

Trabajo Fin de Grado

Grado en Ingeniería en Tecnologías Industriales

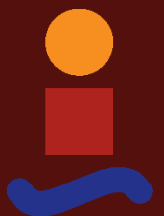
Transformación digital de un Servicio Técnico de Transporte

Autor: Alejandro Lucas Rodríguez

Tutor: Miguel Torres García

**Dpto. de Ingeniería Energética
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla**

Sevilla, 2026



Trabajo Fin de Grado
Grado en Ingeniería en Tecnologías Industriales

Transformación digital de un Servicio Técnico de Transporte

Autor:
Alejandro Lucas Rodríguez

Tutor:
Miguel Torres García

Dpto. de Ingeniería Energética
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2026

Trabajo Fin de Grado: Transformación digital de un Servicio Técnico de Transporte

Autor: Alejandro Lucas Rodríguez

Tutor: Miguel Torres García

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

A mi familia por su apoyo incondicional y su lucha diaria para que esto sea posible. A mi tutor *Miguel Torres García*, por su paciencia, orientación y disposición. A *Ramón González Carvajal* por el asesoramiento sobre la transformación digital. A todos aquellos que en mayor o menor medida contribuyeron a que este trabajo fuese posible.

Alejandro Lucas Rodríguez
Sevilla, 2026

Resumen

Este trabajo parte de un problema muy concreto: el servicio técnico ETMOV gestiona los expedientes de vehículos históricos mediante correos, carpetas compartidas y plantillas en distintos formatos. Este modo de trabajo genera conflictos de versiones, dificulta seguir la trazabilidad de los cambios y obliga a invertir muchas horas en tareas rutinarias a medida que aumenta el número de expedientes.

La solución propuesta organiza cada expediente como un objeto digital con estructura definida y distribuye el trabajo del equipo en dos perfiles principales: *Técnico* y *Supervisor*. El flujo comienza con una hoja de entrada donde el técnico introduce los datos básicos del vehículo; a partir del número de bastidor, el sistema consulta registros de homologación europeos y el historial interno para completar la ficha técnica. Las fotografías que se aportan se procesan con un módulo de análisis de imagen que detecta elementos relevantes y verifica la matrícula, y a partir de esa información se genera un primer borrador del informe técnico, que el técnico revisa y ajusta antes de enviarlo al supervisor para su validación y firma (Técnico → Supervisor → Firmado).

La plataforma se apoya en una arquitectura propia, sin dependencia de servicios en la nube. Se despliega sobre infraestructura virtualizada con Proxmox VE y almacenamiento ZFS, utiliza Kubernetes (MicroK8s) para orquestar los contenedores y combina tres bases de datos con funciones diferenciadas: MongoDB para los expedientes, MinIO para los ficheros y Qdrant para las búsquedas semánticas. La API principal está desarrollada con Node.js/Express y el servicio de procesamiento documental y asistencia a la redacción con Python/FastAPI. El sistema puede conectarse con gestores documentales como SharePoint u OneDrive, de forma que el historial de versiones y el archivo final de documentos se integran en las herramientas corporativas existentes. Además, un ciclo de integración y despliegue continuo basado en GitLab permite automatizar la construcción y actualización del software, manteniendo un registro claro de las versiones instaladas.

Abstract

This work starts from a very concrete problem: the ETMOV technical service manages historical vehicle certification files through emails, shared folders and templates in different formats. This way of working produces version conflicts, makes it difficult to track changes over time and forces the team to spend many hours on repetitive tasks as the number of active files grows.

The proposed solution models each certification file as a structured digital object and organises the work of the team around two main roles: *Technician* and *Supervisor*. The process begins with an intake sheet where the technician enters the vehicle's basic data; from the VIN, the system queries European type-approval registers and the internal history to populate the technical datasheet. The photographs provided are processed by an image analysis module that detects relevant structures and checks the plate number, and that information is used to prepare an initial draft of the technical report. The technician reviews and adjusts that draft before submitting it to the Supervisor for validation and signature (Technician → Supervisor → Signed).

The platform runs on a self-hosted architecture with no dependency on external cloud services. It is deployed on virtualised infrastructure using Proxmox VE and ZFS storage, relies on Kubernetes (MicroK8s) for container orchestration, and combines three databases with distinct roles: MongoDB for certification files, MinIO for file storage and Qdrant for semantic searches. The main API is implemented with Node.js/Express and the document processing and drafting support service with Python/FastAPI. The system can connect to document management tools such as SharePoint or OneDrive, so that version history and final document archiving can be integrated with existing corporate tools. A GitLab-based continuous integration and deployment pipeline automates the build and update process and keeps a clear record of the versions installed.

Índice

<i>Resumen</i>	III
<i>Abstract</i>	V
1 Introducción	1
1.1 Objetivos	1
1.2 Estructura del documento	2
2 Revisión de la Literatura y Estado del Arte	3
2.1 El desafío de la catalogación: del papel al ecosistema digital	3
2.1.1 El expediente técnico como eje central del sistema	3
2.2 Marco normativo y administrativo de los vehículos históricos	4
2.2.1 Criterios y requisitos: impacto directo en el expediente	4
2.2.2 Flujo administrativo	4
2.2.3 Calidad, competencia y requisitos de inspección	5
2.3 Proceso de catalogación	5
2.3.1 Actores involucrados en el procedimiento	5
2.3.2 Documentación inicial	6
2.3.3 Problemas recurrentes en la operativa manual	6
2.4 Gestión documental enfocada a expedientes técnicos	6
2.4.1 Sistemas de gestión documental (DMS) y límites del etiquetado manual	6
2.4.2 Histórico de cambios y trazabilidad	7
2.4.3 Integración con aplicativos de terceros	7
2.4.4 OCR (Optical Character Recognition) e IDP (Intelligent Document Processing)	7
2.4.5 Calidad de los datos extraídos y validación	8
2.5 Digitalización de la inspección	8
2.5.1 Estructuración en la toma de datos	8
2.5.2 Reportaje fotográfico con contexto	8
2.5.3 Trabajo en campo: movilidad y conectividad	8
2.6 Automatización de informes y reutilización de conocimiento	8
2.6.1 Plantillas y generación de documentos	9
2.6.2 Conocimiento acumulado basado en casos previos	9
2.7 Herramientas avanzadas de redacción y consulta de expedientes	9
2.7.1 Retrieval-Augmented Generation (RAG)	9
2.7.2 Niveles de madurez de un sistema RAG	9
2.7.3 Gobernanza y revisión humana	10

2.7.4	De RAG a sistemas agénticos	10
2.8	Soluciones habituales en el mercado y limitaciones para este caso	11
2.8.1	Repositorios y suites colaborativas	11
2.8.2	Gestores documentales y ECM	11
2.8.3	Herramientas específicas de inspección y captura en campo	11
2.9	Seguridad, integridad y cumplimiento en expedientes técnicos	11
2.9.1	Control de acceso y mínimos de trazabilidad	12
2.9.2	Integridad de pruebas y cierre de expediente	12
2.9.3	Copias de seguridad y continuidad	12
2.9.4	Protección de datos y buenas prácticas	12
2.10	Arquitecturas de software para plataformas documentales	12
2.10.1	Microservicios y consistencia	12
2.10.2	Organización por capas	13
3	Estudio comparativo de tecnologías actuales	15
3.1	Frontend	16
3.1.1	Framework de estilos	16
3.1.2	Framework de reactividad	18
3.2	Backend	19
3.3	Inteligencia Artificial	20
3.4	Bases de datos	21
3.4.1	Bases de datos orientadas a documentos (NoSQL)	21
	Características de MongoDB	22
3.4.2	Bases de datos relacionales (SQL)	22
3.4.3	Almacenamiento de objetos y documental	22
	Características de MinIO	23
3.4.4	Bases de datos vectoriales	23
	Características de Qdrant	23
3.5	Orquestación y Despliegue	23
3.6	Integración y Entrega Continua (CI/CD)	24
3.7	Ecosistemas de Despliegue e Infraestructura	25
3.7.1	Modelos de Infraestructura	25
3.7.2	Comparativa Técnica de Entornos	26
4	Elección del Stack Tecnológico	27
4.1	Capa 1: Infraestructura y Virtualización	28
4.1.1	Hipervisor: Proxmox Virtual Environment (VE)	28
	Virtualización con KVM y contenedores LXC	28
	Sistema de archivos ZFS	28
4.2	Capa 2: Orquestación de Cargas de Trabajo	29
4.2.1	Orquestador: Kubernetes	29
	Almacenamiento persistente	29
4.3	Capa 3: Persistencia y gestión de datos	29
4.3.1	Base de datos documental: MongoDB	30
	Comparación con SQL	30
4.3.2	Almacenamiento de objetos: MinIO	31
4.3.3	Base de datos vectorial: Qdrant	31
4.4	Capa 4: Backend y lógica de negocio	31

4.4.1	API principal: Node.js + Express	31
4.4.2	Servicios de procesamiento e IA: Python + FastAPI	31
4.5	Capa 5: Frontend y experiencia de usuario	31
4.5.1	Framework: Vue.js 3	32
4.5.2	Gestión de estado: Pinia	32
4.6	Capa 6: Operación, seguridad y DevOps	32
4.6.1	Gestión de credenciales y configuración	32
4.6.2	Registro de imágenes y análisis de vulnerabilidades	32
4.6.3	CI/CD con GitLab	32
4.7	Resumen del Stack Tecnológico Elegido	33
5	Diseño e Implementación de la Solución	35
5.1	Introducción al Despliegue	35
5.2	Aprovisionamiento de la Infraestructura Base	35
5.3	Arquitectura de Almacenamiento y Red SDN	36
5.3.1	Arquitectura de Almacenamiento	36
	Configuración RAID-Z1 y Resiliencia	36
	Optimización de Datasets	37
5.3.2	Infraestructura de Red Definida por Software (SDN)	37
5.4	Virtualización y Estrategia de Plantilla Maestra	37
5.5	Orquestación con Kubernetes	38
5.5.1	Configuración de Alta Disponibilidad (HA)	38
5.5.2	Ingress y Gestión de Tráfico Seguro	38
5.6	Implementación de la Capa de Datos	38
5.6.1	MongoDB: modelo de documento de inspección	38
5.6.2	MinIO: almacenamiento de evidencias y documentos	39
5.6.3	Qdrant: búsqueda semántica y soporte a la asistencia	39
5.7	Desarrollo del Backend y APIs	39
5.7.1	API Gateway (Node.js + Express)	40
	Autenticación y autorización	40
5.7.2	Servicio de Inteligencia Artificial (Python + FastAPI)	40
	Implementación de RAG para consulta y redacción	41
	Asistente conversacional y agentes MCP	41
5.8	Frontend y Experiencia de Usuario (Vue.js)	42
5.8.1	Gestión de estado con Pinia	42
5.8.2	Editor de informes inteligente	43
5.9	Flujo de Trabajo Operativo con Asistencia de IA	43
5.9.1	Gestión de empleados y roles	43
5.9.2	Apertura del expediente: la hoja de trabajo de entrada	44
5.9.3	Enriquecimiento automático de datos	44
5.9.4	Diagnóstico inicial asistido por fotografías	45
5.9.5	Generación del borrador del informe	46
5.9.6	Ciclo de revisión, aprobación y cierre	46
5.10	DevOps y Ciclo de Entrega Continua	47
5.11	Pruebas y Validación	48
5.11.1	Tests unitarios y de integración	48
5.11.2	Pruebas de carga	48

6 Conclusiones	49
Apéndice A Interfaz de usuario	51
A.1 Pantalla de acceso	51
A.2 Panel de resumen general	51
A.3 Listado de procedimientos	53
A.4 Explorador de ficheros	57
A.5 Perfil del usuario	58
A.6 Ajustes generales de la plataforma	59
A.7 Plataforma. Modelos LLM y Vision	60
A.8 Ajustes. Api y Webhooks	61
A.9 Ajustes. Plantillas	61
A.10 Ajustes. Auditoría	62
Apéndice B Fragmentos de código relevantes	63
B.1 Backend: modelo de datos de un procedimiento	63
B.2 Backend: controlador de procedimientos	64
B.3 Backend: sistema de autenticación JWT	65
B.4 Frontend: interceptor Axios y gestión de respuestas	66
B.5 Frontend: store de autenticación con Pinia	68
<i>Índice de Figuras</i>	71
<i>Índice de Tablas</i>	73
<i>Bibliografía</i>	75

1 Introducción

La catalogación y homologación de vehículos históricos es un proceso técnico y administrativo en el que se combinan normativa específica, documentación procedente de distintas fuentes e inspecciones detalladas sobre el propio vehículo. En la práctica, gran parte del trabajo se apoya todavía en la gestión manual de expedientes, el intercambio de archivos por correo, plantillas en distintos formatos y revisiones sucesivas hasta el cierre del informe. Este modo de trabajo funciona, pero puede dar lugar a errores: se multiplican las versiones del mismo documento, la información queda repartida entre carpetas y plataformas diversas (OneDrive, SharePoint, equipos personales de trabajo, etc.) y tareas repetitivas (copiar datos, comprobar requisitos, buscar antecedentes) consumen tiempo que podría dedicarse al análisis técnico.

Este *Trabajo Fin de Grado* se centra en el diseño de una plataforma que ayude a ordenar y automatizar las partes más repetitivas del proceso, sin sustituir el criterio del responsable que realiza la actuación. La propuesta se alinea con el flujo habitual de un expediente de vehículo histórico: recopilación de documentación y evidencias (ya sea a través de un reportaje fotográfico, acta sobre la inspección realizada u otros métodos que tengan el mismo efecto), inspección y toma de datos, elaboración del informe técnico y su revisión, y preparación del expediente para su tramitación ante el organismo competente. La digitalización se plantea, por tanto, como una herramienta para mejorar la trazabilidad del expediente, reducir errores humanos y acortar los tiempos de generación de documentación, manteniendo siempre el control y la validación en manos del técnico.

1.1 Objetivos

El objetivo principal del proyecto es analizar el proceso de catalogación y homologación de vehículos históricos y proponer una solución software que unifique la gestión documental, la elaboración de informes y la trazabilidad del expediente. A partir de ese objetivo general, se definen los siguientes objetivos específicos:

- **Analizar el proceso actual y sus puntos críticos.**

Identificar las fases habituales del expediente (recopilación documental, inspección, elaboración y revisión del informe, cierre y entrega), los actores implicados y los problemas recurrentes (versionado, duplicidades, dispersión de datos necesarios para la emisión de los informes y tiempos de búsqueda elevados).

- **Definir requisitos funcionales y no funcionales realistas.**

Traducir el flujo de trabajo a requisitos de sistema (multiusuario, roles y permisos, control de versiones, auditoría, plantillas de informe, ficheros adjuntos, exportación a formatos

deseados) junto con requisitos de calidad (seguridad, disponibilidad, mantenibilidad y escalabilidad).

- **Revisar tecnologías y proponer un stack coherente.**

Estudiar alternativas para persistencia de datos, almacenamiento de documentos, orquestación y despliegue, y seleccionar un conjunto de tecnologías que permita construir una solución modular, reproducible y sostenible en el tiempo.

- **Diseñar una arquitectura modular orientada a servicios.**

Plantear una arquitectura por capas y microservicios que facilite la integración con herramientas externas (por ejemplo, repositorios documentales) y permita evolucionar el sistema sin afectar al conjunto.

- **Incorporar herramientas de apoyo a la redacción y revisión.**

Integrar un componente de ayuda basado en recuperación de información sobre expedientes y normativa, orientado a acelerar tareas como la localización de referencias, la consulta de casos anteriores o la preparación de borradores iniciales que siempre se revisan antes de su uso.

1.2 Estructura del documento

El documento se organiza en los siguientes epígrafes, con el objetivo de guiar de forma clara el desarrollo del trabajo:

- **Epígrafe 2. Estado del arte:** se revisan los conceptos, la normativa y los enfoques tecnológicos relevantes para la homologación de vehículos históricos y la digitalización del expediente técnico.
- **Epígrafe 3. Estudio comparativo de tecnologías:** se analizan las alternativas tecnológicas disponibles para cada capa de la plataforma (frontend, backend, bases de datos, orquestación e infraestructura), estableciendo la base para la selección del stack.
- **Epígrafe 4. Elección del stack tecnológico y arquitectura:** se justifica la selección de tecnologías y la arquitectura por capas propuesta.
- **Epígrafe 5. Diseño e implementación:** se detallan los componentes principales del sistema, los modelos de datos y la integración entre servicios.
- **Epígrafe 6. Conclusiones y trabajo futuro:** se recogen las conclusiones, las principales limitaciones y posibles líneas de mejora.

2 Revisión de la Literatura y Estado del Arte

2.1 El desafío de la catalogación: del papel al ecosistema digital

La catalogación de vehículos históricos es una disciplina que combina la protección del patrimonio automovilístico con el rigor de la gestión administrativa. No se trata solo de conservar una máquina, sino de documentar su situación de forma que el resultado sea sólido, verificable y coherente con la normativa vigente. En la práctica, el expediente integra fichas técnicas de hace décadas, fotografías digitales y certificados en formatos muy diversos.

En la operativa diaria de entidades de inspección como ETMOV, el flujo tradicional sigue dependiendo en gran medida de tareas manuales. Los correos no vinculados al expediente, las carpetas de red con estructuras distintas y las plantillas ofimáticas hacen difícil identificar en cada momento cuál es la información válida. Cuando aumenta el número de expedientes, estos problemas se hacen más visibles: aparecen duplicidades, cuesta reconstruir quién cambió qué y se pierde demasiado tiempo buscando datos concretos dentro de documentos dispersos.

La transformación digital de servicios técnicos e inspectores ha sido estudiada ampliamente en la literatura (Vial, 2019), que subraya la necesidad de rediseñar procesos y tecnología de forma conjunta para obtener mejoras sostenibles. En este Trabajo Fin de Grado, digitalizar no significa solo sustituir papel por archivos en la nube. La idea es convertir el expediente en un **activo digital estructurado**: un conjunto ordenado de datos, documentos y pruebas de apoyo que están relacionados entre sí. De esta forma, una fotografía, una ficha técnica o una observación de inspección quedan conectadas con la parte del informe donde se utilizan, y es posible seguir el hilo desde la toma de datos hasta la emisión del informe final.¹

2.1.1 El expediente técnico como eje central del sistema

Para que la plataforma sea útil en el trabajo real, el *expediente técnico* debe entenderse como algo más que una carpeta de documentos. Debe ser el punto desde el que se ordenan los datos del vehículo, las pruebas de apoyo, las revisiones y el informe final. Por ello, el expediente debe cubrir cuatro necesidades principales:

- **Identificación inequívoca:** relacionar de forma clara los datos básicos del vehículo, como el VIN, numeraciones y placas identificativas.
- **Justificación normativa:** apoyar la condición de “Vehículo Histórico” en criterios legales y técnicos concretos.

¹ Descripción del servicio y del proceso: <https://etmov.es/actividades/homologacion-vehiculos-historicos/>.

- **Registro de pruebas:** vincular cada observación de la inspección con los elementos que la respaldan (fotografías, documentos o anotaciones).
- **Generación de informes:** producir un documento final defendible ante la administración, donde cada afirmación remita a una prueba clara.

2.2 Marco normativo y administrativo de los vehículos históricos

El catalizador de esta necesidad de cambio es el nuevo **Real Decreto 892/2024** (Gobierno de España, 2024). Esta actualización del *Reglamento de Vehículos Históricos* no solo reduce la burocracia para el ciudadano, sino que eleva la responsabilidad de los Servicios Técnicos.² Al consolidar criterios como la antigüedad mínima (30 años) y la integridad de los sistemas originales, la norma exige una capacidad de documentación ágil y precisa.

2.2.1 Criterios y requisitos: impacto directo en el expediente

La norma se traduce en pruebas concretas. Criterios como la “originalidad” o la “ausencia de modificaciones sustanciales” ya no pueden quedar como simples notas al pie; requieren un soporte documental y fotográfico que sea fácil de auditar. Esto impone tres requisitos al software propuesto:

- **Necesidad de prueba:** cada afirmación relevante del informe debe poder sostenerse en documentación aportada o en observaciones verificables de la inspección.
- **Gestión de excepciones:** existen casos particulares (singularidad, interés patrimonial, modelos de relevancia) donde la justificación es más narrativa y requiere referencias cualitativas o históricas.
- **Trazabilidad de cambios:** durante la elaboración, el informe puede revisarse varias veces; es importante conservar qué se cambió, por qué y con qué prueba se justificó.

2.2.2 Flujo administrativo

Desde un enfoque operativo, el proceso involucra a diferentes entidades. De forma simplificada: el *Servicio Técnico* elabora un informe favorable, la administración autonómica tramita la resolución y, posteriormente, intervienen estaciones de ITV y la DGT para completar el expediente administrativo. Distintas administraciones autonómicas describen flujos equivalentes, que suelen incluir la obtención del informe favorable, la inspección previa en ITV para la tarjeta ITV y el trámite final ante la Jefatura de Tráfico.³ Además, la DGT ha publicado instrucciones para una aplicación homogénea del reglamento.⁴

Para una plataforma digital, este reparto de responsabilidades implica dos necesidades: (i) **uniformidad interna** (misma estructura de expediente y control de calidad) y (ii) **capacidad de exportación** (preparar el expediente para su entrega o tramitación con terceros en formatos aceptados).

² Real Decreto 892/2024, de 10 de septiembre: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2024-18614.

³ Ejemplo de descripción del procedimiento (Junta de Andalucía): <https://www.juntadeandalucia.es/organismos/industriaenergiaminas/areas/industria/vehiculos/paginas/vehiculos-historicos.html>.

⁴ Instrucción DGT (PROT-VEH-2024-13): https://www.dgt.es/export/sites/web-DGT/.galleries/downloads/muevete-con-seguridad/normas-de-trafico/VEH-vehiculos/2024/Instruccion-PROT-VEH-2024-13_Aplicacion-del-Reglamento-de-Vehiculos-Historicos-aprobado-por-Real-Decreto-892_2024-de-10-de-septiembre.pdf.

2.2.3 Calidad, competencia y requisitos de inspección

En servicios técnicos que actúan como entidades de inspección, la gestión documental se relaciona también con requisitos de competencia e imparcialidad. Normas como **ISO/IEC 17020:2012** (ISO/IEC, 2012) definen requisitos generales para organismos de inspección (consistencia, imparcialidad y competencia).⁵ En el caso de ETMOV, se indica acreditación por ENAC bajo UNE-EN ISO/IEC 17020:2012 para la inspección de vehículos históricos.⁶

Esto se traduce en que el sistema debe implementar de forma nativa el **control de versiones**, la **separación de roles** y la **integridad de datos**. Una plataforma o metodología de trabajo que ignore estos puntos obliga al técnico a recurrir a métodos manuales (como nombrar archivos con fechas en el nombre o guardar correos en carpetas personales), lo cual es frágil, ineficiente y compromete la calidad del servicio.

2.3 Proceso de catalogación

Más allá de la norma, el proceso real se apoya en una toma de datos iniciales (reportaje fotográfico, actas, etc.) y en un flujo de trabajo repetible. De forma resumida, las etapas habituales son:

- **Estudio previo:** análisis de la documentación disponible.
- **Inspección:** verificación de la autenticidad de los documentos asociados al vehículo.
- **Informe de catalogación:** elaboración del informe técnico, incluyendo reportaje fotográfico y documentación de apoyo.
- **Resolución:** presentación ante el órgano competente.
- **Trámites legales:** pasos finales ante la ITV y la DGT.⁷



Figura 2.1 Fases del expediente de catalogación de vehículo histórico.

2.3.1 Actores involucrados en el procedimiento

En un expediente de vehículo histórico pueden intervenir perfiles distintos. Para un sistema informático, interesa capturarlos como *roles* y *responsabilidades*:

- **Propietario/representante:** aporta documentación, atiende requerimientos y valida datos administrativos.
- **Técnico del servicio:** analiza documentación, inspecciona el vehículo, redacta el informe y solicita aclaraciones cuando falta información o existe inconsistencia.
- **Revisor interno:** valida la completitud, la coherencia y el cumplimiento de criterios internos antes del cierre.
- **Administración / terceros:** reciben el expediente para su tramitación y pueden requerir subsanaciones o aclaraciones.

⁵ Resumen de ISO/IEC 17020:2012 (IEC): <https://webstore.iec.ch/en/publication/10521>.

⁶ Referencia en la web de ETMOV: <https://etmov.es/actividades/homologacion-vehiculos-historicos/>.

⁷ Proceso descrito por ETMOV: <https://etmov.es/actividades/homologacion-vehiculos-historicos/>.

2.3.2 Documentación inicial

Habitualmente se incluyen, entre otros:

- Documentación administrativa: permisos, bajas, antecedentes y certificados.
- Documentación técnica: fichas, manuales, homologaciones de época, características técnicas y documentación complementaria.
- Pruebas gráficas: reportaje fotográfico general, detalles de identificación, numeraciones, placas, componentes relevantes y estado de conservación.
- Datos de inspección: observaciones, medidas, configuración actual y contraste con lo aportado.

