Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Ingeniería de Software

Gestión de cadenas de suministros usando blockchain y contratos inteligentes

Autor:

Eduardo García Asensio

Tutor:

Joaquín Nicolás Adiego Rodriguez

Resumen

Este trabajo de fin de grado explora la vanguardia de la tecnología blockchain con el desarrollo de una aplicación descentralizada en la plataforma Ethereum. En un momento en que las criptomonedas generan opiniones polarizadas, con fervientes defensores y críticos acérrimos, este proyecto busca ir más allá del debate. A través de una investigación objetiva, se desentrañan los secretos de los contratos inteligentes y la cadena de suministros, mostrando cómo la descentralización puede revolucionar la transparencia y eficiencia en los procesos industriales. Este proyecto no solo es un puente entre la teoría y la práctica, sino también una invitación a repensar la estructura de confianza en el mundo digital, evaluando si esta tecnología es verdaderamente útil o simplemente una burbuja pasajera.

Además, se realiza un análisis exhaustivo de las tecnologías empleadas y descartadas, justificando cada elección y explorando su potencial impacto en el desarrollo de aplicaciones. La implementación práctica del proyecto, desde la configuración de la red hasta el despliegue de contratos inteligentes, proporciona una guía detallada para aquellos interesados en adentrarse en el mundo de las aplicaciones descentralizadas. Las conclusiones obtenidas ofrecen una visión crítica y fundamentada sobre el estado actual y el futuro de la blockchain, contribuyendo al debate global sobre su viabilidad y utilidad a largo plazo.

Abstract

This undergraduate thesis delves into the forefront of blockchain technology by developing a decentralized application on the Ethereum platform. At a time when cryptocurrencies generate polarized opinions, with fervent advocates and staunch critics, this project seeks to go beyond the debate. Through an objective investigation, the secrets of smart contracts and supply chains are unraveled, demonstrating how decentralization can revolutionize transparency and efficiency in industrial processes. This project is not just a bridge between theory and practice, but also an invitation to rethink the trust structure in the digital world, assessing whether this technology is truly useful or just a passing bubble.

Additionally, an exhaustive analysis of the technologies used and discarded is conducted, justifying each choice and exploring their potential impact on the development of applications. The practical implementation of the project, from network setup to smart contract deployment, provides a detailed guide for those interested in venturing into the world of decentralized applications. The conclusions drawn offer a critical and well-founded perspective on the current state and future of blockchain, contributing to the global debate on its long-term viability and utility.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Introducción

Hace unos años, una nueva tecnología irrumpió con fuerza en nuestro día a día causando mucho ruido. Podíamos oír palabras como Bitcoin, criptomonedas, NFTs, *smart contracts*... sin tener muy claro que había detrás. Este Trabajo pretende mostrar los entresijos de la tecnología que hay detrás de los términos anteriores, la **blockchain**. Esta nueva tecnología rompe con la arquitectura centralizada utilizada comúnmente por la mayoría de aplicaciones, proponiendo un enfoque descentralizado. Esta diferencia, nos llevará a enfrentar atípicos problemas, que una vez resueltos, nos permitirán explotar las ventajas. Aplicaré esta tecnología sobre un sector en particular: las cadenas de suministro, desarrollando una aplicación descentralizada, que resolverá muchos problemas existentes en una arquitectura centralizada, ejemplificando así los pros y contras.

Actualmente, las cadenas de suministros son enormemente complejas. En ellas se ven envueltos muchas entidades diferentes, como productores, transportistas, intermediarios, consumidores... todos ellos con diferentes responsabilidades e intereses. Es habitual que haya conflictos entre las partes, que han de ser resueltos, en ultima instancia por mediadores. Además, gestionar una trazabilidad precisa de un producto, desde su origen hasta el destino final, puede convertirse en una ardua tarea. Estos son algunos de los problemas a los que intentaré dar solución en el presente Trabajo, aplicando la blockchain. Como desarrollaré en el Capítulo 2, esta tecnología permitirá llevar a cabo una trazabilidad de un producto a lo largo de toda la cadena, así como evitar intermediarios agilizando y simplificando el proceso.

1.2. Motivaciones

La finalidad principal que motiva el desarrollo de este trabajo es la adquirir conocimientos acerca de los ecosistemas blockchain. Puesto que es una tecnología relativamente nueva, no se ha aprendido nada sobre ella durante el Grado. Sin embargo, si he estudiado conceptos que aplica la blockchain, como la descentralización entre otros, los cuales me resultan particularmente interesantes.

Por otro lado, la blockchain propone una descentralización del poder, que recae directamente sobre los individuos. Esto confronta con cómo la sociedad está estructurada en la actualidad, ya que todos los ciudadanos centralizamos el poder en unas pocas personas que nos representan, descargando en estas la responsabilidad, confianza y toma de decisiones. Por tanto, otra motivación para la realización de este proyecto, es la de comprender qué alcance real tiene esta tecnología para cambiar el modelo estructural de la sociedad.

Además, la blockchain ha causado mucho ruido estos últimos años, y es interesante descubrir cuánto de ese ruido esta fundamentado, y cuánto corresponde a la burbuja que se ha creado.

Acerca de por qué aplicar la blockchain a las cadenas de suministros, puedo decir que hallar una manera de conectar individuos, sin necesidad de que haya grandes empresas intermediarias lucrándose por el mero hecho de ser grandes, es muy apasionante. Esto conlleva que, por ejemplo, un agricultor pueda comunicarse directamente con una sociedad de consumidores, reavivando así un sector tan perjudicado en estos tiempos.

1.3. Objetivos

El objetivo principal de este trabajo es explorar la tecnología blockchain a fin de adquirir los conocimientos necesarios para tener éxito en un hipotético futuro, donde se quiera desarrollar un modelo de negocio comercial.

Para alcanzar dicho fin, podemos desgranarlo en los siguientes subobjetivos:

- **Subobjetivos técnicos**
  - Desarrollar un prototipo de aplicación descentralizada que aplique sobre el sector de cadenas de suministros, y esté basada en blockchain. Este prototipo ha de explotar los beneficios que aporta la blockchain, para resolver problemas que presentan las cadenas de suministro. De esta manera, la aplicación deberá automatizar procesos, permitir la compra-venta de productos y crear una trazabilidad de estos.
  - Desarrollar y desplegar *smart contracts* que den soporte a la aplicación anteriormente mencionada. Estos actuarán como backend, conteniendo la lógica de la aplicación, y pudiendo ser empleados por cualquier otra aplicación frontend.
  - Crear una red privada basada en blockchain sobre la que probar diferentes configuraciones, con el fin de entender el funcionamiento, permitir desplegar *smart contracts* y dar soporte a la aplicación.
- **Subobjetivos de competencias**
  - Explorar y adquirir conocimientos en algún framework que proporcione soporte para el desarrollo de frontends de aplicaciones descentralizadas.
  - Dominar algún lenguaje específico destinado al desarrollo de *smart contracts*. De esta manera, se pretende aprender la sintaxis, la estructura y buenas prácticas asociadas al lenguaje seleccionado.
  - Evaluar y analizar diferentes tecnologías relevantes para un proyecto, con el objetivo de seleccionar las más adecuadas. Esto implica la investigación y comprensión de diversas opciones tecnológicas, considerando factores como la escalabilidad, seguridad y compatibilidad.
  - Desarrollar sólidas habilidades para la planificación de proyectos. Esto permitirá una ejecución eficiente y exitosa del proyecto, asegurando que se cumplan los objetivos establecidos.

Por otro lado, como objetivo secundario debo destacar el genuino interés en saber qué futuro le espera a la tecnología blockchain. Es decir, si esta nueva tecnología será capaz de remodelar la sociedad tal y como la conocemos, o si por el contrario, será una tecnología útil para ciertos campos, pero sin suponer una revolución como tal.

1.4. Estructura de la memoria

El presente documento está estructurado en los siguientes capítulos:

- **Capítulo 1: Introducción**
- **Capítulo 2: La Blockchain** Este capítulo ofrece una explicación detallada de los fundamentos de la tecnología blockchain. Se abordan conceptos clave como la descentralización, la estructura de la cadena de bloques, los mecanismos de consenso y el proceso de minado de bloques. Además, se analiza la plataforma Ethereum, incluyendo su Máquina Virtual (EVM), contratos inteligentes (smart contracts), gas y aplicaciones descentralizadas (Dapps). Finalmente, se explora la aplicación de la blockchain en la cadena de suministros.
- **Capítulo 3: Tecnologías empleadas** Se detallan las tecnologías y herramientas utilizadas en el desarrollo del proyecto. Esto incluye Ethereum, Geth, Web3.js, ReactJs, React Router, Truffle, Solidity, Remix, Material UI, Visual Studio Code, MetaMask y Astah. Para cada tecnología, se explica su relevancia y aplicación en el proyecto.
- **Capítulo 4: Tecnologías descartadas** En este capítulo se presentan las tecnologías y herramientas consideradas pero finalmente descartadas para el proyecto, como Ganache, MySQL, Axios, React Bootstrap, Firebase y Ethers.js. Se justifican las razones por las cuales estas tecnologías no fueron seleccionadas.

- **Capítulo 5: Planificación** Este capítulo describe la metodología ágil adoptada para la planificación y desarrollo del proyecto. Se abordan los diferentes prototipos utilizados, tanto desechables como evolutivos, y se detallan las etapas del proyecto: fase inicial, fase intermedia y fase final.
- **Capítulo 6: Análisis** Se realiza un análisis exhaustivo del caso de negocio, identificando los actores involucrados y los requisitos del sistema, tanto funcionales como no funcionales. Además, se desarrollan los casos de uso y se realiza una estimación de los costes asociados al proyecto.
- **Capítulo 7: Diseño** Este capítulo se centra en la fase de diseño de la aplicación. Se describe la arquitectura de una Dapp, la arquitectura de los smart contracts y la arquitectura del frontend. Se proporciona un modelo de dominio de los smart contracts y se detalla la estructura de los componentes del frontend.
- **Capítulo 8: Implementación** Se describe la implementación de los diferentes componentes del proyecto. Esto incluye la codificación de los smart contracts (App, StepChain y SupplyChain), la implementación en la blockchain y el desarrollo del frontend utilizando React y MetaMask. Se detalla la interacción con los contratos y la estructura de archivos de la aplicación.
- **Capítulo 9: Conclusiones** En este capítulo final se presentan las conclusiones del proyecto, valorando los resultados obtenidos y reflexionando sobre el trabajo realizado. Además, se plantean posibles líneas de trabajo futuro para continuar desarrollando y mejorando el proyecto.

Finalmente, se adjuntan un manual de despliegue de la aplicación, y un manual de usuario donde se detalla el uso de la misma.

Capítulo 2

La Blockchain

2.1. Fundamentos

Podríamos definir una blockchain como una base de datos descentralizada, segura y pública, la cual es mantenida por cualquiera que esté conectado a ella. Sin embargo, esta definición se queda corta puesto que cuando se habla de blockchain, el concepto al que queremos hacer referencia es realmente al ecosistema que permite, precisamente, conseguir una base de datos descentralizada segura y pública. Para entender cómo conseguir esta blockchain, es necesario comprender las motivaciones y las partes que la conforman.

2.1.1. Descentralización

Antes de nada, hay que explicar qué ventajas aporta un sistema descentralizado frente a uno centralizado, pues son las razones principales que justifican usar la tecnología blockchain. Estas son las siguientes:

- **Resistencia a la censura:** En un sistema descentralizado, la información y las operaciones no están controladas por una entidad central, lo que dificulta la posibilidad de censurar o limitar el acceso a ciertos usuarios o participantes. La red se vuelve más resistente a intentos de bloquear o restringir el flujo de datos.

- **Seguridad:** La descentralización implica que la información se distribuye entre numerosos nodos de la red. Esto hace que sea más difícil para un atacante comprometer la seguridad de toda la red, ya que tendría que controlar una mayoría significativa de los nodos.

- **Transparencia y confianza:** Al descentralizar el control y los datos, cualquier cambio en el sistema es visible para todos los nodos de la red. La confianza se construye a través de la verificación descentralizada, ya que cada nodo puede comprobar la validez de los datos de forma independiente.

- **Resiliencia ante fallos:** La descentralización mejora la resiliencia del sistema ante posibles fallos o ataques. Si un nodo falla, los demás pueden continuar operando, garantizando la continuidad del sistema. Esto contrasta con los sistemas centralizados, donde la falla de un punto central puede resultar en la paralización de todo el sistema.

- **Disponibilidad:** Al distribuir la responsabilidad y la gestión de datos entre múltiples nodos, se mejora la disponibilidad del sistema. La redundancia de la información en varios nodos asegura que, incluso en situaciones de fallo o ataques, la red pueda seguir funcionando y ofreciendo sus servicios.

Para conseguir la descentralización de nuestra base de datos, fundamentaremos su almacenamiento en una red *Peer to Peer* o P2P. En este tipo de redes, no hay un servidor central al cual nos conectemos para obtener la información, sino que esta se distribuye entre multitud de **nodos** interconectados entre sí. Estos nodos pueden actuar tanto de cliente como de servidor, estableciendo así una relación de igual a igual.

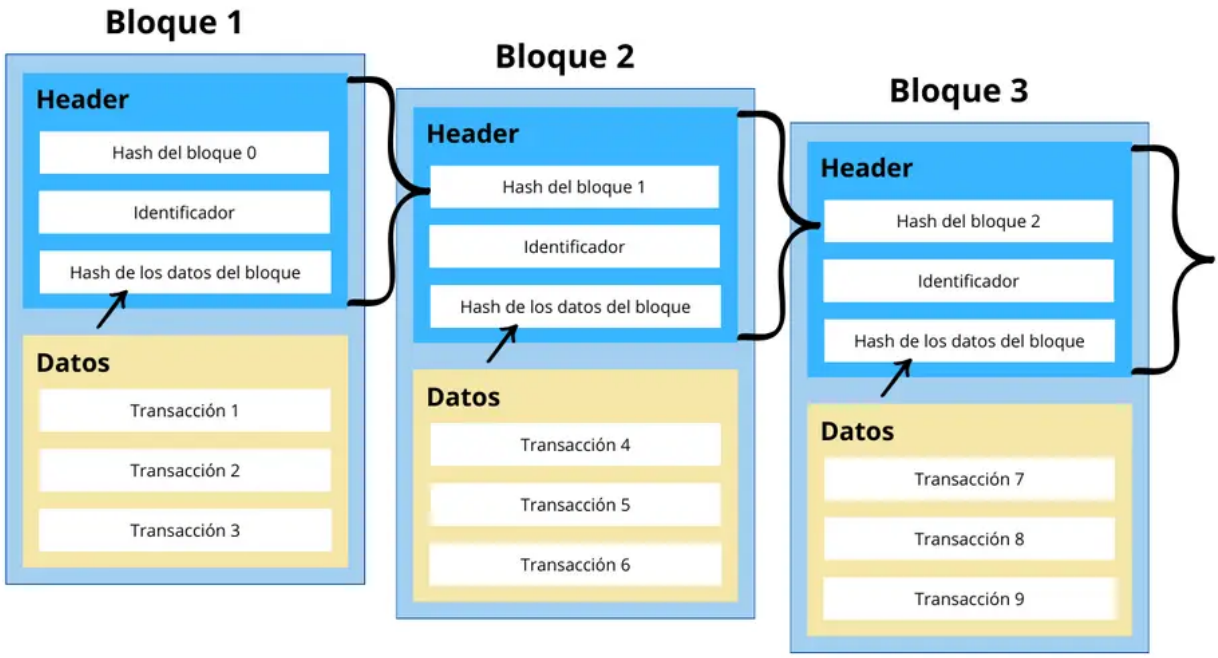


Figura 2.1: Estructura de bloques en una blockchain

2.1.2. Cadena de bloques

Ya tenemos dónde almacenar nuestra información, pero tenemos que aclarar qué vamos a almacenar. Para ello vamos a tomar como referencia la blockchain más conocida, Bitcoin. En ella la información que se almacena no es nada más que un libro de contabilidad, en el cual se guardan todas las transacciones efectuadas. Estas transacciones constan de un emisor, un receptor y la cantidad de bitcoins transferidos. De esta manera, se puede llevar la contabilidad de todos los miembros de la red.

Ahora solo nos falta explicar cómo estructurar estos datos. Como se muestra en la Figura 2.1, las transacciones creadas en un mismo intervalo de tiempo las agruparemos en lo que denominaremos bloque. Este bloque contendrá además, una cabecera con un hash obtenido de encriptar las transacciones del propio bloque, un identificador o "nonce" y un hash de la encriptación de la cabecera de otro bloque. De esta manera conseguimos una cadena de bloques, en la cual, si modificamos cualquier dato de uno de ellos, se verá reflejado en los bloques posteriores.

Por lo tanto, si queremos enviar una determinada cantidad de bitcoins, lo que debemos hacer es crear una transacción con la información pertinente, añadirla a un bloque y conectarlo al final de la cadena. Por último hay que propagar la nueva cadena por la red de nodos para que el cambio quede registrado.

Sin embargo, no podemos olvidar que nuestra red está conformada de nodos, gestionados por multitud de personas, que podrían querer añadir bloques diferentes a la cadena. Esto en sí no sería un problema puesto que podríamos encadenar uno detrás de otro. Pero por desgracia, nuestra red tiene latencias que dificultan la comunicación entre nodos, lo cual podría llevar a que los nodos no tuviesen la misma información guardada. Es por ello necesario elegir entre todos los nodos, cuál va a ser el siguiente bloque en añadirse a la cadena.

2.1.3. Mecanismos de consenso

Los mecanismos de consenso son los protocolos que nos permiten poner de acuerdo a toda una red de nodos sobre qué información es válida y, por tanto, hay que almacenar. Actualmente existen varios de estos protocolos, que pueden ser empleados indistintamente en una blockchain, teniendo diferentes ventajas y desventajas. Estos son los siguientes:

- **Proof of Work (PoW):** La idea que subyace a este protocolo, es permitir votar entre todos los miembros de la red cuál será el último bloque que añadiremos a la cadena. Por motivos de seguridad, cada nodo que quiera votar ha de resolver una prueba criptográfica, que requerirá gastar un coste trabajo, o computacional.

De esta manera se evita el conocido como Sybil Attack[4], donde un atacante podría hacerse con el control de muchos nodos con el fin de que la votación se decantase a su favor.

- **Proof of Stake(PoS):** Este protocolo es similar al anterior, con la diferencia de que no hay que resolver una prueba criptográfica, y por tanto, los costes de energía asociados a validar los bloques son notablemente más bajos. En este protocolo, el bloque se elige de forma aleatoria, entre los participantes que quieran "apostar" una cantidad de sus fondos, teniendo en cuenta que, cuantos más fondos apuestes, más probabilidades tiene tu bloque de salir elegido.
- **Proof of Authority(PoA):** A diferencia de los protocolos anteriores, en este no se produce una votación. Se establece un nodo validador que elige cual será el siguiente bloque. El problema de este protocolo es que perdemos una parte importante de descentralización de la red y debemos depositar nuestra confianza en el nodo validador.

2.1.4. Minado de un bloque

Ahora que ya tenemos un mecanismo de consenso con el cual elegir qué bloque añadir, podemos retomar nuestro ejemplo de Bitcoin. Esta blockchain usa Proof of Work, de tal manera que quién primero resuelva la prueba criptográfica consigue el derecho de añadir su bloque a la cadena, y además añadir una transacción especial en la cual se le otorga una cantidad de bitcoins. Esta recompensa es necesaria porque recordemos que resolver esta prueba tiene un coste computacional, generalmente alto. Este proceso de validar un bloque, resolver la prueba criptográfica y añadirlo a la cadena es denominado **minado**. Cabe mencionar, que la prueba criptográfica consiste concretamente en encontrar un hash de bloque asociado a la información del mismo, que empiece por dieciocho ceros. Para ello es necesario iterar sobre el identificador del bloque, hasta encontrar uno que cumpla la condición. De esta manera, conseguimos que los bloques ya creados sean inmutables, pues un cambio en cualquiera de sus datos acarrearía un cambio en toda la cadena, que sería fácilmente detectable.

2.2. Ethereum

Ethereum es una plataforma de código abierto, basada en la tecnología blockchain, que permite a desarrolladores crear y desplegar aplicaciones descentralizadas, comúnmente llamadas Dapps. Además, los usuarios pueden interactuar con estas, a través de tokens llamados ether(ETH). Para hacer esto posible, incorpora nuevas ideas a las ya planteadas en la sección de fundamentos.

2.2.1. Ethereum Virtual Machine (EVM)

Anteriormente, vimos como Bitcoin almacenaba un libro de contabilidad usando la tecnología blockchain. Ethereum, por su parte, pretende servirse de la blockchain para crear un ordenador distribuido el cual es llamado Máquina Virtual de Ethereum. Todos los nodos participantes de la red guardan una copia de estado de la máquina, de manera que, cuando hay que resolver una solicitud de cálculo, pueda resolverla cualquier nodo. Una vez resuelto el cálculo el cambio producido en la EVM es propagado a toda la red y el resto de nodos se encarga de verificar y validar dicho cambio.

2.2.2. Smart Contracts

Ethereum da soporte a dos tipos de cuentas: Externas y contratos. Las primeras son aquellas que pertenecen a un individuo y por tanto, pueden almacenar, enviar y recibir Ether. Este tipo de cuentas son las mismas que existen en Bitcoin.

Por otra parte, los contratos o *smart contracts* son cuentas especiales que contienen código reutilizable, el cual nos permite cambiar el estado de la EVM. Estos contratos son publicados por desarrolladores, acarreando un coste. Posteriormente, cualquiera puede enviar una transacción al contrato, para así ejecutar el código que contiene.

¿Qué hace a estas piezas de código tan particulares? Pues que al ser públicas y estar soportadas en una red descentralizada, las convierte en una especie de contrato. Sin embargo, a diferencia de un contrato convencional,

donde pueden ser necesarias terceras partes, como notarios o jueces que lo hagan cumplir, un smart contract no necesita de intermediarios, puesto que es autoejecutable. Es decir, si se dan ciertas condiciones, el contrato se ejecuta automáticamente llevándose a cabo las acciones pactadas anteriormente en el. Por ejemplo, imaginemos un caso en el que se ha de realizar un transporte de congelados, en el cual, si se rompe la cadena del frío, el transportista debe pagar una penalización por las pérdidas causadas. En un contrato convencional, necesitaríamos de un perito que compruebe, que efectivamente se ha roto la cadena del frío, y en casos límites, un juez que haga cumplir el contrato. Sin embargo, un *smart contract* podría detecar con un sensor que la cadena del frío se ha roto, y automáticamente penalizar al transportista, enviando Ether de su cuenta a la del vendedor. Como podemos observar, esto es una enorme ventaja a la hora de resolver disputas en la cual se necesiten terceras partes que además llevan un coste asociado.

2.2.3. Gas

En Ethereum, toda transacción tiene un coste computacional asociado. Este coste varía dependiendo del código que mandemos ejecutar a través de los contratos. Para evitar problemas de saturación de ejecución de código basura o bucles infinitos, hay que pagar una pequeña tarifa. Por ello, todas las transacciones que llaman a contratos, tienen asociadas una cantidad de Ether que estamos dispuestos a pagar para así poder ejecutar el código. Este Ether es el denominado **gas**. A medida que se va ejecutando código, el gas va disminuyendo como si de gasolina se tratase, hasta que este se acaba, o termina la ejecución de código que solicitamos.

2.2.4. Aplicaciones descentralizadas(Dapps)

Dado que Ethereum soporta la ejecución de código a través de los *smart contracts*, es posible crear aplicaciones sobre esta plataforma. Y puesto que el soporte es una blockchain, estas aplicaciones son, por ende, descentralizadas. La estructura de estas aplicaciones descentralizadas(Dapps) más ampliamente explicada en la Sección 7.1, suele constar de una interfaz de usuario externa a la blockchain, y una lógica implementada con *smart contracts*. Esto se debe a que ejecutar código sobre la blockchain tiene un coste alto, por lo que se intenta reducir la lógica implementada en smart contracts a la mínima necesaria. De esta manera, la aplicación se beneficia de las ventajas de la descentralización, a la vez que el coste de la misma no se dispara.

2.3. Blockchain y cadena de suministros

Las cadenas de suministro representan el conjunto de procesos interconectados que gestionan el flujo de bienes, servicios, información y capital desde la fase de producción hasta el consumidor final. Estas complejas redes abarcan todas las etapas del ciclo de vida de un producto, desde la obtención de materias primas hasta la entrega al cliente, involucrando a diversos actores y recursos a lo largo de todo el trayecto. En esencia, las cadenas de suministro son como una red de conexiones estratégicas entre proveedores, fabricantes, distribuidores, minoristas y consumidores. Su objetivo principal es garantizar la disponibilidad de productos o servicios en el lugar correcto, en el momento adecuado y en las condiciones deseadas, al mismo tiempo que se busca optimizar los costos y mejorar la eficiencia operativa.

Estas cadenas son fundamentales en el ámbito empresarial y económico, ya que influyen directamente en la competitividad de las organizaciones y en la satisfacción del cliente. Una gestión eficaz de la cadena de suministro implica la coordinación y sincronización de todas las actividades involucradas, desde la planificación de la producción hasta la gestión de inventarios, el transporte y la distribución.

Actualmente, el empleo de la tecnología en el diseño y gestión de las cadenas de suministro modernas es fundamental. Desarrollar y emplear sistemas software, permite a las empresas automatizar sus procesos,agilizando y abaratando costes. Las tecnologías de la información juegan un papel crucial en el diseño y la gestión de las cadenas de suministro modernas. Sistemas avanzados de software, el uso de datos en tiempo real, la automatización y la inteligencia artificial permiten una mayor visibilidad y control sobre los flujos de productos y la información asociada. Esto contribuye a la toma de decisiones más informada y ágil, lo que es esencial en un entorno empresarial dinámico y globalizado.

No obstante, las cadenas de suministro también enfrentan desafíos significativos, como la volatilidad del mercado, los cambios en la demanda, eventos disruptivos (como desastres naturales o pandemias), la complejidad de las operaciones internacionales y la necesidad de adoptar prácticas sostenibles. La resiliencia y flexibilidad son aspectos cada vez más valorados en el diseño de cadenas de suministros capaces de adaptarse a las condiciones cambiantes.

Capítulo 3

Tecnologías empleadas

3.1. Ethereum

Como ya se ha explicado en la Sección 2.2, Ethereum es una plataforma de código abierto basada en la blockchain, que permite a desarrolladores crear y desplegar aplicaciones descentralizadas y a usuarios interactuar con estas a través de tokens llamados ether(ETH). Ethereum permite, por tanto, implementar *smart contracts*, que pueden ser usados por cualquier desarrollador, haciendo florecer un gran ecosistema para el desarrollo de software.

Estas características, junto con su temprana creación en 2015, han hecho que su posición en el mundo de la blockchain sea notoria. Tanto es así que, muchos proyectos emergentes usan la red Ethereum como soporte, antes de implementar su propia red.

Por lo tanto, que sea programable, junto con la enorme comunidad y la gran cantidad de documentación [4], han decantado la balanza a su favor para ser elegida como la tecnología angular sobre la que se realiza este proyecto.

3.2. Geth

Geth(Go Eth) es un cliente desarrollado en Go que implementa el protocolo Ethereum. Esta tecnología es la que nos permitirá desplegar nodos para conectar con una blockchain Ethereum o crear una propia. Geth se encarga de realizar las operaciones típicas de un cliente de blockchain, como la sincronización con la cadena de bloques, la creación de nuevos bloques a través del minado o el envío y recepción de transacciones.

A mayores de la funcionalidad básica que han de proporcionar los clientes Ethereum, proporciona herramientas útiles para la creación y configuración de redes privadas, así como una consola de comandos JavaScript que permite interactuar con la red en un nivel de abstracción superior, y por ello, de forma más intuitiva. También hay que mencionar que permite configurar varios métodos de validación de bloques, basados en PoW (*Proof-of-work*), PoA(*Proof-of Authority*) o PoS(*Proof-of Stake*).

Siendo un cliente oficial, es el más extendido, y por ello cuenta con documentación [18](aunque no muy elaborada) y multitud de guías creadas por usuarios.

Por estos motivos, se ha decidido implementar la red blockchain usando esta tecnología.

3.3. Web3.js

Web3.js es un conjunto de librerías que implementan una API JavaScript para interactuar con la red Ethereum. Estas librerías son las encargadas de comunicarse con un nodo de la red y de proporcionar funciones para usar contratos o enviar ETH. De esta manera, elevamos el nivel de abstracción evitando así usar llamadas JSON-RPC. Nuevamente, Web3.js es ampliamente utilizada por la comunidad, y tiene asociada una documentación amplia con gran cantidad de ejemplos [46]. Cierto es que existe otra librería llamada EthersJs que compite al mismo nivel en uso y documentación. Sin embargo, se ha elegido usar la primera de forma arbitraria puesto que es totalmente válida para las necesidades del proyecto.

3.4. ReactJs

React es una biblioteca de código abierto escrita en JavaScript que se utiliza para construir interfaces de usuario interactivas y reactivas. Fue creada, es desarrollada y mantenida por Facebook. Está enfocada a la creación de interfaces de usuario que necesitan actualizarse de manera eficiente, en respuesta a eventos o cambios en los datos. Para ello se apoya en una arquitectura basada en componentes, que además permiten la reutilización de código.

El *frontend* desarrollado en este proyecto está implementado usando React, y por tanto es necesario ahondar un poco más en cómo funciona.

3.4.1. Componentes

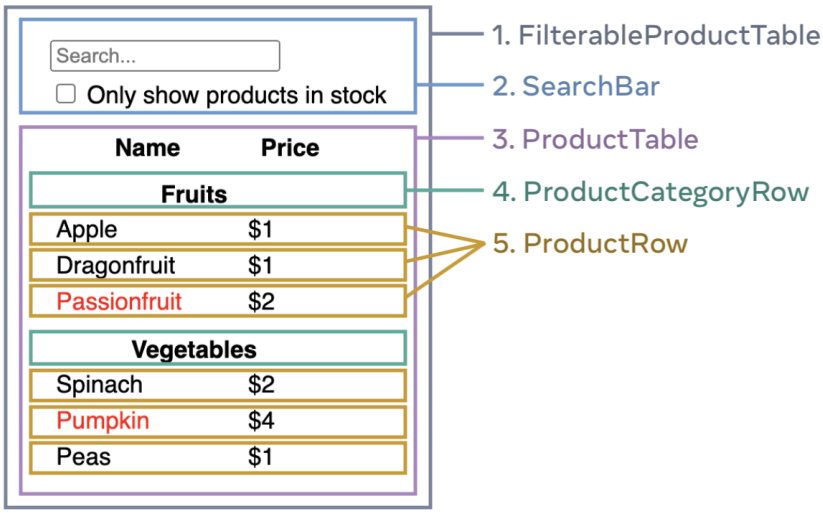


Figura 3.1: Ejemplo de componentes en React

Como podemos observar en la Figura 3.1, las interfaces creadas en React están basadas en componentes. Estos componentes a su vez pueden estar formados por otros componentes, creando así una estructura jerárquica. Además, los componentes son autónomos y pueden contener un estado propio, lo cual les permite ser reutilizables. Para conseguirlo es necesario el uso del lenguaje JSX. Este, es una extensión de Javascript que permite insertar HTML y CSS directamente en el código. Gracias a JSX, los componentes pueden tener una estructura basada en el código HTML que contienen y además, una lógica escrita en JavaScript.

3.4.2. Ciclo de vida

Teniendo presente la Figura 3.2, todo componente en React tiene un ciclo de vida que podemos dividir en tres partes:

1. **Montaje:** Fase en la cual el componente se crea con la información que recibe a través de las propiedades(props), y posteriormente se renderiza.

2. **Actualización:** Una vez el componente es montado, entra en un bucle, a la escucha de posibles cambios. Si los props o el estado del componente son modificados, se inician unas rutinas que gestionan el cambio y, de ser necesario, renderizan el componente de nuevo.

3. **Desmontaje:** Esta es la fase final del componente, en la cual este deja de ser renderizado y procede a liberar los recursos usados.

