UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en ingeniería en Tecnologías y Servicios de Telecomunicación

TRABAJO FIN DE GRADO

Desarrollo de un sistema de monitorización para cadenas de suministro basado en tecnologías de Internet de las cosas y Blockchain

Juan Diego Sierra Fernández
Tutor: Lluís Gifre Renom
Ponente: Sergio López Buedo

Septiembre 2019

Desarrollo de un sistema de monitorización para cadenas de suministro basado en tecnologías de Internet de las cosas y Blockchain

AUTOR: Juan Diego Sierra Fernández
TUTOR: Lluís Gifre Renom

Departamento de Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Septiembre de 2019

Resumen (castellano)

Las tecnologías han cambiado la forma en la que vivimos. Podemos obtener la información en tiempo real de la contaminación que hay en las ciudades, el basurero sabe que un contenedor está lleno antes de recogerlo, controlamos la calefacción de nuestra casa con una aplicación móvil, etc. Estos casos de uso se suelen englobar en el paradigma de "internet de las cosas" (Internet of Things, IoT) que pretende conectar todos los dispositivos y/o máquinas a Internet para que intercambien información y generen beneficio para la sociedad. Para almacenar esta información que se extrae, se utilizan las denominadas bases de datos. Sin embargo, estas últimas no son suficiente cuando se requiere compartir datos entre distintas entidades sin la posibilidad de ser manipulados ilícitamente. Con la llegada de la Cadena de Bloques (Blockchain) se pondrá solución a ese problema.

En este trabajo se desarrollará un caso de uso aplicado a la industria de cadenas de suministro. Para ello, se requiere conocer el panorama de herramientas, protocolos y buenas prácticas esenciales a la hora de desarrollar una aplicación estable que pueda llegar a un entorno de producción real. Se realizará un estudio previo del estado del arte, analizando y entendiendo el funcionamiento de estas tecnologías y las posibilidades que nos brindan para poder desarrollar correctamente el proyecto. Se diseñará la arquitectura teniendo en cuenta las necesidades y servicios que se dan en toda la cadena. Posteriormente, se desarrollará el proyecto con el fin de hacer una demostración real a un posible cliente. Por un lado, se implementará el núcleo del aplicativo, encargado de leer los datos que envían los dispositivos y almacenarlos de forma segura. Por otro lado, se ha usado un pequeño ordenador con sensores de medida en el que se ha programado una aplicación para enviar datos a dicho núcleo del sistema. Con el fin de hacer una demostración visual, se desarrollará una aplicación web que permitirá llevar un control de los dispositivos y los datos que se han recibido.

Como tarea adicional, se ha acudido a un programa de emprendimiento ofrecido por la Universidad Autónoma de Madrid. Se explicará cuáles han sido los pasos aprendidos para analizar la viabilidad del producto y cómo se ha desarrollado y organizado de cara a tener un futuro equipo trabajando en el proyecto. La tarea de emprendimiento se ha detallado en el Anexo A.

Este proyecto, al que se ha denominado con el nombre de VIKYNGO, se encuentra alojado en un repositorio público [1] y se ha desarrollado con licencia Apache Versión 2 [2].

Abstract (English)

Technology has changed the way we live. Nowadays, we can get the information of pollution that exists in the cities in real time, the refuse collector knows that a container is full before picking it up, we control the heating of our house with a mobile app, etc. These use-cases are usually included in the paradigm of "Internet of things" (IoT), which aims to connect all devices and/or machines to the Internet, so that they can exchange information and generate benefits to society. To store this information, we use databases. However, this is not enough when it is required to share data between different entities without the possibility of being illegally manipulated. With the arrival of Blockchain, this problem will be solved.

On this project, a use-case applied to the supply chain industry, will be developed. To make this possible, it is necessary to know the panorama of essential tools, protocols and good practices when developing a stable application that can reach a real production environment. A preliminary study of the state of the art will be made, analyzing and understanding the operation of these technologies and the possibilities that they offer us to be able to correctly develop the project. The architecture will be designed, taking the needs and services that occur throughout the chain into account. Subsequently, the project will be developed in order to make a real demonstration to a possible client. On the one hand, the core of the application will be developed, which is responsible for reading the data sent by devices, storing them securely. On the other hand, a minicomputer with measurement sensors, in which a small application has been programmed to send data to the abovementioned core of the system, which has been used. In order to make a visual demonstration, a web application, in which the devices and the data received will be controlled, will be developed.

As an additional task, we have attended an entrepreneurship program offered by the Autonomous University of Madrid. They will explain which steps have been learned to see the viability of the product and how it has been developed and organized in order to have a future team working on the project. The task of entrepreneurship has been detailed in Annex A.

This project, which has been denominated as VIKYNGO, has been published in a public repository [1] and has been developed with Apache License Version 2.0 [2].

Palabras clave (castellano)

Internet de las cosas, cadena de bloques, cadena de suministro, monitorización, trazabilidad, industria 4.0.

Keywords (inglés)

Internet of Things, Blockchain, supply chain, monitoring, traceability, industry 4.0.

Agradecimientos

Quiero agradecer a mis padres el haberme dado la libertad para afrontar estos 7 años en los que, además de la carrera, tenía otros sueños y objetivos por cumplir.

A mi hermano, por haberme descubierto el interesante mundo de las criptomonedas.

A mis amigos, porque han sido mi fórmula secreta para llegar feliz hasta el final de esta etapa.

A mis compañeros Jamil y Mariale por el sufrimiento compartido en la biblioteca durante toda la carrera.

A los profesores, por los conocimientos que me han transmitido y la persistencia que han tenido en que los entienda.

A Lluís, por confiar en la idea del proyecto y ofrecer su tiempo para el desarrollo de este TFG.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

En este capítulo se explicará la motivación por desarrollar el proyecto de trazabilidad de una cadena de suministro que se ha abordado en este trabajo de final de grado. Se expondrá la planificación de tareas y los objetivos para su desarrollo, así como la previsión de horas de trabajo y la organización que se ha seguido para escribir la memoria.

1.1 Motivación

Las telecomunicaciones es uno de los sectores de mercado más afectados por la creciente demanda del universo digital. En los últimos años ha habido un crecimiento exponencial en el número de dispositivos conectados a la red. Se espera que para el 2020 haya conectados alrededor de 50 billones de dispositivos [3]. Bajo este escenario aparece el paradigma de "internet de las cosas" (Internet of Things, IoT), el cual pretende conectar todos los dispositivos y/o máquinas a Internet para que intercambien información y generen beneficio para la sociedad. La aparición de la tecnología 5G en las redes, en combinación con los dispositivos, permitirán el análisis de datos en tiempo real y la ejecución remota de servicios críticos. Paralelamente han emergido tecnologías de almacenamiento de datos como Blockchain impulsada por el movimiento Cipherfunk [4] en el que se aboga por el uso generalizado de la criptografía para mejorar la privacidad de las personas. Según un estudio de Accenture [5], se espera que para el 2022 el mercado Blockchain esté valorado en los 12 billones de dólares.

Por otro lado, vivimos en un mundo en el que no sabemos a ciencia cierta si los productos que compramos y utilizamos en nuestro día a día son realmente como nos hacen creer que son. A pesar de los certificados de origen y las leyes impuestas para el control de los productos, aparecen noticias como la del atún de contrabando [6] en la que se puede ver que existen mafias que evitan estos controles con los productos que compramos.

Actualmente las empresas involucradas en una cadena de suministro (productores, distribuidores, comerciantes, etc.) invierten tiempo y recursos en comunicarse de forma que el proceso de creación y/o producción, distribución y venta del producto sea lo más eficiente posible. El sistema de funcionamiento que se sigue en estos casos consiste en que cada empresa implicada mantiene sus propios datos y se les obliga a pasar por numerosas auditorias para certificar que los procesos son los correctos. Compartir los datos en una cadena de suministro permite llevar un control del producto en toda la cadena, facilitando la comunicación entre las empresas y mejorando la eficiencia de los procesos. Combinando IoT y Blockchain podemos resolver el problema de la compartición de información en una cadena de suministro

Se desea desarrollar una aplicación la cual cada empresa instalará en su servidor privado de forma que se puedan conectar entre sí a modo de crear una red privada. Esta aplicación será capaz monitorizar los datos de un producto o más productos de la cadena de suministro y almacenarlos en una base de datos en la que no podrán ser manipulados ilícitamente. Este sistema permitirá mejorar la comunicación entre empresas, hacer más eficientes los procesos internos y dar mayor transparencia del producto al cliente final.

1.2 Objetivos

Para desarrollar el proyecto y poder realizar una tabla de tareas junto con una estimación temporal, se han propuesto los siguientes objetivos:

- **Investigación**: Analizar y extraer información sobre el panorama tecnológico de herramientas y protocolos dentro del ecosistema Blockchain e IoT. Saber diseñar la arquitectura de una plataforma como servicio que utilice tecnologías Blockchain e IoT. Hacer pequeñas pruebas de las posibles herramientas a utilizar con el fin de averiguar si son útiles para nuestro proyecto. Adquirir conocimientos y experiencia con las herramientas y lenguajes informáticos que se pretendan aplicar al proyecto.
- **Diseño:** Elegir las tecnologías más convenientes para el caso de uso que se quiere desarrollar. Diseñar la arquitectura teniendo en cuenta los requisitos de nuestra aplicación y los problemas a resolver.
- **Desarrollo**: Implementar y documentar los componentes mencionados en la arquitectura de forma que se comuniquen entre sí y proporcionen las funcionalidades especificadas deseadas.
- **Integración y pruebas**: Probar que los componentes implementados funcionan correctamente y proporcionan las funcionalidades requeridas.
- **Trabajo adicional:** Aprender y tratar de aplicar los conocimientos básicos de emprendimiento que permitan llevar el producto desarrollado en este trabajo a un entorno empresarial.

1.3 Fases del proyecto

Los distintos objetivos mencionados anteriormente se han subdividido en tareas tal y como se detalla en la Figura 1-1, y se representa en el diagrama de Gantt de la Figura 1-2. Para evitar una tabla compleja, algunas tareas reúnen un conjunto de subtareas que van implícitas en estas. Se ha invertido una media de 2h al día desde el comienzo del proyecto y la duración total del desarrollo ha sido de 300h más 50h horas del trabajo adicional en el apartado de emprendimiento.

	Nombre	Duracion
1	**Investigación**	**48 days**
2	Arquitectura de una PaaS	2 days
3	Tecnologías Blockchain	4 days
4	Tecnologías IoT	2 days
5	Arquitectura de un sistem...	2 days
6	Adquirir conocimientos de...	20 days
7	Pruebas con tecnologías ...	10 days
8	Pruebas con tecnologías IoT	5 days
9	**Diseño**	**6 days**
10	Arquitectura global	2 days
11	API	1 day
12	Bases de datos	1 day
13	Interfaz de Usuario	2 days
14	**Desarrollo**	**54 days**
15	**Core**	**20,5 days**
16	API	7 days
17	IBM Cloud	4 days
18	Scripts	10 days
19	Dcocumentación	2 days
20	**IoT**	**9 days**
21	Broker MQTT	2 days
22	Programar Raspberry Pi	5 days
23	**Aplicación web**	**21 days**
24	Lógica de la aplicación	12 days
25	Estética de la aplicación	8 days
26	**Integración y pruebas**	**22 days**
27	Desarrollo de tests unitarios	5 days
28	Corección de errores	10 days
29	Preparación de la demo	5 days
30	**Trabajo final**	**43 days**
31	Implementaciones y mejo...	4 days
32	Contactar con desarrollad...	2 days
33	Buscar medios de difusión...	5 days
34	Redactar memoria	25 days
35	Preparar presentación	7 days
36	**Trabajo adicional**	**25 days**
37	Presentación del proyeco	2 days
38	Asistir a clases de formación	6 days
39	Realizar ejercicios de emp...	6 days
40	Buscar colaboradoes	4 days
41	Organizar al equipo	2 days
42	Búsqueda de clientes	2 days
43	Web corporativa	4 days
	TFG	

Figura 1-1 – Tabla de tareas

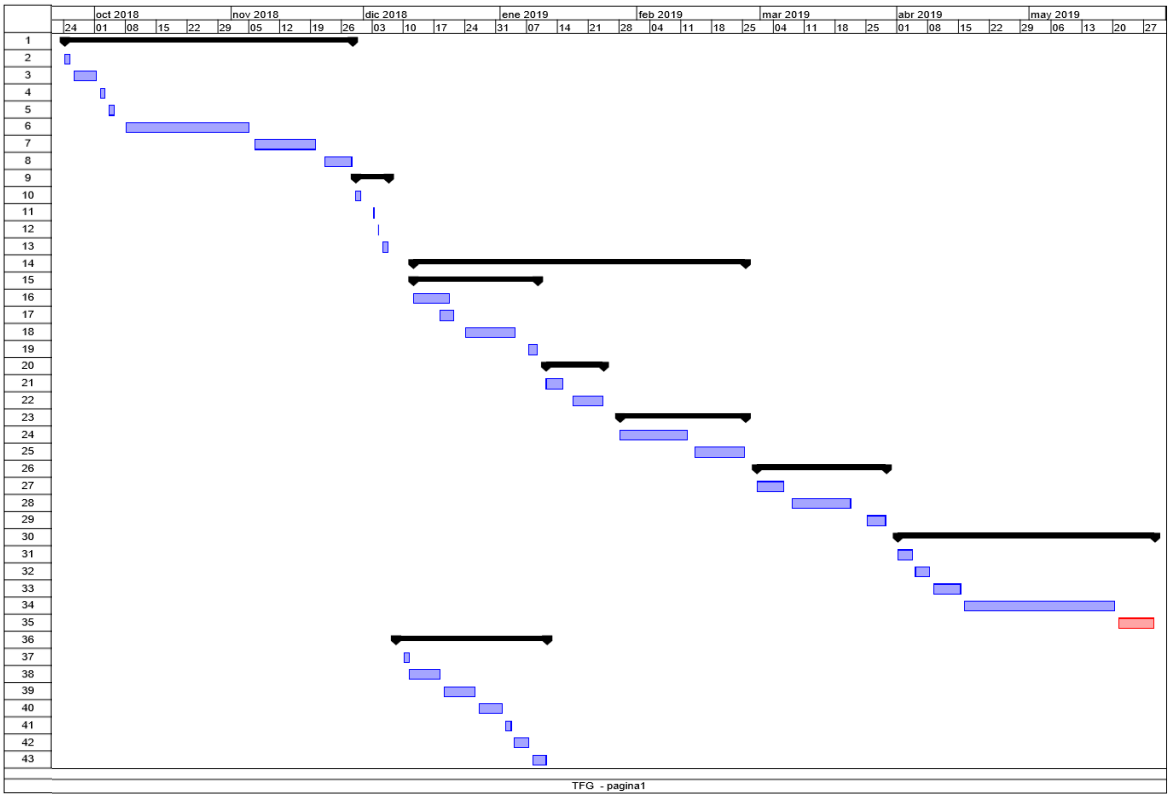

Figura 1-2 – Diagrama de Gantt

1.4 Organización de la memoria

La memoria consta de los siguientes capítulos:

- En el capítulo 2, titulado *Estado del arte*, se describirán los conceptos, tecnologías y protocolos que se han utilizado para implementar la aplicación.
- En el capítulo 3, titulado *Diseño,* se analizará el caso de uso de las cadenas de suministro y se diseñará la arquitectura del proyecto que se quiere desarrollar haciendo un estudio de las tecnologías disponibles.
- En el capítulo 4, titulado *Desarrollo, integración, pruebas y resultados*, se detalla los pasos que se han seguido para desarrollar sistema que se ha desarrollado y se plantean casos de prueba que nos permitirán validar el correcto funcionamiento.
- Finalmente, en el capítulo 5 titulado *conclusiones y trabajo futuro*, se resumen las conclusiones que se derivan de la realización de este proyecto, así como las posibilidades de desarrollo y mejora que puede tener este proyecto de cara al futuro.