En este contexto se entiende por pruebas cualquier elemento que permita justificar una afirmación del informe: desde una fotografía hasta un documento escaneado o una anotación de inspección. Una plataforma digital debe facilitar que cada prueba quede ligada a la parte exacta del informe donde se utiliza, pasando de un expediente entendido como una simple carpeta con archivos PDF a un conjunto de datos y documentos conectados entre sí.

2.3.3 Problemas recurrentes en la operativa manual

A partir de experiencias habituales en servicios técnicos y en procesos basados en expediente, aparecen patrones de problemas que justifican la digitalización:

- **Fragmentación:** información repartida entre correos, carpetas y herramientas distintas.
- **Versiones divergentes:** múltiples copias del informe o anexos sin control claro de cuál es la válida.
- **Repetición:** tareas de copia de datos y redacción de secciones similares entre expedientes.
- **Búsqueda lenta:** localizar antecedentes o normativa relevante consume tiempo y es propenso a fallos.
- **Subsanaciones:** documentación incompleta o incoherente, que podría detectarse antes si existieran validaciones básicas.

Estos problemas se relacionan directamente con los bloques tecnológicos que se revisan a continuación: gestión documental, captura y estructuración de información, automatización parcial del informe y herramientas de apoyo a la redacción y consulta.

2.4 Gestión documental enfocada a expedientes técnicos

La gestión documental en expedientes técnicos ha evolucionado desde repositorios tradicionales hacia enfoques en los que la captura, la clasificación y la búsqueda de información se automatizan progresivamente. En el dominio de vehículos históricos, esta evolución es especialmente relevante porque la información de partida es mayoritariamente no estructurada o semi-estructurada (PDF, imágenes y anexos), y su tratamiento manual consume tiempo y es propenso a errores.

2.4.1 Sistemas de gestión documental (DMS) y límites del etiquetado manual

Los *Document Management Systems (DMS)* clásicos (por ejemplo, *SharePoint*, *Alfresco* u *OpenKM*) se apoyan en metadatos y en flujos donde los usuarios deben ir aprobando cada una de las etapas que conforman la gestión del documento (Newman, 2021). En general, es una solución que da buenos resultados con volúmenes de documentos moderados, pero cuando crece la carga documental, el

etiquetado manual se convierte en un cuello de botella. Además, en expedientes técnicos es frecuente que el dato relevante no esté en el nombre del fichero, sino dentro del documento (una referencia en una tabla, una numeración en una fotografía o un sello en un anexo), lo que limita la búsqueda basada únicamente en metadatos explícitos.

En consecuencia, las soluciones modernas tienden a complementar el *DMS* con mecanismos de extracción automática y con una estructura del expediente más cercana a un modelo de datos que a una simple carpeta. Esto incluye, por ejemplo:

- **Metadatos derivados:** campos extraídos del contenido (bastidor, fechas, marca/modelo, identificadores).
- **Clasificación por tipos:** agrupar documentos en “pruebas de inspección”, “documentación administrativa”, “documentación técnica” y otros conjuntos con significado operativo.
- **Vinculación a tareas:** un documento no solo se almacena; se usa en una fase, se valida y se relaciona con una conclusión.

2.4.2 Histórico de cambios y trazabilidad

En entornos regulados, el control de versiones y la auditoría no son un complemento, sino un requisito. Por norma general, cualquier plataforma documental introduce prácticas como:

- **Versionado de documentos adjuntos:** si se sustituye una ficha técnica o un certificado, debe conservarse el histórico.
- **Versionado del informe:** borradores, revisiones internas y versión final, manteniendo el rastro de cambios.
- **Auditoría:** registro de acciones (quién subió qué, qué se revisó, qué se aprobó y cuándo).
- **Retención y custodia:** conservación del expediente durante periodos largos, con mecanismos de copia de seguridad y recuperación.

Estas prácticas se apoyan normalmente en una combinación de almacenamiento de objetos (para ficheros), base de datos (para metadatos) y un registro de actividad. En el estado del arte se observa que separar contenido y control facilita la escalabilidad a largo plazo.

2.4.3 Integración con aplicativos de terceros

En muchos entornos, parte de la documentación ya reside en suites ofimáticas y repositorios externos (*Office 365*, *Google Drive*, *Nextcloud*, etc.). La tendencia actual es integrarse con estas fuentes, no sustituirlas por completo. Desde el punto de vista del expediente, esto obliga a decidir si se trabaja con *copias gestionadas* (importando documentos al expediente) o con *referencias* (enlaces a sistemas externos). La primera opción suele simplificar la trazabilidad interna; la segunda reduce duplicidad, pero requiere mecanismos para evitar cambios no controlados. En cualquier caso, la integración con repositorios externos aparece como una necesidad práctica para facilitar la adopción y reducir el esfuerzo inicial de migración.

2.4.4 OCR (Optical Character Recognition) e IDP (Intelligent Document Processing)

El *OCR* permite extraer texto de imágenes o *PDF* escaneados (Mori, Suen y Yamamoto, 1992). Sin embargo, el reto habitual no es solo leer texto, sino comprender estructura y contexto (tablas, sellos, campos repetidos, disposición del documento). Por ello, el estado del arte se desplaza hacia *Intelligent Document Processing (IDP)*, que combina OCR con clasificación, extracción de campos, análisis de la estructura del documento y validación básica de los textos identificados.

En documentos técnicos, los modelos que combinan información visual y textual, como *LayoutLM* (Xu et al., 2020), permiten tratar el documento como un objeto completo y no solo como una secuencia de caracteres. Con ello es posible identificar el tipo de documento, localizar campos clave y extraer información específica. Este planteamiento es especialmente útil cuando el expediente incorpora reportajes fotográficos y documentación muy diversa, como ocurre en la catalogación de vehículos históricos.

2.4.5 Calidad de los datos extraídos y validación

Una lectura automática solo es útil si puede validarse. En el estado del arte, la extracción se acompaña de medidas de confianza y de mecanismos de revisión humana. En la práctica, el sistema sugiere campos (por ejemplo, bastidor, fechas o referencias) y el técnico confirma o corrige. Esta estrategia, conocida como *human-in-the-loop* (Monarch, 2021), mejora la productividad sin asumir riesgos excesivos, especialmente cuando la calidad del documento es variable.

2.5 Digitalización de la inspección

Además del tratamiento de documentación existente, una parte importante del expediente nace durante la inspección. Digitalizar esta fase implica capturar datos de forma estructurada desde el primer momento, evitando que la información se reconstruya después a partir de notas sueltas o fotografías sin contexto.

2.5.1 Estructuración en la toma de datos

En inspecciones técnicas es habitual utilizar listas de comprobación para asegurar que no se omiten revisiones. En plataformas digitales, estas *checklists* suelen integrarse como formularios con estados (pendiente, revisado, no aplica) y con campos obligatorios condicionados por el tipo de vehículo o la casuística. Este enfoque aporta dos ventajas: reduce olvidos y proporciona una base objetiva para detectar expedientes incompletos antes del cierre.

2.5.2 Reportaje fotográfico con contexto

Una dificultad frecuente es que el reportaje fotográfico se gestiona como un conjunto de imágenes sin estructura. El estado del arte tiende a organizar la fotografía por *categorías y objetivos* (vista general, identificación, detalles, estado de conservación), y a vincular cada imagen con metadatos mínimos (fecha, autor, objetivo, relación con un criterio o un dato). Con ello, la imagen deja de ser un simple adjunto y pasa a ser una pieza reutilizable y localizable dentro del expediente.

2.5.3 Trabajo en campo: movilidad y conectividad

En escenarios reales, la inspección puede realizarse en ubicaciones con conectividad limitada. Muchas soluciones adoptan estrategias *offline-first* o de sincronización diferida, donde el técnico captura datos e imágenes y la plataforma consolida el expediente al recuperar conectividad. Aunque este TFG no pretende desarrollar una aplicación móvil completa, sí incorpora la idea de que la arquitectura debe poder absorber capturas de pruebas de forma robusta y ordenada, ya sea desde un navegador en portátil, una tablet o una subida posterior.

2.6 Automatización de informes y reutilización de conocimiento

Una vez organizado el expediente, el siguiente paso natural es acelerar la elaboración del informe. En el estado del arte, la automatización documental se apoya en tres pilares: plantillas, extracción de datos y reutilización de casos previos.

2.6.1 Plantillas y generación de documentos

En entornos técnicos es común trabajar con plantillas que fijan estructura, estilo y secciones obligatorias. La automatización suele consistir en completar estas plantillas con datos del expediente: identificación del vehículo, características, resultados de inspección y anexos (por ejemplo, tablas o fotografías seleccionadas). Este enfoque mejora la uniformidad y reduce errores a la hora de cumplimentar los formularios.

2.6.2 Conocimiento acumulado basado en casos previos

Una fuente importante de información es la que se ha generado a partir de expedientes ya cerrados. El estado del arte propone convertir esos expedientes en una base consultable, donde sea posible encontrar:

- expedientes similares por marca/modelo/época,
- redacciones recurrentes que se adaptan a cada caso,
- criterios aplicados en casuísticas típicas,
- listas de documentación habitual según escenario.

Esta reutilización no busca copiar sin criterio; su objetivo es ofrecer puntos de partida, reducir trabajo repetitivo y favorecer la coherencia interna entre expedientes.

2.7 Herramientas avanzadas de redacción y consulta de expedientes

Una vez que los documentos pueden procesarse e indexarse, el siguiente paso es ayudar al técnico en tareas de consulta y redacción sobre la base de la documentación disponible. Las soluciones actuales se apoyan en modelos de lenguaje capaces de trabajar con texto extenso, pero en un contexto técnico y regulado resulta imprescindible limitar el riesgo de respuestas incorrectas o no justificadas.

2.7.1 Retrieval-Augmented Generation (RAG)

Para escenarios donde importa la precisión, no es suficiente con generar texto: es necesario apoyar la respuesta en fuentes verificables. En este sentido, Retrieval-Augmented Generation (RAG) Lewis et al., 2020 se considera una estrategia de referencia: primero recupera fragmentos relevantes de una base documental y, posteriormente, genera una respuesta utilizando ese contexto. En un servicio técnico, esto se traduce en funciones como localizar requisitos concretos, recuperar antecedentes de expedientes similares o reutilizar redacciones previas, manteniendo siempre el enlace a los documentos o fragmentos que justifican cada respuesta. Además, RAG permite actualizar la base de conocimiento sin reentrenar modelos: basta con actualizar los documentos indexados y sus metadatos.

2.7.2 Niveles de madurez de un sistema RAG

De forma práctica, se distinguen niveles de complejidad creciente Gao et al., 2024:

1. **RAG básico:** recuperación simple y generación directa con el contexto recuperado.
2. **RAG avanzado:** mejoras en segmentación, calidad de recuperación, *reranking* y control de evidencia.
3. **RAG modular:** integración con herramientas externas (validadores, reglas, APIs) y agentes que ejecutan pasos adicionales antes de responder.

Esta progresión encaja con el enfoque del TFG: comenzar con búsqueda y asistencia en redacción, y evolucionar hacia comprobaciones automáticas de coherencia del expediente, siempre bajo supervisión del técnico.

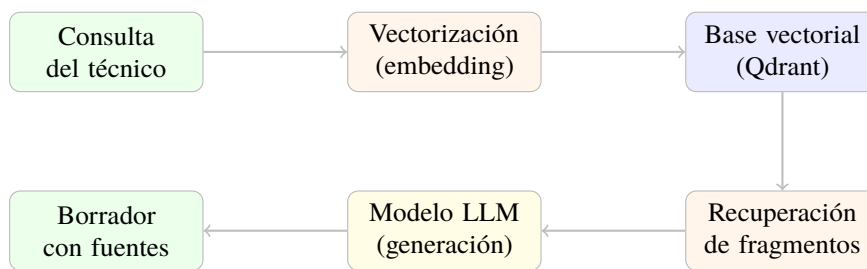


Figura 2.2 Flujo RAG: la consulta se enriquece con fragmentos recuperados antes de generar la respuesta.

2.7.3 Gobernanza y revisión humana

La literatura y las experiencias prácticas insisten en que el uso de modelos de lenguaje debe hacerse con control. Para este dominio, las prácticas más relevantes son: (i) respuestas apoyadas en fuentes identificables, (ii) revisión humana de borradores, (iii) registro de qué documentos se usaron como base y (iv) límites claros sobre qué partes del informe pueden automatizarse y cuáles requieren confirmación explícita.

2.7.4 De RAG a sistemas agénticos

El RAG resuelve bien las consultas directas: buscar un criterio dado, recuperar una sección de un expediente anterior o proponer una redacción a partir de datos concretos. Las limitaciones aparecen cuando la respuesta a una pregunta requiere combinar varias fuentes o realizar varios pasos antes de poder contestar. Ahí es donde entran los sistemas agénticos.

En un flujo RAG clásico la secuencia es fija (buscar → recuperar → generar). En un sistema agéntico, en cambio, el componente de razonamiento puede decidir qué pasos dar en función de lo que necesita. Si el técnico pregunta “¿está este expediente listo para cerrarse?”, el sistema no sabe de antemano qué tiene que revisar. Primero consulta qué campos están completados en MongoDB, después verifica si los archivos obligatorios están subidos en MinIO y, por último, contrasta el resultado con los criterios de cierre del reglamento. Todo eso ocurre dentro de la misma petición, sin que el técnico tenga que lanzar cada consulta por separado.

Este comportamiento se puede articular sobre el estándar *MCP (Model Context Protocol)* (Anthropic, 2024), que define cómo un modelo de lenguaje se comunica con herramientas externas. Cada fuente de datos de la plataforma (MongoDB, MinIO, Qdrant y la base de conocimiento normativo) se expone como un servidor MCP. El modelo decide qué servidores consultar y en qué orden según el contexto de la petición, y siempre adjunta referencias a las fuentes que ha utilizado. Si la información disponible no es suficiente para responder con seguridad, el sistema lo indica en lugar de inventar conclusiones.

Algunos ejemplos de peticiones que un asistente de este tipo puede resolver de forma directa son:

- “¿Qué documentos le faltan a este expediente para poder cerrarlo?”
- “Prepara un borrador del apartado de reformas usando las pruebas que ya están cargadas.”
- “Busca expedientes anteriores con reformas similares a la 8.52 para ver cómo se justificaron.”

El modelo se ejecuta localmente mediante *Ollama* (Ollama Team, 2024), de manera que los datos del expediente no salen de la infraestructura propia. Esta decisión es importante en un entorno

con documentación técnica y datos de titularidad: el sistema trabaja sobre los datos reales de la plataforma sin enviarlos a servicios externos.



Figura 2.3 Ciclo de razonamiento: de la petición a la respuesta fundamentada en pruebas.

2.8 Soluciones habituales en el mercado y limitaciones para este caso

Cuando una organización quiere digitalizar expedientes, lo más frecuente es apoyarse en herramientas ya disponibles (suite ofimática, almacenamiento en nube, correo y hojas de cálculo) y, a medida que el volumen crece, incorporar soluciones específicas. En el mercado existen varias familias de herramientas que resuelven partes del problema, pero no siempre encajan bien con un servicio técnico centrado en vehículos históricos.

2.8.1 Repositorios y suites colaborativas

Soluciones como *SharePoint/OneDrive*, *Google Drive* o *Nextcloud* resuelven bien la colaboración básica: compartir ficheros, controlar permisos y mantener un histórico de versiones en documentos editables. Sin embargo, el seguimiento del proceso (qué falta, en qué estado está, quién lo revisa) suele quedar fuera del repositorio, típicamente en correos, notas o una hoja de cálculo. El resultado es una gestión en paralelo que funciona al principio, pero se vuelve frágil cuando hay varios expedientes activos, revisiones cruzadas o cambios de última hora.

2.8.2 Gestores documentales y ECM

Los *DMS/ECM* aportan un escalón adicional: metadatos, flujos de aprobación y, en algunos casos, plantillas y auditoría. Aun así, en expedientes técnicos como los de vehículos históricos aparecen limitaciones. La primera es el coste de modelado y mantenimiento: para que el sistema sea útil hay que definir modelos de datos, metadatos y flujos, y adaptarlos cuando cambian los requisitos. La segunda está en vincular pruebas (fotografías y observaciones de inspección) con secciones del informe y con decisiones concretas, algo que en herramientas genéricas suele resolverse con trabajo manual.

2.8.3 Herramientas específicas de inspección y captura en campo

En sectores como mantenimiento industrial o inspección de activos existen herramientas para capturar fotos, listas de comprobación y datos in situ desde móvil o tablet. Suelen ser muy útiles para estandarizar pruebas y reducir errores de transcripción. El inconveniente es que rara vez están pensadas para generar un informe técnico para la administración con anexos, redacción y trazabilidad documental completa. En consecuencia, muchas organizaciones acaban exportando datos y volviendo al documento manual.

2.9 Seguridad, integridad y cumplimiento en expedientes técnicos

En un expediente con implicaciones administrativas, la seguridad no debe entenderse únicamente como poner contraseñas. El objetivo es garantizar que la información está disponible cuando se necesita, que no se modifica sin control y que solo acceden quienes deben acceder. Además, el

expediente suele incluir datos personales (titularidad, identificadores, documentación), por lo que es necesario contemplar principios básicos de protección de datos.

2.9.1 Control de acceso y mínimos de trazabilidad

Incluso en sistemas pequeños, es recomendable separar perfiles (por ejemplo, administración, técnico, consulta) y registrar acciones relevantes (subida de documentos, cambios de estado, validación, cierre). Este registro es útil tanto para auditorías internas como para resolver problemas prácticos (qué versión se usó, quién solicitó una aclaración, cuándo se incorporó una prueba).

2.9.2 Integridad de pruebas y cierre de expediente

Una práctica frecuente en expedientes técnicos es distinguir entre contenido editable y versión congelada. Esto se puede implementar con mecanismos sencillos: elaborar versiones finales, generar un paquete de ficheros y fotografías asociadas al informe y mantener el histórico de modificaciones. El objetivo no es complicar el flujo, sino evitar que el expediente cambie sin dejar rastro.

2.9.3 Copias de seguridad y continuidad

El estado del arte en operación de plataformas documentales insiste en que la disponibilidad y la recuperación son parte de la calidad del servicio. Perder fotografías o documentación de un expediente activo puede implicar rehacer inspecciones o solicitar de nuevo documentación al cliente. Por ello, las copias de seguridad, la redundancia de almacenamiento y los mecanismos de recuperación ante fallos deben considerarse requisitos básicos de diseño.

2.9.4 Protección de datos y buenas prácticas

Sin entrar en un análisis jurídico detallado, sí es importante asumir buenas prácticas: principio de minimización (guardar lo necesario), control de acceso, registro de actividad y políticas de retención razonables. En el marco europeo, el RGPD (Parlamento Europeo y Consejo de la Unión Europea, 2016) marca la referencia general en materia de datos personales.⁸ En entornos con relación con la administración pública, también es habitual considerar marcos como el Esquema Nacional de Seguridad (ENS) (Gobierno de España, 2022) como guía de medidas organizativas y técnicas.⁹

2.10 Arquitecturas de software para plataformas documentales

Integrar gestión documental, almacenamiento de pruebas, control de usuarios y herramientas de apoyo al trabajo del técnico requiere una arquitectura que sea mantenible a lo largo del tiempo. Una estrategia habitual es separar responsabilidades en servicios independientes, de forma que cada módulo pueda evolucionar sin bloquear al resto.

2.10.1 Microservicios y consistencia

La arquitectura de microservicios facilita desacoplar componentes con necesidades distintas (por ejemplo, un servicio web de expedientes y un servicio de ingesta y procesamiento documental), permitiendo despliegues independientes y escalado específico (Newman, 2021; Richardson, 2018). Además, el enfoque por servicios encaja con despliegues basados en contenedores y orquestación, donde cada componente se publica, monitoriza y actualiza como una unidad separada.

⁸ Reglamento (UE) 2016/679 (RGPD): <https://eur-lex.europa.eu/eli/reg/2016/679/oj>.

⁹ Esquema Nacional de Seguridad (ENS): https://administracionelectronica.gob.es/pae/Home/pae_Estrategias/pae_Seguridad/ens.html.

2.10.2 Organización por capas

Una forma habitual de hacer la arquitectura software más legible es describirla por capas: infraestructura, orquestación, datos, servicios, interfaz y operación/seguridad. Esta visión no pretende imponer una estructura rígida, sino facilitar que cada decisión tecnológica se entienda en relación con un problema concreto: cómo se despliega, cómo se almacenan los datos, cómo se expone una API o cómo se monitoriza el sistema.

3 Estudio comparativo de tecnologías actuales

El análisis se organiza por capas. Primero se revisan las tecnologías de interfaz (frontend), donde importan la reactividad y la facilidad de uso para los técnicos que trabajan con formularios, documentos y fotografías. Después se estudia la capa de servidor (backend) prestando atención a las opciones que facilitan integrar procesamiento de documentos e IA. A continuación se comparan las alternativas de almacenamiento, diferenciando entre expedientes estructurados, ficheros binarios y vectores para búsqueda semántica. Por último, se analizan los enfoques de orquestación e infraestructura, que determinan cómo se despliega y mantiene la plataforma desarrollada.

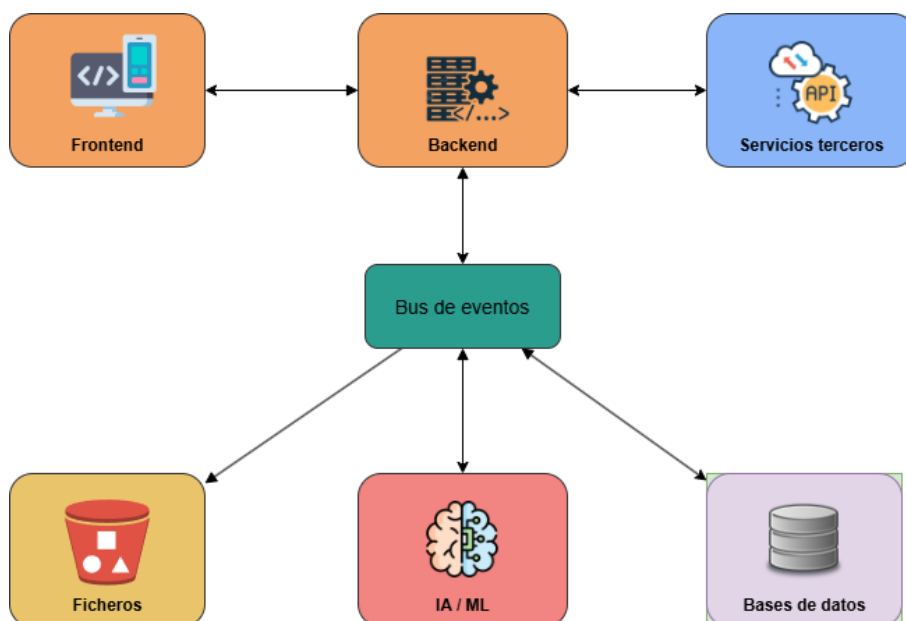


Figura 3.1 Bloques funcionales que conforman la plataforma a diseñar .

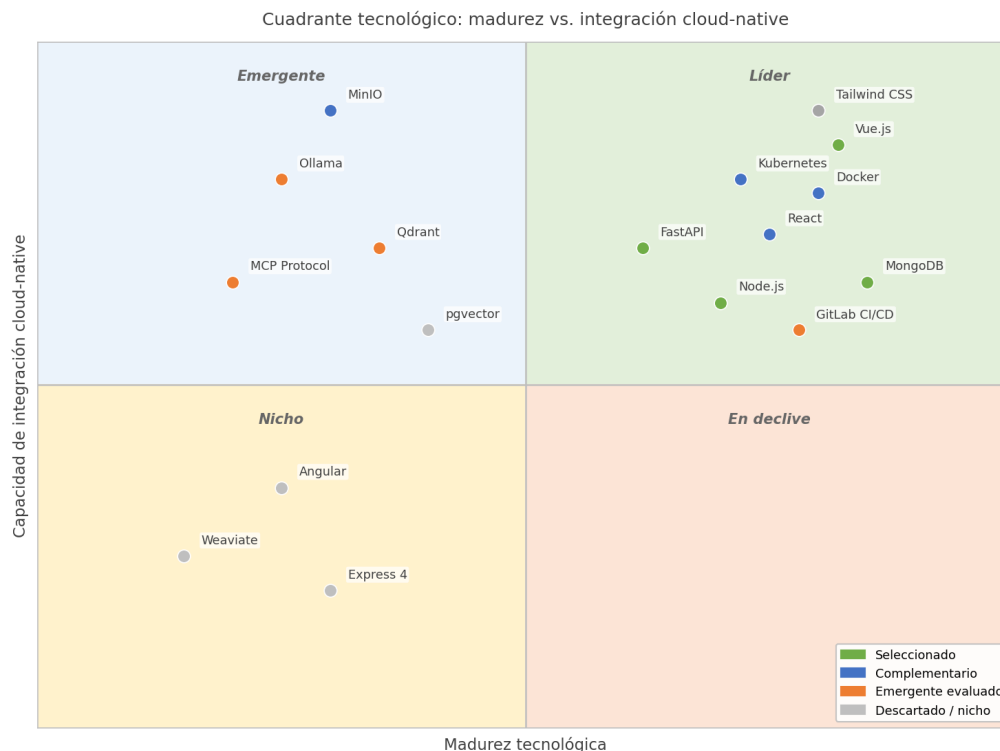


Figura 3.2 Mapa de tecnologías evaluadas en función de su madurez y capacidad de integración en arquitecturas tipo Cloud.