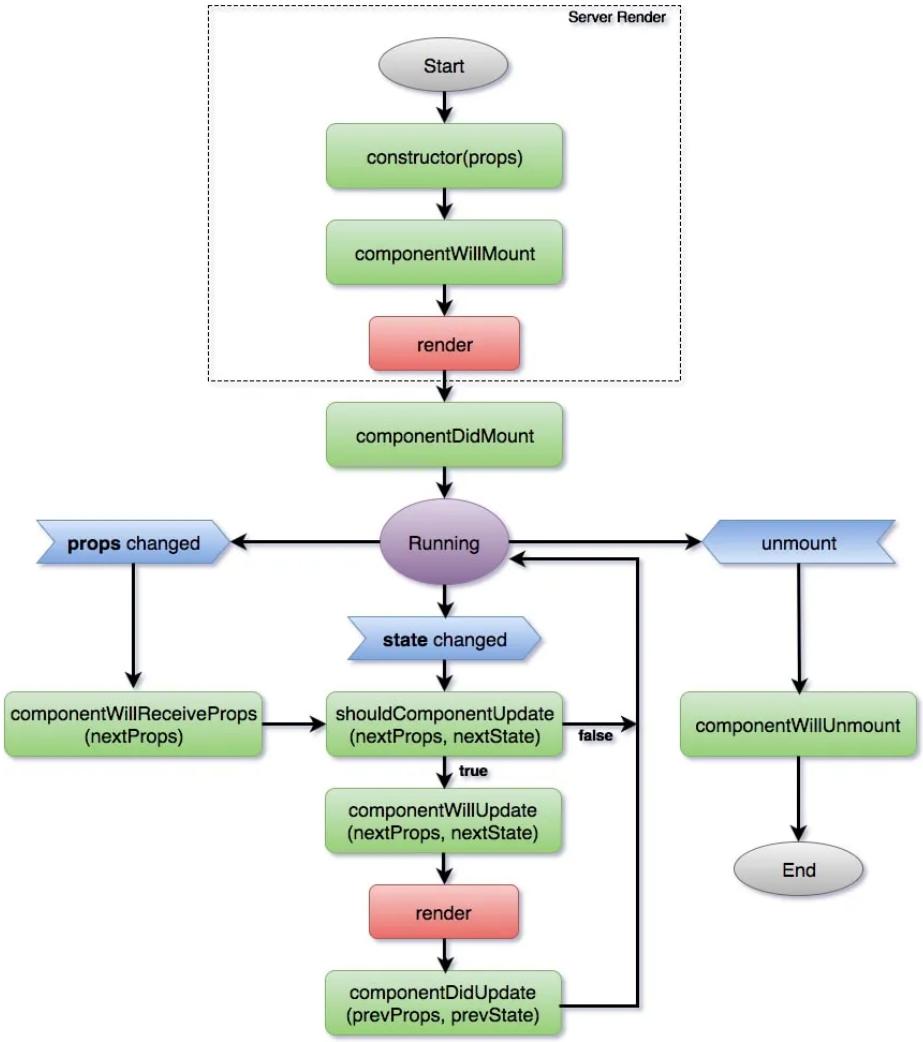


Figura 3.2: Ciclo de vida de un componente React [24]

3.4.3. Jerarquía y Virtual DOM

Anteriormente mencioné que los componentes tienen una jerarquía, ya que unos forman parte de otros más grandes. Esta jerarquía es importante, pues las comunicaciones que tienen los componentes entre sí es unidireccional y descendente. Es decir, la información es transmitida desde el componente padre hacia el componente hijo. Esto se hace a través de las propiedades(props), gracias a las cuales los componentes son notificados de los cambios que se producen y así poder actualizarse en consonancia.

Para hacer más eficiente las actualizaciones que se llevan a cabo en el DOM(Document Object Model), React emplea el denominado Virtual DOM, que es una representación ligera del DOM. De esta manera, cada vez que se produce un cambio, React compara el Virtual DOM con el DOM para ver dónde hay diferencias. Una vez localizadas, React manda una señal solo a los componentes que han de ser renderizados de nuevo, y así mostrar los cambios en pantalla. Estos componentes, a su vez mandan una señal a sus componentes hijos, generando así una cascada de actualizaciones, solo en las ramas que se han visto afectadas por los cambios. Este proceso podemos apreciarlo en la figura *figura*. Por lo tanto, gracias al Virtual DOM, evitamos modificar todos los componentes, consiguiendo así un gran ahorro de recursos.

3.4.4. Hooks

Los Hooks son una característica introducida recientemente en React que permiten a los desarrolladores usar el estado y otras características de React sin necesidad de escribir clases. Antes de los Hooks, la lógica de estado solo podía ser utilizada en componentes de clase. Sin embargo, con la introducción de los Hooks, ahora es posible utilizar el estado y otras características de React en componentes funcionales.

Algunos de los Hooks más comunes empleados en este proyecto son:

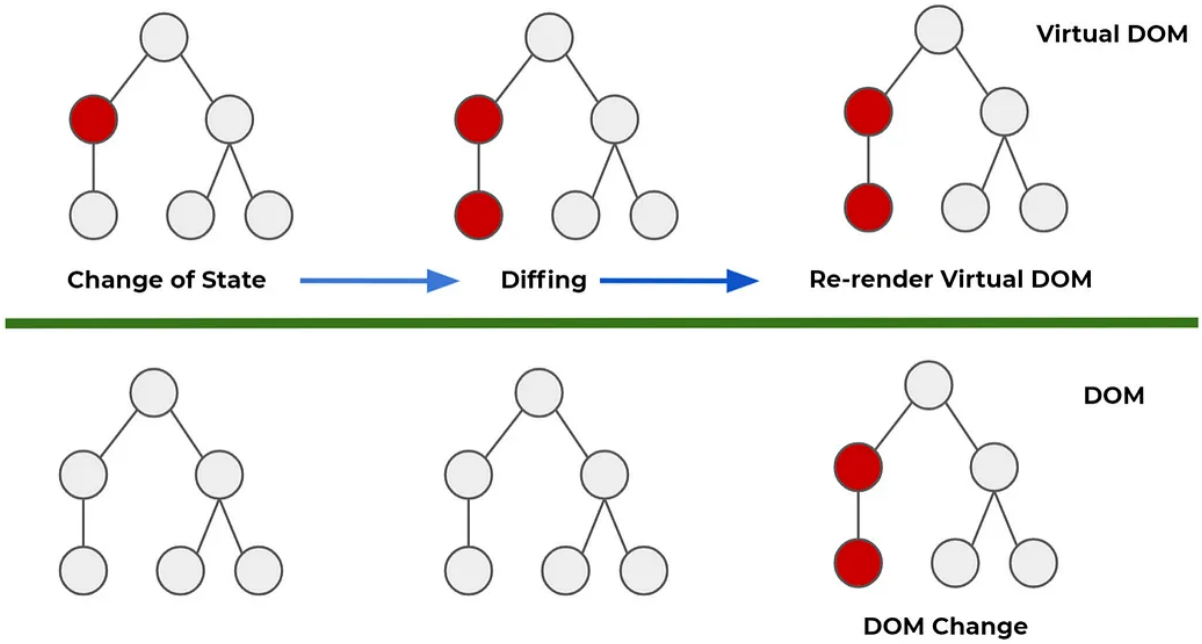


Figura 3.3: React Virtual DOM [38]

- useState: Permite a los componentes funcionales tener estado local. Con este Hook, puedes añadir estado a tus componentes sin necesidad de utilizar clases.

- useEffect: Permite a los componentes funcionales realizar efectos secundarios en la aplicación, como acceder a datos remotos, suscribirse a eventos o manipular el DOM. Este Hook se ejecuta después de cada renderizado y puede ser usado para limpiar recursos cuando el componente es desmontado.

- useContext: Permite acceder al contexto de React dentro de un componente funcional. El contexto de React es una forma de pasar datos a través del árbol de componentes sin necesidad de pasar props manualmente en cada nivel.

Por tanto, los Hooks proporcionan una forma más simple y concisa de escribir componentes funcionales, lo que ha llevado a una mayor adopción de este estilo de programación en la comunidad de React.

3.5. React Router

React Router es una biblioteca de enrutamiento diseñada para aplicaciones web desarrolladas con React. Permite a los desarrolladores definir rutas en su aplicación y vincular estas rutas con componentes específicos. Esto significa que puedes crear aplicaciones de una sola página (SPA) donde el contenido de la página se actualiza dinámicamente sin necesidad de recargar la página completa.

React Router facilita la navegación dentro de la aplicación al mapear las URL del navegador a componentes React específicos, lo que permite que los usuarios naveguen entre diferentes partes de la aplicación sin perder el estado de la misma. Además, proporciona características avanzadas como enrutamiento anidado, redirecciones, gestión de parámetros en la URL.

Por otro lado, React Router también ofrece componentes ya definidos para resolver las tareas de enrutamiento y navegación de la aplicación. Además, el desarrollador puede modificarlos para que se ajusten a unas necesidades concretas.

Se ha elegido el uso de esta biblioteca ya que agiliza el desarrollo de los procesos de enrutamiento y ofrece una documentación sólida acerca de los componenentes que facilita.

3.6. Truffle

Truffle es un conjunto de herramientas que simplifican los procesos de desarrollo relacionados con *smart contracts* en redes basadas en Ethereum. [43]

Como puntos principales a destacar podemos mencionar los siguientes:

- **Compilación y migración** Permite al desarrollador compilar y migrar los *smart contracts* de forma sencilla, pudiendo gestionar los cambios que se van produciendo en estos.
- **Testing** Da soporte para crear pruebas de testeo de *smart contracts*, Esta característica es fundamental en el desarrollo de los mismos, puesto que cualquier defecto que estos puedan tener, conllevan pérdidas directas ya que la ejecución de transacciones va asociada a un gasto computacional.

A medida que se ha ido desarrollando este proyecto, han implementado nuevas herramientas, algunas de ellas útiles, pero no empleadas puesto que ya se ha encontrado alternativa. Por citar algunas de estas nuevas herramientas, puedo destacar una integración con Metamask, y la capacidad de interactuar con *smart contracts* de forma directa, habilitando así un mejor testeo de estos.

3.7. Solidity

Para el desarrollo de *smart contracts*, hace falta un lenguaje que esté especializado en una arquitectura blockchain. Este lenguaje es Solidity, creado a la vez que Ethereum, para dar soporte al desarrollo de *smart contracts*. Entre sus principales características, destaca que es orientado a objetos, por lo que permite herencia, y el uso de librerías creadas por usuarios [44]. Además, dado que los errores en la blockchain pueden ser muy costosos, incluye características específicas que facilita a los desarrolladores escribir *smart contracts* de forma más segura.

Actualmente, es prácticamente el único lenguaje usado en redes Ethereum y por ello es el que he decidido emplear.

3.8. Remix

Remix es el entorno de desarrollo por excelencia para Ethereum [35]. Este IDE permite escribir *smart contracts* usando Solidity. Además los compila y despliega en un entorno virtual donde poder depurar errores, todo ello desde un navegador web, como Firefox.

Sus principales características son:

- Editor de código en Solidity
- Compilación y despliegue de *smart contracts* en un entorno de pruebas.
- Facilita la depuración de *smart contracts* puesto que provee de un entorno que da acceso rápido y fácil a toda la información almacenada en la blockchain.
- Implementa una interfaz gráfica para interactuar con los *smart contracts*, lo cual ayuda mucho a la hora de desarrollarlos.

Estas características lo hacen particularmente interesante para nuevos desarrolladores que necesitan familiarizarse con los complejos procesos de la blockchain. Por tanto, el uso dado a Remix en este proyecto es principalmente la edición, ejecución y depuración de *smart contracts*.

3.9. Material UI

Material UI [12] es una librería de componentes React, que facilita la tarea de diseño de aplicaciones, implementando la filosofía Material Design de Google. Esta librería de código abierto, ofrece multitud de componentes con un diseño uniforme, que permiten usar directamente en la aplicación a desarrollar. Además, estos componentes pueden ser personalizados con facilidad, pudiendo adaptarse a las necesidades concretas de la aplicación.

Al ser una de las librerías más utilizadas en el desarrollo de aplicaciones basadas en React y contar con una extensa documentación, donde se detalla cómo usar cada tipo de componente y cómo poder personalizarlo según las necesidades, se ha elegido como herramienta para darle un diseño homogéneo a la aplicación desarrollada en este proyecto.

3.10. VisualStudio Code

Todo desarrollo de una aplicación requiere de un editor de código que brinde al programador de herramientas para la escritura de código. Visual Studio Code es un editor usado ampliamente pues ofrece soporte a gran cantidad de lenguajes, entre ellos JavaScript y Solidity. Además, complementa la edición de código con extensiones de lo más variopintas, que facilitan el desarrollo. Algunas de las usadas en este proyecto son paletas de colores para edición de código en React, herramientas de testeo para aplicaciones React y GitLens para gestionar las versiones de código.

3.11. MetaMask

Una pieza clave en las redes blockchain son las billeteras, que permiten gestionar cuentas donde almacenar los activos. MetaMask [27] no es más que una billetera hecha extensión de navegador web, que da soporte a redes basadas en Ethereum. Al estar integrada en el navegador, permite conectar directamente con aplicaciones descentralizadas y usar nuestros activos en ellas. Por otro lado, también permite conectarnos a diferentes redes, comprar y transferir sus tokens.

Sus principales características son las siguientes:

- Actúa como extensión de navegador, lo cual permite una conexión con Dapps.
- Permite gestionar diferentes redes basadas en Ethereum, pudiéndote conectar a la red principal, o por el contrario, a redes privadas.
- Da soporte a la gestión de múltiples cuentas en múltiples redes.
- Permite comprar Tokens.
- Es posible realizar transferencias entre cuentas.
- Gestiona las a contratos que pueden hacer las Dapps, pudiéndolas aceptar o rechazar.

MetaMask, es una de las billeteras más extendidas, por su facilidad de uso para los usuarios y por la documentación para desarrolladores que facilita la integración con la herramienta.

En este proyecto, MetaMask es una herramienta clave, pues es la encargada de gestionar los activos de los usuarios, asi como de hacer de intermediario entre la Dapp desarrollada y la blockchain.

3.12. Astah

Astah es una herramienta de modelado visual y diseño de software utilizada por desarrolladores y profesionales de la ingeniería de software para crear diagramas UML (Lenguaje de Modelado Unificado), diagramas de flujo, diagramas de casos de uso, entre otros. Astah ofrece una variedad de características y herramientas que facilitan la visualización y el diseño de sistemas de software complejos.

Con Astah, se pueden crear modelos visuales de sus sistemas, lo que les permite comprender y comunicar de manera efectiva la estructura, el comportamiento y las relaciones dentro del software. La herramienta es utilizada en todas las etapas del ciclo de vida del desarrollo de software, desde la captura de requisitos y el diseño inicial hasta la implementación y la documentación.

En este proyecto, se ha empleado principalmente para diseñar los diagramas de la fase de análisis y diseño, es decir, diagramas de casos de uso, diagramas de secuencia y el modelo de dominio.

Capítulo 4

Tecnologías descartadas

Durante el desarrollo de este proyecto se han usado diversas tecnologías que por diferentes motivos, los cuales se explicarán a continuación, se han ido descartando. Sin embargo, al haber sido utilizadas, en mayor o menor medida, es relevante comentar algunas de ellas.

4.1. Ganache

Ganache es una herramienta de blockchain que se utiliza principalmente para desarrollar y probar Dapps en entornos de desarrollo local. Es parte del ecosistema de Truffle Suite, que es una suite de herramientas populares utilizadas por desarrolladores de Ethereum.

Las principales características y utilidades de Ganache son las siguientes:

- Blockchain Local: Ganache proporciona una blockchain local privada que los desarrolladores pueden ejecutar en sus propias máquinas. Esto permite a los desarrolladores probar sus Dapps en un entorno controlado y sin necesidad de interactuar con la blockchain principal de Ethereum.
- Entorno de Desarrollo Amigable: Ganache viene con una interfaz gráfica de usuario que hace que sea fácil de usar, especialmente para aquellos desarrolladores que no están familiarizados con la línea de comandos. Además, también se puede acceder a través de la línea de comandos para una mayor flexibilidad.
- Redes Personalizables: Los desarrolladores pueden configurar Ganache para que se ajuste a sus necesidades específicas, como ajustar el tamaño del bloque, el número de cuentas disponibles y la velocidad de confirmación de transacciones.
- Datos Simulados: Ganache proporciona una serie de cuentas pre-cargadas con ETH (Ether) simulado, lo que permite a los desarrolladores simular transacciones y pruebas de contratos inteligentes sin la necesidad de usar ETH real.
- Rápido Despliegue y Reinicio: Ganache permite a los desarrolladores reiniciar rápidamente la blockchain local con un solo clic, lo que facilita el ciclo de desarrollo y prueba de las Dapps.

Sin embargo, Ganache al ofrecer una red Ethereum de caja, no permite ajustar muchos parámetros de la configuración de la red. Por otro lado, al ofrecer un entorno completamente configurado, complica el aprendizaje del funcionamiento de una red Ethereum.Es principalmente por estos motivos que finalmente se ha usado Go Ethereum sobre Ganache.

4.2. MySql

MySQL es un sistema de gestión de bases de datos relacional de código abierto ampliamente utilizado en una variedad de aplicaciones y entornos. Permite almacenar, organizar y recuperar datos de manera eficiente utilizando tablas con filas y columnas. MySQL utiliza el lenguaje SQL para interactuar con la base de datos y

ofrece características como escalabilidad, rendimiento, seguridad y soporte de comunidad, lo que lo convierte en una opción popular para proyectos de todos los tamaños.

Sin embargo, utilizar una base de datos MySQL perjudica a la descentralización de la aplicación, puesto que MySQL necesita de un servidor donde se almacene la información. Además, como se explica en la Sección 7.1, es posible evitar el uso de servidores que centralizan la información, apoyándose en el uso directo de la blockchain para guardar los datos.

4.3. Axios

Axios es una biblioteca JavaScript que permite realizar solicitudes HTTP desde el navegador o desde Node.js. Con Axios, los desarrolladores pueden enviar y recibir datos desde servidores remotos de manera asíncrona. Esta biblioteca proporciona una interfaz simple y fácil de usar para realizar operaciones HTTP, como hacer solicitudes GET, POST, PUT, DELETE, entre otras. Además, Axios ofrece características como la gestión automática de conversiones JSON, manejo de errores, interceptores de solicitudes y respuestas, y compatibilidad con promesas, lo que la convierte en una herramienta versátil y poderosa para realizar comunicaciones de red en aplicaciones web y móviles.

En el ámbito de este proyecto, Axios fue usado como método de conexión del frontend desarrollado en React, con el backend que pretendía sustentar el uso de la base de datos MySQL. Por tanto, al desechar el enfoque de usar MySQL, Axios dejó de ser necesario, y por ende, descartado.

4.4. React Bootstrap

React Bootstrap es una biblioteca que combina React con la biblioteca de estilos y componentes Bootstrap. Permite a los desarrolladores crear interfaces de usuario receptivas y atractivas de manera rápida y sencilla utilizando componentes predefinidos de Bootstrap dentro de sus aplicaciones React. Esto permite a los desarrolladores aprovechar las ventajas de la flexibilidad y el diseño receptivo de Bootstrap junto con las capacidades de gestión de estado y el enfoque de componentes de React. React Bootstrap proporciona una amplia gama de componentes reutilizables, como botones, formularios, barras de navegación, tarjetas, modales y más, que siguen las mejores prácticas de diseño y accesibilidad de Bootstrap. Además, ofrece integración con otras bibliotecas y herramientas de desarrollo de React, lo que la convierte en una opción popular para el desarrollo de interfaces de usuario en aplicaciones web modernas.

Sin embargo, a medida que el proyecto avanzaba y requería de funcionalidades más complejas, se han apreciado ciertas carencias y funcionamientos anómalos en los componentes de esta biblioteca. Estos problemas están probablemente asociados a que el uso de Bootstrap en React no está muy extendido. Dado que existía Material UI, otra biblioteca enfocada en resolver los mismos problemas que React Bootstrap, se decidió cambiar a la misma, obteniendo mejores resultados.

4.5. Firebase

Firebase es una plataforma de desarrollo de aplicaciones móviles y web desarrollada por Google. Ofrece una amplia gama de servicios en la nube que permiten a los desarrolladores crear aplicaciones de alta calidad de manera más rápida y sencilla. Entre estos servicios, destacan el de bases de datos y autenticación de usuarios. En un principio, se consideró hacer uso de ambos servicios, pero esto perjudicaba, de nuevo, a la descentralización. Es por ello que se descartaron tanto el uso de base de datos, sustituido por el almacenamiento directo en la blockchain, como la autenticación de usuarios. Esta autenticación ha sido sustituida por el uso de MetaMask, tecnología mucho más extendida en el desarrollo de Dapps, y que no rompe con los principios de descentralización.

4.6. Ethers.js

Ethers.js es una biblioteca desarrollada en Javascript, que interactua con la red Ethereum, y principalmente los contratos inteligentes contenidos en ella. Se centra en proporcionar una API limpia y fácil de usar para desarrolladores que desean construir aplicaciones y servicios basados en Ethereum. Sin embargo, como ya expliqué en el Sección 3.3, se ha decidido no usar esta librería de forma totalmente arbitraria.

Capítulo 5

Planificación

5.1. Metodología

Todo proyecto ha de tener unas pautas que marquen el camino a seguir a lo largo del mismo, para así poder llegar al tan ansiado objetivo final que es concluir el proyecto con éxito. Sin embargo, dependiendo del proyecto, las mejores pautas a seguir pueden ser diferentes. Es por ello necesario hacer un pequeño análisis de las características del proyecto, para así poder elegir una metodología adecuada, que de solución a los problemas que puedan surgir.

Como comenté en la Sección 1.3, el objetivo principal de este proyecto es entender cómo funciona la tecnología blockchain, sirviendo el desarrollo de una aplicación descentralizada como guía exploratoria. Por tanto, es fácilmente deducible que no hay unos requisitos bien definidos de qué ha de desarrollarse. Además, el ámbito de la blockchain es relativamente moderno, lo cual hace que no haya unos estándares de cómo desarrollar una Dapp, y se puedan usar multitud de tecnologías diferentes para dar solución a los problemas que se vayan planteando. Por otro lado, la realización de este proyecto no dispone de financiación, por lo que toda reducción de costes, tanto económicos como de tiempo, es bien recibida.

Para dar solución a las características tan flexibles de este proyecto, se ha decidido aplicar una metodología basada en prototipos. Esta metodología plantea una estrategia de desarrollo que se centra en la construcción de versiones incrementales y parciales del producto final a lo largo del ciclo de vida del proyecto. En lugar de seguir un enfoque tradicional de desarrollo en el que se definen completamente el alcance y los requisitos antes de comenzar la implementación, la metodología basada en prototipos permite a los desarrolladores crear prototipos rápidos e iterativos del sistema.

En el contexto del proyecto mencionado, la elección de una metodología basada en prototipos se justifica por la naturaleza exploratoria y flexible del mismo. Dado que no existen requisitos bien definidos y el desarrollo de una aplicación descentralizada en el ámbito de la blockchain es un territorio relativamente nuevo, la metodología basada en prototipos ofrece la flexibilidad necesaria para adaptarse a los cambios y descubrimientos a lo largo del proceso.

Con esta metodología, se pueden construir versiones tempranas del producto que sirven como modelos experimentales. Estos prototipos permiten a los stakeholders, desarrolladores y otros involucrados en el proyecto, obtener una comprensión más clara de cómo se está progresando y cómo se comporta el sistema en sus etapas iniciales. Además, al tratarse de un proyecto sin financiación, la metodología basada en prototipos permite maximizar la eficiencia al enfocarse en construir y validar funcionalidades clave de manera incremental, evitando gastos innecesarios y optimizando el uso del tiempo disponible.

Existen dos enfoques de plantear esta metodología, aplicando prototipos desechables o prototipos evolutivos, cada uno con sus propias características y aplicaciones. Ambos enfoques serán utilizados a lo largo del proyecto, como se detalla en la Sección 5.2.

5.1.1. Prototipos desechables

Los prototipos desechables son versiones básicas de un sistema, que serán descartadas a lo largo del desarrollo. El hecho de que sean de usar y tirar hace que puedan desarrollarse obviando la rigurosidad habitual, como por ejemplo,

evitando documentar, no siguiendo buenas prácticas... Con ello, se consigue una agilidad clave en el desarrollo prematuro, ahorrando tiempo para invertirlo en la exploración de los requisitos. Las principales características de este enfoque de prototipos son:

- Se crean versiones rápidas y simplificadas del producto final que se construyen con la intención de ser descartados después de cumplir su propósito.
- Están diseñados para explorar conceptos, probar ideas y comprender los requisitos del usuario.
- Se desarrollan de manera rápida y con un costo relativamente bajo, centrándose en aspectos clave del diseño o funcionalidades específicas.
- No están destinados a ser parte integral del producto final; su función principal es informar y guiar el proceso de diseño.

Estas características hacen al enfoque de prototipos desechables ideal para el desarrollo de interfaces de usuario así como de sistemas con los que no se tiene experiencia previa. Por tanto, este enfoque es ideal al inicio del proyecto para clarificar requisitos y expectativas. De esta manera, ayuda a desarrolladores y stakeholders a visualizar y comprender las ideas antes de comprometerse con un diseño completo. Además, permite afrontar requisitos ambiguos y hacer una exploración rápida de diferentes soluciones.

5.1.2. Prototipos evolutivos

Los prototipos evolutivos constituyen un enfoque dinámico para el desarrollo de sistemas, caracterizado por su capacidad para mejorar y refinarse gradualmente hasta alcanzar la forma del producto final. Este método se inicia con un prototipo básico, el cual se expande y perfecciona a medida que se recibe retroalimentación y se identifican requisitos adicionales. Las principales características de este enfoque de prototipos son:

- **Mejora gradual:** A diferencia de los prototipos desechables, los prototipos evolutivos están diseñados para crecer y perfeccionarse con el tiempo. Inician como versiones básicas pero se expanden constantemente a medida que se desarrolla una comprensión más profunda de los requisitos del proyecto.
- **Retroalimentación continua:** La interacción con usuarios y stakeholders es esencial en este enfoque. La retroalimentación obtenida durante las fases iniciales alimenta la evolución del prototipo, permitiendo ajustes, incorporación de características adicionales y adaptación a las necesidades emergentes.
- **Comprensión sólida inicial:** Este enfoque es más efectivo cuando hay una comprensión sólida y detallada de los requisitos desde el principio. Aunque se espera que evolucione, se parte de una base sólida que facilita un desarrollo continuo más efectivo.
- **Interacción temprana:** Permite a los usuarios y stakeholders interactuar con versiones tempranas del producto, lo que facilita la identificación de posibles mejoras y la alineación con las expectativas desde las etapas iniciales del desarrollo.

Por tanto,a diferencia de los prototipos desechables, estos se utilizan cuando se tiene una comprensión inicial sólida de los requisitos, pero se espera que evolucionen a lo largo del tiempo. Además, permiten a los usuarios y stakeholders interactuar con versiones tempranas del producto y proporcionar comentarios que guíen el desarrollo continuo.

5.2. Etapas

5.2.1. Fase inicial

Al ser la primera fase, el conocimiento que se posee acerca de la blockchain es nimio. Por tanto, esta fase es de carácter exploratorio. En ella se pretende aclarar qué herramientas van a utilizarse para el desarrollo de la aplicación, así cómo los requisitos más esenciales. Es por ello que el enfoque metodológico que se aplica es el de prototipos desechables, explicado detalladamente en la Subsección 5.1.1.

5.2.2. Fase intermedia

Una vez aclaradas las tecnologías principales sobre las que se desarrollará el proyecto, empieza la fase que consumirá la mayor parte del desarrollo. En ella se pretende seguir una metodología basada en un prototipo evolutivo, modelo explicado en Subsección 5.1.2. Por un lado, en esta fase se desarrollará un análisis previo de la aplicación que se quiere crear. Se detallarán los actores que darán uso de la aplicación, así como los casos de uso que estos deberán poder realizar. Además, se definirán los requisitos que la aplicación ha de cumplir.

A continuación, será necesario plantear un diseño inicial de la arquitectura de la aplicación. Para ello, me serviré de diferentes diagramas, como pueden ser modelos de dominio y modelos de componentes. En ellos se detallará una arquitectura que podrá sufrir cambios a medida que se avance en el desarrollo de la aplicación.

Finalmente, se procederá al desarrollo de la aplicación, haciendo uso de la información recolectada tanto del análisis como del diseño. Esta será la fase que más larga y que más recursos necesite. Además, al usar la metodología de prototipado evolutivo, los requisitos y el diseño irán sufriendo cambios a lo largo del desarrollo.

5.2.3. Fase final

Por último, en esta fase se llevará a cabo un refinamiento del prototipo final, realizando pruebas finales, y arreglando los defectos que se hayan podido encontrar, así como una documentación de sus partes.

Después de que las pruebas sean satisfactorias y se asegure la calidad del software, se procederá a la documentación detallada de todas las partes del sistema. Esto incluirá la documentación técnica, que describe la arquitectura del sistema, los componentes y cualquier aspecto técnico relevante. También se elaborará un manual de usuario completo que proporcionará instrucciones claras sobre cómo utilizar la aplicación de manera efectiva.

Finalmente, se completará la memoria del proyecto, que será un registro detallado de todo el proceso de desarrollo. La memoria incluirá información sobre el análisis, el diseño, la implementación, las pruebas y cualquier decisión importante tomada durante el desarrollo del proyecto. Esta documentación será crucial para futuras referencias y para comprender el trabajo realizado en el proyecto.

Una vez que todos estos pasos estén completos, se considerará que la fase final ha concluido con éxito y la aplicación estará lista para su despliegue. Es importante destacar que incluso después de la entrega, es posible que se requieran actualizaciones y mantenimiento continuo para garantizar el buen funcionamiento a largo plazo de la aplicación.

5.3. Plan de proyecto

Una vez definido qué modelo de desarrollo se va a utilizar, es necesario precisar las tareas que se deberán llevar a cabo en cada una de las fases. En la Figura 5.1 se muestra la lista de tareas e hitos creados para gestionar el desarrollo del proyecto. A continuación se detallan las tareas:

- **Documentación y aprendizaje**: Se centrará en la recolección de información y el aprendizaje necesario para establecer una base sólida para el proyecto. Incluye la revisión de documentación existente, investigación de tecnologías y metodologías relevantes, y la adquisición de conocimientos que permitirán un desarrollo más efectivo y alineado con los objetivos del proyecto.

- **Desarrollo de prototipos desechables**: Se crearán prototipos preliminares, conocidos como prototipos desechables, cuyo propósito principal es explorar ideas y validar conceptos básicos. Estos prototipos no están destinados a ser parte del producto final, sino que sirven para identificar posibles problemas, experimentar con diferentes enfoques de diseño, y obtener retroalimentación temprana que guiará el desarrollo del prototipo principal.

- **Análisis y diseño del prototipo evolutivo**: Durante esta tarea se realizará un análisis detallado y se elaborará el diseño del prototipo que evolucionará hacia la versión final. Este análisis incluye la evaluación de los resultados obtenidos de los prototipos desechables, la identificación de las funcionalidades esenciales y las mejoras necesarias, y la creación de un plan de diseño que guiará el desarrollo del prototipo evolutivo.

- **Desarrollo del segundo prototipo**: Esta tarea implicará la implementación del segundo prototipo basado en el diseño evolutivo previamente elaborado. El objetivo es desarrollar una versión más avanzada y funcional del producto, incorporando las características y mejoras identificadas durante la fase de análisis y diseño. El segundo prototipo debe ser suficientemente robusto para ser evaluado exhaustivamente, permitiendo identificar áreas adicionales de mejora antes de proceder al refinamiento final.

- **Análisis y diseño de mejoras del prototipo**: En esta tarea se realizará un análisis crítico del segundo prototipo, utilizando la retroalimentación obtenida de las pruebas y evaluaciones realizadas. Se identifican las áreas que requieren mejoras y se diseña un plan de acción para implementar estas mejoras. El objetivo es optimizar el prototipo, enfocándose en aspectos como la funcionalidad, la eficiencia, la usabilidad y la estabilidad del sistema.

- **Desarrollo del refinado del prototipo**: Esta tarea consistirá en el desarrollo del prototipo final, incorporando todas las mejoras y ajustes identificados en la etapa anterior. El objetivo es entregar un producto refinado que cumpla con los requisitos establecidos y esté listo para su implementación o presentación final. Se realizarán pruebas exhaustivas para asegurar que todas las funcionalidades operen correctamente y que el producto cumpla con los estándares de calidad esperados.

- **Documentación y elaboración de la memoria**: En esta tarea se lleva a cabo la documentación final del proyecto y la elaboración de la memoria descriptiva. Esto incluye la compilación de toda la información relevante sobre el desarrollo del prototipo, desde la concepción inicial hasta la entrega final. La memoria detallará los objetivos, los métodos utilizados, los resultados obtenidos, las dificultades encontradas y las soluciones implementadas, proporcionando una visión comprensiva del proyecto y sus logros.

Por otro lado, se han establecido tres hitos que ayudarán a respetar los plazos del proyecto. Son los siguientes:

- **Entrega del primer prototipo**:
  - **Fecha de entrega**: 23 de noviembre de 2022
  - **Descripción**: Primer prototipo inicial basado en los requisitos y análisis preliminares.
- **Entrega del segundo prototipo**:
  - **Fecha de entrega**: 26 de febrero de 2023
  - **Descripción**: Segundo prototipo, desarrollado tras el análisis y diseño evolutivo.
- **Entrega del prototipo refinado**:
  - **Fecha de entrega**: 6 de abril de 2023
  - **Descripción**: Prototipo final con mejoras y refinamientos basados en la retroalimentación del segundo prototipo.

Finalmente, en la Figura 5.2, se presenta un diagrama de Gantt del proyecto, que ayuda a visualizar de forma más sencilla, tanto las tareas como las metas definidas en el proyecto.