2 Estado del arte

Antes de embarcarse en el desarrollo del proyecto, es importante conocer el estado actual de las tecnologías y protocolos que puedan estar relacionadas con el proyecto. Se va a desarrollar una aplicación de extremo a extremo (end-to-end) que dará servicio a toda la cadena de suministro, por lo que es conveniente analizar todas las tecnologías necesarias en el proceso. Esta sección se enfocará en explicar qué es Blockchain e IoT, junto a los protocolos y tecnologías más comunes en estos ámbitos. También se explicará algunos patrones y buenas prácticas para el desarrollo de aplicaciones.

2.1 Industria 4.0

La industria actual es el resultado de un proceso evolutivo en los métodos, herramientas y tecnologías que el ser humano ha ido desarrollando. La primera revolución industrial (industria 1.0) destacó por la invención de la máquina de vapor. Pocos años después, con el descubrimiento de la electricidad y el desarrollo de las cadenas de producción se dio la segunda revolución industrial (industria 2.0). En la tercera revolución (industria 3.0) se introdujeron los dispositivos eléctricos y las máquinas autómatas que permitieron realizar los procesos de forma automática. Y finalmente, la cuarta revolución industrial (industria 4.0) que se produce actualmente, se acuña por incorporar sistemas de inteligencia artificial con el objetivo de mejorar la eficiencia en sistemas de fabricación existentes. Tecnologías como Blockchain, IoT, Big Data, etc. Son las responsables de esta cuarta revolución. Existen grandes empresas que invierten en el desarrollo de proyectos utilizando estas tecnologías en las cadenas de suministro. IBM entre otras, ha desarrollado *Food Trust* [7] en la que hace uso de la tecnología Blockchain para certificar el proceso de producción de los alimentos. Uno de los grandes aspectos negativos que tiene este proyecto de IBM, es que obliga a sus clientes a pasar por sus servidores actuando como una caja negra en la que tienen que depositar la confianza para la administración de los datos, perdiendo la principal característica que ofrece esta tecnología, la descentralización. En España existen otros proyectos como *Trazable* [8] o *Mirror* [9] pero se desconoce el estado de los proyectos.

2.2 Blockchain

La palabra Blockchain deriva del término en inglés “cadena de bloques”, donde cada bloque representa a un conjunto de datos. Estos bloques se encadenan entre sí utilizando algoritmos criptográficos con el objetivo de que la información que hay en su interior no pueda ser manipulada. Al conjunto de bloques se le denomina “libro de cuentas” (ledger) heredado del concepto de libro de cuentas bancario. Esto se debe a que esta tecnología comenzó con la criptomoneda *Bitcoin* [10] y en estos bloques se registran el intercambio de la criptomoneda. Sin embargo, esta tecnología ha ido evolucionando y la información que se almacena en los bloques ya no tiene por qué referirse sólo a transacciones monetarias. En términos generales, Blockchain es una base de datos distribuida con tres características principales: inmutable, descentralizada y transparente.

Cuando se dice que Blockchain es **inmutable** significa que los datos que permanecen en el libro de cuentas no se pueden modificar, es decir, sólo se pueden introducir datos, pero no modificarlos ni eliminarlos. Esta característica se debe a que cada bloque de datos lleva añadido un identificador (resumen o hash) que tiene información del bloque anterior. Un resumen o **hash criptográfico** es el resultado de aplicar un algoritmo matemático a una cadena de datos y que este a su vez devuelve una cadena de caracteres de tamaño fijo. Este algoritmo es determinista, es decir, siempre que se introduzca la misma cadena de

caracteres se obtendrá el mismo resultado. Además, es extremadamente complicado encontrar dos mensajes de entrada que resulten el mismo resumen, y no permite realizar la operación inversa (pasar del resumen a la cadena de entrada). Utilizando las características de este algoritmo se puede demostrar que, si de partida tenemos un hash y queremos demostrar que unos datos han producido dicho hash, basta con aplicarles el algoritmo matemático y comprobar si el resultado coincide con el hash que teníamos. Si dicho resultado coincide, entonces se puede considerar inequívoco que esos eran los datos que produjeron dicho hash. Si en un bloque, añadimos el hash del bloque anterior, como se ilustra en la Figura 2-1, estamos forzando a que la información del bloque anterior no se pueda modificar, ya que si se modifica, el hash cambiará y por tanto no coincidirá con el hash que se había añadido en el bloque posterior, rompiendo así la cadena. Por otro lado, si tratamos de modificar el hash para que coincida con la información manipulada, así mismo estaremos modificando el hash de dicho bloque que lo contiene y este no coincidirá con el hash guardado en el siguiente bloque. El principal algoritmo de obtención del hash es el *SHA256* [11].

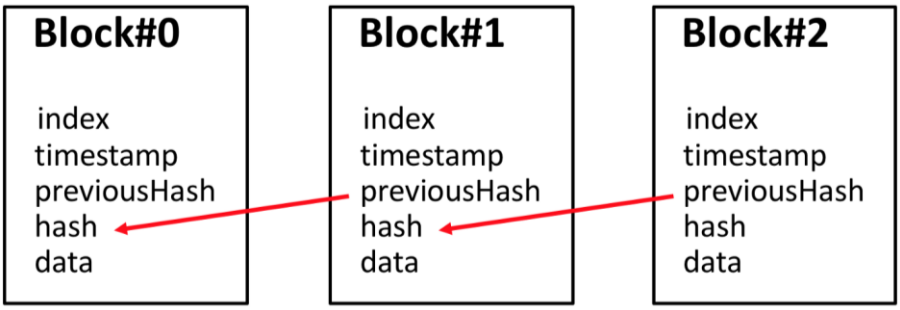

Figura 2-1 – Ejemplo de diagrama de bloques en una Blockchain

Como añadido de seguridad, los sistemas Blockchain detectan si un libro de cuentas no coincide con la mayoría de los nodos y actualizan los bloques faltantes o corruptos. Como se puede ver, manipular un dato se convierte en una tarea prácticamente imposible en redes con suficientes nodos y bloques de datos.

El atributo de **descentralizado** se refiere a que el control de la red no lo gobierna una entidad en sí, sino que es administrada por todas las entidades participantes. Una de las ventajas de poseer esta característica es que si un nodo deja de funcionar, el sistema puede seguir funcionando correctamente. Dependiendo de la red los nodos pueden tener más o menos privilegios. Un ejemplo de red centralizada podría ser el caso de un banco donde nuestras cuentas las administra una entidad central (el banco) y por tanto dependen totalmente de esta. En cambio, una red descentralizada podría ser una cadena de suministro en la que hay implicadas varias empresas y donde nadie tiene el control total de dicha cadena. Por último, la **transparencia** se debe al simple hecho de que todos los participantes de la red tienen acceso a los datos. Existen redes en las que la información se encripta con el objetivo de que otros participantes no puedan acceder a los datos, como es el caso de la red española *Alastria* [12]. Un método muy común en estos casos es encriptar la información utilizando criptografía asimétrica.

La **criptografía asimétrica** es un sistema muy utilizado para enviar información a través de la red con la seguridad de que esta información sólo pueda ser leída por el receptor o receptores que tengan la clave de desencriptación que es diferente a la de encriptación (de ahí el nombre de asimétrica). Este sistema consiste en crear un par de

claves (una clave pública y otra privada) las cuales se utilizan para encriptar y desencriptar el mensaje. En el caso de la criptografía asimétrica, para enviar un mensaje, se encripta con la clave pública del receptor, por lo que sólo este podrá descifrar el mensaje. En la Figura 2-2 se puede ver como BOB le envía un documento a ALICE cifrando el documento con la clave pública de ALICE y esta lo descifra con su clave privada.

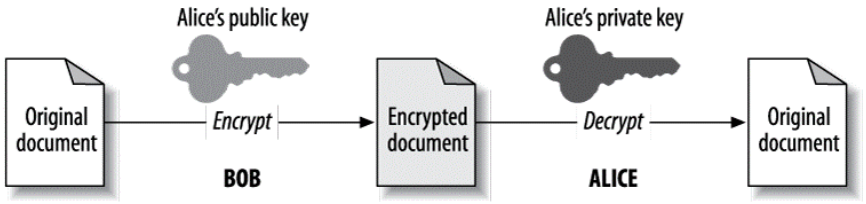

Figura 2-2 – Ejemplo de cifrado de mensaje con criptografía asimétrica

Este sistema también se puede utilizar a la inversa para verificar que alguien ha aceptado una transacción. Si yo encripto una información con mi clave privada y el resultado de esta encriptación se puede desencriptar con mi clave pública, entonces es verídico que yo he sido el firmante de dicha información. A este proceso de firma se le entiende como **firma electrónica.** En Blockchain, este sistema se utiliza para firmar las transacciones o contratos inteligentes.

Los **contratos inteligentes** son piezas de código que se insertan en la cadena de bloques. El hecho de permitir introducir código lógico en los bloques aparece con la criptomoneda *Ethereum* [13]. Utilizar piezas de código abre un mundo de posibilidades dentro de la tecnología Blockchain que con *Bitcoin* no se había alcanzado. Por ejemplo, se podría utilizar para registrar datos públicos como los derechos de una vivienda e incluso venderla utilizando como moneda de cambio el *Ether* (criptomoneda de la red Ethereum) y mediante el uso de la lógica, programar condiciones o cláusulas en dicho contrato.

2.2.1 Tipos de redes Blockchain

Si hay que tener un aspecto en cuenta a la hora de elegir una tecnología Blockchain, es determinar cómo se va a administrar el acceso a los datos. En función a este parámetro las redes se clasifican en:

- **Públicas**: Son las redes en las que cada persona, entidad o participante en la red tiene acceso a través de un programa público al que se accede identificándose con una clave pública. La identidad en este caso es anónima y hay un nivel de desconfianza alto con el resto de los participantes. Estas redes están altamente expuestas al ataque de los hackers. Por ello se requiere un sistema más difícil de franquear que repercute en un mayor tiempo de procesamiento de las transacciones que en las redes privadas.
- **Privadas**: Son las redes en las que los participantes tienen acceso con previo acuerdo de los que forman dicha red. En este caso, los participantes están identificados y son conocidos por el resto de los participantes. Un ejemplo muy obvio podría ser una red formada por bancos. .
- **Híbrida**: Si a una red privada se le da permisos de acceso público a los datos, esta se convierte en una red **híbrida**. Cuando se aplican permisos para ejecutar transacciones o leer datos, se dice que la red es **permisionada**. En el caso más típico, los miembros o partes interesadas de un consorcio operan en una red de Blockchain permisionada, por ejemplo, el caso de la red *Alastria.*

2.2.2 Algoritmos de consenso

Un algoritmo de consenso es un proceso informático utilizado para llegar a un acuerdo entre sistemas distribuidos. Están diseñados para lograr una confiabilidad en una red que involucra nodos con distintos administradores. En el caso de Blockchain, los algoritmos de consenso se utilizan para decidir cómo se inserta el siguiente bloque de la cadena y evitar posibles modificaciones de los datos. Los algoritmos de consenso hacen que una red Blockchain pueda ser descentralizada y fiable ya que la decisión no la toma un agente central si no que se toma por el consenso de una cantidad significativa de nodos.

Los algoritmos de consenso, según de qué tipo sean, pueden tener desventajas que normalmente afectan a la velocidad de procesamiento de las transacciones y la escalabilidad de la red. Esto hace que en sistemas donde se requieren muchas transacciones por segundo o haya una alta cantidad de nodos conectados sea un aspecto crítico. Por ello se han diseñado algoritmos para funcionar mejor en unos sistemas u otros [14]. A continuación, se describen algunos algoritmos:

- **Prueba de trabajo (Proof of Work):** Es el algoritmo utilizado en la famosa criptomoneda *Bitcoin* entre otras. Este algoritmo consiste en que los nodos tienen que resolver un problema matemático y el único método para ello es a base de fuerza bruta aplicando ciclos de prueba y error. El primer nodo que consiga resolver este problema será el encargado de introducir el siguiente bloque en la cadena. A este proceso se le denomina comúnmente como "minar" y a los nodos que realizan este proceso se les denomina "mineros". Las desventajas principales de este sistema son que resolver el problema matemático puede llevar bastante tiempo y que los nodos con mayor procesamiento tienen más probabilidades de ser los encargados de minar. La ventaja que tiene este algoritmo es que es **tolerante a la falla bizantina (Byzantine Fault Tolerant, BFT)** [15], que es la característica que tiene un sistema, generalmente distribuido, a llegar a un consenso para realizar una acción con la posibilidad de que algunos nodos fallen. Este término deriva del *problema de los generales bizantin*os [16] en el que unos generales van a hacer un ataque desde distintos lugares y se tienen que poner de acuerdo para atacar sin que alguno de ellos pueda actuar de traidor e interfiera en el consenso.
- **Prueba de participación (Proof of Stake):** En este caso la probabilidad de minar se establece en función de los tokens o criptomonedas que tiene el nodo. La primera criptomoneda en adoptar ese algoritmo fue *Peercoin* [17]. En este caso se consiguen resolver los aspectos negativos propuestos con el algoritmo anterior. En este algoritmo se incentiva a poseer un mayor capital en la red y eso crea la desventaja de que puede provocar monopolios de control, ya que el nodo con más capital tendrá más probabilidades de gobernar el proceso de inserción del bloque. En este caso no hay proceso de minado, por lo que al nodo se le denomina "validador" (valida que los bloques son correctos). De este algoritmo se pueden desarrollar otros tantos que pueden variar en función de la motivación que se le quiera dar a los nodos (Proof of Activity, Proof of Importance, Proof of Capacity, Proof of Burn, etc).
- **Hashgraph:** Es un algoritmo de consenso creado por Leemon Baird, cuya propiedad intelectual pertenece a Swirlds Corporation. El sistema de funcionamiento es distinto al que tienen las tecnologías Blockchain tradicionales y ofrece una ventaja considerable en el número de transacciones procesadas por segundo. Para hacernos una idea, en una red en la que se hace uso del algoritmo de

consenso PoW, se puede procesar alrededor de 5 transacciones por segundo, sin embargo, haciendo uso del algoritmo Hashgraph se pueden procesar miles de transacciones. El funcionamiento de este algoritmo consiste en que los nodos de la red van guardando un histórico de transacciones propio, cada cierto tiempo estos nodos se comunican entre sí de forma aleatoria y van comprobando si el histórico de transacciones del resto de los nodos es correcto. En la figura [18] se puede ver la diferencia entre una red Blockchain tradicional y el algoritmo Hashgraph.

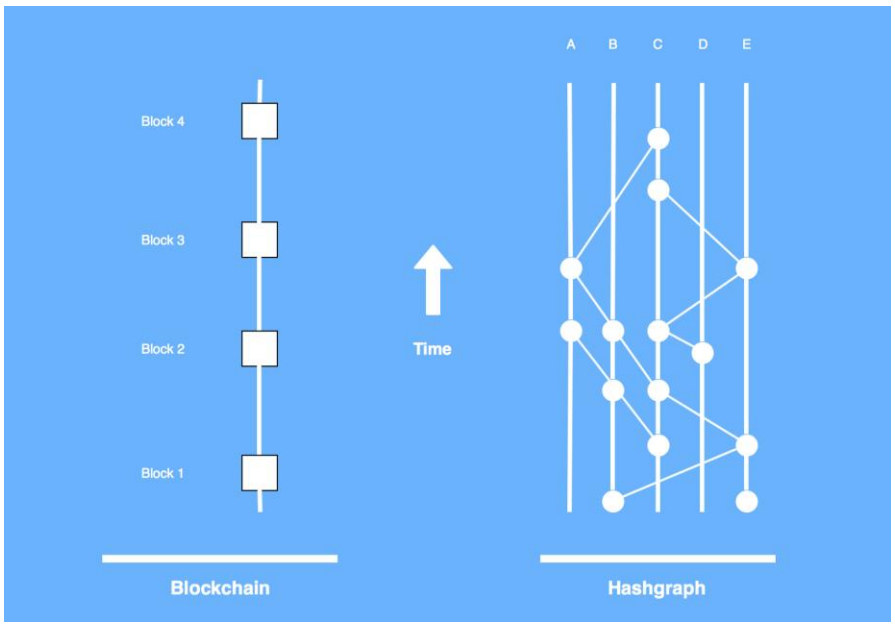

Figura 2-3 Diferencia entre Blockchain tradicional y algoritmo Hashgraph [18]

2.2.3 BigchainDB

BigchainDB es un programa administrado por *IPDB Foundation* [19] y escrito en el lenguaje de programación *Python* que hace uso de las técnicas de Blockchain para implementar una base de datos distribuida enfocada en la administración de activos digitales [20]. Actualmente se encuentra en una versión estable (2.0.0b9) y posee la licencia de *Apache Versión 2.0*. El modo de funcionamiento de *BigchainDB* consiste en tratar las transacciones con unas claves privadas de entrada y unas claves públicas de salida. Cuando se crea una transacción para definir una entidad (elemento al que se le pueden atribuir propiedades), la entrada está vacía y en la salida se escriben las claves públicas de los destinatarios o propietarios de dicha entidad. Si se quiere transferir esta entidad a otros destinatarios, en la siguiente transacción será necesario introducir en la entrada las claves privadas referentes a las claves públicas que se habían introducido en la salida de la transacción anterior, es decir, las claves privadas de los propietarios en el momento que se produce la transacción. *BigchainDB* además posee una implementación a la que le denominan "Cryptoconditions", que permite establecer condiciones para que la transacción se pueda realizar sin que sea necesario utilizar todas las claves privadas de los propietarios. *BigchainDB* no soporta contratos inteligentes, pero existe la red *Cosmos* que hace uso del protocolo *Inter Chain Comunication*, el cual es muy útil para conectar dos tecnologías Blockchain o escalar una red Blockchain [21]. En la sección de *Conclusiones y trabajo futuro* se tratará este tema.

