Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Sistema de Telemetría en Tiempo Real para Motocicletas de Competición

Autor: Carlos Hernández Herrero

Tutor(a): Ángel Herranz Nieva

Madrid, Junio 2025

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería informática
Título: Sistema de Telemetría en Tiempo Real para Motocicletas de Competición
Junio 2025

Autor: Carlos Hernández Herrero

Tutor:
Ángel Herranz Nieva
LENGUAJES Y SISTEMAS INFORMÁTICOS E INGENIERÍA DE SOFTWARE
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

El objetivo principal de este Trabajo de Fin de Grado es el diseño e implementación de un sistema completo de monitorización y visualización de datos de telemetría en tiempo real para una motocicleta de competición. Este sistema está orientado específicamente a su integración en la moto EME-25P, desarrollada por el equipo universitario UPM Motostudent Petrol, al cual pertenezco como miembro activo del departamento de electrónica.

Este sistema tiene como finalidad permitir la lectura de los datos que emite la centralita electrónica (ECU) de la motocicleta y su transmisión a los mecánicos en boxes, donde serán visualizados en tiempo real. Sin embargo, es importante destacar que este sistema no será utilizado durante las carreras de la competición, ya que las normas de la prueba no permiten el uso de telemetría en carrera. Su aplicación se limita, por tanto, a las rodadas de pruebas del equipo, donde el análisis de datos en directo es fundamental para el desarrollo, ajuste y puesta a punto de la moto.

El objetivo final del proyecto es implementar un sistema real que permita leer datos desde la ECU utilizando un dispositivo embebido, como una Raspberry Pi montada en la propia motocicleta. Esta Raspberry estaría conectada al puerto OBD de la moto mediante un adaptador USB, y enviaría los datos al servidor gracias a una conexión a Internet. El servidor procesaría las cadenas de texto recibidas, las interpretaría y las enviaría a través de WebSockets a uno o varios clientes ubicados en los boxes, donde los datos serían graficados para facilitar su análisis en tiempo real por parte del equipo técnico.

No obstante, debido a la complejidad del sistema completo y a las limitaciones de tiempo propias del Trabajo de Fin de Grado, en esta fase inicial se ha desarrollado una versión funcional y totalmente operativa del sistema centrada en la simulación. En lugar de obtener los datos directamente desde la ECU, se utiliza un fichero CSV con datos reales como fuente de información. Esta simulación permite replicar el comportamiento del sistema en condiciones muy similares a las reales, validando tanto el flujo de datos como el procesamiento y visualización de los mismos.

Este desarrollo constituye la base sobre la cual se podrá continuar el trabajo en el futuro, integrando la lectura real desde la ECU en la moto y completando así el ciclo completo del sistema. La arquitectura y el diseño implementados han sido planteados con una visión modular y escalable, permitiendo su adaptación futura sin necesidad de rehacer el sistema desde cero.

Abstract

The main objective of this Bachelor's Thesis is the design and implementation of a complete real-time telemetry data monitoring and visualization system for a racing motorcycle. This system is specifically aimed at its integration into the EME-25P motorcycle, developed by the university team UPM Motostudent Petrol, of which I am an active member of the electronics department.

The purpose of this system is to enable reading the data emitted by the motorcycle's electronic control unit (ECU) and transmitting it to the mechanics in the pits, where it will be visualized in real time. However, it is important to note that this system will not be used during the race itself, since the competition rules prohibit the use of telemetry during the race. Its application is therefore limited to the team's test sessions, where real-time data analysis is fundamental for the development, tuning, and setup of the motorcycle.

The final goal of the project is to implement a real system that allows reading data from the ECU using an embedded device, such as a Raspberry Pi installed on the motorcycle itself. This Raspberry Pi would be connected to the bike's OBD port via a USB adapter and would send the data to a server through an internet connection. The server would process the received text streams, interpret them, and send them via WebSockets to one or more clients located in the pits, where the data would be graphed to facilitate real-time analysis by the technical team.

However, due to the complexity of the complete system and the time constraints of the Bachelor's Thesis, an initial fully functional and operational version of the system focused on simulation has been developed. Instead of obtaining data directly from the ECU, a CSV file with real data is used as the information source. This simulation allows replicating the system's behavior under conditions very similar to the real ones, validating both the data flow and its processing and visualization.

This development constitutes the foundation on which future work can continue, integrating actual reading from the motorcycle's ECU and thus completing the system's full cycle. The implemented architecture and design have been conceived with a modular and scalable vision, allowing future adaptation without needing to rebuild the system from scratch.

Tabla de Contenidos

1. Introducción 6
   1.1 Objetivo 7
2. Trabajo Previo 8
   2.1 Revisión de Soluciones Existentes 9
   2.2 Evaluación de Tecnologías Disponibles 9
   2.3 Tecnologías y Herramientas Seleccionadas 10
   2.4 Conceptos Clave 11
3. Análisis 12
   3.1 Requisitos funcionales 12
   3.2 Requisitos no funcionales 13
   3.3 Alcance del sistema 14
   3.4 Punto de Partida 15
4. Diseño del sistema 15
   4.1 Arquitectura General del Sistema 16
   4.2 Componentes y Paquetes del Sistema 18
      4.2.1 Cliente de Simulación de Datos (simulación de la motocicleta) 18
      4.2.2 Servidor Central (Spring Boot) 19
      4.2.3 Cliente Web de Visualización (cliente en boxes) 20
   4.3 Modelo de datos 20
      4.3.1 Clase ECUData 21
      4.3.2 Clases de sensores 21
      4.3.3 Clase TelemetriaData 22
   4.4 Diagrama de secuencia de flujo de datos 23
      4.4.1 Descripción del flujo de datos 23
   4.5 Comunicación entre los Componentes del Sistema 24
5. Implementación 26
   5.1 Tecnologías Utilizadas 27
   5.2 Estructura del Código 28
   5.3 Lógica del Servidor 29
      5.3.1 Proceso General 29
      5.3.2 Controlador y Recepción de Datos 30

5.3.3 Servicio de Lectura de Ficheros 31
5.3.4 Envío de Datos 31
5.4 Cliente Simulador de la Moto 32
5.4.1 Objetivo del Simulador 32
5.4.2 Funcionamiento del Cliente 32
5.4.3 Adaptación Futura 33
5.5 Cliente de Boxes y Visualización de Datos 34
5.5.1 Funcionalidad General 34
5.5.2 Organización y Diseño Visual 37
5.5.3 Validación del Comportamiento 38
5.6 Servidor Web con Spring Boot 38
5.6.1 Estructura General del Servidor 39
5.6.2 Comunicación con los Clientes 39
5.6.3 Configuración del WebSocket 40
5.6.4 Recepción de Datos y Procesamiento 40
5.6.5 Robustez y validaciones 41
5.6.6 Estado actual y expansión futura 41
5.7 Gestión de Errores y Pruebas 42
5.7.1 Gestión de Errores 42
5.7.2 Pruebas Unitarias 43
6. Conclusión y Trabajo Futuro 43
6.1 Conclusiones 43
6.2 Resultados 45
6.3 Trabajo Futuro 46
6.4 Análisis de Impacto 49
7. Bibliografía y referencias 51

1. Introducción

En el ámbito del motociclismo de competición, la telemetría se ha consolidado como una herramienta esencial para el análisis y la optimización del rendimiento tanto del piloto como del vehículo. Disponer de datos precisos y en tiempo real sobre el comportamiento dinámico de la motocicleta permite a los ingenieros y técnicos tomar decisiones fundamentadas que influyen directamente en el desempeño en pista. No obstante, el acceso a tecnologías de este tipo suele estar limitado a equipos profesionales con grandes recursos.

En este contexto, el presente Trabajo de Fin de Grado se enmarca en el proyecto universitario UPM MotoStudent, una iniciativa en la que estudiantes de ingeniería de la Universidad Politécnica de Madrid diseñan, desarrollan y construyen una moto de competición desde cero para participar en la competición internacional MotoStudent. Esta experiencia multidisciplinar ofrece un entorno real de desarrollo en el que aplicar conocimientos técnicos y enfrentarse a retos propios de la industria.

MotoStudent es una competición bienal de ingeniería y motociclismo que enfrenta a universidades a nivel internacional. En su última edición, se superó el récord de participación con 80 equipos de 19 países diferentes y 4 continentes. El objetivo es diseñar, desarrollar y construir el prototipo de una motocicleta de competición a partir de un kit básico proporcionado por la organización del proyecto. Dicho kit es común para todos los equipos, garantizando la igualdad de condiciones de partida y está compuesto por un motor, pinzas de freno y neumáticos.

La competición se divide en dos fases: MS1, donde se valora el proyecto industrial presentado por el equipo, incluyendo diseño, proceso de fabricación, viabilidad económica e innovación tecnológica; y MS2, que consiste en la fase de validación prestacional mediante pruebas dinámicas y una carrera en el trazado Grand Prix FIM Internacional de MotorLand Aragón.

El equipo UPM MotoStudent Petrol, al cual pertenezco como miembro activo del departamento de electrónica, ha trabajado desde su fundación en 2014 en la constante evolución de sus prototipos de motocicletas con motor de combustión interna. Este proyecto no solo implica el desarrollo técnico de la moto, sino también la gestión integral del equipo como si fuera una empresa. Esto incluye desde el diseño y fabricación de piezas mediante mecanizado de aluminio y fibra de carbono hasta la implementación y mejora de sistemas electrónicos como el dashboard, el sistema de adquisición de datos y la gestión del cableado.

El departamento de electrónica, en particular, se encarga de desarrollar soluciones innovadoras que permitan monitorizar y optimizar el comportamiento de la moto, siendo la telemetría una de las áreas más desafiantes y con mayor impacto para el equipo. A lo largo de los años, UPM MotoStudent ha logrado importantes avances tecnológicos y de sostenibilidad, implementando soluciones innovadoras como el uso de prototipos impresos en

3D a escala y piezas de fibra de carbono fabricadas con procesos ecológicos. Estos logros han sido posibles gracias al apoyo de patrocinadores clave, permitiendo al equipo mantenerse a la vanguardia en eficiencia y tecnología eléctrica.

En el ámbito profesional, la telemetría es una herramienta indispensable en competiciones como MotoGP o el Campeonato Mundial de Superbikes. Equipos como Ducati, Yamaha o Honda utilizan sistemas avanzados de telemetría para monitorizar en tiempo real parámetros como la velocidad, la aceleración, la temperatura de los neumáticos, la presión de frenos y la inclinación de la moto. Esta información permite a los ingenieros ajustar la configuración de la motocicleta y mejorar el rendimiento del piloto en cada sesión de entrenamiento o carrera.

Sin embargo, en la competición MotoStudent, el uso de sistemas de telemetría está restringido durante las carreras oficiales por normativa. Por ello, el presente proyecto se centra en el desarrollo de un sistema de telemetría que pueda ser utilizado durante las sesiones de prueba y entrenamiento, donde sí está permitido su uso. Este sistema permitirá al equipo recopilar y analizar datos clave para la mejora del rendimiento de la motocicleta y la toma de decisiones técnicas.

1.1 Objetivo

El objetivo final del proyecto es implementar un sistema real que permita leer datos desde la ECU utilizando un dispositivo embebido, como una Raspberry Pi montada en la propia motocicleta. Esta Raspberry estaría conectada al puerto OBD de la moto mediante un adaptador USB, y enviaría los datos al servidor gracias a una conexión a Internet. El servidor procesaría las cadenas de texto recibidas, las interpretaría y las enviaría a través de WebSockets a uno o varios clientes ubicados en los boxes, donde los datos serían graficados para facilitar su análisis en tiempo real por parte del equipo técnico.