Entrega del primer prototipo	Milestone	0 días	23/11/2022	23/11/2022
Entrega del segundo prototipo	Milestone	0 días	26/02/2023	26/02/2023
Entrega del prototipo refinado	Milestone	0 días	06/04/2023	06/04/2023
+ Add task or milestone				
Fase inicial				
Documentación y aprendizaje	Task	5 días	19/10/2022	25/10/2022
Desarrollo de prototipos desechables	Task	20 días	26/10/2022	22/11/2022
+ Add task or milestone				
Fase intermedia				
Análisis y diseño del prototipo evolutivo	Task	5 días	29/11/2022	05/12/2022
Desarrollo del segundo prototipo	Task	60 días	05/12/2022	26/02/2023
+ Add task or milestone				
Fase final				
Análisis y diseño de mejoras del prototipo	Task	5 días	26/02/2023	03/03/2023
Desarrollo del refinado del prototipo	Task	25 días	03/03/2023	06/04/2023
Documentación y elaboración de la memoria	Task	20 días	06/04/2023	03/05/2023
+ Add task or milestone				

Figura 5.1: Tareas del Plan de Proyecto

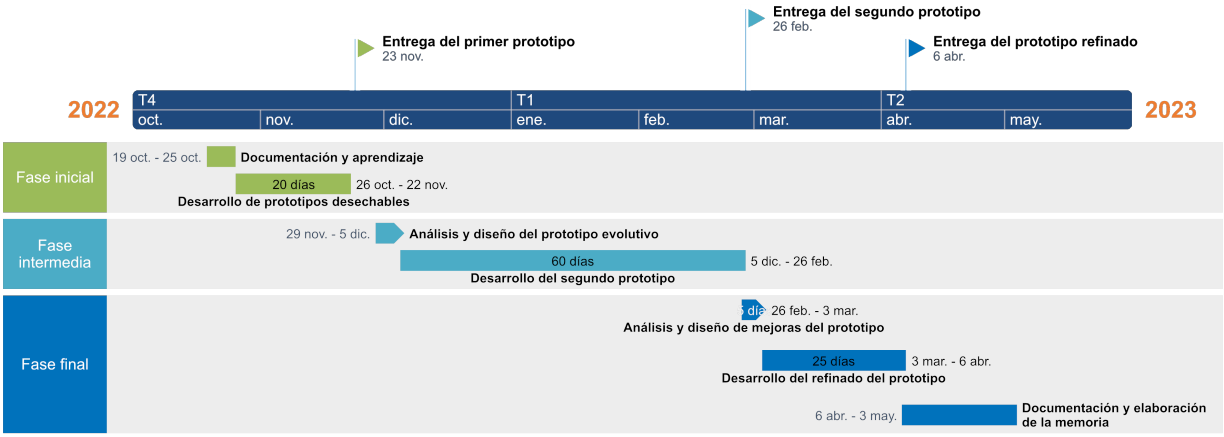


Figura 5.2: TDiagrama de Gantt del Proyecto

Capítulo 6

Análisis

En esta sección, nos adentraremos en el proceso de análisis del proyecto, correspondiente a la fase de prototipado evolutivo. El análisis constituye una fase fundamental en el desarrollo de cualquier sistema o aplicación, ya que nos permite comprender en profundidad el contexto, los actores involucrados y las necesidades que deben ser abordadas. En esta etapa, exploraremos detalladamente los diversos elementos que componen el proyecto, desde los actores principales hasta los requisitos tanto funcionales como no funcionales que guiarán el diseño y la implementación.

El análisis no solo implica identificar los elementos básicos del sistema, sino también comprender las relaciones entre ellos y cómo interactúan para lograr los objetivos planteados. Además, es crucial reconocer los casos de uso más relevantes, aquellos que representan situaciones concretas en las que el sistema será utilizado por los usuarios finales.

A lo largo de este apartado, exploraré cada uno de estos aspectos en detalle, proporcionando una visión clara y completa que servirá como base sólida para las etapas subsiguientes del desarrollo. El objetivo principal es garantizar que las decisiones de diseño y desarrollo estén respaldadas por un profundo entendimiento de las necesidades del usuario y del contexto en el que operará el sistema, todo ello dentro del marco de un prototipado evolutivo que nos permitirá iterar y mejorar continuamente en función de los resultados obtenidos.

6.1. Caso de Negocio

A modo de guía para el desarrollo del prototipo evolutivo, se ha decidido poner el foco en un caso concreto de una cadena de suministro del sector lácteo. Podemos ver una representación en 6.1. Se puede observar que los principales actores involucrados son un ganadero, una veterinario, un transportista y una industria que recibe el producto. Adicionalmente, se contemplan los proveedores de equipamiento y comida del ganadero, un distribuidor que se encarga de conectar a la industria con el consumidor, y un laboratorio que certifica el producto.

El funcionamiento de esta cadena se detalla a continuación:

- El ganadero llega a un acuerdo económico con la industria láctea. Se firma un contrato de duración variable, dependiendo de la incertidumbre del mercado. Puede haber actualizaciones de los precios durante el tiempo que el contrato esté en vigor si los costes de producción se ven incrementados en ese periodo.
- El ganadero se compromete a entregar a la industria láctea un número de litros diarios y a cumplir unos requisitos de calidad y certificación.
- Un transportista contratado por la industria láctea recoge diariamente la leche al ganadero.
- Se recogen muestras de la leche en origen (granja) y en destino (industria) que son analizadas por el laboratorio para verificar que la leche puede ser consumida.
- El informe del laboratorio, tanto en origen como en destino, es entregado tanto al ganadero como a la industria. El transportista verifica el número de litros que recoge en la granja y realiza un albarán de recogida.
- La industria recoge la leche que lleva el transportista y realiza un albarán de recogida.

- Si el informe del laboratorio es favorable, una vez que la leche ha sido recibida por la industria, la factura es emitida por el ganadero para que la empresa transformadora la pague en los términos establecidos en el contrato.
- Se realiza la transferencia monetaria por la industria, tanto al ganadero como al transportista en los plazos establecidos en el contrato.

Por otro lado, la información contenida en los albaranes es la siguiente:

- Lugar y fecha de emisión del albarán
- Código o número del documento
- Datos identificativos del comprador y del vendedor
- Domicilio del comprador y vendedor
- Lugar y fecha de entrega
- Firma y sello del receptor de la mercancía
- Cantidad y descripción de los productos

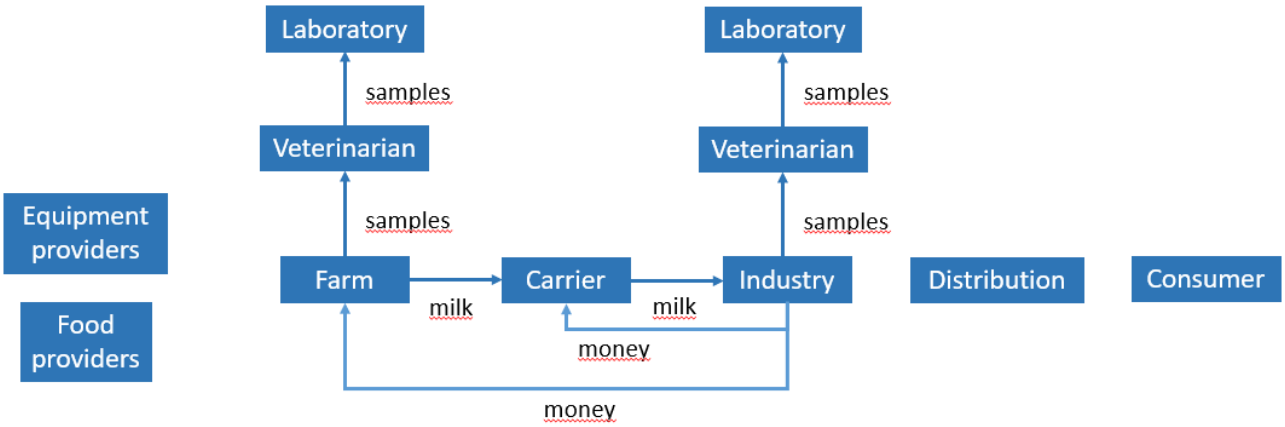


Figura 6.1: Esquema de una cadena de suministro láctea

6.2. Actores

En UML los "actores" de un sistema, son aquellas entidades externas que interactúan con el mismo. Representan roles desempeñados por usuarios, sistemas externos, dispositivos de hardware, o incluso otros sistemas de software. En la aplicación desarrollada, todos los actores involucrados son usuarios con diferentes roles. Se mencionan a continuación:

- **Comprador:** Es aquella persona que tiene el interés de recibir un producto a través de una cadena de suministro. Esta persona, tiene la responsabilidad de crear la cadena de suministro, con los detalles acordados con el resto de las partes, así como de realizar el pago de los productos que recibe.
- **Vendedor:** Es aquella persona que desea vender un producto a través de una cadena de suministro. Una vez el comprador crea la cadena, el vendedor ha de confirmar que está de acuerdo con los detalles, y proveer de productos al comprador durante el tiempo que dure el contrato.
- **Transportista:** Es la persona encargada de conectar al vendedor con el comprador. De esta manera, recogerá los productos que suministra el vendedor, para entregárselos al comprador. A cambio, recibe un pago estipulado en la cadena de suministro.

- **Certificador:** Es la entidad encargada de certificar los productos de una cadena de suministro. En el caso de productos frescos, como puede ser la leche, ha de certificar los productos, tanto en origen, como en destino.
- **Usuario genérico:** Este rol alberga a toda aquella persona que no cumple ninguno de los roles anteriores para con una cadena de suministro. Puesto que la blockchain tiene como principio la transparencia, toda la información referente a una cadena de suministro, es pública a cualquier persona que quiera consultarla. Es por ello que el usuario genérico puede ver toda la información referente a una cadena de suministro, así como los productos y el estado de estos. Sin embargo, un usuario genérico no puede hacer las acciones concretas que pertenecen a los roles anteriores, como podrían ser la creación de un producto, la certificación, etc. Cabe mencionar que todos los roles anteriores heredan de este, lo que significa que todas las acciones que un usuario genérico puede hacer, el resto también pueden.

Estos roles hacen referencia a la relación de una persona para con una cadena de suministro en concreto. Sin embargo, una persona puede ejercer diferentes roles en diferentes cadenas. Un caso común, por ejemplo, es que una persona sea el comprador en una cadena, adquiriendo así un producto, que posteriormente vende en otra cadena de suministro. Por otro lado, es incluso posible ejercer varios roles en una misma cadena, siendo comprador y vendedor. De esta manera, una persona puede reflejar la transformación que realiza sobre un producto, quedando registros en la aplicación. Esto permite tener una trazabilidad completa de los productos, desde su fase más temprana, hasta la entrega a un consumidor final.

6.3. Requisitos

Los requisitos son una parte fundamental en el proceso de análisis y diseño de sistemas. Establecen las funciones y características que el sistema debe cumplir para satisfacer las necesidades de los usuarios y cumplir con los objetivos del negocio. A continuación, se detallarán tanto los requisitos funcionales como los no funcionales del sistema en cuestión.

6.3.1. Requisitos funcionales

Los requisitos funcionales son las especificaciones detalladas que describen las funciones específicas que el sistema debe realizar. Estos requisitos definen las acciones que el sistema debe ser capaz de llevar a cabo y cómo debe comportarse en respuesta a diversas entradas. En esta sección, se enumerarán los requisitos funcionales identificados para el sistema en cuestión.

- RF-1: El sistema deberá permitir conectar cuentas de MetaMask, con la aplicación.
- RF-2: El sistema deberá permitir registrar una cuenta de MetaMask como usuario en la aplicación.
- RF-3 El sistema deberá permitir consultar una lista de los usuarios registrados en la aplicación.
- RF-4: El sistema deberá permitir consultar la información de los usuarios registrados en la aplicación.
- RF-5: El sistema deberá permitir consultar una lista con las cadenas de suministro creadas en la aplicación.
- RF-6: El sistema deberá permitir crear una cadena de suministro a cualquier usuario que tenga una cuenta de MetaMask conectada a la aplicación.
- RF-7: El sistema deberá permitir consultar la información referente a una cadena de suministro.
- RF-8: El sistema deberá permitir a Certificador, Vendedor y Transportista, firmar una cadena de suministro aún no firmada.
- RF-9: El sistema deberá permitir al Comprador, añadir fondos a una cadena de suministro ya firmada.
- RF-10: El sistema deberá permitir al Comprador, Vendedor y Transportista, retirar fondos de una cadena de suministros ya firmada.

- RF-11: El sistema deberá permitir consultar la lista con los productos pertenecientes a una cadena de suministros.
- RF-12: El sistema deberá permitir consultar la información de un producto seleccionado.
- RF-13: El sistema deberá permitir crear productos al Vendedor, en una cadena de suministros ya firmada.
- RF-14: El sistema deberá permitir enviar productos al Transportista, en una cadena de suministros ya firmada.
- RF-15: El sistema deberá permitir certificar productos en origen al Certificador, en una cadena de suministros ya firmada.
- RF-16: El sistema deberá permitir certificar productos en destino al Certificador, en una cadena de suministros ya firmada.
- RF-17: El sistema deberá permitir recibir productos al Comprador, en una cadena de suministros ya firmada.
- RF-18: El sistema deberá permitir tanto al Comprador como al Vendedor de una cadena de suministro, crear una solicitud de modificación del precio de la misma.
- RF-19: El sistema deberá permitir tanto al Comprador como al Vendedor de una cadena de suministro, aceptar una solicitud de modificación del precio de la misma.

6.3.2. Requisitos no funcionales

Los requisitos no funcionales complementan los requisitos funcionales al definir los criterios de calidad, restricciones y características que no están directamente relacionadas con las funciones específicas del sistema, pero que son igualmente importantes para su éxito y eficacia. Estos requisitos abordan aspectos como el rendimiento, la seguridad, la usabilidad y otros atributos del sistema que afectan su calidad global y su capacidad para cumplir con las expectativas de los usuarios y del negocio. En esta sección, se enumeran los requisitos no funcionales identificados, proporcionando una comprensión clara de los estándares y condiciones que el sistema debe cumplir además de sus funciones básicas.

- RNF-1: El sistema deberá implementar un *frontend* desarrollado con React.
- RNF-2: El sistema deberá implementar integrar el uso de MetaMask.
- RNF-3: El sistema deberá implementar un *backend* con la tecnología Go Ethereum.
- RNF-4: El sistema deberá conectar el *backend* y el *frontend* usando la librería Web3.js.
- RNF-5: El sistema deberá implementar la lógica de negocio del *backend* usando smart contracts desarrollados en el lenguaje Solidity.
- RNF-6: El sistema deberá usar Truffle para desplegar smart contracts en la red blockchain.
- RNF-7: El sistema deberá sustituir las direcciones de cuentas por el nombre de usuario, si este está registrado en la aplicación.
- RNF-8: El sistema deberá implementar un menú lateral.
- RNF-9: El sistema deberá implementar un código QR que al leerlo dirija a la vista del producto al que está asociado.

Requisitos de información

RNFI-1: El sistema deberá almacenar la siguiente información de los usuarios registrados:

- Nombre
- Dirección de la cuenta en la blockchain
- Dirección Física
- Código Postal
- Número de teléfono
- Correo electrónico

RNFI-2: El sistema deberá almacenar la siguiente información de las cadenas de suministro:

- Nombre del producto
- Dirección de la cadena de suministro en la blockchain.
- Dirección de la cuenta en la blockchain del comprador
- Dirección de la cuenta en la blockchain del vendedor
- Dirección de la cuenta en la blockchain del transportista
- Dirección de la cuenta en la blockchain del certificador
- Precio unitario del producto
- Precio del transportista
- Fecha de fin de contrato
- Cadenas de suministro que la preceden
- Productos asociados a la cadena de suministro
- Fondos monetarios del comprador,vendedor y transportista asociados a la cadena de suministro
- Firma del vendedor
- Firma del transportista
- Firma del certificador

RNFI-3: El sistema deberá almacenar la siguiente información de los productos:

- Cantidad
- Estado
- Certificado en origen
- Certificado en destino
- Última modificación
- Lista de Steps asociados al producto.

RNFI-4: El sistema deberá almacenar la siguiente información de los Steps:

- Tipo
- Fecha de creación
- Creador
- Producto asociado

6.4. Casos de uso

A continuación se listan los casos de uso que debe implementar la aplicación:

- CU-1 Registrarse como usuario
- CU-2 Consultar los datos de un usuario
- CU-3 Crear cadena de suministro
- CU-4 Consultar lista de cadena de suministro
- CU-5 Consultar cadena de suministro
- CU-6 Consultar lista de productos de una cadena de suministro
- CU-7 Consultar producto de una cadena de suministro
- CU-8 Consultar lista de usuarios
- CU-9 Solicitar modificar precio
- CU-10 Aceptar modificación de precio
- CU-11 Consultar firmas de cadena de suministro
- CU-12 Firmar cadena de suministro
- CU-13 Añadir fondos a una cadena de suministro
- CU-14 Retirar fondos de una cadena de suministro
- CU-15 Crear producto
- CU-16 Editar cadenas de suministro precedentes
- CU-17 Realizar envío de producto
- CU-18 Certificar producto en origen
- CU-19 Certificar producto en destino
- CU-20 Recibir producto

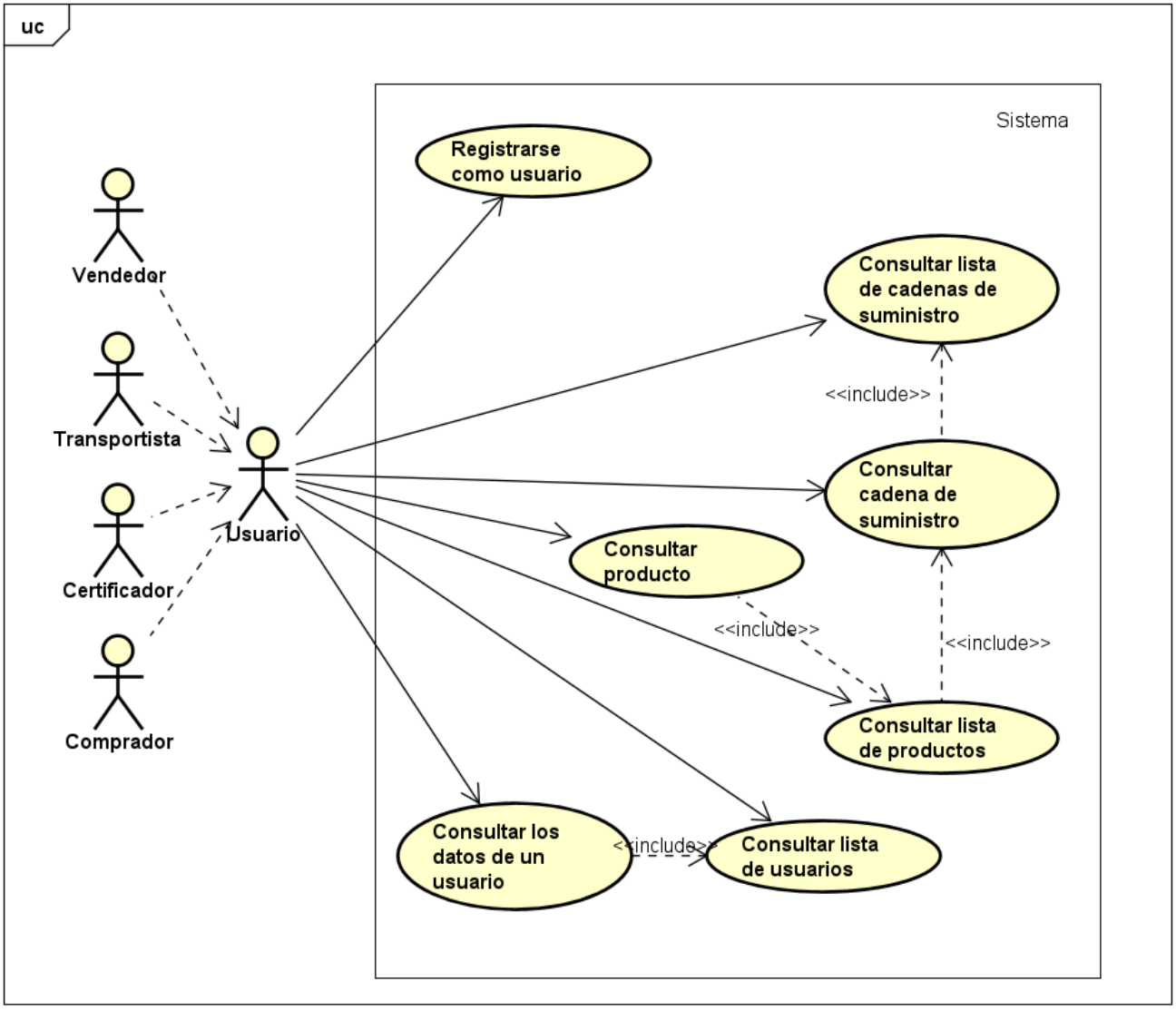


Figura 6.2: Casos de uso genéricos

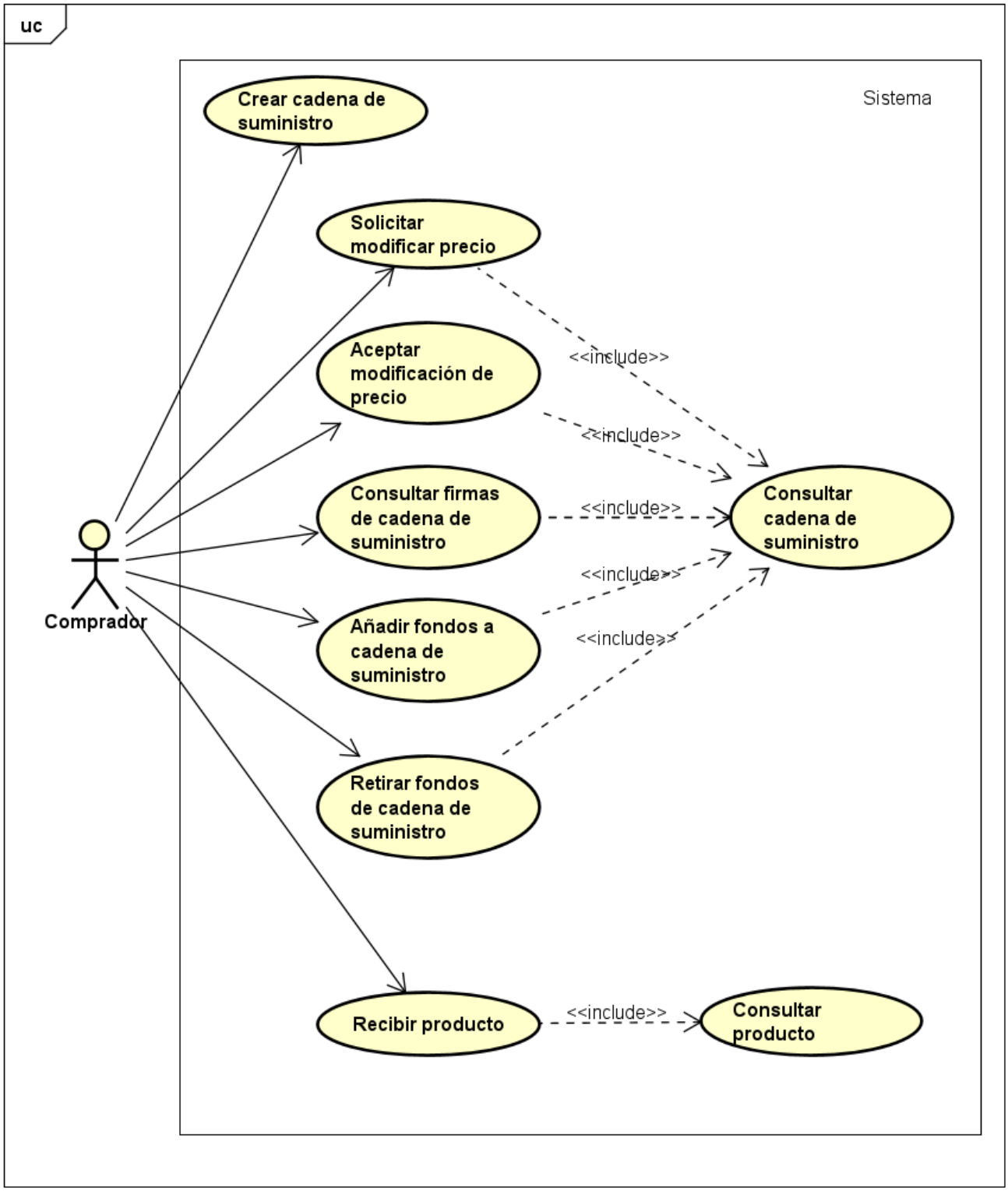


Figura 6.3: Casos de uso del comprador

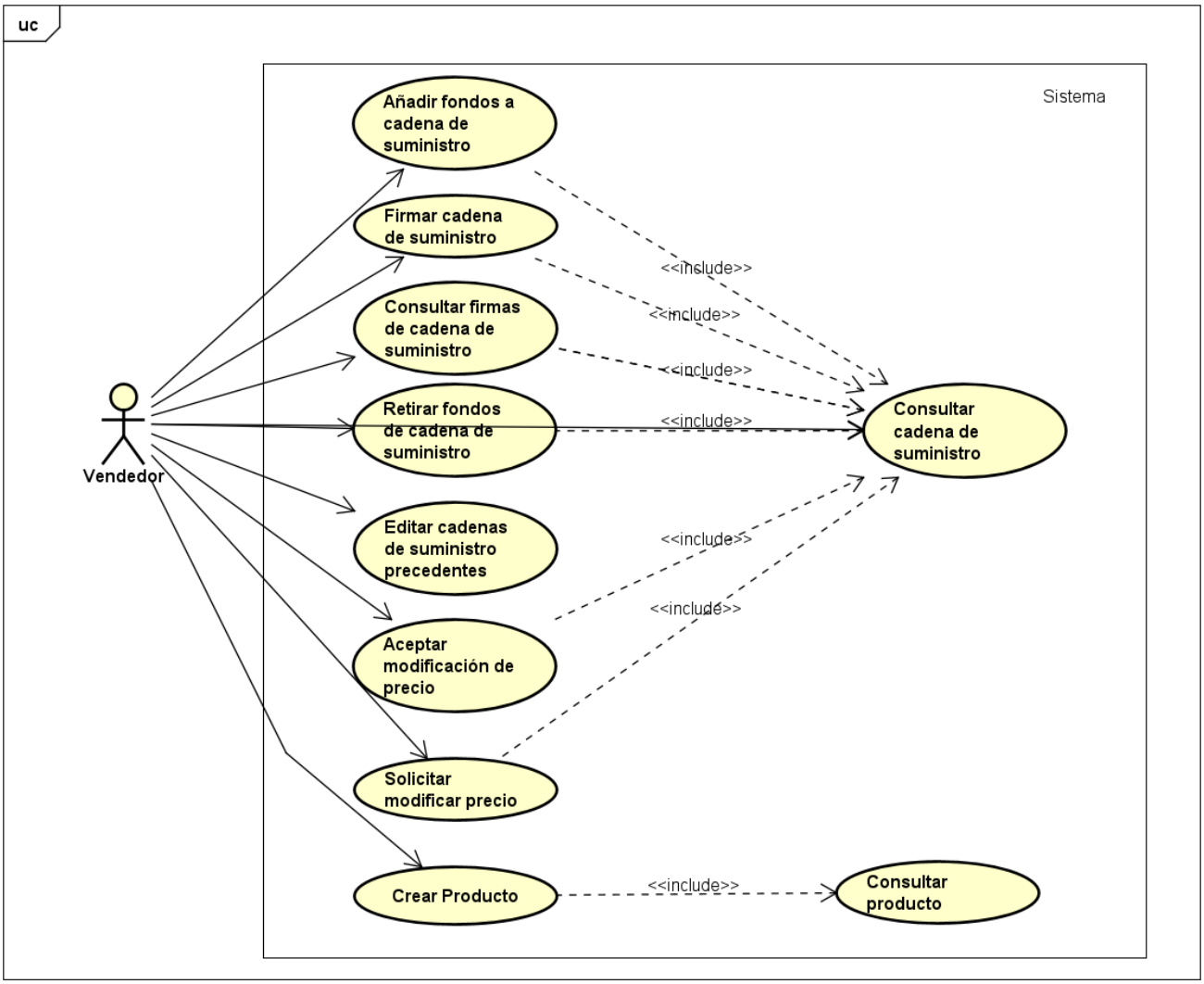


Figura 6.4: Casos de uso del vendedor

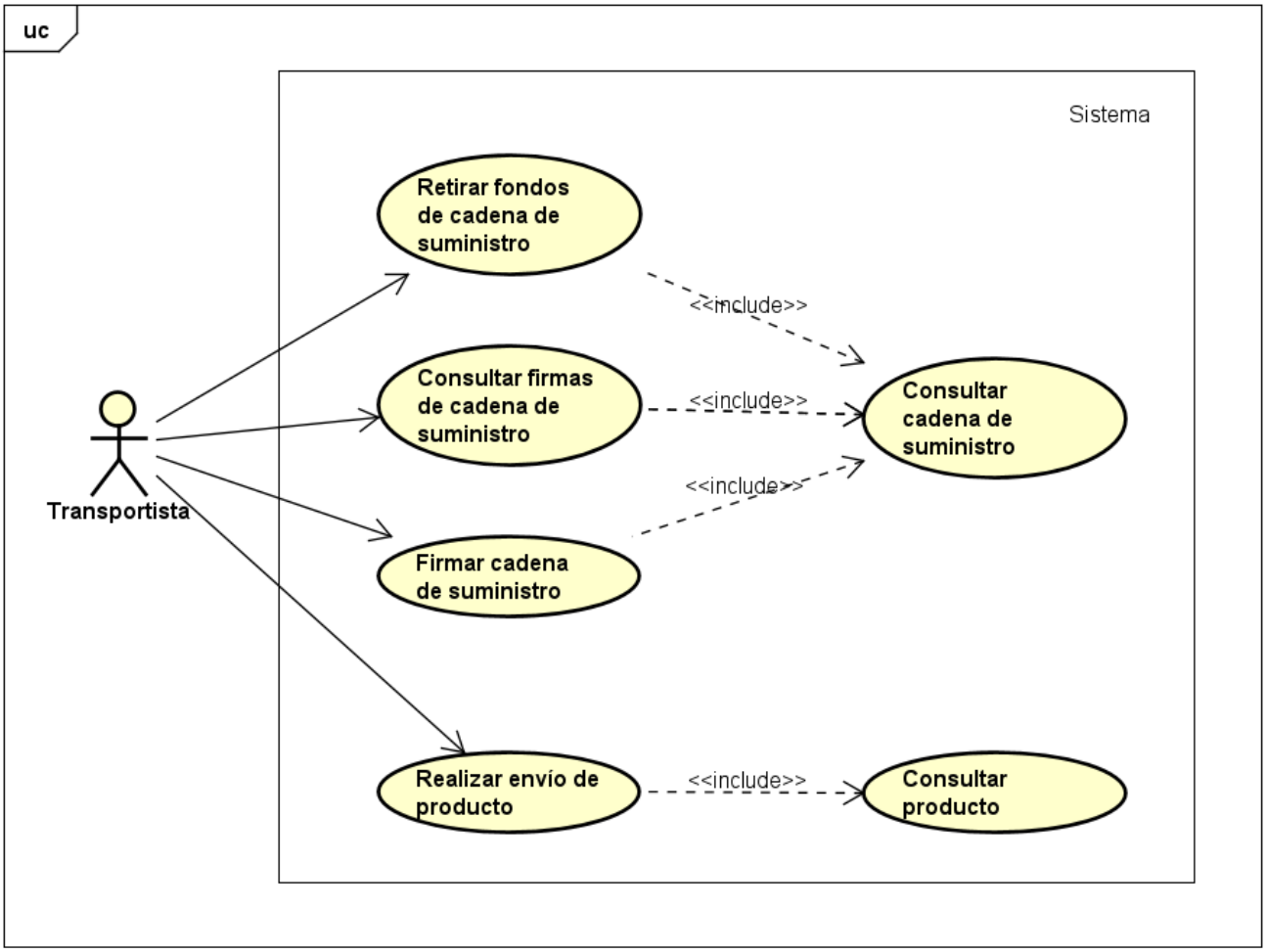


Figura 6.5: Casos de uso del transportista

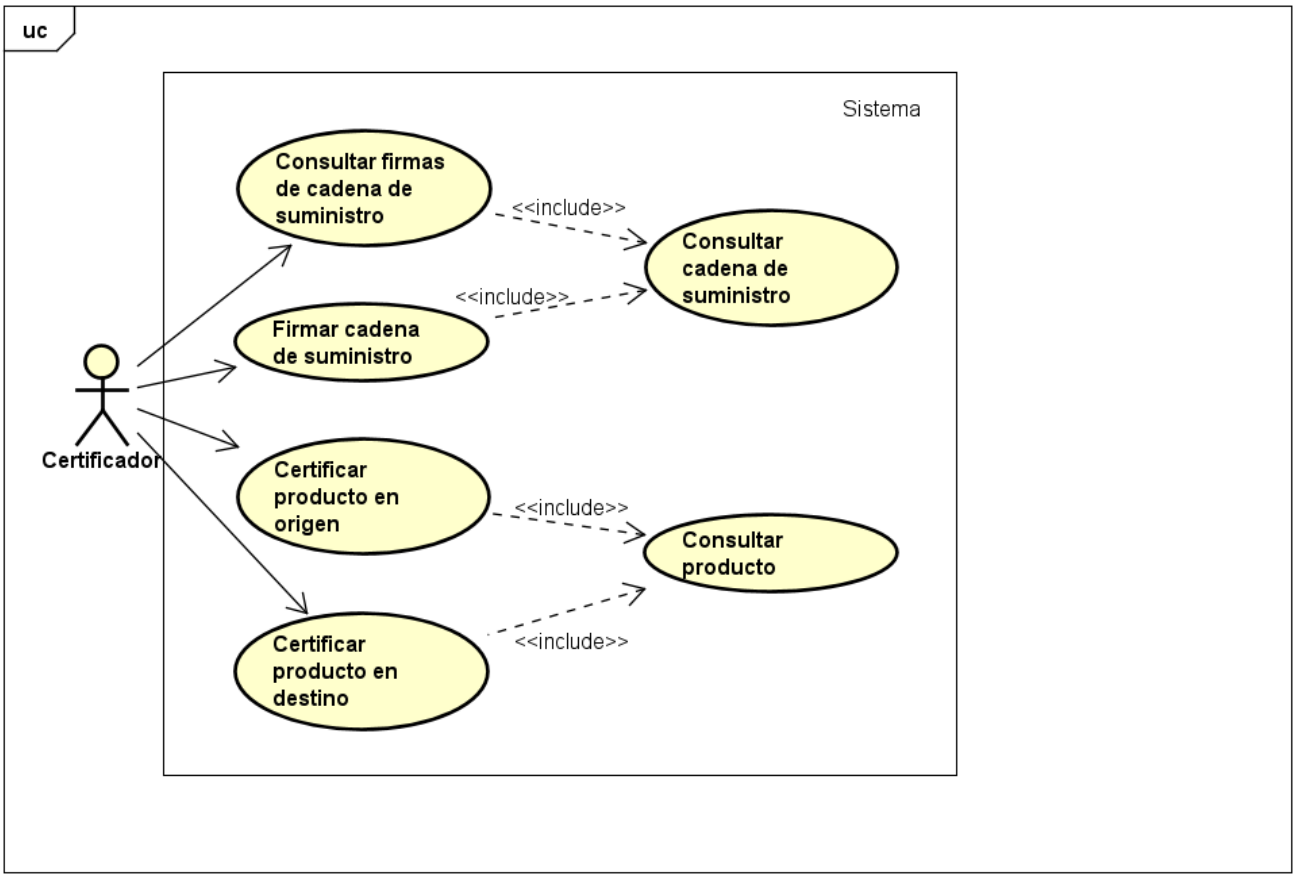


Figura 6.6: Casos de uso del certificador

CU-1	**Registrarse como usuario**
Actor	Usuario
Descripción	El sistema deberá permitir a un usuario registrarse en la aplicación
Precondiciones	El usuario ha de tener una cuenta en MetaMask que esté conectada a la aplicación. El usuario ha accedido al formulario de registro.
Secuencia normal	
Paso	**Acción**
1	El usuario introduce los campos Nombre , Dirección , Código Postal , Número de teléfono y Correo electrónico .
2	El usuario pulsa el botón Registrarse
3	El sistema valida los datos y muestra una transacción en MetaMask con los datos rellenados.
4	El usuario confirma la transacción en MetaMask
Postcondiciones	La transacción se efectúa con éxito y el usuario queda registrado.
Excepciones	
3b	El sistema muestra los campos erróneos. Se continúa en el punto 1.
4b	El usuario decide cancelar la transacción.

