TRABAJOS FIN DE GRADO

Universidad de Jaén

Escuela Politécnica Superior de Jaén

Desarrollo de un Sistema de Trazabilidad en Tiempo Real usando Blockchain y Dispositivos IoT

Autor: Juan Antonio Torres De La Chica

Grado: Ingeniería en Informática

Directores: Bruno Ramos Cruz y Luis Martínez López
Departamento del director: Informática

Fecha: 20/06/2025

Dedicado a mi familia, mis tutores y mis amigos.

RESUMEN

Esta memoria recoge el desarrollo de un sistema de trazabilidad basado en tecnología *Blockchain* (en este caso, Hyperledger Besu) con el uso de distintos dispositivos *IoT* (*Internet of Things*). En dicho sistema se monitorizará mediante una simulación el trazado que siguen los alimentos desde su origen hasta llegar al consumidor aprovechando la versatilidad de los dispositivos *IoT* para recopilar multitud de datos como la posición geográfica, la temperatura a la que se ve sometida el producto, la luminosidad, etc. Es importante destacar que algunas de las características de la tecnología *Blockchain* aportan un importante valor añadido para una trazabilidad segura y fiable tal y como puede ser la INMUTABILIDAD.

Tabla de contenidos

Lista de figuras

Lista de tablas

Lista de listados de código

Capítulo 1

INTRODUCCIÓN

Desde el inicio de este proyecto, mi objetivo personal fue aprender e investigar un campo de la Informática que me llamaba mucho la atención, como eran las tecnologías de Libro Mayor distribuido *(Distributed Ledger Tecnologies, DLT)* también conocida habitualmente como *Blockchain*. A lo largo del tiempo en el que he estado desarrollando este trabajo, me he dado cuenta de que la cantidad de aplicaciones de este conjunto de tecnologías es increíblemente enorme. Mi idea preconcebida era que simplemente tenían cabida en el ámbito de las criptomonedas, *tokens* y NFT (*non-fungible token*). Sin embargo, estos son algunos de las únicas aplicaciones que le suenan a todo estudiante de informática ya que me di cuenta que es una tecnología disruptiva con gran potencial en muchos campos de aplicación aunque a nivel popular fue totalmente engullida por estas temáticas. Mi intención no es utilizar las ventajas de *Blockchain* para dichas temáticas, sino para otros temas que a mi parecer son igual o más importantes como, por ejemplo, la salud, la transparencia y el bienestar de las personas.

1.1. Motivación

Allá en el año 1981 ocurrió una de las mayores tragedias en la historia reciente de España en el ámbito de la salud, mucho antes de la llegada del coronavirus en 2020:

La intoxicación por aceite de colza.

Miles de personas fueron afectadas e intoxicadas por el consumo de aceite desnaturalizado, destinado para uso industrial, que fue vendido como apto para el consumo humano [9]. Este desastre ocurrió debido a la falta de medidas y regulaciones de seguridad alimentaria en el país. Desde entonces, las cosas han cambiado conside-

rablemente. Hoy en día, los productos deben garantizar su procedencia para poder ser vendidos en la mayoría, si no en todos, los establecimientos del país. Sin embargo, estos procesos pueden ser tediosos, llevar mucho tiempo y estar plagados de burocracia.

En los últimos años, distintos sectores dentro de la cadena de suministro han enfrentado una creciente presión para adaptarse a las nuevas exigencias del mercado y de los consumidores. La autenticidad, seguridad y trazabilidad de los productos se han convertido en factores clave, impulsados por la necesidad de garantizar su procedencia y el recorrido que han seguido hasta su destino final. En este contexto, la tecnología *Blockchain* se presenta como una solución innovadora, que mejora la gestión y la transparencia en la cadena de suministro. Tal y como se señala en el libro "*Supply Chain Finance and Blockchain Technology: The Case of Reverse Securitisation*":

> *"La tecnología Blockchain promete cambiar la forma en que las personas y las empresas intercambian valor e información a través de Internet, por lo que parece estar perfectamente posicionada para permitir nuevos niveles de colaboración entre los agentes de la cadena de suministro"[10].*

1.2. Propósito

Explorar la tecnología *Blockchain* como una solución innovadora a las necesidades del mercado es el propósito de mi trabajo de fin de grado, aprovechando sus ventajas con respecto a soluciones tradicionales (e.g. etiquetado y documentación manual, sistemas informáticos centralizados, etc [11]) como pueden ser su modelo descentralizado y seguro para almacenar y verificar los datos, mejorando la fiabilidad y transparencia en los procesos de trazabilidad.

Al implementar *Blockchain* en la trazabilidad se reducirá el riesgo al fraude, la falsificación y los posibles errores humanos. Ganando una mayor confianza en todos los actores de la cadena de suministro, los cuales pueden acceder a información verificada de los productos.

También se busca analizar la integración con dispositivos *IoT* que nos permitiría recolectar los datos necesarios para la trazabilidad alimentaria de forma automática; incrementando la eficiencia en la monitorización de condiciones como temperatura, ubicación de los productos, etc. De esta manera, se plantea una solución que no solo

refuerza la seguridad de los datos, sino que también optimiza la gestión de la cadena de suministro mediante la automatización y digitalización de procesos.

1.3. Objetivos

Se proponen varios objetivos para el éxito del desarrollo de este TFG(Trabajo de Fin de Grado), tanto un propósito general como objetivos específicos.

1.3.1. Propósito general

El propósito general de este TFG consiste en desarrollar e implementar un sistema de trazabilidad que utilice la tecnología *Blockchain* y dispositivos *IoT*, asegurando la transparencia y fiabilidad de los datos en la cadena de suministro. Este sistema permitirá el seguimiento en tiempo real de los productos, garantizando la inmutabilidad de la información, mejorando la transparencia y facilitando una mejor toma de decisiones por parte de los participantes en la cadena de suministro.

1.3.2. Objetivos específicos

Los objetivos específicos para alcanzar el propósito general son:

1. Revisión de la literatura sobre trazabilidad en la cadena de suministro, *Blockchain* e *IoT*, con el fin de identificar el estado del arte y las soluciones existentes.
2. Delimitación del caso de uso a partir de la literatura revisada, seleccionando un escenario donde aplicar la tecnología *Blockchain* para mejorar la trazabilidad.
3. Diseño del modelo *Blockchain*, basado en *Ethereum* y *Solidity*, adaptado a las necesidades del caso de uso definido.
4. Diseño de la integración del modelo *Blockchain* con los dispositivos *IoT*, estableciendo la comunicación entre sensores y la red descentralizada.
5. Implementación del sistema propuesto utilizando tecnologías como *Ethereum* y *Solidity*, evaluando su desempeño en un entorno real y controlado para validar su viabilidad y eficiencia.

1.4. Metodología

Para conseguir los objetivos anteriores, se seguirá la siguiente metodología de trabajo, junto con la planificación mostrada en la figura 1.1:

1. Revisión de la literatura (11 de octubre – 5 de mayo): Se buscarán y analizarán fuentes relevantes sobre las tres importantes temáticas: Trazabilidad, *Blockchain* e *IoT*. Desde el 11 al 20 de octubre, se recopilará el material de estudio. Desde el 21 de octubre hasta la entrega del trabajo se leerán y estudiarán estos materiales. Después de aproximadamente un mes para un estudio inicial, del 1 al 15 de diciembre, se identifica cómo se encuentran actualmente estas áreas y cuáles serán los mayores desafíos a la hora de implementar.

2. Caso de uso (16 de diciembre – 15 de enero): Lo referente al caso de uso de la trazabilidad. Del 16 al 25 de diciembre, se clasificarán y analizarán estudios que hayan abordado este caso de uso aunque fuera con otras tecnologías. Del 26 de diciembre al 5 de enero, se generarán ideas para abordar nuestro caso de uso de trazabilidad con *Blockchain* e *IoT*. Del 6 al 15 de enero, delimitación del alcance del sistema a partir de las necesidades y problemas encontrados.

3. Diseño de la *Blockchain* (16 de enero – 31 de marzo): A partir del caso de uso definido, se elegirá un modelo específico de *Blockchain*. Del 16 al 31 de enero, elección de la estructura de bloques y transacciones. Del 1 al 15 de febrero, se diseñará la arquitectura de la red. Un tiempo después, del 20 al 31 de marzo, se diseñarán y escribirán los *smart contracts* necesarios para el sistema.

4. Integración (16 de febrero – 19 de marzo): Se integrarán los dispositivos *IoT* con *Blockchain*. Del 16 al 26 de febrero, se definirá como hacer la integración. Del 27 al 6 de marzo, se conectarán ambas tecnologías. Del 7 al 19 de marzo, se harán las pruebas de integración iniciales y los ajustes necesarios para mejorar el redimiento.

5. Implementación y validación (16 de febrero – 5 de mayo): El sistema diseñado será implementado utilizando plataformas *Blockchain* como Hyperledger Besu, para probar su rendimiento y eficiencia en un entorno real y controlado. Entre el 16 de febrero y el 15 de abril, se implementará la infraestructura de Hyperledger Besu. Del 16 al 30 de abril, se realizarán pruebas de rendimiento para evaluar la eficiencia del sistema de trazabilidad. Finalmente, del 1 al 5 de mayo, se realizarán los ajustes finales y se documentarán los resultados obtenidos para la elaboración del informe final del proyecto.

1.4.1. Planificación temporal

A continuación, se presenta un diagrama de actividades 1.1 que describe la duración y secuencia de cada actividad.

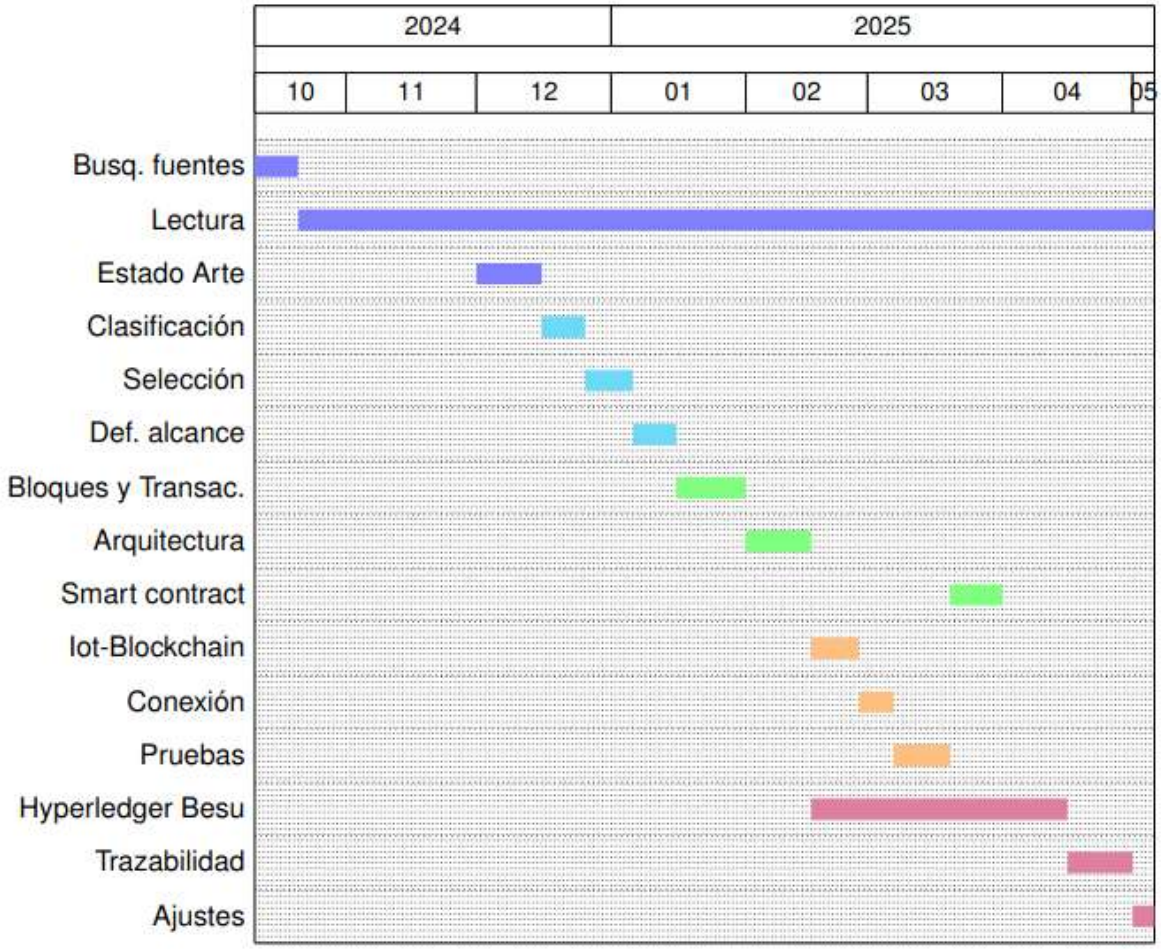

Figura 1.1: Diagrama de actividades.

1.5. Planificación de coste

En el mundo real todo tiene un coste obviamente, aquí la planificación de coste del proyecto teniendo en cuenta tanto los gastos de personal, *software* como de *hardware*.

1.5.1. Costes del *Software*

Se expondrán los costes mensuales, trimestrales y anuales con la intención de tener una vista proporcional de los gastos que tendría el desarrollo del proyecto. Esto se puede apreciar en la tabla 1.1, donde se puede observar que la gran ventaja de utilizar software de código abierto.

Software	1 mes	3 meses	1 año
Hyperledger Besu	0 €	0 €	0 €
Arduino IDE	0 €	0 €	0 €
Hardhat	0 €	0 €	0 €
Vs code	0 €	0 €	0 €
OpenVPN	0 €	0 €	0 €
VirtualBox	0 €	0 €	0 €
Total	**0 €**	**0 €**	**0 €**

Tabla. 1.1: Costos de *sotfware*

El papel de Hyperledger Besu es el más importante en este proyecto. Engloba toda la tecnología de *Blockchain*, comunicación *P2P*(*peer-to-peer*), gestión y creación de bloques, de transacciones, etc; es el cliente *Blockchain*. Arduino IDE para programar los arduinos. Remix para la compilación y testeo de *smart contract*. Hardhat para el despliegue y uso de *smart contract* en la red. VS(Visual Studio) Code para desarrollar el código del sistema. OpenVPN para simular los equipos en una misma red. VirtualBox para el uso de máquinas virtuales.

1.5.2. Costes del Hardware

El coste del *hardware* utilizado no es recurrente en cuanto a su adquisición, pero sí se debe considerar su amortización para una estimación más realista de los costes mensuales del proyecto. En este caso, se ha considerado un periodo de amortización de equivalente a la duración del proyecto (8 meses) para los ordenadores portátiles y el ordenador de sobremesa. Así, se calcula un coste mensual aproximado que refleje mejor el uso de estos recursos durante el desarrollo.

La tabla 1.2 muestra los costes iniciales del hardware y, en el caso de los equipos informáticos, su correspondiente coste mensual tras aplicar la amortización.

Hardware	Costo único	Costo mensual
Dispositivos *IoT*		
Arduino MKR Zero	32 €	-
Arduino MKR GPS Shield	35 €	-
Arduino Uno	28 €	-
Grove - GPS (air530)	11 €	-
NodeMCU v3	4,09 €	-
DHT11	6,49 €	-
Sensor PIR	7,58 €	-
Portátil 1	600 €	75 €
Portátil 2	600 €	75 €
Ordenador de sobremesa	1200 €	150 €
Total	**2524,16 €**	**315,52 €**

Tabla. 1.2: Costos de *hardware* y amortización mensual

El ordenador de sobremesa es el equipo más importante del sistema: contiene cinco máquinas virtuales (VMs), cada una con un nodo, además de otro nodo en el sistema operativo anfitrión que funciona como nodo ADMIN y *Bootnode*. Los portátiles actúan como nodos 'móviles', simulando el desplazamiento de productos. Finalmente, los dispositivos *IoT* son responsables de capturar los datos.

1.5.3. Coste de personal

El único personal en este trabajo es el estudiante. Teniendo en cuenta el salario medio de un ingeniero informático en España [12]. El salario del tiempo contratado se divide en 8 pagas (8 meses). El resumen puede apreciarse en la figura 1.3.

Personal	1 mes	3 meses	8 meses(Total)
Estudiante	2625 €	7875 €	31500 €
Total	**2625 €**	**7875 €**	**21000 €**

Tabla. 1.3: Costos de personal.

1.6. Estructura del TFG

La presente memoria se organiza en varios capítulos que cubren desde las tecnologías utilizadas hasta las conclusiones.

En primer lugar, en el **capítulo 2** se realiza un análisis detallado de las bases teóricas que serán necesarias para poder llevar a cabo el proyecto, entre las que se encuentran tecnologías DLT e IoT.

En el **capítulo 3** se revisa y analiza el problema de la trazabilidad alimentaria el cual fue elegido como caso de uso, analizando sus desafíos y las soluciones tradicionales empleadas en el sector.

En el **capítulo 4** se presentan las tecnologías utilizadas.

En el **capítulo 5** con un enfoque de ingeniería del software, se definen los requisitos del sistema y se presenta su diseño a través de diagramas y el modelado de clases. Se explica la estructura de la arquitectura, así como el desarrollo de la interfaz de usuario y la integración de los diferentes componentes del sistema. Esto es seguido por una sección dedicada a su implementación y evaluación, tras lo que presentaremos los resultados y los discutiremos.

Finalmente, en el **capítulo 6** se presentan las conclusiones del trabajo, reflexionando sobre los resultados obtenidos, las limitaciones encontradas y las posibles mejoras futuras.

La memoria se complementa con una sección de referencias bibliográficas y anexos que incluyen documentación adicional, como manuales de instalación y uso, una lista de acrónimos y un glosario de términos técnicos.

Capítulo 2

BLOCKCHAIN e *IOT*

Con el fin de proporcionar una comprensión suficientemente profunda y fundamentada de las soluciones propuestas en este TFG, es esencial revisar los conceptos y las teorías que lo sustentan. Este capítulo se dedica a establecer las bases teóricas necesarias para el desarrollo del **sistema de trazabilidad con *Blockchain* e *IoT***. A través de una revisión de la literatura más importante, se exploran los principios clave de estas tecnologías. De esta forma, podremos entender las decisiones tomadas a lo largo del trabajo.

2.1. Tecnologías *DLT* y *Blockchain*

Es importante empezar por la tecnología que sentó las bases para la concepción de la primera *Blockchain* por Satoshi Nakamoto en su *White Paper*, "Bitcoin: *A Peer-to-Peer Electronic Cash System*"[13]: Esta es ***DLT***.

DLT se refiere a una infraestructura y conjunto de protocolos que permiten mantener un registro de datos de forma descentralizada y compartida. En un sistema *DLT*, la información se registra y actualiza simultáneamente en múltiples nodos de una red *peer-to-peer*, sin depender de una única entidad central [14].

A diferencia de las bases de datos tradicionales, este no tiene un almacenamiento de datos ni control de gestión centralizados, lo que elimina la figura de un intermediario único en las transacciones. Esto supone que todos los participantes de la red mantienen copias sincronizadas del registro y cualquier cambio legítimo se replica en todas ellas [14]. Por tanto, se logra eliminar el tener un único punto de fallo, generando una red más segura que se reflejaría en mayor confianza en sus datos; un atacante no

podría alterar la información de la red, a no ser que penetrara con éxito en, al menos, el 51 % de los nodos que conforman la red [15].

Figura 2.1: DLT no es lo mismo que Blockchain.

DLT y *Blockchain* suelen usarse como términos relacionados y a veces se confunden, pero no son exactamente lo mismo [14]. En realidad, ***Blockchain*** es un tipo particular de *DLT* con un diseño específico. Todas las *Blockchains* son *DLT*, pero no todas las tecnologías *DLT* implementan una estructura como la cadena de bloques (*Blockchain*) dicha estructura será definida posteriormente en la subsubsección 2.1.3.6. Por ejemplo, existen *DLT* basadas en otras estructuras de datos (como grafos acíclicos dirigidos en el caso de *IOTA Tangle* [16] o el algoritmo *Hashgraph* [17]) [14] que no emplean bloques encadenados, demostrando que *Blockchain* es solo una de varias implementaciones posibles de libros distribuidos.

Las similitudes fundamentales entre estas son la descentralización, la existencia de múltiples nodos replicando el registro y la necesidad de consenso para validar cambios; se verá en mayor detalle en la subsubsección 2.1.2.1. La diferencia clave es que *Blockchain* organiza los datos en grupos de transacciones llamados bloques, que se enlazan secuencialmente mediante funciones (*hash*) formando la cadena inmutable.

2.1.1. ¿Qué es *Blockchain*?

Blockchain son muchas cosas; tiene multitud de definiciones, pero podemos quedarnos por ahora con una. Es una tecnología descentralizada que permite la verificación y registro seguro de transacciones sin necesidad de intermediarios de confianza a través del uso de una estructura de datos que soluciona dos de los mayores problemas de una red descentralizada como pueden ser el doble gasto y la falla bizantina (problema de los generales bizantinos) [18, 19].

La *Blockchain*, como su nombre indica, está formada por bloques los cuales cada uno almacena un conjunto de transacciones validadas y está vinculado al bloque anterior mediante un *hash* criptográfico como se ve en la figura 2.2. Esta estructura encadenada garantiza la integridad de los datos, ya que cualquier intento de modificación en un bloque afectaría a toda la cadena. Debido a la propiedad anterior, solo se pueden agregar nuevas transacciones y no modificar o eliminar las ya registradas [19].

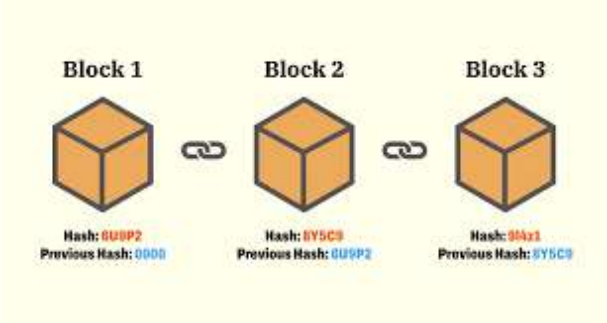

Figura 2.2: Blockchain

Blockchain no pretende reemplazar los sistemas existentes, sino complementarlos proporcionando una infraestructura de confianza [20]. Su diseño descentralizado permite la coexistencia con otras tecnologías de Internet y ha sido adoptado para múltiples aplicaciones, desde criptomonedas hasta soluciones empresariales.

Gracias a su código abierto *Blockchain* ha impulsado el desarrollo de numerosas aplicaciones descentralizadas, promoviendo un ecosistema digital sin control centralizado y con mayor transparencia [19].

2.1.1.1. Diferentes definiciones para el término *Blockchain*

Como se dijo *Blockchain* tiene diversas definiciones que pueden resumirse en tres principales, los cuales es importante saber diferenciar claramente:

1. En primer lugar, desde una perspectiva funcional, *Blockchain* es un mecanismo descentralizado de consenso, donde todos los participantes de la red deben ponerse de acuerdo acerca del estado válido de cada transacción, en contraste con las bases de datos tradicionales, centralizadas y privadas [18]. Bajo este enfoque, las transacciones se organizan en bloques ordenados cronológicamente, creando una única fuente fiable de información compartida entre todos los participantes [18]. Asimismo, *Blockchain* también se entiende como un **algoritmo**, compuesto por un conjunto de instrucciones que gestionan los datos distribuidos entre múltiples participantes. Este algoritmo permite llegar a acuerdos sobre las transacciones mediante un modelo de consenso parecido a una votación democrática [3] como se ilustra en la figura 2.3.

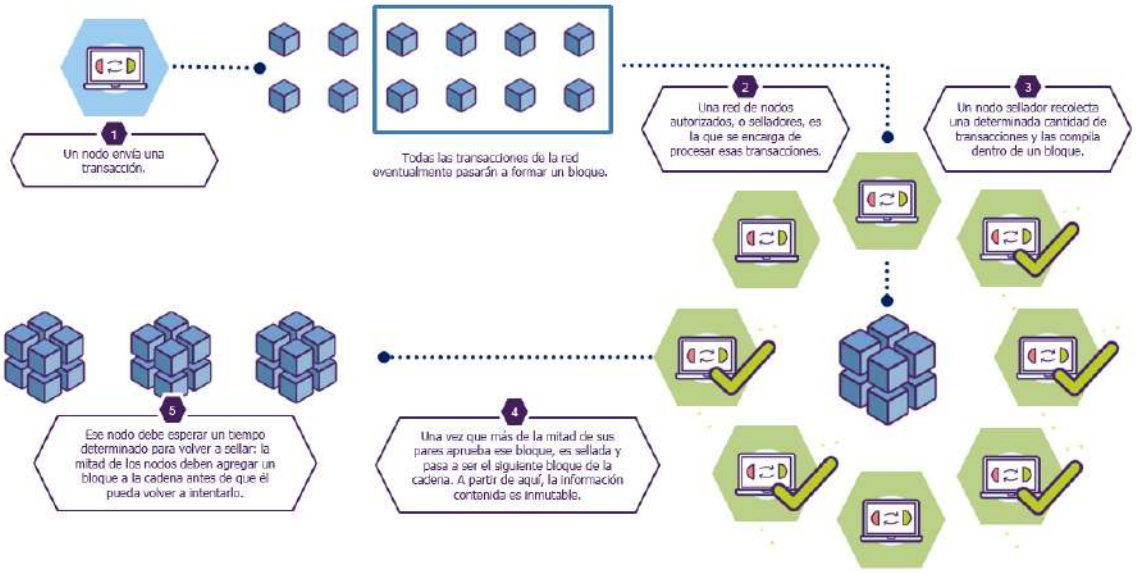

Figura 2.3: Proceso para crear la cadena de bloques [1].

2. Desde el punto de vista técnico, *Blockchain* puede definirse como una **estructura de datos** (figura 2.4) basada en listas enlazadas, en la que cada bloque está conectado al anterior mediante punteros criptográficos (*hash*), garantizando así la integridad y detectando cualquier modificación no autorizada [18]. Este diseño puede compararse con las páginas numeradas de un libro físico, donde cualquier cambio o eliminación se hace evidente [3].

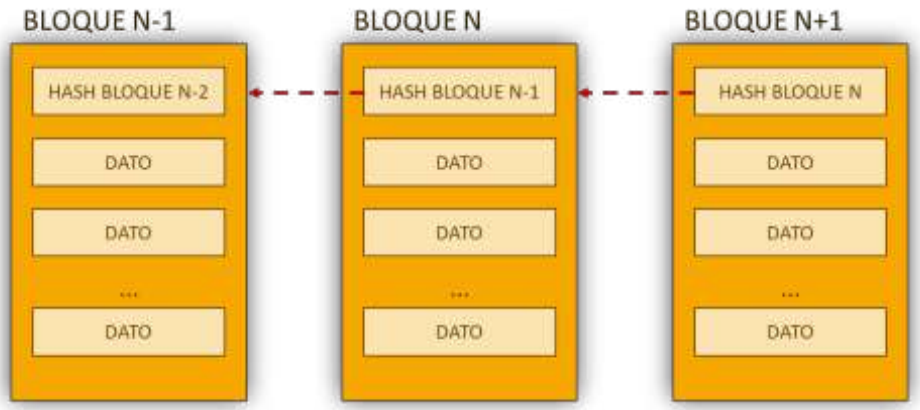

Figura 2.4: Estructura de datos [2].

3. Finalmente, *Blockchain* también constituye un **conjunto de tecnologías** (figura 2.5), que combina estructuras de datos, algoritmos de consenso, técnicas avanzadas de criptografía y red de ordenadores, asegurando así la seguridad y transparencia [3].

Figura 2.5: Conjunto de tecnologías.

En resumen, por *Blockchain* podemos referirnos a un algoritmo, una estructura de datos o un conjunto de tecnologías. En esta memoria usualmente nos referimos a un conjunto de tecnologías.

2.1.2. Características de la tecnología *Blockchain*

Para comprender plenamente el alcance de su impacto y utilidad práctica, es fundamental analizar en detalle las principales características que definen su naturaleza y funcionamiento. Entre ellas nos encontramos la descentralización, la inmutabilidad, la transparencia, la seguridad criptográfica y la trazabilidad.

2.1.2.1. Descentralización

La descentralización es, quizás, la característica más conocida y revolucionaria. A diferencia de las estructuras tradicionales que dependen de una autoridad centralizada para gestionar y validar las operaciones, *Blockchain* distribuye esta responsabilidad entre múltiples participantes, conocidos como nodos. Como afirma Drescher (2017), la esencia de *Blockchain* radica precisamente en esta capacidad de distribuir no solo los datos, sino también el control y la autoridad sobre ellos [3].

En un modelo descentralizado, ningún participante posee control absoluto o privilegiado sobre la red. Esto evita que exista un único punto de fallo, lo que incrementa considerablemente la resistencia del sistema ante ataques, errores o manipulaciones malintencionadas [19]. En consecuencia, *Blockchain* fomenta un entorno más equitativo, transparente y robusto, que es fundamental para aplicaciones que requieren altos estándares de confiabilidad, como los sistemas financieros o registros públicos.

Esta característica es la que ha incentivado a miles de desarrolladores e investigadores a sumergirse en lo que es la Blockchain; un sinfín de oportunidades se abren cuando se piensa en la idea que trae consigo, la desintermediación. La desaparición de entidades que acrediten confianza como los notarios, cámaras de acreditación, etc; supondría de un ahorro para todas las personas y una reducida de costes para las empresas.

2.1.2.2. Inmutabilidad

La inmutabilidad es otro pilar esencial. Una vez que la información se registra en un bloque, esta queda prácticamente sellada mediante mecanismos criptográficos. Esto es debido a que la estructura criptográfica se basa en funciones *hash* y en la conexión directa entre bloques, formando una cadena en la que cualquier intento de alterar un solo dato desencadenaría alteraciones visibles y verificables en todo el sistema [20].

¿Por qué es inmutable? ¿Cómo funciona? Cuando envías un transacción a red esta es almacenada en un bloque con muchas otras transacciones; este conjunto y otros datos, entre los que se encuentra el hash del bloque anterior, son pasados a través de una función hash lo que da un valor fijo de 256 bits (32 bytes, 64 dígitos en hexadecimal que será utilizado como puntero; esta longitud depende de la implementación, por ejemplo en Bitcoin y Ethereum son 256 bits [7, 21]. Si se intentara cambiar un dato de alguna transacción el valor hash obtenido anteriormente no concordaría con el nuevo. Por lo tanto ya no apuntaría a los datos, perdiendo todos los datos desde ese punto debido a que se guarda el hash del bloque anterior también; esto genera una reacción en cadena que destruye la cadena completa como puede verse en la figura 2.6. En la subsección 2.1.3 se profundiza en las funciones hash y claves criptográficas.

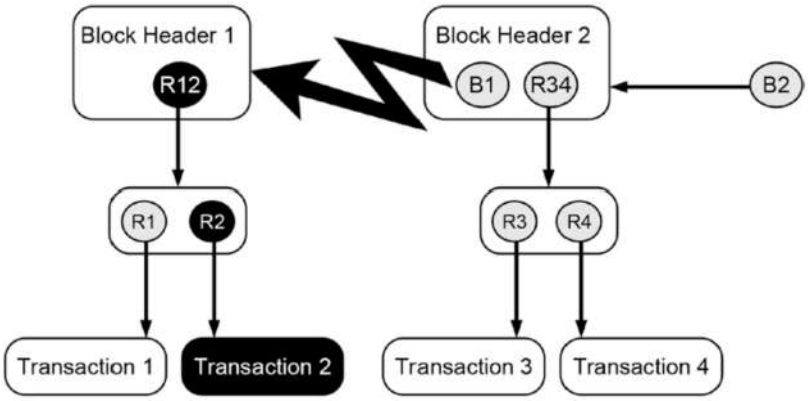

Figura 2.6: Cuando una transacción es cambiada, el hash no coincide [3].

La inmutabilidad aporta una confianza sin precedentes en la integridad de los datos. Los registros almacenados en la cadena son permanentes y auditables, proporcionando seguridad adicional en contextos sensibles como registros de identidad, transacciones monetarias y sistemas de votación electrónica [18]. Por supuesto, también en la trazabilidad alimentaria.

2.1.2.3. Transparencia

Se entiende por transparencia a la accesibilidad abierta y verificable de la información.

Antonopoulos y Gavin Wood (2018) destacan que la transparencia en *Blockchain* no implica necesariamente la exposición de información personal sensible, sino más bien la capacidad de verificar públicamente todas las transacciones y eventos que suceden dentro de la red [21]. Esto se logra a través de las funciones hash, estas

pueden utilizarse como prueba de valor, de posesión o de conocimiento verificable. Puedes mostrar abiertamente el valor hash de una serie de datos, nadie sabrá que serie de datos generaron ese valor hash. Pero sí puedes acreditar luego ante cualquier entidad que te lo solicite como podría ser el departamento de seguridad alimentaria de un país, que esos datos 'Y' corresponden a 'X', y que nunca fueron modificados. Dejas rastro en la cadena de una prueba inmutable de la cuál no puedes negar haber "firmado".

En las redes públicas, esta transparencia facilita auditorías independientes, previniendo actividades fraudulentas y promoviendo un ambiente de confianza compartida [20].

En redes privadas o permisionadas, aunque la transparencia está limitada a un grupo selecto de participantes, continúa proporcionando mecanismos claros y efectivos para validar y monitorear procesos críticos [20].

2.1.2.4. Seguridad criptográfica

La seguridad criptográfica se refiere a características como autorización y validación, que garantizan que solo usuarios autorizados realicen acciones y que estas sean verificadas mediante claves privadas y públicas; confidencialidad y privacidad, que protegen datos y ocultan identidades usando técnicas como direcciones temporales o cifrado; identificación única, donde las claves públicas actúan como IDs y las privadas autentican al usuario; e integridad de la información, asegurada por hashes que detectan cualquier alteración en los datos [3].

Esta seguridad implica el uso avanzado de criptografía, incluyendo funciones hash, firmas digitales y pares de claves asimétricas (públicas y privadas) (se profundiza en la subsubsección 2.1.3.8), que aseguran que únicamente aquellos autorizados puedan acceder y validar transacciones o información específica. Por ejemplo, en la autorización, un usuario firma una transacción con su clave privada, y la red la verifica con la clave pública correspondiente, eliminando intermediarios. Para la confidencialidad, se pueden usar direcciones temporales o cifrado, aunque la privacidad varía según la Blockchain. La identificación única se logra con claves públicas como identificadores y privadas para autenticación. La integridad se protege con hashes, que generan una huella digital única; cualquier cambio en los datos altera el hash, detectando manipulaciones [3, 21].

Estas técnicas proporcionan no solo una 'semi' confidencialidad y privacidad (ya

que en muchas Blockchains las transacciones son visibles, pero se pueden ocultar detalles), sino también mecanismos sólidos para identificar inequívocamente a los participantes y proteger la integridad de la información frente a manipulaciones. De esta forma, Blockchain se convierte en una herramienta robusta y confiable en un contexto digital cada vez más vulnerable [19].

2.1.2.5. Trazabilidad

Finalmente, la trazabilidad emerge como una característica clave que hace posible rastrear detalladamente cualquier elemento o transacción almacenada. La estructura en cadena, junto con la inmutabilidad, permite seguir paso a paso el recorrido histórico de un activo, desde su origen hasta su situación actual, facilitando auditorías precisas y efectivas [3].

Singhal et al. (2018) destacan el valor práctico de esta característica en sectores tan diversos como la industria alimentaria, farmacéutica o las cadenas de suministro logísticas, donde el control del historial completo es esencial para garantizar la calidad y seguridad [19]; razón de la realización de este trabajo como se dijo en la sección 1.1. La trazabilidad, por lo tanto, no solo fortalece la confianza, sino que también permite detectar rápidamente errores o fraudes, mejorando notablemente la eficiencia operativa [18]. Esta subsubsección se ampliará con el capítulo 3.

2.1.3. Componentes de la tecnología *Blockchain*

Una vez entendidas sus características podemos hablar de sus componentes uno por uno, se dejará alguno sin mencionar pero se intentará abarcar todos los posibles. Entre ellos se encuentran:

2.1.3.1. Red de Nodos

En el corazón de cualquier red *Blockchain* se encuentra una red de nodos, que son los dispositivos que participan activamente en la propagación, validación y almacenamiento de las transacciones. Cada nodo en la red puede ser visto como una unidad autónoma que sigue un conjunto común de reglas para interactuar con otros nodos. Estos nodos no son más que ordenadores, móviles, Raspberry Pi, etc que están ejecutando un cliente Blockchain (se profundiza en la subsección 2.1.5). Como explica

Drescher (2017), la descentralización implica que no hay una única entidad controlando la red, sino que el control se distribuye entre los participantes [3].

Esta red es peer-to-peer como se aprecia en la figura 2.7, con conexiones directas entre nodos que establecen vínculos uno a uno. Cada nodo puede tener un número variable de conexiones, dependiendo de su configuración y de las reglas de la red en la que participa. El límite máximo de conexiones está determinado por la implementación específica de la red y el software del nodo.

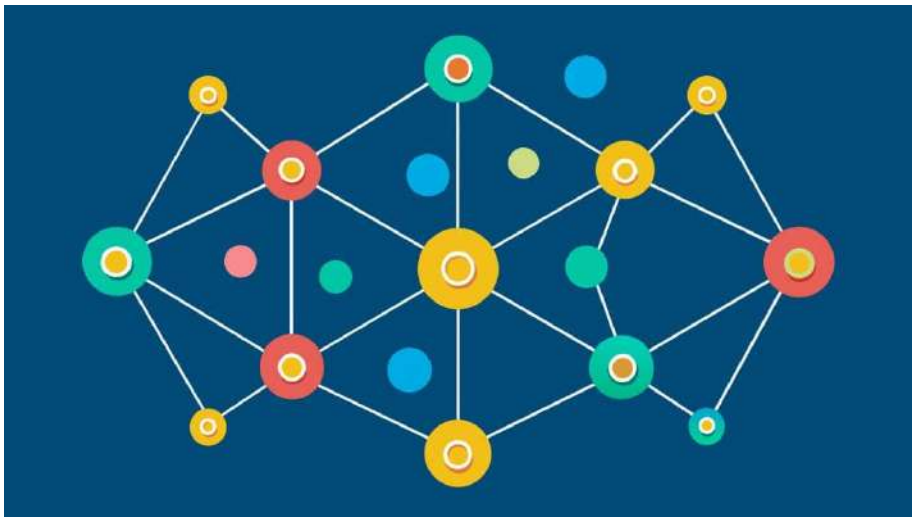

Figura 2.7: Mapa conceptual de una red de nodos [4].

Los protocolos para descubrir nodos y transmitir datos varían según el diseño y propósito de la red, pero generalmente incluyen mecanismos para localizar pares y compartir información de manera eficiente.

Existen diferentes tipos de nodos: los nodos completos mantienen una copia de todo el registro de la *Blockchain* y validan todas las transacciones y bloques, mientras que los nodos ligeros almacenan solo una parte como es la raíz del árbol de *hashes* (*Merkle Tree*) y dependen de los nodos completos para obtener información adicional [19]. Los mineros o validadores (a veces se confunde con validar las transacciones y bloques, es decir, verificar la firma), dependiendo del mecanismo de consenso utilizado, son nodos que están encargados de agregar nuevos bloques a la cadena de bloques, lo que puede requerir un alto poder computacional en redes como Bitcoin, que utiliza el mecanismo *PoW* [18].