No obstante, debido a la complejidad del sistema completo y a las limitaciones de tiempo propias del Trabajo de Fin de Grado, en esta fase inicial se ha desarrollado una versión funcional y totalmente operativa del sistema centrada en la simulación. En lugar de obtener los datos directamente desde la ECU, se utiliza un fichero CSV con datos reales como fuente de información. Esta simulación permite replicar el comportamiento del sistema en condiciones muy similares a las reales, validando tanto el flujo de datos como el procesamiento y visualización de los mismos.

Este desarrollo constituye la base sobre la cual se podrá continuar el trabajo en el futuro, integrando la lectura real desde la ECU en la moto y completando así el ciclo completo del sistema. La arquitectura y el diseño implementados han sido planteados con una visión modular y escalable, permitiendo su adaptación futura sin necesidad de rehacer el sistema desde cero.

En este documento se describe el proceso de desarrollo del sistema de telemetría para motocicletas. Se comienza por presentar el trabajo previo que sirve de base para el diseño y la implementación del sistema, seguido de una definición detallada de los requisitos funcionales y no funcionales que debe cumplir. A continuación, se aborda el diseño arquitectónico del sistema, incluyendo la estructura de los componentes principales, y se detalla el proceso de implementación. Finalmente, se describen las pruebas realizadas para validar el correcto funcionamiento del sistema y garantizar que cumple con los objetivos.

2. Trabajo Previo

Para desarrollar cualquier tipo de proyecto, es fundamental partir de una definición clara y precisa de los requisitos y pasos a seguir.

Lo primero y más importante hacer un análisis de otros proyectos similares. Estudiar soluciones previas permite comprender mejor los retos técnicos y funcionales a los que se enfrenta un sistema de telemetría en entornos reales, además de identificar buenas prácticas, patrones de diseño eficaces y posibles errores a evitar. Esta revisión contribuye a tomar decisiones más informadas y a construir una base sólida sobre la cual desarrollar el sistema propio.

Otro aspecto clave en las etapas iniciales del desarrollo es la evaluación de las diferentes librerías, tecnologías y frameworks disponibles para cada componente del sistema. Existen múltiples opciones tanto para la comunicación en tiempo real como para la visualización de datos o la construcción del backend. Comparar las alternativas según criterios como facilidad de integración, documentación, rendimiento, comunidad activa y compatibilidad con el entorno del proyecto permite elegir las herramientas más adecuadas para lograr una solución eficiente, mantenible y escalable.

Durante el desarrollo del sistema de telemetría, se han seleccionado y empleado diversas herramientas y tecnologías que facilitan la implementación eficiente y escalable de la solución. A continuación, se describen las principales:

Este capítulo presenta una revisión del estado del arte y las tecnologías relacionadas, así como el punto de partida del proyecto, abordando tanto los antecedentes técnicos como las decisiones iniciales sobre herramientas y arquitectura.

2.1 Revisión de Soluciones Existentes

Antes de iniciar el diseño e implementación del sistema, se ha llevado a cabo un análisis de soluciones existentes relacionadas con la telemetría en entornos de automoción, especialmente en contextos de competición. Aunque en el ámbito de MotoStudent no se dispone de sistemas comerciales implantados ampliamente, sí existen referencias relevantes en otros contextos como MotoGP, Fórmula 1 o simuladores de carreras.

Estos sistemas de telemetría comparten ciertos principios fundamentales:

- Recogida continua de datos desde la ECU (Unidad de Control Electrónico).
- Transmisión inalámbrica de los datos en tiempo real hacia una estación remota.
- Almacenamiento y visualización de las sesiones para análisis posterior.

El estudio de estos modelos permite comprender mejor los retos técnicos y funcionales a los que se enfrenta un sistema de este tipo, identificando patrones de diseño eficaces, arquitecturas comunes y posibles errores a evitar, como latencias elevadas o cuellos de botella en la comunicación.

En el caso de MotoStudent, dada la limitación de recursos y la necesidad de adecuación a la normativa de la competición, se plantea una solución personalizada que respete las condiciones de hardware disponibles (ECU propia, Raspberry Pi, red Wi-Fi en boxes) y las restricciones de uso (prohibición de comunicación en pista durante la carrera).

2.2 Evaluación de Tecnologías Disponibles

Una vez definida la necesidad general del sistema, se ha realizado una evaluación técnica de distintas tecnologías disponibles para cada componente:

- **Comunicación en tiempo real**: se han comparado alternativas como WebSockets, MQTT, HTTP polling y gRPC. WebSockets ha sido finalmente la opción elegida por su soporte nativo en navegadores, bidireccionalidad, baja latencia y buena integración con Spring Boot.
- **Visualización de datos**: se han considerado bibliotecas como Chart.js, D3.js y Plotly. Se ha optado por Chart.js debido a su simplicidad de uso, ligereza y buena documentación, suficiente para los requisitos del proyecto.
- **Frameworks backend**: se valoraron alternativas como Express.js (Node.js), Django (Python) y Spring Boot (Java). La elección de Spring Boot responde tanto al dominio del lenguaje como a su robustez,

modularidad, capacidad de integración con WebSockets y compatibilidad con arquitecturas escalables.

Este análisis comparativo ha permitido tomar decisiones informadas en cuanto a qué herramientas utilizar, asegurando una solución eficiente, mantenible y escalable.

2.3 Tecnologías y Herramientas Seleccionadas

A continuación, se detallan las tecnologías y herramientas finalmente empleadas en el proyecto:

- **Editor de texto para desarrollo: IntelliJ IDEA**

  Para el desarrollo del backend en Java se ha utilizado el entorno de desarrollo integrado (IDE) IntelliJ IDEA. Este editor ofrece una amplia gama de herramientas avanzadas para programación en Java, incluyendo autocompletado inteligente, depuración, integración con sistemas de control de versiones y gestión de dependencias mediante Maven, lo que ha facilitado notablemente el desarrollo, organización y mantenimiento del proyecto.

- **Backend: Spring Boot (Java)**

  Framework robusto para aplicaciones web y microservicios, con configuración automática, soporte de WebSockets, y estructura modular. Facilita una arquitectura clara basada en controladores, servicios y entidades.

- **Comunicación en tiempo real: WebSockets con STOMP**

  Permite la comunicación bidireccional entre el cliente (moto) y el servidor, así como entre el servidor y los ordenadores en boxes. Se ha usado la implementación de WebSockets que ofrece Spring, junto al protocolo STOMP para facilitar la suscripción a temas y el envío estructurado de mensajes.

  Nota: Aunque inicialmente se planteó el envío de datos mediante una API REST, se decidió utilizar WebSockets por su mayor adecuación a la naturaleza en tiempo real del sistema. La opción REST se contempla para futuras extensiones, como la consulta de datos históricos.

- **Frontend: HTML, JavaScript y Chart.js**

  La interfaz se ha construido con tecnologías web estándar. Para la representación gráfica se utiliza Chart.js, que permite construir gráficos interactivos y responsivos con bajo coste computacional.

- **Control de versiones: Git y GitHub**

  La gestión de versiones se ha realizado mediante Git, alojando el repositorio en GitHub, lo que permite trazabilidad, control de cambios y colaboración.

2.4 Conceptos Clave

A continuación, se definen los principales conceptos técnicos que resultan relevantes para la comprensión del presente trabajo y que aparecen de forma recurrente a lo largo del mismo:

- **Telemetría**: Técnica que permite la medición y transmisión de datos desde un dispositivo remoto hacia un sistema central, donde dichos datos pueden ser monitorizados y analizados. En el contexto de este trabajo, se refiere a la recopilación de datos en tiempo real desde una motocicleta durante su funcionamiento.

- **ECU (Electronic Control Unit)**: Unidad de Control Electrónico encargada de gestionar diversos parámetros del funcionamiento del motor de un vehículo, como la inyección de combustible, la velocidad, el régimen del motor (RPM), entre otros. Actúa como fuente principal de datos para el sistema de telemetría. En este caso la utilizada en la motocicleta es la ECU AIM Taipan.

- **WebSocket**: Protocolo de comunicación que permite establecer una conexión persistente y bidireccional entre cliente y servidor. Es especialmente útil para aplicaciones en tiempo real, ya que reduce la latencia en la transmisión de datos al evitar solicitudes HTTP repetidas.

- **STOMP (Simple Text Oriented Messaging Protocol)**: Protocolo de mensajería basado en texto que se utiliza comúnmente sobre WebSockets. Facilita la comunicación entre aplicaciones mediante un modelo de publicación-suscripción, permitiendo que múltiples clientes reciban mensajes simultáneamente.

- **Spring Boot**: Framework de desarrollo para Java que simplifica la creación de aplicaciones web y microservicios. Proporciona una estructura predefinida y herramientas integradas que reducen el tiempo de configuración y despliegue, favoreciendo el desarrollo rápido y eficiente del backend.

- **Chart.js**: Biblioteca JavaScript de código abierto que permite crear gráficos interactivos y personalizables en aplicaciones web. Se ha utilizado en este proyecto para representar visualmente los datos de telemetría en el navegador del usuario.

- **Fichero CSV (Comma-Separated Values)**: Formato de archivo utilizado para almacenar datos tabulares en texto plano, donde cada línea representa un registro y los valores se separan por comas. Se emplea en este trabajo como fuente de datos simulada para alimentar el sistema durante las pruebas.
- **Cliente-servidor**: Modelo de arquitectura en el que un cliente (por ejemplo, una aplicación o navegador web) solicita recursos o servicios a un servidor central, que procesa dichas solicitudes y responde con la información correspondiente. Este modelo estructura la comunicación entre la motocicleta, el servidor central y el cliente visualizador.
- **Backend**: Parte del sistema que opera en el servidor y gestiona la lógica del negocio, el procesamiento de datos y la comunicación con otros componentes. En este trabajo, el backend se encarga de recibir, procesar y reenviar los datos de telemetría.
- **Frontend**: Parte del sistema que interactúa directamente con el usuario final. Se refiere a la interfaz gráfica desde la que se visualizan los datos procesados. Está desarrollada con tecnologías web como HTML, CSS y JavaScript.

3. Análisis

A continuación, se detallan las funcionalidades clave que debe cumplir el sistema, han de garantizar su calidad, rendimiento y mantenibilidad:

3.1 Requisitos funcionales

- **Lectura de datos simulados**

  El sistema debe ser capaz de leer datos de telemetría desde un archivo CSV, que actúa como fuente de datos simulados durante las fases de desarrollo y prueba.
- **Envío de datos al servidor desde el cliente de la moto**

  Los datos leídos deben ser enviados mediante una conexión WebSocket desde un componente que simula el cliente situado en la motocicleta hacia el servidor, reproduciendo el comportamiento esperado en un entorno real.

- **Recepción, procesamiento y retransmisión de datos por parte del servidor**

  El servidor debe recibir los datos, procesarlos en el formato adecuado y reenviarlos a los clientes que se encuentran en los boxes, permitiendo la monitorización en tiempo real. Además ha de ser capaz de guardar una copia de los datos recibidos en formato CSV.

- **Visualización en tiempo real**

  Los datos deben ser representados gráficamente en una interfaz web accesible, mediante gráficas dinámicas que actualicen parámetros como las revoluciones por minuto (rpm), temperatura del motor, tensión de batería, entre otros.

- **Acceso desde navegador web**

  La interfaz debe ser accesible a través de cualquier navegador moderno conectado a la red local, sin necesidad de instalar aplicaciones adicionales.

- **Soporte para múltiples clientes simultáneos**

  El sistema debe permitir que varios clientes (por ejemplo, varios dispositivos en boxes) puedan conectarse simultáneamente al servidor y recibir los datos de telemetría en tiempo real.

- **API WebSocket con rutas definidas**

  El backend debe proporcionar una API organizada que permita gestionar las distintas conexiones WebSocket, utilizando rutas específicas para mantener una estructura clara y extensible.

