Universidad de Valladolid

Escuela de Ingeniería Informática

Trabajo Fin de Grado

Grado en Ingeniería Informática, Mención en Tecnologías de la Información

Adquis ición y Visualización de Parámetros de Funcionamiento de Automóvil provenientes de diversos dispositivos en Tiempo Real

Autor:

D. Rodrigo Alonso Pastor

Tutor:

Dr. César Llamas Bello

Dedicado a mis padres, hermano, amigos y a los que confiaban que esto era posible.

Resumen

A pesar de los avances tecnológicos en el mundo de la informática y la automoción, es habitual encontrar usuarios cuyos automóviles, típicamente de cierta antigüedad, no reportan la información suficiente para poder tener un buen cuidado de este, además de usuarios que simplemente están interesados en conocer todos los detalles del funcionamiento de su vehículo.

Este trabajo de fin de grado implementa una solución que ayuda a los usuarios a sacarle más provecho al automóvil con dos funciones principales. Por un lado, les permite comprobar el correcto funcionamiento de este, a través de comprobación de averías y parámetros en tiempo real. Por otro lado, permite la grabación de viajes, es decir, grabar en segundo plano en todo momento los parámetros de funcionamiento del vehículo, posición GPS y otras métricas, función que puede ser útil en diversas situaciones.

Los ámbitos tecnológicos de este proyecto incluyen la investigación y estudio del funcionamiento del procotolo OBD2 para la extracción de parámetros, desarrollo de programas automatizados en Python e integración con bases de datos temporales (TSDB), despliegue de plataforma de visualización de tableros e instalación y gestión de sistemas empotrados.

Abstract

In spite of technological advances in fast-evolving areas such as automotive and technology, it is reasonably common to find users whose automobiles, which tipically are of a certain age, do not report sufficient information to maintain them properly, in addition to users who are simply interested in knowing all the details of the operation of their vehicle.

This End-Of-Degree project implements a solution to help users get the most out of their vehicles with two main functions. On the one hand, it enables the users to check the correct operation of their cars, through fault codes and real-time monitoring of metrics. On the other hand, this project lets the users record trips, i.e. record the vehicle's operating parameters in the background at all times, as well as GPS location and other metrics that may be of interest to the user.

The technological aspects adressed by this project include research and study of the OBD2 protocol for parameter extraction, development of automated programs in Python and their integration with Time-Series Databases, deployment of a dashboard visualization platform, and installation and managment of embedded systems.

Contenidos

1 Introducción **1**
1.1 Motivación . . . 1
1.2 Objetivos y Etapas . . . 1
1.2.1 Objetivos . . . 1
1.2.2 Etapas del proyecto . . . 2
1.3 Recursos utilizados . . . 3
1.4 Caso de negocio . . . 3
1.4.1 Presupuesto . . . 3
1.4.2 Impacto social . . . 3
1.5 Estructura del documento . . . 4

2 Estudio de Antecedentes **5**
2.1 Aplicaciones móviles . . . 5
2.2 Máquinas de diagnosis . . . 7
2.3 Interfaz de Usuario . . . 8
2.3.1 Netdata . . . 8
2.3.2 Perses . . . 9
2.3.3 InfluxDB y Prometheus . . . 10
2.4 Conclusiones del capítulo . . . 10

3 Metodología **13**
3.1 Procedimiento . . . 13
3.2 Planificación del Proyecto . . . 13
3.2.1 Diseño de Tareas . . . 13
3.2.2 Asignación de Recursos . . . 18
3.2.3 Recursos económicos . . . 20
3.3 Tecnologías empleadas . . . 21
3.3.1 Lenguajes de programación . . . 21
3.3.2 Herramientas de Presentación e Interfaz de Usuario . . . 22
3.3.3 Herramientas y Tecnología Software . . . 27
3.3.4 Herramientas y Tecnología Hardware . . . 29

4 Análisis **31**
4.1 Identificación de Usuarios . . . 31
4.2 Paquetes . . . 31

4.3 Requisitos . . . 32
4.3.1 Requisitos funcionales . . . 34
4.3.2 Requisitos no funcionales . . . 34
4.4 Diagramas de definición de bloques . . . 35
4.5 Casos de uso . . . 36
4.5.1 Diagrama de casos de uso . . . 37
4.5.2 Especificación de casos de uso . . . 38

5 Diseño **47**
5.1 Arquitectura . . . 47
5.1.1 Arquitectura física . . . 47
5.1.2 Arquitectura lógica . . . 48
5.2 Diagrama de actividad . . . 49

6 Implementación **51**
6.1 Captura y Procesamiento de Datos . . . 51
6.1.1 Introducción . . . 51
6.1.2 Primera iteración . . . 51
6.1.3 Segunda iteración . . . 53
6.1.4 Resolución de latencia . . . 55
6.1.5 Última iteración . . . 57
6.2 Presentación e Interfaz de Usuario . . . 60
6.2.1 Introducción . . . 60
6.2.2 Primera iteración . . . 60
6.2.3 Segunda iteración . . . 62
6.2.4 Geolocalización en tiempo real . . . 65
6.2.5 Última iteración . . . 68
6.3 Integración de los módulos . . . 72
6.3.1 Introducción . . . 72
6.3.2 Sistema Operativo . . . 73
6.3.3 Instalación de los módulos . . . 76
6.3.4 Tareas de automatización . . . 77

7 Pruebas **79**
7.1 Pruebas realizadas . . . 79
7.2 Consideraciones . . . 79
7.3 Batería de pruebas . . . 79

8 Conclusiones **83**
8.1 Objetivos cumplidos . . . 83
8.2 Características del sistema final . . . 83
8.3 Conclusiones extraídas . . . 84
8.4 Líneas de trabajo futuro . . . 84

Apéndices **89**

A Manual de instalación **91**

A.1 Instalación rápida 91

A.2 Instalación personalizada 92

A.2.1 Sistema Operativo 93

A.2.2 Servicios Python y gestión de dependencias 93

A.2.3 Prometheus-Server 93

A.2.4 Grafana 94

A.2.5 Automatización 94

A.3 Primer inicio y Guía de uso 94

A.3.1 Añadir fuentes de datos 94

A.3.2 Creación de tablero 96

Lista de Figuras

2.1 Listado de Google Play de Torque. 6
2.2 Capturas de pantalla de funcionamiento de aplicación Torque. 6
2.3 Interfaz de tablero de Netdata. 9
2.4 Interfaz de tablero de Perses. 10

3.1 Red de actividades del Plan de Proyecto. 15
3.2 Diagrama de Flujo de Producto del Proyecto. 17
3.3 Detalle de tareas y Diagrama de Gantt del Proyecto. 18
3.4 Interfaz de terminal Python. 22
3.5 Arquitectura de Prometheus y sus componentes [8]. 23
3.6 Vista de menú principal de Grafana. 24
3.7 Vista de tablero de Grafana. 26
3.8 Configuración de panel de Grafana. 26
3.9 Vista de panel D3-Gauge de Grafana. 27
3.10 Vista de editor de Visual Studio Code. 28
3.11 Disposición de los pines del puerto OBD2 tipo A. 30

4.1 Diagrama de paquetes de AutoSampler. 32
4.2 Diagrama de requisitos de AutoSampler. 33
4.3 Diagrama de definición de bloques de la estructura de dispositivos de AutoSampler. 35
4.4 Diagrama de definición de bloques de la estructura de dispositivos de AutoSampler. 36
4.5 Diagrama de casos de uso de AutoSampler. 37

5.1 Diagrama de Despliegue del Proyecto. 48
5.2 Diagrama de Despliegue del Proyecto. 50

6.1 Respuesta del vehículo al comando enviado. 52
6.2 Salida del script del Código 6.1. 53
6.3 Vista del tablero de la primera iteración del módulo de Presentación e Interfaz de Usuario. 62
6.4 Vista de Prometheus-Server de la segunda iteración del módulo de Presentación e Interfaz de Usuario. 63
6.5 Configuración de dirección de Prometheus-Server en Grafana. 63
6.6 Configuración del intervalo de 'scraping' en Grafana. 64
6.7 Configuración de parámetro en Grafana. 64
6.8 Tablero de la segunda iteración del módulo de Presentación e Interfaz de Usuario. 65
6.9 Creación de panel de averías en Grafana. 70
6.10 Primera sugerencia de presentación del módulo de Presentación e Interfaz de Usuario. . . 71

6.11 Segunda sugerencia de presentación del módulo de Presentación e Interfaz de Usuario. . . 72
6.12 Vista de averías del módulo de Presentación e Interfaz de Usuario. 72
6.13 Configuración de adaptadores de red en Raspberry Pi (DietPi). 74
6.14 Configuración de interfaces WiFi en Raspberry Pi (DietPi). 74
6.15 Instalación de módulo de punto de acceso en Raspberry Pi (DietPi). 75
6.16 Configuración de inicio automático en Raspberry Pi (DietPi). 76

A.1 Instalación de Sistema Operativo a través de Raspberry Pi Imager. 91
A.2 Menú de conexiones de Grafana. 95
A.3 Configuración de Dirección de Fuente de Datos en Grafana. 95
A.4 Configuración de Intervalo de Fuente de Datos en Grafana. 95
A.5 Configuración de plugin CSV en Grafana. 96
A.6 Creación de un tablero nuevo en Grafana. 96

Lista de Tablas

3.1 Coste de licencias Software del Proyecto. 20
3.2 Coste de componentes físicos del Proyecto. 20
3.3 Estimación de costes indirectos del Proyecto. 20
3.4 Estimación de costes laborales del Proyecto. 21
3.5 Costes totales del Proyecto. 21

4.1 Caso de Uso 1. Acceso al Sistema 38
4.2 Caso de Uso 2. Crear Tablero 39
4.3 Caso de Uso 3. Consultar Tablero 39
4.4 Caso de Uso 4. Crear Panel 40
4.5 Caso de Uso 5. Consultar Panel 40
4.6 Caso de Uso 6. Modificar Aspecto de Panel 41
4.7 Caso de Uso 7. Modificar Parámetro 41
4.8 Caso de Uso 8. Eliminar Panel 42
4.9 Caso de Uso 9. Eliminar Panel 42
4.10 Caso de Uso 10. Eliminar Tablero 43
4.11 Caso de Uso 11. Añadir una fuente de datos 43
4.12 Caso de Uso 12. Consultar la configuración 43
4.13 Caso de Uso 13. Determinar la dirección de una fuente de datos 44
4.14 Caso de Uso 14. Modificar la frecuencia de actualización 44
4.15 Caso de Uso 15. Modificar el rango de visión 45

6.1 Pruebas de rendimiento en distintos vehículos 56

7.1 Prueba 1 de funcionamiento general del sistema. 80
7.2 Prueba 2 de funcionamiento general del sistema. 80
7.3 Prueba 3 de funcionamiento general del sistema. 80
7.4 Prueba 1 de interacción entre módulos. 80
7.5 Prueba 1 de Captura y Procesamiento de Datos. 81
7.6 Prueba 2 de Captura y Procesamiento de Datos. 81
7.7 Prueba 4 de funcionamiento general del sistema. 81
7.8 Prueba 1 de Presentación e Interfaz de Usuario. 81
7.9 Prueba 2 de Presentación e Interfaz de Usuario. 82
7.10 Prueba 3 de Presentación e Interfaz de Usuario. 82

Lista de Ejemplos de Código

6.1 Ejemplo de funcionamiento de la primera iteración del Módulo de Captura y Procesamiento de Datos . . . 52
6.2 Ejemplo de funcionamiento de la segunda iteración del Módulo de Captura y Procesamiento de Datos . . . 54
6.3 Optimización de métricas de la última iteración del Módulo de Captura y Procesamiento de Datos . . . 58
6.4 Versión reducida de métricas de la última iteración del Módulo de Captura y Procesamiento de Datos . . . 58
6.5 Recogida de averías en la última iteración del Módulo de Captura y Procesamiento de Datos 59
6.6 Proceso de Instalación de Grafana para arquitecturas ARM . . . 60
6.7 Configuración de clientes para Prometheus-Server. . . . 63
6.8 Configuración de intervalo de 'scraping' para Prometheus-Server. . . . 64
6.9 Configuración para habilitar JavaScript en Grafana. . . . 65
6.10 Código de geolocalización via HTML en Grafana. . . . 66
6.11 Código de servidor Flask para geolocalización. . . . 67
6.12 Instalación de plugin CSV para Grafana. . . . 69
6.13 Configuración de plugin CSV para Grafana. . . . 69
6.14 Configuración del mínimo intervalo de actualización de Grafana. . . . 70
6.15 Creación de entorno Python en Conda. . . . 77
6.16 Instalación de módulo GPS. . . . 77
6.17 Configuración de servicio GPS en Raspberry Pi. . . . 77
6.18 Configuración de servicio GPS en Raspberry Pi. . . . 77
6.19 Script de automatización de AutoSampler. . . . 78
A.1 Descarga de Grafana. . . . 92
A.2 Descarga de Prometheus-Server. . . . 92
A.3 Script de instalación rápida de AutoSampler. . . . 92
A.4 Puesta en marcha de servicios Python. . . . 93
A.5 Configuración de Prometheus-Server. . . . 93
A.6 Puesta en marcha de servicios Python. . . . 94
A.7 Puesta en marcha de servicios Python. . . . 94

Capítulo 1

Introducción

El objetivo de este trabajo de fin de grado es el desarrollo de un prototipo funcional de una plataforma de datos sobre el vehículo y la conducción del usuario. Es habitual que los conductores de automóviles quieran conocer más información que la que muestra el propio vehículo, así como registrar y consultar datos sobre viajes pasados.

Este sistema permite a cada usuario monitorizar, registrar y exportar los distintos datos que pueda considerar convenientes. Estos datos incluyen, pero no están limitados a, valores como: distintas temperaturas de operación del vehículo, datos de velocidades y aceleración, voltajes de distintos módulos, estado de pizas electrónicas, así como conocer el estado de avería de diferentes módulos del vehículo.

1.1. Motivación

La motivación principal para la realización de este proyecto es el aprendizaje de conocimientos nuevos o áreas que el alumno no ha visto en profundidad en el grado. La unión de la automoción, sus puertos, protocolos, funcionamiento, etc., con un proyecto de informática supone un gran interés para el alumno, ya que este contenido no se oferta en ninguna asignatura. Por lo tanto, la principal motivación reside en el desafío que presenta el desconocimiento de estas tecnologías.

Además, el desarrollo de sistemas embebidos, y los retos que supone en cuanto a rendimiento o conveniencia también aportan al interés del proyecto.

Por último, existe la motivación de conseguir una herramienta eficaz y comercializable que pueda ser de utilidad para los usuarios potenciales, e incluso para el propio alumno.

1.2. Objetivos y Etapas

1.2.1. Objetivos

Mediante la realización del presente trabajo, se espera conseguir la siguiente lista de objetivos:

- Estudio del funcionamiento del protocolo OBD2 para comunicaciones entre vehículos y sistemas externos.

- Elaboración de un programa que, de manera autosuficiente, sea capaz de obtener y reportar diferentes datos de módulos del vehículo.
- Construcción de un prototipo de sistema empotrado que permita observar en tiempo real, revisar, exportar y analizar los datos reportados en el objetivo anterior.
- Elaborar los tests necesarios para probar el correcto funcionamiento de todas las funcionalidades desarrolladas.

1.2.2. Etapas del proyecto

Las etapas de desarrollo del proyecto son las siguientes:

Primera etapa: Captura de Datos

Se debe desarrollar un componente que sea capaz de comunicarse con el vehículo y solicitar los parámetros que se necesiten, cumpliendo los requisitos que se detallen en la fase de Análisis.

Tareas a realizar:

- Estudio de sistemas equivalentes en el mercado.
- Estudio de funcionamiento de protocolo OBD2.
- Pruebas de comunicación y extracción de parámetros.
- Desarrollo de script automático de extracción y publicación.

Segunda etapa: Interfaz de Usuario

Se debe crear un servicio capaz de comunicarse con el módulo de Captura de Datos, guardar y mostrar los datos extraídos al usuario de forma eficiente.

Tareas a realizar:

- Estudio de sistemas equivalentes en el mercado.
- Búsqueda de plataformas para visualización de datos.
- Creación y configuración de base de datos.
- Creación de tableros e instalación de plugins.

Integración de los módulos

En la última etapa, se combinarán los dos módulos en un sistema empotrado (Raspberry Pi) y se desplegará el módulo de Interfaz.

Tareas a realizar:

- Investigación de opciones disponibles
- Elección y configuración de sistema operativo
- Instalación y comunicación entre los módulos
- Pruebas del sistema final

1.3. Recursos utilizados

Para el desarrollo del proyecto y la construcción del prototipo final se han utilizado los siguientes recursos:

- Ordenador personal con sistemas operativos Windows 10 y GNU/Linux.
- Raspberry Pi 3 Model B.
- Tarjeta de memoria MicroSD.
- Cable microUSB para Raspberry Pi.
- Transformador enchufe toma de mechero a USB.
- Cable ELM327 de OBD2 a USB.
- Módulo GPS USB VK-162

Es importante mencionar que no todos estos recursos son necesarios para el funcionamiento del sistema. Más adelante, en el apartado de implementación se explicará en detalle la flexibilidad del sistema.

1.4. Caso de negocio

1.4.1. Presupuesto

Para aproximar el coste estimado de este proyecto y la fabricación del prototipo se han de tener en cuenta varios tipos de costes. Entre ellos, se encuentran costes físicos, como los materiales del proyecto; costes de personal, la valoración en horas del trabajo empleado, mayormente de desarrollo software. Se desarrollará en profundidad en el capítulo de Metodología.

1.4.2. Impacto social

Las prestaciones que los sistemas informáticos ofrecen hoy en día permite poder mejorar el funcionamiento de muchas estructuras e instituciones, y con ello la vida de los usuarios. Estas mejoras se pueden categorizar en los siguientes aspectos:

Sociales

El proyecto permite al usuario cubrir al usuario interesado una necesidad de información sobre el correcto funcionamiento del vehículo, mejorando indirectamente la calidad de vida de este.

Económicas

Al obtener información detallada de todas las averías del vehículo, entre otros datos, el usuario puede consultarla y así poder evitar averías mayores, las cuales marcaría el propio vehículo a través de avisos o testigos. Usuarios experimentados pueden también realizar informes de eficiencia en la conducción a través de los datos de este sistema.

Tecnológicas

Al haberse planteado de manera abierta y modular, el proyecto sirve como base o plataforma para expansiones, módulos y construcción de distintas funciones. Además, los datos que extrae el sistema pueden servir para futuros análisis y otros proyectos.

1.5. Estructura del documento

El documento se dispone de la siguiente forma:

En el Capítulo 2 se presenta un estudio de antecedentes, análisis de proyectos que comparten similitudes con este, que se han estudiado para este desarrollo.

El Capítulo 3 explica la metodología del desarrollo: planificación, tareas, tecnologías utilizadas, etc.

El Capítulo 4 corresponde al análisis del sistema. Explicará requisitos, casos de uso y elementos SysML de análisis de sistema.

El Capítulo 5 aborda brevemente el Diseño Software del proyecto.

El Capítulo 6 explica detalladamente el proceso de creación del sistema, sus componentes, configuraciones y otros detalles. Es el capítulo más extenso y elaborado.

El Capítulo 7 describe las pruebas realizadas y sus resultados.

Finalmente, en el Capítulo 8 se recogen las conclusiones extraídas del trabajo, así como lineas de trabajo futuras para el desarrollo.

Se incluye también un apéndice correspondiente al manual de instalación y una breve guía de uso.

Capítulo 2

Estudio de Antecedentes

En este capítulo se va a realizar un estudio de los antecedentes, es decir, proyectos similares que ya existen en el mercado y que pueden cumplir casos de uso similares a este. La necesidad de obtener datos adicionales de un vehículo no es nueva, y ya ha sido intentada resolver por varias fuentes.

Aunque ninguna de las opciones que se muestren en este capítulo cumple con los objetivos presentados y con los requisitos y casos de uso que se presentarán en este documento, se analizarán soluciones interesantes que han ayudado a dar forma a parte del proyecto.

2.1. Aplicaciones móviles

Cada vez son más habituales las aplicaciones, que, con un teléfono Android y un receptor OBD2 Bluetooth consiguen transmitir informes del vehículo en tiempo real. Pese a haber muchas que ofrecen prácticamente la misma funcionalidad, este apartado se va a centrar en la más usada y también la más completa.

Torque es una aplicación dedicada a diagnósticos/datos de vehículos. Este sistema permite personalizar varias pantallas con las medidas que el usuario seleccione, así como revisar las averías del vehículo. Aunque pueda parecer que este tipo de sistemas o aplicaciones no interesen al público general, esta aplicación se encuentra en el Top 2 aplicaciones de pago de Google Play Store.

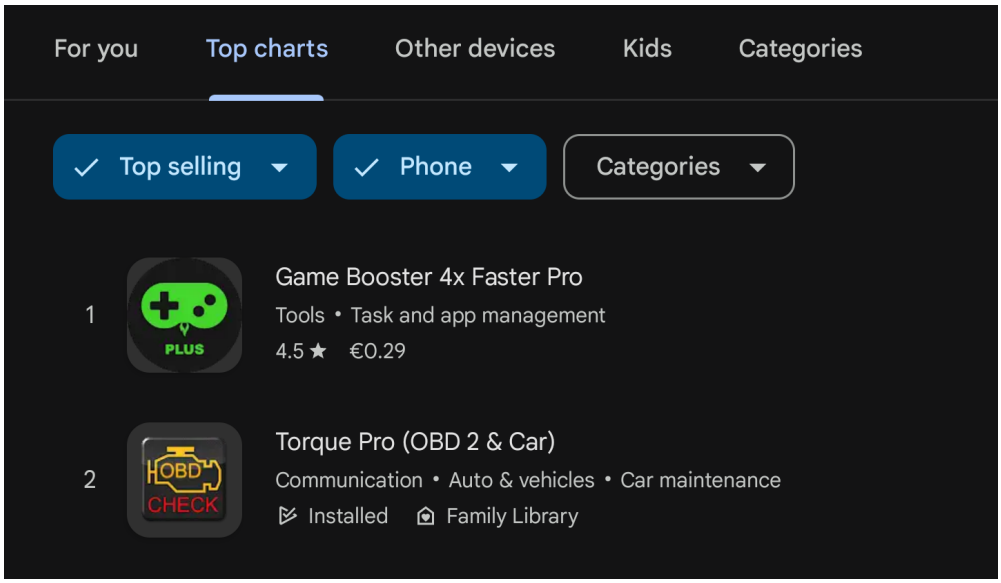


Figura 2.1: Listado de Google Play de Torque.

La aplicación gestiona la conexión Bluetooth al adaptador, y permite al usuario crear "pantallas" con medidas personalizadas. Entre estas se incluyen parámetros del vehículo (velocidades, tempraturas, etc.), así como parámetros del propio teléfono, como el acelerómetro.

Figura 2.2: Capturas de pantalla de funcionamiento de aplicación Torque.

Como se puede ver en la Figura 2.2, la aplicación tiene varias funciones, siendo las tres principales:

- **Métricas en tiempo real:** Permite al usuario crear cuadros para observar parámetros en tiempo real.
- **Vista de averías:** Permite escanear las averías del vehículo conectado y eliminarlas.
- **Vista de gráficos:** Permite, mientras se tiene abierta la pantalla, guardar y visualizar gráficos con parámetros.

Esta solución ofrece mucha funcionalidad a un coste muy reducido, pero tiene varios inconvenientes.

La aplicación carece de de grabación de datos (trata principalmente parámetros en tiempo real), lo que difere totalmente del caso de uso principal de este sistema, que es la grabación de parámetros en "segundo plano", es decir, sin interacción del usuario.

El otro punto negativo es la dependencia absoluta del teléfono móvil. Se necesita tener la aplicación abierta en primer plano para registar cualquier dato. Otra de las ventajas principales de este trabajo es la independencia de terceros dispositivos, pudiendo registrar datos en todo momento, y solo conectando una pantalla, que no tiene por qué ser el teléfono, revisar los datos tomados.

No obstante, algunos aspectos de esta solución, como es la personalización o la flexibilidad de las pantallas, han definido parte de los objetivos y tareas presentados para el módulo de Presentación e Interfaz de Usuario. Se va a buscar una 'interfaz' o experiencia de usuario similar, de manera que sea cómodo y fácil navegar entre los datos, y que el usuario sea capaz de personalizarla para poder ver los parámetros que considere pertinentes.

2.2. Máquinas de diagnosis

Pese a que no guarda mucha relación con este proyecto en lo que a casos de uso se refiere, se van a comentar las máquinas de diagnosis de automóviles. Estas permiten a un usuario experimentado, típicamente usuarios con experiencia o mecánicos revisar todos los parámetros que ofrece un cierto vehículo.

No se considera como un antecedente, pues ni las funciones ni el propósito se equivalen a lo que se presenta en este trabajo; pero si se va a comentar por encima para ver qué detalles o funciones son interesantes y se pueden aprovechar para lo que ataña este trabajo.

Existen varios tipos de máquinas de diagnosis, desde genéricas multimarca con funcionalidad limitada y precio bajo, hasta específicas de cada marca o grupo que permiten mucha más funcionalidad. Por enfocar en un producto en particular, se va a hablar de la herramienta que se ha usado en este proyecto para realizar pruebas de integridad del sistema. Se trata de VCDS, una herramienta de diagnosis que permite, a través de un adaptador OBD2 con una interfaz especial y un ordenador, diagnosticar, programar y visualizar datos de muchos vehículos.

El mayor punto negativo que tiene un sistema como este en lo que se refiere a recolección de datos es la comodidad, pues requiere un ordenador encendido y conectado al vehículo en todo momento.

Elementos a destacar para este proyecto de esta herramienta serían la estabilidad y rapidez del flujo de datos. Esta herramienta ha definido, en parte, decisiones de la arquitectura física del dispositivo.

Al comparar su funcionamiento con la aplicación móvil, se puede ver un decremento en el tiempo de respuesta notable, además de rapidez en la conexión inicial. Es por esto que se ha decidido utilizar una conexión USB para el adaptador del proyecto, pues además de facilitar la automatización de la conexión, permite un flujo más estable y rápido.

2.3. Interfaz de Usuario

Para el módulo de Interfaz de Usuario, se requiere una plataforma que sea capaz de visualizar varias métricas en un tablero, además de una base de datos que sea capaz de gestionar los estrictos requisitos temporales de este sistema. Por ello, se van a listar opciones disponibles que se podrían haber usado, y justificar la decisión final respecto a este módulo.

En primer lugar, se debe hablar brevemente de las plataformas elegidas. Aunque se detallará mucho más extensamente su funcionamiento en los siguientes capítulos, se va a comentar para poder compararlo con las otras opciones.

Para la representación de métricas se ha elegido Grafana. Grafana puede ser considerado el estándar de facto en cuanto a representación de métricas, debido a su popularidad, distintas aplicaciones, y soporte de la comunidad en forma de plugins. En cuanto a la base de datos, se ha elegido Prometheus dada su fácil integración con Grafana y su eficiencia.

A continuación, se van a listar algunos proyectos equivalentes:

2.3.1. Netdata

Netdata[22] es una herramienta de monitorización de métricas que ofrece una solución completa, es decir, tanto la monitorización como la presentación en 'dashboards', lo que facilita la instalación y operación. Cuenta con características interesantes como la automatización de la creación de tableros. Además, ofrece unas estadísticas de consumo más promtedoras que la opción elegida, Prometheus y Grafana, manteniendo mínimo el uso de recursos.

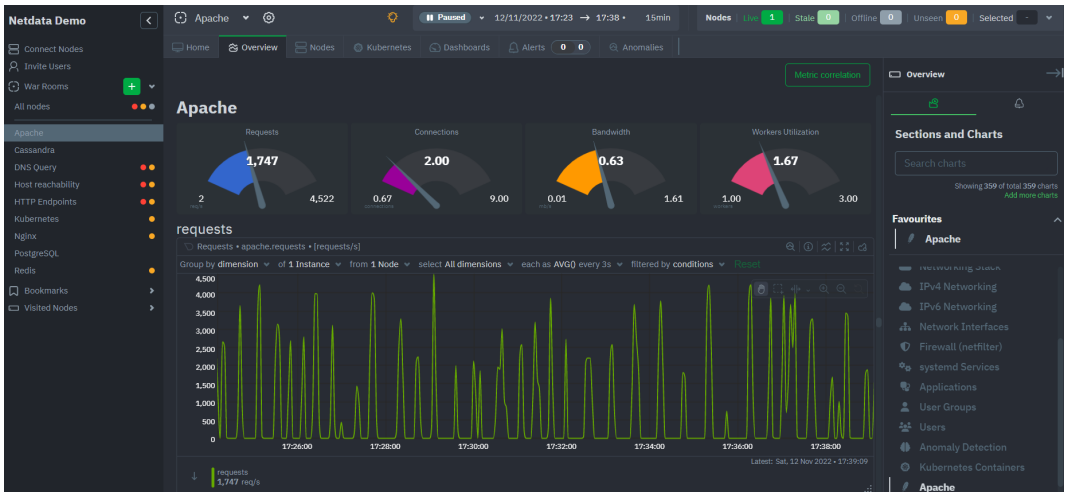


Figura 2.3: Interfaz de tablero de Netdata.

