UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Sistema de trazabilidad para cadenas de suministro con Blockchain en un entorno empresarial

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autora: Alicia Monleón Durá

Tutora: María Alpuente Frasnedo

Curso: 2019-2020

Agradecimientos:
A mis padres Maite, Antonio y Fermín por apoyarme y entenderme.
A mi hermano Juan por aconsejarme y confiar en mí.
A mi tutora María por su atención y paciencia para conmigo.

Abstract

El crecimiento del uso de la tecnología blockchain está siendo exponencial gracias a las ventajas que aporta. Sin embargo, el más mínimo error en un contrato inteligente puede propiciar la pérdida de dinero, bienes, información sensible, etc. Por este motivo es imprescindible aplicar un buen diseño, cuidar el desarrollo y proveer de una documentación clara del mismo, así como aumentar las garantías de que el software desarrollado es correcto. Con la motivación de este reto, en este proyecto exploramos estas tecnologías de un modo conceptual, aplicándolas en el contexto particular de un sistema de trazabilidad para cadenas de suministro que permite acceder a la información de seguimiento, asegurando la transparencia y sostenibilidad de los procesos. La arquitectura de esta solución integra dispositivos IoT que recogen información del entorno, un DLT blockchain para almacenarlos de forma segura e íntegra, un motor de búsqueda y una aplicación móvil que permite el acceso a los productos de la cadena aportando un enfoque práctico al proceso de diseño y desarrollo de contratos inteligentes escritos en C# (y ejecutados en Hyperledger Sawtooth, una plataforma blockchain para empresas).

Palabras clave: Industria 4.0, digitalización, internet de las cosas, tecnología de libro mayor distribuido, cadena de bloques, contratos inteligentes, móvil, motor de búsqueda, cadena de suministro, trazabilidad

Abstract

In recent years, there has been an exponential growth in the use of blockchain technology thanks to the advantages it brings. However, the slightest mistake in a smart contract can lead to the loss of money, goods, sensitive information, etc. For this reason, it is essential to apply a good design, take care of the development and provide clear documentation of it, as well as to increase the guarantees for software correctness. Driven by this challenge, this final project focuses on these technologies conceptually and applies them in the particular context of a tracking system for supply chains that grants access to monitoring information, ensuring both transparency and sustainability. The architecture of the provided software solution seamlessly integrates IoT devices that gather information from the environment, a blockchain DLT that stores the collected information and ensures both safety and integrity, a search engine and a mobile application that allows the chain products to be accessed, providing a practical approach to the design and development of smart contracts written in C# (and executed in Hyperledger Sawtooth, a blockchain platform for companies).

Keywords : Industry 4.0, digitization, internet of things, distributed ledger technology, blockchain, smart contracts, mobile, search engine, supply chain, traceability

Tabla de contenidos

1. Introducción

Nos encontramos en los albores de la cuarta revolución industrial, una revolución dada por la llamada industria 4.0. Entendemos este tipo de industria como aquélla que hace un uso intensivo de Internet y las últimas tecnologías, como lo son la interconexión de dispositivos que interactúan independientemente entre sí, conocido como *Internet of Things*, la tecnología *blockchain* y los *smart contracts*, el desarrollo de IAs que operan con grandes volúmenes de datos (*Big Data*) en tiempo real, la robótica y la realidad virtual y aumentada, entre otras. Es por tanto un momento emocionante en el que la creatividad y la experimentación nos llevan a construir un futuro que, hasta hace poco, sólo existía en nuestra imaginación.

Este acontecimiento histórico está siendo liderado por las empresas y organizaciones en su búsqueda por maximizar la eficiencia de sus operaciones. Los resultados que se esperan de esta evolución son la optimización de los niveles de calidad, la reducción de costes y tiempos de producción y logísticos, un aumento de la competitividad empresarial y una respuesta mejor adaptada a las necesidades del mercado, además de un uso más adecuado de los recursos que contribuye al cuidado del medioambiente. Es de vital importancia para muchas de ellas poder adaptarse a los nuevos tiempos pues, de lo contrario, corren el riesgo de quedar obsoletas y no poder hacer frente a la competencia.

Si bien este trabajo está orientado a un entorno empresarial, su aplicación es igual de válida para el sector público. Por ejemplo, en el sector educativo puede permitir guardar el historial académico y agilizar la equivalencia de estudios con otras entidades y otros países. Asimismo tiene aplicaciones en el sector sanitario, donde un usuario tiene todos sus datos médicos, tales como operaciones, alergias, tratamientos, etc, pudiendo el paciente dar permiso de lectura a aquellos médicos que vayan a atenderle. Además, mediante contratos inteligentes el doctor puede recetar medicamentos y configurar esas recetas de forma que la app notifique al paciente a las horas que debe tomar cada medicamento. El paciente, con esta misma app, puede ir a la farmacia a recoger los medicamentos recetados. Otra ventaja a sumar sería la homogeneización del software utilizado por los centros educativos y sanitarios respectivamente, mejorando la movilidad laboral de los usuarios y evitando que tengan que aprender el uso de nuevos programas y rutinas en función del centro.

1.1. Motivación

Los motivos que me han llevado a realizar este TFG son principalmente tres. El primero, es mi intención de aportar valor a aquellas personas que lean este trabajo cualesquiera que sean sus motivos para hacerlo. Elegí hacerlo sobre el diseño de aplicaciones *full-stack* poniendo especial atención a la tecnología *blockchain* porque creo que ésta aporta beneficios a la sociedad de diversas formas y creo que merece la pena aprender el uso de una herramienta que repercutirá de forma positiva tanto en organizaciones como en individuos. El segundo, es mi deseo de compartir algunas de las cosas que llevo explorando y estudiando desde hace ya varios años. Y el tercero,

es mi intención de continuar dedicándome al desarrollo de estas tecnologías, a la transformación digital y la digitalización de las empresas.

1.2. Objetivos

1. Analizar el ecosistema de las cadenas de suministro con el fin de mostrar los beneficios de un programa basado en *blockchain* para los distintos roles y sus necesidades particulares.
2. Diseñar un programa que permita introducir datos en una *blockchain*.
3. Diseñar el programa de forma que esos datos puedan ser consultados utilizando un motor de búsqueda indexada.
4. Implementar una aplicación *full-stack* de forma que podamos probar su funcionamiento.
5. Que esta implementación sirva como base estructural para posteriormente poder ser adaptada a un modelo de negocio particular.
6. Integrar otra tecnología muy relacionada con las cadenas de suministro como lo son los dispositivos IoT.

1.3. Impacto esperado

¿Por qué una empresa u organización querría utilizar la tecnología *blockchain*? Porque nos proporciona más seguridad y confianza que ninguna otra herramienta informática disponible en el mercado. ¿Por qué es más segura y proporciona mayor confianza? Por varias razones: 1) los datos transmitidos están intrínsecamente encriptados; 2) descentraliza la gestión de las transacciones (dicho de otro modo, no existe un "único punto de falla"); y 3) es fácil detectar cuándo un bloque ha sido manipulado gracias a las funciones *hash*. Para poder hackearla se debería controlar más del 50% de la red en menos tiempo de lo que se tarda en crear un nuevo bloque. La potencia de cálculo que hace falta para llevar esto a cabo es tan inmensa que la hace a efectos prácticos imposible de ser hackeada. Además, los valores de un bloque se incluyen en el siguiente en forma de hash lo que significa que cualquier alteración en uno de los bloques es advertido inmediatamente y, por lo tanto, no tendría ningún efecto manteniendo así la integridad de la información.

Una de las mayores virtudes que ofrece la tecnología *blockchain* es la transparencia. Esta tecnología que nos permite realizar transacciones de valor sin que intervengan intermediarios es una red P2P en la que todos los participantes poseen un mismo "libro contable" que es actualizado simultáneamente para todos ellos. Las transacciones pueden ser monetarias (criptomonedas) o de otra naturaleza (bienes, información, servicios, etc.). Esto posibilita automatizar pagos y transferencias basados en un conjunto predeterminado de condiciones. El Real Instituto Elcano la describe como sigue [Lecuit19]:

> "El blockchain ofrece transparencia (todos los participantes pueden ver la totalidad de la información contenida en la base de datos distribuida), compartición y descentralización (una misma copia de la base de datos en

> todos los nodos), irreversibilidad (una vez registrado un dato, no puede ser modificado o borrado) y desintermediación (sin árbitro central, los participantes toman las decisiones por consenso). La cadena de bloques enlaza la secuencia de transacciones e incorpora una marca de tiempo que da transparencia y trazabilidad a las operaciones sin por ello quebrantar a priori la privacidad de los usuarios (puede conocerse el camino y el contenido aunque no siempre sea factible inferir la identidad del usuario). Los actores pueden adoptar tres roles: usuarios con derecho a tener y consultar una copia de la base de datos distribuida (accessors), participantes con derecho a realizar transacciones (*participants*) y usuarios encargados de validar las transacciones y crear bloques (*miners*). Todos ellos disponen de una copia validada y única de la base de datos.
>
> Cada plataforma establece sus reglas de participación, operación y gobernanza. Las plataformas pueden ser abiertas (públicas) si son accesibles sin restricciones (*permissionless ledgers*), como, por ejemplo, la criptomoneda Bitcoin. Son semipúblicas o autorizadas (*permissioned ledgers*) cuando se condicionan la participación, el derecho a veto de nuevos miembros o la posibilidad de decidir el protocolo de consenso al inicio de la cadena. También pueden ser privadas cuando un actor establece las reglas; en este caso, se desdibuja la diferencia entre una cadena de bloques y una base de datos descentralizada convencional.
>
> El blockchain emplea mecanismos criptográficos de seguridad para acceder, firmar y cifrar las transacciones, los bloques y su encadenado. Las claves privadas pueden estar vinculadas a la identidad de los usuarios o a elementos intermedios; por ejemplo, las carteras digitales con las que la plataforma ofrece el anonimato de las operaciones. Las reglas que ejecutan las transacciones pueden estar establecidas mediante contratos inteligentes; en el blockchain Ethereum, por ejemplo, aseguran un entendimiento común de la transacción entre las partes, en particular sobre las obligaciones contraídas, ofreciendo una visibilidad probatoria limitada a los interesados (las terceras partes del blockchain ajenas al contrato no tienen acceso a sus estipulaciones o a su cumplimiento).”

Podemos observar que se trata de una tecnología con un alto impacto en aplicaciones que requieran de algún tipo de seguimiento. Las organizaciones pueden rastrear la información de forma más sencilla y procesar el historial de forma permanente. Además, la ausencia de una autoridad central o intermediarios hace que la información esté al alcance de todos los participantes de la red *in situ*. Esta simplificación del proceso de transmisión de datos conlleva inherentemente una mayor velocidad en la gestión, lo que permite a las empresas actuar más rápidamente. Esto repercute directamente en menos costes para la organización y una mayor eficiencia en sus procesos y administración de sus recursos. Además, según los expertos será esencial para el Internet de las Cosas [EAEBusinessSchool20]. El mismo artículo afirma que:

> “Tal y como explicaban en Expansión hace algunas semanas, “el 33% de las empresas ya están utilizando la tecnología blockchain o se plantean su uso en breve y un 78% de los que ya la están explorando lo hace como respuesta a los cambios que se producen en su sector o para

desarrollar nuevos modelos de negocio". En este mismo artículo aseguran que algunos directivos "cuentan con la tecnología blockchain para potenciar una nueva generación de aplicaciones transaccionales creadas para fomentar la confianza de sus clientes y ser más competitivas". En este sentido, es importante tener en cuenta que, al vincular a personas, recursos y organizaciones en un ecosistema interactivo, las empresas podrían ofrecer todo tipo de servicios adicionales."

La tecnología blockchain tiene el potencial de ayudar a resolver algunos de los grandes desafíos sociales, medioambientales y económicos de nuestro mundo. y oportunidades que ofrece el futuro del trabajo y analizar cómo las empresas líderes incorporan la tecnología, considerando en sus decisiones criterios éticos y maximizando los impactos positivos, tanto ambientales como sociales [Forética18].

Oportunidades a nivel económico

- Reducción de costes.
- Generación de nuevos modelos de negocio.
- Creación de estándares transversales a las industrias, que generen confianza entre todos los actores; esto es algo muy importante, sobre todo, en aquellas industrias con cadenas de suministro complejas.
- Es clave para el desarrollo del internet de las cosas (IoT) ya que facilita la colaboración y los contratos entre máquinas sin intervención humana y de forma segura.
- Asegurar la privacidad de los datos.
- Reducir los riesgos cibernéticos.

Oportunidades medioambientales

- Cadenas de suministro completamente transparentes que permitan la trazabilidad de producto desde el origen. Esto sentaría las bases para una producción más sostenible.
- Gestión descentralizada de los recursos, lo que permite la toma de decisiones más eficientes y procesos más flexibles y dinámicos.
- Nuevas fuentes de acceso a financiación para sufragar los gastos de proyectos medioambientales.
- Potenciación de la economía circular, permitiendo el seguimiento de recursos y materiales a lo largo de todo el ciclo de vida de los productos.
- Transformación de los mercados energéticos mediante la gestión descentralizada y localizada de la energía.

Consecuentemente vemos cómo estas oportunidades se alinean con varios de los Objetivos de Desarrollo Sostenible desarrollados por la Organización de las Naciones Unidas. Véanse algunos de los ejemplos en los que *blockchain* tiene un impacto directo:

- Objetivo 2: Poner fin al hambre, lograr la seguridad alimentaria y la mejora de la nutrición y promover la agricultura sostenible.

- Objetivo 4: Garantizar una educación inclusiva, equitativa y de calidad y promover oportunidades de aprendizaje durante toda la vida para todos

- Objetivo 5: Lograr la igualdad entre los géneros y empoderar a todas las mujeres y las niñas.

- Objetivo 7: Garantizar el acceso a una energía asequible, segura, sostenible y moderna para todos.

- Objetivo 9: Construir infraestructuras resilientes, promover la industrialización inclusiva y sostenible y fomentar la innovación.

- Objetivo 10: Reducir la desigualdad en y entre los países.

- Objetivo 12: Garantizar modalidades de consumo y producción sostenibles.

- Objetivo 16: Promover sociedades pacíficas e inclusivas para el desarrollo sostenible, facilitar el acceso a la justicia para todos y crear instituciones eficaces, responsables e inclusivas a todos los niveles.

- Objetivo 17: Fortalecer los medios de ejecución y revitalizar la Alianza Mundial para el Desarrollo Sostenible.

Esto es el impacto que podemos esperar de la tecnología *blockchain* en genérico. Respecto al impacto del propio trabajo que está leyendo ahora, más que el impacto que espero, es importante el impacto que deseo de que quién quiera que lo lea pueda aprender algo nuevo, si puede ser que disfrute haciéndolo y que ojalá le sea útil de alguna manera.

1.4. Metodología

En primer lugar haremos un análisis de un escenario para el cual la tecnología *blockchain* resulta especialmente beneficiosa. Posteriormente, realizamos un diseño conceptual en el que observamos los componentes que formarán parte de nuestro sistema. A continuación se irán implementando por etapas en el siguiente orden:

1. Instalación y puesta a punto de la Raspberry.
2. Implementación del programa de lectura de códigos.
3. Implementación del procesador de transacciones y actualización correspondiente en el programa cliente ejecutado en la Raspberry.
4. Implementación de un prototipo en Android que permite hacer llamadas RESTful a Elasticsearch.
5. Actualización del procesador de transacciones para emitir eventos propios de la aplicación.

6. Implementación de un *listener* que escuche estos eventos y escriba los datos recibidos en Elasticsearch.
7. Instalación del programa cliente en la Raspberry.
8. Refactorización del código.
9. Automatización del despliegue utilizando Docker.

1.5. Estructura

Introducción

En esta primera sección se introduce el tema que se trata y posteriormente se divide en seis subapartados. El primero justifica la motivación que hay tras este TFG; el segundo lista los objetivos a cumplir; el tercero señala las ventajas/mejoras que va a suponer el producto/servicio resultante del trabajo para el usuario; el cuarto explica los diferentes pasos que se proponen para el cumplimiento de los objetivos; el quinto es un pequeño índice general comentado que hace un recorrido rápido de aquello que podemos encontrar en los sucesivos capítulos; y el sexto es una lista con las convenciones de marcado que llevan asociado un significado adicional.

Estado de la cuestión

Este capítulo hace un recorrido a lo largo de la historia haciendo referencia a los trabajos previos que hicieron posible el nacimiento de Bitcoin, siguiendo con los contratos inteligentes y el desarrollo de las dapps, hasta la actualidad. Aquí se explica en qué consiste la tecnología *blockchain*, sus características, se mencionan las principales criptomonedas y plataformas existentes en la actualidad y se pone en relieve su papel en la cuarta revolución industrial. Por último se enmarca el propio trabajo presentado en este documento en relación a esta herramienta.

Análisis

En este bloque se explica en qué consiste una cadena de suministro y se resume el proceso que sigue un producto que es vendido en el mercado internacional. Se detallan aspectos del protocolo de importación y exportación para detectar las distintas entidades que pueden estar involucradas y hacerse una idea de la complejidad que supone el sistema, así como aquellos factores a tener en cuenta para el diseño del *software*. Se presenta mediante cuatro roles principales y sus casos de uso para resaltar aquello que resulta fundamental para la cadena y sus participantes.

Diseño de la solución

Se subdivide en tres secciones que explican nuestra solución y la tecnología empleada. Las dos primeras partes, Arquitectura y Diseño detallado, describen y muestran mediante diagramas la estructura y flujo de operaciones de nuestro *software*. La tercera parte, Tecnología utilizada, habla de los aspectos técnicos de las principales herramientas utilizadas: la Raspberry Pi para IoT, Android para móvil, Elasticsearch para el motor de búsqueda, de forma más extensa se profundiza en Hyperledger Sawtooth para la *blockchain*, y también de forma breve sobre Docker y ZeroMQ.

Desarrollo de la solución propuesta

En este apartado se describe parte del código implementando.

Implantación

Esta sección explica cómo instalar el programa cliente en la raspberry y cómo poner en funcionamiento el programa.

Pruebas

En este apartado se muestra la comprobación manual del código implementando.

Conclusiones

Este capítulo hace referencia a los objetivos formulados inicialmente.

Trabajos futuros

En este breve apartado se exponen aquellas mejoras que, de haber tenido más tiempo, podrían haberse aplicado o partes del programa que pueden extenderse.

Referencias

Bibliografía y recursos utilizados.

Glosario

Definición de términos utilizados en el documento.

1.6. Convenciones

Para las palabras en un idioma extrangero se utiliza cursiva salvo nombres propios.

Las palabras en inglés que hagan referencias a elementos del software se escriben entre comillas simples.

Para los fragmentos de código se utiliza el fondo gris.

2. Estado de la cuestión

"The problem, in a nutshell, is that our money currently depends on trust in a third party for its value. As many inflationary and hyperinflationary episodes during the 20th century demonstrated, this is not an ideal state of affairs. Similarly, private bank note issue, while it had various advantages as well as disadvantages, similarly depended on a trusted third party.

Precious metals and collectibles have an unforgeable scarcity due to the costliness of their creation. This once provided money the value of which was largely independent of any trusted third party. Precious metals have problems, however. It's too costly to assay metals repeatedly for common transactions. Thus a trusted third party (usually associated with a tax collector who accepted the coins as payment) was invoked to stamp a standard amount of the metal into a coin. Transporting large values of metal can be a rather insecure affair, as the British found when transporting gold

across a U-boat infested Atlantic to Canada during World War I to support their gold standard. What's worse, you can't pay online with metal.

Thus, it would be very nice if there were a protocol whereby unforgeably costly bits could be created online with minimal dependence on trusted third parties, and then securely stored, transferred, and assayed with similar minimal trust."

Nick Szabo
December 29, 2005

2.1. La era antes de Bitcoin

Desde la década de 1970, la utilización de firmas digitales basadas en criptografía de clave pública ha proporcionado un fuerte control de propiedad. Sobre la base de la criptografía de clave pública, en 1998 Wei Dai describe en su publicación "*b-Money, an anonymous, distributed electronic cash system*" una solución descentralizada al problema de pagos electrónicos. En ella, Dai explica las propiedades básicas de todos los sistemas modernos de criptomonedas; un esquema para que un grupo de seudónimos digitales no rastreables se paguen entre sí con dinero y hagan cumplir contratos entre ellos sin ayuda externa. Definido como "dinero que es imposible de regular", el b-Money de Dai trazó los conceptos centrales que más tarde serían implementados en Bitcoin y otras criptomonedas:

- Requiere una cantidad específica de trabajo computacional (también conocido como Prueba de trabajo).
- El trabajo realizado es verificado por la comunidad que actualiza un libro de contabilidad colectivo.
- El trabajador recibe fondos por su esfuerzo.
- El intercambio de fondos se realiza mediante la contabilidad colectiva y se autentica con hashes criptográficos.
- Los contratos se hacen cumplir mediante la transmisión y firma de transacciones con firmas digitales (es decir, criptografía de clave pública).

Ese mismo año, Nick Szabo diseñó un mecanismo para una moneda digital descentralizada que llamó Bit Gold que, aunque nunca se implementó, se le conoce como un precursor directo de la arquitectura de Bitcoin. Este mecanismo consiste en que los participantes de la red dedican capacidad de cómputo para resolver puzles criptográficos. Esos puzles resueltos son enviados a un registro público donde se asigna la clave pública del participante que resolvió el problema. Cada solución se convierte en parte del siguiente reto, creando un nuevo "eslabón" de la cadena de bloques. Este aspecto del sistema proporciona una forma para que la red verifique y genere nuevas monedas, porque a menos que la mayoría de las partes acuerden aceptar nuevas soluciones, no podrán comenzar con el siguiente enigma.

Seis años después, en 2004 Hal Finney, creó el primer sistema RPOW (*Reusable Proof of Work*) anterior a Bitcoin. Un sistema de prueba de trabajo o PoW es un mecanismo de consenso que, para evitar comportamientos indeseados, requiere que el cliente del servicio realice algún tipo de trabajo que tenga cierto coste y que pueda ser verificado fácilmente en la parte del servidor. El concepto fue inventado por Cynthia Dwork y Moni Naor cuando en 1993 se presentó por primera vez la idea y

técnica para combatir el correo electrónico no deseado mediante la exigencia de una prueba de esfuerzo computacional. El término "*Proof of work*" fue formalizado posteriormente en un artículo de 1999 por Markus Jakobsson y Ari Juels. Ya en 1999, el premio Nobel de economía Milton Friedman dijo [Milton13]:

> *"The one thing that's missing, but that will soon be developed, it's a reliable e-cash. A method where buying on the Internet you can transfer funds from A to B, without A knowing B or B knowing A. The way in which I can take a 20 dollar bill and hand it over to you and there's no record of where it came from. And you may get that without knowing who I am. That kind of thing will develop on the Internet."*

2.2. Nacimiento y primeros años de Bitcoin

Fue por fin el 31 de octubre 2008 cuando, bajo el pseudónimo de Satoshi Nakamoto, se publicó un artículo en la lista de criptografía de metzdowd.com que describía el protocolo Bitcoin. El trabajo titulado "*Bitcoin: A Peer-to-Peer Electronic Cash System*" describe los detalles conceptuales y técnicos de un sistema de pago que permite a las personas enviar y recibir pagos sin involucrar a ninguna institución financiera intermediaria. Éste supuso el inicio de lo que se augura una revolución que no ha hecho más que empezar. Hasta este momento las bases de datos se conectaban a través de redes de computadores, la fusión de estas dos es lo que dio lugar a esta nueva tecnología *blockchain*. Es decir todo nodo conectado a esta red comparte una única realidad; al actualizarse un dato este es actualizado al mismo tiempo en todos los nodos que forman la red. Lo que esto supone y la hace relevante es que se sustenta en ser extremadamente difícil de falsificar y semejante a un gran libro contable, público y distribuido, en el que queda reflejado el histórico de todas las transacciones. La idea detrás de la tecnología *blockchain* se describió en 1991, cuando los científicos de investigación Stuart Haber y W. Scott Stornetta introdujeron una solución computacionalmente práctica para los documentos digitales con sello de tiempo de forma que no pudieran ser modificados o manipulados. El sistema usó una cadena de bloques con seguridad criptográfica para almacenar los documentos con sello de tiempo y en 1992 se incorporaron al diseño los árboles Merkle, lo que lo hizo más eficiente al permitir que varios documentos se juntaran en un solo bloque. Sin embargo, esta tecnología no se utilizó y la patente caducó en 2004, cuatro años antes del inicio de Bitcoin.

