COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

# INTEGRACIÓN DE COMUNICACIONES IOT Y BLOCKCHAIN EN UNA MAQUETA FUNCIONAL DE CADENA DE SUMINISTRO

Autor: Miriam Herbada González

Director: José Luis Gahete Díaz

Co-director: Carlos Miguel Vallez Fernández

Madrid

Junio 2026

**Declaración de originalidad**

Declaro bajo mi responsabilidad que el Proyecto presentado con el título **Integración de Comunicaciones IoT y Blockchain en una Maqueta Funcional de Cadena de Suministro** de la ETS de Ingeniería – ICAI de la Universidad Pontificia Comillas en el curso académico 2025-2026 es de mi autoría y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

**Uso de Inteligencia Artificial**[1]

Declaro bajo mi responsabilidad que (indicar la opción correcta):

☐ No he utilizado Inteligencia Artificial en la elaboración del presente documento.

☒ He utilizado Inteligencia Artificial en la elaboración del presente documento y/o del Anexo B siempre en las condiciones permitidas por la Universidad Pontificia Comillas, es decir, aplicando el Nivel 2 de la Escala de Evaluación de Perkins et al. (2024): "*La IA puede utilizarse para actividades previas a la tarea, como la lluvia de ideas, la descripción y la investigación inicial. Este nivel se centra en el uso de la IA para la planificación, las síntesis y la generación de ideas, pero las evaluaciones deben hacer hincapié en la capacidad de desarrollar y refinar estas ideas de forma independiente*". En concreto, las Inteligencia Artificial ha sido empleada para:

| |
|---|
| La inteligencia artificial se ha utilizado como herramienta de apoyo en la fase previa de organización del trabajo, principalmente para explorar posibles cadenas de suministro, ordenar ideas, estructurar contenidos iniciales y digitalizar esquemas conceptuales elaborados durante el desarrollo del proyecto. |

| |
|---|
|  |
| Firmado (alumno): Miriam Herbada González |
| Fecha: 11 de junio de 2026 |

**Autorización para la entrega del Proyecto**

[1] Esta declaración se refiere al uso de la Inteligencia Artificial generativa para realizar los documentos del Proyecto (Anexo B y Memoria). No aplica a Proyectos donde, por su naturaleza, deban emplear inteligencia artificial como parte de los mismos (aplicación de técnicas de aprendizaje automático, redes neuronales, análisis de datos…)

| El Director del Proyecto | El Co-Director del Proyecto |
| --- | --- |
| Firmado por GAHETE DIAZ JOSE LUIS - ***0287** el día 12/06/2026 con un certificado emitido por AC FNMT Usuarios | Carlos Miguel Vallez Fernández<br>Firmado digitalmente por Carlos Miguel Vallez Fernández<br>Fecha: 2026.06.12 09:52:48 +02'00' |
| Fdo: | Fdo: |
| Fecha: | Fecha: |


GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

# INTEGRACIÓN DE COMUNICACIONES IOT Y BLOCKCHAIN EN UNA MAQUETA FUNCIONAL DE CADENA DE SUMINISTRO

Autor: Miriam Herbada González

Director: José Luis Gahete Díaz

Co-director: Carlos Miguel Vallez Fernández

Madrid

Junio 2026

# Agradecimientos

En primer lugar, me gustaría expresar mi más sincero agradecimiento a José Luis y Carlos, mis tutores, por su orientación, disponibilidad e implicación a lo largo de todo el proyecto. Sus consejos, correcciones y acompañamiento han sido fundamentales para dar forma a este trabajo y para mejorar su calidad, haciendo que el resultado final sea mucho más completo de lo que habría sido sin su ayuda.

También quiero agradecer a mi familia el apoyo que me ha acompañado durante todos estos años. Gracias por estar presentes en cada etapa, por ayudarme a seguir adelante en los momentos más exigentes y por celebrar conmigo cada pequeño avance.

# INTEGRACIÓN DE COMUNICACIONES IOT Y BLOCKCHAIN EN UNA MAQUETA FUNCIONAL DE CADENA DE SUMINISTRO

Autor: Herbada González, Miriam.
Director: Gahete Díaz, José Luis
Co-director: Vallez Fernández, Carlos Miguel
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

## RESUMEN DEL PROYECTO

Este trabajo presenta el diseño e implementación de una maqueta funcional de sistema de trazabilidad para la cadena de suministro del aceite de oliva, basado en la integración de comunicaciones IoT mediante el protocolo MQTT y tecnología blockchain Ethereum. El sistema certifica de forma inmutable varias etapas del proceso de producción, desde la recogida de la aceituna en el campo hasta la extracción final del aceite en la almazara, utilizando un smart contract desplegado en Solidity 0.8.24, un backend Spring Boot con 23 endpoints REST y un simulador de varios sensores industriales containerizado con Docker. Los resultados obtenidos demuestran la viabilidad técnica de la integración y la trazabilidad extremo a extremo verificable mediante hashes de transacción en blockchain.

**Palabras clave:** Blockchain, IoT, MQTT, Trazabilidad, Ethereum, Smart Contracts, Cadena de Suministro

### 1. Introducción

La industria del aceite de oliva virgen enfrenta un problema estructural de trazabilidad: los registros del proceso productivo son en su mayoría manuales, centralizados y susceptibles de manipulación, lo que dificulta la verificación de la autenticidad del producto por parte de compradores, distribuidores y organismos certificadores.

Las tecnologías blockchain e IoT ofrecen una combinación especialmente adecuada para abordar este problema. La red blockchain garantiza la inmutabilidad de los registros mediante encadenamiento criptográfico: modificar cualquier evento registrado requeriría recalcular todos los bloques posteriores, lo que es computacionalmente inviable [1]. Los sensores IoT, por su parte, eliminan la dependencia de la introducción manual de datos, reduciendo la posibilidad de error o manipulación en el origen del dato [3].

Este trabajo propone y valida una arquitectura de cuatro capas que integra ambas tecnologías en un sistema demostrable en entorno de desarrollo controlado, sentando las bases técnicas para una solución transferible a entornos productivos reales [4].

## 2. Definición del proyecto

El objetivo del proyecto es implementar una maqueta funcional que cubra el proceso de producción del aceite de oliva desde el campo hasta la almazara, certificando cada evento crítico de forma inmutable en una red blockchain Ethereum local [2]. El sistema se articula en torno a tres principios de diseño: inmutabilidad, verificabilidad y resiliencia. El principio de inmutabilidad garantiza que cada evento queda sellado criptográficamente en blockchain y no puede ser alterado a posteriori. El de verificabilidad permite que cualquier actor de la cadena compruebe la autenticidad de cualquier registro mediante su hash de transacción, sin depender del sistema como intermediario de confianza. El de resiliencia asegura que el sistema continúa operando, aunque alguno de sus componentes externos no esté disponible.

En la Ilustración 1 se pueden observar las tres fases del proceso productivo que cubre el sistema. La fase de campo y transporte registra la creación del lote con datos de trazabilidad de origen (identificador del contenedor, matrícula del camión y coordenadas GPS), el cierre de la compuerta del camión y cualquier apertura durante el transporte. La fase de almazara registra de forma automática el pesaje de las aceitunas, el volcado en tolva, el lavado con control de calidad del agua, el pesaje en cinta, la molienda, el batido, la separación en decanter, la centrifugación y la extracción final del aceite. Todos los eventos asociados al proceso industrial de la fase de almazara son generados automáticamente por sensores IoT simulados, sin intervención manual del operario más allá de la identificación de la instalación donde se lleva a cabo el proceso.

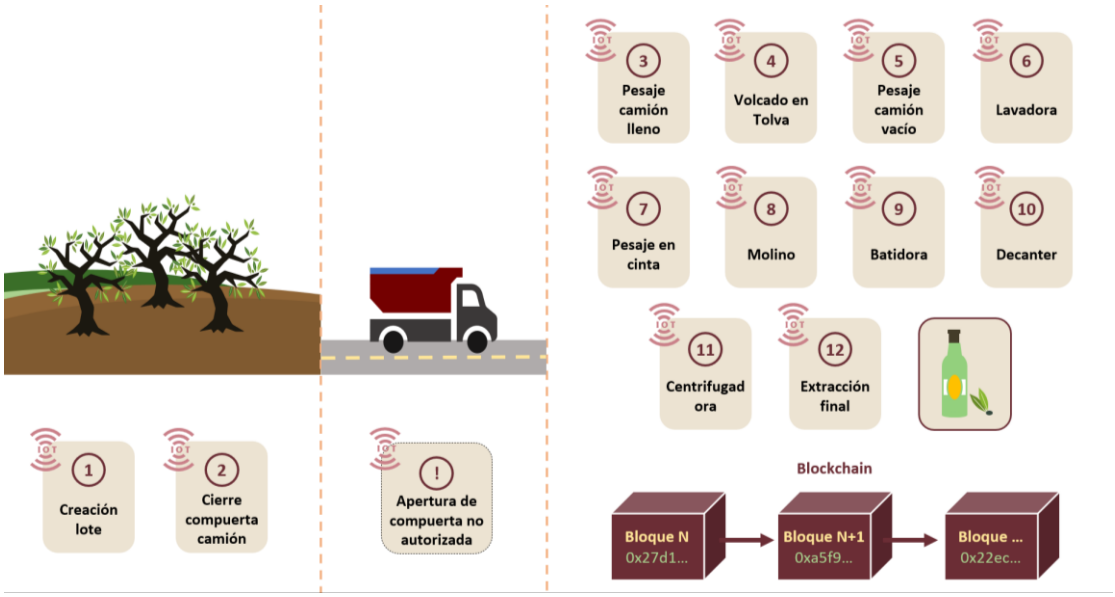


*Ilustración 1 - Tres etapas producción aceite de oliva*

## 3. Descripción del sistema

La arquitectura del sistema se organiza en cuatro capas con responsabilidades claramente diferenciadas, tal y como se muestra en la Ilustración 2.

La capa de presentación es una aplicación web desarrollada con React 19.2, Vite y TypeScript que guía al operario por el flujo completo de registro en tres fases: campo, transporte y almazara, y permite consultar el historial de trazabilidad de cualquier lote.

La capa de aplicación es un backend Spring Boot 3.5.9 con 23 endpoints REST que coordina la escritura simultánea en la base de datos local y en blockchain, e integra la comunicación con los sensores IoT mediante el protocolo MQTT.

La capa de persistencia combina una base de datos H2 en memoria con el smart contract CadenaAceite desplegado en Ganache como registro inmutable. Cada evento se persiste junto con el hash de transacción correspondiente, que actúa como enlace criptográfico verificable entre ambas capas. La codificación de las llamadas al contrato sigue el estándar ABI de Ethereum [5], y la comunicación con el nodo se realiza mediante Web3j [6].

La capa de infraestructura IoT está compuesta por un broker Eclipse Mosquitto y un simulador Python containerizados con Docker Compose, que replica varios tipos de sensores industriales mediante el protocolo MQTT. La arquitectura completa se describe en el apartado 4.2.

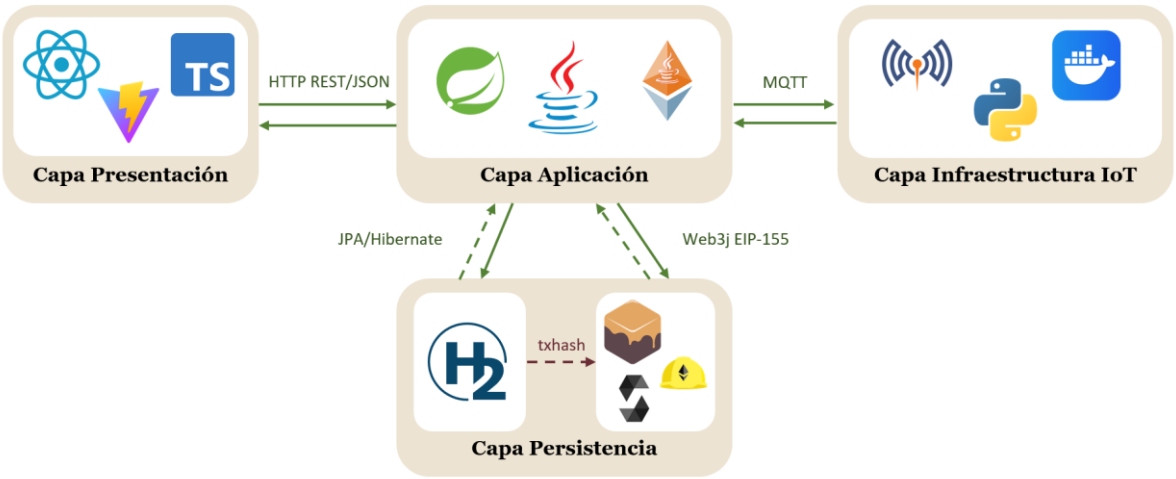


*Ilustración 2 - Arquitectura general del sistema de trazabilidad IoT-Blockchain.*

## 4. Resultados

La validación del sistema se realizó mediante la ejecución del flujo completo de extremo a extremo y una suite de 42 tests automatizados sobre los 23 endpoints de la API REST con 0 fallos. El flujo genera hasta 13 eventos blockchain certificados, uno por cada etapa del proceso, cada uno con un hash de transacción único verificable directamente en el nodo Ganache sin depender del sistema como intermediario. La Ilustración 3 muestra la pantalla de finalización del proceso, con los diez eventos de la fase de almazara certificados en blockchain y sus hashes de transacción disponibles para verificación independiente.

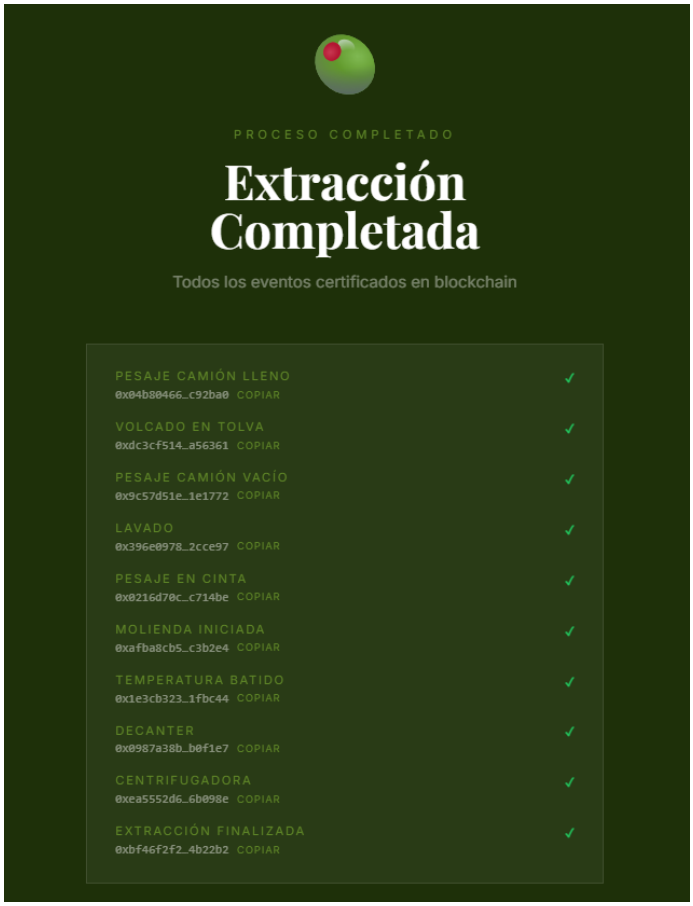


*Ilustración 3 - Pantalla de extracción completada del sistema, mostrando los diez eventos de la fase de almazara certificados en blockchain con sus hashes de transacción verificables.*

## 5. Conclusiones

El sistema desarrollado demuestra la viabilidad técnica de integrar IoT y blockchain en una cadena de suministro agroalimentaria real [4]. La combinación del protocolo MQTT para la adquisición automática de datos de sensores con el registro inmutable en blockchain Ethereum [2] proporciona un nivel de garantía de integridad que los sistemas de trazabilidad

centralizados no pueden ofrecer por diseño. El hash de transacción asociado a cada evento constituye una prueba criptográfica independiente del sistema que puede ser verificada por cualquier actor de la cadena sin confiar en ningún intermediario.

Las principales líneas de trabajo futuro identificadas son el despliegue sobre una red blockchain persistente, la sustitución del simulador por hardware IoT físico, la incorporación de un sistema de roles por actor mediante AccessControl de OpenZeppelin, y la extensión del sistema a las fases de envasado y distribución para cubrir la cadena completa hasta el consumidor final.

## 6. Referencias


[1] Nakamoto, S. "Bitcoin: A Peer-to-Peer Electronic Cash System". 2008. https://bitcoin.org/bitcoin.pdf.

[2] Buterin, V. "Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform". Ethereum Foundation, 2014. https://ethereum.org/en/whitepaper/.

[3] Salah, K.; Nizamuddin, N.; Jayaraman, R.; Omar, M. "Blockchain-Based Soybean Traceability in Agricultural Supply Chain". IEEE Access, vol. 7, pp. 73295-73305, 2019.

[4] T. Frikha, J. Ktari, and H. Hamam, "Blockchain Olive Oil Supply Chain," in Risks and Security of Internet and Systems. CRiSIS 2022, S. Kallel et al., Eds., Lecture Notes in Computer Science, vol. 13857. Springer, 2023, pp. 101–113. https://doi.org/10.1007/978-3-031-31108-6_8.

[5] Ethereum Foundation. "ABI Specification". Ethereum Documentation. https://docs.soliditylang.org/en/latest/abi-spec.html.

[6] Web3j Project. "Web3j: Lightweight Java and Android library for integration with Ethereum clients". https://docs.web3j.io.

# INTEGRATION OF IOT COMMUNICATIONS AND BLOCKCHAIN IN A FUNCTIONAL SUPPLY CHAIN PROTOTYPE

Author: Herbada González, Miriam.
Supervisor: Gahete Díaz, José Luis
Co-supervisor: Vallez Fernández, Carlos Miguel
Collaborating Entity: ICAI – Universidad Pontificia Comillas

## ABSTRACT

This work presents the design and implementation of a functional prototype of a traceability system for the olive oil supply chain, based on the integration of IoT communications using the MQTT protocol and Ethereum blockchain technology. The system immutably certifies several stages of the production process, from olive harvesting in the field to the final oil extraction at the mill, using a smart contract deployed in Solidity 0.8.24, a Spring Boot backend with 23 REST endpoints, and a containerised industrial sensor simulator built with Docker. The results demonstrate the technical feasibility of the integration and end-to-end traceability verifiable through blockchain transaction hashes.

**Keywords:** Blockchain, IoT, MQTT, Traceability, Ethereum, Smart Contracts, Supply Chain

### 1. Introduction

The virgin olive oil industry faces a structural traceability problem: production process records are largely manual, centralised, and susceptible to manipulation, making it difficult for buyers, distributors, and certification bodies to verify product authenticity.

Blockchain and IoT technologies offer a particularly suitable combination to address this challenge. The blockchain network guarantees record immutability through cryptographic chaining: modifying any registered event would require recalculating all subsequent blocks, which is computationally infeasible [1]. IoT sensors, in turn, eliminate dependence on manual data entry, reducing the possibility of error or manipulation at the point of origin [3].

This work proposes and validates a four-layer architecture that integrates both technologies into a system demonstrable in a controlled development environment, establishing the technical foundations for a solution transferable to real production settings [4].

## 2. Project Definition

The objective of this project is to implement a functional prototype covering the olive oil production process from the field to the mill, certifying each critical event immutably on a local Ethereum blockchain network [2]. The system is built around three design principles: immutability, verifiability, and resilience. The immutability principle ensures that each event is cryptographically sealed on the blockchain and cannot be altered after the fact. Verifiability allows any actor in the chain to independently verify the authenticity of any record through its transaction hash, without relying on the system as a trusted intermediary. Resilience ensures that the system continues to operate even when one of its external components is unavailable.

Illustration 1 shows the three phases of the production process covered by the system. The field and transport phase records lot creation with origin traceability data (container identifier, truck licence plate and GPS coordinates), truck gate closure, and any gate openings during transit. The mill phase automatically records olive weighing, hopper unloading, water quality-controlled washing, conveyor belt weighing, milling, malaxation, decanter separation, centrifugation, and final oil extraction. All mill phase events associated with the industrial process are generated automatically by simulated IoT sensors, without manual operator intervention beyond identifying the processing facility.

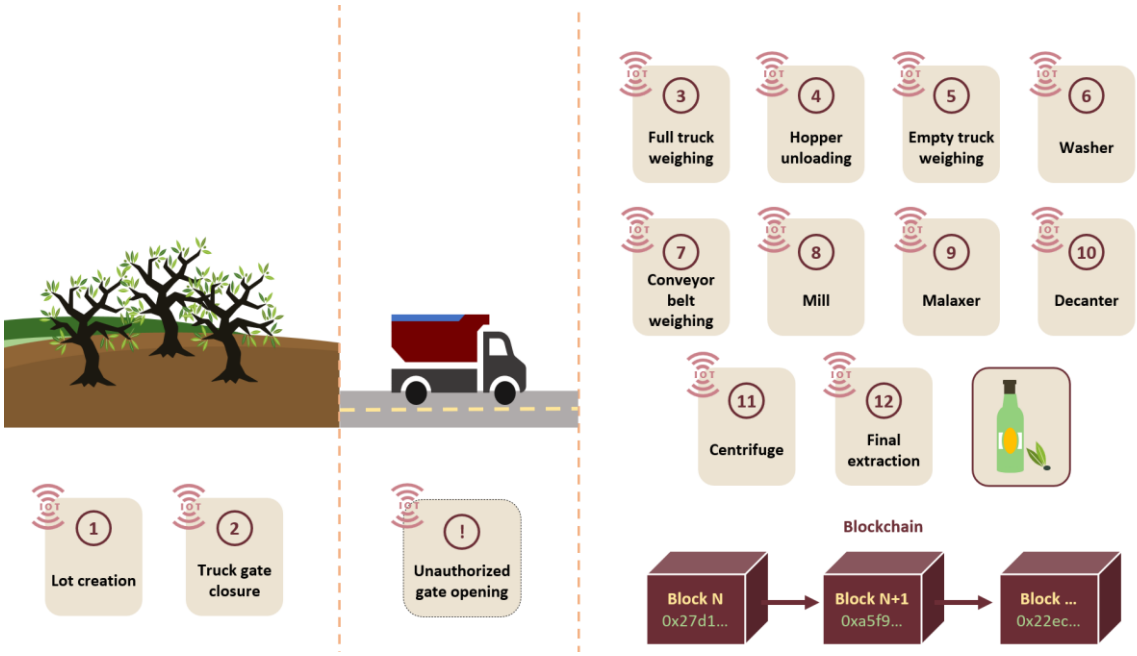


*Illustration 1 - Three stages of the olive oil production process.*

## 3. System Description

The system architecture is organised into four layers with clearly differentiated responsibilities, as shown in Illustration 2.

The presentation layer is a web application developed with React 19.2, Vite, and TypeScript that guides the operator through the complete registration workflow across three phases: field, transport, and mill, and allows querying the traceability history of any lot.

The application layer is a Spring Boot 3.5.9 backend with 23 REST endpoints that coordinates simultaneous writes to both the local database and the blockchain, and integrates communication with IoT sensors via the MQTT protocol.

The persistence layer combines an in-memory H2 database with the CadenaAceite smart contract deployed on Ganache as an immutable registry. Each event is persisted alongside its corresponding transaction hash, which acts as a verifiable cryptographic link between both layers. Contract calls are encoded following the Ethereum ABI standard [5], and node communication is handled through Web3j [6].

The IoT infrastructure layer consists of an Eclipse Mosquitto broker and a Python sensor simulator containerised with Docker Compose, replicating several types of industrial sensors via the MQTT protocol. The complete architecture is described in Section 4.2.

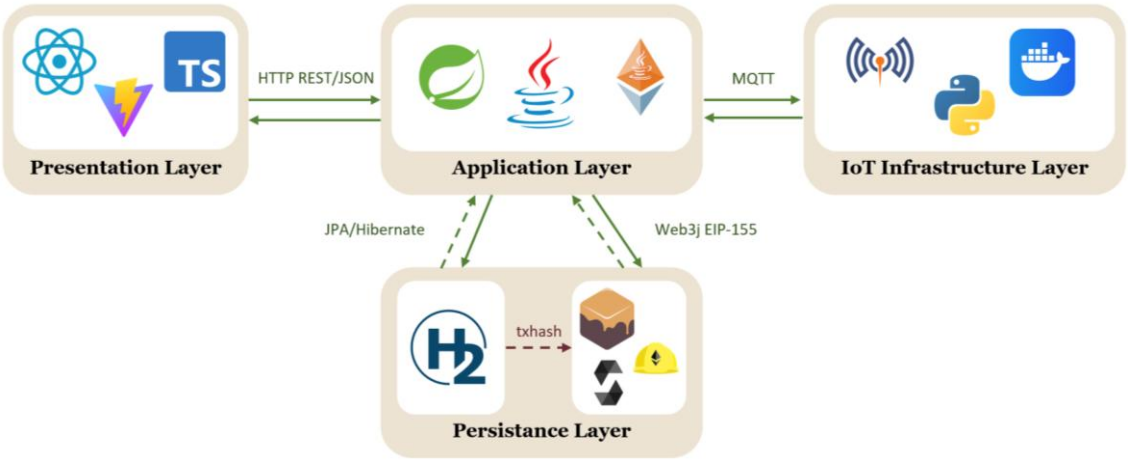


*Illustration 2 - General architecture of the IoT-Blockchain traceability system.*

## 4. Results

System validation was carried out through end-to-end execution of the complete workflow and a suite of 42 automated tests covering all 23 REST API endpoints with 0 failures. The workflow generates up to 13 certified blockchain events, one per process stage, each with a unique transaction hash directly verifiable on the Ganache node without depending on the system as an intermediary. Illustration 3 shows the process completion screen, with the ten mill phase events certified on the blockchain and their transaction hashes available for independent verification.

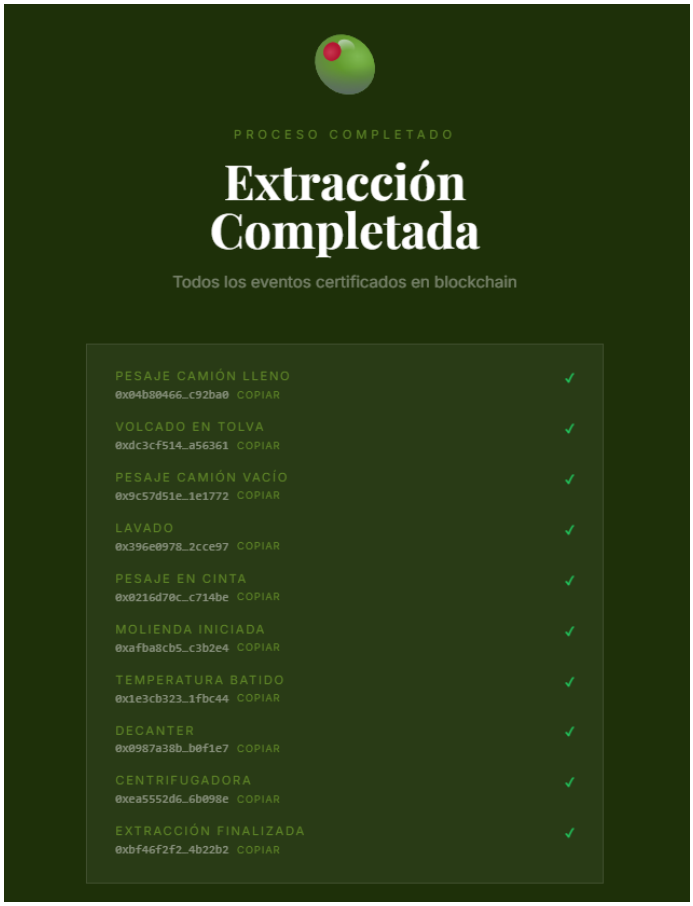


*Illustration 3 - Process completion screen showing the ten mill phase events certified on the blockchain with their verifiable transaction hashes.*

## 5. Conclusions

The developed system demonstrates the technical feasibility of integrating IoT and blockchain in a real agri-food supply chain [4]. The combination of the MQTT protocol for automatic sensor data acquisition with immutable registration on the Ethereum blockchain [2] provides a level of integrity guarantee that centralised traceability systems cannot offer

by design. The transaction hash associated with each event constitutes a cryptographic proof independent of the system, verifiable by any actor in the chain without trusting any intermediary.

The main lines of future work identified are deployment on a persistent blockchain network, replacement of the simulator with physical IoT hardware, incorporation of a role-based access control system using OpenZeppelin's AccessControl, and extension of the system to the bottling and distribution phases to cover the complete chain through to the end consumer.

## 6. References


[1] Nakamoto, S. "Bitcoin: A Peer-to-Peer Electronic Cash System". 2008. https://bitcoin.org/bitcoin.pdf.

[2] Buterin, V. "Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform". Ethereum Foundation, 2014. https://ethereum.org/en/whitepaper/.

[3] Salah, K.; Nizamuddin, N.; Jayaraman, R.; Omar, M. "Blockchain-Based Soybean Traceability in Agricultural Supply Chain". IEEE Access, vol. 7, pp. 73295-73305, 2019.

[4] T. Frikha, J. Ktari, and H. Hamam, "Blockchain Olive Oil Supply Chain," in *Risks and Security of Internet and Systems. CRiSIS 2022*, S. Kallel et al., Eds., Lecture Notes in Computer Science, vol. 13857. Springer, 2023, pp. 101–113. https://doi.org/10.1007/978-3-031-31108-6_8.

[5] Ethereum Foundation. "ABI Specification". Ethereum Documentation. https://docs.soliditylang.org/en/latest/abi-spec.html.

[6] Web3j Project. "Web3j: Lightweight Java and Android library for integration with Ethereum clients". https://docs.web3j.io.

# *Índice de la memoria*

# *Índice de figuras*

# *Índice de tablas*

# Capítulo 1. INTRODUCCIÓN

## *1.1 MOTIVACIÓN DEL PROYECTO*

El sector agroalimentario se enfrenta en la actualidad a una creciente demanda de transparencia por parte de los consumidores, los organismos reguladores y los propios actores de la cadena de suministro. En un contexto de globalización de los mercados, la capacidad de verificar el origen, la calidad y el proceso de producción de un alimento se ha convertido en un factor determinante para diferenciarse.

El aceite de oliva constituye uno de los productos más emblemáticos del sector agroalimentario español y europeo. España es el primer productor mundial, con una producción que supera el 40% del total global según datos del Consejo Oleícola Internacional, y la Unión Europea, encabezada por España, Italia y Grecia, acumula más del 70% de la producción mundial. Solo en la campaña 2022/2023, la producción española alcanzó 1,4 millones de toneladas [1]. Sin embargo, la relevancia económica del sector contrasta con una problemática estructural no resuelta: el fraude en la denominación de origen y la categoría de calidad. El aceite de oliva virgen extra (AOVE) es uno de los alimentos con mayor índice de fraude alimentario en Europa, con prácticas que incluyen el etiquetado incorrecto de categoría, la mezcla con aceites de otras procedencias y la adulteración con aceites de semillas de menor valor. El impacto económico para el sector se estima en cientos de millones de euros anuales.

La normativa europea obliga a disponer de sistemas de trazabilidad en toda la cadena alimentaria (concretamente el Reglamento (CE) n.º 178/2002 [2]) pero los mecanismos de verificación siguen siendo, en gran medida, manuales y centralizados. La información sobre el origen, el transporte y el pesaje de la aceituna se documenta en papel o en sistemas informáticos privados y no interoperables, sin garantía técnica de integridad ni posibilidad de auditoría pública.