Pese a ser una opción con ciertas cualidades y ventajas, no ha sido elegido por dos razones principales:

- La primera razón es su distanciamiento con el tipo de datos de este sistema. Netdata está principalmente enfocado en métricas de servidores y ordenadores similares, y pese a que se pueda adaptar a través de integraciones, se ha preferido elegir una combinación que permita, de forma sencilla, aceptar más tipos de métricas diferentes.

- La segunda razón es la flexibilidad. La visión e idea original de este proyecto implica la personalización y modularidad tanto de métricas como de tableros. Netdata no ofrece funciones que ayuden a estos términos, a diferencia de la opción elegida.

2.3.2. Perses

Perses[23] es una plataforma de visualización de métricas de la comunidad CoreDash, proyecto de Linux Foundation para intentar definir un estándar de visualización de métricas y 'dashboards'. Entre sus principales características ofrece una integración directa con Prometheus, una interfaz sencilla y un desarrollo de código abierto con posibilidad de crear plugins.

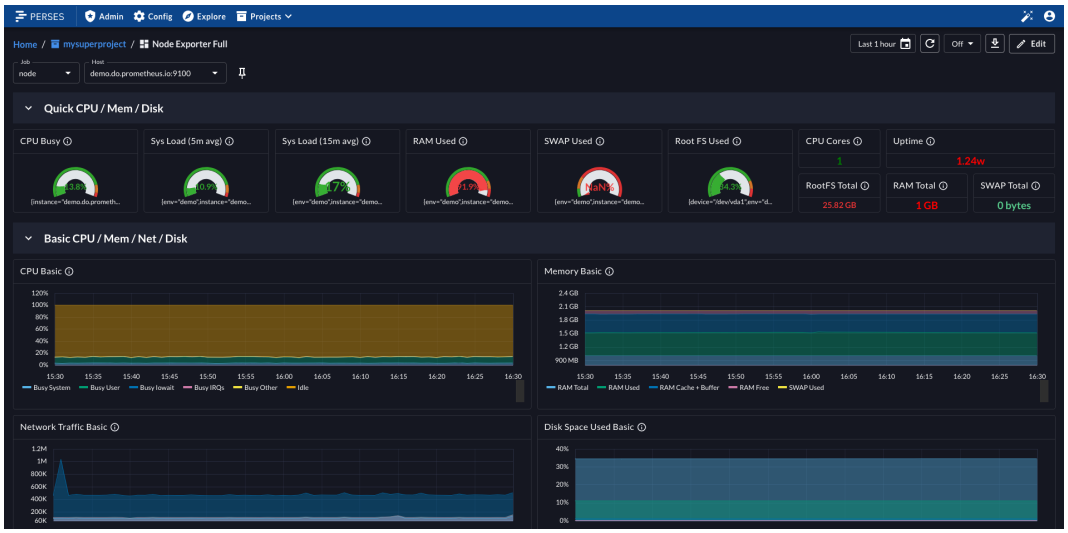


Figura 2.4: Interfaz de tablero de Perses.

Esta propuesta es interesante, y puede que en algunos aspectos se adapte incluso mejor que Grafana para este proyecto, pero no se ha elegido por la falta de soporte actual de plugins. Como se verá más adelante, el proyecto base requiere un plugin de Grafana que no tiene equivalencia en Perses, y las distintas opciones de personalización que se pueden conseguir a través de plugins de Grafana son lo suficientemente importantes como para elegirlo en vez de este.

2.3.3. InfluxDB y Prometheus

Con la plataforma de Interfaz de Usuario elegida, se procede a buscar bases de datos que puedan servir para el proyecto. Desde Grafana se recomienda usar, entre otras, InfluxDB o Prometheus, dada su fácil integración con esta. InfluxDB y Prometheus son dos bases de datos temporales (TSDB) centradas en la eficiencia de los datos, grabación de métricas y logs.

Aunque sirvan casos de uso ligeramente diferentes, ya que Prometheus está más centrado en monitorización y alerta de métricas, mientras que InfluxDB es más hábil gestionando grandes volúmenes de series temporales y logs de eventos; para este proyecto ambas pueden cumplir con los requisitos temporales propuestos.

Por lo que en este caso la decisión es más difícil, pero se ha acabado eligiendo Prometheus por su fácil despliegue y configuración, y por su integración con el lenguaje Python, que ha resultado más sencilla para el alumno.

2.4. Conclusiones del capítulo

En este capítulo se han presentado varias soluciones que permiten recolección de datos de automóviles a otras plataformas y varias propuestas de interfaces de usuario.

Este trabajo intenta, entre otras cosas, resolver los inconvenientes de las soluciones de recolección de datos mostradas. Se intentarán coger los elementos positivos del equipamiento más ‘profesional’,

como es la estabilidad, rapidez y fluidez; combinándolo con elementos notablemente buenos de otras soluciones, como la comodidad de las lecturas y la flexbilidad que supone ser compatible con todo tipo de dispositivos.

Capítulo 3

Metodología

3.1. Procedimiento

Para desarrollar este trabajo de fin de grado, la metodología de desarrollo que se ha elegido es una basada en una planificación previa de los objetivos del proyecto y su realización, pero con la capacidad de modificarlos y reorientar los recursos, ya que, debido al desconocimiento previo de la tecnología a utilizar y sus posibilidades, algunos requisitos se pueden ver ampliados en profundidad o en extensión, aunque se mantengan iguales en líneas generales. Se ha elegido, entonces, un modelo dirigido por plan de cascada, donde se crea una estructura de etapas secuenciales que permiten el inicio de la siguiente una vez finalizan.

En los modelos que se presentarán más adelante, existen tareas paralelas puesto que se podrían realizar a la vez, es decir, son independientes y les precede la misma tarea. No obstante, todas las tareas se han tenido que serializar por optimización, al ser solo una persona trabajando en el proyecto.

Para la implementación, se ha seguido un modelo basado en prototipos, donde se van creando versiones funcionales de un sistema, que evoluciona en cada iteración, hasta llegar a un sistema final. Esto permite actuar con facilidad a la hora de corregir malas implementaciones de un requisito, así como corregir rápidamente problemas que puedan surgir en las pruebas.

3.2. Planificación del Proyecto

3.2.1. Diseño de Tareas

En el capítulo de Introducción, se ha comentado una serie de etapas generales que surgen de los objetivos propuestos para el proyecto. A partir de estas, se debe concretar un conjunto de tareas que definan con propiedad el desarrollo a realizar.

Partiendo entonces de las tres fases iniciales, se pueden definir las siguientes tareas:

- **Estudio de Viabilidad y Antecedentes:** Fase inicial del proyecto, contiene el estudio de antecedentes, preparación tecnológica y documentación. Se define algún objetivo principal.

- **Modelado de alto nivel:** Se modelan las características generales del sistema y se fijan el resto de objetivos.

- **Análisis y requisitos:** Análisis en profundidad del sistema y obtención del listado inicial de requisitos.

- **Prototipado de Captura y Procesamiento de Datos:** Desarrollo, mediante varios prototipos que prueban funcionalidades diferentes, del módulo encargado de obtener y procesar los datos recogidos del vehículo.

- **Prototipado de Presentación e Interfaz de Usuario:** Análogamente al otro módulo, en esta tarea se desarrolla mediante prototipos el módulo de presentación e interfaz de usuario.

- **Integración de los módulos y Prototipado de Sistema Final:** En esta tarea se integran los dos módulos para crear un sistema funcional, y se hacen varios prototipos que se acercan al sistema final.

- **Documentación:** Pese a que cada tarea conlleva una parte de documentación, esta última se refiere a la escritura de esta memoria, manuales de usuario, etc.

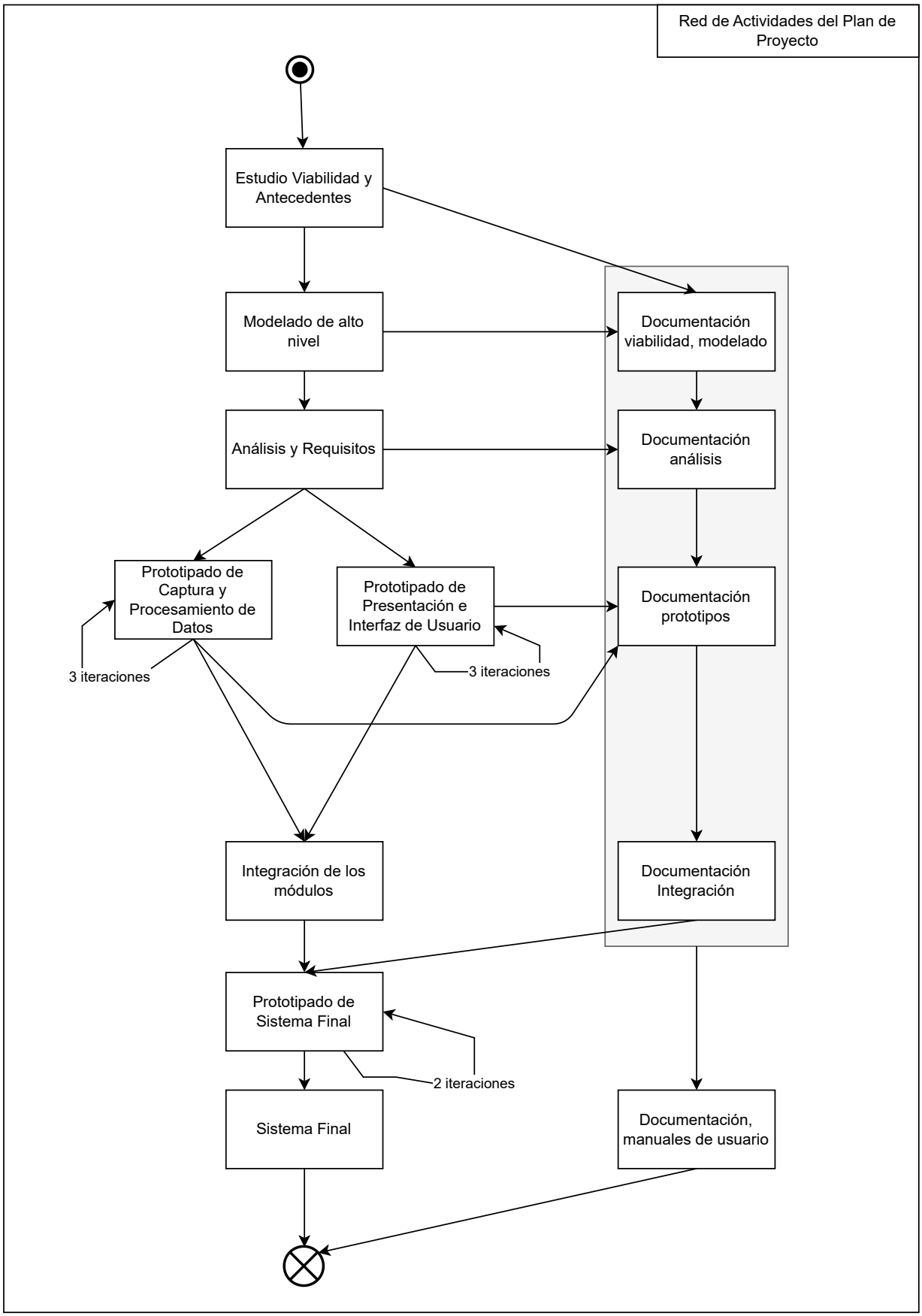


Figura 3.1: Red de actividades del Plan de Proyecto.

En la Figura 3.1 se puede ver la red de actividades de la fase de planificación del proyecto. Como se ha mencionado en el diseño de tareas, hay una fase inicial de estudio, modelado y análisis seguido del desarrollo de los módulos.

Este desarrollo es basado en prototipos, más especificamente, se han estimado tres iteraciones para cada módulo, en interés de aprender del desarrollo mientras se experimenta con la tecnología y sus posibilidades, para, en la tercera iteración, tener un sistema sólido que satisfaga los objetivos y requisitos propuestos. Cada iteración tendrá un enfoque diferente, de forma que se implemente nueva funcionalidad sobre una base estable, como se verá en el capítulo de Implementación.

La fase de desarrollo se ha planificado en paralelo, pues realmente ambos módulos se pueden desarrollar de forma independiente, siendo solo necesario integrarlos cuando se quiera probar el sistema. No obstante, este desarrollo se ha serializado, al ser solo una persona ejecutando el trabajo, de forma que se ha trabajado en el módulo de Captura y Procesamiento de Datos hasta tener una versión usable (aproximadamente la segunda iteración), y luego se ha seguido con el módulo de Presentación e Interfaz de Usuario.

Después de desarrollar los módulos, se procede a integrarlos y prototipar un sistema final que incluya toda la funcionalidad esperada.

En paralelo a todas estas actividades, acompaña una tarea notable de documentación de análisis, desarrollo, integración, etc., que posteriormente se usará para escribir la presente memoria y los manuales de instalación y guía de usuario.

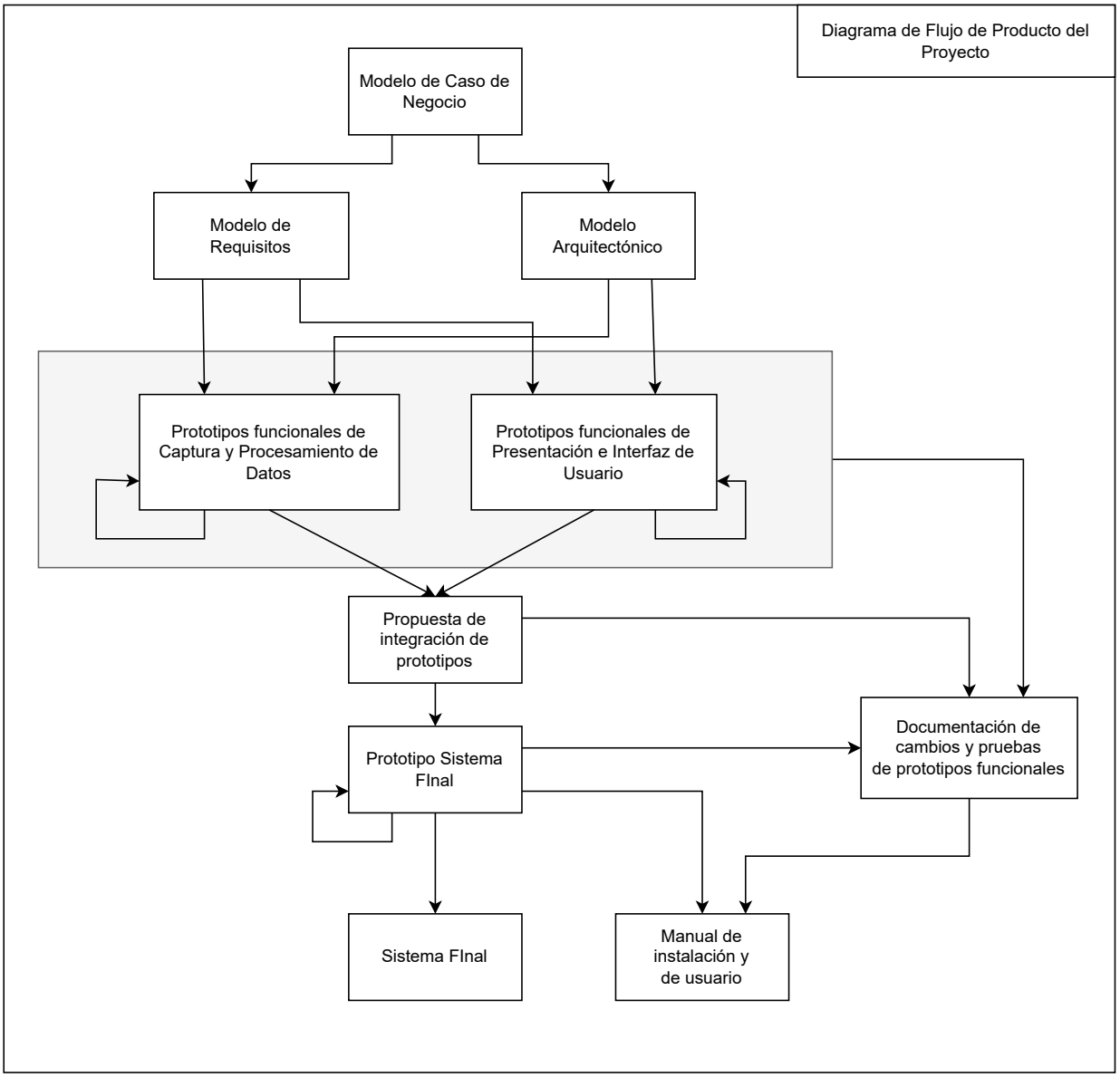
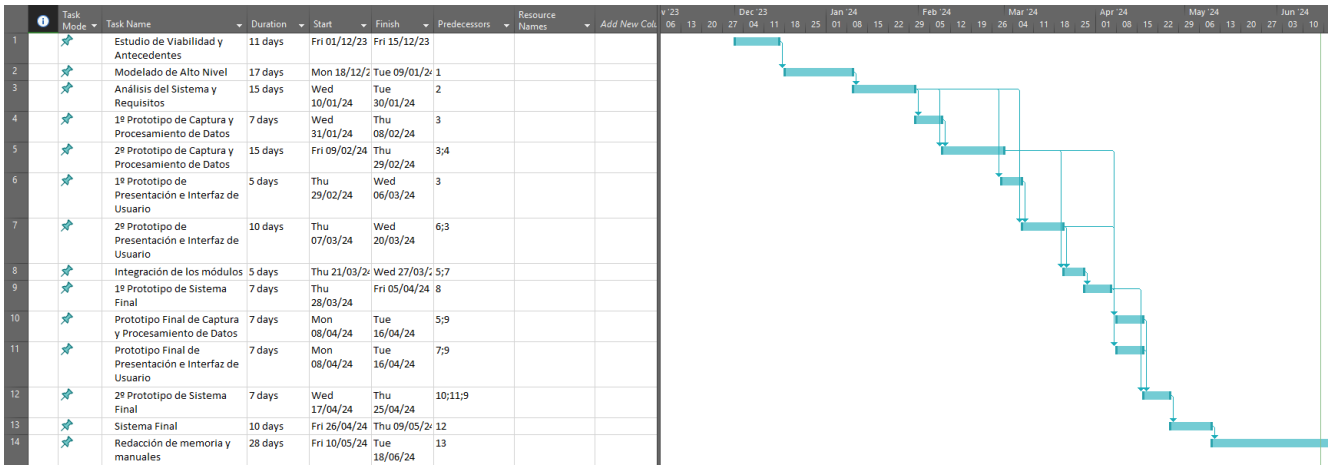


Figura 3.2: Diagrama de Flujo de Producto del Proyecto.

La Figura 3.2 detalla el Diagrama de Flujo de Producto del proyecto. Este detalla, para cada fase explicada anteriormente, los productos que se entregan y que permiten continuar con la siguiente fase.



	Task Mode	Task Name	Duration	Start	Finish	Predecessors	Resource Names
1		Estudio de Viabilidad y Antecedentes	11 days	Fri 01/12/23	Fri 15/12/23		
2		Modelado de Alto Nivel	17 days	Mon 18/12/2	Tue 09/01/24	1	
3		Análisis del Sistema y Requisitos	15 days	Wed 10/01/24	Tue 30/01/24	2	
4		1º Prototipo de Captura y Procesamiento de Datos	7 days	Wed 31/01/24	Thu 08/02/24	3	
5		2º Prototipo de Captura y Procesamiento de Datos	15 days	Fri 09/02/24	Thu 29/02/24	3;4	
6		1º Prototipo de Presentación e Interfaz de Usuario	5 days	Thu 29/02/24	Wed 06/03/24	3	
7		2º Prototipo de Presentación e Interfaz de Usuario	10 days	Thu 07/03/24	Wed 20/03/24	6;3	
8		Integración de los módulos	5 days	Thu 21/03/2	Wed 27/03/2	5;7	
9		1º Prototipo de Sistema Final	7 days	Thu 28/03/24	Fri 05/04/24	8	
10		Prototipo Final de Captura y Procesamiento de Datos	7 days	Mon 08/04/24	Tue 16/04/24	5;9	
11		Prototipo Final de Presentación e Interfaz de Usuario	7 days	Mon 08/04/24	Tue 16/04/24	7;9	
12		2º Prototipo de Sistema Final	7 days	Wed 17/04/24	Thu 25/04/24	10;11;9	
13		Sistema Final	10 days	Fri 26/04/24	Thu 09/05/24	12	
14		Redacción de memoria y manuales	28 days	Fri 10/05/24	Tue 18/06/24	13	

Figura 3.3: Detalle de tareas y Diagrama de Gantt del Proyecto.

En la Figura 3.3 se puede observar la planificación que se ha seguido en el desarrollo del proyecto. Es interesante ver como se ha estructurado el desarrollo del sistema. Se trabaja en cada módulo hasta tener una versión funcional y completa, se integra y se comienza a prototipar el sistema final. Mientras se prototipa el sistema final, se puede concluir el desarrollo de los módulos con la última iteración una vez se ha probado el funcionamiento y su integración. Una vez se concluye el desarrollo, se entrega el sistema final.

3.2.2. Asignación de Recursos

En el primer capítulo se hablaba de recursos materiales utilizados en el sistema entregado. En este apartado, se ampliarán los recursos necesarios para la ejecución de cada tarea del proyecto, omitiendo las fases de viablidad, modelado y análisis, ya que el único componente de coste es el tiempo del futuro Ingeniero realizando el trabajo.

Captura y Procesamiento de Datos Los recursos en esta fase son principalmente herramientas para extraer información del vehículo. Dado que este desarrollo se ejecuta antes que la integración, el sistema se prueba en un ordenador portátil, y se usa también una herramienta de diagnósticos especializada para lanzar pruebas y comprobar los datos.

- Recursos materiales:
  - Ordenador de Sobremesa
  - Ordenador Portátil
  - Ordenador de Diagnóstico
  - Interfaz Vag-Com USB
  - Cable ELM327 de OBD2 a USB
  - Módulo GPS USB VK-162

- Recursos software:
  - Windows 10
  - Windows XP
  - Python 3.8
  - Visual Studio Code
  - VCDS 12.12

Presentación e Interfaz de Usuario

- Recursos materiales:
  - Ordenador de Sobremesa
  - Ordenador Portátil
  - Tablet Android
- Recursos software:
  - Windows 10
  - Ubuntu 20.04 LTS
  - Grafana Enterprise 10.3.3
  - Prometheus 2.45 LTS

Integración de los módulos En esta fase del proyecto, realmente se utilizan todos los materiales listados anteriormente, más algunos necesarios para prototipar el sistema final. No obstante, en la siguiente lista se nombran únicamente los más importantes para la integración.

- Recursos materiales:
  - Ordenador de Sobremesa
  - Raspberry Pi 3 Model B
  - Tablet Android
- Recursos software:
  - Ubuntu 20.04 LTS
  - S.O. Debian DietPi 9.2
  - Raspberry Pi Imager
  - Visual Studio Code

3.2.3. Recursos económicos

Coste de licencias Software

Descripción	**Precio**
Visual Studio Code	0€
Notepad++	0€
Librerías Python	0€
Grafana Enterprise	0€
Prometheus	0€
Total:	**0€**

Tabla 3.1: Coste de licencias Software del Proyecto.

Coste de componentes físicos

Descripción	**Precio**
Raspberry Pi 3B	42,95€
Toma mechero a USB	3,99€
Cable microUSB	1,99€
GPS VK-162	7,63€
Extensor OBD2	4,29€
Total:	**60,85€**

Tabla 3.2: Coste de componentes físicos del Proyecto.

Es necesario mencionar que hay componentes que no se han incluido, por ejemplo la pantalla (Tablet Android), pues son componentes muy flexibles u opcionales, por lo que no se puede estimar un precio.

Estimación de costes indirectos

Pese a que no se puede medir con exactitud los costes indirectos, en la Tabla 3.3 se puede ver una estimación. La electricidad e internet se tienen en cuenta durante el tiempo de realización del proyecto, y el combustible de la realización de diversas pruebas de funcionamiento de módulos y del sistema final.

Descripción	**Precio**
Electricidad	10€/mes
Internet	19,99€/mes
Combustible	30€
Total:	**179,95€**

Tabla 3.3: Estimación de costes indirectos del Proyecto.

Costes laborales

Análogamente, se pueden realizar estimaciones de los costes referentes a la mano de obra del proyecto. Estas estimaciones se han hecho teniendo en cuenta el salario medio por puesto y número de horas de cada uno sobre el total del proyecto.

Descripción	**Precio**
Ingeniero de Software	3.216€
Administrador de Sistemas	1.238€
Total:	**4.454€**

Tabla 3.4: Estimación de costes laborales del Proyecto.

Costes totales

Descripción	**Precio**
Costes Software	0€
Costes Hardware	60,85€
Costes Indirectos	179,95€
Costes Laborales	4.454€
Total:	**4.694,8€**

Tabla 3.5: Costes totales del Proyecto.

3.3. Tecnologías empleadas

3.3.1. Lenguajes de programación

- **Python:**

  Python, creado a finales de los años 80 por Guido van Rossum, es un lenguaje de programación de alto nivel. Entre sus principales características están ser multiparadigma, que permite varios estilos de programación (orientada a objetos, imperativa, etc.); y ser un lenguaje interpretado, es decir, en vez de compilar el código, se va traduciendo dinámicamente, y permite ofrecer al programa un entorno dependiente solo del propio intérprete.

  Todo esto hace que sea un lenguaje extremadamente flexible, y tenga infinidad de aplicaciones, y es precisamente por lo que se ha elegido para la captura y procesamiento de datos del proyecto. Gracias a librerías existentes en Python, se pueden facilitar varios procesos como la comunicación con el puerto OBD o la integración con la base de datos Prometheus, que si se hubiese tratado de otro lenguaje habría dificultado mucho la tarea.

  Además, facilita el componente modular del sistema, permitiendo a otros usuarios integrar módulos de manera muy sencilla.

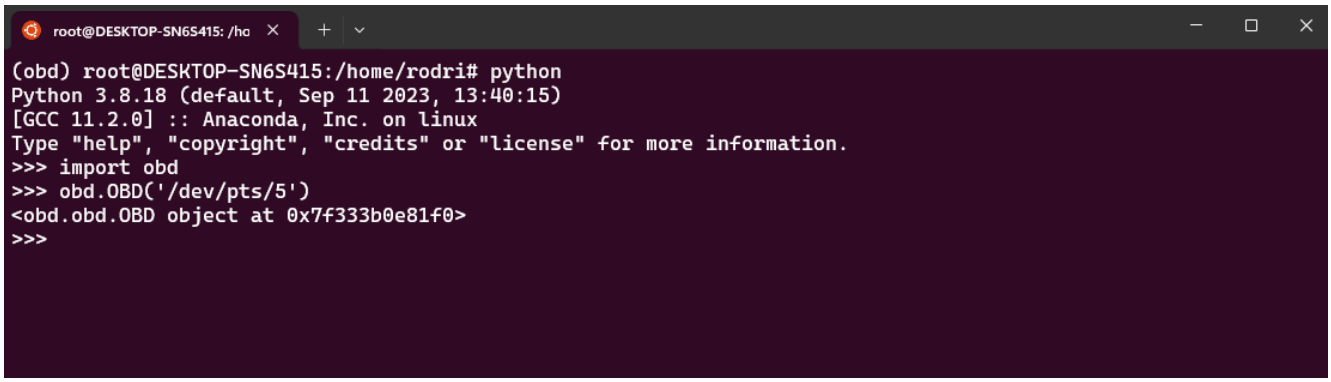


Figura 3.4: Interfaz de terminal Python.

- **Shell:**

  La Shell UNIX, intérprete de comandos de casi todo sistema Linux, además de ofrecer una interfaz de línea de comandos, ofrece la posibilidad de hacer scripts para controlar la ejecución del sistema. En este proyecto, es de vital importancia la automatización de todos los sistemas al momento del inicio, por lo que se han usado scripts que permiten que el usuario no tenga que interaccionar con el sistema.

3.3.2. Herramientas de Presentación e Interfaz de Usuario

En esta sección se va a hablar de las herramientas o plataformas que han dado forma al módulo de Presentación e Interfaz de Usuario. Aunque corresponden a la próxima sección, Herramientas Software, se ha decidido separarlas por su importancia para el proyecto. Se dividirá en dos subsecciones, una para cada herramienta.

Prometheus

Prometheus es un servicio de monitorización y alerta de sistemas de código abierto. Originalmente fundado por SoundCloud en 2012, ha sido adoptado por muchas compañías y organizaciones, hasta convertirse en un proyecto independiente.

Prometheus monitoriza y guarda métricas como series temporales, es decir, la información de cada métrica es guardada junto a la marca de tiempo en la que se recogió, de forma que se crea una cronología con la información de cada parámetro.

Entre las características principales de Prometheus se encuentran:

- Un modelo de datos multidimensional, con series temporales identificadas por un nombre y un par clave/valor.
- Un lenguaje propio de consultas, llamado PromQL, que permite aprovechar la dimensionalidad.
- Recolección de series temporales (métricas) a través de un modelo 'pull' a través de HTTP.

- Multiples modos de gráficos y soporte para “dashboards”.

Antes de continuar, es necesario comentar la importancia de las métricas.

Se puede considerar una métrica como una medida numérica. Aplicado al caso de este proyecto, se considera como métrica todos los parámetros del vehículo y otros módulos, ya que son los objetos de la monitorización. El objetivo de Prometheus (y en última instancia, del proyecto) es monitorizar una serie de métricas en forma de series temporales, grabar los cambios que sufre una métrica a lo largo del tiempo.