Pocos meses después, el 3 de enero de 2009, Satoshi lanzó Bitcoin haciendo realidad una alternativa al sistema financiero actual. El día 12 de ese mismo mes se realizó la primera transacción entre Satoshi Nakamoto y Hal Finney en Bitcoin sin ser respaldado por ningún gobierno o banco central. Para ello utiliza un sistema de prueba de trabajo que impide el doble gasto. El consenso entre todos los nodos que integran la red se alcanza intercambiando información sobre una red no fiable y potencialmente comprometida resolviendo el problema de los generales bizantinos. Desde su puesta en funcionamiento tras la publicación del primer programa cliente, de código abierto, y la creación de los primeros bitcoins, dejó de ser necesario que todos los pagos en el comercio electrónico se hiciesen efectivos a través de entidades

centralizadas de confianza, generalmente bancos y otras empresas financieras que gestionan el seguimiento de todas las transacciones.

En marzo de 2010 apareció el primer mercado de intercambio de criptomonedas bajo el nombre de bitcoinmarket.com (actualmente inactiva). El 22 de mayo del mismo año se hizo la primera compra de bienes cuando Laszlo Hanyecz compró dos pizzas por 10.000 BTC. Este día es conmemorado como el Bitcoin Pizza Day. El valor del Bitcoin era de 0.08$ para cuando se lanzó en julio el mercado de intercambio Mt. Gox. Más tarde, el 15 de Agosto de 2010, Bitcoin encontró su mayor error hasta el momento. El error que Garzik, entre otros, descubrió en el bloque 74638 fue el primer error de inflación de Bitcoin. Dado que el suministro total de la criptomoneda tiene un límite de 21 millones, la adición de 184 mil millones de monedas fue un problema importante, por decirlo suavemente. Un desbordamiento de enteros había causado un valor de transacción total negativo. Tras la apertura de Silk Road en 2011, un mercado online donde usuarios anónimos podian comprar y vender artículos (en su mayoría ilegales) con bitcoins, la criptomoneda empezó a tener mala prensa, al mismo tiempo que el valor de un bitcoin alcanzaba el de un dólar y la gente comenzaba a verlo como la reinvención del oro. Todo ello generó más interés y que cada vez más gente comenzase a minar bitcoins. Dado que el código de Bitcoin es de código abierto (disponible para el público), las personas también comenzaron a crear sus propias criptomonedas.

Entre estas nuevas criptos surge Tether, la primera criptomoneda vinculada 1 a 1 frente al dólar estadounidense. Tether es una moneda estable o *stablecoin* que fue creada con la intención de que sus propietarios pudieran mantener su poder adquisitivo. Por cada USDT en circulación, se agrega $1 USD a una cuenta de ahorros administrada centralmente como garantía. Es decir, si se tienen 1.000 USDT se tienen 1.000 dólares, con la ventaja de que pueden mantenerse en una wallet segura, del mismo modo que se mantiene cualquier otra criptomoneda y prescinde de los bancos. Es muy útil para los comerciantes ya que USDT les permite comercializar sin preocuparse por la volatilidad. Sin embargo, algunas personas dudan de que Tether esté totalmente garantizado. Divisas como la libra inglesa, el real brasileño o el złoty polaco pueden ser ya intercambiadas por Bitcoins. También surge OpenCoin, ahora llamado Ripple, un protocolo de código abierto que se utilizaría como un sistema de pago distribuido entre pares, particularmente en la banca y las finanzas. Wordpress.com y Wikileaks comienzan a aceptar pagos en Bitcoin haciendo que cada vez empresas, equipos e individuos empiecen a darse cuenta de que la tecnología subyacente de Bitcoin, *blockchain*, podría tener otras aplicaciones. VirWoX, por ejemplo, se convierte en el primer mercado en intercambiar Bitcoins contra una moneda virtual: Second Life Lindens.

El 2 de octubre de 2013 el FBI tumbó Silk Road y su “dueño” identificado como Ross Ulbricht fue arrestado y más tarde condenado a cadena perpetua. Después de mucho tiempo cerrado, en el mes de octubre de 2013 el sitio sería restablecido y seguiría funcionando de igual forma que antes, al tiempo que lo harían también otros, como PANDORA Open Market, SheepMarket y The Black Market Reloaded. Pantera Capital, la primera empresa de inversión de Bitcoin de Estados Unidos invierte en mercados de intercambios de criptomonedas como Coinbase, Circle y Bitstamp.

2.3. La llegada de los *smart contracts*

El acontecimiento realmente importante de este año 2013 fue cuando Vitalik Buterin, un desarrollador de 19 años que cofundó la revista Bitcoin, publicó un documento técnico llamado "Ethereum: un contrato inteligente de próxima generación y una plataforma de aplicaciones descentralizadas". Éste propone integrar un lenguaje Turing completo en el sistema de scripting de Bitcoin como mejora del protocolo, aunque el concepto es una idea original de Sergio Demian Lerner que desarrolla en su tesis. Linda Xie lo explica en su artículo para blog.coinbase.com [Xie18] de forma sencilla:

> *"Ethereum is different than Bitcoin in that it allows for smart contracts which can be described as highly programmable digital money. Imagine automatically sending money from one person to another but only when a certain set of conditions are met. For example an individual wants to purchase a home from another person. Traditionally there are multiple third parties involved in the exchange including lawyers and escrow agents which makes the process unnecessarily slow and expensive. With Ethereum, a piece of code could automatically transfer the home ownership to the buyer and the funds to the seller after a deal is agreed upon without needing a third party to execute on their behalf."*

Vitalik y su equipo llevaron a cabo con éxito una ICO (*Initial Coin Offering* o campaña de crowdfunding) para iniciar Ethereum desde julio a agosto de 2014. El sistema salió definitivamente el 30 de julio 2015 como una plataforma open source, en la que cualquier desarrollador puede crear y publicar aplicaciones distribuidas que ejecuten contratos inteligentes. Ethereum también provee una criptomoneda llamada ether (ETH) la cual se puede intercambiar entre cuentas y también es utilizada para compensar a los nodos participantes por los cálculos realizados. No es el propósito de este documento explicar en detalle el funcionamiento de Ethereum, no obstante este artículo de Preethi Kasireddy que puede encontrarse en las referencias lo explica.

La actualización de Homestead fue el primer *hard fork* planificado de la red Ethereum y se implementó el 14 de mayo de 2016 con el número de bloque 1.150.000. En general, la actualización de Homestead incluyó tres mejoras importantes en Ethereum. Primero, eliminó la funcionalidad del *canary contract*, eliminando ese punto de centralización en la red. En segundo lugar, introdujo nuevos códigos en Solidity, el lenguaje de programación utilizado en Ethereum. Por último, presentó la *wallet* Mist, que permitía a los usuarios mantener y transaccionar ETH e implementar contratos inteligentes.

En 2016, una organización autónoma descentralizada llamada The DAO recaudó 150 millones de dólares. En junio, el DAO fue hackeado por un pirata informático desconocido que robó $50 millones en ETH. La comunidad de Ethereum en general decidió bifurcar la cadena para restaurar los fondos a sus billeteras originales y parchear la vulnerabilidad. Sin embargo, el *hard fork* fue polémico, y algunos en la comunidad Ethereum continuaron minando y realizando transacciones en la cadena original. La cadena original, con el ether robado no devuelto, se convirtió

en Ethereum Classic. La mayoría de miembros de la comunidad y los desarrolladores principales continuaron trabajando fuera de la cadena bifurcada, con el ETH robado devuelto a sus propietarios originales, que es lo que ahora conocemos como la cadena de bloques Ethereum.

Desde que Ethereum fue lanzado, han surgido muchas otras *blockchains* que han enfocado su desarrollo de distintas formas. El primer requisito a valorar a la hora de elegir qué plataforma usar es saber si se puede utilizar una *blockchain* pública o *permissionless* (es decir cualquier usuario puede entrar a formar parte de la red y participar en ella) o se requiere una *blockchain* privada o *permissioned* como es en el caso de una empresa, en la que es indispensable mantener la total privacidad de sus operaciones y definir qué nodos pueden unirse a la red. Además, otros factores a tener en cuenta son la escalabilidad (es decir, la capacidad de soportar un aumento considerable de la carga de la red); la velocidad de las transacciones (Bitcoin, por ejemplo, tarda unos 10 minutos o más para verificar y añadir un bloque de transacciones, mientras que el promedio de Ethereum apunta a alrededor de 12 segundos) y los posibles protocolos que pueden ser utilizados para alcanzar el consenso entre los nodos, como los definidos en la siguiente infografía:

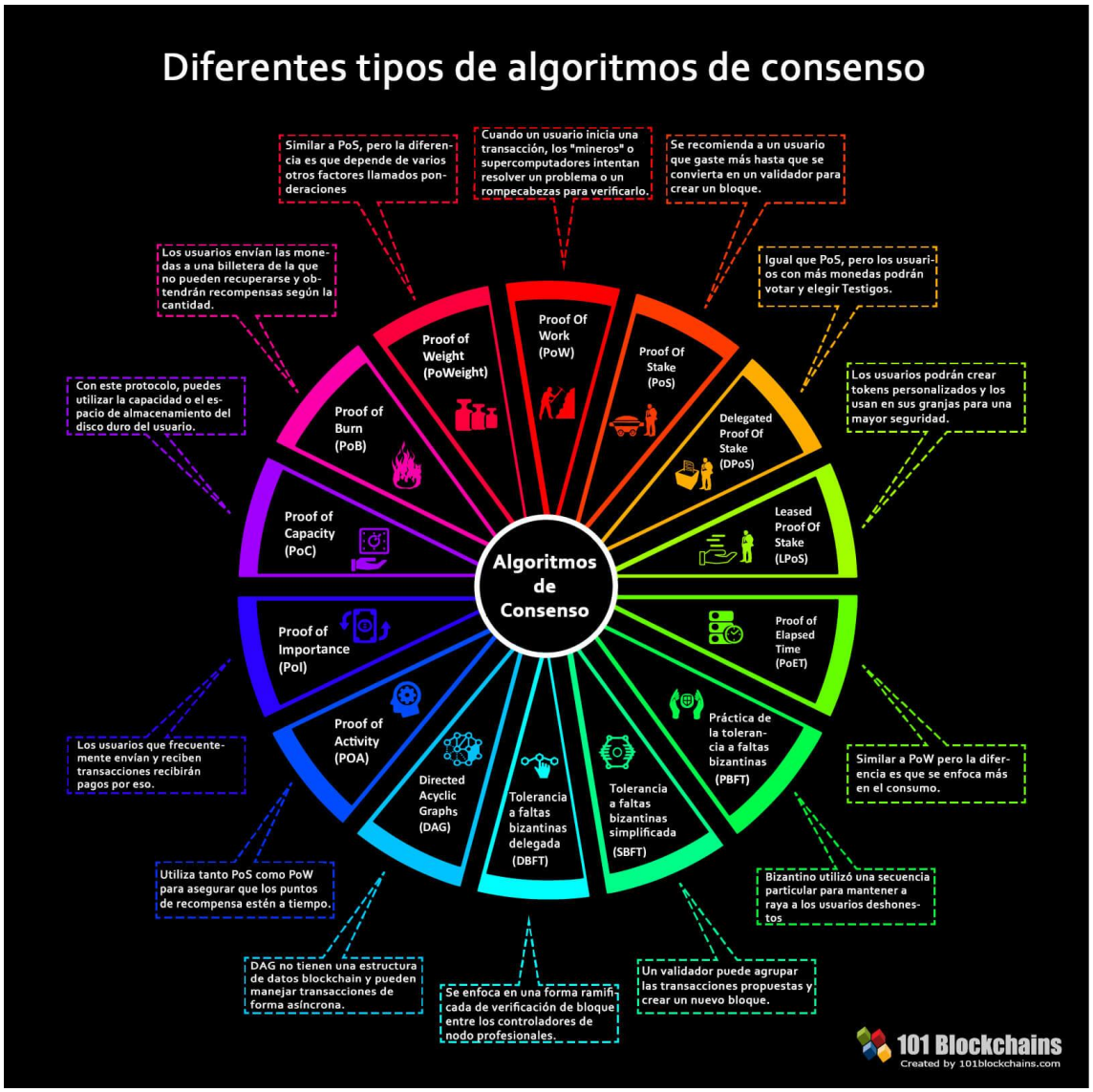

Figura 1. [101 Blockchains20]

Otros aspectos muy a tener en cuenta a la hora de elegir son la actividad a la que está enfocado el proyecto, la popularidad de la plataforma en base a sus contribuidores y valoraciones de expertos, los lenguajes que soporta, la cantidad y calidad de documentación disponible y si se trata de una plataforma gratuita o de pago. Entre las más conocidas se encuentran las de la familia de Hyperledger (Sawtooth, Fabric, Iroha, etc.), EOS, NEO, TRON, IOST, Hedera Hashgraph, OpenChain, Corda, Quorum y Ethereum Enterprise Alliance. No es el propósito de este documento hacer una explicación detallada de las características de cada una; no obstante, la bibliografía contiene información acerca de cada una de ellas así como documentos que las comparan y analizan de manera más exhaustiva.

2.4. Aplicaciones descentralizadas

Una aplicación descentralizada (DApp, dApp, Dapp o dapp) es una aplicación informática que es almacenada y ejecutada por un sistema *blockchain*. Las DApps reducen el impacto de los terceros en las funciones de una aplicación. Éstas conectan

a proveedores y usuarios sin la ayuda de intermediarios, contrariamente a las aplicaciones regulares. Las DApps basadas en *blockchain* son inmunes a la censura, sin necesidad de gastos adicionales y son más propensas a mantener el funcionamiento durante varios ataques. Éstas están mejorando los procesos de pago, las credenciales de usuario y son de confianza, debido a su código open source y los registros públicos de las transacciones.

Acorde al artículo publicado por Cointelegraph [Avan-Nomayo20] las transacciones de las DApp en cadena en 2019 ascendieron a 23.000 millones de dólares con más de 1.900 nuevas aplicaciones añadidas. Sin embargo, Dapp.com sitúa el número de nuevas DApps añadidas para 2019 en alrededor de 1.450, un ligero descenso con respecto a las 1.500 registradas en 2018. Las cifras de Dapp.com muestran que más de 1.300 DApps fueron abandonadas en 2019. Según la plataforma de análisis, una DApp abandonada es aquella en la que no se producen transacciones en un plazo de 30 días.

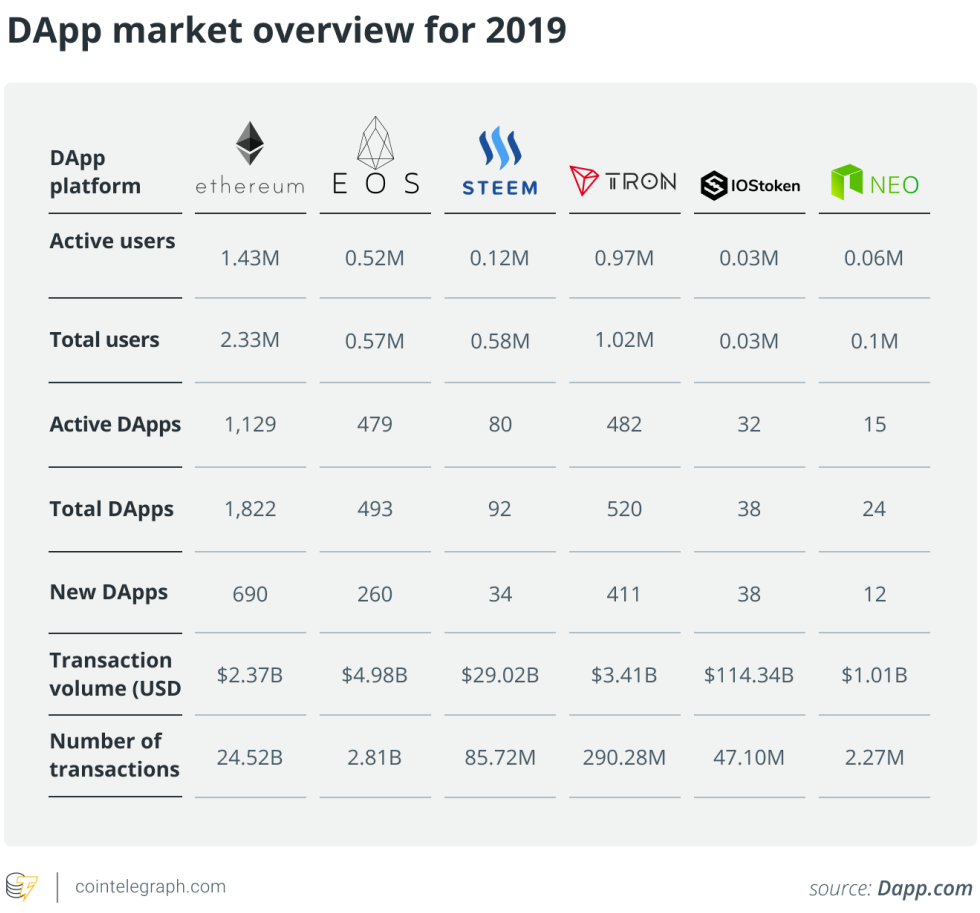

Figura 2. [Avan-Nomayo20]

La retención de los usuarios sigue siendo uno de los principales problemas de las plataformas de DApp. El mismo artículo cita:

> *"El número de usuarios activos en las Dapps en el 2019 se ha duplicado en comparación con 2018, de 1,48M a 3,11M. Hay 2,77 millones de nuevos usuarios que experimentaron aplicaciones descentralizadas. La retención de usuarios sigue siendo un problema para las dapps - sólo quedan 348K usuarios antiguos activos en 2019, lo que representa el 11% de todos los usuarios activos".*

Esto se explica principalmente a que en las DApps basadas en Ethereum es el usuario final quien ha de pagar las tasas de gas (unidad de medida utilizada hasta ahora para ejecutar una transacción en la *blockchain*). En los periodos en el que la red se encuentra congestionada estos costos pueden volverse impracticables para los usuarios causando un flujo de salida significativo. Existen algunas soluciones como por ejemplo el uso de *sidechains* o cadenas laterales.

Otro motivo relevante que afecta a la captación y retención de usuarios es la facilidad de uso. Este constituye un obstáculo para llevar las DApps a las masas. El último artículo referenciado también señala en relación a esto que:

> "Las DApps y los programas web3, en general, también tienen problemas de compatibilidad con los smartphones cuyos navegadores representan el mayor porcentaje de tráfico web. A diferencia de los ordenadores de sobremesa, los navegadores de los smartphones para Android e iOS no tienen acceso a las actualizaciones de la web3 como extensiones y plugins. En una conversación con Cointelegraph, Benjamin Cheng, un alto ejecutivo de la empresa emisora de stablecoins de forma algorítmica Timvi, destacó la necesidad de contar con DApps más fáciles de usar. Según Cheng:
>
> > *"Los usuarios se ocupan de cuestiones tecnológicas como la espera del procesamiento de transacciones, la reorganización de la cadena, etc. Las tecnologías blockchain se encuentran en la etapa de 'geek', aún no aptas para el usuario común, con suerte, esto cambiará con el advenimiento de las soluciones de Nivel 2 (soluciones de Capa 2). Las herramientas para interactuar con la blockchain tampoco son fáciles de usar. Necesitamos gente como Steve Jobs para hacer que la tecnología sea conveniente y fácil para el usuario".*
>
> El entorno para los usuarios de las DApps debe ser familiar para la gente común, lo que significa centrar los esfuerzos en simplificar las interfaces de usuario de estas aplicaciones descentralizadas. Las DApps no pueden alcanzar la escala si su base de usuarios consiste en una micronivel dominada sólo por entusiastas de la tecnología *blockchain* y de la web3."

Mención aparte merecen las DeFi, sistemas monetarios y financieros descentralizados construidos sobre cadenas públicas. Abarcan préstamos, pagos, DEX y criptos derivados, entre otros. Sus defensores afirman que su objetivo es crear

rampas de acceso fáciles para que los económicamente privados de derechos y sub-bancarizados, por ejemplo, tengan acceso a los servicios financieros globales usando protocolos blockchain resistentes a la censura. De acuerdo con el informe de Dapp.com, las aplicaciones enfocadas en DeFi, como las DApps de préstamos, experimentaron un crecimiento significativo de usuarios en 2019. Otro extracto del informe de Dapp.com dice:

> *"Los servicios financieros (por ejemplo, las DApps de préstamos) tienen el crecimiento de usuarios más impresionante en 2019. El número de usuarios de DApps financieras ha aumentado en un 610%, y el volumen de transacciones ha aumentado en un 251%".*

Figura 3. [Avan-Nomayo20]

2.5. Transformación digital

De acuerdo con el artículo publicado por ittrends.es [Blockchain | IT Trends19] los expertos de IDC han detectado un fuerte crecimiento de las soluciones basadas en *blockchain*, que están sentando las bases para crear un mercado potente. Su pronóstico indica que para 2023 este podría alcanzar un valor de unos 15.90 millones de dólares gracias a la adopción en diferentes casos de uso empresariales. También existen detractores que afirman que se está sobreestimando su utilidad, sin embargo muchos sectores entre los que se encuentran la banca, las finanzas y las administraciones públicas están impulsando su uso y desarrollo. Según el artículo citado:

> "Pero parece que la utilidad de la tecnología de cadenas de bloques ya se está demostrando, generando un mercado en crecimiento. Según la última actualización de la guía de gasto de Blockchain de IDC, se prevé

> que el gasto mundial en soluciones Blockchain alcanzará casi 15.900 millones de dólares en 2023. Esto supondrá un crecimiento a una tasa interanual compuesta (CAGR) del 60,2% entre 2018 y 2023. Partiendo de los 2.700 millones de dólares previstos para 2019, este año ya se vería un aumento del 80% con respecto a 2018, cuando la mayoría de soluciones basadas en esta tecnología todavía estaban en desarrollo."

Ciertamente existe incertidumbre, especialmente en las áreas de gobierno y regulación, no obstante, el crecimiento y la adopción de blockchain por parte de las empresas se está acelerando a medida que se entienden los beneficios de usar blockchain para aumentar la eficiencia y mejorar los procesos. El mismo artículo añade que el gasto global en soluciones Blockchain estará liderado por la industria bancaria, que representará en torno a un 30% del total a lo largo de todo el período pronóstico. Le seguirán la fabricación discreta y la fabricación por procesos, que tendrán una participación conjunta del 20% en el mercado global. Esta última será la que más rápidamente aumentará el gasto a lo largo de estos años, para la cual se prevé un crecimiento a una tasa interanual compuesta (CAGR) del 68,8%, lo que le llevará a convertirse en la segunda industria que más gastará en Blockchain para 2023. El artículo también señala que los expertos de IDC pronostican que la fabricación discreta, los servicios profesionales, la venta minorista y los servicios públicos también crecerán por encima de la media general del mercado y que otro punto a remarcar de este informe es que la industria bancaria impulsará el gasto en los dos casos de uso más importantes, que serán los pagos y acuerdos transfronterizos y finanzas comerciales, y los acuerdos posteriores a las transacciones y transferencias. La siguiente infografía muestra un resumen de los casos de uso más relevantes en los que la tecnología *blockchain* supone una disrupción más que significativa:

Figura 4. [Singh19]

2.6. Propuesta

La propuesta de este trabajo es explorar esta novedosa tecnología y aprender a utilizarla mediante una aplicación. Este ejercicio nos servirá primeramente para entender la tecnología y diseñar la solución a nivel de arquitectura y posteriormente para conocer y trabajar los conceptos básicos de su implementación. Para ello tomaremos como contexto el caso de las cadenas de suministro, ya que nos permite profundizar más en un caso particular que nos presenta diferentes roles con distintos requisitos. Esto nos permitirá comprender mejor las necesidades de los usuarios y los procesos, así como observar de cerca algunas de las aplicaciones de la tecnología *blockchain* para éstos.