2.1.3.2. Transacciones

Las transacciones son el núcleo de las interacciones dentro de una *Blockchain*. Cada vez que un usuario realiza una operación como el ejemplo en la figura 2.8, esta operación se registra como una transacción dentro de un bloque. Las transacciones son atómicas, lo que significa que ocurren de forma completa o no ocurren en absoluto, lo que elimina el riesgo de realizar transacciones parciales [19].

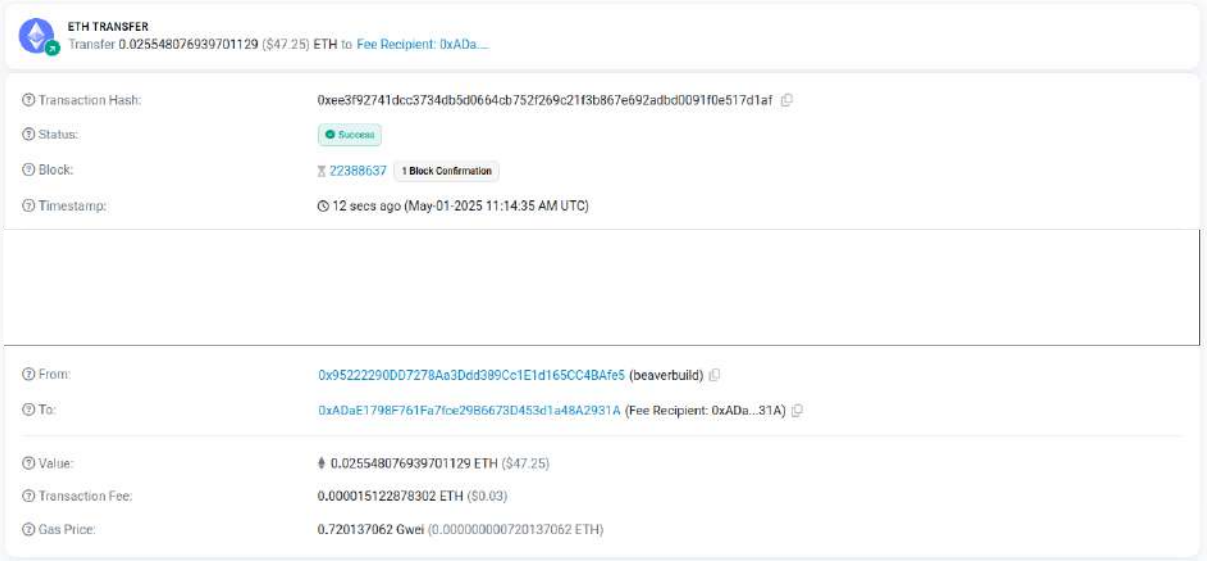

Figura 2.8: Una transaccion en la Mainnet de Ethereum vista con Etherscan.io [5].

En *Blockchain*, cada transacción está vinculada a una dirección que se genera mediante la clave pública del usuario. Para realizar una transacción, el usuario utiliza su clave privada para firmarla digitalmente, asegurando que la transacción es legítima y que solo el propietario de los fondos puede autorizar su uso. La transacción se transmite a la red, donde los nodos verifican su validez. Si es válida, la transacción se incluye en un bloque, que a su vez es validado por el mecanismo de consenso de la red antes de ser añadido [21].

Una vez que una transacción se incluye en un bloque, empieza a recibir lo que se conoce como *confirmaciones*. Cada vez que un nuevo bloque se añade encima del bloque que contiene la transacción [18], se cuenta como una nueva confirmación; en la figura 2.8 puede verse que la transacción de ejemplo es muy reciente porque solo tiene una confirmación. Cuantas más confirmaciones tiene una transacción, más difícil es revertirla (más complicado es que ocurra una bifurcación de la cadena), lo que incrementa su seguridad.

En redes como Ethereum, una transacción puede servir para múltiples propósitos:

enviar criptomonedas (como ETH), interactuar con contratos inteligentes (por ejemplo, para intercambiar tokens o participar en una aplicación descentralizada), desplegar nuevos contratos, o simplemente almacenar datos arbitrarios en la cadena [21].

Una forma útil de entender una transacción es como un mensaje firmado entre pares. El emisor genera un mensaje que contiene una orden (por ejemplo, "envía 1 ETH a la dirección B"), lo firma con su clave privada y lo transmite a la red. Los nodos, actuando como observadores imparciales, validan el mensaje y, si es correcto, lo registran en el libro compartido que es la *Blockchain*.

Las transacciones suelen incluir varios campos importantes [21, 18, 3], veáse la figura 2.8:

- **Hash**: Identificador único de la transacción, generado al aplicar una función hash sobre su contenido.
- **From**: Dirección del emisor.
- **To**: Dirección del destinatario.
- **Value**: Cantidad de criptomoneda a transferir.
- **Data**: Información adicional, como la llamada a una función en un contrato inteligente.
- **Nonce**: Número que asegura que las transacciones del mismo emisor se ejecutan en orden.
- **Signature**: Firma digital del emisor.

Además, en Ethereum y otras redes basadas en contratos inteligentes, se incluyen parámetros relacionados con el coste de la operación [21]:

- **Gas Limit**: Límite máximo de unidades de gas que se pueden consumir.
- **Gas Price**: Precio a pagar por cada unidad de gas (en Gwei), que afecta la prioridad de la transacción.

2.1.3.3. Bloques

En una Blockchain, los bloques son la unidad básica de almacenamiento de datos y sirven para agrupar un conjunto de transacciones verificadas. Cada bloque actúa

como una página en un libro contable distribuido: registra transacciones y enlaza con el bloque anterior, formando una cadena secuencial que garantiza la integridad del historial [20].

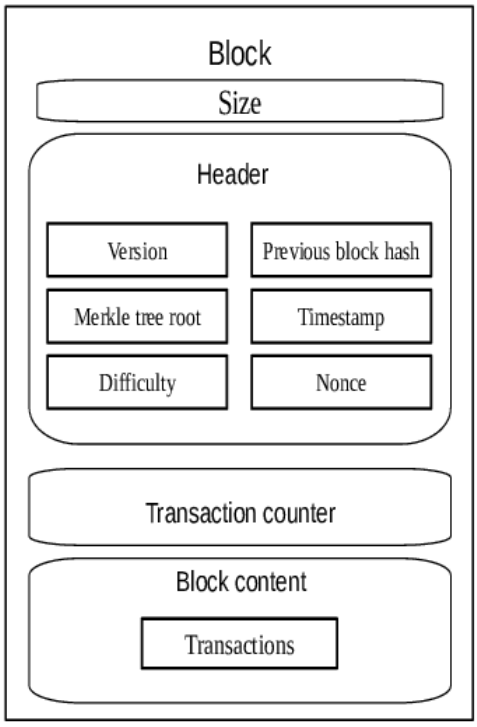

Figura 2.9: Estructura habitual de un bloque [6].

Cada bloque se compone de tres partes principales, como se muestra en la figura 2.9:

- **Cabecera del bloque (Header)**: contiene metadatos esenciales como Previous block hash, Merkle tree root, Timestamp, Version, Nonce y Difficulty.
- **Contador de transacciones**: indica cuántas transacciones hay en el bloque.
- **Contenido del bloque**: una lista ordenada de todas las transacciones incluidas.

La cabecera es especialmente crítica, ya que su hash sirve como identificador único del bloque y es lo que se utiliza para enlazar con el siguiente bloque; por ello, se le dedica una hueco para ella en la subsección 2.1.3.4. Esta estructura encadenada garantiza la inmutabilidad: cualquier intento de modificar una transacción alteraría la raíz del Merkle tree, el hash del bloque, y en consecuencia invalidaría todos los bloques siguientes, lo cual sería evidente para toda la red.

Un bloque muy particular es el llamado bloque génesis, que es el primer bloque de toda la Blockchain. A diferencia de los demás, no hace referencia a un bloque anterior

(su campo previous block hash está vacío o predefinido) y suele ser generado manualmente con parámetros definidos por los desarrolladores de la red. El bloque génesis establece el punto de partida desde el cual se construye toda la cadena posterior.

El tiempo que tarda la red en generar un nuevo bloque se conoce como tiempo de bloque. Este valor depende del protocolo de consenso y tiene impacto directo en la latencia y capacidad de la red. Por ejemplo:

- En Bitcoin, el tiempo promedio de bloque es de aproximadamente 10 minutos [3].
- En Ethereum (post-merge), se generan bloques cada 12 segundos en promedio [21].
- Otras redes, como Solana, tienen tiempos de bloque inferiores al segundo [22].

La propuesta y validación de un nuevo bloque está directamente relacionada con el algoritmo de consenso utilizado. Por ejemplo, en Proof of Work, los mineros compiten para encontrar un *nonce* que produzca un hash válido bajo cierta dificultad. El primero en lograrlo propone un nuevo bloque, que luego es validado por los demás nodos. En Proof of Stake, la elección del validador que propone el siguiente bloque se basa en la cantidad de criptomonedas en juego y otros factores. En ambos casos, solo los bloques validados por consenso son añadidos a la cadena, reforzando así la seguridad y confiabilidad de la red [19].

2.1.3.4. Cabecera de un bloque

La cabecera de un bloque (block header) es una parte crítica de la estructura de cada bloque en una Blockchain, ya que concentra los metadatos necesarios para validar, enlazar y asegurar el contenido del bloque sin necesidad de procesar todas sus transacciones. Esta sección es la responsable de garantizar la integridad criptográfica entre bloques y permite la verificación eficiente de la cadena mediante estructuras como los árboles de Merkle [3].

Una cabecera de bloque generalmente incluye los siguientes campos clave [3]:

- **Hash del bloque anterior**: asegura el enlace con el bloque anterior y garantiza la integridad de la cadena.

- **Raíz del árbol de Merkle (Merkle root)**: es el hash resultante de la parte superior de un árbol de Merkle, que representa la integridad de todos los datos dentro del árbol.

- **Timestamp**: marca temporal que indica cuándo se propuso el bloque. Esta marca contribuye al orden cronológico de los bloques.

- **Nonce**: valor aleatorio que los mineros ajustan durante el proceso de prueba de trabajo (Proof of Work) hasta encontrar un hash válido.

- **Dificultad (difficulty target)**: define cuán difícil es encontrar un hash válido para el bloque. Es ajustado dinámicamente en redes como Bitcoin cada 2016 bloques.

- **Versión**: indica el formato del bloque y puede cambiar con mejoras de protocolo.

- **State root** / **Extra data (dependiendo de la red)**: en blockchains como Ethereum, se incluye información adicional como el estado global del sistema, la raíz del almacenamiento de contratos inteligentes o el gas utilizado [21].

En la figura 2.9, estos campos pueden verse representados en la parte superior del bloque. La cabecera tiene una doble función: por un lado, actúa como identificador único del bloque al ser utilizada para calcular su hash; por otro, sirve como punto de verificación en procesos como la sincronización de nodos, validación ligera de bloques (light clients) y en el consenso distribuido.

Cabe destacar que los campos de la cabecera pueden variar ligeramente entre diferentes protocolos de Blockchain. Por ejemplo, Ethereum incluye raíces de estructuras Patricia-Merkle que representan el estado del sistema [21].

2.1.3.5. Funciones *Hash*

Las funciones hash son algoritmos deterministas que transforman cualquier entrada de datos en una secuencia binaria de longitud fija. Esta transformación es irreversible, lo que significa que no es posible reconstruir los datos originales a partir del hash generado. En la figura 2.10 se puede observar cómo pequeñas variaciones en la entrada producen resultados completamente distintos [3].

Text to Be Hashed	Output
Hello World! 0	4EE4B774
Hello World! 1	3345B9A3
Hello World! 2	72040842
Hello World! 3	02307D5F
...	
Hello World! 613	E861901E
Hello World! 614	**00068A3C**
Hello World! 615	5EB7483F

Figura 2.10: Pequeños cambios implican grandes diferencias [3].

En el contexto de las Blockchain, las funciones hash desempeñan un papel esencial en múltiples niveles. Son utilizadas para:

- Generar identificadores únicos de bloques y transacciones.
- Enlazar bloques entre sí a través el hash de la cabecera del bloque anterior.
- Verificar la integridad de los datos y detectar cualquier intento de manipulación.
- Construir árboles de Merkle que permiten la validación rápida y eficiente de transacciones individuales.

Cada Blockchain emplea funciones hash específicas: Bitcoin utiliza SHA-256 [3] (Secure Hash Algorithm 256 bits), mientras que Ethereum emplea Keccak-256, una variante de la función estandarizada como SHA-3 [21]. Estas funciones producen salidas de 256 bits (32 bytes), lo que representa un equilibrio entre seguridad y eficiencia computacional.

La longitud fija del hash no solo garantiza una representación uniforme para cualquier tamaño de entrada, sino que también permite una verificación extremadamente rápida por parte de los nodos. En lugar de procesar el contenido completo de un bloque o transacción, basta con comparar su hash con el esperado.

Otra aplicación crucial es en la generación de direcciones de cuentas. En Bitcoin, por ejemplo, una dirección se deriva a partir de la clave pública del usuario tras aplicar

múltiples funciones hash (SHA-256 seguido de RIPEMD-160) [3] y añadir una suma de comprobación. Ethereum genera direcciones aplicando Keccak-256 directamente sobre la clave pública y extrayendo los últimos 20 bytes del resultado [21]. Este proceso permite que las direcciones actúen como identificadores compactos, seguros y únicos en la red.

Las funciones hash son la piedra angular de la seguridad, eficiencia y estructura interna de cualquier Blockchain. Sin ellas, no sería posible mantener la integridad de los datos, la trazabilidad de los bloques ni la verificación eficiente de transacciones.

2.1.3.6. Cadena de Bloques (Blockchain)

La cadena de bloques (*Blockchain*) puede entenderse como una estructura de datos secuencial y enlazada, donde cada bloque contiene un conjunto de transacciones y un puntero criptográfico —el *hash*— que lo conecta con el bloque anterior [18].

A diferencia de las listas enlazadas tradicionales, los enlaces entre bloques se basan en funciones hash criptográficas. Esto significa que cualquier alteración en el contenido de un bloque modificaría su hash, invalidando también el hash almacenado en el bloque siguiente y, por lo tanto, rompiendo toda la cadena. Esta propiedad dota a la Blockchain de su característica más distintiva: la inmutabilidad. La estructura es, por diseño, *append-only* [3]: se pueden añadir nuevos bloques al final, pero nunca modificar o eliminar los existentes.

Esta estructura es esencial para prevenir el *doble gasto* (*double spending*), un problema clásico en los sistemas digitales de transferencia de valor. Este ocurre cuando un mismo activo digital se intenta gastar más de una vez. En Blockchain, cada transacción válida se registra de forma permanente en un bloque. Como todos los nodos tienen una copia sincronizada de la cadena, cualquier intento de reutilizar una transacción previamente confirmada es rechazado automáticamente por los nodos al verificar que ese saldo ya fue utilizado. La combinación de inmutabilidad, verificación descentralizada y orden cronológico impide la duplicación de pagos [7].

Ocasionalmente, pueden producirse *bifurcaciones* (*forks*) en la cadena. Estas ocurren cuando dos bloques son propuestos casi simultáneamente y parte de los nodos aceptan uno mientras otros aceptan el otro. Esto genera una cadena temporalmente dividida. Los algoritmos de consenso, como Prueba de Trabajo (PoW) o Prueba de Participación (PoS), resuelven estas divisiones automáticamente: se considera válida la cadena que acumule mayor “trabajo” (en PoW) o mayor participación (en PoS), y el

resto de ramas se descartan. Este mecanismo garantiza la unicidad y coherencia del historial de bloques aceptado por la red [18].

2.1.3.7. Mecanismos de Consenso

El mecanismo de consenso es el proceso mediante el cual los nodos de la red alcanzan un acuerdo sobre qué transacciones son válidas y cuáles no , sin la necesidad de una autoridad central. La existencia de un consenso es lo que permite que la *Blockchain* sea confiable y segura. Existen diversos mecanismos de consenso, y la elección de uno u otro depende de la red en cuestión.

Uno de los mecanismos más conocidos es *PoW*, utilizado por Bitcoin. En *PoW*, los mineros deben resolver problemas matemáticos complejos para validar un bloque y agregarlo a la cadena [18]. Este proceso es intensivo en recursos, ya que requiere una gran cantidad de poder computacional para resolver los problemas y asegurar la integridad de la red.

Por otro lado, *PoS*, que es más eficiente en términos energéticos, selecciona a los validadores en función de la cantidad de criptomonedas que están dispuestos a mantener y poseer en la red. Este mecanismo es utilizado por otras criptomonedas como Ethereum 2.0, y en lugar de depender del poder computacional, depende del compromiso económico de los validadores [21].

Esos dos son los más relevantes y conocidos, pero no son los únicos. Se les dedica la subsección 2.1.4 para describirlos en mayor detalle.

2.1.3.8. Claves Criptográficas y Firmas Digitales

Las claves criptográficas constituyen un pilar fundamental para la seguridad y la autenticidad de las transacciones en una red *Blockchain*. Cada usuario posee un par de claves: una *clave pública*, utilizada para generar su dirección pública (similar a una cuenta de destino), y una *clave privada*, que le otorga control total sobre sus activos digitales. Este esquema asimétrico permite separar la capacidad de verificar de la capacidad de firmar, reforzando la seguridad [3].

Tanto Bitcoin como Ethereum emplean criptografía de curva elíptica mostrada en la figura 2.11, específicamente el algoritmo ECDSA (Elliptic Curve Digital Signature Algorithm), basado en la curva `secp256k1`. Esta curva permite generar claves criptográficas

seguras y eficientes computacionalmente gracias a sus propiedades matemáticas, como la dificultad del problema del logaritmo discreto en curvas elípticas [7].

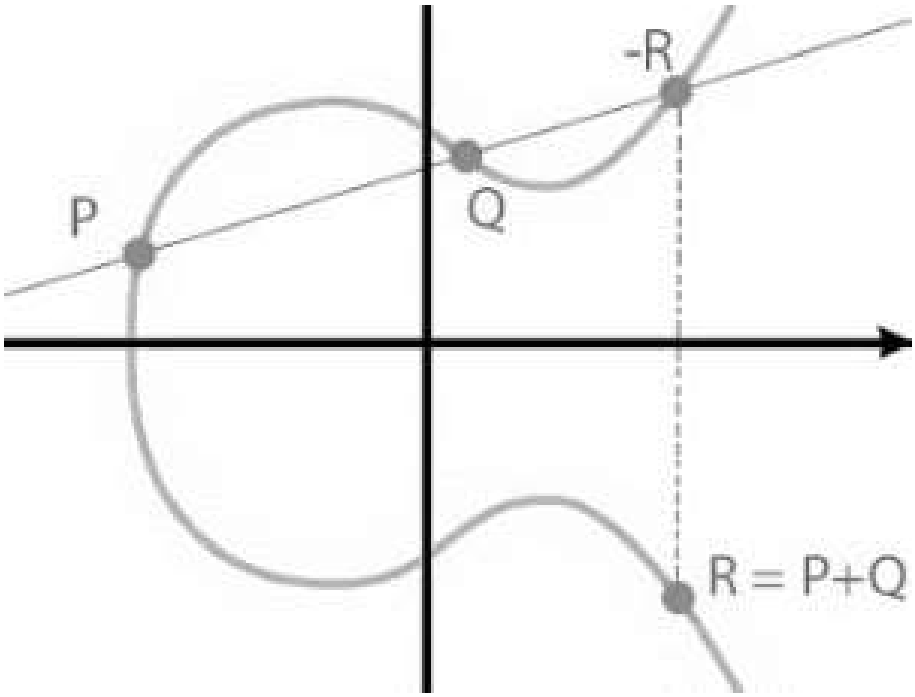

Figura 2.11: ECC y su operación de suma.

Cuando un usuario desea realizar una transacción, firma digitalmente el valor hash del contenido de la transacción —incluyendo campos como emisor, receptor, cantidad y datos adicionales— con su clave privada. El resultado es una *firma digital* que viaja junto a la transacción. Cualquier nodo de la red puede verificar la autenticidad de dicha transacción utilizando la clave pública del firmante, sin necesidad de conocer su clave privada [21].

La firma digital garantiza las siguientes propiedades fundamentales, ya mencionadas en la subsubsección 2.1.2.4:

- **Autenticidad:** asegura que la transacción fue generada por el propietario legítimo de los fondos.
- **Integridad:** cualquier alteración en los datos invalida la firma.
- **No repudio:** el firmante no puede negar posteriormente haber autorizado la transacción.

2.1.3.9. *Smart Contracts*

Los *smart contracts* (contratos inteligentes) son programas autoejecutables desplegados en la cadena de bloques que se activan automáticamente cuando se cumplen condiciones previamente establecidas. Como explican Antonopoulos y Wood en *Mastering Ethereum* [21], estos contratos permiten construir aplicaciones descentralizadas (DApps) que pueden operar de manera autónoma, transparente y segura, sin necesidad de intermediarios.

Un *smart contract* puede ejecutar transacciones, mantener registros persistentes, responder a eventos y administrar activos digitales. Gracias a estructuras como `event`, `struct`, y otros tipos de datos complejos, estos contratos pueden implementar lógica empresarial sofisticada y ofrecer interfaces para que los usuarios y otros contratos interactúen con ellos. Por ejemplo, los `event` permiten emitir señales durante la ejecución del contrato, facilitando el registro de actividades para aplicaciones fuera de la cadena (off-chain); mientras que los `struct` permiten organizar datos relacionados de forma eficiente.

Los contratos en Ethereum son escritos en el lenguaje de alto nivel `Solidity`, compilados a bytecode y desplegados en la red mediante una transacción especial. Una vez desplegado, cualquier usuario o contrato puede interactuar con él llamando a sus funciones públicas a través de nuevas transacciones (estas interacciones son fundamentales para el funcionamiento de cualquier aplicación descentralizada), lo que permite una amplia variedad de casos de uso como [21]:

- Automatización de pagos y transferencias.
- Sistemas de votación transparentes.
- Tokens fungibles (ERC20) y no fungibles (ERC721).
- Organizaciones autónomas descentralizadas (DAOs).

Riesgos de la Inmutabilidad

Una de las mayores ventajas de los *smart contracts* es también uno de sus principales desafíos: su inmutabilidad. Una vez desplegado en la red, un contrato no puede ser modificado, a menos que haya sido diseñado con un mecanismo explícito de actualización. Esta característica garantiza integridad y evita manipulaciones maliciosas, pero también implica que errores en el código son permanentes y potencialmente explotables.

Un ejemplo emblemático de las consecuencias de esta inmutabilidad fue el ataque a *The DAO* en 2016, cuando un contrato mal diseñado permitió a un atacante drenar más de 3.6 millones de ETH. Este evento provocó una bifurcación en Ethereum y sirvió como advertencia sobre la necesidad de mejores prácticas en desarrollo de contratos inteligentes.

Prácticas de Desarrollo Seguro

Para mitigar estos riesgos, se han desarrollado buenas prácticas para el diseño y despliegue de contratos:

- Uso de contratos delegados (*delegatecall*) para permitir actualizaciones.
- Pruebas exhaustivas y auditorías de seguridad independientes.
- Uso de herramientas como `Hardhat` para detectar vulnerabilidades.
- Limitar el uso de funciones críticas mediante modificadores como `onlyOwner`.
- Aplicar patrones de control de errores y recuperación, como `require()`, `revert()` y `assert()`.

Además, es común diseñar contratos con un enfoque modular, limitando la complejidad de cada componente y favoreciendo la reutilización de código bien auditado.

2.1.3.10. Merkle Tree

Los árboles de Merkle, también llamados árboles binarios de hashes, son estructuras fundamentales en el diseño de la mayoría de blockchains como, por ejemplo, Bitcoin. El árbol de Merkle es una estructura de datos criptográfica en forma de árbol binario que permite condensar y verificar de manera eficiente un conjunto de transacciones [7]. Cada hoja del árbol contiene el hash de una transacción individual, y cada nodo intermedio almacena el hash de la concatenación de los hashes de sus hijos. La estructura culmina en un único valor: la raíz de Merkle (*Merkle Root*) como muestra la figura 2.12, que representa un compromiso criptográfico de todo el contenido del bloque [7].

Esta organización se basa en el principio de funciones hash jerárquicas: una estrategia que permite reducir un gran conjunto de datos a un único identificador criptográfico. Este diseño implica que cualquier modificación en una transacción afecta a los

hashes superiores, propagándose hasta la raíz y alterando la huella digital del bloque completo. Por este motivo, la raíz se almacena en la cabecera del bloque, facilitando la validación de la integridad de los datos.

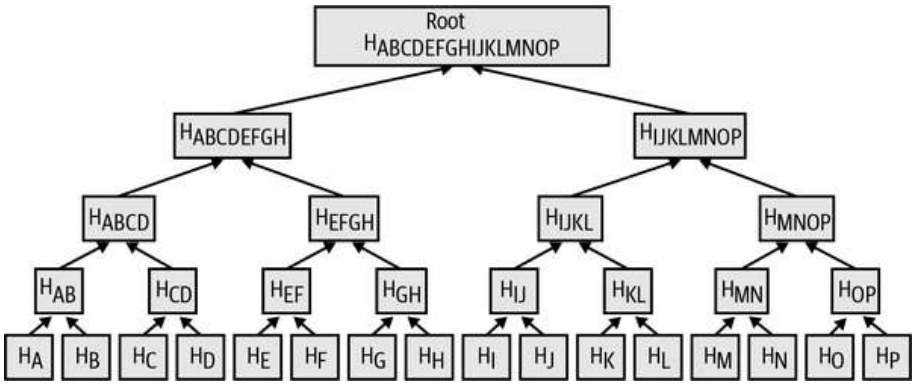

Figura 2.12: Merkle Tree [7].

Una característica importante del árbol de Merkle es su eficiencia para realizar pruebas de inclusión. Para verificar que una transacción está contenida en un bloque, no es necesario descargar el bloque entero: basta con disponer de una prueba parcial de hashes, conocida como *Merkle path*, que requiere solo $\log_2(n)$ elementos para una verificación completa. Esta propiedad es clave para los nodos ligeros que operan bajo el esquema SPV (*Simplified Payment Verification*) [13, 7].

Cuando el número de transacciones es impar, Bitcoin duplica el último hash para mantener una estructura de árbol balanceada. Sin embargo, este enfoque presenta una vulnerabilidad conocida (CVE-2012-2459): la posibilidad de construir árboles con diferentes conjuntos de transacciones pero con la misma raíz de Merkle. Para mitigar este riesgo, Bitcoin Core implementa verificaciones adicionales que aseguran que los árboles con duplicación no introduzcan ambigüedades de validación [7].

A pesar de esta limitación, la eficiencia del árbol de Merkle es innegable: incluso en bloques con miles de transacciones, se puede demostrar la pertenencia de una sola con unas pocas centenas de bytes. Así, los árboles de Merkle representan una solución elegante y práctica para la verificación de integridad en sistemas distribuidos como Bitcoin.

2.1.3.11. Merkle Patricia Trie

Ethereum, en cambio, utiliza una estructura de datos denominada **Merkle Patricia Trie** para gestionar y verificar su estado global, que incluye cuentas, saldos y almacenamiento de contratos inteligentes. Esta estructura combina las propiedades de los

árboles de Merkle y los tries de Patricia, permitiendo una representación eficiente y verificable del estado [21, 23].

En el Patricia Trie, los datos se representan como pares clave-valor, y cada nodo puede ser una hoja, un nodo de extensión o un nodo rama, según su posición y función. Esta estructura soporta operaciones como inserción, búsqueda y verificación en tiempos logarítmicos, lo cual es fundamental para el rendimiento de Ethereum.

La figura 2.13 muestra un ejemplo simplificado del *World State Trie* de Ethereum, ilustrando cómo se estructuran los nodos y cómo las claves se dividen en grupos de 4 bits para el enrutamiento dentro del trie.

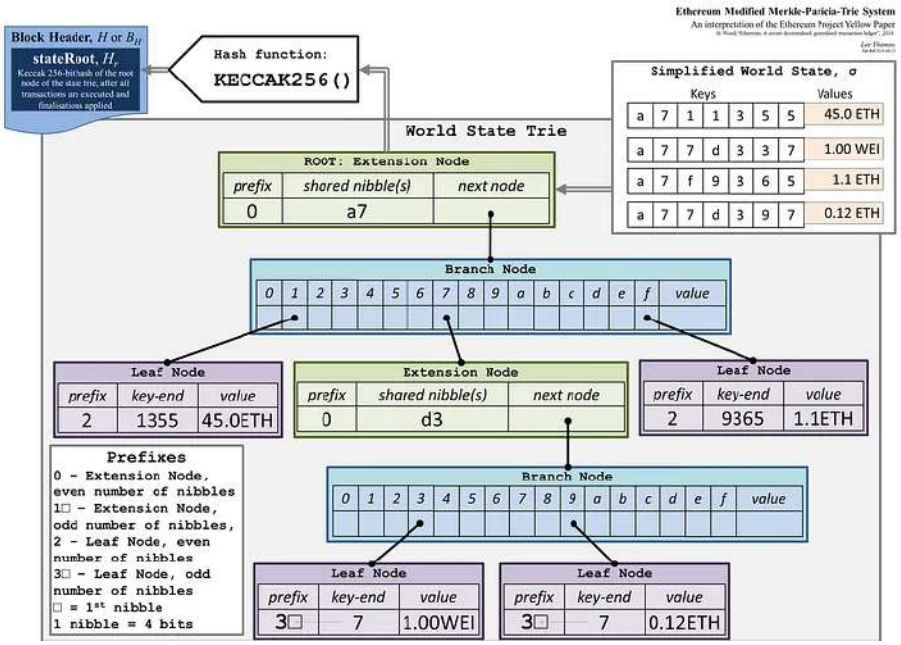

Figura 2.13: Esquema sobre Merkle Patricia Trie [8].

Es una estructura determinista y criptográficamente segura. Cada nodo del trie tiene un *hash* único derivado de su contenido, lo que garantiza que cualquier modificación en los datos subyacentes resulte en un cambio en la raíz. Esta propiedad permite verificar la integridad del estado sin necesidad de acceder a todos los datos individuales.

En Ethereum, existen tres tries principales [21]:

- **State Trie**: almacena el estado global, mapeando direcciones de cuentas a sus respectivos estados.
- **Storage Trie**: para cada contrato inteligente, este trie almacena su almacenamiento interno, mapeando claves a valores.

- **Transaction Trie**: contiene todas las transacciones incluidas en un bloque específico.

Cada uno de estos tries tiene su raíz de Merkle correspondiente, la cual se incluye en la cabecera del bloque. Esto permite una verificación eficiente del estado y las transacciones.

Utiliza codificación *RLP* (Recursive Length Prefix) para serializar los nodos del árbol, y emplea una codificación compacta de las claves, optimizando el almacenamiento y la eficiencia en las operaciones de inserción, búsqueda y eliminación [21, 23].

Sin embargo, también presenta desafíos, especialmente en términos de eficiencia en la actualización del estado y el tamaño de las pruebas. Por ello, Ethereum está explorando la transición hacia estructuras más eficientes, como los **Verkle Trees**, que prometen mejorar la escalabilidad y reducir los requisitos de almacenamiento.

2.1.4. Algoritmos de Consenso más relevantes

El primer algoritmo de consenso que se utilizó en *Blockchain* fue el PoW. Luego aparecieron numerosas variantes de este mismo PoW, hasta que finalmente se dió una ola de nuevos consensos cada uno con sus reglas de protocolo. Se definirán el primer PoW, el PoS de Ethereum, el algoritmo Clique el cuál utilizaremos en este sistema y el IBFT 2.0.

2.1.4.1. PoW

En su artículo fundacional “Bitcoin: A Peer-to-Peer Electronic Cash System” (2008) [13], Satoshi Nakamoto propone un sistema de dinero en efectivo electrónico peer-to-peer basado en un sistema de prueba de trabajo (Proof-of-Work) inspirado en Hashcash de Adam Back. El núcleo de este mecanismo consiste en que cada nodo debe encontrar, para el encabezado de un bloque candidato, un valor (nonce) tal que, al aplicarle dos veces el algoritmo SHA-256, el resultado numérico sea menor o igual a un umbral (target) prefijado, lo que equivale a exigir que el hash comience con un cierto número de bits cero. Dado que SHA-256 produce salidas uniformemente distribuidas, la única forma de dar con ese nonce válido es mediante una búsqueda exhaustiva: el coste esperado crece exponencialmente con cada bit de dificultad añadido, mientras que la verificación de la solución sigue siendo trivial (un único doble hash) [13, 7].

Una vez hallada la combinación adecuada, el bloque —que incluye el hash del bloque previo, la raíz de Merkle de las transacciones seleccionadas, la marca de tiempo y el nonce— se difunde inmediatamente al resto de la red. Cada nodo receptor comprueba que todas las transacciones sean legítimas y no estén duplicadas, y que el hash válido cumpla el requisito de dificultad; si todo es correcto, acepta el bloque y comienza a trabajar sobre el siguiente, enlazándolo con el hash recién verificado. De esta forma, los nodos extienden la cadena que consideran válida en función del poder de cómputo empleado [13, 7].

Para garantizar que el tiempo medio de generación de bloques se mantenga cercano a diez minutos, Nakamoto introduce un ajuste dinámico de la dificultad cada 2 016 bloques. Este procedimiento calcula la relación entre el tiempo real transcurrido y el tiempo esperado; si los bloques se producen demasiado rápido, incrementa la exigencia de bits cero en el target, y si van demasiado lentos, la relaja. De este modo, la red se autorregula frente a variaciones en la potencia global de minado sin intervención externa [13].

La solidez de este esquema radica en que traslada la toma de decisiones desde identidades al poder de cómputo: “un CPU, un voto”. La historia aceptada es siempre la que incorpora la mayor cantidad de trabajo acumulado [13]. Para reescribir cualquier bloque ya enterrado bajo sucesores suficientes, un atacante debería rehacer no solo el PoW de ese bloque sino de todos los posteriores, y además alcanzar y superar el ritmo de producción del resto de la red. La probabilidad de éxito en tal empresa disminuye de manera exponencial a medida que crece la longitud de la cadena honesta.

2.1.4.2. PoS

El mecanismo de consenso basado en Proof-of-Stake (PoS), adoptado por Ethereum con la transición denominada “The Merge”, reemplaza el sistema de prueba de trabajo por un modelo en el que la seguridad de la red descansa en el compromiso económico de los participantes. En lugar de competir computacionalmente por la creación de bloques, los validadores son seleccionados de manera pseudoaleatoria para proponer y atestiguar bloques, y su influencia en el protocolo es proporcional a la cantidad de ETH que han depositado como garantía (*stake*) [24].

Cada validador debe bloquear al menos 32 ETH en el contrato de depósito para ingresar al conjunto activo de validadores. Una vez dentro, su rol principal consiste en participar en la producción y validación de bloques dentro de un sistema estructurado en *slots* y *epochs*. Cada *slot* (de 12 segundos) ofrece una oportunidad para que un

validador sea designado como proponente de bloque. Paralelamente, un subconjunto de validadores (comité) es seleccionado aleatoriamente para votar (atestiguar) sobre el bloque propuesto [25].

Para preservar la honestidad, el protocolo incluye mecanismos de incentivos y sanciones. Los validadores que actúan correctamente reciben recompensas proporcionales a su participación, mientras que aquellos que fallan en su deber de atestiguamiento, o que incurren en comportamientos maliciosos (como emitir votos contradictorios), son penalizados. En casos graves, pueden ser *slashed*, lo que implica la pérdida de parte o la totalidad de su stake y la expulsión del sistema [24].

A diferencia del PoW, el modelo de PoS es considerablemente más eficiente en términos computacionales y energéticos, ya que no requiere resolver problemas computacionales costosos. Asimismo, el modelo de finalización mediante supermayorías permite alcanzar niveles más altos de certeza sobre la inmutabilidad de bloques, reduciendo significativamente el riesgo de reorganizaciones profundas en la cadena.

Este enfoque permite que Ethereum alcance el consenso distribuido y la seguridad económica deseadas, no basándose en el gasto energético, sino en el compromiso financiero directo de sus participantes, lo que representa un avance significativo en sostenibilidad y escalabilidad para sistemas Blockchain de próxima generación.

Aunque no todo puede ser bueno. Debido a su lógica de stake se ha ganado muchos detractores, en el antiguo PoW cualquiera podía obtener monedas mientras que en PoS los que son ricos se vuelven más ricos; lo que ha llevado a que muchos grupos lo denominen un protocolo elitista [26].

2.1.4.3. Clique

Clique es un algoritmo de consenso basado en prueba de autoridad (Proof-of-Authority, PoA), diseñado para su uso en redes Ethereum privadas o redes de prueba donde los participantes son entidades previamente identificadas. A diferencia de los mecanismos como Proof-of-Work o Proof-of-Stake, Clique no depende de la capacidad computacional o el capital económico de los participantes, sino de un conjunto predefinido de validadores denominados signers. Estos signers están autorizados para proponer y firmar bloques. El protocolo opera bajo un esquema de turnos semi-aleatorio en el que los signers se rotan la propuesta de bloques en intervalos regulares. Cada bloque contiene la firma criptográfica del signer correspondiente, que se registra en el campo `extraData` del encabezado, el cual incluye un área de 32 bytes para datos

arbitrarios, una lista de signers autorizados y la firma ECDSA del bloque [27].

La producción de bloques está sujeta a una política de limitación que impide que un signer cree bloques consecutivos más allá de un umbral determinado (`SIGNER_LIMIT`), con el objetivo de evitar la centralización o manipulación del flujo de bloques. Además, la dificultad del bloque se ajusta dependiendo de si fue generado por el signer asignado para ese turno o por otro signer autorizado fuera de turno. Los bloques en turno tienen una dificultad de 2, mientras que los bloques fuera de turno tienen dificultad 1, lo cual permite a los clientes distinguir entre bloques preferidos y secundarios, facilitando así la convergencia hacia una cadena canónica [27].

La gestión del conjunto de signers es dinámica y se lleva a cabo mediante un sistema de votación on-chain. Cada signer puede emitir un voto para agregar o eliminar otro signer. Estas decisiones se efectúan solo en bloques denominados epochs, que ocurren a intervalos regulares definidos por el parámetro `EPOCH_LENGTH`. Para que un voto tenga efecto, se requiere una mayoría absoluta del conjunto de signers activos. Este mecanismo garantiza que los cambios en la autoridad sean verificables, transparentes y consistentes con el consenso de la red [27].

Si bien no está diseñado para sistemas públicos permissionless, su arquitectura resulta ideal para blockchains privadas, consorciadas o experimentales. No es recomendable su uso en producción debido a que puede crear diversificaciones de la cadena cuando hay mucho flujo de transacciones en la red y tiene menos tolerancia a fallos que otros algoritmos [27].

NOTE

El algoritmo Clique ha sido el elegido para el sistema de este TFG.

2.1.4.4. IBFT 2.0

IBFT 2.0 (Istanbul Byzantine Fault Tolerance 2.0) es, también, un protocolo de consenso basado en Proof-of-Authority (PoA) para blockchains privadas o consorciadas, diseñado para ofrecer tolerancia a fallos bizantinos y finalización inmediata de bloques. Utiliza un conjunto predefinido de validadores autorizados, denominados signers (similar al algoritmo Clique), responsables de proponer y validar bloques. Este enfoque garantiza un consenso eficiente y de baja latencia, ideal para entornos controlados con participantes confiables [28].