La arquitectura de *BigchainDB* (Figura 2-4) no es muy compleja. Hace uso de software de terceros como *Tendermint* [22] o *MongoDB* [23]. **Tendermint** es un programa que incorpora un protocolo de consenso que soporta tolerancia a la falla bizantina. *BigchainDB* utiliza este protocolo para tener los nodos en sincronización, validar las transacciones y asegurarse que son almacenadas en todos los nodos. **MongoDB** es una base de datos no relacional en la que se almacena la información de los bloques para que el usuario pueda acceder a esta de forma eficazFigura 2-4 – Arquitectura de BigchainDB 2.0 [24].

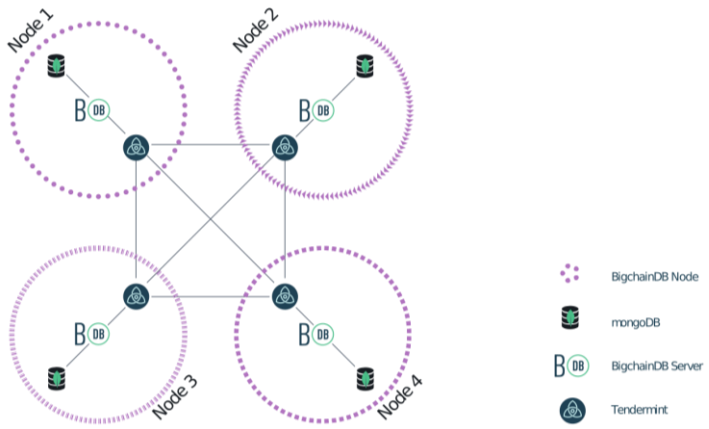

Figura 2-4 – Arquitectura de BigchainDB 2.0 [24]

2.3 Internet de las cosas

Internet de las cosas (Internet of Things, IoT), es un término que ha ido ganando fuerza a medida que han ido evolucionando las redes de comunicaciones. Consiste en la conexión de los dispositivos electrónicos de nuestro día a día a la red de Internet para su monitorización y control. Para explicar qué es IoT, lo mejor es descomponer esta tecnología en capas, como se ilustra en la Figura 2-5: *i*) capa de sensorización, *ii*) capa de red, *iii*) capa de procesamiento de datos, y *iv*) capa de aplicación.

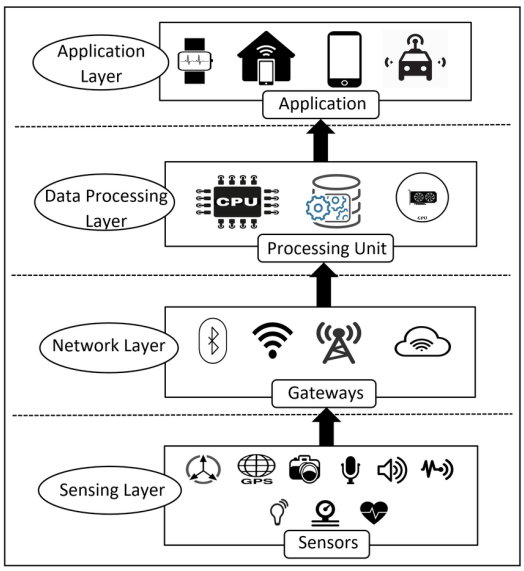

Figura 2-5 – Capas de una aplicación IoT

El objetivo de la **capa de sensorización** es extraer datos del entorno, haciendo uso de sensores, para ser procesados en la capa de aplicación. Dentro de los tipos de sensores existen varias categorías: sensores de movimiento (acelerómetro, giroscopio…), sensores ambientales (luz, temperatura, audio…) y de posición (GPS, magnetismo…).

El objetivo de la **capa de red** es transmitir los datos extraídos en la capa de sensorización a la capa de procesamiento de datos. En esta capa encontramos distintos tipos de estándares de conectividad: por proximidad (Como RFID), Wireless Personal Area Networks (WPAN) (Como Bluetooth), Wireless Local Area Networks (WLAN) (Como Wi-Fi), Lowe-Power Wide Area Networs (LPWAN) (Como LoRa y SIGFOX) y celular (Como 5G). A continuación se detallan los más comunes:

- **RFID (Radio Frequency Identification):** Es un chip transpondedor conectado a una antena que responde cuando recibe un campo electromagnético. A veces son acompañados de una pequeña batería o condensador para asegurar una potencia interrumpida al chip, ya que es necesario que esté recibiendo potencia durante un cierto tiempo, para poder generar la señal de respuesta. Esta tecnología es muy utilizada para etiquetar artículos y rastrearlos. Puede operar, en función del modelo, en distintas frecuencias: baja frecuencia (30 – 500kHz), alta frecuencia (10-15MHz) y ultra alta frecuencia (850 – 950 MHz, 2.4 – 2.5 GHz, 5.8 GHz)

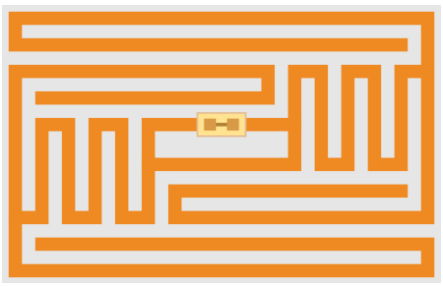

Figura 2-6 – Etiqueta RFID [25]

- **Wi-Fi (Wireless Fidelity):** Es una colección de estándares WLAN (Wireless Local Area Networks) basados en las especificaciones IEEE 802.11 [26], que permite la creación de redes sin la necesidad de utilizar cableado. Las bandas de frecuencia van de las 2.4 GHz a los 5 GHz. En este trabajo se ha utilizado para la conexión del dispositivo IoT a nuestro servicio Blockchain.
- **Bluetooth:** Esta basado en el estándar IEEE 802.15.1 [27]. Es una tecnología sin cables de bajo consumo diseñada para la conexión de dispositivos móviles con un rango de unos 8-10 metros. Esta tecnología opera en la frecuencia de 2.4 GHz.

A un nivel más alto podemos encontrar distintos protocolos de comunicación diseñados para IoT por su bajo consumo de energía y ancho de banda. Los principales son:

- **MQTT (Message Query Telemetry Transport) [28]:** Este protocolo ha sido diseñado por IBM. En la Figura 2-7 se puede ver el funcionamiento de este protocolo que consiste en que un cliente (Client1) se conecta a un servidor MQTT (Broker) y hace una publicación de un mensaje con el tópico “events”. Otro cliente (Client2) también está conectado al servidor escuchando en el tópico “events” de forma que cuando el mensaje que ha publicado “Client1” llega al servidor, este lo reenvía a los clientes subscritos, en este caso a “Client2”. Este protocolo trabaja sobre TCP por lo que se garantiza que los mensajes son entregados sin errores en el

mismo orden en el que se transmitieron. Una de las herramientas más conocidas para utilizar este protocolo es *Mosquitto* [29].

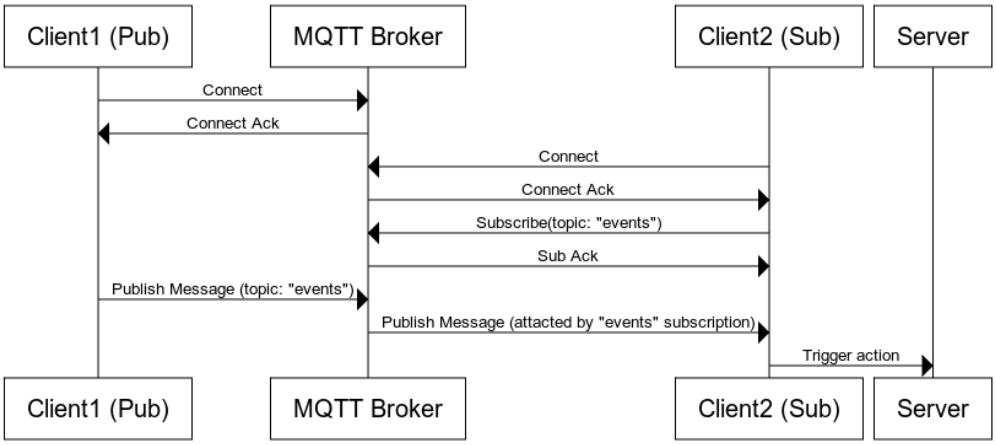

Figura 2-7 – Diagrama del protocolo MQTT

- **XMPP:** Es un protocolo basado en XML (Extensible Markup Language). Las ventajas de este protocolo es su direccionamiento, seguridad y escalabilidad. La desventaja es el sobrecoste de enviar los mensajes en el formato XML.
- **AMQP:** Funciona igual que el protocolo MQTT pero ofrece más funciones complejas y avanzadas, además es algo más complejo y poco ligero.
- **CoAP:** Es un protocolo REST (se explica en la siguiente sección) y trabaja sobre UDP.

El objetivo de la **capa de procesamiento** es almacenar los datos que han llegado de los dispositivos IoT y procesarlos. Para almacenar la información existen variedades de bases de datos. Por un lado, están las **bases de datos relacionales** (como *PostgreSQL* [30]) en las que se utiliza el lenguaje SQL para administrarlas y hacen uso de tablas estructuradas y relacionadas en las que se insertan los datos. Este sistema permite hacer búsquedas de datos rápidamente. Por otro lado, las **bases de datos no relacionales** (como *MongoDB*) son más útiles en estos casos debido a que se pueden almacenar datos no estructurados. Sin embargo, estas bases de datos son mucho más lentas para hacer búsquedas. Para el procesamiento de datos se pueden encontrar programas como *Hadoop* [31].

Finalmente, en la **capa de aplicación** se muestran los datos al usuario y se controlan los dispositivos. En esta capa hay librerías de visualización de datos muy útiles como *D3js* [32] o *Chartjs* [33]. También son muy comunes los cuadros de mando (dashboards) que son programas que permiten inyectar datos y representarlos en gráficas. Ejemplos de dashboards son *Grafana* [34] o *Graphite* [35].

2.4 Patrones y buenas prácticas para el desarrollo de programas

A la hora de diseñar una aplicación hay que tener en cuenta una serie de aspectos como la **accesibilidad** para que personas con algún tipo de discapacidad puedan hacer uso de la aplicación. El consorcio *W3C* ha desarrollado una guía (WCAG 2.1) [36] en la que se detalla los puntos a seguir para tener una web accesible.

Otro punto importante es **documentar** de la forma más detallada y clara posible el funcionamiento de nuestro programa para facilitarles a los usuarios su uso. Para el caso

que se trata en este proyecto, en el que se desarrolla una API REST (se explica más adelante), existen librerías como *Swagger* [37] *apiDoc* [38].

Cuando se trabaja en un equipo, es importante **comentar** el código del programa para facilitar la fluidez y el entendimiento entre los desarrolladores. El **estilo de programación** que estos deben seguir también debe ser lo más parecido y limpio posible. Para ello existen programas denominados *Linters* que se encargan de revisar el código y a partir de una serie de reglas definidas avisar al usuario de si se ha incumplido alguna de estas. Un ejemplo de esta herramienta es ESLint [39].

Dentro de lo que se conoce como **desarrollo ágil** entran aspectos como el uso de repositorios, la integración continua, las pruebas unitarias o el despliegue continuo. El objetivo un desarrollo ágil es automatizar el proceso desde que se produce una modificación en el código de nuestro programa hasta que este está disponible para su ejecuciñon. En este proceso entran herramientas como *Jenkins* [40] que son capaces de ejecutar una serie de acciones cada vez que el código del repositorio se ha modificado y ejecutar pruebas para comprobar que nuestro programa funciona correctamente. Para el caso de este proyecto una herramienta para las pruebas es *Mocha.js* [41].

La **seguridad** es otro aspecto muy importante, por ello existen organizaciones como *OWASP* [42] en la que se detallan aspectos de seguridad a tener en cuenta a la hora de desarrollar una aplicación segura.

La **arquitectura** en una aplicación hace que pueda ser escalable y facilite las cosas a la hora de encontrar errores. Podremos aplicar distintos tipos de arquitecturas dependiendo del caso de nuestra aplicación y el nivel en el que nos encontremos. De forma simple podemos definir 3 aspectos o niveles en los que hay que tener en cuenta la arquitectura: según la distribución de los documentos que tienen el código de nuestro programa (en la sección 3.1 se mostrará más detalladamente), según la comunicación que vaya a tener con otros componentes y según la segmentación de la aplicación en uno o en varios componentes. A continuación, se muestran dos diseños de arquitectura, por un lado, la arquitectura de microservicios que se utiliza de forma global en todo el aplicativo y por otro lado la arquitectura REST que se utiliza para cada componente.

2.5 Arquitectura de Microservicios

La arquitectura de microservicios es un estilo de diseño de aplicaciones que, a diferencia de la arquitectura monolítica que consiste en diseñar las aplicaciones como una única entidad encargada de gestionar todos los servicios, funciona como un conjunto de pequeños servicios que se ejecutan de manera independiente y autónoma, incluso haciendo uso de lenguajes de programación diferentes. Las ventajas de desarrollar una arquitectura de microservicios es que el hecho de dividir una aplicación en distintas partes hace que sea mucho más fácil de desarrollar. Si un componente falla, no tiene por qué afectar a los demás. Por otro lado, se evita la sobrecarga y es más fácil escalar los componentes sobrecargados simplemente arrancando copias del servicio y utilizando un balanceador de carga. Tiene dos desventajas importantes con respecto a la arquitectura monolítica y son que el consumo de memoria es mucho mayor, y puede verse afectado el rendimiento por las latencias introducidas por la red cuando los servicios corren en un entorno distribuido. También hace que sean aplicaciones más difíciles de testear y en arquitecturas con muchos servicios puede ser algo difícil de gestionar.

En la Figura 2-8 se puede ver en el lado izquierdo, la arquitectura monolítica en la que se gestionan tres servicios en la misma entidad y en la derecha la arquitectura de

microservicios en la que una entidad es encargada de un servicio y otra de los otros dos servicios.