Además de estudiar una solución *blockchain* empresarial, este TFG integra de manera explícita esta herramienta con otras tecnologías como IoT, Android (mobile) y NoSQL lo que nos permite abarcar no sólo su propio uso sino que nos presenta su uso en conjunto con otros dispositivos y herramientas disponibles en el mercado. Otra aportación que considero que ofrece este trabajo es el hecho de estar desarrollado con Hyperledger Sawtooth en C# ya que la documentación y proyectos disponibles en el momento de su realización es bastante limitada y en inglés la inmensa mayoría.

3. Análisis

Una cadena de suministro está compuesta por el conjunto de actividades, instalaciones y medios de distribución necesarios en la acción de satisfacer las necesidades de suministro. En otras palabras, es una función estratégica y logística que involucra todas las operaciones que son indispensables para que una mercancía logre llegar al cliente final en óptimas condiciones. La gestión de la misma repercute tanto en los procesos clave de la empresa relacionados con costes, disponibilidad y calidad como en el propio valor que ofrece el producto o servicio al cliente final. Para satisfacer las necesidades de todas las partes de la mejor manera posible, la gestión de la cadena de suministros tiene como objetivos:

- Entregar los bienes y servicios a tiempo.
- Evitar las pérdidas o mermas innecesarias.
- Optimizar los tiempos de distribución.
- Manejo adecuado de inventarios y almacenes.
- Establecer canales de comunicación y coordinación adecuados.
- Hacer frente a cambios imprevistos en la demanda, oferta u otras condiciones.

En una cadena de suministro se involucran distintos roles y entidades como son los proveedores de distintos niveles, los almacenes de Materia Prima, la línea de producción, almacenes de Productos Terminados, canales de distribución, mayoristas, minoristas y el cliente final.

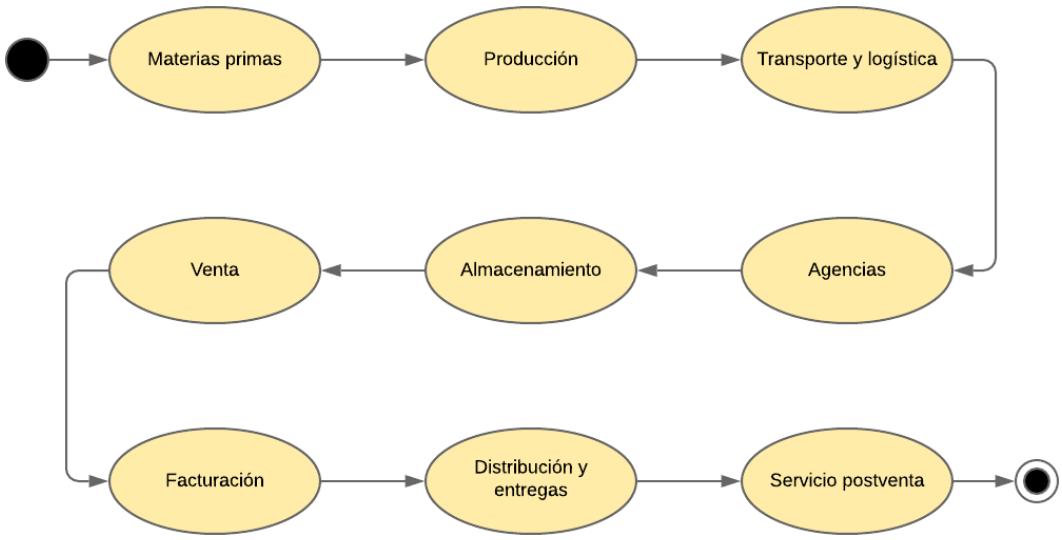

Figura 5.

Además de todas estas partes, existen otras entidades, leyes y reglas que han de aplicarse en el propio proceso. Por ejemplo, en el caso del comercio internacional de alimentos se verán involucrados laboratorios para las pruebas exigidas por el país importador que confirmen que el producto tiene unas características acordes a lo permitido por las leyes sanitarias; el propio gobierno con procesos burocráticos, tanto en materia de salud como en el ámbito económico y fiscal; entidades de crédito, aseguradoras y servicios aduaneros.

La cadena de suministro varía su forma y las actividades que incluye dependen del bien o servicio que estemos estudiando pero, en cualquier caso, independientemente de esto, existen ciertas tareas que son comunes y necesarias para optimizar el resultado final, a saber:

- Planificación.
- Administración de existencias.
- Procesamiento de órdenes de compra.
- Traslados y despacho.
- Seguimiento y control de imprevistos.
- Servicio al cliente.
- Administración de garantías.
- Procesamiento de pagos.

La oportunidad que ofrecen la *blockchain* y los contratos inteligentes (junto con otras tecnologías como el *big data*, el *machine learning* y la inteligencia empresarial) permitirán una administración más ágil y eficiente de estas cadenas, además de una mayor transparencia, seguridad y confianza, aspectos que son fundamentales tanto en las transacciones en sí mismas como en el servicio/producto que se adquiere. Continuemos con el ejemplo anterior para ilustrar de qué manera puede aplicarse el *blockchain* a las actividades mencionadas. Supongamos que tenemos los siguientes roles:

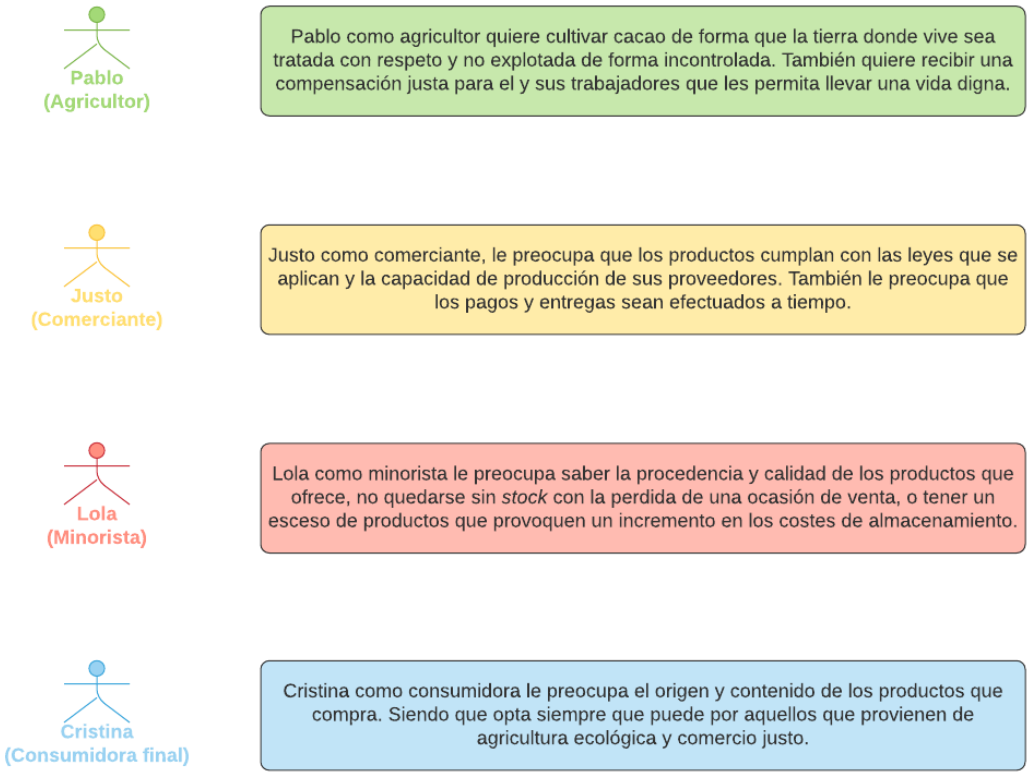

Figura 6.

Gracias a la *blockchain*, Pablo puede acreditar que sus bienes han sido producidos acorde a los estándares de agricultura ecológica y comercio justo, aportando valor al producto en sí y atendiendo sus preocupaciones. Pablo y otros productores también pueden hacerse más visibles al mercado registrándose en un marketplace basado en *blockchain* donde comerciantes como Justo buscan proveedores. En su perfil pueden acreditar su identidad empresarial (probando su existencia en el registro mercantil de su país en base a documentos legales firmados digitalmente); los productos que ofrecen junto con pruebas y documentos de auditores externos que certifiquen la mercancía; las licencias, permisos, etc. para así dar a sus clientes una mayor confianza y crear oportunidades de venta con mayor facilidad.

Justo se ve beneficiado en gran medida por diferentes razones. El uso de los contratos inteligentes permite automatizar los trámites necesarios para las transacciones y transporte de la mercancía. Dependiendo del tipo de mercancía con la que operamos, existen distintos procedimientos que deben ser llevados a cabo. Al realizar una operación comercial internacional debemos tener en cuenta distintos factores. Uno de ellos es la negociación de qué se vende, es decir tanto el producto como sus características, cuánto se vende, a qué precio y bajo qué condiciones.

Para nuestro caso del cacao, Justo tiene un pedido de 5 contenedores de 40 pies, de los cuales 2 de ellos son en paquetes de 5Kg y los otros 3 en paquetes de 1Kg. De estos 3 últimos, uno lleva cacao de tipo A y los otros dos del tipo B. Además, Justo trabaja con varios productores pequeños y medianos y a veces algunos de ellos no pueden hacer frente a toda la demanda que reciben por sí solos, por lo que se puede dar el escenario en el que para una misma operación se requieran varios

proveedores. Todos estos factores influyen en la logística y los costes. Para su compra es necesario negociar el precio así como las condiciones de pago. Los principales medios de pago internacional usados hoy en todo el mundo son pago por adelantado, carta de crédito, remesa documentaria, cuenta abierta y consignación. Por lo tanto, puede que el pago se realice 100% por adelantado, o quizá el 50% por adelantado y el otro 50% a 6 meses. Esto varía enormemente dependiendo del sector, la estrategia logística y la capacidad financiera de cada empresa. Es posible también que haya entidades de crédito involucradas en las transacciones. Esta negociación puede ser perfectamente plasmada en un contrato inteligente que ejecute las órdenes de envío y transacción de pago acorde a lo negociado.

Los *Incoterms* (palabra derivada del sigloide en lengua inglesa ***in**ternational **co**mmercial **terms)*** no son de uso obligatorio, y, por tanto, empresas y particulares tienen libertad para pactar sus propias condiciones. No obstante, su uso está muy generalizado puesto que facilitan acordar las condiciones bajo un marco común. Estas reglas describen las condiciones de entrega de las mercancías y productos en el comercio global, por lo que deben tenerse muy en cuenta a la hora de diseñar los contratos de compraventa. Una red *blockchain* de dispositivos IoT y móviles pueden determinar el momento exacto en el que la transferencia de la custodia se hace efectiva.

Supongamos que el acuerdo se ha hecho en términos FOB en el puerto de Abiyán en Costa de Marfil (*Free On Board (named loading port)* → Libre a bordo (puerto de carga convenido)). En este caso, el vendedor entrega la mercancía sobre el buque y el comprador se hace cargo de designar y reservar el transporte principal (buque). Es decir, en el momento en que la mercancía está cargada en la embarcación, su estado se actualiza en la *blockchain* y automáticamente el comprador tiene la custodia de ésta. En cambio, si el acuerdo es en términos CFR al puerto de Valencia (*Cost and Freight (named destination port)* → coste y flete (puerto de destino convenido)), el vendedor se hace cargo de todos los costes, incluido el transporte principal, hasta que la mercancía llegue al puerto de destino. Sin embargo, el riesgo se transfiere al comprador en el momento que la mercancía se encuentra cargada en el buque, en el país de origen. Si suponemos ahora que el vendedor quiere incluir un seguro, estamos entonces aplicando en esta ocasión CIF al puerto de Valencia (*Cost, Insurance and Freight (named destination port)* → coste, seguro y flete (puerto de destino convenido). Este seguro puede ser proporcionado por una compañía aseguradora cuyo justificante puede ser incluido en los términos del contrato inteligente así como lo pueden estar la documentación de aduanas, o contratos con navieras, y almacenes logísticos. El uso de dispositivos IoT y GPS pueden ser de gran utilidad en la automatización de estas tareas.

Llegados a este punto hay que remarcar, pues es de gran importancia, la correlación de identificadores. Cada entidad registra el ítem en cuestión con un código distinto, sin embargo para la *blockchain* estamos haciendo referencia al mismo ítem. El fabricante puede utilizar el SKU (*stock-keep unit*, referencia de almacén), que son generados por cada empresa o vendedor, y el minorista tendrá otro SKU distinto para ese

mismo ítem. La referencia es un identificador único e interno utilizado por las empresas con el objeto de identificar de forma unívoca un elemento en el inventario físico o financiero. Las referencias no siempre identifican un ítem físico, también pueden identificar a entidades inmateriales pero financieras y facturables. La identificación de elementos con referencia se hace de acuerdo al criterio interno de la empresa: asignando números a productos de forma secuencial o siguiendo algún tipo de estructura significativa. Así pues, la identificación con *SKU* difiere de otros métodos de identificación de productos que sí son controlados como regulaciones y normativas realizadas por organizaciones públicas o privadas (asociaciones de comercios y fabricantes). Otros métodos de seguimiento de entidades, con diferentes regulaciones, son el código universal de producto (UPC), el número de artículo europeo (EAN) y el número global de artículo comercial (GTIN).

Estos códigos no son solamente importantes para la identificación del producto, algunos nos proporcionan información sobre la legislación de un país respecto a un tipo de producto. Este es el caso del *Harmonized Commodity Description and Coding System*, también conocido como *Harmonized System (HS)*, que proporciona un sistema de clasificación de mercancías objeto de comercio internacional. Es decir, es una nomenclatura internacional para la clasificación de productos que permite a los países participantes clasificar los bienes comercializados sobre una base común para fines aduaneros; y permite conocer los impuestos, derechos, regulaciones no arancelarias, etc., aplicables a cada producto.

Un punto más a tener en cuenta es el tipo de cambio, que es el precio de la moneda de un país expresado en términos de la moneda de otro país. En otras palabras, la tarifa por la cual una moneda puede cambiarse por otra. Es importante porque facilita el comercio internacional de bienes y servicios y la transferencia de fondos entre países. También permite la comparación de precios de productos similares en diferentes países. En general, la diferencia de precios entre productos similares determina qué productos se van a comerciar y a que país se van a enviar. Es importante, por lo tanto, saber en qué moneda van a realizarse los pagos y los cobros, qué tipo de cambio se va aplicar y en qué momento, pues no será el mismo tipo en el momento de la firma del contrato de compraventa que en el momento en que la transacción es efectuada, o si el pago está dividido podría aplicarse un tipo distinto. Es un dato muy importante para conocer la rentabilidad que nos va a dar una transacción comercial de carácter internacional.

Por otro lado, Lola está tranquila ya que puede consultar la procedencia y documentación legal asociada al producto y le es más fácil administrar las existencias y procesar órdenes de compra. La aplicación de otras tecnologías emergentes relacionadas con la inteligencia empresarial permiten hacer predicciones de demanda y tomar decisiones estratégicas, optimizando así aún más la gestión de los recursos.

Por su parte, Cristina está satisfecha porque compra productos acordes a sus principios como consumidora y tiene la garantía del origen y tratamiento de los bienes que compra. Otros filtros que Cristina puede aplicar (utilizando motores de búsqueda y bases de datos NoSQL) son por alérgenos, comercio de proximidad, denominación de origen, etc.

Aunque queda fuera del alcance de este ejemplo también hay que resaltar la aplicación para las propias empresas de transporte que, a su vez, pueden tener un mejor control de las mercancías y del empleo de sus flotas.

Una manera de ver este proceso es como una cadena de custodia de un bien cuyo estado y propiedades se encuentran asociados al mismo. Podemos representar

la cadena con el siguiente diagrama de actividades. En él se muestra quién se halla en custodia del bien y qué operaciones se le permite llevar a cabo.

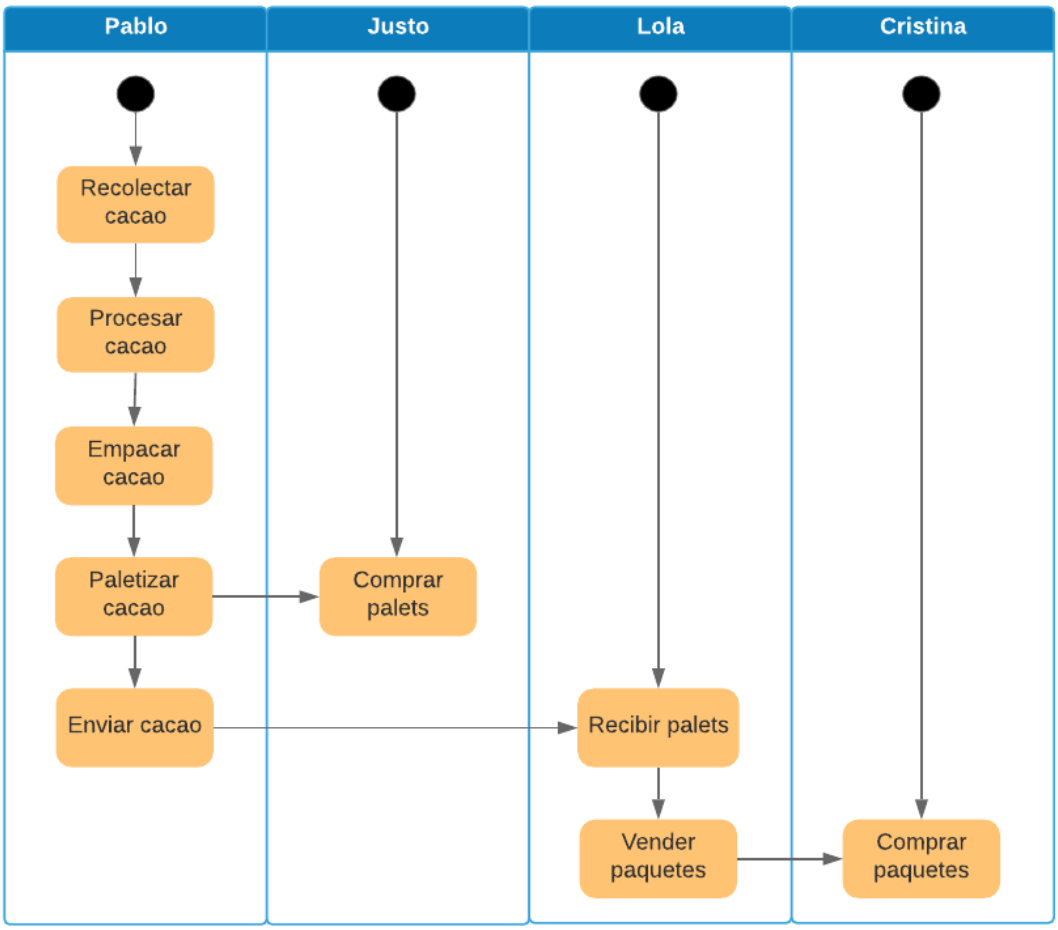

Figura 7.

Vamos ahora a representarlo de forma secuencial de manera que visualicemos la interacción del conjunto de los objetos a través del tiempo:

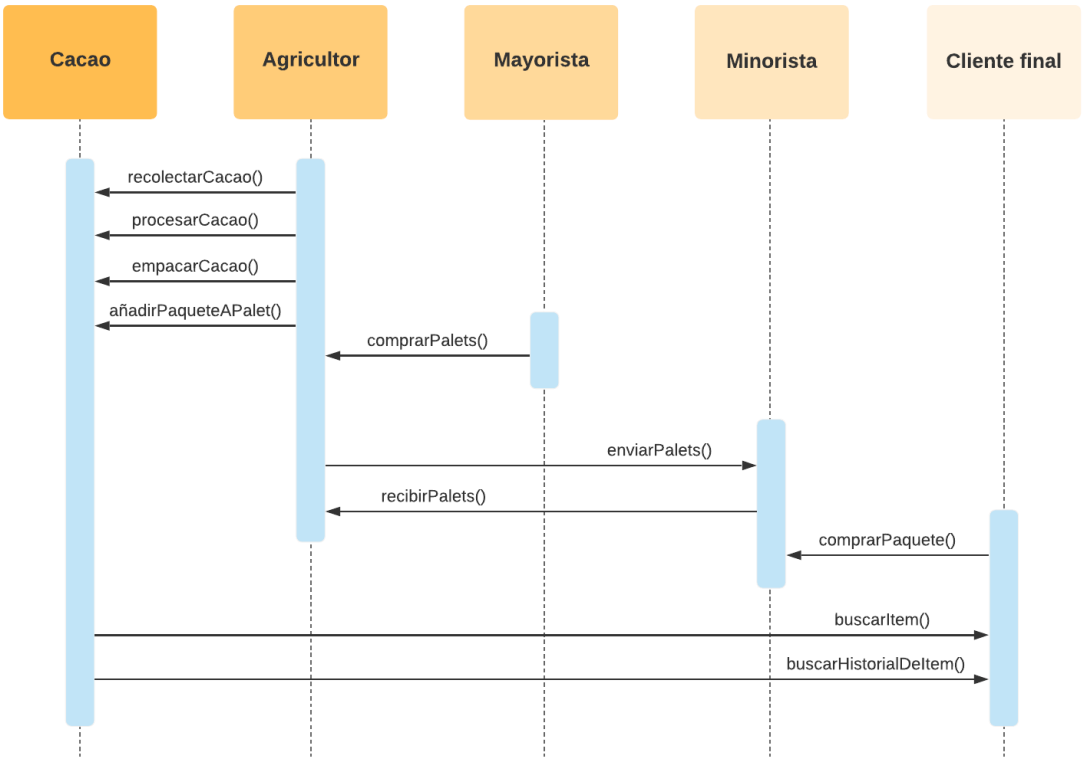

Figura 8.

El siguiente diagrama nos ayuda a comprender mejor los estados y condiciones bajo los que opera nuestro sistema. Con él vemos qué roles pueden llevar a cabo qué acciones en función de qué condiciones y el resultado de estas acciones en el estado del ítem.

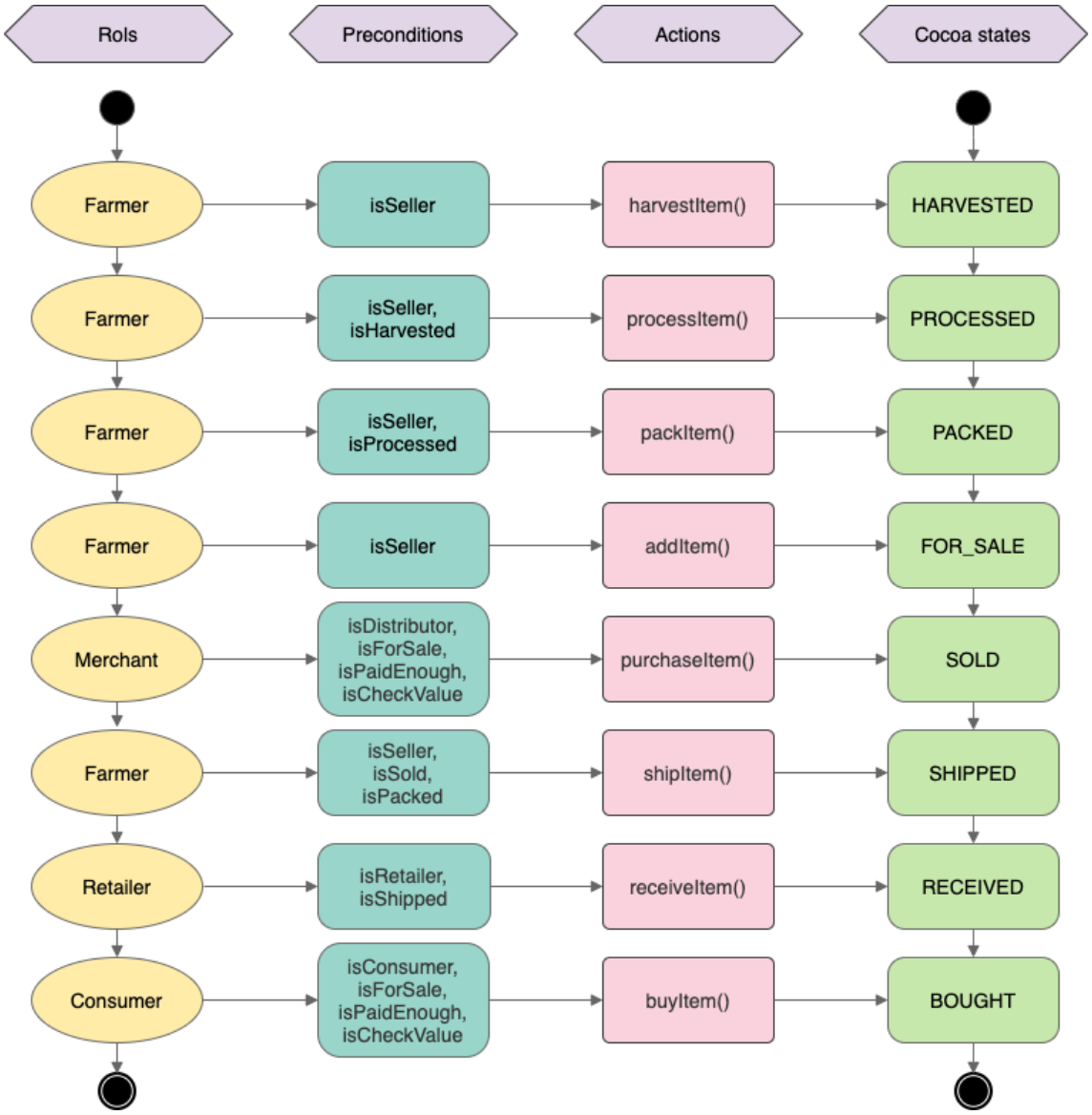

Figura 9.