El protocolo opera bajo un modelo de comunicación asincrónica, donde los nodos

pueden operar de manera independiente sin necesidad de sincronización estricta. Sin embargo, para garantizar la seguridad y la liveness del sistema, se asume que la red eventualmente se vuelve síncrona, es decir, que existe un límite máximo para la latencia de los mensajes. En este contexto, IBFT 2.0 asegura que, incluso en presencia de fallos bizantinos, el sistema puede alcanzar consenso y mantener la integridad de la cadena de bloques [29].

En IBFT 2.0, la producción de bloques se organiza en rondas, donde en cada ronda un validador es seleccionado para proponer un bloque. La selección del proponente se realiza de manera determinista, basada en el historial de bloques y el número de ronda, utilizando una función de selección de proponentes. Una vez que un bloque es propuesto, se sigue un proceso de consenso en tres fases: *Prepare*, *Commit* y *Round Change*. En la fase *Prepare*, el proponente envía un mensaje a los demás validadores indicando su intención de proponer un bloque. Los validadores que estén de acuerdo envían mensajes de preparación. En la fase *Commit*, los validadores envían mensajes de confirmación para indicar que están listos para añadir el bloque a la cadena. Finalmente, en la fase *Round Change*, si se detecta que no se ha alcanzado consenso en la ronda actual, se inicia una nueva ronda con un nuevo proponente [28].

Una característica destacada de IBFT 2.0 es su capacidad para manejar un conjunto dinámico de validadores. Los cambios en el conjunto de validadores se gestionan mediante un mecanismo de votación en cadena, donde los validadores pueden proponer la adición o eliminación de otros validadores. Estos cambios se implementan de manera segura y transparente, garantizando que todos los nodos de la red estén de acuerdo con el nuevo conjunto de validadores antes de que entre en vigor [29].

Además, IBFT 2.0 asegura la finalización inmediata de los bloques, lo que significa que una vez que un bloque es añadido a la cadena, no puede ser revertido sin causar una bifurcación en la red. Esto proporciona una mayor seguridad y previsibilidad en comparación con otros mecanismos de consenso que dependen de probabilidades para la finalización de bloques [29].

Ofrece un protocolo de consenso robusto y eficiente para redes Blockchain privadas o consorciadas, combinando tolerancia a fallos bizantinos, finalización inmediata de bloques y la flexibilidad de un conjunto dinámico de validadores. Su diseño lo hace adecuado para aplicaciones empresariales y escenarios donde la confianza entre los participantes puede ser preestablecida y mantenida [28].

NOTE

En la documentación de Hyperledger Besu se recomienda usar en soluciones en fase de producción [29], unas de las posibles mejoras de este sistema es cambiar el algoritmo de consenso de Clique a IBFT 2.0 en fases más avanzadas del sistema.

2.1.5. ¿Qúe es un cliente Blockchain? Tipos de clientes en el ecosistema Ethereum

Un cliente *Blockchain* es una implementación de software que permite a los usuarios interactuar con la red [30]. Este cliente ejecuta las reglas del protocolo, valida bloques y transacciones, puede mantener una copia del estado actual de la cadena y comunica los datos relevantes a otros nodos. En otras palabras, un cliente es el motor que permite a una computadora convertirse en un nodo de la red.

Dependiendo del ecosistema o arquitectura de la red, existirán distintos tipos de clientes. En Ethereum, existen tres tipos de clientes, la elección entre uno u otro debe adaptarse a las necesidades del usuario:

- **Clientes completos (full clients)**: descargan y verifican toda la cadena de bloques, ejecutan todas las transacciones y almacenan el estado completo del sistema [21]. Ejemplos incluyen `Geth` [31] y `Besu`.
- **Clientes ligeros (light clients)**: descargan solo los encabezados de bloque y confían en nodos completos para consultar información. Son útiles para dispositivos con recursos limitados [21].
- **Clientes de archivo (archive nodes)**: almacenan todos los estados históricos además del estado actual, lo que permite realizar consultas históricas en profundidad, aunque requieren grandes cantidades de almacenamiento [32].

Ethereum también distingue entre clientes de ejecución y clientes de consenso desde la transición a Ethereum 2.0:

- **Cliente de ejecución**: maneja la ejecución de transacciones, el estado de los contratos inteligentes y la máquina virtual de Ethereum (EVM) [33]. Ejemplos: `Geth`, `Besu`.

- **Cliente de consenso**: implementa el protocolo de consenso (actualmente *Proof of Stake*), valida bloques y firma con los validadores [33]. Ejemplo: `Teku` [34].

Ambos tipos de cliente deben estar sincronizados y conectados mediante una interfaz estándar (Engine API) [35] para mantener el funcionamiento adecuado de un nodo de Ethereum post-Merge.

2.1.6. Tipos de Redes *Blockchain*

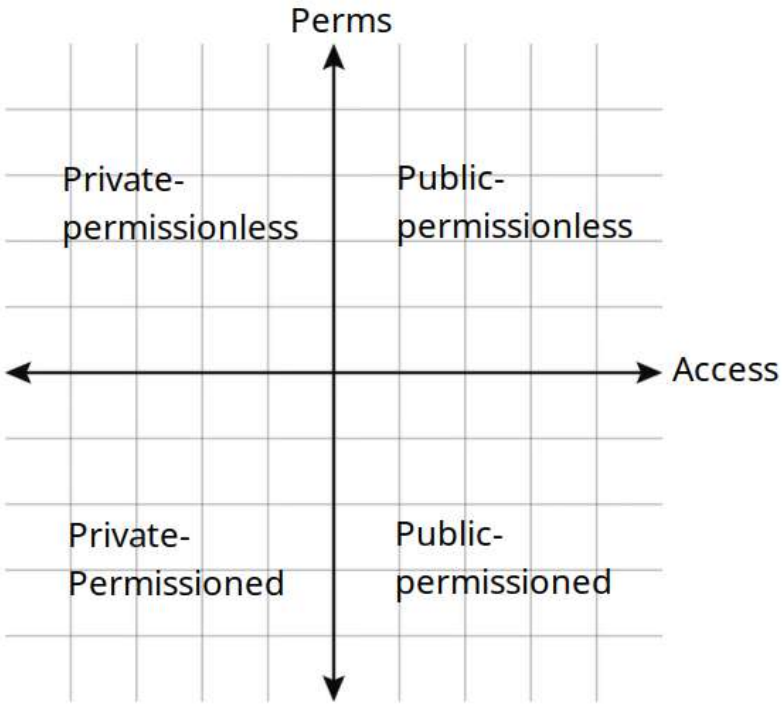

Figura 2.14: Dos ejes que dividen los tipos de redes.

Las redes *Blockchain* se pueden clasificar en función de dos ejes fundamentales: el nivel de **acceso** (quién puede participar en la red) y el nivel de **permiso** (quién puede escribir o validar transacciones) como puede apreciarse en la figura 2.14. Esta clasificación es crucial para adaptar la tecnología *Blockchain* a distintas necesidades y casos de uso [3].

- Moverse hacia redes **más abiertas** (acceso público) aumenta la transparencia, la descentralización y la resistencia a la censura, pero sacrifica rendimiento y privacidad.

- Moverse hacia redes **más restringidas** (permisos controlados) mejora la eficiencia, la escalabilidad y la privacidad, pero reduce la apertura, confianza distribuida y resistencia a fallos.

Este modelo da lugar a cuatro tipos principales de redes:

A) *Public-Permissionless*

Las redes *Public-Permissionless* son aquellas en las que cualquier persona puede participar y enviar transacciones. No existen restricciones para unirse a la red, operar un nodo o interactuar con ella, lo que las hace completamente abiertas y descentralizadas. Estas redes suelen emplear mecanismos de consenso como *proof-of-work* o *proof-of-stake*, ofreciendo incentivos económicos a los validadores. Este tipo de red es ideal para aplicaciones donde la transparencia y la descentralización son prioritarias, como las criptomonedas o las plataformas de contratos inteligentes abiertas [3].

B) *Private-Permissionless*

En las redes *Private-Permissionless*, la escritura está abierta a todos los miembros de la red privada, pero no todo el mundo puede unirse a la red, esta suele estar limitada a un número de participantes conocidos. Esto sugiere un modelo de consorcio, donde, por ejemplo, un grupo de bancos utiliza una *Blockchain* para transferencias internas: cualquier banco miembro puede validar transacciones, pero para ser un banco miembro debes pedir permisos para entrar [3].

C) *Public-Permissioned*

Las redes *Public-Permissioned* permiten que cualquier persona participe en la red, pero restringen el envío de transacciones a entidades específicas autorizadas. Este tipo de red combina transparencia con control, lo que la hace adecuada para escenarios donde los datos deben ser accesibles al público, pero la validación requiere regulación o confianza [3].

D) *Private-Permissioned*

Finalmente, las redes *Private-Permissioned* restringen tanto el acceso como creación de transacciones a entidades autorizadas. Este tipo de red es completamente controlado y se utiliza típicamente en entornos empresariales donde la privacidad y el control son fundamentales. La validación y el acceso a los datos están limitados

a nodos o usuarios autorizados, lo que garantiza la confidencialidad [3]. Este modelo es ideal para sistemas donde la confidencialidad es prioritaria, como en el sector logístico.

2.2. *Internet of Things (IoT)*

La *Internet de las Cosas* (*IoT*, por sus siglas en inglés) es un paradigma tecnológico que ha emergido con fuerza en las últimas décadas, impulsado por los avances en microelectrónica, comunicaciones inalámbricas y computación ubicua. El concepto hace referencia a la extensión de la conectividad de Internet a objetos físicos cotidianos, que adquieren así la capacidad de percibir su entorno, comunicarse entre sí y con sistemas remotos, y en algunos casos tomar decisiones autónomas.

2.2.1. ¿Qué es IoT?

De acuerdo con Serpanos y Wolf [36], IoT puede definirse como un sistema compuesto por dispositivos embebidos (sensores, actuadores, microcontroladores) conectados a redes de comunicación, que cooperan para recopilar, intercambiar y procesar información del mundo físico con fines de monitorización, control o análisis.

Un sistema IoT típico está conformado por los siguientes elementos básicos:

- **Sensores y actuadores:** los sensores permiten observar el entorno mediante la medición de magnitudes físicas (temperatura, presión, humedad, luz, etc.) como el de la figura 2.15, mientras que los actuadores ejecutan acciones sobre el entorno (abrir una válvula, activar una alarma, enviar una transacción, etc.).

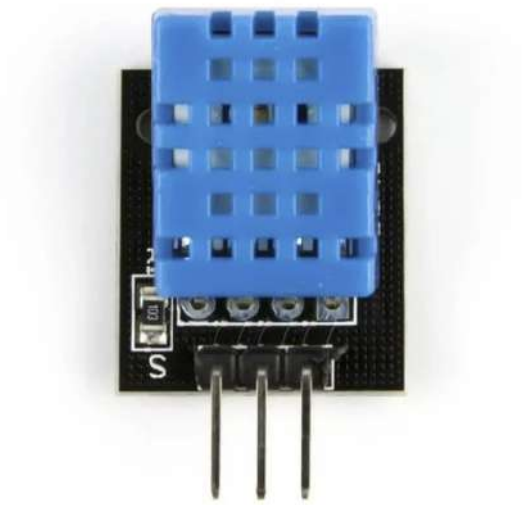

Figura 2.15: Dispositivo DHT11, actuador de temperatura y humedad.

- **Dispositivos de procesamiento:** son microcontroladores o sistemas embebidos (como Arduino, Raspberry Pi, ESP32) que reciben las señales del sensor, las transforman en datos digitales, y pueden realizar operaciones locales como filtrado, agregación o decisiones lógicas.
- **Módulos de comunicación:** permiten la transmisión de datos hacia otros dispositivos o servidores a través de redes alámbricas o inalámbricas. Las tecnologías más comunes incluyen Wi-Fi, Bluetooth, etc., cada una con diferentes alcances, velocidades y consumos energéticos.
- **Infraestructura de red:** abarca desde gateways locales hasta redes de área amplia y conexiones con plataformas en la nube, que permiten el almacenamiento, análisis y visualización de los datos recopilados.
- **Software de integración:** sistemas que permiten el manejo de dispositivos, la interoperabilidad entre ellos, y la exposición de servicios a otras aplicaciones mediante APIs o protocolos estándar.

2.2.2. Arquitectura de referencia

El modelo de referencia para IoT es el propuesto por la ITU (ITU-T Y.2060), que describe una arquitectura de cuatro capas jerárquicas para la industria como puede apreciarse en la figura 2.16:

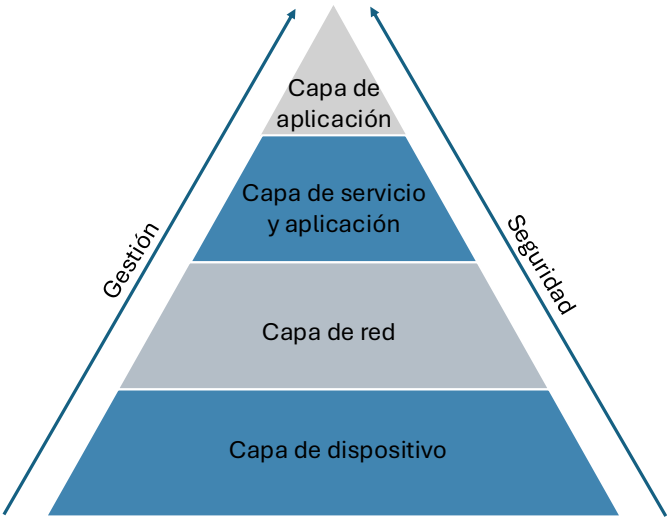

Figura 2.16: Arquitectura propuesta por ITU.

1. **Capa de dispositivo:** incluye sensores, actuadores y dispositivos embebidos que interactúan con el entorno físico, capturando datos o ejecutando acciones.
2. **Capa de red:** se encarga de la conectividad, transmitiendo datos entre dispositivos y sistemas de procesamiento mediante tecnologías alámbricas o inalámbricas.
3. **Capa de soporte de servicio y aplicación:** proporciona funciones como almacenamiento, procesamiento de datos y soporte para aplicaciones específicas.
4. **Capa de aplicación:** abarca las plataformas y servicios que utilizan los datos para generar valor, como monitorización, automatización o análisis.

Además, el modelo ITU incluye dos capas transversales para gestión (configuración, monitoreo, mantenimiento) y seguridad (autenticación, cifrado, protección de datos). Para el IoT Industrial (IIoT), se describe la arquitectura de la Industrial Internet Consortium (IIC), que se basa en cinco dominios funcionales: control, operaciones, información, aplicación y negocio, con funciones transversales como conectividad y análisis [36].

2.2.3. Características clave de los sistemas IoT

Los sistemas IoT (Internet de las Cosas) se caracterizan por una serie de rasgos distintivos que los diferencian de otros tipos de infraestructuras tecnológicas. En primer lugar, se diseñan con un propósito específico, lo que significa que están orientados a

resolver tareas concretas en lugar de constituir redes generales de dispositivos. Además, integran de forma estrecha elementos físicos, como sensores y actuadores, con capacidades de procesamiento computacional, permitiéndoles interactuar directamente con el entorno [36].

Otra característica fundamental es su bajo consumo energético, un requisito esencial para garantizar la viabilidad de muchos dispositivos, especialmente aquellos que funcionan con baterías o fuentes de energía recolectada del ambiente. Asimismo, estos sistemas suelen operar de manera basada en eventos, reaccionando a estímulos del entorno en lugar de realizar muestreos periódicos [36].

En cuanto a los componentes, los nodos que conforman las redes IoT suelen ser dispositivos simples, de bajo costo y con capacidades computacionales limitadas, lo que facilita su implementación masiva. La conectividad, por su parte, se basa mayoritariamente en redes inalámbricas, lo que permite una instalación más flexible y menos invasiva. Finalmente, debido a su carácter distribuido y a la interacción con el mundo físico, los sistemas IoT deben cumplir estrictos requisitos de seguridad y protección de datos para garantizar su confiabilidad y privacidad [36].

Capítulo 3

TRAZABILIDAD ALIMENTARIA

A lo largo de los años, garantizar el seguimiento de los productos alimentarios ha sido un desafío constante. Este capítulo se centra en cómo funciona la trazabilidad alimentaria, qué métodos se han utilizado históricamente y cómo la irrupción de *Blockchain* propone una nueva manera de entender y gestionar este proceso.

Como explican McEntire y Kennedy, la trazabilidad debe entenderse como una arquitectura de información distribuida que conecta datos, decisiones y personas a lo largo de toda la cadena alimentaria [37].

3.1. ¿Qué es la trazabilidad alimentaria?

Según la definición establecida en el Reglamento (CE) Nº 178/2002 y recogida por la Agencia Española de Seguridad Alimentaria y Nutrición (AESAN), “la trazabilidad es la posibilidad de encontrar y seguir el rastro, a través de todas las etapas de producción, transformación y distribución, de un alimento, un pienso, un animal productor de alimentos o una sustancia destinada a ser incorporada en alimentos o piensos o con probabilidad de serlo” [11].

Un sistema de trazabilidad eficaz es clave/necesario para registrar e interconectar información esencial, como el origen de las materias primas y sus proveedores, los procesos de producción y transformación realizados, la identificación de lotes y fechas relevantes, así como la distribución de los productos terminados y sus destinatarios [11].

Esta trazabilidad puede ser hacia atrás (identificando el origen de las materias

primas), interna (seguimiento dentro de una instalación) o hacia delante (conociendo el destino final de los productos) como puede apreciarse en la figura 3.1.

Figura 3.1: Tipos de trazabilidad.

La trazabilidad alimentaria, más allá de ser una obligación legal, se convierte en una herramienta estratégica para gestionar riesgos y garantizar la calidad. Su correcta aplicación permite detectar y retirar rápidamente del mercado productos que puedan representar un peligro, minimizar el impacto de incidentes tanto en la salud pública como en la reputación de las empresas involucradas, facilitar las tareas de inspección y control por parte de las autoridades, y fortalecer la confianza de los consumidores mediante la transparencia y el cumplimiento de la normativa.

La trazabilidad debe entenderse como un “sistema integral” y no como una obligación documental aislada. Involucra a todos los operadores de la cadena alimentaria, exige planificación, coordinación y el uso de herramientas que aseguren la coherencia y accesibilidad de la información en tiempo y forma [11]. La trazabilidad eficaz es, por tanto, una responsabilidad compartida entre productores, transformadores, distribuidores y autoridades, y constituye un eje fundamental de cualquier política moderna de seguridad alimentaria.

3.2. Soluciones clásicas

Tradicionalmente, los sistemas de trazabilidad alimentaria se han basado en estructuras descentralizadas pero no interoperables, donde cada operador mantiene su propio sistema de registro, ya sea manual o informatizado. Aunque estas soluciones han permitido cumplir con los requisitos legales mínimos, presentan importantes limitaciones en términos de eficiencia, transparencia y capacidad de respuesta ante incidentes. A continuación, se detallan las principales soluciones tradicionales, tal como se describen en la guía técnica de la AESAN [11].

3.2.1. Etiquetado físico y documentación en papel

Una de las formas más básicas de trazabilidad consiste en el etiquetado de los productos y el uso de documentación física. Cada operador imprime etiquetas que incluyen información clave como el número de lote, la fecha de producción, el origen de las materias primas y otra información relevante. Esta documentación suele complementarse con albaranes, guías de transporte y fichas técnicas que acompañan al producto a lo largo de la cadena.

Este enfoque, si bien cumple con la exigencia normativa de trazabilidad “una etapa antes y una después”, presenta varios inconvenientes:

- Requiere una elevada carga de trabajo manual.
- Es propenso a errores humanos en la transcripción de datos.
- Los documentos pueden deteriorarse, extraviarse o ser falsificados.
- La información queda aislada en cada empresa, dificultando el seguimiento completo del producto.

Como consecuencia, en situaciones que requieren una rápida retirada de productos o una auditoría exhaustiva, estos sistemas resultan ineficientes y poco fiables.

3.2.2. Identificadores únicos y tecnologías de codificación

Para asociar productos con su información de trazabilidad, se utilizan identificadores únicos. Estos pueden representarse mediante distintas tecnologías, entre ellas:

- **Etiquetas *RFID* (Radio Frequency Identification)** [38]: permiten la identificación sin contacto y a distancia, lo que resulta útil en operaciones logísticas automatizadas.
- **Códigos de barras**: lineales o bidimensionales (por ejemplo, QR), escaneables mediante dispositivos de lectura estándar como las mostradas en la figura 3.2.

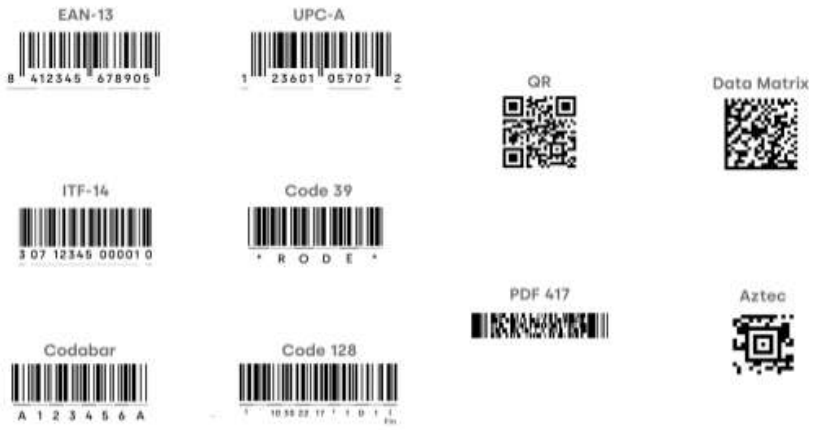

Figura 3.2: Identificadores únicos y tecnologías de codificación.

- **Números de lote**: impresos en los envases, permiten agrupar productos fabricados en una misma tanda de producción.

Estas herramientas mejoran la captura de datos, reducen errores de lectura y aceleran procesos, pero su eficacia depende del sistema informático subyacente. Como apunta la guía de AESAN, si los datos no son correctos, accesibles y verificables entre operadores, el identificador por sí solo no garantiza la trazabilidad real [11].

3.2.3. Sistemas informáticos centralizados

Con el objetivo de mejorar la eficiencia y fiabilidad de los registros frente a los métodos manuales, muchas empresas han implementado soluciones informatizadas como sistemas ERP (Enterprise Resource Planning) [39], SCADA (Supervisory Control and Data Acquisition) [40] o plataformas específicas de trazabilidad. Estas herramientas permiten digitalizar los flujos de información, automatizar tareas rutinarias (como el registro de lotes o la generación de etiquetas) y facilitar búsquedas rápidas dentro de los sistemas internos.

No obstante, estas soluciones suelen operar de manera centralizada y aislada dentro de cada organización, sin conexión directa con los sistemas de proveedores o clien-

tes. Como indica la guía de la AESAN, “los sistemas informáticos, aunque eficaces para uso interno, no garantizan la interoperabilidad con otros operadores sin acuerdos previos y formatos estandarizados” [11]. Esto significa que, aunque cada empresa puede tener control sobre su parte del proceso, la trazabilidad a nivel de cadena se fragmenta en compartimentos estancos.

En la práctica, esta falta de interoperabilidad genera cuellos de botella en situaciones críticas, como retiradas de productos, donde es necesario reconstruir con rapidez el recorrido de un lote desde el origen hasta el punto de venta. Si no existe un protocolo común o integración entre sistemas, estas tareas requieren llamadas, correos electrónicos y validaciones manuales entre actores, lo que ralentiza la respuesta y aumenta el riesgo de error.

Además, el diseño centralizado implica que cada empresa es responsable exclusiva de la integridad y seguridad de sus datos, sin mecanismos externos de verificación. Esto puede dificultar auditorías o investigaciones retrospectivas, especialmente si los registros han sido modificados, eliminados o no están disponibles en el momento necesario.

Como concluyen McEntire y Kennedy, “la digitalización ha mejorado la trazabilidad interna, pero no resuelve el mayor reto: la trazabilidad a nivel de red” [37]. En otras palabras, los sistemas informáticos actuales gestionan bien los datos que una empresa genera, pero no los que necesita de terceros. Este “punto ciego” en la visibilidad interempresarial es uno de los principales catalizadores para explorar soluciones como *Blockchain*.

3.2.4. Obligaciones documentales y registros mínimos

La legislación vigente impone la obligación de registrar y conservar ciertos datos básicos sobre el origen y destino de los productos. Estos incluyen la información ilustrada en la figura 3.3:

- Datos del proveedor inmediato: identificación, productos entregados, fechas, número de lote.
- Información interna: procesos aplicados, lotes mezclados, fechas de transformación o envasado, condiciones de almacenamiento.
- Datos del cliente inmediato: destinatario, fecha de salida, número de lote enviado.

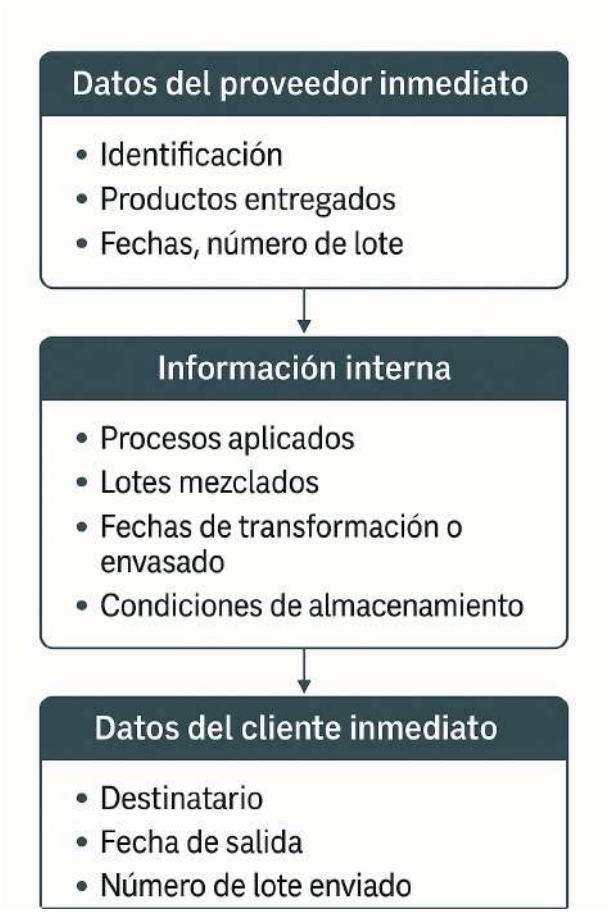

Figura 3.3: Datos obligatorios.

Según la AESAN, “la trazabilidad interna es clave para comprender qué ocurre dentro de la empresa y para reconstruir el flujo del producto si se detecta un problema” [11]. La duración de conservación de los registros depende del tipo de producto y sus riesgos asociados, pero en general se exige que estén disponibles durante un periodo razonable que garantice la protección del consumidor.

3.2.5. Evolución de la trazabilidad: de obligación legal a herramienta estratégica

Aunque inicialmente la trazabilidad se concibió como una exigencia legal tras diversas crisis alimentarias como la mencionada en la sección 1.1, en las últimas décadas su rol ha evolucionado sustancialmente. La trazabilidad ha pasado de ser un mecanismo puramente reactivo a una herramienta estratégica de gestión de riesgos, diferenciación de marca y mejora operativa [37].

En este contexto, las empresas líderes ya no se conforman con cumplir la traza-

bilidad “una etapa antes y una después”, sino que buscan visibilidad total, desde la producción primaria hasta el consumidor final. Esta trazabilidad extendida permite:

- Anticipar y gestionar de forma proactiva los riesgos emergentes (fraude, sostenibilidad, presencia de alérgenos).
- Comunicar atributos de valor añadido (origen local, prácticas sostenibles, cultivo ecológico).
- Optimizar la gestión de inventarios, reducir mermas y fortalecer el control de calidad.

Las empresas más avanzadas ya integran la trazabilidad en sus sistemas de toma de decisiones y comunicación con clientes, especialmente en sectores como la alimentación orgánica o la exportación a mercados con altos estándares regulatorios [37].

3.2.6. Limitaciones estructurales de las soluciones tradicionales

En conjunto, las soluciones clásicas presentan una serie de limitaciones estructurales que condicionan su eficacia:

- **Fragmentación de datos**: la información se encuentra dispersa entre distintos operadores, sin integración automatizada.
- **Falta de transparencia**: cada agente solo tiene visibilidad de su tramo, lo que dificulta auditorías rápidas y completas.
- **Posibilidad de manipulación**: la ausencia de registros inmutables permite la alteración o eliminación de información.
- **Lentitud en la respuesta**: ante una alerta alimentaria, rastrear el origen o destino de un producto puede requerir días o semanas de consultas manuales.

Estas limitaciones explican por qué muchas organizaciones están explorando tecnologías emergentes como *Blockchain*, que permiten crear redes de trazabilidad descentralizadas, verificables y en tiempo real. En la siguiente figura 3.1 puede apreciarse como podría mejorar Blockchain el panorama con respecto a las soluciones tradicionales.

Dimensión	Soluciones tradicionales	*Blockchain*
Inmutabilidad	Los registros pueden ser modificados o eliminados; dependen de los controles internos de cada empresa.	Los datos no pueden alterarse una vez registrados.
Transparencia y confianza compartida	La información está fragmentada en bases de datos aisladas; se requiere confianza en cada parte.	Todos los actores autorizados acceden a un libro mayor común y sincronizado, lo que reduce la necesidad de confianza ciega.
Automatización	Los procesos suelen ser manuales y susceptibles a errores o demoras.	Los contratos inteligentes automatizan condiciones y procesos .
Auditabilidad en tiempo real	Auditorías complejas y lentas que imposibilita ser en tiempo real.	Historial completo y verificable en tiempo real.

Tabla. 3.1: Comparativa entre soluciones tradicionales y tecnología *Blockchain*.

A continuación, se profundiza qué haría esta integración.

3.3. Integración de *Blockchain* en procesos de Trazabilidad

La incorporación de *Blockchain* está revolucionando el concepto y proceso de la trazabilidad alimentaria, como se muestra en la Figura 3.4 conecta a todos los actores.. A través de las características que se enseñaron en el capítulo 2. Esta arquitectura aporta varias mejoras clave frente al modelo clásico como se detalla en [10]:

- **Inmutabilidad de los datos**: Una vez que un evento de trazabilidad (por ejemplo, “lote X fue enviado del proveedor A al fabricante B el día Y”) se registra en la *Blockchain*, no puede ser alterado ni borrado. Esto garantiza la integridad de la información; subsubsección 2.1.2.2. La seguridad de los datos pasa de depender de los controles internos de una empresa a depender de la robustez criptográfica y el consenso de la red distribuida; subsecciones 2.1.2.4 y 2.1.3.7.
- **Transparencia y confianza compartida**: Todos los actores autorizados en la cadena de suministro pueden acceder (según políticas definidas) al registro co-

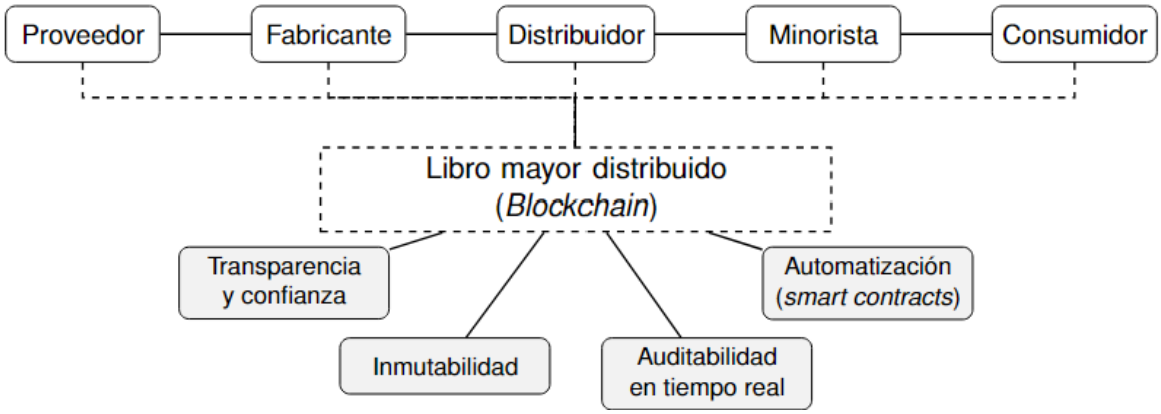

Figura 3.4: Integración de Blockchain y trazabilidad.

mún para verificar el historial de un producto. Ya no se requieren múltiples bases de datos aisladas, sino un libro mayor compartido y sincronizado en tiempo real. Esto incrementa la confianza entre las partes: cada organización puede auditar directamente la información relevante aportada por otras, sin depender solo de su palabra.

- **Automatización mediante *smart contracts***: en trazabilidad, los *smart contracts* pueden automatizar procesos y validar condiciones de forma transparente; subsubsección 2.1.3.9. Por ejemplo, pueden desencadenar acciones como pagos automáticos al cumplirse condiciones (por ejemplo, liberar el pago al transportista solo cuando el contrato verifica en la *Blockchain* que la entrega se realizó en condiciones adecuadas). Esta automatización reduce la intervención manual y el riesgo de errores, a la vez que garantiza que las reglas definidas se cumplan estrictamente.

- **Auditabilidad en tiempo real**: Todas las partes interesadas (productores, reguladores, distribuidores, consumidores autorizados) pueden auditar el historial de un producto prácticamente en tiempo real en la *Blockchain*. La información registrada es rastreable de forma retroactiva completa, creando un rastro verificable de cada evento (producción, procesamiento, transporte, venta). Esto simplifica enormemente las auditorías y el cumplimiento normativo, puesto que todos los participantes poseen todos los datos como se exploró en la subsubsección 2.1.3.1. De cara al consumidor final, se pueden habilitar aplicaciones (por ejemplo, escaneando un código QR en el envase) que muestren el recorrido del alimento “del campo a la mesa” con todos sus hitos verificados en *Blockchain*.

Como se aprecia, el modelo *Blockchain* aporta mejoras sustanciales en seguridad, confianza y eficiencia. No obstante, también conlleva desafíos: su implementación requiere coordinación entre todos los actores de la cadena, estándares comunes y a

veces un cambio cultural para compartir datos de forma transparente [10]. Además, surgen cuestiones de escalabilidad técnica y consumo energético a considerar. Pese a ello, la tendencia en la industria alimentaria es clara: numerosas iniciativas están evolucionando sus sistemas de trazabilidad tradicionales hacia soluciones soportadas por *Blockchain*, a menudo complementadas con otras tecnologías como *IoT*.

Capítulo 4

TECNOLOGÍAS DEL SISTEMA: CLIENTE *BLOCKCHAIN* Y DISPOSITIVOS *IOT*

Este capítulo presenta la tecnologías del sistema de trazabilidad alimentaria basada en tecnologías IoT y blockchain. En primer lugar, se describe el cliente blockchain seleccionado, Hyperledger Besu, detallando las razones de su elección, sus principales características técnicas y su papel dentro del sistema. A continuación, se introducen los dispositivos IoT empleados, especificando los sensores utilizados para la recolección de datos clave (como temperatura, humedad, geolocalización y presencia) y sus condiciones de integración técnica.

4.1. Hyperledger Besu

Se empleará Hyperledger Besu (figura 4.1) como cliente *Blockchain*, cumpliendo un rol fundamental en el sistema. El uso de clientes y software previamente desarrollados acelera la implementación y garantiza la fiabilidad, al tratarse de soluciones ya probadas y seguras. Además, habitualmente, poseen una documentación bien organizada y muy completa [41].

Figura 4.1: Hyperledger Besu.

4.1.1. Motivación y Visión general

La elección de Hyperledger Besu como núcleo del sistema se justifica por su compatibilidad con redes *permissioned* (clave para entornos empresariales), su soporte para altos volúmenes de transacciones, ser un software compatible con el ecosistema Ethereum y su flexibilidad para integrarse con sensores IoT.

Hyperledger Besu es un cliente de Ethereum *open source*, escrito en Java y licenciado bajo Apache 2.0. Su diseño modular y altamente configurable está pensado para entornos empresariales. Proporciona una arquitectura de *plugins* que permite extender o personalizar su funcionalidad sin modificar el núcleo [41]. Es compatible con la arquitectura Ethereum, lo que conlleva una serie de características que habilitan su uso como cliente para redes Ethereum.

4.1.1.1. Compatible con la arquitectura Ethereum

Hyperledger Besu cumple con la especificación de cliente de la *Enterprise Ethereum Alliance* (EEA), implementando todos los componentes esenciales de la plataforma Ethereum:

- **EVM (Ethereum Virtual Machine):** el motor de ejecución que procesa los contratos inteligentes. Besu ejecuta el bytecode de Ethereum de la misma forma que lo haría un nodo de la red pública, garantizando que los smart contracts desarrollados (por ejemplo, en Solidity) funcionen de igual manera en una red Besu privada.
- **Almacenamiento de estado y Blockchain:** Besu mantiene las estructuras de datos de Ethereum, incluyendo el estado mundial (cuentas y balances, almacenamiento de contratos) y la cadena de bloques con todas las transacciones

históricas. Utiliza las mismas estructuras de merkle tries y bases de datos (por defecto RocksDB con esquema Bonsai o Forest para almacenamiento eficiente) que otros clientes Ethereum. El sistema desarrollado utiliza Forest por temas que se comentarán en el diseño [42].

- **Red peer-to-peer Ethereum (devp2p):** Besu implementa el protocolo de red de Ethereum, RLPx, el subprotocolo eth de descubrimiento de nodos y gossip de bloques/transacciones. Esto le permite conectarse y comunicarse con otros nodos Ethereum.

- **API de Ethereum (JSON-RPC/WS):** Besu expone las interfaces de programación estándar de Ethereum (llamadas RPC). Incluye métodos como eth_getBalance, eth_sendTransaction, eth_call, etc., que permiten a aplicaciones externas (dApps, scripts, sistemas de información) interactuar con la Blockchain. De esta forma, desarrolladores pueden usar bibliotecas familiares (web3.js, web3j, ethers.js, etc.) para conectarse a un nodo Besu igual que lo harían con un nodo Geth o Infura (otros clientes distintos como e Besu) en Ethereum público [43].

- **Cuentas y criptografía:** El cliente soporta las claves y direcciones Ethereum (formato EOA), firma de transacciones con ECDSA (secp256k1) u otros algoritmos compatibles, cómputo de hashes (Keccak-256) y en general todos los primitivas criptográficos que conforman la base de la seguridad de Ethereum.

- **Eventos y logs:** Permite el uso de logs de Ethereum (eventos generados por contratos inteligentes) de modo que las aplicaciones pueden suscribirse a eventos de trazabilidad [44].

- **Configurabilidad empresarial:** Al estar alineado con la EEA, Besu agrega ciertas extensiones necesarias en entornos corporativos. Esto se comentará en secciones posteriores.

Besu hereda todo el ecosistema Ethereum pero permite redes independientes y personalizadas mediante un chain ID distinto y un bloque génesis configurable.

- **Chain ID distinto:** se trata de un identificador único para cada red Blockchain. Su función principal es evitar que una transacción válida en una red específica (como una red privada) pueda ser aceptada en otra red (como Ethereum pública), incrementando así la seguridad y aislamiento entre redes [45].

- **Bloque génesis configurable:** es el bloque inicial de la cadena, donde se establecen los parámetros fundamentales de la red. Entre ellos se encuentran el

algoritmo de consenso utilizado, el coste del gas para cada operación, las cuentas predefinidas con saldo inicial y otras condiciones esenciales que determinan el comportamiento de la red desde su inicio [46].