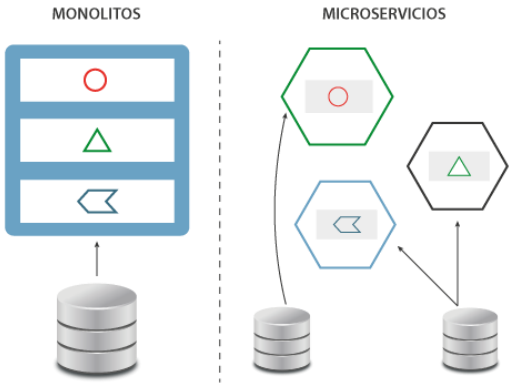

Figura 2-8 – Monolitos vs Microservicios

Una de las herramientas más útiles para combinar con este paradigma de desarrollo de microservicios es *Docker* [43] .**Docker** es una herramienta de código libre que se basa en la empaquetación de aplicaciones en contenedores. Los contenedores se encuentran aislados entre sí y se comportan como máquinas independientes. El servicio que ofrecen los contenedores *Docker* es parecido al de las máquinas virtuales, sin embargo, la diferencia es que *Docker* utiliza su motor, llamado *Docker Engine* (véase Figura 2-9), que se encarga de lanzar y gestionar los contenedores con nuestras aplicaciones gestionando los recursos entre los contenedores, optimizando su uso y eliminando la necesidad de tener sistemas operativos separados para conseguir el aislamiento. Las ventajas de los contenedores son que son más ligeros que una máquina virtual, no es necesario instalar un sistema operativo por contenedor y hacen menor uso de los recursos de la máquina.

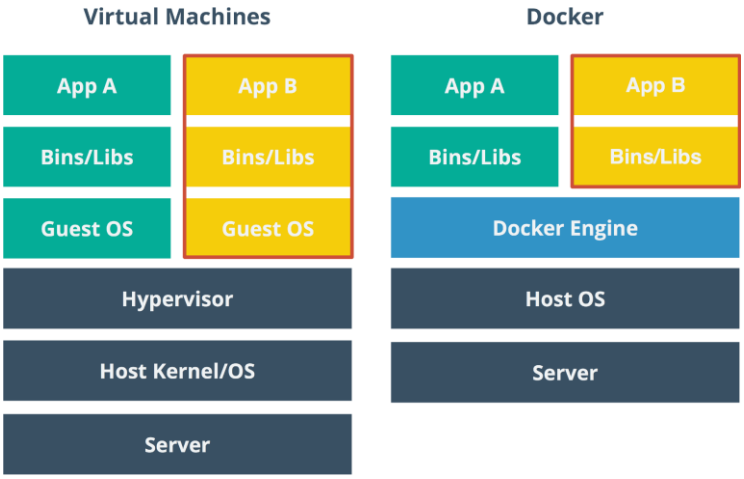

Figura 2-9 – Máquinas Virtuales vs Docker

2.5 Comunicación REST (Representational State Transfer)

REST es un estilo de arquitectura de aplicaciones que se utiliza en sistemas que se comunican a través de HTTP (Hyper Text Transfer Protocol) [44]. El protocolo **HTTP** permite una comunicación entre una aplicación web y el servidor y sirve para realizar una petición de datos y/o recursos, por ejemplo, el código de una página web, una imagen, o un fichero de datos genéricos. Los mensajes HTTP están compuestos de una cabecera (header) en la que se especifica las características del mensaje, y un cuerpo (body) en el que va el mensaje. REST hace uso de este protocolo para la comunicación. Envía peticiones HTTP en las que contiene toda la información necesaria para ejecutarla, lo que permite que ni cliente ni servidor necesiten recordar ningún estado previo a dicha petición. Las operaciones más importantes que se pueden realizar sobre un recurso usando REST son: POST (crear), GET (leer), PUT (editar) y DELETE (borrar). REST soporta el formato JSON (JavaScript Object Notation) y XML (Extensible Markup Language) para transportar los datos. El formato JSON es menos verboso y ocupa menos que el formato XML. Esto hace que sea muy común en las aplicaciones actuales.

2.6 Conclusiones

En este capítulo hemos podido abordar el funcionamiento y los aspectos esenciales para entender la tecnología Blockchain. También se ha explicado el paradigma de IoT, así como sus protocolos de comunicación. Por último, se han citado algunos aspectos importantes a la hora de desarrollar aplicaciones y se han mencionado las arquitecturas que se han seguido a la hora de desarrollar el aplicativo.

3 Diseño

El objetivo de este trabajo es crear una plataforma capaz de dar servicio a toda la cadena de suministro. Diseñar un aplicativo eficaz para este cometido, requiere analizar las necesidades a tener en cuenta en este tipo de entornos. En esta sección se va a analizar el proceso de negocio de la cadena de suministro y se va a confeccionar la lista de requisitos de nuestro aplicativo al que se le ha denominado *VIKYNGO*. También se especificará cómo debe estar organizado el aplicativo y se evaluarán diferentes tecnologías y plataformas existentes que puedan asemejarse a *VIKYNGO*.

3.1 Proceso de suministro

En la producción de manzanas hay numerosas empresas implicadas: recolectores, transportistas, supermercados, etc. Para ofrecer un servicio de transparencia, es importante llevar un control del estado de las manzanas en todo el recorrido. Además, automatizar los pagos entre las distintas empresas implicadas podría reducir una gran cantidad de costes de gestión. A partir de ahora, se tomará el caso de cadena de producción de manzanas como referencia para diseñar el aplicativo. Para ello, dividiremos el proceso en 3 partes: Recolección, transporte y venta.

Al principio de la cadena, se recolectarán las manzanas y se almacenarán en cajas para ser transportadas. El productor tendrá entonces un contrato con las empresas de esta cadena. Un aspecto importante a tener en cuenta es que en muchos casos el valor del producto puede variar en función del tratamiento que se le aplique para producirlo, por ejemplo, en el caso del jamón serrano, el precio puede variar en función de la alimentación que se le dé al animal.

Haciendo uso de dispositivos IoT, podemos obtener la información necesaria para cuantificar o categorizar el trato que ha recibido el producto y en función de este, ajustar su valor económico. Para conseguir esto, necesitaremos poder identificar el producto y tener dispositivos enviando información constantemente sobre el trato aplicado al producto. En este caso, podremos identificar las cajas y a ser posible deberán estar diseñadas para que no puedan ser manipuladas a lo largo del recorrido de la cadena. Con esto aseguramos que los datos se corresponden con los del producto que se almacena en el interior de las cajas al inicio de la cadena. Las cajas se identificarán haciendo uso de la tecnología RFID que se ha tratado en el estado del arte. Dispondremos de dispositivos que obtengan el identificador de las cajas y asocien la información de los sensores con el de la caja identificada.

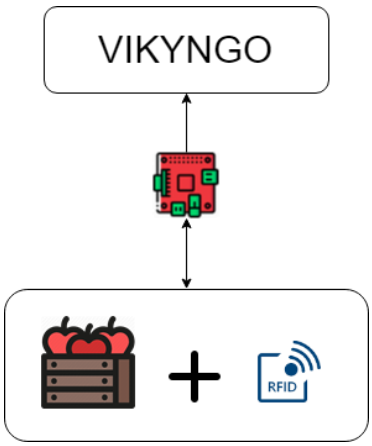

Figura 3-1 – Ejemplo de transferencia de datos de un activo con RFID

Haciendo uso de los contratos inteligentes, se puede diseñar una lógica para que cumplan los contratos legales que se han firmado con el recolector. En ellos se puede establecer el precio de la manzana y las condiciones que puedan afectar a dicho precio, como la época del año. Cuando el recolector entregue la caja al transportista, la aplicación móvil llamará a este contrato que tendrá en cuenta todas las condiciones y los datos que han extraído los dispositivos IoT. Si las condiciones se cumplen tal y como se han pactado, la aplicación registrará la transacción, el recolector recibirá el cobro instantáneamente y será entonces cuando el transportista podrá introducir las cajas en sus camiones.

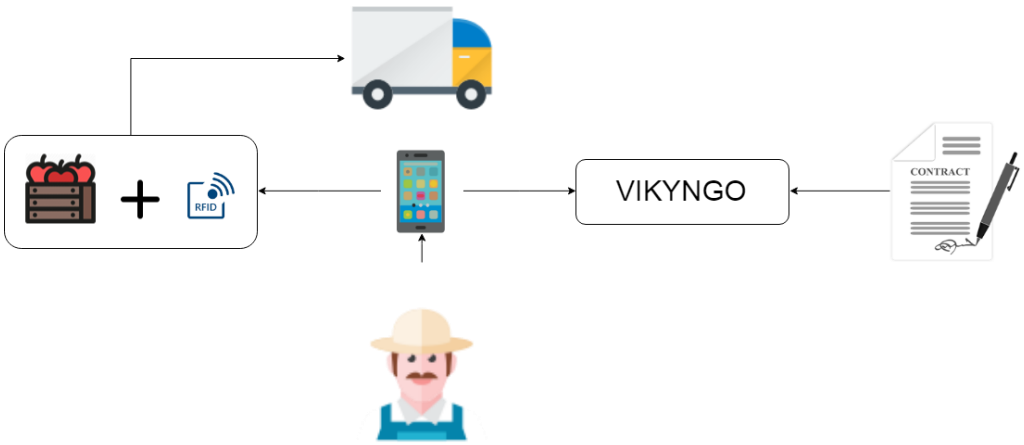

Figura 3-2 – Ejemplo de transacción de un activo

Una vez que las cajas han entrado en el camión, el sistema debe ser capaz de detectar las cajas que están en su interior y poder transmitir los datos de interés de estas. Por ejemplo, localización, vibración o temperatura en el interior del vehículo de transporte.

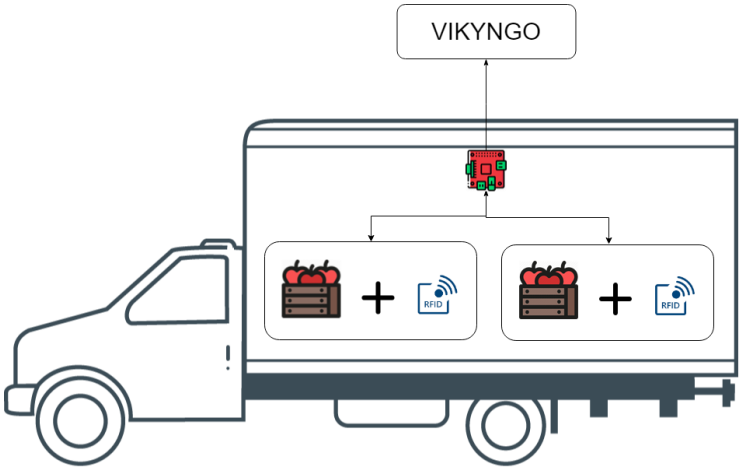

Figura 3-3 – Ejemplo de monitorización en el transporte

Cuando el transportista entregue las cajas tendrá que realizar el mismo proceso que se hizo cuando el productor de manzanas entregó las cajas al transportista. En este caso se podría haber hecho un contrato en el que se refleje el tiempo de entrega o del trato del producto. Si por un casual las cajas han recibido golpes o se han entregado fuera de plazo, el vendedor podría no estar interesado en adquirir esa mercancía.

Una vez que las manzanas han llegado a su destino, se colocan en el escaparate para ser vendidas. En esta situación ya no nos interesa transmitir más datos. Es entonces cuando los usuarios pueden acceder a la información que se ha ido recolectando y almacenando durante toda la cadena. Haciendo uso de una aplicación móvil que sea capaz de leer el código RFID de la caja, se podría enviar una petición a *VIKYNGO* para que este nos devuelva la información relevante al producto identificado y quizá un resumen o valoración del trato del producto.

Teniendo en cuenta el problema expuesto anteriormente, necesitaremos:

- Un aplicativo encargado de administrar la Blockchain y de recibir y almacenar los datos de los dispositivos IoT.
- Una aplicación web donde las empresas pueden administrar los dispositivos y activos y puedan visualizar la información que se está registrando.
- Una aplicación móvil para que el cliente final pueda obtener la información del producto a comprar.
- Un software que se instalará en los dispositivos IoT para que se comuniquen con el servicio Blockchain.

Para este trabajo se ha desarrollado tanto el programa encargado de administrar la Blockchain, como el software IoT y la aplicación web. La aplicación móvil se deja como *Trabajo futuro*.

3.2 Dispositivo IoT

Uno de los dispositivos más famosos es la Raspberry Pi [45], un ordenador del tamaño de una tarjeta de crédito. Para este proyecto utilizaremos el modelo 3B+ por sus potentes características técnicas entre las que destaca un procesador que funciona a 1.4 GHz y una memoria RAM de 1GB. El esquema de las conexiones es el que se muestra en la Figura 3-4. Para el diseño de las conexiones se ha utilizado el programa *Fritzing* [46].

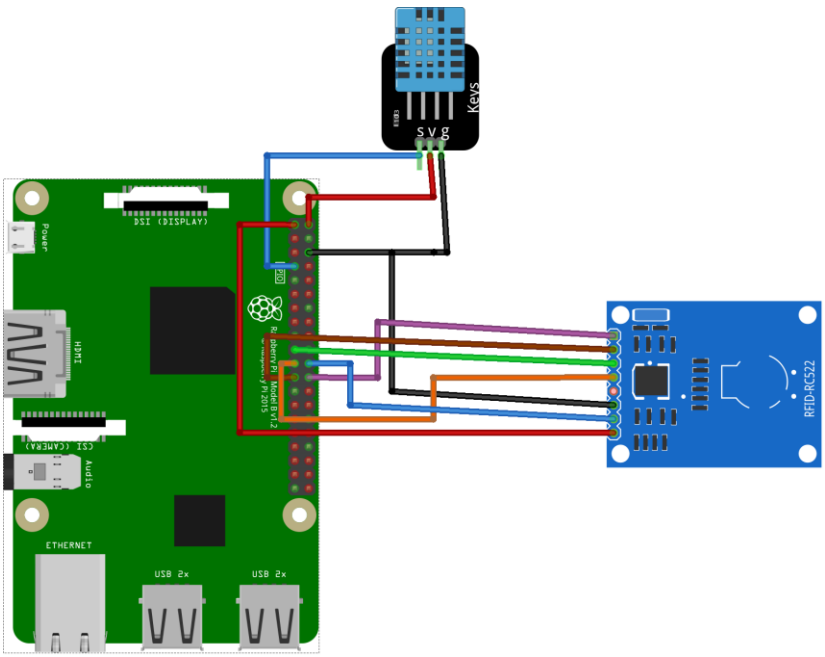

Figura 3-4 – Conexiones de los sensores

Se ha utilizado un sensor de temperatura y humedad DHT11 [47]. Donde los pines de conexión son: un pin de señal, uno de alimentación y otro de tierra.

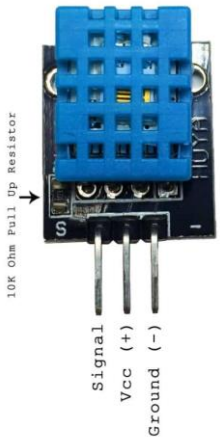

Figura 3-5 – Sensor DHT11

Para detectar la caja de manzanas se ha utilizado un lector RFID RC522 [48]. Analizando los pines en la figura de arriba hacia abajo: SDA (Serial Data Signal), SCK (Serial Clock), MOSI (Master Out Slave In), MISO (Master In Slave Out), IRQ (Interrupt Request), GND (Ground Power), RST (Reset-Circuit) y 3.3V (3.3V Entrada). Para la conexión se utilizarán la **interfaz SPI (Serial Pheriperal Interface)** que es un bus de comunicaciones serie síncrona usado para la transferencia de información entre circuitos integrados que utiliza tres líneas: transmitir (SDO), recibir (SDI) y generar señal de reloj (SCK). Y la interfaz **I2C (Inter-Integrated Circuit)** que es un bus de comunicaciones serie síncrona utilizado para la comunicación entre microcontroladores que utiliza dos líneas: transmitir (SDA) y generar señal de reloj (SCL).

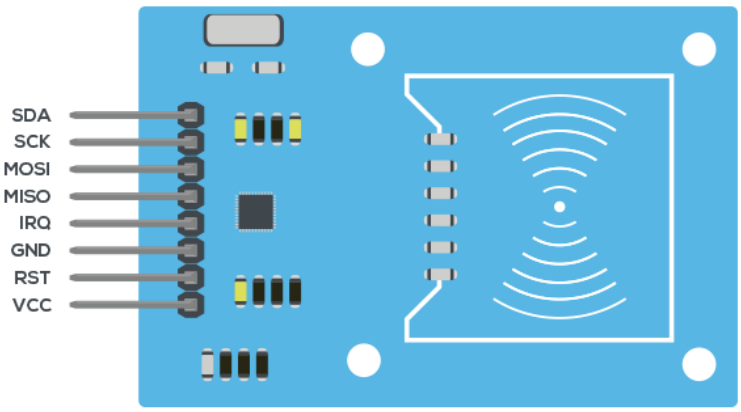

Figura 3-6 – RFID RC522

Por último, se muestra la con los pines del GPIO de la Raspberry Pi Figura 3-7 y la Tabla 3-1 con las conexiones.