3.2 Requisitos no funcionales

- **Framework y lenguaje de programación**

  El backend está desarrollado con el framework Spring Boot en lenguaje Java, lo que permite una estructura modular y escalable.

- **Tecnologías para la visualización**

  La parte frontal está implementada con HTML, JavaScript y la librería Chart.js para la generación de gráficas interactivas y ligeras.

- **Comunicación basada en WebSockets con STOMP**

  Para asegurar una comunicación en tiempo real eficaz, se ha optado por el protocolo WebSocket junto con STOMP, que proporciona una capa adicional de organización y suscripción de mensajes.

- **Frecuencia de actualización configurable (solo versión inicial)**

La lectura y envío de datos desde el cliente simulado se realiza de forma periódica, con una frecuencia configurable. En el estado actual, el sistema lee y envía datos cada 500 milisegundos, aunque este valor puede modificarse según las necesidades.

En este caso, debido a que se trata de la versión inicial, utilizamos una frecuencia de actualización predefinida para poder imitar el comportamiento natural de la transferencia de datos, el cual se da en forma de bloques de datos cada x segundos (depende de la ECU).

- **Escalabilidad futura con conexión OBD**

  Aunque actualmente se utiliza un archivo CSV como fuente de datos, el sistema se ha diseñado para que en el futuro pueda conectarse directamente a la ECU de la motocicleta a través del conector OBD, permitiendo lecturas en tiempo real en condiciones reales de circuito.

- **Rendimiento del servidor**

  El servidor debe mantener un tiempo de respuesta inferior a los 200 milisegundos, lo cual es fundamental para garantizar una experiencia de usuario fluida y sin interrupciones en la visualización de los datos.

- **Separación por capas**

  Se ha seguido una arquitectura basada en capas que separa claramente la lógica de negocio (servicios), la gestión de datos (modelos) y el control de flujo (controladores), lo que facilita el mantenimiento, la ampliación del sistema y la implementación de pruebas unitarias.

3.3 Alcance del sistema

El sistema desarrollado en este Trabajo de Fin de Grado se centra exclusivamente en el diseño y desarrollo de la parte software del servidor encargado de procesar, mostrar y almacenar datos de telemetría provenientes de una motocicleta de competición. El alcance del proyecto incluye:

- La definición y estructura del modelo de datos para representar la información proveniente de la ECU.
- La creación de una API y un canal de comunicación en tiempo real para la recepción y difusión de datos.
- El desarrollo de una interfaz web para la visualización de los datos de telemetría durante la rodada.

No se contempla en este trabajo:

- La adquisición directa de datos desde sensores físicos.

- El desarrollo del cliente embebido en la motocicleta (aunque se simula su comportamiento para pruebas).
- El uso de bases de datos relacionales; se ha optado por almacenamiento mediante ficheros CSV.

Este enfoque permite centrar el esfuerzo en la arquitectura, la comunicación en tiempo real y la representación de los datos, dejando la integración con hardware real o ampliaciones funcionales como posibles líneas de trabajo futuro.

3.4 Punto de Partida

El sistema parte de una situación en la que no se dispone aún de conexión directa con la ECU de la motocicleta ni de un entorno hardware real, por lo que:

- Se simulan los datos de entrada mediante archivos CSV que contienen registros representativos de sesiones reales de conducción.
- El cliente embebido en la motocicleta no está implementado, pero se desarrolla un cliente de simulación que reproduce su comportamiento.
- El backend se ha diseñado con vistas a su evolución, permitiendo en el futuro sustituir la entrada simulada por datos en tiempo real desde la ECU mediante un conector OBD.
- El sistema parte desde cero a nivel de software, por lo que toda la arquitectura, comunicación y visualización han sido diseñadas específicamente para este proyecto, basándose en las necesidades del entorno Motostudent.

Este punto de partida marca el límite entre el desarrollo actual y el trabajo futuro a considerar para su implementación real en circuito.

4. Diseño del sistema

El sistema desarrollado está diseñado para simular un entorno real de telemetría en el contexto de una motocicleta de competición. Su arquitectura se basa en un modelo cliente-servidor distribuido, donde distintos componentes

interactúan para capturar, transmitir, procesar y visualizar datos de sensores en tiempo real.

En una situación real, la fuente de los datos sería la ECU, que recibe los datos los sensores instalados en la motocicleta. En el presente proyecto, dicha fuente se ha sustituido por un archivo CSV que contiene registros simulados, con el objetivo de validar toda la cadena de procesamiento antes de desplegarla en una motocicleta real.

4.1 Arquitectura General del Sistema

En este capítulo se describe la estructura interna del sistema de telemetría diseñado, detallando los componentes principales que lo conforman, su comunicación entre sí y el modelo de datos que permite representar y manipular la información transmitida por la motocicleta en tiempo real.

- **Cliente Moto (Simulador)**

Este componente simula el comportamiento del cliente embebido en la motocicleta. Su responsabilidad es leer datos desde un archivo CSV y enviarlos al servidor a través de WebSockets. En una implementación futura, este cliente podría conectarse directamente a la ECU a través del puerto OBD.

- **Servidor (Backend Spring Boot)**

El servidor es el núcleo del sistema y se encarga de:

- Recibir los datos de telemetría desde el cliente de la moto.
- Procesar los datos y almacenarlos.
- Enviar los datos en tiempo real a los clientes conectados.

El servidor está dividido en varios subcomponentes:

- Controlador WebSocket: se encarga de gestionar las conexiones, recibir los datos entrantes y redirigirlos al servicio correspondiente.
- Servicio de Procesamiento: transforma los datos, los valida y se asegura de que estén en el formato adecuado para su uso posterior.

- Modelo de Datos: define la estructura de los datos de telemetría, incluyendo clases que representan información como rpm, temperatura, voltaje, etc.

- **Interfaz Web (Frontend)**

Este componente representa a los clientes en los boxes. A través de una interfaz web, pueden conectarse al servidor para recibir y visualizar los datos en tiempo real. Utiliza tecnologías como HTML, JavaScript y Chart.js para la representación gráfica de la información.

Esta arquitectura modular permite escalar el sistema en el futuro e integrar fácilmente nuevas fuentes de datos, funcionalidades o mejoras en la visualización.

A continuación, se presenta el diagrama de arquitectura que resume la estructura del sistema y la relación entre sus principales componentes:

Descripción de los componentes:

- **Cliente Moto**: Lee un archivo CSV con registros simulados de la ECU (como RPM, temperatura, voltaje, etc.) y envía estos datos al servidor cada 500 ms usando WebSockets y el protocolo STOMP. Esto permite emular el envío de datos en tiempo real desde una motocicleta en pista.
- **Servidor**: desarrollado con Spring Boot, este componente central se encarga de gestionar todas las conexiones WebSocket entrantes y salientes. Recibe los datos del cliente de la moto, los procesa si es necesario (por ejemplo, para convertir tipos o agrupar información) y los reenvía a todos los clientes suscritos en los boxes. También mantiene la lógica de control de conexiones, rutas y estructura modular del backend.
- **Clientes de Boxes**: son interfaces web accesibles desde navegadores locales que se conectan al servidor para recibir los datos de telemetría. Utilizan STOMP sobre WebSockets para recibir los mensajes en tiempo real y emplean la librería Chart.js para representarlos gráficamente. Cada cliente puede visualizar los parámetros clave con actualizaciones inmediatas.

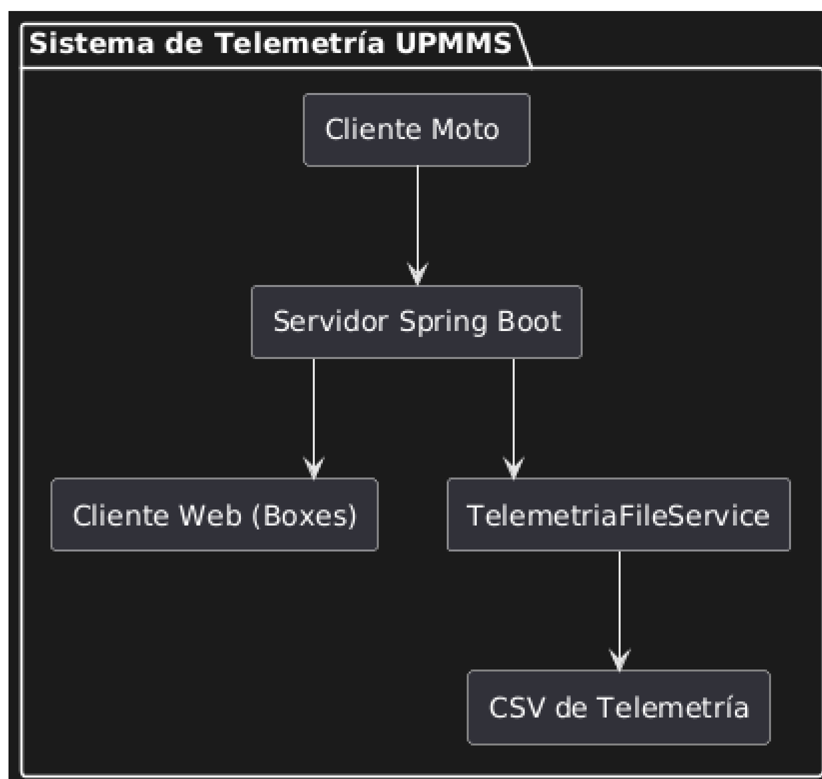


Ilustración 1: Arquitectura del Sistema

4.2 Componentes y Paquetes del Sistema

El sistema ha sido desarrollado empleando una arquitectura en capas, organizada en distintos paquetes Java y componentes funcionales que permiten estructurar el código de forma clara, coherente y escalable. A continuación, se describen los principales componentes del sistema, junto con la responsabilidad de cada uno de ellos.

4.2.1 Cliente de Simulación de Datos (simulación de la motocicleta)

Aunque en el sistema final se prevé la integración de una Raspberry Pi conectada al puerto OBD-II de la motocicleta, en esta primera fase se ha optado por simular el envío de datos a través de un fichero CSV. Este archivo contiene

una traza real de datos obtenidos previamente, y permite validar el sistema completo sin requerir hardware externo.

Esta funcionalidad se implementa mediante una página web que, tras leer el contenido del fichero CSV línea a línea, envía cada línea al servidor a través de un WebSocket abierto con STOMP sobre una ruta específica (/app/datos/telemetria/live). Cada línea representa un conjunto de valores de sensores que se simulan como si fueran enviados en tiempo real desde la motocicleta.

4.2.2 Servidor Central (Spring Boot)

El corazón del sistema es un servidor backend desarrollado en Java con Spring Boot, que se encarga de:

- Recibir los datos de telemetría en formato de texto plano (CSV).
- Interpretar, parsear y convertir esos datos en objetos Java estructurados.
- Enviar los datos formateados a los clientes conectados a través de WebSockets.

Este servidor está organizado en los siguientes paquetes:

com.telemetria.telemetria.controller

Contiene la clase TelemetriaController, que actúa como controlador principal de eventos recibidos por WebSocket. Es la encargada de:

- Recibir los mensajes enviados por el cliente de simulación.
- Interpretarlos y crear los objetos TelemetriaData.
- Publicarlos a los suscriptores a través del SimpMessagingTemplate.

com.telemetria.telemetria.service

Incluye la clase TelemetriaFileService, que contiene métodos para la lectura de los datos desde ficheros CSV, simulando la adquisición desde la ECU. Este servicio también se encarga de parsear cada línea y devolver listas de objetos TelemetriaData.

com.telemetria.telemetria.model

Contiene el modelo de datos, es decir, todas las clases que representan la estructura interna de los datos de telemetría:

- TelemetriaData: clase que agrupa los distintos sensores.
- Subclases de ECUData: RPMData, InclinationData, GearData, SuspCompData, BatteryData y TemperatureData, cada una representando un tipo de sensor diferente.

com.telemetria.telemetria.config

Contiene la configuración de WebSockets, concretamente la clase WebSocketConfig, donde se habilita y registra el endpoint principal del servidor STOMP.

com.telemetria.telemetria.exception

Agrupa las excepciones personalizadas del sistema. Por ejemplo, LecturaFicheroException se lanza en caso de errores al leer el fichero CSV.

