 0E":
            return ((decDataA / 2) - 64) + "";
        case "01 10":
            return (((256 * decDataA) + decDataB) / 100) + "";
        case "01 11":
        case "01 2C":
        case "01 2E":
            return ((100 / 255) * decDataA) + "";
        case "01 1F":
        case "01 21":
            return ((256 * decDataA) + decDataB) + "";
        case "01 22":
            return (0.079 * ((256 * decDataA) + decDataB)) + "";
        case "01 23":
            return (10 * ((256 * decDataA) + decDataB)) + "";
        case "01 24":
        case "01 25":
        case "01 26":
        case "01 27":
        case "01 28":
        case "01 29":
        case "01 2A":
        case "01 2B":
            return ((2 / 65536) * ((256 * decDataA) + decDataB)) + "";
        case "01 2D":
            return (((100 / 128) * decDataA) - 100) + "";

        default:
            return "";
    }
}
```

Como se puede observar en el código, el método recibe tres parámetros que son el PID que se ha enviado y los bytes A y B de información (dado que la aplicación solo solicita información que está formada por 2 bytes como máximo, no es necesario tener en cuenta el resto de los bytes que se podrían recibir). De estos parámetros se elimina cualquier espacio en blanco que pueda existir y, seguidamente, se realiza la conversión a hexadecimal, no sin antes comprobar que la información se ha recibido de manera correcta. Mediante el método *parseInt* definido por defecto en la macro de *String*, se transforma la información recibida en el formato que se desee simplemente pasando como parámetros la información y la base a la que se desea realizar la conversión.

Una vez se ha convertido la información a decimal, se debe aplicar la fórmula correspondiente dependiendo del PID que se trate y, de este modo, ya se obtendrán los datos normalizados.

El otro método con el que realizar la normalización de los valores solicitados se encuentra en la clase *OBDtroubleCodes*, con el que se interpretan los datos recibidos a la solicitud de los errores presentes en el vehículo. El método en cuestión es el que se muestra a continuación:

```
  /** Method that converts the hexadecimal trouble code into a normal code
 * @param troubleCode String: The hexadecimal trouble code
 * @return String
 */
public String convertTroubleCode(String troubleCode){
    /* Deletes any white space present on the data */
    String localTroubleCode = troubleCode.trim();
    /* Checks if the trouble code in not 00 00, which means that there is not trouble code */
    if(!localTroubleCode.equals("")) {
        /* Gets the first number of the hexadecimal code */
        String firstNumber = troubleCode.substring(0, 1);
        /* Depending on it, makes the correct conversion */
        switch (firstNumber) {
            case "0":
                return "P0" + troubleCode.substring(1, 4);
            case "1":
                return "P1" + troubleCode.substring(1, 4);
            case "2":
                return "P2" + troubleCode.substring(1, 4);
            case "3":
                return "P3" + troubleCode.substring(1, 4);
            case "4":
                return "C0" + troubleCode.substring(1, 4);
            case "5":
                return "C1" + troubleCode.substring(1, 4);
            case "6":
                return "C2" + troubleCode.substring(1, 4);
            case "7":
                return "C3" + troubleCode.substring(1, 4);
            case "8":
                return "B0" + troubleCode.substring(1, 4);
            case "9":
                return "B1" + troubleCode.substring(1, 4);
            case "A":
                return "B2" + troubleCode.substring(1, 4);
            case "B":
                return "B3" + troubleCode.substring(1, 4);
            case "C":
                return "U0" + troubleCode.substring(1, 4);
            case "D":
                return "U1" + troubleCode.substring(1, 4);
            case "E":
                return "U2" + troubleCode.substring(1, 4);
            case "F":
                return "U3" + troubleCode.substring(1, 4);
            default:
                return "";
        }
    }
    return "";
}
```

En esta ocasión, al método solo se le pasa un parámetro, que es el código de error en formato hexadecimal. Pero en esta ocasión, en lugar de transformar la información a decimal, simplemente se selecciona el primer número del código de error (hay que destacar que un código de error está formado por 4 números en hexadecimal) y, dependiendo de cual sea este, se deberá de cambiar por uno de los códigos definidos que se pueden visualizar en la Tabla 25 y luego concatenarle el resto de la información recibida.

Tabla 25: Tabla de equivalencias de hexadecimal a identificador del código de error

Valor hexadecimal	Código equivalente
0	P0
1	P1
2	P2
3	P3
4	C0
5	C1
6	C2
7	C3
8	B0
9	B1
A	B2
B	B3
C	U0
D	U1
E	U2
F	U3

6.4. Solicitud y almacenamiento de la ubicación del vehículo

Otra de las funcionalidades que presentaba la aplicación, era la de obtención de la ubicación del dispositivo móvil, de tal manera que cuando se estacionara el vehículo en cualquier lugar, se pudiera guardar la ubicación en la que se localiza. Con ello, los usuarios podrían estar tranquilos si no recuerdan en qué lugar estacionaron. La implementación de esta funcionalidad se realiza en el código que se puede visualizar a continuación:

```
  /** Method that request to the LocationManager for an update of the phone location */
private void saveLocation() {
    /* Checks if the user has granted permission to access location, if not request it*/
    try {
        if (ContextCompat.checkSelfPermission(getContext(),
                Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.PERMISSION_GRANTED) {
            /* Gets the location provider of the phone */
            locationManager = (LocationManager) getActivity()
                    .getSystemService(getContext().LOCATION_SERVICE);
            /* Checks the version of the API */
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.P) {
                /* Checks if the location service is enabled if not shows a message*/
                if(locationManager.isLocationEnabled()){
                    /* Informs the user that location is being saved */
                    this.rlHome.setVisibility(View.VISIBLE);
                    /* Request updates to the provider */
                    locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
                            0, 0, this);
                }
                else {
                    Toast.makeText(getContext(), R.string.homeFragmentActivateLocation,
                            Toast.LENGTH_LONG).show();
                }
            }
        }
        else{
```

```
            Toast.makeText(getContext(), R.string.grantPermisions,
                    Toast.LENGTH_LONG).show();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Una vez se ha pulsado el botón de guardar la ubicación del vehículo, se ejecutará este código. En él, se comprueba en primer lugar si se ha concedido permiso a la aplicación para acceder a la ubicación; en caso negativo se informa al usuario de ello. Si la respuesta es afirmativa, por medio del servicio de localización del dispositivo se solicitan actualizaciones de la ubicación, no sin antes comprobar si el servicio de localización del *smartphone* está activo. En el momento que se obtenga una, se ejecutará la siguiente parte del algoritmo.

```
  /** Method called when the location has changed
 * @param location  Location: the updated location */
@Override
public void onLocationChanged(@NonNull Location location) {
    /* Gets the latitude and longitude */
    this.latitude = location.getLatitude();
    this.longitude = location.getLongitude();
    /* Puts it on the TextView */
    tvCoordinates.setText(latitude + "," + longitude);
    /* Stops the request for new location updates */
    locationManager.removeUpdates(this);
    /* A thread access to the database and saves the location */
    new Thread(new Runnable() {
        @Override
        public void run() {
            MySQLiteOpenHelper.getInstance(getContext())
                    .addLocationToDatabase(latitude, longitude);
            getActivity().runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    /* Informs the user that location has been saved */
                    rlHome.setVisibility(View.INVISIBLE);
                    Toast.makeText(getContext(), R.string.carLocationSaved,
                            Toast.LENGTH_LONG).show();
                }
            });
        }
    }).start();
}
```

En esta segunda parte, se obtiene la latitud y la longitud de la ubicación recibida y se informa de que ya no es necesario recibir más actualizaciones de ubicación. La latitud y la longitud son representadas en la interfaz de manera que sean visibles para el usuario y, una vez lo estén, pulsando sobre el botón con el icono de ubicación se lanzará el servicio de Google Maps [20] y se representará en el mapa la ubicación. Hay que destacar también que, tal y como se muestra en el fragmento de código superior, la latitud y la longitud se guardan en la base de datos presente en la aplicación. Esta base es la denominada SQLite [21].

7. Recorrido por la aplicación CarAnalyzer

CarAnalyzer es una aplicación de diagnóstico del vehículo capaz de interpretar más de 2500 PIDs entre códigos de error y códigos OBD. Toda esta información va desde la más común hasta la que solo se suele revisar cuando se realiza una inspección técnica. La APP está dividida en cuatro pantallas principales en las cuales se proporcionan diferentes funciones. En primer lugar, nada más abrir la aplicación, aparece la pantalla que se puede visualizar en la Figura 32.