La aparición y consolidación de dos tecnologías emergentes, el Internet de las Cosas (IoT) y blockchain, ha abierto nuevas posibilidades para abordar estas limitaciones de forma complementaria. El IoT permite capturar datos físicos como pueden ser el peso, la temperatura o la apertura de compuertas directamente en el origen, eliminando la intervención humana y el error asociado. Blockchain, por su parte, ofrece un mecanismo de registro inmutable y descentralizado: una vez que un dato se escribe en la cadena y queda incluido en un bloque confirmado, su alteración requeriría recalcular ese bloque y todos los posteriores con una potencia de cómputo superior a la del resto de la red, lo que en la práctica lo hace irrefutable. La combinación de ambas tecnologías en un sistema de trazabilidad híbrido representa una línea de investigación y desarrollo activa tanto en la literatura científica como en la industria, con proyectos reales como IBM Food Trust [3] en el sector alimentario o BeefChain [4] en la trazabilidad ganadera.

Este trabajo surge de la necesidad de evaluar la viabilidad técnica de esta aproximación en un entorno controlado y reproducible, mediante el desarrollo de una maqueta funcional aplicada al caso concreto de la cadena de suministro del aceite de oliva, centrándose en las etapas que tienen lugar entre la recogida de las aceitunas hasta la extracción del aceite en la almazara.

## *1.2 PROBLEMÁTICA ABORDADA*

La cadena de suministro del aceite de oliva comprende un conjunto de actores interconectados: agricultores, cooperativas, transportistas, almazaras, envasadores y distribuidores, cuya coordinación eficiente es fundamental para garantizar la calidad y autenticidad del producto final. Sin embargo, los sistemas de información que sostienen esta cadena presentan una serie de limitaciones estructurales que comprometen tanto la integridad de los datos como la capacidad de auditoría externa.

### 1.2.1 CENTRALIZACIÓN DE LOS REGISTROS

La información sobre los lotes de aceitunas y su trayectoria a lo largo de la cadena se almacena en bases de datos controladas por un único actor: la cooperativa, la almazara o la

entidad certificadora. Este modelo centralizado presenta dos vulnerabilidades fundamentales. En primer lugar, cualquier modificación indebida de los registros puede pasar desapercibida, ya que no existe un mecanismo técnico que garantice la inmutabilidad de los datos una vez registrados. En segundo lugar, la auditoría externa resulta costosa y lenta, puesto que requiere acceder a sistemas propietarios de cada actor y confiar en la veracidad de los datos que estos proporcionan. La confianza, en este modelo, es una garantía de buena fe entre las partes, no una garantía técnica verificable por terceros.

### 1.2.2 REGISTROS MANUALES Y PROPENSOS A ERRORES

El pesaje de la aceituna y la asignación de lotes se realiza frecuentemente de forma manual o mediante sistemas informáticos cerrados y no interoperables. El operario introduce el peso en una terminal, que genera un albarán en papel o un registro en una base de datos local. Este proceso introduce múltiples puntos de fallo: errores de transcripción, pérdida de documentos, discrepancias entre los datos registrados por el agricultor y los registrados por la almazara, o simplemente la posibilidad de alterar un registro antes de que sea auditado. En un sector donde el precio pagado al agricultor depende directamente del peso registrado, la fiabilidad de este dato es crítica.

### 1.2.3 FALTA DE TRANSPARENCIA PARA EL CONSUMIDOR

El consumidor final no dispone de mecanismos sencillos y verificables para conocer el origen y el proceso de producción del aceite que adquiere. Las etiquetas y certificaciones de denominación de origen protegida (DOP) o indicación geográfica protegida (IGP) ofrecen una garantía formal, pero su verificación depende de entidades intermedias cuya supervisión es periódica y no en tiempo real. En ausencia de un sistema de trazabilidad transparente, el consumidor debe confiar en la cadena de custodia sin posibilidad de verificación independiente.

### 1.2.4 RIESGO DE FRAUDE Y ADULTERACIÓN

La ausencia de mecanismos criptográficos de verificación hace que los registros de calidad y origen sean susceptibles de alteración o falsificación. Un registro en papel puede

modificarse; una entrada en una base de datos centralizada puede sobrescribirse sin dejar rastro auditable. En el contexto del aceite de oliva, donde la diferencia de precio entre un aceite virgen extra de denominación de origen y un aceite de menor categoría puede superar el 300%, el incentivo económico para falsificar los registros es considerable. Los casos de fraude detectados en el sector ilustran las consecuencias prácticas de esta vulnerabilidad, tal y como reconoce el Reglamento de Ejecución (UE) 2022/2104 [5], que establece métodos de análisis específicos para la determinación de los parámetros de autenticidad del aceite de oliva.

### 1.2.5 CONSECUENCIAS SOBRE EL SECTOR

El conjunto de estas limitaciones tiene consecuencias que van más allá del perjuicio económico inmediato. La reputación de las denominaciones de origen protegidas se ve erosionada cuando los casos de fraude trascienden a los medios de comunicación, afectando indirectamente a productores que sí cumplen con los estándares de calidad. La Comisión Europea ha reconocido esta problemática en su Estrategia "De la granja a la mesa" [6], que identifica la mejora de la trazabilidad alimentaria como una de las prioridades para el periodo 2020-2030.

Frente a este conjunto de problemas, este trabajo propone una arquitectura que aborda de forma directa las limitaciones identificadas: la captura automatizada de datos mediante dispositivos IoT elimina la intervención humana en el registro de los eventos físicos críticos; el anclaje criptográfico de esos eventos en blockchain garantiza su inmutabilidad y verificabilidad por cualquier actor de la cadena sin necesidad de confiar en un intermediario centralizado; y la capa de integración REST permite que estos mecanismos sean accesibles desde cualquier sistema externo de forma estándar e interoperable.

## *1.3 DEFINICIÓN DEL PROYECTO*

Este Trabajo de Fin de Grado consiste en el diseño e implementación de una maqueta funcional de un sistema de trazabilidad para la cadena de suministro del aceite de oliva, que integra tecnologías IoT y blockchain en una arquitectura coherente, funcional y demostrable.

El sistema cubre el proceso completo desde la recogida de la aceituna en el campo hasta la obtención del aceite en la almazara, certificando cada evento crítico de forma inmutable en blockchain.

La solución propuesta se articula en torno a tres principios de diseño:

- Inmutabilidad: cada evento registrado queda sellado criptográficamente en una red blockchain Ethereum local, de forma que ningún actor puede alterarlo a posteriori sin invalidar toda la cadena de bloques posterior.
- Trazabilidad verificable: cualquier actor con acceso al sistema puede consultar el historial completo de un lote y verificar de forma independiente la autenticidad y detalle de cada etapa mediante su hash de transacción.
- Resiliencia: el sistema está diseñado para seguir funcionando, aunque alguno de sus componentes no esté disponible, degradando las funcionalidades opcionales sin interrumpir el servicio principal.

La arquitectura técnica adoptada para implementar estos principios, las tecnologías seleccionadas y las decisiones de diseño que las justifican se describen a partir del Capítulo 3.

# Capítulo 2. ESTADO DE LA CUESTIÓN

## *2.1 TRAZABILIDAD EN LAS CADENAS DE SUMINISTRO*

La trazabilidad en la cadena agroalimentaria se define como la capacidad de capturar, almacenar y transmitir información adecuada sobre un producto alimentario a lo largo de todas las etapas de producción, transformación y distribución, de forma que pueda ser verificado en cualquier momento por los actores implicados [7]. En sectores como el aceite de oliva, donde la calidad, el origen geográfico y los métodos de producción influyen directamente en el valor del producto y en la confianza del consumidor, la trazabilidad resulta especialmente relevante.

Desde sus orígenes, los sistemas de trazabilidad alimentaria han evolucionado desde el registro manual en papel hasta soluciones digitales progresivamente más sofisticadas. Bosona y Gebresenbet [7] identificaron en su revisión sistemática de la literatura que los sistemas de trazabilidad efectivos deben integrar la recogida de datos en el punto de origen, la continuidad del flujo de información a lo largo de toda la cadena y la estandarización de los formatos de intercambio entre actores. Sin embargo, la mayoría de los sistemas implantados en el sector agroalimentario siguen basándose en registros centralizados gestionados por cooperativas o entidades certificadoras, lo que introduce vulnerabilidades en términos de transparencia, interoperabilidad y confianza entre los distintos actores. La manipulación o alteración de los registros supone un riesgo para la fiabilidad de la información almacenada que estos sistemas no pueden mitigar de forma técnica.

En esta misma línea, Ellahi et al. [8] llevan a cabo un análisis de más de 1.500 publicaciones sobre trazabilidad alimentaria basada en blockchain entre 2016 y 2023, identificando un crecimiento exponencial del interés en la combinación de blockchain con tecnologías IoT como respuesta a las limitaciones estructurales de los sistemas centralizados. Los autores concluyen que los sistemas tradicionales fallan en ofrecer soluciones fiables de trazabilidad ante la creciente exigencia de transparencia y responsabilidad en la cadena alimentaria

global, y que las arquitecturas más completas son precisamente aquellas que integran múltiples tecnologías de la Industria 4.0 junto con blockchain.

Los primeros casos de implementación real a escala industrial han confirmado la viabilidad de esta aproximación. Kamath [9] documentó las pruebas piloto de Walmart con IBM Food Trust para la trazabilidad de carne de cerdo y mango, concluyendo que el tiempo necesario para rastrear el origen de un producto desde el punto de venta se redujo de varios días a segundos gracias al registro en blockchain. Este caso estableció un referente industrial que impulsó la adopción de arquitecturas similares en otros sectores agroalimentarios.

En el ámbito específico del aceite de oliva, Frikha et al. [10] propusieron una prueba de concepto de sistema de trazabilidad blockchain aplicado a la cadena de suministro del sector, demostrando su viabilidad técnica para el registro inmutable de las etapas del proceso productivo. Habashneh et al. [11] ampliaron este enfoque con un caso de estudio real en la industria olivarera, validando la integración de blockchain con tecnologías de identificación automática para garantizar la autenticidad del origen y la calidad del producto. Ambos trabajos coinciden en señalar que la combinación de captura automática de datos en origen con registro inmutable en blockchain es el mecanismo más robusto para abordar los problemas estructurales de fraude y adulteración que caracterizan al mercado del aceite de oliva.

## *2.2 TECNOLOGÍAS IOT APLICADAS A TRAZABILIDAD*

El Internet de las Cosas (IoT) ha emergido como tecnología clave para la automatización y monitorización de procesos en la industria agroalimentaria. Mediante el uso de sensores y dispositivos conectados, es posible recoger datos en tiempo real sobre variables relevantes como el peso de los productos entregados, las condiciones ambientales de almacenamiento o el estado de los lotes a lo largo de la cadena, eliminando la intervención manual y los errores asociados.

Desde el punto de vista arquitectónico, los sistemas IoT aplicados a trazabilidad se estructuran habitualmente en tres capas [12], representadas en la Figura 1. La capa de percepción agrupa los sensores y actuadores físicos responsables de capturar datos del entorno real: básculas, sensores de temperatura y humedad, lectores RFID o dispositivos de apertura. La capa de red es responsable de la transmisión de los datos capturados hasta los sistemas de procesamiento, utilizando tecnologías de comunicación como WiFi, LoRa, ZigBee, NB-IoT o Bluetooth según los requisitos de alcance, consumo energético y ancho de banda de la aplicación. La capa de aplicación integra los datos recibidos en plataformas de gestión, bases de datos o sistemas de trazabilidad, donde son procesados, almacenados y puestos a disposición de los distintos actores de la cadena.


*Figura 1. Arquitectura típica de un sistema IoT aplicado a la trazabilidad*

En cuanto a los protocolos de comunicación, el estándar MQTT (Message Queuing Telemetry Transport) es uno de los más utilizados en sistemas IoT industriales, por su bajo consumo de ancho de banda, su cabecera mínima de dos bytes y su modelo de publicación y suscripción, que desacopla al dispositivo sensor del sistema receptor [13]. En este modelo, los dispositivos publican datos en tópicos, entendidos como canales lógicos que organizan los mensajes dentro del sistema. Estos son gestionados por un broker central, es decir, un

servidor intermediario al que se suscriben los sistemas interesados en recibir dicha información. Esta arquitectura resulta especialmente adecuada para entornos donde los sensores tienen capacidades de cómputo limitadas o donde la conectividad de red no es continua, como ocurre frecuentemente en entornos agrícolas o de transporte. Protocolos alternativos como CoAP (Constrained Application Protocol) ofrecen un modelo de petición-respuesta similar a HTTP sobre UDP, siendo preferibles en escenarios donde se requiere transferencia de datos en tiempo real con latencia mínima y recursos computacionales muy restringidos.

La adopción de tecnologías IoT en el sector agroalimentario ha aumentado de forma progresiva en los últimos años. Alimadani et al. [12] señalan como aplicaciones habituales del IoT en el sector alimentario la identificación y trazabilidad de productos mediante RFID, la monitorización de variables ambientales como la temperatura y la humedad, el seguimiento de condiciones de cultivo y el uso de sistemas automatizados de medición y registro en procesos de producción y gestión de residuos. Sin embargo, la dependencia de estos sistemas de infraestructuras centralizadas de almacenamiento, habitualmente basadas en la nube, puede introducir riesgos relacionados con la integridad y el control a largo plazo de los datos registrados. Esta limitación motiva la combinación del IoT con blockchain, donde el primero aporta la capacidad de captura automatizada de datos procedentes del entorno físico y el segundo proporciona un mecanismo de registro distribuido, verificable e inmutable.

## 2.3 *FUNDAMENTOS DE BLOCKCHAIN*

La tecnología blockchain fue introducida por Satoshi Nakamoto en 2008 con la publicación de "Bitcoin: A Peer-to-Peer Electronic Cash System" [14], en el que se proponía un sistema de pagos electrónicos descentralizado entre pares. El problema central que resolvía era el del doble gasto: a diferencia del dinero físico, un activo digital puede copiarse y enviarse a múltiples destinatarios simultáneamente, lo que en los sistemas tradicionales requería de un intermediario central (como un banco) que validara cada transacción y garantizara que el mismo activo no se gastaba dos veces. Nakamoto resolvió este problema sin ningún

intermediario, sustituyendo la confianza en una entidad central por un mecanismo criptográfico de consenso distribuido entre todos los nodos de la red. La innovación fundamental no era la criptografía en sí misma, ya empleada previamente, sino su combinación en una estructura de registro distribuido que hace computacionalmente inviable la alteración del historial de transacciones.

Una red blockchain puede entenderse como un registro distribuido e inmutable en el que las transacciones se agrupan en bloques enlazados criptográficamente. Cada bloque contiene un conjunto de transacciones, un timestamp, un nonce y el hash SHA-256 del bloque anterior, formando una cadena en la que cualquier modificación en un bloque invalida todos los posteriores [15]. Esta estructura garantiza que alterar un registro histórico requeriría recalcular ese bloque y todos los siguientes con una potencia de cómputo superior a la del resto de la red combinada, lo que en la práctica lo hace irrefutable. La Figura 2 ilustra la estructura de encadenamiento de bloques mediante hashes.

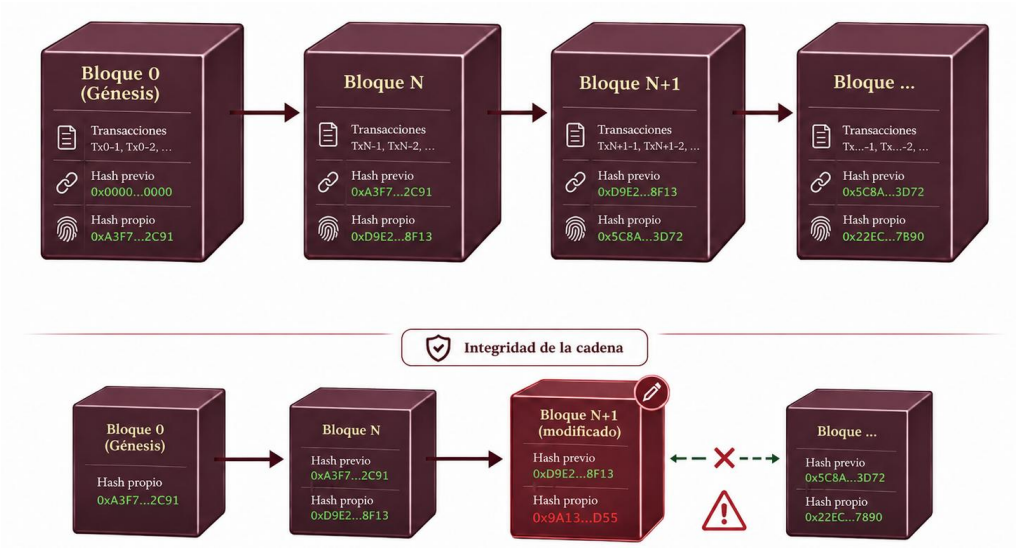


*Figura 2. Estructura de la cadena de bloques.*

Zheng et al. [11] clasifican las redes blockchain en tres categorías según su modelo de acceso. Las blockchains públicas o permissionless, como Bitcoin o Ethereum, permiten la participación de cualquier nodo sin restricciones, ofreciendo máxima descentralización a costa de limitaciones en rendimiento y costes de transacción variables. Las blockchains

privadas o permisionadas, como Hyperledger Fabric, restringen el acceso a nodos autorizados, logrando mayor rendimiento y control a cambio de menor descentralización. Las redes blockchains de consorcio, gestionadas por un conjunto de organizaciones, son habituales en aplicaciones industriales de cadena de suministro, donde se busca combinar la colaboración entre varias entidades con un mayor control sobre el acceso y la validación de la información.

Ethereum, la plataforma blockchain utilizada en este proyecto, fue propuesta por Vitalik Buterin en 2013 [16] y lanzada en 2015. A diferencia de Bitcoin, diseñado exclusivamente para el registro de transacciones económicas, Ethereum incorpora la Ethereum Virtual Machine (EVM), un entorno virtual que permite ejecutar contratos inteligentes dentro de la propia red blockchain, asegurando que todos los nodos obtengan el mismo resultado al procesar las mismas instrucciones.

## *2.4 SMART CONTRACTS PARA LA GESTIÓN DE EVENTOS DE TRAZABILIDAD*

El concepto de smart contract fue propuesto por Nick Szabo en 1994 para describir protocolos que ejecutan automáticamente los términos de un acuerdo cuando se cumplen las condiciones codificadas en ellos, sin necesidad de un intermediario de confianza. Con la aparición de Ethereum y la EVM, este concepto pasó de ser teórico a ser implementable de forma general y programable sobre una infraestructura distribuida [17].

En el contexto de Ethereum, los contratos inteligentes son programas escritos en Solidity que se compilan a bytecode EVM y se despliegan en una dirección única de la red. Una vez desplegados, su código es inmutable y su ejecución es determinista: dado el mismo estado inicial y los mismos parámetros de entrada, todos los nodos de la red producen exactamente el mismo resultado.

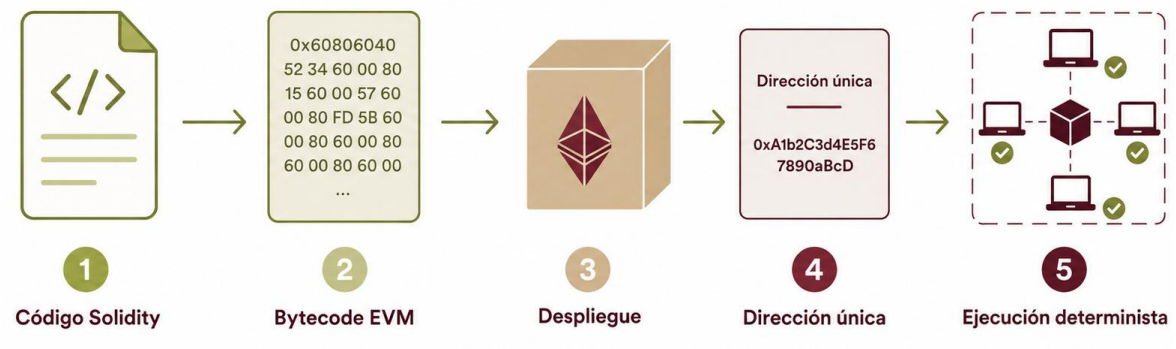


*Figura 3. Ciclo de vida de un smart contract en Ethereum*

Christidis y Devetsikiotis [17] identificaron en 2016 los smart contracts como el mecanismo habilitador clave para la integración de blockchain con sistemas IoT, precisamente por esta capacidad de ejecutar lógica de negocio de forma automática, transparente y sin intermediarios.

Solidity es el lenguaje de programación dominante para el desarrollo de smart contracts sobre Ethereum y redes compatibles con la EVM. Es un lenguaje de alto nivel, estáticamente tipado, con una sintaxis similar a C++ y JavaScript.

## *2.5 ANÁLISIS COMPARATIVO DE SOLUCIONES EXISTENTES*

El análisis de los trabajos existentes permite identificar cinco grandes enfoques tecnológicos para la trazabilidad en cadenas de suministro agroalimentarias, con características, ventajas y limitaciones bien diferenciadas [8].

Los sistemas centralizados tradicionales constituyen el enfoque más extendido en el sector. Basados en registros en papel o en bases de datos relacionales gestionadas por un único actor, ofrecen simplicidad de implantación y bajo coste inicial. Ejemplos representativos en el sector del aceite de oliva son los sistemas ERP de cooperativas como SAP Agro o plataformas sectoriales como Olimerca. Su limitación fundamental es estructural: dependen de la confianza en el actor que controla el registro, son vulnerables a modificaciones no autorizadas y hacen costosa la auditoría externa.

Las plataformas digitales avanzadas integran sistemas ERP, aplicaciones web y bases de datos distribuidas para mejorar la visibilidad de la cadena. Aunque suponen una mejora respecto al registro en papel, siguen siendo arquitecturas centralizadas que no resuelven el problema de fondo: la integridad de los datos depende de la honestidad del operador del sistema y no puede ser verificada de forma independiente por terceros.

Las soluciones IoT puras mejoran significativamente la precisión y la automatización de la captura de datos, eliminando la intervención manual en el registro de variables físicas. Sin embargo, al depender de infraestructuras centralizadas de almacenamiento en la nube, reintroducen los problemas de integridad a largo plazo propios de los sistemas anteriores. La calidad del dato en el origen mejora, pero no su inmutabilidad una vez registrado.

Las soluciones blockchain puras eliminan la dependencia de un actor central mediante un registro distribuido e inmutable. Su principal ventaja es precisamente esa: ningún actor puede alterar un registro sin el consenso del resto de la red. Sin embargo, presentan limitaciones notables en cuanto a escalabilidad, coste de almacenamiento en cadena y capacidad para capturar datos del mundo físico de forma automatizada, ya que por sí solas no disponen de mecanismos para integrar sensores o dispositivos externos.

Las arquitecturas híbridas IoT-blockchain representan la tendencia dominante en la literatura científica reciente [8]. Bajo este enfoque, los dispositivos IoT capturan datos en el entorno físico y blockchain actúa como capa de registro inmutable para los eventos clave, mientras que una base de datos local centralizada actúa como caché de consulta rápida. Este diseño reduce la carga sobre la red blockchain, almacenando en cadena únicamente los hashes o los eventos críticos y no la totalidad de los datos, y mantiene al mismo tiempo un alto nivel de confianza y transparencia, puesto que cualquier actor puede verificar la correspondencia entre los datos locales y los registros on-chain. El análisis realizado por Ellahi et al. [8] confirma que este patrón de diseño es el más adoptado en implementaciones reales de trazabilidad agroalimentaria publicadas entre 2019 y 2023.

## *2.6 PROYECTOS DE REFERENCIA Y CASOS DE USO INDUSTRIALES*

La viabilidad técnica de las arquitecturas IoT-blockchain para trazabilidad alimentaria ha sido demostrada por un conjunto de iniciativas industriales y académicas que sirven como referencia directa para el presente trabajo.

### 2.6.1 IBM FOOD TRUST

El caso más representativo a escala industrial es IBM Food Trust, desarrollado sobre Hyperledger Fabric en colaboración con Walmart, Nestlé y Unilever. En 2016, Walmart comenzó dos proyectos piloto para la trazabilidad de carne de cerdo en China y mango en Estados Unidos. Los resultados fueron determinantes: el tiempo necesario para rastrear el origen de un mango se redujo de casi siete días a 2,2 segundos [9]. A raíz de estos resultados, la plataforma se extendió a más de 25 productos de múltiples proveedores y, en septiembre de 2018, Walmart exigió a todos sus suministradores de verduras de hoja verde la adhesión obligatoria antes de finales de 2019 [18].

Desde el punto de vista técnico, IBM Food Trust se basa en Hyperledger Fabric, un framework de blockchain permisionado de código abierto gestionado por la Linux Foundation. A diferencia de las redes públicas como Ethereum, Hyperledger Fabric no requiere criptomoneda ni mecanismos de consenso intensivos en cómputo: los participantes son entidades conocidas y autorizadas, y el propietario de cada dato controla quién puede acceder a él. Esta arquitectura resulta adecuada para entornos empresariales donde la confidencialidad comercial entre competidores es un requisito, pero introduce una limitación estructural: la dependencia de una plataforma propietaria con un coste de implantación elevado, lo que la hace inaccesible para actores de menor tamaño como cooperativas agrícolas independientes.

### 2.6.2 TRAZABILIDAD BLOCKCHAIN EN EL SECTOR DEL ACEITE DE OLIVA

En el ámbito específico del aceite de oliva, varios trabajos académicos han explorado la aplicación de blockchain para abordar los problemas de fraude y autenticidad del sector. Frikha et al. [10] propusieron un sistema basado en Ethereum con smart contracts para garantizar la trazabilidad de la producción en Túnez, demostrando la aplicabilidad del enfoque en un contexto mediterráneo similar al del presente trabajo. Más recientemente, Vitaskos et al. [13] presentaron un sistema IoT-blockchain específico para el sector del aceite de oliva que integra sensores de calidad con una aplicación web vinculada a Ethereum, con resultados positivos en cuanto a transparencia y reducción del riesgo de fraude. A diferencia del enfoque propuesto en este trabajo, su implementación se centra en la verificación de calidad del producto final, sin cubrir el proceso de extracción completo desde la recolección de la aceituna hasta la obtención del aceite ni la certificación inmutable de cada etapa intermedia del proceso productivo.

### 2.6.3 AGRIDIGITAL

AgriDigital es una plataforma australiana fundada en 2015 que utiliza blockchain para la gestión de contratos y la trazabilidad en la cadena del grano. En diciembre de 2016 ejecutó el primer asentamiento mundial de una transacción de materia prima agrícola sobre blockchain: la venta de 23,46 toneladas de trigo liquidada mediante smart contracts sobre una red privada Ethereum. El contrato inteligente valoró la entrega, verificó la disponibilidad de fondos del comprador y ejecutó la transferencia de propiedad y pago de forma simultánea y automática, eliminando el riesgo de contrapartida habitual en el sector. Desde entonces ha procesado más de 1,6 millones de toneladas de mercancía por un valor de 360 millones de dólares, demostrando la escalabilidad del enfoque en un sector con altos volúmenes de transacciones y múltiples actores distribuidos geográficamente [19].

### 2.6.4 POSICIONAMIENTO DEL PRESENTE TRABAJO

El análisis del estado del arte pone de manifiesto que la combinación de tecnologías IoT y blockchain representa la aproximación más prometedora para abordar los problemas

estructurales de trazabilidad en la cadena agroalimentaria. La literatura académica ha establecido los fundamentos conceptuales de estas arquitecturas híbridas, los casos de uso industriales han demostrado su viabilidad a escala real, y los trabajos específicos sobre el sector del aceite de oliva confirman que el problema del fraude y la falta de transparencia puede abordarse con herramientas tecnológicas maduras y disponibles.

Sin embargo, la revisión de la literatura revela que la mayoría de las soluciones existentes se centran en una parte del problema: la certificación del producto final, la gestión de pagos o la trazabilidad de una etapa concreta, sin cubrir el flujo completo desde la captura automática del dato físico mediante sensores hasta su registro inmutable en blockchain y su consulta desde una interfaz web. La integración de todos estos componentes en una arquitectura cohesionada y reproducible aplicada al sector del aceite de oliva constituye el objetivo de este trabajo, desarrollado en los capítulos siguientes.

# Capítulo 3. DESCRIPCIÓN DE LAS TECNOLOGÍAS

Este capítulo describe las tecnologías seleccionadas para el desarrollo del sistema y justifica cada elección frente a las alternativas consideradas. La selección se realizó atendiendo a tres criterios principales: la madurez y estabilidad de cada tecnología en el momento del desarrollo, la compatibilidad entre los distintos componentes del stack y la adecuación al objetivo académico del proyecto. Para cada tecnología se describe brevemente qué es y qué problema resuelve, a fin de proporcionar el contexto necesario para comprender las decisiones de diseño desarrolladas en los capítulos posteriores

## *3.1 BLOCKCHAIN: ETHEREUM Y GANACHE*

Blockchain es una tecnología de registro distribuido que permite almacenar información de forma compartida, segura y difícilmente modificable. Su funcionamiento se basa en la agrupación de los datos en bloques, que se enlazan entre sí mediante mecanismos criptográficos para formar una cadena ordenada. Cada bloque contiene información registrada, como transacciones o eventos, junto con una referencia al bloque anterior, lo que permite garantizar la continuidad e integridad del sistema.

En este contexto, el hash desempeña un papel esencial como mecanismo de verificación. Un hash es una huella criptográfica generada a partir de unos datos concretos, de forma que cualquier cambio, por pequeño que sea, produce un resultado completamente distinto, como se ilustra en la Figura 4. Las funciones hash son unidireccionales: es computacionalmente inviable reconstruir los datos originales a partir del hash. Ethereum utiliza la función criptográfica Keccak-256 para generar identificadores dentro de la red, como el hash de los bloques, las direcciones de cuenta y los selectores de función de los contratos inteligentes.

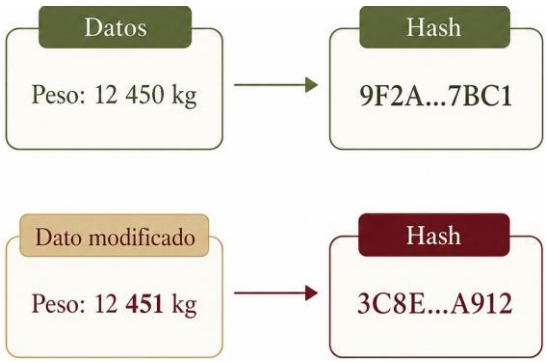


*Figura 4. Alteración de datos y hash asociado.*

Gracias a esta propiedad, la red blockchain puede detectar alteraciones en la información registrada: si se modificara el contenido de un bloque, cambiaría su hash y se rompería la relación con los bloques posteriores. Por ello, el hash permite reforzar la integridad de la cadena y convierte cualquier intento de manipulación en una alteración detectable.

### 3.1.1 QUÉ ES ETHEREUM Y CÓMO FUNCIONA

Ethereum es una plataforma blockchain programable que permite no solo transferir valor entre cuentas, sino también ejecutar código dentro de la propia red mediante contratos inteligentes. Su funcionamiento se basa en un modelo de cuentas, en el que se distinguen dos tipos principales: las cuentas externas, controladas por claves privadas, y las cuentas de contrato, que contienen código ejecutable y almacenan un estado propio dentro de la blockchain. Cuando un usuario o sistema externo desea interactuar con Ethereum, envía una transacción a la red. Esta transacción puede consistir en una transferencia simple de ETH o en una llamada a una función de un contrato inteligente desplegado previamente. En este segundo caso, la transacción activa la ejecución del código del contrato en la EVM, que actúa como entorno común de ejecución para todos los nodos de la red. Gracias a este entorno, las mismas instrucciones producen el mismo resultado en todos los nodos, siempre que se parta del mismo estado inicial y de los mismos datos de entrada.