4.2.3 Cliente Web de Visualización (cliente en boxes)

Este componente es una aplicación HTML + JavaScript, que se conecta al servidor mediante WebSockets, suscribiéndose al canal /topic/telemetria/live.

Utiliza la librería Chart.js para mostrar gráficamente los valores de los sensores de forma continua. Cada tipo de dato (RPM, inclinación, marcha, suspensión delantera y trasera, batería y temperatura) se representa mediante una gráfica individual que se actualiza con cada nuevo dato recibido.

4.3 Modelo de datos

El modelo de datos del sistema ha sido diseñado para representar de forma clara y extensible los distintos parámetros de telemetría que se capturan de la motocicleta. Este modelo se basa en una jerarquía de clases en Java que permiten estructurar los datos en categorías semánticas y facilitar su tratamiento posterior tanto en el servidor como en el cliente de visualización.

4.3.1 Clase ECUData

En la base del modelo se encuentra la clase abstracta ECUData, que representa cualquier dato proveniente de la Unidad de Control Electrónico (ECU). Esta clase contiene los atributos comunes a todos los sensores:

- **sensor**: Nombre del sensor.
- **timestamp**: Fecha y hora en la que se registró el dato.
- **formattedData**: Representación legible del dato (útil para trazas o interfaces).

Esta clase se extiende para representar cada sensor específico, aportando así especialización.

4.3.2 Clases de sensores

Cada tipo de sensor es representado por una subclase de ECUData, incluyendo:

- **RPMData**: contiene un atributo value de tipo int, que representa las revoluciones por minuto del motor.
- **InclinationData**: almacena un double con la inclinación actual de la motocicleta.
- **GearData**: representa la marcha engranada, como un int.
- **SuspCompData**: incluye dos valores double para la compresión de la suspensión delantera y trasera.
- **BatteryData**: contiene un double con el voltaje actual de la batería.
- **TemperatureData**: representa la temperatura del motor como un double.

Estas clases extienden ECUData y sobrescriben sus métodos para construir sus respectivas cadenas formattedData.

4.3.3 Clase TelemetriaData

La clase TelemetriaData representa una muestra completa de telemetría. Es una clase que agrupa todas las anteriores en un solo objeto:

- **rpm**: instancia de RPMData
- **inclination**: instancia de InclinationData
- **gear**: instancia de GearData
- **suspComp**: instancia de SuspCompData
- **battery**: instancia de BatteryData
- **temperature**: instancia de TemperatureData

Este objeto se utiliza como unidad básica de transmisión de datos entre el servidor y el cliente. Una vez parseada una línea de texto recibida (proveniente del fichero CSV simulado o del cliente futuro en la Raspberry Pi), se transforma en una instancia de TelemetriaData que se publica por WebSocket al canal correspondiente.

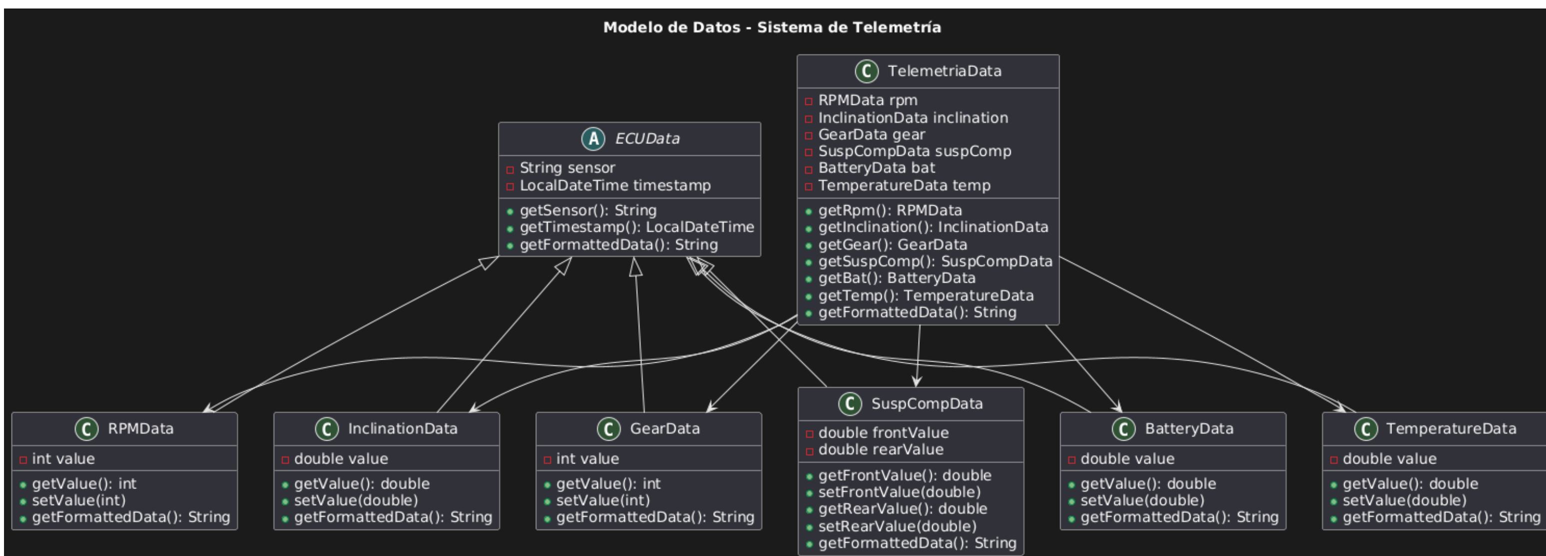


Ilustración 2: Diagrama UML Modelado de Datos

Este modelo orientado a objetos permite tanto la extensibilidad (añadiendo nuevos sensores si fuera necesario) como el mantenimiento del sistema, al estar organizado en torno a principios de diseño limpio y reutilizable.

4.4 Diagrama de secuencia de flujo de datos

Este apartado describe el comportamiento dinámico del sistema durante la operación principal: la recepción y procesamiento de datos de telemetría por parte del servidor y su posterior envío a los clientes para su visualización en tiempo real.

La secuencia refleja el flujo desde la lectura de un dato en el origen (actualmente un fichero CSV simulado), hasta la publicación en el canal WebSocket que los clientes consumen.

4.4.1 Descripción del flujo de datos

El proceso simulado de adquisición y visualización de datos funciona del siguiente modo:

1. **Lectura del dato**: Un cliente (en este caso una página HTML con JavaScript) simula la lectura de datos desde un fichero CSV (vuelta_jarama_moto3.csv). Cada línea representa una muestra de telemetría.

2. **Envío al servidor**: El cliente envía dicha línea a través de WebSocket usando STOMP al endpoint /app/datos/telemetria/live.

3. **Procesamiento en el backend**:

    - El servidor recibe el mensaje y lo divide por comas.

    - Cada valor es parseado y se construyen las instancias de ECUData correspondientes.

    - Estas instancias se agrupan en un objeto TelemetriaData.

4. **Publicación al topic**: El servidor publica este objeto a través de WebSocket en el canal /topic/telemetria/live.

5. **Recepción en el cliente**: El cliente escucha en este canal, recibe el objeto JSON, lo parsea y actualiza los gráficos correspondientes.

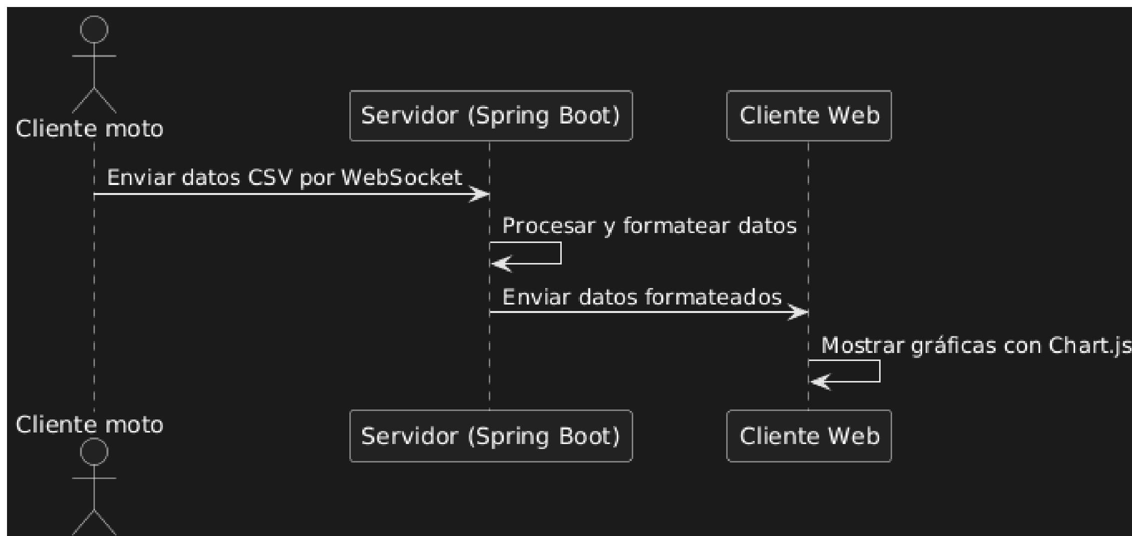


Ilustración 3: Diagrama UML flujo de datos

Este diagrama demuestra cómo el sistema se comporta como una arquitectura reactiva orientada a eventos, en la que los datos fluyen en tiempo real y de manera asincrónica desde el emisor hasta el consumidor, permitiendo una respuesta rápida e inmediata en el análisis.

4.5 Comunicación entre los Componentes del Sistema

La comunicación entre los distintos módulos del sistema se realiza siguiendo una arquitectura basada en eventos y transmisión de mensajes a través de WebSockets. Dado que el objetivo final del proyecto es la lectura de datos en tiempo real desde la motocicleta, se ha diseñado un sistema capaz de recibir flujos continuos de datos, procesarlos y redistribuirlos a los clientes interesados.

En la implementación actual del sistema, la moto aún no está conectada físicamente; por tanto, se ha utilizado una simulación mediante un cliente web que lee datos de un archivo .csv y los envía periódicamente al servidor, emulando el comportamiento esperado del módulo de la moto.

A continuación, se describe cómo interactúan los principales componentes del sistema:

1. **Cliente Web Simulado:** Este componente actúa como un productor de datos. Lee los datos de telemetría desde un archivo y los envía al servidor utilizando un canal WebSocket con el prefijo /app/datos/telemetria/live.

2. **Servidor Spring Boot**:

   - Contiene el WebSocket Controller, que escucha en la ruta especificada y recibe los datos enviados por el cliente.

   - Estos datos se procesan y se convierten en objetos del modelo TelemetriaData, que encapsulan la información individual de cada sensor.

   - El servidor utiliza SimpMessagingTemplate para enviar estos datos formateados al broker STOMP en el canal /topic/telemetria/live.

3. **Broker STOMP integrado**: Actúa como intermediario de los mensajes. Permite que múltiples clientes se suscriban a canales específicos y reciban actualizaciones en tiempo real. En este caso, los mensajes enviados al canal /topic/telemetria/live son distribuidos automáticamente a todos los clientes suscritos.

4. **Cliente Web Visualizador**: Se suscribe al canal /topic/telemetria/live y recibe los datos en tiempo real. Estos datos son graficados utilizando la biblioteca Chart.js para permitir una visualización clara y actualizada del estado de la motocicleta.

Este esquema garantiza una arquitectura desacoplada y escalable, donde el cliente emisor y el cliente visualizador no necesitan conocerse entre sí. El broker intermedio permite escalar el número de clientes suscriptores sin afectar el rendimiento del servidor.