Esto nos lleva a prestar atención a un aspecto de máxima importancia en todo el proceso y, por tanto, en el diseño del software: los permisos. Para todos ellos existe información que quieren mostrar a todo el público, información que sólo algunas entidades deben poder ver, e información que es totalmente privada de la empresa.

4. Diseño de la solución

En este apartado se detalla el diseño propuesto, comenzando por abordar los aspectos arquitectónicos del mismo y siguiendo por el diseño detallado.

4.1. Arquitectura del Sistema

Nuestra solución tiene dos conjuntos de componentes que conforman el modelo de escritura y el modelo de lectura que sigue el patrón *Command–Query*

Separation (CQS) (en castellano “separación de comandos y consultas”), que es un principio de la programación orientada a objetos ideado por Bertrand Meyer como parte de su trabajo pionero sobre el lenguaje de programación Eiffel. El principio afirma que cada método debe ser un comando que realiza una acción, o una consulta que devuelve datos al llamante, pero no ambos. En otras palabras, hacer una pregunta no debe cambiar la respuesta. Más formalmente, los métodos deben devolver un valor sólo si son referencialmente transparentes y, por tanto, no poseen efectos colaterales.

En Hyperledger Sawtooth, el flujo de trabajo de la aplicación es el siguiente:

1. El usuario realiza una acción desde su dispositivo móvil, como inicializar un valor (por ejemplo, "establecer A en 1"). Eli cliente crea una transacción de la siguiente manera:
    a. Lo codifica (serializa) en una carga útil, que podría ser tan simple como {"A": 1}
    b. Envuelve esa carga útil en una transacción firmada y luego en un lote firmado que contiene una o más transacciones
    c. Por último envía el lote al validador a través de una llamada a la API REST.
2. La API REST transmite el lote al validador.
3. El validador confirma que el lote y la transacción son válidos.
4. El procesador de transacciones recibe la transacción y:
    a. Verifica al firmante
    b. Desenvuelve la transacción y decodifica la carga útil
    c. Verifica que la acción sea válida (por ejemplo, asegura que A se pueda establecer en 1)
5. Modifica el estado de una manera que satisfaga la acción (por ejemplo, la dirección ... 000000a se convierte en 1)

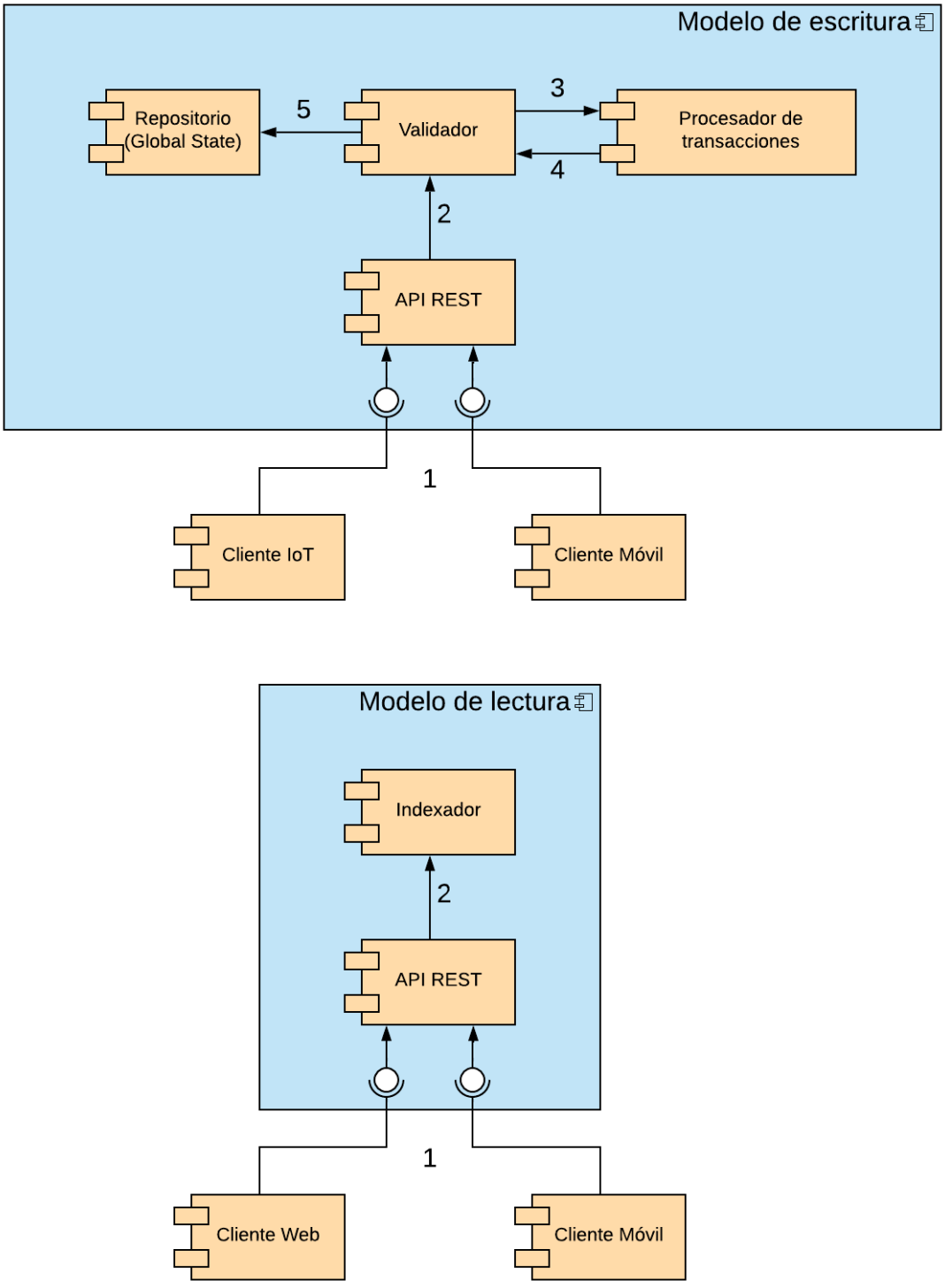

Figura 10.

Una aplicación Sawtooth puede implementar un *listener* de eventos y una base de datos de informes separada para almacenar datos de eventos de estado. Hay que tener en cuenta que esta base de datos de informes fuera de la cadena es independiente de la base de datos de estado en cada nodo de validación. Esta arquitectura permite que pueda hacerse un despliegue bajo demanda del modelo de lectura que represente el estado a un momento específico.

Un cambio en la cadena de bloques desencadena un evento de Sawtooth. Los eventos se emiten cuando se confirma un bloque. Cada aplicación puede definir sus propios eventos. Por convención, los nombres de eventos incluyen el nombre de la aplicación como prefijo. Sawtooth tiene dos eventos principales (identificados con el prefijo 'sawtooth'):

- **sawtooth/commit-block**: Contiene información sobre el bloque comprometido: el ID del bloque, el número, el hash raíz del estado y el ID del bloque anterior
- **sawtooth/state-delta**: Contiene todos los cambios de estado que ocurrieron para un bloque en una dirección específica.

En este modelo, un cliente de suscripción de eventos se suscribe tanto a los eventos Sawtooth state-delta como a los eventos block-commit, luego usa los datos reportados en estos eventos para mantener una copia local del estado *blockchain*, lo que permite una consulta rápida de los datos del estado global.

Aunque la obtención de datos de una base de datos de informes funciona bien con las interfaces RESTful HTTP/JSON típicas, el envío de actualizaciones a la cadena de bloques mediante una API REST requiere una consideración especial para el modelo de firma. Estos dos modelos se han utilizado en el pasado:

- ***Client signing model.***

El flujo de un modelo de firma de cliente es el siguiente:

1. El cliente crea transacciones y lotes y los firma con la clave privada del usuario.
2. El cliente serializa los lotes y los envía a la ruta POST / lotes única en la API REST.
3. La API REST envía los lotes directamente al validador.

Pros	➔ La identidad del usuario es completamente verificable ya que las transacciones se firman localmente utilizando la clave privada del usuario. ➔ Incluso si el servidor está completamente comprometido, no hay forma de falsificar transacciones de un determinado usuario.
Contras	➔ Los *endpoints* RESTful para enviar transacciones no son posibles ya que el servidor solo actúa como intermediario, manteniendo una conexión y enviando datos serializados al validador. ➔ Cada cliente debe implementar la función de creación y firma de transacciones.

Tabla 1.

Este modelo aprovecha por completo la verificación de identidad de *blockchain*, dejando la responsabilidad del usuario para proteger su propia

clave privada. Ejemplos de este modelo incluyen Sawtooth Supply Chain, así como la API REST de Sawtooth.

- ***Server signing model.***

El flujo de un modelo de firma de servidor es el siguiente:

1. El cliente envía solicitudes de actualización como objetos JSON a interfaces RESTful tradicionales.
2. El servidor crea y firma transacciones basadas en el JSON enviado.
3. El servidor maneja el procesamiento por lotes/serialización y envía la transacción al validador.

Pros	➔ El servidor mantiene interfaces RESTful, lo que significa que interactuar con el servidor no es diferente a si el servidor estuviera respaldado por una base de datos tradicional. ➔ El cliente no tiene que manejar ninguna firma o creación de transacciones.
Contras	➔ Debido a que el servidor está a cargo de firmar todas las transacciones, la garantía de verificación de identidad de *blockchain* está algo comprometida. ➔ Un enfoque ingenuo del modelo de firma del servidor firmaría todas las transacciones desde una única clave propiedad del servidor. Esto anularía todas las ventajas de verificación de identidad de *blockchain*, ya que el servidor sería la única fuente de verdad, al igual que con una base de datos tradicional. Esta desventaja se puede mitigar generando un par de claves pública/privada para cada usuario en el servidor y almacenando la clave privada (encriptada) en un servicio de custodia de claves. Luego, el servidor puede recuperar y descifrar la clave privada de un usuario autenticado y firmar transacciones en su nombre. Si bien esto es mejor que el enfoque anterior, sigue siendo tan seguro como los mecanismos de seguridad del servidor.

Tabla 2.

En nuestro diseño hemos optado por el primero ya que podemos verificar la identidad y no es posible falsear las transacciones, además nos ahorra el trabajo de implementar y mantener el servidor de claves.

Es muy importante también a la hora de diseñar tanto a alto como a bajo nivel el tener en cuenta los principios SOLID. SOLID es un acrónimo mnemónico que representa cinco principios básicos de la programación orientada a objetos y el diseño. Cuando estos principios se aplican en conjunto es más probable que un desarrollador cree un sistema que sea fácil de mantener y ampliar con el tiempo. Los principios

SOLID son guías que pueden ser aplicadas en el desarrollo de software para eliminar malos diseños provocando que el programador tenga que refactorizar el código fuente hasta que sea legible y extensible. Puede ser utilizado con el desarrollo guiado por pruebas, y forma parte de la estrategia global del desarrollo ágil de software y desarrollo adaptativo de software.

Los principios son los que siguen:

- ***Single Responsibility Principle*** (Principio de responsabilidad única). Enuncia la noción de que un objeto solo debería tener una única responsabilidad.
- ***Open/Closed Principle*** (Principio de abierto/cerrado). Enuncia la noción de que las "entidades de software deben estar abiertas para su extensión, pero cerradas para su modificación".
- ***Liskov Substitution Principle*** (Principio de sustitución de Liskov). Enuncia la noción de que los "objetos de un programa deberían ser reemplazables por instancias de sus subtipos sin alterar el correcto funcionamiento del programa".
- ***Interface Segregation Principle*** (Principio de segregación de la interfaz). Enuncia la noción de que "muchas interfaces cliente específicas son mejores que una interfaz de propósito general".
- ***Dependency Inversion Principle*** (Principio de inversión de la dependencia). Enuncia la noción de que se debe "depender de abstracciones, no depender de implementaciones".

La distribución de módulos en función de los modelos de lectura y escritura es una aplicación a alto nivel del SRP. Nuestra arquitectura permite la extensión pero no la modificación por lo que también cumple con el OCP. La implementación de las interfaces nos asegura complir con el LSP. Al separar las interfaces de dispositivos IoT de las interfaces móviles estamos cumpliendo con el ISP. Y por último, la Inyección de Dependencias es uno de los métodos que siguen el DIP, y es esto lo que proponemos utilizar en el repositorio, convirtiéndolo en un grafo acíclico.

4.2. Diseño Detallado

Nuestra solución presenta cinco *packages* o *namespaces*: el dominio (entidades y lógica de negocio), el cliente para la Raspberry, el procesador de transacciones (comunicación con la *blockchain*), configuración (para direccionamiento) y el manejador de eventos para la comunicación entre el modelo de escritura y el de lectura descritos en la sección anterior. En el siguiente diagrama de clases podemos ver rápidamente los atributos, funciones y métodos, así como las dependencias con otras clases.

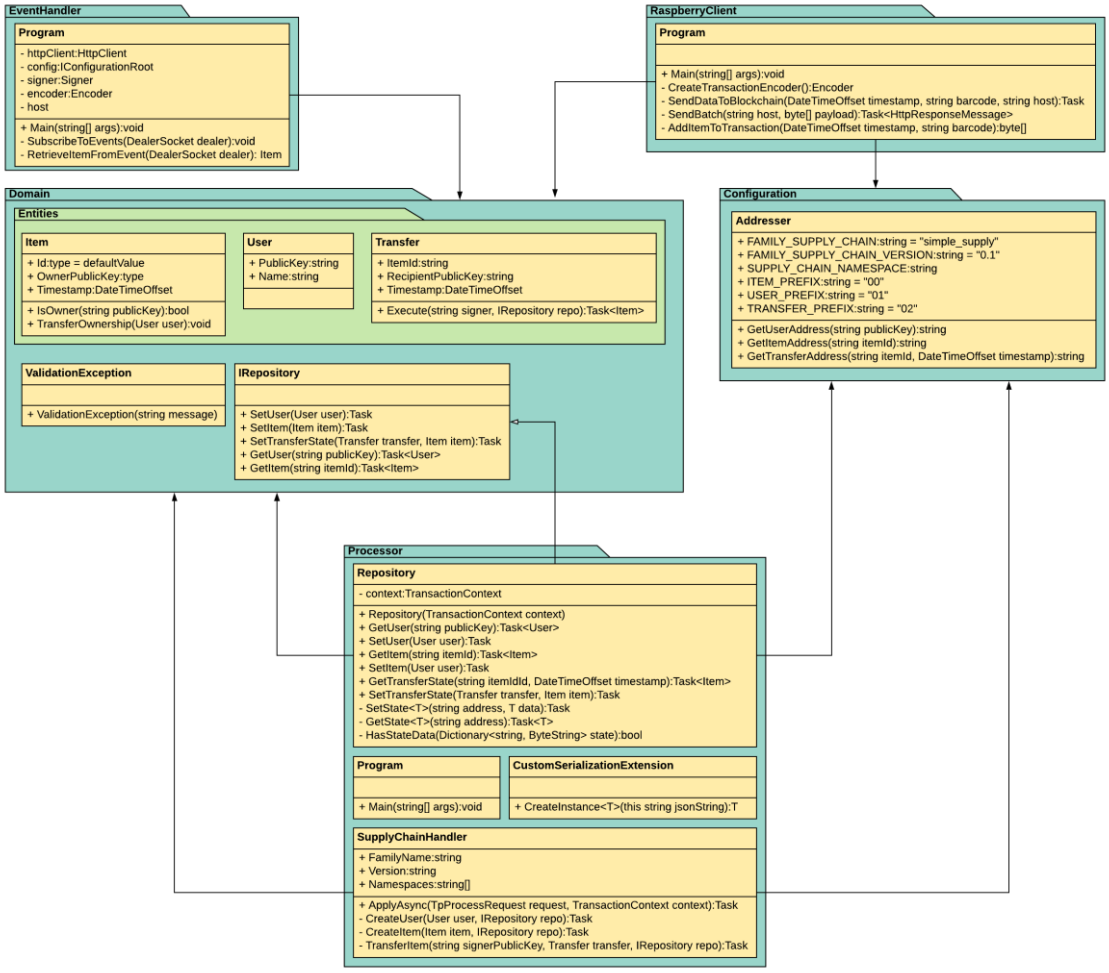

Figura 11.

Cabe destacar el patrón repositorio utilizado para separar la lógica de negocio de la implementación concreta mediante una clase interfaz en el paquete de dominio y su implementación en el paquete del procesador, de esta forma estamos siguiendo el principio de inversión de dependencias. Pese al esfuerzo por separar las distintas capas con el objetivo de crear un *software* con bajo acoplamiento y alta cohesión, independiente de las herramientas elegidas, el caso de *blockchain* es bastante particular. Un claro ejemplo de esta dependencia es la clase 'Addresser' dentro del paquete de configuración ya que se trata de información totalmente relativa al funcionamiento de Sawtooth. Aun así, las dependencias que se generan son dependencias bien controladas que siguiendo buenas prácticas pueden manejarse para dar como resultado un software de calidad. De hecho, no sería demasiado complicado cambiar el motor de persistencia del modelo de escritura siempre y cuando recordemos que habría que implementar las características de autenticación, autorización y auditoría que Sawtooth provee de serie.

4.3. Tecnología Utilizada

En esta sección se describen las principales tecnologías utilizadas, poniendo el foco en Sawtooth por ser este un trabajo sobre la tecnología *blockchain*.

4.3.1. Raspberry Pi

Para la parte relacionada con el IoT de nuestra solución hemos utilizado una Raspberry Pi modelo 3B+ con Raspbian como sistema operativo. La Raspberry Pi es la placa de un ordenador simple compuesto por CPU, memoria RAM, puertos de entrada y salida de audio y vídeo, conectividad de red, ranura SD para almacenamiento, reloj, una toma para la alimentación y conexiones para periféricos de bajo nivel, si bien todo esto y otras características dependen de la gama y el modelo. Existen otras marcas con productos similares como Arduino pero nosotros nos hemos decantado por esta opción porque es un dispositivo muy versátil, económico y con gran cantidad de documentación escrita y audiovisual disponible y con el que ya había trabajado anteriormente. Aunque existe un gran número de sistemas operativos completos y distribuciones ligeras disponibles para su instalación, hemos elegido Raspbian por ser el SO oficial para todos los modelos de Raspberry Pi.

4.3.2. Hyperledger Sawtooth

En el capítulo referente al Estado de la cuestión se ha descrito la tecnología *blockchain* y las diferentes soluciones que ofrece el mercado, por lo que en este apartado vamos a enfocarnos en describir la solución elegida para este proyecto: Hyperledger Sawtooth. Se trata de una solución empresarial para construir, implementar y ejecutar libros de contabilidad distribuidos (también llamados *blockchains*). Proporciona una plataforma extremadamente modular y flexible al estado compartido para implementar actualizaciones basadas en transacciones coordinadas por algoritmos de consenso entre partes que no son de confianza. Sawtooth separa el sistema central del dominio de la aplicación, por lo que los contratos inteligentes pueden especificar las reglas comerciales para las aplicaciones sin necesidad de conocer el diseño subyacente del sistema central.

Antes que nada vamos a recordar cuales son las características comunes de las tecnologías *blockchain* y aclarar algunas particularidades del lenguaje como bien explica el siguiente párrafo extraído de un artículo del BBVA [Fresno18]:

> "Bitcoin, 'blockchain' y ahora DLT, los avances tecnológicos obligan a incorporar al vocabulario términos recientes de enorme impacto. Y, en ocasiones, tanta novedad puede llevar a confusiones o malentendidos. Una de las más habituales es pensar que todas las 'blockchain' son DLT, es decir, tecnologías de registro distribuido ('*Distributed Ledger Technology*, en inglés).
>
> ¿Cuál es la diferencia entre 'blockchain' y una DLT? Es más sencillo de lo que parece. Una 'blockchain', una cadena de bloques, es un tipo de DLT. Es decir, se ha producido un fenómeno frecuente: el éxito de un servicio, producto o aplicación concreta supera tan claramente al 'paraguas' que la engloba que acaba incluso fagocitando su nombre. Pero de la misma forma que no todas las hojas adhesivas son Post-It, no todas las DLT son 'blockchain'.
>
> Desde un punto de vista más técnico, una DLT es simplemente una base de datos que gestionan varios participantes y no está centralizada. No existe una autoridad central que ejerza de árbitro y

verificador. El registro distribuido aumenta la transparencia —dificultando cualquier tipo de fraude o manipulación— y el sistema es más complicado de 'hackear'.

Es probable que todo esto resulte familiar, porque se habla de ello en artículos como este sobre las características de 'blockchain'. Y es que 'blockchain' no es otra cosa que una DLT con una serie de características particulares. También es una base de datos —o registro— compartida, pero en este caso mediante unos bloques que, como indica su propio nombre, forman una cadena. Los bloques se cierran con una especie de firma criptográfica llamada 'hash'; el siguiente bloque se abre con ese 'hash', a modo de sello lacrado. De esta forma, se certifica que la información, encriptada, no se ha manipulado ni se puede manipular. 'Blockchain' debe su fama, entre otras cosas, a que es la tecnología detrás de la famosa criptomoneda 'bitcoin'."

4.3.2.1. Características

Aclarado esto, por completitud pasamos a detallar qué nos ofrece esta herramienta. Las características más distintivas de Sawtooth son:

- **Separación entre el nivel de aplicación y el sistema principal**

Sawtooth facilita el desarrollo y la implementación de una aplicación al proporcionar una separación clara entre el nivel de la aplicación y el nivel del sistema central. Sawtooth proporciona una abstracción de contrato inteligente que permite a los desarrolladores de aplicaciones escribir la lógica del contrato en un lenguaje a su elección.

- **Redes privadas con las funciones de permisos**

Sawtooth está diseñado para resolver los desafíos de las redes autorizadas (privadas). Los *clusters* de nodos Sawtooth se pueden implementar fácilmente con permisos separados. No existe un servicio centralizado que pueda potencialmente filtrar patrones de transacciones u otra información confidencial. La cadena de bloques almacena la configuración que especifica los permisos, como roles e identidades, para que todos los participantes en la red puedan acceder a esta información.

- **Ejecución de transacciones paralelas**

La mayoría de las cadenas de bloques requieren la ejecución de transacciones en serie para garantizar un orden coherente en cada nodo de la red. Sawtooth incluye un programador paralelo avanzado que divide las transacciones en flujos paralelos. En función de las ubicaciones en el estado a las que accede una transacción, Sawtooth aísla la ejecución de transacciones entre sí mientras mantiene los cambios contextuales. Cuando es posible, las transacciones se ejecutan en paralelo, mientras se evita el doble gasto incluso con múltiples modificaciones en el mismo estado. La programación paralela proporciona un aumento potencial sustancial en el rendimiento sobre la ejecución en serie.