Una característica fundamental de Ethereum es que la ejecución de cada operación tiene un coste asociado, denominado gas, que mide la cantidad de trabajo computacional que requiere una transacción o una instrucción ejecutada por la EVM. Operaciones sencillas, como una transferencia, consumen una cantidad limitada de gas, mientras que operaciones más complejas, como escribir datos en el almacenamiento de un contrato, requieren un coste mayor. Este mecanismo evita que se ejecuten operaciones excesivamente costosas o infinitas, ya que toda transacción debe incluir un límite máximo de gas que el usuario está dispuesto a pagar.

El coste final de una transacción depende, por tanto, del gas consumido y del precio del gas en ese momento. Esta característica condiciona directamente el diseño de los smart contracts, ya que obliga a optimizar el código y a decidir cuidadosamente qué información debe almacenarse directamente en la red blockchain. En aplicaciones de trazabilidad, esto resulta especialmente relevante: no es eficiente guardar grandes volúmenes de datos en la cadena, sino únicamente la información esencial, como identificadores, estados, direcciones autorizadas o hashes que permitan verificar la integridad de datos almacenados externamente.

El estado de Ethereum se actualiza cada vez que una transacción válida es incluida en un bloque. En el caso de los smart contracts, esta actualización puede implicar cambios en variables internas del contrato, emisión de eventos o registro de nuevas relaciones entre entidades. De este modo, Ethereum funciona como una máquina de estados distribuida (Figura 5): cada bloque contiene un conjunto ordenado de transacciones que transforman el estado anterior de la red en un nuevo estado compartido y aceptado por todos los nodos.

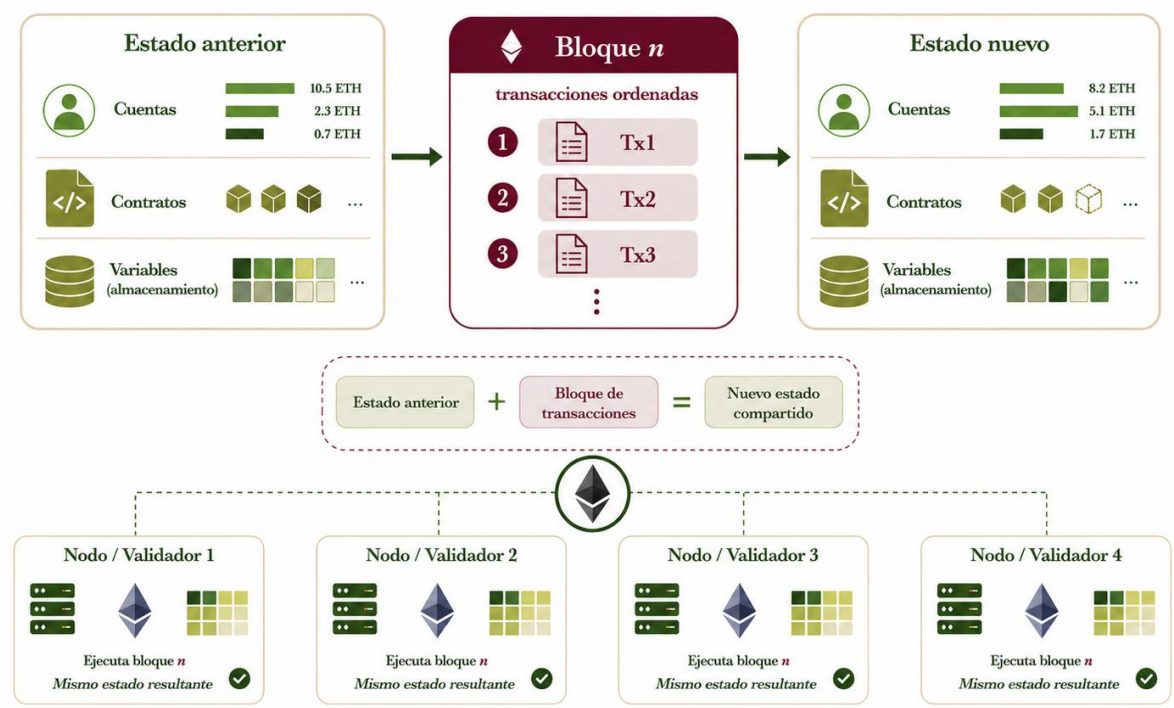


*Figura 5. Ethereum como máquina de estados distribuida.*

Actualmente, Ethereum utiliza un mecanismo de consenso basado en Proof of Stake, en el que los validadores participan en la creación y validación de bloques mediante el bloqueo de ETH como garantía. Este mecanismo permite que la red alcance consenso sobre el orden de las transacciones y el estado resultante, manteniendo la seguridad del sistema sin recurrir al elevado consumo energético característico de los mecanismos basados en Proof of Work [15].

### 3.1.2 POR QUÉ ETHEREUM Y NO OTRAS PLATAFORMAS BLOCKCHAIN

La elección de Ethereum como plataforma blockchain para este proyecto se fundamenta en los tres criterios mencionados anteriormente:

- Hyperledger Fabric, una de las principales alternativas en entornos empresariales, fue descartada debido a que su modelo de red permisionada requiere una infraestructura específica basada en nodos, canales y organizaciones. Esta

arquitectura introduce una complejidad de configuración que no aporta un valor diferencial significativo para un prototipo académico.

- Las redes compatibles con la EVM como Polygon o BNB Chain fueron igualmente descartadas por no ofrecer ventajas adicionales frente a Ethereum para un entorno de desarrollo local.
- El despliegue en Ethereum mainnet o en testnets públicas como Sepolia fue descartado por introducir dependencia de infraestructura externa no controlable y disponibilidad variable de tokens de prueba.

### 3.1.3 GANACHE COMO ENTORNO DE DESARROLLO LOCAL

Ganache es una herramienta de desarrollo que permite desplegar una red Ethereum local de un único nodo, facilitando la prueba de contratos inteligentes y transacciones en un entorno controlado. Desarrollada originalmente por Truffle Suite y mantenida actualmente por la comunidad, Ganache simula el comportamiento completo de la red Ethereum en el entorno local sin necesidad de conectividad a internet ni de ETH real.

Las características que hacen de Ganache la herramienta adecuada para este proyecto son las siguientes:

- Minado instantáneo: cada transacción se incluye en un bloque de forma inmediata, eliminando las esperas de aproximadamente doce segundos propias de la red principal y haciendo las pruebas ágiles y reproducibles.
- Cuentas prefinanciadas: Ganache genera automáticamente diez cuentas con cien ETH de prueba cada una, con sus claves privadas exportables, lo que permite firmar transacciones sin gestión de fondos reales.

Es importante destacar que la elección de Ganache no limita la validez técnica del sistema. El smart contract desarrollado es código Solidity estándar compatible con cualquier red EVM. La librería Web3j, utilizada en el backend para la comunicación con la red blockchain, funciona de forma idéntica apuntando a Ganache, a una testnet pública o a la red principal de Ethereum. El único cambio necesario para desplegar el sistema en un entorno de

producción sería actualizar la URL del nodo y el identificador de cadena en el fichero de configuración del backend.

## *3.2 SMART CONTRACTS: SOLIDITY Y HARDHAT*

### 3.2.1 QUÉ SON LOS SMART CONTRACTS Y SOLIDITY

En este proyecto, los smart contracts se emplean como el componente encargado de registrar y validar los eventos principales de trazabilidad dentro de la red Ethereum. Un smart contract puede entenderse como un programa desplegado en la blockchain que se ejecuta cuando una transacción invoca alguna de sus funciones. Una vez publicado, queda asociado a una dirección única y su ejecución pasa a formar parte del estado compartido de la red.

El contrato no actúa como una base de datos convencional, sino como una capa de registro verificable donde se almacenan los datos esenciales de cada operación y se emiten eventos que permiten reconstruir el historial asociado a cada lote.

La versión utilizada en el proyecto es Solidity 0.8.24, compatible con el entorno de desarrollo Hardhat. Esta versión incorpora comprobaciones automáticas frente a desbordamientos aritméticos, lo que mejora la seguridad del contrato sin necesidad de añadir librerías externas específicas para este propósito.

### 3.2.2 QUÉ ES HARDHAT Y POR QUÉ SE ELIGIÓ FRENTE A TRUFFLE

Hardhat es el entorno de desarrollo de smart contracts más utilizado en el ecosistema Ethereum, desarrollado por Nomic Foundation. Proporciona un compilador Solidity integrado que gestiona múltiples versiones de forma transparente, una red de desarrollo local propia como alternativa a Ganache, y un sistema de plugins para compilación, testing, despliegue, análisis de cobertura de código y generación de informes de consumo de gas. A efectos de este proyecto, se utiliza exclusivamente como compilador y como herramienta de despliegue automatizado en Ganache.

La alternativa principal considerada fue Truffle, el entorno de desarrollo que precedió a Hardhat como estándar del ecosistema. Truffle fue descartado por dos razones. La primera es que su mantenimiento activo ha decaído significativamente desde 2022, con actualizaciones esporádicas y compatibilidad reducida con las versiones más recientes de Node.js. La segunda es que Hardhat ofrece mensajes de error notablemente más informativos, incluyendo stack traces del código Solidity cuando una transacción revierte, lo que facilita el diagnóstico de errores durante el desarrollo.

## *3.3 BACKEND: SPRING BOOT 3 Y JAVA 21*

### 3.3.1 QUÉ ES SPRING BOOT Y CÓMO FUNCIONA

Spring Boot es el framework de referencia en el ecosistema Java para el desarrollo de aplicaciones backend, construido sobre el framework Spring. Su propuesta de valor fundamental es la convención sobre configuración: en lugar de exigir al desarrollador que configure explícitamente cada componente de la aplicación, Spring Boot establece valores por defecto sensatos y configura automáticamente los componentes necesarios en función de las dependencias presentes en el proyecto. Esta filosofía, conocida como autoconfiguration, permite tener una API REST funcional con muy pocas líneas de código de configuración explícita.

El corazón de Spring es el contenedor de inversión de control (IoC), que gestiona el ciclo de vida de los objetos de la aplicación. En lugar de que cada clase instancie sus dependencias directamente, declara qué necesita y Spring se encarga de proporcionarlo. Este patrón, conocido como inyección de dependencias, desacopla los componentes entre sí y facilita tanto la sustitución de implementaciones como la escritura de pruebas. En el contexto de este proyecto, esta capacidad es especialmente relevante: los módulos de blockchain y MQTT son dependencias opcionales que pueden activarse o desactivarse sin modificar el código de la lógica de negocio, simplemente cambiando una propiedad de configuración.

### 3.3.2 POR QUÉ SPRING BOOT FRENTE A OTRAS ALTERNATIVAS

Las alternativas consideradas para el backend fueron Quarkus y Micronaut, dos frameworks Java modernos orientados a tiempos de arranque rápidos y bajo consumo de memoria mediante compilación nativa con GraalVM. Ambos fueron descartados por la misma razón: la integración con Web3j 4.12.2, la librería utilizada para la comunicación con Ethereum presenta compatibilidades no documentadas con la compilación nativa de GraalVM que introducen una complejidad de configuración innecesaria para un prototipo académico. Spring Boot, al ejecutarse sobre la JVM estándar, no presenta estos problemas y ofrece un ecosistema de integraciones más maduro para los componentes del proyecto, incluyendo Spring Data JPA para la persistencia y Spring Validation para la validación de entrada.

### 3.3.3 ARQUITECTURA EN CAPAS Y DECISIONES DE DISEÑO RELEVANTES

El backend sigue la arquitectura en capas estándar de Spring, con tres niveles de responsabilidad claramente separados, como se ilustra en la Figura 6.

- La capa de API, compuesta por los controladores y los DTOs (*Data Transfer Objects*), es la única que conoce el protocolo HTTP y se encarga de traducir las peticiones recibidas en llamadas al servicio, así como los resultados del servicio en respuestas JSON. En este contexto, los DTOs son objetos de transferencia de datos que definen qué información entra y sale de la API, sin exponer directamente las entidades internas del sistema.
- La capa de dominio concentra la lógica de negocio en el servicio central, que orquesta la creación de lotes, el registro de eventos y la consulta de trazabilidad.
- La capa de infraestructura agrupa las integraciones con sistemas externos, concretamente la red blockchain y el broker MQTT, de modo que, si en el futuro se sustituyera Ganache por otro nodo Ethereum o Mosquitto por otro broker, solo sería necesario modificar esta capa.

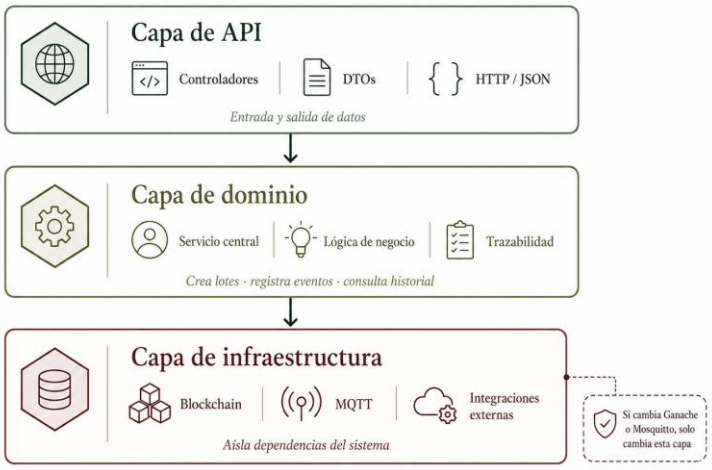


*Figura 6. Arquitectura en capas del backend.*

Una decisión de diseño relevante es que las conexiones con blockchain y MQTT se han configurado como componentes opcionales del sistema. En Spring, estos componentes se gestionan como beans, es decir, objetos creados y administrados automáticamente por el propio framework para que puedan ser utilizados por otras partes de la aplicación. Mediante la anotación `@ConditionalOnProperty`, Spring solo crea los beans relacionados con blockchain y MQTT cuando las propiedades `blockchain.enabled` y `mqtt.enabled` están activadas en el fichero de configuración. Si alguna de estas propiedades está desactivada, el componente correspondiente no se crea y el servicio continúa funcionando sin esa integración. De este modo, el sistema puede arrancar correctamente, aunque Ganache o Mosquitto no estén disponibles, manteniendo las funcionalidades principales y desactivando únicamente las capacidades opcionales asociadas a blockchain o MQTT.

### 3.3.4 ESTRATEGIA DE PERSISTENCIA: H2 COMO CACHÉ DE BLOCKCHAIN

La persistencia de datos en el backend se gestiona mediante Spring Data JPA sobre una base de datos H2 en memoria. H2 es una base de datos relacional escrita en Java que se embebe directamente en la JVM, sin necesidad de instalación ni proceso externo. En desarrollo, funciona en modo “en memoria”, lo que significa que los datos se pierden al reiniciar el backend, un comportamiento coherente con el de Ganache, que también pierde su estado al reiniciarse.

La decisión de usar H2 como caché local de los eventos blockchain responde a tres razones principales:

- Velocidad: las consultas de trazabilidad se resuelven en milisegundos desde H2, mientras que consultar directamente los logs de Ganache mediante `eth_getLogs` puede ser significativamente más lento en contratos con un número elevado de eventos.
- Simplicidad del frontend: gracias a H2, el frontend solo necesita realizar una petición HTTP REST al backend para obtener el historial completo de un lote. De este modo, no es necesario integrar librerías de Ethereum ni conocer el ABI (Application Binary Interface) del contrato, es decir, la interfaz que describe qué funciones y eventos expone el smart contract y cómo deben invocarse o interpretarse desde una aplicación externa.
- Resiliencia: si Ganache no está disponible en el momento de una consulta, el historial sigue siendo accesible desde H2, lo que permite mantener la consulta de trazabilidad, aunque la red blockchain local no esté operativa.

El vínculo entre ambas capas de persistencia es el hash de transacción. Cada vez que un evento se registra en Ganache, el backend guarda en H2 el txHash devuelto por la red junto con los datos del evento. Cualquier actor puede usar ese hash para verificar de forma independiente en Ganache que el dato almacenado en H2 es auténtico y no ha sido alterado.

El proyecto incluye también el driver de PostgreSQL en el fichero de dependencias, lo que permite migrar a una base de datos persistente en producción simplemente actualizando la configuración, sin modificar ningún código Java.

## *3.4 INTEGRACIÓN BLOCKCHAIN: WEB3J*

### 3.4.1 QUÉ ES WEB3J

Web3j es una librería de código abierto que permite a aplicaciones Java interactuar con redes Ethereum. En este proyecto actúa como puente entre el backend Spring Boot y el nodo

Ganache: el backend construye transacciones, las firma localmente con una clave privada y las envía a la blockchain sin delegar en ningún momento la custodia de esa clave a un tercero.

### 3.4.2 EL PROTOCOLO JSON-RPC Y LA CODIFICACIÓN ABI

La comunicación entre Web3j y el nodo Ethereum se realiza a través del protocolo JSON-RPC, una interfaz de llamada a procedimiento remoto transportada sobre HTTP en la que se envía un objeto JSON con el nombre del método a invocar y sus parámetros, y el nodo responde con el resultado.

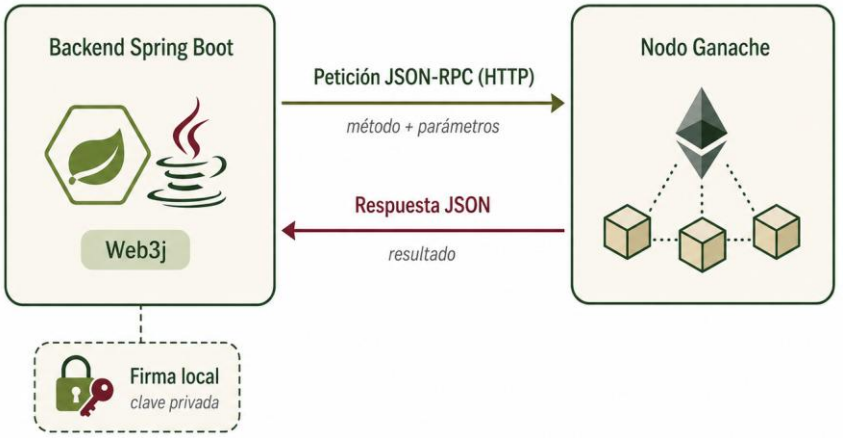


*Figura 7. Protocolo JSON-RPC - flujo de petición y respuesta entre el backend y el nodo Ganache.*

Los métodos relevantes para este proyecto son `eth_getTransactionCount`, que devuelve el nonce de una cuenta; `eth_sendRawTransaction`, que propaga una transacción ya firmada; y `eth_chainId`, que devuelve el identificador de la red.

Para invocar una función de un smart contract, la transacción debe incluir en su campo data la codificación de la llamada siguiendo el estándar ABI de Ethereum, ilustrado en la Figura 8. Los primeros cuatro bytes corresponden al selector de la función, calculado como el hash keccak256 de su firma canónica. A continuación, se añaden los argumentos: los tipos de longitud fija como uint256 se representan como valores de 32 bytes en big-endian, y los tipos dinámicos como string incluyen un offset, la longitud en bytes y el contenido en UTF-8 con padding hasta el siguiente múltiplo de 32 bytes.

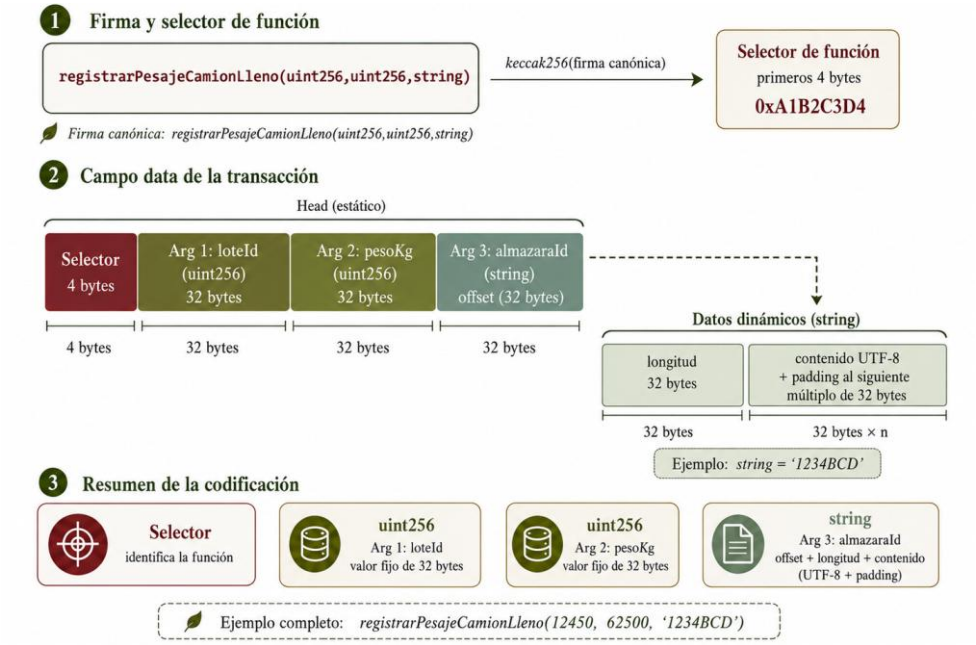


*Figura 8. Codificación ABI del campo data para la función registrarPesajeCamionLleno.*

### 3.4.3 FIRMA DE TRANSACCIONES: ECDSA, RLP Y EIP-155

Las transacciones en Ethereum deben ser firmadas antes de enviarse a la red, ya que la firma permite demostrar que han sido autorizadas por la cuenta emisora sin necesidad de revelar su clave privada. Para ello se utiliza el algoritmo ECDSA sobre la curva elíptica secp256k1, el mismo esquema criptográfico empleado por Ethereum para asociar una clave privada con una dirección de cuenta. Antes de ser firmada, la transacción se serializa mediante RLP (*Recursive Length Prefix*), el formato de codificación utilizado en Ethereum para representar de forma compacta los campos de la transacción. Sobre esa representación codificada se calcula el hash que posteriormente se firma con la clave privada del emisor.

El resultado de la firma se expresa mediante los valores r, s y v. Los dos primeros forman la firma ECDSA propiamente dicha, mientras que v actúa como valor de recuperación, permitiendo identificar la clave pública asociada al firmante a partir de la firma y del mensaje firmado. De esta forma, cualquier nodo puede verificar que la transacción procede realmente de la cuenta indicada y que su contenido no ha sido alterado.

Además, Ethereum incorpora la propuesta EIP-155 para evitar ataques de repetición o replay attacks. Este mecanismo incluye el identificador de red, denominado chainId, dentro de los datos que se firman. Así, una transacción firmada para una red concreta no puede reutilizarse válidamente en otra red compatible con la EVM. En este proyecto, la red local de Ganache utiliza el chainId 1337, valor que se configura en Hardhat y que el backend obtiene dinámicamente del nodo durante el arranque.

### 3.4.4 LAS DOS FORMAS DE USAR WEB3J Y JUSTIFICACIÓN DE LA ELECCIÓN

Web3j ofrece dos formas de interactuar con un smart contract con implicaciones distintas en cuanto a nivel de abstracción, control y dependencias (Figura 9).

La primera es el wrapper generado automáticamente mediante el plugin Maven de Web3j, que a partir del ABI del contrato genera una clase Java con métodos tipados para cada función. Este enfoque es el recomendado por la documentación oficial y el más productivo en proyectos sin restricciones de dependencias. La clase generada extiende `Contract`, perteneciente al artefacto `web3j:contracts`.

La segunda es el uso directo de las APIs primitivas incluidas en `web3j:core:FunctionEncoder` para codificar la llamada ABI, las llamadas JSON-RPC directas para obtener el nonce y enviar la transacción, `RawTransaction` para construir la transacción cruda, y `TransactionEncoder` para firmarla con EIP-155. Este enfoque es más explícito, pero ofrece control total sobre cada paso del proceso.

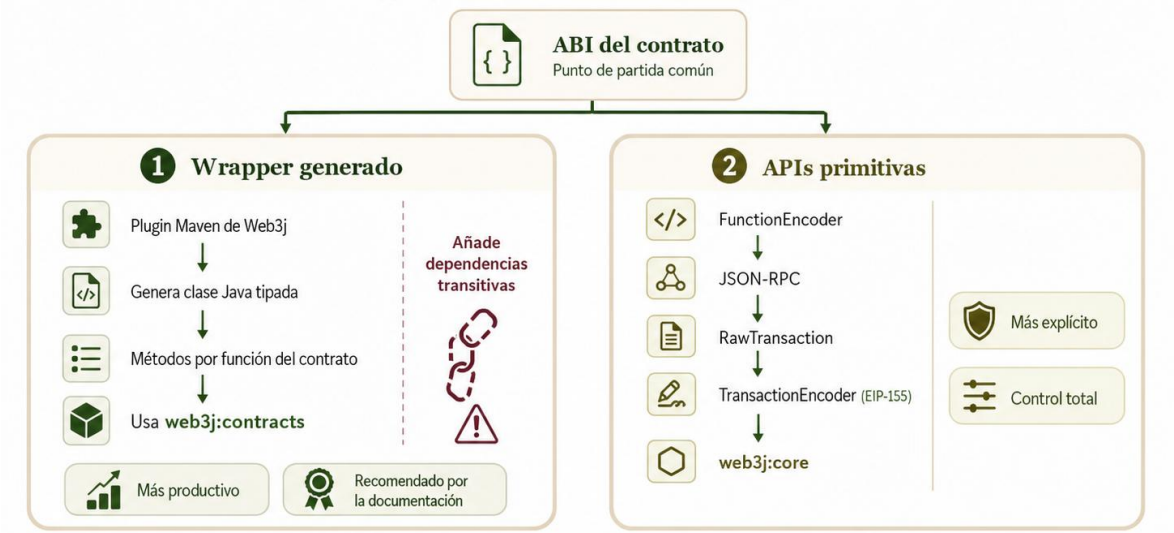


*Figura 9. Comparación de los dos enfoques de integración con Web3j .*

En este proyecto se adoptó esta segunda aproximación por motivos de compatibilidad con el entorno Spring Boot 3. El uso del artefacto `web3j:contracts`, asociado a la generación automática de clases Java para interactuar con el contrato, introducía dependencias transitivas que entraban en conflicto con las versiones de Jackson Databind gestionadas por el BOM (Bill of Materials) de Spring Boot 3. Este BOM actúa como un mecanismo de gestión centralizada de versiones, encargado de definir qué versiones de las distintas dependencias son compatibles dentro del ecosistema de Spring Boot.

Aunque se intentó resolver el conflicto mediante exclusiones explícitas en el fichero pom.xml, la solución no resultó satisfactoria. Por ello, se optó por implementar un wrapper manual, denominado `CadenaAceiteWrapper`, basado exclusivamente en `web3j:core`, eliminando así el conflicto en su origen. Esta decisión permitió mantener una configuración más estable del backend y evitar la incorporación de dependencias adicionales problemáticas.

Además, esta aproximación aporta una ventaja en términos de transparencia técnica. Al construir manualmente las llamadas al contrato, el sistema mantiene un control explícito sobre la codificación ABI, la obtención del nonce, la firma conforme a EIP-155 y el envío de la transacción al nodo. De este modo, cada operación puede auditarse paso a paso, lo que

resulta especialmente coherente con los objetivos de un sistema de trazabilidad basado en blockchain.

## *3.5 PROTOCOLO IOT: MQTT Y ECLIPSE MOSQUITTO*

### 3.5.1 QUÉ ES MQTT Y POR QUÉ EXISTE

MQTT (Message Queuing Telemetry Transport) es un protocolo de mensajería ligero diseñado específicamente para entornos con recursos limitados y conectividad poco fiable. Fue creado en 1999 por Andy Stanford-Clark de IBM y Arlen Nipper para resolver un problema concreto: monitorizar oleoductos a través de conexiones de satélite con un ancho de banda extremadamente reducido, del orden de 9.600 bps, y latencias de varios segundos. El protocolo fue diseñado para ser tan ligero como fuera posible en esas condiciones. Se estandarizó como ISO/IEC 20922 en 2016 y hoy es el estándar de facto en IoT industrial, implementado en millones de dispositivos en todo el mundo.

La característica fundamental que diferencia a MQTT de HTTP es su modelo de comunicación. HTTP sigue el modelo cliente-servidor: el cliente conoce la dirección del servidor, le envía una petición y espera una respuesta. Este modelo implica que el servidor es accesible, tiene una dirección fija y es capaz de gestionar conexiones entrantes. Un sensor IoT industrial, como una báscula de pesaje, no es un servidor: es un dispositivo con recursos de cómputo limitados, posiblemente ubicado en una red privada industrial, que produce datos de forma periódica o en respuesta a eventos físicos. Obligarle a comportarse como un servidor HTTP requeriría que dispusiera de una IP fija accesible desde el exterior, que gestionara conexiones concurrentes y que implementara TLS, capacidades que la mayoría de los dispositivos industriales no tienen o que añadirían una complejidad innecesaria.

MQTT resuelve este problema mediante un modelo de publicación y suscripción (Figura 10). Ningún participante se comunica directamente con otro; todos se comunican a través de un intermediario denominado broker. Los dispositivos que producen datos, llamados publicadores, envían sus mensajes al broker indicando un tópico, que es una cadena de texto jerárquica que actúa como etiqueta de enrutamiento. Los sistemas que consumen esos datos,

llamados suscriptores, se registran en el broker indicando los tópicos que les interesan. El broker recibe los mensajes de los publicadores y los reenvía a todos los suscriptores del tópico correspondiente. Ventajas de este modelo son el desacoplamiento entre productor y consumidor, la asincronía natural y la posibilidad de que múltiples consumidores reciban el mismo mensaje sin que el productor deba conocerlos.

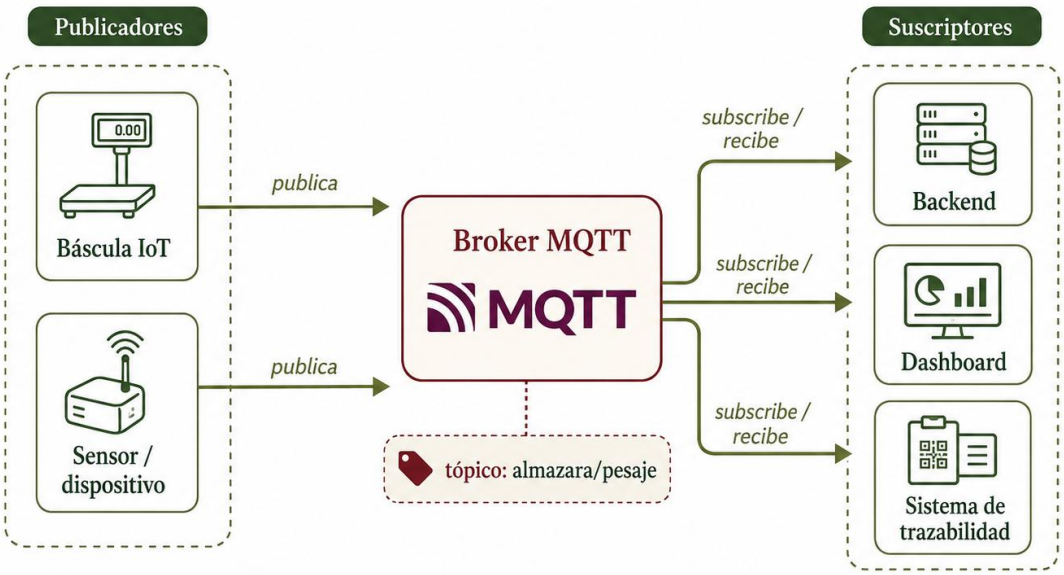


*Figura 10. Modelo publish-subscribe en MQTT.*

### 3.5.2 ECLIPSE MOSQUITTO COMO BROKER MQTT

Eclipse Mosquitto es la implementación de broker MQTT de código abierto más extendida, parte del proyecto Eclipse Foundation. La versión 2, utilizada en este proyecto, implementa los protocolos MQTT 3.1.1 y MQTT 5.0. Mosquitto es ampliamente adoptado tanto en entornos de desarrollo como en entornos de producción embebida por su bajo consumo de recursos, su configuración sencilla y el respaldo de una comunidad activa.