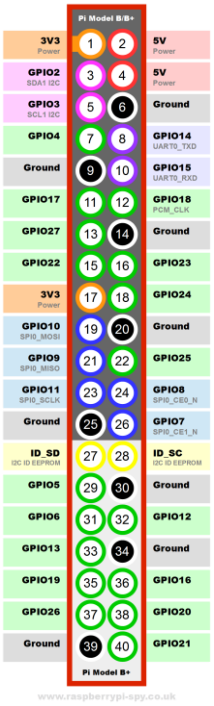

Figura 3-7 – Raspberry Pi 3B+ Pines GPIO

Descripción	Pin RPi 3B+	DHT11	RFID-RC522
VCC 3.3V	#1		VOLTAGE
VCC 5V	#2	VOLTAGE	
GND	#6	GND	GND
GPIO4	#7	SDA	
GPIO10	#19		MOSI
GPIO9	#21		MISO
GPIO11	#23		SCK
GPIO25	#22		RST
GPIO7	#24		SDA

Tabla 3-1 – Conexiones con la Raspberry Pi

3.3 Requisitos del sistema

Antes de embarcarse con una tecnología es esencial entender qué requisitos necesita nuestro aplicativo para saber si es la adecuada. Antes de poder hacer una comparativa vamos a analizar los requisitos que debe tener nuestra herramienta:

- La administración de los nodos debe ser permisionada, pero el acceso a los datos tiene que ser público (red híbrida).
- Debe estar diseñada para manejar activos digitales. Pues es el punto clave de nuestro proyecto.
- Debe ser escalable con un alto procesamiento de transacciones por segundo. Pues estamos en un entorno donde se pueden producir muchas transacciones y eventos en cortos espacios de tiempo.
- Debe tener capacidad de ejecución de contratos inteligentes para los contratos entre empresas de la cadena.
- Debe emplear código libre y ser fácil de desplegar. Pues no tenemos presupuesto para el desarrollo del proyecto, y queremos incrementar su aceptación por parte de las empresas que forman la cadena de suministro.
- Debe emplear una versión estable para evitar problemas que dificulten el desarrollo del proyecto o incrementen los problemas para los usuarios finales.

3.4 Arquitectura de la aplicación

El conjunto tecnológico de Blockchain es muy amplio. Para elegir la tecnología correcta para nuestro proyecto, se ha confeccionado una Tabla 3-2 en la que se comparan cinco tecnologías importantes de código libre. En esta tabla se han representado con colores (verde, naranja y rojo) las calificaciones de cada característica relevante de dichas tecnologías para enfatizar cómo de buenas o malas son.

	Hyperledger Fabric	**Bitcoin**	**Ethereum**	**BigchainDB**	**Quorum**
Permisos	Si	No	Medio	Si	Medio
Contratos Inteligentes	Si	No	Si	Se puede implementar	Si
Activos Digitales	Medio	No	No	Si	No
Escalabilidad	Baja	Alta	Alta	Alta	Alta
Complejidad	Muy Alta	Baja	Baja	Media	Media
Algoritmo de Consenso	Adaptable	PoW	PoS	Tendermint	PoS
Privacidad	Si	No	No	Media	Si
Latencia	Baja	Alta	Alta	Muy baja	Baja
Token	Se puede implementar	Si	Si	Se puede implementar	Si
Versión Estable	Medio	Si	Si	Si	Si

Tabla 3-2 – Comparativa de tecnologías Blockchain

Siguiendo este análisis, *Bitcoin* es descartado por ser una red pública sin permisos. Por otro lado, tanto *Ethereum* como *Quorum* no serían las mejores opciones ya que no están diseñada para funcionar con activos digitales. En cuanto a *Hyperledger Fabric*, podría ser una opción viable, sin embargo, la escalabilidad es un punto muy crítico ya que estamos tratando con casos donde se requerirán muchos nodos. Además, *Hyperledger Fabric* es una tecnología que aún sufre grandes cambios en el código y es de las tecnologías más complejas del conjunto, lo que nos llevaría una gran inversión de tiempo que para un proyecto inicial no es necesario. Finalmente, *BigchainDB* es una tecnología estable, diseñada para trabajar con activos digitales, escalable y con alto procesamiento de transacciones, por lo que parece una buena herramienta para nuestro caso de uso.

En el proceso que se desarrollaba este proyecto salió la noticia de que el mantenimiento de *BigchainDB* se pasaba a la organización IPDB Foundation [49]. Depender de tecnología de terceros implica estar actualizado en las noticias que ocurren con este programa. Ante el anuncio, se decidió contactar con la nueva organización para saber cómo iba a ser el mantenimiento de *BigchainDB*. La organización contestó indicando que el 7 de junio del 2019 publicarían los objetivos de cara al 2019 [50].

A continuación, se procede a elegir la herramienta de trabajo para el desarrollo de la interfaz. Las herramientas más populares actualmente son *Angular* [51], *React* [52] y *Vue.js* [53].

Todas utilizan el lenguaje de programación JavaScript para el desarrollo. En este sentido no hay una clara diferencia a la hora de elegir una u otra. Pero se han intentado extraer los puntos destacables de las 3 para tomar la decisión:

- **Angular**: Es un entorno de trabajo (framework) que además de JavaScript, soporta el lenguaje diseñado por Microsoft, *TypeScript*. *Angular* está orientado a aplicaciones web que van a tener una gran dimensión y es utilizada por Google.
- **React**: Es una librería más flexible que *Angular* estructuralmente, pero más sencilla, que está enfocada en el desarrollo de aplicaciones móviles y mantenida por Facebook.
- **Vue.JS:** Es una librería orientada a aplicaciones simples que ocupa muy poco tamaño y es flexible. Sin embargo, no es tan popular como *Angular* o *React* por lo que habrá menos ejemplos y soporte al desarrollador.

Para este proyecto se ha decidido utilizar *Angular*, debido a que se espera que el proyecto vaya a alcanzar una complejidad más alta en el futuro.

En cuanto al almacenamiento, por un lado, se necesitará guardar información estructurada como pueda ser la información de los usuarios, las cuentas y las entidades. Como base de datos estructurada se ha decidido utilizar *PostgreSQL* por ser de código libre y ser una base de datos más eficiente en proyectos de grandes dimensiones. En la Figura 3-8 se ha diseñado la base de datos relacional. En la tabla "Account" se ha establecido como clave primaria el "ID" de la cuenta. Esta tiene campos como el "type" en el que se indica si es gratuita o de pago, o el campo "BigchainDB" en el que se almacena la localización del nodo asociado. Por otro lado, la tabla "User" que pueden tener acceso a varias cuentas (varios nodos). En esta tabla se almacenan los datos personales de los usuarios y la contraseña encriptada. Por último, la tabla "Entity" en la que se almacenan las claves y los datos de las entidades.

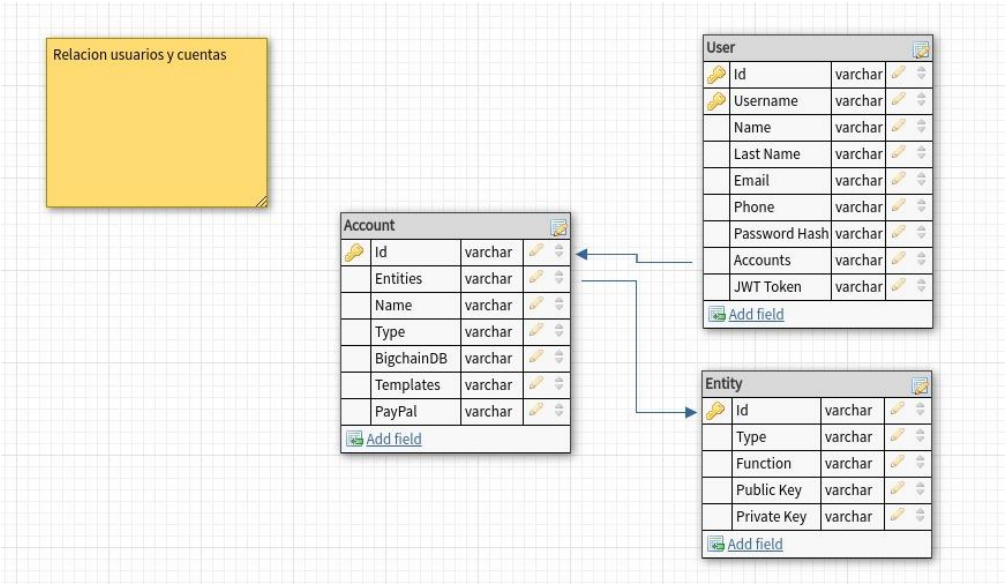

Figura 3-8 – Modelo relacional

Por otro lado, se necesita una base de datos no estructurada para guardar información como documentos o datos de sensores en la que los campos no tienen por qué llevar una estructura fija. En este caso se ha utilizado *MongoDB* ya que es utilizada por *BigchainDB* por lo que conviene llevar una similitud en los programas utilizados.

En la se muestra la arquitectura final de nuestro proyecto. Se ha diseñado siguiendo una arquitectura de microservicios (explicada en el *Estado del arte*) en la que tendremos el servicio de Blockchain con la base de datos en un contenedor y el servicio de la aplicación web en otro contenedor. En IBM Cloud desplegaremos una red Blockchain de cuatro nodos a la que nos conectaremos con un nodo en local. Por último, tendremos el programa de la Raspberry Pi.

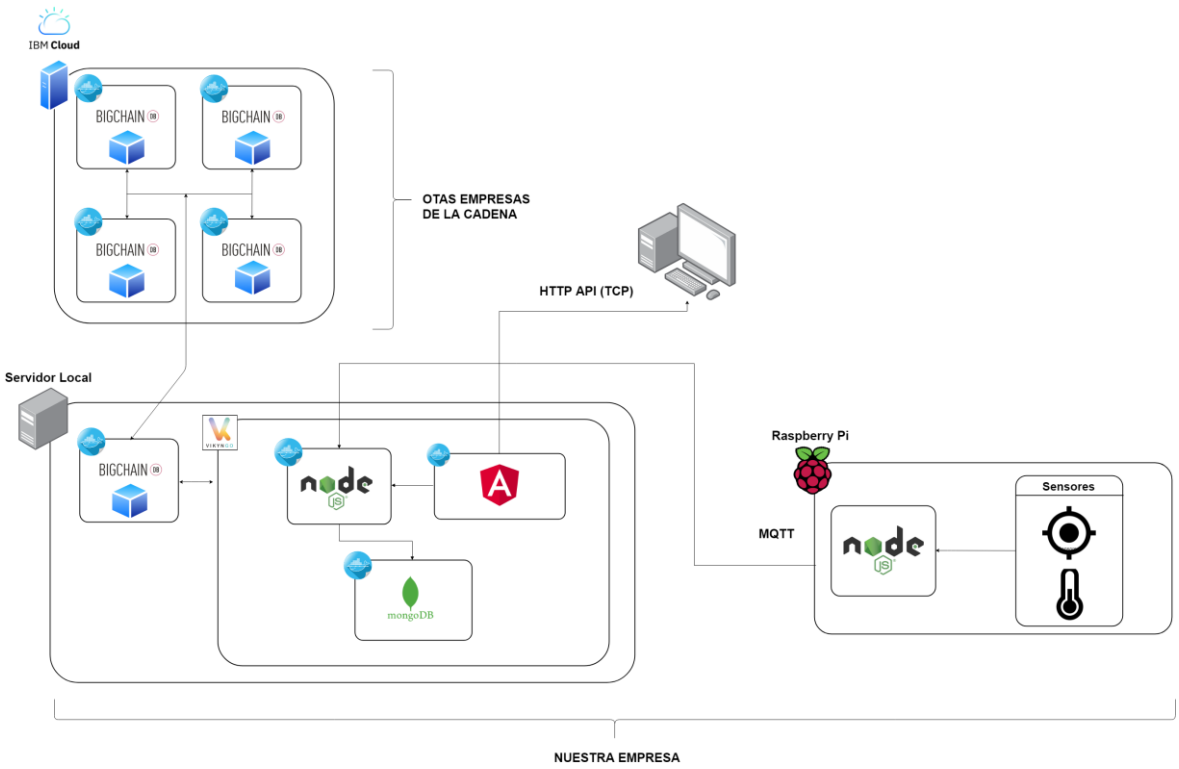

Figura 3-9 – Arquitectura del aplicativo

3.1 Arquitectura del servicio Blockchain

Como se ha comentado en la sección de estado del arte, estructurar el programa permite hacer que sea más fácil de acceder a las funciones, resolver fallos de funcionamiento y realizar pruebas. A continuación, se muestra en la figura la distribución que se ha diseñado para el desarrollo del servidor.

- App: contiene todo el código necesario para que nuestra aplicación funcione. Dentro de esta se encuentras otras carpetas:
    - Actions: Lógica de nuestro servicio.

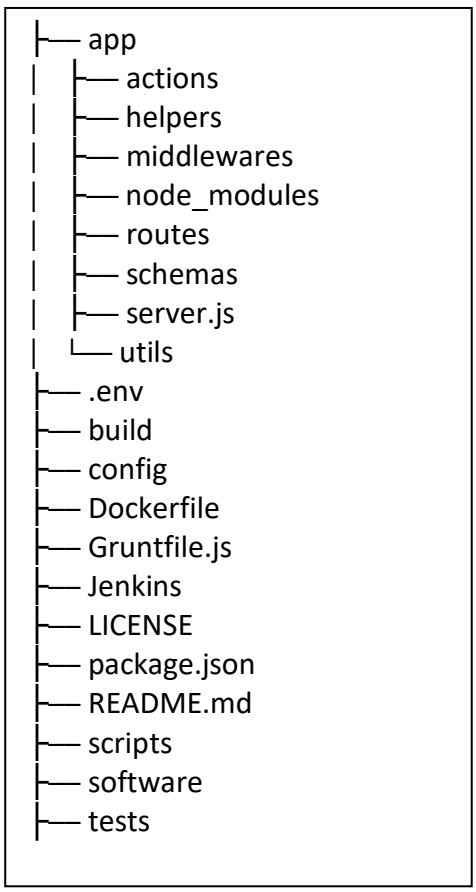

Figura 3-10 – Esqueleto del servicio Blockchain

  - Helpers: Funciones que nos ayudan a conectar con la Blockchain o la base de datos.
  - Middlewares: Programas que se ejecutan antes de
  - Node_modules: Dependencias de nuestro programa.
  - Routes: Todos los métodos disponibles de nuestro servidor.
  - Schemas: Esquemas de validación de las peticiones que recibe nuestro servidor.
  - Server.js: Es el archivo que se ejecuta para arrancar el programa.
  - Utils: Funciones que se utilizan frecuentemente.
- Dockerfile: Contiene la información para crear el contenedor que alberga al servidor.
- Gruntfile.js: Contiene una serie de tareas automáticas como la creación de la documentación o las pruebas unitarias.
- Jenkins: Es el archivo que contiene las tareas para la desplegar el programa automáticamente en el servidor.
- LICENSE: Contiene la licencia del programa.
- Scripts: Aquí se encuentran algunos scripts útiles para algunos servicios.
- Tests: Contiene todos los tests las pruebas unitarias para comprobar que el programa funciona correctamente
- Software: Contiene software que es utilizado en nuestro servicio.
- package.json: Aquí se establecen las librerías de las que depende nuestro servicio.
- README.md: Información sobre nuestro servicio.
- .env: En este archivo se encuentran datos secretos como alguna clave de encriptación.