Cuadro 6.1: CU-1 Registrarse como usuario

CU-2	**Consultar los datos de un usuario**
Actor	Usuario
Descripción	El sistema deberá permitir al usuario consultar los datos referentes a un usuario registrado.
Precondiciones	El usuario ha realizado el CU-8
Secuencia normal	
Paso	**Acción**
1	El usuario pulsa el botón Ver Perfil
2	El sistema muestra los datos referentes al usuario seleccionado.
Postcondiciones	
Excepciones	
Paso	**Acción**

Cuadro 6.2: CU-2 Consultar los datos de un usuario

CU-3	**Crear cadena de sumistro**
Actor	Usuario
Descripción	El sistema deberá permitir al usuario crear una cadena de suministro.
Precondiciones	El usuario ha de tener una cuenta de MetaMask conectada.
Secuencia normal	
Paso	**Acción**
1	El usuario pulsa el botón Crear cadena
2	El sistema muestra el formulario de creación de cadena de suministro.
3	El usuario rellena los campos Nombre del producto , Vendedor , Transportista , Certificador , Precio , Precio del transportista y Fecha fin de contrato .
4	El sistema valida los datos y crea una transacción en MetaMask.
5	El usuario confirma la transacción.
6	El sistema crea la cadena de suministros con los datos introducidos.
Postcondiciones	
Excepciones	
Paso	**Acción**
4b	El sistema muestra los campos erróneos. Se continúa en el paso 3.
5b	El usuario decide cancelar la transacción

Cuadro 6.3: CU-3 Crear cadena de sumistro

CU-4	**Consultar lista de cadenas de suministro**
Actor	Usuario
Descripción	El sistema deberá permitir al usuario consultar una lista de todas las cadenas de suministro creadas.
Precondiciones	
Secuencia normal	
Paso	**Acción**
1	El usuario pulsa el botón Lista de cadenas
2	El sistema muestra una lista con las cadenas de suministro creadas.
Postcondiciones	
Excepciones	
Paso	**Acción**

Cuadro 6.4: CU-4 Consultar lista de cadenas de suministro

CU-5	**Consultar cadena de suministro**
Actor	Usuario
Descripción	El sistema deberá permitir al usuario consultar la información relevante a la cadena de suministro seleccionada.
Precondiciones	El usuario ha realizado el CU-4
Secuencia normal	
Paso	**Acción**
1	El usuario pulsa en la dirección de la cadena de suministro que quiere ver.
2	El sistema muestra una lista con toda la información de la cadenas de suministro seleccionada.
Postcondiciones	
Excepciones	
Paso	**Acción**
2b	Si el Actor es el Vendedor de la cadena de suministro, el sistema habilita las opciones Firmar Cadena , Añadir/Retirar Fondos, Crear Producto, Editar precedentes , Modificar Precio y Lista de productos en el menú lateral.
2c	Si el Actor es el Transportista o el Certificador de la cadena de suministro, el sistema habilita las opciones Firmar Cadena , Añadir/Retirar Fondos y Lista de productos en el menú lateral.
2d	Si el Actor es el Comprador de la cadena de suministro, el sistema habilita las opciones Firmar Cadena , Añadir/Retirar Fondos , Modificar Precio y Lista de productos en el menú lateral.

Cuadro 6.5: CU-5 Consultar cadena de suministro

CU-6	**Consultar lista de productos de una cadena de suministro**
Actor	Usuario
Descripción	El sistema deberá permitir al usuario consultar una lista con todos los productos creados en una cadena de suministro.
Precondiciones	El usuario ha realizado el CU-5
Secuencia normal	
Paso	**Acción**
1	El usuario pulsa en el botón Lista de productos .
2	El sistema muestra una lista con todos los productos creados de la cadena de suministro.
Postcondiciones	
Excepciones	
Paso	**Acción**

Cuadro 6.6: CU-6 Consultar lista de productos de una cadena de suministro

CU-7	**Consultar producto de una cadena de suministro**
Actor	Usuario
Descripción	El sistema deberá permitir al usuario toda la información de referente a un producto.
Precondiciones	El usuario ha realizado el CU-6
Secuencia normal	
Paso	**Acción**
1	El usuario pulsa en la dirección del producto que quiere consultar.
2	El sistema muestra toda la información del producto seleccionado así como los pasos del mismo.
3	El sistema muestra un código QR que enlaza a la vista de información del producto.
Postcondiciones	
Excepciones	
Paso	**Acción**

Cuadro 6.7: CU-7 Consultar producto de una cadena de suministro

CU-8	**Consultar lista de usuarios**
Actor	Usuario
Descripción	El sistema deberá permitir al usuario consultar una lista con todos los usuarios registrados.
Precondiciones	
Secuencia normal	
Paso	**Acción**
1	El usuario pulsa en el botón Lista de Usuarios .
2	El sistema muestra una lista con todos los usuarios registrados.
Postcondiciones	
Excepciones	
Paso	**Acción**

Cuadro 6.8: CU-8 Consultar lista de usuarios

CU-9	**Solicitar modificar precio**
Actor	Comprador,Vendedor
Descripción	El sistema deberá permitir tanto al comprador como al vendedor de una cadena de suministros, crear una solicitud de cambio de precio del producto.
Precondiciones	El Actor ha realizado el CU-5
Secuencia normal	
Paso	**Acción**
1	El Actor pulsa en el botón lateral Modificar precio .
2	El sistema muestra un formulario con el precio actual y un campo para actualizarlo.
3	El Actor introduce el nuevo precio que quiere para el producto y pulsa el botón Modificar precio .
4	El sistema crea una transacción en MetaMask con la información introducida.
5	El actor confirma la transacción desde MetaMask.
6	El sistema crea la solicitud de cambio de precio y lo refleja en pantalla.
Postcondiciones	
Excepciones	
Paso	**Acción**
5b	El Actor decide cancelar la transacción. Se continúa en el Paso 3.

Cuadro 6.9: CU-9 Solicitar modificar precio

CU-10	**Aceptar modificación de precio**
Actor	Comprador,Vendedor
Descripción	El sistema deberá permitir tanto al comprador como al vendedor de una cadena de suministros, aceptar una solicitud de cambio de precio creada por el otro actor.
Precondiciones	Si el Actor es el Comprador: El Vendedor ha realizado el CU-9. Si el Actor es el Vendedor: El Comprador ha realizado el CU-9.
Secuencia normal	
Paso	**Acción**
1	El Actor pulsa en el botón lateral Modificar precio .
2	El sistema muestra un formulario con el precio actual, el nuevo precio solicitado y un botón de Aceptar modificación .
3	El Actor pulsa el botón Aceptar modificación .
4	El sistema crea una transacción en MetaMask con la información introducida.
5	El actor confirma la transacción desde MetaMask.
6	El sistema crea la solicitud de cambio de precio y lo refleja en pantalla.
Postcondiciones	
Excepciones	
Paso	**Acción**
5b	El Actor decide cancelar la transacción. Se continúa en el Paso 3.

Cuadro 6.10: CU-10 Aceptar modificación de precio

CU-11	**Consultar firmas de cadena de suministro**
Actor	Comprador,Vendedor,Certificador,Transportista
Descripción	El sistema deberá permitir al comprador, vendedor, certificador y transportista de una cadena de suministros consultar quiénes la han firmado.
Precondiciones	El Actor ha realizado el CU-5 La cadena de suministro aún no ha sido firmada por todos los miembros.
Secuencia normal	
Paso	**Acción**
1	El Actor pulsa en el botón lateral Firmar cadena .
2	El sistema muestra una tabla que informa si el vendedor, el transportista y el certificador han firmado.
Postcondiciones	
Excepciones	
Paso	**Acción**

Cuadro 6.11: CU-11 Consultar firmas de cadena de suministro

CU-12	**Firmar cadena de suministro**
Actor	Vendedor,Certificador,Transportista
Descripción	El sistema deberá permitir al vendedor, certificador y transportista de una cadena de suministros firmarla.
Precondiciones	El Actor ha realizado el CU-5 La cadena de suministro aún no ha sido firmada por el Actor.
Secuencia normal	
Paso	**Acción**
1	El Actor pulsa en el botón lateral Firmar cadena .
2	El sistema muestra una tabla que informa si el vendedor, el transportista y el certificador han firmado y el botón Firmar .
3	El Actor pulsa el botón Firmar .
4	El sistema crea una transacción en MetaMask.
5	El actor confirma la transacción en MetaMask.
6	El sistema envía la transacción y firma la cadena.
Postcondiciones	
Excepciones	
Paso	**Acción**
5b	El Actor cancela la transacción. Se continúa en el Paso 2.

Cuadro 6.12: CU-12 Firmar cadena de suministro

CU-13	**Añadir fondos a una cadena de suministro**
Actor	Comprador
Descripción	El sistema deberá permitir al comprador transferir fondos de su cuenta personal a una cadena de suministro.
Precondiciones	La cadena de suministros ha sido firmada. El Comprador ha de poseer saldo en su cuenta personal. El Comprador ha realizado el CU-5
Secuencia normal	
Paso	**Acción**
1	El Comprador pulsa en el botón lateral Añadir/Retirar Fondos .
2	El sistema muestra una tabla que informa al comprador de los fondos actuales y permite introducir una cantidad numérica.
3	El Comprador introduce la cantidad que desea transferir y pulsa el botón Añadir Fondos .
4	El sistema crea una transacción en MetaMask con la información.
5	El Comprador confirma la transacción.
6	El sistema envía la transacción y actualiza los fondos que contiene la cadena de suministro.
Postcondiciones	
Excepciones	
Paso	**Acción**
5b	El Comprador cancela la transacción. Se continúa en el Paso 4.

Cuadro 6.13: CU-13 Añadir fondos a una cadena de suministro

CU-14	**Retirar fondos de una cadena de suministro**
Actor	Comprador,Vendedor,Transportista
Descripción	El sistema deberá permitir al comprador, vendedor y transportista retirar los fondos de la cadena de suministro a su cuenta personal.
Precondiciones	La cadena de suministros ha sido firmada. Los fondos de la cadena de suministro, asociados al Actor son mayores que 0. El Actor ha realizado el CU-5
Secuencia normal	
Paso	**Acción**
1	El Actor pulsa en el botón lateral Añadir/Retirar Fondos .
2	El sistema muestra una tabla que informa al comprador de los fondos actuales y un botón de Retirar Fondos .
3	El Actor pulsa el botón Retirar Fondos .
4	El sistema crea una transacción en MetaMask con la información.
5	El Actor confirma la transacción.
6	El sistema envía la transacción y actualiza los fondos que contiene la cadena de suministro.
Postcondiciones	
Excepciones	
Paso	**Acción**
5b	El Comprador cancela la transacción. Se continúa en el Paso 4.

Cuadro 6.14: CU-14 Retirar fondos de una cadena de suministro

CU-15	**Crear producto**
Actor	Vendedor
Descripción	El sistema deberá permitir al vendedor de una cadena de suministros crear un producto con una cantidad asociada.
Precondiciones	La cadena de suministros ha sido firmada. El Vendedor ha realizado el CU-5
Secuencia normal	
Paso	**Acción**
1	El Vendedor pulsa en el botón lateral Crear Producto .
2	El sistema solicita una cantidad que añadir al producto.
3	El Vendedor introduce la cantidad de producto que desea crear y pulsa el botón Crear Producto .
4	El sistema crea una transacción en MetaMask con la información.
5	El Vendedor confirma la transacción.
6	El sistema envía la transacción, crea el producto y el Step Creation asociado al mismo.
Postcondiciones	
Excepciones	
Paso	**Acción**
5b	El Comprador cancela la transacción. Se continúa en el Paso 3.

Cuadro 6.15: CU-15 Crear producto

CU-16	**Editar cadenas de suministro precedentes**
Actor	Vendedor
Descripción	El sistema deberá permitir al vendedor de una cadena de suministros modificar las cadenas de suministro precedentes.
Precondiciones	El Vendedor ha realizado el CU-5 El Vendedor pertenece a otras cadenas de suministro donde ejerce el rol de Comprador
Secuencia normal	
Paso	**Acción**
1	El Vendedor pulsa en el botón lateral Editar precedentes .
2	El sistema muestra una tabla con las posibles cadenas de suministros precedentes.
3	El Vendedor selecciona la o las cadenas que quiere sean precedentes y pulsa el botón Modificar .
4	El sistema crea una transacción en MetaMask con la información.
5	El Vendedor confirma la transacción.
6	El sistema envía la transacción, crea el producto y el Step Creation asociado al mismo.
Postcondiciones	
Excepciones	
Paso	**Acción**
5b	El Vendedor cancela la transacción. Se continúa en el Paso 3.

Cuadro 6.16: CU-16 Editar cadenas de suministro precedentes

CU-17	**Realizar envío de producto**
Actor	Transportista
Descripción	El sistema deberá permitir al transportista de una cadena de suministros realizar el envío de un producto.
Precondiciones	El Transportista ha realizado el CU-7
Secuencia normal	
Paso	**Acción**
1	El Transportista pulsa en el botón Realizar envío .
2	El sistema crea una transacción en MetaMask con la información.
3	El Transportista confirma la transacción.
4	El sistema envía la transacción y crea el Step de envío.
Postcondiciones	
Excepciones	
Paso	**Acción**
3b	El Transportista cancela la transacción. Se continúa en el Paso 3.

Cuadro 6.17: CU-17 Realizar envío de producto

CU-18	**Certificar producto en origen**
Actor	Certificador
Descripción	El sistema deberá permitir al Certificador de una cadena de suministros certificar un producto en origen.
Precondiciones	El Certificador ha realizado el CU-7
Secuencia normal	
Paso	**Acción**
1	El Certificador pulsa en el botón Certificar en Origen .
2	El sistema crea una transacción en MetaMask con la información.
3	El Certificador confirma la transacción.
4	El sistema envía la transacción y crea el Step de Certificación 1.
Postcondiciones	
Excepciones	
Paso	**Acción**
3b	El Certificador cancela la transacción. Se continúa en el Paso 1.

Cuadro 6.18: CU-18 Certificar producto en origen

CU-19	**Certificar producto en destino**
Actor	Certificador
Descripción	El sistema deberá permitir al Certificador de una cadena de suministros certificar un producto en destino.
Precondiciones	El Certificador ha realizado el CU-7
Secuencia normal	
Paso	**Acción**
1	El Certificador pulsa en el botón Certificar en Destino .
2	El sistema crea una transacción en MetaMask con la información.
3	El Certificador confirma la transacción.
4	El sistema envía la transacción y crea el Step de Certificación 2.
Postcondiciones	
Excepciones	
Paso	**Acción**
3b	El Certificador cancela la transacción. Se continúa en el Paso 1.

Cuadro 6.19: CU-19 Certificar producto en destino

CU-20	Recibir producto
Actor	Comprador
Descripción	El sistema deberá permitir al Comprador de una cadena de suministros recibir un producto enviado.
Precondiciones	El Comprador ha realizado el CU-7
Secuencia normal	
Paso	**Acción**
1	El Comprador pulsa en el botón Recibir envío .
2	El sistema crea una transacción en MetaMask con la información.
3	El Comprador confirma la transacción.
4	El sistema envía la transacción y crea el Step de Delivery.
5	El sistema realiza los pagos al transportista y al vendedor.
Postcondiciones	
Excepciones	
Paso	**Acción**
3b	El Certificador cancela la transacción. Se continúa en el Paso 1.

Cuadro 6.20: CU-20 Recibir producto

6.5. Estimación de costes

La estimación de costes para el desarrollo del proyecto se divide en tres categorías principales: software, costos físicos mensuales y costos laborales.

En Tabla 6.21, se enumeran las herramientas de software utilizadas, junto con sus respectivas licencias y precios. Se ha optado por herramientas mayoritariamente de código abierto (Open Source), lo que ha permitido que el costo total en esta categoría sea de 0 €.

Software	Licencia	Precio
Geth	Open Source	0 €
Web3.js	Open Source	0 €
ReactJs	Open Source	0 €
Truffle	Open Source	0 €
Solidity	Open Source	0 €
Remix	Open Source	0 €
Material UI	Open Source	0 €
VisualStudio Code	Open Source	0 €
MetaMask	Open Source	0 €
Astah	Propietaria, posible versión educativa	0 €

Cuadro 6.21: Softwares con su Licencia y Precio

En Tabla 6.22, se detallan los costos físicos asociados al desarrollo del proyecto. Estos incluyen la amortización de hardware, costos de internet, electricidad y tarjeta gráfica. El costo mensual total asciende a 52,49 €, y para una duración del proyecto de 5 meses, el costo total es de 261,96 €.

Concepto	Costo
Ordenador de sobremesa (amortización mensual)	20 €
Internet (mensual)	17,99 €
Electricidad (mensual)	12,50 €
Tarjeta gráfica (amortización mensual)	15 €
Total (5 meses)	261,96 €

Cuadro 6.22: Costos físicos mensuales

En Tabla 6.23, se detallan los costos asociados al trabajo del personal involucrado en el proyecto. Se consideran dos roles principales: programador y desarrollador Blockchain. Según el BOE [9], el sueldo medio de un programador

en España es de 22.892 euros anuales, realizando un 1792 horas. Por tanto, el costo por hora es de 12.77 euros. Se supone una jornada parcial de 20h semanales, donde durante tres meses y medio se ha asumido el rol de programador, y un mes y medio el rol de desarrollador blockchain. Por tanto, el costo total para los sueldos, considerando las horas trabajadas y el salario por hora, asciende a 5919,20 €.

Concepto	**Sueldo por hora**	**Horas**	**Coste total**
Programador	12,77 €	280	3575,60 €
Desarrollador Blockchain	19,53 €	120	2343,60 €
Total			**5919,20 €**

Cuadro 6.23: Costes laborales

Sumando los costos de software, costos físicos y costos laborales, obtenemos el costo total estimado para el proyecto:

- Costos de Software: 0 €
- Costos Físicos: 261,96 €
- Costos Laborales: 5919,20 €

Por tanto, el costo total del Proyecto asciende a 6181,16 €. Esta estimación proporciona una visión clara de los recursos financieros necesarios para el desarrollo del proyecto, destacando la eficiencia en el uso de herramientas de código abierto y una gestión detallada de los costos físicos y laborales.

Capítulo 7

Diseño

7.1. Arquitectura de una Dapp

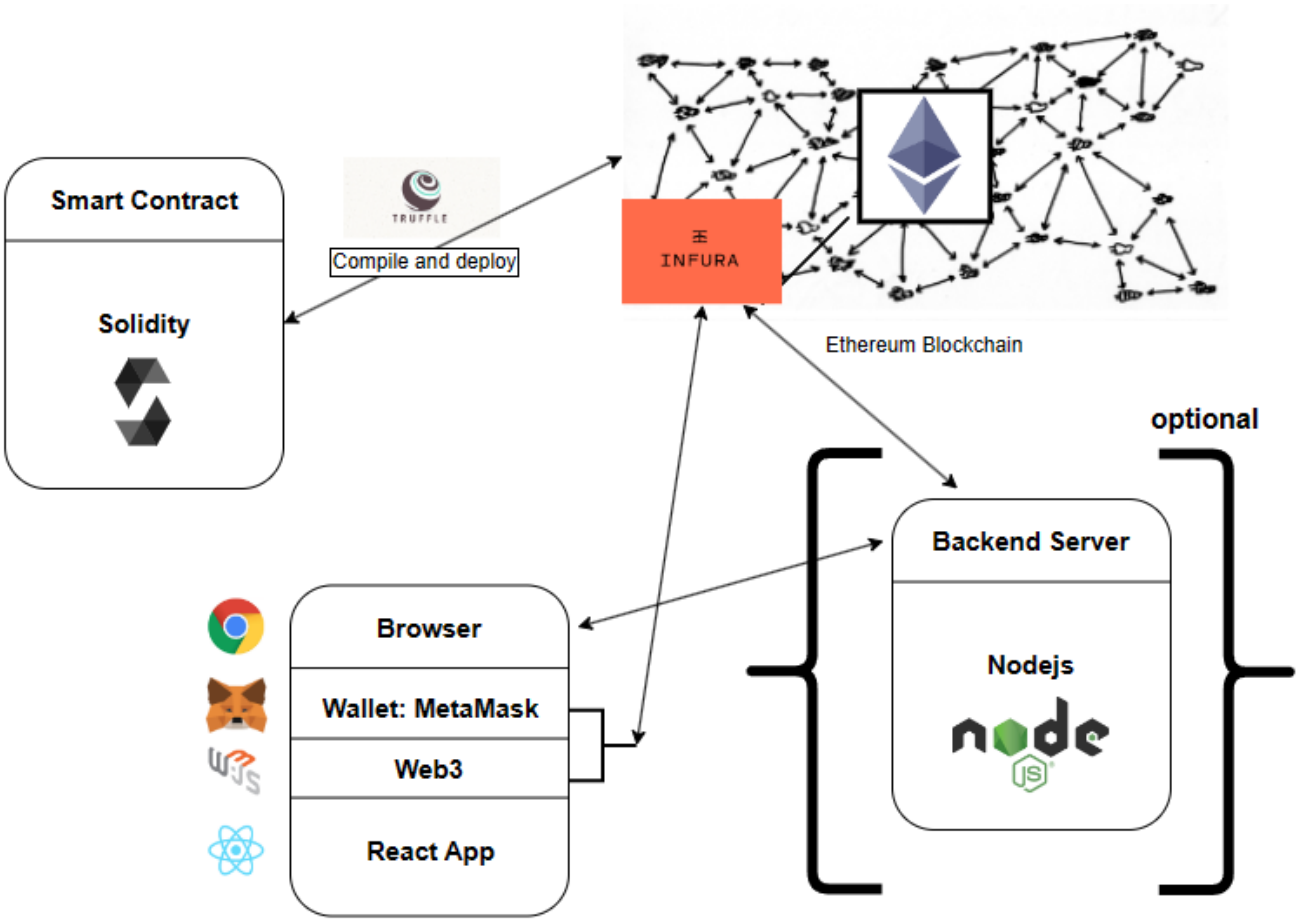


Figura 7.1: Arquitectura Dapp con servidor de respaldo [45]

Para diseñar una aplicación descentralizada, primero hay que entender cuál es la estructura de estas. En la Figura 7.1 podemos observar la arquitectura típica de una Dapp.

Por un lado, como todas las aplicaciones actuales, una Dapp contiene un *frontend* basado, por ejemplo, en React. Esta parte de la aplicación es ejecutada en un navegador, y gestiona la interacción con el usuario, aplicando lógicas sencillas. Junto con React, se necesitan usar herramientas que comuniquen el *frontend* con la red blockchain. En este caso, MetaMask junto con Web3, se encargan de esta función. MetaMask actúa como billetera, la cual recibe las peticiones que se realizan desde React hacia la blockchain. Web3 permite la realización de estas peticiones, así como obtener la información de la blockchain.

Como casi todas las aplicaciones, una Dapp necesita de un *backend* que gestiona la lógica de negocio. Para ello,

se puede usar directamente la red blockchain, donde se programa la lógica en smart contracts. Este modelo es el ideal para explotar las ventajas de usar una blockchain, pues respeta la descentralización. Por otro lado, es habitual tener un servidor de respaldo para ahorrar costes, pues el despliegue de smart contracts puede llegar a tener un alto coste. En este caso, Nodejs es una buena solución para desarrollar las lógicas de negocio que no requieren de descentralización. De esta manera se reducen drásticamente los costes asociados a los smart contracts, sacrificando parte de la descentralización.

Por otro lado, como ya expliqué anteriormente en el Capítulo 2, la red está creada de nodos conectados. Lo ideal en una blockchain es que cada usuario despliegue su propio nodo y se conecte a la red, sin embargo, esta aproximación puede ser complicada, ya que desplegar un nodo no es tarea fácil. Para solucionar este problema, hay diversos servicios, como Infura, que ofrecen nodos a los que un usuario puede conectarse. Esto evita las dificultades asociadas al despliegue, de nuevo, sacrificando en descentralización.

Sin embargo, puesto que el objetivo principal de este proyecto es el de aprender el funcionamiento de la blockchain se ha decidido salir del enfoque tradicional y diseñar una arquitectura más teórica, completamente descentralizada. Como podemos observar en la Figura 7.2, se omite el servidor externo a la blockchain, y por tanto, se desarrolla toda la lógica de negocio sobre los smart contracts programados en Solidity. Para el despliegue de estos contratos se utiliza Truffle. Además, en vez de usar nodos externos, como los de Infura, se ha decidido desplegar nodos individualmente para desplegar una red privada. De esta manera, se puede configurar la red según las necesidades, en vez de depender de la configuración de la red global Ethereum. Además, esto permite evitar los costes asociados al despliegue de contratos, ya que al ser una red privada, se posee mayor control de la moneda de la red.

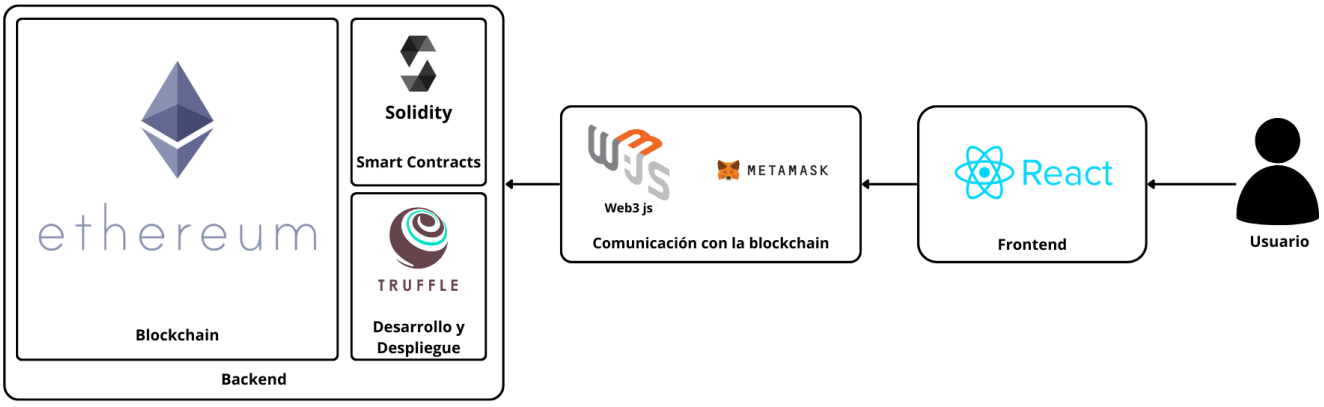


Figura 7.2: Arquitectura Dapp pura

7.2. Arquitectura de Smart contracts

En la figura 7.3 se muestra un modelo orientativo de los smart contracts que componen la lógica de negocio del *backend*.

Como se ha explicado en la sección anterior, los smart contracts son las piezas de código que contienen la lógica de negocio del *backend*. Esta lógica se divide en 3 contratos diferentes: App, SupplyChain y StepChain. App es el contrato principal que gestiona las cadenas de suministro creadas, así como los usuarios registrados y conecta con la cadena de Steps. Por otro lado, SupplyChain es el contrato que gestiona toda la lógica referente a lo que puede hacerse dentro de una cadena de suministro. Por último, StepChain es el contrato encargado de relacionar todos los productos, con sus steps, creados en las cadenas de suministro entre sí. De esta manera, se consigue tener una trazabilidad completa.

En el Capítulo 8 se ahondará más profundamente en cómo funcionan los smart contracts.

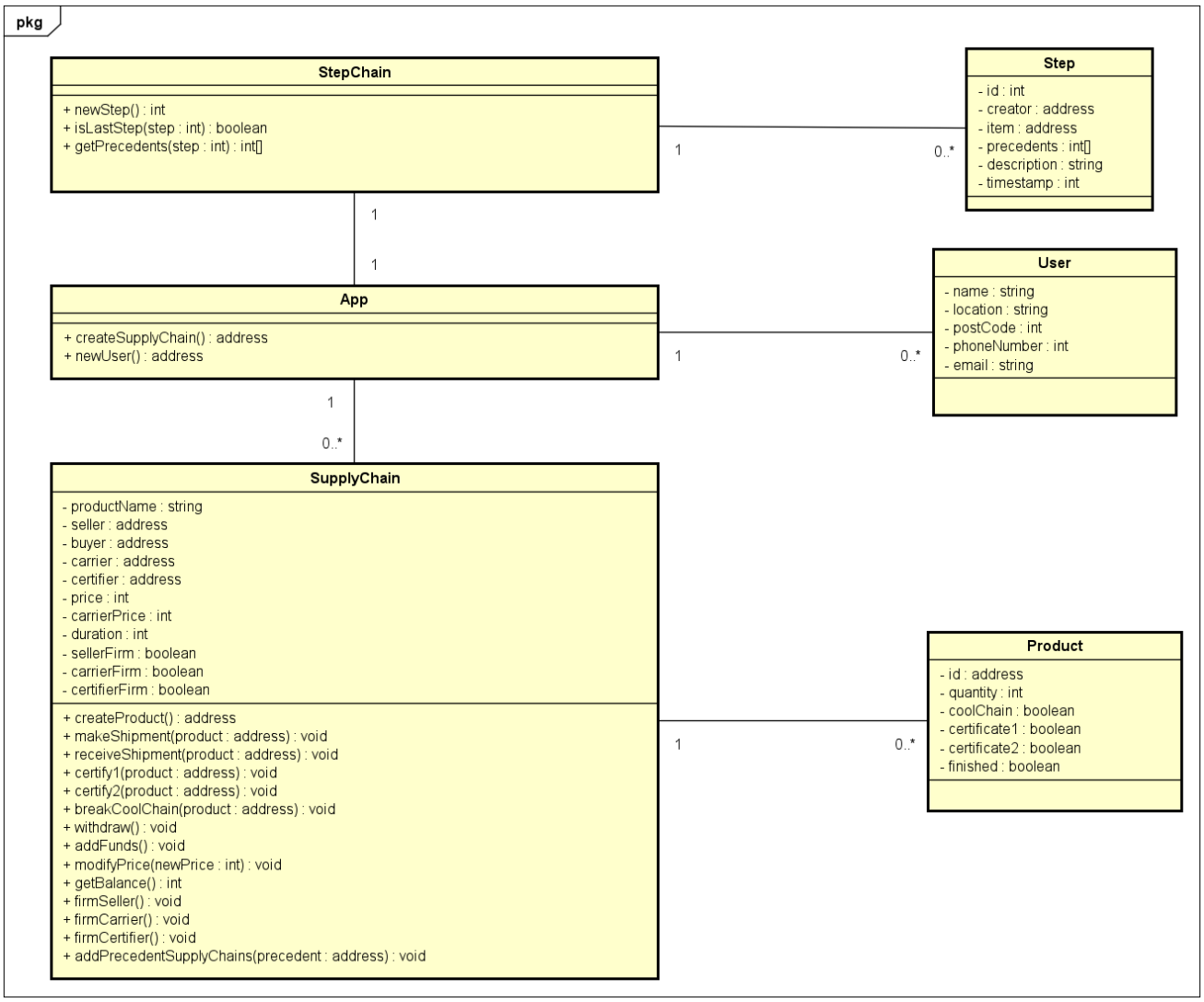


Figura 7.3: Modelo de dominio de los Smart contracts

7.3. Arquitectura del frontend

Para explicar la arquitectura final de los componentes React que se han desarrollado, se ha realizado un diagrama de componentes presentado en la Figura 3.1. En él se muestran los componentes creados, así como sus relaciones principales. Sirviéndonos de este diagrama procedo a explicar la arquitectura:

- **Componentes principales**: Index, AppRouter y AppRoot. Index simplemente renderiza el componente principal y crea el documento HTML. AppRouter es el componente que se encarga principalmente del enrutamiento de la aplicación. Esto significa que gestiona qué componentes han de mostrarse en función de la ruta de la página en la que el usuario esté. Además, gestiona el acceso a los proveedores tanto de la cuenta conectada desde MetaMask como de la red Ethereum a la que se está conectado. Por último, AppRoot se encarga de renderizar el menú lateral de la aplicación, así como de dejar un hueco para los componentes que se deberán renderizar en función de la ruta de la página en la que se esté.