### 3.5.3 ECLIPSE PAHO COMO CLIENTE JAVA

La integración de MQTT en el backend Spring Boot se realiza mediante Eclipse Paho, la librería de cliente MQTT oficial del proyecto Eclipse. La versión Java mqttv3, implementa el protocolo MQTT 3.1.1, compatible con Mosquitto 2. Spring Boot no dispone de un starter

oficial para MQTT puro; la alternativa de Spring Integration MQTT existe, pero introduce conceptos de mensajería propios del framework que añaden complejidad innecesaria para el caso de uso directo que requiere este proyecto.

## *3.6 FRONTEND: REACT, VITE Y TYPESCRIPT*

### 3.6.1 QUÉ SON REACT, VITE Y TYPESCRIPT Y POR QUÉ EXISTEN JUNTOS

El frontend de una aplicación web puede construirse con tecnologías que van desde HTML y JavaScript puro hasta frameworks complejos. La elección del stack condiciona la mantenibilidad del código, la velocidad de desarrollo y la experiencia del usuario. Los tres elementos del stack elegido responden a problemas distintos y se complementan.

- React es una librería de JavaScript orientada a la construcción de interfaces de usuario mediante componentes reutilizables. Su modelo declarativo permite describir la interfaz en función del estado de la aplicación, delegando en la propia librería la actualización eficiente de los elementos visibles en el navegador.
- TypeScript es un superconjunto de JavaScript que incorpora tipado estático. Su objetivo es detectar errores durante el desarrollo y mejorar la mantenibilidad del código, especialmente en aplicaciones donde existen estructuras de datos compartidas entre distintas partes del sistema.
- Vite es una herramienta de desarrollo y construcción para aplicaciones web modernas. Proporciona un entorno de desarrollo rápido y permite generar una versión optimizada de la aplicación para producción.
- Tailwind CSS es un framework de estilos basado en clases de utilidad. Permite definir el aspecto visual de los componentes directamente en el marcado, facilitando la creación de interfaces personalizadas sin depender de componentes prediseñados.

### 3.6.2 POR QUÉ REACT FRENTE A OTRAS ALTERNATIVAS

La elección de React frente a alternativas como Vue.js o Angular se debe a su equilibrio entre flexibilidad, madurez del ecosistema y facilidad de integración. React permite construir

interfaces modulares mediante componentes reutilizables, sin imponer una arquitectura tan rígida como la de Angular ni requerir una configuración excesivamente compleja para el alcance del proyecto.

Angular ofrece una solución completa y robusta, adecuada para aplicaciones empresariales de gran escala, pero introduce una mayor carga conceptual mediante módulos, servicios, decoradores e inyección de dependencias. Vue.js, aunque sencillo y progresivo, cuenta con un ecosistema profesional menos extendido que React. Por ello, React se considera la alternativa más adecuada para desarrollar una interfaz mantenible, escalable y alineada con las necesidades del sistema.

Vite se selecciona como herramienta de desarrollo y construcción por su rapidez, ligereza e integración directa con React. Frente a Create React App, ofrece un entorno de desarrollo más ágil, con tiempos de arranque reducidos y recarga rápida ante cambios en el código.

### 3.6.3 DECISIONES DE DISEÑO DE LA INTERFAZ

La interfaz se organiza en tres rutas principales gestionadas mediante React Router DOM, librería que permite definir la navegación entre las distintas vistas de una aplicación React sin recargar la página completa. En este contexto, el DOM hace referencia a la representación estructurada que el navegador construye de los elementos de la página web y que React actualiza de forma eficiente cuando cambia el estado de la aplicación.

Una decisión técnica relevante en el frontend es el uso de `useRef` como mecanismo de control en el componente de pesaje. En modo estricto de desarrollo, React puede montar los componentes dos veces para detectar efectos secundarios no idempotentes. Sin este control, la solicitud de pesaje al simulador MQTT podría ejecutarse dos veces, generando registros duplicados para un mismo lote. El uso de `useRef` permite conservar una marca interna entre renders sin provocar actualizaciones de la interfaz, garantizando que la operación se lance una sola vez.

## *3.7 INFRAESTRUCTURA: DOCKER*

### 3.7.1 QUÉ ES DOCKER Y CÓMO FUNCIONA

Docker es una plataforma de contenedores que permite empaquetar una aplicación junto con sus dependencias y configuración, de forma que pueda ejecutarse de manera consistente en distintos entornos. A diferencia de una máquina virtual, un contenedor no emula un ordenador completo ni incluye un sistema operativo propio, sino que comparte el kernel del sistema anfitrión y aísla los procesos necesarios para la aplicación. Esto permite reducir el consumo de recursos y acelerar el arranque de los servicios. La configuración de un contenedor se define mediante un Dockerfile, donde se especifican la imagen base, las dependencias, los ficheros que deben copiarse y el comando de ejecución. A partir de este fichero se construye una imagen reutilizable, que sirve como plantilla para crear contenedores.

Docker Compose complementa este proceso permitiendo definir aplicaciones formadas por varios servicios en un único fichero docker-compose.yml. En él se configuran los contenedores necesarios, sus puertos, variables de entorno, volúmenes y dependencias. De este modo, toda la aplicación puede levantarse de forma conjunta mediante un único comando.

### 3.7.2 POR QUÉ DOCKER Y NO UNA MÁQUINA VIRTUAL

La alternativa más directa a Docker para aislar el broker MQTT y el simulador habría sido una máquina virtual, por ejemplo, mediante VirtualBox o VMware. Esta opción fue descartada por tres razones:

- El peso: una máquina virtual con Ubuntu mínimo para ejecutar Mosquitto ocupa entre 3 y 5 GB en disco y requiere asignar de forma fija entre 512 MB y 1 GB de RAM. El contenedor Docker de Eclipse Mosquitto ocupa aproximadamente 12 MB y consume memoria de forma dinámica según la carga real.

- Velocidad de arranque: arrancar una máquina virtual requiere emular el proceso de boot completo del sistema operativo invitado, lo que tarda típicamente entre 30 y 90 segundos. Un contenedor Docker arranca en menos de un segundo porque no hay boot: el proceso simplemente se inicia en el entorno de ficheros ya preparado.
- Reproducibilidad: una máquina virtual garantiza el aislamiento del entorno, pero no garantiza que el software instalado en ella sea exactamente el mismo en dos máquinas distintas, a menos que se distribuya la imagen completa de la VM, que puede pesar decenas de gigabytes.

Con Docker, el `Dockerfile` y el fichero `docker-compose.yml` son ficheros de texto que se versionan en el repositorio junto con el código. Cualquier desarrollador que clone el repositorio y ejecute `docker compose up` obtendrá exactamente el mismo entorno, con la misma versión de Mosquitto y la misma configuración, independientemente del sistema operativo del anfitrión.

### 3.7.3 CÓMO SE UTILIZA EN EL PROYECTO

En este proyecto, Docker Compose se utiliza para levantar los servicios auxiliares necesarios para la comunicación IoT: el broker MQTT Mosquitto y el simulador de dispositivos IoT. Mosquitto se ejecuta a partir de la imagen oficial eclipse-mosquitto:2 y expone el puerto 1883 para permitir la conexión del backend. El simulador IoT se construye mediante un `Dockerfile` basado en python:3.12-slim.

# Capítulo 4. DISEÑO DEL SISTEMA

Este capítulo describe el diseño del sistema desarrollado: los requisitos que debe satisfacer, la arquitectura que los materializa, los actores que interactúan con él y el diseño detallado de cada componente. El objetivo es establecer un marco conceptual completo antes de entrar en los detalles de implementación del capítulo siguiente.

## *4.1 REQUISITOS DEL SISTEMA*

### 4.1.1 REQUISITOS FUNCIONALES

Los requisitos funcionales definen las capacidades que el sistema debe ofrecer a sus usuarios. Se han identificado a partir de los casos de uso de los dos actores principales del sistema: el operario, responsable del registro de eventos a lo largo de toda la cadena de custodia, y el auditor, responsable de la verificación de la trazabilidad de los lotes.

- RF-01 - Creación de lotes con trazabilidad de origen. El sistema debe permitir registrar un nuevo lote de aceituna asociado a un identificador de agricultor, una finca de origen, un identificador de contenedor de recogida, la matrícula del camión de transporte y las coordenadas GPS del contenedor. El identificador legible del lote debe generarse automáticamente con el formato LOT-{año}-{secuencia} y quedar almacenado en la base de datos como identificador único. El lote debe quedar registrado de forma inmutable en blockchain mediante su identificador numérico interno.
- RF-02 - Registro del cierre de compuerta. El sistema debe registrar el cierre de la compuerta del camión de transporte asociado a un lote existente, sellando el evento en blockchain con su timestamp de forma automática.
- RF-03 - Registro de apertura de compuerta no autorizada. El sistema debe permitir registrar una apertura de la compuerta del camión durante el transporte, indicando si es autorizada o no autorizada y la ubicación donde se produjo. Este evento queda

certificado en blockchain como prueba de custodia sin interrumpir el flujo del proceso.

- RF-04 - Registro del pesaje del camión lleno en almazara. El sistema debe solicitar el peso del camión cargado de aceituna a una báscula de entrada de la almazara mediante MQTT y registrar el evento en blockchain.
- RF-05 - Registro del volcado en tolva. El sistema debe registrar el evento de volcado de la aceituna en la tolva de entrada de la almazara, certificando en blockchain el momento en que la carga pasa a custodia de la almazara.
- RF-06 - Registro del pesaje del camión vacío. El sistema debe solicitar el peso del camión vacío a una báscula mediante MQTT y registrar el evento en blockchain, permitiendo calcular el peso neto de aceituna entregada por diferencia.
- RF-07 - Registro del lavado con control de calidad del agua. El sistema debe solicitar automáticamente los parámetros de calidad del agua de lavado a un sensor mediante MQTT, incluyendo temperatura, pH y aptitud del agua, y registrar los datos en blockchain.
- RF-08 - Registro del pesaje en cinta. El sistema debe solicitar el peso de la aceituna sin hojas y limpia en la cinta transportadora mediante MQTT y registrar el evento en blockchain, certificando el peso neto tras el proceso de limpieza.
- RF-9 - Registro de la molienda. El sistema debe solicitar la temperatura del molino al inicio de la molienda mediante MQTT y registrar el evento en blockchain con el timestamp.
- RF-10 - Registro de la temperatura de batido. El sistema debe solicitar la temperatura de la batidora mediante MQTT y registrar el evento en blockchain.
- RF-11 - Registro del decanter. El sistema debe solicitar los datos de separación mediante MQTT, incluyendo los litros de aceite obtenidos y los kilogramos de alpeorujo, y registrar el evento en blockchain.
- RF-12 - Registro de la centrifugadora. El sistema debe solicitar los datos de la centrífuga vertical mediante MQTT, incluyendo las revoluciones por minuto y la temperatura de operación, y registrar el evento en blockchain.

- RF-13 - Registro de la extracción finalizada. El sistema debe solicitar los datos finales de producción mediante MQTT, incluyendo el volumen total de aceite obtenido y el rendimiento del proceso, y registrar el evento en blockchain como cierre de la cadena de trazabilidad del lote.
- RF-14 - Consulta de trazabilidad. El sistema debe permitir consultar el historial completo de eventos de cualquier lote a partir de su identificador, devolviendo la lista de eventos ordenada cronológicamente con el hash de transacción blockchain asociado a cada uno.
- RF-15 - Verificación on-chain. El sistema debe exponer el hash de transacción de cada evento de forma que cualquier actor pueda verificar de forma independiente la autenticidad del registro consultando directamente el nodo blockchain.
- RF-16 - Degradación elegante. El sistema debe continuar funcionando y registrando eventos en la base de datos local cuando el nodo blockchain o el broker MQTT no estén disponibles, indicando en la respuesta que el registro on-chain no ha podido completarse.
- RF-17 - Fallback manual para sensores IoT. Para cada endpoint de solicitud IoT, el sistema debe ofrecer un endpoint alternativo de introducción manual del dato, garantizando la continuidad del servicio cuando el sensor no está disponible.

### 4.1.2 REQUISITOS NO FUNCIONALES

Los requisitos no funcionales definen las restricciones de calidad, rendimiento y operación del sistema.

- RNF-01 - Reproducibilidad. El entorno completo del sistema debe poder levantarse en cualquier máquina con Docker instalado mediante un único comando, sin dependencias de configuración manual del sistema operativo anfitrión.
- RNF-02 - Rendimiento en consultas. Las consultas de trazabilidad deben resolverse en menos de un segundo, independientemente del estado del nodo blockchain, mediante el uso de la base de datos local H2 como caché de eventos.

- RNF-03 - Inmutabilidad verificable. Cada evento registrado debe disponer de un hash de transacción que permita su verificación independiente en el nodo blockchain, sin necesidad de confiar en el backend como intermediario.
- RNF-04 - Configurabilidad. Los parámetros críticos del sistema, como la dirección del contrato, la URL del nodo blockchain, la URL del broker MQTT y el timeout del sensor, deben ser configurables externamente sin modificar el código fuente.
- RNF-05 - Resiliencia. El sistema debe arrancar y operar correctamente, aunque el nodo blockchain o el broker MQTT no estén disponibles en el momento del arranque, degradando las funcionalidades dependientes de esos servicios sin interrumpir el resto.
- RNF-06 - Coherencia física de los datos simulados. Los datos generados por el simulador IoT deben respetar la coherencia física del proceso: el peso del camión vacío debe ser siempre inferior al peso del camión lleno, el peso en cinta debe ser inferior al peso inicial, y los rangos de temperatura de cada sensor deben corresponderse con los valores reales del proceso industrial.
- RNF-07 - Trazabilidad de errores. Ante fallos en la comunicación con blockchain o el sensor IoT, el sistema debe devolver códigos de estado HTTP semánticamente correctos: 404 cuando el lote no existe, 503 cuando el sensor no está disponible y 400 cuando los datos de entrada no superan la validación.
- RNF-08 - Cobertura de tests. El sistema debe disponer de una suite de tests automatizados que cubra los 23 endpoints de la API REST y los métodos principales de la capa de servicio, garantizando el comportamiento correcto tanto en condiciones normales como ante fallos de los sistemas externos.

### 4.1.3 ALCANCE

El sistema implementado cubre el smart contract de trazabilidad en Solidity con control de acceso mediante OpenZeppelin Ownable, la API REST completa con endpoints directos y endpoints de solicitud IoT, la integración blockchain mediante firma de transacciones con protección EIP-155, la simulación de sensores industriales mediante MQTT con Docker

Compose, y una suite de tests automatizados con cobertura de los endpoints y métodos de servicio principales.

El frontend tiene un carácter instrumental: su función es servir como herramienta de demostración del sistema, no como producto final orientado al usuario. El diseño de la interfaz y la experiencia de usuario no constituyen el núcleo técnico del trabajo, que se centra en la arquitectura de integración IoT-blockchain y en el backend.

Dentro del alcance del prototipo, se distingue entre eventos instrumentados mediante comunicación MQTT y eventos operativos registrados por interacción del usuario. Los eventos asociados a mediciones automáticas del proceso, como temperatura, humedad, peso o variables obtenidas mediante sensores se integran a través de los canales MQTT definidos. En cambio, eventos como el cierre del camión o el volcado se modelan como confirmaciones operativas realizadas por el operario desde la interfaz, ya que en esta versión del sistema no se incorpora sensorización específica para verificar físicamente dichas acciones.

Esta decisión responde a una delimitación funcional del prototipo: el objetivo principal es demostrar la integración entre IoT, backend y blockchain para el registro trazable de eventos relevantes, no implementar todos los mecanismos físicos de detección posibles. En un despliegue real, estos eventos podrían automatizarse mediante sensores de posición, finales de carrera o lectores NFC/RFID. No obstante, en el prototipo se registran como eventos manuales certificados en blockchain, manteniendo su valor dentro del historial de trazabilidad del lote.

## *4.2 ARQUITECTURA GENERAL*

El sistema se organiza en cuatro capas funcionales con responsabilidades claramente separadas, conectadas mediante interfaces bien definidas. La Figura 11 muestra el diagrama de arquitectura general del sistema.

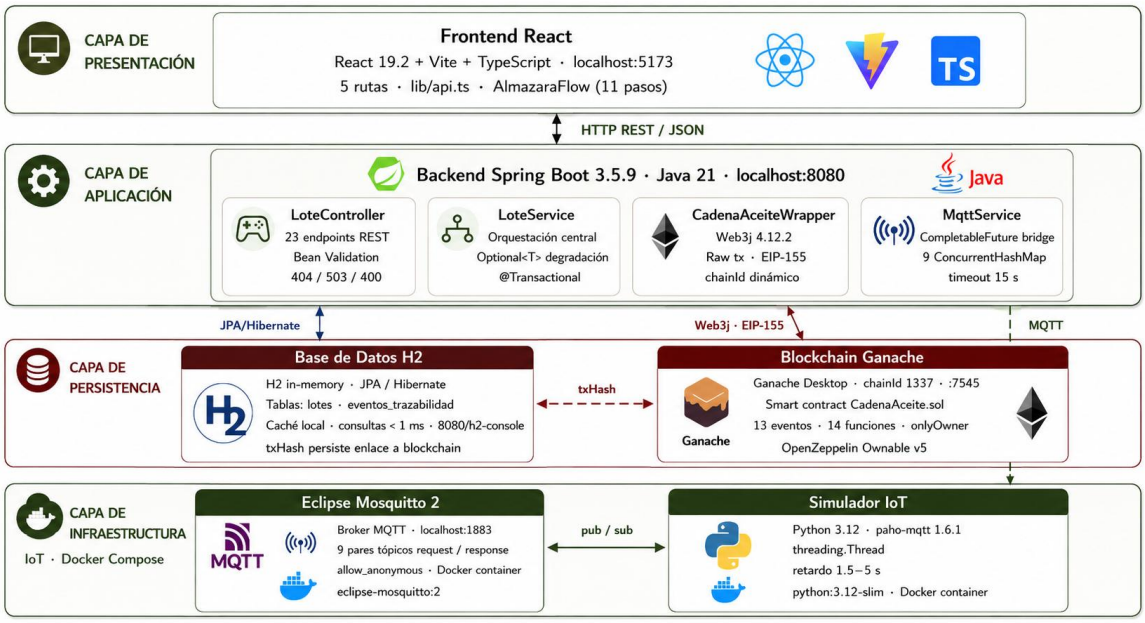


*Figura 11. Diagrama de arquitectura general del sistema.*

La capa de presentación es la interfaz web desarrollada con React, Vite y TypeScript, accesible desde cualquier navegador en localhost:5173 durante el desarrollo. Esta capa tiene un carácter instrumental en el proyecto: su función es guiar al operario a través del flujo de registro de eventos y presentar el historial de trazabilidad al auditor. La comunicación con el backend se realiza exclusivamente mediante peticiones HTTP REST en formato JSON, sin acceso directo ni al nodo blockchain ni al broker MQTT.

La capa de aplicación es el backend desarrollado con Spring Boot 3.5.9 y Java 21, accesible en localhost:8080. Actúa como orquestador central del sistema: recibe las peticiones HTTP del frontend, aplica la lógica de negocio, coordina la escritura simultánea en la base de datos local y en blockchain, y gestiona la comunicación asíncrona con los sensores IoT mediante MQTT. Es la única capa que conoce y coordina todos los demás componentes del sistema.

La capa de persistencia se divide en dos mecanismos complementarios con roles distintos:

- La base de datos H2 embebida en el backend actúa como caché local de eventos y es la fuente de datos para todas las consultas de trazabilidad.

- El smart contract desplegado en Ganache actúa como registro inmutable y fuente de verdad: cada evento escrito en H2 lleva asociado el hash de transacción del registro correspondiente en blockchain, que cualquier actor puede usar para verificar la autenticidad del dato sin depender del backend.

La capa de infraestructura IoT está compuesta por el broker Eclipse Mosquitto y el simulador Python, ambos containerizados con Docker Compose. El broker gestiona el enrutamiento de mensajes entre el backend y el simulador mediante el modelo de publicación y suscripción de MQTT. El simulador replica el comportamiento de once sensores industriales distribuidos a lo largo del proceso, desde el cierre de la compuerta hasta la centrífuga vertical de la almazara, introduciendo retardos realistas de entre 1,5 y 5 segundos por medición y generando datos con rangos físicamente coherentes entre sí.

Cuatro principios transversales guían el diseño de la arquitectura:

- Desacoplamiento: cada capa conoce únicamente la interfaz de la capa con la que se comunica, sin dependencias directas entre capas no adyacentes.
- Opcionalidad: las capas de blockchain e IoT son opcionales y pueden desactivarse de forma independiente mediante configuración, sin afectar al funcionamiento del resto del sistema.
- Verificabilidad: el hash de transacción presente en cada evento de la base de datos local constituye un enlace criptográfico a la red blockchain que permite verificar la integridad del dato sin intermediarios.
- Coherencia física: los datos simulados por el sistema IoT respetan las relaciones físicas reales del proceso de extracción del aceite de oliva, haciendo el sistema directamente transferible a un entorno con sensores reales sin modificar el backend.

## *4.3 ACTORES Y FLUJO DE USUARIO*

### 4.3.1 ACTORES DEL SISTEMA

El sistema contempla dos actores con roles y necesidades diferenciadas, ilustrado en la Figura 12.

- El operario es el actor responsable del registro de eventos a lo largo de toda la cadena de custodia. Representa al trabajador que interviene en las distintas fases del proceso: el agricultor o cooperativista que prepara la carga en el campo, el transportista que lleva la aceituna a la almazara, y el operario de almazara que supervisa el proceso de recepción y extracción. En el sistema, estos tres roles se modelan como un único actor que avanza secuencialmente por las tres fases del flujo de registro. Su única intervención manual es la introducción de los datos de identificación del lote al inicio del proceso, el ID de agricultor y el ID de la almazara donde se lleva a cabo la extracción; el resto de los datos provienen de los sensores IoT.
- El auditor es el actor responsable de la verificación de la trazabilidad de los lotes. Representa a cualquier actor de la cadena de suministro, como un distribuidor, un organismo certificador o el propio agricultor, que necesita consultar el historial de eventos de un lote y verificar su autenticidad. Su interacción con el sistema se realiza a través de la página de trazabilidad, introduciendo el identificador del lote y consultando el historial completo de eventos con sus hashes de transacción.

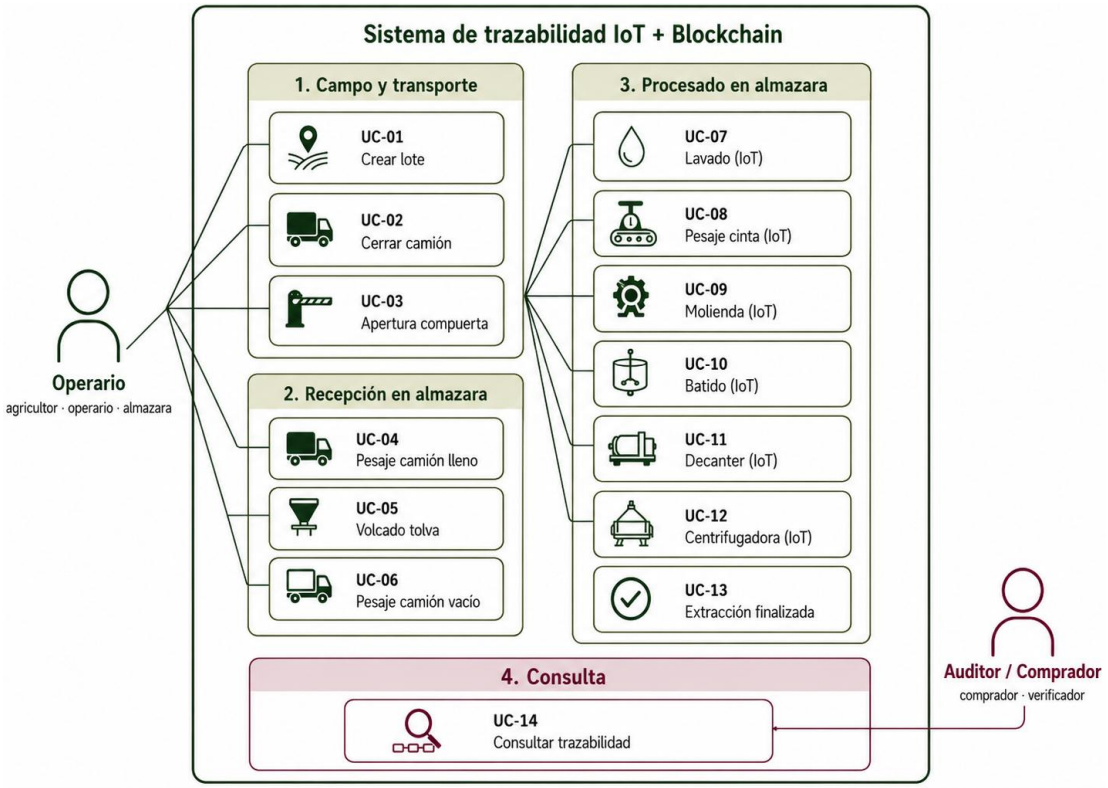


*Figura 12. Diagrama de casos de uso del sistema con los dos actores y sus operaciones asociadas.*

### 4.3.2 FLUJO DE USUARIO DEL OPERARIO

El flujo del operario se articula en tres fases secuenciales que reflejan el recorrido físico de la aceituna desde el campo hasta la almazara. El sistema identifica trece eventos de trazabilidad que cubren el proceso completo desde la recogida de la aceituna en el campo hasta la extracción final del aceite. Cada evento se corresponde con una etapa física del proceso productivo y genera una transacción inmutable en blockchain en el momento en que se registra. La Tabla 1 recoge los eventos identificados con su código interno y descripción.

| Código | Descripción |
|---|---|
| LOTE_CREADO | Creación del lote en campo |
| CAMION_CERRADO | Cierre y sellado del camión |
| APERTURA_COMPUERTA | Apertura de compuerta durante el transporte, autorizada o no |
| PESAJE_CAMION_LLENO | Pesaje del camión cargado en la báscula de la almazara |
| VOLCADO_TOLVA | Descarga de las aceitunas en la tolva de recepción |
| PESAJE_CAMION_VACIO | Pesaje del camión vacío para calcular el peso neto |
| LAVADO | Limpieza de las aceitunas con control de calidad del agua |

| | |
|---|---|
| PESAJE_CINTA | Pesaje de las aceitunas en la cinta transportadora |
| MOLIENDA_INICIADA | Trituración de las aceitunas en el molino |
| TEMPERATURA_BATIDO | Control de temperatura durante el batido de la pasta |
| DECANTER | Separación centrífuga horizontal: aceite, agua y alpeorujo |
| CENTRIFUGADORA | Separación centrífuga vertical para purificar el aceite |
| EXTRACCION_FINALIZADA | Fin del proceso con litros totales y rendimiento |

*Tabla 1. Tabla de eventos de trazabilidad identificados en el sistema.*

### *4.3.2.1 Fase 1: Campo*

El operario accede a la página de inicio, donde selecciona la opción de registrar un nuevo lote (Figura 13).

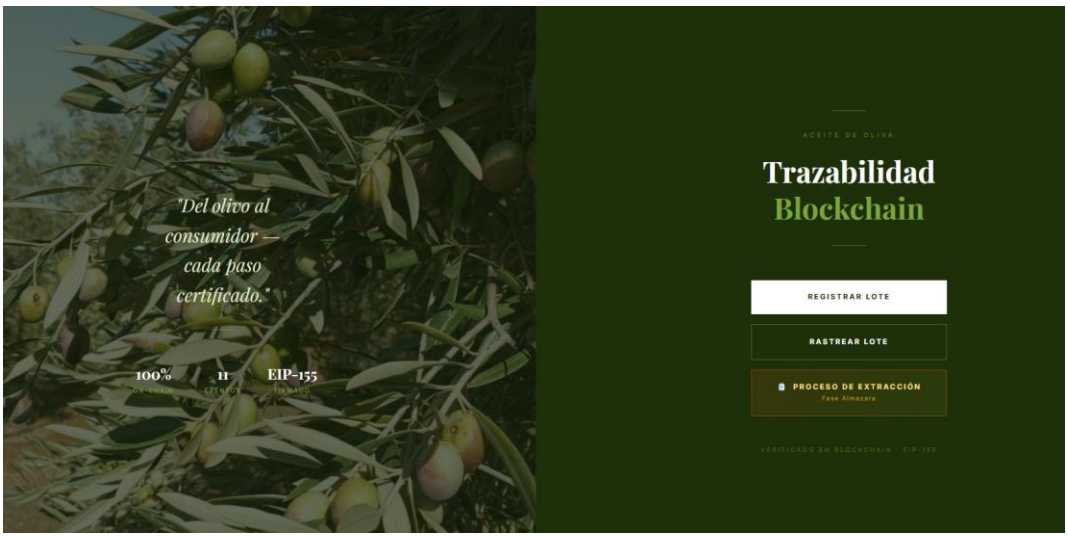


*Figura 13. Captura de pantalla del menú de inicio.*

A continuación, accede a la página de registro e introduce los datos de identificación del lote: el identificador del agricultor, la finca de origen, el identificador del contenedor de recogida y la matrícula del camión de transporte, como se puede ver en la Figura 14.

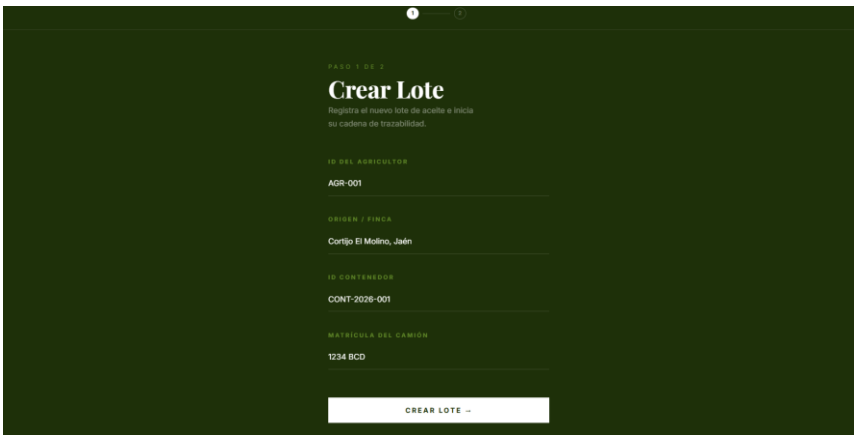


*Figura 14. Captura de la pantalla de registro de campo mostrando el formulario de creación de lote.*

El sistema genera automáticamente el identificador del lote con el formato LOT-{año}-{secuencia} y registra el evento LOTE_CREADO en blockchain con todos los datos de trazabilidad de origen. A continuación, el sistema registra el cierre de la compuerta del camión con el evento CAMION_CERRADO, sellando la carga para el transporte como se muestra en la Figura 15.


*Figura 15. Captura de la pantalla de cierre compuerta.*

### *4.3.2.2 Fase 2: Transporte*

Durante el transporte, el sistema muestra el estado del viaje. En la implementación actual, el operario puede registrar manualmente el evento APERTURA_COMPUERTA (Figura 16) si se produce una apertura no autorizada durante el trayecto, indicando si es autorizada o no autorizada. En un entorno de producción real, este evento sería generado automáticamente

por el sensor físico instalado en la compuerta del camión. El evento queda certificado en blockchain de forma informativa sin interrumpir el flujo del proceso, y tanto si se registra una apertura como si no, el operario continúa a la fase de almazara.

