Universidad Internacional de La Rioja

Escuela Superior de Ingeniería y Tecnología

Grado en Ingeniería Informática

Arquitectura gratuita para monitorización y telemetría de automóviles mediante Android y OBD2

Trabajo fin de estudio presentado por:	Filiberto Cucarella Liñana
Director/a:	Susana Quirós y Alpera
Fecha:	11/09/2024
Repositorio del código fuente:	https://gitlab.com/uni5362831/tfg

Resumen

El sistema OBD2 es esencial para diagnosticar fallos y controlar las emisiones en vehículos modernos. Su utilidad aumenta cuando se combina con telemetría en tiempo real, lo que permite una supervisión constante mientras el automóvil está en marcha. Esto mejora la eficiencia en reparaciones y mantenimiento, además de optimizar el rendimiento del vehículo.

El OBD2, de manera estándar, registra información de sistemas clave como el motor y sensores de emisión. Sin embargo, la transmisión en tiempo real de estos datos permite detectar problemas de inmediato, lo que ayuda a prevenir fallos mayores y a tomar medidas rápidamente.

La monitorización en tiempo real es crucial para evitar reparaciones costosas y cumplir con los estándares de emisiones. Fallos en el sistema de emisiones o problemas críticos, como el sobrecalentamiento del motor, pueden ser peligrosos si no se abordan de inmediato.

Este trabajo implementa una arquitectura gratuita de monitorización y telemetría para automóviles, desarrollando una aplicación Android que, conectada a un dispositivo OBD2 Bluetooth, extrae datos y los envía a una base de datos externa vía internet. Los datos se visualizan en tiempo real desde cualquier lugar del mundo por internet y se reciben alertas cuando se superan ciertos umbrales, vía Telegram.

Palabras clave: OBD2, Telemetría, Diagnóstico, Monitorización, Emisiones

Abstract

The OBD2 system is essential for diagnosing faults and monitoring emissions on modern vehicles. Its usefulness increases when combined with real-time telemetry, allowing constant monitoring while the car is running. This improves efficiency in repairs and maintenance, as well as optimizing vehicle performance.

OBD2, as standard, records information from key systems such as engine and emission sensors. However, the real-time transmission of this data allows problems to be detected immediately, helping to prevent major failures and take action quickly.

Real-time monitoring is crucial to avoid costly repairs and comply with emission standards. Emissions system failures or critical problems, such as engine overheating, can be dangerous if not addressed immediately.

This work implements a free automotive telemetry and monitoring architecture by developing an Android application that, connected to a Bluetooth OBD2 device, extracts data and sends it to an external database via the internet. The data is visualized in real time from anywhere in the world via internet and alerts are received when certain thresholds are exceeded, via Telegram.

Keywords: OBD2, Telemetry, Diagnostics, Monitoring, Emissions

Índice de contenidos

Índice de figuras

1. Introducción

Para cumplir con las nuevas normativas en materia de emisiones, durante las décadas de 1960 y 1970 los fabricantes de automóviles empezaron a incorporar sistemas de diagnóstico llamados OBD.

Sin embargo, estos sistemas iniciales eran limitados y no estaban estandarizados, lo que complicaba su uso y efectividad, ya que cada fabricante tenía su propio sistema. A medida que aumentaba la preocupación por la contaminación del aire y la eficiencia de los vehículos, la Agencia de Protección Ambiental de Estados Unidos (EPA) y la Junta de Recursos del Aire de California (CARB) comenzaron a impulsar la creación de un sistema de diagnóstico más avanzado y uniformizado dando lugar al nacimiento del OBD2 y la obligatoriedad de implementarlo en los automóviles nuevos de EEUU en el año 1996.

El OBD2 mejoró significativamente a su predecesor, al estandarizar los protocolos de comunicación y el conector de diagnóstico, permitiendo el desarrollo de herramientas de escaneo universales que podían utilizarse en cualquier vehículo compatible. Además, OBD2 amplió la capacidad de monitorización del vehículo, permitiendo una vigilancia continua de los sistemas de control de emisiones y la capacidad de registrar códigos de diagnóstico de fallos (DTC). Estos códigos son esenciales para que los técnicos identifiquen problemas específicos.

Desde su introducción, el OBD2 ha desempeñado un papel crucial en la reducción de emisiones y en la mejora de la eficiencia y seguridad de los vehículos. Gracias a su capacidad para proporcionar datos detallados y en tiempo real sobre el rendimiento del vehículo, el OBD2 ha permitido a los propietarios y técnicos detectar y solucionar problemas de manera más eficaz, contribuyendo a un entorno más limpio y a vehículos más confiables y seguros.

1.1. Motivación

La principal motivación de este trabajo es juntar los conocimientos de informática adquiridos por el autor con una de sus mayores aficiones, como es la mecánica del automóvil.

El objetivo de este trabajo es sembrar una semilla para conseguir una arquitectura de monitorización y telemetría de vehículos que sea totalmente gratuita para poder visualizar datos en tiempo real, además de poder consultar datos históricos del vehículo en determinado instante de tiempo.

1.2. Planteamiento del trabajo

La principal aportación de este trabajo es conseguir realizar una arquitectura gratuita que cualquier persona, con un mínimo de conocimientos técnicos, pueda implementar en su domicilio y con coste casi cero. Para ello se hace uso de un dispositivo Bluetooth OBD2 conectado al vehículo, un programa Android desarrollado para este trabajo corriendo en un teléfono móvil, una base de datos para almacenar los valores de las variables y un software de gráficas para visualizarlos.

1.3. Estructura del trabajo

En el Capítulo 2 se presenta el análisis de contexto, se realiza una introducción a los sistemas de diagnóstico en los automóviles, desde sus inicios hasta la actualidad, una breve descripción del sistema Android, pasando por su historia, arquitectura y sus componentes principales, así como también una descripción de cómo es el ciclo de vida de sus aplicaciones. También se realiza una breve descripción del protocolo OBD2, así como también una pequeña introducción al Bus Can, al dispositivo ELM327, a las bases de datos de series temporales y al software de visualización de métricas.

En el Capítulo 3 se presenta el estado del arte, se ha realizado una búsqueda de productos similares en el mercado actual que realicen las funciones buscadas en este trabajo o al menos similares a modo de comparativa.

En el Capítulo 4 se presenta la metodología y la arquitectura implementada. Podemos destacar el desarrollo de la aplicación Android, así como también la integración de ésta con la base de datos de series temporales y el software de visualización.

En el Capítulo 5 se presentan las conclusiones habiéndose alcanzado todos los objetivos iniciales del presente trabajo, así como también se presentan las futuras líneas de investigación.

2. Contexto

De todos es sabido que hay averías en los automóviles muy complejas, porque no es fácil llegar a la causa real del problema, para ello se deben analizar todos los datos y llegar a una conclusión.

Por ello se hace necesario disponer de una herramienta que nos facilite los datos obtenidos del vehículo, esta acción se realiza en los talleres mediante herramientas profesionales que se conectan al puerto de comunicaciones OBD2 y nos proporcionan los códigos de error guardados, así como también nos proporcionan mediciones en tiempo real de distintas variables.

En este punto, disponer de datos en tiempo real del vehículo, así como también datos históricos que nos pueden ayudar a identificar el comportamiento de nuestro vehículo y poder identificar un problema antes de que éste sea más grave.

Actualmente no se encuentra ningún producto gratuito en el mercado que dé la funcionalidad de este trabajo, es decir, el poder monitorizar tu vehículo con coste casi cero tanto en tiempo real como en tiempo pasado.

Las soluciones actuales del mercado analizadas en este trabajo dan los valores en tiempo real, pero no proporcionan ningún modo de guardar esos valores para poder analizarlos en un futuro, o son soluciones hardware de registro de datos con un coste elevado.

2.1. Android

2.1.1. Historia de Android

Un poco de historia:

> Android es un sistema operativo que se basa en el núcleo de Linux. Fue diseñado principalmente para dispositivos móviles con pantalla táctil, como smartphones y tablets, aunque también se utiliza en relojes inteligentes, televisores y automóviles. Originalmente, Android fue desarrollado por la empresa Android Inc., la cual recibió el apoyo de Google y fue adquirida por esta última en 2005. En 2007, Android fue presentado al público junto con la creación de la Open Handset Alliance, un consorcio de empresas de hardware, software y telecomunicaciones, con el objetivo de promover estándares abiertos en dispositivos móviles. El primer teléfono en utilizar el sistema operativo Android fue el HTC Dream, lanzado al mercado en octubre de 2008 (Marín, 2016, p11).

2.1.2. Arquitectura de Android

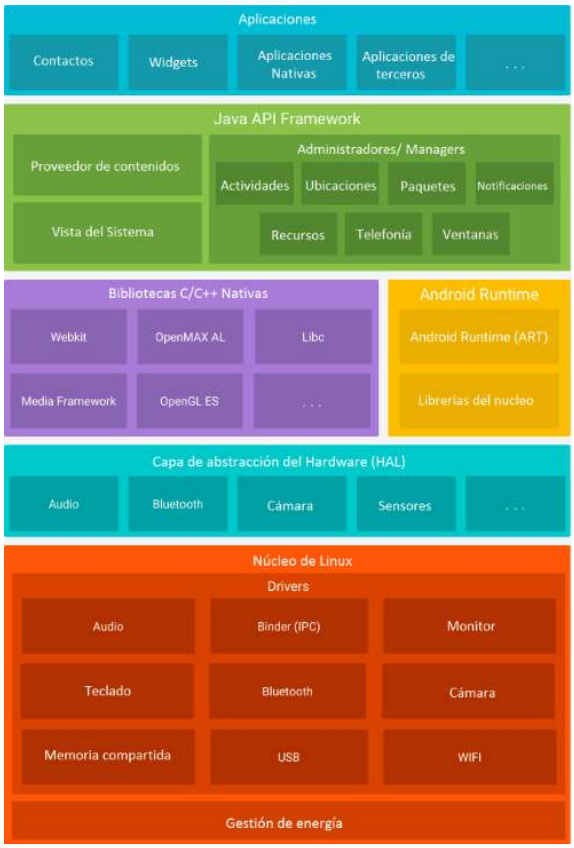


Figura 1: Arquitectura de Android

Fuente: https://developer.android.com/guide/platform/index.html

La arquitectura de Android está diseñada en capas, lo que facilita la modularidad y la flexibilidad. A continuación, se describe cada una de estas capas:

2.1.2.1. Linux Kernel

Es el responsable de interactuar con el hardware del dispositivo. El kernel gestiona tareas fundamentales como la gestión de memoria, la administración de procesos, la seguridad del sistema y los controladores de hardware. Android usa una versión modificada del kernel de Linux en su versión 2.6 para adaptarse a las necesidades específicas de los dispositivos móviles (Lázaro, 2016).

2.1.2.2. Capa de abstracción de hardware (HAL)

La capa de abstracción de hardware (HAL) proporciona interfaces estándar que permiten que el hardware del dispositivo sea accesible para las API de Java de nivel superior. La HAL está formada por varios módulos de bibliotecas, cada uno de los cuales implementa una interfaz para un tipo específico de componente de hardware, como una cámara o el módulo de Bluetooth. Cuando una API necesita interactuar con el hardware del dispositivo, el sistema Android carga el módulo correspondiente del componente de hardware (Developers Android, 2024a).

2.1.2.3. Bibliotecas C/C++ nativas

En una capa superior del kernel de Linux, se encuentran varias bibliotecas de C/C++ que proporcionan las funcionalidades principales del sistema operativo. Soportan reproducción y grabación de varios formatos de audio y video. Gestionan el acceso a la interfaz gráfica del sistema, permitiendo la composición eficiente de diferentes superficies en la pantalla. Proporcionan soporte para gráficos 2D y 3D, es decir, capacidades gráficas avanzadas. Si una aplicación requiere almacenar datos de manera estructurada se le proporciona SQLite para realizar esta tarea. Proporcionan seguridad de red a través de SSL, así como capacidades de navegador web mediante WebKit (Boullosa, 2015).

2.1.2.4. Android Runtime (ART)

Android Runtime (ART) es el entorno de ejecución de aplicaciones introducido como reemplazo de Dalvik, la máquina virtual original de Android. ART utiliza un enfoque llamado Ahead-of-Time (AOT) compilation, que traduce el bytecode de las aplicaciones a código nativo durante la instalación. Esto mejora el rendimiento en comparación con la compilación en tiempo de ejecución (JIT - Just-In-Time) utilizada por Dalvik, aunque ART también admite JIT para ciertas optimizaciones. Se utiliza desde la versión 5.0 de Android llamada Lollipop (Developers Android, 2024b).

2.1.2.5. Framework de Aplicaciones

Según Boullosa (2015):

> Proporciona una plataforma de desarrollo libre de aplicaciones con gran riqueza e innovaciones (sensores, localización, servicios, barra de notificaciones, etc.).
>
> Esta capa ha sido diseñada para simplificar la reutilización de componentes.
>
> Los servicios más importantes que incluye son:
>
> - *Views*: conjunto de vistas
> - *Resource Manager*: proporciona acceso a recursos
> - *Activity Manager*: maneja el ciclo de vida de las aplicaciones y proporciona un sistema de navegación entre ellas.
> - *Notification Manager*: permite a las aplicaciones mostrar alertas personalizadas en la barra de estado.
> - *Content Providers*: Permiten que las aplicaciones compartan datos entre sí de manera segura y controlada (p. 27).

2.1.2.6. Aplicaciones del Sistema

En la capa superior de la arquitectura, se encuentran las aplicaciones del sistema. Estas son las aplicaciones básicas que vienen preinstaladas en dispositivos Android, como el navegador web, la aplicación de contactos y más. Además, incluyen herramientas esenciales para el funcionamiento del dispositivo, como los ajustes del sistema.