- **Login y Home**: Login es el componente que se renderiza en la esquina superior derecha. Este se encarga de escuchar cualquier cambio que se produzca en la cuenta de MetaMask asociada y comunicarlo. Home consiste en el componente que se muestra al iniciar la aplicación.

- **Componentes de usuario**: Estos componentes son EditUser, User y UserList. Son los que gestionan las vistas asociadas a los usuarios. Permiten listar los usuarios, consultar la información de uno en concreto, y crear tu propio usuario.

- **Componentes de contratos**: Para acceder a la lógica de los contratos alojados en la blokchain, se crea un componente por cada contrado, es decir stepContract, appContract y supplyChainContract. Estos componentes aplican el patrón fachada para simplificar la comunicación con los contratos en la blockchain. A su vez, el componente contractManager, aplica el patrón singleton para asegurar que solo hay una entidad de cada contrato. Este gestor de contratos se provee a AppRouter para que distribuya su acceso a todo aquel componente que lo necesite.

- **Componentes de la cadena de suministro**: Estos son todos los componentes que dan funcionalidad a la cadena de suministros. Dependiendo de que quiera hacer el usuario, se muestra el componente para crear una cadena de suministros, la lista con las cadenas de suministros creadas, o el supplyChainLayout para mostrar la información de una cadena en concreto.Este último componente gestiona los espacios que han de ocupar cada uno de los otros componentes que se rendericen.

- **Componentes de los productos:** Estos componentes dan funcionalidad a los productos dentro de una cadena de suministro. Al igual que con los componentes de las cadenas de suministros, hay un componente para crear nuevos productos, otro para listarlos y uno último que sirve de gestor de espacios para el resto de acciones que se pueden realizar sobre un producto en concreto.

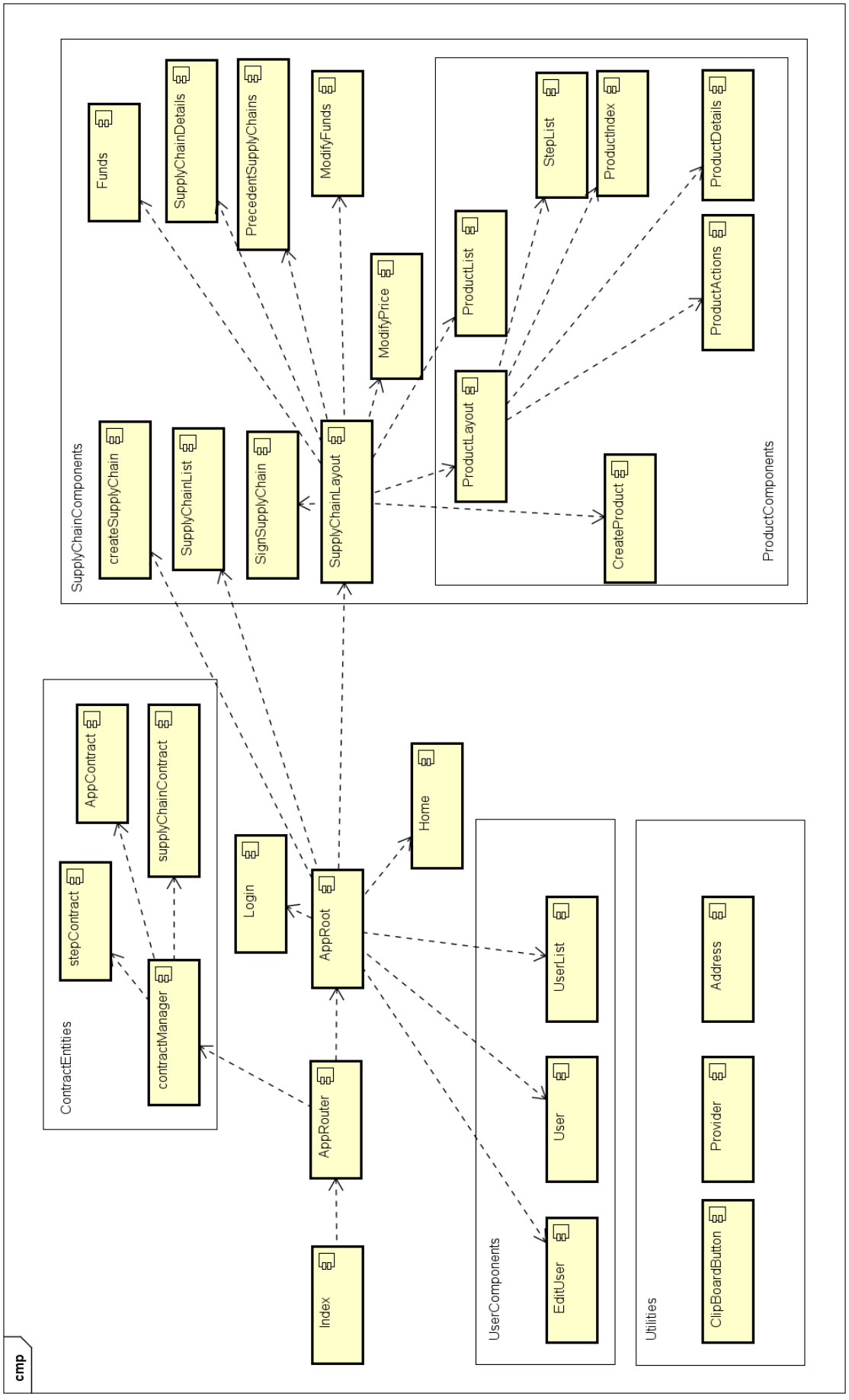


Figura 7.4: Estructura de componentes del frontend React

Capítulo 8

Implementación

Una vez realizado el análisis y el diseño, es el momento de desarrollar la aplicación. Este capítulo pretende explicar esta fase del proceso, explicando de una forma detallada cada una de las partes que componen la aplicación.

8.1. Smart contracts

Tomando como base el modelo de dominio desarrollado en la Sección 7.2, se han desarrollado tres smart contracts que codifican toda la lógica de la aplicación. Estos son los siguientes: **App**, **StepChain** y **SupplyChain**.

8.1.1. App

Este contrato es el situado en un nivel más alto de los tres, y por tanto, el que crea las instancias de los otros dos. Como se puede observar en Figura 8.1, este contrato almacena una lista de cadenas de suministro y otra de usuarios. Además, crea una única instancia del contrato stepChain y define las variables que contendrá cada usuario registrado. Cabe destacar que referente a las cadenas de suministro, lo que se guarda es la dirección del contrato que hace referencia a la cadena. Sin embargo, la dirección de los usuarios que se guarda, hace referencia a la dirección de la cuenta que ha enviado la transacción de registro.

```
address[] public supplyChainList;
StepChain public stepChain = new StepChain();
mapping(address => User) public userList;
address[] public userAddresses;
struct User {
    string name;
    string location;
    uint256 postCode;
    uint256 phoneNumber;
    string email;
    uint256 createdAt;

    bool init;
}
```

Figura 8.1: Variables declaradas en el contrato App

A continuación, en Figura 8.2 se encuentran las funciones del contrato. Por un lado, los *getters* tanto de los usuarios registrados como de las cadenas de suministro creadas. También hay una función para la creación de usuarios, que aparte de registrarlos, comprueba antes que no estén previamente registrados.Por último, se define la función de creación de cadenas de suministros. Esta función crea un nuevo contrato SupplyChain, con los parámetros

suministrados en la transacción, y finalmente guarda la dirección del contrato, para poder acceder a el en caso de que sea necesario.

En definitiva, este contrato es únicamente necesario para mantener un registro de todas las cadenas de suministros y de los usuarios, para acceder a ellos de forma rápida y organizada.

```
function createSupplyChain(
    string memory _productName,
    address _seller,
    address _buyer,
    address _carrier,
    address _certifier,
    uint256 _price,
    uint256 _carrierPrice,
    uint256 _duration
    ) public {
    supplyChainList.push(
        address(
            new SupplyChain(
                _productName,
                _seller,
                _buyer,
                _carrier,
                _certifier,
                _price,
                _carrierPrice,
                _duration,
                address(stepChain)
            )
        )
    );
}
function newUser(
    string memory _name,
    string memory _location,
    uint256 _postCode,
    uint256 _phoneNumber,
    string memory _email
    ) public {

    require(userList[msg.sender].init==false,"User already exists for the related account");
    User memory user =  User(_name,_location,_postCode,_phoneNumber,_email,block.timestamp,true);
    userList[msg.sender]= user;
    userAddresses.push(msg.sender);
}

function getSupplyChainList() external view returns (address[] memory) {···
}



function getUsers() public view returns(User[] memory,address[] memory){···
```

Figura 8.2: Funciones del contrato App

8.1.2. StepChain

Este contrato es el encargado de permitir la trazabilidad de los productos que se creen en las cadenas de suministro. La idea que reside en él es la de construir una red con todos los productos que se creen, registrando los cambios que se produzcan en ellos y conectándolos con los productos que los preceden y suceden [13]. De esta manera, se crea un grafo acíclico por el cual es posible navegar para obtener una trazabilidad completa de cualquier

producto. La Figura 8.3 ilustra este grafo. Se pueden observar cuatro productos diferentes, representados por cada color del Step. En el primero de ellos, la "leche cruda tiene varios pasos: creación, certificación, envío y entrega. Estos pasos son realizados por diferentes usuarios. Por otro lado, el "chocolate"solo tiene los pasos de creación, envío y entrega. Una vez, el usuario "Industriarecibe la leche, puede realizar el proceso de ultrapasteurización. Es interesante comprobar que los procesos internos que realiza un usuario, como es la transformación de la leche cruda a leche UHT , pueden quedar reflejados en el grafo. A su vez, con el chocolate recibido de otro proveedor y la leche UHT por él mismo, la Industria puede reflejar la creación de otro producto: "leche con chocolate". Este ejemplo haría referencia a una parte de la cadena, pero esta podría alargarse tanto como los participantes quisiesen, añadiendo por un lado el proceso de recolección de cacao y su procesamiento, así como la distribución de la leche hasta los puntos de venta y los consumidores finales. Con esta implementación, se ve fácilmente que cualquier producto puede ser rastreado, hasta los orígenes de las materias primas que lo producen. De esta manera podrían resolverse de forma rápida multitud de problemas.

Referente al código, en la Figura 8.4 se puede observar una visión general. Primero, se define la estructura Step, donde el timestamp hace referencia a un id único del Step. A continuación, se crea una variable donde se almacenarán todos los steps que se vayan creando. Por último las funciones que implementan el grafo explicado anteriormente:

- **newStep:** Permite la creación de nuevos steps, tanto de productos nuevos como de productos ya existentes en la cadena.
- **isLastStep:** Esta función en necesaria para asegurar que la creación de nuevos Steps de productos ya existentes se añadan al final. De esta manera, se asegura que todo producto tenga un único estado(como podrían ser: envío, certificado, entregado...).
- **getSteps:** Permite recuperar la información de múltiples Steps.
- **getPrecedents:** Permite recuperar los Steps que preceden a uno en concreto. De esta manera, es posible navegar por el grafo hasta los steps iniciales.

8.1.3. SupplyChain

Este contrato facilita la gestión de una cadena de suministro, permitiendo la creación y seguimiento de productos, el manejo de fondos de las partes involucradas, y asegurando que todas las condiciones necesarias (como certificaciones y mantenimiento de la cadena de frío) se cumplan antes de finalizar el proceso. Cada evento significativo en la cadena de suministro se registra mediante la interacción con el contrato StepChain, proporcionando un historial claro y verificable de las acciones realizadas sobre cada producto.

Para entender la funcionalidad que brinda este contrato, procederé a ir explicando el código realizado. Por un lado, las variables definidas:

- **Participantes:** Estas variables son las que hacen referencia a las direcciones de cuenta del comprador, vendedor, transportista y certificador de la cadena de suministro. Serán necesarias para restringir el acceso a funcionalidades que dependen del rol que tiene el usuario con respecto al contrato.
- **Finanzas:** Entre ellas están las variables que establecen el precio unitario del producto, el precio que ha de pagarse al transportista por envío y los fondos que están almacenados en el contrato. Además, es necesario un mapeo de los fondos asociados a todas las cuentas que interactúan con el contrato. Esta es la función de la variable "pendingWithdrawals".
- **Firmas:** Las variables "sellerFirm", "carrierFirm", "certifierFirm" y "init" indican si los participantes de la cadena de suministro han aceptado los términos de la misma, como pueden ser los precios y la duración. Una vez todos han aceptado, se marca la cadena como iniciada.
- **Productos:** Por un lado, se necesita la estructura "Product" que almacena toda la información necesaria de cada producto creado. Esta información es: un identificador, la cantidad, dos certificados(uno en origen

y otro en destino), una variable que comprueba si la cadena de frío se ha roto y por último, otra variable que señala si el producto ha terminado todos sus procesos. Por otro lado, se necesita una lista que almacene todos los productos creados en la cadena de suministro("productList").

- **Otras variables:**
  - **ProductName:** Almacena el nombre del producto de la cadena de suministro.
  - **Duration:** Marca la duración hasta la cual el contrato es efectivo. Cuando llega dicha fecha, se desactiva.
  - **PrecedentSupplyChains:** Almacena las direcciones de otras cadenas de suministro que se hayan establecido como precedentes.
  - **StepChain:** Dirección del contrato "StepChain" creado para rastrear todos los pasos que se han realizado en los productos.

Una vez expuestas las variables, es momento de explicar todas las funciones que contiene el contrato.

- **Funciones Getters**
  - **getProductList():** Devuelve una lista con todos los productos creados hasta la fecha.
  - **getPrecedents():** Devuelve las direcciones de las cadenas de suministro asignadas como precedentes.
- **Funciones de gestión de productos**
  - **getProduct(address id):** Devuelve un producto específico por su ID.
  - **createProduct(uint256 _quantity, address[] memory _precedentProducts):** Crea un nuevo producto, validando los productos precedentes y generando un ID único basado en la dirección del remitente y el *timestamp*.
  - **makeShipment(address _productId):** Marca un producto para envío.
  - **receiveShipment(address _productId):** Marca la recepción de un producto, actualiza los fondos pendientes y finaliza el producto.
  - **certify1(address _productId):** Marca un producto como certificado en origen.
  - **certify2(address _productId):** Marca un producto como certificado en destino.
  - **breakCoolChain(address _productId):** Marca una ruptura en la cadena de frío de un producto.
- **Funciones de finanzas**
  - **withdraw():** Permite a los participantes retirar sus fondos alamcenados en el contrato.
  - **addFunds():** Permite al comprador agregar fondos al contrato.
  - **modifyPrice(uint256 newPrice):** Permite al vendedor o comprador proponer una nueva modificación de precio, requiriendo que ambas partes acuerden el nuevo precio.
- **Funcioens de firmas e inicialización**
  - **firmSeller():** Registra la firma del vendedor.
  - **firmCarrier():** Registra la firma del transportista.
  - **firmCertifier():** Registra la firma del certificador.
  - **initContract():** Inicializa el contrato si todas las partes han firmado.
- **Otras funciones**
  - **addPrecedents(address[] memory _precedentProducts):** Permite al vendedor agregar productos precedentes.
  - **isValidPrecedentProduct(address productId):** Verifica si un producto precedente es válido.

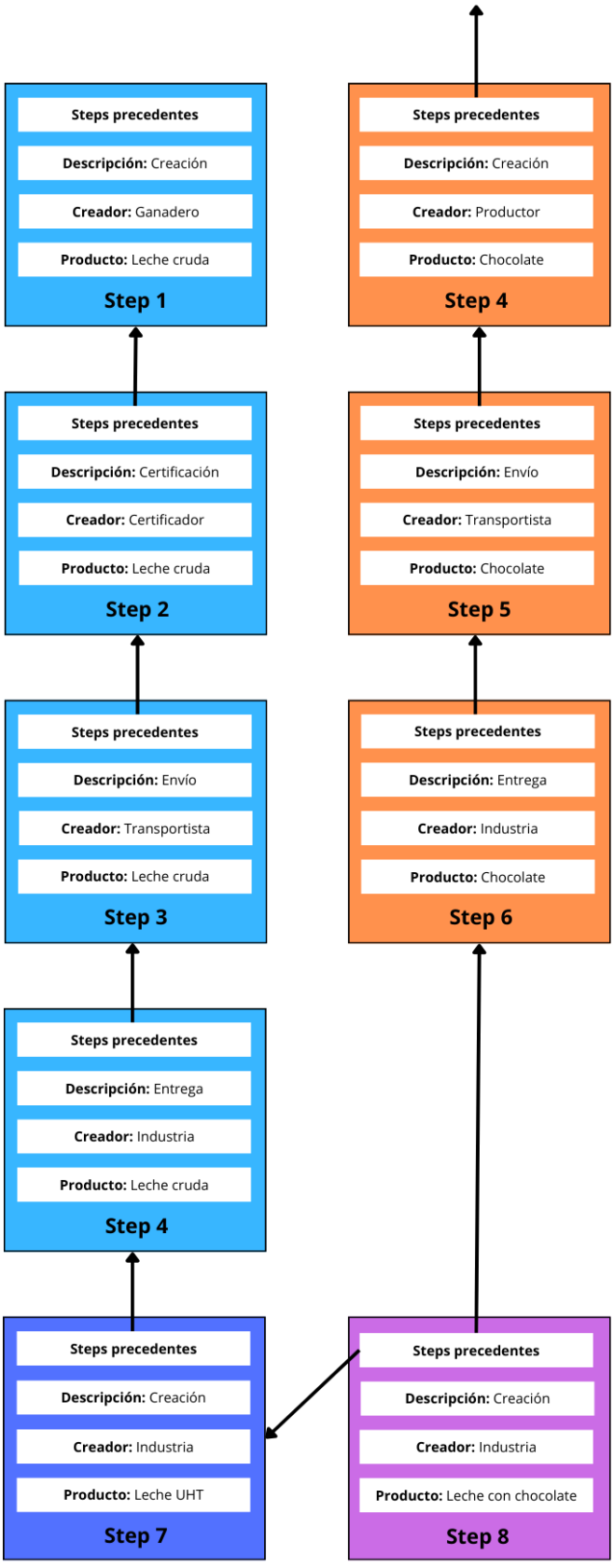


Figura 8.3: Ejemplo de una cadena de steps

```
struct Step {
    address creator;
    address item;
    uint256[] precedents;
    string description;
    uint256 timestamp;
}
/**...
mapping(uint256 => Step) public steps;
/**...
uint256 public totalSteps;

function getSteps() public view returns (Step[] memory) {
    Step[] memory ret = new Step[](totalSteps);
    for (uint256 i = 0; i < totalSteps; i++) {
        ret[i] = steps[i];
    }
    return ret;
}

/**...
mapping(address => uint256) public lastSteps;

/**...
function newStep(...
) public returns (uint256) {...
}
/**...
function isLastStep(uint256 _step) public view returns (bool) {...
}
/**...
function getprecedents(uint256 _step)...
{
    return steps[_step].precedents;
}
```

Figura 8.4: Código de las variables del contrato SupplyChain

```
string public productName;
address public owner;
address public seller;
address public buyer;
address public carrier;
address public certifier;
uint256 public funds;
uint256 public duration;
uint256 public price;
uint256 public carrierPrice;
mapping(address => uint256) public pendingWithdrawals;
address[] public precedentSupplyChains;

//Sign cnotract variables
bool public init= false;
bool public sellerFirm = false;
bool public carrierFirm = false;
bool public certifierFirm = false;

StepChain public stepChain;
struct Product {
    uint256 quantity;
    address id;
    bool coolChain;
    bool certificate1;
    bool certificate2;
    bool finished;
}
uint256 public productCount;
Product[] public productList;
mapping(address => uint256) public productMapping;
```

Figura 8.5: Código del contrato StepChain

8.2. Implementación blockchain

8.2.1. Archivo génesis

En una blockchain, el "genesis" o bloque génesis, es el primer bloque de una cadena de bloques. Este bloque es único porque no tiene predecesor (su parentHash es 0) y, por tanto, define el estado inicial de la red. Para crear una red privada, el primer paso por tanto, es crear este bloque. En Ethereum, el archivo encargado de configurar este bloque es el llamado `genesis.json`. Este archivo puede contener información como la asignación inicial de recursos y los parámetros de consenso que se usarán en la red. Es el punto de partida a partir del cual se construye toda la historia de la blockchain. Para entender cómo funciona el archivo génesis y qué información contiene, usaré

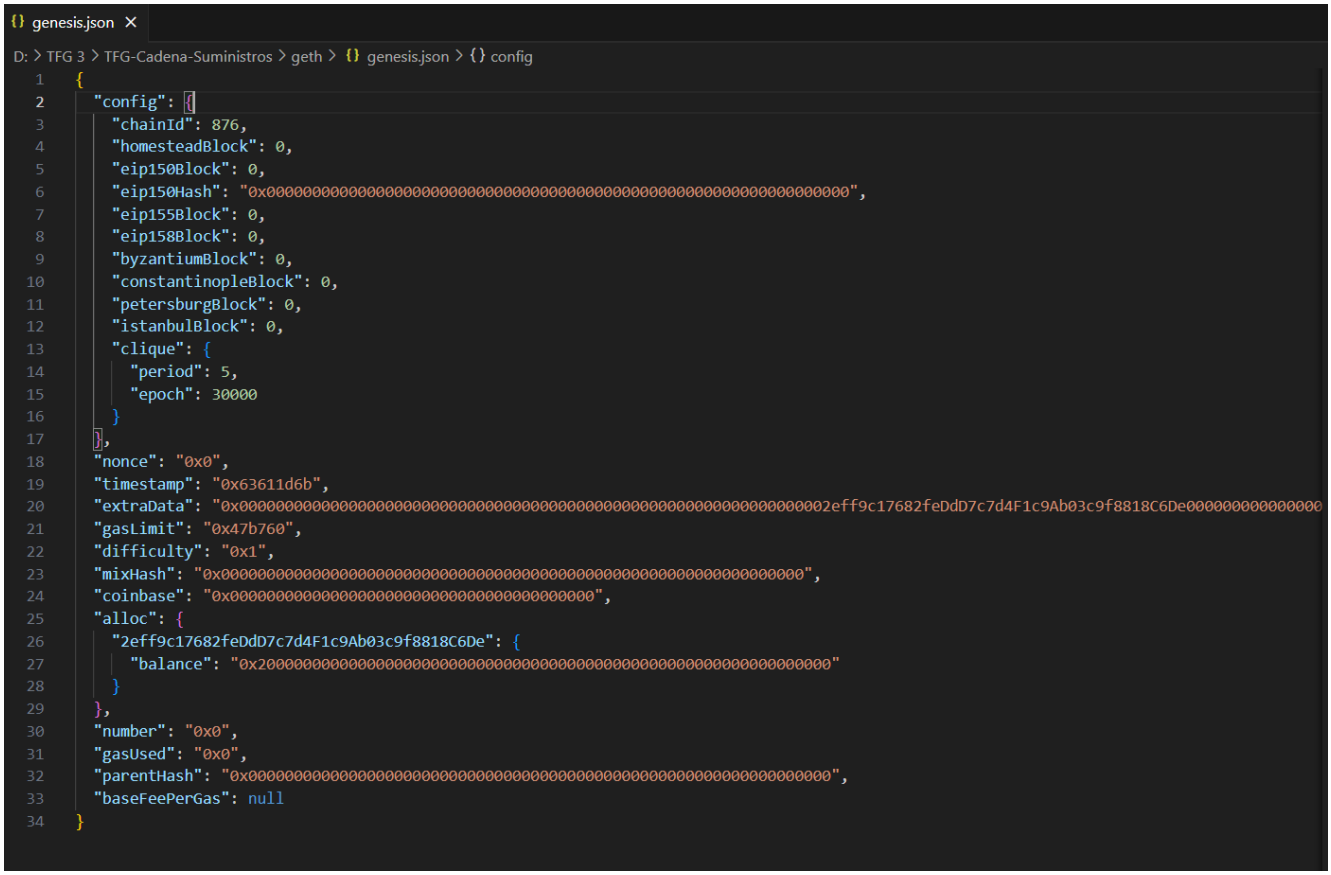


Figura 8.6: Ejemplo de archivo génesis

de ejemplo la Figura 8.6. De toda la información que contiene el génesis, solo los siguientes atributos son únicos de dicho bloque:

- **Config:** Esta sección contiene la configuración de la red, incluyendo la identificación de la cadena y las actualizaciones del protocolo. A su vez, estos atributos podemos dividirlos en varios grupos:
  - **ChainId:** Es un número único que actúa como identificador de la red. De esta manera, solo aquellos nodos con un mismo identificador, podrán conectarse entre sí. Por ejemplo, la red principal Ethereum tiene el identificador de red 1.
  - **Atributos de actualizaciones:** Aquí se incluyen homesteadBlock, eip150Block, istanbulBlock, etc. Todos estos atributos hacen referencia a actualizaciones que se han ido implementando a lo largo del tiempo en el protocolo Ethereum. Cuando una red necesita ser actualizada, lo que se hace es modificar el genesis y añadir un atributo de actualización. El valor que toma, se corresponde con el bloque a partir del cual la actualización toma efecto. Al crear una nueva red privada, se presupone que todas las actualizaciones ya forman parte de la red, y por tanto el bloque al que hacen referencia es el 0.
  - **Algoritmo de consenso:** Por último, se puede especificar el algoritmo de consenso que se quiere emplear en la cadena. Desde el genesis se presentan dos opciones posibles: Clique o Ethash. Por un

lado, clique implementa Proof of Authority, muy útil si lo que se quiere es ahorrar recursos. Se puede especificar un periodo, que marcará cada cuánto tiempo se crean los bloques, y el epoch que señala cada cuánto se reorganizan lso validadores. Por otro lado, Ethash implementa Proof of Work. De esta manera, hará falta emplear mineros que validen las transacciones. En el caso de Ethash no hace falta definir ningún parámetro adicional.

- **Alloc:** Bajo este parámetro se especifica la distribución de Ether que habrá en la red. Como se aprecia en la Figura 8.6, basta con especificar una o varias cuentas y añadir el atributo de balance, con la cantidad de Ether que se quiere que posean. Por otro lado, también es posible desplegar contratos directamente. Esto es posible ya que los contratos están asociados a una dirección de cuenta. Sin embargo, además del balance, habría que añadir otros atributos como el código.

Además de los atributos únicos, el génesis tiene que contener el resto de atributos que tienen todos los bloques. Estos son los siguientes:

- **nonce**: Un valor utilizado en el proceso de minería para generar un hash válido. En el caso del génesis, su valor es arbitrario.
- **timestamp**: Marca de tiempo que indica cuándo se generó el bloque.
- **extraData**: Datos adicionales que pueden ser utilizados para incluir información arbitraria, como la lista de validadores en redes PoA. En el caso de la Figura 8.6, se puede observar cómo se ha insertado en este campo, una dirección de cuenta, correspondiente al validador. Esto es necesario, pues anteriormente se eligió emplear el algoritmo de consenso Clique.
- **gasLimit**: Límite máximo de gas permitido en el bloque.
- **difficulty**: Representa la dificultad de minar el bloque. De este valor depende con qué rapidez se crean nuevos bloques, y por tanto, está estrechamente relacionado con la seguridad en la red. Aunque su valor cambia dinámicamente en bloques posteriores, el bloque génesis ha de establecer una dificultad inicial.
- **mixHash**: Un valor hash utilizado en el proceso de minería.
- **coinbase**: La dirección que recibirá la recompensa por haber minado el bloque. En el caso del génesis, este valor suele ser 0.
- **parentHash**: El hash del bloque padre. Al ser el bloque inicial, no hay padre, y su valor es 0.

8.2.2. Introducción a Geth

Como ya se comentó en la Sección 3.2, Geth es un cliente que implementa el protocolo Ethereum [18]. Por tanto, permite la creación de nodos y su interconexión, formando así una red. Esta herramienta, es principalmente controlada mediante linea de comandos. A pesar de permitir multitud de configuraciones, realizar operaciones básicas es relativamente sencillo. Por ejemplo, para ejecutar un nodo básico en la red principal de Ethereum es tan solo necesario ejecutar el comando `geth`. Para acceder a la mayoría de funcionalidades, es necesario el uso de flags, sin embargo, algunas otras funcionalidades están especificadas como comandos. Algunas de los comandos y flags más importantes son los siguientes:

- `geth --datadir path/to/data`**:** Permite especificar un directorio donde se almacenará toda la información referente a la blockchain.
- `geth --networkid 12345`**:** Todas las redes Ethereum tienen un identificador de cadena, que permite a los nodos de una misma cadena conectarse entre sí. Este flag especifica a qué red queremos conectarnos. Para redes conocidas hay flags especiales que evitan especificar su identificador de cadena, por ejemplo `--mainnet` y `--goerli`.

- `geth account new`**:** Este comando permite la creación de una nueva cuenta, asociada al nodo. Al introducir una contraseña, se crea una clave privada, almacenada en el directorio keystore, asociada a una dirección de cuenta pública.

- `geth --http`**:** Habilita el servidor HTTP-RPC, que permite que al nodo escuchar y responder a solicitudes HTTP. Junto con esta flag, suelen usarse otras como `--http.port portNumber` y `--http.api web3,eth,net...` que complementan a la primera, para permitir interactuar con el nodo, mediante HTTP.

- `geth --nodiscover`**:** Útil para redes privadas, donde se busca que nodos desconocidos no puedan conectarse a la red, manteniendo así la seguridad de la misma. A mayores, existen multitud de otras opciones relacionadas con la gestión de conexiones, que permiten tener un control personalizado de quiénes pueden o no conectarse.

- `geth --mine`**:** En redes basadas en PoW, se pueden seleccionar diferentes configuraciones con respecto al minado. `--mine`, por ejemplo, permite al nodo usar recursos para minar bloques. También es posible seleccionar qué dirección de cuenta recibirá la recompensa (`--miner.etherbase account`) y establecer precios de gas (`--miner.gasprice value`) entre otras opciones.

- `--console`**:** Además de iniciar el nodo, abre una consola que permite interactuar con las APIs directamente. De esta manera, se pueden enviar o firmar transacciones, consultar el estado de la red, activar las opciones de minado, etc.

8.2.3. Despliegue de una red privada básica

Para entender cómo funciona Geth, la mejor manera es experimentar con la herramienta, y un caso idóneo es desplegar una red privada. Conseguir dicho fin, implicará entender cómo se despliega un nodo, cómo se interactúa con el mismo(a través de las APIs) y cómo se conectan el resto de herramientas con la red (MetaMask y Truffle).

El proceso que se explicará a continuación, se corresponde con el seguido en el Apéndice A. Este proceso, desarrolla cómo desplegar un único nodo, cómo usar una cuenta para que valide los bloques(y por tanto, las transacciones), y cómo conectar un cliente MetaMask que dé uso a la aplicación. Al ser una aproximación rápida, este procedimiento es sólo recomendado en entornos de prueba, dónde la seguridad no es un aspecto esencial.

Creación del genesis y una cuenta

El primer paso para crear una red Ethereum, es definir los parámetros de configuración que la darán forma. Para ello, como se vió en la Subsección 8.2.1, se usa el archivo `genesis.json`. Uno de los parámetros claves del genesis, es una cuenta que será necesaria para validar las transacciones. Para crear esta cuenta se usa el siguiente comando: `geth account new --datadir Chain`. Este comando pedirá una contraseña y almacenará la clave privada en ./Chain/keystore. Nótese que se usa el flag –datadir simplemente para almacenar la red en un lugar fácilmente localizable.

Para este ejemplo, se ha utilizado el mostrado en Figura 8.6. Los parámetros más relevantes son:

- **chainId:** Este valor, 876, identifica la red y será relevante para conectar MetaMask y Truffle. A su vez, sería la puerta de entrada para poder conectar otros nodos.