Prometheus ofrece una serie de componentes que componen su ecosistema, de los cuales interesan para este trabajo dos: El servidor principal, que “scrapea” y guarda la información en series temporales; y las librerías cliente, que permiten, en muchos lenguajes de programación, definir y exponer métricas a un servidor HTTP para que sean leídas por el servidor central.

Para entender como se coordinan los componentes, se va a hablar de la arquitectura de Prometheus.

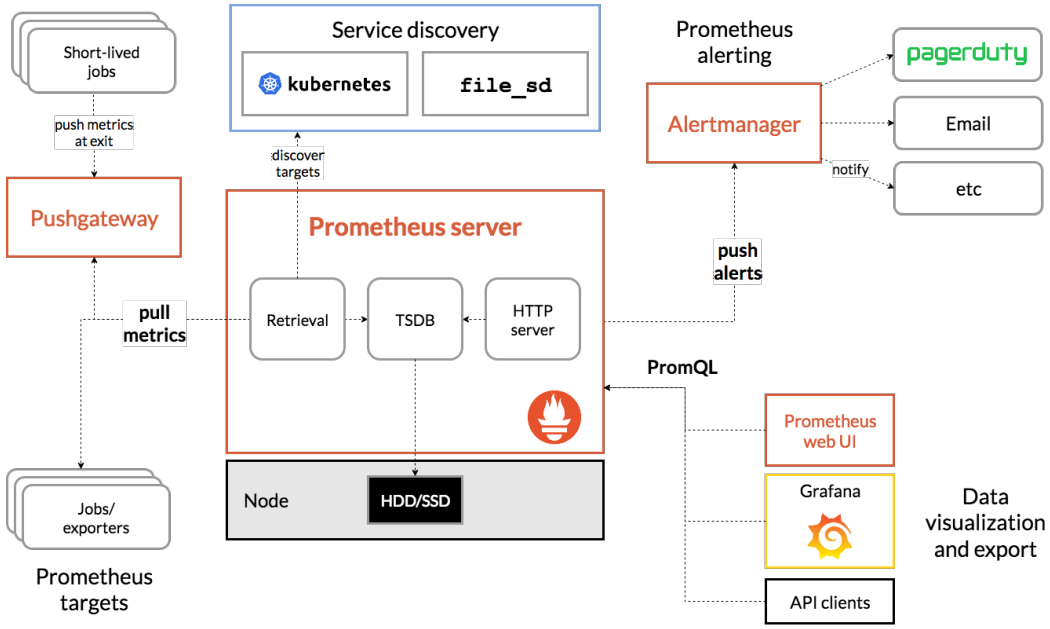


Figura 3.5: Arquitectura de Prometheus y sus componentes [8].

En la Figura 3.5 se puede ver la arquitectura de Prometheus. Los pasos que sigue una métrica para ser grabada siguiendo el modelo “pull” de Prometheus son los siguientes:

- El ‘job/exporter’ que actúa como cliente de Prometheus (una serie de scripts Python en este proyecto) publica las métricas en un ‘endpoints’, un servidor HTTP. Este servidor contiene una lista compuesta de nombres y valores que se van actualizado. En este servidor solo se encuentra la medida actual, es decir, cada cierto tiempo, definido por el funcionamiento del propi cliente, las métricas cambian.

- El Servidor central de Prometheus escanea, o 'scrapea', los 'endpoints' según una tasa de refresco (0.5s/1s para este proyecto), y para cada métrica encontrada en ese servidor HTTP, anota su nombre, valor y la marca de tiempo en la que se ha escaneado. Esta información es guardada en disco y expuesta en otro servidor HTTP para ser accedida posteriormente.
- Una herramienta de visualización de datos, como es Grafana o mismamente la interfaz web que ofrece Prometheus, hace consultas sobre las métricas usando el lenguaje PromQL para acceder a la información.

Por último, comentar que el propósito de uso de Prometheus reside en la eficiencia y la fiabilidad.

Por un lado, el uso exclusivo de series temporales y su modelo de datos le confiere la habilidad de grabar y almacenar métricas en un tiempo razonable, de forma que pueda cumplir con los requisitos temporales de este y muchos proyectos. Usando una base de datos tradicional, sería prácticamente imposible registrar y monitorizar las métricas en un periodo de tiempo aceptable.

Por otro lado, Prometheus está diseñado para ser un sistema fiable. Cada servidor Prometheus es un sistema independiente que no requiere de otros servicios remotos ni red, por lo que conseguirá seguir funcionando en situaciones adversas, lo que lo hace interesante para este proyecto.

Grafana

Grafana es una aplicación web dedicada al análisis y visualización interactiva de datos, de código abierto y multiplataforma. Lanzada por primera vez en 2014 por Torkel Ödegaard como un subproyecto en Orbitz, estaba dedicada exclusivamente al análisis de bases de datos temporales (TSDB) como Prometheus o InfluxDB.

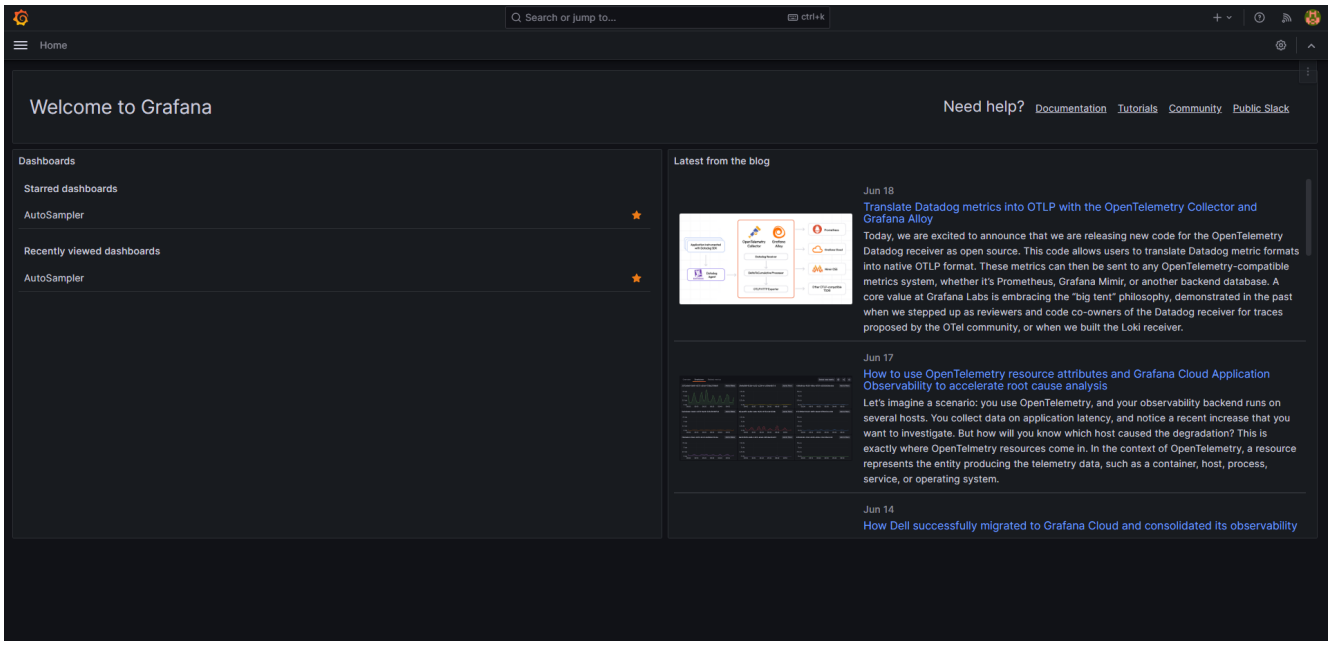


Figura 3.6: Vista de menú principal de Grafana.

Grafana permite consultar, visualizar, explorar y configurar alertas en diferentes métricas,

registros y trazas. Típicamente se utiliza para medir el rendimiento y el estado de servidores, soluciones cloud y otras entidades relacionadas, pero la flexibilidad que ofrece la plataforma permite usos muy variados, como el que se está dando en este proyecto, medir los parámetros de un vehículo.

A continuación, se van a detallar los componentes que forman Grafana, sus características y su utilidad para este proyecto:

- **Servidor central:** El núcleo de la plataforma. Se encarga de la gestión de usuarios, permisos, fuentes de datos, tableros y demás componentes.

  Se ofrecen dos versiones: Grafana Cloud, que permite configurar un servicio Grafana en la nube y visualizar métricas de otros servicios en cloud; y Grafana Enterprise/OSS, que permite desplegar sobre un servidor local una instancia de Grafana. Este último es el que concierne a este trabajo, pues el sistema sobre el que se va a desplegar no contará con conexión a Internet la mayoría del tiempo de operación.

- **Fuentes de datos:** Las fuentes de datos proveen a Grafana las métricas que se utilizarán para la visualización de datos. Actualmente cuenta con más de cien integraciones (plataformas que se pueden conectar a Grafana) disponibles, entre las que se incluyen bases de datos temporales (TSDB), bases de datos SQL y noSQL, etc.

  El funcionamiento general de las fuentes de datos de Grafana es a base de consultas. Toda fuente de datos por defecto en Grafana debe tener un lenguaje que soporte solicitudes para acceder a los datos.

  De este modo, Grafana permite mucha flexibilidad a la hora de acceder a estos, soportando de manera eficiente filtros de distintos tipos. No obstante, esto tiene alguna limitación, como por ejemplo no poder aceptar como métricas información que haya en ficheros locales de un sistema.

- **Tableros:** El componente interactivo principal de Grafana son los tableros. Este componente permite acceder, visualizar e incluso exportar las métricas ubicadas en las fuentes de datos.

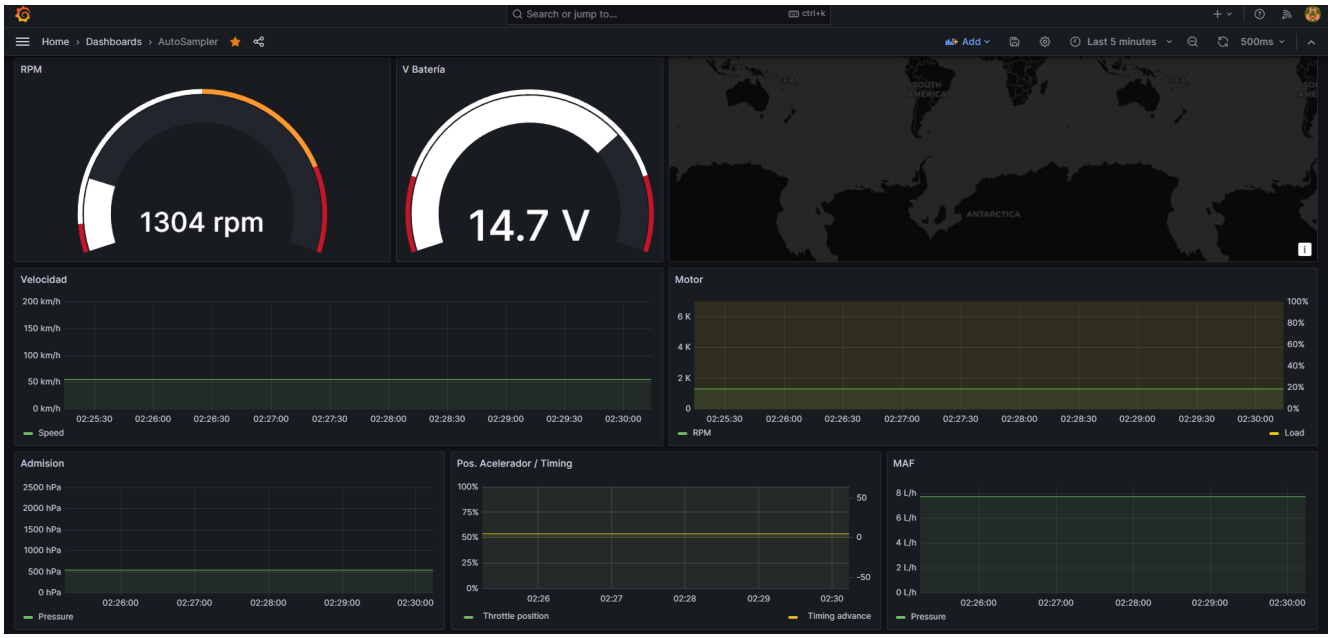


Figura 3.7: Vista de tablero de Grafana.

La unidad mínima de un tablero es el panel, componente que contiene la visualización de una o más métricas. En la Figura 3.7 se puede ver la vista de tablero. Concretamente este contiene 8 paneles, de tres tipos principales:

- Gauge: Visión en tiempo real de una métrica.
- Serie temporal: Vista cronológica de las medidas registradas de una o más métricas.
- Mapa: Vista que ofrece un mapa para representación de coordenadas, entre otras cosas.

El panel de Grafana es un componente muy personalizable, desde la elección de las métricas, ofreciendo opciones como varias consultas, varias fuentes de datos, filtros, operaciones, formatos, leyendas, etc. En la Figura 3.8 se puede ver un ejemplo de una consulta de este proyecto.

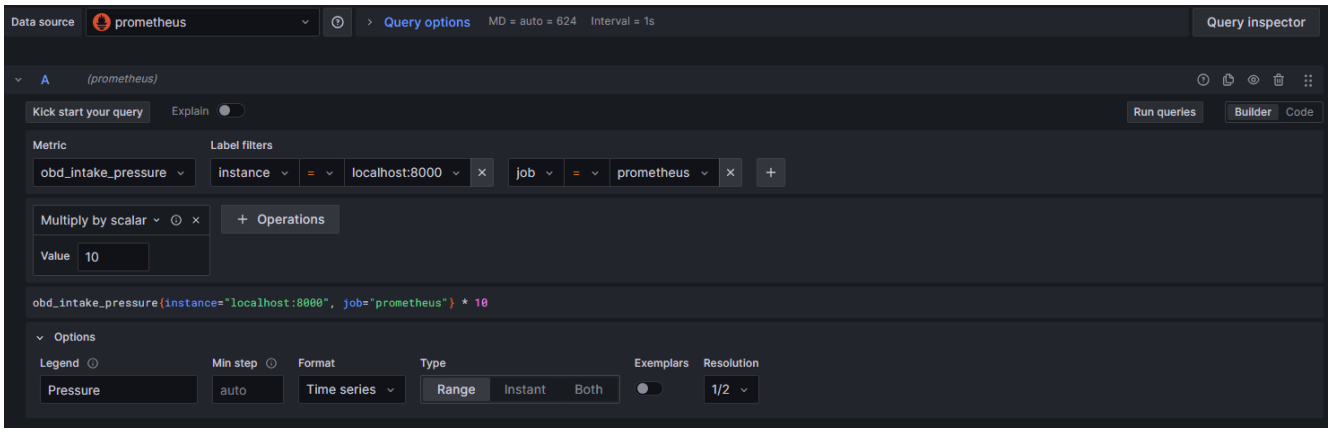


Figura 3.8: Configuración de panel de Grafana.

También ofrece una infinidad de personalización estética sobre los paneles, desde colores, rangos, unidades, valores que enseñar, etc.

- **Plugins:** El último pilar que compone Grafana son los plugins. Estos, impulsados por una comunidad muy activa, permiten mejorar o añadir funcionalidad a la distribución estándar de Grafana. Para la realización del proyecto se ha probado la funcionalidad de diversos plugins, de los que se van a mostrar dos:

  - CSV: A diferencia de las fuentes de datos comentadas en el apartado anterior, este plugin permite leer información de archivos CSV, tanto ubicados en la nube como archivos locales. Esta funcionalidad puede ser interesante para reportar datos de ficheros sencillos que no se modifican en exceso, y cuyo acceso puede ser difícil fuera de la plataforma.

    En el caso del proyecto, las averías del vehículo se escriben en un fichero CSV, que este plugin sencillamente muestra en el tablero, pero acceder al archivo dentro del sistema Linux sería demasiado complejo para un usuario estándar. Esto se desarrollará más extensamente en el capítulo de Implementación.

  - D3-Gauge: Permite usar el diseño de *Gauges* de la plataforma D3 en Grafana. Para este proyecto el uso ha sido exclusivamente con un fin estético, para ofrecer otra versión a la estándar y mostrar las capacidades de personalización de la plataforma.

Figura 3.9: Vista de panel D3-Gauge de Grafana.

3.3.3. Herramientas y Tecnología Software

- **Visual Studio Code:**

  Uno de los editores de texto más populares en la actualidad. Cuenta con extensiones con las que, para cada lenguaje soportado, pueda operar más parecido a como lo hace un IDE completo, con herramientas para resaltar errores, autocompletado, e incluso tareas de depuración.

  En este caso, se ha utilizado principalmente para los scripts de Python de procesamiento de datos, aunque también ocasionalmente para editar otros archivos de código y texto de las partes del proyecto desarrolladas en Windows.

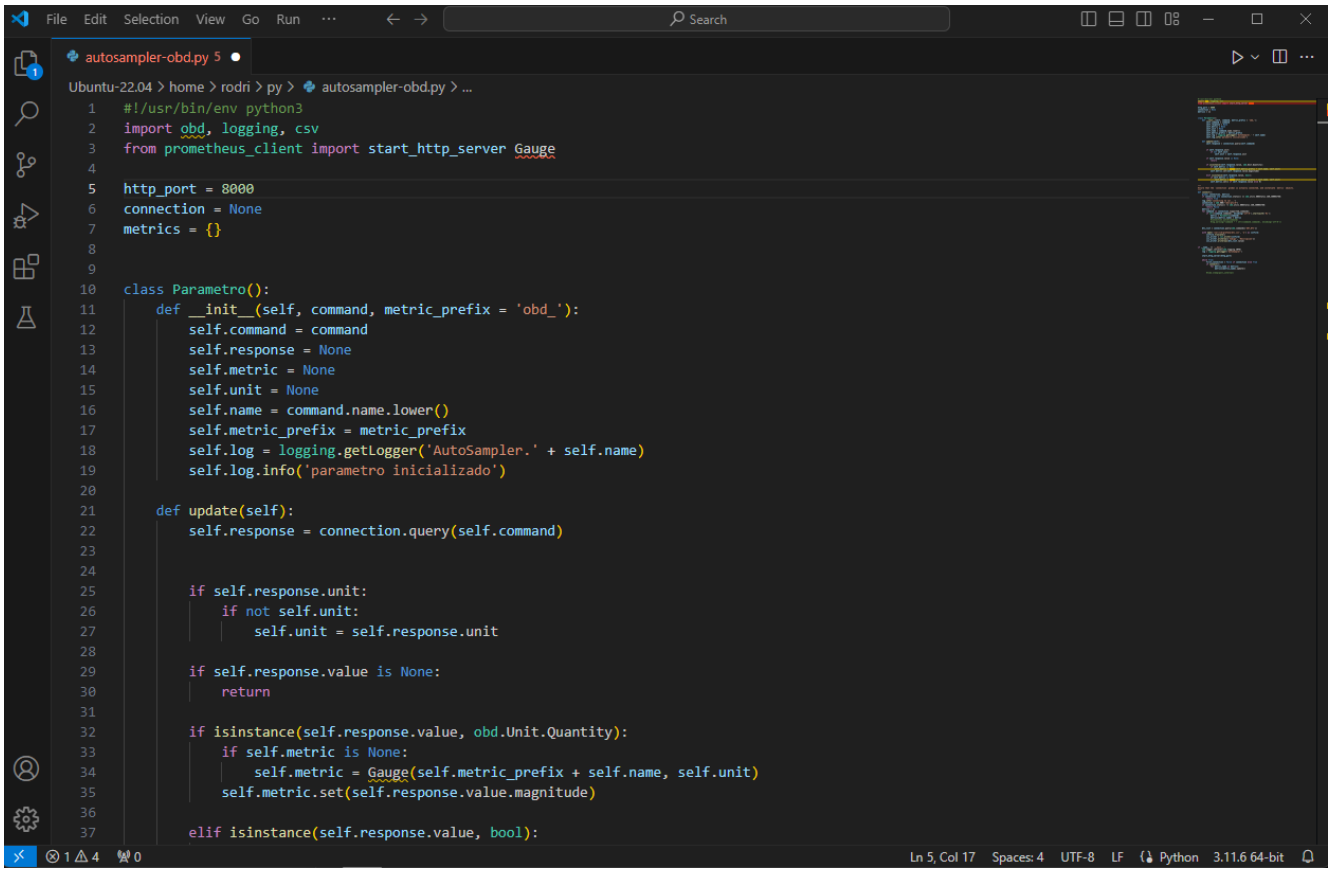


Figura 3.10: Vista de editor de Visual Studio Code.

- **VI:**

  Vi (Visual) es un editor de texto originalmente creado para el sistema operativo Unix. Es una opción muy popular, ya que, además de venir en casi todas las distribuciones de Linux por defecto, es una herramienta muy eficiente y potente para la edición de texto. En este trabajo, se ha usado para prácticamente todos los ficheros de código o texto hechos en Linux. Esto incluye ficheros del sistema, configuración de los diferentes servicios, y archivos Python en la fase de integración.

- **Librerías Python:**

  Pese a que se desarrollará mucho más extensamente en el apartado de Implementación, es importante mencionar el uso de la librería python-OBD por su utilidad en el proyecto. Esta librería maneja la comunicación con el puerto del vehículo, haciendo mucho más sencillo el proceso de conexión y consulta de datos. Como se verá más adelante, la comunicación con el protocolo OBD2 puede ser compleja y tediosa si se hace a bajo nivel, sobre todo teniendo en cuenta los varios protocolos existentes, por lo que esta librería permite hacer solicitudes al vehículo de una manera mucho más sencilla.

- **OBD2:**

  El protocolo OB2 (On-Board Diagnostics) coordina solicitudes hechas desde un terminal con

los ordenadores del vehículo, principalmente con la Unidad de Control del Motor (ECU). El funcionamiento de las solicitudes y las respuestas viene definido en el ISO 15031-5. Esencialmente, el usuario envía un código compuesto por el modo o servicio (de los seis disponibles) seguido del PID (Parameter ID). A continuación se describe un ejemplo de su funcionamiento.

Si se quisiese solicitar el parámetro de la velocidad en ese instante, primero se establecería la conexión con el vehículo mandando una cabecera para establecer el protocolo y otros detalles, por ejemplo: "ATSP0", mensaje que intenta establecer un protocolo automáticamente.

Una vez establecida la conexión se compondría un mensaje con el modo y el PID. En este caso, el modo es 01 (datos en tiempo real) y el PID 0D (correspondiente a la velocidad) "010D" La respuesta, para este caso, podría tendría la forma "410D30", dónde:

- 41 indica que se trata una respuesta del modo 01
- 0D referencia el módulo solicitado, y coincide con el de la solicitud
- 30 es el valor devuelto. En la mayoría de casos, este valor vendrá en hexadecimal y se tendrá que aplicar una fórmula para obtener un valor legible.

El parámetro de la velocidad no necesita ninguna fórmula, por lo que 30 es directamente el valor solicitado en hexadecimal, que serían "48 km/h".

En otros casos, por ejemplo las revoluciones por minuto, el vehículo devolverá 2 bytes de datos (A y B) a los que se tendrá que aplicar la fórmula: $256A + B/100$ para obtener un resultado legible.

3.3.4. Herramientas y Tecnología Hardware

- **OBD2 y ELM327:**

En este apartado se discutirá la parte física del puerto OBD2 y el chip de comunicaciones usado para el proyecto, ELM327. Aunque se podría extender detallando el funcionamiento y utilidad del chip y los pines del puerto, solo se comentará lo que es de utilidad para este proyecto.

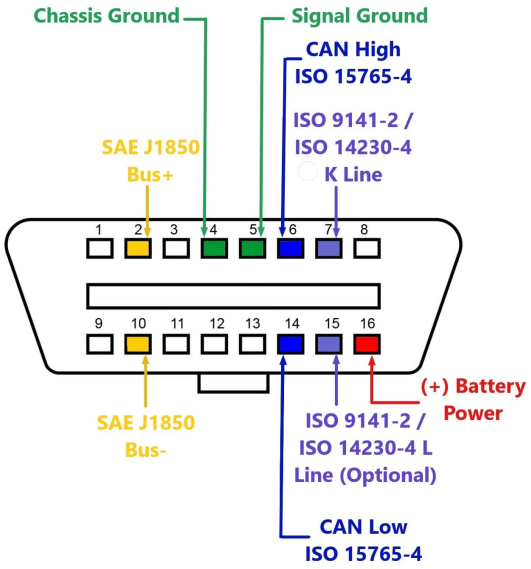


Figura 3.11: Disposición de los pines del puerto OBD2 tipo A.

Este puerto, definido en el estándar SAE J1962, contiene pines para muchos protocolos y distintos modos de comunicación, de los cuales nos centraremos principalmente en tres:

- **ISO 9141-2:** Encontrado en la mayoría de coches desde el año 2000 hasta el 2003. Usa los pines 7 y 15 del conector (K-Line y L-Line) para el envío de datos a través de una comunicación serial asíncrona.
- **ISO 13230-4:** En adelante Keyword Protocol 2000 o KWP. Encontrado desde 2003 hasta la adopción de CAN sobre el 2008. Funcionamiento muy similar al ISO mencionado anteriormente, por lo que usa los mismos pines.
- **ISO 15765:** En Adelante Bus CAN (Controller Area Network). Protocolo usado actualmente. Compone una red de módulos dentro del vehículo, sin un módulo central. Usa los pines 6 y 14 del puerto para la comunicación.

El microcontrolador ELM327, encargado de traducir la interfaz OBD, es una herramienta muy popular en el mercado de diagnósticos, por su compatibilidad con prácticamente todos los protocolos existentes y su reducido precio. Los ELM327 están implementados en microcontroladores de la familia PIC, como el PIC18F2480.

Aunque no transmitan la mayor cantidad de datos y la velocidad de transmisión no sea la mejor, es la opción más óptima para un proyecto de este tipo, ya que se puede asegurar el funcionamiento con una gran mayoría de coches de distintas épocas.

Para cada protocolo, ELM327 soporta las siguientes velocidades de transmisión:

- **ISO 9141-2 y 13230-4 (KWP):** 10.4 kbits/s
- **ISO 15765 (CAN):** 500 kbits/s

Capítulo 4

Análisis

En esta fase del proyecto se desarrolla la obtención de los objetivos y necesidades de este, detallando de forma precisa y analítica las funcionalidades, requisitos y componentes que el prototipo final requiera.

Como se ha comentado en la Introducción, los objetivos principales estaban claros y no debían cambiar, pero debido a una fase esencial de exploración tecnológica que se ha desarrollado previamente y a lo largo del desarrollo de los módulos, hay requisitos específicos y algún objetivo secundario que han sufrido cambios, o se han reemplazado por otros debido a restricciones tecnológicas o una mejor visión del proyecto, pero siempre que se ha efectuado un cambio ha sido en aras de satisfacer mejor los objetivos recogidos inicialmente.

Tanto en este capítulo de Análisis como en el siguiente de Diseño se va a combinar el uso del lenguaje de modelado UML con el lenguaje de especificación de sistemas SysML. Este último permite, de una manera más precisa, definir ciertas partes de este sistema, sobre todo estructuras y comportamientos que no se pueden describir usando elementos estrictamente de software.

4.1. Identificación de Usuarios

El sistema ha sido diseñado para ser usado por un único tipo de usuario. Este usuario es el que instalará, utilizará y administrará el sistema. Esto implica que, aunque para el análisis se recoja por un único tipo de usuarios, el sistema tiene que estar preparado para soportar interacciones de usuarios con distinto nivel técnico.

4.2. Paquetes

En la siguiente figura se puede ver el diagrama de paquetes del sistema. Este modelo de organización dicta como se agrupa y como se va a presentar la información. Este capítulo se centrará en

los paquetes A y B, es decir, Requisitos, Casos de Uso y Estructura del sistema.

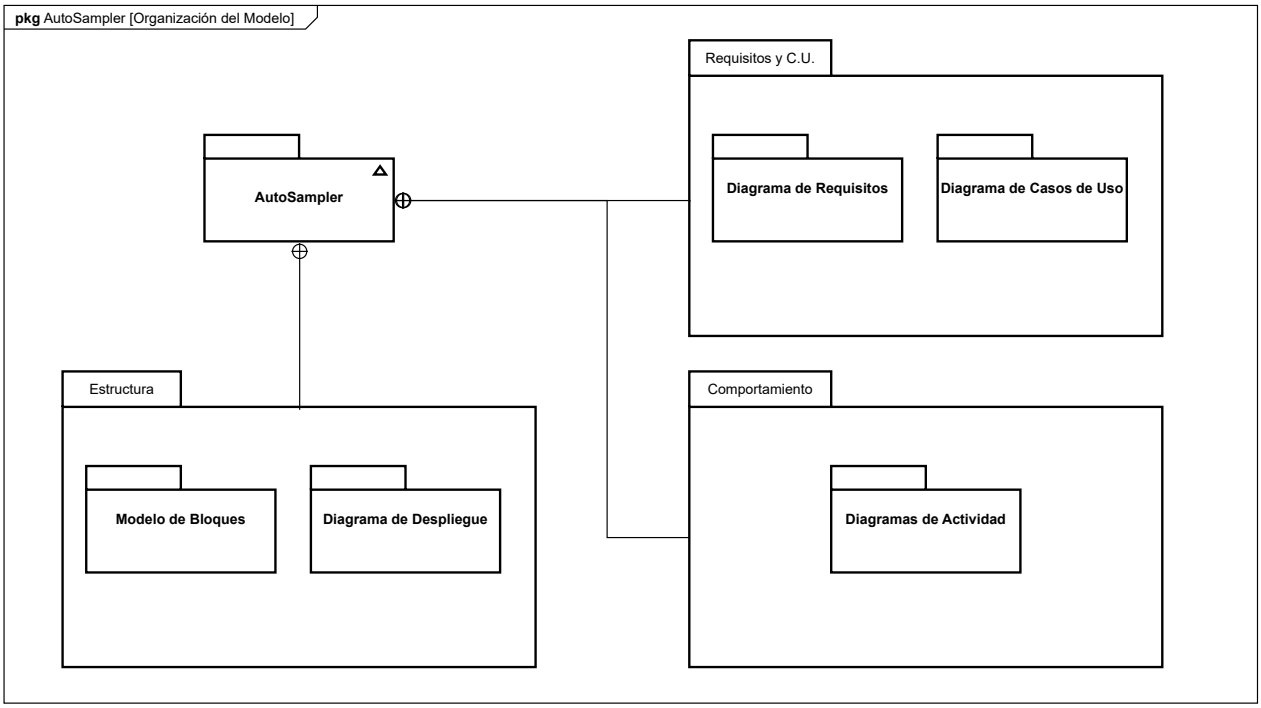


Figura 4.1: Diagrama de paquetes de AutoSampler.