- **Sistema de eventos**

Hyperledger Sawtooth admite la creación y transmisión de eventos. Esto permite a las aplicaciones: suscribirse a eventos que ocurran relacionados con la cadena de bloques, como la confirmación de un nuevo bloque o el cambio a una nueva bifurcación; suscribirse a eventos específicos de la aplicación definidos por una familia de transacciones; y transmitir información sobre la ejecución de una transacción a los clientes sin almacenar esos datos en estado.

- **Compatibilidad del contrato de Ethereum con Seth**

El proyecto de integración Sawtooth-Ethereum, Seth, extiende la interoperabilidad de la plataforma Sawtooth a Ethereum. Los contratos inteligentes EVM (Ethereum Virtual Machine) se pueden implementar en Sawtooth utilizando la familia de transacciones Seth.

- **Algoritmos de consenso conectables**

Sawtooth admite el consenso dinámico, que puede permitir diferentes tipos de consenso en la misma cadena de bloques. El consenso dinámico significa que el algoritmo de consenso se puede cambiar por una cadena de bloques sin reiniciar la cadena de bloques o reiniciar Sawtooth. El consenso es modular ya que cada algoritmo de consenso se implementa como un módulo independiente que se ejecuta como un proceso independiente.

- **Familias de transacciones de muestra**

En Sawtooth, el modelo de datos y el lenguaje de transacciones se implementan en una familia de transacciones. Se espera que los usuarios creen familias de transacciones personalizadas que reflejen los requisitos únicos de sus libros de contabilidad, no obstante, proporciona varias familias de transacciones principales como modelos:

  - **IntegerKey**: se utiliza para probar los libros de contabilidad desplegados.
  - **Configuración**: proporciona una implementación de referencia para almacenar las opciones de configuración en cadena.
  - **Identidad**: maneja los permisos en cadena para las claves de transacción y validación para optimizar la administración de identidades para listas de claves públicas.
  - **Smallbank**: maneja el análisis de rendimiento para evaluaciones comparativas y pruebas de rendimiento al comparar el rendimiento de los sistemas *blockchain*. Esta familia de transacciones se basa en el punto de referencia H-Store Smallbank.
  - **BlockInfo**: proporciona una metodología para almacenar información sobre un número configurable de bloques históricos.

4.3.2.2. Arquitectura

Una vez proporcionada la panorámica general, vamos a explorar la arquitectura de este software. Sawtooth separa las funciones centrales de *blockchain* de la lógica empresarial a través del uso de procesadores de transacciones y aplicaciones cliente.

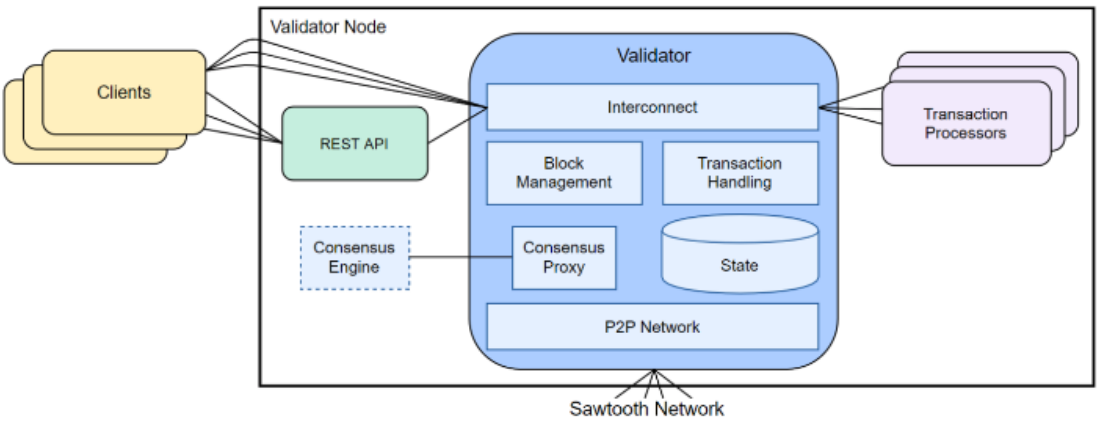

Figura 12. [Hyperledger Sawtooth20]

Como podemos ver en la imagen, existen cuatro componentes principales: el validador, los procesadores de transacciones, y los clientes, que pueden comunicarse con el validador de forma directa o a través de una API REST, que es opcional para comunicarse entre el cliente y el procesador de transacciones. Una aplicación puede utilizar una API REST personalizada, o usar la API REST que proporciona Sawtooth y que simplifica el desarrollo del cliente al adaptar la comunicación cliente-validador a HTTP/JSON estándar. La API REST se ejecuta como un servicio en el nodo validador o en un sistema separado.

El cliente se encarga de la generación de transacciones, la visualización de datos y manejo de eventos como cambios de estado, falla al confirmar un bloque, bifurcaciones, etc. El procesador de transacciones contiene la lógica de negocio.Por su parte, el validador Sawtooth es el componente central que valida lotes de transacciones, combina lotes en bloques, mantiene el consenso con la red Sawtooth para agregar bloques candidatos a la versión de cada nodo de la cadena de bloques y coordina la comunicación entre clientes, procesadores de transacciones y otros nodos de validación. Así pues, cada nodo Sawtooth ejecuta un proceso de validación. Cada validador tiene su propia instancia de *blockchain* y se comunica con los otros validadores a través de una red *peer-to-peer* mediante *gossip protocol*.

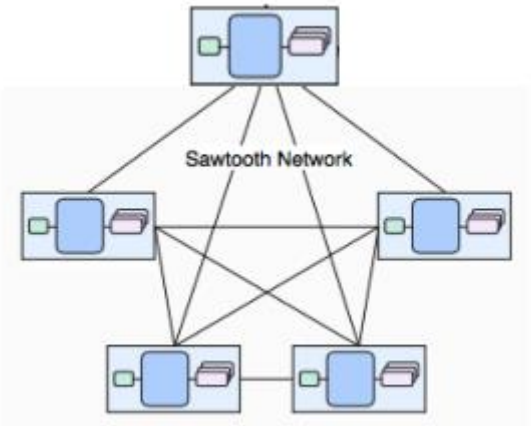

Figura 13. [Hyperledger Sawtooth20]

Una red Sawtooth consta de un conjunto de nodos validadores. Un nodo validador es un sistema host (computadora física, máquina virtual o conjunto de contenedores Docker) que ejecuta un proceso de validación y un conjunto idéntico de procesadores de transacciones. El bloque de génesis se crea solo para el primer nodo validador. Incluye ajustes de configuración en cadena, como el tipo de consenso, que estarán disponibles para los nuevos nodos de validación una vez que se unan a la red. El primer nodo validador en la red no tiene un significado especial, aparte de ser el nodo que creó el bloque génesis (el primer bloque en la cadena de bloques). Sawtooth no tiene el concepto de "nodo principal" o "nodo maestro". En una red Sawtooth, cada nodo tiene el mismo bloque de génesis y trata a todos los demás nodos como pares.

4.3.2.3. Aplicación

La aplicación define las operaciones o tipos de transacciones que se permiten en la cadena de bloques. Una aplicación puede ser una lógica empresarial nativa o una máquina virtual de contrato inteligente. Es decir, puede ser tan simple como un formato de transacción única con validación asociada y lógica de actualización de estado, o tan compleja como una máquina virtual con contabilidad de código de operación y código de bytes almacenado en el estado (en la cadena de bloques) como contratos inteligentes. Sawtooth permite que estas decisiones de diseño se tomen en la capa de procesamiento de transacciones, lo que permite que existan múltiples tipos de aplicaciones en la misma instancia de la red *blockchain*.

4.3.2.3.1. Componentes

Una aplicación Sawtooth se denomina *transaction family* (familia de transacciones) en la Documentación de Sawtooth. Una familia de transacciones define el modelo de datos y el conjunto de transacciones posibles para una aplicación. Cada aplicación define los procesadores de transacciones personalizados para sus requisitos únicos. Por tanto, para crear una aplicación Sawtooth customizada es necesario desarrollar los componentes que siguen:

- Un **modelo de datos** para definir operaciones válidas y especificar la carga útil de la transacción (datos para la aplicación).
- Un **procesador de transacciones** para definir la lógica de negocio de su aplicación. El procesador de transacciones valida los lotes de transacciones y actualiza el estado según las reglas definidas por la aplicación. El procesador de transacciones se ejecuta en cada nodo validador de la red Sawtooth.
- Un **cliente** para generar y enviar transacciones (cambios para la cadena de bloques) al validador; así como mostrar datos y reaccionar a eventos. Un cliente puede ejecutarse en un sistema separado o en el nodo validador.
- Una **API REST** opcional o un servicio RESTful para comunicarse entre el cliente y el validador. Una aplicación puede utilizar la API REST estándar de Sawtooth o proporcionar una API REST personalizada. Ésta se ejecuta como un servicio en el nodo validador o en un sistema separado.

4.3.2.3.2. Conceptos clave

Las modificaciones al estado se realizan creando y aplicando transacciones. Un cliente crea una transacción y la envía al validador. El validador aplica la transacción que provoca un cambio de estado. Las transacciones siempre se envuelven dentro de un lote. Todas las transacciones dentro de un lote se comprometen a declarar juntas o no declarar en absoluto. Por tanto, los lotes son la unidad atómica del cambio de estado. En esta sección vamos a describir la estructura de las transacciones, lotes y bloques de Sawtooth, y explicar cómo Sawtooth maneja el estado y el direccionamiento.

Transaction

Una transacción es un solo cambio en el estado de la cadena de bloques. Está definido como un objeto *protobuf* que contiene un encabezado, una firma y una carga útil. La carga útil de la transacción se utiliza durante la ejecución de la transacción como una forma de transmitir el cambio que se debe aplicar al estado. Solo la aplicación que procesa la transacción deserializa la carga útil. Para todos los demás componentes del sistema, la carga útil es solo una secuencia de bytes. El firmante de la transacción genera la firma digital al firmar el encabezado de la transacción con su clave privada. El campo 'header' es una versión serializada de TransactionHeader. El encabezado está presente en forma serializada para que los bytes exactos se puedan verificar con la firma al recibir la transacción.

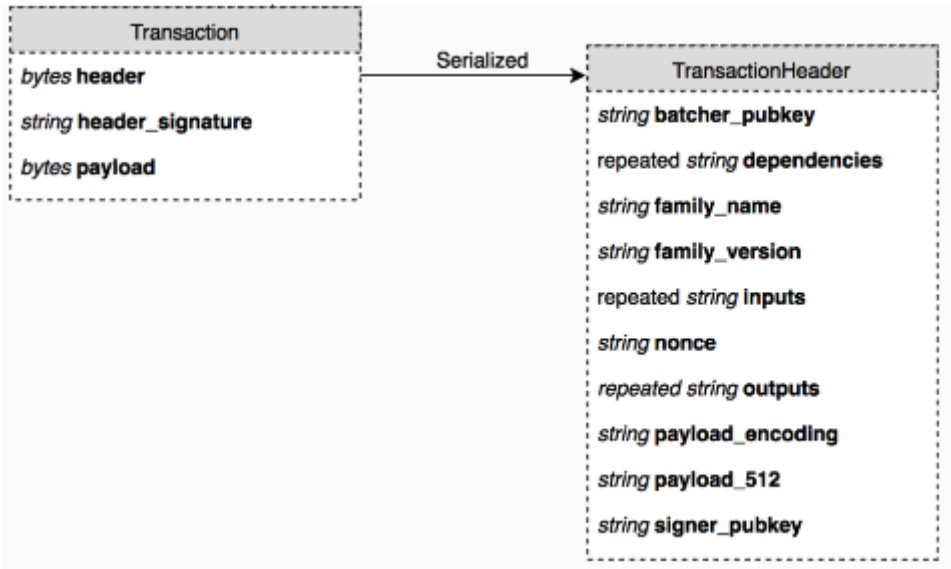

Figura 14. [Hyperledger Sawtooth20]

En la Figura 14, el encabezado de la transacción 'Transaction' contiene los siguientes campos:

- **batcher_pubkey**, debe coincidir con la clave pública utilizada para firmar el lote en el que se incluye esta transacción.
- **family_name**, nombre de la familia de transacciones.
- **family_version**, versión de la familia de transacciones.

- **dependencies**, permite especificar explícitamente las transacciones que deben aplicarse antes de la transacción actual. Las dependencias explícitas son útiles en situaciones en las que las transacciones tienen dependencias pero no se pueden colocar en el mismo lote (por ejemplo, si las transacciones se envían en momentos diferentes).
- Para ayudar en las operaciones de programación en paralelo, los campos de **inputs** y **outputs** de una transacción contienen direcciones de estado. El planificador determina las dependencias implícitas entre transacciones basándose en la interacción con el estado. Las direcciones pueden ser de nodo hoja o direcciones de prefijo parcial que funcionan como comodines y permiten que las transacciones especifiquen las partes del árbol en lugar de solo nodos hoja.
- **nonce**, contiene una cadena aleatoria generada por el cliente. Con el nonce presente, si dos transacciones contienen los mismos campos, el nonce asegura que generarán diferentes firmas de encabezado.
- **paload_encoding**, se utiliza durante la ejecución de la transacción como una forma de transmitir el cambio que se debe aplicar al estado. Solo la familia de transacciones que procesa la transacción deserializa la carga útil; para todos los demás componentes del sistema, la carga útil es solo una secuencia de bytes.
- **payload_512**, contiene un hash SHA-512 de los bytes de la carga útil. Como parte del encabezado, 'payload_512' se firma y luego se verifica, mientras que el campo de carga útil no. Para verificar que el campo de carga útil coincide con el encabezado, se puede comparar un SHA-512 del campo de carga útil con payload_512.
- **signer_pubkey**, debe coincidir con la clave pública utilizada para firmar los bytes del encabezado dando como resultado header_signature.

Batch

O lote en español, es la unidad atómica de cambio de estado y envuelve una o más transacciones. Está definido como un objeto *protobuf* que contiene una lista de transacciones y un encabezado que incluye una firma del creador del lote (a menudo la misma que la del creador de la transacción). Esto quiere decir que si se ha aplicado un lote, todas las transacciones se habrán aplicado en el orden contenido dentro del lote. Si no se ha aplicado un lote (tal vez porque una de las transacciones no es válida), no se aplicará ninguna de las transacciones.

Esto simplifica enormemente la gestión de dependencias desde la perspectiva del cliente, ya que las transacciones dentro de un lote no necesitan que se declaren dependencias explícitas entre ellas. Como resultado, la utilidad de las dependencias explícitas (contenidas en el campo de dependencias en una transacción) está restringida a las dependencias donde las transacciones no se pueden colocar en el mismo lote. Los lotes resuelven un problema importante que no se puede resolver con dependencias explícitas. Supongamos que tenemos las transacciones A, B y C y que el comportamiento deseado es A, B, C se aplica en ese orden y, si alguna de ellas no es válida, no se debe aplicar ninguna. Si intentamos resolver esto usando sólo dependencias, podríamos intentar una relación entre ellas como: C depende de B, B depende de A y A depende de C. Sin embargo, el campo de dependencias no se

puede usar para representar esta relación, ya que las dependencias imponen el orden y lo anterior es cíclico (y por lo tanto no se puede ordenar).

Las transacciones de múltiples familias de transacciones también se pueden agrupar, lo que fomenta aún más la reutilización de familias de transacciones. Por ejemplo, las transacciones para una familia de transacciones de configuración o identidad se pueden agrupar con transacciones específicas de la aplicación. Las transacciones y los lotes también se pueden firmar con diferentes claves. Por ejemplo, una aplicación de navegador puede firmar la transacción y un componente del lado del servidor puede agregar transacciones, crear el lote y firmar el lote. Esto permite patrones de aplicación interesantes, incluida la agregación de transacciones de múltiples transactores en una operación atómica (el lote).

Existe una restricción importante entre transacciones y lotes, que es que la transacción debe contener la clave pública del firmante del lote en el campo 'batcher_public_key'. Esto es para evitar que las transacciones se reutilicen por separado del lote previsto. Entonces, por ejemplo, a menos que tenga la clave privada del *batcher*, no es posible tomar transacciones de un lote y volver a empaquetarlas en un nuevo lote, omitiendo algunas de las transacciones.

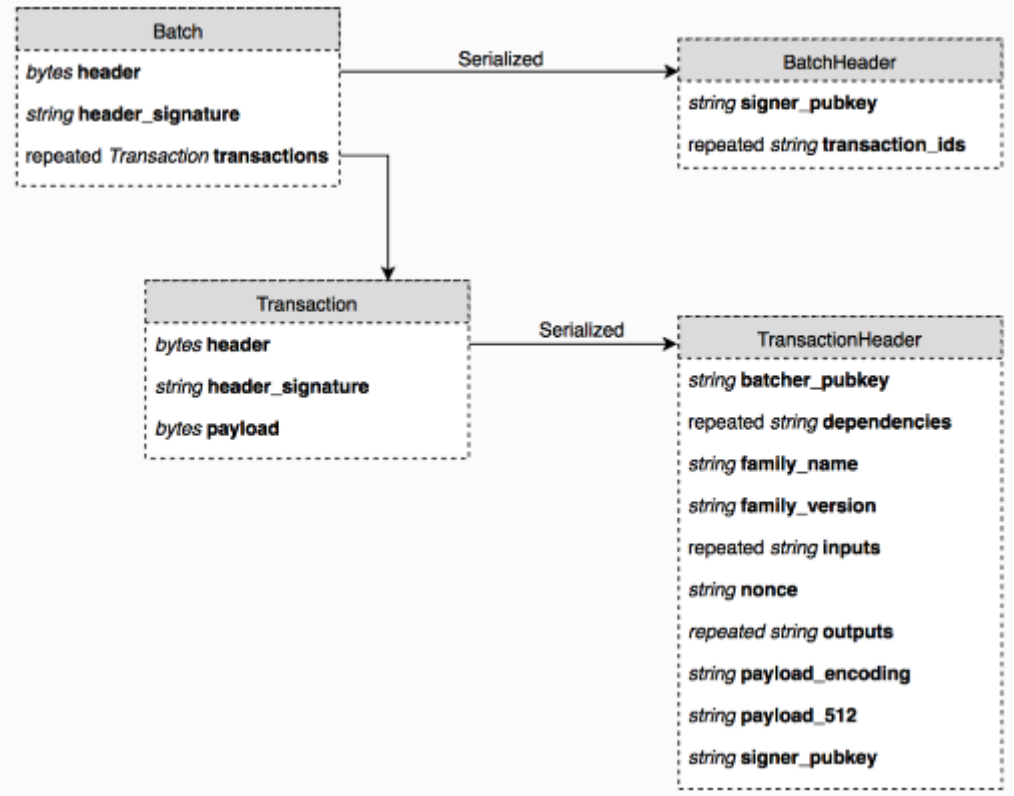

Figura 15. [Hyperledger Sawtooth20]

Siguiendo el patrón presentado en 'Transaction', el campo de encabezado 'Batch' es una versión serializada de 'BatchHeader', tal y como se ilustra en la figura 15. El encabezado está firmado por la clave privada del firmante (no se envía con el lote) y la firma resultante se almacena en 'header_signature'. El encabezado está

presente de forma serializada para que los bytes exactos se puedan verificar con la firma al recibir el lote. El documento serializado resultante se firma con la clave ECDSA privada del operador mediante la curva secp256k1. El campo de 'transactions' contiene una lista de transacciones que componen el lote. Las transacciones se aplican en el orden indicado. El campo 'transaction_ids' contiene una lista de Transaction 'header_signatures' y debe tener el mismo orden que el campo de 'transactions'.

Block

Se trata de un grupo de lotes de Sawtooth. Un bloque tiene un encabezado que incluye una marca de tiempo, un firmante y un hash (ID de bloque único). Después de que se confirma un bloque, el encabezado también identifica el bloque anterior en la cadena de bloques.

State

Sawtooth representa el estado de todas las familias de transacciones en una sola instancia de un árbol Radix Merkle en cada validador. El proceso de validación de bloques en cada validador asegura que las mismas transacciones resulten en las mismas transiciones de estado y que los datos resultantes sean los mismos para todos los participantes de la red. Sawtooth utiliza un árbol Radix Merkle direccionable para almacenar datos para familias de transacciones. Analicemos eso: el árbol es un árbol de Merkle porque es una estructura de datos de copia en escritura que almacena sucesivos hashes de nodo de hoja a raíz ante cualquier cambio en el árbol. Para un conjunto dado de transiciones de estado asociadas con un bloque, podemos generar un único hash raíz que apunte a esa versión del árbol. Al colocar este hash de raíz de estado en el encabezado del bloque, podemos obtener un consenso sobre la versión esperada del estado además del consenso sobre la cadena de bloques. Si las transiciones de estado de un validador para un bloque dan como resultado un hash diferente, el bloque no se considera válido.

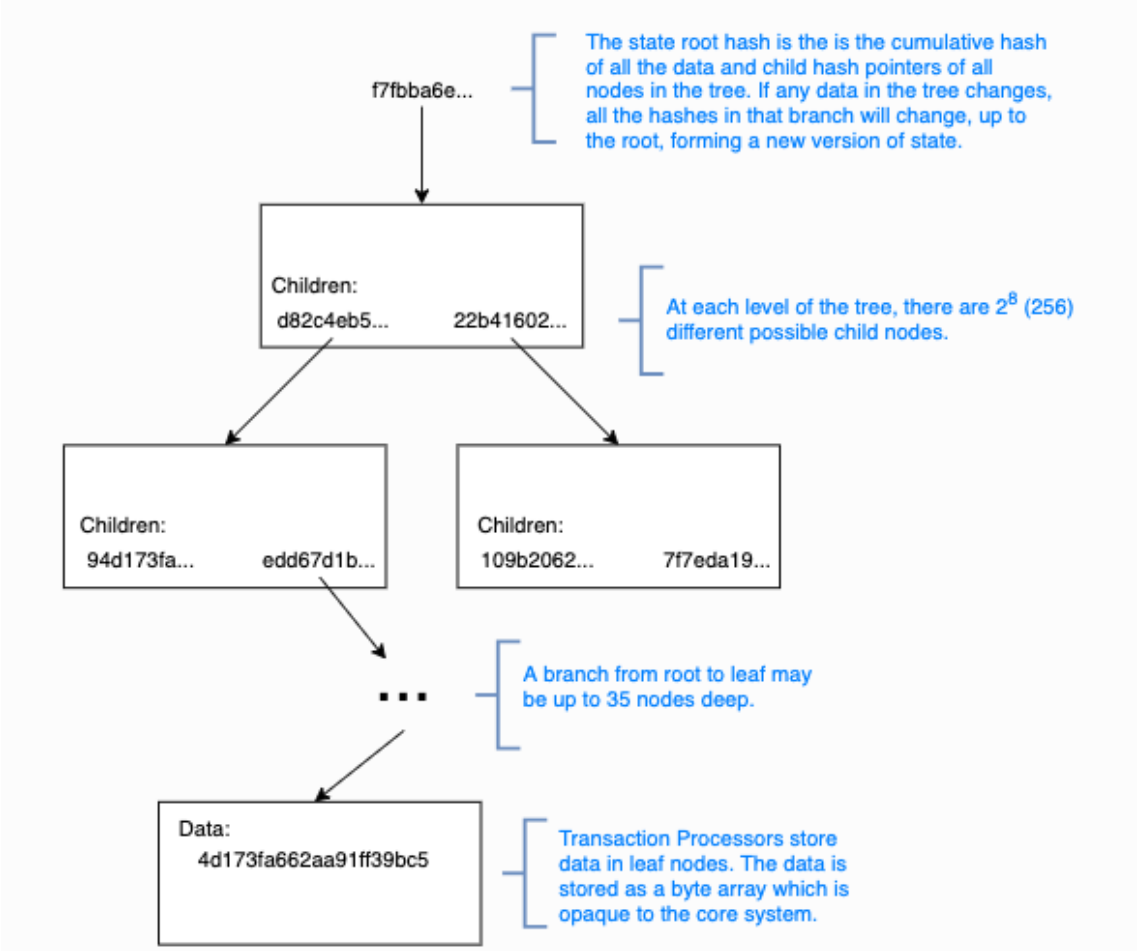

Figura 16. [Hyperledger Sawtooth20]

Addressing

El estado se divide en espacios de nombres que permiten flexibilidad a los autores de familias de transacciones para definir, compartir y reutilizar datos de estado global entre procesadores de transacciones. El árbol es un árbol Radix direccionable porque las direcciones identifican de manera única las rutas a los nodos hoja en el árbol donde se almacena la información. Una dirección es una cadena de 70 caracteres codificada en hexadecimal que representa 35 bytes. En la implementación del árbol, cada byte es un segmento de ruta de Radix que identifica el siguiente nodo en la ruta a la hoja que contiene los datos asociados con la dirección. El formato de la dirección contiene un prefijo de espacio de nombres de 3 bytes (6 caracteres hexadecimales) que proporciona 224 (16.777.216) posibles espacios de nombres diferentes en una instancia determinada de Sawtooth. Los 32 bytes restantes (64 caracteres hexadecimales) se codifican según las especificaciones del diseñador del espacio de nombres y pueden incluir esquemas para subdividir más, distinguir tipos de objetos y mapear identificadores únicos específicos del dominio en porciones de la dirección.