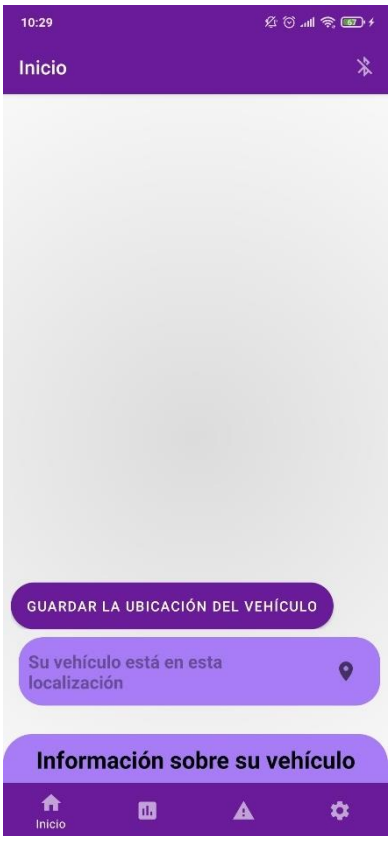


Figura 36: Pantalla de inicio de CarAnalyzer

Como se puede observar, la aplicación está diseñada para ser utilizada en dispositivos móviles en modo vertical. En la imagen se pueden percibir distintos elementos, como una serie de botones para activar el Bluetooth, guardar la ubicación del vehículo y consultar la ubicación guardada. También se puede visualizar un desplegable en el que aparece más información sobre el vehículo. Por último, se encuentra la barra inferior de navegación para alternar entre las distintas pantallas disponibles.

En esta pantalla, por tanto, se pueden realizar las funciones de guardar la ubicación del vehículo y consultarla además de visualizar más información del vehículo; pero esto último no será visible hasta que no se complete el perfil presente en la pantalla de ajustes, la cual se mostrará más adelante.

Si se pulsa sobre el segundo botón de la barra de navegación, se mostrará la pantalla correspondiente a la Figura 33 y en ella hay presentes diferentes contenidos.

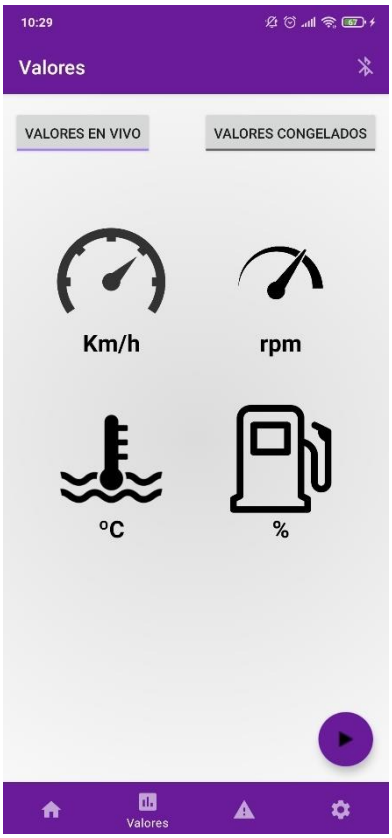


Figura 37: Pantalla de valores de CarAnalyzer

En este caso se pueden observar elementos relativos a la información del vehículo. Estos son los correspondientes a la velocidad, revoluciones por minuto del motor, temperatura del líquido refrigerante y porcentaje restante de combustible. Estos son valores que, generalmente, son mostrados en el vehículo; pero en caso de que exista un motivo por el cual es imposible visualizar estos en el transporte utilizado, en esta pantalla se pueden ver en vivo. Además, también se encuentran diversos botones. Los de la parte superior, para cambiar entre que se muestren los valores en directo mencionados o para observar información que no suele ser accesible a los usuarios generales. El otro botón disponible, el de la parte inferior derecha, tiene como funcionalidad iniciar la captura de los datos del vehículo.

Pasando a la siguiente pantalla, la correspondiente al tercer botón de la barra de navegación, se muestra lo que se puede visualizar en la Figura 34. En este caso, se muestra la funcionalidad de la aplicación de detectar los códigos de error que está presentes en el vehículo. En caso de que el vehículo no posea ningún error, se mostrará una lista vacía; por el contrario, si apareciera algún error, se mostraría una lista con los identificadores de estos y, si se pulsara sobre ellos, se abriría el navegador con una página web para conocer todos los detalles del mencionado error. Para buscar los errores del vehículo, simplemente habría que pulsar el botón inferior derecho de la pantalla.

Figura 39: Pantalla de códigos de error de CarAnalyzer

Por último, se encuentra la pantalla de ajustes de la aplicación (ver Figura 35). En ella se podrá completar el perfil del vehículo que se utiliza para tener información de interés para el conductor reunida en un mismo lugar y de fácil acceso. Además, completar el perfil realizará una pequeña personalización de la APP que será visible en la pantalla de inicio.

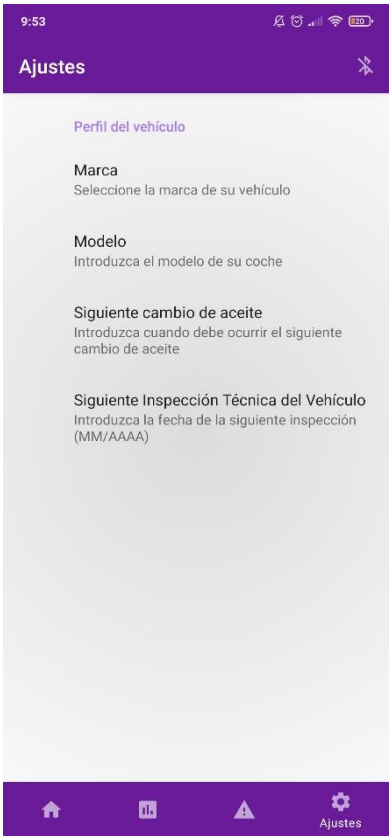


Figura 38: Pantalla de ajustes de CarAnalyzer

Hay que destacar también que la aplicación presenta traducción a 5 idiomas (inglés, español, francés, portugués y catalán) e interfaz en modo oscuro, tal y como se puede visualizar en las imágenes siguientes:

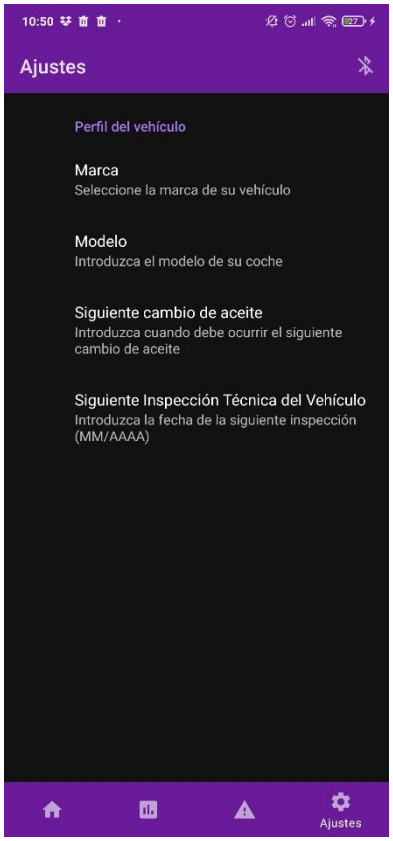


Figura 42: Modo oscuro pantalla de ajustes

Figura 40: Modo oscuro pantalla códigos de error

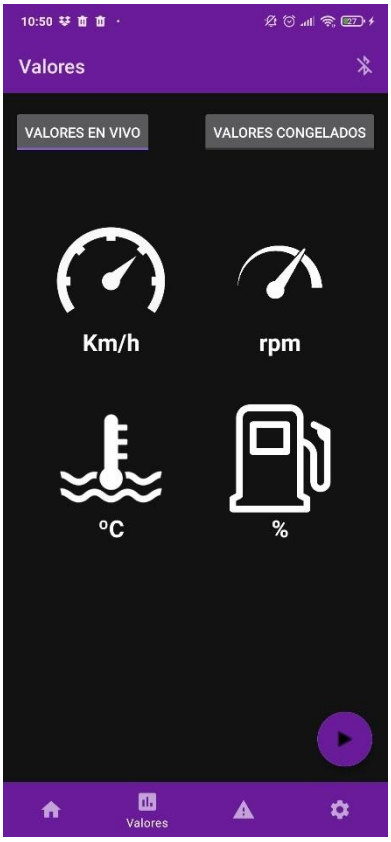


Figura 41: Modo oscuro pantalla de valores

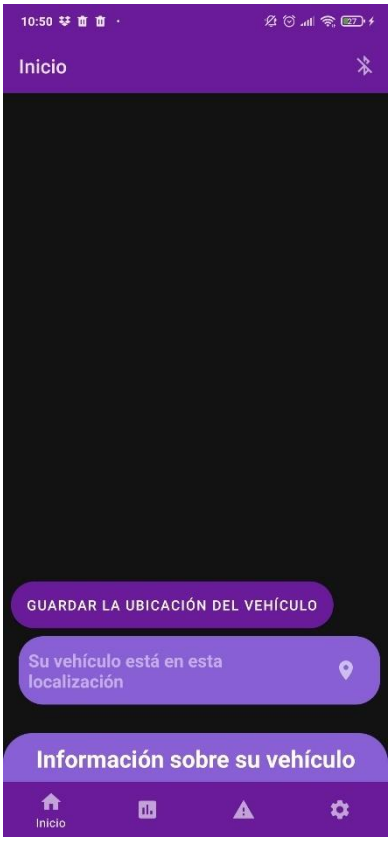


Figura 43: Modo oscuro pantalla de inicio

De este modo, la aplicación presenta una serie de funcionalidades que la hacen competir con las otras aplicaciones existentes en el mercado; pero el punto fuerte de la aplicación desarrollada es la interfaz de usuario, que presenta un estilo actual y renovado, cosa que la competencia no posee. Por último, hay que comentar que esto ha sido un recorrido por las pantallas principales, en caso de que se desee observar la funcionalidad completa de la APP, se puede consultar el apéndice C.

8. Conclusiones

El objetivo de este Trabajo de Fin de Grado ha sido el desarrollo de una aplicación para dispositivos móviles para el diagnóstico y la visualización de la información que se encuentra en el vehículo. Los objetivos que se han cumplido en este desarrollo son los siguientes:

- Elaboración de una interfaz sencilla y visualmente atractiva
- Proporcionar un lugar de almacenamiento común para varios datos de interés sobre el vehículo
- Almacenar de la ubicación en la que se ha estacionado del vehículo
- Mostrar información del vehículo normalmente oculta al usuario general

Para poder conseguir todos estos objetivos, se realizó un estudio intensivo de la competencia para lograr determinar en qué aspectos la aplicación desarrollada podría ser diferencial del resto. También se realizó un profundo análisis del funcionamiento del intérprete ELM327 para conocer tanto el formato de los mensajes que se reciben como la diferente información que se podía solicitar.

En mi opinión, el proyecto en sí ha supuesto un gran reto, pues se ha debido indagar en tecnologías que no se han visto durante toda la carrera, como el estándar OBDII de los vehículos. Además, aunque se poseía una base bastante buena de diseño de aplicaciones móviles, había ciertos aspectos que eran completamente desconocidos, como la realización de una conexión mediante Bluetooth; por lo que también se tuvo que profundizar para adquirir los conocimientos sobre este aspecto.

8.1. Trabajo futuro

Si bien la aplicación es completamente funcional en todos sus aspectos y posee determinados puntos fuertes que la hacen destacar de la competencia, en un futuro se podrían realizar ciertas mejoras las cuales no harían más que mejorar la aplicación.

Una de las mejoras más claras que se puede realizar, es la inclusión de la capacidad de interpretación de un mayor número de PIDs. Si bien la aplicación dispone de un amplio catálogo, existen otras APPs en el mercado que son capaces de interpretar el doble o triple de códigos que la que se ha desarrollado. Por lo tanto, este sería uno de los aspectos a mejorar.

Otra posible mejora sería la elaboración de una interfaz de usuario más compleja. Es cierto que el objetivo era realizar una interfaz sencilla y clara, que no supusiera un obstáculo para los usuarios noveles en este tipo de APPs; pero también hay que destacar que otras aplicaciones disponen de una interfaz mucho más elaborada, aunque la curva de aprendizaje para aprender su funcionamiento sea mayor. De este modo, sería conveniente encontrar un punto medio entre la complejidad de aprendizaje y la mejora de la interfaz.

Una tercera mejora podría ser la inclusión, en la propia APP, de un diccionario en el que se explicara toda la información que se puede obtener del vehículo y su significado. Por ejemplo, para obtener más información sobre un error, se redirige al usuario a una página web externa en la que explica se explica este error. Por lo tanto, la intención sería que no fuera necesario redirigirse a esta web, sino que la propia APP dispone del significado del error y su explicación.

Para finalizar, otras posibles mejoras que se podrían realizar serían la traducción a más idiomas y la personalización, al gusto del usuario, de la aplicación. Estas, aunque menos importantes que el resto, pueden ser de agradecer para todas aquellas personas que utilizaran la APP.

Apéndice A. Comandos AT

A continuación, se mostrará una lista con los comandos AT disponibles para configurar el dispositivo ELM327.

A.1. Comandos generales

Tabla 26: Comandos AT generales

Comandos generales	
Comando	**Descripción**
<CR>	Repite el último comando
BRD hh	Prueba la tasa del divisor de baudios hh
BRT hh	Establece el *timeout* de la tasa de baudios
D	Configuración por defecto
E0, E1	Eco off, on
FE	Olvida los eventos
I	Imprime el ID
L0, L1	Saltos de línea on, saltos de línea off
LP	Pasa a modo bajo consumo
M0, M1	Memoria off, on
RD	Lee los datos almacenados
SD hh	Almacena el byte hh
WS	Arranque en caliente
Z	Reinicia el dispositivo
@1	Muestra la descripción del dispositivo
@2	Muestra el identificador del dispositivo
@3	Almacena el identificador del comando @2

A.2. Comandos OBD

Tabla 27: Comandos AT de OBD

Comandos OBD	
Comando	**Descripción**
AL	Habilita mensajes largos
AMC	Mostrar el recuento del monitor de actividad
AMT hh	Establece el *timeout* del monitor de actividad a hh
AR	Recepción automática
AT0, 1, 2	Sincronización adaptativa off, auto1, auto2
BD	Volcado de *buffer*
BI	Omite la secuencia de inicialización
DP	Muestra el protocolo actual
DPN	Muestra el protocolo por su número
H0, H1	Cabeceras off, on
MA	Monitorizar todo
MR hh	Monitoreo para el receptor hh
MT hh	Monitoreo para el transmisor hh
NL	Mensajes de tamaño normal
PC	Cierra el protocolo
R0, R1	Respuestas off, on
RA hh	Establece la dirección de recepción a hh
S0, S1	Espacios al imprimir off, on
SH xyz	Establece las cabeceras a xyz
SH xxyyzz	Establece las cabeceras a xxyyzz
SH wwxxyyzz	Establece las cabeceras a wwxxyyzz

SP h	Establece al protocolo a h y lo guarda
SP Ah	Establece el protocolo automáticamente y lo guarda
SP 00	Borra el protocolo almacenado
SR hh	Establece la dirección de recepción a hh
S S	Usa el estándar de búsqueda J1978
ST hh	Establece el *timeout* a hh x 4 ms
TA hh	Establece la dirección de prueba a hh
TP h	Prueba el protocolo h
TP Ah	Prueba el protocolo h con búsqueda automática

A.3. Comandos ISO

Tabla 28: Comandos AT de ISO

Comandos ISO	
Comando	**Descripción**
FI	Realiza un inicio rápido
IB 10	Establece la ratio de baudios ISO a 10400
IB 48	Establece la ratio de baudios ISO a 4800
IB 96	Establece la ratio de baudios ISO a 9600
IIA hh	Establece la dirección de inicio ISO a hh
KW	Imprime las palabras reservadas
KW0, KW1	Comprobación de palabras reservadas off, on
SI	Realiza un inicio lento
SW hh	Establece el intervalo de activación a hh x 20 ms
SW 00	Detiene el envío de mensajes de activación
WM	Establece el mensaje de activación

A.4. Comandos J1939 CAN

Tabla 29: Comandos AT de CAN

Comandos J1939 CAN	
Comando	**Descripción**
DM1	Monitoriza mensajes DM1
JE	Utiliza el formato de datos ELM J1939
JHF0, JHF1	Formateo de cabeceras off, on
JS	Utiliza el formato de datos SAE J1939
JMT1	Establece el multiplicador del temporizador a 1
JMT5	Establece el multiplicador del temporizador a 5
MP hhhh	Monitorización para PGN hhhh
MP hhhh n	Monitorización para PGN hhhh y obtiene n mensajes
MP hhhhhh	Monitorización para PGN hhhhhh
MP hhhhhh n	Monitorización para PGN hhhhhh y obtiene n mensajes

Apéndice B. PIDS OBDII

En el siguiente apéndice se muestran la totalidad de PIDS disponibles en los distintos modos del el estándar OBDII.