2.1.3. Componentes Principales de una Aplicación Android

Las aplicaciones Android están compuestas por varios componentes clave, cada uno con una función específica para crear una experiencia de usuario totalmente funcional:

2.1.3.1. Activities

Una actividad representa una pantalla con una interfaz de usuario en una aplicación Android. Las actividades son los bloques de construcción básicos para las interfaces de usuario de las aplicaciones. Cada actividad se presenta al usuario y puede interactuar con otras actividades. El ciclo de vida de una actividad es fundamental para entender cómo se comporta una aplicación en diferentes estados (Developers Android, 2024c).

2.1.3.2. Services

Un servicio es un componente que realiza operaciones en segundo plano sin una interfaz de usuario directa. Los servicios pueden ejecutarse indefinidamente en el fondo o responder a eventos específicos. Ejemplos comunes incluyen reproducir música en segundo plano, realizar sincronización de datos en segundo plano o manejar tareas largas y continuas (Developers Android, 2024d).

2.1.3.3. Broadcast Receivers

Los receptores de difusión permiten que una aplicación responda a mensajes y anuncios del sistema o de otras aplicaciones. Son componentes ligeros que reaccionan a eventos como cambios de conectividad, la recepción de mensajes SMS, o la descarga completa de la batería. Los receptores de difusión no tienen una interfaz de usuario y normalmente actúan para iniciar otros componentes como servicios o actividades (Developers Android, 2024e).

2.1.3.4. Content Providers

Los proveedores de contenido gestionan el acceso a un conjunto centralizado de datos estructurados, permitiendo que diferentes aplicaciones compartan datos de manera segura y controlada. Por ejemplo, una aplicación de contactos puede usar un proveedor de contenido para proporcionar acceso a la lista de contactos de manera que otras aplicaciones puedan consultar o modificar esa información con los permisos adecuados (Developers Android, 2024f).

2.1.4. Desarrollo de Aplicaciones Android

El desarrollo de aplicaciones en Android se realiza principalmente en Java o Kotlin, utilizando el Android Software Development Kit (SDK) y Android Studio como el entorno de desarrollo integrado (IDE) oficial.

2.1.4.1. Android Studio

Android Studio es el IDE oficial para el desarrollo de aplicaciones Android. Basado en IntelliJ IDEA, proporciona herramientas avanzadas de edición de código, diseño de interfaz de usuario, depuración, pruebas y emulación de dispositivos. Algunas características clave incluyen:

- Editor de código inteligente.
- Herramienta de diseño visual.
- Emulador de Android.
- Soporte para Kotlin y Java.

2.1.4.2. Lenguajes de Programación: Java y Kotlin

Java es el lenguaje original para el desarrollo de aplicaciones Android. Es ampliamente utilizado y conocido por los desarrolladores, lo que facilita el mantenimiento.

Kotlin es un lenguaje más moderno que ofrece una sintaxis más clara y menos propensa a errores. Kotlin mejora la productividad al reducir el código y proporciona características avanzadas como funciones de extensión, tipos nulos seguros, y corutinas para la programación asíncrona.

2.1.4.3. Estructura de Proyecto Android

Un proyecto Android típico tiene una estructura de directorios específica que organiza los archivos en módulos. Los principales directorios y archivos incluyen:

- ✓ *src/main/java*: Contiene el código fuente de la aplicación.
- ✓ *src/main/res*: Contiene recursos de la aplicación como layouts, cadenas de texto, y gráficos.
- ✓ *AndroidManifest.xml*: El archivo de manifiesto define los componentes principales de la aplicación, los permisos requeridos, y otros elementos esenciales.

2.1.5. Ciclo de Vida de Componentes de Android

Cada componente de Android tiene un ciclo de vida que gestiona su creación, ejecución y destrucción. Comprender estos ciclos de vida es crucial para desarrollar aplicaciones estables y eficientes.

2.1.5.1. Ciclo de Vida de una Actividad

El ciclo de vida de una actividad está compuesto por una serie de métodos que son llamados en diferentes puntos del tiempo:

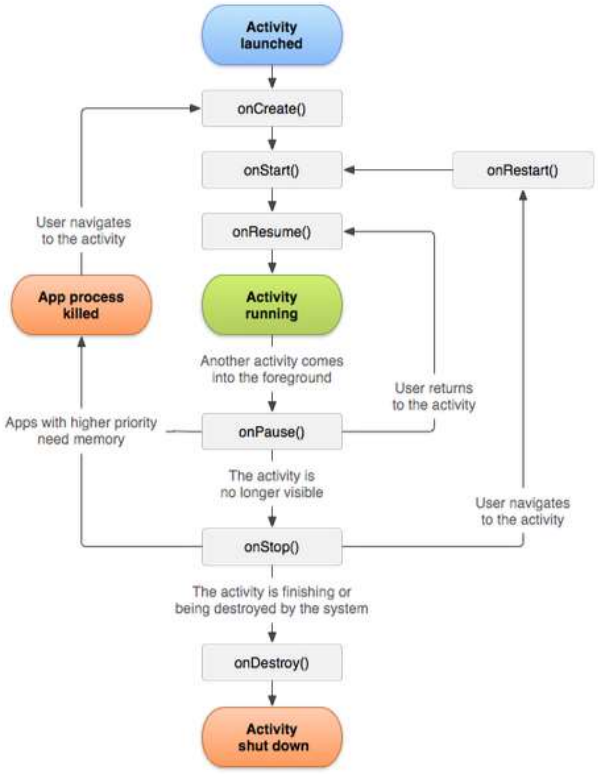


Figura 2: Ilustración simplificada del ciclo de vida de una actividad

Fuente: https://developer.android.com/guide/components/activities/activity-lifecycle?hl=es-419

- ✓ *onCreate ():* Llamado cuando la actividad se crea por primera vez. Aquí es donde se inicializan los componentes básicos de la actividad.
- ✓ *onStart ():* Llamado cuando la actividad se vuelve visible para el usuario.
- ✓ *onResume ():* Llamado cuando la actividad comienza a interactuar con el usuario. Es el estado en el que la actividad se encuentra en primer plano.
- ✓ *onPause ():* Llamado cuando la actividad está parcialmente oculta, generalmente porque otra actividad está tomando el foco, pero no la cubre completamente.
- ✓ *onStop ():* Llamado cuando la actividad ya no es visible para el usuario.
- ✓ *onDestroy ():* Llamado antes de que la actividad sea destruida. Aquí es donde se deben liberar los recursos (Developers Android, 2024g).

2.1.5.2. Ciclo de Vida de un Servicio

Los servicios también tienen un ciclo de vida específico:

- ✓ *onCreate ():* Llamado cuando el servicio se crea por primera vez.
- ✓ *onStartCommand ():* Llamado cada vez que un cliente solicita iniciar el servicio.
- ✓ *onBind ():* Llamado cuando un cliente se une al servicio. No todos los servicios utilizan este método.
- ✓ *onDestroy ():* Llamado cuando el servicio se destruye. Aquí es donde se liberan los recursos y se detienen las operaciones en curso (Developers Android, 2024d).

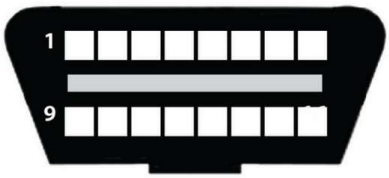
2.2. OBD2

OBD2 es la segunda generación de sistemas de diagnóstico para vehículos, diseñada para proporcionar información en tiempo real sobre el motor y otros sistemas del coche. El sistema OBD1, que se implementó en 1983, se hizo obligatorio en todos los automóviles a partir de 1991 para monitorear los componentes relacionados con las emisiones de gases. A diferencia de su predecesor, OBD2 rastrea una mayor cantidad de componentes del vehículo y alerta al conductor mediante una luz en el tablero si detecta algún problema eléctrico, químico o mecánico (Lozano, 2017).

El puerto OBD2, es estándar y permite a los mecánicos conectar dispositivos de diagnóstico para obtener datos sobre el estado del vehículo y los fallos detectados en tiempo real. En este trabajo, utilizaremos este puerto para recolectar la información necesaria sobre el estado del automóvil.

La obtención de los códigos de error se realiza a través de una interfaz que se comunica con las diferentes unidades de control del vehículo, no solo la del motor, sino también con sistemas como el ABS, ESR, ESP, airbag, entre otros. Con estas interfaces, es posible leer y borrar códigos de error, obtener datos de los sensores y activar distintos actuadores. Los protocolos utilizados por OBD2 son (McCord, 2011):

- ✓ ISO 9141-2 (vehículos europeos)
- ✓ SAE J1850 (vehículos americanos)
- ✓ ISO 14230-4 (KWP2000) (vehículos europeos e industriales)
- ✓ SAE J1708 (vehículos industriales en EE.UU.)
- ✓ Desde 2008, el bus CAN (ISO 15765-4)



1 Vendor Option
2 J1850 Bus +
3 Vendor Option
4 Chassis Ground
5 Signal Ground
6 CAN (J-2234) High
7 ISO 9141-2 K-Line
8 Vendor Option
9 Vendor Option
10 j1850 BUS
11 Vendor Option
12 Vendor Option
13 Vendor Option
14 CAN (J-2234) Low
15 ISO 9141-2 Low
16 Battery Power

Figura 3: Conector OBD2

Fuente: https://ricksfreeautorepairadvice.com/obd-ii-connector-pinout-diagrams/

La comunicación con el sistema OBD2 se realiza según el estándar SAEJ1979, el cual permite interrogar al sistema preguntándole por un PID (Parameter ID) y el sistema nos responde con el valor. El estándar OBD-II SAE J1979 define diez modos de operación (McCord, 2011):

Modo	Descripción
01	Muestra los parámetros disponibles
02	Muestra los datos almacenados por evento
03	Muestra los códigos de fallos de diagnóstico (Diagnostic Trouble Codes, DTC)
04	Borra los datos almacenados, incluyendo los códigos de fallos (DTC)
05	Resultados de la prueba de monitoreo de sensores de oxígeno
06	Resultados de la prueba de monitoreo de componentes del sistema
07	Muestra los códigos de fallo detectados durante el último ciclo de conducción
08	Operación de control de los componentes/sistema a bordo
09	Solicitud de información del vehículo
0A	Códigos de fallos (DTC) permanentes (borrados)

En el presente trabajo usaremos el modo 01 para preguntar por datos disponibles.

De todos los PID disponibles en el modo 01, para este trabajo usaremos los siguientes:

- 05 Temperatura del líquido refrigerante (ºC)
- 10 Flujo aire sensor MAF (gr/sec)
- 0b Presión en el colector de admisión (kpA)
- 0f Temperatura aire en el colector de admisión (ºC)
- 04 Carga del motor (%)
- 0c Revoluciones por minuto (rpm)
- 0d Velocidad (km/h)

Esto es debido a que el automóvil en el que se ha probado el presente trabajo solo admite estos 7 PID.

Cabe mencionar que los fabricantes de automóviles no están obligados a implementar todos los PID incluidos en el estándar.

2.3. Bus CAN

2.9.1 Introducción

Los sistemas de comunicaciones antiguos del automóvil necesitaban de una gran cantidad de cableado, ya que realizaban conexiones punto a punto, debido a que el

numero de centralitas de los coches creció exponencialmente, se pensó que los automóviles

necesitaban de un sistema de comunicaciones en bus que fuera fiable, con inmunidad a los ruidos y que redujera el cableado.

En 1982 la empresa Bosch desarrolla un protocolo de comunicaciones basado en una topología de bus para poder transmitir mensajes, este protocolo se implementa por primera

vez en 1992 en el Mercedes Clase E y se llamó bus CAN.

Desde 2008 el bus CAN es obligatorio para todos los vehículos, ofrece

- ✓ Reducción de cableado.
- ✓ Gran inmunidad a las interferencias y al ruido.
- ✓ Reducción de costes.
- ✓ Mayor velocidad de transmisión de datos.

Bus Can es un protocolo de comunicación en serie, desarrollado por Bosch para facilitar el intercambio de datos entre las unidades de control electrónicas en un automóvil.

Can significa Controller Area Network (Red de Área de Control).

Este sistema permite que una gran cantidad de información sea compartida entre las unidades de control conectadas al sistema, lo que resulta en una reducción significativa tanto en el número de sensores necesarios como en la cantidad de cables en la instalación eléctrica.

De esta manera, se incrementan notablemente las funciones disponibles en los sistemas del automóvil que utilizan Bus Can sin aumentar los costes, además de permitir que estas funciones estén distribuidas entre las diferentes unidades de control (Universidad Tecnológica Nacional, 2009)

2.9.2 Breve definición

Universidad Tecnológica Nacional (2009) define las principales características del protocolo CAN de la siguiente forma:

Transmisión de Información: Los datos que se intercambian entre las unidades de control a través de los dos cables (bus) son paquetes de bits (0 y 1) con una longitud limitada y una estructura de campos bien definida que componen el mensaje.

Identificación de Mensajes: Uno de los campos del mensaje actúa como identificador, especificando el tipo de dato, la unidad de control que lo envía y la prioridad de transmisión. El mensaje no está dirigido a ninguna unidad específica; cada unidad determina si el mensaje es relevante para ella basándose en este identificador.

Capacidad de Transmisión y Recepción: Todas las unidades de control pueden actuar tanto como transmisoras como receptoras, y el número de unidades conectadas al sistema puede variar dentro de ciertos límites.

Solicitud de Información: Si es necesario, una unidad de control puede solicitar información específica a otra mediante uno de los campos del mensaje, conocido como trama remota o RDR.