3.1 Frontend

Parte visible de una aplicación o sitio web con la que interactúan directamente los usuarios, diseñada para ejecutarse en el navegador. Incluye la interfaz gráfica, la estructura de las páginas, los estilos visuales y la lógica necesaria para responder a acciones del usuario, como hacer clic, completar formularios o navegar entre secciones. En la actualidad, una gran amplitud de frameworks incluyen nativamente el soporte reactivo (reactive) y adaptable (responsive), lo que permite construir interfaces dinámicas y fluidas que se adaptan automáticamente al contenido, al contexto y al dispositivo.

3.1.1 Framework de estilos

Un framework con capacidad de adaptación (responsive) es capaz de modificar su presentación y estructura según el tamaño de pantalla o dispositivo desde el cual se accede, ya sea un ordenador de escritorio, una tablet o un teléfono móvil. Este tipo de diseño es esencial para garantizar la accesibilidad y la usabilidad en una amplia gama de contextos de uso.

- **Material UI:** Conjunto de componentes visuales diseñados bajo las guías de diseño de Google Material Design. Permite crear interfaces modernas y totalmente adaptables a distintos dispositivos, especialmente en proyectos con React.
- **Tailwind CSS** Wathan et al., 2024: Framework de utilidades CSS altamente personalizable. Su enfoque "utility-first" permite desarrollar diseños responsive directamente en las clases de los componentes HTML, ofreciendo gran flexibilidad y control visual.

- **Bootstrap:** Uno de los frameworks CSS más conocidos y utilizados para diseño responsive. Proporciona una amplia variedad de componentes y una rejilla flexible que facilita la adaptación automática a distintos tamaños de pantalla.

Entre las opciones evaluadas, **Tailwind CSS** destaca por su enfoque radicalmente distinto al de frameworks como Bootstrap: en lugar de ofrecer componentes visuales predefinidos con estilos difíciles de sobrescribir, adopta el modelo *utility-first*, donde cientos de utilidades atómicas (`flex`, `px-4`, `rounded-xl`) se combinan directamente en el marcado HTML para construir cualquier diseño sin salir del componente. Este enfoque ofrece varias ventajas diferenciales:

- **Integración natural con frameworks de componentes:** Al aplicar los estilos sobre las clases del propio componente, no es necesario gestionar ficheros CSS separados ni preocuparse por colisiones de clases globales, lo que encaja bien con arquitecturas basadas en componentes como Vue o React.
- **Diseño responsive integrado:** Tailwind incorpora un sistema de breakpoints (`sm:`, `md:`, `lg:`) que permite definir estilos adaptativos directamente sobre los elementos, sin necesidad de escribir media queries explícitas en CSS.
- **Soporte nativo para modo oscuro:** La variante `dark:` permite definir estilos alternativos para el modo oscuro en el mismo lugar donde se define el estilo normal, eliminando la necesidad de un sistema de temas adicional.
- **Eliminación de CSS no utilizado (tree-shaking):** En la compilación de producción, el motor de Tailwind analiza todos los ficheros del proyecto y elimina automáticamente cualquier clase que no esté en uso, generando una hoja de estilos final de tamaño mínimo.
- **Consistencia visual sin sistema de diseño previo:** La escala de espaciado, tamaños tipográficos y paleta de colores están predefinidos y son coherentes entre sí, lo que facilita mantener una interfaz visualmente consistente incluso sin un equipo de diseño dedicado.

La versión 4 del framework introduce además un nuevo motor de compilación basado en *Lightning CSS* que reduce sensiblemente los tiempos de construcción respecto a versiones anteriores, y se integra directamente en pipelines de *PostCSS* sin configuración adicional.

Como referencia ilustrativa, el siguiente fragmento muestra cómo se construye un botón de acción primaria sin una sola línea de CSS externo:

```
<button class="inline-flex items-center gap-2 rounded-md bg-indigo-600
    px-4 py-2 text-sm font-semibold text-white shadow-sm
    hover:bg-indigo-500 transition-colors">
  Nuevo expediente
</button>
```

La legibilidad inicial puede resultar menor que la de un framework basado en componentes, pero aporta una ventaja significativa en mantenibilidad: el comportamiento visual de cada elemento queda completamente autocontenido en su propia definición, sin dependencias ocultas en ficheros de estilos externos.

Tabla 3.1 Comparativa de frameworks CSS/UI de estilos.

Framework	Enfoque	Modo oscuro	Tree-shaking	Personalización	Integración con Vue
Tailwind CSS	Utility-first	Nativo (<code>dark:</code>)	Sí (v3/v4)	Muy alta	Muy buena
Bootstrap	Componentes predefinidos	Parcial (v5.3+)	Parcial	Media	Buena (via wrapper)
Material UI	Componentes (Material Design)	Sí	Sí	Media/Alta	Limitada (orientado a React)

3.1.2 Framework de reactividad

Un framework reactivo es aquel que permite que la interfaz de usuario reaccione de forma automática a los cambios en los datos subyacentes, sin necesidad de recargar la página o manipular manualmente el DOM (Document Object Model). Esto se traduce en una experiencia de usuario más rápida, fluida e interactiva, ya que los componentes se actualizan en tiempo real cuando cambian los estados de la aplicación. Entre los frameworks más utilizados podemos encontrar:

- **Vue.js** You y V. Contributors, 2024: Destacado por su simplicidad, ligereza y curva de aprendizaje baja. Permite crear interfaces reactivas de forma intuitiva, ideal para proyectos que buscan rapidez en desarrollo y mantenimiento.
- **React**: Biblioteca ampliamente adoptada, mantenida por Meta, con una gran comunidad y rendimiento optimizado. Favorece el uso de componentes funcionales y una gestión avanzada del estado.
- **Angular (2+)**: Framework completo y ampliamente adoptado en entornos empresariales. Su arquitectura todo-en-uno y su mayor complejidad frente a alternativas más ligeras elevan la curva de aprendizaje, aunque sigue siendo una referencia en proyectos de gran escala.

Por otra parte, disponemos de frameworks sin capacidad "reactiva", pero que aún son muy utilizados por su sencillez, madurez y facilidad de integración en soluciones web. Estos frameworks suelen basarse en el caso clásico de generación de páginas en el servidor, donde cada interacción con el usuario implica una petición *HTTP* completa que devuelve una nueva vista *HTML*. Aunque no permiten actualizar la interfaz en tiempo real sin recargar la página, ofrecen ventajas como un menor consumo de recursos en el cliente, mayor compatibilidad y una curva de aprendizaje más asequible para desarrolladores con experiencia en entornos web tradicionales.

- **Django**: Framework de desarrollo web en *Python* que utiliza un sistema de plantillas para generar vistas *HTML* desde el servidor. Aunque no es reactivo por defecto, es muy utilizado por su rapidez de desarrollo, seguridad y estructura robusta para aplicaciones web completas.
- **Laravel**: Framework *PHP* moderno que permite construir aplicaciones web con arquitectura *MVC (Modelo-Vista-Controlador)*. Su motor de plantillas *Blade* permite generar interfaces desde el backend, aunque no dispone de capacidades reactivas nativas. Es conocido por su sintaxis elegante, facilidad de uso y gran comunidad.
- **Ruby on Rails**: Genera vistas del lado del servidor sin interactividad reactiva nativa, pero destaca por su productividad, simplicidad y ecosistema maduro en entornos empresariales.

Tabla 3.2 Comparativa de frameworks frontend en diseño reactivo y responsive.

Framework	Soporte Reactivo	Soporte Responsive	Facilidad de uso	Popularidad
Vue.js	Sí	Sí	Alta	Alta
React	Sí	Sí	Media	Muy alta
Angular (2+)	Sí	Sí	Baja	Alta
Django (Templates)	No	Parcial*	Alta	Alta
Laravel (Blade)	No	Parcial*	Alta	Alta
Ruby on Rails	No	Parcial*	Media	Media

* El soporte responsive depende del uso explícito de frameworks CSS como Bootstrap o Tailwind

3.2 Backend

El backend es el componente que gestiona la lógica principal: controlar accesos, manejar expedientes, integrar los almacenes de datos y coordinar el procesamiento de documentos. En una plataforma con cargas de trabajo heterogéneas —peticiones web, tareas de OCR y modelos de IA— la elección del framework condiciona tanto el rendimiento como la facilidad de mantenimiento a largo plazo.

Una arquitectura frecuente en plataformas con cargas heterogéneas consiste en separar dos tipos de backend: uno para las operaciones habituales de la aplicación (API de negocio) y otro dedicado al procesamiento intensivo (extracción, OCR e IA). Cada parte puede beneficiarse de herramientas distintas adaptadas a su naturaleza. Los principales frameworks evaluados son:

- **Express.js (Node.js)** Cantelon et al., 2014; Express.js Contributors, 2024: Framework ligero basado en el modelo *event-driven* de Node.js. Su entrada/salida no bloqueante encaja bien con APIs que atienden muchas peticiones concurrentes y delegan trabajo en bases de datos o servicios externos.
- **FastAPI (Python)** Ramírez, 2024: Framework asíncrono sobre *ASGI* (*Asynchronous Server Gateway Interface*). Gracias a *Pydantic*, facilita validar datos de entrada y salida, y resulta especialmente cómodo para exponer servicios de inteligencia artificial y *Machine Learning*.
- **Spring Boot (Java)**: Solución muy extendida en entornos empresariales sobre la *JVM*. Ofrece madurez, buenas herramientas y soporte para transacciones, aunque suele requerir más memoria y una estructura de proyecto más pesada que alternativas ligeras.

También existen lenguajes compilados orientados a servicios de alto rendimiento. Son opciones muy interesantes cuando el objetivo principal es exprimir recursos o construir componentes de infraestructura:

- **GoLang**: Diseñado por *Google* para crear servicios ligeros y concurrentes mediante *goroutines*. Se utiliza con frecuencia en proxies, herramientas cloud-native y componentes del ecosistema Kubernetes.
- **Rust**: Destaca por su seguridad de memoria sin recolector de basura, gracias al *borrow checker*. Es una opción potente para componentes de bajo nivel, procesamiento intensivo o servicios donde el rendimiento y la fiabilidad sean críticos.
- **ASP.NET Core (C#)**: Framework moderno de Microsoft con muy buen rendimiento y herramientas maduras. Encaja especialmente bien en organizaciones que ya trabajan con el ecosistema .NET o Azure.

Tabla 3.3 Comparativa de frameworks backend.

Framework / Lenguaje	Rendimiento	Escalabilidad	Facilidad de uso	Popularidad
Express (Node.js)	Media	Alta	Alta	Muy alta
FastAPI (Python)	Alta	Media/Alta	Alta	Alta
Spring Boot (Java)	Alta	Muy alta	Media/Baja	Muy alta
GoLang	Muy alta	Muy alta	Media	Alta
Rust	Muy alta	Alta	Baja	Media
ASP.NET Core (C#)	Alta	Muy alta	Media	Alta
Laravel (PHP)	Media	Media	Alta	Alta
Django (Python)	Media	Media	Alta	Muy alta
Ruby on Rails	Media	Media	Alta	Media

3.3 Inteligencia Artificial

La IA tiene un papel concreto en esta plataforma: extraer información de documentos (OCR e IDP), facilitar búsquedas por significado y asistir al técnico en la consulta de normativa y antecedentes (RAG). El volumen de documentación de un expediente de vehículo histórico justifica herramientas que reduzcan el trabajo manual de búsqueda y clasificación, pero la integración debe ser selectiva: se incorporan los componentes que resuelven problemas reales, no los que añaden complejidad sin beneficio claro. Para que estas funciones sean posibles se combinan:

- **Extracción de texto e información (IDP):** procesamiento de PDFs e imágenes para identificar datos clave del vehículo, fechas, referencias y firmas.
- **Búsqueda semántica:** representar documentos como vectores numéricos (embeddings) para encontrar contenido relacionado por significado, no solo por palabras exactas.
- **Asistencia conversacional (RAG + agentes):** responder preguntas del técnico apoyándose en documentos reales del sistema, con trazabilidad de fuentes.

Los **embeddings** son la pieza técnica que hace posible la búsqueda semántica. Un modelo de lenguaje convierte cada fragmento de texto en un vector numérico, de forma que textos con significado parecido quedan cerca en ese espacio matemático. Cuando el técnico busca “reformas en suspensión”, el sistema localiza fragmentos de expedientes anteriores relacionados aunque no usen exactamente esas palabras. Estos vectores se almacenan en una base de datos vectorial y se consultan mediante similitud de coseno.

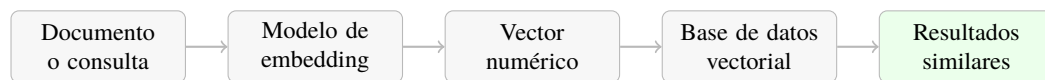


Figura 3.3 Cómo funciona la búsqueda semántica: el texto se convierte en un vector y se compara con los demás por similitud.

- **Modelos de embedding comunes** Reimers y Gurevych, 2019: SBERT, E5, GTE, BGE, QWEN-embedding.
- **Ventajas:** permiten búsquedas por similitud, resumen automático, clasificación temática, y soporte a arquitecturas RAG.
- **Almacenamiento:** los vectores generados pueden ser almacenados en motores vectoriales como Qdrant Qdrant Team, 2023 para consultas rápidas y escalables.

En cuanto al contenido visual, se requieren modelos de **visión por computador**, entrenados para asociar imágenes a representaciones textuales. Esto permite, por ejemplo, que una imagen de una placa de homologación o una etiqueta pueda ser buscada y comparada con imágenes similares en informes anteriores. Algunos modelos permiten incluso generar descripciones automáticas de imágenes o detectar anomalías.

- **Modelos de visión relevantes** Radford et al., 2021: CLIP, BLIP, GIT, Florence, QWEN-vision.
- **Aplicaciones:** búsqueda visual inversa, extracción de información, análisis de etiquetas técnicas o planos escaneados.

Para desplegar estos modelos se pueden emplear diferentes enfoques, que van desde herramientas locales hasta servidores de inferencia optimizados:

- **Ollama** Ollama Team, 2024; Vaswani et al., 2017: ejecuta modelos *LLM* (*Large Language Model*) localmente, útil para entornos sin conexión o sin dependencia externa.
- **vLLM**: ofrece inferencia rápida en *GPU* (*Graphics Processing Unit*), ideal para uso concurrente y *APIs OpenAI-compatible*.

Finalmente, para ampliar la funcionalidad sin necesidad de reentrenar modelos, se introducen los *MCPs* (*Model Context Protocols*), que permiten extender un modelo base mediante herramientas externas. Cada herramienta puede ser una consulta a base de datos, una búsqueda documental, una llamada a una API externa o cualquier función que aporte contexto real al modelo. Con ello se evita el costoso proceso de *fine-tuning* y el modelo puede operar directamente sobre los datos vivos del sistema.

Combinados con un motor de inferencia local como *Ollama*, los MCPs son la pieza que convierte un LLM en un **agente**: un sistema que recibe una tarea en lenguaje natural, decide qué herramientas usar, las invoca en el orden necesario y entrega una respuesta construida sobre datos reales. En lugar de estar atado a un flujo de pantallas, el técnico puede hacer preguntas directas —“¿cuántos expedientes están pendientes de revisión esta semana?” o “busca reformas similares a la que acabo de registrar”— y obtener respuestas basadas en el estado actual de la plataforma, no en suposiciones del modelo.

3.4 Bases de datos

En cualquier sistema de información, elegir bien cómo se almacenan los datos condiciona el rendimiento, la escalabilidad y la facilidad para evolucionar el proyecto. Las bases de datos relacionales (*RDBMS*, *Relational Database Management System*) han sido durante años la opción por defecto por su consistencia y su estructura clara. Sin embargo, muchas aplicaciones actuales necesitan modelos más flexibles, capaces de cambiar con el tiempo y de trabajar con información distribuida. En ese contexto aparecen las tecnologías *NoSQL* (*Not Only SQL*).

En sistemas de gestión documental como el que se analiza, suelen coexistir tres tipos de información muy diferentes: datos estructurados o semiestructurados del expediente, archivos grandes como imágenes y documentos PDF, y vectores numéricos para búsqueda semántica mediante *embeddings*. En lugar de forzar una única base de datos para todo, es más eficiente estudiar qué herramienta encaja mejor con cada tipo de dato.

3.4.1 Bases de datos orientadas a documentos (NoSQL)

Las bases de datos orientadas a documentos son un tipo de almacén *NoSQL* diseñado para gestionar datos semiestructurados. A diferencia de las tablas relacionales, donde cada fila debe cumplir estrictamente con las columnas definidas, estas bases de datos almacenan la información en documentos autocontenidos, típicamente en formatos como JSON (*JavaScript Object Notation*) o BSON (*Binary JSON*).

Esta flexibilidad permite que la estructura de los datos evolucione junto con la aplicación sin obligar a rediseñar todo el esquema cada vez que aparece un nuevo campo. Además, muchas de estas bases de datos pueden escalar horizontalmente mediante *sharding* (fragmentación), repartiendo los datos entre varios servidores.

Dentro de la variedad de herramientas disponibles (MongoDB, DynamoDB, Cassandra, entre otras), **MongoDB** Chodorow, 2013; Stonebraker, 2010; Abramova, Bernardino y Furtado, 2014 destaca como la opción más madura y ampliamente adoptada del ecosistema documental. Es una solución *Open-Source* con amplia documentación y un ecosistema muy activo, lo que la convierte en una candidata natural para proyectos que priorizan la flexibilidad de esquema y la integración con entornos JavaScript o Python.

Características de MongoDB Base de datos orientada a documentos más popular y madura del mercado. Almacena la información en documentos *BSON*, lo que permite tipos de datos ricos y una integración natural con lenguajes de programación modernos, especialmente *JavaScript/Node.js* y *Python*.

- **Flexibilidad de esquema:** Los informes de inspección técnica pueden variar considerablemente en sus campos dependiendo del tipo de inspección o normativa aplicable. Permite almacenar estos registros heterogéneos en una misma colección sin imponer restricciones.
- **Escalabilidad y Alta Disponibilidad:** Mediante el uso de *Replica Sets*, se garantiza redundancia automática y *failover*. Si el nodo primario falla, uno secundario toma el control automáticamente.
- **Indexación potente:** Soporta índices secundarios, índices geoespaciales y búsqueda de texto, lo cual es vital para localizar informes rápidamente.
- **Comunidad y Ecosistema:** Al ser un estándar de la industria, cuenta con drivers oficiales para todos los lenguajes mayores y herramientas de administración robustas como *MongoDB Compass*.

3.4.2 Bases de datos relacionales (SQL)

Las bases de datos relacionales (*SQL*) organizan la información en tablas con filas y columnas predefinidas, relacionándolas mediante claves foráneas. Se basan en el modelo relacional y utilizan *SQL (Structured Query Language)* para la manipulación de datos.

Su principal fortaleza reside en las propiedades *ACID (Atomicity, Consistency, Isolation, Durability)*, que garantizan que las transacciones sean procesadas de forma fiable. Son la opción indiscutible para sistemas donde la integridad de los datos es prioritaria sobre la escalabilidad masiva, como sistemas bancarios, inventarios estrictos o gestión de usuarios con permisos complejos.

Entre los exponentes más destacados encontramos:

- **PostgreSQL:** Considerada la base de datos relacional de código abierto más avanzada. Soporta tipos de datos complejos (incluso JSON), extensiones potentes (como PostGIS) y garantiza una integridad de datos excepcional.
- **MySQL / MariaDB:** Líderes en aplicaciones web, conocidos por su velocidad de lectura y facilidad de uso.

A pesar de su robustez, el modelo relacional presenta ciertas limitaciones cuando los datos son jerárquicos y variables, como ocurre con los expedientes de inspección técnica. Una normalización estricta requeriría múltiples tablas unidas por *JOINS* para reconstruir un único informe, lo que puede penalizar el rendimiento de lectura y aumentar la complejidad del modelo de datos. En estos casos, el modelo documental ofrece una alternativa más natural.

3.4.3 Almacenamiento de objetos y documental

Una limitación común en el diseño de bases de datos es el manejo de archivos binarios grandes (*BLOBs*), como imágenes de alta resolución, vídeos o documentos *PDF* escaneados. Almacenar estos archivos directamente en la base de datos (sea *SQL* o *Mongo*) suele degradar el rendimiento, aumentar el tamaño de los backups y complicar la replicación.

Una solución ampliamente aceptada es utilizar un **Object Storage** (Almacenamiento de Objetos). Este tipo de sistema gestiona los datos como objetos, cada uno con un identificador único y metadatos asociados, accesibles a través de *APIs HTTP* (típicamente compatibles con el protocolo *S3* de Amazon). Entre las alternativas disponibles, **MinIO** MinIO Inc., 2023 destaca por ser *Open-Source*, con compatibilidad total con la API *S3* y un modelo de despliegue ligero, lo que lo convierte en una opción muy valorada tanto en entornos locales como en infraestructuras containerizadas.

Características de MinIO Servidor de almacenamiento de objetos de alto rendimiento, compatible con la API de Amazon S3. Está diseñado para trabajar en sistemas *cloud* y se despliega fácilmente en contenedores Docker Merkel, 2014 o Kubernetes Burns et al., 2022.

- **Rendimiento:** Capaz de servir la información usando todo el ancho de banda de red disponible, ideal para servir imágenes y *PDFs* a la interfaz de usuario con mínima latencia.
- **Simplicidad:** Separa el contenido "pesado" de la lógica de negocio. La base de datos principal *MongoDB* solo guarda la URL o el path del objeto almacenado en *MinIO*.
- **Inmutabilidad y Versionado:** Permite proteger los documentos originales contra modificaciones accidentales, un requisito clave a la hora de proporcionar una trazabilidad mediante un histórico de cambios.

Otras alternativas emergentes en este espacio incluyen sistemas como *RustFS* o soluciones distribuidas como *Ceph*. Sin embargo, *MinIO* destaca por su facilidad de operación y compatibilidad estándar con el ecosistema de desarrollo actual.

3.4.4 Bases de datos vectoriales

Con la integración de la Inteligencia Artificial Generativa y los Grandes Modelos de Lenguaje (LLMs) en la plataforma, surge la necesidad de un nuevo tipo de persistencia: la base de datos vectorial.

Los modelos de IA transforman el texto y las imágenes en "embeddings", que son representaciones numéricas (vectores) en un espacio multidimensional. Para buscar documentos similares semánticamente (por ejemplo, buscar "daños en estructura" y encontrar fotos de grietas, aunque no tengan esa etiqueta explícita), se necesita calcular la distancia matemática entre estos vectores (similitud del coseno, distancia euclídea). Las bases de datos tradicionales no son eficientes para estas operaciones matriciales a gran escala. Dentro del espectro nos encontramos con *Pinecone*, *Chroma*, *Milvus* y **Qdrant** Qdrant Team, 2023, siendo este último especialmente valorado por su bajo consumo de recursos, su facilidad de despliegue en contenedores y su API REST completa.

Características de Qdrant Motor de búsqueda vectorial y base de datos escrita en Rust, diseñada para ofrecer un rendimiento extremo y bajo consumo de recursos.

- **Filtrado Híbrido:** Permite combinar la búsqueda vectorial (semántica) con filtros tradicionales (metadatos). Por ejemplo: "Buscar informes similares a este (vector), pero solo del año 2024 (filtro metadata)".
- **Rendimiento:** Al estar escrito en *Rust*, ofrece latencias muy bajas incluso con millones de vectores.
- **Facilidad de uso:** Dispone de una *API REST* e integración con frameworks de IA.

Esta pieza es fundamental para habilitar las funcionalidades RAG (Retrieval-Augmented Generation) del asistente inteligente, permitiéndole "recordar" y consultar toda la base de conocimientos técnica de la organización antes de responder al usuario.

3.5 Orquestación y Despliegue

Una arquitectura basada en **microservicios** organiza cada módulo funcional —frontend, backend, IA, bases de datos y almacenamiento— como un servicio independiente. Esto permite escalar componentes por separado, actualizar uno sin afectar al conjunto y aislar fallos. Para gestionar este tipo de arquitecturas se dispone de varias herramientas de orquestación, con características muy diferentes según el nivel de complejidad y los requisitos de producción:

- **Docker Compose:**

- Permite definir y ejecutar múltiples contenedores con un único archivo `docker-compose.yml`.
- Facilita el despliegue local y la replicación exacta de entornos en desarrollo y pruebas.
- Adecuado para entornos pequeños, MVPs o servidores individuales.
- No incluye balanceo de carga ni escalado automático.