B.1. Modo 01

Tabla 30: PIDS OBD modo 1

Modo 01			
PID	**Bytes respuesta**	**Descripción**	**Fórmula**
00	4	PIDs implementados del 01 al 20	Se indica con el bit si el PID está implementado (1 si, 0 no)
01	4	Estado monitores diagnóstico desde que se borraron los fallos DTC	-
02	2	Almacena códigos de fallo DTC	-
03	2	Estado del sistema de combustible	-
04	1	Carga calculada del motor	A / 2.55
05	1	Temperatura del líquido de enfriamiento	A - 40
06	1	Ajuste combustible corto plazo (banco 1)	(A / 1.28) - 100
07	1	Ajuste combustible largo plazo (banco 1)	
08	1	Ajuste combustible corto plazo (banco 2)	
09	1	Ajuste combustible largo plazo (banco 2)	
0A	1	Presión del combustible	3A
0B	1	Presión absoluta del colector de admisión	A
0C	2	RPM del motor	(256A + B) / 4
0D	1	Velocidad del vehículo	A
0E	1	Avance del tiempo	(A / 2) - 64
0F	1	Temperatura colector de admisión	A – 40
10	2	Velocidad flujo del aire MAF	(256A + B) / 100
11	1	Posición del acelerador	A / 2.55
12	1	Estado del aire secundario controlado	-
13	1	Presencia sensores oxígeno	A0 – A3 Banco 1 A4 – A7 Banco 2

14	2	Sensor oxígeno 1 A (Voltaje) B(Ajuste combustible)	A / 200 (B / 1.28) - 100
15	2	Sensor oxígeno 2 A (Voltaje) B(Ajuste combustible)	
16	2	Sensor oxígeno 3 A (Voltaje) B(Ajuste combustible)	
17	2	Sensor oxígeno 4 A (Voltaje) B(Ajuste combustible)	
18	2	Sensor oxígeno 5 A (Voltaje) B(Ajuste combustible)	
19	2	Sensor oxígeno 6 A (Voltaje) B(Ajuste combustible)	
1A	2	Sensor oxígeno 7 A (Voltaje) B(Ajuste combustible)	
1B	2	Sensor oxígeno 8 A (Voltaje) B(Ajuste combustible)	
1C	1	Estándar OBD implementado	-
1D	1	Sensores oxigeno presentes en el banco 4	
1E	1	Estado entradas auxiliares	Si A0 = 1 activo Se A0 = 0 PTO
1F	1	Tiempo en marcha del motor	256A + B
20	1	PIDs implementados del 01 al 20	Se indica con el bit si el PID está implementado (1 si, 0 no)
21	2	Distancia recorrida con luz de fallo encendida	256A + B
22	2	Presión del tren de combustible (colector de vacío)	0.079(256A + B)
23	2	Presión medidor tren de combustible	10(256A + B)
24	4	Sensor de oxígeno 1 AB (Relación combustible-aire) CD (Voltaje)	(256A + B) / 32768 (256C + D) / 8192
25	4	Sensor de oxígeno 2 AB (Relación combustible-aire) CD (Voltaje)	
26	4	Sensor de oxígeno 3 AB (Relación combustible-aire) CD (Voltaje)	
27	4	Sensor de oxígeno 4 AB (Relación combustible-aire) CD (Voltaje)	
28	4	Sensor de oxígeno 5 AB (Relación combustible-aire) CD (Voltaje)	
29	4	Sensor de oxígeno 6 AB (Relación combustible-aire) CD (Voltaje)	

2A	4	Sensor de oxígeno 7 AB (Relación combustible-aire) CD (Voltaje)	
2B	4	Sensor de oxígeno 8 AB (Relación combustible-aire) CD (Voltaje)	
2C	1	EGR comandado	A / 2.55
2D	1	Fallo EGR	(A / 1.28) - 100
2E	1	Purga evaporativa comandada	A / 2.55
2F	1	Nivel de combustible	A / 2.55
30	1	Cantidad de calentamientos desde que se borraron los fallos	A
31	2	Distancia recorrida desde que se borraron los fallos	256A + B
32	2	Presión vapor del sistema evaporativo	((256A + B) / 4) - 8192
33	1	Presión barométrica absoluta	A
34	4	Sensor de oxígeno 1 AB (Relación combustible-aire) CD (Actual)	(256A + B) / 32768 (C + D / 256) - 128
35	4	Sensor de oxígeno 2 AB (Relación combustible-aire) CD (Actual)	
36	4	Sensor de oxígeno 3 AB (Relación combustible-aire) CD (Actual)	
37	4	Sensor de oxígeno 4 AB (Relación combustible-aire) CD (Actual)	
38	4	Sensor de oxígeno 5 AB (Relación combustible-aire) CD (Actual)	
39	4	Sensor de oxígeno 6 AB (Relación combustible-aire) CD (Actual)	
3A	4	Sensor de oxígeno 7 AB (Relación combustible-aire) CD (Actual)	
3B	4	Sensor de oxígeno 8 AB (Relación combustible-aire) CD (Actual)	
3C	2	Temperatura del catalizador (Banco 1, sensor 1)	(256A + B) / 10 - 40

3D	2	Temperatura del catalizador (Banco 1, sensor 2)	
3E	2	Temperatura del catalizador (Banco 2, sensor 1)	
3F	2	Temperatura del catalizador (Banco 2, sensor 2)	
40	4	PIDs implementados del 41 al 60	Se indica con el bit si el PID está implementado (1 si, 0 no)
41	4	Estado de los monitores	-
42	2	Voltaje del módulo de control	(256A + B) / 1000
43	2	Valor absoluto de carga	(256A + B) / 2.55
44	2	Relación combustible-aire	(256A + B) / 32768
45	1	Posición relativa del acelerador	A / 2.55
46	1	Temperatura del aire ambiental	A - 40
47	1	Posición absoluta del acelerador B	A / 2.55
48	1	Posición absoluta del acelerador C	
49	1	Posición absoluta del acelerador D	
4A	1	Posición absoluta del acelerador E	
4B	1	Posición absoluta del acelerador F	
4C	1	Actuador comandado del acelerador	
4D	2	Tiempo con luz de fallo encendida	256A + B
4E	2	Tiempo desde que se borraron todos los fallos	-
4F	4	Valor relación combustible-aire, voltaje sensor de oxígeno, corriente del sensor y presión del colector	A B C 10D
50	4	Velocidad flujo de aire del sensor de flujo de aire masivo	10A B, C, D reservados
51	1	Tipo de combustible	-
52	1	Porcentaje de combustible etanol	(100 / 255)A
53	2	Presión absoluta del vapor del sistema de evaporación	(256A + B) / 200
54	2	Presión del vapor del sistema de evaporación	(256A + B) - 32767
55	2	Ajuste del sensor de oxígeno secundario de plazo corto (A banco 1, B banco 3)	(100 / 128)A – 100