Gestión de Conflictos: Cualquier unidad de control puede introducir un mensaje en el bus siempre que esté libre. Si dos unidades intentan transmitir al mismo tiempo, el conflicto se resuelve mediante la prioridad del mensaje indicada por el identificador.

Mecanismos de Seguridad: El sistema cuenta con mecanismos que aseguran la correcta transmisión y recepción de los mensajes. Si un mensaje contiene un error, se anula y se retransmite correctamente. Además, una unidad con problemas puede informar a las demás a través del mensaje. Si la unidad no puede recuperarse, se desconecta del sistema, pero el resto del sistema continúa funcionando.

La información se transmite a través de dos cables trenzados que conectan todas las unidades de control del sistema. Esta información se transmite mediante la diferencia de voltaje entre los dos cables, de modo que un alto voltaje representa un 1 y un bajo voltaje representa un 0. La combinación adecuada de unos y ceros forma el mensaje que se transmite.

En uno de los cables, los valores de voltaje varían entre 0V y 2.25V, por lo que se le llama cable L (Low). En el otro cable, el cable H (High), los valores de voltaje oscilan entre 2.75V y 5V. Si la línea H se interrumpe o se conecta a tierra, el sistema operará utilizando la señal del cable Low con respecto a la tierra. Si la línea L se interrumpe, ocurrirá lo contrario. Esta configuración permite que el sistema continúe funcionando incluso si uno de los cables está cortado o conectado a tierra.

Es fundamental mantener el trenzado de ambos cables, ya que esto neutraliza los campos magnéticos. Por lo tanto, no se debe alterar ni el trenzado ni la longitud de estos cables bajo ninguna circunstancia.

En el bus can existen resistencias conectadas a los extremos de los cables H y L, previenen fenómenos de reflexión que podrían interferir con el mensaje.

Por razones de economía y seguridad operativa, estas resistencias están ubicadas dentro de algunas de las unidades de control del sistema.

Las unidades de control que se conectan al sistema Bus Can necesitan compartir información, ya sea que pertenezcan al mismo sistema o no. En el ámbito de los automóviles, por lo general, las unidades de control del motor, ABS y cambio automático están conectadas a un bus, mientras que las unidades de control relacionadas con el sistema de confort están en otro bus de menor velocidad. El Bus Can está orientado hacia el mensaje en lugar del destinatario. La información en la línea se transmite en forma de mensajes estructurados, donde una parte del mensaje es un identificador que indica el tipo de dato que contiene. Todas las unidades de control reciben el mensaje, lo filtran y solo las unidades que necesitan ese dato lo utilizan. Todas las unidades de control conectadas al sistema pueden tanto enviar como recibir mensajes desde la línea. Cuando el bus está libre, cualquier unidad conectada puede comenzar a transmitir un nuevo mensaje.

Si varias unidades intentan enviar un mensaje al mismo tiempo, entonces la unidad con mayor prioridad es la que transmitirá el mensaje.

El controlador de la unidad envía los datos junto con su identificador y la solicitud de inicio de transmisión, asegurándose de que el mensaje sea correctamente enviado a todas las unidades de control asociadas. Para enviar el mensaje, debe primero encontrar el bus libre y, en caso de colisión con otra unidad intentando transmitir simultáneamente, tener una prioridad más alta. Una vez que esto ocurre, las demás unidades de control actúan como receptores.

Después de que todas las unidades de control reciben el mensaje, verifican el identificador para determinar si el mensaje es relevante para ellas. Aquellas unidades de control que necesiten los datos del mensaje los procesan, si no, simplemente ignoran el mensaje.

El sistema de bus Can está equipado con mecanismos para detectar errores en la transmisión de mensajes. Todos los receptores realizan una verificación del mensaje analizando una parte del mismo, conocida como campo CRC. Además, las unidades emisoras aplican otros mecanismos de control, como la monitorización del nivel del bus, la presencia de campos de formato fijo en el mensaje (verificación de la trama), y análisis estadísticos de fallos por parte de las unidades de control. Estas medidas garantizan que las probabilidades de error en la emisión y recepción de mensajes sean extremadamente bajas, convirtiendo al sistema en altamente seguro.

2.4. ELM327

2.4.1. ¿Qué es el ELM327?

Figura 4: Dispositivo ELM327

Fuente: https://www.norauto.es/p/interface-nk-bluetooth-2.0-obd-ii-elm327-26738.html

El ELM327 es un dispositivo desarrollado por la empresa Elm Electronics, diseñado para interpretar comandos de diagnóstico del sistema OBD-II (On-Board Diagnostics II) de vehículos. Este dispositivo actúa como un traductor entre la interfaz OBD-II del vehículo y dispositivos externos, como ordenadores, smartphones o tablets, que desean comunicarse con el vehículo para obtener datos o realizar diagnósticos.

El ELM327 es famoso por su capacidad para soportar múltiples protocolos de comunicación OBD-II, lo cual lo hace versátil y aplicable a una amplia gama de vehículos fabricados desde 1996 en adelante.

2.4.2. Protocolos de Comunicación Compatibles

El ELM327 soporta varios protocolos de comunicación, lo que lo hace adaptable a diferentes modelos de automóviles que utilizan diferentes estándares (Elm Electronics Inc, 2010):

- SAE J1850 PWM (Pulse Width Modulation): Utilizado principalmente por vehículos Ford en América del Norte. Funciona a una velocidad de 41.6 kbps y utiliza una señal de voltaje de modulación de ancho de pulso para la comunicación.

- SAE J1850 VPW (Variable Pulse Width): Utilizado por GM y otros fabricantes en América del Norte. Este protocolo opera a una velocidad de 10.4 kbps y también utiliza modulación de ancho de pulso, pero con diferentes características en comparación con PWM.

- ISO 9141-2: Utilizado principalmente en vehículos europeos y asiáticos, incluidos algunos de los primeros modelos de vehículos japoneses. Este protocolo es similar en funcionalidad al KWP2000 pero tiene diferentes velocidades de comunicación y requisitos de hardware.

- ISO 14230-4 (KWP2000 - Keyword Protocol 2000): Otro protocolo común en vehículos europeos y asiáticos, utilizado para el diagnóstico de vehículos. Es una evolución del ISO 9141 y permite velocidades de comunicación más rápidas.

- ISO 15765-4 (CAN - Controller Area Network): Este es el protocolo más moderno y ampliamente utilizado, implementado en la mayoría de los vehículos a partir de 2008. Ofrece una comunicación robusta y rápida a través de un bus de datos CAN, que permite velocidades de hasta 1 Mbps.

2.4.3. Arquitectura Interna del ELM327

El microcontrolador ELM327 está basado en una arquitectura que facilita la interpretación y la traducción de los comandos OBD-II. Su núcleo interno generalmente es un microcontrolador PIC de la empresa Microchip Technology, adaptado con firmware específico de Elm Electronics Inc. A continuación, se describen algunos componentes clave:

Unidad Central de Procesamiento (CPU): La CPU es el núcleo del ELM327 y es responsable de ejecutar las instrucciones del firmware, gestionando tanto la entrada como la salida de datos entre el vehículo y el dispositivo de usuario.

Memoria Flash: Almacena el firmware del ELM327, que incluye el conjunto de instrucciones que el microcontrolador utiliza para operar y gestionar las comunicaciones.

Memoria RAM: Utilizada para el almacenamiento temporal de datos durante la operación. Permite el almacenamiento de comandos y respuestas intermedias.

Transceptores de Línea: Permiten la comunicación con diferentes protocolos OBD-II. El ELM327 incluye transceptores que convierten las señales del vehículo en un formato comprensible para el microcontrolador.

Convertidores A/D (Analógico a Digital): Utilizados para interpretar señales analógicas desde el vehículo, convirtiéndolas en datos digitales que pueden ser procesados por el microcontrolador.

2.4.4. Funcionamiento del ELM327

2.4.4.1. Comunicación y Transferencia de Datos

El ELM327 se conecta al puerto OBD-II del vehículo, que suele estar ubicado cerca del tablero o en la columna de dirección.

Al conectar el dispositivo, el ELM327 inicia una secuencia de inicio en la que detecta automáticamente el protocolo de comunicación del vehículo y establece una conexión.

Una vez detectado, ajusta su configuración interna para comunicarse correctamente con el vehículo.

Una vez que el protocolo está configurado, el ELM327 puede recibir comandos desde un dispositivo externo (como un smartphone o computadora) y traducirlos al formato necesario para la ECU (Engine Control Unit) del vehículo.

Después de recibir los comandos de la ECU, el ELM327 traduce las respuestas de vuelta al formato comprensible para el dispositivo externo. Estos datos pueden incluir códigos de error, datos en tiempo real de sensores, entre otros.

2.4.5. Conexión y Configuración Bluetooth/Wi-Fi

El ELM327 viene en variantes que soportan comunicación a través de Bluetooth, Wi-Fi o USB. Las versiones Bluetooth y Wi-Fi permiten la conexión inalámbrica con dispositivos móviles y computadoras, mientras que la versión USB se utiliza principalmente para conexiones por cable.

Los adaptadores ELM327 con Bluetooth crean una conexión serial con dispositivos externos, que permite enviar y recibir datos OBD-II. Requiere emparejamiento con el dispositivo móvil y suele ser compatible con aplicaciones móviles de diagnóstico.

Las versiones Wi-Fi crean un punto de acceso local al que los dispositivos externos pueden conectarse. Esto es útil cuando se requiere mayor velocidad de transferencia de datos o cuando el dispositivo móvil no tiene capacidad Bluetooth.

2.4.6. Aplicaciones Avanzadas del ELM327

2.4.6.1. Diagnóstico de Vehículos y Mantenimiento Preventivo

El ELM327 se utiliza ampliamente para el diagnóstico de vehículos, proporcionando a los usuarios acceso a información detallada sobre el estado del motor y otros sistemas.

Los códigos de error o DTC (Diagnostic Trouble Codes) son mensajes almacenados en la ECU del vehículo que indican problemas con los sistemas del vehículo. El ELM327 puede leer estos códigos y proporcionar descripciones que ayudan a los usuarios a identificar problemas específicos.

Además de los códigos de error, el ELM327 puede proporcionar datos en tiempo real de sensores del vehículo, como temperatura del motor, velocidad del vehículo, presión de aceite, nivel de combustible, entre otros. En el presente trabajo se utilizará esta información para mostrarla en una aplicación Android y luego enviarla por internet a una base de datos externa para su posterior visualización.

2.4.7. Desarrollo de Aplicaciones Personalizadas

El ELM327 permite a los desarrolladores crear aplicaciones personalizadas que interactúan con los vehículos a través de la interfaz OBD-II.

Esto es posible gracias a su interfaz de comandos AT, que permite enviar instrucciones específicas al microcontrolador y recibir su respuesta (Elm Electronics Inc, 2010).

Ejemplos de comandos AT:

AT Z Reinicia el dispositivo.

AT I Muestra la identificación del dispositivo.

AT RV Lee el voltaje de la batería.

AT SP Selecciona el protocolo de comunicación.

2.5. Bases de datos para series temporales

Algunas definiciones:

> Una serie temporal es una sucesión de datos observados que tienen un valor que podemos denominar X, a lo largo de un periodo de tiempo determinado, que podemos llamar T.
>
> La forma habitual de representar las series temporales es mediante un gráfico de la serie temporal, donde:
>
> En el eje x del gráfico se representan los valores de tiempo, sería una variable independiente.
>
> Y el eje y muestra los valores de la variable observada, objeto del estudio, que es una variable dependiente (Garzó, 2023, p. 12).

> Una base de datos de series de tiempo podría definirse como un sistema gestor de bases de datos, que como su propio nombre indica, presta especial atención a los aspectos temporales, contando con un modelo de datos temporal y una versión temporal del lenguaje de consulta estructurado (Nazco, 2018, p. 4).

Como podemos observar en las definiciones anteriores, una serie temporal no es más que el valor de una variable (o varias) medida a intervalos regulares de tiempo, así pues, una base de datos de series temporales es un gestor preparado específicamente para tratar con series temporales.

Los datos que se pretende enviar en este trabajo no son datos estructurados, son medidas realizadas periódicamente de los valores de las variables, con lo cual una base de datos relacional no es el sistema óptimo para guardar estos datos, por esta razón se toma la decisión de guardar la información en una base de datos de series temporales la cual está diseñada específicamente para guardar este tipo de datos.