- **Docker Swarm:**

- Sistema de orquestación nativo de Docker que permite crear clústeres distribuidos.
- Ofrece descubrimiento de servicios, replicación automática y balanceo de carga integrado.
- Ideal como paso intermedio entre Compose y Kubernetes, con una curva de aprendizaje baja.
- Utiliza el mismo formato de definición que Docker Compose, lo que facilita la migración.

- **Kubernetes:**

- Plataforma de orquestación de contenedores avanzada y extensible.
- Permite autoescalado, actualizaciones sin downtime, tolerancia a fallos y gestión declarativa.
- Soporta integración con herramientas externas como NGINX Ingress, Prometheus, Helm o ArgoCD.
- Es la opción más robusta y utilizada en entornos de producción complejos o cloud-native.

Gracias a esta modularidad, el despliegue de la plataforma puede adaptarse fácilmente a distintos entornos: desde entornos locales o de desarrollo con Docker Compose, pasando por despliegues simplificados con Docker Swarm, hasta arquitecturas empresariales completas gestionadas mediante Kubernetes.

3.6 Integración y Entrega Continua (CI/CD)

En el contexto del desarrollo moderno de software, resulta fundamental automatizar los procesos de construcción, prueba y despliegue de las aplicaciones. Esta automatización no solo mejora la eficiencia del equipo de desarrollo, sino que también garantiza la coherencia entre entornos, reduce errores humanos y permite realizar entregas frecuentes y confiables. Esta práctica se conoce como *Integración Continua (CI)* y *Entrega Continua (CD)*.

- **Integración Continua (Continuous Integration):** Consiste en integrar los cambios realizados en el código fuente de manera frecuente (varias veces al día) en un repositorio central, donde se ejecutan automáticamente pruebas y validaciones para detectar errores lo antes posible.
- **Entrega Continua (Continuous Delivery):** Se refiere a la capacidad de desplegar automáticamente las nuevas versiones del software en entornos de pre-producción o producción tras superar las etapas de prueba, manteniendo siempre el producto en un estado listo para liberar.

Para implementar estas prácticas existen múltiples herramientas que permiten definir flujos de trabajo automatizados mediante pipelines. Las soluciones más utilizadas son:

- **Bitbucket + Bamboo:**
 - Bitbucket es un sistema de control de versiones basado en Git, desarrollado por Atlassian.
 - Bamboo es la herramienta de CI/CD también de Atlassian, integrada con Bitbucket y Jira.
 - Permite definir pipelines de construcción, pruebas y despliegue, con control de versiones y trazabilidad.
- **Jenkins:**
 - Jenkins es una de las herramientas de CI/CD más veteranas y ampliamente adoptadas.
 - Se caracteriza por su flexibilidad y su extenso ecosistema de plugins.
 - Permite construir pipelines complejos, integrarse con cualquier sistema de control de versiones y desplegar a múltiples entornos.
- **GitLab + GitLab Runner** GitLab Inc., 2024; Kim et al., 2016:
 - GitLab es una plataforma DevOps completa que incluye control de versiones, gestión de proyectos y CI/CD integrados.
 - GitLab Runner es el componente que ejecuta las tareas definidas en los `.gitlab-ci.yml` para compilar, testear y desplegar el software.
 - Ofrece una integración nativa y sencilla, ideal para proyectos donde se busca centralizar todo el ciclo de vida del desarrollo.

La elección de la herramienta dependerá del ecosistema del equipo de desarrollo, la infraestructura disponible y las necesidades de trazabilidad y seguridad. En cualquier caso, la implementación de CI/CD representa una mejora sustancial en la calidad del software, la velocidad de entrega y la fiabilidad del sistema en producción.

3.7 Ecosistemas de Despliegue e Infraestructura

Actualmente, una solución tecnológica puede desplegarse de tres formas principales. La primera es **Bare Metal**, donde la aplicación se ejecuta directamente sobre el servidor físico para aprovechar al máximo el hardware. La segunda es la **Virtualización**, que introduce un hipervisor para repartir recursos entre varias máquinas aisladas. La tercera es el modelo **Cloud**, donde la infraestructura la gestiona un proveedor externo y se paga según uso. La elección depende del control que se quiera mantener, la facilidad de administración y el coste operativo asumible.

3.7.1 Modelos de Infraestructura

1. **Bare Metal (Servidor Físico):** Consiste en ejecutar las aplicaciones directamente sobre el hardware, sin capas intermedias. Ofrece el mejor rendimiento, especialmente cuando se necesita acceso directo a GPU o baja latencia de disco, aunque reduce la flexibilidad para aislar servicios.
2. **Entornos Virtualizados (On-Premise):** Representan un equilibrio entre control y flexibilidad. Un hipervisor permite ejecutar varios sistemas operativos aislados sobre el mismo servidor físico, aprovechando mejor los recursos disponibles.
 - **VMware vSphere:** Enfocado a la estabilidad empresarial con herramientas de automatización de alto nivel, aunque con un modelo de licenciamiento de coste creciente.

- **Proxmox VE** Proxmox Server Solutions GmbH, 2024: Basado en estándares abiertos, permite combinar la potencia de KVM con la ligereza de los contenedores LXC, siendo una opción altamente eficiente para arquitecturas de microservicios.

3. Ecosistema Cloud (AWS, Azure, Google Cloud): Ofrece infraestructura gestionada por un proveedor externo. Simplifica el arranque y permite escalar bajo demanda, pero puede elevar mucho el coste mensual cuando se necesitan grandes volúmenes de almacenamiento persistente o instancias con GPU para IA.

3.7.2 Comparativa Técnica de Entornos

La tabla siguiente contrasta las diferencias clave entre los tres modelos de despliegue:

Tabla 3.4 Contraste entre Bare Metal, Virtualización y Cloud.

Característica	Bare Metal	Virtualización (Proxmox)	Cloud Público
Aislamiento	Total (Físico)	Lógico (VM/LXC)	Lógico (Multi-tenant)
Rendimiento	Máximo (100 %)	Alto (~95-98 %)	Variable (Instancia)
Escalabilidad	Limitada/Manual	Alta (Clustering)	Casi Infinita
Control Hardware	Total	Alto (Passthrough)	Nulo
Inversión (CapEx)	Alta inicial	Media/Alta	Nula
Gasto (OpEx)	Muy Bajo	Bajo	Muy Alto

4 Elección del Stack Tecnológico

La arquitectura se organiza en capas con funcionalidades claras: infraestructura virtualizada, orquestación de contenedores, persistencia de datos, APIs encargada de la lógica de negocio, frontend e integración continua. Esta separación facilita actualizar o sustituir una parte sin afectar a las demás, y permite una toma de decisiones de forma independiente. La Figura 4.1 muestra el esquema general del sistema.



Figura 4.1 Propuesta de arquitectura por capas.



Figura 4.2 Tecnologías que conforman el stack de la plataforma ETMOV.

4.1 Capa 1: Infraestructura y Virtualización

La base de la plataforma es la infraestructura donde se ejecutarán los servicios. En este proyecto se consideraron tres enfoques habituales: *bare metal* (máximo rendimiento pero menor flexibilidad), *cloud pública* (elasticidad y servicios gestionados, a costa de dependencia y costes variables) y *nube privada virtualizada*. Se ha optado por esta última porque permite controlar el entorno (importante cuando se trabaja con documentación sensible), aprovechar mejor el hardware disponible y mantener una operación razonable para el alcance del TFG.

4.1.1 Hipervisor: Proxmox Virtual Environment (VE)

Se selecciona *Proxmox* Proxmox Server Solutions GmbH, 2024 como plataforma de virtualización por tres motivos principales: (i) es software abierto y ampliamente utilizado, (ii) integra virtualización completa y contenedores de sistema, y (iii) facilita la gestión del almacenamiento con ZFS Bonwick et al., 2004; Patterson, Gibson y Katz, 1988. Alternativas como *VMware vSphere* son muy maduras, pero su coste y modelo de licenciamiento hacen que no encajen bien con un despliegue académico o de pequeña organización.

En una fase de *desarrollo o pilotaje* donde el control de costes es importante pero se requiere potencia para la indexación vectorial y el procesamiento de imágenes, este entorno virtualizado ofrece la mejor relación entre flexibilidad de gestión y ahorro operativo.

Virtualización con KVM y contenedores LXC

Proxmox combina dos formas de aislar servicios:

- **KVM (máquinas virtuales):** recomendado para servicios que requieren un aislamiento más fuerte o un sistema operativo completo (por ejemplo, nodos del clúster de Kubernetes o servicios con requisitos específicos).
- **LXC (contenedores de sistema):** útil para servicios auxiliares (monitorización, herramientas de apoyo, etc.), con un uso de recursos más eficiente y una puesta en marcha rápida.

En la práctica, esta combinación permite asignar recursos de forma realista: lo crítico va en VMs y lo auxiliar puede ir en contenedores, manteniendo un buen equilibrio entre aislamiento y eficiencia.

Sistema de archivos ZFS

Otro punto a favor de Proxmox es su integración con *ZFS*, que aporta ventajas relevantes cuando el sistema gestiona expedientes y evidencias durante años:

- **Copy-on-Write:** reduce el riesgo de inconsistencias ante cortes inesperados.
- **Verificación de integridad:** permite detectar corrupción silenciosa de datos, especialmente importante en ficheros y backups.
- **Snapshots:** facilita puntos de restauración y copias consistentes, útiles para pruebas y para recuperación ante fallos.

4.2 Capa 2: Orquestación de Cargas de Trabajo

Una vez definida la infraestructura, el siguiente paso es cómo ejecutar y mantener los distintos servicios (API, frontend, bases de datos, reportajes fotográficos y servicios de IA). La plataforma se apoya en contenedores para facilitar despliegues repetibles y coherentes entre entornos. A partir de ahí, se consideraron dos opciones: *Docker Swarm* (simple y directo) y *Kubernetes* Burns et al., 2022 (más completo y estándar en producción).

En un ámbito académico la opción de *Docker Swarm* gana terreno, pero en un entorno orientado a un MVP (*Producto Mínimo Viable*), la opción recomendada sería *Kubernetes*.

4.2.1 Orquestador: Kubernetes

Para una solución por servicios que incluye componentes con estado (bases de datos y almacenamiento) y necesidades de red más elaboradas, Kubernetes ofrece capacidades que simplifican la operación: actualizaciones controladas, reinicio automático de pods, descubrimiento de servicios y gestión de configuración y secretos. Aunque su curva de aprendizaje es mayor, se justifica porque la plataforma tiene vocación de crecer y porque el despliegue se documenta como parte del trabajo.

La plataforma se expone al público mediante *Ingress Controller*. Se utiliza para centralizar el enrutado HTTP/HTTPS, aplicar TLS y definir reglas de acceso a los distintos servicios (por ejemplo, separar API y frontend por rutas o subdominios). Este enfoque evita configurar proxies manualmente por cada microservicio y facilita mantener una entrada única y controlada a la plataforma.

Almacenamiento persistente

Kubernetes permite definir *Persistent Volumes* para componentes con estado (como MongoDB o MinIO). Esto resulta importante para que, si un pod se reinicia o se mueve, los datos sigan estando disponibles. En el alcance de este trabajo, el objetivo no es construir un sistema de alta disponibilidad “de operador cloud”, sino garantizar persistencia, reproducibilidad y un despliegue consistente.

A la fecha de realización de este trabajo existen diversas alternativas para desplegar un ecosistema *Kubernetes* (*K8s*), entre ellos se ha optado por *MicroK8s* por su facilidad de instalación y por ofrecer un conjunto de addons que acelera la puesta en marcha. Esto permite centrarse en el diseño de la plataforma y no en la complejidad de un clúster “desde cero”.

4.3 Capa 3: Persistencia y gestión de datos

Un expediente de vehículo histórico mezcla información estructurada (datos del vehículo, estados del expediente, revisiones) con grandes volúmenes de ficheros (PDFs, imágenes, anexos). Además, si se incluye búsqueda semántica y asistencia inteligente, aparece un tercer tipo de dato: embeddings vectoriales. Por eso se utiliza un motor distinto para cada tipo de dato, eligiendo en cada caso la herramienta más adecuada según su naturaleza.

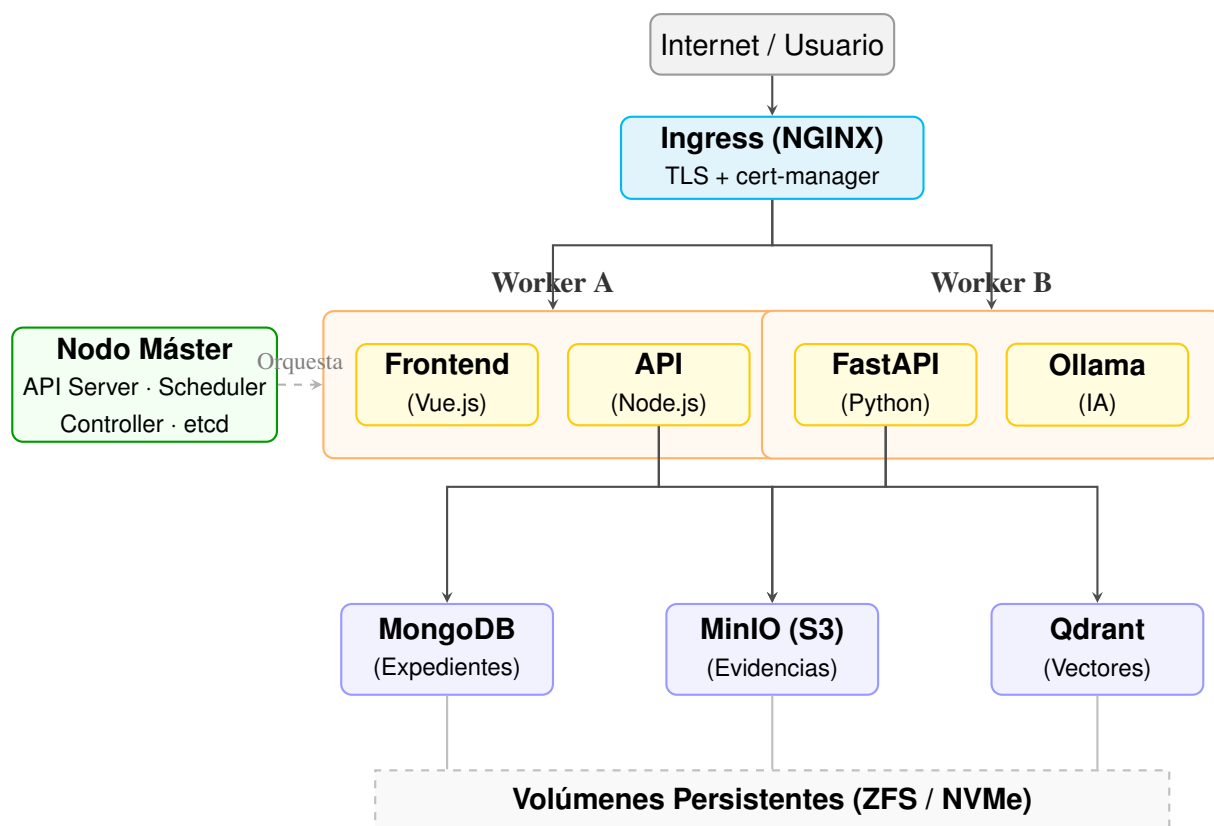


Figura 4.3 Topología lógica del clúster Kubernetes: nodo máster, nodos trabajadores y persistencia..

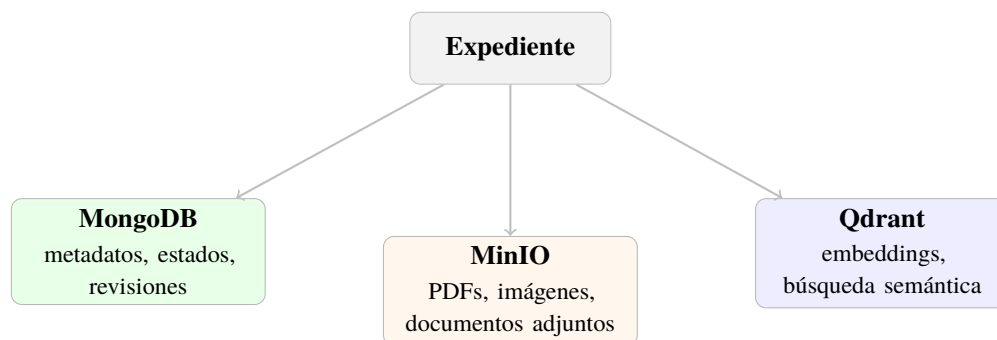


Figura 4.4 Tres motores, tres tipos de dato: cada tecnología gestiona lo que mejor conoce.

4.3.1 Base de datos documental: MongoDB

Se selecciona *MongoDB* Chodorow, 2013 para almacenar el expediente como documento, porque el contenido real varía entre casos: no todos los vehículos aportan la misma documentación, y los anexos pueden cambiar según el tipo de vehículo y su historia. Un modelo documental permite añadir o ajustar campos sin forzar migraciones complejas, y encaja bien con APIs basadas en JSON.

Comparación con SQL

Una base de datos relacional (por ejemplo, *PostgreSQL*) es una opción excelente cuando el esquema es muy estable y las relaciones están claramente normalizadas. En este proyecto, sin embargo, el expediente se comporta como un “contenedor” de información que crece con anexos y evidencias,

y que se consulta habitualmente como un todo. Por eso se prioriza el modelo documental, que simplifica lecturas y reduce fricción al evolucionar el modelo de datos.

4.3.2 Almacenamiento de objetos: MinIO

Los documentos adjuntos que enriquecen la elaboración de un expediente (fotos, PDFs y documentación escaneada) se almacenan en un sistema de objetos compatible con *S3*. Se utiliza *MinIO* MinIO Inc., 2023 para separar los ficheros grandes de la base de datos, mantener un acceso eficiente desde el frontend y permitir políticas de versionado o retención cuando sea necesario. *MongoDB* guarda referencias y metadatos, mientras que *MinIO* almacena el contenido.

4.3.3 Base de datos vectorial: Qdrant

Para funciones como búsqueda semántica (por ejemplo, localizar antecedentes similares o encontrar fragmentos relevantes de normativa y documentación), se utiliza un motor vectorial. *Qdrant* Qdrant Team, 2023 ofrece una API sencilla, buen rendimiento y soporte para combinar búsqueda vectorial con filtros por metadatos (por ejemplo, limitar resultados por categoría documental o fecha del expediente).

4.4 Capa 4: Backend y lógica de negocio

La plataforma necesita una capa de servicios que gestione usuarios, permisos, expedientes y flujo de trabajo (creación, edición, revisión y cierre), además de integrar procesamiento documental y asistencia inteligente. Se separan dos tipos de backend: uno orientado a APIs y orquestación, y otro orientado a tareas de procesamiento.

4.4.1 API principal: Node.js + Express

La API principal se construye con *Node.js* Cantelon et al., 2014 y *Express* Express.js Contributors, 2024. Gestiona el flujo habitual de la aplicación: autenticación y control de acceso mediante JWT, operaciones sobre expedientes (creación, consulta, edición y cierre) y coordinación de las subidas de documentos hacia MinIO. Su modelo de E/S no bloqueante responde bien a múltiples peticiones concurrentes sin necesidad de configuración compleja de hilos, lo que lo hace adecuado para una API web de uso moderado.

4.4.2 Servicios de procesamiento e IA: Python + FastAPI

Las tareas de procesamiento intensivo —OCR, extracción de datos y generación de embeddings— se separan en un servicio independiente construido con *Python* y *FastAPI* Ramírez, 2024. Python es la opción natural para este tipo de trabajo dada la madurez del ecosistema de librerías de IA disponibles. Ejecutar este servicio por separado evita que analizar un PDF largo bloquee o ralentice la API principal, y permite escalar el procesamiento de forma independiente si el volumen de documentos crece.

4.5 Capa 5: Frontend y experiencia de usuario

La interfaz de usuario sigue una arquitectura *SPA* (*Single Page Application*): una aplicación que carga una vez y navega entre vistas sin recargar la página. En un servicio técnico donde el técnico trabaja a la vez con formularios, documentos y fotografías, este modelo reduce las interrupciones de contexto. El objetivo de diseño es simple: que toda la información relevante de un expediente esté accesible desde una misma pantalla.

4.5.1 Framework: Vue.js 3

Se elige *Vue.js 3* You y V. Contributors, 2024 por su curva de aprendizaje razonable y su enfoque basado en componentes, que facilita dividir la interfaz en partes reutilizables: ficha del vehículo, panel de evidencias, editor del informe. La *Composition API* de Vue 3 permite organizar la lógica de cada componente de forma más clara que el modelo de opciones anterior, lo que ayuda a mantener el código a medida que la interfaz crece. Se combina con *Tailwind CSS* Wathan et al., 2024 para definir estilos directamente en los componentes sin necesidad de hojas de estilo separadas.

4.5.2 Gestión de estado: Pinia

Pinia Morote y P. Contributors, 2024 actúa como el almacén de estado global de la aplicación. Su función principal es evitar que cada componente tenga que solicitar de nuevo los mismos datos. Un *store* centraliza la información compartida —el expediente activo, la sesión del usuario, los filtros de búsqueda— y la pone a disposición de todas las vistas sin peticiones redundantes, manteniendo la interfaz coherente incluso cuando el técnico navega entre secciones.

4.6 Capa 6: Operación, seguridad y DevOps

Para que el despliegue sea reproducible y el código sea trazable, se aplican prácticas básicas de DevOps: un pipeline de CI/CD automatizado, gestión de secretos separada del código fuente y un registro privado de imágenes de contenedor. En un proyecto de alcance académico estas prácticas tienen valor real: documentan cómo se construye y despliega cada versión, y permiten reproducir el entorno sin pasos manuales.

4.6.1 Gestión de credenciales y configuración

Las contraseñas, claves de API y parámetros de configuración se mantienen fuera del código fuente. En Kubernetes esto se gestiona mediante *Secrets* y *ConfigMaps*: los primeros almacenan valores sensibles cifrados, los segundos opciones de configuración no confidenciales. Este enfoque sigue las pautas del *twelve-factor app* y evita que credenciales acaben en el historial de git por error.

4.6.2 Registro de imágenes y análisis de vulnerabilidades

Para mantener controladas las imágenes de contenedor, se utiliza un registro privado (*Harbor*) y un escáner de vulnerabilidades (*Trivy*) Aqua Security, 2024. El objetivo es detectar dependencias con *CVEs* relevantes antes del despliegue, y mantener un mínimo de higiene de seguridad en el ciclo de vida del software.

4.6.3 CI/CD con GitLab

Se emplea *GitLab CI/CD* GitLab Inc., 2024 para automatizar tareas repetitivas: ejecutar pruebas, construir imágenes y desplegar versiones. Esto reduce errores manuales y deja trazabilidad sobre qué versión del código genera cada despliegue, algo muy útil tanto en desarrollo como a la hora de documentar el trabajo.

4.7 Resumen del Stack Tecnológico Elegido

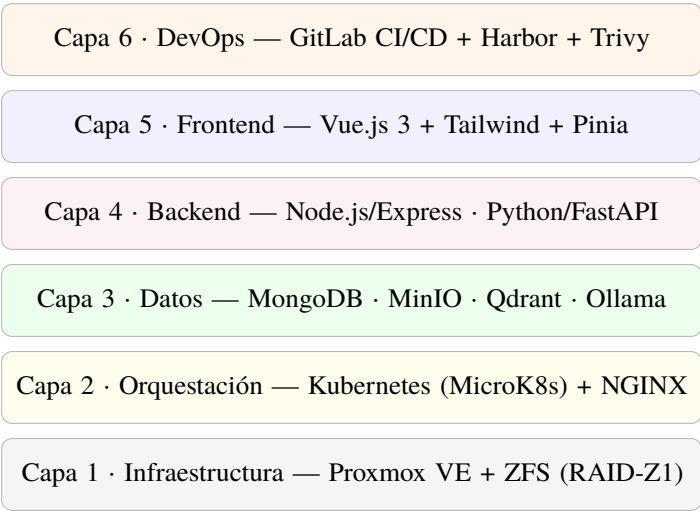


Figura 4.5 Stack tecnológico final organizado por capas.

A modo de cierre, la siguiente tabla recoge el conjunto de herramientas seleccionadas para cada capa de la plataforma. Cada tecnología se eligió por razones concretas que se han desarrollado en las secciones anteriores; aquí se presenta el resultado final como referencia rápida.

Tabla 4.1 Resumen del Stack Tecnológico.

Capa de la Arquitectura	Tecnología Seleccionada	Logo
Infraestructura	Proxmox VE	
Orquestación	Kubernetes (MicroK8s)	
Frontend	Vue.js 3 + Tailwind CSS	
Backend (API)	Node.js + Express	
Procesamiento IA	Python + FastAPI	
Base de Datos Documental	MongoDB	
Base de Datos Vectorial	Qdrant	
Almacenamiento de Objetos	MinIO (S3 Compatible)	
Orquestación de LLM	Ollama	
CI/CD + Registry	GitLab CI/CD + Harbor	

5 Diseño e Implementación de la Solución

5.1 Introducción al Despliegue

Tras la definición de la arquitectura y la selección del stack tecnológico, este capítulo detalla el proceso de implementación técnica de la plataforma *ETMOV*.¹ Se aborda desde el aprovisionamiento de la infraestructura física hasta el desarrollo de los componentes de software de alto nivel y la integración de los modelos de IA.