Esta capacidad de personalización permite adaptar la red Blockchain a los requisitos específicos del proyecto, sin perder compatibilidad con las herramientas, contratos y estándares del ecosistema Ethereum.

4.1.2. Características de Hyperledger Besu

En esta subsección se explican las principales características de Hyperledger Besu como pueden ser los algoritmos de consenso disponibles, las APIs, etc.

4.1.2.1. Algoritmos de consenso disponibles

El algoritmo a utilizar se indica y configura en el bloque génesis. Besu soporta varios mecanismos de consenso [47]:

- **Proof of Work (PoW):** Implementa el protocolo de consenso original de Ethereum (Ethash) utilizado en la red principal hasta 2022. Besu podía operar en redes PoW de Ethereum tradicional, aunque Ethereum migró luego a PoS. Se menciona porque fue de los primeros; pero en la actualidad está DEPRECATED [48].
- **Proof of Stake (PoS):**Besu maneja la ejecución de transacciones y el estado, mientras que la validación de bloques PoS la llevan a cabo otros nodos de consenso. Besu no realiza staking ni validación PoS por sí mismo, sino que se integra bajo este nuevo esquema [49]. Se necesitaría usar clientes adicionales como Teku [50].
- **Proof of Authority (PoA):** Orientada a redes privadas o consorciadas donde los nodos validadores son conocidos. Besu soporta varios algoritmos BFT de PoA, entre ellos:
  - **Clique:** algoritmo PoA basado en sellos de tiempo donde un grupo de validadores produce bloques en ronda [51].
  - **IBFT 2.0:** versión mejorada de IBFT con mayor tolerancia a fallos bizantinos y finalización más segura de cada bloque [29].

- **QBFT:** Quorum BFT de próxima generación, introducido más recientemente, que optimiza el rendimiento y seguridad en redes de consorcio. Este algoritmo reduce la probabilidad de bifurcaciones y está diseñado para casos de uso empresariales exigentes [52].

Gracias a este soporte, una red puede elegir el mecanismo de consenso que mejor se adapte a sus necesidades. Por ejemplo, en redes de suministro permisionadas es común usar IBFT 2.0 o QBFT para lograr finalización inmediata de las transacciones, mientras que en Ethereum público Besu simplemente sigue las reglas de consenso PoS impuestas por la red .

Este sistema usará Clique por razones de alcance de proyecto.

4.1.2.2. APIs

Hyperledger Besu proporciona APIs estándar que facilitan tanto la integración con sistemas externos como la monitorización de la propia red. Las principales APIs que ofrece Besu son:

- **API JSON-RPC (HTTP/WS):** Es la interfaz principal para consultar y enviar transacciones a la Blockchain Ethereum [53]. Besu soporta el conjunto de métodos RPC de Ethereum común, por ejemplo:
  - eth_blockNumber, eth_getBlockByNumber para obtener bloques.
  - eth_getTransactionByHash para ver transacciones.
  - eth_call para ejecutar localmente una función de contrato inteligente sin realizar transacción (lecturas).
  - eth_sendRawTransaction para enviar transacciones
  - eth_getLogs para obtener eventos (útil para extraer eventos de trazabilidad de los contratos).

  Besu puede habilitar esta API tanto sobre HTTP como sobre WebSocket, permitiendo suscripción a eventos en tiempo real vía WS (método eth_subscribe). Esto resulta útil para el componente IoT: por ejemplo, un servicio podría suscribirse a eventos de “nuevo registro de lote” para notificar a ciertas máquinas.
- **API de permisionamiento (RPC extendido):** Cuando se activa el permisionamiento on-chain o local, Besu expone métodos para gestionar las listas de permisos [54].

- **API de gestión (Web3):** Métodos como web3_clientVersion (que devuelve la versión de Besu), net_peerCount (número de peers conectados) o admin_peers (información de peers) ayudan a monitorizar el estado de la red: carga, errores, etc. Besu implementa estos métodos de administración comunes a clientes Ethereum [55].

4.1.2.3. Otros

Hyperledger Besu permite un despliegue de nodos completamente personalizado, lo que facilita la configuración de la red según las necesidades específicas del caso de uso. Esta flexibilidad permite ajustar parámetros clave —como el algoritmo de consenso, la frecuencia de los bloques o las políticas de acceso— para adaptarse a distintos entornos operativos o requisitos funcionales. Además, pudiendo ejecutarse en servidores locales, contenedores Docker, entornos en la nube o arquitecturas orquestadas con Kubernetes. También cuenta con una interfaz de línea de comandos (CLI) que permite automatizar la configuración y puesta en marcha de nodos, así como herramientas de monitorización integradas (como Prometheus y Grafana) para analizar el estado y rendimiento de la red. Todo ello convierte a Besu en una solución robusta, flexible y lista para entornos de producción exigentes.

4.1.3. Ejemplo: Besu en trazabilidad de cadena de suministro

Un caso de uso real de Besu es la plataforma **AURA**, un consorcio impulsado por LVMH para productos de lujo. Lanzado en 2019, AURA utiliza una red *permissioned* sobre Besu desplegada en Azure para rastrear el origen y la autenticidad de artículos de lujo a lo largo de la cadena de suministro [56].

Figura 4.2: Logo de AURA

4.2. Dispositivos IoT utilizados

Para garantizar un sistema de trazabilidad alimentaria eficiente, es fundamental recopilar datos clave en tiempo real a lo largo de toda la cadena de suministro. Entre los datos necesarios se incluyen: temperatura y humedad ambiental, localización geográfica del producto, detección de presencia o movimiento (para identificar manipulación o actividad en puntos críticos), así como marcas temporales que permitan reconstruir la ruta y condiciones del producto.

4.2.1. Datos requeridos para la trazabilidad

La trazabilidad alimentaria implica la capacidad de seguir el movimiento de un alimento a través de las distintas etapas de la cadena de suministro: producción, transformación y distribución. Según las directrices de la AESAN [11] y los reglamentos europeos como el (CE) 178/2002 [57], es necesario garantizar la recolección de información precisa y verificable en cada punto de control. En el contexto de una solución tecnológica basada en IoT, los datos esenciales que deben ser recopilados incluyen:

- **Condiciones ambientales:** se refieren a variables como la temperatura y la humedad relativa del entorno en el que se encuentra el producto. Estas condiciones son críticas para productos perecederos, como frutas, lácteos o carnes [cite], ya que fuera de los rangos adecuados se compromete la seguridad alimentaria. En este sistema, dichas variables son monitorizadas en tiempo real mediante sensores como el DHT11, permitiendo detectar desviaciones y activar alarmas o acciones correctivas automatizadas.
- **Ubicación geográfica:** registrar de manera precisa la posición del lote durante el transporte permite reconstruir el recorrido completo del alimento. La ubicación GPS también es útil para verificar si se ha respetado la ruta logística establecida o si ha habido paradas no autorizadas. Para ello, se emplea un módulo como el Arduino MKR GPS Shield, que proporciona coordenadas en tiempo real con precisión métrica.
- **Actividad o manipulación no autorizada:** conocer si el contenedor ha sido abierto, manipulado o si ha habido movimientos no planificados es vital para detectar posibles incidentes de seguridad o deterioro de la mercancía. Este tipo de eventos se detecta mediante sensores PIR (Passive Infrared), capaces de registrar presencia humana o movimiento dentro de su rango de detección.

- **Marcas temporales (timestamp):** cada evento capturado por los sensores debe ir asociado a una marca de tiempo fiable que permita su ordenamiento cronológico y análisis posterior. Esto es fundamental para auditorías, estudios de causa-efecto y cumplimiento normativo. El microcontrolador NodeMCU v3 sincroniza su reloj interno a través de NTP (Network Time Protocol), garantizando precisión en el registro de eventos, incluso si se producen de forma desconectada temporalmente [cite].

La recopilación automatizada de estos datos, mediante sensores conectados a una arquitectura IoT distribuida, garantiza no solo la fiabilidad de la trazabilidad, sino también su auditoría en tiempo real. Además, permite el cumplimiento de los principios de transparencia, responsabilidad y respuesta rápida ante crisis de seguridad alimentaria [11].

4.2.2. Componentes utilizados

A continuación en la tabla 4.1 se describen los dispositivos IoT empleados en el sistema propuesto, organizados de acuerdo con la arquitectura de referencia de IoT (ITU-T Y.2060) [36], en la capa de dispositivo:

Dispositivo	Función en el sistema	Características técnicas clave
NodeMCU v3 (ESP8266)	Microcontrolador central con conectividad WiFi. Gestiona sensores y transmite datos a nodos blockchain.	Procesador L106 32-bit RISC a 80 MHz; 4 MB de flash; 64 KB de SRAM; WiFi 802.11 b/g/n; 11 pines digitales I/O; 1 pin analógico; alimentación 3.3V; consumo 10uA–170mA. [58]
DHT11	Sensor de temperatura y humedad para monitorizar condiciones ambientales del producto.	Rango de temperatura: 0–50 ℃ (±2 ℃); humedad: 20–90 % RH (±5 % RH); resolución: 1 ℃ y 1 % RH; tiempo de respuesta: 6–15 s; alimentación 3.3–5.5V; interfaz digital de un solo cable. [59]
Arduino MKR GPS Shield	Módulo de geolocalización para registrar coordenadas en tiempo real.	Basado en el módulo u-blox SAM-M8Q GNSS; comunicación vía UART/I2C; alimentación 3.3V; batería de respaldo CR1216; dimensiones: 45x25 mm; peso: 14 g. [60]
Sensor PIR	Detecta movimiento o presencia cerca del producto. Útil para registrar posibles manipulaciones.	Rango de detección: hasta 6 m; ángulo de detección: 110 ° x 70 °; salida digital alta (3V) cuando se detecta movimiento; alimentación 5–12V; consumo bajo; tiempo de respuesta ajustable. [61]

Tabla. 4.1: Dispositivos IoT utilizados en el sistema de trazabilidad alimentaria.

4.2.3. Consideraciones técnicas

La selección e integración de los dispositivos debe responder a las características clave de los sistemas IoT mencionadas en la sección anterior: bajo consumo energético, comunicación inalámbrica eficiente, bajo coste y respuesta orientada a eventos. A continuación se resumen algunos aspectos técnicos relevantes:

- **Comunicación y conectividad:** El NodeMCU v3 emplea WiFi (IEEE 802.11 b/g/n), adecuado para entornos con infraestructura de red disponible. Los senso-

res como el DHT11 y el PIR se comunican a través de buses digitales simples (GPIO), mientras que el módulo GPS utiliza UART para transmisión de datos [58, 59, 61, 60].

- **Velocidad y frecuencia de muestreo:** El DHT11 ofrece lecturas cada segundo, lo que es suficiente para monitorización ambiental. El GPS opera con una tasa de actualización entre 1–10 Hz, y el sensor PIR reacciona casi instantáneamente a cambios térmicos en el entorno [59, 61, 60].
- **Latencia y tolerancia a fallos:** La latencia depende en gran medida de la red WiFi. Para entornos móviles o rurales, se recomienda implementar un mecanismo de almacenamiento temporal local en el NodeMCU que permita transmitir los datos una vez restaurada la conectividad [58].
- **Consumo energético:** Si bien el NodeMCU tiene un consumo moderado, el módulo GPS puede incrementar significativamente la demanda energética. Esto debe considerarse especialmente en sistemas autónomos alimentados por baterías [58].
- **Integración en la arquitectura IoT:** Estos dispositivos conforman la capa física y de red del modelo ITU, permitiendo una recopilación automatizada de datos que serán enviados a plataformas de trazabilidad y almacenamiento seguro basadas en blockchain.

La elección de estos componentes refleja un compromiso entre simplicidad, funcionalidad y eficiencia, además permite construir un sistema IoT orientado a eventos, reactivo y seguro, alineado con los principios fundamentales de las soluciones modernas para trazabilidad alimentaria.

Capítulo 5

INGENIERÍA SOFTWARE

En este capítulo se presenta el desarrollo del sistema de trazabilidad aplicando el ciclo de vida del software en cascada, con el objetivo de estructurar y justificar cada decisión técnica tomada. En él las etapas se llevan a cabo una detrás de otra de forma lineal, así sólo cuando la primera fase se termina se puede empezar con la segunda, y así progresivamente.

El ciclo se estructura en las siguientes fases:

- **Análisis**: Definición de requisitos funcionales y no funcionales, casos de uso y objetivos del sistema.
- **Diseño**: Propuesta del sistema, incluyendo los subsistemas IoT, blockchain y la interfaz web.
- **Implementación**: Desarrollo de los componentes descritos en el diseño, incluyendo la configuración de nodos, programación de dispositivos, desarrollo de contratos inteligentes y la aplicación web.
- **Pruebas**: Validación funcional del sistema mediante una simulación de trazabilidad, observando la interacción entre las capas y el cumplimiento de requisitos.
- **Despliegue**: Puesta en funcionamiento del sistema completo en el entorno de pruebas.

Esta estructura metodológica permite observar cómo se aplica la ingeniería del software en un entorno realista. Puede observarse mejor el ciclo descrito en la siguiente figura 5.1.

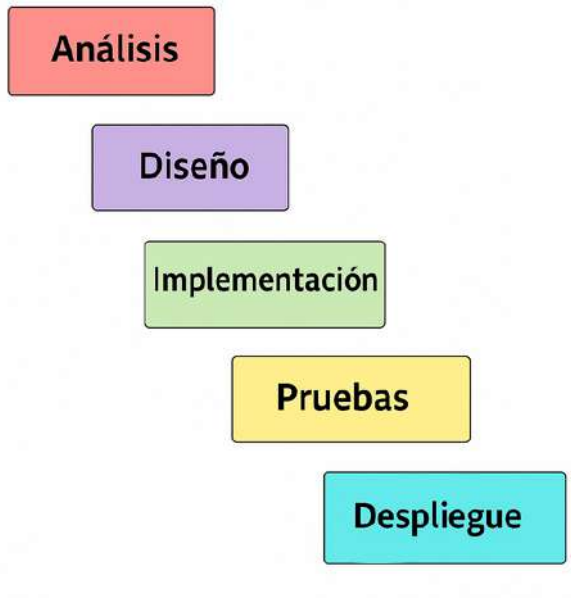

Figura 5.1: Fases del ciclo en cascada.

5.1. Análisis

Esta es la primera fase del ciclo en cascada, donde se analizan los requisitos del sistema empezando por una definición del alcance del proyecto.

5.1.1. Definición del sistema

El sistema de trazabilidad basado en tecnologías *Blockchain* e *Internet of Things* (IoT), es fundamental que garantice la capacidad de seguir un producto desde su punto de origen hasta el consumidor final. Este seguimiento detallado permite mejorar la transparencia, la seguridad alimentaria y la eficiencia de las operaciones logísticas.

Con el fin de demostrar la viabilidad técnica del sistema propuesto, se desarrollará una simulación que reproduce su funcionamiento en un entorno controlado. En esta simulación, se recrea el recorrido completo de un producto a lo largo de la cadena de suministro, pasando por distintos puntos de control o etapas numeradas (por ejemplo, Punto 1, Punto 2, Punto 3, etc.), hasta llegar a su destino final.

Para simplificar el modelo y enfocarnos en la trazabilidad técnica, se omite la representación detallada de los actores habituales involucrados en la cadena (como productores, distribuidores, minoristas, etc.). En su lugar, se introduce un único agente abstracto encargado de transportar el producto a través de las diferentes etapas. Este agente será responsable de registrar en la blockchain los datos correspondientes

a cada etapa.

La simulación generada permitirá evaluar cómo se registra, transmite y consulta la información trazada mediante los dispositivos IoT conectados al sistema y cómo esta se almacena de forma inmutable en la *blockchain*. Así, se pone a prueba la arquitectura planteada en un escenario simplificado pero representativo del mundo real, facilitando el análisis de su comportamiento, escalabilidad y posibles mejoras.

5.1.2. Análisis del sistema

El sistema propuesto tiene como objetivo principal demostrar la trazabilidad integral de productos utilizando tecnologías Blockchain e IoT. Para comprender claramente cómo interactúan los usuarios y los distintos elementos que conforman el sistema, se ha realizado un análisis basado en casos de uso y la definición de requisitos funcionales y no funcionales que guían su desarrollo.

5.1.2.1. Casos de uso

Los casos de uso permiten entender fácilmente qué acciones pueden realizar los usuarios y qué funcionalidades les ofrece el sistema. Considerando que se trata de una simulación orientada a mostrar la trazabilidad completa mediante Blockchain e IoT, se han identificado los siguientes casos clave:

- **CU1 - Alta de producto:** Permite introducir un producto nuevo al sistema, asignándole automáticamente un identificador único que asegura su seguimiento durante todo el proceso logístico. Esto es fundamental para garantizar la unicidad de cada producto, evitando errores o duplicaciones durante su seguimiento.
- **CU2 - Registro automático de datos IoT:** Sensores IoT instalados en diferentes etapas capturan datos relevantes como ubicación, temperatura, humedad y tiempo. Estos datos se envían automáticamente y quedan registrados en la blockchain, garantizando integridad y transparencia. La automatización del registro reduce significativamente errores humanos y asegura la consistencia de los datos.
- **CU3 - Consulta de trazabilidad:** El usuario puede acceder a todo el historial del producto utilizando su identificador. La información se presenta de forma crono-

lógica y detallada, reflejando los registros almacenados en la blockchain. Esto facilita auditorías precisas y rápidas para cualquier producto.

- **CU4 - Visualización de datos:** Los datos recopilados por sensores se presentan gráficamente para facilitar su análisis. Estas gráficas permiten monitorear variables críticas del producto a lo largo del proceso, facilitando una rápida identificación de desviaciones o problemas potenciales.

- **CU5 - Consulta de información de la red:** Permite obtener información técnica acerca de transacciones y bloques generados por la blockchain, así como detalles específicos sobre los productos registrados. Esto es especialmente relevante para validar el funcionamiento interno del sistema y realizar auditorías técnicas y de calidad.

Estos casos de uso, visibles en la figura 5.2, son fundamentales para definir las interacciones básicas del sistema y proporcionan una base sólida para derivar los requisitos funcionales y no funcionales.

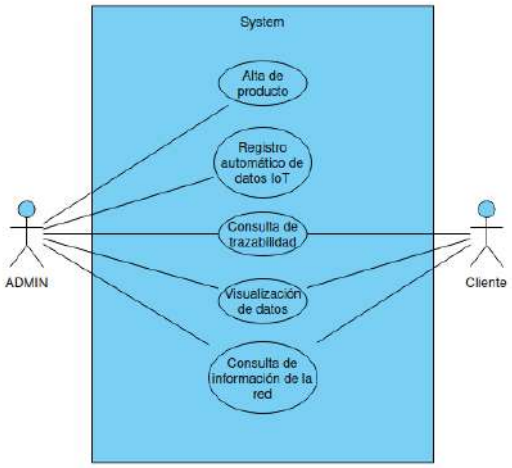

Figura 5.2: Diagrama de casos de uso.

En el sistema se identifican dos actores principales: Admin y Cliente. El Admin representa al desarrollador, encargado de gestionar el sistema, registrar automáticamente los datos provenientes de los dispositivos IoT y supervisar el funcionamiento general de la red. Por otro lado, el Cliente es un usuario externo que accede al sistema con el objetivo de consultar la trazabilidad de los productos, visualizar los datos recopilados y obtener información relevante de la red de forma transparente.

5.1.2.2. Requisitos funcionales

Los requisitos funcionales son las capacidades concretas que el sistema debe ofrecer para cumplir su objetivo. Se han identificado los siguientes requisitos esenciales:

- **RF1.** Recuperar toda la información de trazabilidad de un producto mediante su identificador único. Esto permite al usuario verificar fácilmente cualquier aspecto del historial del producto.
- **RF2.** Permitir al usuario consultar detalladamente la trazabilidad del producto, incluyendo datos como ubicación, temperatura, humedad, número de manipulaciones acumuladas y hora exacta de cada etapa. Detallar esta información es crucial para auditorías precisas.
- **RF3.** Facilitar el alta simulada de nuevos productos, generando automáticamente un identificador para comenzar su seguimiento. Esto asegura la uniformidad y simplifica el proceso inicial.
- **RF4.** Recoger datos automáticamente mediante sensores IoT y registrarlos en la blockchain sin intervención manual. Automatizar esta función evita posibles errores humanos y mejora la fiabilidad.
- **RF5.** Consultar información detallada sobre las transacciones almacenadas en la blockchain a través de su hash.
- **RF6.** Acceder a la información específica sobre los bloques generados en la blockchain, fundamental para supervisar y gestionar la red.
- **RF7.** Consultar información detallada y específica de cada producto trazado, necesaria para garantizar una trazabilidad completa y efectiva.
- **RF8.** Generar representaciones gráficas claras para visualizar la evolución de variables críticas asociadas a la trazabilidad, facilitando una rápida interpretación de los datos.

5.1.2.3. Requisitos no funcionales

Los requisitos no funcionales establecen criterios clave relacionados con la calidad general del sistema, incluyendo aspectos como rendimiento, usabilidad y escalabilidad. Los principales requisitos no funcionales son:

- **RNF1.** Contar con una interfaz web intuitiva que permita a usuarios sin conocimientos técnicos consultar fácilmente la información. Esto facilita la adopción del sistema en diferentes contextos operativos.
- **RNF2.** Garantizar que la transmisión de datos IoT se realice en tiempo real o con un retardo mínimo, asegurando la actualidad y precisión de la información, aspecto crítico en procesos logísticos.
- **RNF3.** Ofrecer una arquitectura escalable que permita monitorizar múltiples productos simultáneamente sin afectar negativamente al rendimiento del sistema, esencial en entornos industriales reales.
- **RNF4.** Utilizar una arquitectura modular que permita incorporar fácilmente nuevas funcionalidades o integrar nuevas tecnologías en el futuro, asegurando la sostenibilidad y adaptabilidad del sistema.
- **RNF5.** Proporcionar interfaces gráficas responsive que se adapten eficazmente a diferentes dispositivos y tamaños de pantalla, permitiendo un acceso ubicuo a la información.
- **RNF6.** Garantizar tiempos de respuesta inferiores a 2 segundos para consultas básicas de información sobre trazabilidad, asegurando una experiencia eficiente y efectiva para el usuario.

5.2. Diseño

El diseño define la estructura técnica que da soporte a los requisitos identificados en la fase de análisis. Esta sección presenta la arquitectura general del sistema, los componentes funcionales principales, el diseño de los datos manejados, las interfaces de usuario, y el flujo de operación que permite realizar la trazabilidad de los productos mediante tecnologías Blockchain e IoT.

A continuación, se detalla la arquitectura general y cada uno de los elementos que la componen.

5.2.1. Red de nodos

Para lograr un sistema verdaderamente descentralizado y distribuido, es esencial contar con múltiples máquinas capaces de ejecutar nodos que colaboren entre sí. En

esta simulación, la red de nodos se organiza en tres niveles diferenciados como puede apreciarse en la figura 5.3:

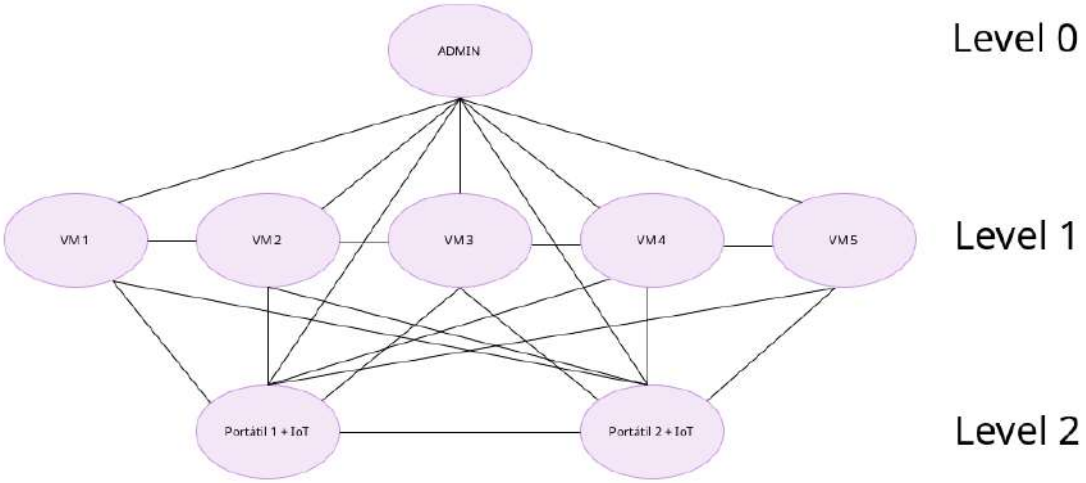

Figura 5.3: Esquema de la red.

- **Level 0 (Administrador):** Un nodo denominado `ADMIN` actúa como el administrador central de la red. Su función principal es gestionar los permisos para la incorporación de nuevos participantes, asegurando que solo los nodos autorizados puedan unirse. Este nodo también sirve como *bootnode*, el punto de entrada inicial para otros nodos.

- **Level 1 (Nodos fijos):** Cinco máquinas virtuales (VMs) ejecutan clientes Besu y se conectan automáticamente al nodo `ADMIN`. Estas VMs representan nodos estables que residen en el mismo host físico que el nodo administrador, lo que simplifica su integración en la red. Su rol principal es procesar, validar y almacenar datos en la cadena de bloques.

- **Level 2 (Nodos móviles):** Dos portátiles, cada uno asociado a dispositivos IoT, funcionan como nodos móviles. Estos dispositivos recopilan datos de trazabilidad y los transmiten a la red a través de los portátiles. Este nivel simula la captura de datos en tiempo real.

Esta estructura jerárquica permite una combinación de control centralizado (vía el nodo `ADMIN`) y descentralización operativa entre los nodos fijos y móviles.

Conectividad y visibilidad

La comunicación efectiva entre todos los nodos es un pilar fundamental del sistema. Para garantizar que las máquinas sean mutuamente accesibles, se ha implementado una VPN basada en OpenVPN. A continuación, se describen los detalles de la conectividad:

- **Configuración de la VPN:**
  - El nodo `ADMIN` y los dos portátiles se conectan directamente a la VPN, obteniendo direcciones IP asignadas por esta red virtual.
  - Las cinco VMs, al estar alojadas en el mismo host físico que el nodo `ADMIN`, acceden al túnel local de la VPN sin necesidad de una conexión adicional. Esto optimiza la comunicación dentro del sistema.
- **Puertos y firewall:** Para que Besu funcione correctamente, se deben abrir y configurar los siguientes puertos en cada máquina:
  - `30303`: Usado para la comunicación peer-to-peer entre nodos.
  - `8545`: Puerto por defecto para la API JSON-RPC sobre HTTP, que permite consultas y gestión remota.

  Además, las reglas del firewall en cada máquina deben permitir el tráfico entrante y saliente en estos puertos, y la VPN debe configurarse para no bloquear los protocolos utilizados por Besu (como TCP y UDP).

Arquitectura private-permissioned

La red se diseña como *private* (privada) y *permissioned* (permisionada), lo que significa que está restringida a participantes autorizados y no es accesible al público. Esta característica se implementa mediante las siguientes configuraciones:

- **Control de acceso:**
  - El nodo `ADMIN` actúa como la autoridad de permisos, utilizando las APIs de permisionamiento de Besu. Solo los nodos incluidos en su lista de permitidos pueden unirse a la red.
  - Además, el `ADMIN` funciona como *bootnode*, proporcionando a los nuevos nodos la información necesaria para conectarse.

- **Consenso:**
  - Se emplea el algoritmo *PoA - Clique*. En este modelo, solo los nodos designados como *signers* pueden validar y sellar bloques en la cadena. En esta implementación, únicamente el nodo `ADMIN` tiene este rol, centralizando la validación mientras se mantiene la distribución de datos entre los nodos.

Configuración de clientes y nodos

La configuración de cada nodo es un proceso crítico para garantizar la uniformidad y el funcionamiento de la red. A continuación, se detallan los puntos necesarios para configurar cada máquina:

1. **Generación de Claves**
   Cada nodo debe contar con un par de claves criptográficas (privada y pública). La clave pública deriva la dirección única del nodo, que actúa como su identificador en la red.
2. **Identificación de Nodos mediante Enodes**
   Cada nodo debe generar un identificador `enode`, que integra su dirección IP, puerto y clave pública. Este diseño facilita el *descubrimiento y la conexión segura* entre nodos, siendo un componente esencial para la arquitectura peer-to-peer de la red.
3. **Configuración del Archivo Genesis**
   Todos los nodos deben de compartir un archivo `genesis.json` común, que establece los parámetros iniciales de la red. Asegurando que la red inicie desde un estado consistente y que todos los nodos operen bajo las mismas condiciones.
4. **Arranque de Besu:** Configurar el inicio de Besu en cada nodo para que se ajuste a las siguientes necesidades:
   - Asegurar la persistencia de la cadena de bloques y el estado de cada nodo, el sistema necesita un directorio dedicado donde almacenar bloques y estados.
   - Garantizar que todos los nodos compartan el mismo bloque génesis y parámetros de red (consenso, signers iniciales), es imprescindible utilizar un único archivo `genesis.json` común.
   - Designar al nodo ADMIN como punto de entrada (*bootnode*).

- Ofrecer los servicios de consulta y gestión remota, cada nodo necesita exponer su API JSON-RPC por HTTP en todas las interfaces y en el puerto 8545.
- Cubrir las operaciones de aplicación, diagnóstico y consenso, el sistema debe habilitar los módulos ETH, NET y WEB3, el módulo específico de consenso CLIQUE, el pool de transacciones y las herramientas de debug.
- Garantizar la interoperabilidad con clientes y herramientas externas (p. ej. paneles de visualización o scripts de prueba), la configuración debe permitir orígenes y hosts múltiples.

5. **Configuración específica del nodo ADMIN:** Habilitar las funciones de permisionamiento en el `ADMIN`, asegurando que solo las direcciones en su lista blanca puedan unirse a la red.

Archivo génesis común

Como parte del diseño de una red blockchain privada y permisionada, todos los nodos deben compartir un mismo archivo génesis, que define las condiciones iniciales y los parámetros de funcionamiento de la red desde el bloque cero. Este archivo representa el punto de partida común y su coherencia es esencial para garantizar la correcta sincronización entre los participantes.

Los elementos clave para este archivo son:

- **Identificador de red (`chainId`)**: Se asigna un identificador único, como `1982`, que permite distinguir esta red de otras redes públicas o privadas y evita colisiones en transacciones firmadas.
- **Política de gas sin costes**: Para facilitar la experimentación y despliegue de contratos sin restricciones, se diseña una red sin tasas de gas, estableciendo el coste de ejecución en cero.
- **Parámetros de gas y contratos**: Se definen valores máximos para el límite de gas por bloque y para el tamaño de los contratos, con el fin de permitir transacciones complejas y contratos inteligentes extensos sin restricciones artificiales.
- **Consenso basado en Clique**: El algoritmo de consenso elegido es *Proof of Authority* (PoA), en su variante Clique. Se establece un intervalo fijo de 60 segundos entre bloques y se configura el sistema para evitar la creación de bloques vacíos, lo que garantiza que cada bloque contenga información relevante.

- **Definición del signer inicial**: Se incluye la dirección del nodo `ADMIN` como único firmante autorizado en el bloque génesis, lo cual permite un control centralizado del sellado de bloques al inicio de la red.
- **Asignación de balances iniciales**: Las direcciones correspondientes al nodo `ADMIN` y a los nodos móviles (portátiles) reciben una dotación inicial de fondos, lo que les permite desplegar e interactuar con contratos sin depender de transferencias externas.

Este diseño del archivo génesis proporciona una base estable, coherente y funcional para una red orientada a la trazabilidad, con reglas predefinidas que garantizan su operatividad desde el primer bloque. Con estos pasos, cada nodo queda plenamente configurado para participar en la red de manera segura y coordinada.

NOTE

Normalmente, se sitúa el archivo génesis en el directorio base del nodo debido a que las opciones de despliegue utilizan rutas relativas.

5.2.2. Diseño del sistema IoT

El sistema IoT está diseñado para capturar datos físicos relevantes (ambientales, geoespaciales y de movimiento) y transmitirlos al sistema de registro blockchain mediante una estructura estandarizada. La arquitectura se basa en la integración de sensores conectados a un microcontrolador central, que procesa los datos y los transmite vía puerto serial a un portátil como se muestra en la figura 5.4.

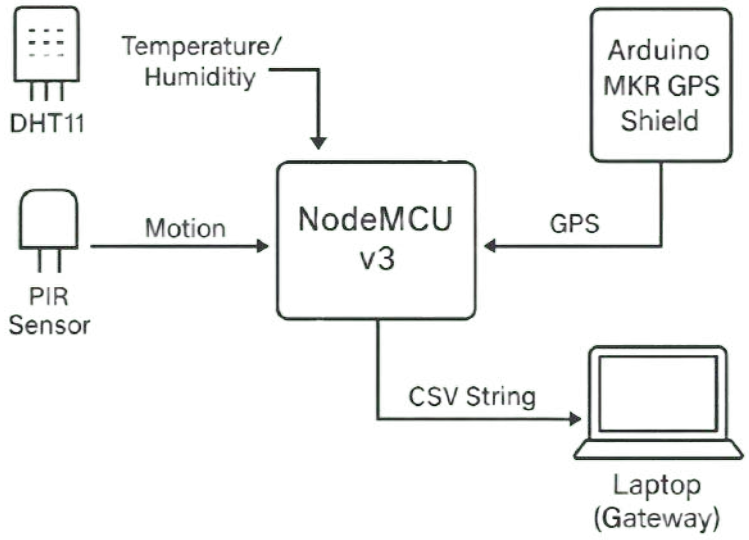

Figura 5.4: Arquitectura del sistema IoT.

Componentes empleados

Los elementos seleccionados para el diseño del sistema son:

- **NodeMCU v3**: microcontrolador basado en ESP8266, encargado de leer y procesar datos de sensores, y de transmitirlos por el puerto serial al portátil.
- **DHT11**: sensor digital de temperatura y humedad, conectado a un pin GPIO del NodeMCU.
- **Sensor PIR**: sensor pasivo de infrarrojos para detección de movimiento, también conectado a un GPIO.
- **Arduino MKR GPS Shield**: módulo de geoposicionamiento, conectado al NodeMCU mediante UART (Universal Asynchronous Receiver/Transmitter) a través de pines digitales.

Arquitectura y conexiones

El circuito propuesto se centra en el NodeMCU como núcleo del sistema. Todos los sensores se conectan a él, incluyendo el GPS, que comunica sus datos mediante

UART usando los pines digitales del microcontrolador. El esquema de conexión es el siguiente:

- **DHT11** y **PIR**: conectados a pines GPIO estándar para lectura digital.
- **GPS Shield**:
  - `VCC` del GPS conectado a `3V3` del NodeMCU.
  - `GND` del GPS conectado a `GND` del NodeMCU.
  - `TX` del GPS conectado al pin `D1` del NodeMCU (configurado como RX).
  - `RX` del GPS conectado al pin `D2` del NodeMCU (TX), aunque no se utiliza en esta etapa.

Este montaje permite al NodeMCU recibir directamente los datos de localización.

Requisitos del código embebido

El código cargado en el NodeMCU debe cubrir las siguientes tareas:

- Leer valores del sensor DHT11 a intervalos fijos.
- Detectar movimiento con el sensor PIR, e ir guardando el acumulado.
- Leer datos del GPS a través de UART desde el pin D1.
- Integrar los valores en una cadena de texto con formato predefinido.
- Transmitir la cadena por el puerto serial USB al portátil, que actuará como nodo blockchain de entrada.

Formato de los datos transmitidos

Para facilitar el procesamiento posterior, los datos se formatean como una cadena de texto plano, con campos separados por comas y finalizada en un salto de línea:

```
timestamp,temperatura,humedad,latitud,longitud,movimiento\n
```

Donde:

- `timestamp`: instante de captura en formato UNIX epoch.
- `temperatura`, `humedad`: datos leídos del DHT11.
- `latitud`, `longitud`: coordenadas GPS.
- `movimiento`: valor acumulado de numero de veces activado.

Ejemplo:

```
1715079600,22.4,45.0,40.4165,-3.7026,12\n
```

Este formato permite al sistema receptor (portátil) interpretar y reenviar los datos a través de transacciones hacia la red blockchain.

NOTE

Aunque los datos deben pasar por el gateway 5.2.4. Este es simplemente la herramienta que crea la transacción añadiendo estos datos y la firma. No se modifican los datos. El gateway y el sistema IoT forman un único ser. Ambos se alimentan de la misma máquina. No hay oportunidad en la que los datos puedan ser alterados maliciosamente por un tercero en el proceso.

5.2.3. Diseño del contrato de emisión de eventos de trazabilidad

Uno de los pilares del sistema propuesto es la capacidad de registrar eventos relevantes a lo largo del ciclo de vida de los productos. Para ello, se diseña un contrato inteligente específico cuya función principal es la emisión de eventos estructurados, ligados a un identificador único por producto. Estos eventos representan puntos de trazabilidad capturados por el sistema IoT y transmitidos a la red blockchain.

Objetivo y funcionalidad del contrato

El contrato tiene una única responsabilidad: emitir eventos asociados a productos individuales. Cada producto se identifica mediante un entero único (`id`), y a cada evento se le asocia un campo de datos en forma de cadena estructurada. La funcionalidad esperada incluye:

- Registrar un evento de trazabilidad por cada punto recibido del sistema IoT.
- Asociar dicho evento a un identificador único de producto.
- Permitir que terceros consulten el histórico de eventos de un producto de forma eficiente.

Estructura lógica y requisitos

El contrato debe cumplir con los siguientes principios:

- **Eventos indexados**: Cada evento emitido por el contrato debe estar asociado a un campo indexado (`id`), de modo que puedan filtrarse y consultarse eventos por producto sin recorrer toda la cadena.
- **Formato de datos**: El campo de datos se transmite como una cadena de texto en formato CSV, con los siguientes campos: timestamp, temperatura, humedad, latitud, longitud y movimiento. Esta estructura permite almacenar múltiples valores en un único evento y facilita su procesamiento posterior.
- **Minimización del almacenamiento en cadena**: Para reducir el coste computacional y evitar el uso excesivo de almacenamiento, se opta por un diseño sin variables de estado ni arrays en memoria. Todos los datos se transmiten a través de eventos, que son más económicos y eficientes para casos donde solo se requiere consulta histórica.
- **Acceso público**: La función de emisión debe estar disponible públicamente, permitiendo a los nodos móviles (portátiles) escribir directamente en la blockchain, siempre que dispongan del balance inicial necesario para cubrir el coste de gas.

NOTE

La red está diseñada para ser libre de gas; pero aún así, aunque no se cobre luego gas, se necesita de un balance mínimo para poder enviar transacciones.

Consideraciones para el despliegue en la red

El despliegue del contrato debe considerar las siguientes condiciones específicas del entorno:

- **Mismo ChainID de la red definida**: El contrato está diseñado para desplegarse sobre la red privada definida previamente. Se asegura la compatibilidad mediante la configuración del `chainId` en la transacción de despliegue.
- **Configuración de gas**: El contrato requiere asignación de un límite razonable de gas en el momento del despliegue (ej. 5,000,000 unidades) para cubrir la transacción inicial.

Procesamiento posterior de eventos

Aunque el contrato no almacena datos en variables persistentes, el sistema receptor (la aplicación web) debe ser capaz de:

- Filtrar eventos por identificador de producto.
- Descomponer la cadena de datos en los campos definidos.
- Generar estructuras JSON o equivalentes para su análisis o visualización.