4.3. Requisitos

En esta sección se describen los requisitos necesarios para la entrega de un sistema que cumpla con los objetivos propuestos.

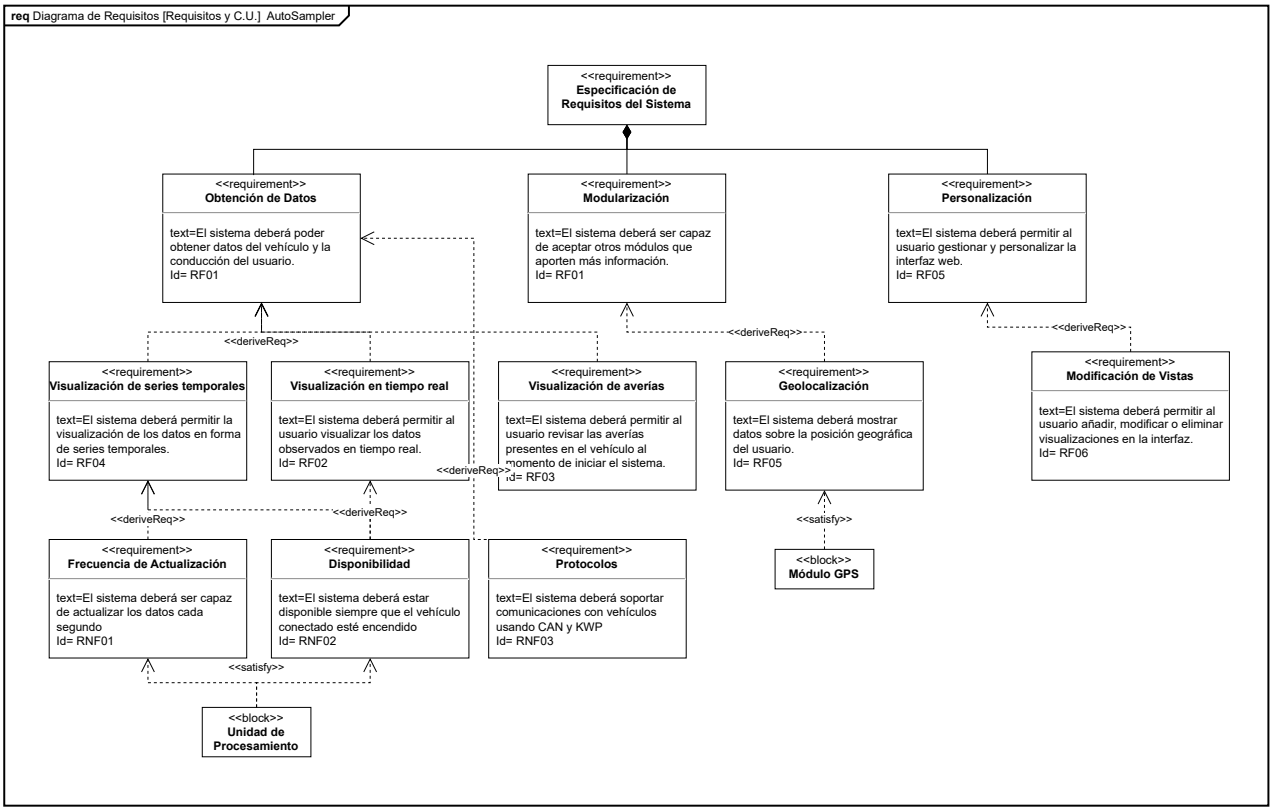


Figura 4.2: Diagrama de requisitos de AutoSampler.

Esta figura muestra el diagrama de requisitos. En este se puede ver la trazabilidad entre requisitos y estructuras del modelo del sistema, por tanto, se ve como los requisitos más específicos dependen de objetivos o requisitos más generales, y como, en última instancia, la mayoría de requisitos dependen de, o son satisfechos por "bloques" del sistema.

Por ejemplo, el requisito con ID RF-01: Obtención de datos, comprende prácticamente todas las funciones del módulo de Captura y Procesamiento de Datos; sus derivados directos, con ID RF-02 y Por ejemplo, el requisito con ID RF-01: Obtención de datos, comprende prácticamente todas las funciones del módulo de Captura y Procesamiento de Datos; sus derivados directos, con ID RF-02 y RF-04, definen comportamientos más específicos que debe soportar el sistema; y de estos derivan las restricciones en forma de requisitos no funcionales que se aplican para el funcionamiento esperado.

Por último, para seguir con este ejemplo, se puede ver que las restricciones que se aplican son satisfechas directamente por la unidad de procesamiento, ya que concretamente estas se refieren a periodos de actualización y disponibilidad.

4.3.1. Requisitos funcionales

Los requisitos funcionales tienen la función de describir con exactitud las funcionalidades que el sistema deberá implementar.

- RF-01: El sistema deberá poder obtener datos sobre el vehículo y la conducción del usuario.
- RF-02: El sistema deberá permitir al usuario visualizar los datos observados en tiempo real.
- RF-03: El sistema deberá permitir al usuario revisar las averías presentes en el vehículo al momento de iniciarse.
- RF-04: El sistema deberá permitir la visualización de los datos en forma de series temporales.
- RF-05: El sistema deberá mostrar datos sobre la posición geográfica del usuario.
- RF-06: El sistema deberá ser capaz de aceptar otros módulos que aporten más información.
- RF-07: El sistema deberá permitir al usuario gestionar y personalizar la interfaz.
- RF-08: El sistema deberá permitir al usuario añadir visualizaciones a la interfaz.
- RF-09: El sistema deberá permitir al usuario modificar visualizaciones en la interfaz.
- RF-10: El sistema deberá permitir al usuario eliminar visualizaciones de la interfaz.
- RF-11: El sistema deberá permitir al usuario exportar los datos a otros formatos.

4.3.2. Requisitos no funcionales

Estos requisitos determinan las restricciones que se aplican sobre el sistema para asegurar el funcionamiento correcto. Esto incluye especificaciones sobre el diseño del sistema, características técnicas y filosofías de funcionamiento de las soluciones.

- RNF-01: El sistema deberá ser capaz de obtener y actualizar los datos cada segundo.
- RNF-02: El sistema deberá estar disponible siempre que el vehículo conectado esté encendido.
- RNF-03: El sistema deberá soportar comunicaciones con vehículos por el bus CAN y KWP.
- RNF-04: El sistema y todos sus módulos deberán iniciarse automáticamente una vez iniciada la máquina.
- RNF-05: El sistema deberá poder desplegar un servidor Grafana en la red local para el acceso a los datos.
- RNF-06: El sistema deberá poder ser desplegado en cualquier ordenador que use un sistema operativo GNU/Linux, independientemente de las características o arquitectura.

4.4. Diagramas de definición de bloques

Los diagramas de definición de bloques SysML muestran los elementos de definición del modelo del sistema y las relaciones estructurales que hay entre ellos. Se entienden por elementos de definición las estructuras, o bloques, que forman la base del resto del modelo.

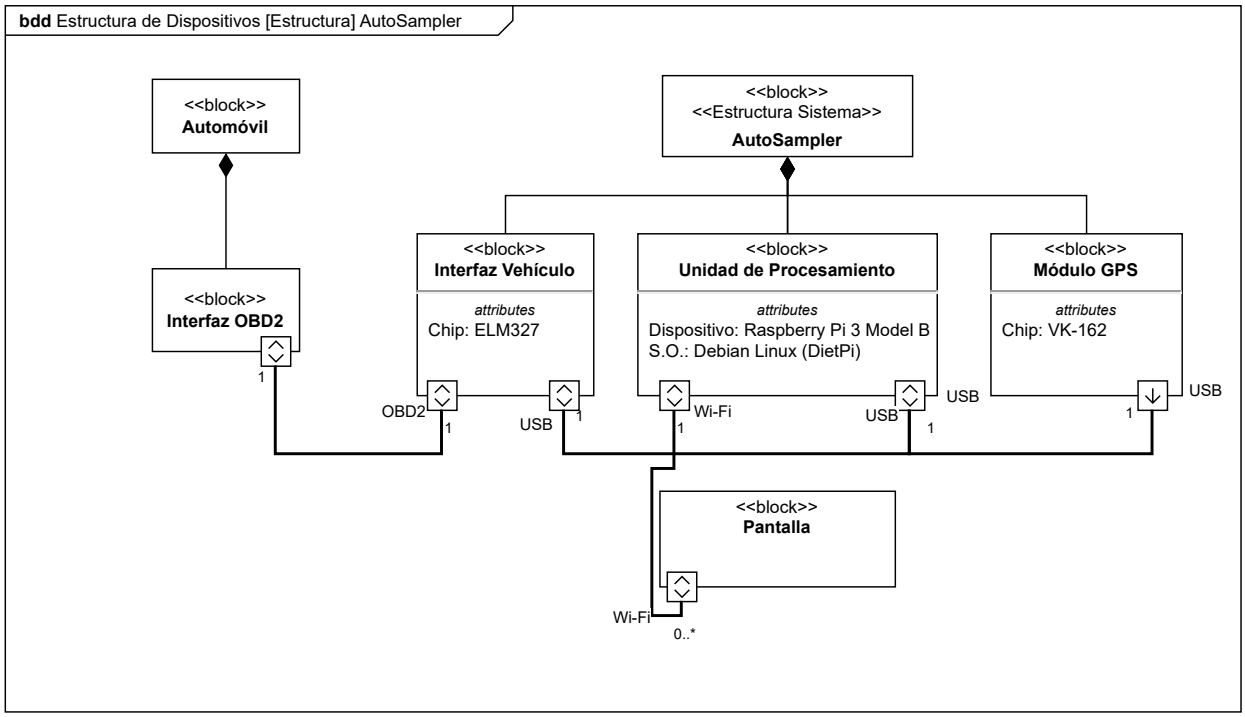


Figura 4.3: Diagrama de definición de bloques de la estructura de dispositivos de AutoSampler.

En esta figura se puede ver la estructura de dispositivos del sistema. Por un lado, está el sistema principal, AutoSampler, que cuenta con la unidad de procesamiento, la unidad que realiza el cómputo de los parámetros, coordina todos los módulos y despliega la interfaz de usuario; la interfaz del vehículo, explicada en profundidad en el anterior capítulo; y el módulo GPS.

Por otro lado, están representados también los sistemas externos, como el vehículo y su interfaz OBD2, o la pantalla de visualización, conectada a través de Wi-Fi.

A continuación, se muestra el BDD de AutoSampler.

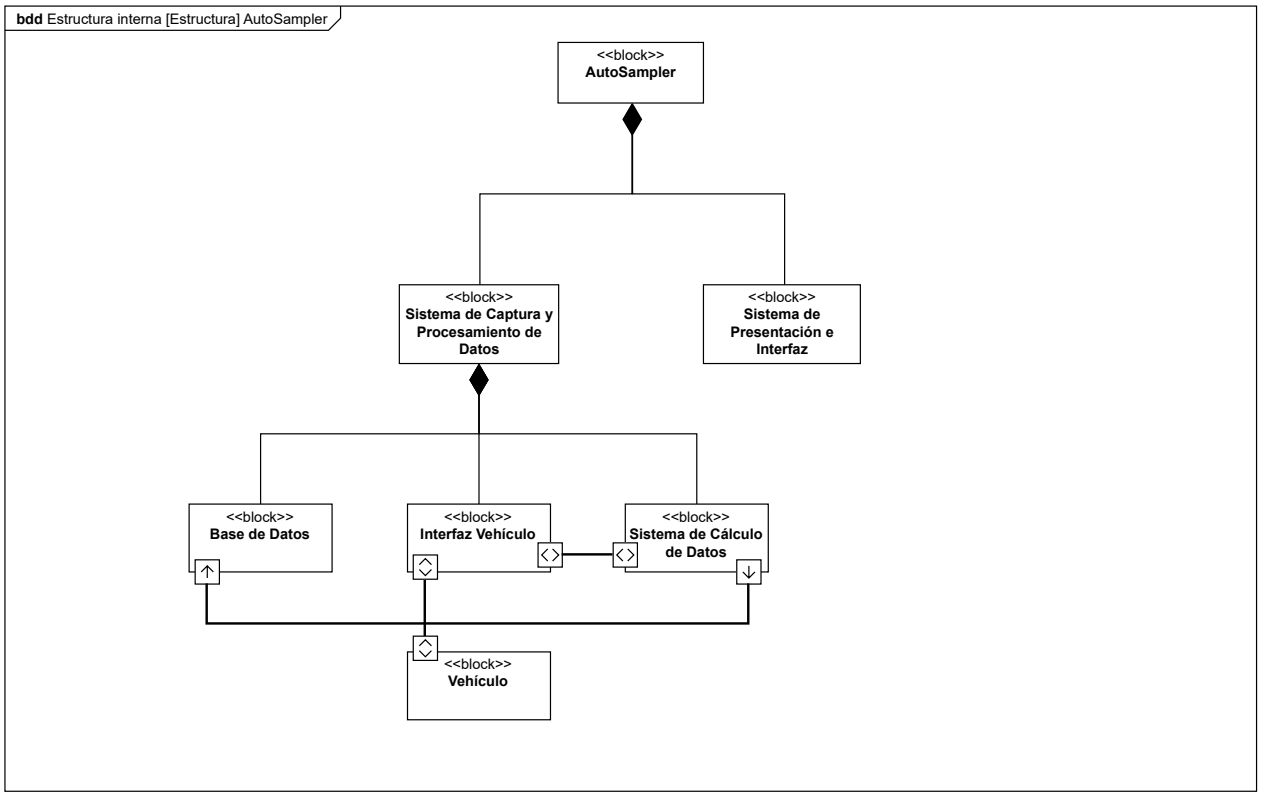


Figura 4.4: Diagrama de definición de bloques de la estructura de dispositivos de AutoSampler.

En este se puede observar más detalladamente el funcionamiento interno de la Unidad de Procesamiento del anterior diagrama. Sus funciones están divididas en dos bloques: Captura y Procesamiento de Datos, encargado de extraer y calcular la información del vehículo y de los módulos; y Presentación e Interfaz, encargado de la disponibilización de la información para las pantallas conectadas.

Es interesante también comentar el flujo de datos dentro del sistema. El sistema de cálculo de datos solicita los parámetros a la interfaz del vehículo, bloque físico que se encuentra en los dos diagramas, que los recoge de la interfaz OBD2 del vehículo. Con los datos calculados, el sistema lo envía a la base de datos para su visualización.

4.5. Casos de uso

Un caso de uso es la descripción de una acción o actividad que un usuario realiza en el uso del sistema. Los casos de uso deben quedar especificados en la fase de planificación el proyecto, para determinar el comportamiento específico del sistema ante las acciones de un usuario.
En este trabajo, dado que la mayor parte del sistema no requiere interacción con el usuario, solo se van a tener en cuenta los casos de uso de la interfaz de usuario, ya sean tareas de administración, consulta de datos, personalización, etc.

4.5.1. Diagrama de casos de uso

Similarmente a la definición de caso de uso, un diagrama de caso de uso es una descripción de las actividades que deberá realizar alguien o algo para llevar a cabo algún proceso. Las entidades que participan en un diagrama de caso de uso se denominan actores. En este caso, el actor es el usuario identificado al principio del capítulo.

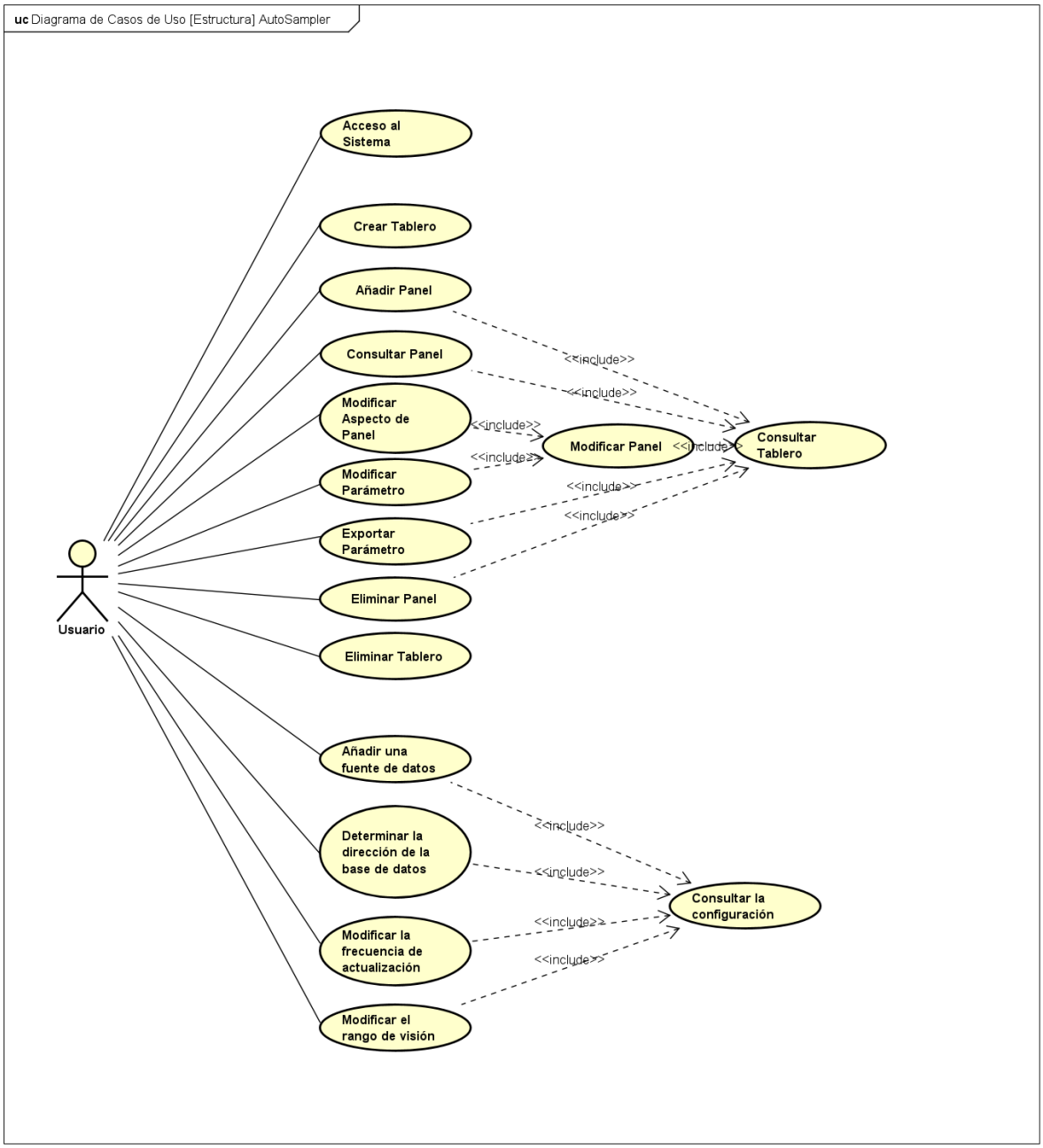


Figura 4.5: Diagrama de casos de uso de AutoSampler.

4.5.2. Especificación de casos de uso

A continuación, se presenta la lista de todos los casos de uso contemplados para la interacción con el sistema:

- CU-01: Acceso al sistema.
- CU-02: Crear Tablero.
- CU-03: Consultar Tablero.
- CU-04: Añadir Panel.
- CU-05: Consultar Panel.
- CU-06: Modificar Aspecto de Panel.
- CU-07: Modificar Parámetro.
- CU-08: Eliminar Panel.
- CU-09: Eliminar Tablero.
- CU-10: Añadir una fuente de datos.
- CU-11: Consultar la Configuración.
- CU-12: Determinar la dirección de la fuente de datos.
- CU-13: Modificar la frecuencia de actualización.
- CU-14: Modificar el rango de visión.

Para cada caso de uso, se contempla también la siguiente especificación del flujo de proceso:

CU-01	**Acceso Al Sistema**
Actor	Usuario
Descripción	El sistema deberá permitir al usuario acceder a la plataforma de visualización.
Secuencia Normal	1. El usuario accede al navegador en el dispositivo Pantalla. 2. El usuario escribe la dirección del servidor de AutoSampler. 3. El sistema le permite el acceso.
Postcondición	El usuario ha accedido al sistema.
Excepciones	
Variación	Acción
3b	Si se han configurado credenciales, el sistema deberá verificarlas antes de permitir el acceso.

Tabla 4.1: Caso de Uso 1. Acceso al Sistema

CU-02	**Crear Tablero**
Actor	Usuario
Descripción	El sistema deberá permitir al usuario crear tableros en la plataforma.
Precondición	(CU-01) El usuario ha accedido al sistema.
Secuencia Normal	1. El usuario accede al menú de creación. 2. El usuario configura las opciones del tablero. 3. El sistema confirma la acción y crea el tablero.
Postcondición	El usuario ha creado un tablero.
Excepciones	
Variación	Acción
2b	Si el usuario cancela la acción, se le devuelve al menú principal y se cancela el caso de uso.

Tabla 4.2: Caso de Uso 2. Crear Tablero

CU-03	**Crear Tablero**
Actor	Usuario
Descripción	El sistema deberá permitir al usuario consultar los paneles de tableros creados.
Precondiciones	(CU-01) El usuario ha accedido al sistema. (CU-02) El usuario ha creado un tablero.
Secuencia Normal	1. El usuario accede al menú principal. 2. El usuario selecciona un tablero. 3. El sistema confirma la acción y muestra los paneles del tablero.
Postcondición	El usuario ha accedido al tablero.
Excepciones	
Variación	Acción
2b	Si no hay ningún vehículo conectado, los datos en tiempo real se mostrarán en blanco.

Tabla 4.3: Caso de Uso 3. Consultar Tablero

CU-04	**Añadir Panel**
Actor	Usuario
Descripción	El sistema deberá permitir al usuario crear paneles dentro de los tableros.
Precondiciones	(CU-01) El usuario ha accedido al sistema. (CU-02) El usuario ha creado un tablero.
Secuencia Normal	1. El usuario accede a la vista del tablero. 2. El usuario selecciona crear un nuevo panel. 3. El usuario configura los parámetros, rangos y tipos. 4. El sistema confirma la acción y crea el panel.
Postcondición	El usuario ha añadido un panel.
Excepciones	
Variación	Acción
3b	Si el usuario cancela la acción, se le devuelve a la vista del tablero y se cancela el caso de uso.

Tabla 4.4: Caso de Uso 4. Crear Panel

CU-05	**Consultar Panel**
Actor	Usuario
Descripción	El sistema deberá permitir al usuario consultar el/los parámetros observados en un panel.
Precondiciones	(CU-01) El usuario ha accedido al sistema. (CU-02) El usuario ha creado un tablero. (CU-04) El usuario ha creado un panel.
Secuencia Normal	1. El usuario accede a la vista del tablero. 2. El usuario selecciona un panel existente. 3. El sistema confirma la acción y muestra la información del panel.
Postcondición	El usuario tiene acceso al panel.
Excepciones	
Variación	Acción
3b	Si el usuario cancela la acción, se le devuelve a la vista del tablero y se cancela el caso de uso.

Tabla 4.5: Caso de Uso 5. Consultar Panel

CU-06	**Modificar Aspecto de Panel**
Actor	Usuario
Descripción	El sistema deberá permitir al usuario modificar el aspecto visual de un panel.
Precondiciones	(CU-01) El usuario ha accedido al sistema. (CU-02) El usuario ha creado un tablero. (CU-04) El usuario ha creado un panel.
Secuencia Normal	1. El usuario accede a la vista del tablero. 2. El usuario selecciona un panel existente. 3. El usuario configura el aspecto visual. 4. El sistema confirma la acción y muestra el panel modificado.
Postcondición	El usuario ha modificado el aspecto de un panel.
Excepciones	
Variación	Acción
3b	Si el usuario cancela la acción, se le devuelve a la vista del tablero y se cancela el caso de uso.

Tabla 4.6: Caso de Uso 6. Modificar Aspecto de Panel

CU-07	**Modificar Parámetro**
Actor	Usuario
Descripción	El sistema deberá permitir al usuario modificar el/los parámetros observados en un panel.
Precondiciones	(CU-01) El usuario ha accedido al sistema. (CU-02) El usuario ha creado un tablero. (CU-04) El usuario ha creado un panel.
Secuencia Normal	1. El usuario accede a la vista del tablero. 2. El usuario selecciona un panel existente. 3. El usuario configura los parámetros. 4. El sistema confirma la acción y muestra el panel modificado.
Postcondición	El usuario ha modificado el parámetro de un panel.
Excepciones	
Variación	Acción
3b	Si el usuario cancela la acción, se le devuelve a la vista del tablero y se cancela el caso de uso.

Tabla 4.7: Caso de Uso 7. Modificar Parámetro

CU-08	**Exportar Parámetro**
Actor	Usuario
Descripción	El sistema deberá permitir al usuario exportar datos observados en un panel.
Precondiciones	(CU-01) El usuario ha accedido al sistema. (CU-02) El usuario ha creado un tablero. (CU-04) El usuario ha creado un panel.
Secuencia Normal	1. El usuario accede a la vista del tablero. 2. El usuario selecciona un panel existente. 3. El usuario selecciona la opción de exportar. 4. El usuario confirma las opciones de exportación. 5. El sistema confirma la acción y muestra el tablero modificado.
Postcondición	El usuario ha exportado un parámetro.
Excepciones	
Variación	Acción
4b	Si el usuario cancela la acción, se le devuelve a la vista del tablero y se cancela el caso de uso.

Tabla 4.8: Caso de Uso 8. Eliminar Panel

CU-09	**Eliminar Panel**
Actor	Usuario
Descripción	El sistema deberá permitir al usuario eliminar un panel del tablero.
Precondiciones	(CU-01) El usuario ha accedido al sistema. (CU-02) El usuario ha creado un tablero. (CU-04) El usuario ha creado un panel.
Secuencia Normal	1. El usuario accede a la vista del tablero. 2. El usuario selecciona un panel existente. 3. El usuario selecciona la opción de eliminarlo. 4. El usuario confirma la acción. 5. El sistema confirma la acción y muestra el tablero modificado.
Postcondición	El usuario ha eliminado un panel.
Excepciones	
Variación	Acción
4b	Si el usuario cancela la acción, se le devuelve a la vista del tablero y se cancela el caso de uso.

Tabla 4.9: Caso de Uso 9. Eliminar Panel

CU-10	**Eliminar Tablero**
Actor	Usuario
Descripción	El sistema deberá permitir al usuario eliminar un tablero..
Precondiciones	(CU-01) El usuario ha accedido al sistema. (CU-02) El usuario ha creado un tablero.
Secuencia Normal	1. El usuario accede a la vista del tablero. 2. El usuario selecciona la opción de eliminarlo. 3. El usuario confirma la acción. 4. El sistema confirma la acción y muestra la pantalla principal.
Postcondición	El usuario ha eliminado un tablero.
Excepciones	
Variación	Acción
3b	Si el usuario cancela la acción, se le devuelve a la vista del tablero y se cancela el caso de uso.

Tabla 4.10: Caso de Uso 10. Eliminar Tablero

CU-11	**Añadir una fuente de datos**
Actor	Usuario
Descripción	El sistema deberá permitir al usuario añadir una fuente de datos a la plataforma.
Precondición	(CU-01) El usuario ha accedido al sistema.
Secuencia Normal	1. El usuario accede a la vista de configuración. 2. El usuario selecciona la opción de fuente de datos. 3. El usuario introduce los parámetros de la nueva fuente. 4. El sistema confirma la acción y muestra la pantalla principal.
Postcondición	El usuario ha añadido una fuente de datos.
Excepciones	
Variación	Acción
3b	Si el usuario cancela la acción, se le devuelve a la vista de configuración y se cancela el caso de uso.
3c	Si la nueva fuente de datos es inválida, se avisa al usuario y se le devuelve al paso 3.

Tabla 4.11: Caso de Uso 11. Añadir una fuente de datos

CU-12	**Consultar la configuración**
Actor	Usuario
Descripción	El sistema deberá permitir al usuario consultar la configuración de la plataforma.
Precondición	(CU-01) El usuario ha accedido al sistema.
Secuencia Normal	1. El usuario accede al menú principal. 2. El usuario solicita modificar los parámetros del sistema. 3. El sistema confirma la acción y muestra la configuración.
Postcondición	El usuario ha modificado la configuración.