5.2 Aprovisionamiento de la Infraestructura Base

El punto de partida de la plataforma es un servidor físico de alto rendimiento. Sobre este equipo se ha desplegado *Proxmox VE* para convertir el hardware disponible en un entorno flexible, capaz de alojar varias máquinas virtuales y servicios aislados. Las decisiones principales se han tomado pensando en dos objetivos: proteger los expedientes técnicos y dejar margen para ejecutar tareas de IA con suficiente capacidad de cómputo:

- **Configuración del Host:** Se ha priorizado el modo *UEFI* y la activación de *IOMMU*, permitiendo que el sistema esté preparado para el paso de dispositivos (*passthrough*). Esta decisión es clave para que las VMs puedan acceder directamente a la potencia de cálculo de la GPU en tareas intensivas de IA.
- **Estrategia de Almacenamiento:** Se han integrado tres unidades *SSD NVMe* bajo el sistema de archivos *ZFS* Bonwick et al., 2004. La elección de *RAID-Z1* Patterson, Gibson y Katz, 1988 responde a la necesidad de balancear la velocidad de lectura (esencial para el procesamiento de imágenes) con la tolerancia a fallos, permitiendo que el servicio no se detenga ante una avería física de un disco.
- **Gestión de Recursos:** Se ha optimizado el *Adaptive Replacement Cache (ARC)* de *ZFS* para limitar su consumo de memoria RAM, asegurando que la mayor parte de este recurso esté disponible para los modelos de lenguaje que orquestan la asistencia inteligente.

Con esta configuración, las máquinas virtuales se apoyan en una base estable, con protección frente a fallos de disco y preparada para crecer si aumenta la carga de trabajo.

¹ La plataforma está desplegada y accesible en <https://etmov.famluro.es>.

5.3 Arquitectura de Almacenamiento y Red SDN

5.3.1 Arquitectura de Almacenamiento

La arquitectura de almacenamiento se ha diseñado como el repositorio principal de expedientes y evidencias del servicio técnico. Se han empleado unidades *PCIe 4.0* para reducir cuellos de botella durante la subida y consulta de documentación fotográfica.

Configuración RAID-Z1 y Resiliencia

El uso de **RAID-Z1** sobre ZFS proporciona una capa de abstracción que protege al expediente técnico contra la corrupción silenciosa de datos (*bit rot*). A diferencia de los sistemas tradicionales, ZFS verifica la integridad de cada bloque mediante sumas de verificación (*checksumming*) en tiempo real. Las ventajas operativas para ETMOV son:

- **Disponibilidad Continua:** El sistema tolera la pérdida total de una unidad física sin interrumpir el acceso a los expedientes activos.
- **Rendimiento de E/S:** La lectura simultánea sobre los tres discos permite que las tareas de OCR y generación de *embeddings* se ejecuten con latencias mínimas.
- **Ahorro mediante Compresión:** Se ha activado el algoritmo *LZ4* en los sub-volúmenes de backups, lo que permite reducir el espacio en disco sin penalizar el rendimiento de la CPU.

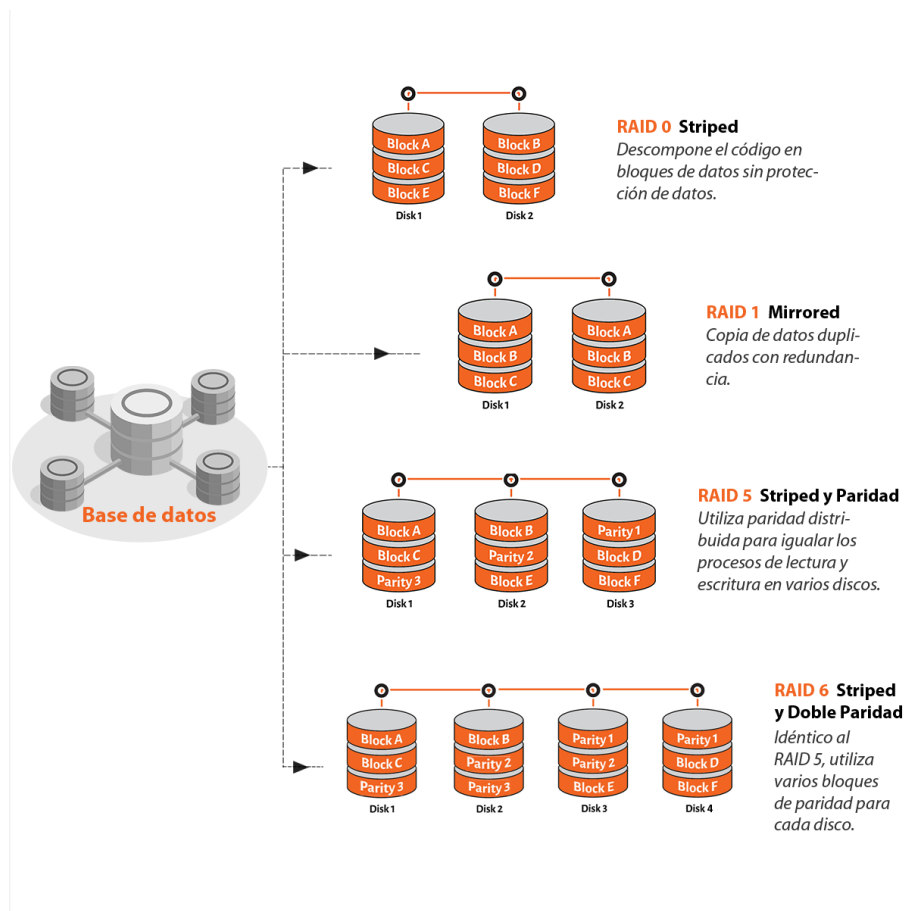


Figura 5.1 Configuración RAID-Z1 .

Optimización de Datasets

Para exprimir la velocidad de las unidades del almacenamiento y la gestión lógica de *ZFS*, se han configurado sub-volúmenes con parámetros específicos:

Dataset *vm-disks*: Se ha establecido un "*recordsize=64k*". Este ajuste es crítico para optimizar el rendimiento de las imágenes de disco de las máquinas virtuales, ya que reduce la amplificación de escritura y alinea los bloques de datos con los patrones de acceso de los sistemas virtualizados.

Dataset *backups*: Se ha activado la compresión *LZ4*. Este algoritmo es extremadamente eficiente en *CPU* ya permite comprimir copias de seguridad y *snapshots* en tiempo real, aumentando la vida útil de los *NVMe* al escribir menos datos físicos sin sacrificar la velocidad de transferencia.

5.3.2 Infraestructura de Red Definida por Software (SDN)

Para organizar el tráfico entre los servicios del clúster (frontend, API e IA) se ha implementado una red definida por software (*SDN*) basada en puentes virtuales, tal como se ilustra en la Figura 5.2. Esta topología permite que las máquinas virtuales y los microservicios se comuniquen dentro del propio servidor, reduciendo latencia y manteniendo aislado el tráfico relacionado con los expedientes.

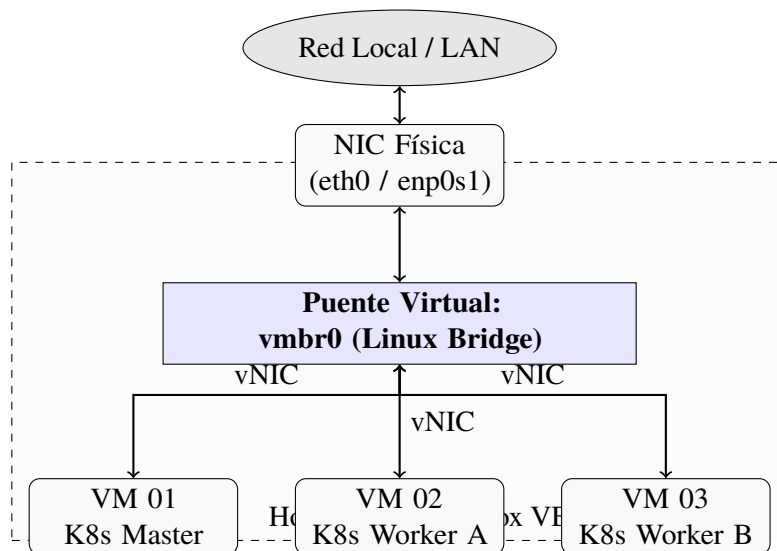


Figura 5.2 Topología de red SDN: Abstracción del tráfico físico a lógico mediante puentes virtuales..

5.4 Virtualización y Estrategia de Plantilla Maestra

Para garantizar la repetibilidad del entorno de trabajo, no se ha realizado una instalación manual de cada servidor. En su lugar, se ha creado una **Plantilla Maestra (Golden Image)** sobre *Ubuntu Server 24.04 LTS*. Esta plantilla incorpora las optimizaciones críticas de rendimiento analizadas en capítulos previos:

- **Controladores *VirtIO*** Kochhar y Arora, 2016: Se emplearon para la virtualización para-virtualizada de disco y red, lo que minimiza el consumo de CPU en tareas de entrada/salida intensivas.
- **Integración con el Hipervisor:** Se configuró el *QEMU Guest Agent* para permitir que Proxmox realice *snapshots* consistentes con el sistema de archivos, asegurando copias de seguridad sin corrupción.

- **Aprovisionamiento Elástico:** La plantilla permite clonar nodos del clúster de Kubernetes en segundos, asegurando que todos los entornos (desarrollo, pruebas y producción) sean idénticos.

Esta metodología de "Infraestructura como Código" permite que el despliegue del clúster sea homogéneo, eliminando errores de configuración individual y facilitando el mantenimiento a largo plazo de la plataforma.

5.5 Orquestación con Kubernetes

Para gestionar la complejidad de los microservicios, se ha desplegado un clúster de **Kubernetes** Burns et al., 2022; Lukša, 2019 utilizando la distribución *MicroK8s*. La orquestación no es solo una elección tecnológica, sino la garantía de que el servicio técnico de ETMOV sea ininterrumpido.

5.5.1 Configuración de Alta Disponibilidad (HA)

El clúster se compone de tres nodos virtuales que operan bajo un esquema de **Alta Disponibilidad**. Mediante el motor de consenso *Dqlite*, el clúster garantiza que, ante la caída de un nodo, el resto de la infraestructura pueda reubicar los pods (contenedores) de forma automática. Esta capacidad es vital para mantener la integridad de los informes técnicos en proceso sin pérdida de sesión para el usuario.

5.5.2 Ingress y Gestión de Tráfico Seguro

Para la exposición de los servicios al exterior, se ha configurado un *Ingress Controller* basado en **NGINX** NGINX Inc., 2024. Este componente actúa como un proxy inverso inteligente que centraliza el tráfico seguro mediante certificados TLS. La automatización de este proceso se realiza a través de *cert-manager* cert-manager Authors, 2024, asegurando que toda la documentación que viaja entre el cliente y el servidor esté siempre cifrada, cumpliendo con los estándares de seguridad y protección de datos analizados en el Capítulo 2.

5.6 Implementación de la Capa de Datos

La **capa de datos** es el corazón de la plataforma: es donde se guarda el expediente (datos del vehículo, estados, revisiones), las evidencias (PDFs e imágenes) y la información necesaria para la búsqueda inteligente. En vez de forzar una única tecnología para todo, se adopta una estrategia sencilla de “cada cosa en su sitio”:

La gestión de datos se ha resuelto mediante una **distribución del almacenamiento por tipo de dato**: cada motor se encarga de lo que mejor sabe hacer, lo que mejora tanto el rendimiento como la capacidad de búsqueda del conjunto.

5.6.1 MongoDB: modelo de documento de inspección

En MongoDB Chodorow, 2013 se diseñó un **documento agregado** que concentra la información que normalmente se consulta “de golpe” cuando se abre un expediente: identificación del expediente, estado, datos del vehículo, reformas/observaciones y trazabilidad básica. Este enfoque reduce la necesidad de “reconstruir” el informe uniendo muchas tablas, y se ajusta bien a un flujo en el que cada expediente puede tener variaciones (no todos aportan la misma documentación o presentan las mismas reformas).

El siguiente ejemplo es ilustrativo (no pretende ser el único posible). Muestra cómo se organiza un expediente con su vehículo, reformas y un registro de auditoría de acciones:

```
{
  "_id": "ObjectId('...')",
  "expediente": "EXP-2024-001",
  "estado": "BORRADOR",      // BORRADOR, EN_REVISION, REVISADO, FIRMADO
  "vehiculo": {
    "bastidor": "WWZZZ...",
    "marca": "SEAT",
    "modelo": "LEON",
    "datos_tecnicos": {
      "cv_fiscales": 14.9,
      "neumaticos": ["225/45 R17", "225/40 R18"]
    }
  },
  "reformas": [
    {
      "codigo": "8.52",
      "descripcion": "Reforma de carrocería",
      "evidencias": [
        "s3://bucket-evidencias/foto_aleron_trasero.jpg",
        "s3://bucket-evidencias/cert_taller.pdf"
      ],
      "estado_validacion": "OK"
    }
  ],
  "traza_auditoria": [
    { "usuario": "ing_perez", "accion": "MODIFICAR_ALTURA",
      "fecha": "ISO8601..." }
  ]
}
```

5.6.2 MinIO: almacenamiento de evidencias y documentos

Para la gestión de los archivos binarios (imágenes y PDFs) se ha desplegado un sistema compatible con *Amazon S3* mediante **MinIO** MinIO Inc., 2023. Esta separación permite que la base de datos MongoDB sea extremadamente ligera y rápida, delegando el tráfico pesado de imágenes a un sistema optimizado para objetos inmutables. Se han definido políticas de retención de datos para evitar el borrado accidental de evidencias críticas para el cumplimiento normativo.

5.6.3 Qdrant: búsqueda semántica y soporte a la asistencia

La verdadera inteligencia del sistema reside en su motor vectorial. Mediante **Qdrant** Qdrant Team, 2023; Reimers y Gurevych, 2019, se han indexado los informes históricos convirtiendo el lenguaje técnico en vectores matemáticos (*embeddings*). Esto permite habilitar la búsqueda semántica: un técnico puede localizar precedentes de catalogación no solo por marca o modelo, sino por la similitud técnica de las reformas aplicadas en el pasado.

5.7 Desarrollo del Backend y APIs

La plataforma se apoya en una API que centraliza la lógica de negocio: autenticación, gestión de expedientes, control de estados, subida de evidencias y consulta de información. Para que el diseño sea más mantenible, se separan dos responsabilidades:

1. **API principal (Node.js + Express):** gestiona el flujo normal de la aplicación web, operaciones CRUD de expedientes y acceso controlado por usuarios.
2. **Servicio de procesamiento e IA (Python + FastAPI):** ejecuta tareas más pesadas (OCR, extracción y cálculo de embeddings) sin bloquear la experiencia del usuario.

Esta separación tiene una ventaja sencilla: si el procesamiento de documentos tarda (lo cual es normal en PDFs largos), el usuario puede seguir trabajando en el resto del sistema.

5.7.1 API Gateway (Node.js + Express)

La API principal actúa como puerta de entrada única para todas las operaciones de la plataforma: autenticación de usuarios, gestión del ciclo de vida de los expedientes y control de acceso a los recursos. Su diseño sigue el patrón de *controlador-servicio-modelo*, separando la lógica de negocio de la capa de acceso a datos y facilitando el mantenimiento a largo plazo.

Autenticación y autorización

El sistema de autenticación se basa en **JSON Web Tokens (JWT)** Jones, Bradley y Sakimura, 2015 con caducidad de siete días. Al recibir las credenciales del usuario, el servidor verifica la contraseña mediante *bcrypt* Provos y Mazieres, 1999 (factor de coste 10) y, en caso de éxito, emite un token firmado con el secreto de aplicación. Este token viaja en la cabecera `x-access-token` de cada petición posterior y es verificado por el middleware de autorización antes de que la solicitud alcance cualquier controlador.

- **Rotación de tokens:** El sistema mantiene un máximo de diez tokens activos por usuario, eliminando automáticamente los más antiguos caducados, lo que previene la acumulación de sesiones obsoletas.
- **Registro de actividad:** Cada inicio de sesión queda registrado con fecha, hora y dirección IP, proporcionando una traza de auditoría completa consultable desde el perfil de usuario.
- **OAuth2 con Google** Hardt, 2012: Se integra la librería *google-auth-library* para ofrecer inicio de sesión federado, eliminando la necesidad de gestionar contraseñas para los usuarios que prefieran este método.

Este esquema cumple con los estándares de seguridad de ecosistemas *Zero Trust*: ningún recurso es accesible sin un token válido, y la revocación de acceso se realiza en tiempo real eliminando el token del listado activo del usuario.

5.7.2 Servicio de Inteligencia Artificial (Python + FastAPI)

El flujo de trabajo para procesar un PDF se puede resumir en cuatro pasos:

1. **Preparación:** se convierten páginas a imágenes para facilitar la detección (cuando el PDF no contiene texto seleccionable).
2. **OCR** Mori, Suen y Yamamoto, 1992; Xu et al., 2020: se detecta texto y, cuando es posible, estructura (tablas, bloques).
3. **Extracción de datos clave:** se identifican elementos útiles (bastidor/VIN, fechas, referencias) y se devuelven como sugerencias para revisión.

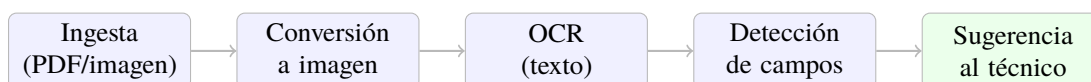


Figura 5.3 Pipeline de procesamiento documental: del archivo al dato sugerido.

Un punto importante para un trabajo académico es el enfoque **human-in-the-loop** Monarch, 2021: el sistema *sugiere*, pero el técnico *confirma o corrige*. Esto reduce errores y hace el proceso defendible.

Implementación de RAG para consulta y redacción

Para que la IA no redacte a partir de información genérica o no comprobada, se implementa un flujo **RAG** (*Retrieval-Augmented Generation*) Lewis et al., 2020. La idea es sencilla: antes de generar una respuesta, el sistema busca documentos reales relacionados con la consulta y utiliza esos fragmentos como contexto. Así, el borrador se apoya en evidencias disponibles dentro de la plataforma.

- **Recuperación de evidencias:** Qdrant localiza los fragmentos de expedientes, normativa o documentación que más se parecen a la consulta del técnico.
- **Generación acotada al contexto:** el modelo recibe esos fragmentos junto con la pregunta, de forma que la respuesta queda limitada a la información recuperada.

Este diseño reduce el riesgo de alucinaciones. Si la consulta no encuentra documentos suficientes o las fuentes recuperadas no permiten responder con seguridad, el sistema debe indicarlo en lugar de inventar una explicación.

El flujo se resume en los siguientes pasos:

1. La consulta del técnico se convierte en un vector mediante un modelo de *embeddings*.
2. Qdrant compara ese vector con los documentos indexados y ordena los resultados por similitud.
3. Se seleccionan los K fragmentos más relevantes y se incorporan al contexto del modelo.
4. El modelo genera una propuesta con `temperature=0.0`, priorizando respuestas reproducibles y fieles a las fuentes.

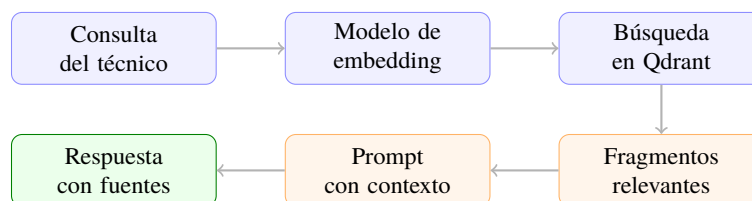


Figura 5.4 Flujo RAG de consulta y redacción: la respuesta se genera a partir de fragmentos recuperados..

Asistente conversacional y agentes MCP

El flujo RAG se amplía mediante servidores **MCP (Model Context Protocol)** Anthropic, 2024. Estos servidores actúan como conectores entre el modelo y las fuentes reales de la plataforma. En lugar de obligar al técnico a buscar manualmente en cada sistema, el agente interpreta la petición y consulta la herramienta adecuada.

Esta capacidad se articula inyectando como contexto operativo las siguientes herramientas nativas:

- **MCP de expediente:** acceso de lectura a los documentos de MongoDB; el agente puede consultar estado, campos, reformas registradas y trazabilidad de un expediente concreto.
- **MCP de documentos:** lista los archivos almacenados en MinIO para un expediente y recupera su contenido textual cuando es necesario.
- **MCP semántico:** lanza búsquedas por similitud en Qdrant para encontrar secciones de expedientes anteriores relevantes a la consulta actual.

- **MCP normativo:** busca en la base de conocimiento indexada (reglamento, instrucciones de la DGT, criterios internos del servicio técnico).

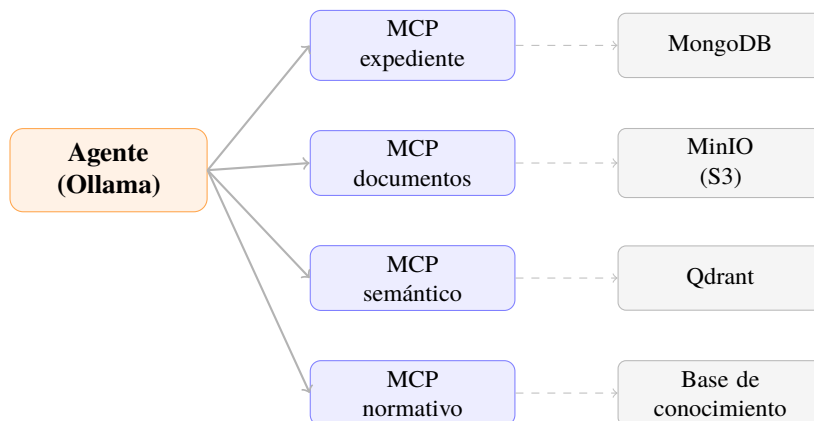


Figura 5.5 Arquitectura del asistente agéntico: el agente (Ollama) orquesta cuatro servidores MCP, cada uno conectado a su fuente de datos..

El modelo se ejecuta localmente con *Ollama* Ollama Team, 2024, lo que garantiza que los datos del expediente no salen de la infraestructura propia. Cuando el técnico pregunta algo como “¿qué documentos le faltan a este expediente para cerrarlo?”, el agente invoca el MCP de expediente para leer qué hay cargado, consulta los criterios de completitud en el MCP normativo y devuelve una respuesta concreta con las referencias exactas. Si la información no es suficiente para responder con seguridad, el sistema lo indica en lugar de inventar. Esa transparencia es lo que hace defendible el uso de IA en un contexto con implicaciones administrativas.

5.8 Frontend y Experiencia de Usuario (Vue.js)

La capa de presentación es la parte de la plataforma con la que trabaja el usuario. En una herramienta técnica de este tipo, la interfaz no debe centrarse en efectos visuales, sino en reducir pasos innecesarios: localizar un expediente, revisar documentos, consultar evidencias y editar el informe sin cambiar continuamente de aplicación.

El frontend se ha construido con **Vue.js 3** You y V. Contributors, 2024, aprovechando su enfoque por componentes. La interfaz se divide en bloques funcionales: ficha del expediente, visor de documentos y fotografías, panel de evidencias y editor del borrador del informe.

5.8.1 Gestión de estado con Pinia

Para evitar que cada componente tenga que pedir los mismos datos una y otra vez, se utiliza *Pinia* Morote y P. Contributors, 2024 como almacén global de estado. Dos *stores* concentran la información principal:

- **useAuthStore:** mantiene la sesión del usuario, gestiona el token de acceso y detecta errores de autenticación.
- **useExpedienteStore:** guarda el expediente activo y lo comparte con las vistas que lo necesitan, evitando pasar datos manualmente entre muchos componentes.

Además, se incorporó **autosave con debounce**. Esto significa que los cambios del borrador no se envían al servidor en cada pulsación, sino cuando el usuario deja de escribir durante un breve

intervalo. Así se reducen peticiones innecesarias y se protege el avance del trabajo ante cierres de navegador o cortes de conexión.

5.8.2 Editor de informes inteligente

El editor de informes se ha diseñado para que el técnico pueda redactar y comprobar el expediente desde una misma pantalla. Sus elementos principales son:

- **Visor de documentos:** permite revisar PDFs y evidencias sin salir de la plataforma.
- **Campos conectados al expediente:** los datos normalizados se leen y actualizan desde MongoDB, manteniendo coherencia entre formulario e informe.
- **Sugerencias asistidas:** los valores extraídos por OCR o por el agente de IA se muestran como propuestas que el técnico debe confirmar antes de incorporarlas.