Este diseño liviano, basado en eventos indexados, garantiza eficiencia, escalabilidad y compatibilidad con herramientas de visualización y análisis sin comprometer la simplicidad ni la estabilidad del contrato.

5.2.4. Integración de los sistemas: Gateway

La convergencia entre el sistema IoT (captura de datos físicos) y el sistema blockchain (registro inmutable de eventos) se materializa en un módulo de integración alojado en los portátiles (nodos móviles de Nivel 2). Dicho módulo actúa como pasarela —o *gateway*—, como se muestra en la figura 5.5 y debe ejecutar las funciones descritas a continuación:

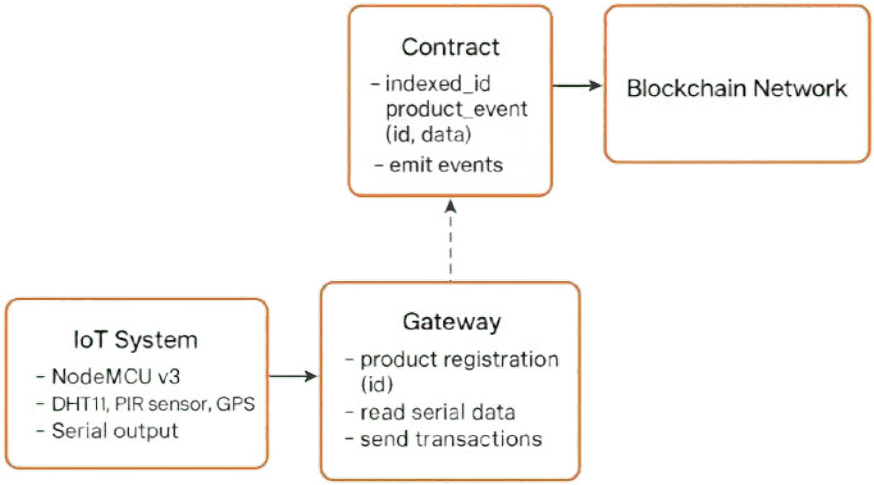

Figura 5.5: Flujo de integración entre los sistemas.

Flujo de datos y responsabilidad de la pasarela

1. **Alta de producto (*id*)**: Asignar un identificador entero único.
   - La pasarela interpreta el valor `id` como la operación de "alta" del producto.
   - No se requiere un contrato específico de registro: basta con que el primer evento emitido con ese `id` quede anclado en la blockchain; ello constituye su creación oficial.
2. **Adquisición de datos**:
   - Escuchar de forma continua el puerto serial donde el NodeMCU publica cadenas CSV terminadas en \n.
   - Parámetros a definir: 9 600 baudios, delimitador de línea, interfaz `COMx` en Windows y `/dev/ttyUSBx` en Linux.
3. **Prevalidación**:
   - Validación de formato y rango numérico; descarte de lecturas corruptas.
   - Resolución y autogestión del *nonce* para evitar colisiones de transacciones.
4. **Envio a blockchain**:
   - Formar una transacción dirigida al contrato de trazabilidad y envíar a la red privada.
   - Utiliza gas limit holgado y *gasPrice* simbólico (red sin costes).

Requisitos de diseño para la pasarela

- **Sincronización de producto:** control interno del ciclo de vida de cada `id` para evitar eventos huérfanos o duplicados.
- **Escalabilidad:** modularidad para ampliar el número de productos o la complejidad de los datos sin rediseñar la lógica central.
- **Compatibilidad de red:** parámetros de transacción alineados con el archivo génesis.

Interacciones entre capas

Se presenta en la tabla 5.1 cómo interactúan las distintas capas:

Capa	Interfaz	Responsabilidad principal
Percepción IoT	Puerto serial (USB)	Emitir cadena CSV por producto (`id`) con datos ambientales, coordenadas GPS y estado de movimiento (PIR).
Pasarela lógica	Biblioteca EVM	Registrar el alta del producto (`id`), firmar y enviar las transacciones a la red PoA.
Red blockchain	JSON-RPC 8545	Anclar los eventos indexados en bloques de 60 s y garantizar su orden e integridad mediante consenso Clique.

Tabla. 5.1: Interacción entre las capas del sistema de trazabilidad.

5.2.5. Interfaz - Aplicación Web

La aplicación web es la puerta de entrada para desarrolladores, auditores y clientes que necesitan inspeccionar, en tiempo real, la integridad de la cadena de suministro. Se documenta aquí su diseño lógico y visual con el objetivo de justificar las decisiones adoptadas, facilitar su mantenimiento y demostrar su alineación con los objetivos globales del proyecto.

Objetivo y alcance

Propósito general

La interfaz web actúa como único punto de observabilidad del ecosistema Blockchain–IoT. Está diseñada exclusivamente para **consultar** datos de productos, bloques, transacciones y métricas de red, sin capacidad alguna de modificar el estado de la cadena. Este enfoque *read-only* reduce la superficie de ataque y simplifica las tareas de auditoría.

Objetivos específicos

1. Consolidar en una misma vista datos de productos, bloques y transacciones.
2. Mostrar el estado operativo de la red (número de nodos, altura de bloque, ritmo de generación) para diagnosticar incidentes de consenso.
3. Facilitar la toma de decisiones logísticas mediante mapas, gráficas y tablas.

Alcance del primer lanzamiento

La primera versión del sistema permite consultar información clave relacionada con productos, puntos, bloques y transacciones, además de ofrecer una visualización actualizada de métricas de red con un desfase máximo de 60 segundos. La interfaz ha sido diseñada siguiendo un enfoque desktop-first, aunque se adapta correctamente a distintos tamaños de pantalla mediante un diseño responsive.

Este lanzamiento no contempla la posibilidad de firmar ni enviar transacciones directamente desde el navegador. Tampoco incluye funcionalidades para el alta manual de productos, ya que este proceso se realiza automáticamente a través de la pasarela. La gestión de usuarios, nodos o permisos de red queda fuera del alcance actual, al igual que el sistema de alertas o notificaciones push, cuya implementación está prevista para fases futuras.

En cuanto a los supuestos y limitaciones, se parte de un escenario con un volumen moderado de datos —en torno a decenas de registros simultáneos—, por lo que no se considera necesario implementar un sistema de paginación por streaming masivo. Se asume que el nodo `JSON-RPC` estará disponible de forma continua. Además, las métricas provenientes de los sensores se actualizan cada 60 segundos, en línea con el intervalo de generación de bloques definido por el mecanismo de consenso PoA.

Relación con los requisitos globales

La interfaz cumple los requisitos funcionales **RF1**, **RF2**, **RF5**, **RF6**, **RF7** y **RF8** al permitir: recuperar y detallar la trazabilidad de cualquier producto, explorar bloques y transacciones, y visualizar variables críticas mediante gráficas interactivas. También satisface los requisitos no funcionales **RNF1**, **RNF5** y **RNF6** gracias a su interfaz intuitiva, *responsive* y con tiempos de respuesta inferiores a 2 s para consultas básicas. Los requisitos restantes se abordan en las capas IoT, pasarela y red de nodos.

Arquitectura de la solución

- **Front-end**: React 18 con Vite; enrutamiento con React Router 6 y *code-splitting* dinámico por ruta.
- **Estilos**: Tailwind CSS; barra superior en gris `#1F2937`, color de acento azul `#2563EB`, tipografía sans-serif (14 px base). Diseño *desktop-first*.
- **Acceso a datos**: Ethers v6 mediante proveedor HTTP `JSON-RPC`. El endpoint se define en `.env`.

Mapa de navegación

- **Barra superior fija**: Logo «**TBI**» y tres botones ->*Trazabilidad*| *Blocks*| *TXs*. El botón activo se destaca para indicar la sección corriente.
- **Buscador global**: Campo de texto centrado en la sección principal/trazabildiad para buscar por ID de producto, hash de bloque y hash de transacción. Pulsar `Enter` redirige a la vista correspondiente.
- **Rutas principales**
  - `/` → Dashboard de trazabilidad.
  - `/blocks` → Explorador de bloques.
  - `/txs` → Explorador de transacciones.

  Rutas de detalle: `/product/:id`, `/block/:number`, `/block/:hash`, `/tx/:hash`.
- **Navegación contextual** En las tablas de bloques y transacciones, hacer clic en el hash abre la vista de detalle; un icono que permite *copiar al portapapeles* con *tooltip* “Copiado”.

Vistas principales

La interfaz web se articula en seis vistas funcionales. La Tabla 5.2 resume para cada una de ellas (i) el objetivo que persigue dentro del flujo de usuario y (ii) sus elementos y datos mostrados. Esta visión global permite verificar que las vistas cubren todos los requisitos RF1, RF2, RF5, RF6, RF7 y RF8 relacionados con la exploración de productos, bloques y transacciones.

Vista	Objetivo	Elementos UI / Datos mostrados
Dashboard	Visión rápida del estado global y acceso a búsquedas.	Tarjetas (nº nodos, productos, bloques, fecha último bloque); lista de últimos puntos; buscador universal; mapa Leaflet que centra el producto seleccionado.
Detalle de producto	Analizar la ruta y condiciones de un producto concreto.	Mapa con polilínea completa; gráficas temperatura/humedad; histograma de eventos PIR; tabla de puntos.
Explorador de bloques	Auditoría cronológica de la cadena.	Tabla infinita (hash abreviado con *copy-to-clipboard* y enlace al detalle del bloque), número bloque, nuemero de transacciones y fecha.
Detalle de bloque	Inspeccionar un bloque específico.	Metadatos de cabecera (hash completo, signer, gas, `extraData`); lista de transacciones con enlace a la vista de detalle.
Explorador de transacciones	Revisión rápida de actividad reciente.	Tabla con hash (enlace + *copy-to-clipboard*), `from`, `to`, valor, fecha.
Detalle de transacción	Verificación y trazabilidad de una operación puntual.	Campos estándar (hash, blockHash, gas, status) y decodificación del evento `indexed_id_product_event`.

Tabla. 5.2: Vistas principales de la interfaz web y datos mostrados.

La primera columna de la Tabla 5.2 identifica la vista (ruta principal en la barra de navegación). La segunda explica el propósito concreto de cada pantalla (detección rápida de anomalías, auditoría técnica, etc.), mientras que la tercera detalla los componentes clave: tarjetas, tablas con paginación infinita, mapas interactivos y gráficas de serie temporal. De este modo el lector puede comprobar de un vistazo qué información ofrece cada vista y cómo se satisfacen los requisitos funcionales y no funcionales previamente definidos.

Wireframes

Las Figuras 5.6–5.11 muestran la disposición de los elementos UI en cada vista antes de la implementación definitiva. Todos los wireframes siguen el mismo esquema visual: cabecera con logo y navegación, zona de contenido principal y acciones secundarias (botones, tablas o gráficos) en la parte inferior o lateral.

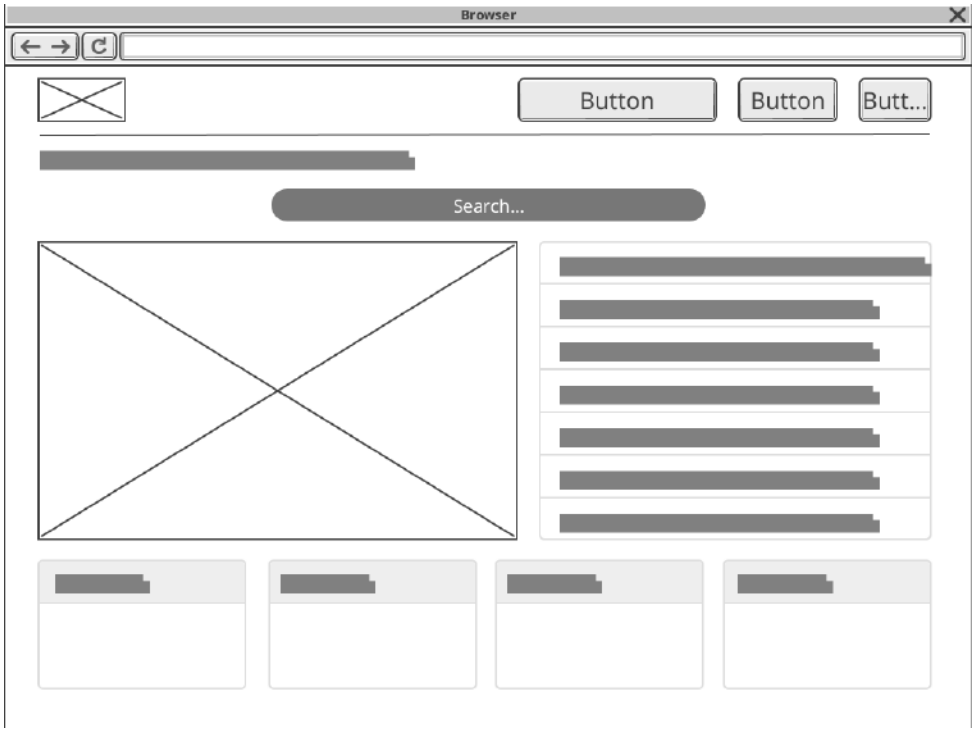

Figura 5.6: Wireframe del *Dashboard*.

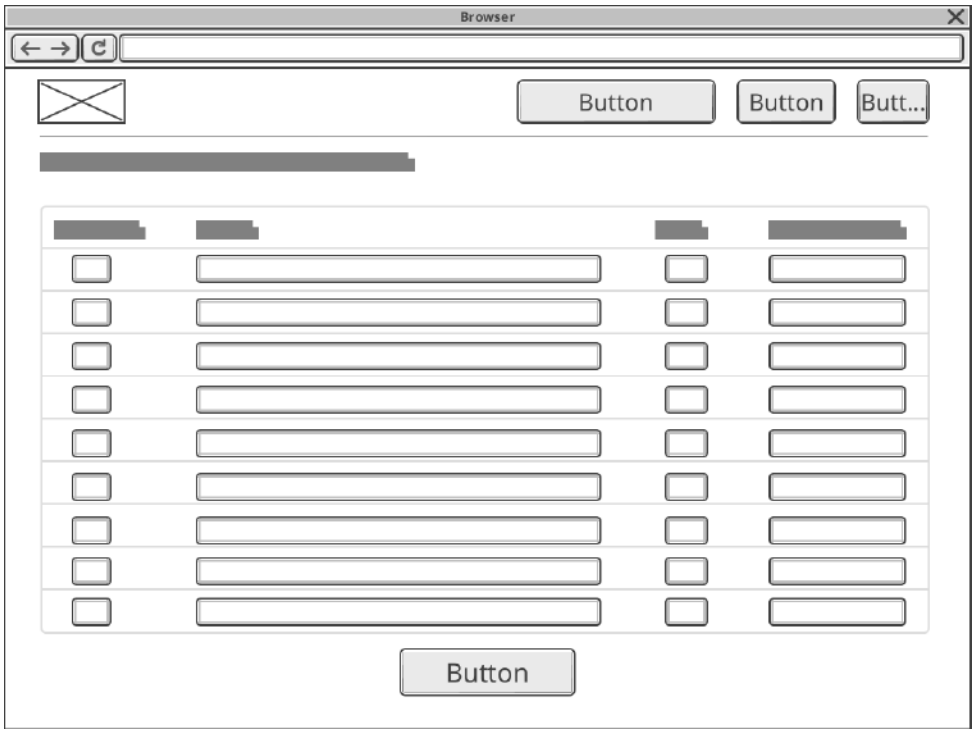

Figura 5.7: Wireframe del *Explorador de Bloques*.

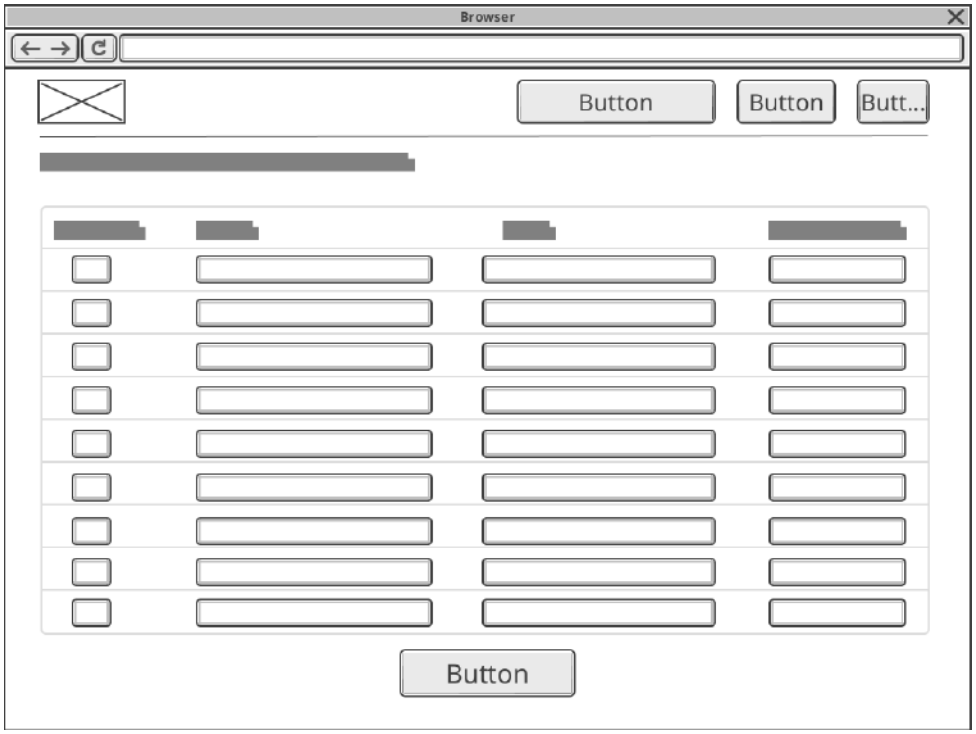

Figura 5.8: Wireframe del *Explorador de Transacciones*.

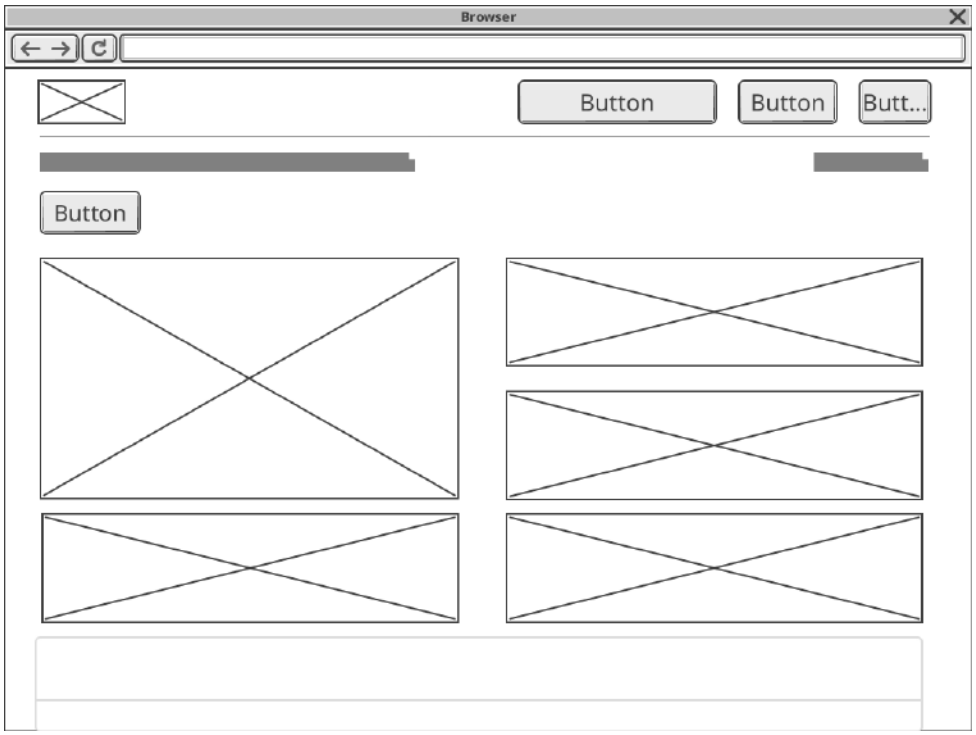

Figura 5.9: Wireframe del *Detalle de Producto*.

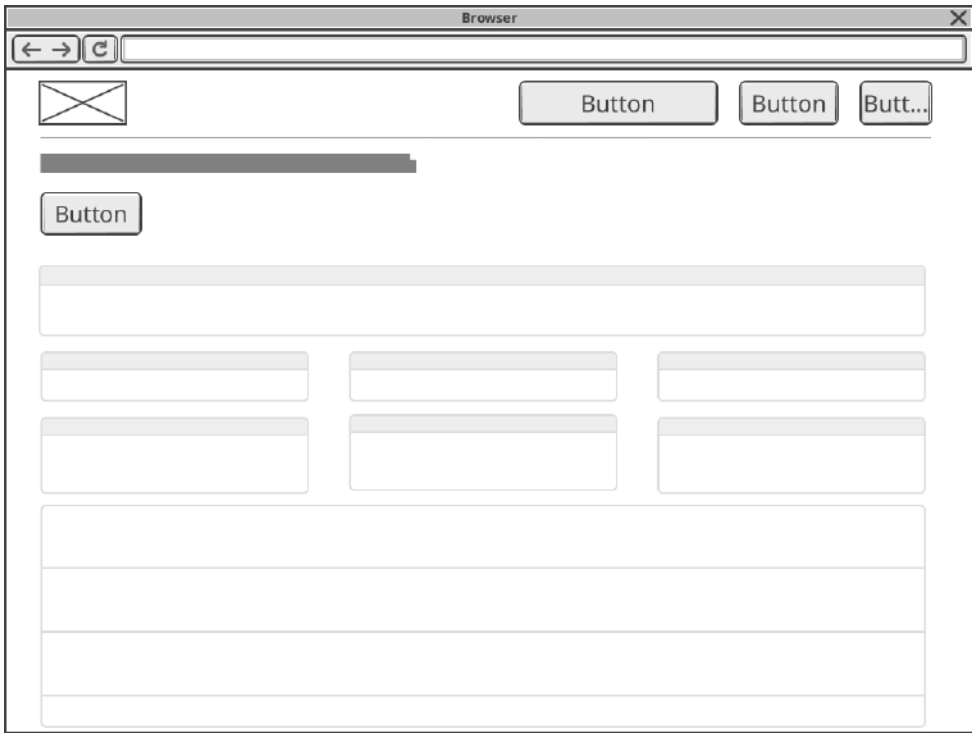

Figura 5.10: Wireframe del *Detalle de Bloque*.

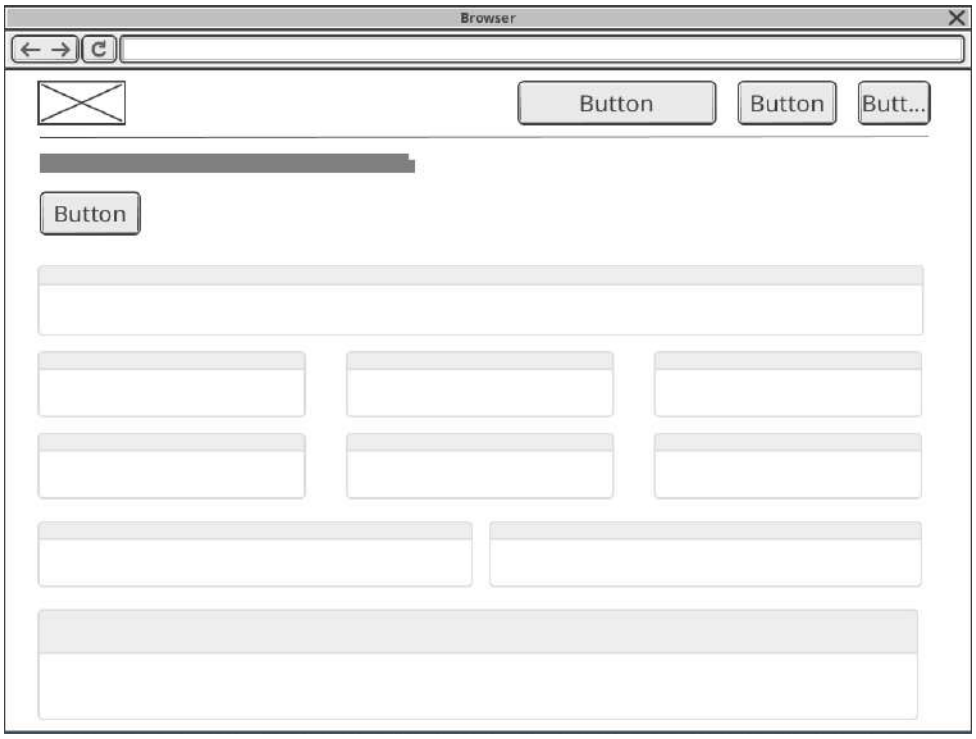

Figura 5.11: Wireframe del *Detalle de Transacción*.

5.2.5.1. Storyboards

Para ejemplificar el uso de la interfaz del sistema se van a exponer algunos *storyboards* sobre cómo lo utilizaría un usuario.

A) Consulta de información detallada de un producto y un punto del trayecto (figura 5.12). Se comienza en la página principal 5.6, en dicha ventana puedes elegir el producto que desear consultar. Puede hacerse por la barra de búsqueda central o la lista de productos de la derecha. Al elegir un producto se envía al usuario a la página de detalles del producto correspondiente 5.9, donde puede ver el resumen de información. A continuación, revisar la lista de puntos y elegir el deseado. Por último, se obtiene toda información relacionada con dicho punto/transacción 5.11.

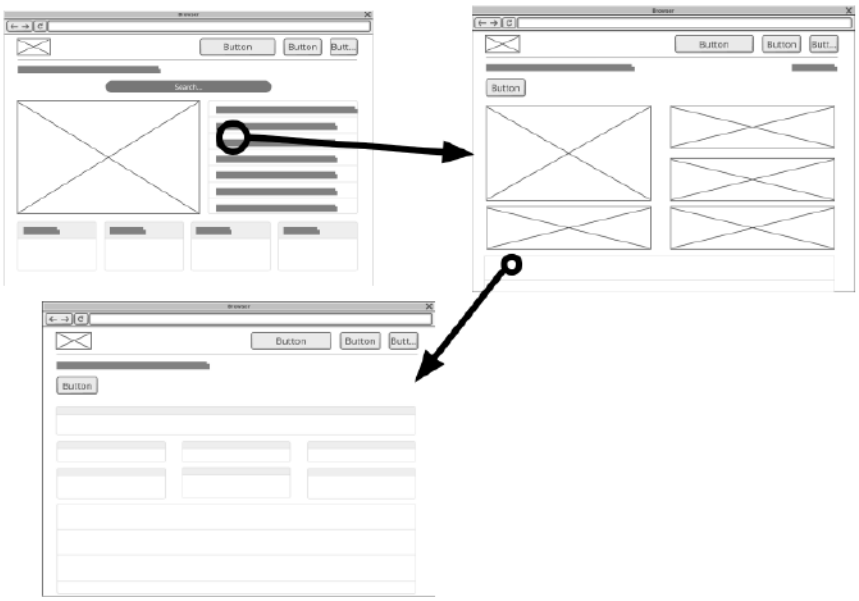

Figura 5.12: Storyboard A.

B) Consulta de una transacción o bloque por hash conocido (figura 5.13). Se comienza en la vista principal donde a través de la barra de búsqueda central se introduce el hash a buscar. De tal forma, puede consultarse toda información relacionada con dicho bloque 5.10 o transacción.

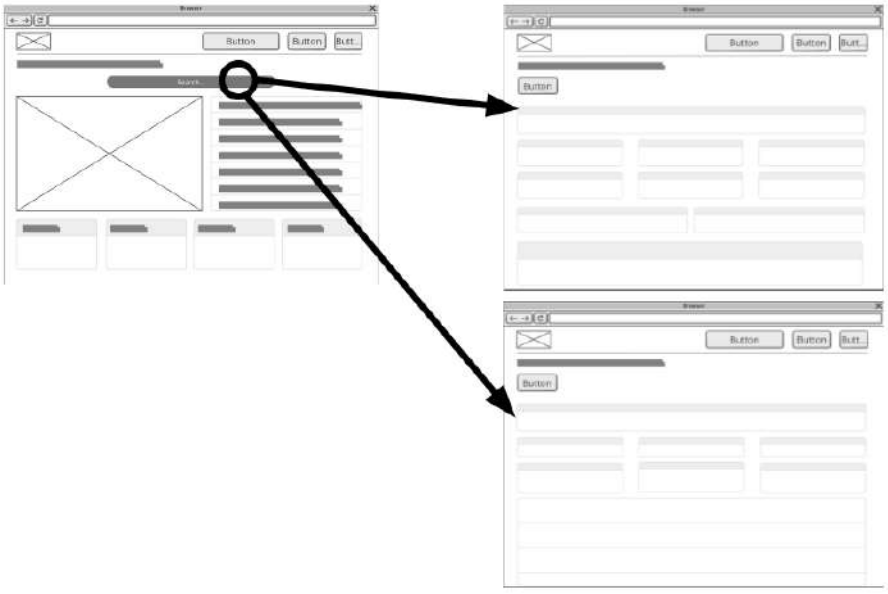

Figura 5.13: Storyboard B.

5.3. Implementación

Se describe, paso a paso, la puesta en marcha del proyecto **«Desarrollo de un Sistema de Trazabilidad en Tiempo Real usando Blockchain y Dispositivos IoT»**. Su contenido permite que cualquier alumno de futuros TFG reproduzca el prototipo —desde la compilación del firmware IoT hasta la interfaz web— y comprenda las decisiones técnicas adoptadas.

Repositorio y objetivo didáctico

El código fuente completo está disponible en:

https://github.com/Juanan151/Trazability

El repositorio se publica de forma abierta para facilitar su reutilización. Además, la memoria posee un apéndice con listados de códigos para poder referenciar y explicar en detalle el porqué de la decisión de ciertas implementaciones.

El énfasis se pone en la **reproducibilidad**: cada sección indica los comandos exactos, scripts y archivos de configuración necesarios para repetir la instalación en un entorno limpio.

5.3.1. Entorno y dependencias

La implementación se realizó en diferentes sistemas operativos y entornos. Los sistemas operativos son Linux (Ubuntu 22.04) y Windows 11; Hyperledger Besu, Arduino IDE [62] y VS Code (Visual Studio Code) [63] funcionan en ambos sistemas y sin la necesidad de pasos adicionales para ninguno de ellos. No se detallan los recursos que tienen estas máquinas ya que no son relevantes para el funcionamiento del sistema.

Sin embargo, se detallan los requisitos de las VMs, que se especifican en la documentación de Besu [64] para un correcto funcionamiento, en la tabla 5.3.

Concepto	Requisito recomendado
Tecnologías de virtualización	Activar **Intel VT-x** y **VT-d** en la BIOS/UEFI.
Hyper-V en Windows	Deshabilitar la característica “Hyper-V” para evitar conflictos con VirtualBox.
Memoria de la VM	**6 GB** de RAM (valor recomendado por la documentación).
Almacenamiento	Disco virtual de **10 GB** mínimo (20 GB recomendado).
Tipo de disco	Formato **VDI** si se necesita compartir el archivo con otras aplicaciones (*alternativa:* VHD).

Tabla. 5.3: Requisitos recomendados para la máquina virtual de los nodos Besu.

Una de las dificultades más grandes encontradas en el desarrollo del proyecto fue los conflictos de dependencias y la cantidad de herramientas, librerías y frameworks que ya no tienen mantenimiento. Debido a la velocidad a la que se investiga y desarrolla la tecnología Blockchain, una de las mayores dificultades es encontrar tutoriales y guías actualizadas para realizar estos sistemas. La única salida posible pasa por recurrir a la documentación oficial la cual no es muy clara por el mismo motivo. Es por ello, que es esencial mencionar las versiones del software en las que se ha desarrollado el sistema de trazabilidad. En la figura 5.14 puede verse un gráfico que ayuda a entender donde han estado involucradas estas herramientas. Además, en la tabla 5.4 pueden consultarse todas las versiones; también se incluyen las versiones del resto de software. También para se aporta la figura .

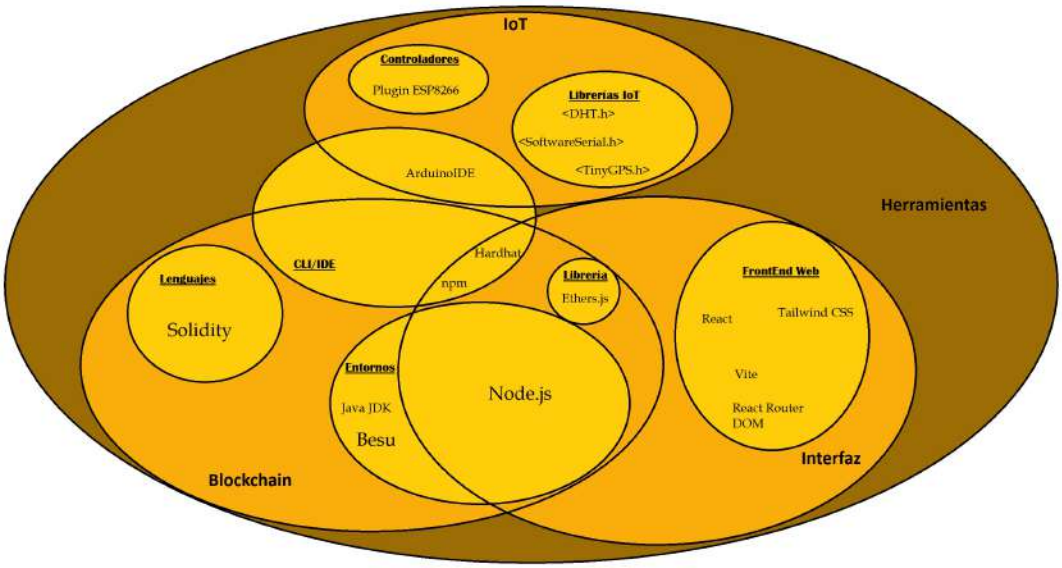

Figura 5.14: Esquema gráfico sobre las herramientas empleadas.

Categoría	Software / Herramienta	Versión	Observaciones
Entorno de ejecución	Besu	25.2	Nodo *ADMIN* y VMs; requiere JDK 21.
	Node.js [65]	22.14	Base para scripts Hardhat y pasarela.
	Java JDK [66]	21.0.1	Necesario para ejecutar Besu.
Herramientas CLI / IDE	npm [67]	10.9.2	Instalación de dependencias JS/TS.
	Hardhat [68]	2.22.19	Compilación, pruebas y despliegue de contratos.
	Arduino IDE	2.3.5	Compilación y flasheo del firmware NodeMCU.
Lenguaje	Solidity [69]	0.8.24	Compilado con *evmVersion = london*.
Librería	Ethers.js [70]	6.14.0	Interacción JS con la red Besu (vía JSON-RPC).
Frontend web	React [71]	19.1.0	Librería base para construir la interfaz modular.
	Vite [72]	6.2.0	Herramienta de desarrollo y empaquetado frontend.
	Tailwind CSS [73]	4.1.0	Plugin de CSS para Vite.
	React Router DOM [74]	7.6.0	Gestión de rutas cliente en aplicaciones SPA.
Librerías IoT	*<DHT.h>*	1.4.4	Lectura de temperatura y humedad (sensor DHT11).
	<SoftwareSerial.h>	1.0	Comunicación UART por software entre módulos.
	<TinyGPS++.h>	1.0.3	Procesamiento de datos GPS desde el shield MKR.
Controladores / Drivers	Plugin ESP8266 (ArduinoIDE) [75]	3.1.2	Instalado vía *package_esp8266com_index.json*; permite seleccionar NodeMCU como placa.

Tabla. 5.4: Entornos, herramientas, lenguajes, drivers y librerías utilizadas durante la implementación.

5.3.2. Configuración de la red

Una vez instalados los clientes de Besu, activada la VPN y configuradas las VMs como se enseña en el posterior Apéndice D. El siguiente paso es configurar lo que será la red *private-permissioned*. Se crea un archivo génesis común y se arranca el nodo con las opciones necesarias para que se ajuste al diseño establecido para la red. Se distinguen entre dos configuraciones distintas: la del nodo **ADMIN** y la de los nodos regulares. Además, para facilitar la configuración de un nodo se puede crear un archivo *config_file.toml* donde se añadan las opciones de despliegue.

En los códigos C.2 y C.3 que corresponden al archivo de configuración del nodo ADMIN y un nodo regular respectivamente, se puede apreciar que hay muchas opciones comunes. Se quiere que la red se comporte tal y como se propone en 5.2.1; para ello, los nodos deben actuar de forma similar.

5.3.2.1. Nodo ADMIN

El nodo ADMIN tiene opciones habilitadas muy importantes que lo habilitan como administrador en este entono de prueba. Incluyen configuraciones específicas para la gestión de permisos, tanto a nivel de API como mediante un archivo de nodos permitidos a entablar conexión con él. Esta API permite añadir o eliminar nodos autorizados, así como gestionar cuentas con privilegios administrativos. Resulta fundamental en redes consorciadas donde los participantes deben cumplir con requisitos de gobernanza.

El archivo *permissions_config.toml* actúa como fuente de verdad para las políticas de permisos. Define explícitamente qué nodos están autorizados a conectarse a la red (mediante enode, IP o DNS), así como qué cuentas están habilitadas para determinadas operaciones. En el sistema, los nodos son los propios usuarios, por lo que no se define ninguna lista de cuentas permitidas.

Estas capacidades adicionales del nodo **ADMIN** responden a su rol como autoridad de gestión dentro de la red. Al centralizar el control de acceso, se facilita la gobernanza y se garantiza una administración coherente del ecosistema *Blockchain* privado.

5.3.2.2. Nodos regulares

Por su parte, los nodos no ADMIN (nodos regulares) incluyen una opción específica que les permite descubrir y conectarse al nodo inicial/ADMIN de la red.

El nodo listado actúa como punto de entrada a la red para los nodos que se inician posteriormente. Dicho nodo corresponde al nodo **ADMIN**.

El uso de *bootnodes* es esencial en redes privadas, ya que no existe una infraestructura pública de descubrimiento de nodos como en redes públicas. Esta opción garantiza que los nuevos nodos puedan conocer al menos un participante de la red al iniciar, desde el cual obtener información sobre otros pares activos mediante el protocolo de descubrimiento.

En una red permisionada, los nodos descubiertos no solo deben existir, sino estar autorizados según la política de permisos definida. Por ello, aunque el nodo ADMIN se configure como *bootnode*, no podrán acceder a la red si no se encuentran en su lista de permitidos.

Todos los nodos regulares deben generar su enode y enviarlo al ADMIN. Este debe añadirlo al archivo permissions_config.toml .

5.3.2.3. Archivo génesis común

Todos los nodos deben tener el mismo archivo génesis para poder formar parte de la misma red. El archivo *genesis.json* (código C.1) define el estado inicial y las reglas de funcionamiento de la red. Se configura para cumplir el diseño propuesto en la subsubsección 5.2.1.

5.3.3. Sistema IoT

El sistema IoT está construido sobre un microcontrolador NodeMCU v3, al cual se conectan tres sensores: un DHT11 para medir temperatura y humedad, un sensor PIR para detectar movimiento y un módulo GPS basado en un **Arduino MKR GPS Shield**. Todos los componentes se alimentan a través del puerto micro-USB del NodeMCU.

El sensor DHT11 y el sensor PIR incluyen internamente las resistencias necesarias para su correcto funcionamiento, por lo que no se requiere electrónica adicional.