Comparativa de bases de datos para series temporales:

BASES DE DATOS									
Características	**InfluxDB**	**Graphite**	**OpenTSDB**	**Prometheus**	**Blueflood**	**Druid**	**Hawkular**	**Heroic**	**KairosDB**
Versión Inicial	2013	2006	2011	2015	2013	2012	2014	2014	2013
Licencia	Código Abierto	Código Abierto	Código Abierto	Código Abierto	Código Abierto	Código Abierto	Código Abierto	Código Abierto	Código Abierto
Basado en la nube	No	No	No	No	No	No	No	No	No
Lenguaje de implementación	Go	Python	Java	Go	Java	Java	Java	Java	Java
Sistemas operativos de servidor	Linux OS X	Linux Unix	Linux Windows	Linux Windows	Linux OS X	Linux OS X Unix	Linux Windows OS X	Linux	Linux OS X Windows
Esquema de datos	Esquema Libre	Sí	Esquema Libre	Sí	Esquema Predefinido	Sí	Esquema Libre	Esquema Libre	Esquema Libre
Mecanografía	Datos numéricos y cadenas	Datos numéricos	Datos numéricos para métricas, cadenas para etiquetas	Datos numéricos únicamente	Sí	Sí	Sí	Sí	Sí
Soporte XML	No	No	No	No	No	No	No	No	No
Índices secundarios	No	No	No	No	No	Sí	No	Sí	No
SQL	SQL-like query language	No	No	No	No	Soporte SQL a través del adaptador Apache Calcite	No	No	No
APIs y otros métodos de acceso	HTTP API JSON over UDP	HTTP API Sockets	HTTP API Telnet API	RESTful HTTP/JSON API	HTTP REST	JSON over HTTP	HTTP REST	HQL HTTP API	Graphite Protocol HTTP REST Telnet API
Lenguajes de programación soportados	.Net Clojure Erlang Go Haskell Java JavaScript(Node.js) Lisp Perl PHP Python R Ruby Rust Scala	JavaScript(Node.js) Python	Erlang Go Java Python R Ruby	.Net C++ Go Haskell Java JavaScript(Node.js) Python Ruby	Java Python	Clojure JavaScript PHP Python R Ruby Scala	Go Java Python Ruby	Java Python	Java JavaScript PHP Python
Scripts del lado del servidor	No	No	No	No	No	No	No	No	No
Desencadenantes	No	No	No	No	No	No	Sí	No	No
Métodos de particionamiento	Sharding	Ninguno	Sharding	Sharding	Sharding	Sharding	Sharding	Sharding	Sharding
Métodos de replicación	Factor de replicación seleccionable	Ninguno	Factor de replicación seleccionable	Sí	Factor de replicación seleccionable	Sí, a través de HDFS, S3 u otros motores de almacenamiento	Factor de replicación seleccionable	Sí	Factor de replicación seleccionable
Mapa reducido	No	No	No	No	No	No	No	No	No
Llaves exteriores	No	Ninguno	No	No	No	No	No	No	No
Conceptos de transacción	No	No	No	No	No	No	No	No	No
Concurrencia	Sí	No	Sí	Sí	Sí	Sí	Sí	Sí	Sí
Durabilidad	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
Capacidades de memoria	Sí	Sí	No	No	No	No	No	No	No
Conceptos de usuario	Simple gestión de derechos a través de cuentas de usuario	No	No	No	No	No	No	No	No

Figura 5: Comparativa de bases de datos de series temporales

Fuente: (Nazco, 2018, p. 7)

Se puede observar en la web *https://db-engines.com/en/ranking/time+series+dbms* que en Septiembre de 2024 la base de datos de series temporales más utilizada en el mundo era InfluxDB:

☐ include secondary database models — 44 systems in ranking, September 2024

Rank Sep 2024	Rank Aug 2024	Rank Sep 2023	DBMS	Database Model	Score Sep 2024	Score Aug 2024	Score Sep 2023
1.	1.	1.	InfluxDB	Time Series, Multi-model	22.12	-0.50	-9.15
2.	2.	2.	Kdb	Multi-model	7.73	-0.24	-1.21
3.	3.	3.	Prometheus	Time Series	7.56	+0.39	-0.06
4.	4.	4.	Graphite	Time Series	5.19	-0.16	-0.26
5.	5.	5.	TimescaleDB	Time Series, Multi-model	4.06	-0.01	-1.33
6.	6.	↑ 7.	Apache Druid	Multi-model	2.85	-0.10	-0.35
7.	7.	↑ 10.	QuestDB	Time Series, Multi-model	2.81	-0.06	+0.43
8.	8.	↓ 6.	DolphinDB	Multi-model	2.72	-0.05	-1.32
9.	9.	9.	TDengine	Time Series, Multi-model	2.48	-0.03	-0.14
10.	10.	↑ 11.	GridDB	Time Series, Multi-model	1.91	-0.07	-0.26
11.	11.	↓ 8.	RRDtool	Time Series	1.70	-0.02	-1.43
12.	12.	12.	OpenTSDB	Time Series	1.59	0.00	-0.52
13.	13.	13.	Fauna	Multi-model	1.55	-0.01	-0.13
14.	14.	↑ 18.	Apache IoTDB	Time Series	1.31	+0.03	+0.33
15.	15.	↓ 14.	VictoriaMetrics	Time Series	1.23	+0.03	-0.14
16.	16.	↓ 15.	Amazon Timestream	Time Series	1.22	+0.03	+0.09
17.	17.	↓ 16.	M3DB	Time Series	1.02	+0.04	-0.05
18.	↑ 19.	↓ 17.	eXtremeDB	Multi-model	0.79	+0.10	-0.23

Figura 6: Ranking de bases de datos de series temporales

Fuente: https://db-engines.com/en/ranking/time+series+dbms

Según Garzó (2023), InfluxDB es una base de datos de series temporales de código abierto, diseñada para gestionar y consultar grandes volúmenes de datos temporales en tiempo real. Algunas características de esta base de datos son:

- InfluxDB está optimizada para manejar grandes cantidades de datos de series temporales, permitiendo realizar consultas y escrituras con gran rapidez.
- Los datos se almacenan de manera eficiente, lo que maximiza la capacidad disponible para los registros.

- Es posible escalar InfluxDB de manera horizontal, facilitando el manejo de grandes volúmenes de datos temporales.
- InfluxDB se comunica a través de una API RESTful, lo que la hace flexible y compatible con otros sistemas, permitiendo un intercambio seguro de información a través de la red.

Así pues, observamos que InfluxDB es la base de datos para series temporales más utilizada actualmente, por otro lado, es la que más lenguajes de programación soporta y además se ejecuta en Linux y es escalable horizontalmente, también maneja grandes cantidades de datos de manera óptima, por lo tanto, se toma la decisión de usar InfluxDB como base de datos para guardar los datos leídos del automóvil.

2.6. Visualización de métricas

Grafana es una plataforma de análisis y visualización de métricas que permite a los usuarios crear cuadros de mando interactivos y personalizables (dashboards) a partir de una amplia variedad de fuentes de datos. Está diseñado principalmente para ofrecer una visión en tiempo real de los datos almacenados en sistemas distribuidos, lo que lo convierte en una herramienta poderosa para monitorizar infraestructuras y aplicaciones.

Una de las características más atractivas de Grafana es su capacidad para crear *dashboards* personalizables que pueden incluir gráficos, tablas, medidores, mapas de calor, y más. Los dashboards pueden ser diseñados para visualizar datos de una manera que sea intuitiva para el usuario. Algunos componentes de visualización son:

- ✓ *Gráficos de líneas y áreas:* Ideales para mostrar tendencias de datos en el tiempo, como el uso de CPU o el tráfico de red.
- ✓ Tablas: Muestran datos tabulares detallados con la posibilidad de aplicar filtros, ordenación y paginación.
- ✓ *Mapas de calor:* Utilizados para visualizar patrones de alta y baja densidad en los datos.
- ✓ *Alertas:* Permiten la creación de alertas basadas en reglas definidas por el usuario, que pueden dispararse si se alcanzan ciertos umbrales.

3. Estado del Arte

A continuación, se hace un breve recorrido para describir las algunas aplicaciones existentes en el ámbito de monitorización de las variables de un automóvil en el contexto de este trabajo:

3.1. Torque

- Aplicación Android creada por Ian J. Hawkins que tiene una versión gratuita y otra de pago, permite leer códigos de error guardados en el vehículo, así como también permite leer valores en tiempo real.

En su versión gratuita supuestamente te permite guardar los datos para su posterior explotación y visualización, pero según la prueba realizada en este trabajo esta característica no funciona ni tampoco hay una explicación clara del formato en el cual se envían los datos y las indicaciones sobre cómo explotarlos después.

Figura 7: Aplicación Torque

Fuente: https://obd2-elm327.es/torque-android

3.2. Obd auto doctor

Aplicación Android muy similar a la anterior, permite leer códigos de fallos registrados en el vehículo además de realizar diagnósticos predeterminados, también permite leer sensores en tiempo real, no permite el envío de datos a ningún servidor externo.

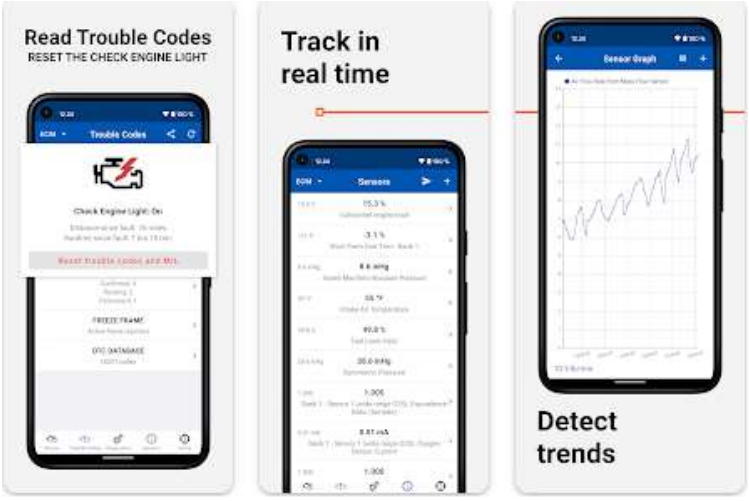


Figura 8: Aplicación OBD Auto Doctor

Fuente: https://play.google.com/store/apps/details?id=com.obdautodoctor&hl=es_419&pli=1

3.3. DashCommand

Aplicación desarrollada por Palmer Performance Engineering, permite la monitorización en tiempo real de muchos parámetros del vehículo, muy parecida a Torque, pero no tiene la posibilidad de enviar datos a un servidor externo para su posterior análisis.

Figura 9: Aplicación DashCommand

Fuente: https://play.google.com/store/apps/details?id=com.palmerperformance.DashCommand&hl=es

3.4. Car Scanner

Muy parecida a la anterior, tiene versión para Android y para Iphone, en su versión gratuita tiene muchas limitaciones como por ejemplo no deja borrar los fallos detectados, además tiene publicidad. No tiene la posibilidad de enviar los datos leídos de los sensores externamente.

Figura 10: Aplicación Car Scanner

Fuente: https://play.google.com/store/apps/details?id=com.ovz.carscanner&hl=es

3.5. Diagnóstico Tor OBD2

Aplicación que permite leer y borrar códigos de avería, no permite ver los valores de los sensores y por lo tanto tampoco exportar éstos a un servidor externo.

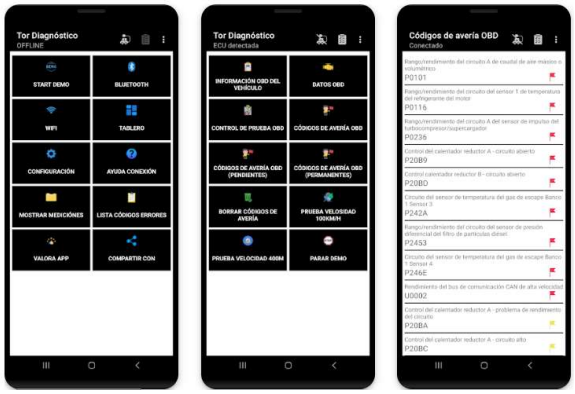


Figura 11: Diagnóstico Tor OBD2

Fuente: https://play.google.com/store/apps/details?id=com.aplications.main.torobid&hl=es

3.6. OBD Mary

Aplicación que dispone de la posibilidad de ver datos en tiempo real de los distintos sensores, también dispone de leer códigos de avería, además dispone de la función grabación de datos la cual se puede usar para guardar los distintos valores de una variable durante un tiempo.

No dispone de la funcionalidad de los datos medidos de los sensores hacia un servidor externo.

Figura 12: Obd Mary

Fuente: https://play.google.com/store/search?q=obd%20mary&c=apps&hl=es

3.7. CAN bus data loggers de la empresa CSS electronics

Esta empresa ofrece sus dispositivos hardware los cuales se conectan al bus can del automóvil y permiten transmitir los datos por USB, WIFI o 3G/4G.

Esta solución no proporciona un servicio para visualizar los datos, pero facilita un cuadro de mando gratuito que podemos bajarnos para la aplicación de visualización de datos llamada Grafana. Es decir, esta solución nos vende el hardware para recolección de datos y nos proporciona un cuadro de mandos el cual podemos usar para montarnos nosotros mismos nuestra infraestructura.

Figura 13: Productos de CSS Electronics

Fuente: https://www.csselectronics.com/

4. Objetivos y metodología de trabajo

Después de analizar el contexto y el estado del arte actual, se observa que no existe ninguna aplicación Android ni arquitectura gratuita y libre que nos permita monitorizar un vehículo en tiempo real y a tiempo pasado, así como también que una persona con pocos conocimientos técnicos pueda conseguir monitorizar su vehículo con coste casi cero.

4.1. Objetivo general

Conseguir una infraestructura gratuita que permita monitorizar un automóvil en tiempo real y transmitir esos datos a una base de datos externa para así poder realizar un análisis de los datos recogidos en cualquier momento.

4.2. Objetivos específicos

Implementar una aplicación Android que permita visualizar en el teléfono móvil los datos extraídos del automóvil. Esta aplicación se conectará mediante Bluetooth a la interfaz ELM327 y preguntará constantemente al automóvil por el valor de las variables, correrá en un teléfono móvil con sistema operativo Android.

Además, a los datos obtenidos del automóvil, se le añadirán los datos de posición GPS obtenidos del teléfono y todo junto se enviará al servidor de métricas mediante la tecnología 4G del teléfono.

Instalar una máquina virtual GNU-Linux mediante el software gratuito de virtualización Virtualbox

Instalar el software de servidor de bases de datos para series temporales Influxdb.

Instalar el software de visualización de métricas llamado Grafana.