Con este enfoque, ningún dato aparece en el informe sin revisión. Cada anotación técnica puede relacionarse con su documento o evidencia de origen, lo que mejora la trazabilidad y facilita justificar el expediente ante una revisión posterior.

5.9 Flujo de Trabajo Operativo con Asistencia de IA

La plataforma *ETMOV* no se limita a ser un repositorio de documentos: es un entorno de trabajo colaborativo donde los roles Técnico, Supervisor y Administrador coordinan el ciclo de vida estructurado del expediente. Cada fase de ese ciclo incorpora asistencia de IA para reducir la carga administrativa, pero preservando en todo momento la decisión final en manos del rol responsable en cada fase.

5.9.1 Gestión de empleados y roles

El sistema implementa un modelo de **control de acceso basado en roles (RBAC)** Sandhu et al., 1996 con tres perfiles diferenciados: **Técnico**, **Supervisor** y **Administrador**. Estos roles reflejan la cadena operativa del expediente: el Técnico prepara y documenta, el Supervisor revisa y valida técnicamente, y el Administrador gestiona la plataforma y realiza el cierre definitivo.

Técnico Es el perfil encargado de la elaboración inicial del expediente. Puede crear o editar los expedientes que tenga asignados, rellenar la hoja de entrada, cargar documentación y fotografías, consultar la asistencia de IA y validar o rechazar las sugerencias generadas por el sistema. No puede cerrar ni firmar expedientes.

Supervisor Es el perfil encargado de revisar los expedientes elaborados por los técnicos. Puede comprobar la coherencia entre datos, evidencias y justificaciones técnicas, devolver el expediente al Técnico con observaciones o elevarlo al Administrador cuando la revisión sea favorable.

Administrador Es el perfil con permisos completos sobre la plataforma. Puede gestionar usuarios, plantillas, parámetros del sistema y modelos de IA. Además, realiza la validación final, firma y cierra definitivamente los expedientes revisados.

Esta separación garantiza que el expediente avance siguiendo una cadena de revisión clara, y que el historial de auditoría de cada acción (usuario, fecha, campo modificado) quede registrado en MongoDB.

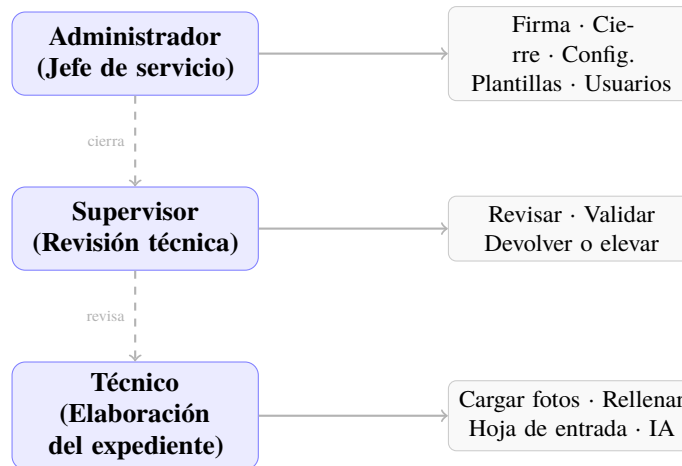


Figura 5.6 Modelo RBAC de la plataforma: tres roles con permisos diferenciados y cadena de revisión Técnico → Supervisor → Administrador..

5.9.2 Apertura del expediente: la hoja de trabajo de entrada

El proceso comienza cuando un vehículo entra en el servicio técnico. El técnico asignado abre un nuevo expediente en la plataforma y cumplimenta la **hoja de trabajo de entrada**, que actúa como ficha de recepción estructurada. Los campos mínimos obligatorios son:

- **Número de bastidor (VIN):** identificador único del vehículo de 17 caracteres. Es el dato pivote sobre el que se construye todo el enriquecimiento posterior.
- **Tipo de procedimiento:** seleccionado de un catálogo definido por el administrador (Motocicleta, Turismo, Vehículo industrial, etc.), que determina qué plantilla de informe se aplicará.
- **Referencia interna:** código alfanumérico del servicio para identificar el expediente en el sistema de gestión propio.
- **Técnico asignado:** el Supervisor o el Administrador asigna el expediente a un Técnico concreto, que desde ese momento lo verá en su lista de trabajo pendiente.

Con estos datos mínimos el expediente queda en estado BORRADOR y el técnico puede comenzar a trabajar sobre él de forma inmediata, sabiendo que el sistema guardará automáticamente cualquier cambio mediante el mecanismo de *autosave con debounce* descrito en la sección anterior.

5.9.3 Enriquecimiento automático de datos

Una vez introducido el VIN, la plataforma lanza en segundo plano una cadena de consultas a fuentes externas para enriquecer automáticamente la ficha técnica del vehículo. Este enriquecimiento evita que el técnico tenga que buscar manualmente datos que ya están disponibles en registros públicos o semipúblicos, reduciendo el tiempo de apertura de un expediente y eliminando errores de transcripción.

Las fuentes consultadas siguen una estrategia de prioridad decreciente:

1. **Base de datos interna de expedientes anteriores (Qdrant):** si el servicio ya ha tramitado vehículos con el mismo VIN o del mismo modelo/año, el motor vectorial recupera la ficha técnica de aquel expediente y la propone como punto de partida. Esto es especialmente útil para revisiones periódicas del mismo vehículo.

2. **Registro de homologaciones europeas (e-HRAS / WVTA):** mediante la consulta al identificador de homologación embebido en el VIN, el sistema obtiene la ficha técnica de tipo del vehículo: categoría, masa máxima autorizada, dimensiones, potencia, normativa de emisiones y equipamientos de serie.
3. **Fuentes abiertas enriquecidas mediante el agente IA:** para campos que no están disponibles en bases de datos estructuradas (particularidades de un modelo de serie limitada, historial de reformas homologadas para ese tipo, etc.), el agente MCP puede lanzar consultas sobre la base de conocimiento indexada del servicio y sobre documentación técnica de fabricantes.

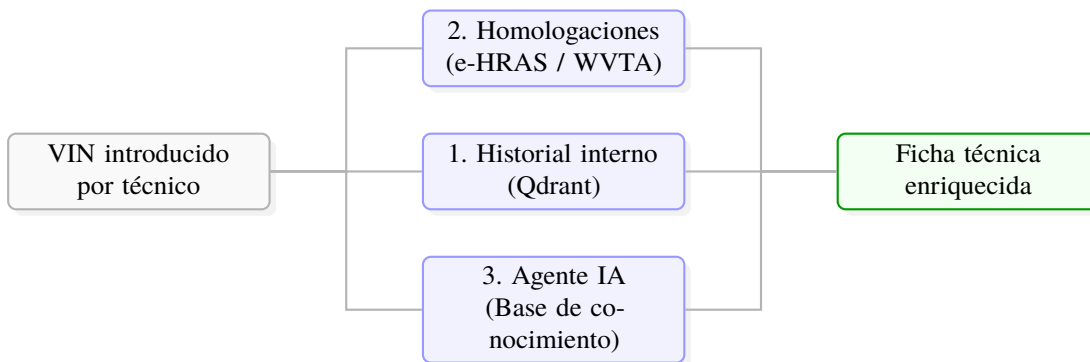


Figura 5.7 Cadena de enriquecimiento automático de datos a partir del VIN: tres fuentes con prioridad decreciente convergen en la ficha técnica del expediente..

Todos los campos rellenados automáticamente se presentan al técnico con una indicación visual de su origen y una confianza estimada. El técnico puede aceptar el valor, modificarlo o rechazarlo. Ningún dato enriquecido se persiste sin confirmación explícita, garantizando que la responsabilidad de la información que aparece en el informe final recae siempre en la persona.

5.9.4 Diagnóstico inicial asistido por fotografías

Paralelamente al enriquecimiento de datos, el Técnico fotografía el vehículo siguiendo una secuencia estandarizada: frontal, trasera, laterales, interior, placa de identificación y zonas con reformas o anomalías. Las imágenes se suben a la plataforma y se almacenan en MinIO bajo el prefijo del expediente.

Una vez cargadas, el servicio de IA en Python/FastAPI procesa cada imagen con un modelo de visión Radford et al., 2021 para extraer información relevante:

- **Lectura de matrícula y número de bastidor:** mediante OCR orientado a placas y etiquetas de chasis, el sistema verifica que la matrícula fotografiada coincide con la del expediente abierto. Si hay discrepancia, lanza una alerta inmediata al técnico.
- **Detección de reformas visibles:** el modelo de visión identifica elementos que difieren del equipamiento de serie conocido (spoilers, llantas no originales, modificaciones de suspensión perceptibles, etc.) y los propone como puntos de atención en la hoja de trabajo.
- **Evaluación del estado general:** se genera una descripción textual del estado aparente del vehículo que el técnico puede incluir o descartar en el informe.

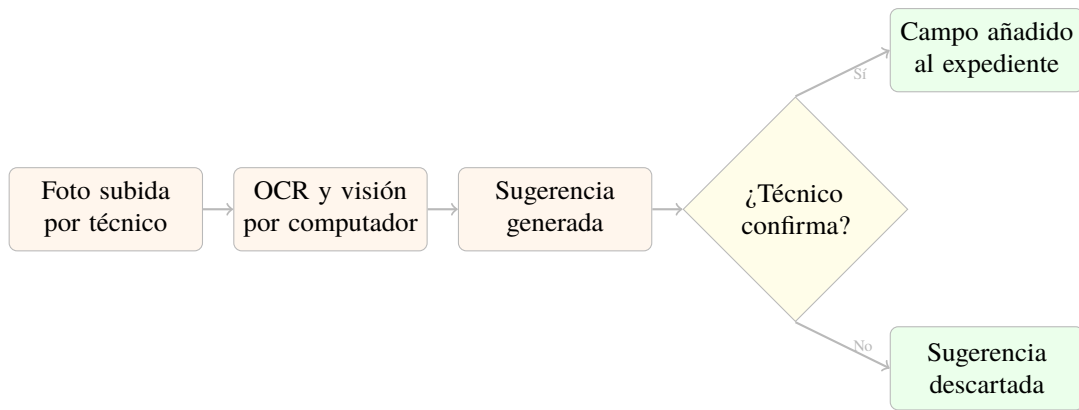


Figura 5.8 Flujo de diagnóstico asistido por fotografía: la IA sugiere, el técnico decide..

Este enfoque *human-in-the-loop* es especialmente relevante en el contexto de la catalogación de vehículos históricos, donde la valoración final implica criterios subjetivos —el nivel de autenticidad, el estado de conservación, la coherencia de las modificaciones con la época del vehículo— que un sistema automatizado no puede determinar de forma inequívoca.

5.9.5 Generación del borrador del informe

Con la ficha técnica enriquecida y las observaciones fotográficas confirmadas, el técnico solicita la **generación del borrador**. El agente de IA ejecuta el siguiente proceso:

1. **Recuperación de precedentes:** Qdrant busca los cinco expedientes más similares al actual en el espacio vectorial de embeddings, considerando tipo de vehículo, categoría de reformas y contexto normativo aplicable.
2. **Síntesis del contexto:** los fragmentos más relevantes de esos expedientes —las secciones de justificación técnica y los apartados de observaciones— se inyectan como contexto en el *prompt* del modelo de lenguaje local (Ollama).
3. **Generación con temperatura cero:** el modelo genera el borrador con `temperature=0.0`, priorizando la reproducibilidad y la fidelidad al lenguaje técnico de los precedentes sobre la creatividad. Esto reduce el riesgo de alucinaciones y mantiene la coherencia terminológica del servicio.
4. **Presentación al técnico:** el borrador se carga en el editor de informes con todos sus campos marcados como *pendiente de revisión*. El técnico debe confirmar o modificar cada sección antes de poder elevar el expediente al estado EN_REVISIÓN.

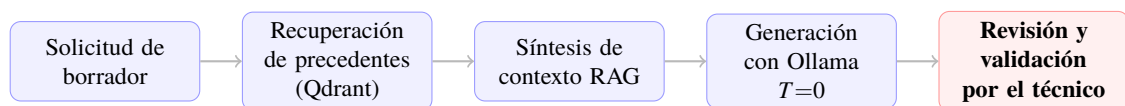


Figura 5.9 Pipeline de generación del borrador: el modelo de IA construye una propuesta basada en precedentes, que el técnico valida campo a campo..

5.9.6 Ciclo de revisión, aprobación y cierre

Una vez que el Técnico ha validado el borrador, el expediente transita al estado EN_REVISIÓN. El Supervisor recibe una notificación y accede al expediente para revisar:

- La coherencia entre los datos de la ficha técnica y las fotografías cargadas.
- La corrección de las justificaciones técnicas y su alineación con la normativa vigente.
- La completitud documental (certificados de taller, fichas de homologación, fotografías de detalle requeridas).

Si el Supervisor detecta alguna deficiencia, devuelve el expediente al Técnico con anotaciones concretas. Si la revisión es satisfactoria, eleva el estado a REVISADO y lo traslada al Administrador para la firma definitiva. El Administrador realiza la comprobación final y cierra el expediente, que pasa a estado FIRMADO: a partir de ese momento ningún campo puede modificarse y el informe queda disponible para su descarga en PDF desde MinIO.

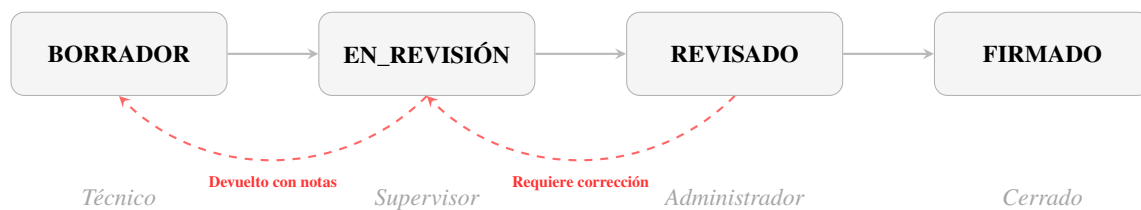


Figura 5.10 Estados del expediente y roles responsables de cada transición. Las flechas discontinuas representan el flujo de devolución y corrección interna..

Este flujo garantiza que cada expediente pasa por revisión técnica del Supervisor y validación final del Administrador antes de quedar firmado, y que el historial de cada transición —con el usuario y el timestamp correspondiente— queda registrado de forma permanente. La IA actúa exclusivamente como herramienta de apoyo en las fases de redacción y enriquecimiento; la validación y la firma son siempre responsabilidad de una persona.

5.10 DevOps y Ciclo de Entrega Continua

Aunque el proyecto se desarrolla dentro del alcance de un TFG, se han aplicado prácticas habituales de ingeniería de software profesional. La **Integración y Entrega Continua (CI/CD)** permite que cada cambio en el código siga un proceso repetible: se valida, se construye, se empaqueta y se despliega con menos intervención manual.

Para organizar este flujo, se utilizan *runners* de **GitLab** GitLab Inc., 2024; Kim et al., 2016:

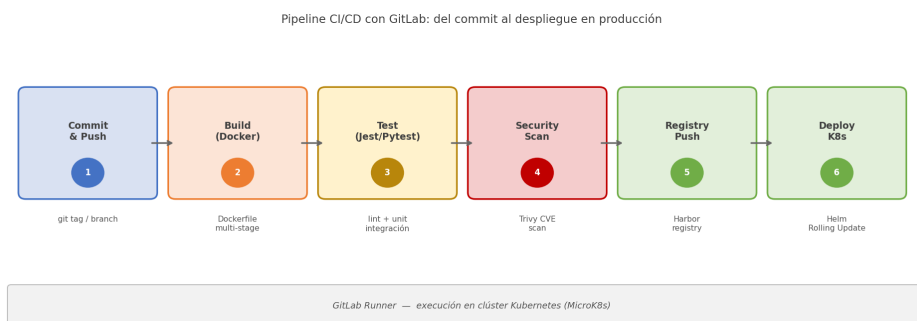


Figura 5.11 Secuencia de validación, contención y empaquetado del orquestador GitLab CI/CD..

1. **Compilación y generación (Build):** construcción del frontend Vue y creación de las imágenes Docker del backend y los servicios Python.

2. **Testing:** ejecución de validaciones automáticas antes de permitir el despliegue.
3. **Análisis de seguridad (Security Scan):** revisión de dependencias e imágenes para detectar *Common Vulnerabilities and Exposures* (CVEs).
4. **Publicación de imágenes (Registry Push):** subida de las imágenes generadas al registro privado.
5. **Despliegue (Deploy):** actualización del clúster *K8s* mediante manifiestos Helm Helm Authors, 2024 y estrategias de *Rolling Updates*.

El principal beneficio de este enfoque es la trazabilidad: se sabe qué versión del código se ha desplegado, qué pruebas ha superado y qué imagen de contenedor está ejecutándose en producción.

5.11 Pruebas y Validación

La validación de la plataforma se centra en comprobar que los componentes principales funcionan de forma coherente. Las pruebas se orientan a los puntos más sensibles del sistema: autenticación mediante JWT, operaciones críticas sobre la base de datos y subida de documentos o imágenes.

5.11.1 Tests unitarios y de integración

La estructura modular permite probar el sistema en dos niveles complementarios:

- **Backend:** pruebas sobre rutas HTTP, autenticación, validación de datos y escritura en MongoDB.
- **Frontend:** comprobación de componentes, estados de carga, permisos de usuario y bloqueo de acciones cuando el expediente no permite modificaciones.

5.11.2 Pruebas de carga

La respuesta del sistema ante carga concurrente se ha evaluado con **k6** Grafana Labs, 2024. Las pruebas simulan varios usuarios realizando peticiones al mismo tiempo, especialmente en operaciones de subida de anexos, que son las que más pueden afectar al rendimiento.

1. Medir la degradación del tiempo de respuesta cuando aumenta el número de peticiones simultáneas.
2. Comprobar que el servicio de procesamiento en Python/FastAPI puede absorber tareas pesadas sin bloquear la API principal.

Los resultados permiten comprobar que separar la API principal del servicio de procesamiento ayuda a mantener una experiencia estable para el usuario. Las tareas pesadas, como OCR o generación de *embeddings*, pueden ejecutarse en segundo plano sin bloquear el flujo normal de consulta y edición de expedientes.

6 Conclusiones

Este Trabajo de Fin de Grado ha partido de un problema muy concreto del día a día: la forma en que ETMOV gestiona los expedientes de vehículos históricos, apoyándose en correos, carpetas compartidas y hojas de cálculo. A lo largo del proyecto se ha ido comprobando que el principal cuello de botella no estaba tanto en la parte técnica de la inspección, sino en todo lo que rodea al expediente: recopilar documentos, localizar la versión correcta, justificar cada decisión y conseguir que varias personas trabajen sobre el mismo caso sin estorbarse. La plataforma desarrollada intenta dar una respuesta razonable a ese escenario, organizando el expediente en torno a un flujo de trabajo claro y dejando que sean las herramientas las que se adapten al procedimiento, y no al revés.

El resultado es un sistema que estructura el trabajo en cuatro grandes pasos: entrada de datos y enriquecimiento automático, generación de un borrador asistido, revisión interna y cierre del expediente. En la práctica, esto se traduce en que el técnico ya no tiene que empezar siempre desde una plantilla en blanco ni buscar correos antiguos para saber qué se acordó en una revisión anterior. El módulo de visión ayuda a sacar partido a las fotografías, el asistente basado en RAG recupera antecedentes y redacciones previas y el control de versiones evita perder el hilo cuando el informe pasa por varias manos. Todo ello se ha construido sobre una arquitectura modular, desplegada en la infraestructura propia de ETMOV, con servicios separados para datos, ficheros, búsquedas semánticas y procesamiento de documentos.

Más allá de la tecnología concreta elegida, la principal aportación del trabajo es haber demostrado que es posible digitalizar casi todo el recorrido del expediente sin sacar al técnico de su terreno ni sustituir su criterio. El sistema propone, ordena y recuerda, pero la última palabra sigue siendo del ingeniero que firma el informe. Este equilibrio entre automatización y control humano ha guiado muchas de las decisiones de diseño: desde la elección de modelos de IA que se ejecutan en local hasta la forma de registrar la trazabilidad de los cambios y de archivar la versión final del expediente.

El proyecto, aun así, deja abierta una lista amplia de mejoras. Queda margen para integrar mecanismos de firma electrónica cualificada, conectar de forma directa con bases de datos externas de homologación, llevar parte del flujo de trabajo a una aplicación móvil pensada para la inspección en campo o entrenar modelos de visión más específicos para vehículos históricos. También sería interesante explotar los datos acumulados para ofrecer cuadros de mando sobre tiempos de tramitación, carga de trabajo o tipos de expediente más frecuentes. Todas estas líneas encajan con la arquitectura ya desplegada, por lo que pueden abordarse de forma incremental, sin necesidad de replantear el sistema desde cero.

Apéndice A

Interfaz de usuario

En este apéndice se documenta la interfaz de usuario de la plataforma *ETMOV*, accesible en <https://etmov.famluro.es>. La aplicación web ha sido desarrollada con *Vue.js 3* y *Tailwind CSS*, siguiendo un diseño limpio y funcional orientado al uso profesional por parte de los técnicos del servicio.

A.1 Pantalla de acceso

La pantalla de acceso a la plataforma muestra el logotipo de *ETMOV — Estudios Técnicos de Movilidad* junto al logo de la Universidad de Sevilla. El formulario permite el acceso mediante credenciales propias (email y contraseña) o a través de *Google OAuth*, facilitando la incorporación de nuevos técnicos sin necesidad de crear contraseñas adicionales.

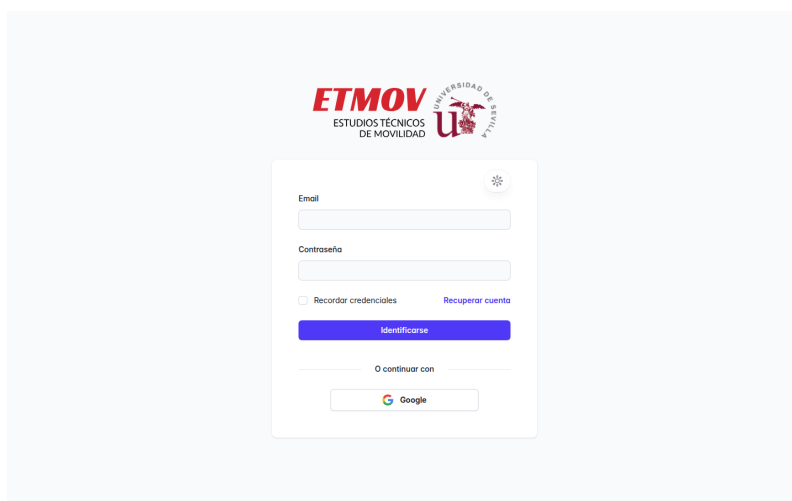


Figura A.1 Pantalla de acceso a la plataforma ETMOV con autenticación local y OAuth de Google..

A.2 Panel de resumen general

Tras autenticarse, el usuario accede al panel de resumen general (*dashboard*). Esta vista condensa los aspectos críticos a tratar: número de procedimientos activos, pendientes de revisión y pendientes de ser firmados.

La barra de navegación lateral agrupa las secciones principales: *Resumen general*, *Procedimientos*, *Ficheros* y la sección de administración con *Ajustes*. En la parte inferior se muestra la versión de la plataforma y el acceso al perfil del usuario autenticado.

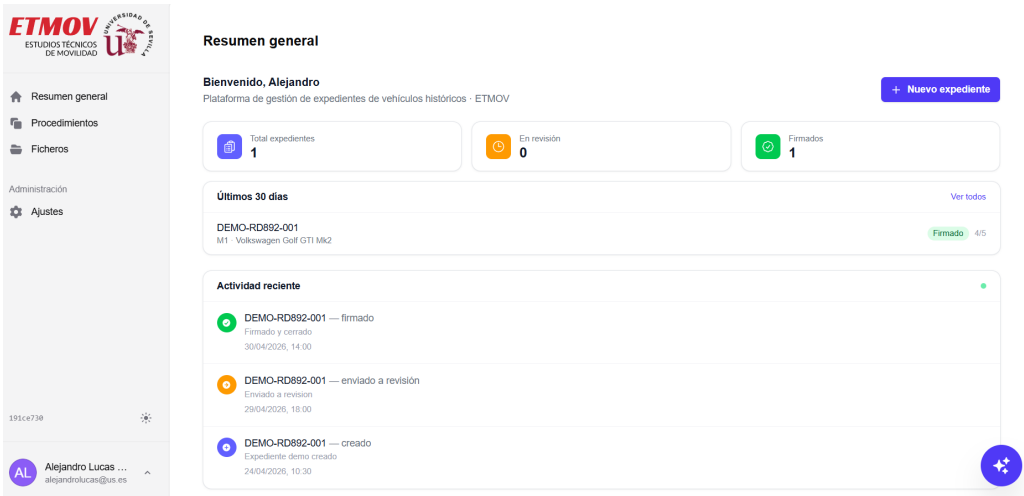


Figura A.2 Panel de resumen general (*dashboard*) con indicadores de actividad y registro de eventos recientes..