- **Clique:** Se ha elegido Clique debido a que implementa un consenso basado en PoA. Las redes privadas, habitualmente son pequeñas, y por tanto, su seguridad aplicando un mecanismo PoW puede ser cuestionable. Gracias a usar PoA, nos aseguramos de que las transacciones serán validadas por una cuenta de confianza. Además, evitamos tener que invertir grandes recursos de computación.

- **Extradata:** En el caso de usar Clique, este campo es importante, puesto que en él se incluyen las cuentas que se encargarán de validar las transacciones. En nuestro caso, hay que añadir la cuenta que hemos creado anteriormente.

- **Alloc:** Como la cuenta validadora será el administrador de la red, se le asigna el suministro de Ether. De esta manera, será el encargado de distribuir el Ether a quién corresponda.

Ejecución del nodo

Una vez definido el genesis, es necesario inicializar la red. Para ello, se usa el comando `geth --datadir Chain init genesis.json` Este comando crea la carpeta ./Chain/geth con información necesaria para la ejecución de la nueva red.

A continuación, hay que desplegar el nodo. El comando empleado ha sido el siguiente:

```
geth --networkid 876--datadir Chain --nodiscover --http --http.api web3,eth,personal,net --http.port 8406--http.
    corsdomain "*" --http.addr "0.0.0.0" --http.vhosts "*" --unlock "0x2eff9c17682feDdD7c7d4F1c9Ab03c9f8818C6De" --
    mine --allow-insecure-unlock --miner.etherbase="0x2eff9c17682feDdD7c7d4F1c9Ab03c9f8818C6De"
```

Como se puede observar, hay multitud de flags que han sido necesarias. Cada una tiene una utilidad que explico a continuación:

- `--networkid 876`: Especifica el ID de la red a la que se quiere conectar. Como en el genesis se eligió el ID 876, es el que debemos emplear aquí.
- `--datadir Chain`: Define el directorio de datos donde se almacenará la cadena de bloques y los datos de la cuenta.
- `--nodiscover`: Desactiva la funcionalidad de descubrimiento de nodos, lo que significa que el nodo no buscará otros nodos ni será encontrado por ellos. De esta manera, evitamos posibles problemas de ataques a nuestra red.
- `--http`: Habilita la interfaz HTTP para poder interactuar con el nodo mediante solicitudes HTTP. Esto es necesario ya que MetaMask utiliza HTTP para enviar las transacciones al nodo.
- `--http.api web3,eth,personal,net`: Especifica las APIs que estarán disponibles a través de HTTP. En este caso, las APIs disponibles son `web3`, `eth`, `personal` y `net`.
- `--http.port 8406`: Define el puerto en el que la interfaz HTTP escuchará. En este caso, el puerto es 8406. Este puerto será el usado tanto por Metamask como por Truffle para conectar con el nodo desplegado.
- `--http.corsdomain "*"`: Configura el encabezado CORS (Cross-Origin Resource Sharing) en el servidor HTTP. CORS es un mecanismo de seguridad que permite que los recursos de una página web sean solicitados desde un dominio diferente al de la propia página. De esta manera, permitimos conectar a múltiples clientes con nuestro nodo, y que lo use para comunicarse con la red blockchain y sus contratos. Sin embargo, esto no es una buena práctica para ninguna de las dos partes, como ya explicaremos en la próxima sección.
- `--http.addr "0.0.0.0"`: Configura la dirección IP a la que el servidor HTTP estará vinculado. En este caso, 0.0.0.0 indica que el servidor escuchará en todas las interfaces de red disponibles en el sistema. Esto significa que el servidor será accesible tanto desde la red local como desde Internet
- `--http.vhosts "*"`: Configura los hosts virtuales del servidor HTTP. Un host virtual permite que un único servidor físico aloje múltiples dominios o sitios web. El asterisco indica que se aceptarán todas las solicitudes de hosts virtuales, lo que significa que el servidor responderá a todas las solicitudes de nombres de dominio sin importar cuál sea el host.
- `--unlock "0x2ef...6De"`: Desbloquea la cuenta especificada por la dirección para poder usarla en transacciones y minería. Esta cuenta corresponde con la que se creó anteriormente para validar las transacciones.
- `--mine`: Inicia el proceso de minería en el nodo. Al estar usando Clique, lo que indica es que las cuentas de validación pueden empezar a comprobar los bloques.
- `--allow-insecure-unlock`: Permite el desbloqueo de cuentas de manera insegura. Esto es necesario puesto que necesitamos desbloquear la cuenta validadora. Como estamos exponiendo el nodo de forma que pueda recibir peticiones HTTP, Geth bloquea el despliegue del nodo ya que incurre en problemas graves de seguridad, donde mediante peticiones HTTP se podría controlar la cuenta que hemos desbloqueado y usarla para enviar transacciones. Para evitar este bloqueo, Geth permite el uso del flag `allow-insecure-unlock`. Sin embargo, no debería usarse en un entorno real, como veremos en la próxima sección.

- `--miner.etherbase="0x2ef...6De"`: Especifica la cuenta que recibirá las recompensas de la minería. Aunque usando PoA, no hay recompensas de minado, Geth obliga a seleccionar una cuenta. De nuevo, usamos la cuenta creada anteriormente.

Una vez ejecutado el comando anterior, nos pedirá introducir la contraseña de la cuenta que hemos creado. Tras introducirla, ya tendremos un nodo operativo ejecutándose en nuestra máquina y validando bloques.

Despliegue con Truffle

Una vez se ha desplegado el nodo, podríamos usar las APIs y la cuenta creada, para desplegar directamente, sin embargo, como ya se mencionó en la Sección 3.6, se ha elegido usar ya que es Truffle una práctica herramienta para la compilar y desplegar smart contracts de forma sencilla. Por un lado, Truffle necesita de un fichero de configuración `truffle-config.js`. A pesar de que ofrece un amplio abanico de posibilidades [43], en este proyecto se ha usado el archivo que se muestra en la Figura 8.7. En este caso, usamos el archivo de configuración para tres tareas:

- Modificar el directorio dónde se almacenarán los archivos ABI de los contratos, generados al compilar.
- Definir las características de la red en la que queremos desplegar los smart contracts. Como se puede ver, hay que indicar la IP de la máquina donde se está ejecutando el nodo (en este caso la máquina local), así como el puerto que se ha asignado para la escucha(este puerto ha de coincidir con el que se seleccionó anteriormente en el despliegue del nodo). Además el parámetro `from`, indica qué cuenta se empleará para desplegar los contratos. Se podría usar cualquiera, pero como ya hemos creado una anteriormente, y posee Ether, la utilizaremos.
- Definir los parámetros del compilador, entre ellos la versión y el optimizador.

Por otro lado, hace falta un script que será el encargado de desplegar los contratos cuando se ejecute. En la Figura 8.8 se ve el código empleado. En nuestro caso, simplemente se cargan los contratos en el script, para posteriormente desplegarlos. Sin embargo, en este script es común añadir configuraciones avanzadas para los contratos, interacción con contratos ya desplegados o con los mismos y muchas otras configuraciones. Finalmente, es necesario ejecutar los comandos `truffle compile` y `truffle migrate` para llevar a cabo el despliegue de los contratos.

truffle-config.js

D: > TFG 3 > TFG-Cadena-Suministros > truffle > truffle-config.js > <unknown> > networks

```
module.exports = {
  contracts_build_directory: "../client/src/build/contracts",
    // See <http://truffleframtruework.com/docs/advanced/configuration>
    // for more about customizing your Truffle configuration!
    networks: {
      development: {
        host: "127.0.0.1",
        port: 8406,
        network_id: "*", // Match any network id
        production:true,
        from: "0x2eff9c17682feDdD7c7d4F1c9Ab03c9f8818C6De"

      },
    },
    compilers: {
        solc: {
          version: "0.8.8",
          settings: {
            optimizer: {
              enabled: true,
              runs: 10
            }
          }
        }
    },

};
```

Figura 8.7: Archivo de configuración de Truffle

D: > TFG 3 > TFG-Cadena-Suministros > truffle > migrations > 1_deploy_simple_storage.

```
const App = artifacts.require("App");
const StepChain = artifacts.require("StepChain");
const SupplyChain = artifacts.require("SupplyChain");
module.exports = function (deployer) {
  deployer.deploy(App);

};

```

Figura 8.8: Archivo de despliegue usando Truffle

Conexión con MetaMask

MetaMask es considerado frontend de la aplicación, y el uso de su API se realiza desde los componentes React. Sin embargo, MetaMask es el intermediario entre React y la red blockchain. Es por ello que hace falta configurar una conexión. Esta configuración se hace desde la extensión de MetaMask, siguiendo los siguientes pasos:

1. Abrir Metamask/Configuración/Redes/Agregar una red/Agregar una red manualmente. Nos encontraremos un formulario para rellenar.
2. Rellenar el formulario como en la Figura 8.9:
   - **Nombre de la red:** Podemos escribir el nombre que queramos para identificar esta red.
   - **Dirección URL de RPC:** Esta dirección tiene que corresponderse con la del nodo al que queremos conectarnos. Anteriormente, en la creación del nodo, le asignamos el puerto 8406. Como estamos en local, la dirección se corresponde por tanto a http://127.0.0.1:8406. En caso de que quisiéramos conectarnos desde otra máquina, deberíamos obtener la dirección de la máquina en la que hemos desplegado el nodo. Como anteriormente hicimos una configuración que permite conectarse al nodo de forma externa al localhost, sería una configuración funcional. Sin embargo, no sería del todo segura.
   - **Identificador de cadena:**. Este identificador ha de corresponderse al empleado en el flag –networkid usado en el despliegue de la red Ethereum. Como se ha usado un identificador de cadena relativamente común, aparece un mensaje de aviso. Sin embargo, al ser una red privada, y no buscar otros nodos, no hay problema.
   - **Símbolo de cadena:** Podemos escribir el nombre que queramos para identificar el token usado. Por ejemplo: “SC”

Tras guardar la configuración de conexión a la red, ya tendríamos una red blockchain desplegada y operativa. Esta red, con un único nodo, se encarga simultáneamente de recibir peticiones HTTP de clientes, y de minar bloques con la cuenta que hemos creado.

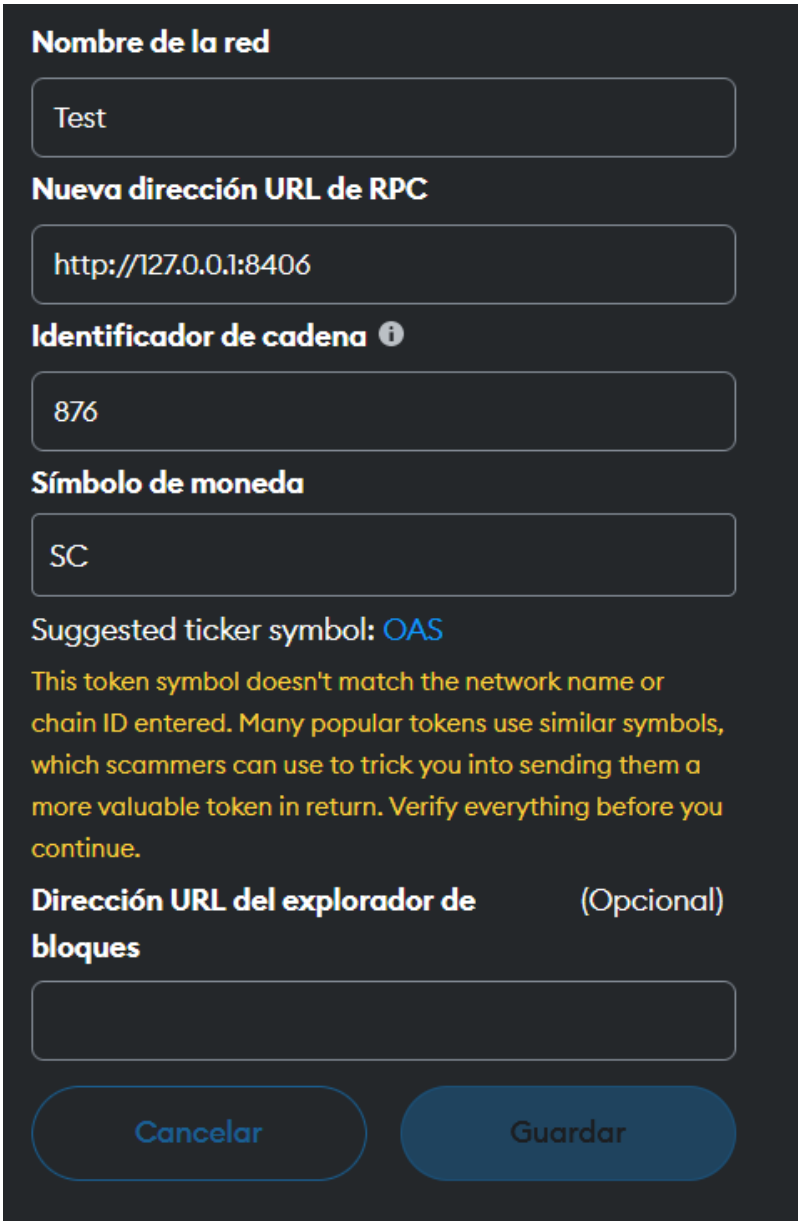


Figura 8.9: Conexión con una red blockchain desde MetaMask

8.2.4. Despligue de una red privada multi-nodo

En el despliegue visto anteriormente, se ha explicado cómo desplegar un único nodo, que hace las funciones de minado y de servidor HTTP. Sin embargo, tener un solo nodo, puede dar lugar a múltiples problemas como pueden ser de seguridad, disponibilidad y fiabilidad. Por ejemplo, si sucede un error por el cual el nodo deja de funcionar, la blockchain no sería accesible, además de que es posible que haya una pérdida de datos. Por otro lado, al tener que ejecutar las operaciones de minado y ofrecer la API HTTP, es posible que un atacante tome el control de la cuenta personal de administración. Por estos motivos, en esta sección, se aborda un modelo con tres nodos, pero que es fácilmente escalable. Son los siguientes:

- **Nodo administrador:** Será el encargado de desplegar los contratos, así como de validar los bloques mediante el mecanismo PoA.
- **Nodo de escucha HTTP:** Recibirá peticiones HTTP de clientes que no deseen desplegar su propio nodo, permitiendo la interacción con la blockchain. De esta manera, los clientes podrán acceder a la Dapp y usar su billetera en MetaMask, sin necesidad de configuraciones de nodos.
- **Nodo independiente:** Este nodo pretende servir de ejemplo a cómo debería usarse la blockchain en una situación perfecta. La idea que subyace en crear un nodo propio, es por la que se fundamenta la tecnología

blockchain. Teniendo un nodo propio, puedes ejecutar tus propias transacciones en un entorno seguro, sin necesidad de confiar en terceros. Así, un cliente que desea usar la Dapp desarrollada, conecta su billetera de MetaMask a su propio nodo, el cual se conecta directamente con la red.

Antes de comenzar a desplegar los nodos, hay que considerar ciertos aspectos:

- Dado que se emplean varias máquinas, hay que tener especial cuidado con la configuración del Firewall y del router. Si no se tienen en consideración, es muy probable que la comunicación entre nodos no funcione.
- Hay muchas maneras de crear una red blockchain. Todas ellas pueden ser correctas, y diferir en pequeños matices que hay que tener en cuenta para que sea segura.
- En esta sección se aborda una configuración bastante más realista. Sin embargo, se ha desarrollado en un entorno aislado de pruebas para evitar ataques, ya que son muy comunes.
- A pesar de que se pueden desplegar varios nodos en una máquina, cada nodo se ha desplegado en una máquina diferente. Esto ha sido necesario por varias razones: evitar posibles interferencias entre nodos, por motivos de seguridad de la red y para acercarse más a una red real.

Inicialización de los nodos

El proceso descrito a continuación es prácticamente común para los tres nodos que vamos a desplegar. Por tanto, habrá que realizarlo en las tres máquinas que se empleen.

Al igual que en el despliegue visto en la Subsección 8.2.3, lo primero es empezar por la configuración del génesis. Dado que no hay motivos para cambiar esta configuración inicial, se ha vuelto a emplear el génesis mostrado en la Figura 8.6. Como también vamos a necesitar una cuenta que despliegue los contratos y valide las transacciones, los pasos serán los mismos. Por tanto, se ejecuta el comando `geth account new --datadir Chain`(este paso solo es necesario en el nodo administrador). A continuación se inicializa el nodo con `geth --datadir Chain init genesis.json`.

A partir de aquí, el proceso cambia. El primer paso que tendremos que realizar es obtener el enode de nuestro nodo. Este término se utiliza comúnmente en el contexto de Ethereum para referirse a la identificación única de un nodo en la red. Este enode es sensible a la dirección IP y al puerto en el que está escuchando el nodo, por lo que puede cambiar si la dirección IP o el puerto del nodo cambian. Hay multitud de maneras de obtener el enode, pero una muy sencilla consiste en interactuar con la API admin que nos ofrece geth. Para ello debemos ejecutar el nodo y acceder a la consola:

`geth --datadir Chain --nodiscover console` A continuación, ejecutamos el comando de la API que permite mostrar la información del enode: `admin.nodeInfo.enode`. Será necesario guardar esta información para el futuro. Finalmente, ya podemos salir de la consola con `exit`. En la Figura 8.10 se muestra el proceso.

Figura 8.10: Obtención del enode

Definición de los nodos estáticos

A pesar de que hay otras maneras para conectar los nodos, como puede ser el uso de un bootnode, se ha preferido usar el método de nodos estáticos. De esta manera, evitamos tener que configurar el firewall para que nos defienda de la conexión de nodos no deseados. Además, en caso de cometer algún error, seguiríamos protegidos.

Los nodos estáticos son nodos configurados para establecer conexiones persistentes. A diferencia de los nodos normales, que pueden conectar y desconectar dinámicamente con otros nodos según la topología de la red, los nodos estáticos mantienen conexiones permanentes con un conjunto predefinido de otros nodos. En redes privadas,

donde los participantes están predeterminados y conocidos, los nodos estáticos pueden usarse para asegurar la conectividad entre los participantes de manera constante, reduciendo la exposición a posibles ataques de red o al riesgo de conexiones no confiables.

Para conectar los nodos de esta manera es necesaria la creación del archivo `config.toml`. Este archivo tiene multitud de parámetros para configurar, pero en nuestro caso, solo necesitamos definir la lista de nodos estáticos que deseamos. Llegados a este punto, deberíamos tener ya los tres enodes, correspondientes a cada nodo. Definimos el archivo como se muestra en Figura 8.11, usando los enodes que hemos obtenido. Como se puede observar, hay que hacer unas pequeñas modificaciones. Al final del enode, están definidas las IPs de los mismos. A cada enode, hay que asignarle la IP que contienen en la red privada. Desde Windows, el comando `ipconfig` es útil para dicho fin. El puerto no hace falta modificarlo, puesto que es por defecto el 30303. Finalmente, se elimina el discport, que no es necesario.

Este archivo que hemos creado, deberemos añadirlo a todas las máquinas, guardándolo en el mismo directorio que la carpeta Chain que creamos anteriormente.

Figura 8.11: Estructura del archivo de configuración de nodos

Despliegue del nodo administrador

Una vez inicializados los nodos y creado el archivo de configuración, ya podemos desplegar los nodos. En el caso del nodo administrador, tendremos que desplegar antes, los contratos. Debido a que truffle utiliza el procedimiento HTTP, deberemos activarlo temporalmente. El comando usado es el siguiente:

```
geth --networkid 876--datadir Chain --nodiscover --unlock "0x2eff9c17682feDdD7c7d4F1c9Ab03c9f8818C6De" --mine --miner
    .etherbase="0x2eff9c17682feDdD7c7d4F1c9Ab03c9f8818C6De" --config config.toml --http --http.api web3,eth,personal
    ,net --http.corsdomain "*" --http.port 8406--allow-insecure-unlock
```

La mayoría de flags ya los expliqué anteriormente, así que un breve resumen:

- **–nodiscover:** Este flag evita conexiones con nodos no deseados. Sin embargo, no inhabilita la conexión con los nodos estáticos.
- **Desbloqueo de cuenta y minado:** Es necesario desbloquear la cuenta y minar, para que quede registrado el contrato en la blockchain. Tanto la cuenta como el etherbase deberán ser actualizadas con la cuenta creada anteriormente.
- **Interfaz HTTP:** Es necesario permitir la conexión mediante HTTP, para que truffle funcione correctamente.
- **–config:** Este flag permite transmitir al nodo los parámetros de configuración que hemos definido en el archivo `config.toml`.

Ahora ya es posible desplegar los contratos usando truffle. El procedimiento es el mismo que se usó en la Sección 8.2.3. Por tanto, usando el archivo `truffle-config` de la Figura 8.7, ejecutamos los comandos `truffle compile` y `truffle migrate`. Es importante actualizar el archivo de configuración con la cuenta que hemos creado.

Aunque podríamos dejar el nodo desplegado, es aconsejable interrumpirlo para lanzarlo de nuevo con una ejecución más segura:

```
geth --networkid 876--datadir Pruebas2 --nodiscover --unlock "0x2eff9c17682feDdD7c7d4F1c9Ab03c9f8818C6De" --mine --
    miner.etherbase="0x2eff9c17682feDdD7c7d4F1c9Ab03c9f8818C6De" --config config.toml
```

Se puede apreciar que se ha eliminado el acceso a la interfaz HTTP, lo cual nos permite también, eliminar el flag `--allow-insecure-unlock`.

Por tanto, hemos desplegado un nodo que ejerce las tareas típicas, pero a mayores, valida los bloques mediante PoA, en un entorno seguro.

Despliegue del nodo de escucha HTTP

Este nodo tiene como tarea dar acceso a la red a aquellos clientes que no desean albergar su propio nodo. Por tanto, será necesario habilitar la interfaz HTTP, así como dar acceso a todo aquel que quiera conectarse. El siguiente comando es el empleado:

```
geth --networkid 876--datadir Chain --nodiscover --http --http.api web3,eth --http.corsdomain "*" --http.port 8406--
    http.addr "0.0.0.0" --http.vhosts "*" --config config.toml
```

Como se ve, hay que activar la mayoría de flags relacionados con HTTP. Por un lado, se permite el uso de las APIs web3 y eth. Por otro, se permite el acceso de cualquier dominio el CORS y las solicitudes de creación de host virtuales desde cualquier dominio. Se ha intentado restringir el acceso de estas características solo a una subred privada, pero geth no permite estas configuraciones. Por tanto, para una configuración segura, sería necesario configurar adecuadamente el firewall para restringir el acceso adecuadamente.

Finalmente, es necesario configurar MetaMask. Aquellos usuarios que deseen usar el nodo expuesto, deberán crear una nueva red manualmente y añadir la URL de RPC de la máquina(en este caso la IP y el puerto expuestos), tal y como se muestra en la Figura 8.9.

Despliegue de un nodo independiente

El despliegue de este nodo es sencillo, ya que su única tarea es conectar la máquina en la que se ejecuta, con la red blockchain. Para ello, nos servimos de un comando bastante sencillo:

```
geth --networkid 876--datadir Chain --nodiscover --http --http.api web3,eth --http.corsdomain "*" --http.port 8406--
    config config.toml
```

Desafortunadamente, si se quiere conectar MetaMask a un nodo, es necesario hacerlo vía HTTP, pues es una extensión de navegador. Por tanto, es necesario habilitar los flags HTTP que se muestran. Sin embargo, las conexiones externas no son fructíferas. Aun así, es recomendable activar correctamente el firewall para evitar posibles brechas en los protocolos.

De nuevo, es necesario configurar MetaMask para poder hacer uso del nodo. La Figura 8.9 sirve de referencia, otra vez.

8.3. Frontend React/Metamask

En esta sección, se pretende explicar a grandes rasgos, el funcionamiento de la aplicación desarrollada en React. El desarrollo de esta aplicación ha sido fundamentado sobre el modelo de componentes presentado en Sección 7.3, y como resultado, se ha obtenido la estructura de proyecto mostrada en Figura 8.12.

8.3.1. Componentes principales

Estos componentes son los que se encuentran en más alto nivel de la aplicación. Index es el componente principal, cuyo única tarea es crear el ReactDOM y renderizar el componente AppRouter.

El componente AppRouter es el que utiliza la biblioteca React Router vista en Sección 3.5. Su función principal es gestionar el enrutamiento de toda la aplicación, utilizando el resto de componentes, y creando las rutas para cada uno de ellos. En la Figura 8.13 se puede observar el código. Mediante los componentes “Route” es posible definir la estructura. Para ello, se añade la ruta, el componente que va a renderizarse, y si son necesarias, las funciones loader y action. Estas funciones sirven para gestionar la información que tendrá el componente seleccionado al cargarse por primera vez, o al realizar el usuario alguna acción.

El componente Login, es el que se encarga de la conexión con la API de MetaMask. De esta manera, la aplicación tiene acceso a las cuentas del usuario, así como de registrar cualquier cambio en estas que se produzca.

Además, AppRoot es el componente que se encarga del menú lateral. Permite al usuario acceder a todas las funcionalidades de la aplicación, así como indicar en qué sección se encuentra. Por otro lado, también asigna el espacio dónde se cargarán el resto de componentes.

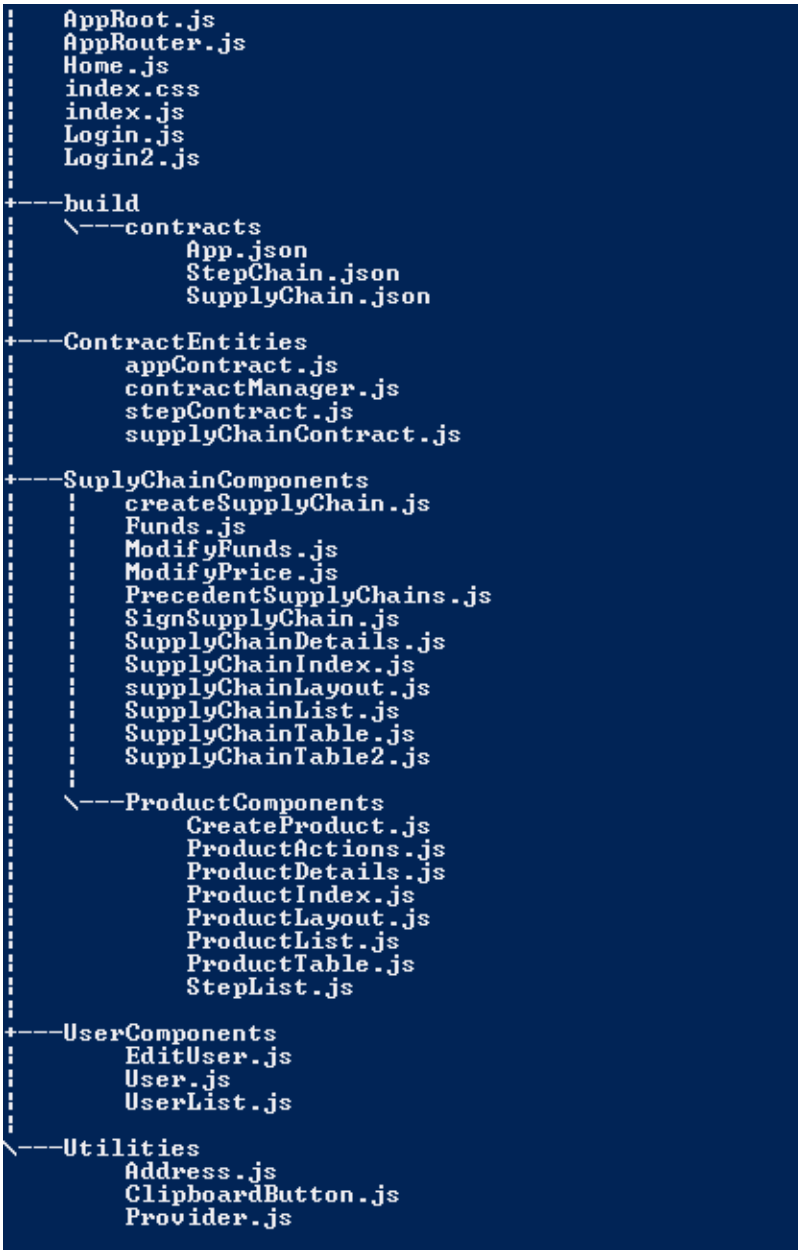
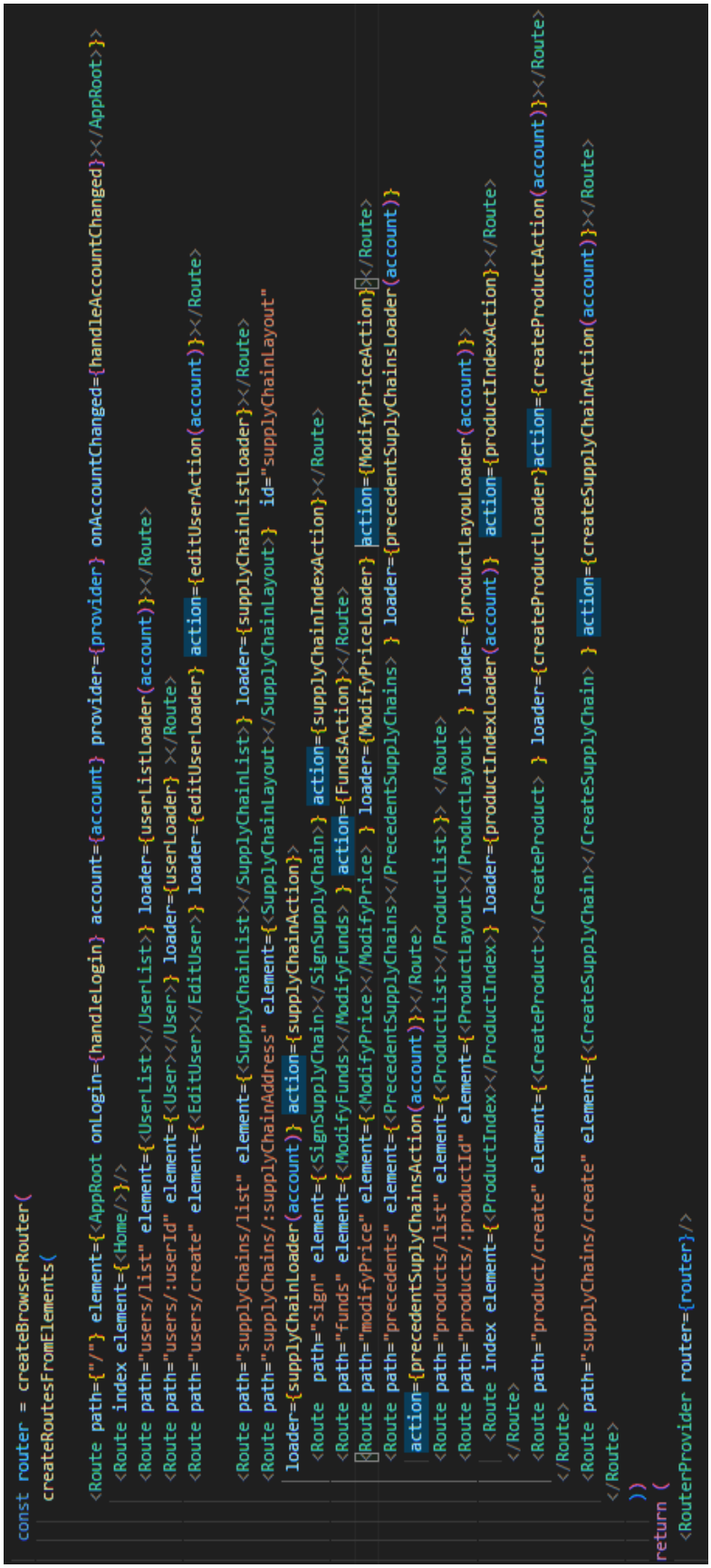


Figura 8.12: Estructura de archivos de la aplicación React



```
const router = createBrowserRouter(
  createRoutesFromElements(
    <Route path={"/"} element={<AppRoot onLogin={handleLogin} account={account} provider={provider} onAccountChanged={handleAccountChanged}></AppRoot>}>
      <Route index element={<Home/>}/>
      <Route path="users/list" element={<UserList></UserList>} loader={userListLoader(account)}></Route>
      <Route path="users/:userId" element={<User></User>} loader={userLoader} ></Route>
      <Route path="users/create" element={<EditUser></EditUser>} loader={editUserLoader} action={editUserAction(account)}></Route>

      <Route path="supplyChains/list" element={<SupplyChainList></SupplyChainList>} loader={supplyChainListLoader}></Route>
      <Route path="supplyChains/:supplyChainAddress" element={<SupplyChainLayout></SupplyChainLayout>}  id="supplyChainLayout"
       loader={supplyChainLoader(account)} action={supplyChainAction}>
        <Route  path="sign" element={<SignSupplyChain></SignSupplyChain>} action={supplyChainIndexAction}></Route>
        <Route path="funds" element={<ModifyFunds></ModifyFunds> } action={FundsAction}></Route>
        <Route path="modifyPrice" element={<ModifyPrice></ModifyPrice> } loader={ModifyPriceLoader} action={ModifyPriceAction}></Route>
        <Route path="precedents" element={<PrecedentSupplyChains></PrecedentSupplyChains> } loader={precedentSuplyChainsLoader(account)}
         action={precedentSuplyChainsAction(account)}></Route>
        <Route path="products/list" element={<ProductList></ProductList>} ></Route>
        <Route path="products/:productId" element={<ProductLayout></ProductLayout> } loader={productLayouLoader(account)}>
          <Route index element={<ProductIndex></ProductIndex>} loader={productIndexLoader(account)}  action={productIndexAction}></Route>
        </Route>
        <Route path="product/create" element={<CreateProduct></CreateProduct> } loader={createProductLoader}action={createProductAction(account)}></Route>
      </Route>
      <Route path="supplyChains/create" element={<CreateSupplyChain></CreateSupplyChain> } action={createSupplyChainAction(account)}></Route>
    </Route>
  )
)
return (
  <RouterProvider router={router}/>
```

Figura 8.13: Enrutamiento de la aplicación React

8.3.2. Entidades de los contratos

En esta carpeta se encuentran los componentes que conectan con los contratos desplegados en la red Ethereum. La idea es aplicar el patrón fachada, de manera que los componentes de la aplicación que necesiten recibir o enviar datos de los contratos, simplemente llamen a una función que gestiona la interacción. Para ello se ha creado un componente por contrato: AppContract, stepContract y supplyChainContract. En la Figura 8.14 tenemos dos ejemplos de la cómo se realiza la comunicación con el contrato.