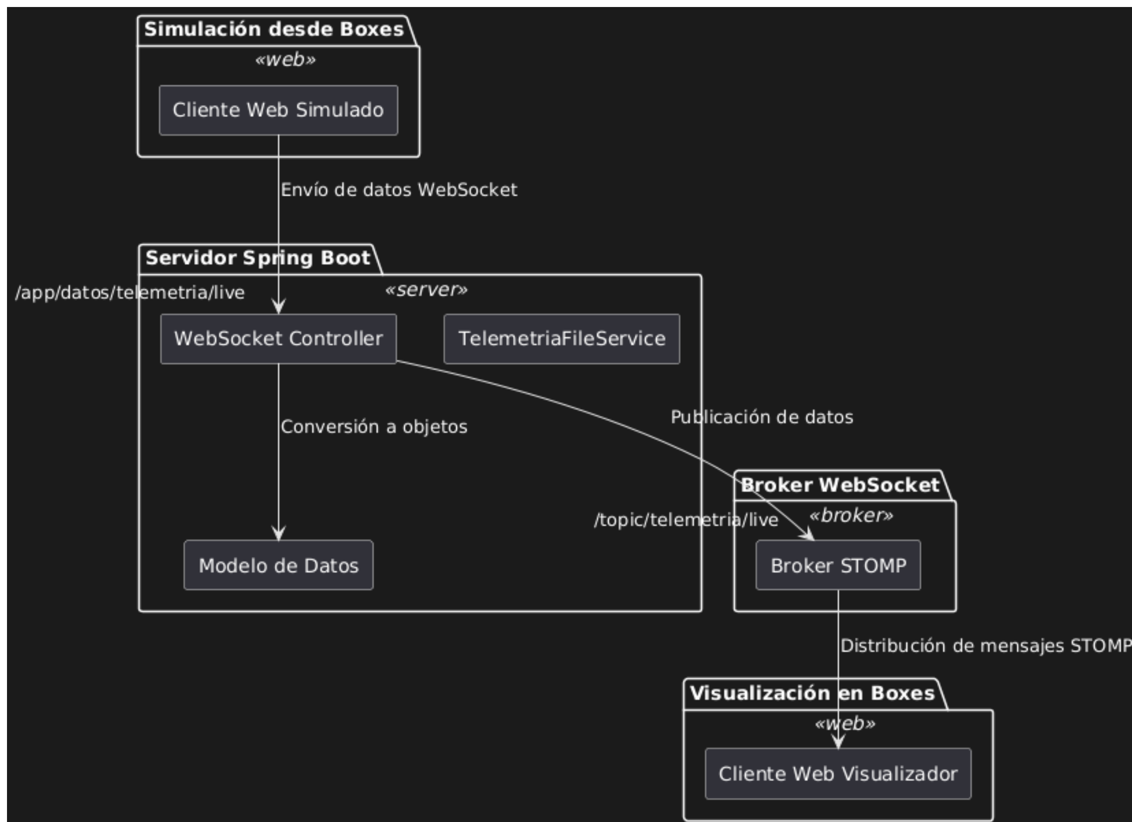


Ilustración 4: Diagrama UML de comunicación entre componentes

5. Implementación

Este capítulo describe el proceso de desarrollo del sistema de telemetría diseñado para la motocicleta de competición EME-25P del equipo PETROL de la UPM. Aunque el sistema final está pensado para funcionar en un entorno físico real, con una Raspberry Pi conectada a la ECU de la moto, en esta primera fase se ha optado por simular la lectura de datos usando un cliente web que lee un fichero CSV.

A pesar de ser una versión inicial, el sistema implementa toda la arquitectura completa de adquisición, envío, procesamiento y visualización en tiempo real de los datos, validando así su funcionamiento y permitiendo futuras integraciones directas con el hardware real.

En este capítulo se detallan las tecnologías empleadas, la estructura del código, los módulos desarrollados y fragmentos representativos que ilustran la lógica implementada.

5.1 Tecnologías Utilizadas

Como se comentó en el apartado 2.3, hay una serie de tecnologías que han sido utilizadas a lo largo del desarrollo del proyecto, por tanto, voy a realizar un pequeño repaso de estas para tenerlas en cuenta.

El sistema desarrollado para la monitorización de telemetría en motocicletas de competición ha sido implementado utilizando una combinación de tecnologías modernas tanto en el lado del servidor como en el lado del cliente. A continuación, se describen las principales herramientas y librerías utilizadas en este proyecto:

Backend (Servidor)

- **Java 21**

  Lenguaje principal del desarrollo del servidor. La elección se justifica por su robustez, capacidad de manejo de concurrencia, y soporte en tiempo de ejecución por parte del framework Spring.

- **Spring Boot 3.4.3**

  Framework principal del backend. Permite el desarrollo rápido de aplicaciones web, con una estructura basada en anotaciones y una configuración sencilla de servicios web y WebSocket.

- **STOMP sobre WebSocket**

  Protocolo de mensajería usado para la comunicación en tiempo real entre el servidor y los clientes. Gracias a Spring Boot, se configura de manera declarativa.

- **SockJS**

  Librería cliente que permite la compatibilidad de WebSocket con navegadores que no soportan esta tecnología de forma nativa.

- **JUnit 5**

  Herramienta utilizada para la realización de pruebas unitarias y de integración del backend.

Frontend (Clientes Web)

- **HTML5, CSS3 y JavaScript**

Tecnologías base utilizadas para construir los dos clientes web (emisor y receptor de datos).

- **Chart.js**

  Librería JavaScript para la representación de datos en gráficas de línea, que permite mostrar en tiempo real los valores recibidos desde el servidor.

- **SockJS-client + STOMP.js**

  Permite establecer la conexión con el servidor WebSocket desde el navegador, enviar datos y suscribirse a canales de escucha.

Entorno de desarrollo

- **IntelliJ IDEA 2024.1**

  Entorno de desarrollo utilizado para el backend, con soporte para Spring Boot y Maven.

- **Navegador Chrome**

  Herramienta principal de prueba y depuración del cliente web.

- **Git + GitHub**

  Control de versiones del proyecto y alojamiento del código fuente.

5.2 Estructura del Código

La implementación del sistema se ha estructurado siguiendo una arquitectura modular clara, favoreciendo la escalabilidad, la reutilización de componentes y la separación de responsabilidades. El proyecto está desarrollado en Java utilizando el framework Spring Boot, lo que permite una organización basada en paquetes con responsabilidades bien definidas.

La estructura general del código fuente es la siguiente:

- **config:** contiene la configuración necesaria para habilitar y gestionar la comunicación vía WebSocket.
- **controller:** maneja los canales STOMP, recibe mensajes desde el cliente y reenvía datos al servidor de visualización.

- **exception:** agrupa las clases de errores personalizadas, como LecturaFicheroException, para manejar de forma clara situaciones anómalas.
- **model:** define el modelo de datos que representa la información de telemetría, incluyendo las clases base y sus extensiones para cada tipo de sensor.
- **service:** incluye la lógica relacionada con la lectura de datos desde el fichero CSV y el formateo necesario para su procesamiento.
- **TelemetriaUpmmsApplication:** es la clase que lanza el servidor Spring Boot y configura el contexto de ejecución.

5.3 Lógica del Servidor

El servidor es el núcleo del sistema, encargado de recibir los datos de telemetría, procesarlos y redistribuirlos a los clientes conectados para su visualización en tiempo real. Esta lógica ha sido desarrollada utilizando el framework Spring Boot, aprovechando su soporte para aplicaciones reactivas y su integración con WebSocket y STOMP para la transmisión eficiente de datos.

5.3.1 Proceso General

El servidor expone un endpoint WebSocket accesible desde los clientes, que se conectan mediante SockJS y STOMP. Una vez conectados, los clientes pueden enviar datos a rutas específicas o suscribirse a canales donde el servidor publica actualizaciones en tiempo real.

En el sistema actual, los datos se simulan mediante un fichero CSV con valores reales grabados previamente. Un cliente de simulación lee línea a línea ese fichero, emula el envío desde la moto y manda esos datos como cadenas de texto CSV (formato string) a través del canal /app/datos/telemetria/live. Esta información llega al servidor en bruto y se procesa mediante un controlador que extrae y formatea los datos.

5.3.2 Controlador y Recepción de Datos

El componente principal que gestiona la recepción y procesamiento de datos es la clase TelemetriaController. Este controlador tiene definidas varias rutas mediante la anotación @MessageMapping, que actúan como equivalentes a los endpoints REST tradicionales, pero para WebSocket.

En particular, el método datosCliente() recibe mensajes con los datos en crudo desde la moto (o simulador). Este método se encarga de:

1. Parsear la cadena CSV separándola por comas.
2. Convertir cada campo a su tipo correspondiente (int o double).
3. Actualizar las instancias de los objetos del modelo de datos.
4. Crear una nueva instancia de TelemetriaData con la información formateada.
5. Llama a la funcion guardarEnCSV para generar un archivo .CSV con los datos leidos
6. Enviar el objeto completo a través del canal /topic/telemetria/live a todos los clientes suscritos.

```
@MessageMapping("/datos/telemetria/live")
public void datosCliente(@Payload String message) {

    String[] datos = message.split(",");

    rpm.setValue(Integer.parseInt(datos[0]));
    inc.setValue(Double.parseDouble(datos[1]));
    gear.setValue(Integer.parseInt(datos[2]));
    sus.setFrontValue(Double.parseDouble(datos[3]));
    sus.setRearValue(Double.parseDouble(datos[4]));
    bat.setValue(Double.parseDouble(datos[5]));
    temp.setValue(Double.parseDouble(datos[6]));

    TelemetriaData data = new TelemetriaData(rpm, inc, gear, sus,
bat, temp);

    guardarEnCSV(data);

    messagingTemplate.convertAndSend("/topic/telemetria/live",
data);
}
```

Código del método datosCliente

5.3.3 Servicio de Lectura de Ficheros

Aunque en el diseño original del sistema los datos pueden provenir de un fichero, actualmente ese sistema no está implementado del todo. Actualmente está esta versión preliminar funciona pero es posible que genere errores de lectura. Este archivo contiene líneas con datos secuenciales representando valores reales tomados durante una prueba.

El servicio TelemetriaFileService se encarga de:

- Leer el archivo línea a línea, ignorando la cabecera.
- Procesar cada línea y convertirla en un objeto TelemetriaData utilizando el método readTelemetria().
- Devolver una lista con los objetos ya procesados para poder simular su envío a través del canal WebSocket.

```
@MessageMapping("/datos/telemetria/file")
public void enviarData() {
    List<TelemetriaData> datos =
telemetriaFileService.leerDatosDesdeFichero("src/run/resources/test
data/vuelta_jarama_moto3.csv");
    for (TelemetriaData dato : datos) {
        messagingTemplate.convertAndSend("/topic/telemetria/file",
dato.getFormattedData());
        try {
            Thread.sleep(500); // Simula un envío cada medio
                                   segundo
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
}
```

Código de enviarData

5.3.4 Envío de Datos

El envío de los datos a los clientes conectados se realiza utilizando el objeto SimpMessagingTemplate, que permite publicar objetos a un destino específico (/topic/telemetria/live). Los clientes suscritos a este canal reciben automáticamente los nuevos datos y los procesan para su visualización gráfica en tiempo real.

Esto permite un flujo de información constante, con baja latencia y gran versatilidad, ya que múltiples clientes pueden recibir los datos al mismo tiempo sin recargar el servidor.

5.4 Cliente Simulador de la Moto

Dado que el sistema aún no está conectado físicamente a la motocicleta ni se dispone de una Raspberry Pi configurada para la transmisión de datos en tiempo real, se ha desarrollado un cliente simulador. Este simulador emula el comportamiento del módulo que se integrará en la moto, permitiendo probar y validar la infraestructura del servidor y el flujo de datos completo.