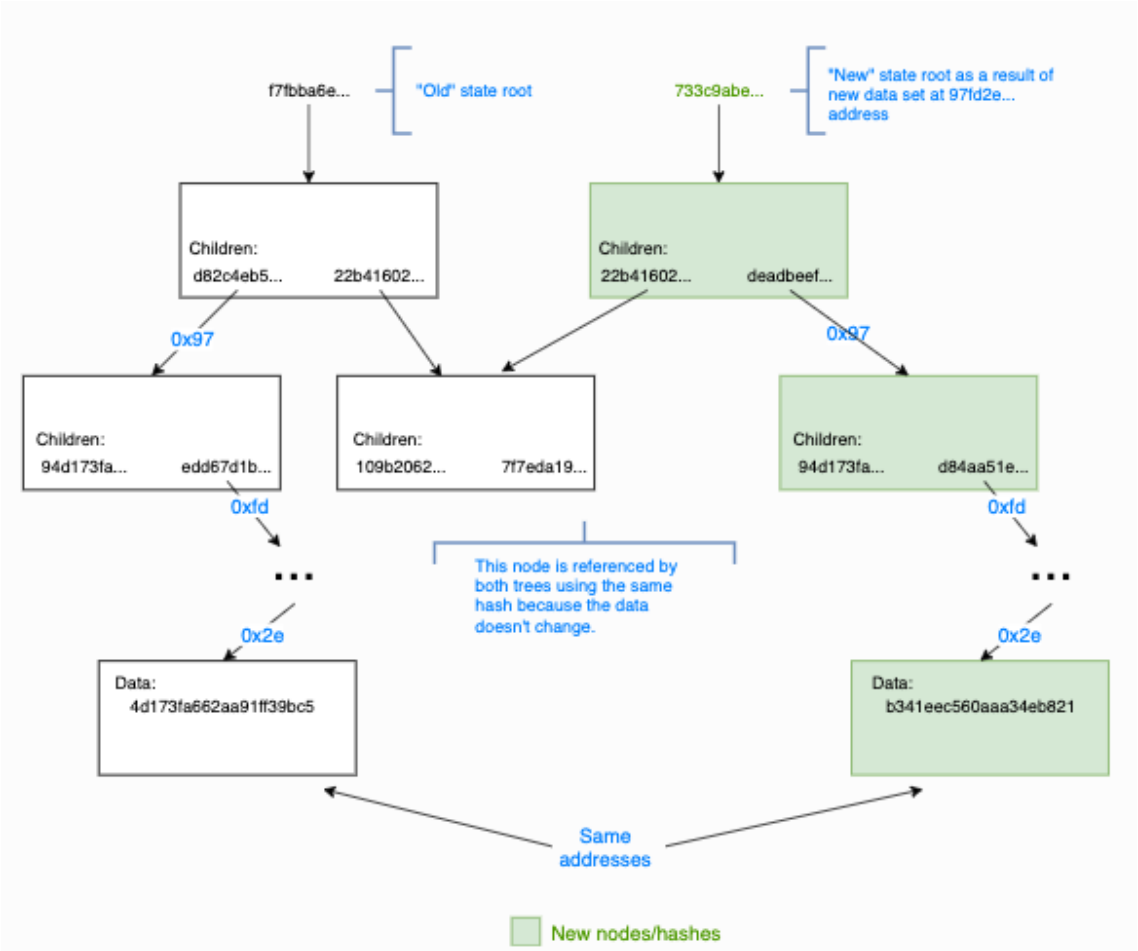

Figura 17. [Hyperledger Sawtooth20]

Además de las preguntas sobre la codificación de direcciones, los diseñadores de espacios de nombres también deben definir el mecanismo de serialización y las reglas para serializar/deserializar los datos almacenados en las direcciones. El procesador de transacciones específico del dominio realiza llamadas *get(dirección)* y *set(dirección, datos)* contra una versión del estado que proporciona el validador: *get(dirección)* devuelve la matriz de bytes que se encuentra en esa dirección y *set(dirección, datos)* establece la matriz de bytes almacenada en esa dirección. La matriz de bytes es opaca para el sistema central. Solo tiene significado cuando se deserializa mediante un componente específico del dominio según las reglas del espacio de nombres. Es fundamental seleccionar un esquema de serialización que sea determinista, en todas las ejecuciones de la transacción, en las plataformas y en las versiones del marco de serialización. Deben evitarse las estructuras de datos que no imponen la serialización ordenada (por ejemplo, conjuntos, mapas, dictados). El requisito es producir consistentemente la misma matriz de bytes en el espacio y el tiempo. Si no se produce la misma matriz de bytes, el hash del nodo hoja que contiene los datos será diferente, al igual que todos los nodos principales de regreso a la raíz. Esto dará como resultado que las transacciones y los bloques que las contienen se consideren válidos en algunos validadores e inválidos en otros, dependiendo del comportamiento no determinista.

Global state

También llamado *global state agreement* (acuerdo de estado global), representa el consenso de todos los participantes sobre el estado de la cadena de bloques. En una cadena de bloques, el consenso es el proceso de construir un acuerdo entre un grupo de participantes que desconfían mutuamente. Los algoritmos para lograr consenso con fallas arbitrarias generalmente requieren alguna forma de votación entre un conjunto conocido de participantes. Los enfoques generales incluyen el consenso al estilo de Nakamoto, que elige a un líder a través de algún tipo de lotería, y variantes de los algoritmos tradicionales de tolerancia a fallas bizantinas (BFT), que utilizan múltiples rondas de votos explícitos para lograr el consenso.

Sawtooth abstrae los conceptos centrales del consenso y aísla el consenso de la semántica de transacciones. La interfaz admite la conexión de varias implementaciones de consenso. Más importante aún, Sawtooth permite diferentes tipos de consenso en la misma cadena de bloques. El consenso se selecciona durante la configuración inicial de la red y se puede cambiar en una cadena de bloques en ejecución con una transacción. Sawtooth actualmente admite estas implementaciones de consenso:

- *Proof of Elapsed Time* (PoET, Prueba de tiempo transcurrido), un algoritmo de consenso diseñado para ser un protocolo de alto grado de producción capaz de admitir grandes poblaciones de redes. PoET se basa en la ejecución segura de instrucciones para lograr los beneficios de escalado de un algoritmo de consenso de estilo Nakamoto sin los inconvenientes de consumo de energía del algoritmo de PoW (prueba de trabajo).
- Simulador de PoET, que proporciona un consenso de estilo PoET en cualquier tipo de hardware, incluido un entorno de nube virtualizado.
- Modo de desarrollo, un algoritmo simplificado de líder aleatorio que es útil para el desarrollo y las pruebas.

4.3.2.3.3. Permisos

La siguiente tabla describe los tres tipos de escenarios que se pueden dar en una red Sawtooth en función de sus competencias:

Network Scenario	Competencias
Public Network	➔ Permitir que todos los firmantes de lotes envíen lotes ➔ Permitir que todos los firmantes de transacciones envíen transacciones ➔ Permitir que todos los nodos se unan a la red del validador
Consortium Network	➔ Permitir que todos los firmantes de lotes envíen lotes ➔ Permitir que todos los firmantes de transacciones envíen transacciones

	➔ Permitir que solo nodos específicos se unan a la red del validador ➔ Permitir que solo los nodos específicos participen en el consenso ➔ Admite permisos de transacciones basados en políticas
Private Network	➔ Permitir que solo los firmantes de lotes específicos envíen lotes ➔ Permitir que solo los firmantes de transacciones específicas envíen transacciones ➔ Permitir que solo nodos específicos se unan a la red del validador ➔ Permitir que solo los nodos específicos participen en el consenso ➔ Restringir el tipo de transacciones que pueden firmar los operadores ➔ Restringir el acceso al espacio de direcciones a un conjunto limitado de operadores ➔ Admite permisos de transacciones basados en políticas

Tabla 3.

Para implementar los distintos escenarios que acabamos de describir, Sawtooth incluye los siguientes mecanismos para el diseño de permisos:

- Permiso de claves de transacción y lote, que controla la aceptación de transacciones y lotes según las claves de firma.
- Permiso de la clave del validador, que controla qué nodos pueden establecer conexiones con la red del validador.
- Aplicación de políticas, un conjunto de reglas DENY y PERMIT que se pueden usar para controlar el acceso a la red del validador y determinar qué operadores pueden participar en la red.

4.3.2.3.4. Eventos y recibos de transacciones

Una aplicación Sawtooth puede suscribirse a eventos del core de Sawtooth y eventos específicos de la aplicación que ocurren en la cadena de bloques, como un nuevo bloque que se confirma o cambia a una nueva bifurcación; luego informa a los clientes sobre la ejecución de la transacción sin almacenar esos datos en el estado. Una aplicación también puede solicitar la actualización de eventos emitiendo una

solicitud de suscripción de actualización con una ID de bloque específica. El validador Sawtooth envía datos para todos los eventos que han ocurrido después de que se comprometió ese bloque. Una suscripción de evento incluye el tipo de evento, una dirección (como una dirección específica, un rango o un patrón) y filtros opcionales para los atributos del evento. La mayoría de las suscripciones a eventos utilizan filtros para centrarse en los eventos específicos de interés.

4.3.3. Elasticsearch

En este trabajo es también muy relevante la tecnología Elasticsearch, que de acuerdo con la documentación que acompaña la distribución [Elastic20], se trata de un motor de búsqueda y análisis distribuido *open source* para todos los tipos de datos, incluidos textuales, numéricos, geoespaciales, estructurados y desestructurados. Está desarrollado en Apache Lucene y es conocido por sus API REST simples, naturaleza distribuida, velocidad y escalabilidad. Es el componente principal del Elastic Stack, un conjunto de herramientas open source para la ingesta, el enriquecimiento, el almacenamiento, el análisis y la visualización de datos. Puede usarse para una variedad de casos de uso: búsqueda de aplicaciones, de sitio web, empresarial, *logging* y analíticas de log, métricas de infraestructura y monitorización de contenedores, monitorización de rendimiento de aplicaciones, análisis y visualización de datos geoespaciales, analítica de seguridad y analítica de negocios.

Los datos sin procesar fluyen hacia Elasticsearch desde una variedad de fuentes, incluidos *logs*, métricas de sistema y aplicaciones web. La ingesta de datos es el proceso mediante el cual estos datos son parseados, normalizados y enriquecidos antes de su indexación en Elasticsearch. Una vez indexados en Elasticsearch, los usuarios pueden ejecutar consultas complejas sobre sus datos y usar agregaciones para recuperar resúmenes complejos de sus datos. Desde Kibana, los usuarios crean visualizaciones poderosas de sus datos, comparten dashboards y gestionan el Elastic Stack.

4.3.4. Android

Android es un sistema operativo móvil desarrollado por Google, basado en Kernel de Linux y otros software de código abierto. Fue diseñado para dispositivos móviles con pantalla táctil, como teléfonos inteligentes, tabletas, relojes inteligentes (Wear OS), automóviles (Android Auto) y televisores (Android TV). Es el sistema operativo móvil más utilizado del mundo, con una cuota de mercado superior al 80%.

Para el desarrollo de la app hemos empleado Kotlin, un lenguaje de programación de tipado estático que corre sobre la máquina virtual de Java y que también puede ser compilado a código fuente de JavaScript. El líder de desarrollo Andrey Breslav ha dicho que Kotlin está diseñado para ser un lenguaje de programación orientado a objetos de calidad industrial, y para ser un lenguaje mejor que Java pero todavía ser plenamente interoperable con código Java, permitiendo a las compañías hacer una migración gradual de Java a Kotlin.

También utilizamos Retrofit, un cliente REST para Android desarrollado por Square. La biblioteca proporciona un marco para autenticar e interactuar con una API y enviar solicitudes de red con OkHttp. Esta biblioteca hace que la descarga de datos JSON o XML desde una API web sea bastante sencilla. Una vez que se descargan los datos, se analizan en un POJO (Plain Old Java Object) que debe definirse para cada "recurso" en la respuesta. Para esto último usamos GSON, una biblioteca de código abierto para el lenguaje de programación Java que permite la serialización y deserialización entre objetos Java y su representación en notación JSON.

4.3.5. Docker

Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos. Docker utiliza características de aislamiento de recursos del kernel Linux, tales como cgroups y espacios de nombres (namespaces) para permitir que "contenedores" independientes se ejecuten dentro de una sola instancia de Linux, evitando la sobrecarga de iniciar y mantener máquinas virtuales. Dicho de otro modo, Docker implementa una API de alto nivel para proporcionar contenedores livianos que ejecutan procesos de manera aislada.

Mediante el uso de contenedores, los recursos pueden ser aislados, los servicios restringidos, y se otorga a los procesos la capacidad de tener una visión casi completamente privada del sistema operativo con su propio identificador de espacio de proceso, la estructura del sistema de archivos, y las interfaces de red. Contenedores múltiples comparten el mismo núcleo, pero cada contenedor puede ser restringido a utilizar solo una cantidad definida de recursos como CPU, memoria y E/S.

Usar Docker para crear y gestionar contenedores puede simplificar la creación de sistemas altamente distribuidos, permitiendo que múltiples aplicaciones, las tareas de los trabajadores y otros procesos funcionen de forma autónoma en una única máquina física o en varias máquinas virtuales. Esto permite que el despliegue de nodos se realice a medida que se dispone de recursos o cuando se necesiten más nodos, lo que permite una plataforma como servicio (PaaS - *Platform as a Service*) de estilo de despliegue y ampliación de los sistemas.

4.3.6. ZeroMQ

ZeroMQ (también escrito ØMQ, 0MQ o ZMQ) es una biblioteca de comunicaciones de alto rendimiento orientada a mensajes, destinada a la construcción de aplicaciones distribuidas. Está basada en colas de mensajes, pero a diferencia de otros middlewares orientados a mensajes, ZeroMQ no necesita un broker intermedio.

5. Desarrollo de la solución propuesta

Para poder crear una aplicación Sawtooth es necesario añadir el paquete NuGet Sawtooth.sdk y PeterO.Cbor en los *namespaces* ‘Processor’, ‘RaspberryClient’

'EventHandler' y 'Configuration'. Comenzamos por el *namespace* 'RaspberryClient' que contiene la siguiente clase 'Program.cs':

```
public class Program
{
   private static readonly HttpClient httpClient = new
HttpClient();
   private static readonly Signer signer = new Signer();
   private static readonly Encoder encoder =
CreateTransactionEncoder();
   private static readonly IConfigurationRoot config = new
ConfigurationBuilder()
       .AddJsonFile("appsettings.json")
       .Build();

   private static readonly string Host =
config["sawtoothProcessor"];

   public static async Task Main(string[] args)
   {
       while (true)
       {
           Console.WriteLine("Enter barcode: ");
           var value = Console.ReadLine();

           if (string.IsNullOrEmpty(value)) continue;

           var timestamp = DateTimeOffset.UtcNow;
           await SendDataToBlockchain(timestamp, value);
           value = "";
       }
   }

   private static Encoder CreateTransactionEncoder()
   {
       var batcherPublicKey =
signer.GetPublicKey().ToHexString();
       var settings = new EncoderSettings()
       {
           BatcherPublicKey = batcherPublicKey,
           SignerPublickey = batcherPublicKey,
           FamilyName = Addresser.FAMILY_SUPPLY_CHAIN,
           FamilyVersion =
Addresser.FAMILY_SUPPLY_CHAIN_VERSION
       };
       settings.Inputs.Add(Addresser.SUPPLY_CHAIN_NAMESPACE);
       settings.Outputs.Add(Addresser.SUPPLY_CHAIN_NAMESPACE);
       return new Encoder(settings, signer.GetPrivateKey());
```

```
    }

    private static async Task
SendDataToBlockchain(DateTimeOffset timestamp, string barcode)
    {
        var payload = AddItemToTransaction(timestamp, barcode);
        var response = await SendBatch(Host, payload);
        Console.WriteLine(await
response.Content.ReadAsStringAsync());
    }

    private static async Task<HttpResponseMessage>
SendBatch(string host, byte[] payload)
    {
        var content = new ByteArrayContent(payload);
        content.Headers.Add("Content-Type", "application/octet-
stream");
        var response = await
httpClient.PostAsync($"http://{host}:8008/batches", content);
        return response;
    }

    private static byte[] AddItemToTransaction(DateTimeOffset
timestamp, string barcode)
    {
        var item = new Item
        {
            Id = barcode,
            OwnerPublicKey =
signer.GetPublicKey().ToHexString(),
            Timestamp = timestamp
        };

        var cborObject = CBORObject.NewMap()
            .Add("command", "create_item")
            .Add("item", JsonSerializer.Serialize(item));

        return
encoder.EncodeSingleTransaction(cborObject.EncodeToBytes());
    }
}
```

La función ‘Main’ es un bucle a la espera de leer un código; una vez lo recibe llama al método ‘SendDataToBlockchain’ junto con el *timestamp* que crea una transacción, la envuelve en un batch y lo envía con una llamada a la API REST. Para construir el ‘Encoder’ que nos permite generar el batch y firmarlo hacemos uso de la clase ‘Addresser’ dentro del *namespace* ‘Configuration’.

```
public static class Addresser
{
   private const string ITEM_PREFIX = "00";
   private const string USER_PREFIX = "01";
   private const string TRANSFER_PREFIX = "02";
   public const string FAMILY_SUPPLY_CHAIN = "simple_supply";
   public const string FAMILY_SUPPLY_CHAIN_VERSION = "0.1";

   public static readonly string SUPPLY_CHAIN_NAMESPACE =

FAMILY_SUPPLY_CHAIN.ToByteArray().ToSha512().ToHexString().Sub
string(0, 6);

   public static string GetUserAddress(string publicKey) =>
       SUPPLY_CHAIN_NAMESPACE + USER_PREFIX +

publicKey.ToByteArray().ToSha512().ToHexString().Substring(0,
62);

   public static string GetItemAddress(string itemId) =>
       SUPPLY_CHAIN_NAMESPACE + ITEM_PREFIX +

itemId.ToByteArray().ToSha512().ToHexString().Substring(0,
62);

   public static string GetTransferAddress(string itemId,
DateTimeOffset timestamp) =>
       SUPPLY_CHAIN_NAMESPACE + TRANSFER_PREFIX +
       (itemId +
timestamp.ToString("O")).ToByteArray().ToSha512().ToHexString(
).Substring(0, 62);
}
```

Esta clase también es utilizada por el repositorio y el manejador de las transacciones que se encuentran en el *namespace* 'Processor' junto con la clase 'Program.cs'. Haber extraído estas constantes y funciones a una clase nos facilita la reutilización del código y minimiza la posibilidad de fallos y *bugs* ya que se comparte el acceso a una única fuente en vez de reescribirla en cada una de las otras clases. El repositorio nos permite interactuar con el estado de la *blockchain* a través de los métodos 'setState()' y 'getState()' utilizados en las funciones que leen o escriben datos como un ítem, un usuario y una transacción. En nuestro caso:

```
public class Repository : IRepository
{
   private readonly TransactionContext context;

   public Repository(TransactionContext context)
   {
       this.context = context;
   }

   public Task<User> GetUser(string publicKey) =>
```

```
GetState<User>(Addresser.GetUserAddress(publicKey));

   public Task SetUser(User user) =>
SetState(Addresser.GetUserAddress(user.PublicKey), user);

   public Task<Item> GetItem(string itemId) =>
GetState<Item>(Addresser.GetItemAddress(itemId));

   public async Task SetItem(Item item)
   {
       await SetState(Addresser.GetItemAddress(item.Id),
item);
       await context.AddEventAsync("item/create", new
Dictionary<string, string>
       {
           { "address", Addresser.GetItemAddress(item.Id) },
           { "item", JsonSerializer.Serialize(item)}
       }, ByteString.Empty);
   }

   public Task<Transfer> GetTransferState(string itemIdId,
DateTimeOffset timestamp) =>

GetState<Transfer>(Addresser.GetTransferAddress(itemIdId,
timestamp));

   public Task SetTransferState(Transfer transfer, Item item)
   {
       var itemAddress = Addresser.GetItemAddress(item.Id);
       var transferAddress =
Addresser.GetTransferAddress(transfer.ItemId,
transfer.Timestamp);
       return context.SetStateAsync(new Dictionary<string,
ByteString>
       {
           {itemAddress,
ByteString.CopyFromUtf8(JsonSerializer.Serialize(item))},
           {transferAddress,
ByteString.CopyFromUtf8(JsonSerializer.Serialize(transfer))}
       });
   }

   private Task SetState<T>(string address, T data) =>
       context.SetStateAsync(new Dictionary<string,
ByteString>
       {
           {address,
ByteString.CopyFromUtf8(JsonSerializer.Serialize(data))},
       });

   private async Task<T> GetState<T>(string address) where T :
class
   {
       var state = await context.GetStateAsync(new[]
{address});
```

```
        return !HasStateData(state) ? null :
state[address].ToStringUtf8().CreateInstance<T>();
    }

    private static bool HasStateData(Dictionary<string,
ByteString> state) =>
        state != null && state.Any() &&
!state.First().Value.IsEmpty;
}
```

El repositorio está libre de tener que ejecutar y comprobar las reglas de negocio. Nótese que la función 'SetItem(Item item)' añade un evento que se disparará al hacerse *commit* de una transacción de este tipo en el estado de la *blockchain*. Este repositorio es llamado por la clase 'SupplyChainHandler' que se encarga de recibir las transacciones y procesarlas. Por ejemplo, si hemos enviado una transacción cuyo propósito es crear un nuevo ítem primero la decodificará, leerá la clave "command" y, según sea el valor, llamará a una función u otra; en nuestro caso 'CreateItem(payload["item"].AsString().CreateInstance<Item>(), repository)'. Véase la implementación de la clase a continuación:

```
public class SupplyChainHandler : ITransactionHandler
{
    public string FamilyName => Addresser.FAMILY_SUPPLY_CHAIN;
    public string Version =>
Addresser.FAMILY_SUPPLY_CHAIN_VERSION;
    public string[] Namespaces => new[]
{Addresser.SUPPLY_CHAIN_NAMESPACE};

    public async Task ApplyAsync(TpProcessRequest request,
TransactionContext context)
    {
        var payload =
CBORObject.DecodeFromBytes(request.Payload.ToByteArray());
        Console.WriteLine(payload);

        var repository = new Repository(context);
        switch (payload["command"].AsString())
        {
            case "create_user":
                await
CreateUser(payload["user"].AsString().CreateInstance<User>(),
repository);
                return;
            case "create_item":
                await
CreateItem(payload["item"].AsString().CreateInstance<Item>(),
repository);
                return;
            case "transfer_item":
                await
TransferItem(request.Header.SignerPublicKey,
payload["transfer"].AsString().CreateInstance<Transfer>(),
```

```
repository);
                return;
            }
        }

        private async Task CreateUser(User user, IRepository repo)
        {
            var existingUser = await repo.GetUser(user.PublicKey);
            if (existingUser != null)
            {
                throw new InvalidTransactionException($"User with
the public key {user.PublicKey} already exists.");
            }

            await repo.SetUser(user);
        }

        private async Task CreateItem(Item item, IRepository repo)
        {
            var existingItem = await repo.GetItem(item.Id);
            if (existingItem != null)
            {
                throw new InvalidTransactionException($"Item with
the id {item.Id} already exists.");
            }

            await repo.SetItem(item);
        }

        private async Task TransferItem(string signerPublicKey,
Transfer transfer, IRepository repo)
        {
            try
            {
                var item = await transfer.Execute(signerPublicKey,
repo);
                await repo.SetTransfer(transfer, item);
            }
            catch (ValidationException e)
            {
                throw new InvalidTransactionException(e.Message);
            }
        }
}
```