El módulo de posicionamiento se implementa mediante un *MKR GPS Shield*, que normalmente está diseñado para montarse sobre placas Arduino de la serie MKR. En este proyecto, se ha adaptado su uso conectando manualmente los pines UART del shield a los pines D1 (TX) y D2 (RX) del NodeMCU, utilizando para ello la biblioteca *SoftwareSerial*. Esta elección permite mantener libre el puerto serial hardware, reservado para la comunicación con el sistema host.

Dado que tanto el NodeMCU como el MKR GPS Shield operan a 3.3 voltios, la conexión directa entre sus líneas de datos es segura sin necesidad de conversores de nivel lógico. El montaje completo forma un sistema ligero y portátil capaz de registrar datos ambientales, eventos de movimiento y coordenadas geográficas, que posteriormente son transmitidos en tiempo real a través del puerto serie.

Se muestra a continuación el montaje del sistema IoT 5.15 a falta de conectarlo por USB a un portátil.

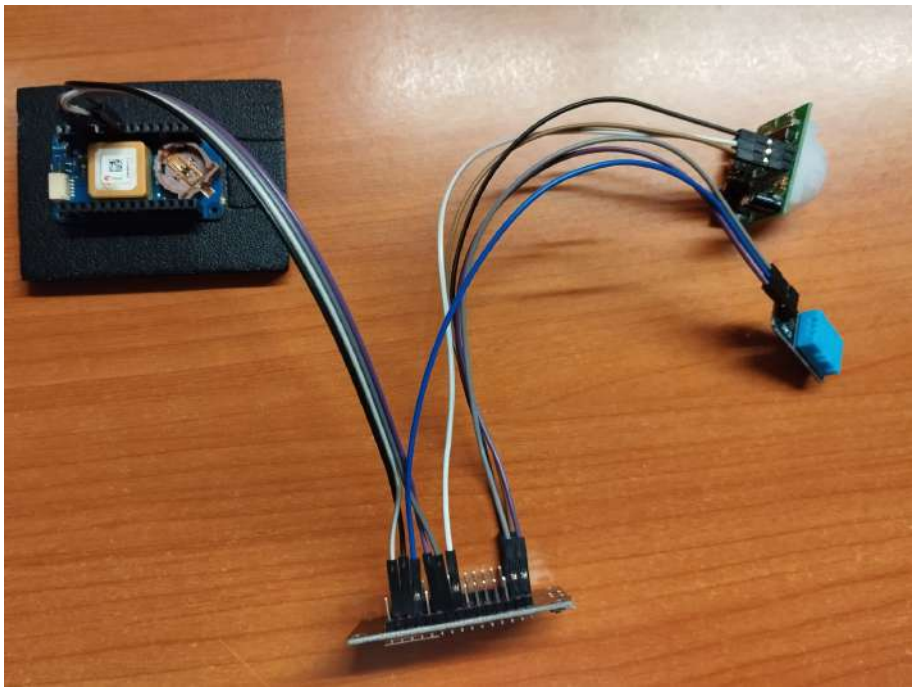

Figura 5.15: Montaje completo del circuito IoT.

5.3.3.1. Firmware del sistema IoT

El firmware desarrollado para el dispositivo IoT tiene como objetivo la lectura y transmisión de datos ambientales, de movimiento y de localización. El código C.4 está escrito en C++ para la plataforma Arduino IDE y ejecutado sobre un microcontrolador NodeMCU v3. El sistema se apoya en tres bibliotecas clave: *DHT.h* para la lectura de sensores de temperatura y humedad, *TinyGPS++.h* para el análisis de tramas provenientes del módulo GPS, y *SoftwareSerial.h* para gestionar la comunicación UART secundaria.

El sensor de movimiento PIR se monitoriza mediante interrupciones, lo que permite registrar eventos sin necesidad de hacer sondeos constantes.

5.3.4. Smart Contract

Para el despliegue del contrato mostrado en el Listado C.6, se utilizó el framework HardHat, una herramienta robusta y ampliamente adoptada en el ecosistema Ethereum. Su flexibilidad permite compilar, desplegar y testear contratos en redes personalizadas.

La configuración se realiza en el archivo *hardhat.config.js*, mostrado en el Listado C.5, donde se especifica tanto la versión del compilador Solidity como los parámetros de la red Besu.

En esta configuración se especifica que el contrato se compilará utilizando Solidity 0.8.24, compatible con la versión *london* de la EVM, activada desde el bloque génesis de la red. Se define una red personalizada llamada *besu*, con la URL del nodo local (*http://localhost:8545*) y el *chainId* correspondiente (*1982*). Además, se proporciona una cuenta firmante utilizando su clave privada en formato hexadecimal, que coincide con una cuenta autorizada en el archivo *permissions_config.json*.

El contrato *indexed_id_product* fue diseñado con un enfoque minimalista, ya que su única funcionalidad es la emisión de eventos. Este diseño responde a la necesidad de mantener la red privada ligera, evitando almacenamiento en cadena y minimizando el coste de gas, y teniendo en mente la escalabilidad del proyecto. El contrato registra interacciones relevantes mediante eventos, lo cual permite que la interfaz web los recupere de forma eficiente y segura.

Este enfoque orientado a eventos permite al sistema mantener un historial confia-

ble, económico y fácilmente indexable, especialmente útil en entornos de trazabilidad, donde el análisis posterior se realiza fuera de la cadena.

5.3.4.1. Script de despliegue

El despliegue del contrato inteligente se realiza mediante un script manual implementado con *ethers.js* dentro del entorno de trabajo que proporciona Hardhat. Este enfoque, lejos de los sistemas de despliegue automatizado basados en plantillas o migraciones, permite un control completo sobre los parámetros de la transacción y una comprensión detallada del proceso de interacción con la red *Blockchain*.

Hardhat actúa como capa intermedia entre el contrato fuente en Solidity y la red permisionada. Se encarga de compilar el contrato con la versión especificada del compilador, generar los artefactos binarios (*bytecode*) y las interfaces ABI necesarias para su despliegue. Además, simplifica el acceso a cuentas firmantes y la conexión con nodos personalizados como los desplegados con Besu.

El script completo puede verse en el Listado C.7.

5.3.5. *Gateway*: Pasarela entre la Blockchain y el sistema IoT

Para enlazar el sistema embebido con la red *Blockchain*, se ha desarrollado un componente software intermedio o *gateway*, implementado en Node.js. Su propósito es actuar como puente entre los datos generados por el microcontrolador (en este caso, enviados por puerto serie) y el contrato inteligente desplegado en la red Besu. El código completo puede consultarse en el Listado C.8.

En paralelo, se establece una conexión con la red *Blockchain* mediante *ethers.js*, una biblioteca que ofrece una API para interactuar con contratos Ethereum desde JavaScript. La elección de esta herramienta, frente a alternativas como *web3.js* [76], se justifica por su mejor compatibilidad con Hardhat y su arquitectura más flexible. Además, *ethers.js* permite construir y firmar transacciones con granularidad.

5.3.6. Interfaz Web

La interfaz web fue desarrollada como punto final del sistema de trazabilidad, permitiendo consultar desde un navegador los datos almacenados en la red *Blockchain*

permisionada.

Para su implementación se utilizó el framework *React* junto con *Vite* como herramienta de empaquetado y el *plugin Tailwind CSS* para la estética. Esta elección responde a la necesidad de construir una aplicación rápida, modular y fácilmente mantenible. *React* permite una clara separación de responsabilidades a través de componentes reutilizables, mientras que Vite ofrece tiempos de arranque prácticamente instantáneos y recarga en caliente durante el desarrollo.

La aplicación está estructurada en dos bloques principales: componentes reutilizables (*components*) y páginas completas (*pages*). Dentro de estas páginas destacan tres rutas clave: el explorador de bloques, el explorador de transacciones y la sección de trazabilidad por producto. En cada una de ellas se hace uso del archivo *rpcClient.js* (GitHub (rpcClients.js)) para establecer conexión directa con el nodo Besu vía JSON-RPC utilizando *ethers.js*.

El formulario principal permite al usuario introducir un identificador de producto o un hash de bloque o transacción. A partir de ahí, la aplicación construye un filtro indexado sobre los eventos emitidos por el contrato *indexed_id_product*, recuperando los datos registrados por el gateway IoT. Estos datos, codificados como cadenas CSV en los eventos, se transforman en objetos estructurados y se visualizan a través de componentes como tablas dinámicas, gráficos de evolución y mapas basados en *Leaflet.js*.

La presentación general está contenida en el componente *MainLayout* GitHub (MainLayout.jsx), que encapsula tanto el menú de navegación (*Tabs* GitHub (Tabs.jsx)) como el estilo común entre vistas. El resultado es una interfaz eficiente, clara y perfectamente integrada con la experiencia del sistema de trazabilidad.

5.3.6.1. Vista principal del sistema de trazabilidad

La pantalla principal del sistema, mostrada en la Figura 5.16, está implementada en el componente *TraceabilityExplorer.jsx* GitHub (TraceabilityExplorer.jsx).

5.3.6.2. Vista de detalles del producto rastreado

La vista de detalle, representada en la Figura 5.17, permite explorar la evolución temporal de un producto concreto, desde sus parámetros físicos hasta su localización

GPS. Esta interfaz está implementada íntegramente en el componente *ProductDetail.jsx* GitHub (ProductDetail.jsx).

5.3.6.3. Vista del explorador de bloques

La pantalla mostrada en la Figura 5.19 corresponde a la vista de exploración de bloques, implementada en el componente *BlockExplorer.jsx* GitHub (BlockExplorer.jsx).

5.3.6.4. Vista de detalle de bloque

La vista ilustrada en la Figura 5.20 muestra el detalle completo de un bloque específico, permitiendo inspeccionar tanto su metadata como las transacciones que contiene, implementado en BlockDetail.jsx GitHub (BlockDetail.jsx).

5.3.6.5. Vista del explorador de transacciones

La vista presentada en la Figura 5.21 corresponde al explorador de transacciones de la red. Esta interfaz permite recorrer, consultar y acceder al detalle de las transacciones emitidas, tanto hacia contratos como entre cuentas. Está gestionada por el componente *TransactionExplorer.jsx* GitHub (TransactionExplorer.jsx).

5.3.6.6. Vista de detalle de transacción

La pantalla ilustrada en la Figura 5.22 corresponde a la vista de detalle de una transacción específica. Está implementada por el componente *TransactionDetail.jsx* GitHub (TransactionDetail.jsx).

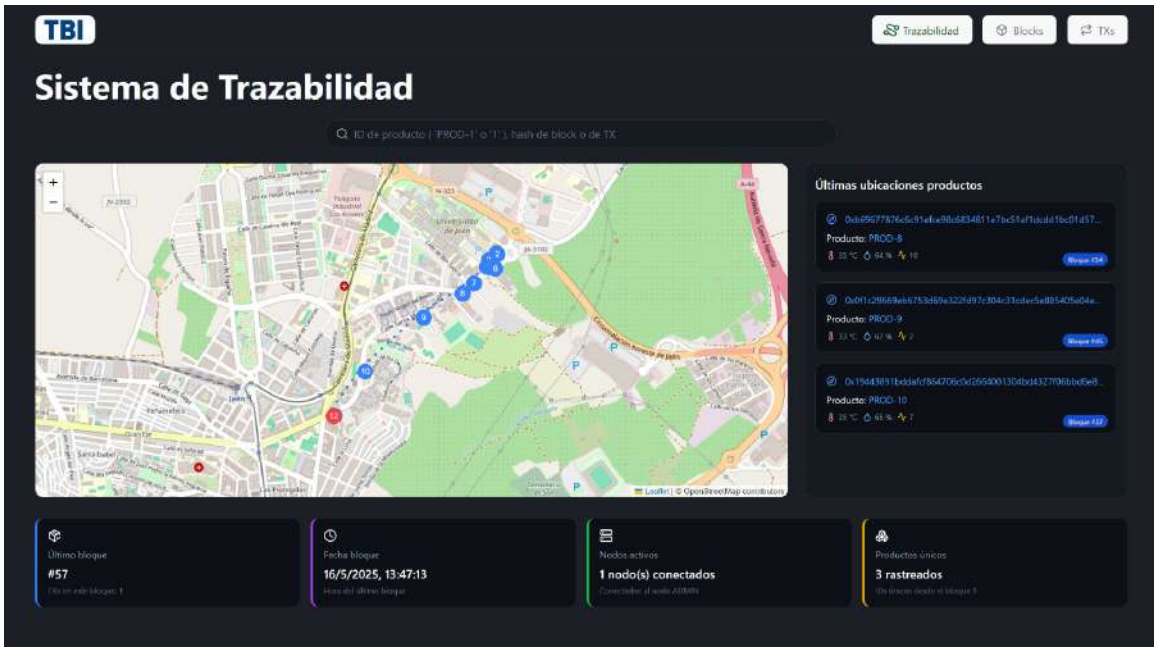

Figura 5.16: Vista general del sistema de trazabilidad. Se muestran los productos rastreados, su ubicación, y estadísticas extraídas directamente desde la *Blockchain*.

Figura 5.17: Vista de detalle de un producto. Se visualiza el recorrido GPS, métricas medias y evolución temporal.

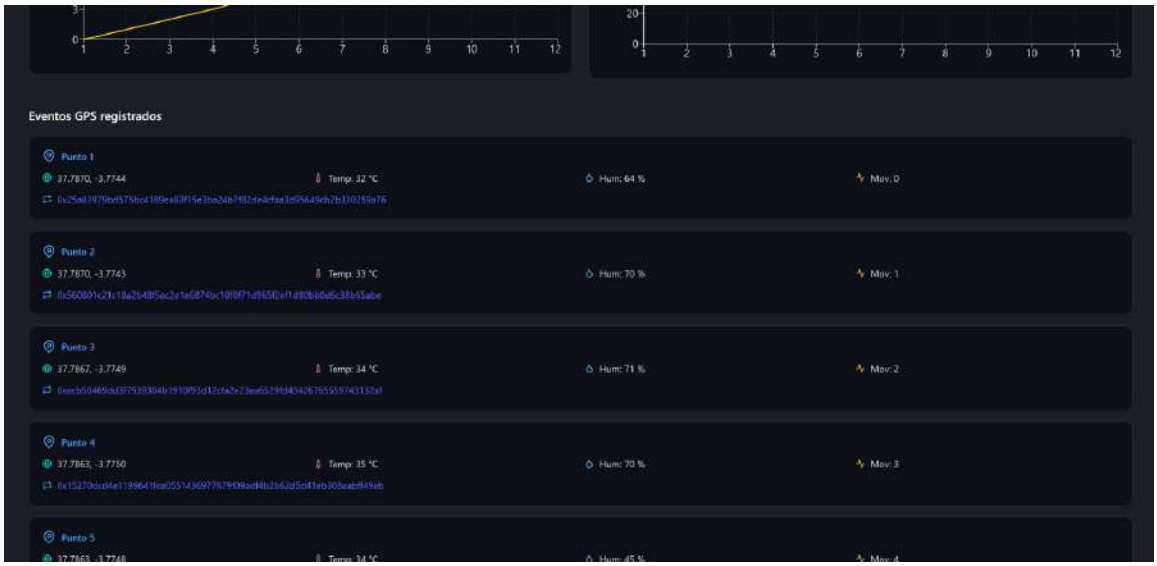

Figura 5.18: Eventos GPS registrados del producto. Cada evento contiene información sensórica y hash de la transacción en *Blockchain*.

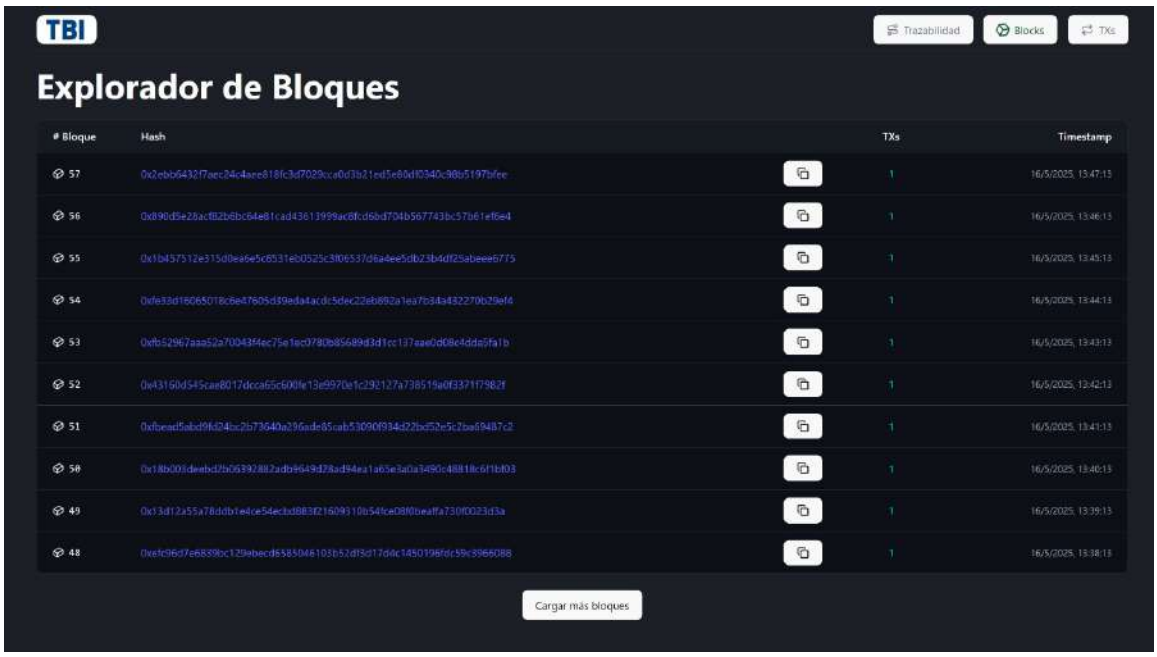

Figura 5.19: Vista del explorador de bloques. Se muestran los últimos bloques minados, número de transacciones y timestamp.

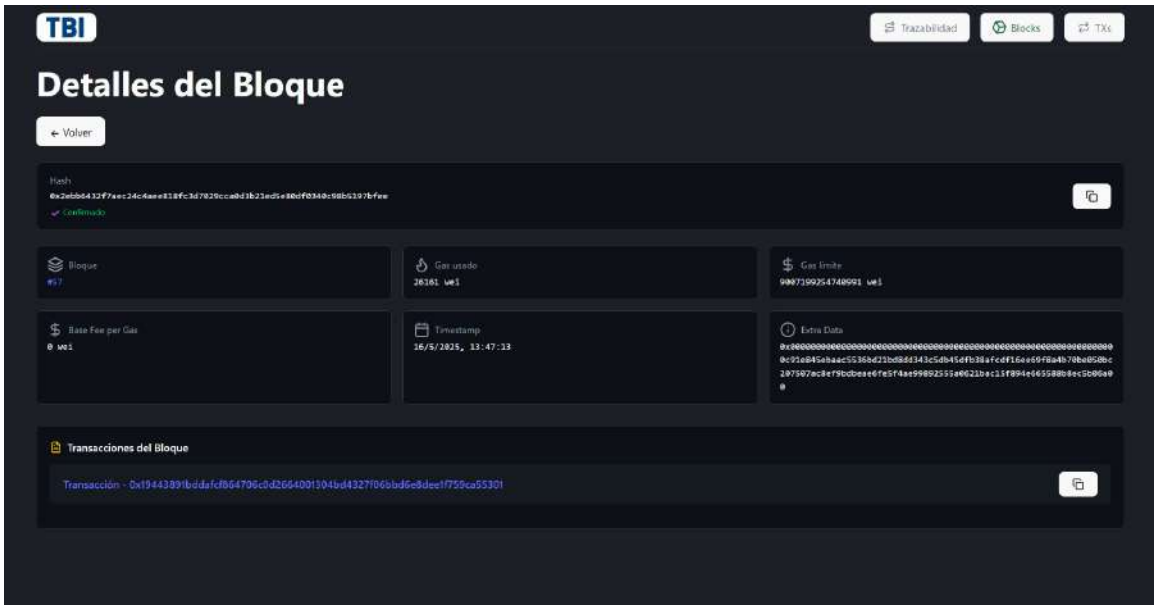

Figura 5.20: Vista de detalle de un bloque. Se incluyen gas usado, límite, timestamp, extra data y listado de transacciones.

Figura 5.21: Vista del explorador de transacciones. Permite navegar por las transacciones de la red permisionada, ordenadas por bloques recientes.

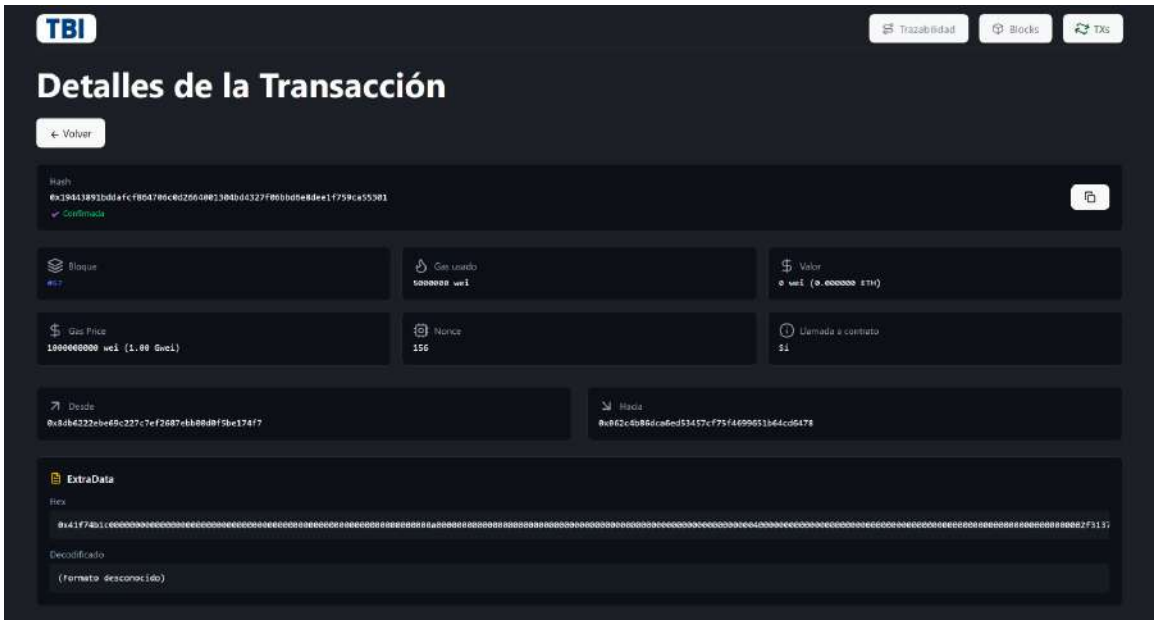

Figura 5.22: Vista detallada de una transacción. Se muestran datos como gas, valor, nonce, direcciones y datos adjuntos codificados.

5.4. Pruebas

Con el objetivo de verificar el correcto funcionamiento del sistema implementado se llevaron acabo una serie de pruebas para cada uno de los subsistemas.

5.4.1. Enfoque de las pruebas

Desde el punto de vista metodológico, la mayoría de las pruebas realizadas en este trabajo se encuadran dentro del enfoque de pruebas de caja negra, ya que se evalúa el comportamiento observable del sistema sin acceder ni analizar directamente su lógica interna. Se comprueba, por ejemplo, que al invocar una función del contrato inteligente se emite el evento esperado, o que los datos transmitidos por el microcontrolador son correctamente registrados y recuperados, sin inspeccionar cómo se procesan internamente.

Algunas pruebas pueden considerarse de tipo caja gris, ya que, aunque se accede al sistema a través de sus interfaces públicas (scripts, API JSON-RPC), se aprovecha el conocimiento del diseño interno —como el uso de eventos indexados en el contrato inteligente— para construir filtros eficientes y recuperar información.

5.4.2. Pruebas de verificación del funcionamiento de la red

Una vez levantada la red permisionada y con todos los nodos en ejecución, se realizaron tres pruebas básicas para comprobar su correcto funcionamiento: una transacción de prueba, la consulta del balance de una cuenta y la verificación de conexiones mediante la API.

1. Transacción de prueba

Se utilizó un script en JavaScript con la librería *ethers.js* (v6) ejecutado desde el entorno Hardhat para realizar una transferencia de 1 ETH entre dos cuentas de la red, código C.10.

El script permite verificar que las transacciones son aceptadas y procesadas correctamente por el nodo, el sistema de gas funciona, incluso si no hay costes reales

(gracias a *zeroBaseFee*), y los bloques se generan correctamente y contienen transacciones.

2. Consulta de balance

Para comprobar que el estado de la red refleja correctamente las operaciones realizadas, se desarrolló un segundo script que consulta el balance de una cuenta concreta, código C.11.

Esta prueba permite verificar la sincronización de estados entre nodos funciona correctamente y el saldo de una cuenta se actualiza tras una transacción.

3. Verificación de nodos conectados

Por último, se utilizó la API JSON-RPC expuesta por el nodo Besu para verificar que los nodos están efectivamente conectados entre sí. En el entorno Windows, se realizó la siguiente petición mediante PowerShell:

```
Invoke-WebRequest -Uri "http://localhost:8545" -Method Post `
  -Body '{"jsonrpc":"2.0","method":"net_peerCount","params":[],"id":1}' `
  -ContentType "application/json"
```

Esta petición devuelve el número de pares conectados al nodo en ejecución. Es útil para confirmar que el nodo ADMIN está recibiendo conexiones entrantes desde el resto de nodos, las direcciones configuradas en *bootnodes* y en el archivo de permisos son correctas y la red está efectivamente operativa como sistema distribuido.

Resultado

Las tres pruebas se completaron satisfactoriamente, lo que confirma que:

- La red fue correctamente iniciada y configurada.
- Los nodos están conectados entre sí y pueden intercambiar información.
- Las cuentas preasignadas y los mecanismos de consenso están funcionando como se esperaba.

5.4.3. Prueba de recepción de datos vía puerto serie

Para comprobar el correcto funcionamiento del firmware embebido descrito, se desarrolló una prueba de recepción de datos en el sistema host mediante un pequeño script en Node.js. Este código utiliza la biblioteca *serialport*, ampliamente utilizada para leer flujos de datos procedentes de dispositivos conectados a través de puertos seriales (USB en este caso).

El script C.13 escucha continuamente la información enviada por el microcontrolador a través del puerto conectado a una velocidad de 9600 baudios, que coincide con la configurada en el firmware.

El uso del *ReadlineParser* permite interpretar correctamente cada línea CSV enviada desde el firmware como una unidad de datos.

Durante la prueba, los datos generados por el microcontrolador fueron recibidos en tiempo real, en el mismo formato definido en el firmware (temperatura, humedad, latitud, longitud, número de eventos de movimiento). Esto confirmó que tanto la comunicación serial como el empaquetado de datos funcionan correctamente de extremo a extremo.

5.4.4. Prueba de funcionamiento del contrato desplegado

Una vez desplegado el contrato inteligente en la red permisionada, se procedió a realizar una prueba de funcionamiento para verificar que la lógica del contrato y la conexión con la red eran correctas. La prueba consistió en invocar la función *indexed_id_product_function()* del contrato, emitiendo un evento con datos arbitrarios. El script utilizado puede verse en el Listado C.12.

Este script se conecta al contrato ya desplegado utilizando su dirección y el ABI generado por Hardhat. La cuenta firmante se obtiene de la configuración de red y se utiliza para autorizar la transacción. La función invocada genera un evento que asocia un identificador de producto con una cadena de texto, simulando un caso de uso real como un punto en un proceso de trazabilidad.

Se configuraron manualmente los parámetros de gas y el *chainId* para asegurar compatibilidad mencionada anteriormente. La inclusión del *nonce* explícito garantiza que la transacción se ordene correctamente en el pool, evitando errores de repetición o conflicto.

La ejecución exitosa de este script permitió comprobar que el contrato puede ser invocado desde el exterior con los parámetros esperados y que el evento emitido queda registrado.

5.4.5. Recuperación de eventos

Una vez que los datos del sistema IoT han sido correctamente transmitidos a la red *Blockchain* mediante el gateway, se implementó una prueba para verificar que la información registrada es recuperable y estructurable desde la capa de aplicación. Esta validación se realiza recuperando los eventos emitidos por el contrato inteligente, específicamente aquellos generados por la función *indexed_id_product_function()*.

El script C.9, desarrollado en Node.js con *ethers.js*, establece conexión con el contrato a través de su dirección y una definición mínima del ABI, suficiente para filtrar eventos y acceder a sus argumentos. Esta estrategia permite simplificar el cliente al máximo, evitando la necesidad de importar todo el artefacto de compilación generado por Hardhat.

Para recuperar los datos, se utiliza el sistema de filtros que proporciona *ethers.js*. Se construye un filtro sobre el evento indexado utilizando el identificador de producto como clave de búsqueda. Esta posibilidad es consecuencia directa del modificador *indexed* aplicado en la declaración del evento dentro del contrato inteligente, lo que permite a la *Blockchain* construir índices eficientes para la consulta.

La función *queryFilter()* consulta los registros históricos emitidos por el contrato que coinciden con el filtro indicado, devolviendo una lista de eventos en orden cronológico. Cada uno de estos eventos contiene como argumento una cadena de texto estructurada en formato CSV, generada por el gateway al recibir datos del sistema IoT.

El script divide esa cadena y reconstruye los campos originales: marca temporal, temperatura, humedad, latitud, longitud y número de detecciones de movimiento. Los datos se almacenan en un array de objetos estructurados que pueden ser utilizados para visualización, análisis posterior o exportación. Esta reconstrucción demuestra que el modelo basado en eventos es suficiente para conservar la trazabilidad de los datos, incluso sin almacenamiento directo en la *Blockchain*.

La ejecución del script con un identificador válido devuelve un conjunto de eventos correspondientes al recorrido completo de un producto. De este modo se confirma que el flujo de datos —desde el sistema IoT, pasando por el gateway, hasta el contrato

inteligente y su posterior recuperación— funciona correctamente y permite garantizar integridad, orden y trazabilidad de la información registrada.

5.4.6. Prueba en entorno real

Tras la validación funcional de todos los componentes del sistema en entorno controlado —incluyendo el firmware embebido, la comunicación con el nodo, el contrato inteligente y la interacción con la *Blockchain*— se procedió a realizar una prueba de campo en condiciones reales, con el objetivo de evaluar la fiabilidad del sistema en movilidad y en un entorno físico no supervisado.

La prueba consistió en montarse en el interior de un vehículo y activar la recolección de datos durante un recorrido urbano. Se siguieron los mismos pasos indicados en el apéndice E.

La prueba demostró que el sistema es capaz de operar de forma robusta y estable en movimiento, integrando dispositivos físicos, red privada *Blockchain* y procesamiento distribuido en un flujo de datos continuo y coherente.

5.4.6.1. Pruebas preiniciales

Antes de adoptar la configuración final basada en NodeMCU y sensores ambientales, se realizaron diversas pruebas funcionales con otro conjunto de hardware compuesto por un **Arduino MKR Zero** y un **Arduino MKR GPS Shield**. Esta fase experimental tenía como objetivo verificar el comportamiento del sistema de trazabilidad *Blockchain* utilizando datos geoespaciales reales, aprovechando el soporte completo del GPS integrado.

El sistema IoT utilizado durante estas pruebas recolectaba y transmitía datos como latitud, longitud, velocidad, número de satélites y altitud. Estos valores eran procesados por el microcontrolador y enviados en formato CSV al portátil mediante una conexión serie. El *gateway* Node.js —idéntico al empleado en la versión final— leía estos datos y los convertía en eventos que eran emitidos al contrato inteligente desplegado en la red basada en Hyperledger Besu. Tanto el contrato como la configuración de red y permisos se mantuvieron invariables a lo largo de todas las pruebas.

La interfaz web desarrollada para esta fase difería de la final, ya que estaba centrada en representar los datos recibidos en tiempo real de forma visual. A través de

distintas vistas se podía monitorizar el estado del sistema: desde mapas con coordenadas en vivo hasta visualización de la velocidad, altura y datos técnicos asociados al GPS. Esta interfaz permitía comprobar que los datos llegaban correctamente, que los eventos se emitían como se esperaba, y que los filtros por identificador funcionaban de forma adecuada.

En las Figuras 5.23 a 5.28 se muestran capturas de pantalla representativas de esta interfaz web en su fase preinicial.

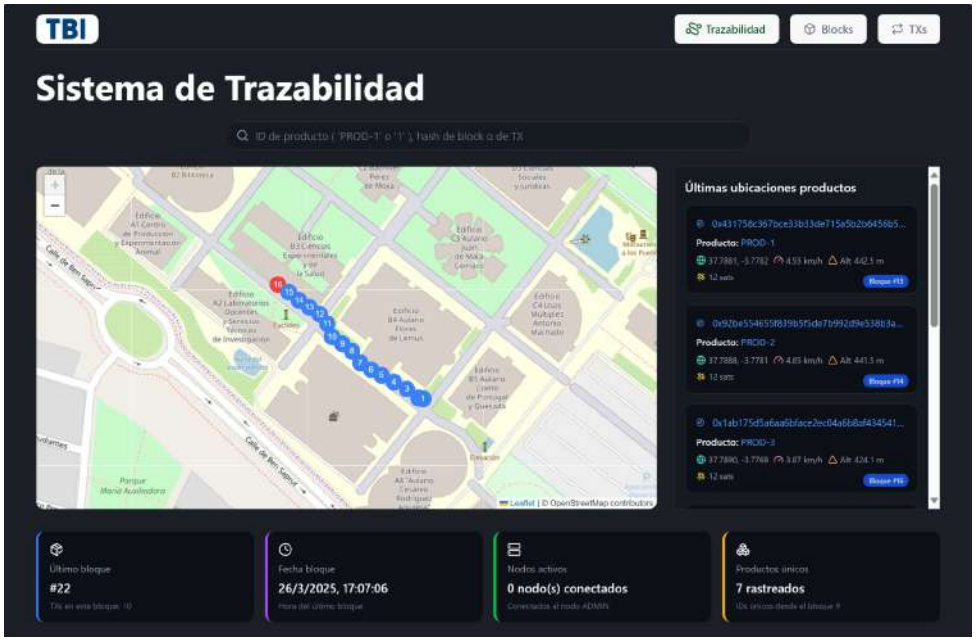

Figura 5.23: Versión anterior 5.16.

Figura 5.24: Versión anterior 5.17 y 5.18.

Figura 5.25: Versión anterior 5.19.

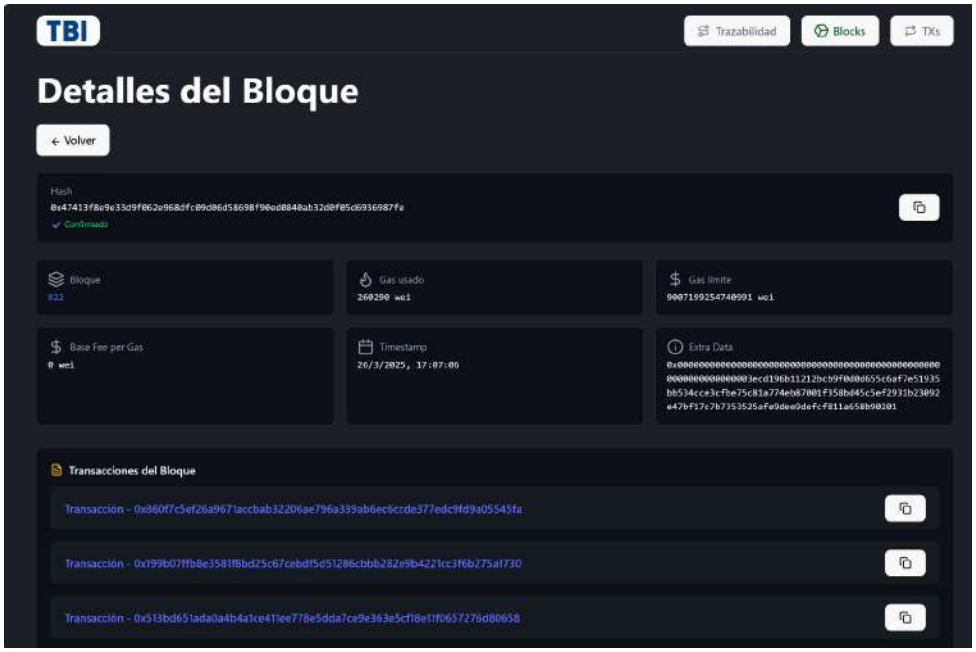

Figura 5.26: Versión anterior 5.20.

Figura 5.27: Versión anterior 5.21.

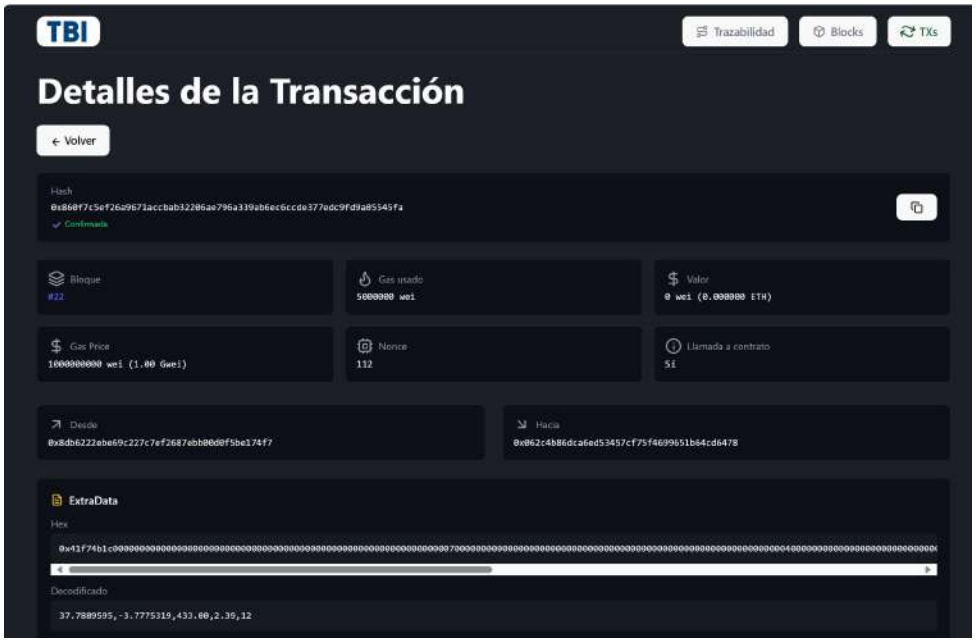

Figura 5.28: Versión anterior 5.22.

Estas pruebas permitieron validar de forma temprana la arquitectura del sistema, confirmando que tanto el contrato como la infraestructura *Blockchain* eran funcionales incluso con diferentes configuraciones IoT e interfaces web. También permitieron detectar ciertos límites, como la dependencia del GPS a cielo abierto, la sensibilidad a la velocidad de lectura del puerto serie y la necesidad de homogeneizar los formatos de datos en el gateway. A partir de esta experiencia, se consolidaron las decisiones de diseño que dieron lugar a la versión final del sistema.

5.4.7. Validación y discusión

Una vez completadas todas las pruebas funcionales, tanto en entorno controlado como en condiciones reales, se procede a analizar el comportamiento general del sistema y a discutir su validez como solución de trazabilidad distribuida basada en IoT y *Blockchain*. Esta sección recoge los aprendizajes derivados de todo el proceso de desarrollo, prueba y validación, abordando tanto los aciertos técnicos como las limitaciones detectadas.