En todo momento dispondremos de la posibilidad de consultarle a nuestro asistente por dudas que se puedan originar. Para ello deberemos pulsar sobre la burbuja flotante situada en el pie de pagina en la posición derecha.

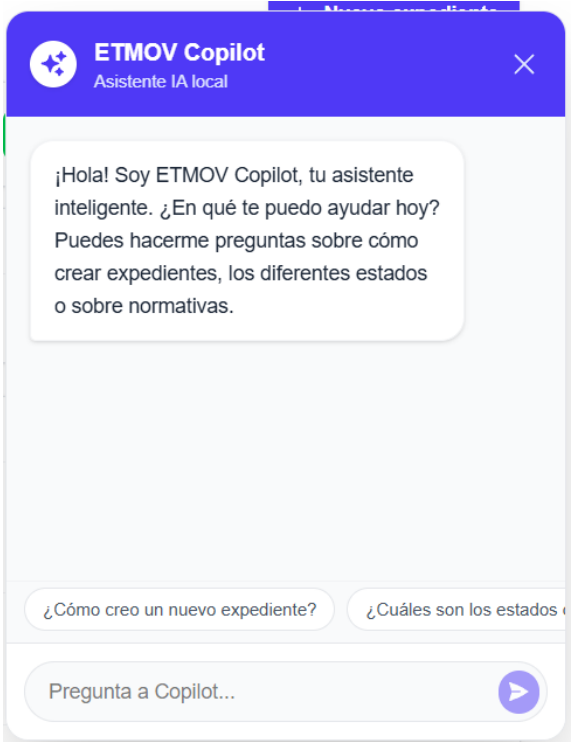


Figura A.3 Panel de resumen general (*dashboard*) con indicadores de actividad y registro de eventos recientes..

A.3 Listado de procedimientos

La sección de *Procedimientos* muestra en formato un historial de todos los expedientes técnicos registrados en el sistema. Cada fila presenta la fecha de creación, la referencia del expediente, el tipo de vehículo, la plantilla aplicada, el estado de tramitación y las opciones disponibles. Esta vista facilita la localización rápida de cualquier expediente y sirve como punto de entrada para la consulta o modificación de sus datos. Además dispone de un buscador para localizar ágilmente los expedientes creados.

FECHA	REFERENCIA	TIPO	PROCEDIMIENTO	VIN / MATRÍCULA	TÉCNICO	ESTADO	ACCIONES
04/05/2026	DEMO-RD892-001	M1	Grupo B - turismos/furgonetas M1/N1	WWZZZ19ZGW000001	Aleja...	Firmado	Abrir Eliminar

1 expediente

Figura A.4 Vista de listado de procedimientos con columnas de referencia, tipo, plantilla, estado y opciones..

En esta sección se permite la creación de un nuevo expediente mediante el botón denominado «+ Nuevo Expediente». Una vez iniciado el proceso, la información se estructura en tres bloques principales: en primer lugar, los datos iniciales destinados a la identificación del expediente; en segundo lugar, los datos técnicos necesarios para su correcta elaboración; y, finalmente, el reportaje fotográfico junto con la documentación anexa requerida para la emisión del informe.

Figura A.5 Vista de la creación de un nuevo expediente..

Procedimiento RD 892/2024

Grupo B - turismos/furgonetas M1/N1 · Datos · Ficha Reducida

1% completado

Campos 1/62

Docs 0/6

Fotos 0/14

Hoja de datos normativa

Guardar

Datos del expediente

Identificación

Fabricacion y procedencia

Masas y dimensiones

Constitucion ge

Solicitante

Inspector

05/05/2026

Fecha de la visita

05/05/2026

Fecha de emision

Lugar de inspeccion

Estado del vehiculo

Ubicación

< Anterior

1 / 8

Siguiente >

Figura A.6 Paso 2 en la creación de un nuevo expediente, enriquecimiento del informe con datos asociados. Datos de formulario.

Documentación aportada

Ficha tecnica o ficha de características original

Permiso de circulacion o documentacion original del pais

DNI/NIF del titular o solicitante

Certificado DGT cuando proceda

Certificado del club/laboratorio acreditado

Bibliografia o documentacion tecnica complementaria

Reportaje fotográfico de los documentos anexos

Vista frontal

Vista trasera

Lateral derecho

Figura A.7 Paso 2 en la creación de un nuevo expediente, enriquecimiento del informe con datos asociados. Ficheros adjuntos y reportaje fotográfico.

Informe técnico Generando... [Editar](#)

AGENTE IA deepseek-v4-flash · visión qwen3-vl:235b

-
- Iniciando agente de borrador IA
- Configuración cargada
Modelo de texto: deepseek-v4-flash. Modelo de visión: qwen3-vl:235b.
-
- Analizando fotografías
Llamando al modelo de visión qwen3-vl:235b para detectar reformas y defectos.
-
- Resultado del análisis visual
No hay fotografías adjuntas en este expediente.
-
- Generando borrador
Llamando al modelo deepseek-v4-flash de Ollama con el contexto del expediente.

Sin informe técnico. Usa el botón "Generar borrador IA" para que el sistema proponga un contenido basado en expedientes anteriores.

Notas del técnico [Editar](#)

Figura A.8 Paso 2 en la creación de un nuevo expediente, enriquecimiento del informe con datos asociados. Generación de borrador con IA con los elementos proporcionados.

 **Enviar a revisión**
El expediente quedará pendiente de firma del supervisor

Referencia	Tipo
asdfasf	M1
Vehículo	Compleitud
	1%

 El expediente está poco completado. Considera revisar los datos antes de enviarlo.

Nota para el supervisor (opcional)

Cancelar Enviar a revisión

Figura A.9 Paso 2 en la creación de un nuevo expediente, enriquecimiento del informe con datos asociados. Envío a revisión por parte de un supervisor.

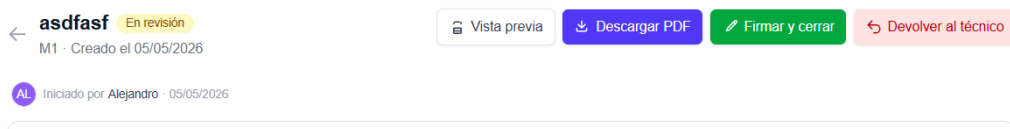


Figura A.10 Paso 3 en la creación de un nuevo expediente. Devolución del informe al técnico para re-elaboración o firma digital para su cierre por parte de un supervisor.

This image shows a modal dialog box with a white background and rounded corners. At the top left is a red circle with a white left arrow icon. To its right is the title 'Devolver al técnico' in bold, followed by the text 'El expediente se devuelve con anotaciones para que el técnico lo corrija.' Below this is a table with two columns: 'Expediente' and 'Estado actual'. The first row has the values 'asdfasf' and 'En revisión'. Below the table is a label 'Indica qué debe corregir el técnico *' followed by a text input field with the placeholder text 'Indica qué debe corregir el técnico'. At the bottom right are two buttons: 'Cancelar' and 'Confirmar' (which is red with white text).

Figura A.11 Paso 3 en la creación de un nuevo expediente. Devolucion de informe con nota anexa.

 **Revisión y firma del expediente**
asdfasf · M1

ESTADO DEL EXPEDIENTE

1%
Compleitud

1/62
Campos

0/20
Docs + Fotos

VIN / N.º bastidor
asdfa

Matrícula
—

Marca
—

Modelo
—

Año de fabricación
—

Color
—

 **Firma digital con AutoFirma**

Firmante
Alejandro
alejandrolucas@us.es

Fecha
05/05/2026

Se generará el PDF del expediente y AutoFirma solicitará el certificado para incrustar una firma PAdES en el documento.

Observaciones de la firma (opcional)

Conforme con el contenido del expediente...

 Devolver al técnico

Cancelar

Firmar con AutoFirma

Figura A.12 Paso 3 en la creación de un nuevo expediente. Firma del documento digital.

A.4 Explorador de ficheros

La sección *Ficheros* actúa como un repositorio centralizado desde el que es posible consultar todos los documentos y recursos multimedia que se han incorporado como anexos a los distintos procedimientos documentales. Desde esta vista, el usuario puede localizar de manera rápida los ficheros asociados a cada expediente, empleando filtros por origen y tipo de documento, así como un cuadro de búsqueda específico. Esta funcionalidad facilita la organización y recuperación de la documentación anexa, mejora la trazabilidad de los expedientes y contribuye a reducir el tiempo dedicado a la gestión manual de archivos dentro del flujo de trabajo.

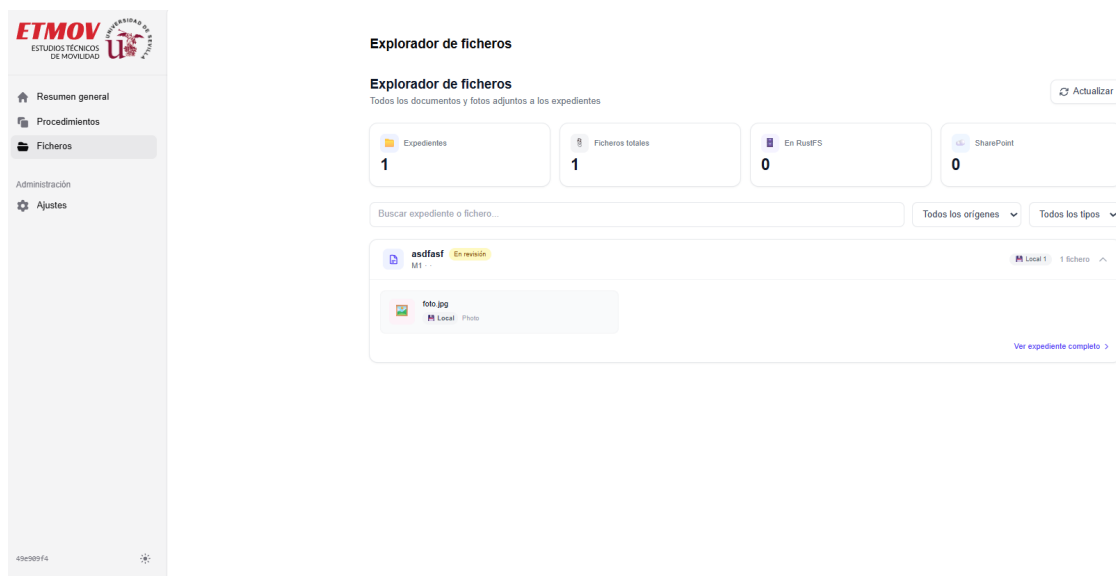


Figura A.13 Explorador de ficheros enfocado a los expedientes existentes.

A.5 Perfil del usuario

La sección de perfil permite al técnico consultar y modificar sus datos de cuenta. En la parte superior se muestra un resumen con el nombre completo, nivel de permisos asignado y la fecha y hora del último acceso al sistema. La línea de tiempo de *últimas conexiones* registra los tres accesos más recientes con su dirección IP, ofreciendo una traza de auditoría visible para el propio usuario. El formulario inferior permite actualizar datos personales y cambiar la fotografía de perfil.

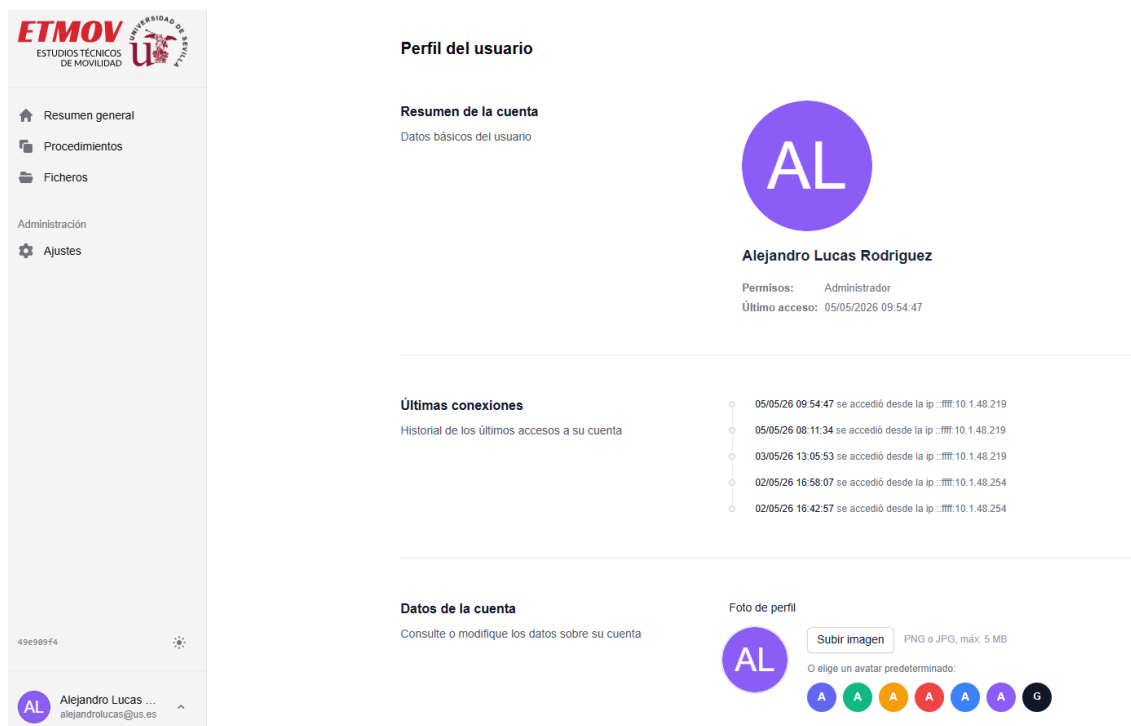


Figura A.14 Pantalla de perfil del usuario con historial de accesos y formulario de edición de datos..

A.6 Ajustes generales de la plataforma

El panel de ajustes generales está organizado en cuatro pestañas: *Plataforma*, *Plantillas*, *Usuarios* e *IA*. La pestaña *Plataforma*, visible en la Figura A.15, permite configurar el servidor de correo SMTP para el sistema de notificaciones: servidor, puerto, credenciales y activación del servicio. Desde aquí también se puede enviar un correo de prueba para verificar la configuración antes de activarla en producción. Además, desde esta misma sección es posible establecer diversos parámetros globales de la plataforma.

La pestaña *IA* permite seleccionar y configurar los modelos de inteligencia artificial utilizados por la aplicación, definiendo tanto el proveedor como el modelo concreto y los parámetros asociados a su comportamiento (por ejemplo, temperatura o número máximo de tokens). De este modo, el sistema puede adaptarse a distintos escenarios de uso, ya sea priorizando la velocidad de respuesta o la precisión en la generación de contenido. Esta configuración centralizada facilita la actualización y el mantenimiento de los modelos sin necesidad de modificar el código de la aplicación.

Por otro lado, el panel de ajustes incluye la configuración de los servicios de persistencia de datos empleados por la plataforma. Desde aquí es posible definir los parámetros de conexión a la base de datos documental en MongoDB, así como el servicio de vectorización y búsqueda semántica basado en Qdrant. Esta sección permite establecer direcciones de los servidores, puertos, credenciales y opciones avanzadas, garantizando una integración coherente entre los distintos componentes de la arquitectura.

Finalmente, el sistema ofrece opciones para configurar el almacenamiento de archivos en servicios compatibles con S3, donde se definen los *buckets*, las claves de acceso y las regiones asociadas. Además, el panel muestra el estado de los servicios principales (base de datos, motor vectorial, almacenamiento S3 y servicios de IA), indicando si la conexión es correcta o si se ha producido algún error. Esta información de monitorización básica resulta especialmente útil para tareas de administración y para la detección temprana de incidencias en el entorno de producción.

ETMOV
ESTUDIOS TÉCNICOS DE MOVILIDAD

Ajustes generales

Plataforma
Configuración SMTP, notificaciones y estado de servicios

Notificaciones email
Configure los parámetros y opciones para el sistema de alerta y notificaciones

Servidor SMTP
smtp.outlook.es

Puerto SMTP
587

Usuario
alejandrolucas@us.es

Contraseña

☒ **Habilitar servicio**
Al habilitar el servicio todas las notificaciones serán enviadas también por email.

Figura A.15 Panel de ajustes generales: configuración del servidor SMTP para notificaciones por correo electrónico..

A.7 Plataforma. Modelos LLM y Vision

La sección *Ajustes* integra un panel de configuración de inteligencia artificial concebido como el “centro neurálgico” de la capacidad cognitiva de la plataforma. Desde este módulo no solo se seleccionan los modelos de lenguaje (LLM), los modelos de visión para *Intelligent Document Processing* (IDP) y los modelos de *embeddings* empleados en los flujos RAG, sino que se orquesta cómo se combinan entre sí para dar soporte al ciclo completo de análisis de expedientes. Esta arquitectura configurable convierte al sistema en una solución extremadamente versátil, capaz de adaptarse tanto a escenarios de experimentación en laboratorio como a despliegues en producción con altas exigencias de fiabilidad y rendimiento.

La potencia de este panel radica también en la libertad de elección de infraestructura que ofrece: es posible trabajar con instancias locales de Ollama, preservando la soberanía del dato y reduciendo la dependencia de terceros, o bien apoyarse en servicios en la nube para escalar bajo demanda y aprovechar modelos de última generación. A ello se suma la posibilidad de ajustar de forma detallada el comportamiento del asistente (tono, estilo de respuesta, límites de contexto, nivel de iniciativa, etc.), de modo que la IA pueda alinearse con las políticas internas de la organización y con los objetivos específicos del proyecto. En consecuencia, esta sección no se limita a “activar” la IA, sino que la convierte en un componente gobernable, auditable y evolutivo, clave para garantizar que la solución pueda crecer y mejorar sin requerir cambios estructurales en el desarrollo del sistema.

Inteligencia Artificial Ollama

Resumen de la configuración y estado del servicio de IA local y análisis de expedientes.

Configura tu instancia de Ollama. En Docker, si Ollama está en tu equipo, usa <http://host.docker.internal:11434>; fuera de Docker puedes usar <http://localhost:11434>.

URL Ollama

https://ollama.com

API Key

.....

Proveedor

Ollama

Estado de conexión

Ollama online

Modelos activos

Modelo de Asistente Conversacional (Chat)

deepseek-v4-flash

Modelo de Embeddings (RAG)

gpt-oss-20b

Modelo de Visión (IDP)

qwen3-vl-235b

Agente de borrador y homologación

Prompts y contexto usados cuando el sistema genera el borrador y revisa fotografías para localizar reformas o defectos bloqueantes.

Prompt del borrador técnico

Instrucciones para redactar el informe técnico en JSON...

Figura A.16 Configuración de los prompts con los modelos de IA..

A.8 Ajustes. Api y Webhooks

La sección *API y Webhooks* centraliza la configuración de las integraciones con sistemas de terceros, permitiendo que ETMOV se exponga como una plataforma abierta y extensible. A través de la gestión de claves de API, el administrador puede generar y revocar tokens de acceso que habilitan a aplicaciones externas (como CRMs, gestores de talleres o herramientas de analítica) a consumir de forma segura los servicios de la plataforma. De este modo, se controla de manera granular quién puede acceder a la información y qué sistemas están autorizados a interactuar con los expedientes.

Complementariamente, el apartado de *Webhooks* permite definir endpoints de notificación hacia sistemas externos, de manera que ETMOV pueda enviar eventos en tiempo real cuando se produzcan acciones relevantes, como la creación o actualización de un expediente, la emisión de un informe o la detección de un cambio de estado. Esta capacidad abre la puerta a múltiples integraciones: actualización automática de estados en un CRM, disparo de flujos de trabajo en herramientas de orquestación (Zapier, n8n), registro de métricas en plataformas de monitorización o sincronización con sistemas corporativos de gestión documental. En conjunto, esta sección convierte a la aplicación en un nodo más dentro del ecosistema digital de la organización, reduciendo la necesidad de procesos manuales y favoreciendo la automatización extremo a extremo.

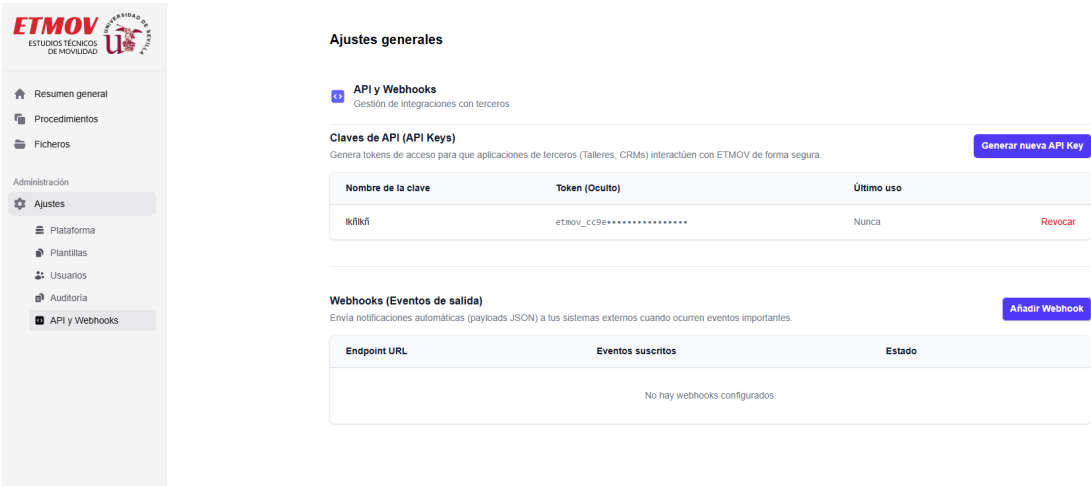


Figura A.17 Generación de API y Webhooks..

A.9 Ajustes. Plantillas

La sección *Plantillas* está concebida como el punto de gestión central de los modelos de informe que utiliza la plataforma. Desde este panel es posible crear, editar y habilitar distintas plantillas asociadas a los grupos de vehículos o tipos de procedimiento, definiendo para cada una de ellas un nombre visible, el fichero Excel base sobre el que se construye el informe y, en su caso, la ficha reducida correspondiente. De esta forma, se separa la lógica de presentación del informe de la parte de implementación, lo que facilita actualizar el formato documental sin alterar el resto del sistema.

Además, la interfaz permite definir las secciones que compondrán el informe final seleccionando qué hojas del libro de Excel formarán parte del documento generado (por ejemplo, apartados de resultados, anexos fotográficos o bibliografía). Cada plantilla puede, a su vez, asociarse a un flujo de trabajo específico, determinando los pasos que seguirá el expediente creado a partir de ella. Esto convierte a las plantillas en un elemento clave para estandarizar la generación de informes técnicos y garantizar que todos los expedientes de un mismo tipo sigan una estructura homogénea y un proceso de tramitación coherente.

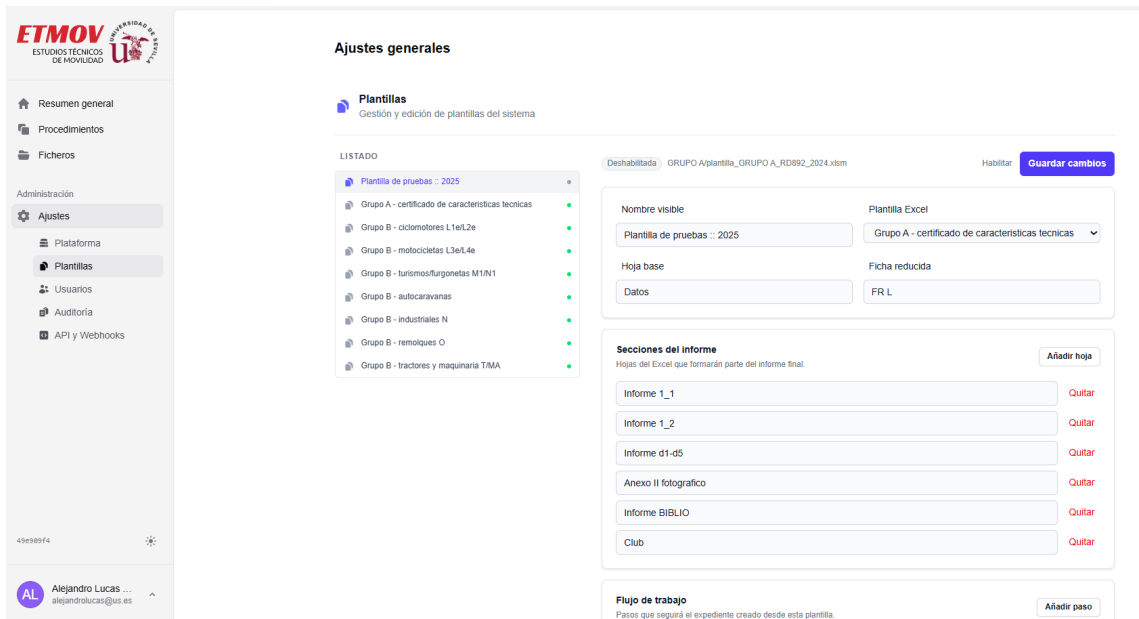


Figura A.18 Plantillas disponible para la creación de expedientes nuevos..