4.3. Requisitos funcionales

A continuación, se describen los requisitos funcionales que debe implementar la aplicación desarrollada en este trabajo, ésta corre en un teléfono móvil con sistema operativo Android.

Identificador: RQF-01
Nombre: Listar dispositivos Bluetooth vinculados **Descripción**: La aplicación deberá ser capaz de mediante la pulsación de un botón mostrar en pantalla los dispositivos Bluetooth vinculados previamente en el smartphone. **Importancia**: Alta

Identificador: RQF-02
Nombre: Conectar mediante Bluetooth al dispositivo ELM327 **Descripción**: Cuando el usuario seleccione el dispositivo elm327 de la lista de dispositivos vinculados, la aplicación se conectará a él mediante Bluetooth e informará del estado de la conexión mediante el textview “estado”. **Importancia**: Alta

Identificador: RQF-03
Nombre: Arrancar **Descripción**: La aplicación deberá ser capaz de mediante la pulsación del botón "empezar" a preguntar al vehículo los PID y mostrar los valores en los texview correspondientes. **Importancia**: Alta

Identificador: RQF-04
Nombre: Envío de datos a la base datos para series temporales **Descripción**: Mediante un switch la aplicación permitirá o no el envío de datos a la base de datos para series temporales remota definida en el textview “Servidor” **Importancia**: Alta

Identificador: RQF-05
Nombre: Mostrar número de identificación del vehículo **Descripción**: A través de un textview la aplicación deberá ser capaz de mostrar el número de identificación del vehículo (VIN) **Importancia**: Alta

Identificador: RQF-06
Nombre: Mostrar posición GPS **Descripción**: A través de los textview correspondientes, la aplicación deberá ser capaz de mostrar la posición GPS actual del teléfono **Importancia**: Alta

Identificador: RQF-07
Nombre: Indicador visual a color de revoluciones por minuto **Descripción**: Mediante un progressBar, la aplicación indicará cuando las rpm están por debajo de 2000 en color rojo y por encima de 2000 en color verde **Importancia**: Baja

Identificador: RQF-08
Nombre: Salir de la aplicación **Descripción**: A través de un menú se realizará la salida ordenada de la aplicación **Importancia**: Baja

4.4. Requisitos no funcionales

A continuación, se describen los requisitos no funcionales que debe cumplir la aplicación desarrollada en este trabajo.

Identificador: RNF-01
Nombre: Menor latencia posible **Descripción**: Mediante ajustes empíricos de la aplicación, se intentará tener la menor latencia posible en la visualización de datos **Importancia**: Baja

Identificador: RNF-02
Nombre: Accesibilidad **Descripción**: La visualización de métricas deberá estar en una fuente grande **Importancia**: Alta

Identificador: RNF-03
Nombre: Compatibilidad **Descripción**: La aplicación deberá ser compatible con Android **Importancia**: Alta

4.5. Arquitectura de la solución

El dispositivo ELM327 se conectará al puerto OBD2 del vehículo y creará una conexión Bluetooth para que el teléfono Android se pueda conectar a él.

Después la aplicación Android mostrará los datos en pantalla y enviará los datos al servidor de métricas InfluxDB, posteriormente estas métricas se visualizarán mediante un dashboard de Grafana y cuando los valores sobrepasen un umbral, se enviarán alarmas a Telegram.

A continuación, se describe la arquitectura implementada en este trabajo mediante un esquema:

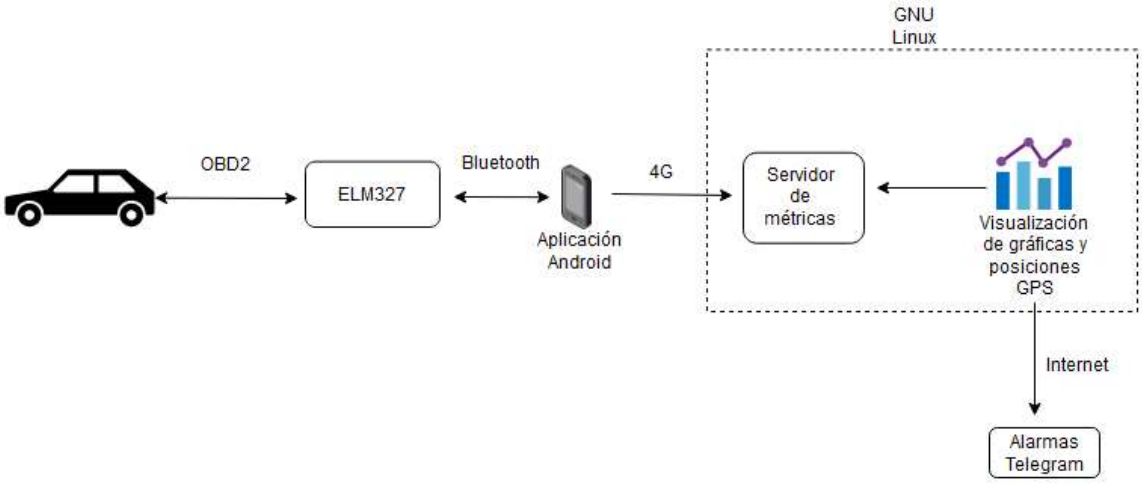


Figura 14: Arquitectura de la solución

Fuente: Elaboración propia

Descripción esquemática de la funcionalidad que ofrece una interfaz OBD:

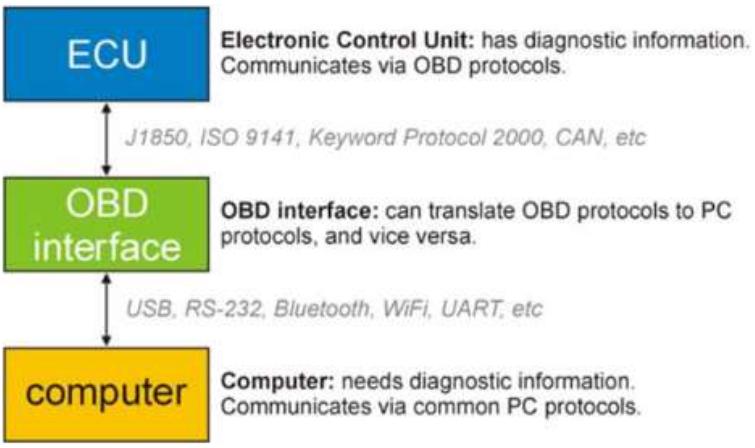


Figura 15: Esquema de conexión OBD

Fuente: Javier San José González, TFG 2018

El dispositivo ELM327 proporciona una interface de comunicación por medio de comandos AT a través de Bluetooth.

La comunicación que realizará la aplicación Android será del tipo “pregunta-respuesta” (SAE J1939) es decir estará continuamente preguntando por los PID y el ELM327 estará continuamente respondiéndole.

La aplicación Android mostrará los datos en tiempo real por pantalla, después añadirá la posición GPS obtenida del teléfono y enviará esa información al servidor de métricas que en este trabajo será el software Influxdb.

Posteriormente el software de visualización de métricas que en este trabajo será Grafana se conectará al servidor de métricas para ofrecer una interfaz de explotación de datos.

En un primer momento se implementó un servidor elasticsearch para guardar los datos y con su producto de visualización kibana se intentaron realizar visualizaciones de las series temporales, pero no se obtuvieron resultados satisfactorios debido a que elasticsearch está más enfocado a buscar cadenas de texto dentro de logs que al almacenamiento de series temporales.

Para realizar la aplicación Android, se ha partido de una ya existente y que su licencia permite la reutilización de código, está ubicada en:

https://github.com/bauerjj/Android-Simple-Bluetooth-Example

Esto ha permitido realizar más rápidamente la conexión Bluetooth.

La aplicación desarrollada en Android muestra los dispositivos vinculados previamente por Bluetooth presionando el botón "Dispositivos vinculados", después damos clic en el dispositivo y establecemos la comunicación Bluetooth con el dispositivo ELM327, el estado de la conexión se mostrará entonces como "conectado al dispositivo OBDII".

Después al hacer clic en el botón "empezar" la aplicación empezará a realizar preguntas al dispositivo ELM327 y mostrará las respuestas en pantalla:

Figura 16: Aplicación Android realizada 1

Fuente: Elaboración propia

Por defecto se enviarán los datos a la url "servidor", si no queremos que se envíen y solo queremos visualizar datos podemos poner en "off" el switch "envía datos cloud".

Como funcionalidad añadida, se dispone de una barra en verde que nos indicará que las revoluciones por segundo están por encima de 2000, en caso contrario la barra se mostrará en rojo.

Figura 17: Aplicación Android realizada 2

Fuente: Elaboración propia

4.5.1. Código fuente de la aplicación Android

El código de la aplicación Android y del dashboard de Grafana están en el repositorio público: https://gitlab.com/uni5362831/tfg

La aplicación cuando arranca espera la entrada por el usuario del dispositivo OBD2 previamente vinculado al cual se tiene que conectar y después de presionar el botón "empezar" se envían comandos de inicialización a la interfaz ELM327 como se puede observar en el código fuente:

```
307     private void initialize() {
308
309         if (mConnectedThread != null) {
310             mConnectedThread.write("ATZ"+ "\r" + "\n");//Reset
311             try {
312                 Thread.sleep(200);
313             } catch (InterruptedException e) {
314                 throw new RuntimeException(e);
315             }
316             mConnectedThread.write("ATPP0CSV23" + "\r" + "\n");//115200 baudios
317             try {
318                 Thread.sleep(200);
319             } catch (InterruptedException e) {
320                 throw new RuntimeException(e);
321             }
322             mConnectedThread.write("ATPP0CON"+ "\r" + "\n" );//Enable Programable Parameter
323             try {
324                 Thread.sleep(200);
325             } catch (InterruptedException e) {
326                 throw new RuntimeException(e);
327             }
328             mConnectedThread.write("ATZ" + "\r" + "\n");//Reset
329             try {
330                 Thread.sleep(200);
331             } catch (InterruptedException e) {
332                 throw new RuntimeException(e);
333             }
334             mConnectedThread.write("ATSP0"+ "\r" + "\n");//Automatic protocol
335             try {
336                 Thread.sleep(200);
337             } catch (InterruptedException e) {
338                 throw new RuntimeException(e);
339             }
340             mConnectedThread.write("ATE0"+ "\r" + "\n" );//echo OFF
341             try {
342                 Thread.sleep(200);
343             } catch (InterruptedException e) {
344                 throw new RuntimeException(e);
345             }
```

Figura 18: Inicialización de la interfaz ELM327

Fuente: Elaboración propia

Después de la inicialización de la interfaz ELM327, se crean todos los Threads de java que serán los encargados tanto de preguntar por el valor de las variables, como de enviar estos valores a la base de datos de series temporales:

```
367             SenderCoolantTemp worker_temp_coolant = new SenderCoolantTemp(mConnectedThread);
368             worker_temp_coolant.start();
369             worker_temp_coolant.setPriority(5);
370
371             SenderSpeed worker_speed = new SenderSpeed(mConnectedThread);
372             worker_speed.start();
373             worker_speed.setPriority(5);
374
375             SenderRpm worker_rpm = new SenderRpm(mConnectedThread);
376             worker_rpm.start();
377             worker_rpm.setPriority(5);
378
379
380             SenderIntakeManifoldAbsolutePressure worker_intake_manifold_absolute_pressure = new SenderIntakeManifoldAbsolutePressure(mConnectedThr
381             worker_intake_manifold_absolute_pressure.start();
382             worker_intake_manifold_absolute_pressure.setPriority(5);
383
384
385             SenderMotorLoad worker_motor_load = new SenderMotorLoad( mConnectedThread);
386             worker_motor_load.start();
387             worker_motor_load.setPriority(5);
388
389             SenderIntakeManifoldAirTemperature worker_intake_manifold_air_temperature = new SenderIntakeManifoldAirTemperature(mConnectedThread);
390             worker_intake_manifold_air_temperature.start();
391             worker_intake_manifold_air_temperature.setPriority(5);
392
393             SenderAirFlowSpeedMaf worker_air_flow_speed_maf = new SenderAirFlowSpeedMaf(mConnectedThread);
394             worker_air_flow_speed_maf.start();
395             worker_air_flow_speed_maf.setPriority(5);
396
397             SenderInflux worker_influx = new SenderInflux(this, mConnectedThread);
398             worker_influx.start();
399             worker_influx.setPriority(5);
400
```

Figura 19: Creación de Threads

Fuente: Elaboración propia

Como se observa en el fragmento de código anterior, se crean y se ponen en funcionamiento 7 Threads de Java (correspondientes a las 7 variables de las cuales deseamos conocer sus valores) los cuales irán transmitiendo el comando AT correspondiente para obtener los valores de cada variable, por ejemplo, en el caso de las revoluciones por minuto se puede observar en el siguiente fragmento de código, que se transmite la cadena *010C* para preguntar por las revoluciones por minuto cada 5ms:

SenderRpm.java 600 B

```java
package com.systemsandcloud.tfg;

public class SenderRpm extends Thread {
    private ConnectedThread mConnectedThread;

    public SenderRpm(ConnectedThread m) {
        mConnectedThread = m;
    }

    @Override
    public void run() {
        while (true) {

            try {
                if ( mConnectedThread != null) {
                    mConnectedThread.write("010C" + "\r" + "\n");
                    Thread.sleep(5);
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

}

```

Figura 20: Thread rpm

Fuente: Elaboración propia

Se han realizado pruebas empíricas para ajustar el periodo de cada Thread, esto ha permitido no tener una espera elevada de tiempo en los Threads, ya que todos ellos compiten por el mismo medio de transmisión, por ejemplo, para la temperatura del refrigerante se pregunta cada 500ms por el valor, aprovechando que la temperatura es una variable que tiene mucha inercia:

SenderCoolantTemp.java 521 B

```java
package com.systemsandcloud.tfg;

public class SenderCoolantTemp extends Thread {
    private ConnectedThread mConnectedThread;

    public SenderCoolantTemp(ConnectedThread m) {
        mConnectedThread = m;
    }

    @Override
    public void run() {
        while (true) {

            try {
                mConnectedThread.write("0105"+ "\r" + "\n");
                Thread.sleep(500);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

}
```

Figura 21: Thread temperatura del refrigerante

Fuente: Elaboración propia

Un octavo Thread de Java, el cual cada 250ms mira los valores que hay en ese instante de tiempo mostrándose en la aplicación Android, recoge estos datos, crea un *punto* y lo envía a InfluxDB tal como se puede observar en el siguiente fragmento de código:

```
50    public void run() {
51         while (true) {
52
53                              try {
54                                  if ((mConnectedThread != null) && sw_sender_enable.isChecked()) {
55
56                                      LAT = (String) latitude_tv.getText();
57                                      LON = (String) longitude_tv.getText();
58
59                                      TEMP = (String) temp_tv.getText();
60                                      RPM = (String) rpm_tv.getText();
61                                      SPEED = (String) speed_tv.getText();
62                                      VIN = (String) vin_tv.getText();
63                                      intake_manifold_absolute_pressure = (String) intake_manifold_absolute_pressure_tv.getText();
64                                      motor_load=(String) motor_load_tv.getText();
65                                      intake_manifold_air_temperature=(String) intake_manifold_air_temperature_tv.getText();
66                                      air_flow_speed_maf=(String) air_flow_speed_maf_tv.getText();
67
68                                      punto = Point
69                                          .measurement("coches")
70                                          .addTag("vin",VIN)
71                                          .addField("temperatura_refrigerante", Integer.parseInt(TEMP))
72                                          .addField("revoluciones_por_minuto", Integer.parseInt(RPM))
73                                          .addField("velocidad", Integer.parseInt(SPEED))
74                                          .addField("latitud", Float.parseFloat(LAT))
75                                          .addField("longitud", Float.parseFloat(LON))
76                                          .addField("presion_absoluta_colector_admisión", Integer.parseInt(intake_manifold_absolute_pressure))
77                                          .addField("carga_motor", Integer.parseInt(motor_load))
78                                          .addField("temperatura_aire_colector_admisión", Integer.parseInt(intake_manifold_air_temperature))
79                                          .addField("caudal_aire_sensor_maf", Integer.parseInt(air_flow_speed_maf))
80                                          .time(Instant.now(), WritePrecision.NS);
81
82                                      writeApi.writePoint(bucket, org, punto);
83
84                                  }
85                                  Thread.sleep(250);
86                              } catch (Exception e) {
87                                  e.printStackTrace();
88                              }
89                              punto = null;
90         }
91    }
92 }
```

Figura 22: Creación y envío del punto a InfluxDb

Fuente: Elaboración propia

La recogida de datos se establece en 2 fases, en la primera se está continuamente observando cuando hay datos disponibles que leer y cuando hay un mensaje se mira de que tipo es y se envía a la segunda fase.

Primera fase:

```
switch (code) {
    case "4105":
        //Log.v(TAG","he recibido TEMP desde el coche");
        Message readMsg0 = mHandler.obtainMessage(MainActivity.MESSAGE_READ, 0, -1, partialMessage.getBytes());
        readMsg0.sendToTarget();
        break;
    case "410C":
        //Log.v(TAG,"he recibido RPM desde el coche");
        Message readMsg1 = mHandler.obtainMessage(MainActivity.MESSAGE_READ, 1, -1, partialMessage.getBytes());
        readMsg1.sendToTarget();
        break;
    case "410D":
        //Log.v(TAG,"he recibido SPEED desde el coche");
        Message readMsg2 = mHandler.obtainMessage(MainActivity.MESSAGE_READ, 2, -1, partialMessage.getBytes());
        readMsg2.sendToTarget();
        break;


    case "4902":
        //Log.v(TAG,"he recibido VIN desde el coche");
        String res_aux;
        for (int i = 1; i <= 4; i++) {
            res_aux = br.readLine();
            res_aux = res_aux.replaceAll(">", "");
            res_aux = res_aux.replaceAll("\\s+", "");

            partialMessage += res_aux;

            res_aux = "";
        }
        Message readMsg3 = mHandler.obtainMessage(MainActivity.MESSAGE_READ, 3, -1, partialMessage.getBytes());
        readMsg3.sendToTarget();
        break;


    case "410B":
        //Log.v(TAG,"he recibido intake_manifold_absolute_pressure desde el coche");
        Message readMsg4 = mHandler.obtainMessage(MainActivity.MESSAGE_READ, 4, -1, partialMessage.getBytes());
        readMsg4.sendToTarget();
        break;

    case "4104":
        //Log.v(TAG,"he recibido motor_load desde el coche");
        Message readMsg5 = mHandler.obtainMessage(MainActivity.MESSAGE_READ, 5, -1, partialMessage.getBytes());
        readMsg5.sendToTarget();
        break;

    case "410F":
        //Log.v(TAG,"he recibido intake_manifold_air_temperature desde el coche");
        Message readMsg6 = mHandler.obtainMessage(MainActivity.MESSAGE_READ, 6, -1, partialMessage.getBytes());
        readMsg6.sendToTarget();
        break;

    case "4110":
        //Log.v(TAG,"he recibido air_flow_speed_maf desde el coche");
        Message readMsg7 = mHandler.obtainMessage(MainActivity.MESSAGE_READ, 7, -1, partialMessage.getBytes());
        readMsg7.sendToTarget();
        break;
    default:
        Log.v(TAG, "Code not identified: " + code);
        Log.v(TAG, "Message:" + partialMessage);
}
```

Figura 23: Recogida de datos 1

Fuente: Elaboración propia

En la segunda fase es cuando se extrae la información del valor de la variable y se escribe en la aplicación Android:

```
public void handleMessage(Message msg) {

    if (msg.what == MESSAGE_READ) {
        readMessage = new String((byte[]) msg.obj, StandardCharsets.UTF_8);

        switch (msg.arg1) {
            case 0://coolant temp
                aux1 = readMessage.substring(10, 12);
                A = Integer.parseInt(aux1, 16);
                txt_temp_coolant.setText(String.valueOf(Math.abs(A - 40)));
                break;

            case 1://rpm
                aux1 = readMessage.substring(10, 12);
                aux2 = readMessage.substring(12, 14);
                A = Integer.parseInt(aux1, 16);
                B = Integer.parseInt(aux2, 16);
                value = (((256 * A) + B) / 4);
                txt_rpmm.setText(String.valueOf(value));
                progressBar.setProgress(value);
                if (value > 2000) {
                        progressBar.getProgressDrawable().setColorFilter(
                                Color.GREEN, android.graphics.PorterDuff.Mode.SRC_IN);

                    }
                    else{
                        progressBar.getProgressDrawable().setColorFilter(
                                Color.RED, android.graphics.PorterDuff.Mode.SRC_IN);


                    }

                break;



                case 2://speed
                    aux1 = readMessage.substring(10, 12);
                    A = Integer.parseInt(aux1, 16);
                    txt_speedd.setText(String.valueOf(A));
                    break;

                case 3://VIN
                    aux1 = readMessage;
                    String[] chunks = aux1.split(patern_vin);
                    aux2="";
                    for (int i=1;i<6;i++) {
                        byte[] byteArray = hexStringToByteArray(chunks[i].substring(2));
                        String s2 = new String(byteArray, StandardCharsets.UTF_8);
                        aux2+=cleanTextContent(s2);
                    }
                    txt_vinn.setText(aux2.substring(2));
                    break;
                case 4://intake_manifold_absolute_pressure
                    aux1 = readMessage.substring(10, 12);
                    A = Integer.parseInt(aux1, 16);
                    txt_intake_manifold_absolute_pressuree.setText(String.valueOf(A));
                    break;
                case 5://motor_load
                    aux1 = readMessage.substring(10, 12);
                    A = Integer.parseInt(aux1, 16);
                    j = A/2.55;
                    txt_load_motorr.setText(String.valueOf((int)Math.round(j)));
                    break;
```

```
case 6://intake_manifold_air_temperature
    aux1 = readMessage.substring(10, 12);
    A = Integer.parseInt(aux1, 16);
    txt_intake_manifold_air_temperaturee.setText(String.valueOf(Math.abs(A - 40)));
    break;
case 7://air_flow_speed_maf
    aux1 = readMessage.substring(10, 12);
    aux2 = readMessage.substring(12, 14);
    A = Integer.parseInt(aux1, 16);
    B = Integer.parseInt(aux2, 16);
    txt_air_flow_speed_maff.setText(String.valueOf(((256 * A) + B) / 100));
    break;
default:
    Log.v(TAG, "msg.arg1:" + msg.arg1);
```

Figura 24: Recogida de datos 2

Fuente: Elaboración propia

Se ha realizado un análisis de calidad de código con *sonarqube* obteniendo los siguientes resultados:

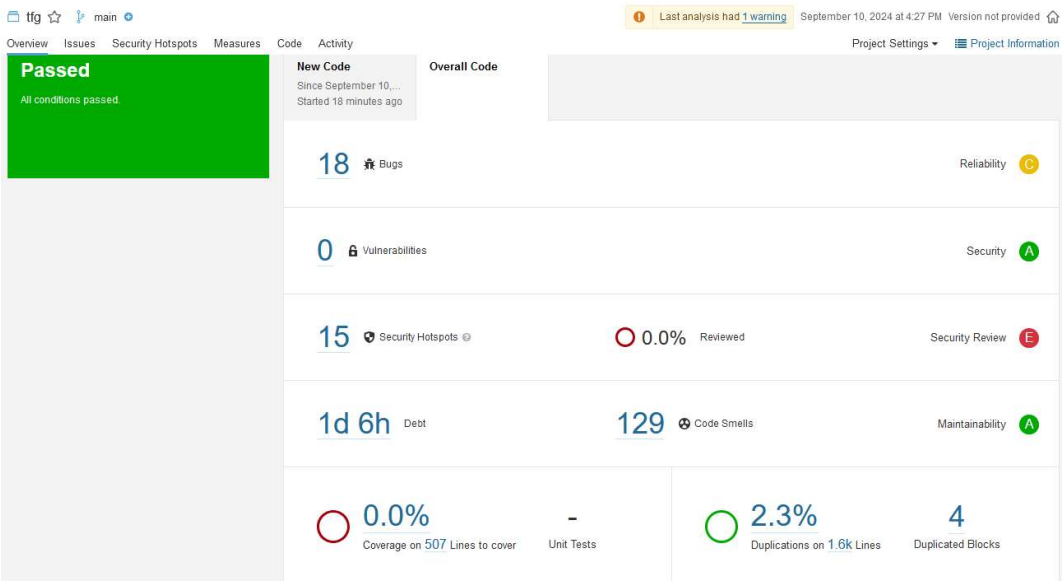


Figura 25: Resultados SonarQube

Fuente: Elaboración propia

Como se puede observar, el código realizado tiene margen de mejora ya que hay 129 smells y 18 bugs.

En el ámbito de sonarqube un smell es *algo que hace que el código sea difícil de entender*, por lo tanto, para que el mantenimiento a largo plazo del código sea factible, se deberían solucionar estos problemas.

4.6. Base de datos para series temporales Influxdb

Para la instalación en la máquina GNU-Linux del software Influxdb se han seguido las instrucciones de instalación del fabricante las cuales se detallan a continuación:

```
curl --silent --location -O \
https://repos.influxdata.com/influxdata-archive.key
echo "943666881a1b8d9b849b74caebf02d3465d6beb716510d86a39f6c8e8dac7515  influxdata-archive.key" \
| sha256sum --check - && cat influxdata-archive.key \
| gpg --dearmor \
| tee /etc/apt/trusted.gpg.d/influxdata-archive.gpg > /dev/null \
&& echo 'deb [signed-by=/etc/apt/trusted.gpg.d/influxdata-archive.gpg] https://repos.influxdata.com/debian stable main' \
| tee /etc/apt/sources.list.d/influxdata.list
# Install influxdb
sudo apt-get update && sudo apt-get install influxdb2
sudo service influxdb start
```

Figura 26: Instrucciones de instalación de Influxdb en Debian

Fuente: https://docs.influxdata.com/influxdb/v2/install/?t=Linux

En la base de datos de series temporales Influxdb se pueden visualizar los datos para un rango temporal, aunque esta operación no sería necesaria en la fase de explotación del producto:

Figura 27: Explorador de datos en Influxdb

Fuente: Elaboración propia

En la misma interfaz web de Influxdb se indica cómo se debe conectar desde código java:

Build with Gradle

```
dependencies {
  implementation "com.influxdb:influxdb-client-java:6.6.0"
}
```

Initialize the Client

```
import com.influxdb.annotations.Measurement;
import com.influxdb.client.InfluxDBClient;
import com.influxdb.client.InfluxDBClientFactory;
import com.influxdb.client.WriteApi;
import com.influxdb.client.domain.WritePrecision;
import com.influxdb.client.write.Point;
import com.influxdb.query.FluxTable;

public class InfluxDB2Example {
  public static void main(final String[] args) {

    // You can generate an API token from the "API Tokens Tab" in the UI
    String token = System.getenv("INFLUX_TOKEN");
    String bucket = "android";
    String org = "micasa";

    InfluxDBClient client = InfluxDBClientFactory.create("http://influxdb.systemsandcloud.com:8086", token.toCharArray());
  }
}
```

Figura 28: Instrucciones para conectar a Influxdb en Java 1

Fuente: Elaboración propia

Write Data

Option 1: Use InfluxDB Line Protocol to write data

```
String data = "mem,host=host1 used_percent=23.43234543";

WriteApiBlocking writeApi = client.getWriteApiBlocking();
writeApi.writeRecord(bucket, org, WritePrecision.NS, data);
```

COPY TO CLIPBOARD

Option 2: Use a Data Point to write data

```
Point point = Point
  .measurement("mem")
  .addTag("host", "host1")
  .addField("used_percent", 23.43234543)
  .time(Instant.now(), WritePrecision.NS);

WriteApiBlocking writeApi = client.getWriteApiBlocking();
writeApi.writePoint(bucket, org, point);
```

Figura 29: Instrucciones para conectar a Influxdb en Java 2

Fuente: Elaboración propia

4.7. Visualización de series temporales con Grafana

La parte de visualización de series temporales se ha realizado con un software gratuito llamado Grafana corriendo en la máquina virtual GNU-Linux.