3.2 Diseño de la aplicación Web

Con el fin de diseñar una plataforma lo más profesional posible, se han estudiado plataformas ya existentes de IoT. Por un lado, la plataforma *Altair SmartCore* [54] que en la Figura 3-11 se muestra una captura de pantalla de la ventana inicial.

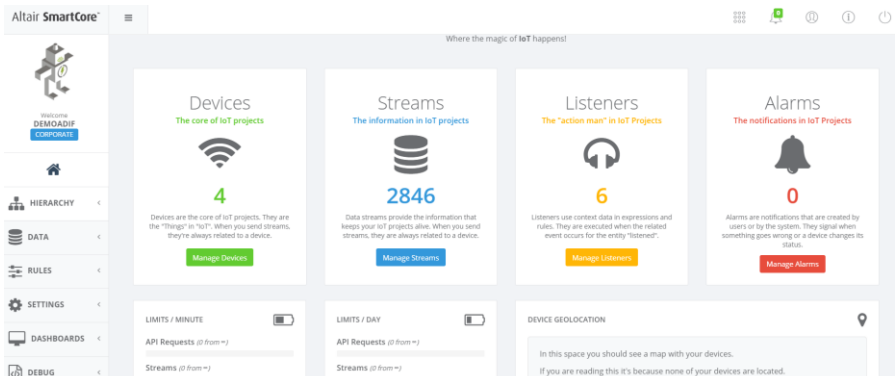

Figura 3-11 – Plataforma de Altair SmartCore

En esta plataforma se puede ver aspecos como el resumen que muestra en los paneles centrales: “devices” que representa a los dispositivos conectados a la plataforma, “streams” que son los mensajes que los dispositivos han enviado a la plataforma, “listeners” que son piezas de código que se ejecutan cuando un “stream” llega a la plataforma y “alarms” que son avisos (sms, e-mail, etc) que se programan para recibir información de los “streams” que llegan a la plataforma. También es interesante el menú de la izquierda, que se muestra en todo momento de la navegación, y la forma en la que se plantea un modelo jerárquico para abordar un proyecto IoT en el que un proyecto puede estar formado por servicios que su vez pueden estar formados por grupos y que estos pueden estar formados por dispositivos. Haciendo uso de esta idea, en *VIKYNGO* se le ha dado un enfoque distinto y se hará uso de las claves asimétricas para diseñar una jerarquía estricta que se comentará en la sección *Entidades*.

Por otro lado, se ha analizado la plataforma *Losant* [55]. Esta plataforma es compleja y mucho menos intuitiva que *Altair SmartCore*. Es una plataforma potente que cuenta con muchas funcionalidades, sin embargo, para una primera aproximación no es un buen modelo a tener en cuenta ya que queremos desarrollar las funcionalidades básicas que puedan resolver los principales problemas de la cadena de suministro. A pesar de esto, se ha encontrado un aspecto interesante que podría servir para *VIKYNGO*, y es el hecho de poder diseñar una plantilla de visualización de datos para los clientes (se tratará en la *Conclusiones y trabajo futuro*).

Después del análisis realizado, a continuación, se muestran las secciones de que dispondrá la aplicación web de WIKYNGO:

- **Inicio**: Se muestra un mensaje de bienvenida, algunos consejos para empezar, noticias relacionadas sobre las versiones del programa, acceso al foro y tutoriales.
- **Resumen**: Se muestra una ventana con distinta información de la cuenta como nodos, mensajes o dispositivos conectados en la Blockchain.
- **Nodos**: Permite administrar (arrancar, parar, reiniciar y configurar) los nodos.
- **Entidades**: Permite administrar (crear, eliminar, transferir) y representar en una gráfica jerárquica las entidades.
- **Datos**: Se muestran los datos almacenados en la Blockchain con el uso de gráficas.
- **Tokens**: Permite configurar y generar un token para una red privada.
- **Contratos**: Permite crear contratos para ser llamados al transferir entidades.
- **Vista de cliente**: Permite crear una plantilla con gráficas para mostrar al cliente que accede a la información de un producto de la cadena.
- **Documentación**: Ofrece documentación de la API para desarrolladores.
- **Ayuda**: Preguntas frecuentes. Contacto con los administradores.
- **Configuración**: Configuración de la cuenta como administradores de esta, sistema de pago, o estado (status) de la cuenta (libre o de pago)
- **Componentes**: Componentes desarrollados por la comunidad para ser utilizados en la plataforma.
- **Recursos**: Programa disponible compatible con VIKYNGO. Por ejemplo, el programa que hemos diseñado para la Raspberry Pi.

- **Incidencias**: Sección donde depositar incidencias o errores encontrados en la plataforma.

3.2.1 Entidades

Se ha denominado entidad a cualquier elemento de la cadena de suministro que vaya a tener datos asociados en la Blockchain, bien sea porque vaya a crear transacciones o bien porque se vayan a almacenar metadatos de este elemento. Las entidades deben tener una clave pública y privada para poder firmar las transacciones. Además, las entidades pueden ser poseedoras de otras entidades teniendo el derecho a operar con ellas o introducir metadatos relacionados con estas.

Tomando como ejemplo nuestra cadena de producción de manzanas, las entidades pueden ser tanto las cajas de manzanas, como los recolectores que las recogen, el transportista o incluso el mismo camión que contiene las cajas. De esta forma, podemos crear una jerarquía y tener un control del estado de todas las entidades para así poder trazarlo. De esta forma podríamos tener una jerarquía la que se muestra en la Figura 3-12.

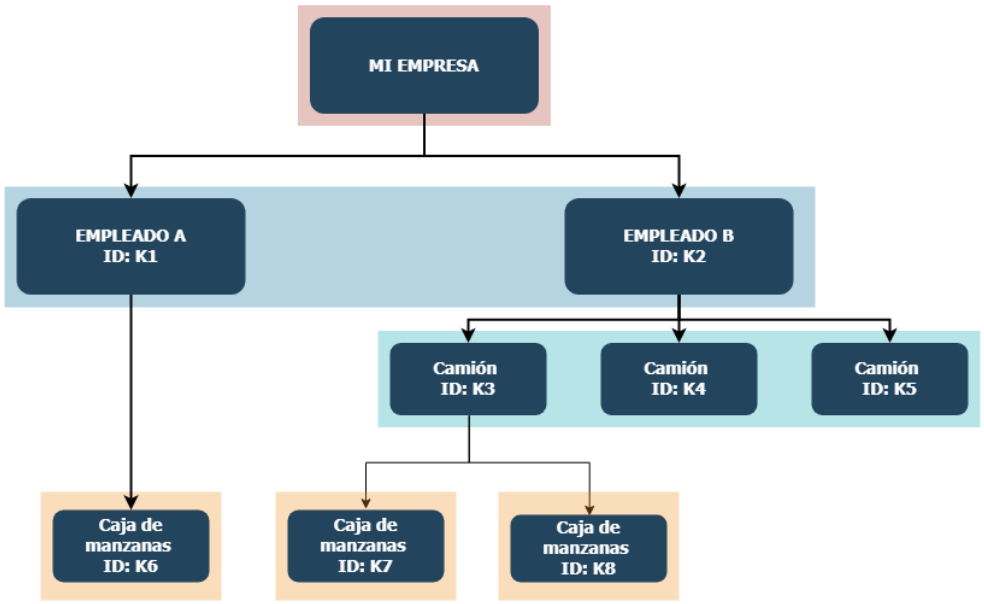

Figura 3-12 – Jerarquía de entidades

En la Figura 3-12 todos los elementos de la jerarquía están identificados con claves públicas (donde Kx representa una clave pública). Así el "EMPLEADO A (ID=K1)", que podría ser un recolector de manzanas, es poseedor de "Caja de manzanas (ID=K6)". Por otro lado, el "EMPLEADO B (ID=K2)" tiene en posesión 3 camiones. A su vez, tiene un camión que posee 2 cajas de manzanas. Si el "EMPLEADO A (ID=K1)" decidiese transferir la caja a "EMPLEADO B (ID=K2)", podría hacerlo de dos formas distintas: Bien ejecutando una transacción directamente a uno de los camiones del "EMPLEADO B (ID=K2)" o bien transfiriendo la caja al "EMPLEADO B (ID=K2)" y este a su vez transfiriéndoselo a uno de los camiones que posee. Cabe recalcar, que el "EMPLEADO A (ID=K1)" no podrá introducir metadatos de las cajas de manzanas que pertenezcan al "EMPLEADO B (ID=K2)".

Este sistema permite llevar un control estricto de las relaciones de posesión entre entidades y por tanto es conveniente para el proyecto. En principio para este trabajo, con

saber el estado actual de las entidades es suficiente. En la sección *Conclusiones y trabajo futuro*.

3.3 Conclusiones

En este capítulo hemos diseñado el proyecto partiendo desde el caso global de una cadena de suministro, entendiendo el proceso a modo de desarrollar una aplicación útil. Posteriormente se ha hecho un análisis en las tecnologías Blockchain y las plataformas IoT del mercado, atendiendo a las necesidades de nuestro sistema. Finalmente, haciendo uso de las técnicas mencionadas en el estado del arte, se ha diseñado la arquitectura de nuestra aplicación.

4 Desarrollo, Integración, pruebas y resultados

A continuación, se explica el desarrollo y las pruebas de la arquitectura mencionada en el apartado anterior. Esta sección se dividirá en tres apartados: servicio Blockchain, dispositivo IoT y aplicación web. Como se comentó en la sección 2.4 y se tratará en la sección 5.2, es recomendable de cara a una aplicación escalable que las pruebas estén automatizadas. En este proyecto se utilizará la herramienta *Postman* [56] para probar la API. También se interactuará con la web para comprobar que se comunica correctamente con nuestra API. Antes de poder realizar las pruebas es necesario crear una red Blockchain a modo de simular los nodos del resto de las empresas. Este proceso se detalla en el Anexo B. Todo el código que se explica a continuación se encuentra en el Anexo C. En este anexo se han introducido las funciones más relevantes para el proyecto, por lo que no aparece todo el código del aplicativo. El código completo se encuentra en el repositorio de VIKYNGO [1].

Antes de poder comprobar que nuestro programa funciona correctamente es necesario arranchar nuestra aplicación. Para realizar este proceso de la forma más automática posible, se han desarrollado 3 tipos distintos de archivos: un archivo denominado "Dockerfile" que se encuentra en cada programa y en el que se especifica la configuración que debe seguir *Docker* para que este arrancar cada programa en un contenedor, un archivo denominado "Docker-compose" en el que se configura la conexión y comunicación de los distintos contendores y un script en lenguaje *Bash* que descarga el contenido de los repositorios y ejecuta el archivo "Docker-composer". Para ejecutar el programa bastará con introducir el comando "`sudo make up`".

4.1 Servicio Blockchain

La función de este servicio es controlar e interactuar con la Blockchain. El servicio Blockchain se tiene que encargar de comprobar que el usuario tenga permisos de acceso, servir una función para arrancar, parar y resetar el nodo, crear transacciones, extraer información de la Blockchain y leer los eventos de los dispositivos IoT.

Para desarrollar este servicio se ha utilizado el entorno *Node.js*. Este entorno trabaja en tiempo real, es de código abierto y multiplataforma. Está especialmente creado para trabajar en el lado del servidor y desarrollar aplicaciones en *JavaScript*. Para desarrollar la API se utilizará el entorno de trabajo (framework) *Express.js* [57], diseñado para interpretar peticiones HTTP.

4.1.1 Autenticación

Para que nuestra aplicación sea segura, se ha implementado un método de obtención de un símbolo (token) de acceso que se muestra en la Tabla C 1. De esta forma, los usuarios que quieran acceder a las funcionalidades de nuestro servicio deberán acompañar en la cabecera del mensaje dicho token. Para la autenticación se ha utilizado la librería *jsonwebtoken* [58] muy útil para la gestión de los tokens. En esta función se muestra el código para generar la documentación (líneas 1-27). Por simplicidad se ha omitido este código en el resto de las funciones. En esta función se habilita un recurso con método POST llamado "`/auth`" que recibe en el cuerpo del mensaje el usuario y la contraseña (líneas 29-30), posteriormente se conecta a la base de datos de *MongoDB* (líneas 32-33) y se hace una petición para extraer la contraseña (líneas 38-40). En caso de ser correcta, se genera un token con el nombre de usuario y el tiempo válido para utilizar los servicios

(líneas 51-52). En caso de que la contraseña sea incorrecta, se devuelve un mensaje HTTP con código de estado "401 No Autorizado (Unauthorised) (línea 48)".

Para comprobar que el token es válido, se ha creado una función (Tabla C 2) que se ejecuta cada vez que se llama a un método de nuestro servicio. Esta función comprueba si el token es válido y que no ha expirado (línea 11). En caso de no ser válido, se devuelve un mensaje HTTP con código de estado "401 No Autorizado (Unauthorised) (línea 13). Con el fin de no mostrar datos secretos en el código (como la semilla de encriptación), se utiliza una librería llamada *dotenv* [59] que lee un archivo donde se encuentran estos datos secretos y los añade a las variables de entorno al arrancar la aplicación.

Una vez diseñado se procede a probar la función.

SOLICITUD

Método:

POST /auth

Cabecera:

```
Content-Type: application/json
```

Cuerpo:

```
{
    "username": "diego",
    "password": "password"
}
```

RESPUESTA

Estado:

200

Cuerpo:

```
{
   "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImRpZWdvIiwia
WF0IjoxNTYzMjk3NTUwfQ.TeUmmIlSLxMgL-wK27AamIVUp2dg36fKyUsUzwIgPOQ"
}
```

Tabla 4-1 – POST /auth

4.1.2 Generación de claves

Antes de comenzar a interactuar con la Blockchain es necesario generar un par de claves para poder registrar una entidad. Para ello se han creado 3 funciones, una para generar una clave pública y privada, otra para obtener todas las claves y otra para borrar una clave. Estas funciones se asemejan bastante por lo que sólo se explicará la función que genera la claves.

Esta función la podemos encontrar en la Tabla C 3, donde podemos que en las líneas 9-10 generamos la clave haciendo uso de una palabra secreta que se almacena en el archivo *.env* y el timespamp. Estas claves se almacenan en *MongoDB* poniendo como identificador a la clave pública para que, en caso de introducir dos claves públicas iguales, no se nos permita hacerlo. En la Tabla 4-2 podemos ver como creamos la clave para un dispositivo con el nombre "raspberry pi" y con identificador "12345". La respuesta que nos devuelve el método es la respuesta que produce *MongoDB* al insertar el documento.

SOLICITUD

Método:

POST /api/v1/keys

Cabecera:

Content-Type: application/json

Authorization:

"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImRpZWdvIiwiaWF0IjoxNTYzMjk3NTUwfQ.TeUmmIlSLxMgL-wK27AamIVUp2dg36fKyUsUzwIgPOQ"

Cuerpo:


```
{
    "data": {
          "type": "device",
          "name": "raspberry pi",
          "id": "12345"
      }
}
```


RESPUESTA

Estado:

200

Respuesta:


```
{
   "result": {
       "n": 1,
       "ok": 1
   },
   "connection": {
       "id": 1,
       "host": "mongo",
       "port": 27017
   },
   "ops": [
       {
           "username": "diego",
           "_id": "69nVysuaLCSvGfSbQ6whNfYaxaxAMczmhULJRoyXKdQ9",
           "privateKey":
"5Y8JHkF7DrN8U2h9m1SPRH68wr3HopjuDbqcDhq9DGfc",
           "data": {
               "type": "device",
               "name": "raspberry pi",
               "id": "12345"
           }
       }
   ],
   "insertedCount": 1,
   "insertedId": "69nVysuaLCSvGfSbQ6whNfYaxaxAMczmhULJRoyXKdQ9",
   "n": 1,
   "ok": 1
}
```


Tabla 4-2– POST /api/v1/keys

4.1.3 Interacción con la Blockchain

Para controlar el nodo se han creado funciones para arrancar, parar, reiniciar, configurar y obtener el estado del nodo. En la Tabla C 4 se muestra la función para iniciar el nodo, en el

que se puede observar que se accede a la carpeta de *BigchainDB* y se ejecuta un script en lenguaje Bash (líneas 2-3).