5.4.1 Objetivo del Simulador

El propósito principal de este simulador es leer los datos históricos desde un fichero CSV y enviarlos línea por línea al servidor en intervalos regulares, emulando un flujo de datos en tiempo real como si provinieran directamente desde la moto.

Este enfoque permite validar:

- El procesamiento de datos en el servidor.
- La conversión de los datos crudos a objetos formateados.
- La correcta difusión de los datos a los clientes conectados.
- La visualización en tiempo real en las gráficas del cliente de boxes.

5.4.2 Funcionamiento del Cliente

El cliente está implementado en un fichero HTML/JavaScript, que se ejecuta en un navegador web. Este cliente establece una conexión WebSocket con el servidor utilizando SockJS y STOMP, y una vez conectado:

1. Realiza una petición fetch() al fichero CSV (vuelta_jarama_moto3.csv).

2. Divide el contenido en líneas (ignorando la cabecera).

3. En intervalos de 500 ms (simulando la frecuencia de muestreo), envía cada línea como un string al endpoint /app/datos/telemetria/live del servidor.

4. Los mensajes se imprimen en consola para seguimiento y depuración.

```
5. function start() {
       fetch('vuelta_jarama_moto3.csv')
           .then(response => response.text())
           .then(csvText => {
               const lines = csvText.trim().split('\n').slice(1);
   // omitir encabezado

               let i = 0;
               const interval = setInterval(() => {
                   if (i >= lines.length) {
                       clearInterval(interval);
                       console.log('Envío finalizado');
                       return;
                   }

                   const line = lines[i];
                   stompClient.send('/app/datos/telemetria/live',
   {}, line);
                   console.log('Enviado:', line);

                   i++;
               }, 500);
           });
   }
```

Función start que envía los datos de un fichero de prueba

Este simulador ha permitido realizar múltiples pruebas de integración, visualización y análisis sin necesidad de depender del hardware de la motocicleta.

5.4.3 Adaptación Futura

Este componente será reemplazado en el futuro por una Raspberry Pi conectada físicamente a la ECU de la moto mediante un adaptador USB-OBD. El flujo de datos se mantendrá, pero en lugar de provenir de un fichero, se capturarán en

tiempo real desde los sensores del vehículo y se enviarán usando un programa embebido desarrollado específicamente para este hardware.

5.5 Cliente de Boxes y Visualización de Datos

Uno de los componentes más visibles y esenciales del sistema es el cliente web que opera en boxes, cuyo objetivo principal es mostrar en tiempo real los datos de telemetría que llegan desde el servidor. Este cliente facilita al equipo técnico el seguimiento del comportamiento de la motocicleta durante las rodadas de prueba, permitiendo una evaluación inmediata de las condiciones del vehículo.

5.5.1 Funcionalidad General

El cliente se ejecuta en un navegador web moderno y está desarrollado completamente en HTML, CSS y JavaScript. Hace uso de las librerías:

- **SockJS** y **STOMP.js** para la conexión en tiempo real con el servidor mediante WebSockets.
- **Chart.js** para la generación dinámica de gráficas que visualizan los datos recibidos.

El funcionamiento se puede resumir en los siguientes pasos:

1. Se establece una conexión con el servidor al endpoint /ws.
2. Se realiza la suscripción al canal /topic/telemetria/live.
3. A medida que los datos llegan del servidor (en formato JSON), se procesan y actualizan automáticamente los gráficos correspondientes en el cliente.
4. Se representan 7 gráficas: RPM, Inclinación, Marcha, Suspensión delantera, Suspensión trasera, Temperatura del motor y Voltaje de batería.

En las siguientes imágenes se puede observar la interfaz que observarían los clientes desde boxes, viendo los datos actualizarse en tiempo real con su respectivos valores y marcas temporales.

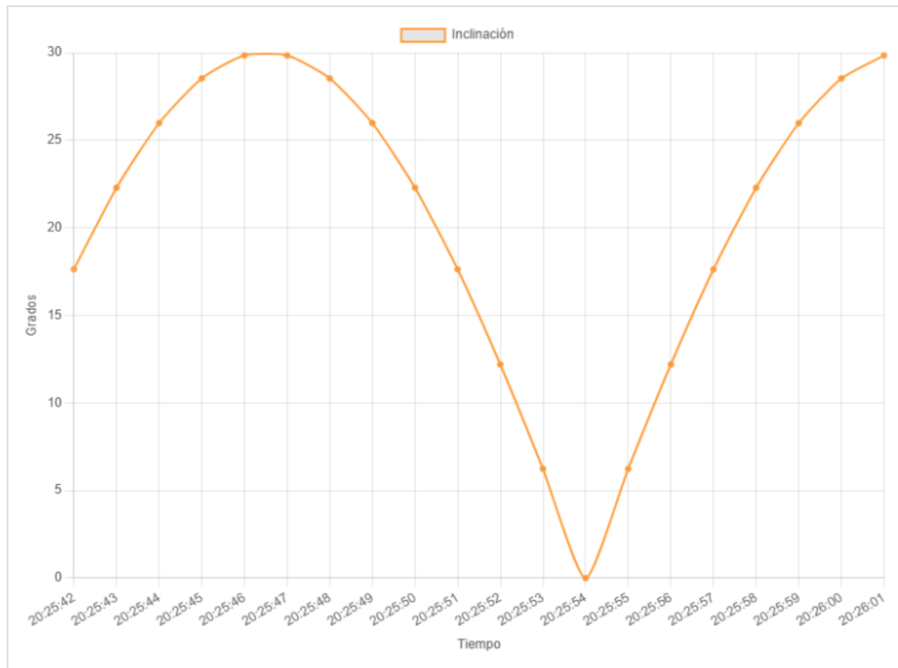
Cada uno de los sensores tiene su propia gráfica y color para poder ser distinguidos de forma rápida y sencilla. Para esta demostración se han

utilizado unos valores generados por una IA que simulan una vuelta al circuito del Jarama (los datos no son reales ni tienen por qué ser exactos y correctos, son una mera representación para la muestra del correcto funcionamiento del sistema).

Telemetría en Tiempo Real

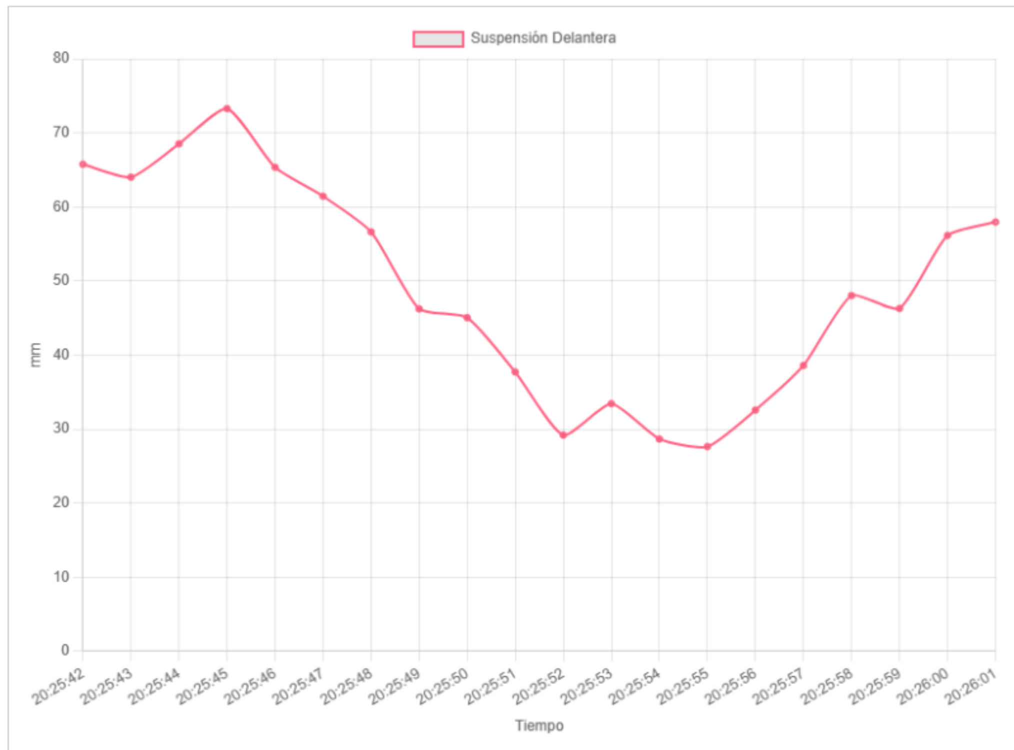
5.

Ilustración 5: Valores RPM



6.

Ilustración 6: Valores Inclinación



7.

Ilustración 7: Valores Suspensión Delantera

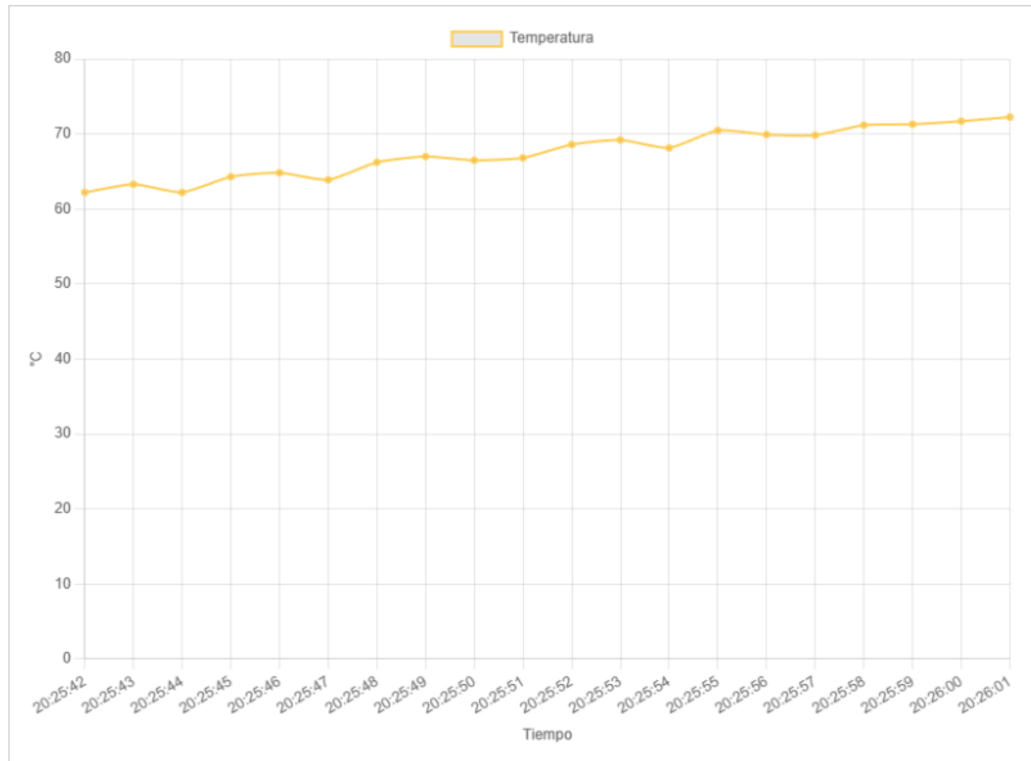
8. 

Ilustración 8: Valores Temperatura

5.5.2 Organización y Diseño Visual

Cada gráfica se ubica en una sección distinta, con una leyenda identificativa, colores personalizados y ejes correctamente etiquetados. El eje X representa el tiempo (tomado como la hora local al recibir el dato), y el eje Y representa el valor medido.