Por otro lado, el componente ContractManager aplica del patrón Singleton, para segurar que sólo haya una instancia de los componentes anteriores, así como del proveedor Web3. De esta manera, se previenen errores y se ahorra uso de memoria.

```
async getUser(id){
    let user= await this.contract.methods.userList(id).call();
    user.account=id
     return user;
    ;
}

async createUser(account,data){
    this.contract.methods.newUser(data.name,data.location,data.postCode,data.phoneNumber,data.email).send({
        from: account,
        gas: String(await this.contract.methods.newUser(data.name,data.location,data.postCode,data.phoneNumber,data.email).estimateGas({from:account})*2)
    })
}
```

Figura 8.14: Interacción con los contratos

8.3.3. Componentes de usuario

Como la aplicación ha de gestionar los usuarios que se quieran registrar, es necesario crear los componentes que se encargan de esta funcionalidad. Para ello, los componentes creados son: EditUser, User y UserList. User se encarga de crear la vista que muestra toda la información relevante de un usuario en cuestión. EditUser permite tanto la creación, como la modificación de los usuarios ya registrados. Por último, UserList define la vista que muestra todos los usuarios que están registrados en la aplicación. Estos tres componentes, al tener poca funcionalidad y basarse principalmente en mostrar información, usan otros componentes de Material UI, para darle un aspecto solvente.

8.3.4. Componentes de cadena de suministro

La mayor parte de la funcionalidad implementada es referente a las cadenas de suministro. Se pueden distinguir tres componentes principales:

- **CreateSupplyChain:** contiene la funcionalidad que permite crear nuevas cadenas de suministro. Para ello, se define el formulario que solicita los datos pertinentes al usuario. Además, es necesario añadir la validación de los campos. Por último, se define la función "action" que gestiona la llamada a la entidad del contrato SupplyChain. De esta manera, se consigue que MetaMask solicite aprobar la transacción con la información del formulario.

- **SupplyChainList:** envuelve el componente SupplyChainTable para darle el formato adecuado, y define la llamada, de nuevo a través de la función "action", a la entidad del contrato SupplyChain. Por su parte, SupplyChainTable es un componente que muestra una tabla con la información de las cadenas de suministro, previamente suministrada.

- **SupplyChainLayout:** es un componente soporte, que facilita la gestión del resto de componentes de la cadena de suministros. Lo consigue definiendo la función "loader" que podemos ver en Figura 8.15. En ella, se recopila toda la información referente a la cadena de suministro, accediendo al contrato desplegado en la blockchain. Además, se comprueba qué rol tiene el usuario, para darle acceso a las funcionalidades correspondientes. Respecto a la vista que presenta, se muestra una alerta que indica si la cadena ha sido firmada y el componente SupplyChainDetails, que contiene toda la información relevante de la cadena. Finalmente, define un espacio libre, donde se mostrarán el resto de componentes, según se navegue por la aplicación.

El resto de componentes son secundarios, y cada uno implementa una funcionalidad distinta. Son los siguientes:

- **PrecedentSupplyChains:** Utiliza el componente base SupplyChainTable para mostrar información de las cadenas de suministro que son precedentes de la actual. Es necesario solicitar al contrato la lista de cadenas precedentes, desde la función "action".
- **ModifyPrice:** Gestiona la funcionalidad que permite tanto a comprador como a vendedor, solicitar un cambio de precio.
- **ModifyFunds:** Permite a los usuarios con algún rol específico en la cadena, añadir o retirar fondos de la misma. Para ello, utiliza el componente Funds, personalizándolo levemente.
- **SignSupplyChain:** Este componente, permite al vendedor, transportista y certificador firmar la cadena. Esto quiere decir, que el firmante, ratifica los datos con los que se ha creado la cadena. Una vez todos han firmado, la cadena de suministros se inicia.

```
export const loader = account=> async ({ request, params }) => {
  contractManagerInstance.setSuplyChainContract(params.supplyChainAddress);
  const supplyChainInfo =  await contractManagerInstance
    .getSuplyChainContract()
    .getSupplyChainInformation();
  const supplyChainSignatures = await contractManagerInstance
    .getSuplyChainContract()
    .getSupplyChainSignatures();
    let precedentsInfo = await  contractManagerInstance
    .getSuplyChainContract().getPrecedentsInfo()
    let role=[];
    if (!account) role.push("viewer")
    else {if (
        //provider.role === "seller" &&
        supplyChainInfo.seller &&
        account.toLowerCase() === supplyChainInfo.seller.toLowerCase()
      ) {
        role.push("seller");
      }  if (
        //provider.role === "carrier" &&
        supplyChainInfo.carrier &&
        account.toLowerCase() === supplyChainInfo.carrier.toLowerCase()
      ) {
        role.push("carrier");
      }  if (
        //provider.role === "buyer" &&
        supplyChainInfo.buyer &&
        account.toLowerCase() ===supplyChainInfo.buyer.toLowerCase()
      ) {
        role.push("buyer");
      }  if (
        //provider.role === "certifier" &&
        supplyChainInfo.certifier &&
        account.toLowerCase() === supplyChainInfo.certifier.toLowerCase()
      ) {
        role.push("certifier");
      } if(role.length===0) {
        role.push("viewer");
      }
    }
    const supplyChainFunds =await  contractManagerInstance.getSuplyChainContract().getFunds(account);

      let productListInfo= await  contractManagerInstance
      .getSuplyChainContract().getProductListInfo()
   // await Promise.all(supplyChainFunds,supplyChainInfo,supplyChainSignatures)
  return { supplyChainInfo, supplyChainSignatures ,role,supplyChainFunds,productListInfo,precedentsInfo };
};
```

Figura 8.15: Loader de SupplyChainLayout

8.3.5. Componentes de producto

Una vez la cadena de suministro ha sido iniciada, es posible interactuar con la funcionalidad de los productos. Todos los componentes de productos, son renderizados por el componente SupplyChainLayout y se muestran en el espacio designado. Al igual que con los componentes de las cadenas de suministro, la estructura de los de productos es la misma. Hay tres componentes principales: ProductLayout que gestiona los componentes secundarios, CreateProduct que permite crear productos, y productList que muestra todos los productos de la cadena de suministro sirviéndose de la tabla creada en ProductTable.

Los componentes secundarios son los siguientes:

- **ProductActions:** Muestra una serie de botones que da acceso al usuario a las acciones que pueden realizarse sobre el producto. Los botones que se muestran, dependen directamente del rol que tiene el usuario en la cadena de suministro.
- **StepList:** Despliega una lista con la información de todos los Steps que se han realizado para el producto seleccionado.

8.3.6. Componentes de utilidad y artefactos

En la carpeta build, se almacenan los artefactos generados por el compilador de Solidity. Estos archivos JSON contienen metadatos y detalles importantes sobre los contratos desplegados. Específicamente, estos archivos contienen la siguiente información:

- **ABI (Application Binary Interface):** La ABI es una descripción del contrato que permite que la aplicación React interactúe con el contrato en la blockchain. Contiene una lista de todas las funciones y eventos que el contrato puede utilizar, junto con sus parámetros y tipos de datos.
- **Bytecode:** Este es el código binario que es ejecutado por la máquina virtual de Ethereum (EVM). Es el resultado de compilar el código fuente Solidity. Este bytecode es lo que se despliega en la blockchain cuando se implementa el contrato.
- **Metadata:** Información adicional sobre el contrato, como el nombre del contrato, las bibliotecas utilizadas, versiones del compilador, y otra información relevante.
- **Redes (Networks):** Este apartado puede incluir información sobre las implementaciones del contrato en distintas redes (mainnet, testnets, etc.), incluyendo direcciones del contrato desplegado y block hashes de las transacciones de despliegue.

Por otro lado, en la carpeta Utilities, están aquellos componentes que son ampliamente en toda la aplicación, resolviendo funcionalidades típicas. Por ejemplo, en Dapps es muy común tener que mostrar direcciones de cuenta o de contratos. Estas direcciones son relativamente largas, y a veces es necesario contraerlas para simplificar la vista. En este caso, el componente Address resuelve este problema, acortando las direcciones y habilitando links y cambio de colores en caso de que así lo especifiques.

8.3.7. Despliegue de la aplicación

Desplegar una aplicación React, es muy sencillo. Los pasos a realizar son los siguientes:

- Abrir una terminal de comandos y posicionarse en la carpeta `client`.
- Ejecutar el siguiente comando para instalar las dependencias:
  - `npm install`

  Este comando instalará todas las dependencias necesarias para la aplicación.
- Para un despliegue rápido, en un entorno de pruebas, se puede usar el siguiente comando:

- `npm start`

Si, por el contrario, queremos una versión optimizada, deberemos usar los siguientes comandos:

- `npm run build`
- `npm install -g serve`
- `serve -s build`

Finalmente, en el puerto 3000 de la máquina en la que se ha desplegado, se puede acceder a la aplicación. Si se quiere acceder desde otra máquina, solo sería necesario asegurarse que la conexión no se ve interrumpida por firewalls, y acceder con la dirección IP y el puero de la máquina en que se ha desplegado.

Capítulo 9

Conclusiones

9.1. Conclusiones

En este trabajo, se ha desarrollado una aplicación descentralizada (Dapp) utilizando la tecnología blockchain, específicamente Ethereum. A través del proceso de investigación y desarrollo, se han logrado los siguientes objetivos y se han obtenido las siguientes conclusiones:

- **Comprensión de la Tecnología Blockchain:** Se ha adquirido una comprensión profunda de los fundamentos de la tecnología blockchain, incluyendo la descentralización, las cadenas de bloques y los mecanismos de consenso. Esta comprensión es crucial para desarrollar aplicaciones seguras y eficientes en el ámbito de la cadena de suministro.
- **Desarrollo de Smart Contracts:** Se han diseñado y desarrollado varios smart contracts utilizando Solidity. Estos contratos inteligentes son fundamentales para la automatización de procesos dentro de la Dapp, asegurando la transparencia y la inmutabilidad de las transacciones.
- **Integración de Herramientas y Tecnologías:** Se ha integrado una variedad de herramientas tecnológicas como Geth, Web3.js, ReactJs y MetaMask. Cada una de estas herramientas ha jugado un papel importante en la creación de una aplicación completa y funcional.
- **Aplicación en la Cadena de Suministro:** La implementación de la Dapp en un caso de uso de cadena de suministro ha demostrado su potencial para mejorar la trazabilidad y la transparencia en la gestión de productos. Esta aplicación práctica valida la viabilidad de utilizar blockchain en entornos industriales.

En resumen, este proyecto ha demostrado la capacidad de la tecnología blockchain para revolucionar la gestión de la cadena de suministro, ofreciendo una solución innovadora que puede ser adaptada a diversas industrias.

9.2. Trabajo futuro

A pesar de los logros alcanzados, existen varias áreas que se pueden mejorar y expandir en futuros trabajos:

- **Optimización de Costos de Gas:** Los costos de transacción en Ethereum pueden ser elevados. Investigaciones futuras podrían enfocarse en la optimización del uso del gas, así como en la exploración de otras plataformas blockchain que ofrezcan costos más bajos, como Binance Smart Chain o Polygon.
- **Interfaz de Usuario :** Aunque se ha desarrollado un frontend funcional, la experiencia del usuario siempre puede mejorar. Futuras iteraciones podrían centrarse en una interfaz más intuitiva y amigable, incorporando feedback de los usuarios para realizar ajustes que faciliten la interacción con la Dapp.
- **Integración de IoT:** La integración de dispositivos IoT para la recopilación automática de datos en la cadena de suministro podría proporcionar un nivel adicional de automatización y precisión. Esto permitiría una monitorización en tiempo real de los productos y condiciones.

- **Adopción de Estándares de Industria:** Colaborar con organismos de estandarización para alinear la Dapp con los estándares industriales podría facilitar su adopción en un entorno más amplio. Esto incluye la interoperabilidad con otros sistemas y plataformas utilizadas en la cadena de suministro.
- **Seguridad y Auditoría:** Aunque la seguridad es una ventaja inherente de la blockchain, siempre es posible mejorar. La realización de auditorías de seguridad exhaustivas y la implementación de mejores prácticas de desarrollo seguro pueden proteger aún más los activos y datos de los usuarios.
- **Pruebas en Entornos Reales:** Ampliar las pruebas a entornos de producción reales en diversas industrias proporcionará datos valiosos sobre el rendimiento y la eficacia de la Dapp. Esto permitirá realizar ajustes finos y mejorar su aplicabilidad y robustez.

En conclusión, este trabajo ha sentado una base sólida para el uso de la tecnología blockchain en la cadena de suministro. Sin embargo, existe un vasto potencial para futuras mejoras y aplicaciones que pueden expandir significativamente el impacto y la eficiencia de estas tecnologías en el mundo real.

Apéndice A

Manual de despliegue

El manual que se expone a continuación, pretende guiar a aquella persona que quiera desplegar la aplicación presentada en el Proyecto de la forma más sencilla posible. Para ello será necesario, por un lado, desplegar una red Ethereum privada, desplegar los contratos en dicha red, conectar MetaMask a la red, y desplegar la aplicación React.

A.1. Dependencias

Antes de comenzar el despliegue de la aplicación, es necesario contar con las siguientes herramientas:

- **Go Ethereum**: La versión empleada ha sido Saradril (v1.11.1). Se puede descargar desde el sitio web oficial de Go Ethereum: `https://geth.ethereum.org/downloads`.
- **NodeJs**: La versión empleada ha sido 18.14.1 de NodeJs. Se puede descargar desde el sitio web oficial de NodeJs: `https://nodejs.org/en/download/`.
- **Truffle**: La versión empleada ha sido v5.7.7. Se puede instalar usando npm con el siguiente comando: `npm install -g truffle`. Para más información consultar `https://trufflesuite.com/docs/truffle/how-to/install/`.
- **Metamask**: La extensión de Metamask debe estar instalada en el navegador web. Se puede descargar desde el sitio web oficial de Metamask: `https://metamask.io/download/`.

A.2. Despliegue de una red Ethereum

Hay una infinidad de formas de desplegar una red Ethereum que dé soporte a nuestras aplicaciones descentralizadas. En este apartado, se explicará cómo desplegar un único nodo privado, basado en Proof of Authority. De esta manera, el proceso será más sencillo, pero perfectamente válido para probar la aplicación.

Los pasos a seguir son los siguientes:

- Abrir una terminal de comandos y posicionarse en la carpeta `geth`.
- Crear una cuenta que será la que firme las transacciones. Para ello, ejecutar el siguiente comando en la terminal:
  - `geth account new --datadir Chain`

  Este comando pedirá una contraseña y creará una cuenta en `./Chain/keystore`, la cual se utilizará para firmar las transacciones.
- Modificar el archivo `genesis.json`. En la sección `extraData`, añadir la dirección de la cuenta creada en el paso anterior, sin el prefijo `"0x"`. También se puede añadir esta cuenta en la sección `alloc` para que tenga un balance inicial. En la Figura A.1 se puede ver un ejemplo.

- Inicializar la red usando el siguiente comando en la terminal:
  - `geth --datadir Chain init genesis.json`

  Este comando creará la carpeta `./Chain/geth` con la información necesaria para la ejecución de la nueva red.

- Para arrancar el nodo creado, ejecutar el siguiente comando:

```
geth --networkid 876--datadir Chain --nodiscover --http --http.api web3,eth,personal,net --http.corsdomain "*"
    --http.port 8406--unlock "address" --mine --allow-insecure-unlock --miner.etherbase="address"
```

Habrá de sustituirse "address" por la dirección de la cuenta que hemos creado anteriormente. Una vez ejecutado el comando, se nos pedirá la contraseña de la cuenta, para así poder desbloquearla.

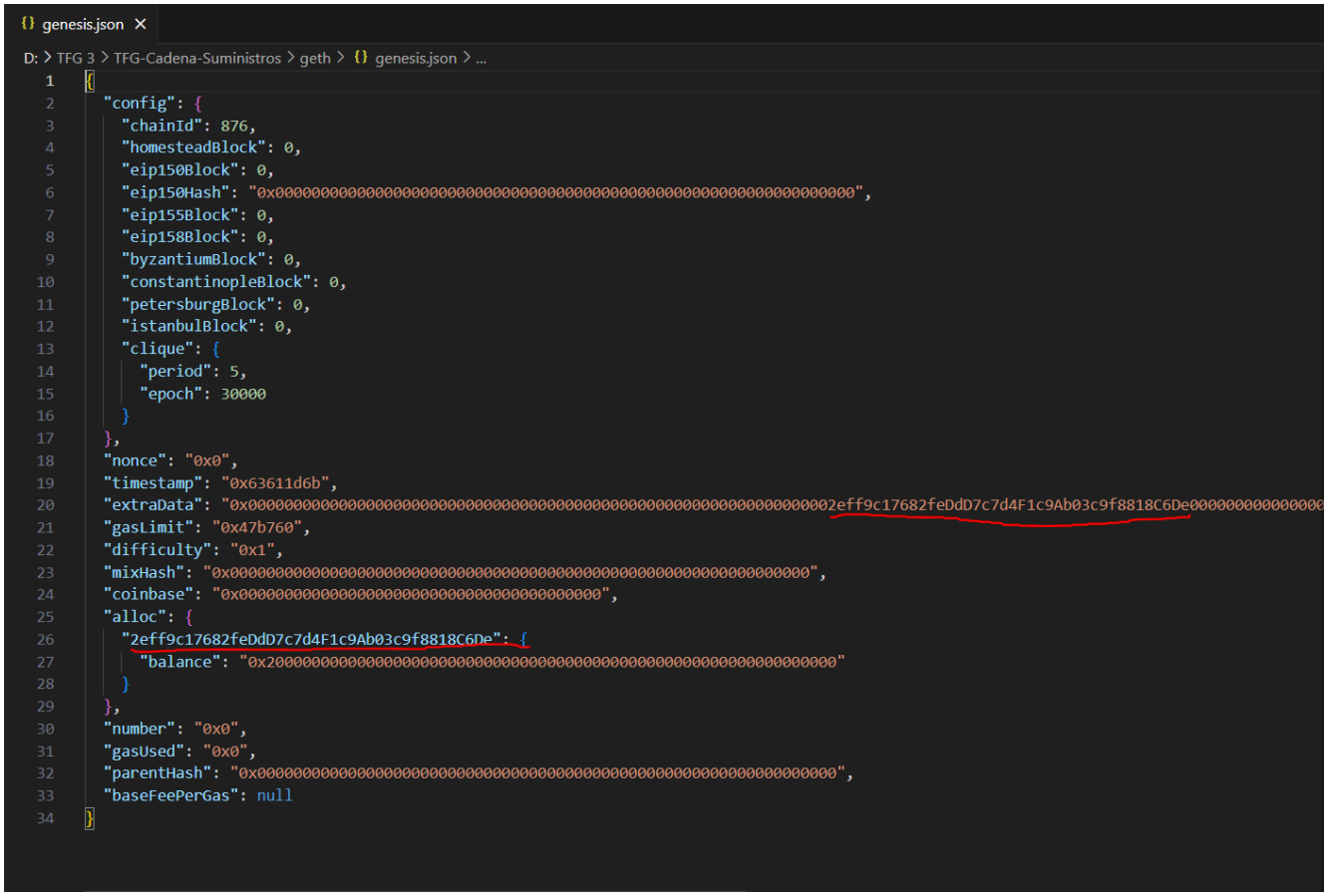


Figura A.1: Modificación del archivo genesis

A.3. Despliegue de contratos

Una vez activa la red blockchain, es necesario desplegar los contratos en la misma. Para ello, hay que seguir los siguientes pasos:

- Abrir una terminal de comandos y posicionarse en la carpeta `truffle`.
- Como se muestra en la Figura A.2, hay que modificar el archivo `truffle-config`.En este archivo, hay que modificar el atributo `from` con la dirección de la cuenta creada en el apartado "Despliegue de red Ethereum". Esto nos permitirá desplegar los contratos usando dicha cuenta.
- Ejecutar los comandos `truffle compile` y `truffle migrate`.

truffle-config.js

D: > TFG 3 > TFG-Cadena-Suministros > truffle > truffle-config.js > <unknown>

```
module.exports = {
  contracts_build_directory: "../client/src/build/contracts",
    // See <http://truffleframtruework.com/docs/advanced/configuration>
    // for more about customizing your Truffle configuration!
    networks: {
      development: {
        host: "127.0.0.1",
        port: 8406,
        network_id: "*", // Match any network id
        production:true,
        from: "0x2eff9c17682feDdD7c7d4F1c9Ab03c9f8818C6De"

      },
      develop: {
        port: 8545
      }
    },
    compilers: {
        solc: {
          version: "0.8.8",
          settings: {
            optimizer: {
              enabled: true,
              runs: 10
            }
          }
        }
    },

};
```

Figura A.2: Modificación del archivo truffle-config

A.4. Despliegue de la aplicación React

Para desplegar la aplicación, se deben seguir los siguientes pasos:

- Abrir una terminal de comandos y posicionarse en la carpeta `client`.
- Ejecutar el siguiente comando para instalar las dependencias:
  - `npm install`

  Este comando instalará todas las dependencias necesarias para la aplicación.
- Para un despliegue rápido, en un entorno de pruebas, se puede usar el siguiente comando:
  - `npm start`

  Si, por el contrario, queremos una versión optimizada, deberemos usar los siguientes comandos:

- `npm run build`
- `npm install -g serve`
- `serve -s build`

Finalmente, en el puerto 3000 de la máquina en la que se ha desplegado, se puede acceder a la aplicación. Si se quiere acceder desde otra máquina, solo sería necesario asegurarse que la conexión no se ve interrumpida por firewalls, y acceder con la dirección IP y el puero de la máquina en que se ha desplegado.

A.5. Configuración de MetaMask

Para configurar la extensión Metamask, se deben seguir los siguientes pasos:

- Abrir Metamask, ir a `Configuración`, `Redes` , `Agregar una red` , `Agregar una red manualmente`. Nos encontraremos con un formulario para rellenar.
- Rellenar el formulario con la siguiente información:
  - **Nombre de la red**: Podemos escribir el nombre que queramos para identificar esta red.
  - **Nueva dirección Url de RPC**: `http://127.0.0.1:8406`. De esta manera accedemos al nodo que hemos desplegado en nuestra máquina. En caso de querer conectar a un nodo externo, habría que poner la dirección IP correspondiente a dicho nodo.
  - **Identificador de cadena**: `876`. Este identificador ha de corresponderse al empleado en el flag `--networkid` usado en el despliegue de la red Ethereum.
  - **Símbolo de cadena**: Podemos escribir el nombre que queramos para identificar el token usado. Por ejemplo: "SC".
- Finalmente, guardamos la nueva red creada.

A.6. Información adicional

Una vez desplegada la red Ethereum, los contratos y la aplicación React, y configurado Metamask, podemos empezar a usar la aplicación. Sin embargo, nos encontraremos con que no tenemos fondos en las cuentas nuevas. Para añadir fondos a estas cuentas, deberemos enviarlos desde la cuenta creada en el despliegue de la red Ethereum. Para ello, podremos hacerlo desde geth, usando la consola, o podremos importar esta cuenta en Metamask. Recomiendo la segunda opción:

- Abrir Metamask y dirigirse a **Importar cuenta** / **Archivo JSON**.
- Añadir el archivo JSON correspondiente, que se encuentra en `geth/Chain/keystore`.
- Introducir la contraseña que desbloquea la cuenta.
- Enviar los fondos a la nueva cuenta, a través de la interfaz de Metamask.

Si las cuentas que se van a usar ya han sido utilizadas en otras redes, es necesario limpiar su historial. Para ello, se debe:

- Acceder a la configuración de Metamask.
- Seleccionar la opción **Avanzado**.
- Hacer clic en **Restablecer cuenta**. Este proceso debe hacerse para cada una de las cuentas que se utilizarán en la red.

Apéndice B

Manual de usuario

El presente manual tiene el objetivo de servir como apoyo para el uso de la aplicación. En él, se explica cómo interactuar tanto con la aplicación como con MetaMask.

B.1. Conectar con MetaMask

Figura B.1: Conexión con MetaMask

Antes de utilizar cualquier funcionalidad de la aplicación, es recomendable conectar la cuenta que querremos usar, la cual está almacenada en Metamask. Para ello, debemos pulsar el botón de conectar presente en la esquina superior derecha. A continuación, se abrirá una ventana emergente solicitando que se seleccionen las cuentas de nuestra billetera MetaMask que queremos conectar, como se muestra en Figura B.1. Al continuar y aceptar los términos, el botón de conexión desaparecerá y mostrará la cuenta que hemos conectado. Cabe mencionar que podemos conectar varias cuentas, pero solo una estará seleccionada. La cuenta seleccionada es la que se mostrará en la aplicación, y la que recibirá las peticiones de transacción que se realicen.

B.2. Gestionar transacciones en MetaMask

Cada vez que se quiera realizar una acción que implique modificar datos en la blockchain, se crea una transacción que MetaMask envía para realizar los cambios. Estas transacciones tienen un coste asociado, en el cual se incluye una tarifa de gas. Antes de que se ejecute cualquier transacción, Metamask nos pide la confirmación de la misma,

restando los costes de realizarla de la cuenta que está activa. En la Figura B.2 se puede ver una transacción. Una vez pulsamos el botón de confirmación, MetaMask envía la transacción. La resolución de la misma no es automática, pero una vez resuelta, MetaMask nos notifica que se ha relizado con éxito.

Hay que tener en cuenta que para poder realizar transacciones, la cuenta que se vaya a emplear debe tener fondos para pagar, al menos, la tarifa de gas.

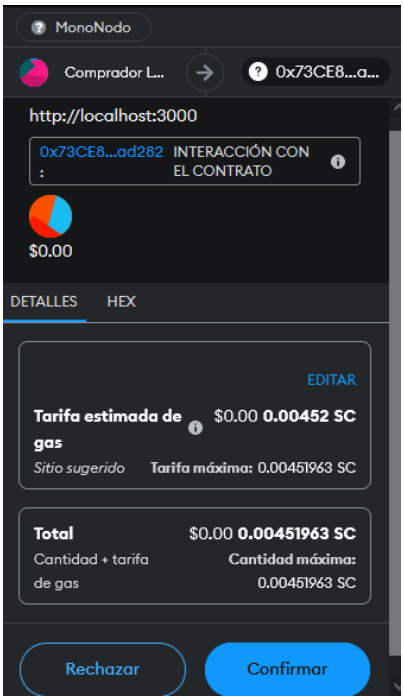


Figura B.2: Transacción de MetaMask

B.3. Registrarse como usuario

A pesar de que para usar la aplicación, no es necesario registrarse en ella, esto es útil para que el resto de usuarios puedan identificar cuál es tu cuenta. Para registrar una cuenta, hay que navegar en el menú lateral a la lista de usuarios. Si la cuenta aún no ha sido registrada, aparecerá una opción para registrarse, como se muestra en la Figura B.3. Tras rellenar el formulario emergente, se crea una transacción en MetaMask, que una vez validada, registra la nueva cuenta con la información suministrada.

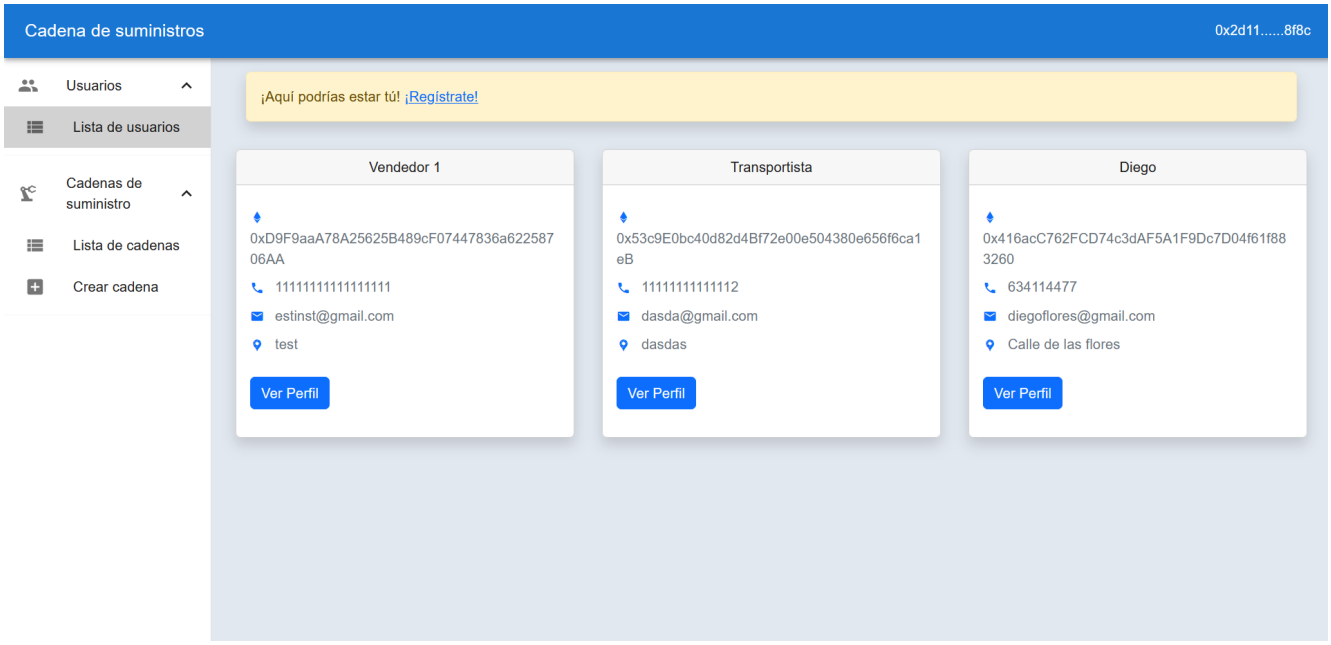


Figura B.3: Lista de usuarios con opción de registro

B.4. Crear una cadena de suministro

La acción de crear una cadena de suministro ha de ser realizada por la persona que ejerce el rol de comprador de la misma. Para ello, el comprador tiene hay que usar el menú lateral. En el apartado de cadenas de suministro, se puede encontrar el botón de crear una cadena. Al pulsarlo, se despliega un formulario como el presentado en la Figura B.4. Es importante tener claro qué cuentas serán las que empleen el resto de miembros, pues solo las cuentas que se seleccionen, tendrán acceso a la gestión de la cadena de suministros. Una vez rellenado el formulario, y pulsado el botón de creación de la cadena, se crea la transacción que ha de ser validada a través de MetaMask. La cuenta usada en el envío de la transacción, se corresponderá con el comprador de la cadena. Finalmente, cuando la transacción se resuelve, se puede dar por creada correctamente la cadena.

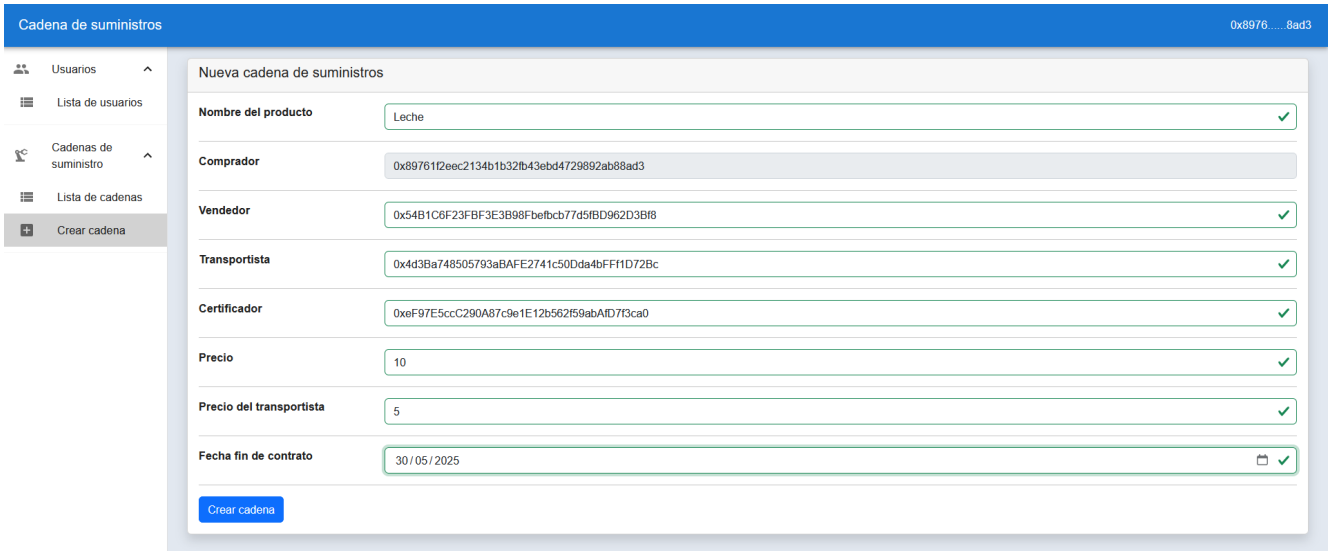


Figura B.4: Creación de una cadena de suministro

B.5. Firmar una cadena de suministro

Para poder empezar a usar toda la funcionalidad de la cadena de suministros, es necesario que todas las partes implicadas la firmen. Como se puede observar en la Figura B.5, en el menú lateral se ha de seleccionar

el botón de firmar cadena. Esto despliega una lista con información de quiénes han firmado, y un botón para firmar. Una vez se pulsa, se envía una transacción a MetaMask, que tras aceptarla, valida nuestra aprobación de la cadena de suministros. Cuando todos los miembros han dado su aprobación, se pueden empezar a usar el resto de funcionalidades.