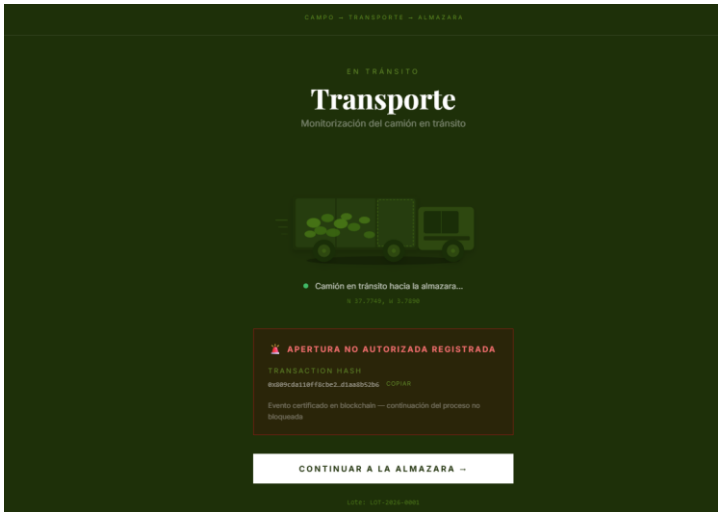


*Figura 16. Captura de pantalla del transporte.*

#### *4.3.2.3 Fase 3: Almazara*

La fase de almazara comprende once pasos secuenciales que cubren el proceso completo de recepción y extracción. En cada paso, el sistema conecta automáticamente con el sensor correspondiente mediante MQTT, recibe el dato medido y lo registra en blockchain sin intervención del operario. El inicio del proceso se da cuando se introduce, como se puede observar en la Figura 17, el identificador de la almazara. En la implementación actual este dato se introduce manualmente por el operario, dado que el sistema no dispone de hardware de identificación automática en la entrada de la instalación. No obstante, la arquitectura MQTT adoptada permite sustituir este paso por un sensor de identificación automática, como un lector RFID o un sensor de presencia con geolocalización, que publique el identificador en el tópico correspondiente sin modificar ningún componente del backend, siendo esta una de las extensiones naturales del sistema descritas en el apartado de trabajo futuro.


*Figura 17. Captura de pantalla del inicio de la extracción.*

El primer paso registra el pesaje del camión lleno. Al llegar a la almazara, la báscula de entrada mide el peso total del camión cargado de aceituna y registra el evento PESAJE_CAMION_LLENO (Figura 18).

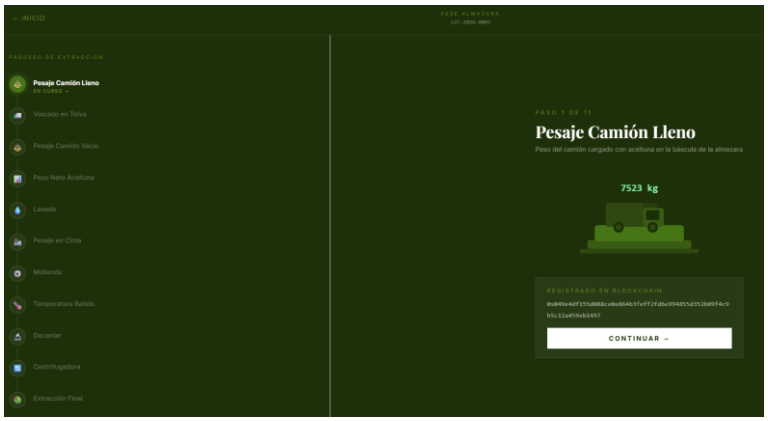


*Figura 18. Captura de pantalla de pesaje camión lleno.*

El segundo paso registra el volcado en tolva. El operario confirma el volcado de la aceituna en la tolva de entrada y el sistema registra el evento VOLCADO_TOLVA, certificando el momento en que la carga pasa a custodia de la almazara (Figura 19).

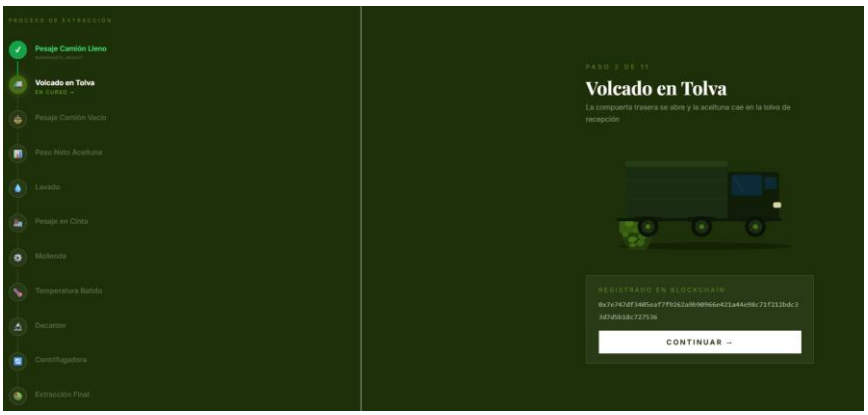


*Figura 19. Captura de pantalla de volcado en tolva.*

El tercer paso registra el pesaje del camión vacío. La báscula de salida mide el peso del camión sin carga y registra el evento PESAJE_CAMION_VACIO (Figura 20).

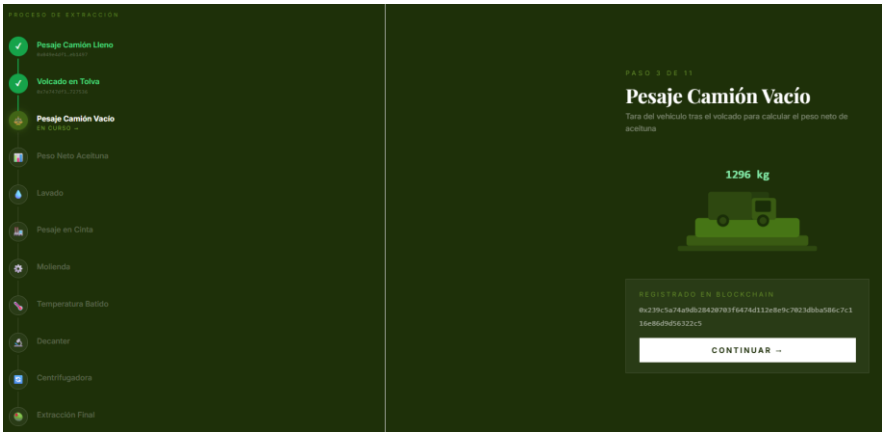


*Figura 20. Captura de pantalla de pesaje camión vacío.*

El cuarto paso calcula automáticamente el peso neto de aceituna como diferencia entre el pesaje lleno y el pesaje vacío. Este paso es exclusivamente visual y no genera evento blockchain (Figura 21).

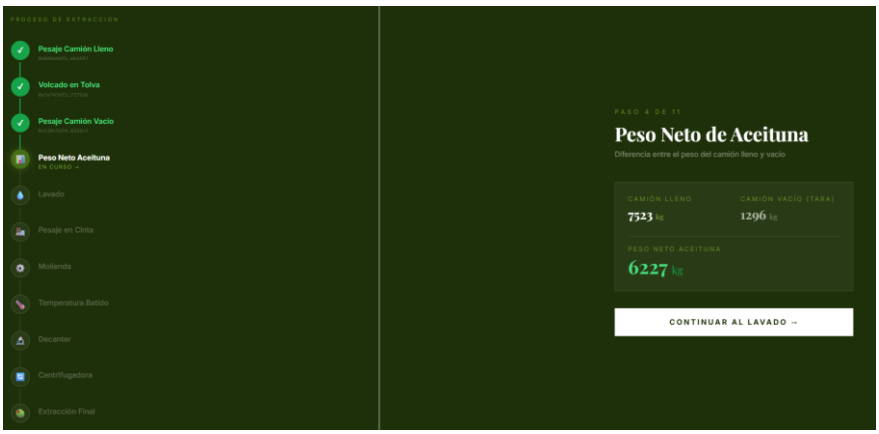


*Figura 21. Captura de pantalla de peso neto aceitunas.*

El quinto paso registra el lavado. El sensor de calidad del agua mide la temperatura, el pH y la aptitud del agua de lavado y registra el evento LAVADO, certificando que la aceituna fue procesada con agua en condiciones óptimas (Figura 22).

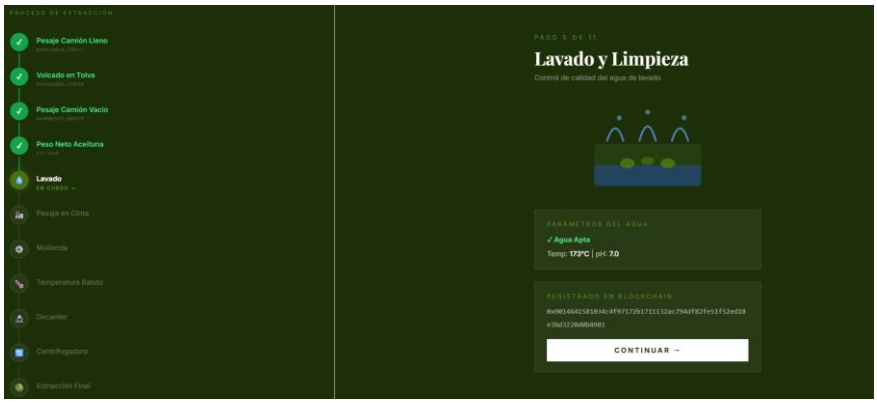


*Figura 22. Captura de pantalla de lavado y limpieza de las aceitunas.*

El sexto paso registra el pesaje en cinta. La báscula de la cinta transportadora mide el peso de la aceituna limpia que entra al proceso de extracción y registra el evento PESAJE_CINTA (Figura 23).


*Figura 23. Captura de pantalla del pesaje en cinta.*

El séptimo paso registra la molienda. El sensor de temperatura del molino registra las condiciones al inicio de la trituración con el evento MOLIENDA_INICIADA, certificando que el proceso comenzó dentro del plazo recomendado desde la recepción (Figura 24).


*Figura 24. Captura de pantalla de la molienda.*

El octavo paso registra la temperatura de batido. La sonda registra la temperatura de la batidora con el evento TEMPERATURA_BATIDO (Figura 25).

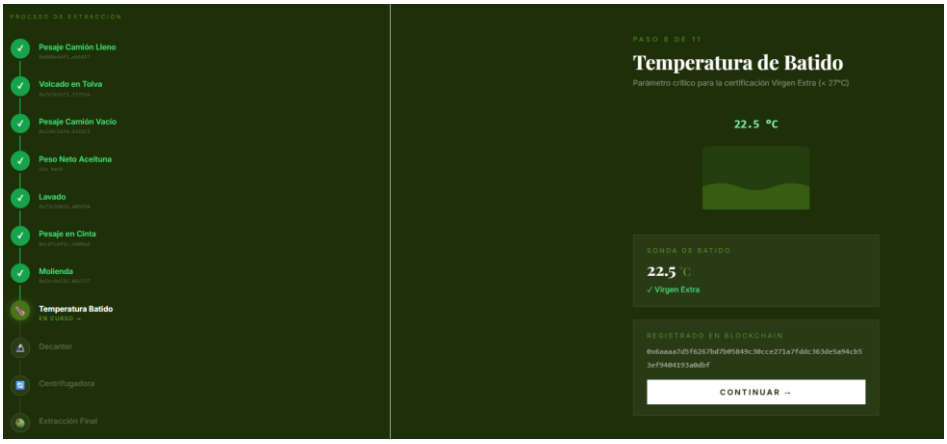


*Figura 25. Captura de pantalla del batido.*

El noveno paso registra el decanter. Los caudalímetros de la centrífuga horizontal miden los litros de aceite obtenidos y los kilogramos de alpeorujo y registran el evento DECANTER (Figura 26).

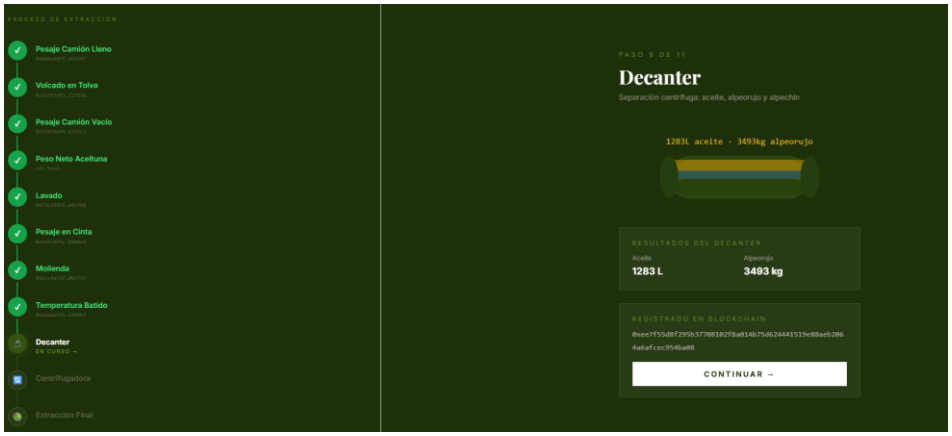


*Figura 26. Captura de pantalla del decanter.*

El décimo paso registra la centrifugadora. Los sensores de la centrífuga vertical registran las revoluciones por minuto y la temperatura de operación con el evento CENTRIFUGADORA (Figura 27).

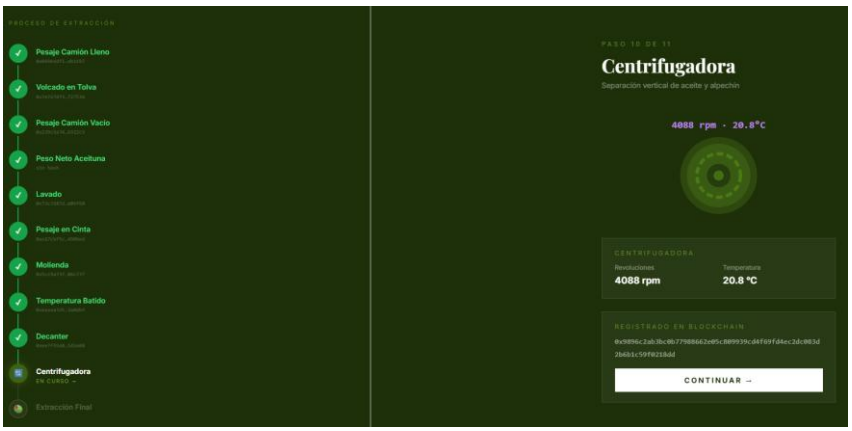


*Figura 27. Captura de pantalla de la centrifugadora.*

El undécimo y último paso registra la extracción finalizada. El sistema registra el volumen total de aceite producido y el rendimiento del proceso con el evento EXTRACCION_FINALIZADA, cerrando la cadena de trazabilidad del lote (Figura 28).

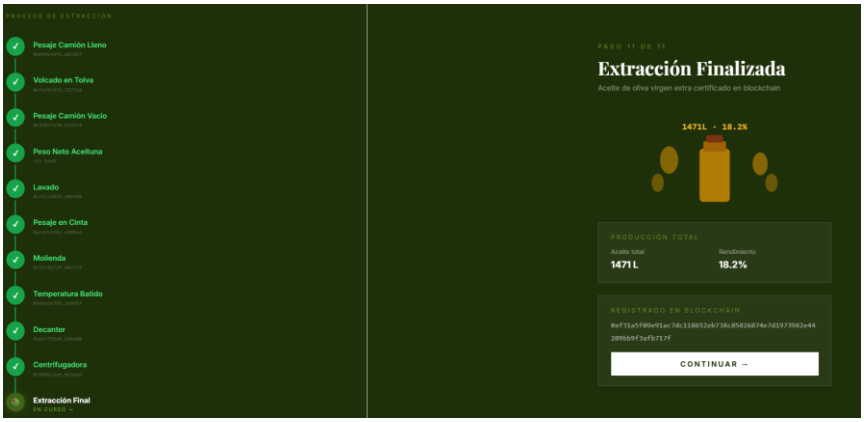


*Figura 28. Captura de pantalla de la extracción finalizada.*

La Figura 29 muestra el diagrama del proceso de extracción de aceite de oliva en la almazara con los puntos de integración IoT identificados. Cada símbolo IoT acompañado de un indicador de verificación representa un punto del proceso donde el sistema recibe datos de un sensor mediante el protocolo MQTT y los registra de forma inmutable en blockchain.

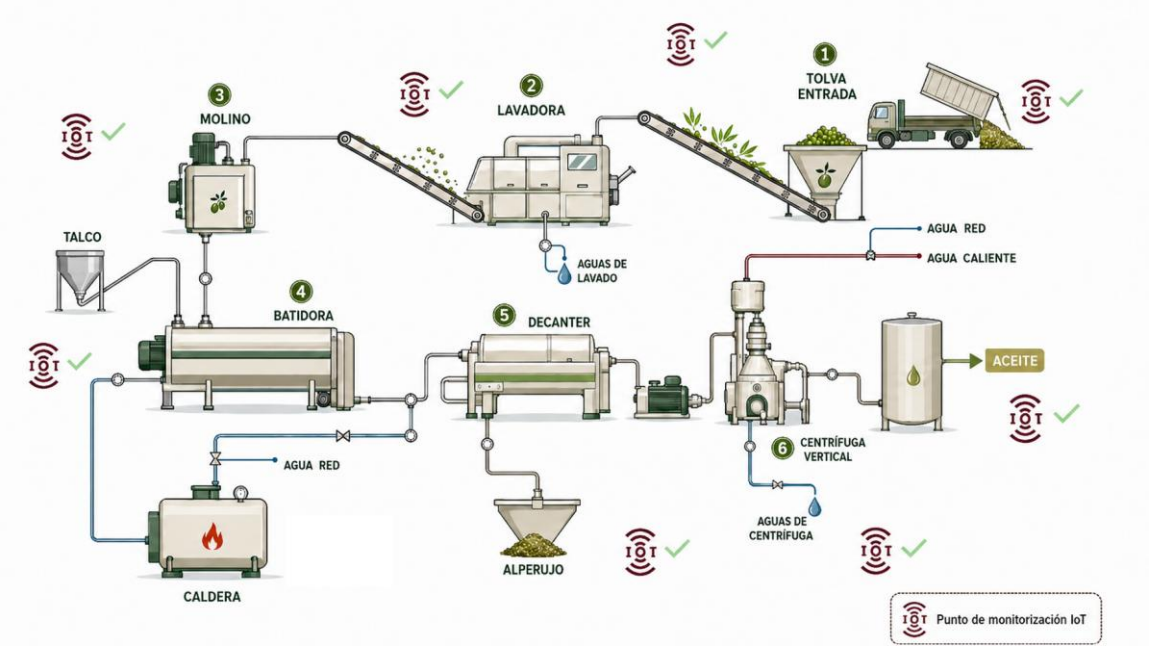


*Figura 29. Flujo completo de datos registrados dentro de la almazara.*

### 4.3.3 FLUJO DE USUARIO DEL AUDITOR

El flujo del auditor es independiente del flujo del operario y no requiere seguir ningún orden secuencial. El auditor accede a la página de trazabilidad e introduce el identificador del lote que desea consultar, como se muestra en la Figura 30. El sistema recupera de la base de datos local H2 el historial completo de eventos del lote, ordenado cronológicamente, y lo presenta en forma de línea temporal. Cada evento muestra el tipo de operación, el timestamp, los datos específicos del sensor cuando aplica, y el hash de transacción blockchain. El auditor puede copiar cualquier hash de transacción para verificarlo de forma independiente directamente en el nodo Ganache, confirmando que el dato presentado por el sistema coincide con el registro inmutable en blockchain.

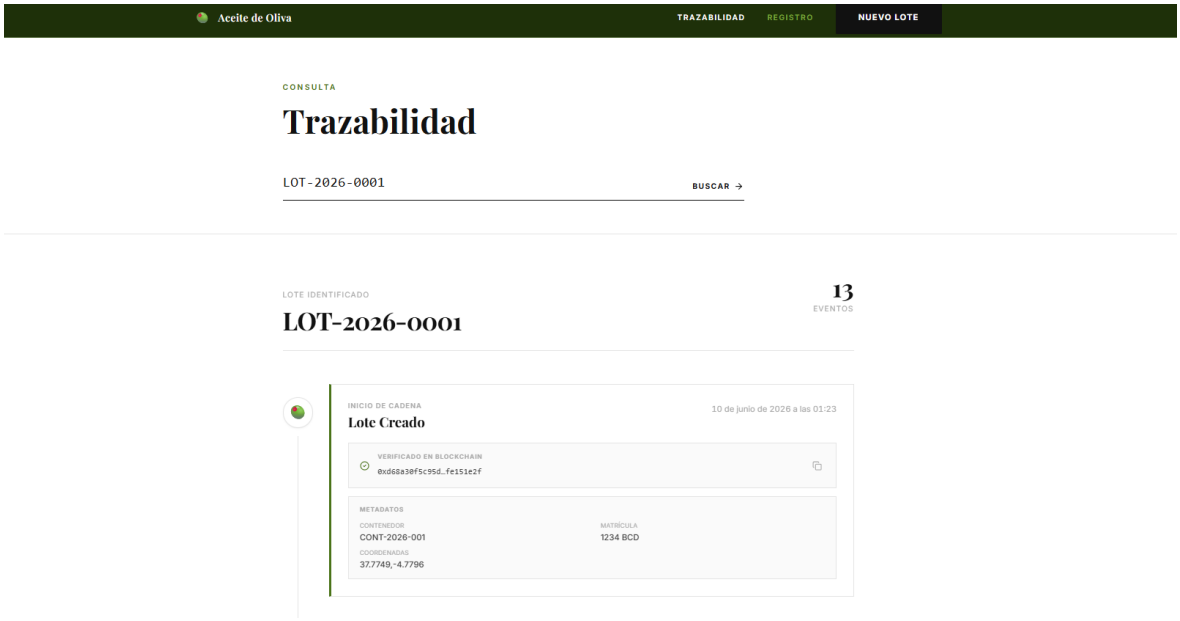


*Figura 30. Captura de la página de trazabilidad mostrando el historial completo de un lote con los 13 eventos certificados y sus hashes de transacción.*

## 4.4 DISEÑO DEL SMART CONTRACT

El smart contract `CadenaAceite` constituye la capa de registro inmutable del sistema. Su diseño responde a un principio de mínima complejidad: el contrato no contiene lógica de negocio ni validaciones complejas, sino que actúa exclusivamente como registro de eventos, dejando toda la orquestación al backend. Esta decisión reduce el coste de gas de cada operación y minimiza la superficie de error en el código desplegado en blockchain, cuya inmutabilidad impide correcciones posteriores.

El contrato adopta, además, un modelo de registro en dos capas complementarias. La primera capa corresponde al almacenamiento interno del contrato, donde cada evento queda guardado en el array asociado a su lote dentro del `mapping` principal. Esta capa permite que el backend recupere el historial completo de un lote mediante la función `obtenerEventos()`, sin depender de herramientas externas de indexación. La segunda capa corresponde a los eventos Solidity emitidos en cada operación, que quedan registrados en los logs de las

transacciones. Estos logs están pensados para facilitar la consulta, filtrado y auditoría externa del sistema, especialmente gracias al uso del parámetro `loteId` como campo `indexed`.

De este modo, cada operación relevante queda registrada simultáneamente en el estado persistente del contrato y en los logs de la red blockchain. La primera capa proporciona una consulta directa y estructurada desde el backend, mientras que la segunda ofrece una vía de verificación externa, indexable y auditable por herramientas compatibles con Ethereum. Esta separación refuerza la trazabilidad del sistema, ya que permite consultar el historial de un lote tanto desde la lógica de la aplicación como desde la propia infraestructura blockchain.

### 4.4.1 CONTROL DE ACCESO

El contrato hereda de Ownable de OpenZeppelin v5, lo que garantiza que únicamente la cuenta que desplegó el contrato puede invocar las funciones de escritura. En el constructor se llama a `Ownable(msg.sender)`, registrando automáticamente como propietario la cuenta de Ganache cuya clave privada está configurada en el backend. Todas las funciones de escritura llevan el modificador `onlyOwner`, de forma que cualquier intento de registrar un evento desde una cuenta no autorizada es rechazado por la EVM antes de ejecutar ninguna lógica. En un entorno de producción, este mecanismo se extendería a un sistema de roles por actor mediante `AccessControl` de OpenZeppelin.

### 4.4.2 ESTRUCTURA DE DATOS

El almacenamiento principal es un mapping que asocia cada identificador de lote con su lista de eventos: `mapping(uint256 => EventoOnChain[]) private _eventos`. Cada evento se representa mediante la estructura `EventoOnChain`, compuesta por tres campos: el tipo de evento como cadena de texto, el peso en kilogramos como entero sin signo de 256 bits y el timestamp Unix del momento del registro. El uso de uint256 como tipo del identificador de lote responde a que es el tipo nativo de Solidity y el más eficiente en términos de gas. El identificador numérico utilizado en el contrato corresponde al campo id autogenerado por la base de datos H2, que actúa como clave de vinculación entre ambas capas de persistencia.

### 4.4.3 EVENTOS SOLIDITY

El smart contract define trece eventos Solidity, que quedan registrados en los logs de las transacciones generadas durante la ejecución del sistema. Todos ellos incluyen el parámetro `loteId` marcado como `indexed`, lo que permite filtrar de forma eficiente los eventos asociados a un lote concreto mediante consultas al nodo, sin necesidad de recorrer todo el almacenamiento interno del contrato.

Los eventos se organizan de acuerdo con las dos fases principales del proceso de trazabilidad. En primer lugar, la fase de campo y transporte incluye los eventos `LoteCreado`, `CamionCerrado` y `AperturaCompuertaRegistrada`. El evento `LoteCreado` registra la información inicial del lote, incluyendo el identificador del agricultor, la cooperativa, el contenedor, la matrícula del camión, las coordenadas GPS y el timestamp. El evento `CamionCerrado` registra la confirmación del cierre del camión, junto con el identificador de la cooperativa y el timestamp. Por su parte, `AperturaCompuertaRegistrada` permite registrar la apertura de la compuerta, incluyendo un valor booleano de autorización, la ubicación y el timestamp.

En segundo lugar, la fase de almazara comprende los eventos vinculados a la recepción, tratamiento y extracción del aceite. Los eventos `PesajeCamionLlenoRegistrado`, `VolcadoTolvaRegistrado` y `PesajeCamionVacioRegistrado` recogen las operaciones iniciales de recepción del camión. A continuación, `LavadoRegistrado` registra la aptitud del agua, la temperatura y el pH del proceso de lavado, mientras que `PesajeCintaRegistrado` almacena el peso de la aceituna limpia. La etapa de extracción queda representada mediante los eventos `MoliendaIniciada`, `TemperaturaBatidoRegistrada` y `DecanterRegistrado`. Finalmente, `CentrifugadoraRegistrada` recoge las revoluciones por minuto y la temperatura de la centrifugadora, y `ExtraccionFinalizada` registra el volumen total de aceite obtenido y el rendimiento final del proceso.

Para representar valores decimales, como temperaturas, pH o rendimientos, se adopta la convención de multiplicar por diez el valor real antes de registrarlo en el contrato. Esta decisión responde a que Solidity no dispone de tipos de coma flotante nativos. De este modo,

un valor como 24,5 °C se almacena como 245, conservando un decimal de precisión mediante aritmética entera. Esta aproximación resulta más predecible y eficiente en términos de gas que el uso de representaciones alternativas más complejas.

### 4.4.4 FUNCIONES

El contrato expone catorce funciones: trece funciones de escritura y una función de lectura. Las funciones de escritura se corresponden con los trece tipos de evento definidos en el sistema y tienen visibilidad `external`, además de estar protegidas mediante el modificador `onlyOwner`. Cada una de estas funciones registra una operación concreta del proceso de trazabilidad y genera una transacción independiente, asociada a su correspondiente hash.

La función de lectura, denominada `obtenerEventos`, tiene visibilidad `external view` y permite consultar el historial completo de eventos asociados a un lote. Al tratarse de una función de solo lectura, no modifica el estado del contrato y no consume gas cuando se invoca externamente mediante una consulta al nodo.

La elección de visibilidad `external` frente a `public` responde a criterios de eficiencia. Dado que estas funciones no se invocan internamente desde el propio contrato, `external` permite trabajar directamente con los argumentos recibidos en `calldata`, evitando copias innecesarias a memoria, especialmente en parámetros de tipo `string`. Esta decisión contribuye a reducir el coste de ejecución de las transacciones y hace más eficiente la interacción con el contrato.

En la Figura 31 se puede observar un resumen del contenido de contrato CadenaAceite.

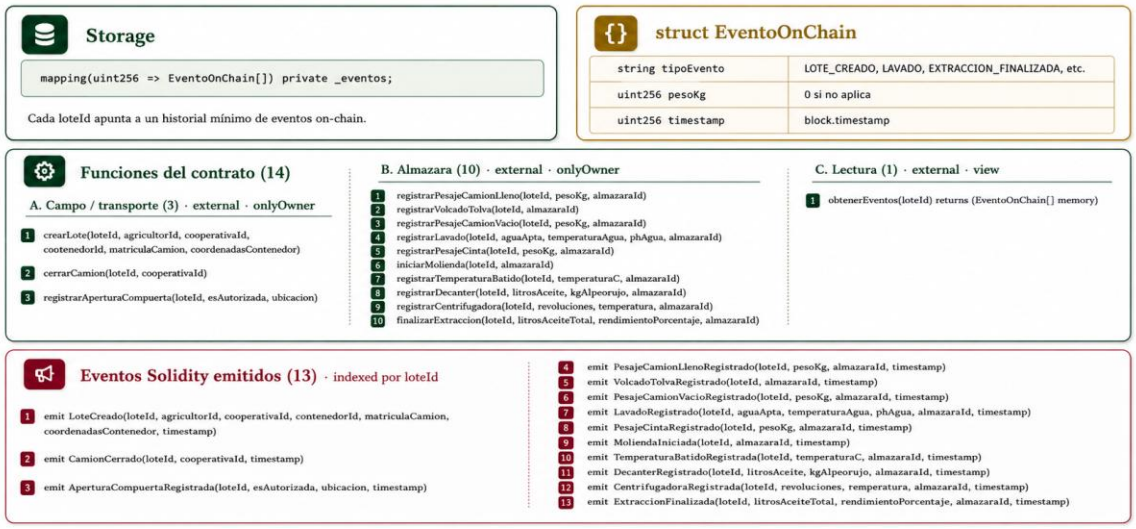


*Figura 31.Diagrama de las funciones, eventos y estructura de datos del contrato CadenaAceite.*

## 4.5 DISEÑO DE LA API REST

La API REST constituye la interfaz de comunicación entre el frontend y el backend. Su diseño sigue los principios REST habituales: cada recurso se identifica mediante una URL, las operaciones se expresan mediante métodos HTTP y las respuestas utilizan códigos de estado para indicar el resultado de la petición. Los cuerpos de petición y respuesta se serializan en formato JSON, y la URL base de los endpoints definidos para la gestión de lotes es `/api/lotes`.

Los endpoints se organizan en dos grupos principales según su forma de obtención del dato. Por un lado, los endpoints directos reciben la información en el cuerpo de la petición y la persisten inmediatamente. Por otro, los endpoints de solicitud IoT, identificados mediante el sufijo `/solicitar`, publican una petición en el broker MQTT, esperan la respuesta del sensor correspondiente con un tiempo máximo de espera de 15 segundos y persisten el dato recibido. Para cada endpoint de solicitud IoT se define un endpoint directo equivalente, que actúa como mecanismo de respaldo manual cuando el sensor no está disponible.