56	2	Ajuste del sensor de oxígeno secundario de plazo largo (A banco 1, B banco 3)	(100 / 128)B – 100
57	2	Ajuste del sensor de oxígeno secundario de plazo corto (A banco 2, B banco 4)	
58	2	Ajuste del sensor de oxígeno secundario de plazo largo (A banco 2, B banco 4)	
59	2	Presión absoluta del tren de combustible	10(256A + B)
5A	1	Posición relativa del acelerador	(100 / 255)A
5B	1	Tiempo de vida del banco de baterías híbridas	(100 / 255)A
5C	A	Temperatura del aceite del motor	A – 40
5D	2	Sincronización de la inyección de combustible	((256A + B) / 128) - 210
5E	2	Velocidad del combustible del motor	(256A + B) / 20
5F	1	Requisitos de emisiones del vehículo	-
60	4	PIDs implementados del 61 al 80	-
61	1	Porcentaje de torque solicitado por el conductor	A - 125
62	1	Porcentaje actual de torque del motor	A -125
63	2	Torque de referencia del motor	256A + B
64	5	Datos del porcentaje de torque del motor	A – 125 B – 125 C – 125 D – 125 E – 125
65	2	Entrada o salida auxiliar implementada	-
66	5	Sensor de flujo de aire masivo	-
67	3	Temperatura del enfriador del motor	-
68	5	Sensor de temperatura de aire de entrada	-
69	5	EGR comandado y fallo de EGR	-
6A	5	Control comandado flujo de aire de entrada de Diesel y posición relativa	-
6B	5	Temperatura de recirculación del gas del escape	-
6C	5	Control comandado del actuador del acelerador y posición relativa	-
6D	5	Sistema de control de presión del combustible	-
6E	5	Sistema de control de presión de inyección	-
6F	3	Presión de entrada del compresor del turbocargador	-
70	9	Control de presión de aumento	-
71	5	Control del turbo de geometría variable	-
72	5	Control de la compuerta de desperdicio	-
73	5	Presión del escape	-
74	5	RPM del turbocargador	-
75	7	Temperatura del turbocargador	-
76	7	Temperatura del turbocargador	-
77	5	Temperatura del enfriador del aire de carga	-
78	9	Temperatura del gas de escape (Banco 1)	-
79	9	Temperatura del gas de escape (Banco 2)	-
7A	7	Filtro de partículas diésel	-
7B	7	Filtro de partículas diésel	-
7C	9	Temperatura del filtro de partículas diésel	((256A + B) / 10) - 40
7D	1	Estado del área de control NOx NTE	-
7E	1	Estado del área de control NOx NTE	-
7F	13	Tiempo que el motor ha estado en marcha	-

80	4	PIDs implementados del 81 al A0	-
81	21	Tiempo de marcha del motor para el dispositivo auxiliar de control de emisiones	-
82	21	Tiempo de marcha del motor para el dispositivo auxiliar de control de emisiones	-
83	5	Sensor NOx	-
84	1	Temperatura del colector	-
85	10	Sistema de NOx	-
86	5	Sensor del material de partículas	-
87	5	Presión absoluta del colector de admisión	-
88	13	Sistema de inducción SCR	-
89	41	Tiempo en marcha del dispositivo auxiliar de control de emisiones	-
8A	41	Tiempo en marcha del dispositivo auxiliar de control de emisiones	-
8B	7	Postratamiento del diésel	-
8C	16	Sensor de oxígeno	-
8D	1	Posición del acelerador G	-
8E	1	Fricción del motor, porcentaje de torque	A – 125
8F	5	Sensor PM (Banco 1 y 2)	-
90	3	Información del sistema OBD del vehículo	-
91	5	Información del sistema OBD del vehículo	-
92	2	Control del sistema de combustible	-
93	3	Contadores del sistema OBD del vehículo	-
94	12	Sistema de advertencia e inducción NOx	-
98	9	Sensor de temperatura del escape	-
99	9	Sensor de temperatura del escape	-
9A	6	Datos del sistema, batería y voltaje del vehículo híbrido	-
9B	4	Datos del sensor de fluido de escape diésel	-
9C	17	Datos del sensor de oxígeno	-
9D	4	Tasa de combustible del motor	-
9E	2	Tasa de flujo del escape del motor	-
9F	9	Porcentaje de uso del sistema de combustible	-
A0	4	PIDs implementados del A1 al C0	-
A1	9	Datos corregidos del sensor NOx	-
A2	2	Tasa de combustible del cilindro	-
A3	9	Presión del sistema de evaporación	-
A4	4	Marcha actual de la transmisión	-
A5	4	Dosificación de líquido del escape diésel	-
A6	4	Odómetro	((2^24)A + (2^16)B + (2^8)C + D) / 10
C0	4	PIDs implementados del C1 al E0	-
C3	-	-	-
C4	-	-	-

B.2. Modo 02

En este modo se utilizan los mismos PIDs que en el modo 1.

B.3. Modo 03

Tabla 31: PIDS OBD modo 3

Modo 03			
PID	**Bytes respuesta**	**Descripción**	**Fórmula**
-	6N	Solicita los códigos de fallo	3 códigos por mensaje

B.4. Modo 04

Tabla 32: PIDS OBD modo 4

Modo 04			
PID	**Bytes respuesta**	**Descripción**	**Fórmula**
-	0	Borra los códigos de fallo y apaga la luz de mal funcionamiento	-

Apéndice C. Manual de usuario de CarAnalyzer

A continuación, se mostrará las funcionalidades completas de la aplicación y su funcionamiento.

C.1. Primeros pasos en la aplicación

Recién instalada la aplicación, si se inicia aparecerán una serie de diálogos. El primero de ellos es para conceder permiso a la aplicación para acceder a la ubicación del dispositivo. El segundo de los diálogos da la bienvenida a los usuarios y realiza un pequeño resumen de cómo utilizar la aplicación. Este diálogo dispone de una caja que se puede marcar para que, en caso de que se desee, no vuelva a aparecer cada vez que se inicie de nuevo la APP.

Figura 44: Diálogo inicial

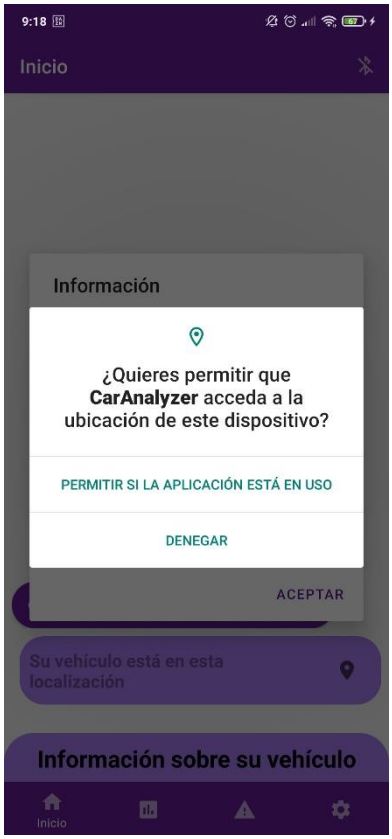


Figura 45: Solicitud de acceso a la ubicación

C.2. Creación del perfil del vehículo

Una vez leído el diálogo inicial, tal y como se recomienda se deberá ir a la pantalla de ajustes que se muestra en la Figura 17 y crear el perfil del vehículo. Para ello se irán rellenando los diferentes campos que se presentan en la pantalla con los valores que correspondan al vehículo en cuestión.