Para su instalación se ha seguido las instrucciones del fabricante:

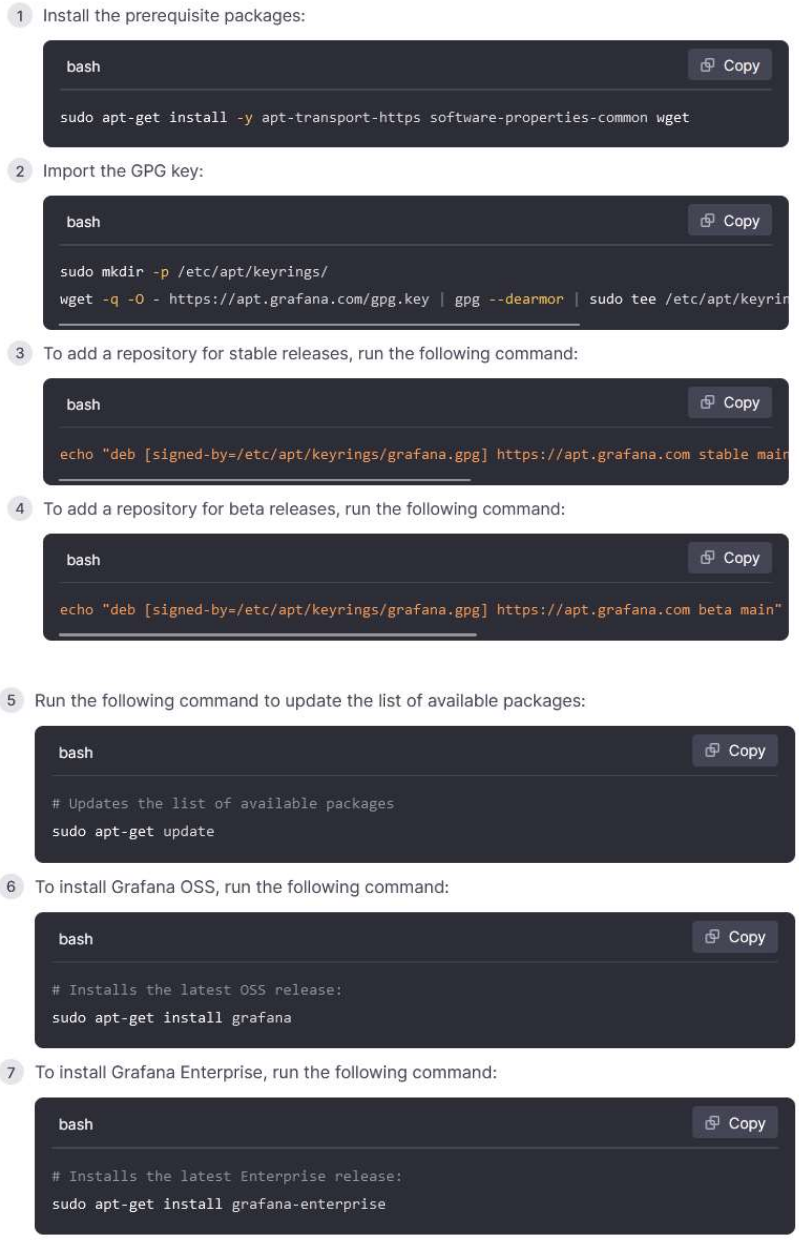


Figura 30: Instrucciones de instalación de Grafana en Debian

Fuente: https://grafana.com/docs/grafana/latest/setup-grafana/installation/debian/

Nada más entrar a Grafana la primera acción que deberemos realizar es seleccionar el VIN del vehículo (número de identificación del vehículo) y seguidamente seleccionar un rango de tiempo para poder visualizar las señales en ese espacio de tiempo:

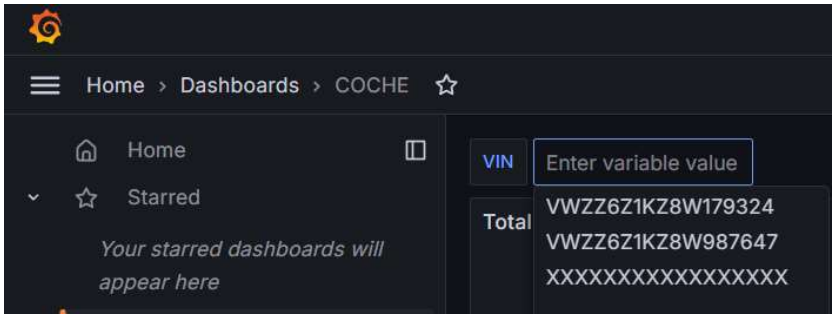


Figura 31: Selección del VIN en Grafana

Fuente: Elaboración propia

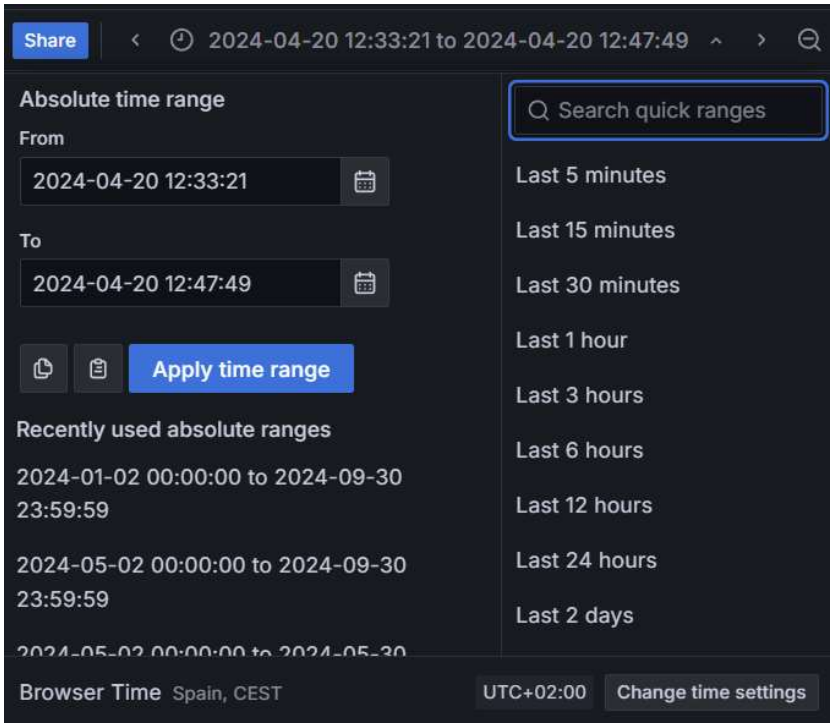


Figura 32: Selección del rango temporal a visualizar en Grafana

Fuente: Elaboración propia

Tal como se ha comentado anteriormente, las señales que vamos a visualizar son:

Señales para visualizar	
Azul claro	Temperatura del líquido refrigerante (ºC)
Rosa	Flujo aire sensor MAF (gr/sec)
Naranja	Presión en el colector de admisión (kpA)
Azul oscuro	Temperatura aire en el colector de admisión (ºC)
Rojo	Carga del motor (%)
Amarillo	Revoluciones por minuto (rpm)
Verde	Velocidad del vehículo (km/h)

Se puede observar también en la leyenda de la gráfica que contiene las 7 señales conjuntamente:

Figura 33: Colores de las señales en Grafana

Fuente: Elaboración propia

La primera gráfica que vemos son las 7 señales juntas (velocidad, revoluciones por minuto, temperatura del refrigerante, presión absoluta del colector de admisión, carga del motor, temperatura del aire en el colector de admisión y caudal de aire medido en el sensor MAF):

Figura 34: Visualización de las 7 señales conjuntas en Grafana

Fuente: Elaboración propia

En esta misma gráfica se pueden seleccionar las señales que queremos visualizar, por ejemplo, si solo nos interesa visualizar la velocidad, las revoluciones por minuto y la presión en el colector de admisión, haremos clic en esas 3 señales de la leyenda:

Figura 35: Visualización de 3 señales conjuntas en Grafana

Fuente: Elaboración propia

Además de la gráfica que contiene todas las señales, se han implementado gráficas individuales por cada una de las señales:

Figura 36: Gráficas individuales 1

Fuente: Elaboración propia

Figura 37: Gráficas individuales 2

Fuente: Elaboración propia

En la figura anterior que muestra el caudal de aire en el sensor MAF, se observa en rojo el umbral establecido por el cual se enviarán alarmas cuando se supere.

Figura 38: Gráficas individuales 3

Fuente: Elaboración propia

Al pasar el ratón por encima de la gráfica automáticamente nos muestra los valores de todas las señales en ese preciso instante de tiempo, así como también el punto del mapa donde estaba el automóvil en ese momento:

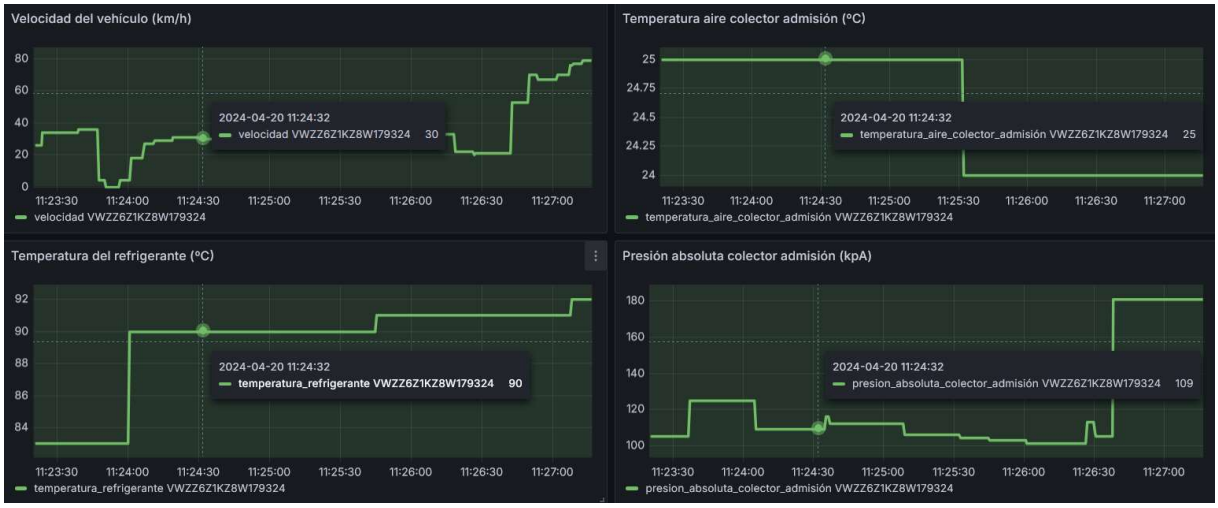


Figura 39: Muestra valores instantáneos en todas las gráficas simultáneamente

Fuente: Elaboración propia

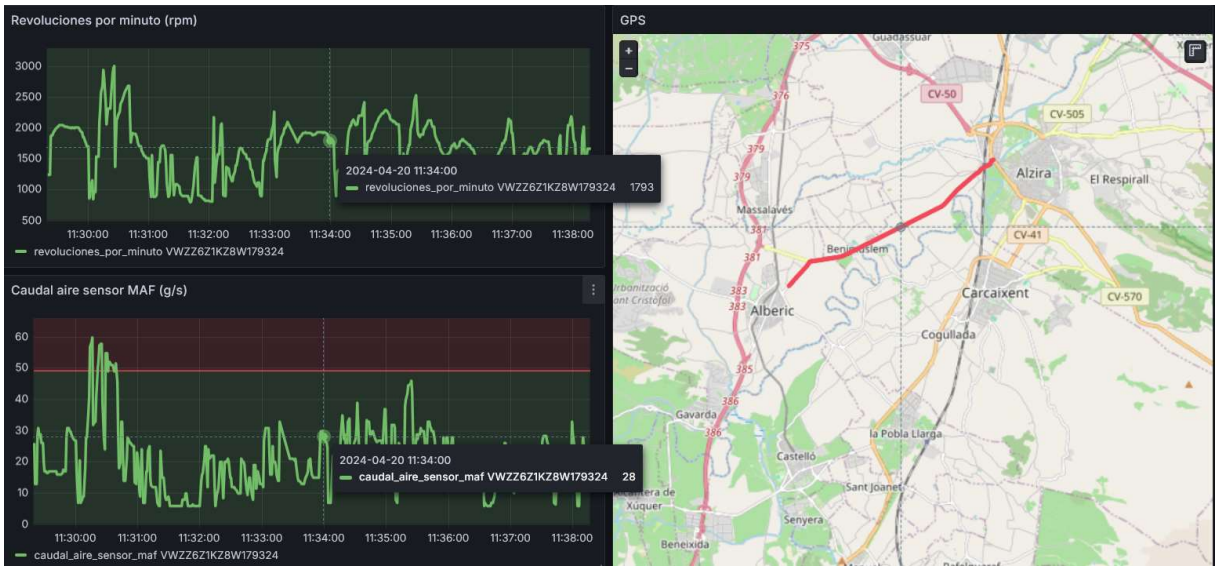


Figura 40: Puntero del ratón mostrando los valores puntuales conjuntamente en todas las gráficas