### 4.5.1 ENDPOINTS DE CAMPO Y TRANSPORTE

| Método | Endpoint | Descripción | Entrada principal | Respuesta |
|---|---|---|---|---|
| POST | `/api/lotes` | Crea un nuevo lote. | Agricultor, finca de origen, contenedor, matrícula y coordenadas GPS opcionales. | Lote creado con identificador generado. |
| POST | `/api/lotes/{loteId}/cerrar` | Registra el cierre de la puerta del camión. | No requiere cuerpo. | Evento registrado y hash de transacción. |
| POST | `/api/lotes/{loteId}/apertura-compuerta` | Registra una apertura de puerta durante el transporte. | Autorización y ubicación. | Evento registrado y hash de transacción. |

*Tabla 2. Tabla con resumen de endpoints asociados al campo y transporte del lote.*

### 4.5.2 ENDPOINTS DE RECEPCIÓN EN ALMAZARA

| Método | Endpoint | Modalidad | Descripción | Entrada principal |
|---|---|---|---|---|
| POST | `/api/lotes/{loteId}/pesaje-camion-lleno/solicitar` | IoT | Solicita al sensor el pesaje del camión cargado. | Identificador de la almazara. |
| POST | `/api/lotes/{loteId}/pesaje-camion-lleno` | Manual | Registra manualmente el pesaje del camión cargado. | Identificador de la almazara y peso. |
| POST | `/api/lotes/{loteId}/volcado-tolva` | Manual | Registra el volcado de la aceituna en la tolva. | Identificador de la almazara. |
| POST | `/api/lotes/{loteId}/pesaje-camion-vacio/solicitar` | IoT | Solicita al sensor el pesaje del camión vacío. | Identificador de la almazara. |
| POST | `/api/lotes/{loteId}/pesaje-camion-vacio` | Manual | Registra manualmente el pesaje del camión vacío. | Identificador de la almazara y peso. |

*Tabla 3. Tabla con resumen de endpoints de recepción del lote en almazara.*

### 4.5.3 ENDPOINTS DE PROCESADO EN ALMAZARA

Los endpoints de procesado en almazara siguen un patrón dual. La variante terminada en `/solicitar` obtiene el dato desde el sensor correspondiente mediante MQTT, mientras que

la variante directa recibe los datos en el cuerpo de la petición como mecanismo de respaldo manual.

| Endpoint base | Variante IoT | Variante manual | Datos registrados |
|---|---|---|---|
| `/lavado` | `/lavado/solicitar` | `/lavado` | Temperatura del agua, pH y aptitud del agua. |
| `/pesaje-cinta` | `/pesaje-cinta/solicitar` | `/pesaje-cinta` | Peso de la aceituna limpia en cinta transportadora. |
| `/molienda` | `/molienda/solicitar` | `/molienda` | Inicio de molienda y temperatura del molino. |
| `/temperatura-batido` | `/temperatura-batido/solicitar` | `/temperatura-batido` | Temperatura de batido. |
| `/decanter` | `/decanter/solicitar` | `/decanter` | Litros de aceite y kilogramos de alpeorujo. |
| `/centrifugadora` | `/centrifugadora/solicitar` | `/centrifugadora` | Revoluciones por minuto y temperatura. |
| `/extraccion-finalizada` | `/extraccion-finalizada/solicitar` | `/extraccion-finalizada` | Litros totales de aceite y rendimiento final. |

*Tabla 4. Tabla con resumen de endpoints de procesado del lote en almazara.*

### 4.5.4 ENDPOINT DE CONSULTA

| Método | Endpoint | Descripción | Respuesta |
|---|---|---|---|
| GET | `/api/lotes/{loteId}/trazabilidad` | Devuelve el historial completo de eventos asociados a un lote. | Identificador del lote y lista cronológica de eventos con tipo, peso si aplica, timestamp, hash de transacción y metadatos JSON. |

*Tabla 5. Tabla con endpoint de consulta del historial de trazabilidad.*

### 4.5.5 MANEJO DE ERRORES

El backend gestiona tres situaciones de error con códigos semánticamente correctos. Cuando el lote especificado no existe devuelve 404 con el mensaje de error en texto plano. Cuando el sensor IoT no está disponible o no responde en el tiempo límite devuelve 503 con una descripción del motivo. Cuando los datos de entrada no superan la validación devuelve 400 con un mapa JSON que identifica cada campo con su mensaje de error específico.

## 4.6 *MODELO DE DATOS*

La persistencia local del sistema se gestiona mediante dos entidades JPA mapeadas a tablas en la base de datos H2. El modelo de datos está diseñado para ser un espejo eficiente de los eventos registrados en blockchain, añadiendo los campos necesarios para las consultas REST sin duplicar información innecesaria (Figura 32).

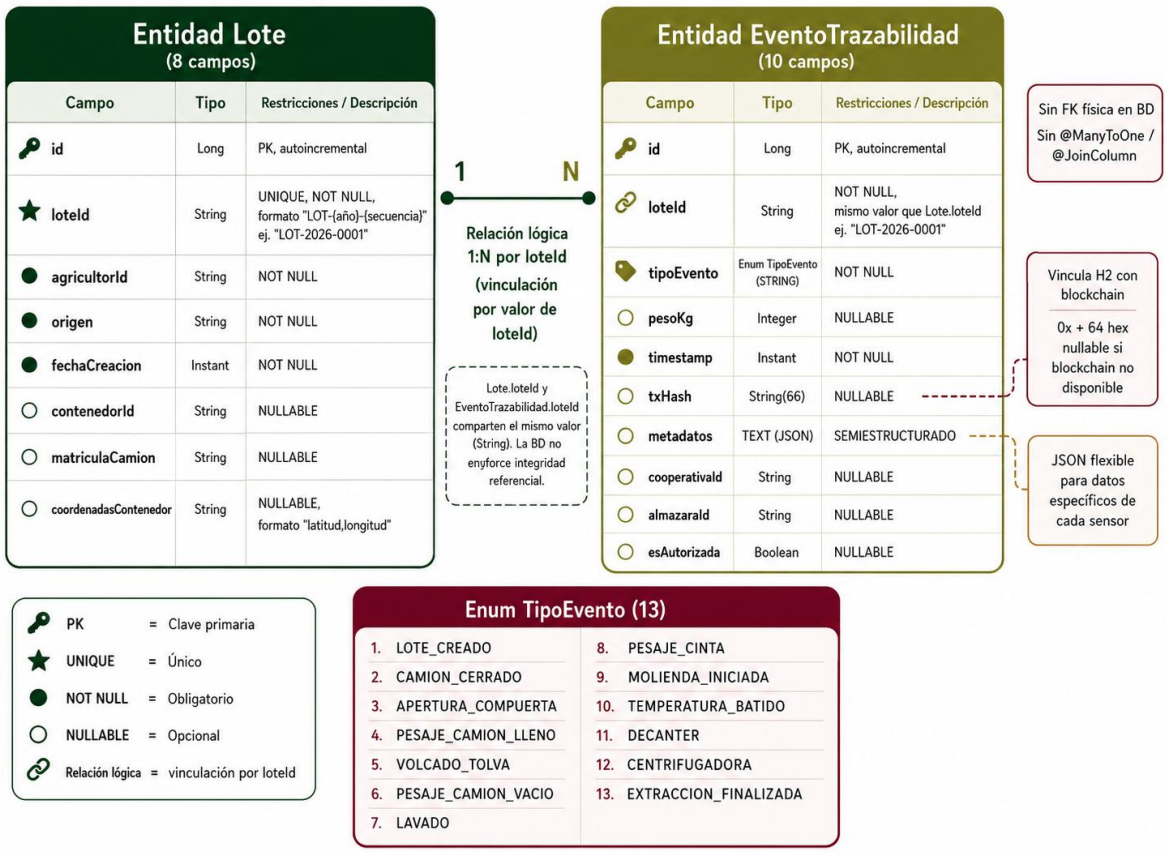


*Figura 32. Diagrama entidad-relación del modelo de datos.*

### 4.6.1.1 Entidad Lote

La entidad `Lote` representa un lote de aceituna y se mapea a la tabla lotes. Tiene ocho campos. El campo `id` es la clave primaria numérica autoincremental generada por la base de datos, y es el valor que se utiliza como identificador del lote en el smart contract. El campo `loteId` es el identificador legible con formato LOT-{año}-{secuencia}, definido como único en la

base de datos y no nulo. Este identificador se utiliza en la API REST y en el frontend; su presencia se limita a la base de datos H2 y no tiene representación en blockchain, donde el lote se referencia exclusivamente mediante el campo `id` numérico.

Los campos `agricultorId` y `origen` almacenan la información del agricultor y la procedencia de la aceituna, ambos no nulos. El campo `fechaCreacion` almacena el timestamp del momento de creación del lote como `Instant`, también no nulo.

Los tres campos adicionales incorporados para la trazabilidad de origen son `contenedorId`, `matriculaCamion` y `coordenadasContenedor`, todos ellos `nullable`. El campo `contenedorId` identifica el contenedor de recogida la aceituna en el campo. El campo `matriculaCamion` identifica el vehículo de transporte. El campo `coordenadasContenedor` almacena la posición GPS del contenedor en el momento de la carga, en formato de cadena de texto con latitud y longitud separadas por coma. Estos tres campos se persisten también en blockchain en el momento de la creación del lote, certificando la trazabilidad de origen desde el primer evento de la cadena.

La entidad dispone de tres constructores. El constructor protegido sin argumentos es requerido por JPA para instanciar entidades desde los resultados de una consulta SQL mediante reflexión. El constructor de tres argumentos acepta `loteId`, `agricultorId` y `origen`, y es compatible con el formato original de creación de lotes sin datos de trazabilidad adicionales, garantizando la compatibilidad con el frontend existente. El constructor de seis argumentos añade `contenedorId`, `matriculaCamion` y `coordenadasContenedor`, y es el utilizado cuando el frontend envía los datos de trazabilidad completos.

#### *4.6.1.2 Entidad EventoTrazabilidad*

La entidad `EventoTrazabilidad` representa un evento registrado en la cadena de trazabilidad y se mapea a la tabla `eventos_trazabilidad`. Tiene diez campos:

- El campo `id` es la clave primaria numérica autoincremental.
- El campo `loteId` es el identificador del lote al que pertenece el evento, no nulo, y actúa como clave de relación con la entidad Lote.

- El campo `tipoEvento` almacena el tipo de evento mediante el enum `TipoEvento`, persistido como cadena de texto mediante la anotación `@Enumerated(EnumType.STRING)`, lo que garantiza que la base de datos sea legible directamente y evita inconsistencias si el orden del `enum` cambia en el futuro.
- El campo `pesoKg` almacena el peso en kilogramos como entero y es `nullable`, ya que solo tiene valor en eventos de pesaje.
- El campo `timestamp` almacena la marca de tiempo del registro como `Instant`, no nulo.
- El campo `txHash` almacena el hash de transacción blockchain con una longitud máxima de 66 caracteres, `nullable` cuando blockchain no estaba disponible en el momento del registro.
- El campo `metadatos` es un campo de tipo `TEXT` que almacena en formato JSON los datos específicos de cada tipo de sensor, como la temperatura y el pH en el lavado, las revoluciones y temperatura en la centrifugadora, o los litros de aceite y kilogramos de alpeorujo en el decanter.
- El campo `cooperativaId` almacena el identificador de la cooperativa de origen del lote, `nullable`, y está presente únicamente en los eventos de la fase de campo y transporte.
- El campo `almazaraId` almacena el identificador de la almazara donde se procesa el lote, `nullable`, y está presente en todos los eventos de la fase de almazara.
- El campo `esAutorizada` es un booleano `nullable` que solo tiene valor en los eventos de tipo APERTURA_COMPUERTA, indicando si la apertura detectada por el sensor fue autorizada o no.

El uso de un campo JSON semiestructurado en lugar de columnas separadas por tipo de evento responde a tres razones. La primera es la flexibilidad: añadir un nuevo tipo de evento con nuevos parámetros no requiere modificar el esquema de la base de datos. La segunda es la simplicidad: una única entidad cubre todos los tipos de evento sin necesidad de herencia ni tablas adicionales. La tercera es la legibilidad: los metadatos son directamente inspeccionables en la consola H2 durante el desarrollo.

#### *4.6.1.3 Enumeración TipoEvento*

El `enum` `TipoEvento` define los 13 tipos de evento descritos en la Tabla 1.

#### *4.6.1.4 Vinculación entre H2 y blockchain*

El campo `txHash` de la entidad `EventoTrazabilidad` es el elemento que vincula la base de datos local con el registro on-chain. Su longitud máxima de 66 caracteres corresponde exactamente a la longitud de un hash de transacción Ethereum: el prefijo 0x seguido de 64 dígitos hexadecimales. Cuando un evento se registra correctamente en blockchain, el backend persiste en H2 el `txHash` devuelto por Ganache. Cuando blockchain no está disponible, el evento se persiste igualmente con `txHash` nulo, garantizando que la trazabilidad local no se interrumpe. En ambos casos el sistema informa al frontend del estado de la certificación on-chain de forma transparente.

## *4.7 DISEÑO DEL FLUJO IOT*

La integración de los sensores IoT en el sistema presenta un reto de diseño fundamental: el protocolo MQTT es asíncrono y orientado a eventos, mientras que la API REST que consume el frontend es síncrona y orientada a petición-respuesta. El diseño del flujo IoT resuelve esta incompatibilidad mediante un patrón de puente síncrono-asíncrono implementado en el backend, que permite que cada endpoint `/solicitar` bloquee el hilo HTTP a la espera de la respuesta del sensor sin exponer esta complejidad al frontend.

### 4.7.1 ARQUITECTURA DE COMUNICACIÓN IOT

La comunicación IoT involucra cuatro componentes: el frontend, el backend, el broker Mosquitto y el simulador del sensor. El frontend no interactúa directamente con ningún componente MQTT: realiza una petición HTTP al endpoint `/solicitar` correspondiente y espera una respuesta HTTP, sin conocimiento del protocolo subyacente. Toda la complejidad de la comunicación MQTT queda encapsulada en el backend, concretamente en la clase `MqttService`.

El simulador Python, containerizado con Docker Compose, implementa el comportamiento de nueve sensores industriales. Cada sensor escucha en su tópico de solicitud, introduce un retardo aleatorio que simula el tiempo de medición real, y publica la respuesta en el tópico de respuesta correspondiente. El uso de `threading.Thread` por cada solicitud garantiza que el simulador puede gestionar múltiples peticiones concurrentes sin bloquear el bucle principal de MQTT.

### 4.7.2 TÓPICOS MQTT DEL SISTEMA

El sistema utiliza nueve pares de tópicos request/response, uno por cada tipo de sensor. Los rangos de datos publicados por el simulador están calibrados para respetar la coherencia física del proceso: el peso del camión lleno es siempre mayor que el peso del camión vacío, el peso en cinta es siempre menor que el peso del camión lleno menos el vacío, y las temperaturas se expresan multiplicadas por diez para preservar un decimal de precisión.

| Tópico MQTT | Dispositivo o sensor asociado | Datos devueltos | Rango de valores |
|---|---|---|---|
| `olive/camionlleno/request` | Báscula de entrada de la almazara | Peso del camión cargado | 7.000-8.000 kg |
| `olive/camionvacio/request` | Báscula de salida de la almazara | Peso del camión vacío | 1.000-2.000 kg |
| `olive/lavado/request` | Sensor de calidad del agua | Temperatura, pH y aptitud del agua | Temperatura: 150-220 (15,0-22,0 °C); pH: 65-75 (6,5-7,5); aptitud: verdadero |
| `olive/cinta/request` | Báscula de la cinta transportadora | Peso de la aceituna limpia | 3.500-4.440 kg |
| `olive/molienda/request` | Sensor de temperatura del molino | Temperatura del molino | 200-280 (20,0-28,0 °C) |
| `olive/batido/request` | Sonda de la batidora | Temperatura de batido | 220-265 (22,0-26,5 °C) |
| `olive/decanter/request` | Caudalímetros del decanter | Litros de aceite y kilogramos de alpeorujo | Aceite: 1.050-1.600 L; alpeorujo: 2.600-3.600 kg |
| `olive/centrifugadora/request` | Sensores de la centrífuga | Revoluciones por minuto y temperatura | RPM: 3.000-5.000; temperatura: 180-240 (18,0-24,0 °C) |

| | | | |
|---|---|---|---|
| `olive/extraccion/request` | Contador de producción final | Litros totales de aceite y rendimiento final | Aceite total: 1.000-1.550 L; rendimiento: 150-200 (15,0-20,0 %) |

*Tabla 6. Tabla de tópicos MQTT del sistema.*

### 4.7.3 EL PATRÓN COMPLETABLEFUTURE COMO PUENTE SÍNCRONO-ASÍNCRONO

El puente entre el mundo síncrono de la API REST y el mundo asíncrono de MQTT se implementa mediante `CompletableFuture<T>` de Java. Para cada tipo de sensor existe un `ConcurrentHashMap` independiente que almacena los futuros pendientes, usando el `loteId` como clave de correlación entre la solicitud y la respuesta.

Cuando el backend recibe una petición HTTP de solicitud IoT, crea un `future` vacío del tipo correspondiente al sensor, lo registra en el mapa con el `loteId` como clave, publica el mensaje de solicitud en el tópico request con el `loteId` en el payload, y bloquea el hilo HTTP llamando a `future.get()` con un timeout de 15 segundos. Paralelamente, el `callback` `messageArrived` de Paho, que se ejecuta en el hilo interno del cliente MQTT, recibe la respuesta del sensor en el tópico response correspondiente, extrae el `loteId` y los datos del mensaje JSON, localiza el `future` en el mapa y lo completa con los datos recibidos, desbloqueando el hilo HTTP. Si el sensor no responde en 15 segundos, el timeout expira y el sistema lanza una `SensorNoDisponibleException`, que el `controller` convierte en una respuesta HTTP 503. En cualquier caso, el bloque finally del método elimina la entrada del mapa para evitar fugas de memoria.

El uso de `ConcurrentHashMap` en lugar de `HashMap` es imprescindible porque el mapa es accedido simultáneamente desde el hilo HTTP que escribe la entrada y desde el hilo interno de Paho que la completa y la elimina.

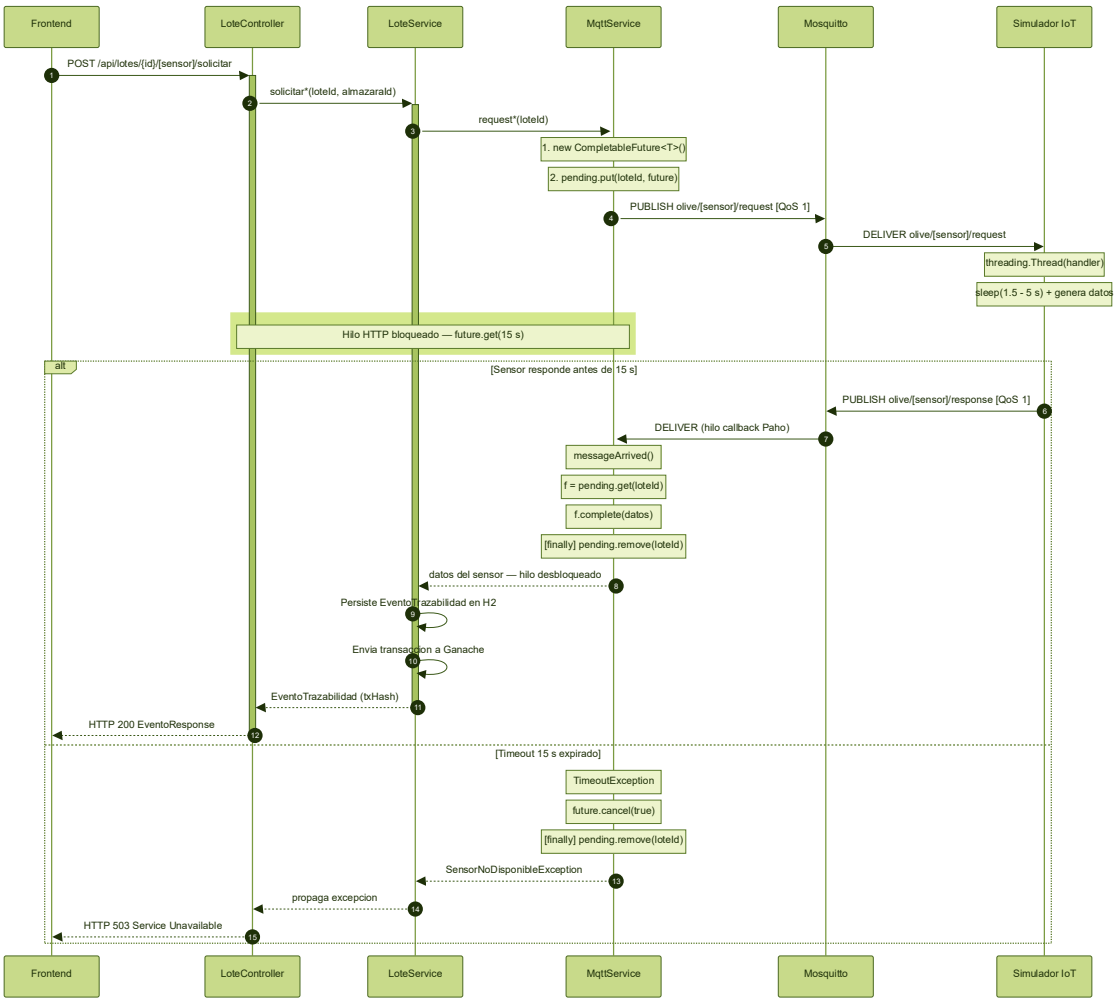


*Figura 33. Diagrama de secuencia del flujo IoT.*

## 4.8 DISEÑO DEL FRONTEND

El frontend se diseña como una aplicación de página única con cinco rutas gestionadas por React Router DOM.

| Ruta | Componente |
|---|---|
| `/` | LandingPage |
| `/registro` | RegistroFlow |
| `/campo/transporte` | TransporteAperturaPage |
| `/almazara` | AlmazaraFlow |
| `/trazabilidad` | TrazabilidadPage |

*Tabla 7. Tabla de rutas gestionadas por React Router DOM.*

Cada ruta responde a un caso de uso distinto y está diseñada de forma independiente, con su propio layout y lógica de navegación. Como se indicó en el apartado de alcance, el frontend tiene un carácter instrumental en este proyecto: su función es servir como herramienta de demostración del sistema subyacente, no como producto final orientado al usuario. La Figura 34 ilustra la estructura del frontend.

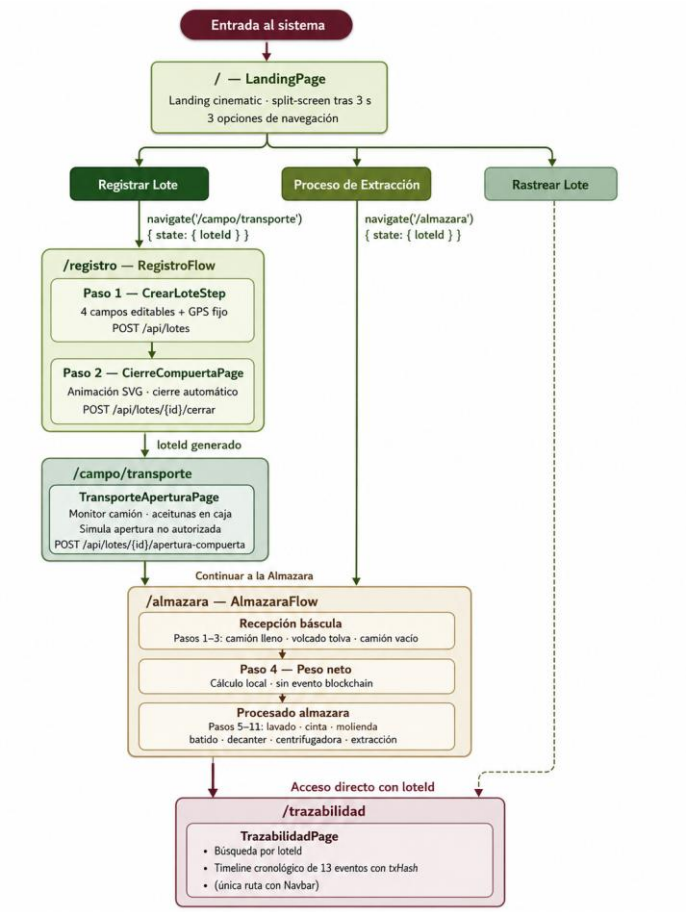


*Figura 34. Diagrama de navegación del frontend con las cinco rutas y los flujos de transición entre ellas.*

### 4.8.1 ESTRUCTURA DE RUTAS

El sistema de rutas está definido en el componente raíz `App.tsx` mediante React Router DOM. Las rutas de landing, registro, transporte y almazara no incluyen `navbar`, ya que son experiencias guiadas donde la navegación libre podría interrumpir el flujo del operario. La ruta de trazabilidad incluye navbar porque es una herramienta de consulta libre.

La ruta raíz `/` renderiza `LandingPage`, la página de entrada al sistema. La ruta `/registro` renderiza `RegistroFlow`, el flujo de registro de campo. La ruta `/campo/transporte` renderiza `TransporteAperturaPage`, la pantalla de monitorización del transporte. La ruta `/almazara` renderiza `AlmazaraFlow`, el flujo de registro en almazara. La ruta `/trazabilidad` renderiza `TrazabilidadPage`, la página de consulta de trazabilidad.

### 4.8.2 LANDING PAGE

La landing page es la primera pantalla que ve el usuario al acceder al sistema. Presenta tres opciones de navegación: el registro de un nuevo lote, el proceso de extracción en almazara y la consulta de trazabilidad.

### 4.8.3 FLUJO DE REGISTRO EN EL CAMPO

El componente `RegistroFlow` implementa el flujo de registro de campo en dos pasos secuenciales. El estado del flujo, incluyendo el paso actual y el `loteId` generado, vive en el componente raíz y se pasa a cada subcomponente mediante `props`. El primer paso renderiza el formulario de creación de lote, con campos para el identificador del agricultor, la finca de origen, el identificador del contenedor y la matrícula del camión. El segundo paso es el componente `CierreCompuertaPage`, que registra automáticamente el cierre de la compuerta del camión mostrando una animación SVG del camión con la compuerta cerrándose. Al completarse el segundo paso, el flujo navega automáticamente a `/campo/transporte` pasando el `loteId` en el estado de la ruta.

### 4.8.4 PÁGINA DE TRANSPORTE

La página de transporte muestra el estado de la monitorización del camión durante el trayecto a la almazara. Presenta una animación SVG del camión en tránsito con aceitunas visibles en la caja de carga. Incluye un botón para simular la detección de una apertura no autorizada de la compuerta, que llama al endpoint `/apertura-compuerta` y muestra el hash de transacción resultante en una tarjeta de alerta. Este evento es puramente informativo y no bloquea la continuación del flujo. El botón de continuar navega a `/almazara` pasando el `loteId` en el estado de la ruta.

### 4.8.5 FLUJO ALMAZARA

El componente `AlmazaraFlow` es el más complejo del frontend. Implementa los once pasos del proceso de almazara mediante una arquitectura de dos columnas. La columna izquierda muestra un diagrama de progreso vertical con las once estaciones del proceso, donde cada estación puede estar en estado pendiente, activo o completado, con transiciones de color animadas. La columna derecha muestra el paso actual con su animación SVG específica y los datos recibidos del sensor.

El flujo comienza con una pantalla de entrada del identificador de almazara. Una vez completado, los pasos se ejecutan secuencialmente con avance manual: cada paso conecta automáticamente con el sensor IoT al montarse, muestra la animación de conexión y recepción, presenta el dato recibido con su hash de transacción, y espera la confirmación del operario para avanzar al siguiente.

Los pasos de pesaje del camión lleno y vacío reutilizan el SVG del camión de `CierreCompuertaPage`, mostrando aceitunas en la caja de carga en el primer caso y la caja vacía en el segundo, junto con la animación de la báscula inclinándose. El paso de volcado en tolva muestra una animación del camión inclinándose y las aceitunas cayendo, sin llamada a la API IoT. El paso de peso neto es un cálculo visual que muestra la diferencia entre los dos pesajes sin generar evento blockchain. Los pasos de lavado, molienda, batido, decanter, centrifugadora y extracción muestran animaciones SVG específicas para cada proceso industrial.

Al completar el undécimo paso, el sistema muestra una pantalla de éxito con el resumen de todos los hashes de transacción obtenidos y un botón que navega a la página de trazabilidad.

### 4.8.6 PÁGINA DE TRAZABILIDAD

La página de trazabilidad permite consultar el historial completo de cualquier lote introduciendo su identificador. El sistema recupera de H2 todos los eventos ordenados cronológicamente y los presenta en una línea temporal vertical. Cada tarjeta de evento muestra el tipo de operación con su etiqueta descriptiva, el timestamp, los datos específicos

del sensor parseados desde el campo metadatos cuando aplica, y el hash de transacción con un botón de copia. Los eventos con `txHash` nulo muestran un indicador de ausencia de verificación blockchain, comunicando de forma transparente el estado de la certificación.

La página soporta los 13 tipos de evento del `enum` `TipoEvento`, con configuración visual específica para cada uno incluyendo etiqueta, subtítulo y color del borde de la tarjeta. Los metadatos JSON se parsean con manejo de errores silencioso, de forma que un campo malformado no interrumpe la visualización del resto del historial.

# Capítulo 5. DESARROLLO DEL SISTEMA

Este capítulo describe la implementación concreta del sistema, presentando los fragmentos de código más significativos de cada componente y explicando las decisiones técnicas adoptadas. El código completo está disponible en el repositorio del proyecto.

## *5.1 ESTRUCTURA DEL REPOSITORIO*

El proyecto se organiza como un monorepo con cuatro directorios principales en la raíz:

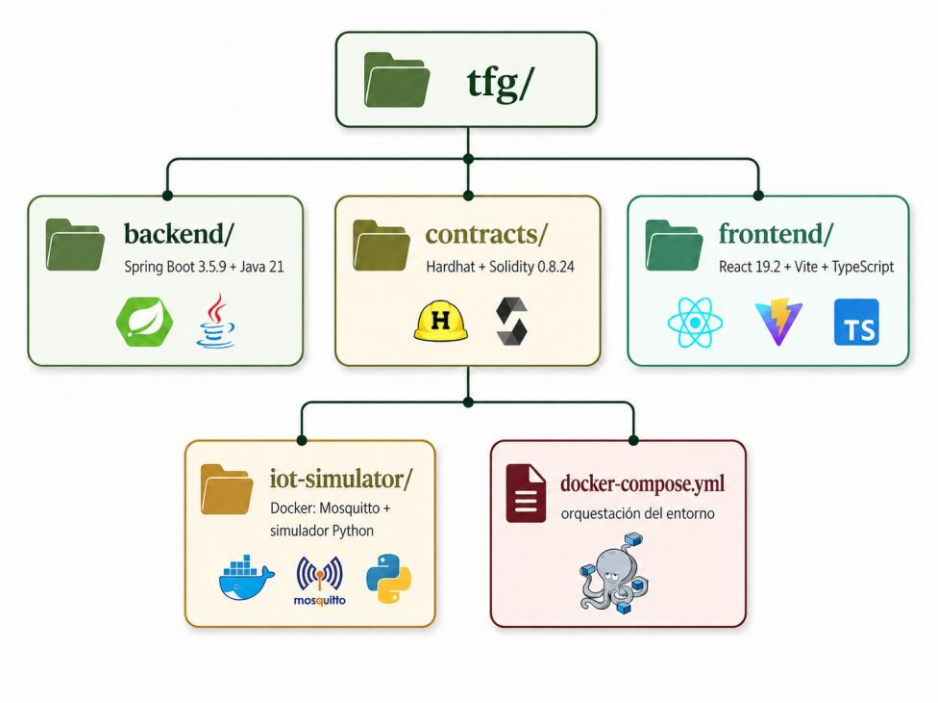


*Figura 35. Estructura del proyecto.*

El entorno completo se levanta en el siguiente orden: primero Ganache, que debe estar activo antes de desplegar el contrato; después el contrato mediante Hardhat, cuya dirección resultante se copia en `application.properties`; a continuación, el backend Spring Boot; luego la infraestructura IoT mediante `docker compose up -d`; y finalmente el frontend con `npm run dev`.