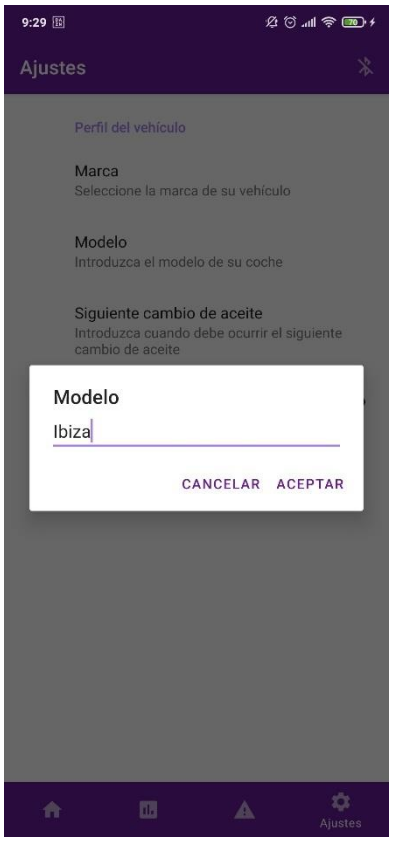


Figura 46: Campo modelo

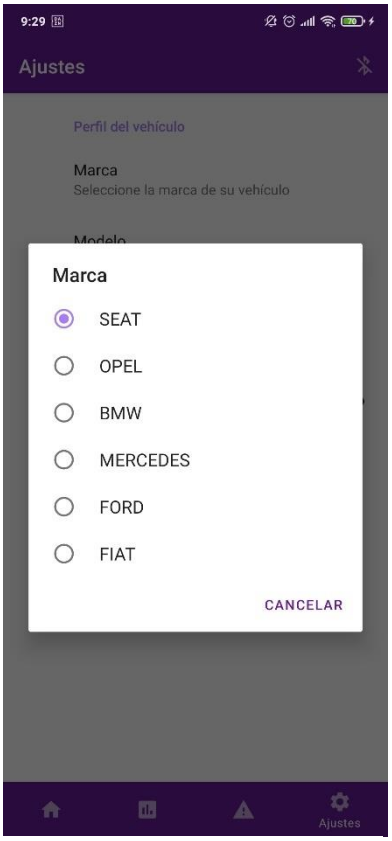


Figura 48: Campo marca

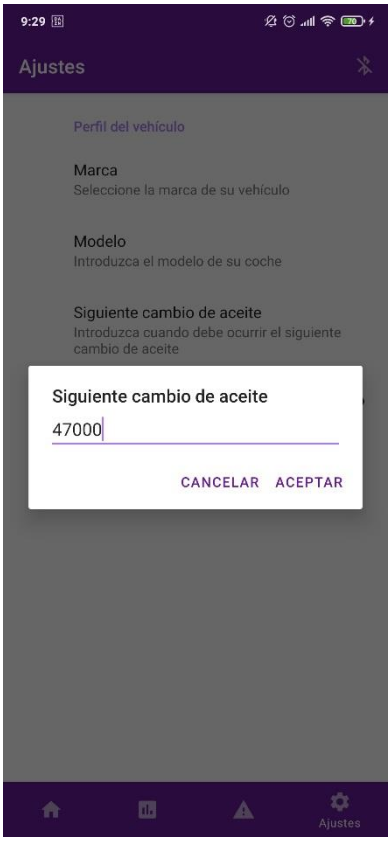


Figura 49: Campo cambio de aceite

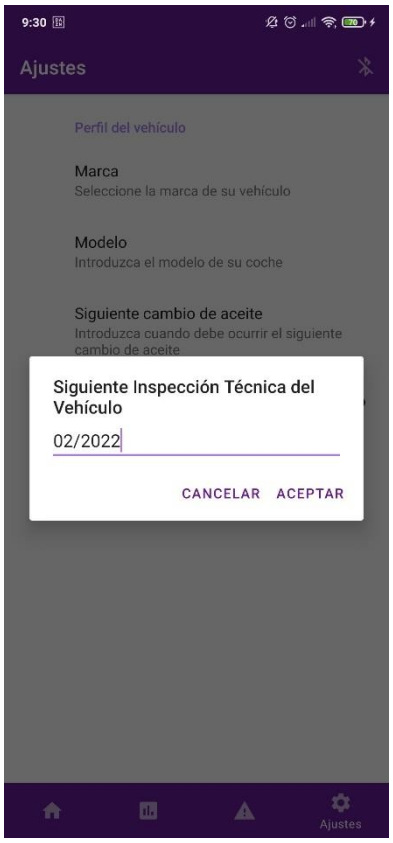


Figura 47: Campo inspección técnica

Cuando se ha completado el perfil, los cambios podrán ser visibles en la pantalla de inicio.

Figura 50: Cambios en la pantalla de inicio

C.3. Conexión con dispositivo OBD

Realizado el perfil de vehículo, ya se puede pasar a la conexión de la aplicación con el dispositivo OBD. Para ello se debe pulsar sobre el icono superior derecho de la APP y, si el Bluetooth no está activado, aparecerá un mensaje para permitir o denegar la activación del este. En caso contrario, si el Bluetooth si está activado, se mostrará un diálogo para seleccionar a que dispositivo conectarse de los que el teléfono móvil tiene vinculados (si no se muestra el dispositivo deseado, se deberá ir a ajustes del teléfono y vincular el OBD).

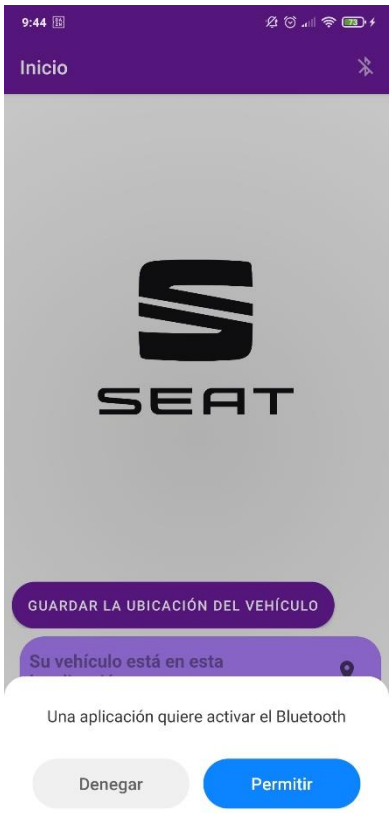


Figura 51: Solicitud de activación del Bluetooth

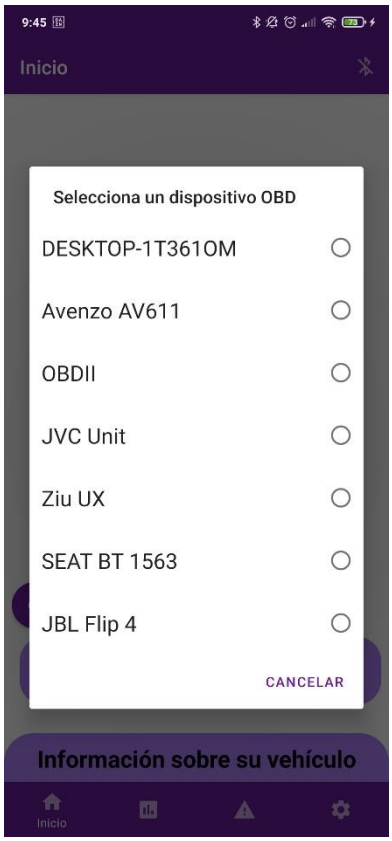


Figura 52: Diálogo de selección del dispositivo

Cuando se seleccione el dispositivo OBD, si todo ha ido correctamente, se mostrará un mensaje flotante para indicar que la conexión se ha completado de forma satisfactoria.

C.4. Visualización de los valores del vehículo

En este punto se está preparado para utilizar las funcionalidades de la aplicación. Si se navega a la pantalla de valores, se podrán visualizar los valores en directo del vehículo. Para ello hay que, en primer lugar, encender el motor del vehículo y, en segundo lugar, pulsar sobre el botón de la parte inferior derecha para comenzar el análisis. Los valores irán apareciendo en la pantalla y se actualizarán cada cierto tiempo.

En caso de que se quiera ver otra información, se pulsará sobre el botón de “valores congelados” y se visualizarán otros valores. Hay que destacar que, en algunos casos, no es necesario tener encendido el motor del vehículo para la visualización de la información, sino que con tener el contacto activo basta. En este caso aparecerá una lista de valores con toda la información que se puede obtener del vehículo con la aplicación.

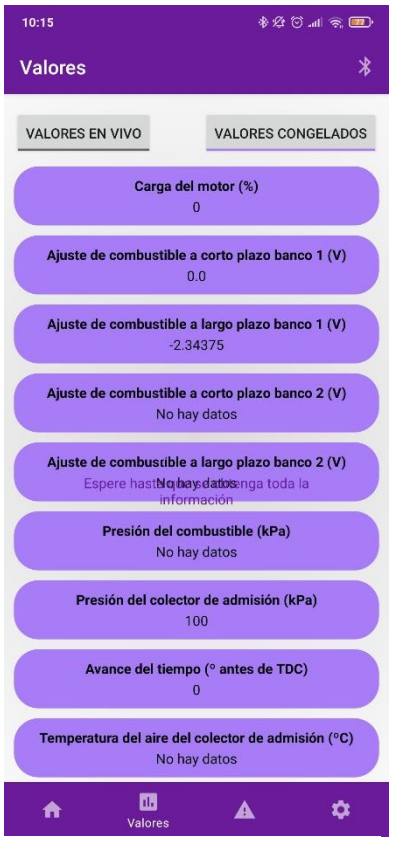


Figura 53: Valores congelados

Figura 54: Valores en vivo

C.5. Consulta de códigos de error

Si se desean visualizar los errores que se presentan en el vehículo, se deberá navegar a la pantalla de códigos de error. En ella, si se pulsa sobre el botón inferior derecho, comenzará la búsqueda de errores y, según las condiciones del vehículo que se está utilizando, la aplicación informará con mensajes flotantes lo que se realiza. Por ejemplo, si el vehículo no presenta errores, se informará que no hay errores en el vehículo y se mostrará la lista vacía (Figura 16). En caso de que sí que haya, comenzará la búsqueda de errores y, si se encuentran, se mostrarán en la lista los identificadores de estos.