5.4.7.1. Comprobación integral y evaluación de la arquitectura

El sistema ha demostrado ser funcional en todas las fases del ciclo operativo: desde la captura de datos físicos hasta su registro estructurado en una red *Blockchain* permisionada. Durante las pruebas de campo, se validó que el sistema IoT era capaz de recolectar correctamente datos ambientales y geográficos (temperatura, humedad, movimiento y coordenadas GPS), transmitirlos al nodo por puerto serie, procesarlos mediante un gateway intermedio y finalmente generar eventos en un contrato inteligente desplegado sobre una red Besu privada.

Este comportamiento no fue puntual: a lo largo del proceso se llevaron a cabo múltiples pruebas preiniciales con distintas configuraciones de hardware. Se utilizaron, por ejemplo, un Arduino MKR Zero junto a un MKR GPS Shield para recolectar información geoespacial básica como latitud, longitud, velocidad y altitud. Aunque los sensores y la interfaz web eran distintos, la infraestructura de red, el contrato y el gateway fueron los mismos. Esto permitió comprobar que la arquitectura central era robusta, y que los módulos podían intercambiarse sin alterar el funcionamiento general.

Este tipo de modularidad es una de las principales fortalezas del sistema. La separación entre el firmware, el gateway, el contrato y la red permite sustituir o actualizar cualquiera de los componentes sin reestructurar el sistema completo. El gateway, por ejemplo, fue reutilizado sin cambios entre pruebas con distintos formatos de entrada. Del mismo modo, el contrato inteligente se mantuvo inalterado en todos los ensayos, demostrando que su diseño era lo suficientemente general como para soportar distintos tipos de datos y nodos emisores.

También se comprobó que la infraestructura *Blockchain* soportaba adecuadamente una operación continua, sin presentar bloqueos, errores de sincronización ni conflictos de permisos. El uso de una red permisionada permitió un control total sobre la topología de nodos y facilitó la depuración durante el desarrollo, sin las fricciones propias de

redes públicas (latencia, costes de gas, congestión, etc.).

5.4.7.2. Justificación del enfoque basado en eventos

Una de las decisiones más relevantes a nivel de diseño fue no almacenar información dentro del contrato inteligente, sino utilizar eventos como mecanismo de registro. Esta elección técnica responde a una combinación de factores de eficiencia, compatibilidad y adecuación al caso de uso.

Desde el punto de vista del rendimiento, emitir un evento es una operación mucho más ligera que almacenar datos on-chain. Esto se traduce en un menor uso de gas por transacción, menor consumo de almacenamiento en los nodos de la red y mayor velocidad de ejecución. Aunque en el contexto de una red privada como Besu no existen incentivos económicos, el coste computacional sigue siendo un factor a considerar, especialmente cuando se plantean sistemas escalables con múltiples nodos IoT.

A nivel funcional, el modelo orientado a eventos se adapta perfectamente a sistemas de trazabilidad. En estos escenarios, lo importante no es mantener un "estado actual", sino registrar un historial confiable, cronológicamente ordenado e inmutable. Cada evento emitido en la red representa una acción concreta asociada a un identificador, y puede ser recuperado más adelante mediante filtros eficientes. El uso de campos `indexed` en los eventos permite además segmentar fácilmente la información desde el cliente, sin necesidad de recorrer el historial completo de bloques.

Este enfoque también simplifica la lógica del contrato. Al no requerirse almacenamiento ni validaciones complejas, el código es más compacto, menos propenso a errores y más fácil de auditar. Esta simplicidad se alinea con uno de los principios clave del sistema: mantener la *Blockchain* como una capa de verificación y registro, delegando el análisis y el procesamiento de los datos a capas superiores (por ejemplo, el frontend o el backend del sistema).

5.4.7.3. Limitaciones observadas

A pesar de sus puntos fuertes, el sistema también presenta algunas limitaciones. La principal es su dependencia de la calidad de la señal GPS, lo que restringe su uso en espacios cerrados o zonas urbanas con mala cobertura satelital. Además, el uso del puerto serie como canal de comunicación entre el dispositivo IoT y el nodo no escala fácilmente a múltiples nodos ni a distancias largas. Aunque es útil para entornos de

prueba y controlados, no resulta viable para escenarios distribuidos.

Por otro lado, el hecho de que el contrato no almacene datos implica que la recuperación del historial debe realizarse fuera de la *Blockchain*, mediante herramientas específicas que consulten los logs. Aunque esto es razonable para muchos sistemas de trazabilidad, puede no ser suficiente si se requiere mantener una representación permanente del estado de cada producto dentro de la propia red.

5.4.7.4. Conclusión de la validación

En conjunto, las pruebas realizadas, tanto preiniciales como en entorno real, confirman que el sistema propuesto cumple con los objetivos funcionales definidos. La arquitectura modular, el diseño ligero del contrato y el enfoque orientado a eventos permiten construir una solución de trazabilidad basada en *Blockchain* que es eficiente, replicable y adaptable a distintos contextos. Además, su validación progresiva en distintas fases del desarrollo refuerza la solidez técnica de la propuesta y su aplicabilidad práctica en escenarios reales.

5.5. Despliegue

Por último, terminando el ciclo de software esta la fase de despliegue donde se comentará el estado actual de despliegue del sistema y posibles mejoras a futuro.

Actualmente, el sistema está desplegado localmente; es totalmente funcional y probado. El despliegue local se ha llevado acabo con las herramientas de desarrollo integradas en Vite.

En un futuro, puede realizarse un despliegue auto-hosteado de la interfaz web que tenga conexión local a un nodo de la red. Otra de las opciones es desplegar la interfaz en la nube y configurar un nodo para que sea accesible desde el exterior (un puente en el NAT que lo protege, etc).

Capítulo 6

CONCLUSIONES Y POSIBLES MEJORAS

Cuando se planteó este Trabajo Fin de Grado la cuestión no giraba en torno a la mera conectividad de sensores, ni a la exuberancia mediática de la *Blockchain*; el núcleo del reto era más simple y, a la vez, más exigente: ¿es posible trasladar al mundo físico la promesa de inmutabilidad y transparencia propia de los registros distribuidos, sin incurrir en la complejidad y el coste de las redes públicas ni renunciar a la simplicidad? En otras palabras, el proyecto nació del deseo de comprobar si un conjunto modesto de dispositivos IoT podía generar datos que viajaran por un trayecto tecnológicamente coherente—desde la lectura del sensor hasta un *ledger* permisionado—y llegar intactos, verificables y listos para ser auditados. Bajo esa premisa se diseñó un sistema cuyo terreno de pruebas es la trazabilidad, porque allí donde un producto físico cambia de manos o de entorno, la confianza se vuelve frágil y valiosa a partes iguales. El propósito inicial fue, por tanto, validar que la cadena de bloques puede actuar como notario de eventos físicos sin la sobrecarga de mecanizar tarifas, minería o economías de tokens; y hacerlo con una arquitectura lo suficientemente ligera como para que un microcontrolador alimentado por batería participe en ese mismo pacto de confianza. Todo lo que se narra en los capítulos precedentes—desde la elección de Hyperledger Besu hasta la emisión de eventos indexados—responde a esa pregunta germinal que, una vez resuelta de forma práctica, justifica la relevancia y la pertinencia del trabajo realizado.

6.1. Aportación esencial

La principal aportación de este proyecto reside en haber convertido un conjunto heterogéneo de tecnologías—sensores, firmware, pasarela intermedia, contrato inteligente y red permisionada—en un flujo coherente de confianza. Cada pieza, por sí sola, pertenece a dominios tecnológicos muy dispares; sin embargo, orquestadas bajo una lógica de eventos, encajan como engranajes de una misma transmisión. El firmware embebido no se limita a leer sensores: prepara el dato para que sea comprensible fuera del microcontrolador. El *gateway* no es un mero proceso de agregación: traza un puente entre el mundo analógico y las estructuras de la *Blockchain*. El contrato inteligente, lejos de almacenar estados voluminosos, certifica la existencia de cada evento y lo liga a un identificador indexado, permitiendo a cualquier observador reconstruir la secuencia sin ambigüedades. Por su parte, la red Hyperledger Besu aporta gobernanza y control—algo crucial cuando la transparencia debe convivir con la confidencialidad operativa—mientras que la interfaz web traduce la abstracción de los hashes en un relato visual interpretable por el usuario final.

Lo innovador no radica en la novedad individual de cada componente, sino en haber demostrado que pueden convivir pareciendo parte de un mismo ser.

6.2. Aprendizajes transversales

Más que un prototipo funcional, este proyecto se ha convertido en un curso intensivo sobre los cimientos de la tecnología *Blockchain*. Partiendo de la lectura en frío de los primeros *white-papers*, el trabajo ha empujado a aprender cómo se calculan los *hashes* SHA-3, por qué las firmas ECDSA otorgan no repudio, de qué forma un árbol de Merkle comprime miles de transacciones en un único resumen y por qué un bloque sin *parentHash* válido es un bloque huérfano. Comprender estos mecanismos en su grado más elemental permitió, luego, valorar las implicaciones de elegir un consenso u otro: pasar de la prueba de trabajo al PoA de Clique no es solo reducir consumo energético, es asumir un modelo de confianza explícita y aceptar la responsabilidad de gestionar a los firmantes.

Trabajar con Hyperledger Besu no hizo sino poner carne a esos conceptos. Configurar `chainId`, `londonBlock`, listas de permisos y perfiles "`PRIVATE`" mostró que una red puede ser tan abierta o tan cerrada como dicte su gobernanza, y que seguridad y eficiencia se negocian en un espacio de parámetros continuos. Al mismo tiempo,

depurar *nonces*, estimar gas y observar cómo la EVM ejecuta opcodes enseñó que cada transacción es, en realidad, una pequeña máquina de estados que se replica de manera determinista en todos los nodos.

El contrato inteligente diseñado para este proyecto—mínimo en líneas de código pero elocuente en su uso de eventos—sirvió como puente para interiorizar la dualidad *on-chain/off-chain*. Emitir un evento indexado, ver cómo queda grabado en los *logs* y luego recuperarlo con un filtro de *topics* ancló la idea de que el auténtico valor de la *Blockchain* no es almacenar datos a granel, sino dotar de veracidad a los hechos que se narran. El evento se convierte, así, en una firma de tiempo distribuida que cualquier parte puede verificar sin pedir permiso a un intermediario.

A nivel de herramientas, el ecosistema resultó ser tan importante como la teoría. Hardhat, `ethers.js` y las utilidades de Besu demostraron que la comunidad Ethereum ha destilado la complejidad en flujos de trabajo accesibles: compilar, desplegar, probar, migrar; todo queda a unos cuantos comandos, siempre que se comprenda lo que ocurre bajo la superficie.

En suma, el aprendizaje más valioso tras meses de experimentación es que la *Blockchain* no es un software monolítico, sino un espacio de diseño donde criptografía, teoría de juegos y sistemas distribuidos convergen. Haber transitado ese espacio—del sensor al *ledger*—ha permitido no solo construir un sistema operativo, sino también adquirir una comprensión integral que será transferible a cualquier variante futura de tecnologías de registro distribuido.

Más allá de la cadena de bloques, el trabajo ha servido de campo de pruebas para dominar el ecosistema que la rodea: en la capa física, programar un NodeMCU v3 obligó a pulir interrupciones, gestión de memoria y temporización precisa; en la pasarela, la combinación de `serialport` y `ethers.js` enseñó a orquestar flujos asíncronos, firmar transacciones y depurar estados intermedios; en infraestructura, levantar túneles VPN y automatizar nodos con scripts evidenció que la gobernanza de la red es tan crítica como el código que corre sobre ella; y, por último, en la interfaz, convertir *hashes* y coordenadas en visualizaciones comprensibles recordó que, sin una puerta de entrada usable, ningún registro distribuido aporta valor al usuario. Ese trayecto interdisciplinar completa el aprendizaje: de la física del sensor al relato visual, cada capa se ha convertido en una lección práctica que trasciende el caso de estudio y cimenta un perfil profesional capaz de navegar entre hardware y aplicaciones descentralizadas.

6.3. Limitaciones

Las pruebas de laboratorio y de campo confirmaron la solidez del flujo extremo-a-extremo, pero también dejaron a la luz los bordes donde el sistema todavía muestra carencias. El primero se sitúa en la propia frontera física: la conexión por puerto serie resulta idónea para prototipos y entornos controlados, sin embargo, en cuanto el microcontrolador se aleja del portátil —o simplemente vibra dentro de un vehículo— el cable limita la escalabilidad. Un enlace inalámbrico resolvería ese punto, aunque implicaría reabrir cuestiones de consumo energético y seguridad de transporte.

En la capa de consenso, Clique se portó con disciplina. Aun así, la literatura recuerda que su finalidad no es estricta y que redes con muchos firmantes o con enlaces inestables pueden sufrir bifurcaciones temporales. Esa advertencia sugiere que, para una red corporativa extensa, convendría migrar a IBFT 2.0 o a un protocolo con finalización instantánea y tolerante a fallos bizantinos.

El contrato inteligente presenta su propia paradoja. Su minimalismo —emitir eventos sin tocar estado— abarata gas y simplifica la auditoría, pero también fuerza a recrear el último estado fuera de la cadena. A escala de prototipo eso es manejable; en despliegues prolongados, la necesidad de procesar miles de eventos para conocer el “dónde-está-ahora” podría convertirse en una carga. Introducir un balance entre eventos y almacenamiento selectivo aparece como camino natural de evolución.

Finalmente, la *Blockchain* sólo certifica la inmutabilidad de lo que recibe, no su veracidad. Sin un mecanismo de firma criptográfica en el propio sensor, un nodo malicioso podría inyectar datos espurios que la red sellaría sin pestañear. Dotar al microcontrolador de claves privadas seguras y firmar cada mensaje antes de que abandone el entorno físico es, por tanto, un paso necesario para cerrar el círculo de confianza cuando este abandone el nodo portátil y evolucione a un usuario individual de la red.

Estas limitaciones no invalidan el sistema; más bien lo delimitan. Conocerlas permite proyectar mejoras concretas y evita caer en la complacencia. El prototipo funciona, pero su horizonte de madurez se dibuja justo en las fronteras aquí esbozadas.

6.4. Posibles mejoras

Las limitaciones identificadas no son un fin de trayecto, sino un mapa de oportunidades. Abordarlas permitiría llevar la plataforma desde un prototipo robusto a una

solución industrial plenamente operativa. Las líneas de evolución más inmediatas se resumen a continuación:

- **Conectividad inalámbrica (Wi-Fi)**. Sustituir la conexión serie por un enlace inalámbrico dotaría al nodo IoT de autonomía física y permitiría desplegar unidades en ubicaciones dispersas sin necesidad de un ordenador intermedio.
- **Red de nodos más amplia y heterogénea**. Escalar la infraestructura hacia múltiples gateways o nodos IoT operando en paralelo pondría a prueba la tolerancia del consenso.
- **Verificación criptográfica de datos de entrada**. Firmar cada paquete en el propio microcontrolador añadiría una capa de autenticidad que blinda el sistema frente a la inyección de datos falsificados.
- **Almacenamiento híbrido**. Complementar el contrato con datos on-chain selectivos.
- **Interfaz de usuario avanzada**. Una UI enriquecida con dashboards, filtros temporales y exportación de datos facilitaría la explotación analítica y acercaría la plataforma a usuarios no técnicos.
- **Migrar a consenso IBFT 2.0**. Sustituir Clique por un protocolo con finalización instantánea y tolerante a fallos bizantinos reduciría el riesgo de bifurcaciones y permitiría ampliar el número de validadores sin pérdida de estabilidad.
- **Contenerización y despliegue orquestado**. Empaquetar nodos Besu y gateways en contenedores Docker, y gestionarlos con Kubernetes, simplificaría la puesta en producción, la replicación y las actualizaciones continuas.
- **Monitorización y métricas de red**. Exponer métricas Prometheus en nodos y gateways y visualizarlas con Grafana permitiría anticipar cuellos de botella y automatizar alertas de degradación.
- **Cumplimiento regulatorio y privacidad**. Alinearse con los requisitos de la AESAN u otros marcos normativos.

Materializar estos pasos convertiría el prototipo en un sistema escalable, seguro y orientado a producción, capaz de integrarse con cadenas de suministro reales.

Apéndice A

Acrónimos

- **TFG** – Trabajo de Fin de Grado
- **DLT** – Distributed Ledger Technology
- **IoT** – Internet of Things
- **GPS** – Global Positioning System
- **PIR** – Passive Infrared Sensor
- **AESAN** – Agencia Española de Seguridad Alimentaria y Nutrición
- **EVM** – Ethereum Virtual Machine
- **ABI** – Application Binary Interface
- **RPC** – Remote Procedure Call
- **UI** – User Interface
- **CSV** – Comma-Separated Values
- **VPN** – Virtual Private Network
- **VM** – Virtual Machine
- **SCADA** – Supervisory Control and Data Acquisition
- **ERP** – Enterprise Resource Planning
- **NFT** – Non-Fungible Token
- **ID** – Identifier

- **JSON** – JavaScript Object Notation
- **API** – Application Programming Interface
- **CLI** – Command Line Interface
- **DAG** – Directed Acyclic Graph
- **CPU** – Central Processing Unit
- **IDE** – Integrated Development Environment
- **WiFi** – Wireless Fidelity
- **P2P** – Peer-to-Peer
- **PoW** – Proof of Work
- **PoS** – Proof of Stake
- **IBFT** – Istanbul Byzantine Fault Tolerance

Apéndice B

Glosario

- **ABI (Application Binary Interface)**
  Interfaz binaria que define cómo interactuar con un contrato inteligente desde fuera de la blockchain. Es utilizada para codificar y decodificar datos cuando se llaman funciones del contrato desde aplicaciones externas, como interfaces web.

- **AESAN (Agencia Española de Seguridad Alimentaria y Nutrición)**
  Organismo español responsable de velar por la seguridad alimentaria. Establece normativas sobre trazabilidad de productos alimenticios y supervisa su cumplimiento a lo largo de la cadena de suministro.

- **Algoritmo de consenso**
  Mecanismo utilizado en blockchain para que los nodos de la red acuerden el estado válido de las transacciones. En este proyecto se utilizan algoritmos como Clique e IBFT 2.0.

- **Arduino**
  Placa de desarrollo de código abierto utilizada en sistemas electrónicos embebidos. En este trabajo se emplean modelos como Arduino Uno y MKR Zero para la adquisición de datos desde sensores IoT.

- **Bloque génesis**
  Primer bloque de una blockchain. Es creado manualmente y contiene la configuración inicial de la red, incluyendo el algoritmo de consenso, el ChainID y las cuentas preconfiguradas.

- **Blockchain**
  Tecnología de registro distribuido que permite almacenar datos de forma segura,

inmutable y descentralizada mediante una cadena de bloques enlazados criptográficamente.

- **Clique**
  Algoritmo de consenso tolerante a fallos bizantinos basado en la firma de bloques por parte de validadores preautorizados. Es rápido y adecuado para redes privadas.

- **Contrato inteligente (Smart Contract)**
  Programa informático desplegado en la blockchain que se ejecuta automáticamente al cumplirse ciertas condiciones predefinidas. Permite automatizar reglas sin necesidad de intermediarios.

- **DHT11**
  Sensor digital que mide temperatura y humedad. Utilizado en el sistema IoT del proyecto para monitorizar condiciones ambientales en tiempo real.

- **DLT (Distributed Ledger Technology)**
  Tecnología que permite almacenar información de forma distribuida entre múltiples nodos sin necesidad de una autoridad central. Blockchain es un tipo específico de DLT.

- **Ether**
  Criptomoneda nativa de la red Ethereum. Se utiliza para pagar las tarifas de gas necesarias para ejecutar transacciones o contratos inteligentes.

- **ethers.js**
  Librería JavaScript para interactuar con la blockchain Ethereum. Permite enviar transacciones, leer eventos de contratos inteligentes y acceder a datos de la red.

- **Firmware**
  Conjunto de instrucciones de bajo nivel programadas en los dispositivos IoT para controlar su funcionamiento. En el proyecto se escribe el firmware para el NodeMCU en C/C++.

- **Gas**
  Unidad de medida que determina el coste computacional de ejecutar operaciones en Ethereum. Cada instrucción tiene un coste en gas, y este se paga en ether.

- **Hash**
  Función criptográfica que convierte una entrada de datos en una cadena alfanumérica de longitud fija. Se utiliza para garantizar integridad, anonimato y como identificador único.

- **Hyperledger Besu**
  Cliente blockchain compatible con Ethereum, desarrollado por la Fundación Hyperledger. Diseñado para redes empresariales con soporte para redes permisionadas y distintos algoritmos de consenso.

- **IBFT 2.0**
  Algoritmo de consenso tolerante a fallos bizantinos basado en votación entre nodos validadores. Requiere un quórum para agregar bloques y garantiza la finalidad de las transacciones.

- **IoT (Internet of Things)**
  Conjunto de dispositivos físicos interconectados a través de Internet capaces de recopilar, transmitir y procesar datos. En este TFG se emplean sensores y microcontroladores para monitorizar productos alimentarios.

- **Merkle Tree**
  Estructura de árbol utilizada en blockchain para resumir y verificar grandes volúmenes de datos. Cada nodo es un hash de sus nodos hijos, permitiendo validar datos de forma eficiente.

- **NodeMCU v3**
  Placa de desarrollo basada en el chip ESP8266 con conectividad WiFi. Usada como microcontrolador principal en el sistema IoT por su bajo coste y versatilidad.

- **Nonce**
  Número que se incrementa con cada transacción enviada desde una dirección. Evita ataques de repetición y garantiza el orden correcto de ejecución en la blockchain.

- **Puerto serie**
  Canal de comunicación física entre el microcontrolador (como NodeMCU) y el ordenador. Permite transmitir datos sensóricos desde el IoT al gateway.

- **RPC (Remote Procedure Call)**
  Protocolo que permite a un cliente ejecutar funciones en un servidor remoto. En blockchain se usa para interactuar con nodos Ethereum a través de interfaces como JSON-RPC.

- **Sensor PIR**
  Sensor de infrarrojos pasivo que detecta movimiento. Utilizado en el sistema para registrar la manipulación o desplazamiento de productos.

- **Solidity**
  Lenguaje de programación orientado a contratos inteligentes, utilizado para desarrollar aplicaciones sobre la Ethereum Virtual Machine (EVM).
- **Tangle**
  Estructura de datos basada en grafos acíclicos dirigida (DAG), utilizada por la red IOTA como alternativa a la cadena de bloques tradicional.
- **Token**
  Unidad de valor creada dentro de una blockchain, que puede representar activos digitales o derechos específicos en un sistema descentralizado.
- **Topic (Indexed Event)**
  Campo indexado dentro de un evento emitido por un contrato inteligente. Permite filtrar registros fácilmente mediante consultas en la blockchain.
- **Transacción**
  Acción registrada en la blockchain que puede consistir en transferir fondos, ejecutar un contrato inteligente o almacenar información. Cada transacción queda registrada de forma inmutable.
- **Trazabilidad**
  Capacidad de seguir el historial, aplicación o localización de un producto a lo largo de toda la cadena de suministro. Fundamental en seguridad alimentaria y calidad del producto.
- **VirtualBox**
  Software de virtualización que permite crear máquinas virtuales para simular múltiples nodos blockchain en un único equipo físico.
- **Visual Studio Code (VS Code)**
  Editor de código fuente ampliamente utilizado en desarrollo de software. Soporta extensiones útiles para trabajar con Solidity, JavaScript y otras tecnologías utilizadas en el TFG.
- **Web3.js**
  Librería JavaScript que permite la interacción con la blockchain Ethereum desde aplicaciones web. Facilita el envío de transacciones y la lectura de contratos inteligentes.
- **VPN (Virtual Private Network)**
  Red privada virtual que permite interconectar nodos blockchain distribuidos físicamente como si estuvieran en una red local, asegurando la comunicación y evitando exposición directa a Internet.

Apéndice C

Listado completo de códigos

C.1. Códigos de configuración de la red

```
{
  "config": {
    "chainId": 1982,
    "londonBlock": 0,
    "zeroBaseFee": true,
    "contractSizeLimit": 2147483647,
    "clique": {
      "blockperiodseconds": 60,
      "epochlength": 30000,
      "createemptyblocks": false
    }
  },
  "coinbase": "0x0000000000000000000000000000000000000000",
  "difficulty": "0x1",
  "extraData": "0x0000000000000000000000000000000000000000000000
    0000000000000000000badfe4969619ed7f72bfb6fc3b4488b947
    ec2845000000000000000000000000000000000000000000000000000000
    0000000000000000000000000000000000000000000000000000000000
    0000000000000000000000",
  "gasLimit": "0x1fffffffffffff",
  "mixHash": "0x0000000000000000000000000000000000000000000000000
    0000000000000000",
  "nonce": "0x0",
```

```
  "timestamp": "0x5c51a607",
  "alloc": {
    "badfe4969619ed7f72bfb6fc3b4488b947ec2845": {
      "balance": "0xae6ae8e5ccbfb04590405997ee2d52d2b330726137b875053c36d94e974d162f"
    },
    "8db6222ebe69c227c7ef2687ebb00d0f5be174f7": {
      "balance": "0xae6ae8e5ccbfb04590405997ee2d52d2b330726137b875053c36d94e974d162f"
    },
    "182e2470adb9014fc71bb070ffb2ebf38a18499c": {
      "balance": "0xae6ae8e5ccbfb04590405997ee2d52d2b330726137b875053c36d94e974d162f"
    }
  },
  "number": "0x0",
  "gasUsed": "0x0",
  "parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000"
}
```

Listado C.1: Archivo génesis

```
#node path
data-path="data"

#rpc options
rpc-http-enabled=true
rpc-http-api=["ETH","NET","WEB3","CLIQUE","ADMIN","DEBUG","PERM","TXPOOL"]
rpc-http-cors-origins=["*"]
host-allowlist=["*"]

#profile
profile="PRIVATE"

#chain
genesis-file="genesis.json"

```

```
permissions-nodes-config-file-enabled=true
permissions-nodes-config-file="permissions_config.toml"
```

Listado C.2: Archivo de configuración del nodo ADMIN.

```
#node path
data-path="data"

#rpc options
rpc-http-enabled=true
rpc-http-api=["ETH","NET","WEB3","CLIQUE","ADMIN","DEBUG","
   TXPOOL"]
rpc-http-cors-origins=["*"]
host-allowlist=["*"]

#profile
profile="PRIVATE"

#chain
genesis-file="genesis.json"

#network bootnode
bootnodes=["enode://cc2c1f1b3c49c5f277ccaab4fab8b5c500f59e787
   0f8b81c80435309e9406e939d0c5a3666446bfd10b1015957eec59
   fddace975bfe40b371970ee3abdb49e1f@192.168.255.10:30303"]
```

Listado C.3: Archivo de configuración de los nodos regulares.

C.2. Código embebido IoT

```
#include <DHT.h>
#include <SoftwareSerial.h>
#include <TinyGPS++.h>

#define DHTPIN        D5
#define DHTTYPE       DHT11
#define PIRPIN        D3
#define GPS_RX_PIN    D2
```

```
#define GPS_TX_PIN   D1

DHT dht(DHTPIN, DHTTYPE);
volatile unsigned int motionCount = 0;

TinyGPSPlus gps;
SoftwareSerial gpsSerial(GPS_RX_PIN, GPS_TX_PIN);

void ICACHE_RAM_ATTR handleMotion() {
  motionCount++;
}

void setup() {
  Serial.begin(9600);
  dht.begin();

  pinMode(PIRPIN, INPUT);
  attachInterrupt(digitalPinToInterrupt(PIRPIN), handleMotion
     , RISING);

  gpsSerial.begin(9600);
}

void loop() {
  while (gpsSerial.available()) {
    gps.encode(gpsSerial.read());
  }

    if (gps.location.isValid()) {
      float temperature = dht.readTemperature();
      float humidity    = dht.readHumidity();

      double lat = gps.location.lat();
      double lon = gps.location.lng();

      String csv = "," +
                   String(temperature, 1) + "," +
                   String(humidity,    1) + "," +
                   String(lat, 6) + "," +
```

```
                    String(lon, 6) + "," +
                    String(motionCount) +
                    "\n";

      Serial.print(csv);
  }
}
```

Listado C.4: Código embebido en el NodeMcu v3.

C.3. Smart Contract

```
require("@nomicfoundation/hardhat-toolbox");

  module.exports = {
    solidity: {
      version: "0.8.24",
      settings: {
        evmVersion: "london"
      },
    },
    networks: {
      besu: {
        url: "http://localhost:8545",
        chainId: 1982,
        accounts: ['0x0709c36acb5a539568840291e2a2ffadf92ac09
           189ed4eb5b64ef33d2f60c237'],
      }
    }

  };
```

Listado C.5: Archivo `hardhat.config.js`.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

contract indexed_id_product {
```

```
5
6       event indexed_id_product_event(int256 indexed id, string
            data);
7
8       function indexed_id_product_function(int256 id, string
            memory data) public {
9           emit indexed_id_product_event(id, data);
10      }
11  }
```

Listado C.6: Contrato `indexed_id_product.sol`

```
1  const { ethers } = require("hardhat");
2
3  async function main() {
4    const [deployer] = await ethers.getSigners();
5    console.log("Desplegando␣contrato␣con:", deployer.address);
6
7    const ContractFactory = await ethers.getContractFactory("
       indexed_id_product");
8
9    const contract = await ContractFactory.deploy({
10     gasLimit: 5000000,
11     gasPrice: ethers.parseUnits("1", "gwei"),
12     nonce: await deployer.getNonce(),
13     chainId: 1982,
14   });
15
16   console.log("Esperando␣confirmacion...");
17   const receipt = await contract.deploymentTransaction().wait
       (1);
18
19   console.log("Contrato␣desplegado␣correctamente:");
20   console.log("Direccion:", contract.target);
21   console.log("Tx␣Hash:", receipt.hash);
22   console.log("Gas␣usado:", receipt.gasUsed.toString());
23   console.log("Bloque:", receipt.blockNumber);
24   console.log("Remitente:", receipt.from);
25 }
```

```
26
27 main().catch((err) => {
28   console.error("Error al desplegar:", err);
29   process.exit(1);
30 });
```

Listado C.7: Código de despliegue del contrato C.6

C.4. Códigos de la pasarela

```
1  const { ethers } = require("hardhat");
2  const { SerialPort } = require("serialport");
3  const { ReadlineParser } = require("@serialport/parser-
     readline");
4
5  async function main() {
6    const port = new SerialPort({
7      path: "COM5",
8      baudRate: 9600,
9    });
10   const parser = port.pipe(new ReadlineParser({ delimiter: "\
       n" }));
11
12   const [wallet] = await ethers.getSigners();
13   console.log("Cuenta usada:", wallet.address);
14
15   const contractAddress = "0x062C4B86dcA6Ed53457cf75F4699651B
       64CD6478";
16
17   const contract = await ethers.getContractAt(
18     "indexed_id_product",
19     contractAddress
20   );
21
22   let products = [
23     { id: 1, totalPoints: 16, completedPoints: 0 },
24     { id: 2, totalPoints: 15, completedPoints: 0 },
```

```
    { id: 3, totalPoints: 20, completedPoints: 0 },
    { id: 4, totalPoints: 13, completedPoints: 0 },
    { id: 5, totalPoints: 14, completedPoints: 0 },
    { id: 6, totalPoints: 15, completedPoints: 0 },
    { id: 7, totalPoints: 18, completedPoints: 0 },
    { id: 8, totalPoints: 13, completedPoints: 0 },
  ];

  let currentProductIndex = 0;

  let nonce_actual = await wallet.getNonce();

  async function tx_function(id, data) {
    try {
      const tx = await contract.indexed_id_product_function(
          id, data, {
        gasLimit: 5000000,
        gasPrice: ethers.parseUnits("1", "gwei"),
        nonce: nonce_actual,
        chainId: 1982,
      });
      nonce_actual = nonce_actual + 1;
      console.log("Enviada TX:", tx.hash);
    } catch (error) {
      console.error("Error en TX:", error);
    }
  }

  async function tx_trazabilidad(id, data) {
    const product = products[currentProductIndex];
    console.log(
      `Producto ${id}, punto ${product.completedPoints + 1}:
         ${data}`
    );

    await tx_function(id, data);
    product.completedPoints++;

    if (product.completedPoints >= product.totalPoints) {
```

```
62        console.log(`Producto ${id} completado.`);
63        if (currentProductIndex < products.length - 1) {
64          currentProductIndex++;
65        }
66      }
67    }
68
69    parser.on("data", (line) => {
70      const rawData = line.trim();
71      const timestamp = Date.now();
72      const data = `${timestamp}${rawData}`;
73
74      const currentProduct = products[currentProductIndex];
75        if (currentProductIndex < products.length - 1) {
76          tx_trazabilidad(currentProduct.id, data);
77        } else {
78          console.log("Todos los productos completados.");
79        }
80    });
81  }
82
83  main()
84    .then()
85    .catch((error) => {
86      console.error(error);
87      process.exit(1);
88    });
```

Listado C.8: Código principal de la pasarela.

```
1  const { ethers } = require("hardhat");
2
3  async function main(id) {
4      const [owner] = await ethers.getSigners();
5      const contractAddress = "0x062C4B86dcA6Ed53457cf75F469965
         1B64CD6478";
6
7      const indexedIdProductABI = [
8          "event indexed_id_product_event(int256 indexed id,
```

```
            string␣data)",
        "function␣indexed_id_product_function(int256␣id,␣
            string␣memory␣data)␣public"
    ];

    const contract = new ethers.Contract(contractAddress,
        indexedIdProductABI, owner);

    const filter = contract.filters.indexed_id_product_event(
        id);

    const events = await contract.queryFilter(filter);

    const eventData = [];

    events.forEach((event) => {
        const dataParts = event.args.data.split(',');

        if (dataParts.length === 6) {
            const eventObj = {
                timestamp: String(dataParts[0]),
                temperatura: parseFloat(dataParts[1]),
                humedad: parseFloat(dataParts[2]),
                latitud: parseFloat(dataParts[3]),
                longitud: parseFloat(dataParts[4]),
                movimiento: parseInt(dataParts[5], 10)
            };

            eventData.push(eventObj);
        }
    });

    console.log(JSON.stringify(eventData, null, 2));
}

main(3)
    .then(() => process.exit(0))
    .catch((error) => {
        console.error(error);
```

```
44          process.exit(1);
45      });
```

Listado C.9: Código para recoger todos los logs.

C.5. Pruebas

C.5.1. Prueba de transacción

```
const { ethers } = require("hardhat");

async function main() {
  const [sender] = await ethers.getSigners();
  console.log("Cuenta remitente:", sender.address);
  console.log("Cuenta destinataria: 0xbadfe4969619ed7f72bfb6
     fc3b4488b947ec2844");

  const tx = await sender.sendTransaction({
    to: '0xbadfe4969619ed7f72bfb6fc3b4488b947ec2844',
    value: ethers.parseEther("1.0"),
    gasLimit: 21000,
    gasPrice: ethers.parseUnits("1", "gwei"),
    nonce: await sender.getNonce(),
    chainId: 1982,
  });

  console.log("Transaccion enviada. Hash:", tx.hash);
  const receipt = await tx.wait(1);
  console.log("Confirmada en bloque:", receipt.blockNumber);
}

main().catch((error) => {
  console.error("Error:", error);
  process.exit(1);
});
```

Listado C.10: Código de prueba de transacción.

C.5.2. Prueba de consulta de balance

```
const { ethers } = require("hardhat");

async function main() {

  const [account] = await ethers.getSigners();
  console.log("Cuenta:", account.address);

  const balanceWei = await ethers.provider.getBalance(account
     .address);

  const balanceEther = ethers.formatEther(balanceWei);
  console.log(`Balance: ${balanceEther} ETH`);
}

main()
  .then(() => process.exit(0))
  .catch((error) => {
    console.error("Error:", error);
    process.exit(1);
  });
```

Listado C.11: Código de prueba de consulta de balance.

C.5.3. Prueba del contrato

```
const { ethers } = require("hardhat");


async function main() {

  const [wallet] = await ethers.getSigners();
  console.log("Cuenta usada:", wallet.address);

  const id = 0;
  const data = "PRUEBA DE FUNCIONAMIENTO";

```

```
  const contractAddress =
    "0x062C4B86dcA6Ed53457cf75F4699651B64CD6478";

  const contract = await ethers.getContractAt(
    "indexed_id_product",
    contractAddress,
    wallet
  );

  console.log(contract);

  console.log(`Enviando TX para producto ${id} con datos: ${
     data}`);
  try {
    const tx = await contract.indexed_id_product_function(id,
        data,  {
        gasLimit: 5000000,
        gasPrice: ethers.parseUnits("1", "gwei"),
        nonce: await wallet.getNonce(),
        chainId: 1982,
      });
    console.log("Enviada TX. Hash:", tx.hash);

  } catch (error) {
    console.error("Error en TX:", error);
  }

}

main()
  .then(() => console.log("Script en ejecucion..."))
  .catch((error) => {
    console.error(error);
    process.exit(1);
  });
```

Listado C.12: Código de prueba del contrato.

C.5.4. Prueba del sistema IoT

```
const { SerialPort } = require("serialport");
const { ReadlineParser } = require("@serialport/parser-
   readline");

const port = new SerialPort({
    path: "COM5",
    baudRate: 9600,
  });
const parser = port.pipe(new ReadlineParser({ delimiter: "\n"
    }));

async function main() {
  parser.on("data", (line) => {
    const data = line.trim();
    console.log(data);
  });
}

main()
  .then()
  .catch((error) => {
    console.error(error);
    process.exit(1);
  });
```

Listado C.13: Código de prueba del sistema IoT.

Apéndice D

Manual de instalación

El objetivo de este manual es ayudar al lector que quiere reproducir este sistema, una por una de todas las partes del sistema.

Se omitirá el tutorial de cómo instalar las herramientas como Hardhat, React, etc; en sus respectivos repositorios ya existen guías.

Prerequisitos tener instalados Node.js, Npm, Hardhat, ethers.js, ArduinoIDE y React.

Todo el código se encuentra en el Github, excepto los relacionados con la configuración red; empezar por descagar el repositorio.

D.1. Configuración de la red

El primer paso es descargar la versión del Cliente Besu que se necesita para gestionar un nodo. Se pueden ejecutar tanto en Docker como a través de los binarios. Se decide usar los binarios por su facilidad de uso.

NOTE

Se recomienda siempre usar la última versión disponible del cliente. Este recibe actualizaciones periódicas muy frecuentemente; durante el desarrollo de este trabajo recibió hasta 4 actualizaciones.

En el GitHub oficial de Hyperledger Besu [77] se pueden encontrar todas las versiones hasta la fecha. Descargamos la última y la descomprimimos en la carpeta personal si se está en Linux o en 'C:' si se está en Windows. Es importante añadir la carpeta

bin al PATH para poder usar la línea de comandos.

Con Besu listo en la máquina, se crea un archivo *config_file.toml* donde se añadan las opciones de despliegue.

En los códigos C.2 y C.3 que corresponden al archivo de configuración del nodo ADMIN y un nodo regular respectivamente, se puede apreciar que hay muchas opciones comunes. Entre las opciones comunes encontramos:

```
#node path
data-path="data"
```

Esta línea define el directorio donde el nodo almacenará todos sus datos (estado, bloques, claves, etc.). En este caso, se utiliza una ruta relativa *"data"*, por lo que se creará una carpeta en el mismo directorio desde donde se lance Besu.

En esta carpeta se debe guardar la clave privada del nodo. Puede generarse automáticamente al generar la clave pública mediante el comando:

```
besu public-key export --data-path=data --to=public_key.txt
```

Esta clave pública será necesaria para crear los enodes de cada uno de los nodos; sin estos sería imposible configurar el listado de permitidos.