Esta clase implementa la 'ITransactionHandler' proporcionada por el SDK de Sawtooth y debe ser añadida al procesador de transacciones de la siguiente forma:

```
class Program
{
    private static readonly IConfigurationRoot config = new
ConfigurationBuilder()
        .AddJsonFile("appsettings.json", optional: true)
```

```
        .AddEnvironmentVariables()
        .Build();

    static void Main(string[] args)
    {
        Console.WriteLine("Hello World!");
        var validatorAddress =
string.IsNullOrEmpty(config["VALIDATOR"]) ?
"tcp://127.0.0.1:4004" : config["VALIDATOR"];

        if (!Uri.TryCreate(validatorAddress, UriKind.Absolute,
out var _))
            throw new Exception($"Invalid validator address:
{validatorAddress}");

        var processor = new
TransactionProcessor(validatorAddress);
        processor.AddHandler(new SupplyChainHandler());
        processor.Start();
    }
}
```

De esta manera, la lógica de negocio queda contenida en el *namespace* 'Domain' donde podemos encontrar las entidades y la interfaz de repositorio que actúa como un contrato de las operaciones que debe poder soportar el programa. A continuación, vemos el ejemplo para la clase 'Item.cs' y la interfaz 'IRepository':

```
public class Item
{
    public string Id { get; set; }
    public string OwnerPublicKey { get; set; }
    public DateTimeOffset Timestamp { get; set; }

    public bool IsOwner(string publicKey) => publicKey ==
OwnerPublicKey;

    public void TransferOwnership(User user) => OwnerPublicKey
= user.PublicKey;
}
```

```
public interface IRepository
{
    Task SetUser(User user);
    Task SetItem(Item item);
    Task SetTransfer(Transfer transfer, Item item);
    Task<User> GetUser(string publicKey);
    Task<Item> GetItem(string itemId);
    Task<Transfer> GetTransfer(string itemIdId, DateTimeOffset
timestamp);
}
```

Una vez que hemos implementado esto, podemos comprobar que la clase 'Program.cs' aparece dentro del *namespace* 'EventHandler' que nos permite escuchar los eventos que son emitidos al hacerse un *commit* y reenviarlos a un endpoint de Elasticsearch donde serán indexados. Es necesario añadir el paquete NuGet NEST para Elasticsearch y NetMQ para utilizar la librería de mensajería.

```
class Program
{
   private static readonly IConfigurationRoot config = new
ConfigurationBuilder()
        .AddJsonFile("appsettings.json", optional: true)
        .AddEnvironmentVariables()
        .Build();
   private static readonly string validatorAddress =
config["VALIDATOR"];
   private static readonly string elasticSearchApi =
config["ELASTICSEARCH"];

   static async Task Main(string[] args)
   {
        using var dealer = new DealerSocket();
        dealer.Connect(validatorAddress);
        SubscribeToEvents(dealer);
        var client = new ElasticClient(new
ConnectionSettings(new Uri(elasticSearchApi)));
        while (true)
        {
            var item = RetrieveItemFromEvent(dealer);
            await client.IndexAsync(item, i =>
i.Index("items"));
        }
   }

   private static void SubscribeToEvents(DealerSocket dealer)
   {
        var subscription = new EventSubscription
        {
            EventType = "item/create"
        };

        var filter = new EventFilter
        {
            Key = "address",
            MatchString = ".*",
            FilterType = EventFilter.Types.FilterType.RegexAny
        };

        subscription.Filters.Add(filter);
        var request = new ClientEventsSubscribeRequest();
        request.Subscriptions.Add(subscription);

        var correlationId = Guid.NewGuid().ToString();
        var outgoingMsg = new Message
        {
```

```
            CorrelationId = correlationId,
            MessageType =
Message.Types.MessageType.ClientEventsSubscribeRequest,
            Content = request.ToByteString()
        };

        var zmqMessage = new NetMQMessage();
        zmqMessage.Append(outgoingMsg.ToByteArray());
        dealer.SendMultipartMessage(zmqMessage);

        var responseStrings = dealer.ReceiveMultipartStrings();
    }

    private static Item RetrieveItemFromEvent(DealerSocket
dealer)
    {
        var obj = dealer.ReceiveMultipartMessage();
        var msg =
Message.Parser.ParseFrom(obj.First.ToByteArray());
        var resp =
ClientEventsGetRequest.Parser.ParseFrom(msg.Content.ToByteArra
y());
        var itemJson =
resp.Subscriptions.First().Filters.Single(f => f.Key ==
"item").MatchString;
        var item = JsonSerializer.Deserialize<Item>(itemJson);
        Console.WriteLine(itemJson);
        return item;
    }
}
```

Por último, para que nuestra app pueda leer la información de Elasticsearch añadimos las siguientes dependencias en el archivo ‘gradle’ del módulo ‘app’:

```
implementation 'com.squareup.retrofit2:retrofit:2.3.0'
implementation 'com.squareup.retrofit2:converter-gson:2.3.0'
implementation 'com.squareup.okhttp3:logging-interceptor:3.9.1'
```

A continuación creamos una interfaz que contiene las llamadas a los métodos de HTTP para servicios RESTful:

```
import retrofit2.Call
import retrofit2.http.GET

interface ElasticsearchApiService {

  @GET("/items/_search")
  fun getItems(): Call<ItemsResponse>
}
```

Y para finalizar hacemos la llamada para obtener la lista de ítems de la siguiente forma:

```
internal fun getItemsList() {
  val retrofit = Retrofit.Builder()
      .baseUrl(BaseUrl)
      .addConverterFactory(GsonConverterFactory.create())
      .build()
  val service = retrofit.create(ElasticsearchApiService::class.java)
  val call = service.getItems()

  call.enqueue(object : Callback<ItemsResponse> {
      override fun onResponse(call: Call<ItemsResponse>, response:
Response<ItemsResponse>) {
          if (response.code() == 200) {

              val itemsResponse = response.body()!!
              val gson = Gson()
              val json = gson.toJson(itemsResponse)
              val response = gson.fromJson(json, ItemsResponse::class.java)
              val items = response.hits?.get("hits")?.asJsonArray
              itemsList = items?.toList() as List<JsonObject>
              setupRecyclerView(findViewById(R.id.item_list))
          }
      }

      override fun onFailure(call: Call<ItemsResponse>, t: Throwable) {
          Log.d("JSON: ", t.message.toString());
      }
  })
}
```

6. Implantación

Para la puesta en marcha y funcionamiento del software desarrollado hemos utilizado Docker (salvo el cliente de la Raspberry, cuya instalación se describe más adelante). Esta parte consta de tres ficheros: un 'docker compose' (.yml o .yaml) y dos 'dockerfiles'. El primero se encuentra en el directorio raíz de nuestra solución y en él están descritos los servicios que necesitan ser puestos en funcionamiento así como las dependencias entre ellos, el contenedor en el que se encuentran, los puertos que utilizan, los *entrypoints* que ejecutan, etc.

```
version: "2.1"

services:

 elasticsearch:
   image: docker.elastic.co/elasticsearch/elasticsearch:7.9.0
   container_name: elasticsearch
```

```
    environment:
      - discovery.type=single-node
    ports:
      - "9200:9200"
      - "9300:9300"

  kibana:
    image: docker.elastic.co/kibana/kibana:7.9.0
    depends_on:
      - elasticsearch
    environment:
      SERVER_NAME: kibana
      ELASTICSEARCH_HOSTS: http://elasticsearch:9200
    ports:
      - "5601:5601"

  settings-tp:
    image: hyperledger/sawtooth-settings-tp:1.0
    container_name: sawtooth-settings-tp-default
    depends_on:
      - validator
    entrypoint: settings-tp -vv -C tcp://validator:4004

  validator:
    image: hyperledger/sawtooth-validator:1.0
    container_name: sawtooth-validator-default
    expose:
      - 4004
    ports:
      - "4004:4004"
    entrypoint: "bash -c \"\
        sawadm keygen && \
        sawtooth keygen my_key && \
        sawset genesis -k /root/.sawtooth/keys/my_key.priv && \
        sawadm genesis config-genesis.batch && \
        sawtooth-validator -vv \
          --endpoint tcp://validator:8800 \
          --bind component:tcp://eth0:4004 \
          --bind network:tcp://eth0:8800 \
        \""

  rest-api:
    image: hyperledger/sawtooth-rest-api:1.0
    container_name: sawtooth-rest-api-default
    ports:
      - "8008:8008"
    depends_on:
      - validator
    entrypoint: sawtooth-rest-api -C tcp://validator:4004 --
bind rest-api:8008

  shell:
    image: hyperledger/sawtooth-all:1.0
    container_name: sawtooth-shell-default
    depends_on:
```

```
      - rest-api
    entrypoint: "bash -c \"\
        sawtooth keygen && \
        tail -f /dev/null \
        \""

  supply-chain-tp:
    image: supply-chain-tp
    container_name: supply-chain-tp
    depends_on:
      - validator
    environment:
      - VALIDATOR=tcp://validator:4004

  supply-chain-event-handler:
    image: supply-chain-event-handler
    container_name: supply-chain-event-handler
    depends_on:
      - validator
      - elasticsearch
    environment:
      - VALIDATOR=tcp://validator:4004
      - ELASTICSEARCH=http://elasticsearch:9200
```

Todos los servicios nos vienen ya dados salvo 'supply-chain-tp' y 'supply-chain-event-handler'. Para ello, hemos creado un fichero 'Dockerfile' en la carpeta del 'Processor' y del 'EventHandler', respectivamente, que se muestran a continuación.

```
FROM mcr.microsoft.com/dotnet/core/runtime:3.1
COPY publish/ App/
WORKDIR /App
ENTRYPOINT ["dotnet", "Processor.dll"]
```

```
FROM mcr.microsoft.com/dotnet/core/runtime:3.1
COPY publish/ App/
WORKDIR /App
ENTRYPOINT ["dotnet", "EventHandler.dll"]
```

Una vez ejecutemos el comando 'docker-compose up', el programa estará listo para que nuestro cliente pueda interactuar con él. Seguidamente se describe la puesta en marcha de este en nuestro dispositivo IoT. Para empezar, es necesario hacer un *publish* del proyecto 'RaspberryClient'. Seguidamente, y suponiendo que ya tenemos la Raspberry con el SO instalado y configurada para tener acceso a internet, creamos una carpeta compartida donde incluiremos la carpeta que acabamos de publicar del programa cliente (para una descripción detallada, ver [Emmet19]). Una vez hecho esto, sólo queda abrir un terminal, navegar hasta la carpeta y ejecutar 'dotnet run'.

7. Pruebas

En este apartado se describe el proceso de validación seguido para comprobar que el *software* desarrollado funciona correctamente. El primer paso, ya descrito en el apartado anterior, consiste en ejecutar el comando 'docker-compose up' dentro del directorio donde se encuentra nuestra solución. Veremos algo similar a la siguiente imagen, que proporciona las garantías básicas de funcionamiento.

```
aliciamonleon@MacBook-Pro-de-Alicia Supply Chain Traceability System % docker-compose up
Creating network "supplychaintraceabilitysystem_default" with the default driver
Creating sawtooth-validator-default ... done
Creating elasticsearch              ... done
Creating supply-chain-tp                        ... done
Creating sawtooth-settings-tp-default           ... done
Creating sawtooth-rest-api-default              ... done
Creating supply-chain-event-handler             ... done
Creating supplychaintraceabilitysystem_kibana_1 ... done
Creating sawtooth-shell-default                 ... done
Attaching to sawtooth-validator-default, elasticsearch, supply-chain-tp, supply-chain-even
t-handler, sawtooth-rest-api-default, sawtooth-settings-tp-default, supplychaintraceabilit
ysystem_kibana_1, sawtooth-shell-default
```

Figura 18.

Una forma de comprobar el estado de los contenedores que acabamos de levantar es utilizar la interfaz gráfica que nos proporciona Docker.

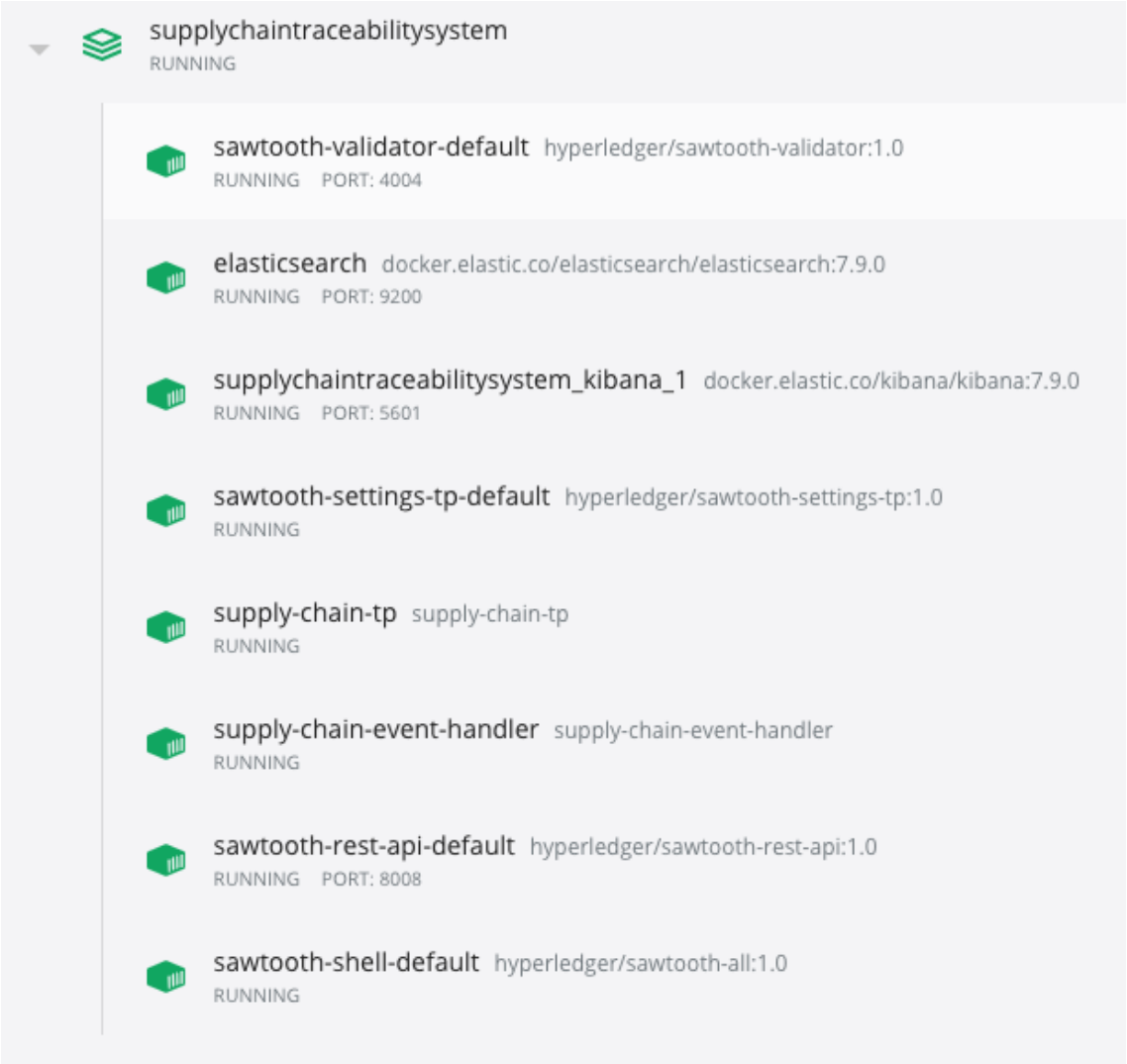

Figura 19

Una vez este paso es completado ejecutamos el programa cliente como ya se ha explicado anteriormente. A continuación podemos introducir nuestro código y la consola nos devolverá una dirección http.

```
aliciamonleon@MacBook-Pro-de-Alicia RaspberryClient % dotnet run
Enter barcode:
TestItem1
{
  "link": "http://127.0.0.1:8008/batch_statuses?id=2be7ce298a28137
}
Enter barcode:
```

Figura 20

Si la abrimos con el navegador podemos ver el estado del lote. Hay cuatro estados posibles: ‘COMMITTED’, ‘INVALID’, ‘PENDING’, y ‘UNKNOWN’. Si todo es correcto deberá aparecernos el primero.

```
{
  "data": [
    {
      "id": "2be7ce298a2813709b903853d3ae365e7231437abf354d
      "invalid_transactions": [],
      "status": "COMMITTED"
    }
  ],
  "link": "http://127.0.0.1:8008/batch_statuses?id=2be7ce29
}
```


Figura 21

Ahora vamos a comprobar que los datos almacenados en Sawtooth también han sido almacenados en Elasticsearch. Para hacerlo es tan sencillo como dirigirnos a la aplicación de consola que proporciona Kibana en http://localhost:5601/app/dev_tools#/console e introducimos el comando 'GET items/_search' obtenemos como respuesta un JSON con los ítems que hayamos introducido y la información que incluyen.


```
{
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "items",
        "_type" : "_doc",
        "_id" : "TestItem1",
        "_score" : 1.0,
        "_source" : {
          "id" : "TestItem1",
          "ownerPublicKey" :
            "04e49dc25d8c7236810550a00a163502188f9513bca6a12ccc3811a84c7d0c47498767fab
            2d3ee46a9621b69fad1ae3dea0194884f78756076a5bbd5bd41ab9006",
          "timestamp" : "2020-09-01T14:43:32.4191820+00:00"
        }
      }
    ]
  }
}
```


Figura 22

Por último, queda comprobar que nuestra app es capaz de obtener estos mismos datos y mostrarlos al usuario de una forma más *user friendly*. Para ello

podemos ejecutarla en un móvil o emulador. En nuestro caso el resultado es el que se muestra en la siguiente captura de pantalla.

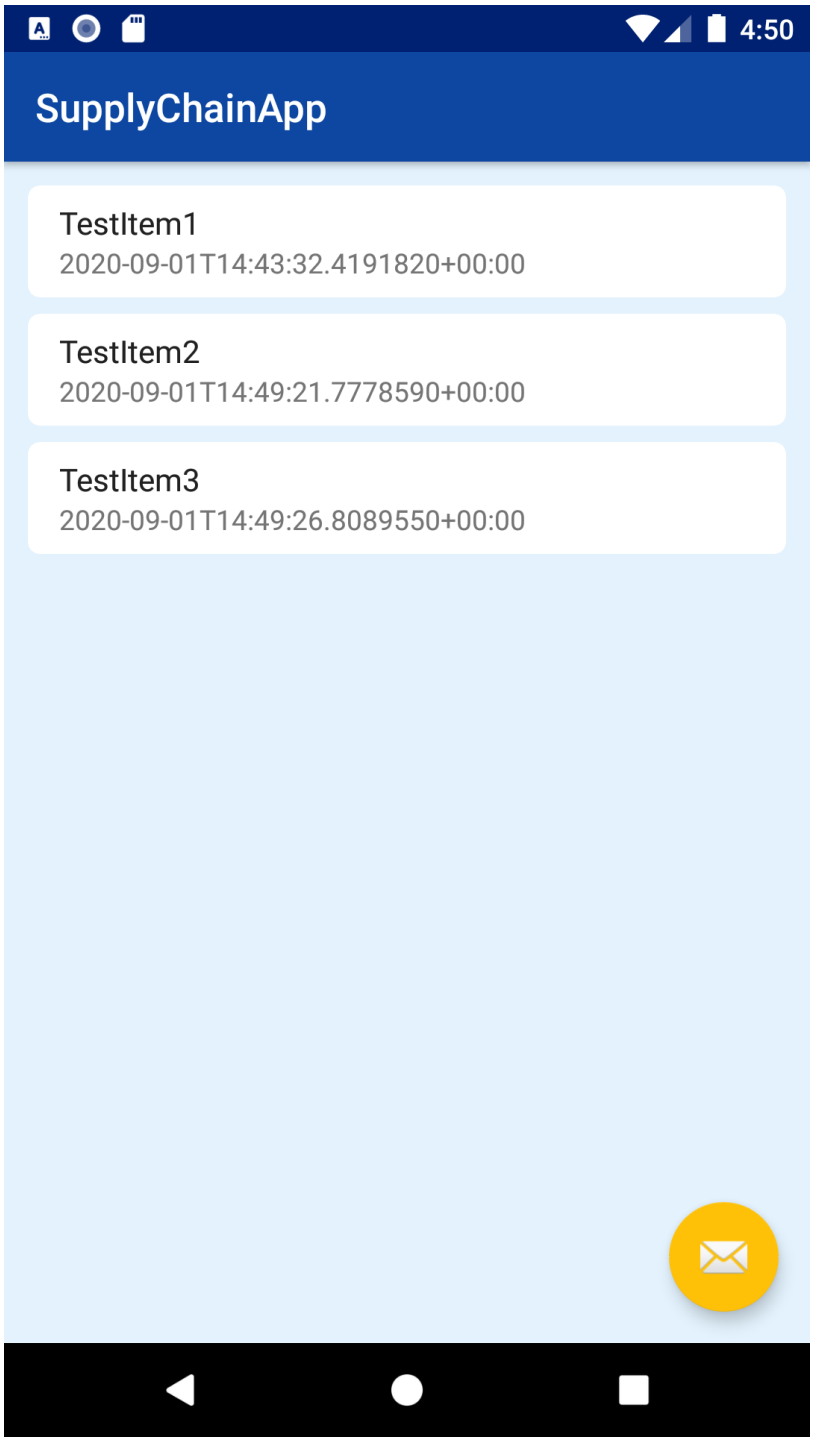

Figura 23

Si bien esto es suficiente para comprobar que la solución funciona en un escenario restringido como el aquí planteado, en ningún caso esto garantiza que su integración en un programa de mayor complejidad sería correcta y válida en cualquier escenario. Antes de poner en producción el software, sería extremadamente importante realizar tanto tests unitarios como test de integración, pruebas de

componentes, etc, resultando muy conveniente automatizar todas estas pruebas, si bien este proceso trasciende el alcance de este trabajo académico

8. Conclusiones

En el mundo en el que nos encontramos, donde organismos tanto públicos como privados ponen cada vez más de manifiesto la importancia de los datos (sin olvidar que una inmensa mayoría pertenecen a personas las cuales exigen ser soberanos y soberanas de los mismos), su obtención, la seguridad y la escalabilidad, son tres de los mayores retos que se plantean. En este trabajo hemos expuesto cómo incrementar significativamente la seguridad mediante la tecnología *blockchain*, la escalabilidad mediante los motores de búsqueda indexada y su obtención a través de dispositivos IoT, si bien estos dos últimos se han tratado más de un modo conceptual que profundizando en detalles técnicos. De acuerdo con los objetivos primarios de este trabajo, orientados a entender la *blockchain* desde varios puntos de vista, se ha proporcionado una panorámica acerca de sus orígenes y su historia; se ha tratado desde las perspectivas sociales, económicas y medioambientales; se ha investigado el mercado; se ha llevado a cabo un análisis sobre un contexto particular de su aplicación; y se ha desarrollado un proyecto práctico en el que se aborda la creación de una arquitectura de *software* que permite su integración con otras dos tecnologías de vanguardia

La mayor dificultad que tuve fue al elegir el lenguaje de programación. Consideré hacerlo en Python por ser el utilizado en la documentación y el SDK disponible de mayor madurez según pone en la documentación oficial, y por tanto, había mucho más material en internet en el que apoyarse. No obstante, después de valorar esta opción, debido a varias dificultades con los *environments* y las diferentes versiones de Python, decidí desarrollarlo en C#. Me pareció una opción interesante ya que no existe apenas documentación y esto podía suponer un valor añadido, todo lo opuesto al SDK de Python. Además resultó bastante sencillo usar la implementación para .NET, con las ventajas que esta plataforma, ya madura, aporta por sí misma. Por otro lado, hace años ya que empecé a trastear con Raspberry y Arduino, sin embargo apenas sabía sobre bases de datos NoSQL o no relacionales como el motor de búsqueda de Elasticsearch. Ahora conozco más sobre estas bases de datos y seguiré aprendiendo para, aunque no sea mi especialidad, saber cuales son las características de los diferentes tipos que engloba y cual es la adecuada en cada contexto. Al mismo tiempo, seguiré aprendiendo más acerca de lo que hasta ahora considero mis especialidades: aplicaciones *blockchain* con Hyperledger y aplicaciones móviles en Android.