Además, si se pulsa sobre el identificador del error, se abrirá el navegador y se mostrará una página web con información sobre el dicho error.

Figura 55: Web informativa del error

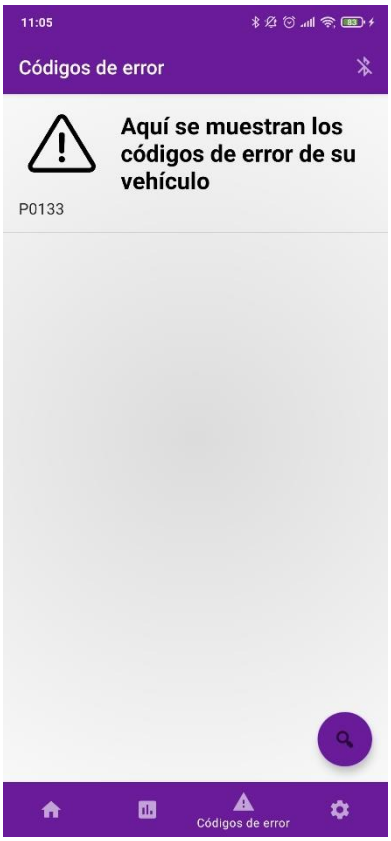


Figura 56: Errores presentes en el vehículo

C.6. Almacenamiento y visualización de la ubicación del vehículo

En caso de que se desee almacenar la ubicación en la que se ha estacionado el vehículo, en primer lugar, se deberá navegar a la pantalla de inicio y, con el servicio de ubicación del teléfono activo y los permisos de acceso a este concedidos a la aplicación, se deberá de pulsar sobre el botón de “Guardar ubicación del vehículo”, tal y como se pudo ver en la Figura 32. Luego de realizar esta acción, aparecerá una barra de progreso indicando al usuario que espere mientras se obtiene y almacena la ubicación y, una vez almacenada, se representará la latitud y la longitud en pantalla de la manera que se puede ver en la Figura 57. De este modo, si se desea ver la ubicación en el mapa simplemente hay que pulsar sobre el icono de ubicación con lo que se abrirá la aplicación de mapas y se podrá observar la localización del vehículo.

Figura 58: Ubicación almacenada representada

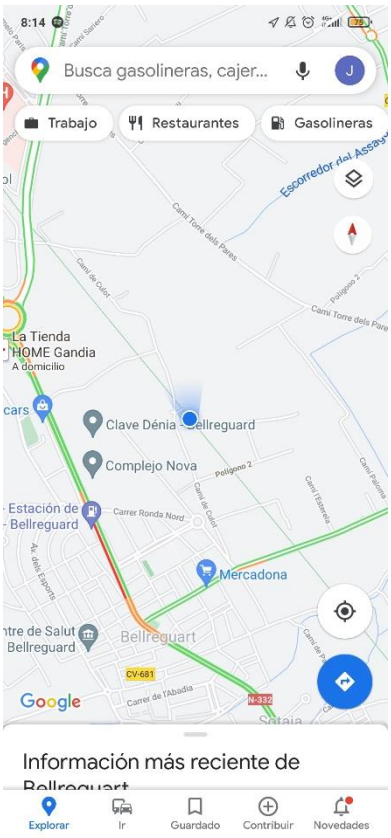


Figura 57: Representación de la ubicación en el mapa

Apéndice D. Implementación del envío y la recepción de mensajes

En este apéndice se va a mostrar cómo se ha implementado la gestión del envío y la recepción de los mensajes para la comunicación con el OBD.

D.1. Gestión de los mensajes de valores en vivo

Tal y como se ha visto en el código del apartado relacionado con la implementación de las funcionalidades, existe un hilo encargado de la gestión de los valores que aparecen en la pantalla de valores en vivo. El hilo envía un mensaje, luego consulta el estado de este cada cierto tiempo y, cuando está preparado, lo obtiene. De este modo, para el envío de mensajes se ejecuta el siguiente método:

```
  /** Method that writes the message to the target
 * @throws IOException */
public void sendMessage() throws IOException {
    /* Iterates the list of PIDs */
    if(this.iterator.hasNext()){
        /*Gets the PID and puts the iterator to the next position */
        this.pid = this.iterator.next();
        /* Writes the message */
        this.bluetoothThread.write(this.pid + "\r");
    }else{
        /* Resets the iterator */
        this.iterator = this.listPidsLV.iterator();
        /*Gets the PID and puts the iterator to the next position */
        this.pid = this.iterator.next();
        /* Writes the message */
        this.bluetoothThread.write(this.pid + "\r");
    }
}
```

Como se puede observar, el hilo comprueba si el iterador que posee la lista de PIDs tiene un siguiente elemento disponible. En caso afirmativo lo obtiene y lo escribe para su envío mediante el método *write* del hilo Bluetooth. En cambio, si no hay un siguiente elemento disponible, reinicia el iterador para volver al primer PID. De este modo, se consigue una monitorización continúa de los valores que están presentes en el vehículo.

Una vez enviado el mensaje, como se ha comentado, se espera a que su estado pase a preparado y entonces se obtiene mediante el siguiente método:

```
  /** Method that gets the message, converts it and puts to the corresponding place
 * @param res String: The message received
 * @return String
 * */
public String getMessage(String res){
    /* Indicates that the message has been consumed */
    this.bluetoothThread.setMessageState(false);
    /* Divides the message into an array */
    String[] splits = res.split(" ");
    /* String for the hexadecimal data */
    String byteA = "", byteB = "", decData = "";
    /* Checks the length of the array in order to know how many bytes has been received */
    if(splits.length == 6){
        /* Gets the hexadecimal data */
        byteA = splits[3].trim();
        byteB = splits[4].trim();
        if(!byteA.equals("") && !byteB.equals("")) {
```

```
            /* Converts the data */
            decData = this.obdPids.convertHexToDec(this.pid, byteA, byteB);
        }
    }else if(splits.length == 5){
        /* Gets the hexadecimal data */
        byteA = splits[3].trim();
        if(!byteA.equals("")) {
            /* Converts the data */
            decData = this.obdPids.convertHexToDec(this.pid, byteA, byteB);
        }
    }
    return decData;
}
```

En este caso se pueden observar diversos aspectos interesantes. En primer lugar, se indica que el mensaje ha sido consumido para realizar la correcta gestión del estado. Luego se divide el mensaje en un array utilizando como método de división el espacio en blanco; de este modo se consigue separar cada elemento en una posición distinta. Cuando ya se ha dividido, se comprueba si el formato es el correcto. Esto se realiza mediante la comprobación de la longitud del mensaje de manera que, si es de una longitud diferente a la deseada, es que ha habido algún problema en la transmisión y se debe descartar. En caso de que todo sea correcto, se obtienen los datos de interés, eliminando de estos posibles espacios en blanco que se hayan podido añadir en el elemento y se transforma la información al formato adecuado con el método correspondiente.

D.2. Gestión de los mensajes de valores congelados

Para la gestión de los valores congelados, el hilo correspondiente con esta funcionalidad también ejecuta una serie de métodos muy parecidos a los visualizados en el apartado anterior. De hecho, el método para la recepción del mensaje es exactamente el mismo que el mostrado en el apartado anterior. Por lo tanto, la única diferencia estaría en el envío del mensaje. El código de este se puede visualizar a continuación:

```
  /** Method that writes the message to the target
 * @throws IOException */
public void sendMessage() throws IOException {
    /* If there are more elements in the list, continue. If not, stop */
    if(this.iterator.hasNext()){
        /*Gets the PID and puts the iterator to the next position */
        this.pid = this.iterator.next();
        /* Writes the message */
        this.bluetoothThread.write(this.pid + "\r");
    }else{
        /* Stops the thread */
        this.reference.get().stop();
        /* Resets the iterator */
        this.iterator = this.listPidsFV.iterator();
    }
}
```

Como se puede ver, también se comprueba si el iterador sobre la lista de PIDs tiene un siguiente elemento y, si lo tiene, se envía. La diferencia es que, en el caso anterior, para monitorizar el estado de los valores había que hacer un recorrido cíclico sobre la lista, pero, en este caso, no se debe realizar, ya que solo se desea el valor en un momento determinado. De este modo, cuando no hay un siguiente PID en la lista para el envío, simplemente se indica al hilo que se debe detener y el iterador se reinicia para su próxima utilización.