Para crear las entidades se ha utilizado tanto el controlador oficial de *JavaScript* para *BigchainDB* [60] y un controlador específico para el manejo de activos digitales (*js-driver-orm* [61]). En la Tabla C 5 se muestra la función en la que se crea un activo utilizando *js-driver-orm*. En el Anexo B se han hecho pruebas para comprobar si un nodo está funcionando por lo que aquí haremos pruebas sobre la función de crear una entidad. En la Tabla C 5 se habilita un recurso con método POST llamado "`/create`" (línea 1). Se crea un objeto con la clave pública del activo (línea 2), después se define el modelo del activo (el modelo son los campos que va a tener el activo, como temperatura o humedad) (línea 5) y se llama a la función que registra un activo en la Blockchain (línea 3). Se puede ver como se crea un activo utilizando el par de claves (pública y privada) y el campo "`data`" que almacena la información de los sensores de temperatura y humedad (línea 7). Si el activo ha sido creado con éxito se devuelve un mensaje HTTP con código de estado "201 Creado" (línea 9). Otras funciones que se han implementado y se pueden ver en el código del repositorio son: transferencia de activos Tabla C 7, registro de datos de un activo, eliminación de un activo, obtención de datos de un activo, parar el nodo o arrancar el nodo. Para comprobar que funciona correctamente procedemos a probar nuestro recurso que se puede ver en la Tabla 4-3. La respuesta que recibimos es la que devuelve BigchainDB al crear la entidad:

SOLICITUD

Método:

POST /api/v1/entities

Cabecera:

```
Content-Type: application/json
Authorization:
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImRpZWdvIiwia
WF0IjoxNTYzMjk3NTUwfQ.TeUmmIlSLxMgL-
wK27AamIVUp2dg36fKyUsUzwIgPOQ"
```

Cuerpo:

```
{
    "publicKey":
"69nVysuaLCSvGfSbQ6whNfYaxaxAMczmhULJRoyXKdQ9",
    "modelEntity": "dispositivo",
    "entityInfo": {
        "id": "12345"
    },
    "ownerKeyPair": {
        "publicKey":
"69nVysuaLCSvGfSbQ6whNfYaxaxAMczmhULJRoyXKdQ9",
        "privateKey":
"5Y8JHkF7DrN8U2h9m1SPRH68wr3HopjuDbqcDhq9DGfc"
    },
    "data": {
        "temperature": 20
    },
    "ownerName": "diego"
}
```

RESPUESTA

Estado:

201

Respuesta:


```
{
    "_name": "dispositivo",
    "_schema": {
        "id": "12345"
    },
    "_connection": {
        "path": "http://localhost:9994/api/v1/",
        "headers": {},
        "conn": {
            "path": "http://localhost:9994/api/v1/",
            "headers": {}
        }
    },
    "_appId": "global",
    "transactionHistory": [
        {
            "inputs": [
                {
                    "owners_before": [
                        "69nVysuaLCSvGfSbQ6whNfYaxaxAMczmhULJRoyXKdQ9"
                    ],
                    "fulfills": null,
                    "fulfillment": "pGSAIEyKhxoLSHyHxLNzZoxhHu8LnvD2wyhb_VyZDwD3P75-gUD8hnzR6vLa7QBXRQEq6wyLAIrZBzWQZIY5Iv4R4_wG-JPT0QvzUrxqZ7tHoBqPu93-HVPkf0HjsUjldUuWvMwE"
                }
            ],
            "outputs": [
                {
                    "public_keys": [
                        "69nVysuaLCSvGfSbQ6whNfYaxaxAMczmhULJRoyXKdQ9"
                    ],
                    "condition": {
                        "details": {
                            "type": "ed25519-sha-256",
                            "public_key": "69nVysuaLCSvGfSbQ6whNfYaxaxAMczmhULJRoyXKdQ9"
                        },
                        "uri": "ni:///sha-256;bmcZ91_KKh7oQ5Oicj-aDv2g7hC61Q-ypZL1j249SuE?fpt=ed25519-sha-256&cost=131072"
                    },
                    "amount": "1"
                }
            ],
            "operation": "CREATE",
            "metadata": {
                "temperature": 20
            },
            "asset": {
                "data": {
                    "schema": {
```

```
                            "id": "12345"
                        },
                        "id": "id:global:dispositivo:1fdce690-b13d-
4d2b-b33c-aebe2f4389c1"
                    }
            },
            "version": "2.0",
            "id":
"ada562b1a8718b97e319daac421fad104fb17c62f9757a21832944f87d7404
39"
        }
    ],
    "id": "id:global:dispositivo:1fdce690-b13d-4d2b-b33c-
aebe2f4389c1",
    "data": {
        "temperature": 20
    }
}
```


Tabla 4-3 – POST api/v1/entities

Ahora que hemos creado un dispositivo vamos a comprobar que nuestro método para extraer datos sobre la Blockchain funciona correctamente. En la Tabla 4-4 podemos ver que nuestro método nos devuelve que existe una entidad (la que hemos creado en el apartado anterior).

SOLICITUD

Método:

GET /api/v1/data

Cabecera:


```
Content-Type: application/json
```


Cuerpo:


```
{
    "username": "diego",
    "password": "password"
}
```


RESPUESTA

Estado:

201

Cuerpo:


```
{
    "entities": 1,
    "blocks": 4,
    "metadata": 1,
    "transactions": 1
}
```


Tabla 4-4 – GET/api/v1/data

4.1.4 Bróker MQTT

Para poder escuchar los eventos de los dispositivos IoT que nos estarán enviando la información de los sensores, se utiliza un bróker MQTT. Como se ha comentado en el *Estado del arte* este protocolo es muy común en la arquitectura de aplicaciones IoT. En este caso, se ha utilizado *Mosquitto* [29] como bróker MQTT por su sencillez. El algoritmo implementado se muestra en la Tabla C 6. El algoritmo empieza conectándose al bróker y suscribiéndose a todos los tópicos que empiecen por "asset/" (# es un comodín que indica "cualquiera") (línea 2). Cuando llega un mensaje con tópico "asset/create" (línea 7), entonces se analiza el mensaje recibido. En este caso, si el tópico es "asset/créate", se crea un activo llamando a la función de la sección 4.1.2.

4.2 Dispositivo IoT

Se ha instalado en la Raspberry Pi el sistema operativo *Raspbian Strech* [62] y se ha programado para que lea una tarjeta que lleva en su interior un chip RFID (simulando la caja de manzanas). En la Tabla C 8 se puede ver como se hace la lectura de la tarjeta cada 1 segundo (variable Interval en línea 58). En el caso de detectar una tarjeta y obtener su ID (líneas 5-20), si se ha arrancado el programa en modo "crear", la Raspberry envía un evento con el topic "`asset/créate`" (líneas 27-29), si se ha arrancado en modo "data", extrae la temperatura y la humedad del sensor y se envía al bróker (líneas 30-41).

4.3 Aplicación web

Para poder interactuar con el programa se ha creado una interfaz web. A continuación, se muestran las secciones desarrolladas.

4.3.1 Entidades

Se ha generado una sección en la plataforma, aquí se muestra el resultado para la generación de una entidad:

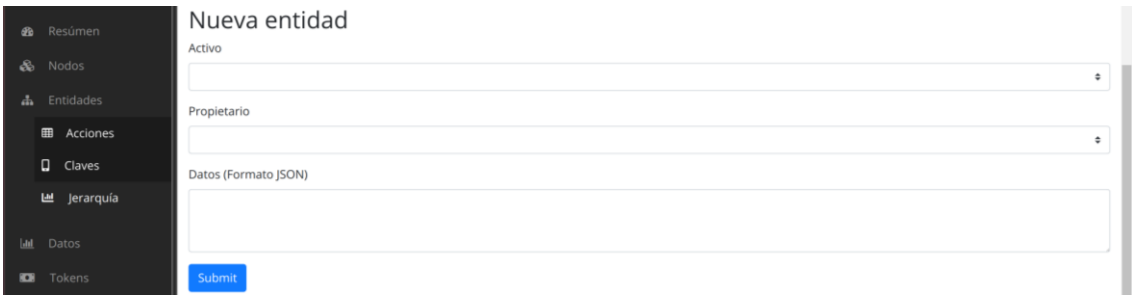

Figura 4-1 – Vista sección crear entidad

Para esta vista se ha generado un formulario HTML. Este formulario carga los activos y los propietarios de la base de datos y nos da la opción de elegir entre varios. Al hacer clic en "`Submit`" se recoge la información seleccionada y la empaqueta en una petición POST que envía al servidor de *VIKYNGO*.

```
1   createAsset(modelname, modelInfo, publicKey, ownerKeyPair, ownerName,
2   data){
3       return this.http.post('http://localhost:3000/api/assets/create',
4   {modelName:modelname, modelInfo: modelInfo, publicKey: publicKey,
5   ownerKeyPair: ownerKeyPair, ownerName: ownerName, data:
```

```
6   data}).subscribe(data => {
7         return data
8       })
9     }
```

Tabla 4-6 – Función crear activo

También se ha desarrollado el algoritmo para generar la jerarquía.

Figura 4-2 – Jerarquía de entidades

El algoritmo funciona recibiendo todos los activos que ha registrado el usuario "Test" y crea un array de objetos JSON con campos "`name`" y "`owner`" (líneas 3-4), a continuación, recorre este array llamando a la función recursiva "`findChild`" que va obteniendo de los hijos desde el principio de la jerarquía (el padre).

4.3.2 Sección de visualización de datos

La sección de visualización de datos se ha hecho la librería *PrimeNG* [63] por tener experiencia previa con ella. En esta sección se puede elegir la entidad a representar y el campo (temperatura, humedad…) que se quiere monitorizar y se ha programado una gráfica que mostrará los datos en la pantalla a modo de tener una demostración visual que se verá más adelante.

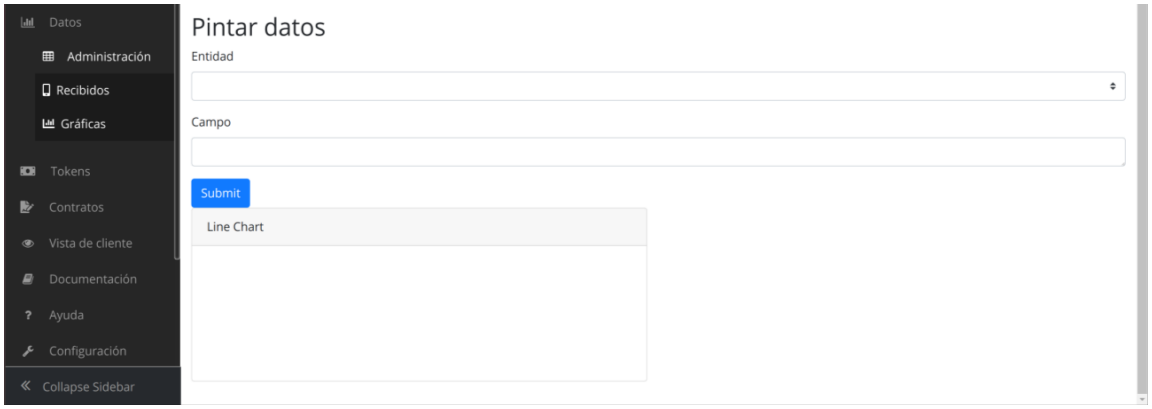

Figura 4-3 – Vista sección pintar datos

4.3.3 Sección de resumen

En esta sección se muestra el resumen de los datos más relevantes. La caja azul muestra los dispositivos que tenemos conectados, la caja amarilla muestra los activos registrados, la caja verde muestra los datos registrados y la caja roja los nodos funcionando.

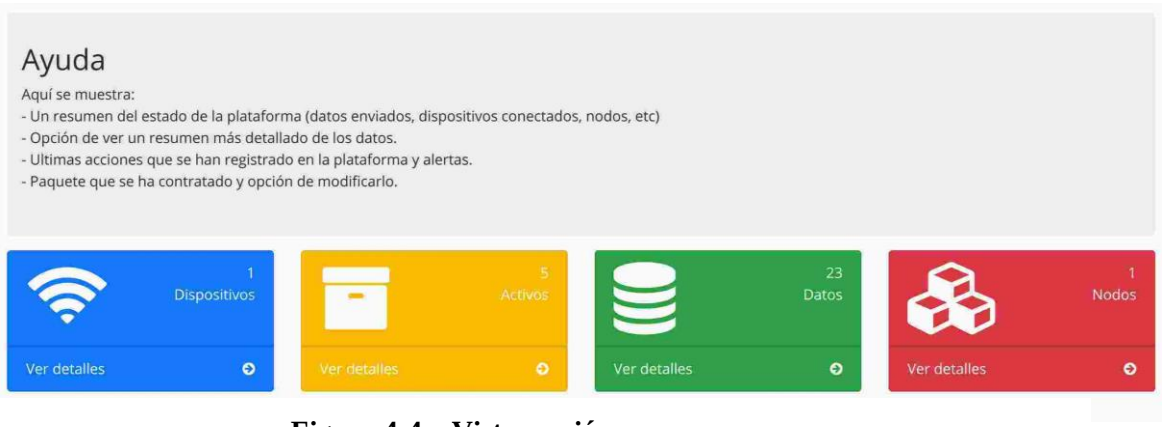

Figura 4-4 – Vista sección resumen

4.3.4 Sección de gestión de claves

En esta sección se ha desarrollado una tabla donde se muestran las claves en una tabla. La sección tiene cuatro entradas donde se pueden introducir datos como el nombre el tipo de entidad, la función que realizan y el sesgo para crear la clave.

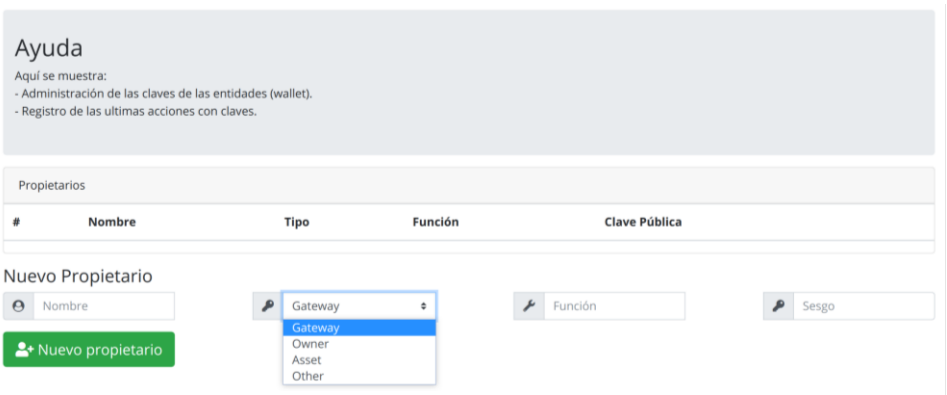

Figura 4-5 – Vista sección claves

4.3.4 Interacción con la web

A modo de comprobar si se monitoriza la temperatura, se van a registrar la información del sensor de temperatura. Para ello, el primer paso es obtener la clave pública y privada para nuestra Raspberry Pi en la plataforma.

Figura 4-6 – Vista crear clave Raspberry Pi

Ponemos la Raspberry Pi en modo "data" y acercamos la tarjeta para registrar transacciones. Se han hecho ocho transacciones y se ha calentado el sensor para que variase la temperatura. Para ver los resultados, accedemos a la sección donde hemos implementado la gráfica de datos y mostramos la temperatura.