## *5.2 SMART CONTRACT CADENAACEITE*

### 5.2.1 EL CONTRATO

El contrato `CadenaAceite.sol` hereda de `Ownable` de OpenZeppelin v5 e implementa 13 eventos y 14 funciones. El constructor registra como propietario la cuenta desplegante:

```
solidypragma solidity ^0.8.24;
import "@openzeppelin/contracts/access/Ownable.sol";

contract CadenaAceite is Ownable {

    struct EventoOnChain {
        string  tipoEvento;
        uint256 pesoKg;
        uint256 timestamp;
    }

    mapping(uint256 => EventoOnChain[]) private _eventos;

    constructor() Ownable(msg.sender) {}

    function crearLote(
        uint256 loteId,
        string calldata agricultorId,
        string calldata cooperativaId,
        string calldata contenedorId,
        string calldata matriculaCamion,
        string calldata coordenadasContenedor
    ) external onlyOwner {
        _eventos[loteId].push(EventoOnChain("LOTE_CREADO", 0, block.timestamp));
        emit LoteCreado(loteId, agricultorId, cooperativaId,
            contenedorId, matriculaCamion, coordenadasContenedor,
block.timestamp);
    }

    function registrarPesajeCamionLleno(
        uint256 loteId,
        uint256 pesoKg,
        string calldata almazaraId
    ) external onlyOwner {
        _eventos[loteId].push(EventoOnChain("PESAJE_CAMION_LLENO", pesoKg,
block.timestamp));
        emit PesajeCamionLlenoRegistrado(loteId, pesoKg, almazaraId,
block.timestamp);
    }

    // ... resto de funciones con el mismo patrón

    function obtenerEventos(uint256 loteId)
        external view returns (EventoOnChain[] memory) {
        return _eventos[loteId];
    }
}
```

El contrato adopta un enfoque de almacenamiento dual: cada función añade un registro al array `interno _eventos` y emite simultáneamente un evento Solidity. El array permite recuperar el historial completo mediante `obtenerEventos()` sin un nodo de archivo, mientras que los eventos Solidity quedan indexados en los logs para su consulta eficiente por herramientas externas.

### 5.2.2 EL SCRIPT DE DESPLIEGUE

El script `deploy.js` se encarga de desplegar el contrato inteligente utilizando la API de ethers.js v6 integrada en Hardhat. Tras iniciar el despliegue, el script espera a que el contrato quede correctamente publicado mediante `contrato.waitForDeployment()` y recupera su dirección con `await contrato.getAddress()`, ambos métodos propios de la versión 6 de la librería.

Una vez completado el despliegue, el script muestra por consola las propiedades que deben configurarse en el fichero `application.properties`: `blockchain.enabled`, `blockchain.contract.address` y `blockchain.wallet.privateKey`. Además, guarda automáticamente la dirección del contrato, la cuenta desplegante y el timestamp del despliegue en el fichero `contracts/deploy-output.json`, reduciendo así los pasos manuales y evitando posibles errores de configuración.

Durante el desarrollo se detectó que, si la variable de entorno `GANACHE_PRIVATE_KEY` no está definida al ejecutar Hardhat, el fichero `hardhat.config.js` configura la lista de cuentas como vacía mediante `accounts: []`. En consecuencia, la llamada a `ethers.getSigners()` devuelve un array vacío y el script falla con un error poco descriptivo relacionado con una propiedad `address` indefinida. Para evitar este problema, la clave privada debe proporcionarse explícitamente al ejecutar el despliegue:

```
GANACHE_PRIVATE_KEY=0x...
npx hardhat run scripts/deploy.js --network ganache
```

## *5.3 BACKEND SPRING BOOT*

### 5.3.1 MODELO DE DOMINIO

La capa de dominio del backend representa los conceptos principales del sistema desde el punto de vista de la aplicación. En esta capa se modelan los lotes de aceituna y los eventos de trazabilidad asociados a cada lote, manteniendo una representación persistente en base de datos que complementa al registro inmutable realizado en blockchain. Mientras que el smart contract actúa como capa de certificación y verificación externa, el dominio del backend proporciona una estructura más flexible para gestionar la información necesaria por la aplicación, la API REST y el frontend.

La entidad `Lote` contiene la información básica del lote y los datos necesarios para identificar su origen. Además de la clave primaria numérica utilizada internamente por la base de datos, incorpora un identificador legible con formato `LOT-{año}-{secuencia}`, generado por el sistema para facilitar su consulta por parte del operario. Este identificador permite buscar y comunicar un lote de forma sencilla, mientras que la clave primaria numérica se emplea para vincular de forma eficiente la base de datos con el contrato inteligente.

La entidad dispone de tres constructores para cubrir distintos escenarios de creación. El constructor protegido sin argumentos es requerido por JPA para poder instanciar la entidad. El constructor básico mantiene compatibilidad con el formato inicial del sistema, mientras que el constructor completo permite crear lotes con todos los datos de trazabilidad de origen enviados desde el frontend, como el contenedor, la matrícula del camión o las coordenadas GPS. Los eventos se representan mediante la entidad `EventoTrazabilidad`, que almacena el tipo de evento, el instante de registro, el peso cuando aplica, el hash de la transacción blockchain y un campo de metadatos.

### 5.3.2 CAPA DE SERVICIO

`LoteService` es el núcleo de la aplicación. Sus dos decisiones de diseño más relevantes son la gestión de dependencias opcionales y el patrón de escritura dual.

Las dependencias de blockchain y MQTT se inyectan como `Optional<T>`, de forma que Spring inyecta `Optional.empty()` cuando los servicios están desactivados por configuración. Esto permite que el sistema arranque y funcione correctamente, aunque Ganache o Mosquitto no estén disponibles:

```
@Service
public class LoteService {

    private final Optional<CadenaAceiteWrapper> blockchain;
    private final Optional<MqttService> mqttService;

    public LoteService(...,
                       Optional<CadenaAceiteWrapper> blockchain,
                       Optional<MqttService> mqttService) {
        this.blockchain  = blockchain;
        this.mqttService = mqttService;
    }
}
```

El método `crearLote` ilustra el patrón de escritura dual y el manejo de los nuevos campos de trazabilidad:

```
public Lote crearLote(String agricultorId, String origen, String cooperativaId,
                      String contenedorId, String matriculaCamion,
                      String coordenadasContenedor) {

    Lote lote = crearLoteConRetry(agricultorId, origen,
        contenedorId, matriculaCamion, coordenadasContenedor);

    String txHash = enviarABlockchain(() ->
        blockchain.get().crearLote(
            BigInteger.valueOf(lote.getId()), agricultorId, cooperativaId,
            contenedorId, matriculaCamion, coordenadasContenedor
        )
    );

    String metadatos = buildJson(
        "\"contenedorId\":\"" + escapeJson(nullToEmpty(contenedorId)) + "\"",
        "\"matriculaCamion\":\"" + escapeJson(nullToEmpty(matriculaCamion)) +
"\"",
        "\"coordenadasContenedor\":\"" +
escapeJson(nullToEmpty(coordenadasContenedor)) + "\""
    );

    eventoRepository.save(new EventoTrazabilidad(
        lote.getLoteId(), TipoEvento.LOTE_CREADO, null, txHash,
        cooperativaId, null, null, metadatos
    ));
    return lote;
}
```

El método `crearLoteConRetry` resuelve el problema de condición de carrera en la generación del identificador de lote. Si dos peticiones concurrentes generan el mismo `loteId`, la restricción `UNIQUE` de la base de datos provoca una `DataIntegrityViolationException`. En lugar de propagar el error, el método reintenta automáticamente hasta cinco veces con un offset incremental:

```
private Lote crearLoteConRetry(String agricultorId, String origen,
                               String contenedorId, String matriculaCamion,
                               String coordenadasContenedor) {
    int intentos = 0;
    while (true) {
        String loteId = generarLoteId(intentos);
        try {
            return loteRepository.save(
                new Lote(loteId, agricultorId, origen,
                    contenedorId, matriculaCamion, coordenadasContenedor)
            );
        } catch (DataIntegrityViolationException e) {
            intentos++;
            if (intentos >= 5) throw e;
            log.warn("Colisión de loteId '{}', reintentando ({}/5)...", loteId,
intentos);
        }
    }
}
```

El método privado `enviarABlockchain` centraliza toda la lógica de interacción con blockchain mediante una interfaz funcional:

```
private String enviarABlockchain(BlockchainCall call) {
    if (blockchain.isEmpty()) {
        log.debug("Blockchain deshabilitada. Saltando llamada.");
        return null;
    }
    try {
        return call.execute();
    } catch (Exception e) {
        log.warn("Error blockchain: {}. El evento se guarda solo en H2.",
e.getMessage());
        return null;
    }
}

@FunctionalInterface
private interface BlockchainCall {
    String execute() throws Exception;
}
```

Este patrón garantiza que cualquier fallo en la comunicación con Ganache no interrumpe el flujo de negocio: el evento se persiste en H2 con `txHash` nulo y la aplicación continúa.

El método `registrarVolcadoTolva` ilustra el patrón de los eventos de almazara que no tienen sensor IoT asociado:

```
@Transactional
public EventoTrazabilidad registrarVolcadoTolva(String loteId, String
almazaraId) {
    Lote lote = loteRepository.findByLoteId(loteId)
        .orElseThrow(() -> new LoteNoEncontradoException(loteId));

    String txHash = enviarABlockchain(() ->
        blockchain.get().registrarVolcadoTolva(
            BigInteger.valueOf(lote.getId()), almazaraId)
    );

    String metadatos = buildJson(
        "\"almazaraId\":\"" + escapeJson(almazaraId) + "\""
    );

    return eventoRepository.save(new EventoTrazabilidad(
        loteId, TipoEvento.VOLCADO_TOLVA, null, txHash,
        null, almazaraId, null, metadatos
    ));
}
```

Los métodos de solicitud IoT siguen un patrón uniforme: verifican que MQTT está disponible, llaman al método de `MqttService` correspondiente y delegan en el método de persistencia:

```
public EventoTrazabilidad registrarPesajeCamionLleno(String loteId, String
almazaraId) {
    if (mqttService.isEmpty()) {
        throw new SensorNoDisponibleException(
            "MQTT deshabilitado. Use el endpoint /pesaje-camion-lleno con peso
manual."
        );
    }
    int pesoKg = mqttService.get().requestPesajeCamionLleno(loteId);
    return persistirPesajeCamionLleno(loteId, pesoKg, almazaraId);
}
```

### 5.3.3 CAPA DE API

`LoteController` expone los 23 endpoints REST. Los DTOs se implementan como records de Java 21. Cada tipo de sensor tiene su propio record de petición con validaciones específicas mediante `Bean Validation`.

El `controller` gestiona tres tipos de error con manejadores específicos: `LoteNoEncontradoException` devuelve 404, `SensorNoDisponibleException` devuelve 503 y

`MethodArgumentNotValidException` devuelve 400 con un mapa que identifica cada campo con su mensaje de error.

## *5.4 INTEGRACIÓN BLOCKCHAIN CON WEB3J*

### 5.4.1 CONFIGURACIÓN CONDICIONAL

BlockchainConfig utiliza `@ConditionalOnProperty` para que Spring no procese la clase cuando `blockchain.enabled` no es true. El `chainId` se obtiene dinámicamente del nodo en el arranque mediante `web3j.ethChainId()`, lo que garantiza que la firma EIP-155 siempre usa el valor correcto. El `factory method` está anotado con `@Nullable` para que Spring inyecte `Optional.empty()` en lugar de fallar en el arranque cuando Ganache no está disponible.

### 5.4.2 EL WRAPPER MANUAL

`CadenaAceiteWrapper` implementa el flujo completo de envío de transacciones usando las APIs primitivas de Web3j. El método privado `sendTransaction` ejecuta cuatro pasos. El primero obtiene el `nonce` de la cuenta firmante mediante `eth_getTransactionCount`. El segundo construye la transacción sin firmar con `RawTransaction.createTransaction`. El tercero firma con EIP-155 mediante `TransactionEncoder.signMessage(rawTransaction, chainId, credentials)`. El cuarto envía la transacción firmada al nodo mediante `eth_sendRawTransaction`:

```
private String sendTransaction(Function function) throws IOException {
    String encodedFunction = FunctionEncoder.encode(function);

    BigInteger nonce = web3j
        .ethGetTransactionCount(credentials.getAddress(),
DefaultBlockParameterName.LATEST)
        .send()
        .getTransactionCount();

    RawTransaction rawTransaction = RawTransaction.createTransaction(
        nonce, GAS_PRICE, GAS_LIMIT, contractAddress, BigInteger.ZERO,
encodedFunction
    );

    byte[] signedMessage = TransactionEncoder.signMessage(rawTransaction,
chainId, credentials);

    EthSendTransaction response = web3j
```

```
            .ethSendRawTransaction(Numeric.toHexString(signedMessage))
            .send();

        if (response.hasError()) {
            throw new IOException("Error en blockchain: " +
response.getError().getMessage());
        }
        String hash = response.getTransactionHash();
        if (hash == null || hash.isBlank()) {
            throw new IOException(
                "Ganache no devolvió txHash. Causas: (1) contrato no desplegado en "
                + contractAddress + ", (2) chainId incorrecto, (3) nonce ya usado."
            );
        }
        return hash;
}
```

El wrapper incluye un método auxiliar `nullToEmpty` que convierte valores nulos a cadenas vacías antes de pasarlos al codificador ABI, ya que Solidity acepta strings vacíos pero no nulos:

```
private static String nullToEmpty(String value) {
    return value != null ? value : "";
}
```

El gas price está fijado en 20 Gwei y el gas limit en 300.000 unidades, suficiente para todas las funciones del contrato que consumen aproximadamente 80.000 unidades cada una.

## *5.5 INTEGRACIÓN IOT CON MQTT*

### 5.5.1 CONFIGURACIÓN DEL CLIENTE PAHO

`MqttConfig` crea el bean del cliente Paho con `MemoryPersistence` en lugar del persistidor de ficheros por defecto, evitando la creación de directorios no deseados en el directorio del proyecto. La opción `setAutomaticReconnect(true)` garantiza la reconexión automática si el broker se reinicia.

### 5.5.2 EL SERVICIO MQTT

`MqttService` gestiona once pares de tópicos request/response mediante once `ConcurrentHashMap` independientes, uno por tipo de dato. Esta separación evita colisiones de clave entre sensores que podrían procesar solicitudes concurrentes del mismo lote:

```
private final ConcurrentHashMap<String, CompletableFuture<Integer>>
    pendingScale      = new ConcurrentHashMap<>(),
```

```
    pendingCinta     = new ConcurrentHashMap<>(),
    pendingPostLimpieza = new ConcurrentHashMap<>(),
    pendingBatido    = new ConcurrentHashMap<>(),
    pendingCamionLleno = new ConcurrentHashMap<>(),
    pendingCamionVacio = new ConcurrentHashMap<>(),
    pendingMolienda  = new ConcurrentHashMap<>();

private final ConcurrentHashMap<String, CompletableFuture<LavadoData>>
    pendingLavado    = new ConcurrentHashMap<>();

private final ConcurrentHashMap<String, CompletableFuture<DecanterData>>
    pendingDecanter  = new ConcurrentHashMap<>();

private final ConcurrentHashMap<String, CompletableFuture<ExtraccionData>>
    pendingExtraccion = new ConcurrentHashMap<>();

private final ConcurrentHashMap<String, CompletableFuture<CentrifugadoraData>>
    pendingCentrifugadora = new ConcurrentHashMap<>();
El patrón de puente síncrono-asíncrono es el mismo para todos los sensores. El
método requestPesajeCamionLleno ilustra el patrón para sensores simples de tipo
entero:
javapublic int requestPesajeCamionLleno(String loteId) {
    CompletableFuture<Integer> future = new CompletableFuture<>();
    pendingCamionLleno.put(loteId, future);
    try {
        publish(topicCamionLlenoRequest,
            "{\"loteId\":\"" + loteId + "\"}");
        return future.get(timeoutSeconds, TimeUnit.SECONDS);
    } catch (TimeoutException e) {
        future.cancel(true);
        throw new SensorNoDisponibleException(
            "El sensor de pesaje no respondió en " + timeoutSeconds + "s");
    } catch (Exception e) {
        future.cancel(true);
        throw new SensorNoDisponibleException(
            "Error en comunicación MQTT: " + e.getMessage());
    } finally {
        pendingCamionLleno.remove(loteId);
    }
}
```

El callback `messageArrived` enruta cada mensaje al mapa pendiente correcto según el tópico de respuesta recibido, parsea el JSON del payload y completa el `future` correspondiente con los datos del sensor.

Los records de datos compuestos encapsulan las respuestas de los sensores con múltiples campos:

```
public record LavadoData(boolean aguaApta, int temperaturaAgua, int phAgua) {}
public record DecanterData(int litrosAceite, int kgAlpeorujo) {}
public record ExtraccionData(int litrosAceiteTotal, int rendimientoPorcentaje) {}
public record CentrifugadoraData(int revoluciones, int temperatura) {}
```

### 5.5.3 EL SIMULADOR PYTHON

El simulador implementa once handlers independientes, uno por sensor, cada uno ejecutado en un hilo separado mediante `threading.Thread`. El uso de hilos independientes es imprescindible porque el callback `on_message` de Paho se ejecuta en el hilo de red interno: bloquearlo con `time.sleep()` impediría procesar más mensajes durante la espera. Los rangos de datos están calibrados para respetar la coherencia física del proceso.

## *5.6 FRONTEND REACT*

### 5.6.1 CAPA DE COMUNICACIÓN HTTP

El módulo `api.ts` centraliza las 26 funciones de comunicación con el backend y define las interfaces TypeScript que representan el contrato de la API. La interfaz `EventoResponse` incluye los campos opcionales `cooperativaId`, `almazaraId`, `esAutorizada` y metadatos añadidos para los nuevos tipos de evento. El `helper` genérico `handleResponse<T>` centraliza el manejo de errores HTTP: cuando el backend devuelve un código de error, extrae el texto del `body` y lo convierte en un error JavaScript con el mensaje específico del dominio.

### 5.6.2 EL FLUJO DE REGISTRO DE CAMPO

`RegistroFlow` gestiona el flujo de campo en dos pasos. El primer paso es el formulario de creación de lote con los cinco campos, de los cuales tres son opcionales. Al confirmar, el sistema navega automáticamente al segundo paso, `CierreCompuertaPage`, que gestiona el pesaje mediante MQTT y el cierre de compuerta, mostrando la animación SVG del camión. Al completarse, navega a `/campo/transporte` pasando el `loteId` en el estado de la ruta.

### 5.6.3 LA PÁGINA DE TRANSPORTE

`TransporteAperturaPage` muestra una animación SVG del camión en tránsito con aceitunas visibles en la caja de carga. El botón de simulación de apertura no autorizada utiliza un `useRef` como guard de ejecución única para evitar múltiples registros ante clics repetidos. El

evento se registra en blockchain y el hash de transacción aparece en una tarjeta de alerta sin interrumpir el flujo.

### 5.6.4 ALMAZARAFLOW

AlmazaraFlow implementa los once pasos mediante el hook personalizado useIotFlow, compartido por todos los pasos con sensor IoT. El hook encapsula la máquina de estados `connecting → receiving → confirmed → error`, el guard `useRef` contra la doble ejecución en StrictMode, el timer de avance automático y la lógica de reintento:

```
function useIotFlow<T>(
  apiCall: () => Promise<T>,
  onSuccess: (result: T) => void
) {
  const [phase, setPhase] = useState<IotPhase>('connecting');
  const requestRef = useRef(false);
  const timerRef   = useRef<ReturnType<typeof setTimeout> | null>(null);

  const execute = useCallback(async () => {
    if (requestRef.current) return;
    requestRef.current = true;
    try {
      const result = await apiCall();
      setPhase('confirmed');
      timerRef.current = setTimeout(() => onSuccess(result), 2000);
    } catch (err) {
      setPhase('error');
    }
  }, [apiCall, onSuccess]);

  useEffect(() => {
    execute();
    return () => { if (timerRef.current) clearTimeout(timerRef.current); };
  }, [execute]);

  function handleRetry() {
    requestRef.current = false;
    setPhase('connecting');
    execute();
  }

  return { phase, handleRetry };
}
```

El paso del volcado en tolva es el único sin llamada a la API IoT. Utiliza un `useEffect` con una secuencia de `setTimeout` que anima el camión inclinándose y las aceitunas cayendo.

### 5.6.5 TRAZABILIDADPAGE

La página de trazabilidad soporta los 14 tipos de evento del `enum` con configuración visual específica para cada uno. Los metadatos JSON se parsean con manejo de errores silencioso mediante try/catch, de forma que un campo malformado no interrumpe la visualización del resto del historial. Para los eventos con metadatos, la página muestra los datos específicos del sensor en una cuadrícula compacta debajo del hash de transacción.

## *5.7 PROBLEMAS ENCONTRADOS Y SOLUCIONES ADOPTADAS*

El desarrollo del sistema presentó una serie de problemas técnicos cuya resolución contribuyó a la calidad final del código. Se documentan los más relevantes a continuación.

### 5.7.1 INCOMPATIBILIDAD DE WEB3J 4.12.2 CON EL WRAPPER GENERADO AUTOMÁTICAMENTE

El wrapper Java generado por el plugin de Web3j utilizaba la clase `TransactionException`, no disponible en el artefacto `web3j:core` sino en `web3j:contracts`, que introducía conflictos de versión con el BOM de Spring Boot 3. La solución fue implementar un wrapper manual con las APIs primitivas de Web3j, que además ofrece control total sobre cada paso del proceso de transacción.

### 5.7.2 GANACHE RECHAZABA LAS TRANSACCIONES SILENCIOSAMENTE

Durante las primeras pruebas, Ganache devolvía `txHash: null` sin error explícito. El diagnóstico reveló que el `chainId` utilizado en la firma era incorrecto. La solución fue obtener el chainId dinámicamente del nodo mediante `web3j.ethChainId()` en el arranque, en lugar de leerlo de la configuración.

### 5.7.3 EL BACKEND NO ARRANCABA CUANDO GANACHE NO ESTABA ACTIVO

Con `blockchain.enabled=true` pero Ganache apagado, Spring abortaba el arranque del contexto porque el `factory method` de `CadenaAceiteWrapper` lanzaba la excepción de

conexión. La solución fue anotar el `factory method` con `@Nullable` y capturar la excepción devolviendo `null`, lo que hace que Spring inyecte `Optional.empty()` en `LoteService`.

### 5.7.4 PAHO CREABA UN DIRECTORIO NO DESEADO EN EL PROYECTO

El persistidor por defecto de Paho creaba un directorio en el directorio de trabajo con el nombre derivado del `clientId` y la URL del broker. La solución fue usar `MemoryPersistence`, suficiente para el escenario de simulación donde la pérdida de mensajes en caída del proceso es aceptable.

### 5.7.5 EJECUCIÓN DUPLICADA DE LOS EFECTOS DE IOT DOS VECES

React StrictMode monta los componentes dos veces en desarrollo, lo que provocaba dos peticiones al sensor IoT y dos transacciones blockchain para el mismo lote. La solución fue el guard con `useRef` en el hook `useIotFlow`, que garantiza que la solicitud se lanza exactamente una vez por montaje.

### 5.7.6 COLISIONES DE LOTEID EN PETICIONES CONCURRENTES

Dos peticiones simultáneas podían generar el mismo `loteId`, provocando una `DataIntegrityViolationException` en la segunda inserción. La solución fue el método `crearLoteConRetry`, que captura la excepción y reintenta hasta cinco veces con un offset incremental en el contador.

# Capítulo 6. ANÁLISIS DE RESULTADOS

Este capítulo presenta las pruebas realizadas sobre el sistema implementado y los resultados obtenidos, con el objetivo de verificar que el sistema satisface los requisitos funcionales y no funcionales definidos en el capítulo 4. Las pruebas se realizaron con el entorno completo activo: Ganache Desktop en el puerto 7545 con `chainId` 1337, el contrato desplegado en la dirección 0xcd99A25153bc49510EAEB897c713996e83284cd3, el backend Spring Boot en el puerto 8080, la infraestructura IoT levantada con Docker Compose y el frontend accesible en el puerto 5173.

## *6.1 ENTORNO DE PRUEBAS*

Las pruebas se realizaron sobre el siguiente entorno. El backend Spring Boot 3.5.9 se ejecutó sobre Java 21.0.2 con la base de datos H2 en memoria y blockchain habilitada. El contrato `CadenaAceite` se desplegó en Ganache Desktop con `chainId` 1337 y minado instantáneo. La infraestructura IoT se levantó mediante Docker Compose con Eclipse Mosquitto 2 como broker y el simulador Python con paho-mqtt 1.6.1. El frontend React 19.2 se sirvió mediante el servidor de desarrollo de Vite.

Las pruebas funcionales del backend se realizaron mediante peticiones HTTP con curl desde línea de comandos, verificando de forma independiente el comportamiento de la API sin pasar por el frontend. Los resultados de las transacciones blockchain se verificaron en la interfaz gráfica de Ganache Desktop. Las pruebas automatizadas se ejecutaron mediante Maven Surefire sin ningún servicio externo activo, usando mocks de Mockito para aislar las dependencias de blockchain y MQTT.

## *6.2 PRUEBAS DE LA API REST*

Las pruebas de la API REST verifican el comportamiento de los 23 endpoints del sistema sobre el lote LOT-2026-0003.

- Creación de lote con trazabilidad de origen

```
curl -X POST http://localhost:8080/api/lotes \
  -H "Content-Type: application/json" \
  -d '{"agricultorId":"AGR-001","origen":"Cortijo El Molino, Jaén",
       "contenedorId":"CONT-2026-001","matriculaCamion":"1234 BCD",
       "coordenadasContenedor":"37.7749,-4.7796"}'
```

Respuesta HTTP 201:

```
{
  "id": 3,
  "loteId": "LOT-2026-0003",
  "agricultorId": "AGR-001",
  "origen": "Cortijo El Molino, Jaén",
  "fechaCreacion": "2026-06-06T07:17:20.141589Z",
  "contenedorId": "CONT-2026-001",
  "matriculaCamion": "1234 BCD",
  "coordenadasContenedor": "37.7749,-4.7796"
}
```

El sistema generó automáticamente el identificador LOT-2026-0003 y persistió los cinco campos de trazabilidad de origen. El campo id: 3 es el identificador numérico interno utilizado como clave en el smart contract.

- Pesaje del camión lleno en almazara

```
curl -X POST http://localhost:8080/api/lotes/LOT-2026-0003/pesaje-camion-
lleno/solicitar \
  -H "Content-Type: application/json" \
  -d '{"almazaraId":"ALM-JAEN-01"}'
```

Respuesta HTTP 200:

```
{
  "tipoEvento": "PESAJE_CAMION_LLENO",
  "pesoKg": 7428,
  "timestamp": "2026-06-06T07:20:04.587801700Z",
  "txHash": "0x27d1edebfbb2b1dfde0dbf80dfe9197a86b35cff984e33ad8a630a6fb97f0ebe",
  "almazaraId": "ALM-JAEN-01",
  "metadatos": "{\"pesoKg\":7428,\"almazaraId\":\"ALM-JAEN-01\"}"
}
```

- Pesaje del camión vacío en almazara

Respuesta HTTP 200:

```
{
  "tipoEvento": "PESAJE_CAMION_VACIO",
  "pesoKg": 1077,
  "txHash": "0x22ec7a2ec576ac8de6daf8cf933053ee8d420235770de040731a15b301d3dc3a",
  "metadatos": "{\"pesoKg\":1077,\"almazaraId\":\"ALM-JAEN-01\"}"
}
```


El peso neto de aceituna calculado por diferencia es 7.428 - 1.077 = 6.351 kg, coherente con los rangos configurados en el simulador.

- Lavado con control de calidad del agua

Respuesta HTTP 200:


```
{
  "tipoEvento": "LAVADO",
  "txHash": "0x431d9907f7f8531b79591f59d20adb6d8f4013d41307bb593f8328166258a5ed",
  "metadatos":
"{\"aguaApta\":true,\"temperaturaAgua\":153,\"phAgua\":75,\"almazaraId\":\"ALM-
JAEN-01\"}"
}
```


Los valores del sensor se interpretan aplicando la convención de factor 10: temperatura de 15,3°C y pH de 7,5, ambos dentro de los rangos óptimos para el lavado de aceituna.

- Temperatura de batido

Respuesta HTTP 200:


```
{
  "tipoEvento": "TEMPERATURA_BATIDO",
  "txHash": "0x724fa79143b9953da958ea35a5b379b5c7c0e35acfb944a80a16f4aee02f03ad",
  "metadatos": "{\"temperaturaC\":246,\"almazaraId\":\"ALM-JAEN-01\"}"
}
```


La temperatura de 24,6°C está por debajo del límite de 27°C establecido para la certificación de aceite de oliva virgen extra, lo que confirma que el lote cumple las condiciones de calidad.

- Decanter

Respuesta HTTP 200:


```
{
  "tipoEvento": "DECANTER",
  "txHash": "0x97bf6b08608b04c797b173243af1c8fd0081604583415 39f816b15caeaadf5e2",
  "metadatos": "{\"litrosAceite\":1444,\"kgAlpeorujo\":3180,\"almazaraId\":\"ALM-
JAEN-01\"}"
}
```


- Centrifugadora

Respuesta HTTP 200:


```
{
  "tipoEvento": "CENTRIFUGADORA",
  "txHash": "0xd25c83fa9e54fce0a2ddf0cd0d5b1a1975b005ddbb8965e0c4501460849c7c78",
  "metadatos": "{\"revoluciones\":3667,\"temperatura\":239,\"almazaraId\":\"ALM-JAEN-01\"}"
}
```


Las 3.667 rpm y una temperatura de 23,9°C son valores representativos de una centrífuga vertical en condiciones normales de operación.

- Extracción finalizada

Respuesta HTTP 200:


```
{
  "tipoEvento": "EXTRACCION_FINALIZADA",
  "txHash": "0xd09e24903b053567f2c4bfd2c2f85952faf130746700bd6a707d77626d22cdaf",
  "metadatos":
"{\"litrosAceiteTotal\":1502,\"rendimientoPorcentaje\":167,\"almazaraId\":\"ALM-JAEN-01\"}"
}
```


El rendimiento de 16,7% es coherente con los valores típicos de extracción de aceite de oliva, que oscilan entre el 15% y el 25% según la variedad y el estado de madurez de la aceituna.

- Consulta de trazabilidad completa

```
curl http://localhost:8080/api/lotes/LOT-2026-0003/trazabilidad
```

A continuación, se muestra un fragmento representativo de la respuesta HTTP 200 obtenida para el lote LOT-2026-0001, con los 13 eventos en orden cronológico y su `txHash` válido:


```
{
  "loteId": "LOT-2026-0001",
  "eventos": [
    {
      "tipoEvento": "LOTE_CREADO",
      "pesoKg": null,
      "timestamp": "2026-06-10T03:15:01.922787Z",
      "txHash":
"0x494b1f3472e881ea1c2a2ab1d2a754faf1a1050f7d13a855284fcc8283636d18",
      "cooperativaId": null,
      "almazaraId": null,
      "esAutorizada": null,
      "metadatos": "{\"contenedorId\":\"CONT-2026-001\",\"matriculaCamion\":\"1234 BCD\",\"coordenadasContenedor\":\"37.7749,-4.7796\"}"
    },
    {
      "tipoEvento": "CAMION_CERRADO",
```

```
      "pesoKg": null,
      "timestamp": "2026-06-10T03:15:06.161995Z",
      "txHash":
"0x29683d293ce6995428f86f320befbcc58d40011f5f2ab941ce09dd91a2c77f3f",
      "cooperativaId": null,
      "almazaraId": null,
      "esAutorizada": null,
      "metadatos": null
    },

    // ...

    {
      "tipoEvento": "EXTRACCION_FINALIZADA",
      "pesoKg": null,
      "timestamp": "2026-06-10T03:16:13.189949Z",
      "txHash":
"0x54aca68da091926581cbd1bd5da54bdbf1c3f2224708f4693bd7291a45c10e46",
      "cooperativaId": null,
      "almazaraId": "ALM-JAEN-01",
      "esAutorizada": null,
      "metadatos":
"{\"litrosAceiteTotal\":1177,\"rendimientoPorcentaje\":196,\"almazaraId\":\"ALM-
JAEN-01\"}"
    }
  ]
}
```


- Manejo de error 404

```
curl http://localhost:8080/api/lotes/LOT-2026-9999/trazabilidad
```

Respuesta HTTP 404:

```
Lote no encontrado: LOT-2026-9999
```

El sistema devuelve el código 404 con un mensaje específico del dominio que identifica el lote inexistente, satisfaciendo el requisito RNF-07 de trazabilidad de errores.