Además, se ha cuidado el aspecto visual para:

- Reducir la altura de las gráficas y maximizar el ancho para una mejor lectura.
- Utilizar colores diferenciados por sensor.
- Mostrar únicamente los últimos 20 valores para mantener claridad y evitar saturación visual.

```
charts.rpm.data.labels = dataStorage.labels;
charts.rpm.data.datasets[0].data = dataStorage.rpm;
charts.rpm.update();
```

Pedazo del código que actualiza las gráficas, en este caso es de la gráfica de RPM

5.5.3 Validación del Comportamiento

Durante las pruebas, el cliente fue capaz de recibir, interpretar y mostrar correctamente todos los datos emitidos desde el simulador. Se verificó que:

- Las gráficas se actualizan sin interrupciones.
- No se producen errores de conexión durante el envío.
- Los datos representados se corresponden con los valores enviados.

Este módulo es completamente funcional y será el mismo que se utilice cuando el sistema se conecte con la moto real, sin necesidad de grandes modificaciones, lo que permite una transición directa entre la fase de simulación y la fase operativa.

5.6 Servidor Web con Spring Boot

El **servidor** constituye el **núcleo lógico** del sistema, encargado de recibir los datos emitidos por el cliente embarcado en la motocicleta (simulado en este caso mediante un CSV), procesarlos y redistribuirlos a todos los clientes suscritos. Este módulo se ha desarrollado utilizando el framework **Spring Boot**, que permite construir aplicaciones web modernas de forma sencilla, estructurada y altamente mantenible.

5.6.1 Estructura General del Servidor

La aplicación sigue una arquitectura de paquetes organizada por funcionalidades:

```
com.telemetria.telemetria
|
├── controller        // Controladores WebSocket y HTTP
├── service           // Servicios de lógica de negocio
├── model             // Clases del modelo de datos (ECUData y derivadas)
├── config            // Configuración de WebSockets
├── exception         // Excepciones personalizadas
└── TelemetriaUpmmsApplication.java // Clase principal
```

Esquema en consola del sistema de ficheros del servidor

5.6.2 Comunicación con los Clientes

El servidor expone un punto de entrada WebSocket en /ws, gestionado por la clase WebSocketConfig. Este punto de entrada permite la comunicación bidireccional con clientes STOMP.

Cuando un cliente envía un mensaje al endpoint /app/datos/telemetria/live, este es gestionado por el TelemetriaController. Allí, el mensaje es interpretado como una línea de CSV, parseado y convertido en un objeto TelemetriaData. A continuación, el objeto es emitido al canal /topic/telemetria/live, al cual están suscritos todos los clientes visuales.

5.6.3 Configuración del WebSocket

El siguiente fragmento muestra la configuración del WebSocket:

```
@Configuration
@EnableWebSocketMessageBroker // Activar los websockets
public class WebSocketConfig implements
WebSocketMessageBrokerConfigurer {


    @Override
    public void configureMessageBroker(MessageBrokerRegistry
registry) {
        registry.enableSimpleBroker("/topic");
        registry.setApplicationDestinationPrefixes("/app");
    }

    @Override
    public void registerStompEndpoints(StompEndpointRegistry
registry) {

registry.addEndpoint("/ws").setAllowedOrigins("http://localhost:
63342").withSockJS();


    }

}
```

Código de la clase WebSocketConfig

Esta configuración indica a Spring que todos los mensajes enviados al prefijo /app serán dirigidos a los controladores, y que las respuestas se emitirán a través del prefijo /topic.

5.6.4 Recepción de Datos y Procesamiento

La función que recibe los datos desde el cliente simplemente los parsea (dividiéndolos por comas) y los convierte en objetos de tipo RPMData, InclinationData, GearData, etc. Estos datos se encapsulan en una instancia de TelemetriaData y se publican para su consumo por parte del cliente.

Si el sistema leyese esta línea:

```
7000,0.0,1,49.05,71.42,13.0,90.24
```

Al hacer un getFormattedData() de cada dato quedaría lo siguiente:

```
RPM: 7000
Inclination: 0.0°
Current Gear: 1
Suspension Compression - Front: 49.05mm, Rear: 71.42mm
Battery Voltage: 13.0V
Temperature: 90.24°C
```

Y si lo hicieses con el getFormattedData() del telemetriaData te quedaría como:

```
RPM: 7000, Inclination: 0.0°, Gear: 1, Front Suspension:
49.05mm, Rear Suspension: 71.42mm, Battery: 13.0V, Temperature:
90.24°C
```

5.6.5 Robustez y validaciones

Aunque el flujo principal está basado en una simulación, se han implementado mecanismos de validación y control de errores:

- Si los datos recibidos no tienen el formato esperado, se descartan silenciosamente.
- Se define una excepción personalizada LecturaFicheroException que se lanza si el archivo de datos no puede ser leído correctamente.
- El servidor es tolerante a errores en los mensajes, asegurando que una línea defectuosa no interrumpe la recepción del resto.

5.6.6 Estado actual y expansión futura

En su estado actual, el servidor cumple su propósito dentro del entorno simulado. Cuando se complete la integración con el hardware real en la motocicleta, este mismo servidor será capaz de recibir los datos reales sin necesidad de modificar su lógica central, ya que espera un mensaje de texto con el mismo formato que el simulado.

5.7 Gestión de Errores y Pruebas

Durante el desarrollo del sistema, se ha prestado especial atención a la robustez del software y a su capacidad para gestionar errores comunes de forma controlada y clara. Además, se ha implementado una batería de pruebas unitarias y de integración para asegurar que cada parte del sistema funcione correctamente, tanto de forma aislada como en conjunto.

5.7.1 Gestión de Errores

El sistema implementa control de errores principalmente en el módulo de lectura de datos y en los procesos de transmisión. Por ejemplo, si el fichero CSV no se encuentra, está malformado o contiene datos no válidos, el sistema lanza una excepción personalizada llamada LecturaFicheroException. Esta clase permite identificar claramente los errores relacionados con la adquisición de datos y distinguirlos de otros posibles fallos.

Esto proporciona una forma clara y modular de manejar errores, permitiendo al desarrollador depurar fácilmente problemas y al sistema continuar funcionando cuando sea posible, notificando al usuario o dejando trazas adecuadas en el registro de logs.

```
public List<TelemetriaData> leerDatosDesdeFichero(String file)
throws LecturaFicheroException {
    try (BufferedReader br =
Files.newBufferedReader(Paths.get(file))) {
        return br.lines()
                .skip(1) // Saltar la cabecera
                .map(this::readTelemetria)
                .collect(Collectors.toList());
    } catch (IOException e) {
        throw new LecturaFicheroException("Error al leer el
fichero de telemetría", e);
    }
}
```

Código de leerDatosDesdeFichero() en el que se observa el uso de un error personalizado

5.7.2 Pruebas Unitarias

Para validar el correcto funcionamiento de los componentes del sistema, se ha implementado un conjunto de pruebas unitarias utilizando el framework JUnit 5. Las pruebas incluyen:

- Verificación del funcionamiento de los constructores de las clases del modelo (RPMData, SuspCompData, TelemetriaData, etc.).
- Validación de los valores devueltos por los métodos getValue() y getFormattedData().
- Prueba de integración básica para comprobar que el controlador procesa correctamente los datos formateados y los reenvía al canal correspondiente.
- Prueba específica que simula la excepción LecturaFicheroException para validar la correcta gestión de errores.
- Una serie de pruebas manuales para comprobar la correcta transferencia de datos y enviados y recibidos en el cliente.

Estas pruebas pueden ejecutarse de forma independiente al servidor gracias al uso de JUnit.

6. Conclusión y Trabajo Futuro

6.1 Conclusiones

El desarrollo de este sistema de telemetría representa un avance significativo en la capacidad del equipo UPM Motostudent para monitorizar, analizar y optimizar el rendimiento de su motocicleta de competición. A lo largo de este Trabajo de Fin de Grado se ha logrado materializar una solución tecnológica robusta y escalable que integra múltiples disciplinas: desde la adquisición y procesamiento de datos en tiempo real, hasta la visualización interactiva y accesible para los ingenieros en pista.

Aunque en esta primera fase el sistema funciona con datos simulados, la arquitectura planteada está diseñada con la visión clara de ser implantada en

un entorno real, conectando directamente con los sensores de la motocicleta mediante hardware como Raspberry Pi y lectores OBD-II. Este enfoque garantiza que la herramienta no solo sirva para análisis retrospectivos, sino que se convierta en un aliado imprescindible en las sesiones de pruebas y competiciones en vivo.

El proyecto ha puesto especial énfasis en la modularidad y la escalabilidad, lo que permitirá incorporar nuevas variables de telemetría, mejorar la experiencia de usuario en el cliente web y ampliar las funcionalidades, como la integración de alertas en tiempo real, el uso de una base de datos relacional para guardar los datos, o incluso la implementación de algoritmos avanzados de análisis de datos que apoyen la toma de decisiones del equipo.

Además, el desarrollo ha abierto la puerta a múltiples líneas de mejora y expansión. Desde la optimización de la transmisión y el procesamiento de datos en entornos con limitaciones de ancho de banda, hasta la mejora de la calidad y variedad de las gráficas, que pueden evolucionar hacia visualizaciones 3D o dashboards personalizados según el rol del usuario. También se contempla la exploración de diferentes modelos de Raspberry Pi para balancear rendimiento y consumo energético, garantizando así la viabilidad práctica en la moto.

Este trabajo ha supuesto un gran reto tanto a nivel personal como profesional, ya que me ha permitido combinar de manera directa los conocimientos adquiridos durante mis estudios con una de mis pasiones más profundas: las motos y el mundo del motor. La experiencia me ha brindado una valiosa oportunidad para adoptar una perspectiva práctica y realista del desarrollo de software aplicado a sistemas embebidos y entornos de competición, fortaleciendo habilidades de trabajo en equipo, resolución de problemas complejos y gestión de proyectos tecnológicos.

En definitiva, este proyecto no solo refleja un logro técnico, sino que sienta las bases de un sistema integral, versátil y preparado para afrontar los desafíos técnicos y operativos del automovilismo universitario. Su futuro desarrollo promete no solo perfeccionar la calidad y fiabilidad de los datos, sino también convertirse en un recurso estratégico fundamental para el éxito del equipo, marcando un antes y un después en la manera en que se entiende y mejora el rendimiento de la motocicleta en competición.

Por último, quisiera expresar mi más sincero agradecimiento al equipo UPM Motostudent Petrol por permitirme formar parte de este proyecto tan ambicioso y emocionante. Ha sido un privilegio poder aportar mi granito de arena en el diseño y desarrollo de una motocicleta de competición universitaria, aprendiendo y creciendo junto a un grupo de estudiantes y compañeros con un alto nivel técnico y una gran pasión por el motor. Asimismo, quiero agradecer especialmente a mi tutor, Ángel Herranz Nieva, por haber aceptado la propuesta

de este Trabajo de Fin de Grado y por su constante apoyo, ánimo y asesoramiento durante todas las fases del desarrollo, sin los cuales este proyecto no habría sido posible.

6.2 Resultados

Durante el desarrollo de este Trabajo de Fin de Grado se ha conseguido implementar un sistema funcional para la lectura, procesamiento y visualización de datos de telemetría destinados a una motocicleta de competición universitaria, simulando su funcionamiento real a partir de un fichero de datos.

A nivel técnico, los principales resultados alcanzados son los siguientes:

- **Lectura y tratamiento de datos de telemetría**

Se ha desarrollado un módulo capaz de leer archivos CSV que contienen datos reales de una sesión en pista, estructurados con las principales variables extraídas de la ECU: revoluciones por minuto (RPM), grado de inclinación, marcha engranada, compresión de suspensión delantera y trasera, voltaje de batería y temperatura del motor.

La información se procesa y transforma en objetos del tipo TelemetriaData, utilizando una jerarquía de clases bien estructurada, lo que facilita la extensión a futuros sensores.