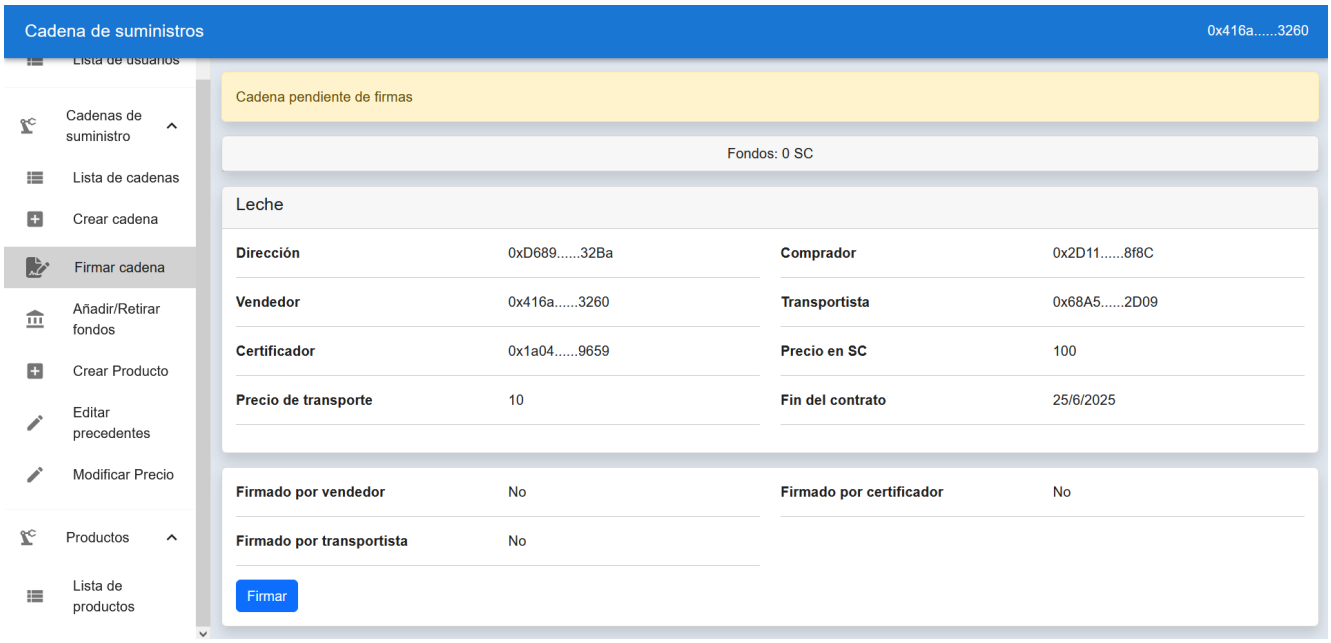


Figura B.5: Firmar una cadena de suministro

B.6. Añadir y retirar fondos

Accediendo en el menú lateral, los miembros de la cadena tienen acceso a interactuar con los fondos que tienen en ella. Si es el vendedor o el transportista, únicamente puede retirar los fondos que ha ganado a través de la cadena de suministro. Para ello, ha de clicar el botón de retirar y aceptar la transacción que se crea en MetaMask. Si es el comprador, además, puede añadir fondos de los que se descontarán los costes de los productos comprados. El funcionamiento consiste en añadir la cantidad deseada, y posteriormente aceptar la transacción creada.

Figura B.6: Añadir o retirar fondos

B.7. Modificar el precio

Tanto el comprador como el vendedor de una cadena, pueden solicitar una actualización del precio de la misma. Para ello, en el menú lateral se ha de seleccionar la modificación de precio. Como se aprecia en la Figura B.7, el solicitante tiene que introducir el nuevo precio que desee. Al aceptar la solicitud en MetaMask, la otra persona podrá ver la solicitud cuando entre en la cadena de suministro. En la Figura B.8, se ve como la segunda persona puede aceptar la solicitud de cambio de precio. Una vez se acepta (con la consiguiente validación en MetaMask), todos los productos que se creen a partir de entonces, tendrán en cuenta el precio actualizado.

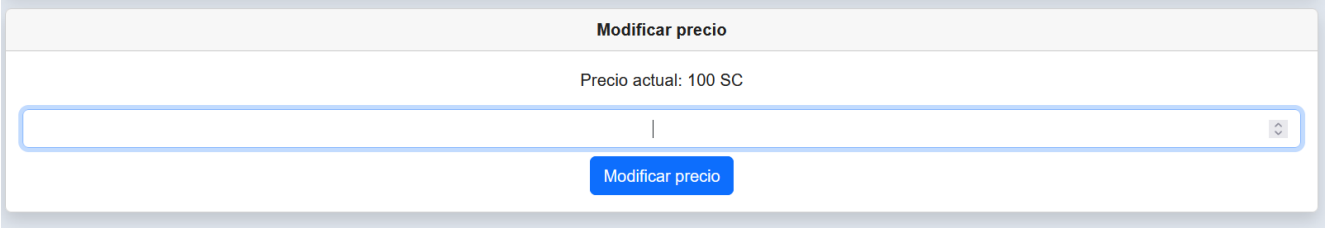


Figura B.7: Solicitud modificación de precio

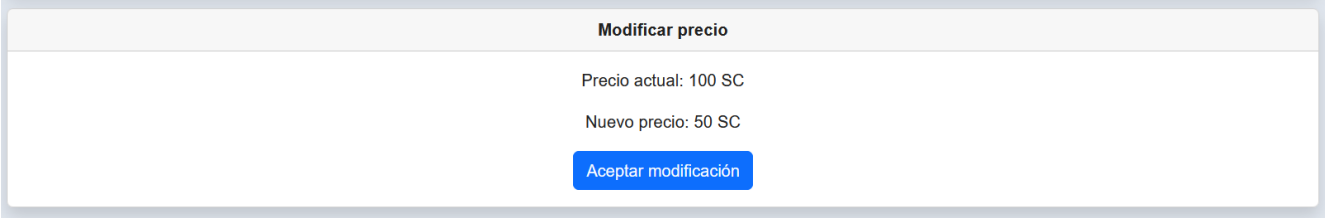


Figura B.8: Aceptar solicitud de modificación de precio

B.8. Añadir una cadena de suministro precedente

Es posible que un usuario de la aplicación esté asociado en varias cadenas de suministro de forma simultánea. Además, puede ejercer el rol de comprador en una, y de vendedor en otra. Este sería el caso de una empresa transformadora, que compra leche cruda a través de una cadena de suministro, y posteriormente, vende leche ultrapasteurizada a una empresa de supermercados. Para estos casos, la empresa puede relacionar ambas cadenas de suministro, para poder tener una traza de los productos y su transformación. Para ello, como se puede observar en la Figura B.9 el vendedor de una cadena de suministros, ha de ir al menú lateral y seleccionar la opción de editar precedentes. Una vez abierta esta sección, el vendedor, puede seleccionar aquellas cadenas en las que ejerce el rol de comprador. Una vez aceptada la transacción en MetaMask, la cadena de suministro actual, reflejará en su información, las demás cadenas seleccionadas como precedentes. Esto se puede ver en la Figura B.10. A partir de entonces, cada vez que el vendedor cree un lote de Leche UHT, deberá asignar de qué lote ha obtenido la leche cruda.

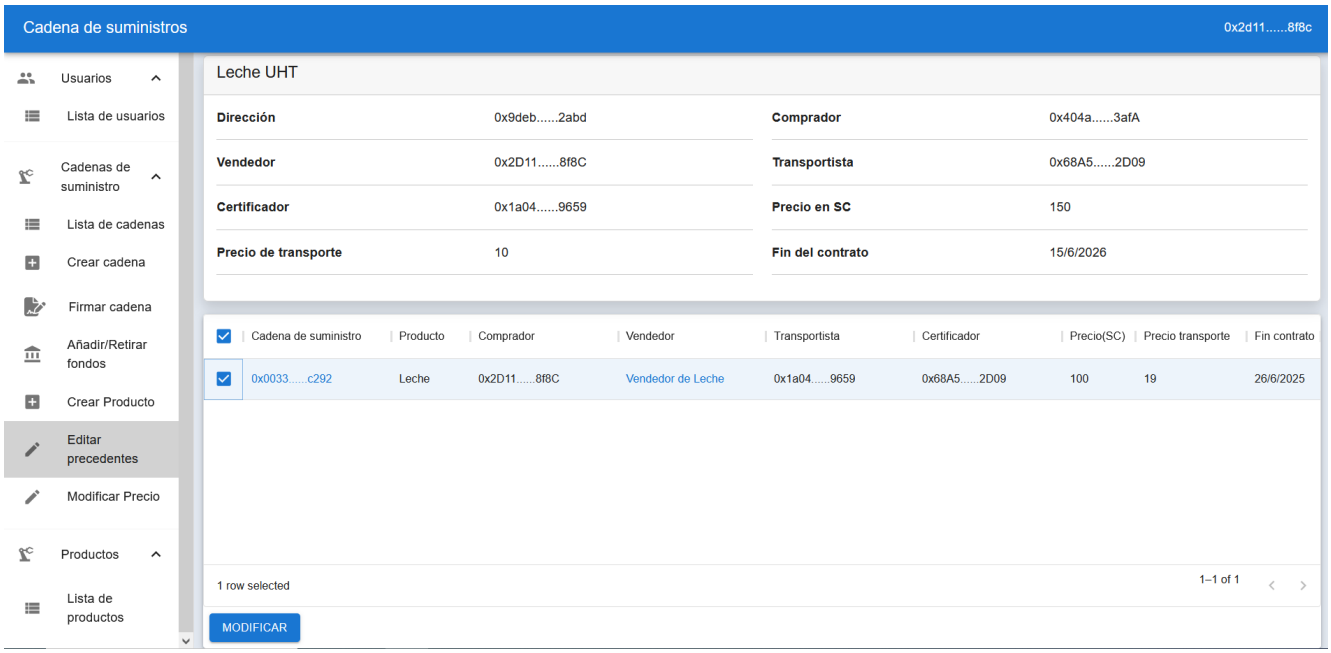


Figura B.9: Adición de una cadena de suministro como precedente

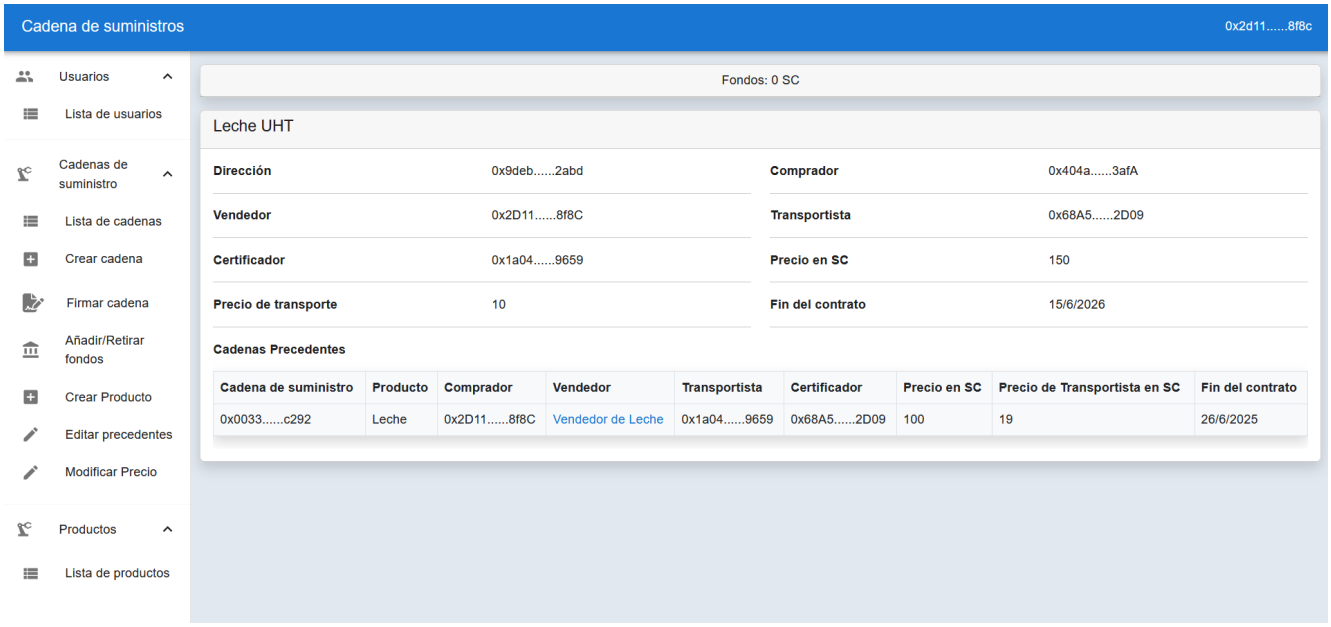


Figura B.10: Cadena de suministro con cadena precedente

B.9. Crear un producto

Una vez la cadena de suministro ha sido ratificada por todos los miembros, el vendedor puede empezar a crear los productos, que serán transportados al comprador. Como se aprecia en la Figura B.11, se selecciona la creación de producto en el menú lateral y tan solo hay que especificar la cantidad que contendrá el lote. Tras la validación

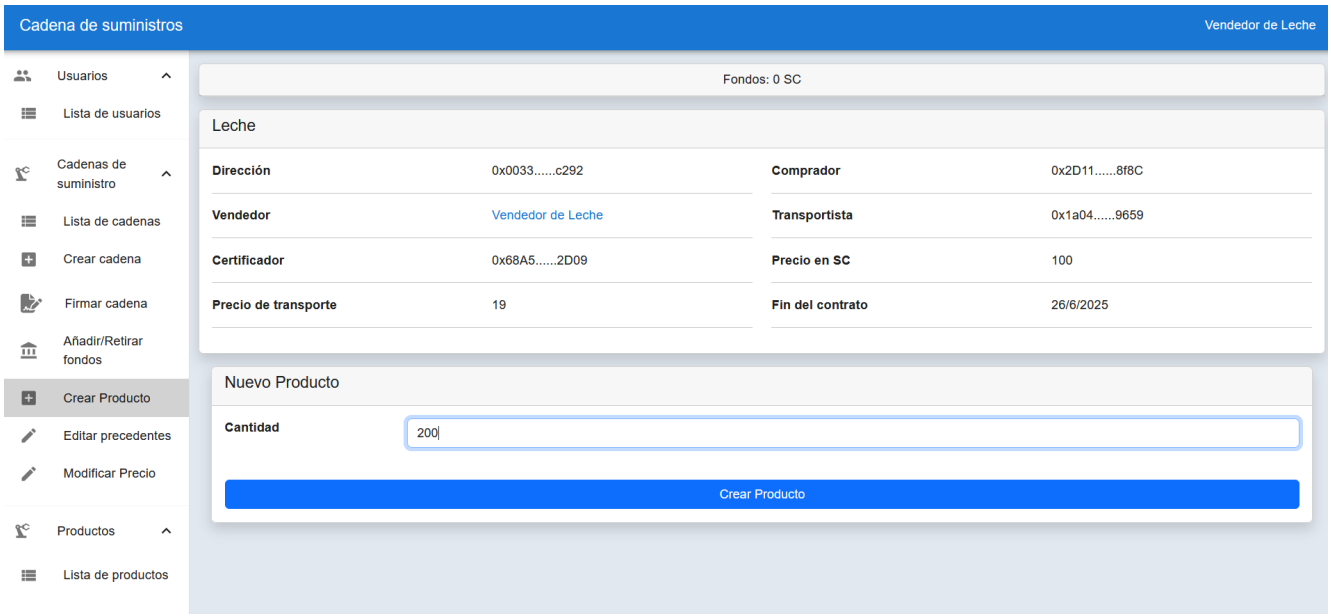


Figura B.11: Creación de un producto

de la transacción en MetaMask, el producto se habrá creado. Para acceder a el, hay que dirigirse a la lista de productos en el menú lateral, y seleccionarlo. En la Figura B.12 se puede comprobar la información del mismo. Como acaba de ser creado, el step en el que se encuentra es en el de creación.

Por otro lado, si la cadena en la que se crea un producto, tiene asignado otra cadena como precedente, será necesario seleccionar los productos que preceden al mismo. Estos tendrán que ser productos que hayan sido entregados, de la cadena precedente. En la Figura B.13 se puede ver un ejemplo.

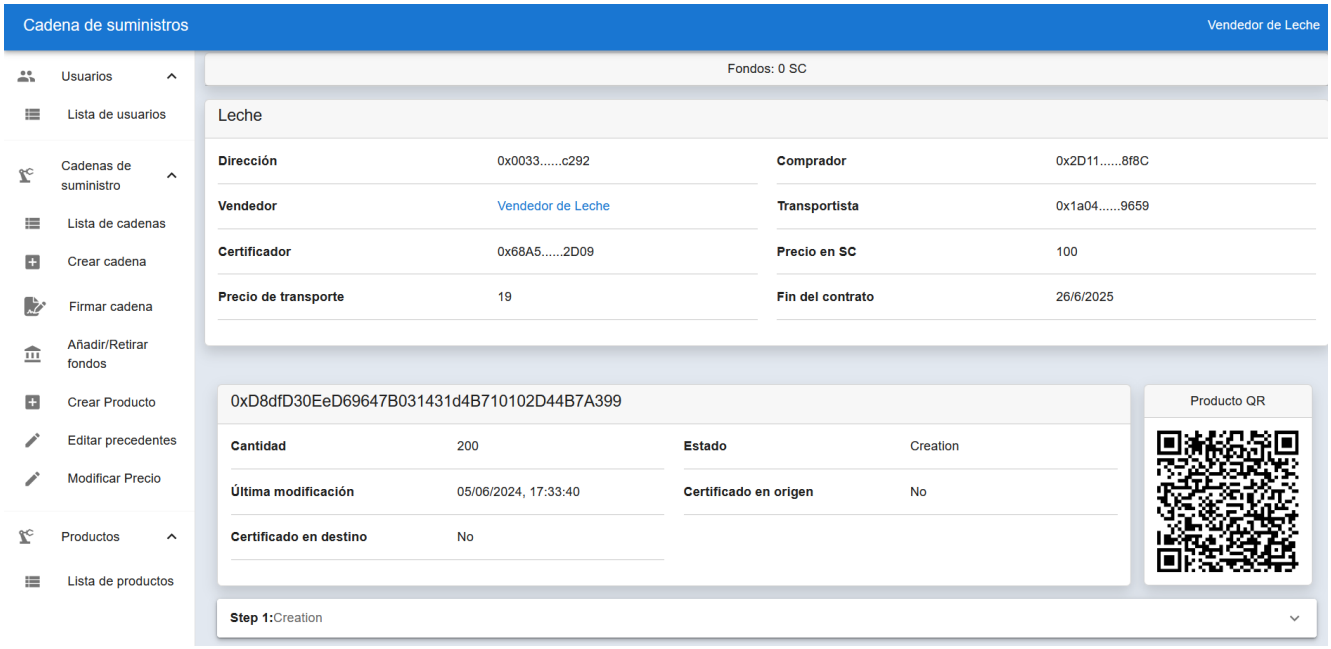


Figura B.12: Información de un producto

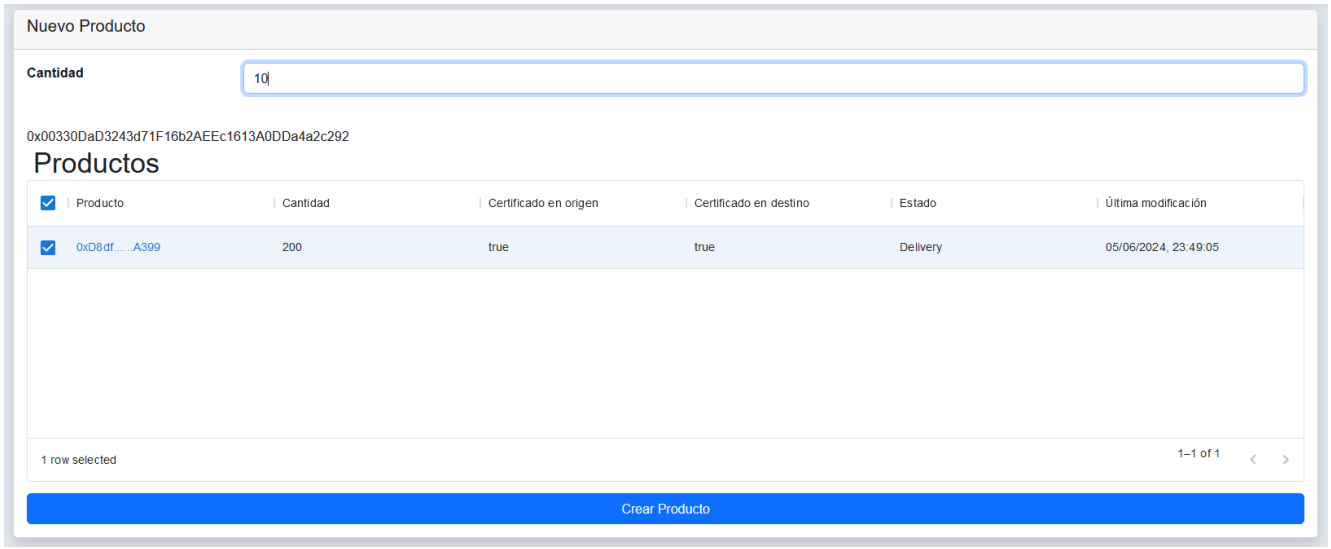


Figura B.13: Información de un producto

B.10. Interactuar con un producto

Cuando el vendedor de una cadena de suministros ha creado un nuevo producto, es posible que el resto de miembros de la cadena puedan realizar sus acciones asociadas. Cada una de estas acciones pedirá una validación a través de MetaMask, y creará un nuevo step con la información. Las acciones que se pueden realizar dependen del rol que tiene el usuario con la cadena de suministro y son las siguientes:

- Transportista:
  - Realizar envío
- Certificador:
  - Certificar en origen
  - Certificar en destino
- Comprador:
  - Recibir envío

En la Figura B.14 se puede apreciar un producto con los diferentes steps que ha sufrido, y pendiente de ser recibido por el comprador. Como al recibir un envío se desencadenarán los pagos al vendedor y al transportista, es necesario que el comprador tenga suficiente saldo en la cadena de suministro. Una vez finalizada la transacción de recibir envío, se habrá descontado de la cuenta del comprador el saldo correspondiente, y se marcará el producto como finalizado.

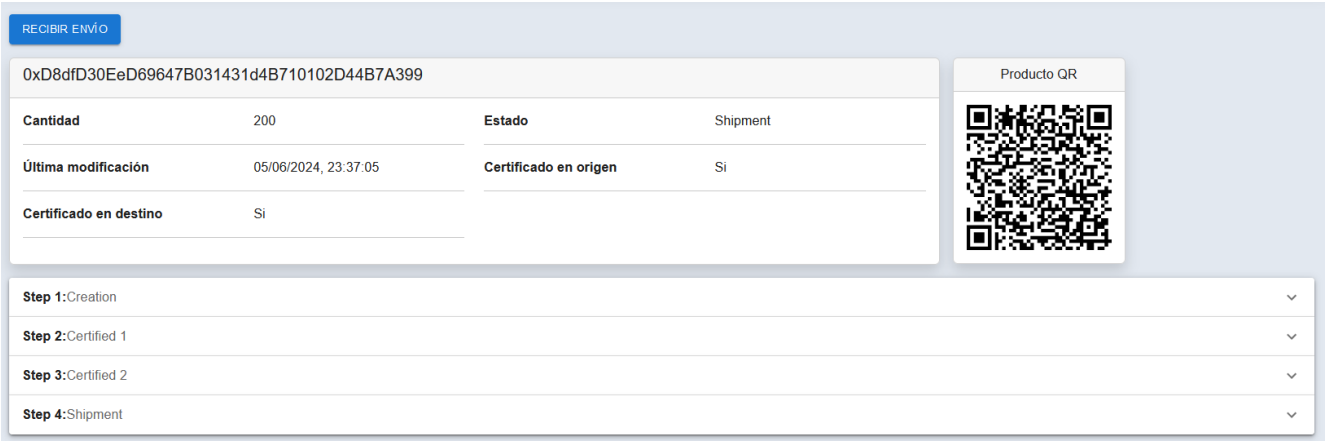


Figura B.14: Steps de un producto

Bibliografía

[1] Adebola Adeniran. *Cómo usar localStorage*. Último acceso 20-02-2023. URL: https://www.freecodecamp.org/news/how-to-persist-a-logged-in-user-in-react/.

[2] Francisco José García et al. *Modelos de desarrollo software USal*. Último acceso 20-01-2024. URL: https://repositorio.grial.eu/bitstream/grial/1940/1/IS_I%20Tema%203%20-%20Modelos%20de%20Proceso.pdf.

[3] Kailash Chandra Bandhu et al. *Ejemplo Cadena de suministro*. Último acceso 21-03-2024. URL: https://link.springer.com/article/10.1007/s11042-022-14238-4.

[4] Katerina Pace et al. *Documentación para desarrolladores de Ethereum*. Último acceso 15-03-2024. URL: https://ethereum.org/es/developers/docs/.

[5] Mark Otto et al. *Bootstrap Documentations*. Último acceso 04-10-2022. URL: https://getbootstrap.com/docs/5.2/getting-started/introduction/.

[6] rizedr et al. *Ethereum contract abstraction*. Último acceso 27-09-2022. URL: https://www.npmjs.com/package/@truffle/contract.

[7] Muhammad Altabba. *Cómo arreglar errores de Web3*. Último acceso 13-09-2022. URL: https://github.com/ChainSafe/web3.js#web3-and-create-react-app.

[8] Anónimo. *Login/SingUp Tutorial*. Último acceso 06-09-2022. URL: https://contactmentor.com/login-form-react-js-code/.

[9] BOE. *BOE-A-2021 Resolución de 4 de febrero de 2021*. Último acceso 23-05-2024. URL: https://www.boe.es/eli/es/res/2021/02/04/(3).

[10] BokkyPooBah. *Cómo compilar un contrato y enviar transacciones*. Último acceso 26-09-2022. URL: https://ethereum.stackexchange.com/questions/3285/how-to-get-return-values-when-a-non-view-function-is-called/3293#3293.

[11] Vitalik Buterin. "A next-generation smart contract and decentralized application platform". En: *white paper* (2014).

[12] MUI community. *Material UI*. Último acceso 15-01-2024. URL: https://mui.com/material-ui/getting-started/.

[13] Alberto Cuesta. *Ejemplo cadena de suministros*. Último acceso 17-10-2022. URL: https://medium.com/hackernoon/implementing-a-supply-chain-in-the-ethereum-blockchain-dcc91ea718ab.

[14] davdotsol. *Ejemplo conexión con MetaMask*. Último acceso 18-10-2022. URL: https://github.com/davdotsol/react-and-web3-demo-app.

[15] *Documentación Solidy*. Último acceso 20-02-2024. URL: https://solidity-es.readthedocs.io/es/latest.

[16] Informática DP. *Tutorial Base de datos en React*. Último acceso 06-09-2022. URL: https://www.youtube.com/watch?v=nqu9Zt68J_M.

[17] *Ejemplo Pharma Chain*. Último acceso 09-09-2022. URL: https://github.com/sherwyn11/Pharma-Chain.

[18] Autores Go Ethereum. *Documentación Geth*. Último acceso 12-02-2024. URL: https://geth.ethereum.org/docs/.

[19] Hyperledger Foundation. *Problemas y soluciones que resuelve la Blockchain*. Último acceso 06-09-2022. URL: https://www.youtube.com/watch?v=ywHIiUfWx5c.

[20] Francisco Giordano. *Crear un Token ERC20*. Último acceso 06-09-2022. URL: https://forum.openzeppelin.com/t/how-to-implement-erc20-supply-mechanisms/226.

[21] murtza gondal. *Ejmeplo de transacción*. Último acceso 10-01-2023. URL: https://ethereum.stackexchange.com/questions/84180/how-to-send-signed-transaction-to-blockchain-from-different-account.

[22] Ukeje Goodness. *Aplicación de la blockchain a cadenas de suministros*. Último acceso 25-09-2022. URL: https://blog.logrocket.com/examples-applications-smart-contracts/.

[23] Google. *Documentación de la API de Firebase para JavaScript*. Último acceso 09-09-2022. URL: https://firebase.google.com/docs/reference/js.

[24] Trey Huffine. *Ciclo de vida de componentes React*. Último acceso 17-01-2024. URL: https://levelup.gitconnected.com/componentdidmakesense-react-lifecycle-explanation-393dcb19e459.

[25] Fábio José. *Flujos de una cadena de suministros*. Último acceso 25-09-2022. URL: https://medium.com/coinmonks/build-a-smart-contract-to-sell-goods-6cf73609d25.

[26] Suresh Konakanchi. *Ejemplo Aplicación de cadena de suministros*. Último acceso 17-10-2022. URL: https://github.com/GeekyAnts/sample-supply-chain-ethereum.

[27] MetaMask. *Aplicación MetaMask*. Último acceso 16-02-2024. URL: https://metamask.io/.

[28] MetaMask. *MetaMask API Documentación*. Último acceso 04-10-2022. URL: https://docs.metamask.io/guide/rpc-api.html#table-of-contents.

[29] *Montaje del hook useAuth*. Último acceso 08-09-2022. URL: https://usehooks.com/useAuth/.

[30] Mozilla. *Promises en JavaScript*. Último acceso 11-09-2022. URL: https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Promise.

[31] *Problemas y soluciones que resuelve la Blockchain en cadena de suministros*. Último acceso 06-09-2022. URL: https://101noticias.com/tecnologia-blockchain-para-cadenas-de-suministro-quien-la-usa-y-como/.

[32] *React-Bootstrap Documentación*. Último acceso 04-10-2022. URL: https://react-bootstrap.github.io/.

[33] Remix. *React Router Documentación*. Último acceso 04-10-2022. URL: https://reactrouter.com/en/main.

[34] Remix. *React Router: Quick-Start*. Último acceso 12-09-2022. URL: https://v5.reactrouter.com/web/guides/quick-start.

[35] Remix. *Remix IDE*. Último acceso 03-03-2023. URL: https://remix.ethereum.org/.

[36] Remix. *Remix IDE*. Último acceso 15-01-2024. URL: https://remix-project.org/.

[37] Maximilian Schwarzmüller. *Ejemplo Intro a React*. Último acceso 23-01-2023. URL: https://github.com/academind/react-router-6.4-intro.

[38] Suraksha Singh. *Virtual DOM de React*. Último acceso 17-01-2024. URL: https://medium.com/@surksha8/virtual-dom-and-real-dom-understanding-the-differences-da8f3fab4261.

[39] *Solucionar polyfill error*. Último acceso 20-03-2023. URL: https://github.com/web3/web3.js#troubleshooting-and-known-issues.

[40] Meta Open Source. *Documentación Beta React*. Último acceso 18-01-2023. URL: https://beta.reactjs.org/.

[41] Meta Open Source. *Documentación React*. Último acceso 08-09-2022. URL: https://es.reactjs.org/.

[42] Creative Team. *Plantillas Bootstrap/React*. Último acceso 06-09-2022. URL: https://www.creative-tim.com/templates/react-bootstrap.

[43] Truffle Team. *Documentación Truffle*. Último acceso 15-01-2024. URL: https://trufflesuite.com/docs/truffle/.

[44] TutorialsPoint. *Solidity Documentación*. Último acceso 21-02-2023. URL: https://www.tutorialspoint.com/solidity/index.htm.

[45] Markus Waas. *Arquitectura Dapp*. Último acceso 22-03-2024. URL: https://soliditydeveloper.com/solidity-overview-2020.

[46] *Web3 Documentación*. Último acceso 08-02-2023. URL: https://web3js.readthedocs.io/en/v1.8.0/.

[47] Wikipedia. *Qué es un Sybil Attack*. Último acceso 13-01-2024. URL: https://en.wikipedia.org/wiki/Sybil_attack/.

[48] Yutai. *Sistema de cuentas de usuario en Ethereum*. Último acceso 24-09-2022. URL: https://blog.wetrust.io/designing-user-account-systems-in-ethereum-apps-f824fe625412.

[49] Matt Zabriskie. *Documentación Axios*. Último acceso 21-09-2022. URL: https://axios-http.com/.