## *6.3 VERIFICACIÓN BLOCKCHAIN*

Los 13 eventos del lote LOT-2026-0005 generaron 13 transacciones distintas en Ganache, cada una con su hash único.


*Figura 36. Captura de la pestaña Transactions de Ganache Desktop mostrando una transacción del lote LOT-2026-0005 con su hash completo y el contrato destinatario.*

Como parte de la verificación blockchain, se comprobó en Ganache Desktop que las transacciones generadas por el backend quedaban efectivamente registradas en la red local. La Figura 36 muestra el detalle de una de las transacciones asociadas al lote LOT-2026-0005, donde puede observarse que se trata de una llamada a contrato (`CONTRACT CALL`) dirigida a la dirección del smart contract `CadenaAceite`.

La captura permite verificar varios aspectos relevantes del funcionamiento del sistema. En primer lugar, el campo `TO CONTRACT ADDRESS` confirma que la transacción se envía al contrato desplegado en Ganache, y no a una cuenta externa. En segundo lugar, el campo `VALUE` aparece con valor `0.00 ETH`, lo que indica que la operación no implica una transferencia económica, sino únicamente la ejecución de lógica del contrato inteligente. Asimismo, el consumo de gas registrado, del orden de 80.000 unidades, es coherente con una operación de escritura sobre el estado del contrato, en la que se almacena un evento de trazabilidad y se emite el evento Solidity correspondiente.

Además, la presencia de datos en el campo `TX DATA` confirma que la transacción incluye una llamada codificada a una función del contrato, con sus parámetros serializados según la ABI de Ethereum. Por tanto, no se trata de una simulación ni de una operación puramente interna del backend, sino de una transacción real enviada al nodo, minada en un bloque de la red blockchain local y asociada a un hash único.

La verificación en Ganache confirma que los eventos del lote generan transacciones independientes en la blockchain local y que cada una de ellas queda identificada por un

`txHash` propio. Dado el minado instantáneo de Ganache, el tiempo entre el envío de la petición HTTP desde el frontend y la inclusión de la transacción en un bloque fue inferior a un segundo en las pruebas realizadas.

Esta comprobación permite validar el funcionamiento extremo a extremo del sistema: el frontend inicia la operación, el backend procesa la petición, interactúa con el contrato inteligente mediante Web3j, firma y envía la transacción, y Ganache la incorpora a un bloque. De este modo, el hash almacenado en H2 puede contrastarse posteriormente con el registro on-chain, permitiendo detectar cualquier discrepancia entre la base de datos local y la información certificada en blockchain.

La propiedad de inmutabilidad queda demostrada por la propia naturaleza del registro blockchain: una vez incluida una transacción en un bloque, la modificación posterior de los datos almacenados en H2 no altera el registro on-chain. Por ello, cualquier actor con acceso al hash de transacción puede verificar de forma independiente que el evento fue registrado en el contrato, lo que refuerza la trazabilidad del sistema frente a esquemas basados únicamente en registros centralizados.

## 6.4 PRUEBAS DE LA INTEGRACIÓN IOT

La integración MQTT se verificó implícitamente en las pruebas de los endpoints `/solicitar`, que requieren que el broker Mosquitto y el simulador Python estén activos y comunicados. Los logs del backend durante las pruebas muestran el ciclo completo de comunicación para cada sensor:

```
MQTT: solicitud pesaje-camion-lleno enviada para loteId=LOT-2026-0004
MQTT: [camionlleno] peso=7631 kg para loteId=LOT-2026-0004
MQTT: solicitud pesaje-camion-vacio enviada para loteId=LOT-2026-0004
MQTT: [camionvacio] peso=1724 kg para loteId=LOT-2026-0004
MQTT: solicitud lavado enviada para loteId=LOT-2026-0004
MQTT: [lavado] temp=185 ph=69 aguaApta=true para loteId=LOT-2026-0004
MQTT: solicitud pesaje-cinta enviada para loteId=LOT-2026-0004
MQTT: [cinta] peso=4203 kg para loteId=LOT-2026-0004
MQTT: solicitud molienda enviada para loteId=LOT-2026-0004
MQTT: [molienda] temperaturaC=241 (24.1°C) para loteId=LOT-2026-0004
MQTT: solicitud temperatura-batido enviada para loteId=LOT-2026-0004
MQTT: [batido] temperaturaC=253 (25.3°C) para loteId=LOT-2026-0004
MQTT: solicitud decanter enviada para loteId=LOT-2026-0004
```

```
  MQTT: [decanter] litrosAceite=1180 kgAlpeorujo=3420 para loteId=LOT-2026-0004
  MQTT: solicitud centrifugadora enviada para loteId=LOT-2026-0004
  MQTT: [centrifugadora] revoluciones=3750 temperatura=205 para loteId=LOT-2026-
0004
  MQTT: solicitud extraccion-finalizada enviada para loteId=LOT-2026-0004
  MQTT: [extraccion] litrosAceiteTotal=1180 rendimientoPorcentaje=197 para
loteId=LOT-2026-0004
```

Los retardos observados entre la solicitud y la respuesta de cada sensor oscilaron entre 1,5 y 5 segundos, coherentes con los retardos aleatorios configurados en el simulador para imitar el tiempo de medición de sensores industriales reales. El timeout de 15 segundos no se activó en ninguna de las pruebas realizadas.

La resiliencia del sistema ante la pérdida de conexión MQTT fue verificada durante las pruebas. Cuando el broker Mosquitto se detuvo de forma imprevista, el log del backend registró el aviso correspondiente y el cliente Paho intentó reconectar automáticamente gracias a la opción `setAutomaticReconnect(true)`. Los endpoints `/solicitar` invocados durante la desconexión devolvieron el código 503 con el mensaje correspondiente, sin afectar al funcionamiento del resto del sistema.

## 6.5 *SUITE DE TESTS AUTOMATIZADOS*

La suite de tests automatizados del backend comprende 42 casos de prueba distribuidos en tres clases, con resultado de 0 fallos y 0 errores en la ejecución final:

```
  Tests run: 25, Failures: 0, Errors: 0  — LoteControllerTest
  Tests run: 16, Failures: 0, Errors: 0  — LoteServiceTest
  Tests run:  1, Failures: 0, Errors: 0  —
OliveTraceabilityBackendApplicationTests
  ──────────────────────────────────────────
  Tests run: 42, Failures: 0, Errors: 0
  BUILD SUCCESS — 31.762 s
```

`LoteControllerTest` contiene 25 casos que verifican el comportamiento de los endpoints REST mediante `MockMvc`, incluyendo los `happy paths` de todos los endpoints IoT y directos, los casos de error 404 y 503, y la validación 400 con mapa de errores por campo.

`LoteServiceTest` contiene 16 casos que verifican la lógica de negocio del servicio de forma aislada mediante mocks de Mockito, cubriendo el comportamiento con blockchain habilitada, con blockchain deshabilitada, y ante fallos de blockchain.

Un aspecto notable de esta clase es el test `crearLote_blockchainFalla_guardaEventoConTxHashNullSinLanzarExcepcion`, que verifica que cuando el wrapper de blockchain lanza una excepción, el servicio la captura silenciosamente, persiste el evento en H2 con `txHash` nulo y no propaga el error al `caller`. Durante la ejecución de este test, aparece en el log un stack trace de `RuntimeException`: Ganache caído que es completamente esperado: es la excepción simulada por el mock que el servicio debe absorber sin relanzar.

`OliveTraceabilityBackendApplicationTests` contiene 1 caso de integración que verifica que el contexto de Spring Boot carga correctamente con todas sus dependencias.

Los tests se ejecutan sin necesidad de Ganache, Mosquitto ni Docker activos, ya que todas las dependencias externas están mockeadas. Esto garantiza que la suite puede ejecutarse en cualquier entorno de integración continua sin infraestructura adicional.

## 6.6 *VERIFICACIÓN DEL FLUJO COMPLETO END-TO-END*

La verificación end-to-end permite comprobar que todos los componentes del sistema funcionan de forma integrada, desde la interacción del usuario en el frontend hasta el registro de los eventos en blockchain y su posterior consulta.

Durante las pruebas se ejecutó el flujo completo de trazabilidad de un lote. El frontend permitió avanzar por las distintas fases del proceso, el backend coordinó las peticiones, el simulador IoT proporcionó las mediciones mediante MQTT y cada evento quedó registrado tanto en la base de datos H2 como en el smart contract desplegado en Ganache. Para cada operación registrada en blockchain se obtuvo un `txHash` distinto, asociado al evento correspondiente.

La consulta posterior desde la página de trazabilidad permitió visualizar el historial completo del lote en orden cronológico, incluyendo los datos específicos de cada evento y sus hashes de transacción. De este modo, se verifica que la arquitectura propuesta funciona como un sistema integrado de trazabilidad, no solo como un conjunto de módulos independientes.

# Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

## *7.1 CONCLUSIONES*

Este Trabajo de Fin de Grado tenía como objetivo principal diseñar e implementar una maqueta funcional de un sistema de trazabilidad para la cadena de suministro del aceite de oliva que integrara comunicaciones IoT y blockchain de forma cohesionada, verificable y demostrable. Los resultados obtenidos permiten afirmar que este objetivo se ha cumplido satisfactoriamente.

El sistema implementado demuestra de forma práctica que la combinación de IoT y blockchain resuelve los problemas estructurales identificados en el estado del arte. La centralización de los registros, que en los sistemas tradicionales introduce vulnerabilidades de integridad, se elimina mediante el anclaje criptográfico de cada evento en blockchain: el hash de transacción asociado a cada registro permite a cualquier actor de la cadena verificar de forma independiente la autenticidad del dato sin necesidad de confiar en un intermediario centralizado. La dependencia de registros manuales se reduce mediante la integración con once sensores IoT simulados mediante el protocolo MQTT, que cubren el proceso completo desde la báscula de campo hasta la centrífuga vertical de la almazara. La falta de transparencia para el comprador o auditor se aborda mediante la página de trazabilidad, que expone el historial completo de hasta 13 eventos certificados de cualquier lote con sus hashes de transacción verificables directamente en el nodo blockchain.

Respecto a los objetivos específicos definidos, todos han sido alcanzados. Se diseñó una arquitectura en cuatro capas con separación clara de responsabilidades y cuatro principios transversales de desacoplamiento, opcionalidad, verificabilidad y coherencia física. Se implementó el contrato inteligente `CadenaAceite` en Solidity 0.8.24 con 13 eventos y 14 funciones, incluyendo control de acceso mediante `OpenZeppelin Ownable v5`. Se desarrolló una API REST completa con Spring Boot 3.5.9 y 23 endpoints funcionales organizados en dos fases. Se integró la comunicación con sensores IoT mediante el protocolo MQTT con

broker Eclipse Mosquitto y simulador Python containerizado con Docker Compose, con rangos de datos físicamente coherentes entre sí. Se desarrolló el frontend React con cinco rutas funcionales y animaciones SVG específicas para cada proceso industrial de la almazara. Se validó el funcionamiento end-to-end del sistema mediante una suite de 42 tests automatizados con 0 fallos, verificando la coherencia entre los datos almacenados en H2, las respuestas de la API y los registros on-chain en Ganache.

Desde el punto de vista técnico, el proyecto ha demostrado la viabilidad de integrar tecnologías con paradigmas de comunicación heterogéneos en un sistema cohesionado. La combinación de un protocolo síncrono como HTTP REST con un protocolo asíncrono como MQTT en el mismo backend, es posible mediante el patrón `CompletableFuture`, ilustra la complejidad real que introduce la integración IoT en sistemas empresariales. La resolución de la incompatibilidad entre Web3j 4.12.2 y Spring Boot 3 mediante un wrapper manual con firma EIP-155 demuestra la capacidad de adaptar librerías de ecosistemas distintos sin renunciar al control sobre el protocolo subyacente.

Una reflexión relevante para la defensa del diseño adoptado es la justificación de blockchain frente a una base de datos tradicional con logs de auditoría. Un sistema de auditoría centralizado ofrece inmutabilidad práctica pero no técnica: el administrador de la base de datos tiene la capacidad técnica de modificar cualquier registro. Blockchain ofrece inmutabilidad técnica: modificar un registro requeriría recalcular todos los bloques posteriores con una potencia de cómputo superior a la del resto de la red. Los casos de uso industriales analizados en el estado del arte, desde IBM Food Trust hasta los trabajos específicos sobre el sector olivarero, confirman que esta arquitectura está siendo adoptada por actores reales del sector con resultados positivos en términos de transparencia y confianza del consumidor.

## *7.2 LIMITACIONES*

El sistema desarrollado es un prototipo académico cuyas limitaciones son inherentes a su naturaleza y están documentadas desde el inicio del proyecto en el apartado de alcance del capítulo 4.

La limitación más significativa desde el punto de vista de la trazabilidad es que Ganache es una red blockchain local que solo existe mientras está activa en la máquina de desarrollo. En producción, el sistema debería desplegarse sobre una red blockchain persistente, ya sea una testnet pública como Sepolia, una red privada permisionada o una red de consorcio entre cooperativas y almazaras. La arquitectura del sistema está preparada para este cambio: actualizar la URL del nodo y el `chainId` en `application.properties` es suficiente para apuntar a cualquier red EVM-compatible sin modificar ningún componente del sistema.

La segunda limitación es que todos los sensores IoT son simulados mediante el script Python del simulador. El sistema no se conecta a ningún hardware físico. Sin embargo, la arquitectura MQTT adoptada está diseñada específicamente para abstraer el origen del dato: sustituir el simulador por sensores industriales reales que publiquen en los mismos tópicos MQTT no requeriría ningún cambio en el backend ni en el frontend. La coherencia física de los rangos del simulador garantiza además que los datos generados son representativos de un proceso real.

La tercera limitación es la ausencia de autenticación y autorización en la API REST. Cualquier actor con acceso a la red puede registrar eventos en nombre de cualquier lote. En producción, el control de acceso a nivel de contrato debería extenderse del patrón `onlyOwner` actual a un sistema de roles por actor mediante `AccessControl` de OpenZeppelin, de forma que solo la almazara correspondiente pueda registrar eventos de recepción, o solo el transportista pueda registrar el cierre de compuerta.

La cuarta limitación es la base de datos H2 en memoria, que pierde todos los datos al reiniciar el backend. El proyecto incluye el driver de PostgreSQL en el fichero de dependencias, por

lo que la migración a una base de datos persistente requiere únicamente actualizar la configuración de `datasource` sin modificar ningún código Java.

## *7.3 TRABAJO FUTURO*

Las líneas de trabajo futuro identificadas se ordenan de menor a mayor alcance, desde extensiones inmediatas del prototipo hasta evoluciones que cambiarían fundamentalmente la naturaleza del sistema.

### 7.3.1 INTEGRACIÓN CON HARDWARE IOT REAL

La extensión más directa e impactante del sistema es sustituir el simulador Python por hardware físico. Un sensor de peso basado en una celda de carga con amplificador HX711 conectado a un microcontrolador ESP32 con soporte MQTT nativo sería suficiente para convertir el sistema en una solución IoT real sin modificar ningún componente del backend. Para los sensores de temperatura y pH, los modelos DS18B20 y sensores analógicos de pH con convertidor ADC son opciones ampliamente disponibles y compatibles con el mismo patrón de publicación MQTT. Esta extensión daría al proyecto una dimensión física que reforzaría significativamente su credibilidad ante actores reales del sector.

### 7.3.2 DESPLIEGUE EN RED BLOCKCHAIN PÚBLICA O SEMIPÚBLICA

El despliegue en una testnet pública de Ethereum como Sepolia permitiría que los hashes de transacción fueran verificables desde cualquier navegador mediante exploradores de bloques como Etherscan, sin necesidad de acceso al nodo local. Para un entorno de producción real, una red de consorcio basada en Hyperledger Besu en modo IBFT, compartida entre cooperativas, almazaras y organismos certificadores, ofrecería el equilibrio adecuado entre descentralización, rendimiento y control de acceso. En este escenario, la dirección del contrato y el `chainId` de la red de consorcio serían los únicos parámetros a actualizar en la configuración del backend.

### 7.3.3 CONTROL DE ACCESO POR ACTORES MEDIANTE ROLES

La extensión del modelo de control de acceso actual, basado en `onlyOwner`, a un sistema de roles por actor mediante `AccessControl` de OpenZeppelin permitiría que cada actor de la cadena firmara sus propias transacciones con su wallet privada. El evento PESAJE_CAMION_LLENO quedaría firmado por la cuenta de la almazara receptora, el evento CAMION_CERRADO por la cuenta de la cooperativa de origen, y el evento EXTRACCION_FINALIZADA por la cuenta del responsable de producción. Esta extensión convertiría el sistema de trazabilidad en un registro de responsabilidades distribuidas, donde cada evento certifica no solo qué ocurrió sino también quién lo certificó criptográficamente.

### 7.3.4 NOTIFICACIONES AUTOMÁTICAS A LOS ACTORES DE LA CADENA

El sistema actual no notifica a los actores cuando ocurre un evento en su lote. Una extensión natural sería añadir un servicio de notificaciones que enviara alertas por correo electrónico o SMS al agricultor cuando el lote llega a la almazara, cuando se completa el proceso de extracción, o cuando se detecta una anomalía como una apertura de compuerta no autorizada. Esta funcionalidad no requeriría cambios en el contrato ni en el backend principal: bastaría con añadir un subscriber MQTT o un listener de eventos JPA que disparara las notificaciones.

### 7.3.5 PAGOS AUTOMÁTICOS AL AGRICULTOR MEDIANTE SMART CONTRACTS

La integración más transformadora para el sector sería la automatización de los pagos a los agricultores mediante smart contracts. Cuando el evento PESAJE_CAMION_LLENO y el evento PESAJE_CAMION_VACIO quedan confirmados en blockchain, el contrato podría calcular automáticamente el peso neto de aceituna entregado y ejecutar la transferencia de tokens al agricultor proporcional a ese peso y al precio por kilogramo pactado `on-chain` en el momento de la creación del lote. Este mecanismo eliminaría completamente la dependencia de intermediarios en el proceso de liquidación, convirtiendo el sistema de trazabilidad en una plataforma de comercio descentralizado para el sector olivarero.

### 7.3.6 EXTENSIÓN A LAS FASES DE ENVASADO Y DISTRIBUCIÓN

El sistema actual cubre el proceso desde el campo hasta la extracción del aceite en la almazara. Una extensión natural sería añadir los eventos correspondientes a las fases posteriores de la cadena: el envasado con su lote de producción y fecha de caducidad, el etiquetado con los datos de denominación de origen, y la salida hacia el distribuidor. Con estos eventos adicionales, el consumidor final podría escanear el código QR de una botella de aceite y ver en blockchain el historial completo desde la aceituna en el árbol hasta el aceite envasado listo para su distribución, incluyendo la temperatura de batido que certifica la calidad virgen extra. Este es el escenario de valor final que justifica la inversión en infraestructura blockchain para el sector agroalimentario.

# Capítulo 8. BIBLIOGRAFÍA

[1] Consejo Oleícola Internacional (COI), "World Olive Oil Figures," 2023. [En línea]. Disponible en: https://www.internationaloliveoil.org/

[2] Parlamento Europeo y del Consejo, "Reglamento (CE) n.º 178/2002 por el que se establecen los principios y requisitos generales de la legislación alimentaria," *Diario Oficial de la Unión Europea*, 28 enero 2002.

[3] IBM, "IBM Food Trust: Blockchain for food traceability," 2023. [En línea]. Disponible en: https://www.ibm.com/blockchain/solutions/food-trust

[4] BeefChain, "Blockchain beef traceability," 2023. [En línea]. Disponible en: https://beefchain.com

[5] Comisión Europea, "Reglamento de Ejecución (UE) 2022/2104 de la Comisión, de 29 de julio de 2022, por el que se establecen los métodos de análisis para la determinación de los parámetros de autenticidad del aceite de oliva," *Diario Oficial de la Unión Europea*, 29 julio 2022.

[6] Comisión Europea, "Estrategia 'De la granja a la mesa' para un sistema alimentario justo, saludable y respetuoso con el medio ambiente," COM(2020) 381 final, 2020.

[7] T. Bosona y G. Gebresenbet, "Food traceability as an integral part of logistics management in food and agricultural supply chain," *Food Control*, vol. 33, n.º 1, pp. 32–48, sep. 2013, doi: 10.1016/j.foodcont.2013.02.004.

[8] R. M. Ellahi, L. C. Wood, y A. E.-D. A. Bekhit, "Blockchain-based frameworks for food traceability: A systematic review," *Foods*, vol. 12, n.º 16, art. 3026, ago. 2023, doi: 10.3390/foods12163026.

[9] R. Kamath, "Food traceability on blockchain: Walmart's pork and mango pilots with IBM," *The Journal of The British Blockchain Association*, vol. 1, n.º 1, jun. 2018, doi: 10.31585/jbba-1-1-(10)2018.

[10] T. Frikha, J. Ktari, y H. Hamam, "Blockchain olive oil supply chain," en *Risks and Security of Internet and Systems. CRiSIS 2022*, S. Kallel et al., Eds., *Lecture Notes in Computer Science*, vol. 13857. Cham, Suiza: Springer, 2023, pp. 107–122, doi: 10.1007/978-3-031-31108-6_8.

[11] A. Habashneh, A. Assayed, y A. AlMajali, "Using blockchain for agro-food traceability: A case study from olive oil industry," en *Industry 4.0 Technologies:*

*Sustainable Manufacturing Supply Chains*, K. E. K. Vimal et al., Eds., *Environmental Footprints and Eco-design of Products and Processes*. Singapore: Springer, 2024, pp. 37–52, doi: 10.1007/978-981-99-4819-2_3.

[12] R. Alimadani, A. A. Adedeji, y M. Narimani, "A review of IoT applications in food processing and related fields," *Journal of Food Processing and Preservation*, vol. 2025, art. 3064441, 2025, doi: 10.1155/jfpp/3064441.

[13] V. Vitaskos, K. Demestichas, S. Karetsos, y C. Costopoulou, "Blockchain and Internet of Things technologies for food traceability in olive oil supply chains," *Sensors*, vol. 24, n.º 24, art. 8189, dic. 2024, doi: 10.3390/s24248189.

[14] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [En línea]. Disponible en: https://bitcoin.org/bitcoin.pdf

[15] Z. Zheng, S. Xie, H. Dai, X. Chen, y H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," en *Proc. 2017 IEEE Int. Congr. Big Data (BigData Congress)*, Honolulú, HI, EE. UU., jun. 2017, pp. 557–564, doi: 10.1109/BigDataCongress.2017.85.

[16] V. Buterin, "A next-generation smart contract and decentralized application platform," Ethereum Foundation, 2013. [En línea]. Disponible en: https://ethereum.org/en/whitepaper/

[17] K. Christidis y M. Devetsikiotis, "Blockchains and smart contracts for the Internet of Things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016, doi: 10.1109/ACCESS.2016.2566339.

[18] Linux Foundation Decentralized Trust, "How Walmart brought unprecedented transparency to the food supply chain with Hyperledger Fabric," s.f. [En línea]. Disponible en: https://www.lfdecentralizedtrust.org/case-studies/walmart-case-study

[19] A. Kamilaris, A. Fonts, y F. X. Prenafeta-Boldú, "The rise of blockchain technology in agriculture and food supply chains," *Trends in Food Science & Technology*, vol. 91, pp. 640–652, sep. 2019, doi: 10.1016/j.tifs.2019.07.034.

# ANEXO I: ALINEACIÓN DEL PROYECTO CON LOS ODS

Este proyecto se alinea principalmente con los Objetivos de Desarrollo Sostenible 9, 12 y 16 de la Agenda 2030, al proponer una solución tecnológica orientada a mejorar la trazabilidad, transparencia y fiabilidad de la cadena de suministro agroalimentaria.

En relación con el **ODS 9: Industria, innovación e infraestructura**, el proyecto aplica tecnologías digitales como IoT, blockchain y smart contracts a un sector tradicional como el agroalimentario. La integración de dispositivos IoT para la captura automática de datos y de una red blockchain para su registro verificable contribuye al desarrollo de infraestructuras digitales más robustas, interoperables y transparentes. De este modo, se favorece la modernización de los procesos industriales y logísticos asociados a la producción de aceite de oliva.

Respecto al **ODS 12: Producción y consumo responsables**, el sistema propuesto permite mejorar el seguimiento de los lotes a lo largo de la cadena de suministro, desde su origen hasta las fases de procesamiento. Una trazabilidad más completa facilita la identificación de incidencias, la verificación del origen de la materia prima y la consulta de información relevante sobre el proceso productivo. Esto puede contribuir a una gestión más eficiente de los recursos, a la reducción de errores en la cadena y a una mayor transparencia hacia consumidores y entidades participantes.

Finalmente, el proyecto también guarda relación con el **ODS 16: Paz, justicia e instituciones sólidas**, al incorporar un mecanismo de registro que dificulta la manipulación posterior de la información. El uso de blockchain permite que los eventos relevantes queden asociados a transacciones verificables, reforzando la confianza entre los distintos actores de la cadena. Esta característica favorece la rendición de cuentas y la transparencia en procesos donde la integridad de los datos resulta fundamental.

# ANEXO II: GLOSARIO DE TÉRMINOS

- **ABI (Application Binary Interface):** Interfaz que define cómo interactuar con un smart contract desplegado en la blockchain. Especifica las funciones disponibles, sus parámetros y los tipos de datos que devuelve, permitiendo que aplicaciones externas invoquen el contrato correctamente.
- **API REST:** Estilo de arquitectura para servicios web que utiliza el protocolo HTTP y sus métodos estándar (GET, POST, PUT, DELETE) para exponer recursos e intercambiar datos, habitualmente en formato JSON.
- **Blockchain:** Estructura de datos distribuida formada por bloques de información encadenados criptográficamente. Cada bloque contiene un conjunto de transacciones y el hash del bloque anterior, lo que garantiza la inmutabilidad e integridad del historial registrado sin necesidad de una autoridad central.
- **Broker MQTT:** Servidor intermediario en una arquitectura MQTT que recibe los mensajes publicados por los clientes y los distribuye a los suscriptores correspondientes según el topic al que estén suscritos.
- **Cadena de suministro:** Conjunto de actores, procesos y flujos de información que intervienen desde la producción de una materia prima hasta la entrega del producto final al consumidor.
- **Smart contract (Contrato inteligente):** Programa autoejecutable desplegado en una blockchain que codifica reglas de negocio y se ejecuta automáticamente cuando se cumplen las condiciones predefinidas, sin necesidad de intermediarios.
- **Degradación elegante (Graceful degradation):** Capacidad de un sistema para seguir funcionando con funcionalidad reducida ante el fallo de uno de sus componentes, en lugar de detenerse completamente. En este proyecto, si la red blockchain no está disponible, el sistema continúa operando utilizando únicamente la base de datos H2.
- **Dual-write:** Patrón arquitectónico en el que cada operación de escritura se realiza simultáneamente en dos sistemas de almacenamiento, en este caso la base de datos H2 (caché rápida) y la blockchain Ganache (fuente de verdad), vinculados mediante el hash de transacción.
- **Ethereum:** Plataforma blockchain de código abierto que extiende el concepto de Bitcoin incorporando soporte nativo para contratos inteligentes y aplicaciones descentralizadas (dApps), mediante su máquina virtual (EVM).
- **Evento (blockchain):** Mecanismo de notificación emitido por un smart contract cuando se produce una acción relevante. Los eventos quedan registrados en el log de la transacción y pueden ser consultados por aplicaciones externas para conocer el estado de la cadena.
- **Fraude alimentario:** Práctica ilícita consistente en adulterar, falsificar o etiquetar incorrectamente un producto alimentario con fines económicos. En el sector del aceite de oliva, es especialmente frecuente la mezcla de aceite virgen extra con aceites de menor calidad.
- **Ganache:** Blockchain Ethereum local de uso privado orientada al desarrollo y las pruebas. Simula el comportamiento de la red Ethereum principal sin coste de gas real, lo que permite desplegar y probar contratos inteligentes en un entorno controlado.
- **Gas:** Unidad que mide el coste computacional de ejecutar operaciones en la red Ethereum. Cada transacción o llamada a un smart contract consume una cantidad de gas proporcional a su complejidad.

- **Hash / Función hash:** Función criptográfica que transforma cualquier entrada de datos en una cadena de longitud fija y única. En blockchain, cada bloque contiene el hash del bloque anterior, formando la cadena e impidiendo la alteración retroactiva de los registros.
- **Inmutabilidad:** Propiedad de la blockchain por la que los datos una vez escritos no pueden ser modificados ni eliminados. Cualquier alteración invalidaría el hash del bloque afectado y todos los posteriores.
- **IoT (Internet of Things):** Paradigma tecnológico en el que objetos físicos equipados con sensores, actuadores y conectividad de red son capaces de recopilar y transmitir datos de forma autónoma, integrándose en sistemas de información digitales.
- **JSON-RPC:** Protocolo de llamada a procedimiento remoto (RPC) que utiliza JSON como formato de codificación de mensajes. En este proyecto se emplea para la comunicación directa entre el backend Spring Boot y el nodo Ganache, sin recurrir a las clases base de Web3j.
- **Lote:** Unidad de trazabilidad que agrupa un conjunto de aceitunas u aceite de oliva producidos bajo las mismas condiciones y en el mismo periodo. Cada lote recibe un identificador único (en este proyecto, con el formato LOT-{año}-{secuencia}) que le acompaña a lo largo de toda la cadena.
- **MQTT (Message Queuing Telemetry Transport):** Protocolo de mensajería ligero basado en el modelo publicación-suscripción, diseñado para entornos con recursos limitados o conexiones de baja fiabilidad. Es ampliamente utilizado en aplicaciones IoT para la transmisión de datos de sensores.
- **Sensor:** Dispositivo electrónico que detecta y mide una magnitud física del entorno (temperatura, peso, humedad, etc.) y la convierte en una señal digital que puede ser procesada o transmitida.
- **Sistema distribuido:** Conjunto de nodos de computación independientes que se comunican a través de una red y cooperan para alcanzar un objetivo común, presentándose al usuario como un sistema coherente y unificado. La blockchain es un ejemplo de sistema distribuido donde no existe una autoridad central de control.
- **Solidity:** Lenguaje de programación orientado a objetos, con tipado estático, diseñado específicamente para escribir contratos inteligentes que se ejecutan sobre la Ethereum Virtual Machine (EVM).
- **Topic (MQTT):** Canal lógico sobre el que se publican y reciben mensajes en una arquitectura MQTT. Los clientes suscritos a un topic reciben automáticamente todos los mensajes publicados en él.
- **Trazabilidad:** Capacidad de seguir el rastro de un producto a lo largo de todas las etapas de su cadena de suministro, desde el origen hasta el consumidor final, registrando los datos relevantes de cada fase de forma verificable.
- **txHash (Transaction Hash):** Identificador único generado criptográficamente para cada transacción registrada en la blockchain. Actúa como enlace entre el registro en la base de datos local (H2) y el registro inmutable en la blockchain, permitiendo verificar la autenticidad de cualquier evento.
- **Web3j:** Biblioteca Java de código abierto que permite la integración de aplicaciones Java con nodos Ethereum, facilitando el despliegue de contratos inteligentes y el envío y consulta de transacciones mediante llamadas JSON-RPC.