A.10 Ajustes. Auditoría

La sección *Auditoría* registra de forma cronológica todas las operaciones realizadas en la plataforma, almacenando cada acción, petición e interacción junto con la información necesaria para conocer en todo momento qué ha ocurrido, por qué y quién la ha llevado a cabo. De este modo, el sistema proporciona una trazabilidad completa que resulta especialmente útil para tareas de supervisión, detección de incidencias y cumplimiento de requisitos normativos.

Asimismo, la interfaz de auditoría permite filtrar los registros en función de distintos criterios, incluyendo rangos de fecha, tipo de operación o usuario responsable, lo que facilita localizar rápidamente eventos relevantes dentro de grandes volúmenes de información. Además, los datos auditados pueden exportarse a formato CSV para su análisis externo, integración con otras herramientas o elaboración de informes más detallados.

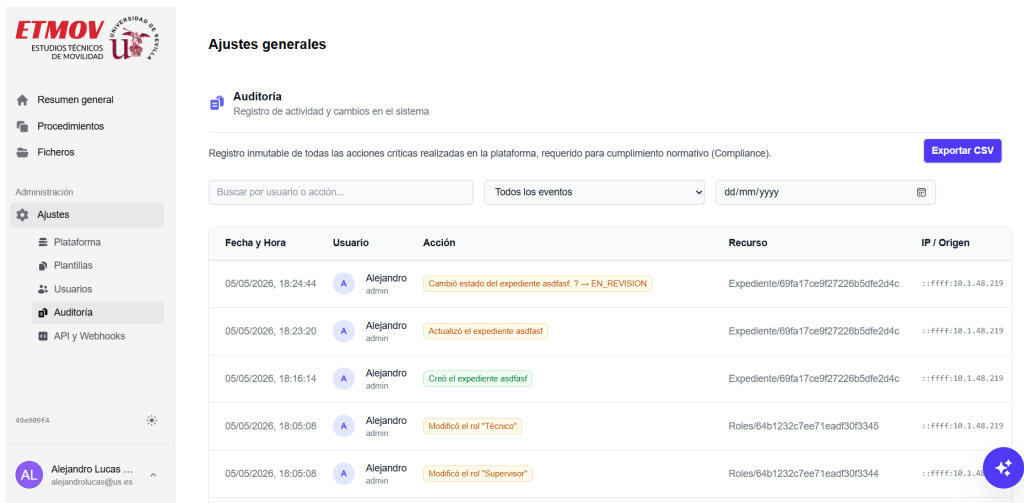


Figura A.19 Historial con todas las acciones que han ocurrido en la web. Auditoría y trazabilidad..

Apéndice B

Fragmentos de código relevantes

En este apéndice se recogen los fragmentos de código más representativos de la plataforma *ETMOV*, seleccionados por su relevancia arquitectónica o por ilustrar patrones de diseño clave empleados en el desarrollo. Los listados están organizados por capa de la aplicación. La nomenclatura de roles utilizada en los fragmentos se mantiene coherente con el flujo operativo de la plataforma: **Técnico**, **Supervisor** y **Administrador**. Cuando aparece el término *usuario* en minúscula se emplea en sentido genérico, no como nombre de rol.

B.1 Backend: modelo de datos de un procedimiento

El Código B.1 muestra el esquema *Mongoose* que define la estructura de un expediente técnico en la base de datos MongoDB. El campo `author` identifica al **Técnico** que crea o prepara el expediente, `reviewed` recoge la revisión del **Supervisor** y `signed` representa la firma o cierre por parte del **Administrador**. El campo `data` de tipo `Object` proporciona flexibilidad para albergar la ficha técnica específica de cada tipo de vehículo sin necesidad de alterar el esquema. La opción `timestamps` añade automáticamente los campos `createdAt` y `updatedAt`, garantizando la trazabilidad temporal de cada expediente.

Código B.1 Esquema Mongoose del modelo Procedures (MongoDB).

```
module.exports = mongoose => {
  var schema = mongoose.Schema(
    {
      reference: { type: String, default: '' },
      type: { type: String, default: '' },
      // Tecnico que crea o prepara el expediente
      author: { type: mongoose.Schema.ObjectId, ref: "users" },
      // Revision tecnica realizada por el Supervisor
      reviewed: {
        type: Object,
        default: { status: false, user: null }
      },
      // Firma y cierre definitivo por el Administrador
      signed: {
        type: Object,
        default: { status: false, user: null }
      },
    },
    { timestamps: true }
  );
```

```

    data: { type: Object, default: '' },
    template: { type: mongoose.Schema.ObjectId, ref: "templates" }
  },
  {
    timestamps: true,
    versionKey: false
  }
);
return schema;
};

```

B.2 Backend: controlador de procedimientos

El Código B.2 presenta las funciones principales del controlador de procedimientos. La función `getProcedure` ilustra el patrón de gestión de errores utilizado en toda la API: validación de parámetros de entrada, consulta a la base de datos con `.lean()` para obtener objetos JavaScript planos (más eficientes que los documentos Mongoose con prototipo) y respuesta con códigos de estado HTTP semánticos.

Código B.2 Controlador de expedientes: funciones `getAllProcedures` y `getProcedure`.

```

const Procedures = require("../models").db.models.Procedures;

async function getAllProcedures(req, res) {
  try {
    const procedures = await Procedures.find({}).lean();
    if (!procedures.length) {
      return res.status(200).send({
        message: 'No se han encontrado procedimientos',
        code: 3000
      });
    }
    return res.status(200).send({ data: procedures, code: 2001 });
  } catch (err) {
    console.error("getAllProcedures", err);
    return res.status(500).send({
      message: 'Internal server error', code: 3000
    });
  }
}

async function getProcedure(req, res) {
  try {
    const { id } = req.params;
    if (!id) {
      return res.status(400).send({
        message: 'ID del procedimiento no proporcionado',
        code: 3000
      });
    }
  }
}

```

```

    }
    const procedure = await Procedures.findById(id).lean();
    if (!procedure) {
        return res.status(404).send({
            message: 'Procedimiento no encontrado', code: 3000
        });
    }
    return res.status(200).send({ data: procedure, code: 2001 });
} catch (err) {
    console.error("getProcedure", err);
    return res.status(500).send({
        message: 'Error interno del servidor', code: 3000
    });
}
}
}

module.exports = { getAllProcedures, getProcedure };

```

B.3 Backend: sistema de autenticación JWT

El Código B.3 muestra el núcleo de la función de autenticación. El flujo verifica las credenciales con `bcrypt.compare`, genera un token JWT con caducidad de siete días y actualiza el documento del usuario en MongoDB mediante `$push` con `$slice` para mantener un historial de las últimas diez sesiones activas y diez actividades de acceso, respectivamente. El campo `role`, cargado mediante `populate`, debe corresponderse con uno de los tres roles funcionales definidos para la plataforma: **Técnico**, **Supervisor** o **Administrador**.

Código B.3 Función de autenticación: verificación de credenciales y emisión de JWT.

```

async function login(req, res) {
    try {
        const findUser = await User.findOne({ email: req?.body?.email });
        let passwordIsValid = false;
        if (findUser) {
            passwordIsValid = await bcrypt.compare(
                req?.body?.password, findUser.password
            );
        }
        if (!findUser || !passwordIsValid) {
            return res.status(401).send({
                message: 'Email o password incorrecto', code: 3000
            });
        }
        // Limpiar tokens expirados del usuario
        for (const listedToken of findUser.token || []) {
            try {
                await jwt.verify(listedToken, appConfig.secret);
            } catch {
                await User.updateOne(

```

```

        { _id: findUser._id },
        { $pull: { token: listedToken } }
    );
}
}
// Emitir nuevo token con TTL de 7 dias. El rol debe ser
// TECNICO, SUPERVISOR o ADMINISTRADOR.
const token = jwt.sign(
    {
        _id: findUser._id,
        email: findUser.email,
        name: findUser.name,
        role: findUser.role
    },
    appConfig.secret,
    { expiresIn: '7d' }
);
// Persistir token y registrar actividad
const userData = await User.findOneAndUpdate(
    { _id: findUser._id },
    {
        $push: {
            token: { $each: [token], $slice: -10 },
            loginActivities: { $each: [{ loginDate: new Date(),
                                      loginIP: req.ip }], $slice: -10 }
        }
    },
    { new: true }
).populate('role');

return res.status(200).send({ user: userData, token });
} catch (err) {
    console.error("auth.login", err);
    return res.status(500).send({
        message: 'Error interno del servidor', code: 3000
    });
}
}
}

```

B.4 Frontend: interceptor Axios y gestión de respuestas

El Código B.4 ilustra el patrón de interceptores de Axios empleado en la capa de servicios del frontend. El interceptor de peticiones inyecta automáticamente el token JWT en la cabecera `x-access-token`. El interceptor de respuestas interpreta los códigos internos de la API (2000, 2001, 3000...) para gestionar notificaciones tipo *toast* de forma centralizada, evitando duplicar esta lógica en cada servicio individual.

Código B.4 Interceptores Axios: inyección de token y gestión centralizada de respuestas.

```
const service = axios.create({
  baseURL: process.env.VUE_APP_BACKEND_URI || '/api',
  timeout: 10000,
  headers: { "Content-type": "application/json" }
});

// Interceptor de peticiones: adjunta el token JWT
service.interceptors.request.use(config => {
  const usersStore = useUsersStore();
  if (usersStore.token) {
    config.headers['x-access-token'] = usersStore.token;
  }
  return config;
}, error => Promise.reject(error));

// Interceptor de respuestas: gestiona codigos internos
service.interceptors.response.use(
  response => {
    const { code, message } = response.data;
    if (code === 2000) {
      useLayoutStore().showToastMessage({
        status: response.status, message, style: "success"
      });
    } else if (code === 3000) {
      useLayoutStore().showToastMessage({
        status: response.status, message, style: "error"
      });
    }
    return response.data;
  },
  error => {
    // Redirige al login si el token ha caducado (401)
    if (error.response.status === 401) {
      useUsersStore().v_cleanToken();
    }
    useLayoutStore().showToastMessage({
      status: error.response.status,
      message: "Error en la comunicacion con la API",
      style: "error"
    });
    return Promise.reject(error);
  }
);

export default service;
```

B.5 Frontend: store de autenticación con Pinia

El Código B.5 muestra el *store* de autenticación implementado con *Pinia*. La acción `login` delega en el servicio `UserService`, actualiza el estado reactivo de la aplicación y persiste el token en una cookie mediante `setUserToken`. La acción `v_decodeToken` resuelve el ciclo completo de inicialización de sesión: decodifica el JWT, recupera los datos completos del usuario desde la API y los almacena en el estado global para que cualquier componente pueda acceder a ellos sin realizar peticiones adicionales.

Código B.5 Store Pinia de autenticación: acciones de login y decodificación de token.

```
export const useUsersStore = defineStore('users', {
  state: () => ({
    id: null,
    token: getUserToken(),
    v_userdata: null,
    v_decodedToken: null,
    // Rol funcional: TECNICO, SUPERVISOR o ADMINISTRADOR
    role: null
  }),

  actions: {
    async login(userData) {
      const res = await UserService.login({
        email: userData.email,
        password: userData.password
      });
      if (res.code === 3000) throw new Error('Credenciales incorrectas');
      this.token = res?.data?.token;
      this.v_userdata = res?.data?.user;
      this.role = res?.data?.user?.role || null;
      setUserToken(res?.data?.token);
      return res;
    },

    v_decodeToken() {
      return new Promise(async (resolve, reject) => {
        if (this.v_decodedToken !== null) {
          return resolve(this.v_decodedToken);
        }
        try {
          const tokenToParse = this.token || await getUserToken();
          if (!tokenToParse) return resolve(null);
          const data = parseJwt(tokenToParse);
          this.v_decodedToken = data;
          await this.getUserData(data._id);
          this.role = this.v_userdata?.role || data?.role || null;
          resolve(data);
        } catch (error) {
          reject(error);
        }
      });
    }
  }
});
```

```
    }  
  });  
}  
}  
});
```


Índice de Figuras

2.1	Fases del expediente de catalogación de vehículo histórico	5
2.2	Flujo RAG: la consulta se enriquece con fragmentos recuperados antes de generar la respuesta	10
2.3	Ciclo de razonamiento: de la petición a la respuesta fundamentada en pruebas	11
3.1	Bloques funcionales que conforman la plataforma a diseñar	15
3.2	Mapa de tecnologías evaluadas en función de su madurez y capacidad de integración en arquitecturas tipo Cloud	16
3.3	Cómo funciona la búsqueda semántica: el texto se convierte en un vector y se compara con los demás por similitud	20
4.1	Propuesta de arquitectura por capas	27
4.2	Tecnologías que conforman el stack de la plataforma ETMOV	28
4.3	Topología lógica del clúster Kubernetes: nodo máster, nodos trabajadores y persistencia.	30
4.4	Tres motores, tres tipos de dato: cada tecnología gestiona lo que mejor conoce	30
4.5	Stack tecnológico final organizado por capas	33
5.1	Configuración RAID-Z1	36
5.2	Topología de red SDN: Abstracción del tráfico físico a lógico mediante puentes virtuales.	37
5.3	Pipeline de procesamiento documental: del archivo al dato sugerido	40
5.4	Flujo RAG de consulta y redacción: la respuesta se genera a partir de fragmentos recuperados.	41
5.5	Arquitectura del asistente agéntico: el agente (Ollama) orquesta cuatro servidores MCP, cada uno conectado a su fuente de datos.	42
5.6	Modelo RBAC de la plataforma: tres roles con permisos diferenciados y cadena de revisión Técnico → Supervisor → Administrador.	44
5.7	Cadena de enriquecimiento automático de datos a partir del VIN: tres fuentes con prioridad decreciente convergen en la ficha técnica del expediente.	45
5.8	Flujo de diagnóstico asistido por fotografía: la IA sugiere, el técnico decide.	46
5.9	Pipeline de generación del borrador: el modelo de IA construye una propuesta basada en precedentes, que el técnico valida campo a campo.	46
5.10	Estados del expediente y roles responsables de cada transición. Las flechas discontinuas representan el flujo de devolución y corrección interna.	47
5.11	Secuencia de validación, contención y empaquetado del orquestador GitLab CI/CD.	47
A.1	Pantalla de acceso a la plataforma ETMOV con autenticación local y OAuth de Google.	51
A.2	Panel de resumen general (<i>dashboard</i>) con indicadores de actividad y registro de eventos recientes.	52

A.3	Panel de resumen general (<i>dashboard</i>) con indicadores de actividad y registro de eventos recientes.	52
A.4	Vista de listado de procedimientos con columnas de referencia, tipo, plantilla, estado y opciones.	53
A.5	Vista de la creación de un nuevo expediente.	53
A.6	Paso 2 en la creación de un nuevo expediente, enriquecimiento del informe con datos asociados. Datos de formulario	54
A.7	Paso 2 en la creación de un nuevo expediente, enriquecimiento del informe con datos asociados. Ficheros adjuntos y reportaje fotográfico	54
A.8	Paso 2 en la creación de un nuevo expediente, enriquecimiento del informe con datos asociados. Generación de borrador con IA con los elementos proporcionados	55
A.9	Paso 2 en la creación de un nuevo expediente, enriquecimiento del informe con datos asociados. Envío a revisión por parte de un supervisor	55
A.10	Paso 3 en la creación de un nuevo expediente. Devolución del informe al técnico para re-elaboración o firma digital para su cierre por parte de un supervisor	56
A.11	Paso 3 en la creación de un nuevo expediente. Devolucion de informe con nota anexa	56
A.12	Paso 3 en la creación de un nuevo expediente. Firma del documento digital	57
A.13	Explorador de ficheros enfocado a los expedientes existentes	58
A.14	Pantalla de perfil del usuario con historial de accesos y formulario de edición de datos.	58
A.15	Panel de ajustes generales: configuración del servidor SMTP para notificaciones por correo electrónico.	59
A.16	Configuración de los prompts con los modelos de IA.	60
A.17	Generación de API y Webhooks.	61
A.18	Plantillas disponible para la creación de expedientes nuevos.	62
A.19	Historial con todas las acciones que han ocurrido en la web. Auditoría y trazabilidad.	62

Índice de Tablas

3.1	Comparativa de frameworks CSS/UI de estilos	17
3.2	Comparativa de frameworks frontend en diseño reactivo y responsive	18
3.3	Comparativa de frameworks backend	19
3.4	Contraste entre Bare Metal, Virtualización y Cloud	26
4.1	Resumen del Stack Tecnológico	33

Bibliografía

- Abramova, Vera, Jorge Bernardino y Pedro Furtado (2014). «Which NoSQL database? A performance overview». En: *Open Journal of Databases (OJDB)* 1.2. https://www.ronpub.com/ojdb/OJDB_2014v1i2n02_Abramova.pdf, págs. 17-24.
- Anthropic (2024). *Model Context Protocol (MCP): An Open Standard for Connecting AI Assistants to Data Sources*. <https://modelcontextprotocol.io>.
- Aqua Security (2024). *Trivy: Vulnerability Scanner for Container Images, File Systems, and Git Repositories*. <https://trivy.dev>.
- Bonwick, Jeff et al. (2004). «The Zettabyte File System». En: *Proceedings of the 2nd Usenix Conference on File and Storage Technologies (FAST)*. <https://www.usenix.org/conference/fast-04/zettabyte-file-system>.
- Burns, Brendan et al. (2022). *Kubernetes: Up and Running*. 3rd. Sebastopol, CA: O'Reilly Media. ISBN: 978-1098110208.
- Cantelon, Mike et al. (2014). *Node.js in Action*. Shelter Island, NY: Manning Publications. ISBN: 978-1617290558.
- cert-manager Authors (2024). *cert-manager: X.509 Certificate Management for Kubernetes*. <https://cert-manager.io>.
- Chodorow, Kristina (2013). *MongoDB: The Definitive Guide*. 2nd. Sebastopol, CA: O'Reilly Media. ISBN: 978-1449344689.
- Express.js Contributors (2024). *Express — Fast, Unopinionated, Minimalist Web Framework for Node.js*. <https://expressjs.com>.
- Gao, Yunfan et al. (2024). «Retrieval-Augmented Generation for Large Language Models: A Survey». En: *arXiv preprint arXiv:2312.10997*. <https://arxiv.org/abs/2312.10997>.
- GitLab Inc. (2024). *GitLab CI/CD Documentation*. <https://docs.gitlab.com/ee/ci/>.
- Gobierno de España (2022). *Real Decreto 311/2022, de 3 de mayo, por el que se regula el Esquema Nacional de Seguridad*. Boletín Oficial del Estado, núm. 106. BOE-A-2022-7191. <https://www.boe.es/eli/es/rd/2022/05/03/311>.
- (2024). *Real Decreto 892/2024, de 10 de septiembre, por el que se aprueba el Reglamento de Vehículos Históricos*. Boletín Oficial del Estado, núm. 219. BOE-A-2024-18614. https://www.boe.es/diario_boe/txt.php?id=BOE-A-2024-18614.
- Grafana Labs (2024). *k6 — Modern Load Testing Tool, Built for Developer Happiness*. <https://k6.io>.
- Hardt, D. (2012). *RFC 6749: The OAuth 2.0 Authorization Framework*. Internet Engineering Task Force (IETF). <https://datatracker.ietf.org/doc/html/rfc6749>.
- Helm Authors (2024). *Helm — The Kubernetes Package Manager*. <https://helm.sh>.
- ISO/IEC (2012). *ISO/IEC 17020:2012 — Conformity assessment — Requirements for the operation of various types of bodies performing inspection*. <https://www.iso.org/standard/52994.html>. Geneva, Switzerland.

- Jones, Michael B., John Bradley y Nat Sakimura (2015). *RFC 7519: JSON Web Token (JWT)*. Internet Engineering Task Force (IETF) Standards Track. <https://datatracker.ietf.org/doc/html/rfc7519>.
- Kim, Gene et al. (2016). *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. Portland, OR: IT Revolution Press. ISBN: 978-1942788003.
- Kochhar, Khushboo e Indu Arora (2016). «Performance Evaluation of VirtIO Network Drivers in KVM Based Virtual Machine». En: *International Journal of Advanced Research in Computer Engineering and Technology (IJARCET)* 5.5, págs. 1354-1358.
- Lewis, Patrick et al. (2020). «Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks». En: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 33. Curran Associates, Inc., págs. 9459-9474.
- Lukša, Marko (2019). *Kubernetes in Action*. 2nd. Shelter Island, NY: Manning Publications. ISBN: 978-1617297618.
- Merkel, Dirk (2014). «Docker: Lightweight Linux Containers for Consistent Development and Deployment». En: *Linux Journal*. Vol. 2014. 239. <https://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment>.
- MinIO Inc. (2023). *MinIO High Performance Object Storage*. S3-compatible object storage. <https://min.io>.
- Monarch, Robert (Munro) (2021). *Human-in-the-Loop Machine Learning: Active Learning and Annotation for Human-Centered AI*. Shelter Island, NY: Manning Publications. ISBN: 978-1617296741.
- Mori, Shunji, Ching Y. Suen y Kazuhiko Yamamoto (1992). «Historical review of OCR research and development». En: *Proceedings of the IEEE* 80.7, págs. 1029-1058. doi: [10.1109/5.156468](https://doi.org/10.1109/5.156468).
- Morote, Eduardo San Martin y Pinia Contributors (2024). *Pinia — The Intuitive Store for Vue.js*. <https://pinia.vuejs.org>.
- Newman, Sam (2021). *Building Microservices: Designing Fine-Grained Systems*. 2nd. Sebastopol, CA: O'Reilly Media. ISBN: 978-1492034025.
- NGINX Inc. (2024). *NGINX Ingress Controller for Kubernetes*. <https://kubernetes.github.io/ingress-nginx/>.
- Ollama Team (2024). *Ollama: Run Large Language Models Locally*. <https://ollama.com>.
- Parlamento Europeo y Consejo de la Unión Europea (2016). *Reglamento (UE) 2016/679 del Parlamento Europeo y del Consejo, de 27 de abril de 2016, relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales*. Diario Oficial de la Unión Europea, L 119. RGPD/GDPR. <https://eur-lex.europa.eu/eli/reg/2016/679/oj>.
- Patterson, David A., Garth A. Gibson y Randy H. Katz (1988). «A Case for Redundant Arrays of Inexpensive Disks (RAID)». En: *ACM SIGMOD Record* 17.3, págs. 109-116. doi: [10.1145/971701.50214](https://doi.org/10.1145/971701.50214).
- Provos, Niels y David Mazieres (1999). «A Future-Adaptable Password Scheme». En: *Proceedings of the 1999 USENIX Annual Technical Conference*. USENIX Association, págs. 32-32.
- Proxmox Server Solutions GmbH (2024). *Proxmox Virtual Environment: Open-Source Server Virtualization Platform*. <https://www.proxmox.com/proxmox-virtual-environment>.
- Qdrant Team (2023). *Qdrant: High-performance, massive-scale Vector Database*. <https://qdrant.tech>.
- Radford, Alec et al. (2021). «Learning Transferable Visual Models From Natural Language Supervision». En: *Proceedings of the 38th International Conference on Machine Learning (ICML)*. <https://arxiv.org/abs/2103.00020>, págs. 8748-8763.
- Ramírez, Sebastián (2024). *FastAPI — Modern, Fast (high-performance), Web Framework for Building APIs with Python*. <https://fastapi.tiangolo.com>.

- Reimers, Nils e Iryna Gurevych (2019). «Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks». En: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. <https://arxiv.org/abs/1908.10084>, págs. 3982-3992.
- Richardson, Chris (2018). *Microservices Patterns: With Examples in Java*. Shelter Island, NY: Manning Publications. ISBN: 978-1617294549.
- Sandhu, Ravi S. et al. (1996). «Role-Based Access Control Models». En: *IEEE Computer* 29.2, págs. 38-47. DOI: [10.1109/2.485845](https://doi.org/10.1109/2.485845).
- Stonebraker, Michael (2010). «SQL Databases v. NoSQL Databases». En: *Communications of the ACM* 53.4, págs. 10-11. DOI: [10.1145/1721654.1721659](https://doi.org/10.1145/1721654.1721659).
- Vaswani, Ashish et al. (2017). «Attention Is All You Need». En: *Advances in Neural Information Processing Systems* 30. <https://arxiv.org/abs/1706.03762>.
- Vial, Gregory (2019). «Understanding digital transformation: A review and a research agenda». En: *The Journal of Strategic Information Systems* 28.2, págs. 118-144. DOI: [10.1016/j.jsis.2019.01.003](https://doi.org/10.1016/j.jsis.2019.01.003).
- Wathan, Adam et al. (2024). *Tailwind CSS — A Utility-First CSS Framework*. <https://tailwindcss.com>.
- Xu, Yiheng et al. (2020). «LayoutLM: Pre-training of Text and Layout for Document Image Understanding». En: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, págs. 1192-1200. DOI: [10.1145/3394486.3403172](https://doi.org/10.1145/3394486.3403172).
- You, Evan y Vue.js Contributors (2024). *Vue.js 3 — The Progressive JavaScript Framework*. <https://vuejs.org>.