D.3. Gestión de los mensajes de códigos de error

El último de los hilos, el encargado de la pantalla de códigos de error, presenta un código en el envío mucho más sencillo. De hecho, simplemente envía el PID 01 01, que cuestiona si hay códigos de error almacenados en el vehículo y dependiendo de la respuesta que se reciba, se realizarán más acciones o simplemente finalizará el hilo. Por tanto, los cambios más drásticos tienen lugar en la recepción del mensaje y el código se puede ver a continuación:

```
  /** Method that gets the message and manage it
 * @param res String: The message received
 * */
public void getMessage(String res){
    /* Indicates that the message has been consumed */
    this.bluetoothThread.setMessageState(false);
    /* Divides the message into an array */
    String[] splits = res.split(" ");
    /* Checks the PID sent */
    if(this.pid.equals("01 01")) {
        /* If length is not correct, an error occurred. Try again */
        if (splits.length != 8) {
            /* Makes a Toast in order to inform the user */
            this.reference.get().getActivity().runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    Toast.makeText(reference.get().getContext(),
                            R.string.troubleCodesFragmentTryAgain, Toast.LENGTH_LONG).show();
                }
            });
            this.start = false;
        }
        else {
            /* If the data is 00, that means that there are no trouble codes */
            if (splits[3].equals("00")) {
                /* Makes a Toast in order to inform the user */
                this.reference.get().getActivity().runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        Toast.makeText(reference.get().getContext(),
                                R.string.troubleCodesFragmentNoData, Toast.LENGTH_LONG).show();
                    }
                });
                this.start = false;
            } else {
                /* Makes a SnackBar in order to inform the user */
                this.reference.get().getActivity().runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        Snackbar.make(reference.get().getView(),
                                R.string.troubleCodesFragmentSearching, Snackbar.LENGTH_LONG)
                                .setAction(R.string.OK, null).show();
                    }
                });
                /* Changes the PID to be sent */
                this.pid = "03";
            }
        }
    }
    else {
        /* If length is less than the minimum, that means that there are not trouble codes */
        if(splits.length < 4){
            /* Makes a Toast in order to inform the user */
```

```
            this.reference.get().getActivity().runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    Toast.makeText(reference.get().getContext(),
                            R.string.troubleCodesFragmentNoData, Toast.LENGTH_LONG).show();
                }
            });
        }
        else {
            /* Gets the trouble codes and converts it*/
            if((splits[1] != null) && (splits[2] != null)){
                this.troubleCode1 = this.obDtroubleCodes.convertTroubleCode(splits[1].trim()
                        + splits[2].trim());
            }
            if((splits[3] != null) && (splits[4] != null)){
                this.troubleCode2 = this.obDtroubleCodes.convertTroubleCode(splits[3].trim()
                        + splits[4].trim());
            }
            if((splits[5] != null) && (splits[6] != null)){
                this.troubleCode3 = this.obDtroubleCodes.convertTroubleCode(splits[5].trim()
                        + splits[6].trim());
            }
            if(!this.troubleCode1.equals("")) {
                this.reference.get().getActivity().runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        if (!troubleCode1.equals("")) {
                            reference.get().updateUI(troubleCode1);
                        }
                        if (!troubleCode2.equals("")) {
                            reference.get().updateUI(troubleCode2);
                        }
                        if (!troubleCode3.equals("")) {
                            reference.get().updateUI(troubleCode3);
                        }
                    }
                });
            }
        }
        this.start = false;
    }
}
```

En este caso se puede ver un código de recepción mucho más extenso. Una vez se ha enviado el mensaje para la consulta de cuántos errores hay en el vehículo, se comprueba la información recibida. Si esta no está en formato correcto, es que ha ocurrido un error, por lo que se informa al usuario que vuelva a intentarlo. En caso de que la transmisión haya sido satisfactoria, se revisa si la información de interés es equivalente a la cadena “00”, lo cual querrá decir que en el vehículo no existe ningún error. Pero, si la cadena no es equivalente a esta, se modificará el PID a enviar por aquel que cuestiona sobre cuáles son esos errores. Este se enviará y transcurrido un tiempo, se obtendrá la respuesta. En la respuesta, los errores se insertan en grupos de 3, por lo que se pueden recibir hasta 3 errores del vehículo en una misma consulta. Cada error estará formado por dos pares de números en hexadecimal, sobre los cuales se eliminará todo espacio en blanco que se haya podido añadir en ellos y se realizará la correspondiente transformación mediante la Tabla 25. De este modo, se obtendrá el identificador del error.

Referencias

[1] *Wikipedia*, OBD. URL de la página: https://es.wikipedia.org/wiki/OBD. [Último acceso: 04 febrero 2021].

[2] *Android Phoria*, 7 aplicaciones para leer o limpiar errores de tu coche con OBD2, 2020. URL de la página: https://androidphoria.com/aplicaciones/mejores-aplicaciones-escanear-errores-coche-obd2. [Último Acceso: 04 febrero 2021].

[3] OBD Auto Doctor. URL a la página de la aplicación: https://www.obdautodoctor.com/android.php. [Último Acceso: 04 febrero 2021].

[4] Torque Lite. URL a la página de la aplicación: https://torque-bhp.com/. [Último Acceso: 04 febrero 2021].

[5] DashCommand. URL a la página de la aplicación: https://www.palmerperformance.com/. [Último Acceso: 04 febrero 2021].

[6] Car Scanner ELM OBD2. URL a la página de la aplicación: https://www.carscanner.info/. [Último Acceso: 04 febrero 2021].

[7] EODB Facile. URL a la página de la aplicación: https://www.outilsobdfacile.fr/. [Último Acceso: 04 febrero 2021].

[8] SAE. URL de la página: https://www.sae.org/. [Último acceso: 04 febrero 2021].

[9] ISO. URL de la página: https://www.iso.org/home.html. [Último acceso: 04 febrero 2021].

[10] Manual ELM327, OBD to RS232 Interpreter. URL de acceso al PDF: https://www.elmelectronics.com/wp-content/uploads/2016/07/ELM327DS.pdf [Último acceso: 06 febrero 2021].

[11] NinjaMock. URL de la página: https://ninjamock.com/. [Último acceso: 04 junio 2021]

[12] Arquitectura *Model View Presenter* (MVP). URL de la página: https://devexperto.com/mvp-android/. [Último acceso: 04 junio 2021]

[13] Arquitectura *Model View Controller* (MVC). URL de la página: https://codigofacilito.com/articulos/mvc-model-view-controller-explicado. [Último acceso: 04 junio 2021]

[14] Arquitectura *Model View ViewModel* (MVVM). URL de la página: https://inmediatum.com/blog/ingenieria/mvvm-que-es-y-como-funciona/. [Último acceso: 04 junio 2021]

[15] Android Studio. URL de la página:
https://developer.android.com/studio?hl=es. [Último acceso: 04 junio 2021]

[16] Java. URL de la página:
https://www.java.com/es/about/whatis_java.jsp. [Último acceso: 04 junio 2021]

[17] Servicio de Google Play sobre la localización. URL de la página:
https://developers.google.com/android/guides/setup. [Último acceso: 04 junio 2021]

[18] Biblioteca para los ajustes de la aplicación (*Preference*). URL de la página:
https://developer.android.com/jetpack/androidx/releases/preference.
[Último acceso: 04 junio 2021]

[19] Biblioteca *Material Design*. URL de la página:
https://material.io/develop/android. [Último acceso: 04 junio 2021]

[20] Plataforma Google Maps. URL de la página:
https://developers.google.com/maps/documentation?hl=es.
[Último acceso 04 junio 2021]

[21] SQLite. URL de la página:
https://www.sqlite.org/index.html. [Último acceso: 04 junio 2021]