Fuente: Elaboración propia

Podemos observar en la parte GPS del dashboard de Grafana el recorrido realizado en el espacio de tiempo seleccionado:

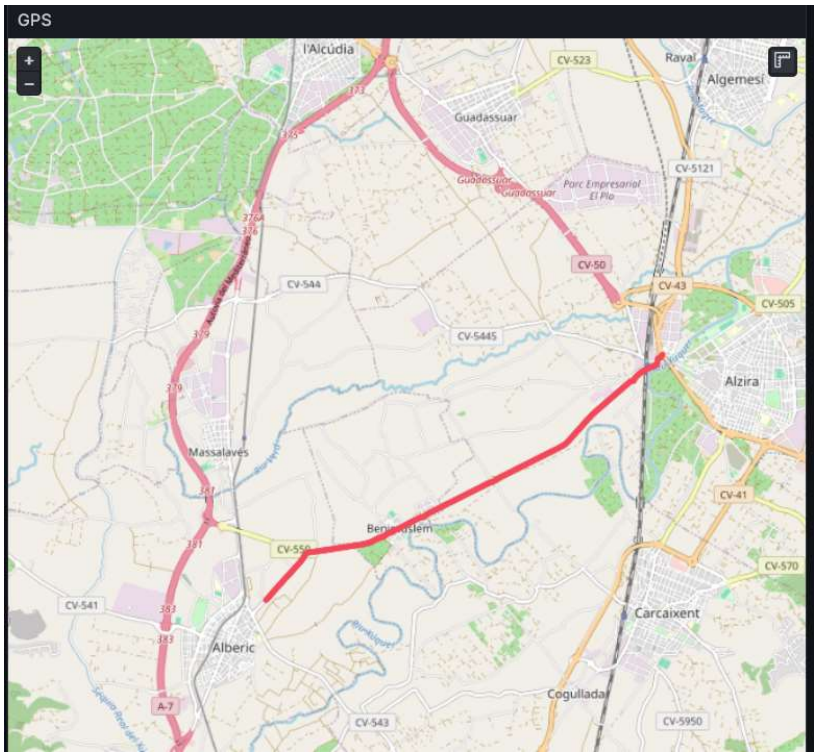


Figura 41: Mapa de Grafana mostrando recorrido realizado en rojo

Fuente: Elaboración propia

4.8. Alarmas

Se han establecido alarmas en Grafana, cuando el valor de la gráfica de Grafana supere un umbral, entonces se envía a un canal de Telegram una notificación, por ejemplo, cuando la temperatura del refrigerante supera el umbral establecido:

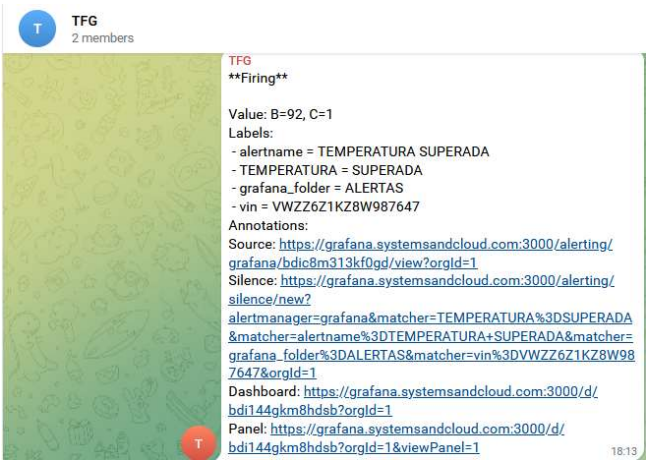


Figura 42: Notificación de Alarma activa en Telegram

Fuente: Elaboración propia

Y cuando baja la temperatura se resuelve la alarma:

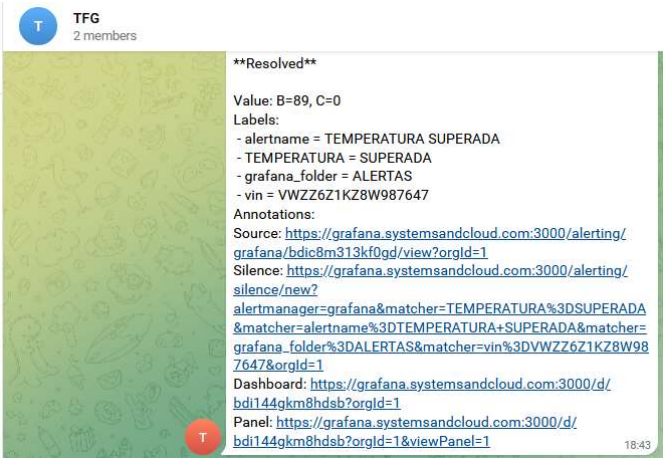


Figura 43: Notificación de Alarma resuelta en Telegram

Fuente: Elaboración propia

4.9. Servidor GNU-Linux

Se ha instalado Debian 11 en una máquina virtual mediante el software de virtualización Virtualbox:

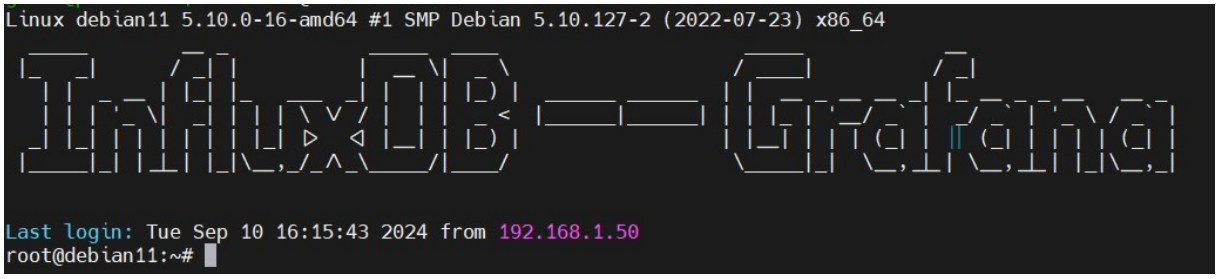


Figura 44: Debian

Fuente: Elaboración propia

Se ha dado de alta un registro DNS wilcard para que *.systemsandcloud.com apunte a la ip pública del router del hogar.

Se ha redirigido (NAT) el puerto TCP 8086 del router del hogar hacia la ip de la máquina virtual para que el tráfico que envía el teléfono móvil llegue hasta Influxdb.

5. Conclusiones y trabajo futuro

5.1. Conclusiones del trabajo

El objetivo general del presente trabajo el cual es conseguir una infraestructura gratuita, la cual permita monitorizar la velocidad, revoluciones por minuto, temperatura del líquido refrigerante, carga del motor, caudal de aire en el sensor MAF, presión absoluta del colector de admisión y temperatura del aire en el colector de admisión de un automóvil en tiempo real y transmitir esos datos a una base de datos externa, para así poder realizar un análisis de los datos recogidos en cualquier momento mediante un software de visualización de métricas, se ha conseguido realizar correctamente.

En cuanto a los objetivos específicos los cuales se nombran a continuación:

- ✓ Implementar una aplicación Android que permita visualizar en el teléfono móvil los datos extraídos del automóvil. Esta aplicación se conectará mediante Bluetooth a la interfaz ELM327 y preguntará constantemente al automóvil por el valor de las variables, correrá en un teléfono móvil con sistema operativo Android.
- ✓ A los datos obtenidos del automóvil, se le añadirán los datos de posición GPS obtenidos del teléfono y todo ello se enviará al servidor de métricas mediante la tecnología 4G del teléfono.
- ✓ Instalar una máquina virtual GNU-Linux mediante el software gratuito de virtualización Virtualbox
- ✓ Instalar el software de servidor de bases de datos para series temporales Influxdb.
- ✓ Instalar el software de visualización de métricas llamado Grafana.

Todos ellos se han conseguido realizar correctamente, además, aprovechando funcionalidades de Grafana, se han implementado alarmas que se notifican en un canal de Telegram cuando un valor de determinada variable supera un valor previamente establecido.

También se han investigado los protocolos de comunicaciones en automóviles, así como la interfaz que nos ofrecen para extraer datos.

Se ha demostrado que se puede tener en el domicilio personal una infraestructura capaz de monitorizar el vehículo a coste casi cero, así como también explotar los datos en cualquier momento posterior.

5.2. Líneas de trabajo futuro

En un futuro en lugar de realizar “pregunta-respuesta” obteniendo los PID correspondientes a los valores de las variables en ese instante de tiempo, sería interesante conectarse directamente al bus can mediante una placa electrónica MCP2515 por ejemplo, y en lugar de “pregunta-respuesta” obtener los datos directamente del bus can para luego ser enviados al smartphone y de ahí a la base de datos de series temporales por internet.

Referencias bibliográficas

Boullosa. F. (2015). APPQUEOLOGY 2.0. [Trabajo fin de grado, Universitat de Barcelona]. Dipòsit digital de la Universitat de Barcelona. https://diposit.ub.edu/dspace/bitstream/2445/66890/2/memoria.pdf

CiA. can-cia. (s. f.). *Recuperado el 23 de Julio de 2024* de https://www.can-cia.org/can-knowledge/

CSS Electronics. OBD2 PID Overview. *Recuperado el 6 de Julio de 2024* de https://www.csselectronics.com/pages/obd2-pid-table-on-board-diagnostics-j1979

DB-Engines. (2024). DB-Engines Ranking of Time Series DBMS. *Recuperado el 10 de Septiembre de 2024* de https://db-engines.com/en/ranking/time+series+dbms

Developers Android. (2024a). Arquitectura de la plataforma. *Recuperado el 2 de Agosto de 2024 de* https://developer.android.com/guide/platform?hl=es-419

Developers Android. (2024b). Tiempo de ejecución de Android y Dalvik. *Recuperado el 5 de Agosto de 2024* de https://source.android.com/docs/core/runtime?hl=es-419

Developers Android. (2024c). Introducción a las actividades. *Recuperado el 5 de Agosto de 2024 de* https://developer.android.com/guide/components/activities/intro-activities?hl=es-419

Developers Android. (2024d). Descripción general de los servicios. *Recuperado el 12 de Agosto de 2024 de* https://developer.android.com/develop/background-work/services?hl=es-419

Developers Android. (2024e). Descripción general de las transmisiones. *Recuperado el 18 de Agosto de 2024 de* *https://developer.android.com/develop/background-work/background-tasks?hl=es-419*

Developers Android. (2024f). Conceptos básicos sobre proveedores de contenido. R*ecuperado el 18 de Agosto de 2024 de* *https://developer.android.com/guide/topics/providers/content-provider-basics?hl=es-419*

Developers Android. (2024g). Ciclo de vida de la actividad. *Recuperado el 9 de Agosto de 2024 de* *https://developer.android.com/guide/components/activities/activity-lifecycle?hl=es-419*

Elm Electronics Inc. (2010). ELM327 OBD to RS232 Interpreter. https://www.elmelectronics.com/DSheets/ELM327DSH.pdf

Garzó. E. (2023). Bases de datos para series temporales: estudio. [Trabajo fin de grado, Universidad de Alcalá]. Biblioteca digital Universidad de Alcalá. http://hdl.handle.net/10017/58730

Ian J. Hawkins. (s. f.). Torque. *Recuperado el 3 de Julio de 2024* de https://torque-bhp.com

Lázaro, F. (2015). Investigación forense de dispositivos móviles Android. RA-MA.

Lozano, J.V. (2017). DriverAnalizer Clasificación de conductores a partir de la información de diagnóstico de un vehículo mediante técnicas de aprendizaje automático. [Trabajo fin de grado, Universidad de Alicante]. Repositorio institucional de la Universidad de Alicante. http://hdl.handle.net/10045/67811

Marín, A. (2016). Aplicación Android para el internet de las cosas. [Trabajo fin de grado, Universidad de Málaga]. Repositorio institucional de la Universidad de Málaga. https://riuma.uma.es/xmlui/bitstream/handle/10630/12733/Adrian%20Marin%20TFG%20%283%29.pdf?sequence=1

McCord, K. (2011). Automotive Diagnostic Systems. https://books.google.es/books?id=kyEtsrPk9ZQC&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false

Nazco, W. (2018). Almacenamiento y visualización de series temporales. [Trabajo fin de grado, Universidad La Laguna]. RIULL.

http://riull.ull.es/xmlui/handle/915/10417

OBD Solutions. (s. f.). OBDLink. *Recuperado el 8 de Julio de 2024* de http://www.obdlink.com

Pacheco, JE. (2011). *Monitoreo de hábitos de manejo por medio de una red CAN automotriz. [Tesis profesional, Universidad de las Américas Puebla]. Colección de tesis digitales.* http://catarina.udlap.mx/u_dl_a/tales/documentos/lmt/pacheco_h_je/

Universidad Tecnológica Nacional. (2009). *Facultad regional de Buenos Aires. Departamento de electrónica.* https://sge.frba.utn.edu.ar/upload/Materias/95-0429/archivos/Cap8_2009_CAN.pdf