- **Servidor WebSocket para transmisión en tiempo real**

Se ha implementado un servidor basado en Spring Boot con soporte para WebSockets mediante el protocolo STOMP, capaz de recibir y reenviar datos de telemetría en tiempo real a los clientes conectados.

Este servidor actúa como nodo intermedio entre la motocicleta y los ordenadores situados en boxes, simulando la arquitectura prevista para el sistema final.

- **Cliente web para visualización gráfica**

Se ha diseñado un cliente web utilizando HTML5, JavaScript y Chart.js, que se conecta al servidor mediante WebSocket y muestra en tiempo real todas las variables de telemetría a través de gráficos dinámicos.

El cliente actual permite visualizar múltiples variables simultáneamente (RPM, marcha, temperatura, entre otras) y ha sido ajustado para ofrecer una experiencia visual clara y escalable.

- **Sistema modular y escalable**

Toda la arquitectura del sistema ha sido diseñada de forma modular para facilitar su mantenimiento y futura ampliación.

Se ha aplicado una separación adecuada entre la lógica del controlador, el servicio de lectura de datos, el modelo de datos y la interfaz de usuario, siguiendo principios recomendados de diseño de software.

- **Validación mediante pruebas automatizadas y manuales**

Se han implementado pruebas unitarias empleando JUnit, que permiten validar el correcto funcionamiento de los constructores y del procesamiento de datos.

A su vez el sistema ha sido testeado a mano para comprobar detalles que las pruebas automatizadas no pudiesen detectar o que fuesen demasiado complejos para poder ser automatizados.

Estos resultados demuestran que el sistema es plenamente funcional en un entorno de pruebas y sientan una base sólida para su futura integración en la motocicleta real del equipo UPM Motostudent. Aunque actualmente se trabaja con datos simulados a través de un archivo CSV, el sistema está preparado para integrarse con una Raspberry Pi y un lector OBD-II, lo que permitirá su uso durante las sesiones de pruebas reales en pista.

6.3 Trabajo Futuro

El sistema desarrollado durante este Trabajo de Fin de Grado constituye una base funcional y escalable para la telemetría en motocicletas de competición universitaria. No obstante, para que el sistema sea plenamente operativo e implementable en la moto real del equipo UPM Motostudent, es necesario avanzar en varias áreas y completar ciertas funcionalidades. A continuación, se detallan los principales aspectos pendientes y posibles mejoras:

1. Completar la implementación de servicios y gestión de sesiones WebSocket

- Actualmente, el servicio TelemetriaService está parcialmente desarrollado y no se encuentra en uso dentro del sistema.
- Este componente es esencial para la gestión eficiente de las conexiones WebSocket, controlando múltiples sesiones de clientes y garantizando la correcta distribución de datos en tiempo real.

- Su integración futura mejorará la escalabilidad y robustez del sistema de comunicación entre la moto y los ordenadores en boxes.

2. Desarrollo y optimización del sistema de almacenamiento en fichero

- El módulo de lectura de datos desde archivos CSV funciona parcialmente para sesiones históricas, aún está en desarrollo, pero el sistema de almacenamiento de la rodada actual en curso funciona de forma simple.
- Se podrían explorar opciones para mejorar el formato y eficiencia del almacenamiento, considerando formatos binarios o bases de datos ligeras para facilitar la gestión y recuperación de grandes volúmenes de datos.

3. Mejora de la calidad y usabilidad en la visualización de datos

- El cliente web actual ofrece visualización mediante gráficos básicos con Chart.js, adecuados para un primer prototipo.
- Se recomienda investigar y evaluar librerías de gráficos más avanzadas que permitan mayor interactividad, personalización, y mejor rendimiento en tiempo real.
- También es relevante optimizar el diseño de la interfaz para facilitar la interpretación rápida y precisa de los datos durante la competición, incluyendo alertas visuales y configuración dinámica de las variables mostradas.

4. Selección y adaptación del hardware embarcado (Raspberry Pi)

- En la fase de integración real, será necesario encontrar una placa capaz de ejecutar el código necesario y debe tener algún módulo de conexión por USB y otro para una tarjeta SD que permita el acceso a internet. Por tanto, se deberán evaluar diferentes modelos en función de criterios como capacidad de procesamiento, consumo energético, conectividad y compatibilidad con sensores y dispositivos OBD-II.
- A su vez, podría ser necesario reajustar el actual código o añadir nuevo para que pueda conectarse a internet.
- Además, se deberá diseñar un sistema robusto de montaje y protección física para el hardware en la moto, garantizando su funcionamiento en condiciones de vibración y temperatura propias de la competición.

5. Integración con hardware real y adquisición de datos OBD-II

- Hasta ahora, el sistema ha trabajado con datos simulados mediante archivos CSV. La integración con la Raspberry Pi y un lector OBD-II es fundamental para capturar datos en vivo de la motocicleta.
- Se deberá implementar el cliente que lea en tiempo real los datos desde el hardware OBD-II, validando la sincronización y consistencia de la información recibida.
- También será importante asegurar la comunicación continua y sin interrupciones entre la Raspberry Pi y el servidor central durante las sesiones de competición.

6. Extensión del modelo de datos y soporte para nuevos sensores

- El diseño basado en una jerarquía de clases para la telemetría permite la incorporación futura de nuevos sensores y variables.
- Se prevé ampliar el sistema para incluir datos adicionales relevantes para el equipo, como presión de neumáticos, consumo de combustible o datos ambientales.
- Esto requerirá adaptar tanto el procesamiento en el servidor como la visualización en el cliente web.

7. Pruebas, validación y optimización del sistema completo

- Para garantizar la fiabilidad en condiciones reales, será necesario realizar pruebas exhaustivas integrando todos los componentes hardware y software, añadiendo las pruebas (JUnit Tests) necesarios y realizando multitud de pruebas.
- Se recomienda planificar campañas de pruebas en pista para ajustar parámetros, detectar posibles fallos y optimizar el rendimiento del sistema en escenarios reales.
- También será útil desarrollar herramientas de diagnóstico y monitoreo para facilitar el mantenimiento y mejora continua.

6.4 Análisis de Impacto

El desarrollo de este sistema de monitorización y visualización de datos de telemetría en tiempo real tiene un impacto significativo en varios ámbitos, que contribuyen al valor y la utilidad del proyecto.

Impacto técnico:

El sistema mejora notablemente el proceso de desarrollo y ajuste de la motocicleta de competición EME-25P, al facilitar la visualización inmediata de los datos provenientes de la centralita electrónica (ECU). Esto permite a los técnicos y mecánicos tomar decisiones informadas durante las rodadas de prueba, acelerando el ciclo de ensayo y error para optimizar el rendimiento de la moto. Además, al diseñar la arquitectura del sistema con un enfoque modular y escalable, se facilita la integración futura de mejoras, como la conexión directa con la ECU real y la incorporación de nuevos sensores o funcionalidades, sin necesidad de rehacer completamente el sistema. Esto convierte al proyecto en una base sólida para desarrollos posteriores.

Impacto económico:

El sistema contribuye a la optimización de recursos y ahorro de tiempo durante las pruebas. Al disponer de datos en tiempo real, el equipo puede detectar y corregir rápidamente posibles fallos o ajustar parámetros técnicos sin necesidad de repetir excesivas sesiones de ensayo, lo que reduce costes asociados a combustible, desgaste mecánico y horas-hombre. Asimismo, al emplear una solución basada en dispositivos embebidos accesibles y relativamente económicos, como la Raspberry Pi, se consigue un sistema asequible que puede ser implementado y mantenido por equipos con recursos limitados, como es habitual en entornos universitarios.

Impacto social y formativo:

Este proyecto aporta un valor educativo relevante para los miembros del equipo UPM Motostudent Petrol, especialmente para los estudiantes del departamento de electrónica. La participación en el diseño e implementación de un sistema real de telemetría fomenta el aprendizaje práctico de tecnologías modernas como comunicaciones WebSocket, programación en Java, procesamiento de datos en tiempo real y sistemas embebidos. Esto potencia la formación técnica y profesional de los integrantes, preparándolos para futuros retos en la industria automovilística y de competición.

Impacto en la competición:

Aunque las reglas oficiales de la competición no permiten el uso de telemetría en carrera, la aplicación del sistema durante las sesiones de prueba ofrece una ventaja competitiva indirecta. Al mejorar la calidad y rapidez del análisis de datos, el equipo puede realizar ajustes más precisos y eficientes en la moto, elevando el nivel de rendimiento y fiabilidad durante la competición. De esta manera, el sistema contribuye a maximizar el potencial del vehículo sin contravenir la normativa.

7. Bibliografía y referencias

«Getting Started | Using WebSocket to build an interactive web application», *Getting Started | Using WebSocket To Build An Interactive Web Application*. https://spring.io/guides/gs/messaging-stomp-websocket

«Chart.js | chart.js». https://www.chartjs.org/docs/latest/

The Coding Train, «1.1: fetch() - Working With Data & APIs in JavaScript», *YouTube*. 17 de mayo de 2019. [En línea]. Disponible en: https://www.youtube.com/watch?v=tc8DU14qX6I

The Coding Train, «1.2 Tabular Data - Working With Data & APIs in JavaScript», *YouTube*. 23 de mayo de 2019. [En línea]. Disponible en: https://www.youtube.com/watch?v=RfMkdvN-23o

The Coding Train, «1.3: Graphing with Chart.js - Working With Data & APIs in JavaScript», *YouTube*. 23 de mayo de 2019. [En línea]. Disponible en: https://www.youtube.com/watch?v=5-ptp9tRApM

Sockjs, «GitHub - sockjs/sockjs-client: WebSocket emulation - Javascript client», *GitHub*. https://github.com/sockjs/sockjs-client

spring-guides, «GitHub - spring-guides/gs-messaging-stomp-websocket: Using WebSocket to build an interactive web application :: Learn how to the send and receive messages between a browser and the server over a WebSocket», *GitHub*. https://github.com/spring-guides/gs-messaging-stomp-websocket

UPM Motostudent, "UPM Motostudent Blog," 2025. [En línea]. Disponible en: https://blogs.upm.es/upm-motostudent/

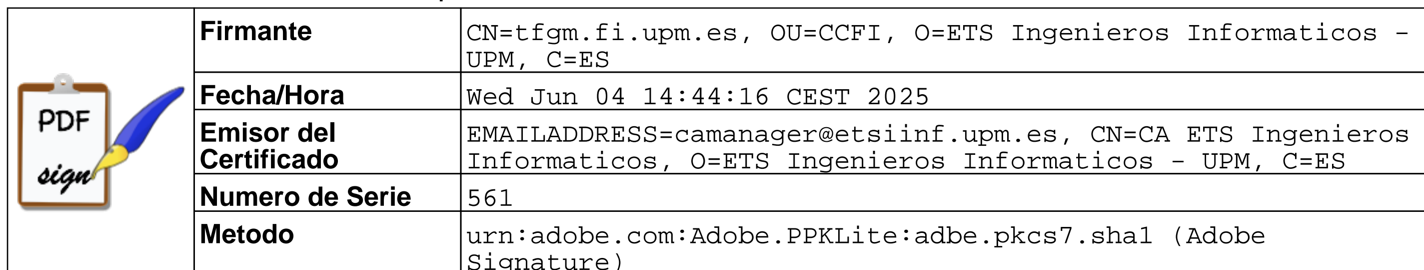
«JUnit 5». https://junit.org/junit5/

MotoStudent, "Página oficial de MotoStudent,". https://www.motostudent.com/

«AIM ECU Taipan - Documentation». https://www.aimsports.com/en/products/ecu-taipan/documentation.htm

"Spring Boot Reference Documentation," *Spring*, [Online]. Available: https://docs.spring.io/spring-boot/documentation.html

Este documento esta firmado por



	Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Wed Jun 04 14:44:16 CEST 2025
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)