Tabla 4.12: Caso de Uso 12. Consultar la configuración

CU-13	**Determinar la dirección de una fuente de datos**
Actor	Usuario
Descripción	El sistema deberá permitir al usuario determinar la dirección de una fuente de datos.
Precondición	(CU-01) El usuario ha accedido al sistema.
Secuencia Normal	1. El usuario accede a la vista de configuración. 2. El usuario selecciona la opción de fuente de datos. 3. El usuario selecciona una fuente de datos existente. 4. El usuario introduce la nueva dirección. 5. El sistema confirma la acción y muestra la vista de configuración.
Postcondición	El usuario ha modificado la dirección de una fuente de datos.
Excepciones	
Variación	Acción
4b	Si el usuario cancela la acción, se le devuelve a la vista de configuración y se cancela el caso de uso.

Tabla 4.13: Caso de Uso 13. Determinar la dirección de una fuente de datos

CU-14	**Modificar la frecuencia de actualización**
Actor	Usuario
Descripción	El sistema deberá permitir al usuario modificar la frecuencia de actualización de los datos de un tablero.
Precondiciones	(CU-01) El usuario ha accedido al sistema. (CU-02) El usuario ha creado un tablero.
Secuencia Normal	1. El usuario accede a la vista del tablero. 2. El usuario accede a la configuración del tablero. 3. El usuario selecciona la opción de frecuencia de actualización. 4. El usuario modifica la lista de frecuencias aceptadas. 5. El sistema confirma la acción y muestra la vista del tablero.
Postcondición	El usuario ha modificado la frecuencia de actualización del tablero.
Excepciones	
Variación	Acción
4b	Si el usuario cancela la acción, se le devuelve a la vista de configuración y se cancela el caso de uso.
4c	Si la nueva frecuencia es inválida, se avisa al usuario y se le devuelve al paso 4.

Tabla 4.14: Caso de Uso 14. Modificar la frecuencia de actualización

CU-15	**Modificar el rango de visión**
Actor	Usuario
Descripción	El sistema deberá permitir al usuario modificar el rango de visión temporal de los datos de un tablero.
Precondiciones	(CU-01) El usuario ha accedido al sistema. (CU-02) El usuario ha creado un tablero.
Secuencia Normal	1. El usuario accede a la vista del tablero. 2. El usuario accede a la configuración del tablero. 3. El usuario selecciona la opción de rango de tiempo. 4. El usuario modifica el rango de las series temporales. 5. El sistema confirma la acción y muestra la vista del tablero.
Postcondición	El usuario ha modificado el rango temporal de un tablero.
Excepciones	
Variación	Acción
4b	Si el usuario cancela la acción, se le devuelve a la vista de configuración y se cancela el caso de uso.
4c	Si el nuevo rango es inválido, se avisa al usuario y se le devuelve al paso 4.

Tabla 4.15: Caso de Uso 15. Modificar el rango de visión

Capítulo 5

Diseño

En este capítulo se detallará el proceso de diseño del sistema. Según su definición, es el proceso por el que un agente crea una especificación de un artefacto de software, pensado para cumplir unos objetivos.

El diseño de este sistema ha supuesto varios desafíos, tanto por la complejidad de este y sus requisitos, como por su distanciamiento de un proyecto de software convencional.

Es por esto que la estructura de este capítulo variará ligeramente, ya que, aunque se mantenga una visión general del diseño del sistema, es importante mencionar los procesos de diseño y filosofías que se han utilizado para los módulos, así como las diferencias que ha supuesto su despliegue en un sistema empotrado.

5.1. Arquitectura

En esta sección se va a analizar la arquitectura elegida para el desarrollo del trabajo. Se va a dividir en dos partes: Arquitectura física y Arquitectura lógica.

5.1.1. Arquitectura física

En la arquitectura física, han existido varias circunstancias que han definido las decisiones tomadas:

En primer lugar, se quería asegurar la conexión correcta y estable entre el vehículo y AutoSampler, por esto, se decidió que la conexión entre el adaptador ELM327 y la Unidad de Procesamiento debía ser por USB. Esto significa que la Unidad de Procesamiento (Raspberry Pi) debe estar a escasos metros del puerto OBD2 del vehículo, por lo que, para hacer el dispositivo cómodo y útil, debía quedar oculto cerca del puerto. Análogamente, el módulo GPS elegido para el proyecto también se ha conectado por USB.

Por lo tanto, la otra conexión principal del sistema, la pantalla, se debía hacer de forma inalámbrica, más concretamente por Wi-Fi, ya que permite velocidades suficientes para el flujo de datos de este sistema. Se posibilitan dos modos de funcionamiento: La Unidad de Procesamiento crea un punto de acceso al que se conecta la pantalla, o el usuario crea uno con su teléfono móvil,

al que se conecta AutoSampler. En el apartado de Implementación, se desarrollarán extensamente los dos modos de funcionamiento y se discutirá los pros y contras de cada uno.

Teniendo en cuenta estos condicionantes, se han determinado los siguientes elementos que conforman la arquitectura física:

Nodos

- Vehículo
- Interfaz OBD2
- Unidad de Procesamiento (Raspberry Pi 3B)
- Módulo GPS
- Pantalla

En la Figura 5.1 se puede consultar el diagrama de despliegue.

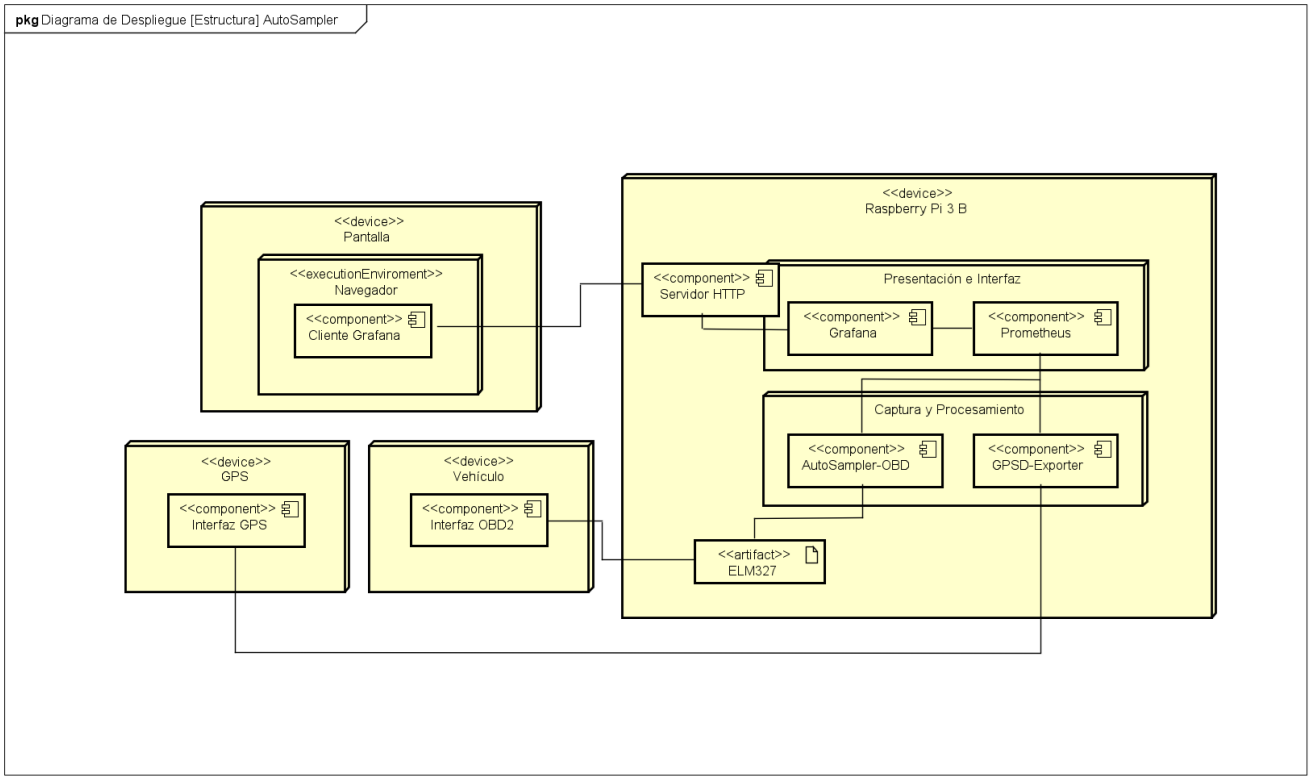


Figura 5.1: Diagrama de Despliegue del Proyecto.

5.1.2. Arquitectura lógica

La identificación de una arquitectura lógica que permitiese cumplir con los requisitos y la flexibilidad propuesta para este sistema ha sido una tarea compleja. Por ello, se va a hacer una diferenciación entre los dos módulos principales.

Módulo de Presentación e Interfaz de Usuario

Este módulo, encargado de presentar la información al usuario, no se corresponde realmente con trabajo de desarrollo software en el marco de este proyecto, si no más de tecnologías de la información y administración de sistemas.

No es relevante entonces hablar de patrones arquitectónicos, ya que las plataformas utilizadas para la resolución del módulo no han sido creadas específicamente para este, y se desconoce (ya que no concierne realmente al proyecto) como funcionan internamente. Por lo tanto, se encapsulará el funcionamiento general del módulo como una parte de la arquitectura que se presentará a continuación.

Módulo de Captura y Procesamiento de Datos

Este módulo engloba todas las soluciones que capturan datos, los procesan y los publican para el otro módulo. Esto incluye en este proyecto al componente principal, la interfaz OBD2; pero también todos los módulos que se pueden incluir, como es el caso del módulo GPS.

Pese a que no hay ningún patrón que se ajuste exactamente al comportamiento de los sistemas que componen este módulo, se van a considerar como Microservicios. Los Microservicios proponen un enfoque arquitectónico donde el software está compuesto por pequeños servicios independientes, escalables, etc. Cada microservicio se puede implementar de forma independiente. Los microservicios se comunican a través de interfaces.

Más aplicado a este caso, cada módulo de captura sería un componente, implementando soluciones independientes a problemas "pequeños". En otras palabras, cada módulo del sistema, de manera independiente, captura datos y realiza cálculos diferentes. Cada uno realiza una interfaz donde publica sus parámetros finales, y la implementación de estos componentes no es relevante para el funcionamiento de los otros componentes o módulos, es decir, se pueden reemplazar por otras soluciones que proporcionen la misma interfaz.

Por último, estas interfaces pueden ser usadas por otros componentes como entrada para otros cálculos, y, en última instancia, todos los datos serán recogidos por el otro módulo principal, Presentación e Interfaz de Usuario, al que se podría considerar como otro microservicio, ya que va a requerir todas las interfaces para su funcionamiento.

5.2. Diagrama de actividad

Para finalizar el capítulo, resulta interesante mostrar en un diagrama el flujo de los datos del sistema, es decir, como el módulo de Captura y Procesamiento se conecta con el vehículo, recoge los datos y los publica en la interfaz.

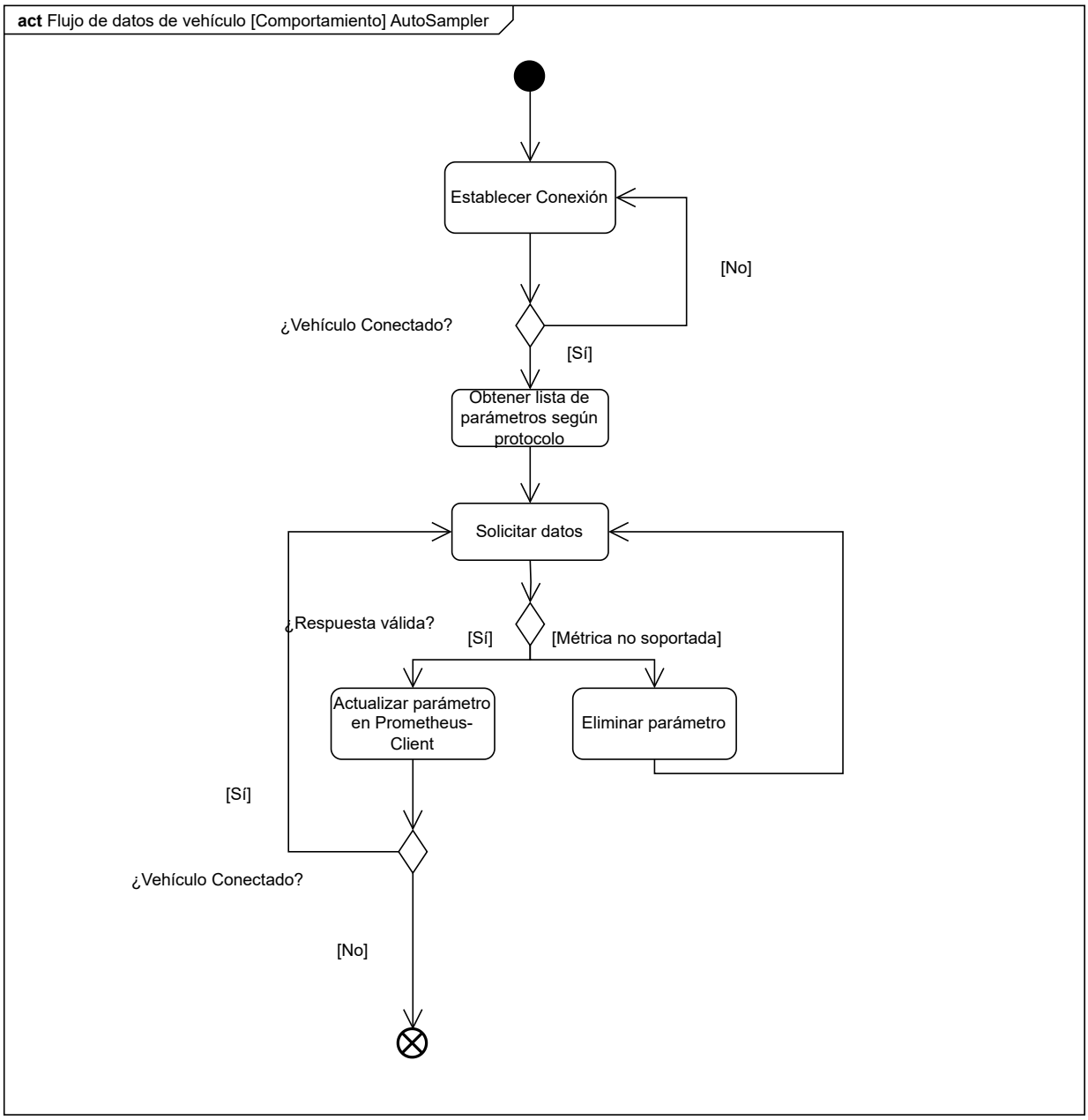


Figura 5.2: Diagrama de Despliegue del Proyecto.

Capítulo 6

Implementación

En este capítulo se tratará la implementación del proyecto, incluyendo las actividades vistas en la red de actividades: Desarrollo y prototipado de los módulos e Integración. Al igual que en el resto del documento, la estructura de este capítulo se va a dividir en los dos módulos principales del sistema, se estudiará cada iteración de los prototipos que han compuesto el sistema final y se justificarán las decisiones tomadas.

6.1. Captura y Procesamiento de Datos

6.1.1. Introducción

Este módulo se corresponde con la primera etapa del desarrollo, ya que, aunque se ha mostrado que las tareas de desarrollo se podían paralelizar, se ha preferido serializarlas al ser solo una persona trabajando en el proyecto.

Es también la parte más extensa en cuanto a desarrollo de software, ya que requiere diseñar una solución que permita, de forma eficiente, extraer parámetros del automóvil y publicarlos para el otro módulo en un periodo de tiempo aceptable.

Cada iteración del prototipo se centra en un enfoque diferente y añadir más funcionalidad, que proporciona una enseñanza a aplicar en el sistema final.

6.1.2. Primera iteración

La primera iteración del módulo se centra en la recogida de datos del vehículo, y se define un objetivo básico: Recoger un parámetro del vehículo en tiempo real.

Como se ha explicado en la metodología, la comunicación OBD2, la elección de protocolos y cálculos para obtener parámetros legibles puede ser tediosa, por ello se va a utilizar una librería de Python que permite mandar comandos de "alto nivel", y recibir una respuesta aceptable.

El primer paso era comprobar que la interfaz OBD2 a través del chip ELM327 funciona correctamente, por lo que, a través de un ordenador portátil Linux, se conectó el adaptador, se encendió el vehículo y se ejecutó el siguiente comando:

screen /dev/ttyUSB0

Por norma general, al solo haber este adaptador conectado, siempre se encontrará en el dispositivo ttyUSB0, aunque, de todos modos, no es un problema para el sistema porque la librería permite "escaneo" de dispositivos.

Una vez dentro del adaptador, se prueba a enviar el comando "010C", correspondiente a la velocidad del motor, para el que se recibe la siguiente respuesta:

Figura 6.1: Respuesta del vehículo al comando enviado.

Esta respuesta nos indica, en las dos primeras partes, que es una respuesta al comando que se ha enviado, y las dos tramas de datos, "14" y "5F" componen la respuesta. Aplicando la fórmula pertinente, $256A + B/4$, se obtiene que la velocidad del motor es: 1.303 RPM, valor acode a lo observado en el cuadro del vehículo y coherente a la velocidad del motor de un coche recién encendido.

Se puede escribir entonces un script Python que automatice esta conexión, envíe un comando y recupere la respuesta. En este recorte de código se puede observar como es el resultado:

```
1 import obd, time
2
3 connection = obd.OBD()
4
5 while True:
6   rpm = connection.query(obd.commands['RPM'])
7   print(rpm.value)
8   time.sleep(0.5)
```

Código 6.1: Ejemplo de funcionamiento de la primera iteración del Módulo de Captura y Procesamiento de Datos

Con esta sencilla secuencia se debería poder extraer, cada 0.5 segundos, el valor legible de la velocidad del motor del vehículo conectado. Al probarlo, se obtiene la siguiente respuesta:

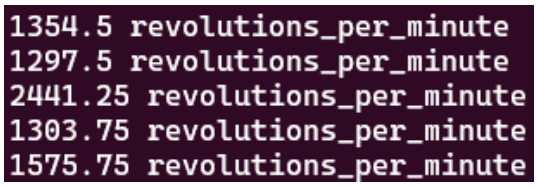


Figura 6.2: Salida del script del Código 6.1.

Se puede concluir entonces que el script funciona y el objetivo principal para esta iteración ha sido cumplido.

Por último, ya que según se va complicando el desarrollo se necesita mayor cantidad de pruebas y más extensas, se ha puesto en marcha un simulador de OBD2. Aunque no se extenderá la descripción porque no es relevante para el trabajo, permite habilitar un dispositivo ubicado en */dev* que simula comunicaciones a través de OBD2 con un automóvil. Esto es especialmente útil para poder probar las cosas en un entorno de desarrollo más cómodo.

A partir de este punto, los datos que se muestren serán del simulador, que devuelve respuestas con datos aleatorios, por lo que ya no se hablará más de la coherencia de los datos recibidos hasta el apartado de integración y el capítulo de pruebas.

6.1.3. Segunda iteración

Sabiendo como extraer datos del vehículo, el enfoque de la siguiente iteración es disponibilizar esos datos para el otro módulo. Es decir, publicarlos en un servidor dónde Prometheus-Server, el servicio del otro módulo que utilizará la interfaz publicada por este, pueda acceder. Esta iteración también debe obtener el mayor número de parámetros posibles del automóvil.

Es necesario en este punto hacer una distinción sobre el funcionamiento de Prometheus. Este servicio tiene una estructura "cliente-servidor", en el cual el cliente publica una página web con los datos actuales en un puerto definido de la máquina, y el servidor revisa según una frecuencia establecida esas páginas cliente para anotar los datos y así obtener una cronología. Por tanto, a partir de ahora, se hablará de Prometheus-Client para este módulo, y Prometheus-Server para el módulo de Presentación e Interfaz de Usuario.

De la integración a través de librería de Python que ofrece Prometheus-Client, interesa especialmente la función Gauge, ya que esta es la que permite publicar la información a la interfaz. El flujo que se deberá seguir es: Asegurar la conexión al vehículo, y para cada parámetro solicitarlo al vehículo y publicarlo inmediatamente en la interfaz.

Esto debería suplir la demanda de información en el periodo requerido. En el siguiente código fuente se puede ver, de manera simplificada, qué se ha añadido respecto a la anterior iteración.

```
1  import obd, time
2  from prometheus_client import start_http_server, Gauge
3
4  class Parameter():
5    def __init__(self, command):
6    self.command = command
7    self.query = None
8    self.metric = None
9    self.name = command.name.lower()
10
11
12   def update(self):
13     self.query = connection.query(self.command)
14
15     if self.query.value is None:
16       return
17
18     if isinstance(self.query.value, obd.Unit.Quantity):
19       if self.metric is None:
20         self.metric = Gauge(self.name, self.query.unit)
21         self.metric.set(self.query.value.magnitude)
22
23
24
25 if __name__ == '__main__':
26   connection = obd.OBD('/dev/pts/3')
27   start_http_server(8000)
28
29   metrics = {}
30
31   for command in connection.supported_commands:
32     metric = Parameter(command)
33     metrics[metric.name] = metric
34
35   while True:
36
37     for metric_name in metrics:
38     metrics[metric_name].update()
39
40   time.sleep(0.5)
```

Código 6.2: Ejemplo de funcionamiento de la segunda iteración del Módulo de Captura y Procesamiento de Datos

Ahora existe una clase Parámetro, que contiene todas las propiedades y funciones que se deben esperar de cada uno: El comando al que se refiere, la última respuesta del vehículo y la métrica de Prometheus-Client, que se crea y actualiza con la función update.

En el programa principal, se crea una lista de métricas, que, iterando la lista de comandos soportados según el tipo de protocolo de la conexión, crea las métricas y las añade a la lista. El bucle es parecido a la iteración anterior, con la diferencia que se recorre la lista de métricas y se actualiza cada una.

Esta solución es buena, pues cumple con los dos objetivos indicados para esta iteración, ya que soporta el mayor número de parámetros para cada protocolo y publica los resultados en la interfaz. No obstante, al probar el módulo se obtuvieron resultados alarmantes.

Al probarlo en el vehículo personal del alumno, sucedió que los parámetros tardaban excesivamente en actualizarse. Las medidas se tomaban cada, aproximadamente, 10 segundos, suceso que en el simulador no ocurría. Esto llevó a una investigación sobre el tiempo de respuesta del módulo que se resolverá en la siguiente sección.

6.1.4. Resolución de latencia

Antes de concluir el desarrollo del módulo con el último prototipo, es interesante hacer mención a lo mencionado en la conclusión de la última sección.

La diferencia de funcionamiento entre el simulador y el vehículo era tan notable que dificultaba mucho la utilidad del sistema en un entorno real. Se propusieron varios escenarios:

- La Unidad de Procesamiento (ordenador portátil personal en esta etapa del desarrollo) no tenía capacidad de procesamiento suficiente para procesar las solicitudes.
- El vehículo tenía algún problema con la comunicación.
- El adaptador ELM327 tenía problemas de funcionamiento.

La primera opción se descartó rápidamente, haciendo pruebas con ordenadores con más potencia y obteniendo resultados muy similares. Esta, aunque poco probable, hubiese tenido consecuencias considerables para la integración, pues el sistema debe funcionar en un ordenador compacto, como una Raspberry Pi.

Para la segunda opción, se estableció una batería de pruebas en distintos vehículos para estimar la posible causa de la latencia. El objetivo era determinar si el tipo de vehículo, año o protocolo podía causar un retardo en las comunicaciones. En esta se medía, para cada vehículo, el tiempo que tardaba en recorrer la lista entera de comandos, es decir, cuanto tardaba en actualizarse un parámetro. Cabe mencionar que los cuatro vehículos pertenecían a marcas diferentes, por ver si el funcionamiento interno de cada uno podía influir en la latencia, pero al no ser el caso, se ha omitido la información.

Los resultados de las pruebas son los siguientes:

Año	Protocolo	Periodo
2001	ISO 9141-2	10s
2003	KWP	4s
2004	KWP	4s
2008	CAN	<1s

Tabla 6.1: Pruebas de rendimiento en distintos vehículos

El primer vehículo, el personal del alumno, es el que impulsó estas pruebas. En él, los parámetros se actualizaban cada mucho tiempo, y pocos daban una respuesta aceptable comparado a la lista de soportados. Activando las opciones de registros, se podía observar como tardaba mucho en procesar cada petición, siendo esto desde que AutoSampler solicitaba el dato hasta que se publicaba en la interfaz.

El segundo y tercer vehículos ofrecían resultados similares, descartando que el chip o el módulo se comunicase mejor con ciertos grupos o marcas. En ambos, los parámetros tardaban prácticamente la mitad en actualizarse comparado al primero. No obstante, estos resultados seguían sin ser aceptables para el funcionamiento correcto y útil.

El cuarto vehículo otorgaba resultados esperanzadores. Aunque no funcionase tan rápido como el simulador, los datos que se veían permitían un uso efectivo del sistema. La diferencia es aún más llamativa si se tiene en cuenta que el cuarto vehículo soportaba una gran parte de comandos que el resto no, dado el protocolo por el que se conectaba a AutoSampler.

Según estas pruebas, se puede determinar que sobre todo el año del vehículo tiene un efecto claro en la rapidez o efectividad del módulo. Esto se puede extrapolar directamente al protocolo que usa cada uno, ya que según el año de fabricación requerirá que soporte un protocolo u otro. Para ver esto con claridad, es importante recuperar la información dada en el capítulo de Metodología sobre los tres protocolos principales usados en Europa:

- **ISO 9141-2:** Primer protocolo sobre pin K-Line. Encontrado desde la fecha de la regulación europea de OBD2 (A partir de 2001 para coches gasolina) hasta 2003.
- **ISO 13230-4:** KWP. Actualización directa sobre el anterior. Encontrado desde 2003 hasta la adopción de CAN sobre el 2008.
- **ISO 15765:** CAN. Soporta considerablemente más velocidad de transmisión y más parámetros. Encontrado desde, aproximadamente, 2006 hasta la actualidad.

Esto implica que el principal problema de latencia es basado en el protocolo de conexión, siendo el cuarto vehículo el único que soporta el bus CAN.

Pese a que los dos primeros protocolos listen oficialmente la misma velocidad de transmisión, se podría inferir también que el segundo y tercer vehículo soportan el protocolo KWP, mientras que el primero pertenece al pequeño grupo de vehículos que soportan comunicaciones estándar

por OBD2 sin soportar KWP, y la pérdida de velocidad extra en estos se puede deber a diversas razones, entre las que se incluye la propia optimización de la unidad ELM327 que se está usando.

De estas pruebas se puede concluir entonces que, para asegurar el funcionamiento correcto del sistema en todos los vehículos posibles, se va a requerir una extensa tarea de optimización y pruebas sobre el módulo.

Uno de los objetivos principales del proyecto es la flexibilidad que se debe ofrecer al usuario para la instalación, por lo que toda la optimización se deberá hacer sobre AutoSampler, y no sobre dispositivos externos como es el chip ELM327, que pueden variar en calidad y prestaciones; de forma que la gran mayoría de usuarios puedan utilizar el sistema con normalidad, independientemente del equipamiento del que dispongan.

6.1.5. Última iteración

La última iteración debe representar como se comportará el módulo en el sistema final. Toda la funcionalidad ha de quedar implementada y se debe optimizar el módulo teniendo en cuenta lo visto en la sección anterior. Por tanto, se marcan los siguientes objetivos para esta iteración:

- Optimizar el módulo para su correcto funcionamiento con los tres protocolos principales.
- Implementar la funcionalidad de aviso sobre averías.
- Implementar el módulo GPS.

Ya que en este caso los objetivos son muy distintos entre sí, se va a dividir en subsecciones y hablar de cada solución por separado.

Optimización del módulo

Para optimizar el módulo y asegurar su correcto funcionamiento, el factor más determinante es el número de parámetros. Visto que la latencia proviene del tiempo que tarda el adaptador en procesar una solicitud, se va a optimizar el número de solicitudes que debe procesar. Esto es una tarea relativamente compleja, pues no se quiere perder funcionalidad.

Lo primero es observar qué se está enviando realmente al vehículo. Como se habló en el capítulo de Metodología, los comandos OBD2 están compuestos por el modo y el PID. La lista de comandos soportados que ofrece la librería incluye todos los modos, entre los que se encuentran:

- Modo 01: Obtener un parámetro en tiempo real
- Modo 02: Modo de 'Freeze Frames'. Métricas de cuando sucedió la última avería.
- Modo 03: Obtener averías.

- Modo 04: Limpiar averías.

De todos estos y otros modos experimentales que se incluyen, realmente solo interesa enviar en bucle los comandos del primer modo, ya que, son los que reportan parámetros en tiempo real, además de querer evitar comportamientos extraños, como limpiar las averías en bucle. Por lo tanto, esa es la optimización más directa que se puede hacer sobre el prototipo. A la lista de comandos soportados, se podría añade lo siguiente:

```
1
2 for command in connection.supported_commands:
3   if str((command.command), encoding='utf-8').startswith('01'):
4     metric = Parameter(command)
5     metrics[metric.name] = metric
```

Código 6.3: Optimización de métricas de la última iteración del Módulo de Captura y Procesamiento de Datos

De este modo, solo se añadirán como Parámetros las métricas del modo 01.