```
#rpc options
rpc-http-enabled=true
rpc-http-api=["ETH","NET","WEB3","CLIQUE","ADMIN","DEBUG","
   TXPOOL"]
rpc-http-cors-origins=["*"]
host-allowlist=["*"]
```

Estas opciones habilitan la interfaz RPC vía HTTP, imprescindible para que aplicaciones externas puedan interactuar con el nodo; se usará para la interfaz web. Se activan APIs fundamentales como:

- *ETH*, *NET*, *WEB3*: para operaciones básicas y consultas.

- *CLIQUE*: necesaria para gestionar el consenso basado en Proof of Authority (PoA).
- *ADMIN*, *DEBUG*, *TXPOOL*: útiles para monitorización, depuración y gestión del nodo.

WARNING

Es importante destacar que tanto *rpc-http-cors-origins=["*"]* como *host-allowlist=["*"]* permiten conexiones desde cualquier origen y cualquier IP. Esta configuración no se recomienda en entornos en producción debido a riesgos de seguridad. Se han activado en esta red para habilitar la colaboración con otro proyecto externo a este.

```
#profile
profile="PRIVATE"
```

Activa un perfil predefinido orientado a redes privadas. Este perfil:

- Permite transacciones sin coste de gas (*gasPrice = 0*).
- Aplica un modo secuencial de validación de transacciones (*txpool="sequence"*).
- Simplifica los requisitos para la puesta en marcha de redes no públicas.

La elección del perfil *PRIVATE* responde a la naturaleza de la red que se desea desplegar: una red privada y permisionada, donde los participantes están identificados y autorizados previamente. En este tipo de entornos, las necesidades de seguridad y control son diferentes a las de una red pública como Ethereum Mainnet, y por tanto se buscan configuraciones más flexibles y eficientes.

- **Transacciones sin coste de gas (*gasPrice = 0*):** al tratarse de una red privada, no es necesario incentivar económicamente la inclusión de transacciones en bloques. Elimina la fricción que supondría calcular y pagar tarifas por operaciones, lo que es especialmente útil en entornos de pruebas, desarrollo o consorcios donde los participantes no compiten.
- **Modo secuencial de validación (*txpool="sequence"*):** impone un orden rígido en la aceptación de transacciones por cuenta. Esto reduce significativamente los conflictos derivados de transacciones pendientes y simplifica la lógica de envío

en clientes y contratos. Es útil cuando se conoce de antemano la fuente de las transacciones y se busca mantener coherencia.

- **Facilita el despliegue:** el perfil *PRIVATE* establece por defecto varios parámetros que aceleran la puesta en marcha del nodo, omitiendo configuraciones de producción que no son necesarias en este tipo de redes. Por ejemplo, no se exige una política estricta de gas, ni se habilitan mecanismos complejos de monitoreo o protección contra ataques de red.

En resumen, este perfil está diseñado para maximizar la eficiencia y la simplicidad en redes donde la confianza entre los participantes ya está establecida y donde el rendimiento y la facilidad de administración son más importantes que la resistencia a actores maliciosos.

```
#chain
genesis-file="genesis.json"
```

Hace referencia a la ubicación del archivo *genesis.json*, el cual define la configuración inicial de la *Blockchain*.

NOTE

Es fundamental que **todos los nodos compartan exactamente el mismo archivo génesis** para formar parte de la misma red.

D.1.1. Nodo ADMIN

El nodo ADMIN tiene opciones habilitadas muy importantes que lo habilitan como administrador en este entono de prueba. Incluyen configuraciones específicas para la gestión de permisos, tanto a nivel de API como mediante un archivo de nodos permitidos a entablar conexión con él:

```
rpc-http-api=["PERM", ...]
permissions-nodes-config-file-enabled=true
permissions-nodes-config-file="permissions_config.toml"
```

La inclusión de la API *PERM* habilita al nodo para gestionar dinámicamente listas de control de acceso, lo cual es esencial en una red *permissioned*. Esta API permite añadir o eliminar nodos autorizados, así como gestionar cuentas con privilegios administrativos. Resulta fundamental en redes consorciadas donde los participantes deben cumplir con requisitos de gobernanza.

La opción *permissions-nodes-config-file-enabled=true* indica que el nodo debe cargar y aplicar las reglas definidas en el archivo *permissions_config.toml*. Esto no sólo permite validar los nodos entrantes durante el arranque, sino también hacer cumplir de forma automática las políticas de conexión definidas.

NOTE

Aunque sólo el nodo ADMIN tenga activada la API *PERM*, los efectos de sus decisiones pueden replicarse en los demás nodos si estos también tienen habilitada la validación de permisos. En redes más seguras, se recomienda que todos los nodos carguen el archivo de configuración, aunque sólo uno esté autorizado a modificarlo.

D.1.2. Nodos regulares

```
#network bootnode
bootnodes=["enode://cc2c1f1b3c49c5f277cca...3abdb49e1f@192.168.255.10:30303"]
```

Esta línea define un nodo de arranque (*bootnode*) utilizando su identificador *enode* y su dirección IP/puerto. El nodo listado actúa como punto de entrada a la red para los nodos que se inician posteriormente. Dicho nodo corresponde al nodo **ADMIN**.

NOTE

Es importante que la dirección IP y el puerto configurados en *bootnodes* coincidan con los expuestos por el nodo ADMIN, y que no existan restricciones de red (como firewalls) que impidan la conexión inicial.

D.1.3. Archivo génesis común

Todos los nodos deben tener el mismo archivo génesis para poder formar parte de la misma red. El archivo *genesis.json* (código C.1) define el estado inicial y las reglas de funcionamiento de la red.

```
"config": {
  "chainId": 1982,
  "londonBlock": 0,
  "zeroBaseFee": true,
  "contractSizeLimit": 2147483647,
  "clique": {
    "blockperiodseconds": 60,
    "epochlength": 30000,
    "createemptyblocks": false
  }
}
```

- ***chainId***: El valor *1982* identifica de forma única esta red privada, evitando ataques de *replay* entre distintas redes Ethereum.
- ***londonBlock: 0***: Aplica desde el bloque génesis las reglas de la bifurcación London (EIP-1559) [78].
- ***zeroBaseFee: true***: Desactiva la tarifa base progresiva de EIP-1559, es necesario este parámetro a true para poder configurar la red como libre de gas.
- ***contractSizeLimit***: Se establece un tamaño máximo de contrato extremadamente alto, el máximo, que facilita pruebas complejas.
- ***clique***: Define los parámetros del consenso Proof of Authority.
  - *blockperiodseconds: 60*: Generación de bloques cada 60 segundos. Es un tiempo conveniente para las pruebas ya que los dispositivos IoT envian transacciones cada 5 segundos. En la prueba de entorno real 5.4.6, no es así.
  - *epochlength: 30000*: Se renueva el conjunto de validadores cada 30.000 bloques. Es un valor recomendado pero en este sistema al tener un solo firmante como es el ADMIN, no es muy útil.

- *createemptyblocks: false*: Solo se generan bloques si existen transacciones, optimizando el uso de almacenamiento. No se quiere que la red cree en todo momento bloques cuando no hay nadie usándola. En entornos reales, esto es inadecuado ya que hay muchísimo flujo de transacciones.

```
"extraData": "0x...badfe4969619ed7f72bfb6fc3b4488b947ec2845
   ...",
"gasLimit": "0x1fffffffffffff",
```

- ***extraData***: Campo crítico en Clique. Incluye:
  - 32 bytes de relleno.
  - Dirección del nodo validador ADMIN: *0xbadfe4...*.
  - 65 bytes de firma (rellenados con ceros en el bloque génesis).
- ***gasLimit***: Se establece un límite de gas extremadamente alto, permitiendo la ejecución de transacciones sin restricciones. Es el máximo que se puede poner.

El ADMIN se convierte en el único capaz de escribir en la cadena de bloques.

```
"alloc": {
  "badfe4969...ec2845": { "balance": "0xae6a..." },
  "8db6222e...be174f7": { "balance": "0xae6a..." },
  "182e2470...a18499c": { "balance": "0xae6a..." }
}
```

Se asigna un balance muy elevado a tres cuentas predefinidas: el ADMIN y los dos portátiles. Esto es necesario para poder desplegar e interactuar con los contratos; aunque la red está libre de costes de gas se necesita de un mínimo de balance para poder desplegar un contrato. Para ahorrar futuros problemas, se le asigna el máximo balance que se permite.

Además de los parámetros personalizados mencionados, el archivo génesis contiene otros campos requeridos por el protocolo Ethereum, que tienen valores por defecto y no afectan directamente al comportamiento de la red:

- ***coinbase***: Dirección nula, sin uso en el consenso Clique.
- ***difficulty***: Valor mínimo (*0x1*), obligatorio pero sin efecto práctico en PoA.
- ***mixHash***, ***nonce*** y ***parentHash***: Valores estándar del bloque génesis.
- ***timestamp***: Marca de tiempo inicial, sin implicaciones operativas.
- ***number*** y ***gasUsed***: Definen el bloque como génesis y sin uso inicial de gas.

D.1.4. Uso de VPN

Dado que el nodo **ADMIN** no puede ser expuesto directamente al exterior mediante NAT, ya que se encuentra bajo el NAT de la universidad, se opta por establecer una red privada virtual (*VPN*) como solución de conectividad segura entre los distintos nodos de la red. La VPN se implementa utilizando **OpenVPN**, una herramienta robusta y ampliamente utilizada para crear túneles cifrados entre dispositivos a través de redes no confiables como Internet.

En este despliegue, todos los nodos (incluido el nodo ADMIN) se conectan como *clientes* a un servidor OpenVPN central, que actúa exclusivamente como intermediario de red. Esta configuración permite que los nodos se comporten como si estuvieran en una misma red local privada, sin necesidad de exposición pública ni redirección de puertos.

El uso de VPN permite una comunicación segura y directa entre nodos, utilizando direcciones IP privadas asignadas por el servidor VPN. Además, elimina la necesidad de exponer puertos públicos o configurar reglas complicadas de NAT.

Para conectarse a la VPN, cada nodo importa su archivo de configuración *.ovpn* personalizado. Este archivo incluye las credenciales, certificados y parámetros necesarios para establecer el túnel VPN con el servidor central.

NOTE

La creación y configuración detallada de la infraestructura OpenVPN queda fuera del alcance del presente TFG.

Una vez establecida la conexión VPN, los nodos pueden comunicarse entre sí utilizando las IPs privadas asignadas. Estas direcciones deben utilizarse en los campos *bootnodes* y en el archivo de permisos *permissions_config.toml*, de forma que la co-

municación y la validación de nodos se realice exclusivamente a través de la red virtual segura.

D.1.5. VMs

Para creación de las VMs se opta por usar VitualBox [79] para la gestión y creación de las máquinas y usar el sistema operativo Linux en concreto su distribución Ubuntu 22.04. Descargamos las iso de la página oficial [80] y creamos las máquinas con las especificaciones mínimas requeridas 5.3 debido a la limitación de recursos en el host, ADMIN.

Dado que todas las máquinas virtuales residen en el mismo sistema host que el nodo **ADMIN**, no es necesario conectarlas a través de la VPN. Al operar sobre el mismo entorno físico, pueden comunicarse utilizando la red interna, sin comprometer la seguridad ni la conectividad entre nodos.

Una vez instalada la imagen de Ubuntu en cada máquina virtual, se siguen los mismos pasos anteriores, adaptando los parámetros necesarios como las rutas locales, las IPs internas y los archivos de permisos.

D.1.6. Arranque de los nodos

Una vez completada la configuración de los archivos necesarios (*config_file.toml*, *genesis.json*, claves y permisos), el siguiente paso consiste en iniciar los nodos que formarán parte de la red.

El arranque de un nodo puede realizarse mediante un único comando:

```
1 besu --config-file=config_file.toml
```

Para mayor comodidad, especialmente en entornos Linux, es posible automatizar el proceso creando un script de inicio, por ejemplo *start.sh*, que contenga dicho comando. Este script se ejecutaría con:

```
1 sh start.sh
```

Se recomienda iniciar en primer lugar el nodo **ADMIN**, ya que los demás nodos están configurados para conectarse a través de él mediante el parámetro *bootnodes*. Una vez el nodo ADMIN esté en ejecución, los nodos restantes podrán establecer conexión y sincronizarse correctamente.

Con todos los nodos en funcionamiento, la red permisionada se encuentra completamente formada y lista para operar.

D.2. Sistema IoT

Ya se habló como montar el sistema IoT y el código del dispositivo se encuentra en el repositorio GitHub.

D.3. Smart Contract

Usar el script de despliegue C.7 para lanzar el contrato usando Hardhat.

D.4. Gateway: Pasarela entre la Blockchain y el sistema IoT

Para usar correctamente el programa C.8, se va explicar un poco cómo funciona.

La implementación hace uso de la biblioteca *serialport* para establecer una conexión directa con el dispositivo IoT a través del puerto *COM5*, configurado a una velocidad de *9600 baudios*, consistente con la configuración del firmware como se hizo en la prueba del sistema IoT. Para procesar las líneas de texto enviadas por el microcontrolador, se emplea el módulo *ReadlineParser*, que fragmenta el flujo en líneas delimitadas por salto de línea. Esta decisión responde a la necesidad de manejar los datos de forma estructurada y en tiempo real, sin realizar bloqueos ni búferes complejos.

El script obtiene automáticamente la clave firmante desde la configuración de Hardhat, y accede al contrato previamente desplegado utilizando su dirección y nombre. No se

recurre a archivos ABI externos, ya que Hardhat proporciona toda la información de compilación de forma integrada.

Se define localmente una estructura de productos que simula una cadena de producción. Cada producto posee un identificador, un número total de puntos de trazabilidad y un contador de puntos ya completados. Esta aproximación permite modelar procesos secuenciales como si cada lectura del sistema IoT representara una etapa superada por un producto dentro del flujo logístico.

NOTE

Se considera esta etapa, el alta de un producto.

Al recibirse una línea de datos desde el microcontrolador, esta se concatena con una marca temporal generada en el nodo móvil (portátil). Esta decisión permite añadir una referencia temporal confiable desde el lado del gateway, evitando depender del reloj del dispositivo embebido, que podría carecer de sincronización. Añadir un timestamp a la línea tiene el propósito de evitar la pérdida de esta información en caso de una caída de Internet; se almacenarían en el pool del cliente sin enviarse y se enviarían al volver la conexión.

A continuación, se invoca al contrato en la *Blockchain*, enviando los datos mediante la función que emite el evento indexado.

Cada vez que un producto alcanza su número total de puntos trazables, el sistema pasa automáticamente al siguiente. Esta lógica permite mantener un flujo automatizado de registro sin necesidad de intervención manual.

Conociendo como funciona se puede modificar los datos que hagan falta como por ejemplo el identificador del puerto serie si es necesario.

D.5. Interfaz Web

Crear un proyecto con React-Vite y añadir TailWindCSS [73]. Copiar el código correspondiente a la interfaz del repositorio e instalar las dependencias del **package.json** con *npm*.

Con todo esto listo, ya se tiene instalado todo el sistema.

Apéndice E

Manual de usuario

El manual de usuario tiene como objetivo guiar en una primera toma de contacto a cualquier persona que haya reproducido el sistema de este TFG. Se da por hecho que el lector ha seguido ya los pasos del anterior manual D.

Para trazar un producto deben seguirse los siguientes pasos como preparación inicial. Conectar físicamente el sistema IoT al equipo elegido para actuar como nodo móvil, lanzar el script prueba Node.js C.13 de escucha serial para verificar que los datos comienzan a fluir correctamente.

> **NOTE**
>
> El Arduino MKR GPS Shield puede tardar unos minutos en obtener la señal de los satélites. Se recomienda tener al aire libre donde con una mejor visibilidad del cielo, conseguirá antes la señal.

Con el flujo de datos confirmado, proceder a activar la VPN (figura E.1), paso necesario para permitir la comunicación entre el nodo portátil y el nodo `ADMIN`, que usualmente se encontraría en otra ubicación.

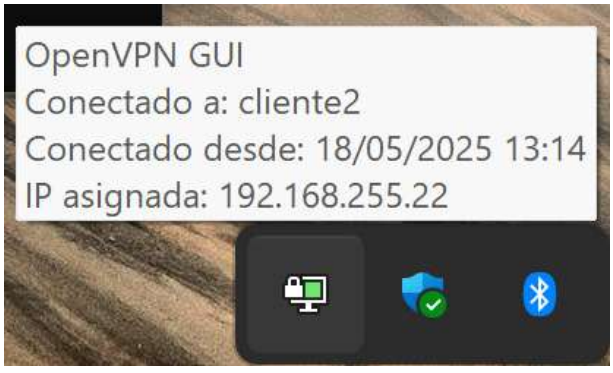

Figura E.1: VPN activada.

Una vez establecida la conexión privada, el nodo portátil está conectado a la red permisionada de Besu, garantizando así su sincronización como un nodo autorizado, como se ve en las figuras E.2-E.3.

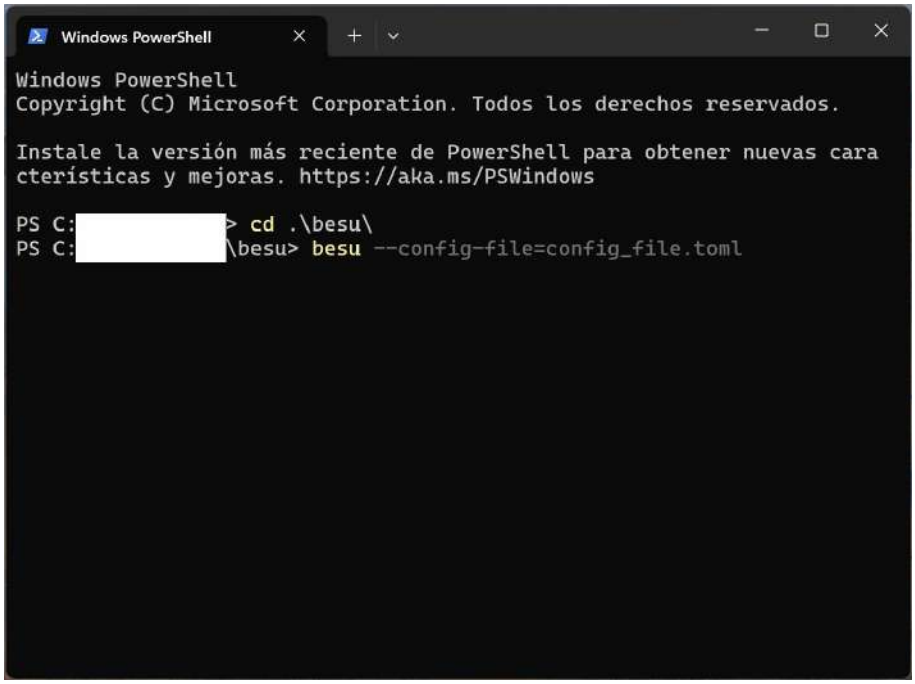

Figura E.2: Línea de comando para arrancar Besu en potátil Windows.

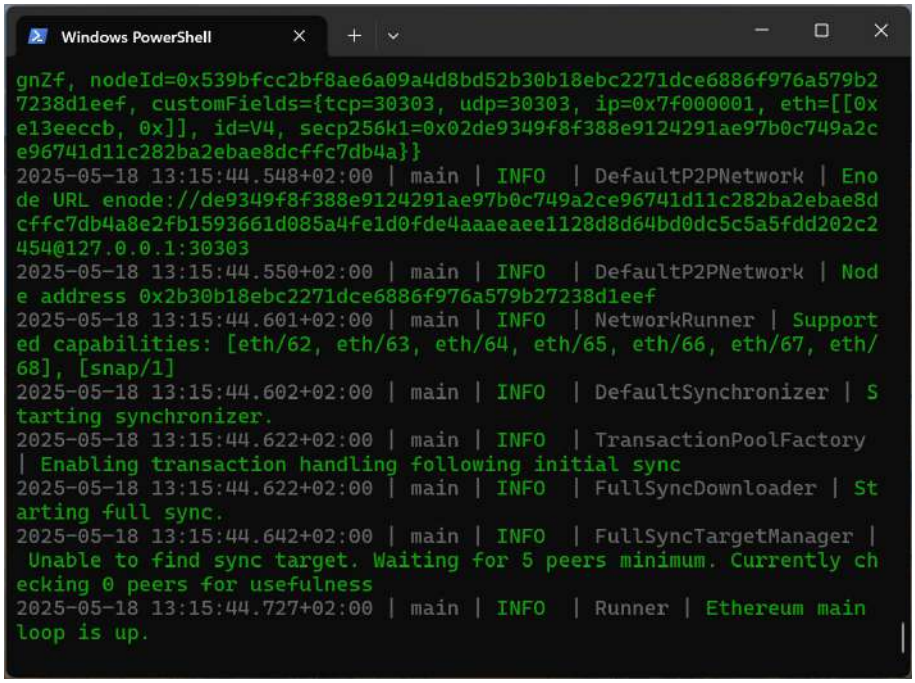

Figura E.3: Besu arrancado sin fallos.

A continuación, se activa el componente *gateway*, encargado de leer los datos del puerto serie, interpretarlos y registrar los eventos en la *Blockchain*.

En ese momento, el sistema está completamente operativo y comienza a trazar información en tiempo real, enviando cada registro como evento al contrato inteligente previamente desplegado. Opcionalmente, también se puede iniciar la interfaz web, lo que permite visualizar los datos conforme son recibidos y procesados, facilitando una supervisión activa del recorrido, ejemplo en la figura E.4.

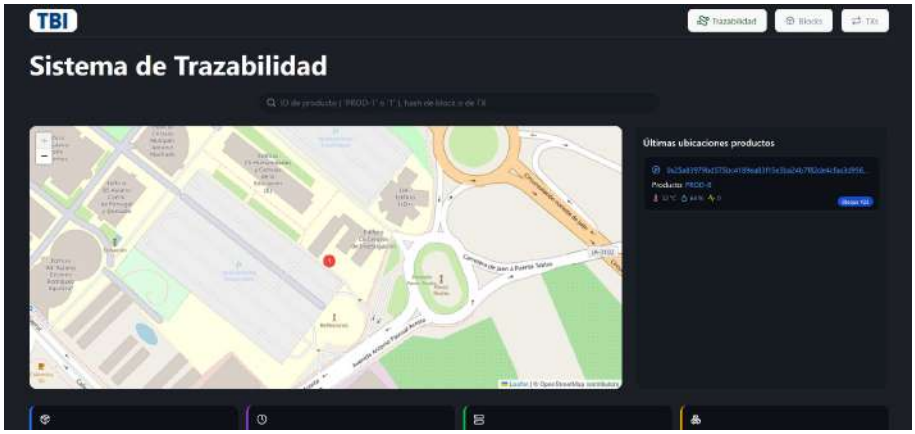

Figura E.4: Visualización de la trazabilidad en tiempo real.

Con todos los elementos funcionando de forma coordinada, se puede emprender un trayecto. Durante la marcha, el sistema continuará capturando datos y generando eventos sin interrupciones. Las coordenadas GPS variarñan según el desplazamiento, el sensor PIR seguirá acumulando detecciones y los parámetros ambientales fluctuaran conforme cambiaban las condiciones externas. Esta información será encapsulada en eventos que se almacenaban en la red, permitiendo reconstruir posteriormente el recorrido con total precisión y trazabilidad.

Bibliografía

[1] Blockchain Federal Argentina. Protocolos de consenso, 2023. URL https://bfa.ar/blockchain/protocolos-de-consenso. Consultado el 24 de abril de 2025.

[2] Agencia Española de Protección de Datos. Blockchain (ii): Conceptos básicos y protección de datos, 2020. URL https://www.aepd.es/prensa-y-comunicacion/blog/blockchain-II-conceptos-basicos-proteccion-de-datos. Consultado el 24 de abril de 2025.

[3] Daniel Drescher. *Blockchain Basics: A Non-Technical Introduction in 25 Steps*. Apress, Berkeley, CA, 2017. ISBN 978-1-4842-2603-2. doi: 10.1007/978-1-4842-2604-9. URL https://doi.org/10.1007/978-1-4842-2604-9.

[4] ChainMyne. ¿qué son los nodos en blockchain? guía para hnwis, 2025. URL https://chainmyne.com/es/what-are-nodes-in-blockchain-hnwi-guide/. comprobado en 2025-04-25.

[5] Etherscan. Transacción ethereum 0xee3f92741dcc3734..., 2025. URL https://etherscan.io/tx/0xee3f92741dcc3734db5d0664cb752f269c21f3b867e692adbd0091f0e517d1a comprobado en 2025-04-25.

[6] Regio-Michelin. Blockchain block structure, 2025. URL https://www.researchgate.net/figure/Blockchain-block-structure_fig1_325136332. comprobado en 2025-04-25.

[7] Andreas M. Antonopoulos and David A. Harding. *Mastering Bitcoin: Programming the Open Blockchain*. O'Reilly Media, 3 edition, 2023. ISBN 978-1-098-15009-9. URL https://learning.oreilly.com/library/view/mastering-bitcoin-3rd/9781098150082/. comprobado en 2025-04-25.

[8] Daniel Olshansky. 5p;1r — ethereum's modified merkle patricia trie, 2022. URL https://medium.com/coinmonks/5p-1r-ethereums-modified-merkle-patricia-trie-6956f5888398. comprobado en 2025-04-25.

[9] MARTA CURIEL. 40 aniversario así ocurrió, así contamos la intoxicación por aceite de colza en españa, 2021. URL https://www.rtve.es/noticias/20210531/asi-ocurrio-asi-contamos-colza/2096404.shtml/. comprobado en 2025-05-01.

[10] E. Hofmann, U.M. Strewe, and N. Bosia. *Supply Chain Finance and Blockchain Technology: The Case of Reverse Securitisation*. SpringerBriefs in Finance. Springer International Publishing, 2017. ISBN 9783319623719. doi: 10.1007/978-3-319-62371-9. URL https://books.google.es/books?id=tLIvDwAAQBAJ.

[11] Agencia Española de Seguridad Alimentaria y Nutrición. Aesan, 2001. URL https://www.aesan.gob.es/AECOSAN/docs/documentos/publicaciones/seguridad_alimentaria/guia_trazabilidad.pdf. comprobado en 2025-05-01.

[12] Talent.com. Salario ingeniero informático en españa, 2025. URL https://es.talent.com/salary?job=ingeniero+informÃątico. Consultado el 2025-05-01.

[13] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. URL https://bitcoin.org/bitcoin.pdf/. comprobado en 2025-05-01.

[14] Horst Treiblmaier and Trevor Clohessy, editors. *Blockchain and Distributed Ledger Technology Use Cases: Applications and Lessons Learned*. Progress in IS. Springer, Cham, 2020. ISBN 978-3-030-44337-5. doi: 10.1007/978-3-030-44337-5. URL https://doi.org/10.1007/978-3-030-44337-5.

[15] IBM. ¿qué es la seguridad de blockchain?, sin fecha. URL https://www.ibm.com/think/topics/blockchain-security. comprobado en 2025-04-23.

[16] IOTA Documentation. About iota, 2025. URL https://docs.iota.org/about-iota/. Consultado el 2025-05-01.

[17] Hedera Documentation. Hedera: A public hashgraph network & governing council, 2023. URL https://files.hedera.com/hh_whitepaper_v2.2-20230918.pdf. Consultado el 2025-05-01.

[18] I. Bashir. *Mastering Blockchain*. Packt Publishing, 2017. ISBN 978-1-78712-544-5. URL https://books.google.es/books?id=urkrDwAAQBAJ.

[19] Bikramaditya Singhal, Gautam Dhameja, and Priyanshu Sekhar Panda. *Beginning Blockchain: A Beginner's Guide to Building Blockchain Solutions*. Apress, Berkeley, CA, 2018. ISBN 978-1-4842-3443-3. doi: 10.1007/978-1-4842-3444-0. URL https://doi.org/10.1007/978-1-4842-3444-0.

[20] B. Ramamurthy. *Blockchain in Action*. In Action. Manning, 2020. ISBN 9781617296338. URL https://books.google.es/books?id=Al4BEAAAQBAJ.

[21] Andreas M. Antonopoulos and Gavin Wood. *Mastering Ethereum: Building Smart Contracts and DApps*. O'Reilly Media, 2018. ISBN 978-1-4919-7194-9. URL https://github.com/ethereumbook/ethereumbook.

[22] Solana Foundation. Solana: Web3 infrastructure for everyone, 2025. URL https://solana.com/. comprobado en 2025-04-25.

[23] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger, 2025. URL https://ethereum.github.io/yellowpaper/paper.pdf. comprobado en 2025-04-25.

[24] Ethereum Foundation. The merge, 2025. URL https://ethereum.org/en/upgrades/merge/. comprobado en 2025-04-25.

[25] Ethereum Foundation. Proof-of-stake (pos) — ethereum.org, 2023. URL https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/. comprobado en 2025-04-25.

[26] Bitpanda Academy. Los algoritmos de consenso: proof-of-stake, 2023. URL https://www.bitpanda.com/academy/es/lecciones/los-algoritmos-de-consenso-proof-of-stake/. comprobado en 2025-04-25.

[27] Péter Szilágyi. Eip-225: Clique proof-of-authority consensus protocol, 2017. URL https://eips.ethereum.org/EIPS/eip-225. comprobado en 2025-04-25.

[28] Roberto Saltini and David Hyland-Wood. Ibft 2.0: A safe and live variation of the ibft blockchain consensus protocol for eventually synchronous networks, 2019. URL https://arxiv.org/abs/1909.10194. Accedido el 25 de abril de 2025.

[29] Hyperledger Besu. Configurar el consenso ibft 2.0 en una red privada, 2025. URL https://besu.hyperledger.org/stable/private-networks/how-to/configure/consensus/ibft. Accedido el 25 de abril de 2025.

[30] CoinMarketCap Academy. Client, 2021. URL https://coinmarketcap.com/academy/glossary/client. comprobado en 2025-04-25.

[31] Go Ethereum Documentation. Geth — cliente de ejecución de ethereum, 2025. URL https://geth.ethereum.org/. Consultado el 2025-05-01.

[32] Ethereum Foundation. Archive nodes, 2025. URL https://ethereum.org/en/developers/docs/nodes-and-clients/archive-nodes/. comprobado en 2025-04-25.

[33] Hyperledger Besu. Node clients, 2025. URL https://besu.hyperledger.org/public-networks/concepts/node-clients. comprobado en 2025-04-25.

[34] ConsenSys Documentation. Teku — cliente de consenso de ethereum 2.0 para staking institucional, 2025. URL https://consensys.io/teku. Consultado el 2025-05-01.

[35] Hyperledger Besu Documentation. Use the engine api, 2025. URL https://besu.hyperledger.org/public-networks/how-to/use-engine-api. Consultado el 2025-05-01.

[36] Dimitrios Serpanos and Marilyn Wolf. *Internet-of-Things (IoT) Systems: Architectures, Algorithms, Methodologies*. Springer, 2018. ISBN 978-3-319-69714-7. doi: 10.1007/978-3-319-69715-4. URL https://doi.org/10.1007/978-3-319-69715-4. comprobado en 2025-04-25.

[37] Jennifer McEntire and Andrew W. Kennedy, editors. *Food Traceability*. Food Microbiology and Food Safety. Springer Cham, 2019. ISBN 978-3-030-10900-4. doi: 10.1007/978-3-030-10902-8. URL https://doi.org/10.1007/978-3-030-10902-8. Publicado el 22 de mayo de 2019.

[38] Dipole RFID. Etiquetas rfid: qué son y qué aplicaciones tienen, 2025. URL https://www.dipolerfid.es/blog-rfid/etiquetas-rfid-y-aplicaciones. Consultado el 2025-05-05.

[39] SAP. ¿qué es erp?, 2025. URL https://www.sap.com/spain/products/erp/what-is-erp.html. Consultado el 2025-05-01.

[40] COPA-DATA. ¿qué es scada?, 2025. URL https://www.copadata.com/es/productos/zenon-software-platform/que-es-scada/. Consultado el 2025-05-01.

[41] Hyperlegder Foundation. Hyperledger besu oficial site, 2019. URL https://besu.hyperledger.org/. comprobado en 2025-05-01.

[42] Hyperledger Besu. Data storage formats, 2025. URL https://besu.hyperledger.org/public-networks/concepts/data-storage-formats. comprobado en 2025-04-25.

[43] Hyperledger Besu. Access the besu api, 2025. URL https://besu.hyperledger.org/public-networks/how-to/use-besu-api. comprobado en 2025-04-25.

[44] Hyperledger Besu. Events and logs, 2023. URL https://besu.hyperledger.org/public-networks/concepts/events-and-logs. comprobado en 2025-04-25.

[45] Hyperledger Besu. Network id and chain id, 2025. URL https://besu.hyperledger.org/public-networks/concepts/network-and-chain-id. comprobado en 2025-04-25.

[46] Hyperledger Besu. Genesis file, 2025. URL https://besu.hyperledger.org/public-networks/concepts/genesis-file. comprobado en 2025-04-25.

[47] Hyperledger Besu. Consensus protocols, 2025. URL https://besu.hyperledger.org/private-networks/how-to/configure/consensus. comprobado en 2025-04-25.

[48] Hyperledger Besu. Configure mining, 2025. URL https://besu.hyperledger.org/public-networks/how-to/use-pow/mining. comprobado en 2025-04-25.

[49] Hyperledger Besu. Proof of stake consensus, 2025. URL https://besu.hyperledger.org/public-networks/concepts/proof-of-stake. comprobado en 2025-04-25.

[50] Hyperledger Besu. Node clients: Execution clients, 2025. URL https://besu.hyperledger.org/public-networks/concepts/node-clients#execution-clients. comprobado en 2025-04-25.

[51] Hyperledger Besu. Configure clique consensus, 2025. URL https://besu.hyperledger.org/private-networks/how-to/configure/consensus/clique. comprobado en 2025-04-25.

[52] Hyperledger Besu. Configure qbft consensus, 2025. URL https://besu.hyperledger.org/private-networks/how-to/configure/consensus/qbft. comprobado en 2025-04-25.

[53] Hyperledger Besu. Use json-rpc over http, websocket, and ipc, 2025. URL https://besu.hyperledger.org/stable/public-networks/how-to/use-besu-api/json-rpc. comprobado en 2025-05-04.

[54] Hyperledger Besu. Private network api methods: Perm (permissioning) methods, 2025. URL https://besu.hyperledger.org/private-networks/reference/api#perm-permissioning-methods. comprobado en 2025-05-04.

[55] Hyperledger Besu. Besu api: web3 methods, 2025. URL https://besu.hyperledger.org/public-networks/reference/api#web3-methods. comprobado en 2025-05-04.

[56] Consensys, 2019. URL https://cdn.consensys.io/uploads/AURA_ConsenSys_Press-Release_May-16-2019-1.pdf.

[57] Parlamento Europeo. Reglamento (ce) n.º 178/2002 del parlamento europeo y del consejo, de 28 de enero de 2002, por el que se establecen los principios y los requisitos generales de la legislación alimentaria, se crea la autoridad europea de seguridad alimentaria y se fijan procedimientos relativos a la seguridad alimentaria, 2002. URL https://www.boe.es/buscar/doc.php?id=DOUE-L-2002-80201. Publicado en el DOCE n.º 31, de 1 de febrero de 2002, págs. 1–24. Consultado el 2025-05-05.

[58] MakeIt.ca. Nodemcu details and specifications, 2025. URL https://www.make-it.ca/nodemcu-details-specifications/. Consultado el 2025-05-01.

[59] Aosong Electronics. Dht11 - technical data sheet (translated version), 2018. URL https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf?srsltid=AfmBOoqxlt_JXHyY8x3NbjtoZfGhDOjvP1yd4T-wQi4_t8hhdCsn9rXK. Consultado el 2025-05-01.

[60] Arduino Documentation. Mkr gps shield, 2025. URL https://docs.arduino.cc/hardware/mkr-gps-shield/. Consultado el 2025-05-01.

[61] Adafruit Industries. Pir (passive infrared) proximity motion sensor, 2025. URL https://cdn-learn.adafruit.com/downloads/pdf/pir-passive-infrared-proximity-motion-sensor.pdf. Consultado el 2025-05-01.

[62] Arduino Documentation. Arduino software, 2025. URL https://www.arduino.cc/en/software. Consultado el 2025-05-01.

[63] Microsoft. Visual studio code, 2025. URL https://code.visualstudio.com/. Consultado el 2025-05-01.

[64] Hyperledger Besu Documentation. System requirements, 2025. URL https://besu.hyperledger.org/private-networks/get-started/system-requirements#vm-requirements. Consultado el 2025-05-01.

[65] Node.js Documentation. Node.js — ejecuta javascript en cualquier parte, 2025. URL https://nodejs.org/es. Consultado el 2025-05-01.

[66] Oracle Documentation. Java downloads, 2025. URL https://www.oracle.com/java/technologies/downloads/. Consultado el 2025-05-01.

[67] npm Documentation. npm | home, 2025. URL https://www.npmjs.com/. Consultado el 2025-05-01.

[68] Hardhat Documentation. Hardhat — entorno de desarrollo para ethereum, 2025. URL https://hardhat.org/. Consultado el 2025-05-01.

[69] Solidity Documentation. Solidity programming language, 2025. URL https://soliditylang.org/. Consultado el 2025-05-01.

[70] Ethers.js Documentation. Ethers.js v6 documentation, 2025. URL https://docs.ethers.org/v6/. Consultado el 2025-05-01.

[71] React Documentation. React — la biblioteca para interfaces de usuario web y nativas, 2025. URL https://es.react.dev/. Consultado el 2025-05-01.

[72] Vite Documentation. Vite — herramienta de desarrollo frontend de próxima generación, 2025. URL https://vite.dev/. Consultado el 2025-05-01.

[73] Tailwind CSS Documentation. Tailwind css — rapidly build modern websites without ever leaving your html, 2025. URL https://tailwindcss.com/. Consultado el 2025-05-01.

[74] React Router Documentation. React router — enrutamiento declarativo para react, 2025. URL https://reactrouter.com/. Consultado el 2025-05-01.

[75] ESP8266 Community. Esp8266 arduino core package index, 2025. URL http://arduino.esp8266.com/stable/package_esp8266com_index.json. Consultado el 2025-05-01.

[76] Web3.js Documentation. Web3.js — documentación oficial, 2025. URL https://docs.web3js.org/. Consultado el 2025-05-01.

[77] Hyperledger Besu Documentation. Releases de hyperledger besu, 2025. URL https://github.com/hyperledger/besu/releases. Consultado el 2025-05-01.

[78] NiceHash Blog. ¿qué es la bifurcación de londres y qué pasará después del eip-1559?, 2025. URL https://www.nicehash.com/blog/post/what-is-the-london-fork-and-what-will-happen-after-eip-1559?lang=es. Consultado el 2025-05-01.

[79] Oracle VirtualBox Documentation. Virtualbox — virtualización de código abierto para uso personal y empresarial, 2025. URL https://www.virtualbox.org/. Consultado el 2025-05-01.

[80] Ubuntu Documentation. Descargar ubuntu desktop, 2025. URL https://ubuntu.com/download/desktop. Consultado el 2025-05-01.