8.1. Relación del trabajo desarrollado con los estudios cursados

Durante el Grado en Ingeniería Informática hemos aprendido a utilizar distintos lenguajes y herramientas que hemos utilizado a lo largo de este trabajo. Desde que entramos en primero aprendimos a programar en Java que me ha servido para desarrollar aplicaciones en Android. Esto me ha conducido a aprender Kotlin y librerías

como Retrofit. Un par de cursos después aprendí C# y UML en la asignatura de Ingeniería del Software. Ese mismo semestre también estudiamos los *middleware*, más concretamente ZeroMQ, y despliegue de servicios con Docker los cuales se emplean en este proyecto, además de conceptos como algoritmos de consenso, escalabilidad, etc. La especialidad que he cursado, Tecnologías de la Información, me ha aportado una gran cantidad de conocimiento que ha sido imprescindible, como por ejemplo Integración de Aplicaciones y Sistemas y Servicios en Red. Estas asignaturas junto con el resto me han proporcionado competencias como:

- Capacidad para comprender el entorno de una organización y sus necesidades en el ámbito de las tecnologías de la información y las comunicaciones.
- Saber aplicar los conocimientos al trabajo o vocación de una forma profesional y poseer las competencias que suelen demostrarse por medio de la elaboración y defensa de argumentos y la resolución de problemas dentro del área de estudio.
- Capacidad para seleccionar, diseñar, desplegar, integrar y gestionar redes e infraestructuras de comunicaciones.
- Razonar de manera abstracta, analítica y crítica, sabiendo elaborar y defender argumentos en su área de estudio y campo profesional.
- Aprender de manera autónoma nuevos conocimientos y técnicas adecuadas para la concepción, el desarrollo, la evaluación o la explotación de sistemas informáticos.
- Capacidad de concebir sistemas, aplicaciones y servicios basados en tecnologías de red, incluyendo Internet, web y móvil.
- Conocimiento y aplicación de las características, funcionalidades y estructura de los Sistemas Distribuidos, las Redes de Computadores e Internet y diseñar e implementar aplicaciones basadas en ellas.
- Localizar información relevante desde diferentes fuentes e investigar las novedades tecnológicas en su ámbito de trabajo y en áreas afines.

9. Trabajos futuros

Una posible ampliación que se puede llevar a cabo y que no se ha realizado debido al aumento de complejidad que supone es la implementación de los roles y la configuración de permisos. Otra mejora a considerar sería incrementar las funciones de interacción y muestra de datos por la parte *front-end*. También sería muy interesante para un trabajo futuro el estudio y desarrollo práctico de Sawtooth desde el punto de vista del administrador, donde se trate la configuración de la red y despliegue en producción.

10. Referencias

Afghan Girls Can Now 'Participate in the Global Economy' with Bitcoin & Learning Code. (2015, 19 junio). Cointelegraph.

https://cointelegraph.com/news/afghan-girls-can-now-participate-in-the-global-economy-with-bitcoin-learning-code

Asano, T. (2020, 12 mayo). *Comparison of Several Types of Blockchains (Public・Private・Consortium)*. Medium. https://medium.com/cosmosgaminghub/blockchain-comparison-51f881c8399f

[Avan-Nomayo20] Avan-Nomayo, O. (2020, 28 enero). *Retención de los usuarios: El Santo Grial para las DApps que se mueven más allá del estatus de palabra de moda.* Cointelegraph. https://es.cointelegraph.com/news/user-retention-the-holy-grail-for-dapps-moving-beyond-buzzword-status

[101 Blockchains20] *Best Blockchain Infographics Collection*. (2020, 8 agosto). 101 Blockchains. https://101blockchains.com/blockchain-infographics/

[Forética18] *Blockchain: una cadena de bloques transparente y segura para una economía circular y eficiente*. (2018, 18 diciembre). Forética. https://foretica.org/blockchain-una-cadena-de-bloques-transparente-y-segura-para-una-economia-circular-y-eficiente/

Cointelegraph. (2020, 1 enero). *Qué es Ripple - Últimas noticias sobre Ripple*. https://es.cointelegraph.com/tags/ripple

colaboradores de Wikipedia. (2019, 11 octubre). *Cadena de suministro*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Cadena_de_suministro

colaboradores de Wikipedia. (2020a, marzo 31). *Stock-keeping unit*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Stock-keeping_unit

colaboradores de Wikipedia. (2020b, abril 16). *Command–query separation*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Command%E2%80%93query_separation

colaboradores de Wikipedia. (2020c, abril 16). *Gson*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Gson

colaboradores de Wikipedia. (2020d, julio 6). *Clasificación arancelaria*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Clasificaci%C3%B3n_arancelaria

colaboradores de Wikipedia. (2020e, agosto 3). *ZeroMQ*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/ZeroMQ

colaboradores de Wikipedia. (2020f, agosto 4). *SOLID*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/SOLID

colaboradores de Wikipedia. (2020g, agosto 10). *Proxy (patrón de diseño)*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Proxy_(patr%C3%B3n_de_dise%C3%B1o)

colaboradores de Wikipedia. (2020h, agosto 12). *Docker (software)*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Docker_(software)

colaboradores de Wikipedia. (2020i, agosto 15). *Android*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Android

colaboradores de Wikipedia. (2020j, agosto 23). *Raspberry Pi*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Raspberry_Pi

colaboradores de Wikipedia. (2020k, agosto 27). *Objetivos de Desarrollo Sostenible*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Objetivos_de_Desarrollo_Sostenible

Consuming APIs with Retrofit | *CodePath Android Cliffnotes*. (s. f.). Codepath. Recuperado 3 de septiembre de 2020, de https://guides.codepath.com/android/consuming-apis-with-retrofit

[Emmet19] Emmet. (2019, 10 julio). *How to Setup a Raspberry Pi Samba Server*. Pi My Life Up. https://pimylifeup.com/raspberry-pi-samba/

[Blockchain | IT Trends19] *El mercado de soluciones blockchain crecerá rápidamente hasta 2023*. (2019, 12 agosto). Blockchain | IT Trends. https://www.ittrends.es/blockchain/2019/08/el-mercado-de-soluciones-blockchain-crecera-rapidamente-hasta-2023

[Elastic20] Elastic. (s. f.). *¿Qué es Elasticsearch?* Recuperado 3 de septiembre de 2020, de https://www.elastic.co/es/what-is/elasticsearch

Fm, Y. (2017, 13 septiembre). *The DAO y el caso del robo de los 50 millones de dólares en Ethereum (Insert Coin 1x01)*. Xataka. https://www.xataka.com/seguridad/the-dao-y-el-caso-del-robo-de-los-50-millones-de-dolares-en-ethereum-insert-coin-1x01#:%7E:text=ENTRETENIMIENTO-,The%20DAO%20y%20el%20caso%20del%20robo%20de%20los%2050,en%20Ethereum%20(Insert%20Coin%201x01)&text=El%20pasado%20julio%20un%20usuario,como%20una%20alternativa%20al%20Bitcoin.

[Fresno18] Fresno, B. G. (2018, 26 septiembre). *¿Cuál es la diferencia entre una DLT y «blockchain»?* BBVA NOTICIAS. https://www.bbva.com/es/diferencia-dlt-blockchain/

Gichigi, T. (2020, 31 agosto). *A brief history of blockchain - Coinmonks*. Medium. https://medium.com/coinmonks/a-brief-history-of-blockchain-70c519d3053

Gupta, V. (2018, 1 noviembre). *Why Global Trade Will Inevitably Move to the Blockchain*. Medium. https://medium.com/humanizing-the-singularity/why-global-trade-will-inevitably-move-to-the-blockchain-ab3f66bd20f8

H. (s. f.). *hyperledger/education-sawtooth-simple-supply*. GitHub. Recuperado 3 de septiembre de 2020, de https://github.com/hyperledger/education-sawtooth-simple-supply

[Hyperledger Sawtooth20] *Hyperledger Sawtooth | Hyperledger Sawtooth*. (s. f.). Hyperledger. Recuperado 3 de septiembre de 2020, de https://sawtooth.hyperledger.org/

Hyperledger Sawtooth for Application Developers. (s. f.). LearnThings. Recuperado 3 de septiembre de 2020, de https://learnthings.online/course/2020/03/06/hyperledger-sawtooth-for-application-developers

Informe OBS: La tecnología Blockchain trae una seguridad y una transparencia que no se había conocido hasta ahora | OBS Business School. (s. f.). OBS Business School. Recuperado 3 de septiembre de 2020, de

https://obsbusiness.school/es/informe-de-investigacion/informe-obs-la-tecnologia-blockchain-trae-una-seguridad-y-una-transparencia-que-no-se-habia-conocido-hasta-ahora

Isuamfon. (2020, 30 julio). *The Complete History Of Bitcoin: How Satoshi, the World's First Decentralized Came To Be*. Medium. https://medium.com/@IWILLTEACHUCRYPTO/the-complete-history-of-bitcoin-how-satoshi-the-worlds-first-decentralized-came-to-be-d5c0ef1cb067

Kasireddy, P. (2020, 14 enero). *How does Ethereum work, anyway?* Medium. https://medium.com/@preethikasireddy/how-does-ethereum-work-anyway-22d1df506369

[Lecuit19] Lecuit, J. A. (2019, 12 noviembre). *La seguridad y la privacidad del blockchain, más allá de la tecnología y las criptomonedas*. Real Instituto Elcano. http://www.realinstitutoelcano.org/wps/portal/rielcano_es/contenido?WCM_GLOBAL_CONTEXT=%2Felcano%2Felcano_es%2Fzonas_es%2Fari106-2019-alonsolecuit-seguridad-y-privacidad-del-blockchain-mas-alla-de-tecnologia-y-criptomoneda

[Milton13] *Milton Friedman, Land value tax and internet currencies*. (2013, 26 abril). [Vídeo]. YouTube. https://www.youtube.com/watch?v=j2mdYX1nF_Y

Muñoz, A. (2019, 9 septiembre). *¿Qué es un SKU? Todo lo que debes saber*. Saleslayer. https://blog.saleslayer.com/es/que-es-un-sku-todo-lo-que-debes-saber

Nogales, B. (2019, 8 agosto). *¿Qué es Tether (USDT)? La stablecoin más importante*. bitcoin.es. https://bitcoin.es/criptomonedas/que-es-tether/

[EAEBusinessSchool20] *¿Qué es el blockchain y qué ventajas aporta a las empresas?* (s. f.). EAE Business School. Recuperado 3 de septiembre de 2020, de https://www.eaeprogramas.es/blog/negocio/finanzas-economia/que-es-el-blockchain-y-que-ventajas-aporta-las-empresas

Rodriguez, N. (2018, 1 diciembre). *Hyperledger vs Corda R3 vs Ethereum: La guía definitiva*. 101 Blockchains.

https://101blockchains.com/es/hyperledger-vs-corda-r3-vs-ethereum-la-guia/

Rodriguez, N. (2019a, enero 17). *La mejor comparación de los tipos de registros distribuidos: Blockchain vs Hashgraph vs Dag vs Holochain*. 101 Blockchains. https://101blockchains.com/es/blockchain-vs-dag-vs-hashgraph-vs-holochain/

Rodriguez, N. (2019b, agosto 18). *Los mejores proyectos Blockchain de código abierto*. 101 Blockchains. https://101blockchains.com/es/blockchain-de-codigo-abierto/

Rodriguez, N. (2019c, septiembre 2). *Uso de Blockchain: lista de 20+ casos de uso de la tecnología Blockchain*. 101 Blockchains. https://101blockchains.com/es/uso-de-blockchain/

Roldán, P. N. (2020, 9 julio). *Cadena de suministro*. Economipedia. https://economipedia.com/definiciones/cadena-de-suministro.html

Ruiz, A. (2020, 2 julio). *¿Blockchain qué es y qué ventajas tiene?* Tecnología para los negocios. https://ticnegocios.camaravalencia.com/servicios/tendencias/blockchain-que-es-y-que-ventajas-tiene/

Sawtooth –. (2020, 23 abril). Hyperledger. https://www.hyperledger.org/use/sawtooth

Schwarz, C. (2019, 3 septiembre). *Ethereum 2.0: A Complete Guide. Ewasm. - ChainSafe*. Medium. https://medium.com/chainsafe-systems/ethereum-2-0-a-complete-guide-ewasm-394cac756baf

Sedgwick, K. (2019, 1 marzo). *Bitcoin History Part 10: The 184 Billion BTC Bug*. Bitcoin News. https://news.bitcoin.com/bitcoin-history-part-10-the-184-billion-btc-bug/

[Singh19] Singh, N. (2019, 2 julio). *Blockchain Usage: List of 20+ Blockchain Technology Use Cases*. 101 Blockchains. https://101blockchains.com/blockchain-usage/

Szabo, N. (2005, 29 diciembre). *Bit Gold | Satoshi Nakamoto Institute*. Nakamoto Institute. https://nakamotoinstitute.org/bit-gold/

Table of Contents — Sawtooth v1.0.5 documentation. (s. f.). Hyperledger. Recuperado 3 de septiembre de 2020, de https://sawtooth.hyperledger.org/docs/core/releases/1.0/contents.html

Teach, Learn, and Make with Raspberry Pi – Raspberry Pi. (s. f.). Raspberrypi. Recuperado 3 de septiembre de 2020, de https://www.raspberrypi.org/

Tipo de Cambio de Divisas. (2013, 26 mayo). Mundomercados. http://mundomercados.blogspot.com/2013/05/tipo-de-cambio-de-divisas.html

Vinay Gupta at Michel Bauwens & the Promise of the Blockchain. (2016, 25 febrero). [Vídeo]. Vimeo. https://vimeo.com/161183966

Where tech, women, and the future of finance converge. (2018, 1 agosto). Medium. https://blog.coinbase.com/fereshteh-forough-interview-1e73a684788f?gi=7209cd1bd448

Wikipedia contributors. (2020, 3 septiembre). *Kotlin (programming language)*. Wikipedia. https://en.wikipedia.org/wiki/Kotlin_(programming_language)

[Xie18] Xie, L. (2018, 19 junio). *A beginner's guide to Ethereum - The Coinbase Blog*. Medium. https://blog.coinbase.com/a-beginners-guide-to-ethereum-46dd486ceecf?gi=fd14dfe54479

Glosario

***Accessor*:** Actor de una *blockchain* con derecho a leer y conservar una copia de la cadena.
Apache Lucene: API de código abierto para recuperación de información, originalmente implementada en Java. Es útil para cualquier aplicación que requiera indexado y búsqueda a texto completo.
Back-end: Término de ingeniería de software que hace referencia a la capa de acceso a datos.
***Backup*:** En ciencias de la información e informática es una copia de seguridad de los datos originales que se realiza con el fin de disponer de un medio para recuperarlos en caso de su pérdida.
Bitcoin: Protocolo, proyecto de código abierto y red *peer-to-peer* que se utiliza como criptomoneda, sistema de pago y mercancía.

***Blockchain*:** Estructura de datos cuya información se agrupa en conjuntos (bloques) a los que se les añade metainformaciones relativas a otro bloque de la cadena anterior en una línea temporal.
Cartera digital: También conocida como billetera electrónica o e-Wallet, se refiere a un dispositivo electrónico, un servicio de banca móvil o una aplicación móvil que permite a una parte realizar transacciones electrónicas con otra parte que intercambia unidades de moneda digital por bienes y servicios.
***Cluster*:** Se aplica a los sistemas distribuidos de granjas de computadoras unidos entre sí normalmente por una red de alta velocidad y que se comportan como si fuesen un único servidor.
Criptografía: Se ha definido, tradicionalmente, como el ámbito de la criptología que se ocupa de las técnicas de cifrado o codificado destinadas a alterar las representaciones lingüísticas de ciertos mensajes con el fin de hacerlos ininteligibles a receptores no autorizados.
Criptomoneda: También conocida como criptodivisa (del inglés cryptocurrency) o criptoactivo es un medio digital de intercambio que utiliza criptografía fuerte para asegurar las transacciones, controlar la creación de unidades adicionales y verificar la transferencia de activos usando tecnologías de registro distribuido.
***Crowdfunding*:** Mecanismo colaborativo de financiación de proyectos desarrollado sobre la base de las nuevas tecnologías. Prescinde de la tradicional intermediación financiera y consiste en poner en contacto a promotores de proyectos que demandan fondos mediante la emisión de valores y participaciones sociales o mediante la solicitud de préstamos, con inversores u ofertantes de fondos que buscan en la inversión un rendimiento.
DIP: Principio de inversión de la dependencia (*Dependency Inversion Principle*).
Docker: Proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos.
ECDSA: *Elliptic Curve Digital Signature Algorithm* es una modificación del algoritmo DSA que emplea operaciones sobre puntos de curvas elípticas en lugar de las exponenciaciones que usa DSA.
Elastic Stack: Grupo de productos de código abierto de Elastic diseñado para ayudar a los usuarios a tomar datos de cualquier tipo de fuente y en cualquier formato y buscar, analizar y visualizar esos datos en tiempo real.
Elasticsearch: Servidor de búsqueda basado en Lucene que provee un motor de búsqueda de texto completo, distribuido y con capacidad de multitenencia con una interfaz web RESTful y con documentos JSON.
Ethereum: Cadena de bloques de código abierto descentralizada con funcionalidad de contrato inteligente.
***Front-end*:** Término de ingeniería de software que hace referencia a la capa de presentación de datos.
***Full-stack*:** Un desarrollador *full-stack* es un desarrollador o ingeniero que trabaja tanto con el *front-end* como con el *back-end*.
Geek: Término que se utiliza para referirse a la persona fascinada por la tecnología y la informática.
***Gossip protocol*:** Procedimiento o proceso de comunicación entre pares por computadora que se basa en la forma en que se propagan las epidemias.

Hackear: Acceder sin autorización a computadoras, redes o sistemas informáticos, o a sus datos.
***Hard fork*:** Actualización importante del protocolo que obliga a todos los usuarios a pasar al nuevo software si quieren seguir usando la misma cadena de bloques.
Hash: Una función criptográfica hash- usualmente conocida como "hash"- es un algoritmo matemático que transforma cualquier bloque arbitrario de datos en una nueva serie de caracteres con una longitud fija.
HTTP: Protocolo de transferencia de hipertexto (en inglés, *Hypertext Transfer Protocol*) es el protocolo de comunicación que permite las transferencias de información en la World Wide Web.
Hyperledger Sawtooth: Plataforma blockchain empresarial para crear redes y aplicaciones de contabilidad distribuida.
IoT: El término IoT (del inglés *Internet of Things*) hace referencia a los sistemas de dispositivos físicos que reciben y transfieren datos a través de redes inalámbricas sin la intervención humana.
ISP: Principio de segregación de la interfaz (*Interface Segregation Principle*).
JSON: Acrónimo de JavaScript Object Notation, (notación de objeto de JavaScript) es un formato de texto sencillo para el intercambio de datos.
Kernel: En informática, un núcleo o kernel es un software que constituye una parte fundamental del sistema operativo, y se define como la parte que se ejecuta en modo privilegiado.
Kibana: Panel de visualización de datos de código abierto para Elasticsearch.
Linux: Familia de sistemas operativos de código abierto tipo Unix basados en el núcleo de Linux, un núcleo del sistema operativo lanzado por primera vez el 17 de septiembre de 1991 por Linus Torvalds.
***Listener*:** En el proceso de gestión de eventos, es el objeto de la clase que implementa la interfaz de escucha y que contiene el método de respuesta al evento.
LSP: Principio de sustitución de Liskov (*Liskov Substitution Principle*).
***Merkle Tree*:** Un árbol hash de Merkle (en inglés *Merkle Hash Tree*) o árbol de merkle o árbol hash es una estructura de datos en árbol, binario o no, en el que cada nodo que no es una hoja está etiquetado con el hash de la concatenación de las etiquetas o valores (para nodos hoja) de sus nodos hijo.
***Middleware*:** Lógica de intercambio de información entre aplicaciones es un *software* que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, o paquetes de programas, redes, *hardware* o sistemas operativos.
***Miner*:** Actor de una *blockchain* que valida una transacción y crea bloques aplicando reglas de blockchain para la "aceptación" de la comunidad.
Nakamoto: Satoshi Nakamoto es la persona o grupo de personas que crearon el protocolo Bitcoin y su software de referencia.
NoSQL: Método de almacenamiento no necesariamente estructurado que no tiene por qué almacenarse en tablas.
OCP: Principio de abierto/cerrado (*Open/Closed Principle*).
OkHttp: OkHttp es un cliente HTTP y HTTP/2 eficiente para aplicaciones de Android y Java.
***Open source*:** *Software* cuyo código fuente y otros derechos que normalmente son exclusivos para quienes poseen los derechos de autor, son publicados bajo una licencia de código abierto o forman parte del dominio público.
P2P: *Peer-to-peer*.

***Participant*:** Actor de una *blockchain* con derecho a realizar entradas.
***Peer-to-peer*:** Una red *peer-to-peer*, red de pares, red entre iguales o red entre pares es una red de ordenadores en la que todos o algunos aspectos funcionan sin clientes ni servidores fijos, sino una serie de nodos que se comportan como iguales entre sí.
***Platform as a Service*:** Plataforma en la que el proveedor de servicios ofrece acceso a un entorno basado en cloud en el cual los usuarios pueden crear y distribuir aplicaciones.
***Plugins*:** Aplicación (o programa informático) que se relaciona con otra para agregarle una función nueva y generalmente muy específica.
***Proof of Work* (PoW):** Mecanismo de consenso que con el fin de desincentivar y dificultar comportamientos indeseados como ataques DDoS o *spam* requiere que el cliente del servicio realice algún tipo de trabajo que tenga cierto coste y que sea verificado fácilmente en la parte del servidor.
Protobuf: *Protocol Buffers* es un método de serialización de datos estructurados.
Raspberry Pi: Serie de ordenadores de placa reducida, ordenadores de placa única u ordenadores de placa simple (SBC) de bajo costo.
Refactorización: (del inglés *refactoring*) es una técnica de la ingeniería de *software* para reestructurar un código fuente, alterando su estructura interna sin cambiar su comportamiento externo.
RESTful: La transferencia de estado representacional (en inglés *Representational State Transfer*) o REST es un estilo de arquitectura *software* para sistemas hipermedia distribuidos como la World Wide Web.
Retrofit: Cliente HTTP con seguridad de tipos para Android y Java.
Secp256k1: Tipo de curva elíptica utilizada como curva ECDSA en el modelo criptográfico de Bitcoin.
***Smart contracts*:** Un contrato inteligente es un programa informático que facilita, asegura, hace cumplir y ejecuta acuerdos registrados entre dos o más partes.
SRP: Principio de responsabilidad única (*Single Responsibility Principle*)
Tether (USDT): Criptomoneda controvertida con tokens emitidos por Tether Limited. Anteriormente afirmó que cada token estaba respaldado por un dólar de los Estados Unidos, pero el 14 de marzo de 2019 cambió el respaldo para incluir préstamos a empresas afiliadas.
ZeroMQ: Biblioteca de comunicaciones de alto rendimiento orientada a mensajes, destinada a la construcción de aplicaciones distribuidas.