Al probarlo, esta optimización resulta suficiente para los protocolos KWP y CAN, permitiendo que funcione con normalidad. Para el otro protocolo, dado que el tiempo de respuesta es lo suficientemente lento para que sea imposible tener un número aceptable de parámetros, se propone una idea más drástica:

```
1
2 metrics = {}
3 metrics[obd.commands['RPM'].name] = Parameter(obd.commands['RPM'])
4 metrics[obd.commands['SPEED'].name] = Parameter(obd.commands['SPEED'])
5 metrics[obd.commands['COOLANT_TEMP'].name] = Parameter(obd.commands['
    COOLANT_TEMP'])
6 metrics[obd.commands['THROTTLE_POS'].name] = Parameter(obd.commands['
    THROTTLE_POS'])
7 metrics[obd.commands['ELM_VOLTAGE'].name] = Parameter(obd.commands['
    ELM_VOLTAGE'])
```

Código 6.4: Versión reducida de métricas de la última iteración del Módulo de Captura y Procesamiento de Datos

Esta será otra versión del módulo, destinada a funcionar únicamente en vehículos del primer grupo, que soporta solo el protocolo ISO 9141-2, y recoge una serie de parámetros básicos de funcionamiento. De todos modos, el usuario que posea tiene la elección sobre qué parámetros monitorizar, e incluso qué versión de AutoSampler decide ejecutar, si prefiere un mayor número de parámetros frente a un tiempo de respuesta mínimo.

Implementación de averías

Otra de las funcionalidades pendientes para este módulo es la del aviso de averías del vehículo. Este componente debe encargarse de, al momento de iniciar el sistema, comprobar las averías del vehículo y disponibilizarlas en una interfaz para el otro módulo.

Las averías se solicitan al vehículo con el único comando del modo de funcionamiento 03. La estructura de las averías consiste de un código alfanumérico estándar, como puede ser "P0104", seguido de una descripción sobre el error. Esto presenta una problemática con el funcionamiento de Prometheus-Client, que solo soporta variables numéricas.

Por tanto, se requerirá una nueva interfaz que exponga estos datos al otro módulo. El código fuente referido a este componente se puede ver a continuación:

```
1 dtc_list = connection.query(obd.commands['GET_DTC'])
2
3 with open('/var/lib/grafana/dtc.csv', 'w') as csvfile:
4   csvfile.truncate()
5   csv_writer = csv.writer(csvfile)
6   csv_writer.writerow(['Codigo', 'Descripcion'])
7   csv_writer.writerows(dtc_list.value)
```

Código 6.5: Recogida de averías en la última iteración del Módulo de Captura y Procesamiento de Datos

Este código recoge las averías en una lista, abre un archivo colocado en el directorio de la plataforma Grafana del otro módulo, y reemplaza su contenido con las averías presentes en el vehículo. En la siguiente sección se verá como se tratan esos datos en el otro módulo.

Cabe mencionar que no se han podido realizar excesivas pruebas de este componente, pues en los vehículos disponibles al alumno para probar no se registraba ninguna avería, ni con AutoSampler ni con una máquina de diagnósticos más específica. No obstante, en los registros se ha podido ver que el comando funciona correctamente y no hay fallos de comunicación, por lo que se puede concluir que funciona.

Por último, es importante recalcar una función que no se ha implementado. En el apartado anterior se vio que al igual que existe un modo para comprobar las averías, existe otro para borrarlas del vehículo. Esta función, pese a que se pueda considerar interesante para ciertos casos de uso, se ha decidido no implementar.

Esto se debe a que va en contra de la filosofía del sistema. Desde un primer momento, AutoSampler se ha concebido como un sistema de información, dedicado a informar al usuario de ciertos parámetros y alertas sobre su vehículo, y como tal, no debe poder modificar ningún aspecto de este. Al igual que el propio vehículo no permite al usuario eliminar ninguna alerta, ya que está diseñado para que lo revise personal cualificado con cierto equipamiento, AutoSampler tampoco lo permite, y las alertas desaparecerán automáticamente cuando queden eliminadas del vehículo por otros medios.

Implementación de Módulo GPS

La implementación del módulo GPS es una parte muy importante del desarrollo del módulo, ya que además de ser una parte muy importante de los datos que pretende recoger AutoSampler, sienta un precedente sobre cómo deben funcionar los módulos en el sistema.

Se plantearon varios modos de funcionamiento para este componente, entre ellos, conectar el teléfono móvil del usuario al sistema y obtener de este su señal GPS, o aprovechar la conexión de una pantalla (típicamente un teléfono o una tablet con sistemas operativos Android/iOS) para extraer su posicionamiento a través de HTML. De estos se hablará en la implementación del otro módulo, Presentación e Interfaz de Usuario.

Finalmente se decidió por un módulo GPS estándar a través de USB, por su fiabilidad y coste. Para obtener los datos de este módulo, se ha usado un programa ya existente escrito en Python: *gpsd-exporter*. Este publica directamente en una interfaz de Prometheus-Client las métricas del servicio gpsd, programa que usa los datos del receptor USB para la geolocalización, por lo que las únicas tareas requeridas para este componente serán la configuración del servicio, y su inclusión en Prometheus-Server.

6.2. Presentación e Interfaz de Usuario

6.2.1. Introducción

La implementación de este módulo se corresponde con la segunda etapa del desarrollo, ya que ha sido empezado a ser desarrollado cuando el otro módulo estaba en fases avanzadas. Más concretamente, cuando se completó la segunda iteración de los prototipos.

Esta parte del proyecto es la menos exigente en cuanto a desarrollo de software, ya que se centra principalmente en diseño de interfaz, administración e integración de sistemas.

Al igual que se ha hecho con el otro módulo, se dividirá en iteraciones que tratan enfoques y funcionalidad distinta, con un componente evolutivo y un objetivo principal común de conseguir un sistema final funcional.

6.2.2. Primera iteración

La primera iteración de este módulo es una toma de contacto con la tecnología que se va a utilizar. Se centrará mayormente en la instalación local de la plataforma Grafana y el diseño de un tablero funcional que permita mostrar los parámetros.

El primer paso es instalar Grafana. Se trata de un proceso relativamente fácil y automático, a través de los siguientes comandos:

```
1 sudo apt-get install -y adduser libfontconfig1 musl
2 wget https://dl.grafana.com/enterprise/release/grafana-enterprise_11
    .0.0_arm64.deb
3 sudo dpkg -i grafana-enterprise_11.0.0_arm64.deb
```

Código 6.6: Proceso de Instalación de Grafana para arquitecturas ARM

Una vez instalado, se puede poner en marcha con *systemctl start grafana-server*. La plataforma se encuentra en el puerto 3000 de la máquina. Al acceder, se pide configurar unas credenciales para el acceso y luego te permite crear un tablero.

Estas credenciales se piden cada vez que se quiere acceder a la plataforma. Para este sistema, con acceso controlado y cuya información se guarda exclusivamente en el dispositivo, se va a proponer para próximas iteraciones eliminar el inicio de sesión, en favor de una mejor experiencia de usuario.

El próximo paso es crear un tablero que permita seguir los parámetros del otro módulo. Existen varias maneras de hacerlo, pero en esta fase del desarrollo se crearán paneles usando la intefaz gráfica. De todas las posibilidades que ofrece la plataforma, interesan principalmente tres tipos para esta iteración:

- **Gauge:** Visión en tiempo real de un parámetro.
- **Time Series:** Visión de rango de uno o varios parámetros.
- **Geomap:** Permite visualizar coordenadas GPS en un mapa.

Más concretamente para este proyecto, las *Gauges* se utilizarán para parámetros que típicamente se quieren ver en tiempo real, como es la velocidad instantánea, distintas temperaturas de operación del vehículo, o voltajes.

En cambio, las series temporales pueden resultar interesantes para parámetros que se quieran revisar a lo largo de un trayecto, como pueden ser por ejemplo la posición del acelerador y la pendiente del terreno, combinando así datos de los dos componentes.

En la siguiente figura se puede ver como podría quedar una versión preeliminar del tablero:

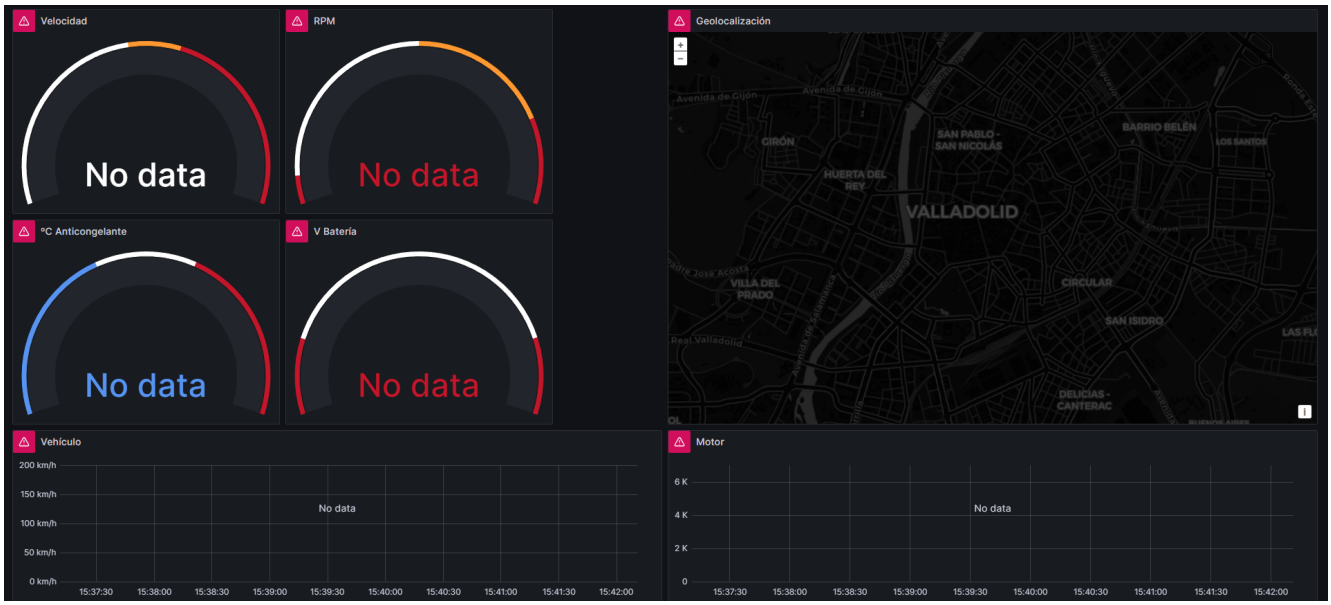


Figura 6.3: Vista del tablero de la primera iteración del módulo de Presentación e Interfaz de Usuario.

Con este tablero se puede concluir la fase de experimentación y así la primera iteración.

Se puede ver en las *Gauges* que se han configurado límites por colores, para poder comprobar de un vistazo si todo está funcionando como debería, por ejemplo, si la temperatura del motor se elevase más de lo previsto, o si se está superando un límite de velocidad establecido. Todos los paneles muestran una advertencia, pues todavía no se ha configurado ninguna fuente de datos, tarea pendiente para la segunda iteración.

6.2.3. Segunda iteración

Esta iteración se centra en conseguir un sistema funcional, que consiga extraer un parámetro del vehículo y mostrarlo en Grafana. Por lo tanto, se plantean dos objetivos:

- Instalar e integrar Prometheus-Server con Grafana.
- Conectar los dos módulos y probar el sistema.

El primer paso es descargar, extraer y ejecutar Prometheus-Server, información disponible en el Anexo. En este, se despliega un servidor que permite comprobar las métricas que observa. Al iniciarlo por primera vez, se obtiene lo siguiente:

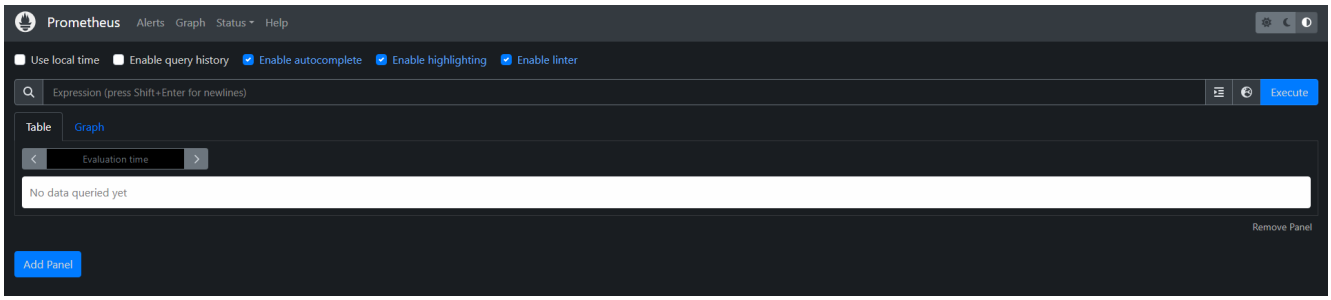


Figura 6.4: Vista de Prometheus-Server de la segunda iteración del módulo de Presentación e Interfaz de Usuario.

Una vez comprobado que funciona, se deben añadir las interfaces del otro módulo. Para ello, Prometheus-Server ofrece un archivo de configuración en el que se tendrán que añadir las siguientes líneas:

```
1 - job_name: "prometheus"
2
3   static_configs:
4     - targets:
5       -"localhost:8000"
6       -"localhost:9999"
```

Código 6.7: Configuración de clientes para Prometheus-Server.

De este modo, Prometheus-Server buscará en estas dos direcciones, correspondientes a los dos componentes del módulo de Captura y Procesamiento de Datos, en el puerto 8000 los datos del vehículo, y en el 9999 los datos del módulo GPS.

Para que Grafana pueda utilizar estos datos, hay que configurar este Prometheus-Server como fuente de datos de la plataforma. En las siguientes dos figuras se ve la configuración básica que se puede hacer para añadir la fuente de datos.

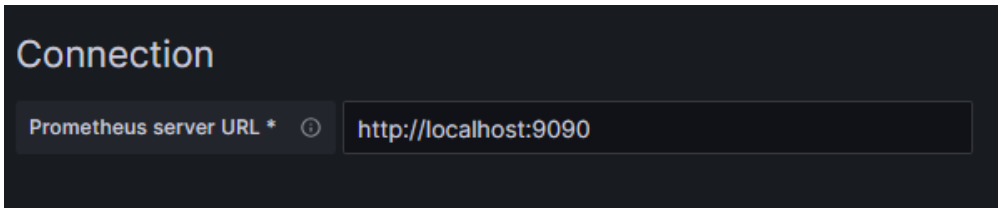


Figura 6.5: Configuración de dirección de Prometheus-Server en Grafana.

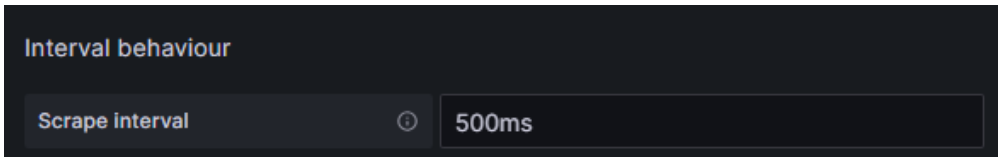


Figura 6.6: Configuración del intervalo de 'scraping' en Grafana.

Por último, se debe configurar también el intervalo en el archivo de Prometheus-Server modificado anteriormente, de la siguiente manera:

```
1   global:
2     scrape_interval: 1s
```

Código 6.8: Configuración de intervalo de 'scraping' para Prometheus-Server.

Con esto configurado, se debería poder obtener, en los paneles del tablero creado anteriormente, una opción para configurar los parámetros. Las opciones seleccionadas para este parámetro en concreto son medir la velocidad en un instante para mostrarlo en una *Gauge*.

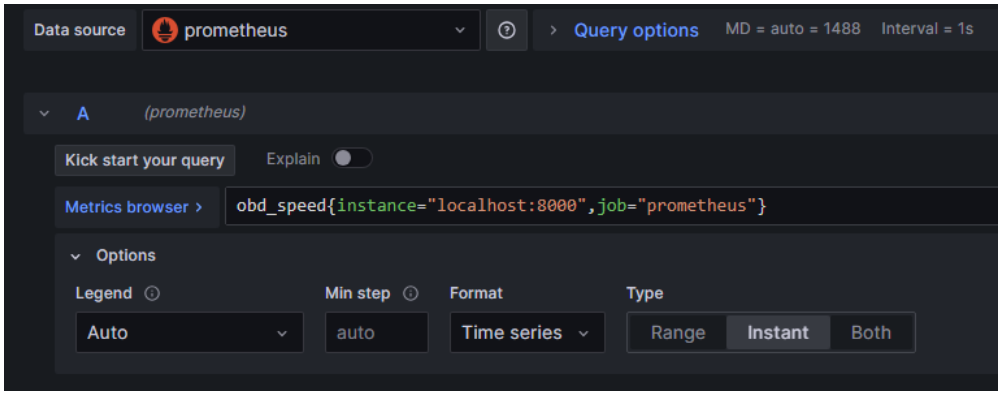


Figura 6.7: Configuración de parámetro en Grafana.

El tablero actualizado y funcional se puede ver en la siguiente figura:

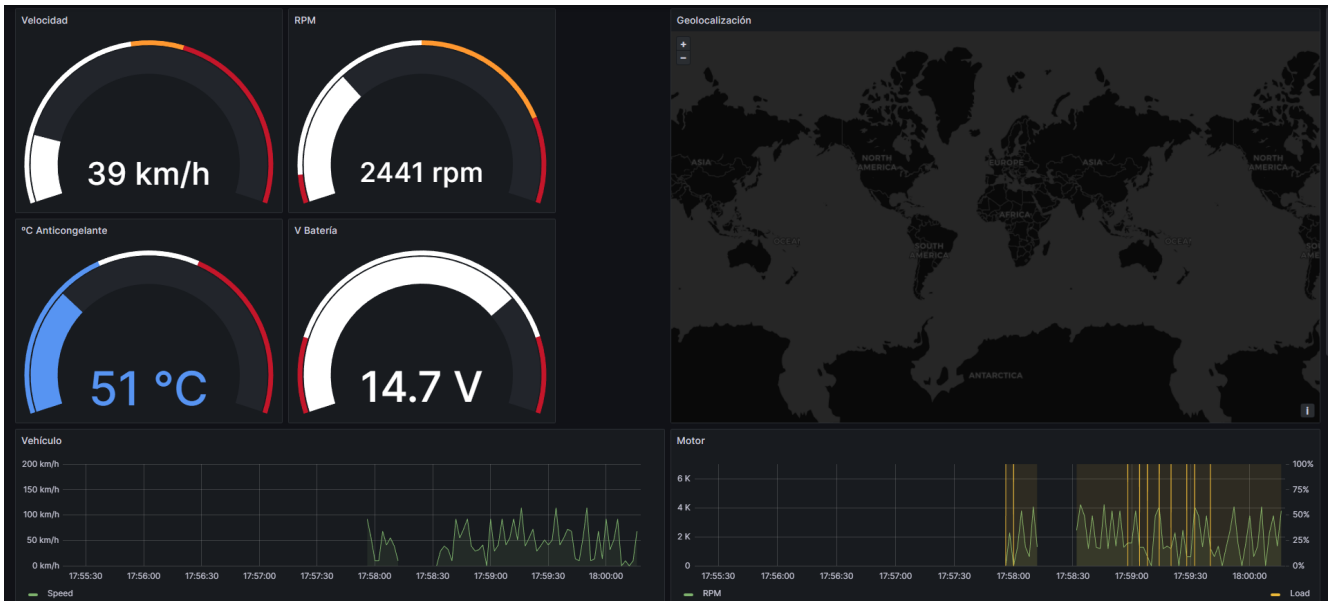


Figura 6.8: Tablero de la segunda iteración del módulo de Presentación e Interfaz de Usuario.

Nótese que no se tiene información sobre la geolocalización pues en esta fase del desarrollo las pruebas se hacen sobre el simulador comentado en la anterior sección.

6.2.4. Geolocalización en tiempo real

Antes de continuar con la última iteración, se debe hacer un inciso sobre la geolocalización. Se ha comentado en la sección anterior que se propusieron varias alternativas, entre ellas la más interesante fue utilizar el GPS de la pantalla a través del API de Geolocalización a través de HTML. Se va a desarrollar su implementación, ya que puede resultar interesante para otros usos, y justificar por qué finalmente no se utilizó.

La idea básica es aprovechar que el usuario debe acceder a través de un navegador, solicitar permiso a su ubicación si el dispositivo lo permite y mostrar en el panel *Geomap* la localización.

Para esto se necesitan tres componentes principales:

- Un plugin de Grafana que permita ejecutar JavaScript dentro de un panel.
- Código JavaScript y HTML que permita recoger la ubicación del usuario.
- Un servidor en la Unidad de Procesamiento que reciba las solicitudes y las publique en una interfaz Prometheus-Client.

El plugin que permite esta función se llama DynamicText. Este añade un nuevo tipo de panel a Grafana que permite ejecutar código de JavaScript. Para que funcione, hay que deshabilitar una opción del archivo de configuración de Grafana, ubicado en */etc/grafana/grafana.ini*, y añadir la siguiente línea:

```
1    disable_sanitize_html = true
```

Código 6.9: Configuración para habilitar JavaScript en Grafana.

Esto permite ejecutar código "no seguro"dentro de un panel HTML de Grafana. Con el plugin instalado y esta configuración hecha, se procede a incluir en un panel el siguiente código:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Geolocation Panel</title>
</head>
<body>
<div id="locationPanel">
<h2>User's Location</h2>
<p id="locationText">Fetching location...</p>
</div>

<script>
function showLocation(position) {
  var latitude = position.coords.latitude;
  var longitude = position.coords.longitude;
  var accuracy = position.coords.accuracy;

  var locationData = {
    latitude: latitude,
    longitude: longitude,
    accuracy: accuracy
  };

  fetch('http://localhost:5000/location', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify(locationData)
  })
  .then(response => {
    if (!response.ok) {
      throw new Error('Failed to save location');
    }
    console.log('Location sent successfully');
  })
  .catch(error => {
    console.error('Error sending location:', error);
```

```
41      });
42    }
43
44    function errorHandler(err) {
45      const locationText = document.getElementById('locationText');
46      locationText.textContent = `Error fetching location: ${err.message
            }`;
47    }
48
49    function getLocationAndSend() {
50      if (navigator.geolocation) {
51        navigator.geolocation.getCurrentPosition(showLocation,
              errorHandler);
52      } else {
53        const locationText = document.getElementById('locationText');
54        locationText.textContent = 'Geolocation is not supported by this
              browser.';
55      }
56    }
57
58    getLocationAndSend();
59
60    setInterval(getLocationAndSend, 1000);
61    </script>
62
63    </body>
64    </html>
```

Código 6.10: Código de geolocalización via HTML en Grafana.

Este es el encargado de solicitar la ubicación al dispositivo cliente, y enviarla cada segundo a un servidor localizado en el puerto 5000 de la máquina. Este servidor debe tratar las métricas y publicarlas en la interfaz. Para el servidor se ha elegido usar Flask, un micro web framework escrito en Python, suficiente para este propósito, y conveniente, ya que no requiere la instalación de ningún programa más en la máquina.

El código fuente de este servidor Flask es el siguiente:

```
1   from flask import Flask, request, jsonify
2   from flask_cors import CORS
3   from prometheus_client import start_http_server, Gauge
4   import json
5
6   app = Flask(__name__)
7   CORS(app)
8
9   latitude_metric = Gauge('latitude', 'Latitude␣of␣GPS␣location')
10  longitude_metric = Gauge('longitude', 'Longitude␣of␣GPS␣location')
```

```
11  accuracy_metric = Gauge('accuracy', 'Accuracy␣of␣GPS␣location')
12
13  @app.route('/location', methods=['POST'])
14  def save_location():
15    location = request.json
16    latitude = location['latitude']
17    longitude = location['longitude']
18    accuracy = location['accuracy']
19
20    latitude_metric.set(latitude)
21    longitude_metric.set(longitude)
22    accuracy_metric.set(accuracy)
23
24    print('Location␣saved:', location)
25    return 'Location␣saved␣successfully'
26
27  if __name__ == '__main__':
28
29  start_http_server(4000)
30
31  app.run(debug=False)
```

Código 6.11: Código de servidor Flask para geolocalización.

Este script funciona parecido a los enseñados en el módulo de Captura y Procesamiento de Datos. Crea tres *Gauges*, una para cada métrica, y cuando recibe el evento de acceso al servidor, actualiza la interfaz con los nuevos datos.

Esta solución es válida, y en las pruebas realizadas funcionaba correctamente, incluso en dispositivos que no tienen GPS pero sí internet (como puede ser un ordenador). Si se ha terminado eligiendo la opción del módulo GPS ha sido por dos razones principales:

- El módulo GPS reporta mucha más métricas además de la latitud y longitud, como pueden ser la pendiente, velocidad, o incluso los satélites a los que está conectado. En un sistema dedicado a la información, es de mucho valor recibir todas las métricas posibles.
- Esta solución presentada tiene la dependencia de tener una pantalla conectada. En otras palabras, si se tiene el sistema conectado sin una pantalla, solo recogiendo datos, los del geoposicionamiento no se van a guardar. Esto reduce los usos que tiene un componente de este estilo, ya que solo puedes ver dónde estas en tiempo real, frente a poder revisar todos los viajes en cualquier momento con el módulo GPS.

Esto y el bajo coste que tiene un módulo GPS como el mostrado en la Metodología para este proyecto, hace de este una opción muy robusta.

6.2.5. Última iteración

La última iteración de este módulo debe recoger toda la funcionalidad restante, ya que es la versión que se debe implementar en el sistema final. Como ya es habitual, se presentan los objetivos

para la iteración:

- Añadir la funcionalidad de visualización de averías
- Optimizar la visualización de métricas en tiempo real

En este caso, también se tratarán por separado.

Funcionalidad de averías

Una funcionalidad que no se había planteado todavía es la visualización de averías en un panel. Esto ha sido principalmente porque no se tuvo una solución viable hasta la última iteración del otro módulo. Sabiendo que las averías se escriben en un fichero *csv*, se deberá buscar un plugin que permita usarlos como fuente de datos.

Esta funcionalidad no existe de serie pues Grafana requiere que las fuentes de datos sean çonsultables", es decir, que se hagan consultas para acceder a los datos. Generalmente esto incluye bases de datos, y algún otro elemento, pero en ningún caso un fichero dentro de la máquina. Por esto, se ha elegido el plugin ÇSV", que permite añadir exactamente la funcionalidad buscada.

La instalación es muy sencilla, escribiendo este comando en la terminal:

```
1    grafana-cli plugins install marcusolsson-csv-datasource
```

Código 6.12: Instalación de plugin CSV para Grafana.

Para su funcionamiento, el plugin tiene dos requisitos: El archivo debe estar en un directorio al que pueda acceder Grafana, por eso se ha elegido la carpeta de datos de Grafana, ya que el otro módulo se ejecuta con permisos de superusuario y puede guardar el archivo en cualquier directorio; y añadir las siguientes líneas a la configuración:

```
1    [plugin.marcusolsson-csv-datasource]
2    allow_local_mode = true
```

Código 6.13: Configuración de plugin CSV para Grafana.

Esta configuración permite cargar archivos locales, ya que por defecto solo soporta archivos en la nube.

Una vez instalado y configurado, se puede añadir como fuente de datos, similarmente a cómo se ha hecho en la segunda iteración con Prometheus-Server. Para visualizar los datos, se puede crear un sencillo panel con una Tabla y esta fuente de datos, de la siguiente manera:

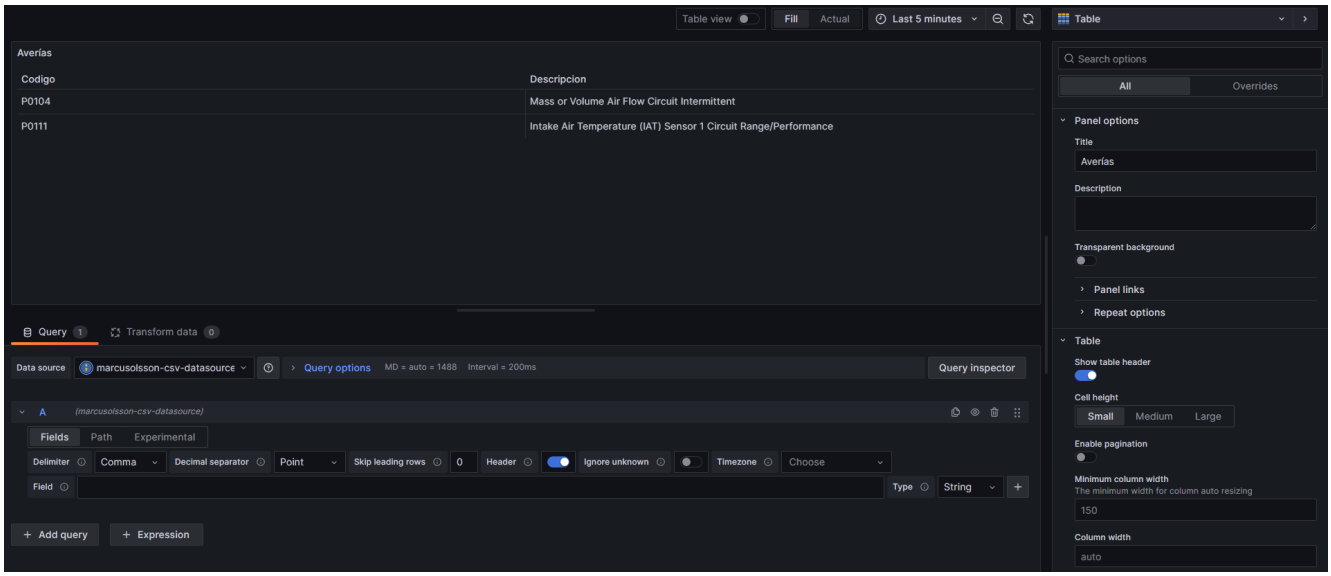


Figura 6.9: Creación de panel de averías en Grafana.

Por lo tanto, la funcionalidad ha sido implementada correctamente, y se puede acceder a los datos sobre las averías de manera rápida y cómoda, tal como se esperaba en la fase de Planificación del Sistema.

Optimización de tableros

Una tarea pendiente hasta este punto es modificar las opciones de refresco que Grafana permite por defecto. Ya que esta plataforma no está tan pensada para ver métricas en tiempo real, existen algunas limitaciones en cuanto a la actualización de los tableros.

En la segunda iteración se vio como Prometheus-Server está configurado para "scrapear"(anotar las métricas vistas en las interfaces) cada 1s. Grafana tiene acceso para consultar esta base de datos, pero por defecto solo actualizará las métricas cada 5 segundos como mínimo. Esto no es suficiente para el visionado de ciertas métricas en tiempo real, por lo tanto se va a modificar.

Si se intenta realizar el caso de uso de modificar la frecuencia de actualización y se introduce un valor inferior a 5 segundos, va a resultar en un error, pues es el límite puesto por defecto. Para modificarlo, se puede editar la siguiente línea del archivo de configuración:

```
1    min_refresh_interval = 5s
```

Código 6.14: Configuración del mínimo intervalo de actualización de Grafana.

Simplemente cambiando esta variable a un valor suficientemente bajo, véase 100ms o 200ms, se puede configurar cualquier rango de actualización.

Antes de terminar esta sección, es necesario anotar comportamientos vistos haciendo pruebas. Pese a que se puedan configurar intervalos de actualización muy pequeños, no es la opción más óptima en la mayoría de los casos.

En el caso del simulador, se ha conseguido hacer funcionar con intervalos realmente bajos, y poder ver las métricas con mucha más tasa de refresco. En cambio, como se ha discutido antes, en el sistema real tanto el número de parámetros configurados como el tipo de protocolo pueden hacer que el sistema sufra de latencia.

En el mejor de los casos, un vehículo que soporte CAN, y el módulo de Captura funcionando en su versión completa, las pruebas realizadas demostraron que es más estable tener una tasa de refresco de un segundo. Cuando se probó con 500ms, en diversas ocasiones Prometheus-Server anotaba los datos algo antes de que se modificaran en la interfaz, consiguendo que tarde otros 500ms en actualizarse. Este comportamiento llevaba a una experiencia de usuario algo confusa, ya que los datos a veces tardaban algo más, y la observación en tiempo real no era estable.

En última instancia, esto siempre depende de los requisitos temporales del usuario, número de parámetros observados, tipo de vehículo, etc. Por lo tanto, el valor recomendado será de 1s, pero abierto a que sea el propio usuario el que encuentre la tasa que más se ajuste a sus necesidades.

Estas dos tareas completan los objetivos marcados para la última iteración. Por tanto, se puede concluir el desarrollo de este módulo. A continuación, se van a presentar algunas figuras detallando el funcionamiento final del módulo, así como alguna opción de personalización que ofrece la plataforma:

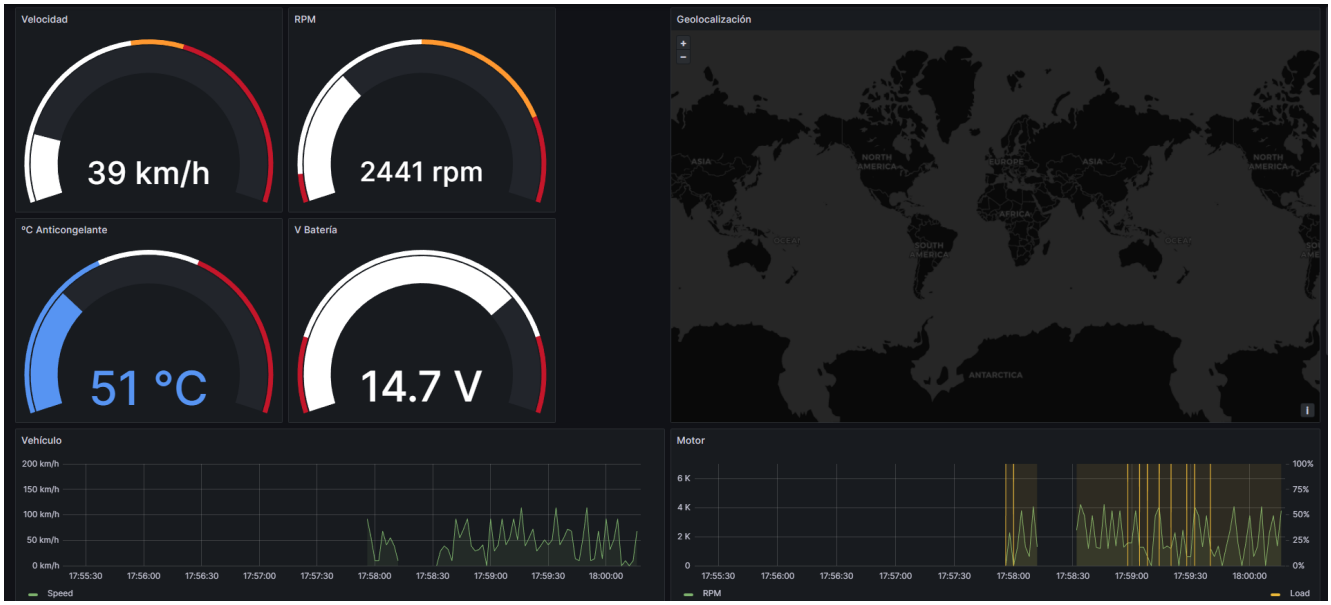


Figura 6.10: Primera sugerencia de presentación del módulo de Presentación e Interfaz de Usuario.

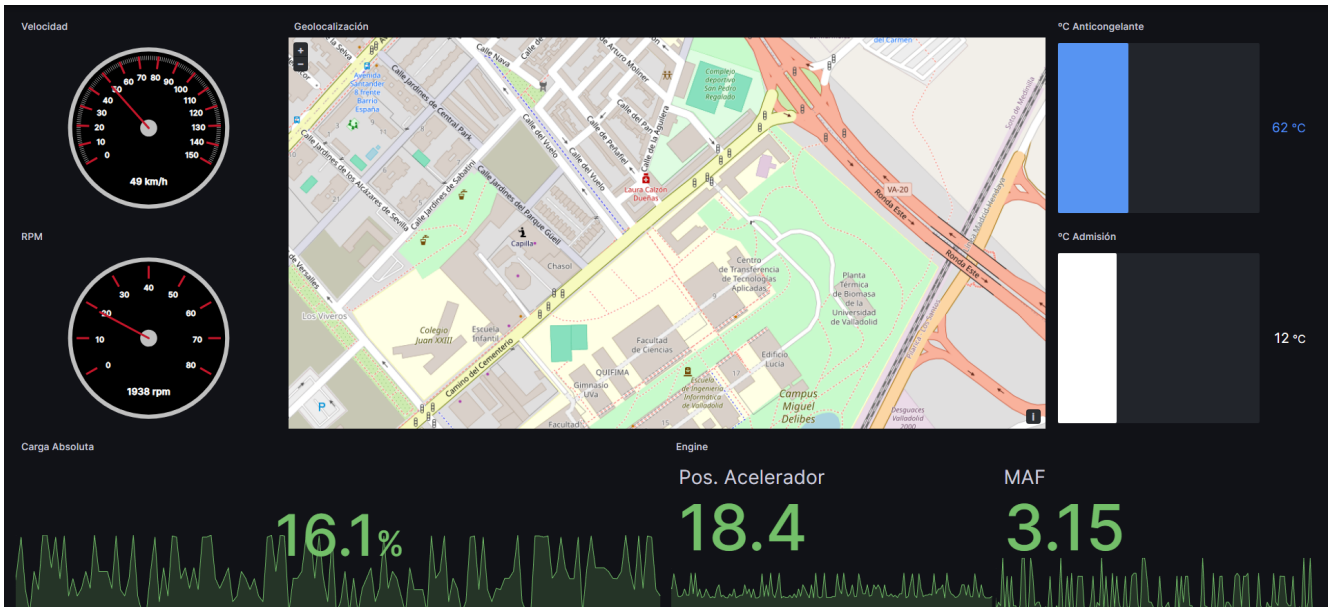


Figura 6.11: Segunda sugerencia de presentación del módulo de Presentación e Interfaz de Usuario.

Figura 6.12: Vista de averías del módulo de Presentación e Interfaz de Usuario.

6.3. Integración de los módulos

6.3.1. Introducción

En esta sección se tratará la integración de los dos módulos y su implementación en un sistema empotrado que cumpla con las especificaciones propuestas en los requisitos.

Uno de los objetivos principales del proyecto es la flexibilidad tanto de uso como de instalación, ya que se propone que pueda ser desplegado en máquinas muy diversas. No obstante, las tareas de instalación, configuración y optimización de esta sección van a ser justificadas y explicadas para la lista de materiales explicada en la Metodología, es decir, una Raspberry Pi 3 Model B y sus módulos.

Esto se va a hacer por enfocar el contenido e intentar ser lo más preciso posible, puesto que la implementación de estas tareas puede variar considerablemente dependiendo del sistema operativo, arquitectura o características.

Esta sección se dividirá en tres subsecciones. La primera estará centrada en la instalación y configuración del sistema operativo que se utiliza en el sistema final, la segunda en la puesta en marcha de los módulos de AutoSampler, y la tercera será la automatización de procesos.

6.3.2. Sistema Operativo

La elección de un sistema operativo para este proyecto es una tarea compleja, pues requiere un equilibrio de funcionaliad, ya que debe tener unos servicios mínimos para poder desplegar AutoSampler; eficiencia, no debe tener muchos procesos pues se va a requerir la potencia de procesamiento de la Raspberry para el proyecto; y facilidad de instalación, de cara a ser instalado por usuarios con menos experiencia técnica.

Finalmente se ha optado por utilizar un sistema llamado *DietPi*. Se ha elegido por tres razones principales:

- Es una instalación mínima de Linux basada en Debian, que mantiene un uso muy moderado de recursos: CPU, RAM...
- Ofrece versiones fáciles de instalar para la mayoría de sistemas ARM: Raspberry Pi, Orange Pi, Odroid, PINE, etc., además de contar con una versión para sistemas x86.
- Facilita, a través de programas propios, tareas de administración específicas para sistemas empotrados.

En el caso de la Raspberry Pi, se puede instalar fácilmente en una tarjeta SD a través de Raspberry Pi Imager, como se puede ver en el manual de instalación indicado en el Anexo. La configuración inicial es bastante sencilla, y no requiere ninguna acción especial.

Una vez el sistema operativo está instalado y funcionando, hay dos funciones que deben modificarse para este proyecto.

Conectividad

La primera es la conectividad. Se ha mencionado antes que se iba a dar soporte a dos modos de funcionamiento en lo que conexiones de pantallas se refiere.

El primer modo, modo de uso recomendado para la mejor experiencia de usuario, es dotar de internet a AutoSampler cuando se quieran revisar los datos. Es decir, a través de un punto de acceso en el teléfono móvil del usuario. En este caso, se debería de configurar el sistema operativo de la siguiente manera:

Desde la línea de comandos, se ejecuta *dietpi-config*. Esto mostrará al usuario un menú visual de configuración. Después, se debe entrar a la opción *7: Network Options: Adapters*.

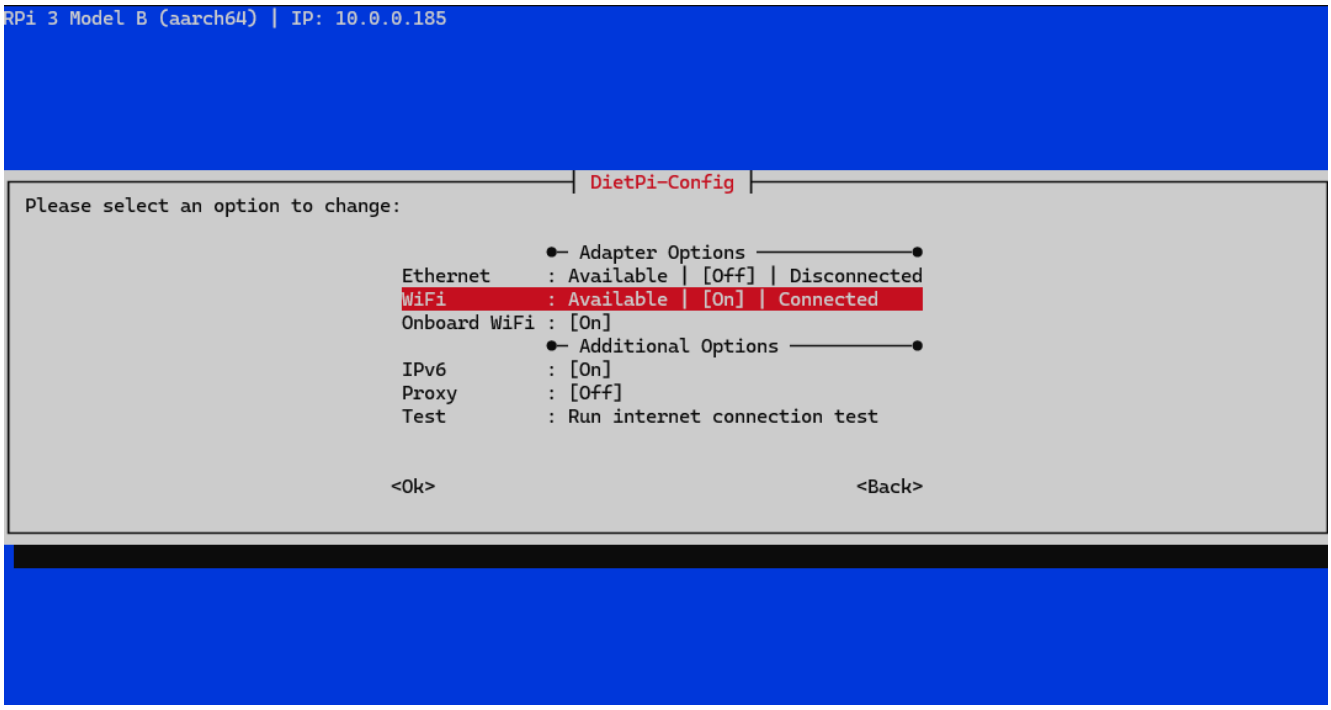


Figura 6.13: Configuración de adaptadores de red en Raspberry Pi (DietPi).

De esta, se debe asegurar que la opción *Onboard WiFi* esté habilitada. Después, se debe entrar al menú WiFi para configurar la conexión. En este menú sencillo se deberán de configurar las interfaces WiFi pertinentes para que AutoSampler se pueda conectar automáticamente a la red.

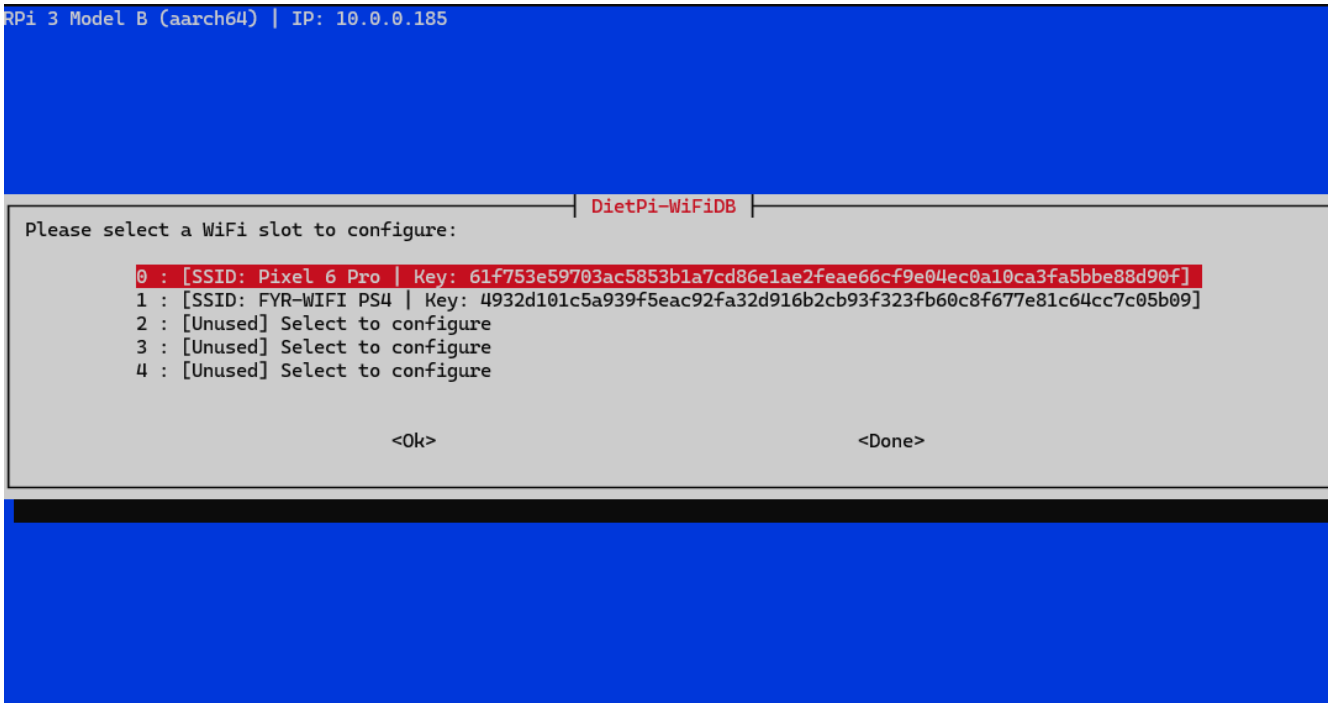


Figura 6.14: Configuración de interfaces WiFi en Raspberry Pi (DietPi).

Una vez configurado de esta forma, el sistema se intentará conectar, en el orden de esta lista, a las redes. Es interesante también activar la opción *Auto-Reconnect* vista en la Figura 6.14, habilitando poder reconectarse a una red si se pierde la conexión.

El segundo modo de funcionamiento es crear un punto de acceso en la Raspberry Pi, y directamente conectar la pantalla a esta. Este proceso también es sencillo, ya que lo único que se debe hacer es descargar el módulo *WiFi Hotspot* de la librería de módulos de DietPi.

Para ello, se debe ejecutar el comando *dietpi-software*, y encontrar el módulo en su opción de búsqueda.

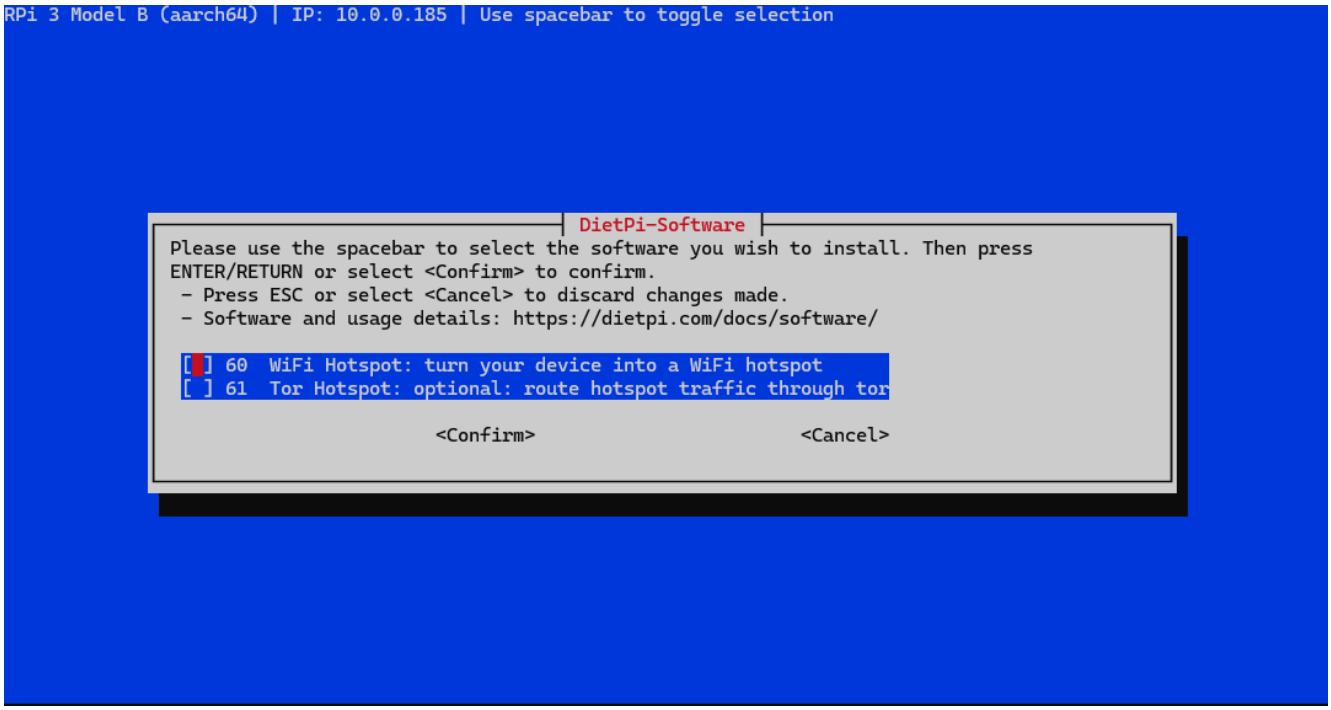


Figura 6.15: Instalación de módulo de punto de acceso en Raspberry Pi (DietPi).

Una vez instalado, se lanzará automáticamente el punto de acceso con los ajustes por defecto, y sobreescribirá las opciones de adaptador de la Figura 6.14 con las opciones del punto de acceso.

De los dos modos presentados, se recomienda el uso del primero. La razón principal es tener acceso a internet cuando se revisan las métricas, así módulos como el *Geomap* pueden renderizar el mapa, que en ningún caso está descargado en el almacenamiento de AutoSampler. La otra razón es, en el caso de que el usuario utilize su teléfono móvil como pantalla, al conectarse al punto de acceso no solo perderá acceso a internet, si no que también se cortarán conexiones habituales en vehículos actuales, como puede ser Apple Carplay/Android Auto inalámbrico, ya que estos sistemas también requieren conectar el teléfono a un punto de acceso.

En las pruebas que se han hecho, la opción de conectar AutoSampler al punto de acceso del móvil ha resultado realmente cómoda, ya que permite visualizar las métricas con normalidad, así utilizar estos sistemas mencionados sin ningún problema. El único punto negativo es el ligero incremento en el uso de la batería del teléfono, pero al solo tener que activarlo cuando se quieren

revisar las métricas, no resulta un gran inconveniente.

Inicio de sesión

La otra configuración principal que se va a realizar sobre la máquina es activar el inicio de sesión automático. Esto se va a hacer para facilitar la tarea de automatización que se verá posteriormente. Esta función permite que, al arrancar el sistema, inicie sesión automáticamente, pero no quitar la contraseña, es decir, que por ejemplo para acceder por SSH se necesitará la contraseña del usuario.

Esto se puede hacer en la configuración, accediendo de nuevo a través de *dietpi-config*. En este, se debe acceder a la opción *9: AutoStart Options*.

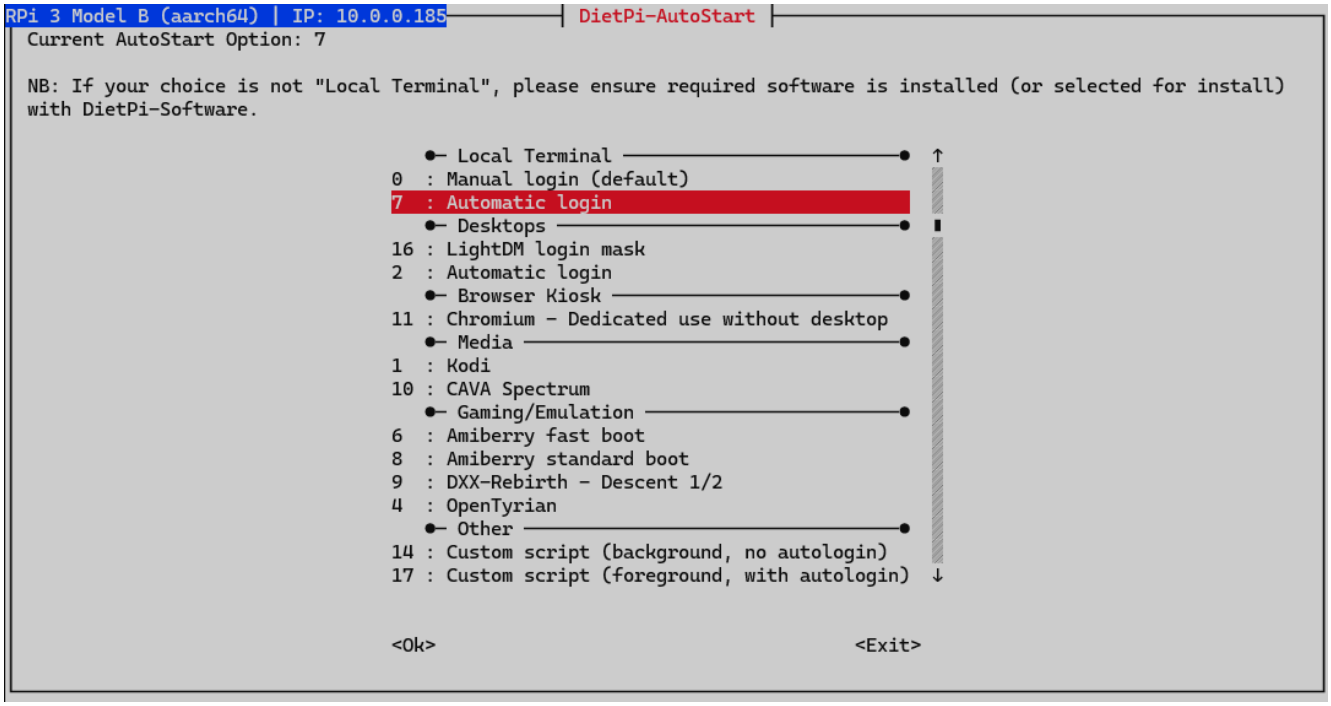


Figura 6.16: Configuración de inicio automático en Raspberry Pi (DietPi).

La opción que se debe activar es *Automatic Login* en la sección *Local Terminal*.

6.3.3. Instalación de los módulos

En esta subsección se tratará la instalación y puesta en marcha de los servicios requeridos para que funcione AutoSampler. No se va a hacer un manual de instalación completo pues esa información se puede encontrar en el Anexo, solo se va a detallar los pasos sobre como se han implementado los módulos en el sistema.

La primera tarea es instalar un gestor de paquetes y entornos Python, dado que el módulo de Captura y Procesamiento de Datos está diseñado para funcionar con Python 3.8. Para la Raspberry Pi, se ha decidido instalar Miniforge3, una instalación mínima de Conda enfocada a funcionar en

distintas arquitecturas. Una vez instalado, se crea y activa el entorno que se va a utilizar para AutoSampler:

```
1    conda create -n obd python=3.8
2    conda activate obd
```

Código 6.15: Creación de entorno Python en Conda.

Con el paquete instalado, se pueden pasar a la máquina los diferentes archivos Python, instalar las dependencias y probar el módulo.

Para el módulo GPS, se debe instalar *gpsd* y el programa Python *gpsd-exporter*. El servicio gps se pueden instalar fácilmente a través de *apt*, y análogamente, el programa Python a través de *pip*:

```
1    apt install cgps
2    pip install gpsd_prometheus_exporter
```

Código 6.16: Instalación de módulo GPS.

Por último, se debe configurar *gpsd* para que encuentre el dispositivo. En el caso del receptor USB, siempre estará ubicado en el dispositivo *ttyACM0*, por lo que, en el fichero */etc/default/gpsd* se modificará la siguiente línea:

```
1    DEVICE = "/dev/ttyACM0"
```

Código 6.17: Configuración de servicio GPS en Raspberry Pi.

Con todo configurado, se puede lanzar el proceso de recolección de datos GPS con el siguiente comando, donde -p indica el puerto de la interfaz, y -s la unidad de la velocidad GPS:

```
1    gpsd-exporter -p 9999 -s kph
```

Código 6.18: Configuración de servicio GPS en Raspberry Pi.

El módulo de Presentación e Interfaz de Usuario se instala igual que se ha descrito en la sección anterior, descargando los ficheros para la arquitectura correcta y realizando las configuraciones pertinentes. Para copiar el tablero, se puede exportar en formato JSON e importarlo en la Raspberry.

6.3.4. Tareas de automatización

La tarea pendiente para poder probar el sistema final es automatizar los servicios. En otras palabras, para poder conectarlo al automóvil y que funcione directamente, los módulos deben iniciarse automáticamente.

Para ello, se ha creado un script bash que inicializa los módulos cuando se inicia sesión, que como se ha mencionado antes, es un proceso automático en este sistema. El script se puede encontrar en el siguiente código fuente:

```
1  #!/bin/bash
2  eval "$(conda␣shell.bash␣hook)"
3  conda activate obd
4  cgps &
5  sleep 5
6  systemctl start grafana-server
7  python py/autosampler-obd.py &
8  gpsd-exporter -b 0.0.0.0 -p 9999 -s kph &
9  ./prometheus/prometheus --config.file=prometheus/prometheus.yml &
10 wait
```

Código 6.19: Script de automatización de AutoSampler.

Este script se debe incluir al final del fichero *.bashrc* en la carpeta root del sistema. Se podría iniciar en segundo plano incluyendo el script en los ficheros de inicio, contenidos en el directorio */etc/rc.d*, pero para asegurar el funcionamiento correcto de Conda se ha decidido hacer una vez iniciada sesión.

Capítulo 7

Pruebas

7.1. Pruebas realizadas

Dado el desarrollo basado en prototipos que se ha llevado a cabo para este proyecto, la mayoría de pruebas o comprobaciones que prueban el comporamiento correcto del sistema se han hecho mientras se desarrollaba. A pesar de ello, las pruebas que se van a presentar en este capítulo comprueban una vez terminada la implementación, sobre el sistema final, si el comportamiento del sistema es el esperado frente a las acciones propuestas.

7.2. Consideraciones

Antes de presentar las pruebas, es necesario considerar dos aspectos importantes sobre estas.

Pese a que el resultado de las pruebas es positivo, y se cree que el sistema funcione en la mayoría de situaciones, solo se ha podido probar en un entorno y con unas condiciones determinadas, es decir, con un abanico pequeño de vehículos, pantallas, hardware, etc. Por lo tanto, puede que el resultado de las pruebas cambie en un entorno de prueba más desfavorable o desafiante para el sistema.

Por otro lado, esta batería de pruebas representa el comportamiento completo del sistema ante varias situaciones, pero en ningún caso representa el orden cronológico de desarrollo o de uso. Algunas pruebas se han extraído directamente de casos de uso, mientras otras son pruebas de robustez del sistema y su cumplimiento de los requisitos presentados.

7.3. Batería de pruebas

Prueba 1	**Inicio automático**
Objetivo	Todos los servicios se inician automáticamente una vez conectado el sistema.
Actor	Usuario
Precondición	El usuario ha conectado AutoSampler.
Acción esperada	El sistema se inicia y comienza a registrar datos.
Resultado	**Positivo**

Tabla 7.1: Prueba 1 de funcionamiento general del sistema.

Prueba 2	**Conexión de pantalla**
Objetivo	El usuario puede acceder a la información de AutoSampler a través de un navegador web.
Actor	Usuario
Precondiciones	El usuario ha conectado AutoSampler y ha escrito la dirección de AutoSampler en el navegador.
Acción esperada	El sistema muestra en la dirección y puerto especificados la plataforma Grafana.
Resultado	**Positivo**

Tabla 7.2: Prueba 2 de funcionamiento general del sistema.

Prueba 3	**Visualización de parámetros**
Objetivo	El usuario puede visualizar los parámetros configurados entrando al tablero
Actor	Usuario
Precondiciones	El usuario ha conectado AutoSampler y arrancado el vehículo.
Acción esperada	El sistema muestra los parámetros del vehículo en tiempo real sobre los paneles.
Resultado	**Positivo**

Tabla 7.3: Prueba 3 de funcionamiento general del sistema.

Prueba 4	**Listado de parámetros**
Objetivo	El usuario puede visualizar todas las métricas configuradas de todos los módulos en la creación de paneles.
Actor	Usuario
Precondiciones	El usuario ha conectado AutoSampler y arrancado el vehículo.
Acción esperada	El sistema muestra todos los parámetros compatibles, además de las variables GPS
Resultado	**Positivo**

Tabla 7.4: Prueba 1 de interacción entre módulos.

Prueba 5	**Integridad de parámetros de vehículo**
Objetivo	El usuario puede comprobar que los datos que reporta el sistema se corresponden con las acciones tomadas.
Actor	Usuario
Precondición	El usuario ha tomado alguna acción en el vehículo (por ejemplo, alcanzar una velocidad de 30 km/h).
Acción esperada	El dato que muestra el sistema se corresponde con la acción tomada.
Resultado	**Positivo**

Tabla 7.5: Prueba 1 de Captura y Procesamiento de Datos.

Prueba 6	**Integridad de Geolocalización**
Objetivo	El usuario puede comprobar que los datos del módulo GPS se actualizan correctamente.
Actor	Usuario
Precondición	El usuario ha movido el vehículo de posición.
Acción esperada	El sistema registra el viaje y la nueva posición.
Resultado	**Positivo**

Tabla 7.6: Prueba 2 de Captura y Procesamiento de Datos.

Prueba 7	**Capacidad de respuesta**
Objetivo	El usuario puede ver datos nuevos cada segundo.
Actor	Usuario
Precondición	El usuario ha arrancado el vehículo.
Acción esperada	El sistema registra el viaje y la nueva posición.
Resultado	**Positivo**

Tabla 7.7: Prueba 4 de funcionamiento general del sistema.

Prueba 8	**Múltiples conexiones**
Objetivo	El usuario puede acceder simultáneamente al sistema con varias pantallas.
Actor	Usuario
Precondiciones	El usuario ha conectado AutoSampler y accedido al servidor con más de un dispositivo.
Acción esperada	El sistema soprota las peticiones y funciona con normalidad en todas las pantallas.
Resultado	**Positivo**

Tabla 7.8: Prueba 1 de Presentación e Interfaz de Usuario.

Prueba 9	**Rango de visión**
Objetivo	El usuario puede consultar los datos de viaje de los 15 días anteriores.
Actor	Usuario
Precondiciones	El usuario ha registrado datos en los últimos 15 días y selecciona la visión de un día concreto.
Acción esperada	El sistema muestra los datos de viaje del día seleccionado.
Resultado	**Positivo**

Tabla 7.9: Prueba 2 de Presentación e Interfaz de Usuario.

Prueba 10	**Exportación de datos**
Objetivo	El usuario puede exportar los datos registrados en el sistema.
Actor	Usuario
Precondiciones	El usuario ha registrado datos en los últimos 15 días, selecciona un rango de visión y un panel y selecciona exportar los datos.
Acción esperada	El sistema exporta los datos del panel seleccionado en el rango específicado.
Resultado	**Positivo**

Tabla 7.10: Prueba 3 de Presentación e Interfaz de Usuario.

Capítulo 8

Conclusiones

En este capítulo se muestran los objetivos cumplidos, las conclusiones extraídas, las líneas de trabajo futuro y la valoración personal del presente Trabajo de Fin de Grado.

8.1. Objetivos cumplidos

Para poder interpretar los resultados y conclusiones de este proyecto, se deben comparar con los objetivos principales propuestos.

En líneas generales, se puede concluir que se han cumplido los objetivos propuestos para el proyecto. Se ha creado un prototipo funcional que cumple con los objetivos y requisitos comentados, es decir, se ha construido un sistema autosuficiente que es capaz de recoger una alta cantidad de parámetros de funcionamiento de cualquier vehículo al que se conecte, almacenarlos, y mostrarlos al usuario de manera eficiente y cómoda. La implementación de este sistema ha sido validada a través de pruebas que verifican el funcionamiento y la efectividad de este.

8.2. Características del sistema final

Además de los objetivos cumplidos, es importante comentar las características técnicas del sistema final:

- **Rendimiento:** Teniendo en cuenta las consideraciones del capítulo de Implementación, el sistema consigue evaluar y mostrar las métricas de los diferentes módulos (vehículo, GPS, etc.) cada segundo, independientemente del año o protocolo del vehículo.
- **Escalabilidad:** El sistema está preparado para aceptar información de otros sensores (como sensores externos del vehículo) y otras fuentes de datos. La información referida a añadir un nuevo módulo se encuentra en el Apéndice.
- **Extensibilidad:** La plataforma Grafana otorga al sistema capacidades altas de extensibilidad, ofreciendo la construcción de plugins o uso de existentes que aumenten o extiendan la funcionalidad del sistema. Se podría construir, por poner un ejemplo concreto, un plugin que envíe la geolocalización a un API de algún proovedor de mapas y pinte sobre el mapa existente puntos de interés (por ejemplo, gasolineras) cercanos.

- **Mantenibilidad:** Las plataformas mencionadas, así como las librerías de Python usadas están en constante evolución y soporte, y el sistema será compatible con nuevas versiones de estas. No obstante, el sistema ha sido diseñado para un bajo consumo tanto eléctrico como de red, por lo que no 'requiere' mantenimiento y los paquetes no se actualizarán automáticamente.
- **Seguridad:** Los datos del sistema pueden ser expuestos a usuarios que tengan acceso al punto de acceso WiFi al que se conecta AutoSampler, o a personas que tengan acceso al dispositivo físico. En las líneas de trabajo futuras que se presentarán, se tratará como mejorar la seguridad.

8.3. Conclusiones extraídas

Del cumplimiento de los objetivos propuestos y la construcción del sistema se pueden extraer dos conclusiones principales con matices diferentes:

Por un lado, la conclusión inmediata de lo comentado es la sensación de entrega de un sistema funcional, finalizado y pulido, que no solo cumple con los requisitos propuestos si no que cumple la visión inicial que se tenía del sistema. Esto indica que no se han tenido que hacer cambios significativos en el diseño o la implementación, y se han satisfecho todas las incógnitas que se tenían inicialmente de una forma buena.

Por otro lado, del trabajo realizado también se puede extraer el aprendizaje que ha supuesto en términos de materias desconocidas para el alumno hasta la realización de este. Tanto el funcionamiento a bajo nivel de los protocolos trabajados, tratamiento y entendimiento de los datos, como destreza en administración de sistemas, instalación de servicios y conocimiento general de sistemas empotrados, ventajas y limitaciones.

8.4. Líneas de trabajo futuro

El prototipo funcional que este proyecto presenta sienta las bases de una plataforma funcional y eficiente para los usos que se han descrito. No obstante, esta base da lugar a líneas de trabajo que plantean mejoras u objetivos a mayores para este.

En primer lugar, se podrían buscar formas de optimizar la recogida de datos. Ya bien modificar u optimizar la librería de Python para que se ajuste más a los propósitos de este proyecto, o a más bajo nivel controlar las solicitudes en un programa que sea capaz de extraer datos más eficientemente. Esto ayudaría al tiempo de respuesta de los datos, dotando al sistema de usos añadidos.

En segundo lugar, se podría crear una plataforma propia para la visión de los datos. En el proyecto se ha usado Grafana, dado que para el tiempo del que se disponía era una opción muy sólida que permitía cumplir todos los requisitos propuestos; pero se podría crear una plataforma específica, con menos funcionalidad, más sencilla de usar y que esté centrada en la eficiencia de los datos.

En segundo lugar, se podría reforzar la seguridad del sistema. Dadas las configuraciones realizadas sobre el sistema operativo, cualquier individuo que tenga acceso físico al dispositivo

tendrá también acceso a los datos. Se podría cambiar este comportamiento, construyendo otro sistema de automatización que permita securizar los datos y lanzar los servicios.

En último lugar, un ámbito del proyecto que no se ha tratado es la instalación física. Pese a que se sale de las competencias típicas que se aprenden en este grado y que se pretende aprender en el presente trabajo, puede ser interesante tratar de mejorar la instalación física mediante soportes, moldes, conectores personalizados, etc.

Bibliografía

[1] Wikipedia, *OBD-II PIDs*, en *Wikipedia*, 4 de mayo de 2024. URL: https://en.wikipedia.org/w/index.php?title=OBD-II_PIDs&oldid=1222175766 (visitado 01-06-2024).

[2] Lenny Delligatti, *SysML Distilled: A Brief Guide to the Systems Modeling Language*. Upper Saddle River, NJ Munich, 8 de nov. de 2013, 304 págs., ISBN: 978-0-321-92786-6. (visitado 25-05-2024).

[3] Wikipedia, *Use case*, en *Wikipedia*, 19 de abr. de 2024. URL: https://en.wikipedia.org/w/index.php?title=Use_case&oldid=1219721783 (visitado 06-06-2024).

[4] CSS Electronics. "OBD2 explained - a simple intro," CSS Electronics. (2023), URL: https://www.csselectronics.com/pages/obd2-explained-simple-intro (visitado 30-05-2024).

[5] The Raspberry Pi Foundation. "Raspberry pi documentation." (2024), URL: https://www.raspberrypi.com/documentation/ (visitado 18-06-2024).

[6] Python Software Foundation. "Python Documentation." (2024), URL: https://docs.python.org/3/ (visitado 30-05-2024).

[7] Grafana Labs. "Grafana documentation," Grafana Labs. (2024), URL: https://grafana.com/docs/grafana/latest/ (visitado 20-06-2024).

[8] The Linux Foundation. "Overview | prometheus." (2024), URL: https://prometheus.io/docs/introduction/overview/ (visitado 20-06-2024).

[9] Breandan Whitfield. "python-OBD Documentation." (2024), URL: https://python-obd.readthedocs.io/en/latest/ (visitado 20-06-2024).

[10] Dimitrios Rimpas, Andreas Papadakis y Maria Samarakou, "OBD-II sensor diagnostics for monitoring vehicle operation and consumption," *Energy Reports*, Technologies and Materials for Renewable Energy, Environment and Sustainability, vol. 6, págs. 55-63, 1 de feb. de 2020, ISSN: 2352-4847. DOI: 10.1016/j.egyr.2019.10.018. URL: https://www.sciencedirect.com/science/article/pii/S2352484719308649 (visitado 01-05-2024).

[11] Malintha Amarasinghe, Sasikala Kottegoda, Asiri Liyana Arachchi, Shashika Muramudalige, H. M. N. Dilum Bandara y Afkham Azeez, "Cloud-based driver monitoring and vehicle diagnostic with OBD2 telematics," en *2015 Fifteenth International Conference on Advances in ICT for Emerging Regions (ICTer)*,

ago. de 2015, págs. 243-249. DOI: 10.1109/ICTER.2015.7377695. URL: https://ieeexplore.ieee.org/abstract/document/7377695 (visitado 10-06-2024).

[12] The Raspberry Pi Foundation. "Setting up your raspberry pi." (2024), URL: https://projects.raspberrypi.org/en/projects/raspberry-pi-setting-up (visitado 06-06-2024).

[13] DietPi Team. "DietPi.com docs," DietPi Documentation. (2024), URL: https://dietpi.com/docs/ (visitado 18-06-2024).

[14] Brendan Bank, *gpsd-prometheus-exporter*, 22 de mayo de 2024. URL: https://github.com/brendanbank/gpsd-prometheus-exporter (visitado 30-04-2024).

[15] Richard Zetterberg, *rzetterberg/obd_ exporter*, 7 de jun. de 2024. URL: https://github.com/rzetterberg/obd_exporter (visitado 30-04-2024).

[16] Eric S. Raymond. "GPSd Documentation." (8 de mar. de 2018), URL: https://gpsd.gitlab.io/gpsd/ (visitado 10-06-2024).

[17] Armin Ronacher. "Flask Documentation (3.0.x)." (2024), URL: https://flask.palletsprojects.com/en/3.0.x/ (visitado 09-05-2024).

[18] Grafana Labs. "Download Grafana | Grafana Labs." (2024), URL: https://grafana.com/grafana/download?platform=arm (visitado 19-06-2024).

[19] The Linux Foundation. "Prometheus | download." (2024), URL: https://prometheus.io/download/ (visitado 19-06-2024).

[20] Anaconda Inc. "Miniconda — Anaconda documentation." (2024), URL: https://docs.anaconda.com/miniconda/ (visitado 15-06-2024).

[21] Conda-Forge Community, *Miniforge*, 17 de jun. de 2024. URL: https://github.com/conda-forge/miniforge (visitado 20-06-2024).

[22] Netdata. "GitHub - netdata/netdata: The open-source observability platform everyone needs!" (2024), URL: https://github.com/netdata/netdata (visitado 18-06-2024).

[23] The Linux Foundation, *Perses*, 26 de ene. de 2021. URL: https://github.com/perses/perses (visitado 20-06-2024).

[24] Rodrigo Alonso Pastor. "AutoSampler · GitLab," GitLab. (19 de jun. de 2024), URL: https://gitlab.inf.uva.es/rodralo/autosampler/-/tree/main (visitado 20-06-2024).

Apéndices

Apéndice A

Manual de instalación

En este apéndice se detalla el manual de instalación del proyecto, los pasos que se deben seguir para poner el proyecto en funcionamiento una vez se tienen a disposición los materiales necesarios. Se van a presentar dos opciones, una instalación sencilla si los materiales coinciden con los expuestos en esta memoria, y una versión algo más compleja que puede funcionar con prácticamente cualquier hardware que cumpla los requisitos.

A.1. Instalación rápida

El primer paso es preparar la Raspberry Pi para albergar el sistema operativo. Con la tarjeta SD de esta conectada a un ordenador, se procede a descargar el programa Raspberry Pi Imager y el sistema operativo DietPi. Una vez instalado, se puede ejecutar e instalar el sistema operativo en la tarjeta.

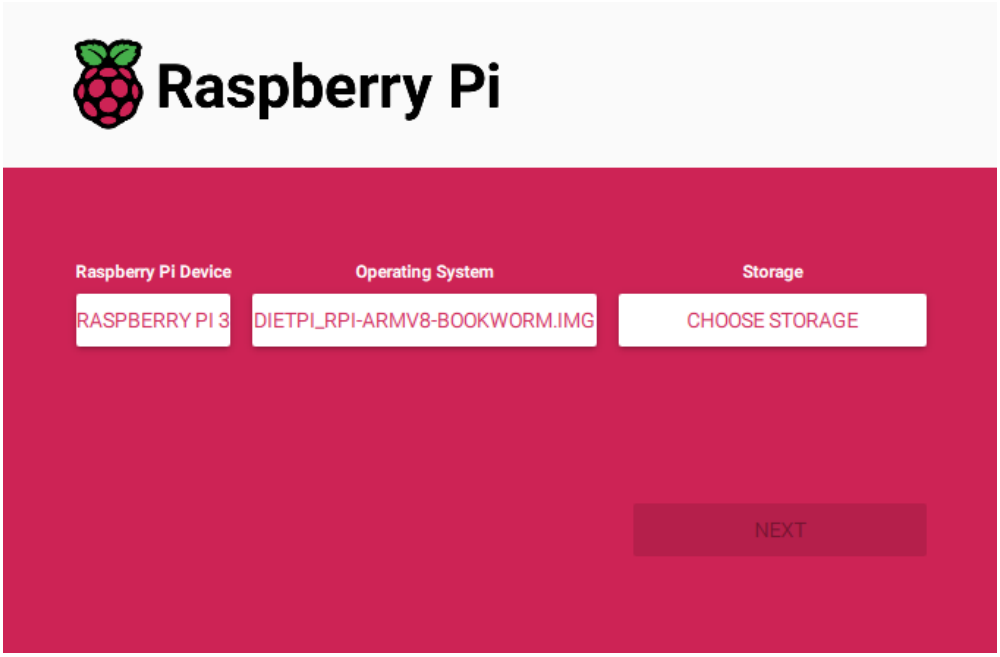


Figura A.1: Instalación de Sistema Operativo a través de Raspberry Pi Imager.

Después de iniciarlo y realizar la configuración inicial, se deben hacer los cambios al sistema comentados en la Implementación: Configurar la Conectividad y modificar el Inicio de Sesión.

Para instalar los programas, paquetes y dependecias se ha preparado un script que automatiza el proceso.

Este proceso se debe hacer desde el directorio root, es decir, */root*. Primero, se debe descargar e instalar Grafana, ejecutando los siguientes comandos:

```
1 sudo apt-get install -y adduser libfontconfig1 musl
2 wget https://dl.grafana.com/enterprise/release/grafana-enterprise_10
    .3.3_arm64.deb
3 sudo dpkg -i grafana-enterprise_10.3.3_arm64.deb
```

Código A.1: Descarga de Grafana.

Después, se debe descargar Prometheus, con el siguiente comando

```
1   wget https://github.com/prometheus/prometheus/releases/download/v2
      .53.0/prometheus-2.53.0.linux-arm64.tar.gz
```

Código A.2: Descarga de Prometheus-Server.

Por último, debe descomprimir la carpeta que se puede descargar aquí[24], y ejecutar el script de instalación, con los siguientes dos comandos:

```
1   unzip autosampler-instalacion.zip
2   ./instalacion.sh
```

Código A.3: Script de instalación rápida de AutoSampler.

Este proceso instalará todos los paquetes necesarios, configuraciones y automatizaciones. Algunos paquetes tienen instalaciones interactivas, por lo que el usuario deberá confirmar un par de veces ciertas opciones.

Cuando se complete, el usuario deberá reiniciar el sistema y AutoSampler arrancará automáticamente.

El último paso es crear un tablero en Grafana para visualizar los datos. Se puede importar el tablero usado para la implementación, a través de su fichero JSON.

A.2. Instalación personalizada

En esta sección se detalla la instalación personalizada de todos los componentes, para usuarios experimentados que decidan montar el sistema con otras opciones Hardware o otra configuración.

A.2.1. Sistema Operativo

En cuanto al sistema operativo, se ha comentado a lo largo del documento que se puede ejecutar en prácticamente cualquier sistema Linux. No obstante, se sigue recomendando DietPi por su bajo consumo, flexibilidad y opciones, ya que ofrecen también una versión para sistemas x86.

A.2.2. Servicios Python y gestión de dependencias

Para ejecutar el servicio Python, se necesitan los siguientes requisitos:

- Python 3.8
- Paquetes setuptools, wheel, prometheus_client, pyserial
- gpsd_prometheus_exporter

Para gestionar las versiones de Python y sus dependencias, se recomienda instalar Miniconda, disponible para ARM[21] y x86[20], aunque también se podría crear una imagen en Docker u otros servicios.

El servicio Python de AutoSampler se puede descargar del siguiente enlace.

Para ejecutar los servicios, se recuerdan los comandos y sus opciones:

```
1   gpsd-exporter -b 0.0.0.0 -p 9999 -s kph
2   python autosampler-obd.py
```

Código A.4: Puesta en marcha de servicios Python.

A.2.3. Prometheus-Server

Las descargas de Prometheus-Server se pueden encontrar para ambas arquitecturas en su página web[19].

Una vez descomprimido, se debe modificar la configuración por defecto para incluir los servicios Python, de este modo:

```
1     global:
2     scrape_interval: 0s500ms
3
4     scrape_configs:
5     - job_name: "prometheus"
6
7     static_configs:
8     - targets:
9     - "localhost:8000"
10    - "localhost:9999"
```

Código A.5: Configuración de Prometheus-Server.

A.2.4. Grafana

Análogamente a Prometheus, las descargas del servidor de Grafana se encuentran en el siguiente enlace[18].

Una vez instalado, se debe tener en cuenta el plugin CSV que se puede instalar con el comando:

```
1    grafana-cli plugins install marcusolsson-csv-datasource
```

Código A.6: Puesta en marcha de servicios Python.

Se debe también cambiar la configuración, ubicada en */etc/grafana/grafana.ini* para ajustar el tiempo de respuesta y permitir archivos CSV locales:

```
1    [plugin.marcusolsson-csv-datasource]
2    allow_local_mode = true
3
4    [dashboards]
5    min_refresh_interval = 500ms
```

Código A.7: Puesta en marcha de servicios Python.

Por último, se puede importar el tablero por defecto mostrado en este documento.

A.2.5. Automatización

Para finalizar la instalación, se incluye el script para automatizar la puesta en marcha de los servicios. Este corresponde a la instalación simple, por lo que habrá que modificarlo teniendo en cuenta la estructura de directorios, versiones de los programas, etc.

A.3. Primer inicio y Guía de uso

El sistema es sencillo de utilizar. Lo único que debe hacer el usuario es acceder a la plataforma Grafana, ubicada en el puerto 3000 de la Raspberry Pi. La primera vez que se inicia, pedirá registrar unas credenciales, añadir una fuente de datos y crear un tablero.

A.3.1. Añadir fuentes de datos

Accediendo al menú lateral, se puede configurar una fuente de datos en el apartado de *Connections*. En la barra de búsqueda, se puede buscar directamente 'Prometheus'.

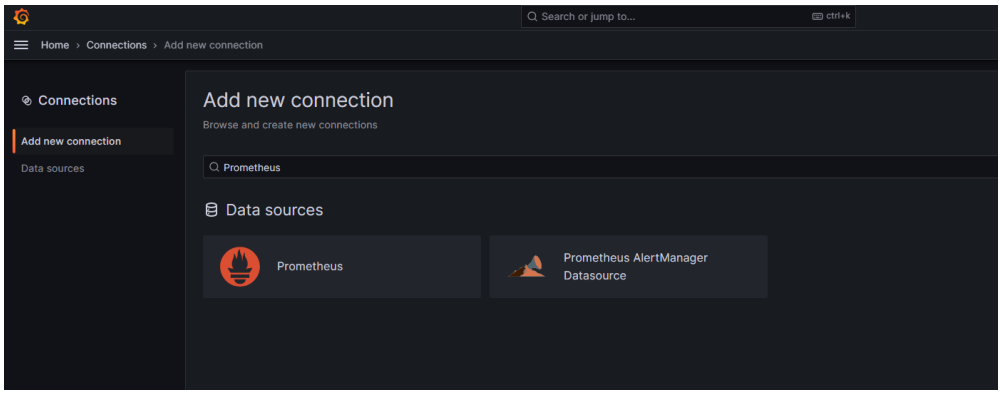


Figura A.2: Menú de conexiones de Grafana.

Al añadirla, se deben configurar únicamente dos opciones: La dirección y el intervalo de 'scrape'. Si se usa la instalación por defecto, se deben poner exactamente los valores vistos en las dos siguientes Figuras:

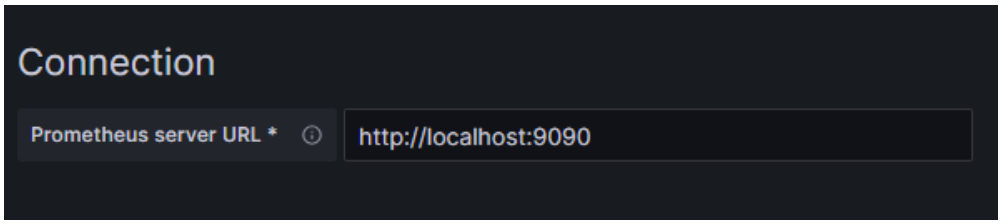


Figura A.3: Configuración de Dirección de Fuente de Datos en Grafana.

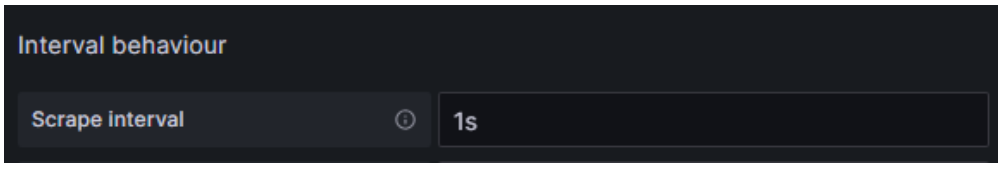


Figura A.4: Configuración de Intervalo de Fuente de Datos en Grafana.

Una vez hecho, se puede cerrar.

Se debe hacer lo mismo para la fuente de averías, buscando 'CSV' en la barra de búsqueda, y configurando la siguiente dirección:

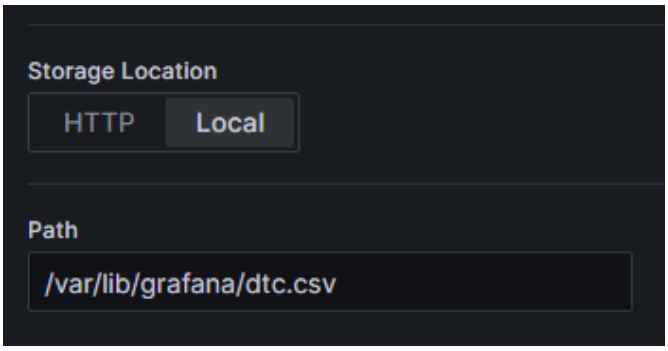


Figura A.5: Configuración de plugin CSV en Grafana.

A.3.2. Creación de tablero

Accediendo al menú lateral, se selecciona la opción *Dashboards*. En esta, se puede crear un tablero con el botón ubicado en la esquina superior derecha.

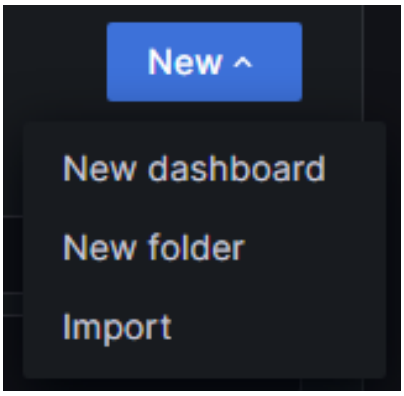


Figura A.6: Creación de un tablero nuevo en Grafana.

En el botón de importar, se puede subir directamente el fichero JSON que contiene el tablero usado en este trabajo y que funciona directamente.

Con este primer inicio completado, lo único que debe hacer el usuario es acceder a la plataforma cuando quiera comprobar las métricas, averías y demás módulos.