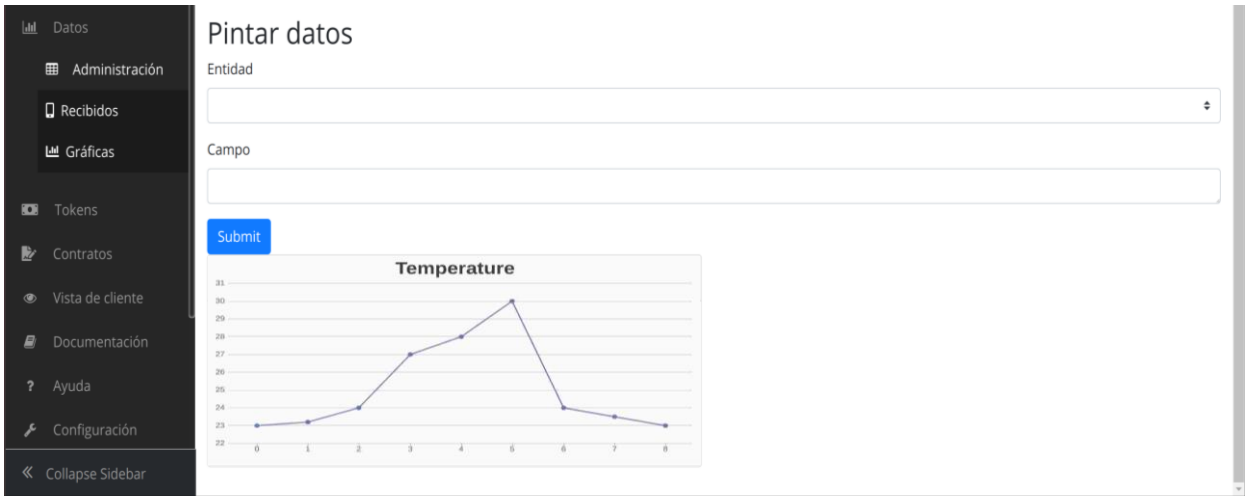

Figura 4-7 – Vista de la temperatura registrada

4.4 Conclusiones

En esta sección hemos podido comprobar cómo se han implementado las funciones para que nuestro aplicativo funcione correctamente. Como se ha podido ver, se han desarrollado funciones tanto para generar las claves necesarias que nos permiten crear las entidades en nuestra cadena de suministro, como las funciones que permiten administrar un nodo Blockchain y su interacción con este y la inyección de datos en el libro de cuentas. También Se ha podido comprobar el aspecto de la plataforma que hace uso de estas funciones. Todo este sistema se ha desarrollado haciendo uso de la arquitectura de Microservicios, empaquetando todos los componentes en un contenedor que se comunican entre sí, y automatizando todo el proceso para que se pueda arrancar el programa con un simple comando.

5 Conclusiones y trabajo futuro

5.1 Conclusiones

La aparición de tecnologías disruptivas como Blockchain y el desarrollo de las redes de comunicaciones y dispositivos electrónicos están mejorando los procesos y sistemas de nuestro día a día. Los dispositivos IoT consiguen digitalizar el mundo analógico en el que vivimos, proviniéndonos de grandes cantidades de datos que tenemos que almacenar y mantener. Gracias a la inmutabilidad que la tecnología Blockchain ofrece, podemos estar seguros de que esos datos no serán alterados. Poco a poco se empiezan a utilizar estas tecnologías en sectores como la industria, especialmente en las cadenas de suministro.

En este trabajo se ha abordado el proceso de las cadenas de suministro y se ha estudiado cómo funcionan las tecnologías Blockchain e IoT y qué utilidad nos pueden dar para mejorar los procesos y añadir nuevos servicios que hasta ahora eran imposibles. Se ha demostrado que se puede mejorar el control y la monitorización de los activos en toda la cadena y añadir contratos inteligentes establecer cláusulas entre los participantes de la cadena. Todo ello haciendo uso de metodologías y tecnologías de software, como la encriptación, los microservicios y los protocolos de comunicación. Además, se ha visto la potencia del lenguaje JavaScript ya que hemos conseguido desarrollar tanto la parte del cliente como del servidor haciendo uso de herramientas y entornos de trabajo de este lenguaje.

Este trabajo será de código libre y se publicará en un repositorio público [1]. A partir del momento en el que se publica este trabajo, será posible que se añadan colaboradores al proyecto con el objetivo de desarrollar los puntos que se mencionan en la siguiente sección.

5.2 Trabajo futuro

Ante la motivación de seguir desarrollando este proyecto, se han ido planteando una serie de mejoras de cara al futuro de *VIKYNGO*.

- **Contratos inteligentes**: Se propone como implementación, añadir la máquina virtual de Ethereum (EVM) con el objetivo de poder ejecutar Smart Contracts de forma más potente. Para ello, se podría crear una sección donde se pudiese programar y compilar contratos en tiempo real (parecido a la plataforma de Remix **¡Error! No se encuentra el origen de la referencia.**) y estos se almacenasen en BigchainDB igual que cualquier entidad, de forma que se pudiesen ejecutar en cualquier momento.
- **App Móvil**: En este trabajo se ha utilizado una Raspberry Pi para realizar la transacción de un activo. Como se ha visto en la sección de diseño, las transacciones se deberían hacer a través de una aplicación móvil a la que los empleados tengan acceso. Para el desarrollo de esta aplicación se propone utilizar el lenguaje Java, muy común en el desarrollo de aplicaciones móviles. Esta aplicación tiene que ser capaz de detectar los chips RFID con los que se identifican las cajas, para ello será necesario acceder al sensor NFC que incorporan los teléfonos móviles de última generación **¡Error! No se encuentra el origen de la referencia.**. Una vez identificado el producto, la aplicación tiene que mostrar una ventana como la que se ha diseñado para la aplicación web en la que permita hacer una transacción de dicho activo llamando a un Smart Contract.

- **Automatización de procesos:** Desarrollar un script o scripts para automatizar el proceso de creación y adición de nodos a la Blockchain de forma que un usuario registrado sólo necesite darle a un botón para ejecutar todo el despliegue. Los scripts no son muy críticos por lo que el lenguaje de programación podría ser JavaScript.
- **Versión de pruebas**: Acabar los servicios necesarios en *VIKYNGO* (tanto el núcleo del software como las secciones de la aplicación web) para poder llevar la aplicación a un entorno de pruebas y poder realizar una simulación en una cadena de producción real.
- **Investigación**: Investigar las brechas de seguridad y líneas que quedan por resolver como: estudiar la optimización y la eficiencia de las tecnologías y plataformas Blockchain para reducir la energía de consumo en los dispositivos IoT, evitar la manipulación del código que presentan los dispositivos IoT y la manipulación de los sensores que obtienen los datos, desarrollar proyectos a gran escala para cadenas de suministro a fin de localizar nuevos problemas que no se detectan a pequeña escala, etc.

Referencias

[1] Diego Sierra Fernandez, Github. [Online] https://github.com/jdiegosierra
[2] Licencia Apache, https://www.apache.org/licenses/LICENSE-2.0
[3] Daniel Wellers, "Is this the future of the Internet of Things", Noviembre 2017. [Online] https://www.weforum.org/agenda/2015/11/is-this-future-of-the-internet-of-things
[4] Eric Hughes, "A Cypherpunk's Manifesto", Marzo 1993. [Online] https://www.activism.net/cypherpunk/manifesto.html
[5] Accenture, "Tracing The Supply Chain". [Online] https://www.accenture.com/_acnmedia/PDF-93/Accenture-Tracing-Supply-Chain-Blockchain-Study-PoV.pdf
[6] Carlos Quilez, "Atún *gourmet* en realidad era atún de contrabando", Mayo 2019. [Online] https://eltaquigrafo.com/atun-goumet-en-realidad-era-atun-de-contrabando/4339/
[7] IBM Food Trust. [Online] https://www.ibm.com/es-es/blockchain/solutions/food-trust
[8] Trazable. [Online] https://trazable.io
[9] Mirror. [Online] https://getmirror.io
[10] Bitcoin. [Online] https://bitcoin.org
[11] SHA256. [Online] https://csrc.nist.gov/CSRC/media/Publications/fips/180/2/archive/2002-08-01/documents/fips180-2withchangenotice.pdf
[12] Alastria. [Online] https://alastria.io/
[13] Ethereum. [Online] https://www.ethereum.org/
[14] "ConsensusPedia: An Encyclopedia of 30+ Consensus Algorithms". [Online] https://hackernoon.com/consensuspedia-an-encyclopedia-of-29-consensus-algorithms-e9c4b4b7d08f
[15] "Tolerancia a las fallas bizantinas", Wikipedia. [Online] https://es.wikipedia.org/wiki/Tolerancia_a_faltas_bizantinas
[16] "Problema de los generales Bizantinos". [Online] https://es.wikipedia.org/wiki/Problema_de_los_generales_bizantinos
[17] Peercoin. [Online] https://peercoin.net
[18] Hasgraph https://hackernoon.com/blockchains-vs-hashgraphs-66a2058c8b43
[19] IPDB Foundation
[20] "How BigchainDB is good for Asset Registrations & Transfers". [Online] https://docs.bigchaindb.com/en/latest/assets.html
[21] "Cosmos Inter-Blockchain Communication (IBC) Protocol". [Online] https://cosmos.network/docs/spec/ibc/
[22] Tendermint. [Online] https://tendermint.com
[23] MongoDB. [Online] https://www.mongodb.com
[24] BigchainDB Whitepaper. [Online] https://www.bigchaindb.com/whitepaper/bigchaindb-whitepaper.pdf
[25] RFID. [Online] https://es.wikipedia.org/wiki/RFID
[26] IEEE 802.11. [Online] http://www.ieee802.org/11/
[27] IEEE 802.15. [Online] https://standards.ieee.org/content/ieee-standards/en/standard/802_15_4-2015.html
[28] MQTT. [Online] http://mqtt.org
[29] Mosquitto. [Online] https://mosquitto.org/
[30] PostgreSQL. [Online] https://www.postgresql.org/

[31] Apache Hadoop. [Online] https://hadoop.apache.org/
[32] D3js. [Online] https://d3js.org/
[33] Chartjs. [Online] https://www.chartjs.org/
[34] Grafana. [Online] https://grafana.com/
[35] Graphite. [Online] https://graphiteapp.org
[36] WCAG2.1. [Online] https://www.w3.org/TR/WCAG21/
[37] Swagger. [Online] https://swagger.io/
[38] Apidoc. [Online] http://apidocjs.com/
[39] ESLint. [Online] https://eslint.org/
[40] Jenkins. [Online] https://jenkins.io/
[41] Mocha. [Online] https://mochajs.org/
[42] OWASP. [Online] https://www.owasp.org/index.php/Main_Page
[43] Docker. [Online] https://www.docker.com
[44] HTTP RFC7230. [Online] https://tools.ietf.org/html/rfc7230
[45] Raspberry Pi. [Online] https://www.raspberrypi.org
[46] Fritzing. [Online] http://fritzing.org/home
[47] DHT11 Humidity & Temperature Sensor. [Online] https://www.mouser.com/ds/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf
[48] MFRC522. [Online] https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf
[49] IPDB Foundation. [Online] https://github.com/ipdb
[50] Manifiesto IPDB Foundation. [Online] https://medium.com/ipdb-blog/ipdb-foundation-assumes-governance-of-bigchaindb-software-and-testnet-51235322e14c
[51] Angular. [Online] https://angular.io/
[52] React. [Online] https://es.reactjs.org/
[53] Vuejs. [Online] https://vuejs.org/
[54] Altair smartcore. [Online]
[55] Losant. [Online] https://www.losant.com/
[56] Postman. [Online] https://www.getpostman.com/
[57] Express.js. [Online] https://expressjs.com/es
[58] jsonwebtoken. [Online] https://www.npmjs.com/package/jsonwebtoken
[59] dotenv. [Online] https://www.npmjs.com/package/dotenv
[60] BigchainDB JavaScript Driver. [Online] https://github.com/bigchaindb/js-bigchaindb-driver
[61] BigchainDB JavaScript ORM Driver. [Online] https://github.com/bigchaindb/js-driver-orm
[62] Raspbian. [Online] https://www.raspberrypi.org/downloads/raspbian/
[63] PrimeNG. [Online] https://www.primefaces.org/primeng/#/
[64] Script 4 Nodes BigchainDB. [Online] https://github.com/bigchaindb/bigchaindb/tree/master/pkg/scripts
[65] Postman. [Online] https://www.getpostman.com/
[66] Anna Corberó, Linkedin. [Online] https://www.linkedin.com/in/anna-corber%C3%B3-551b2147
[67] VIKYNGO, UAM Emprende. [Online] https://uamemprende.es/portfolio/1518
[68] "El método Lean Startup", Eric Ries. [Ref-libro] https://www.amazon.es/m%C3%A9todo-Lean-Startup-utilizando-innovaci%C3%B3n/dp/842340949X
[69] ScrumStudy. [Online] https://www.scrumstudy.com

Glosario

API	Application Programming Interface
UDP	User Datagram Protocol
HTTP	Hypertext Transfer Protocol
TCP	Transmission Control Protocol
IOT	Internet Of Things
XML	Extensible Markup Language
REST	Representational state transfer
MQTT	Message Queuing Telemetry Transport
RDIF	Radio Frequency Identification
XMPP	Extensible Messaging and Presence Protocol
AMQP	Advanced Message Queuing Protocol
CoAP	Constrained Application Protocol
BFT	Byzantine Fault Tolerance
UAM	Universidad Autónoma de Madrid
MVP	Minimum Viable Product
PoW	Proof of Work
PoS	Proof of Stake
RFID	Radio Frequency Identification
OWASP	Open Web Application Security Project
SPI	Serial Peripheral Interface
I2C	Inter-Integrated Circuit

Anexos

A. Emprendimiento

Para este proyecto se ha hecho un trabajo adicional de emprendimiento en el que se ha acudido a un programa de la Universidad Autónoma de Madrid y que se ha creído conveniente documentar debido a la intención de llevar el aplicativo a un entorno real. En este anexo no se profundizará mucho en documentar la información aprendida durante el programa, pero sí dejar constancia de los ejercicios que se hicieron. También se comentarán otros aspectos relacionados con el emprendimiento, como ha sido la búsqueda de un equipo de colaboradores, la organización que se ha diseñado de cara a un futuro desarrollo del proyecto con distintos integrantes y la difusión de proyecto en charlas. Por último, se comentará el proceso de concesión de 1000 euros mensuales que dio IBM para el desarrollo de este proyecto en la plataforma IBM Cloud.

- ***UAM Emprende***

UAM Emprende es un programa en el que se imparten clases, ejercicios y charlas sobre emprendimiento. A este curso se pueden presentar todos los alumnos de la Universidad Autónoma de Madrid. Sin embargo, sólo *VIKYNGO* y otros 19 proyectos más, fueron seleccionados para acceder al curso. Durante el programa, VIKYNGO recibió un premio al mejor planteamiento de idea inicial (Figura A-0-1). El proyecto se puede encontrar en la web oficial del programa [67]. Durante este curso se han tratado conceptos como "Lean Startup", "Propuesta de Valor" y "Tablero de experimentación Javelin".

Figura A-0-1 – Premio UAM Emprende

El **Método Lean Startup** [68] Figura A-0-2 es un proceso que se aplica para la creación de nuevas empresas en entornos de mucha incertidumbre. Está basado en la creación de nuevos productos y servicios por medio del aprendizaje validado, la experimentación y la iteración en los lanzamientos del producto, a través del mínimo producto viable (MVP), teniendo al cliente como fuente del aprendizaje.

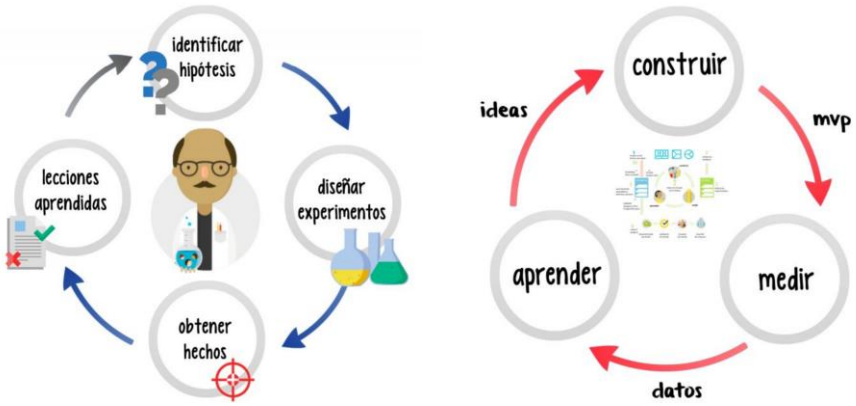

Figura A-0-2 – Método Lean Startup

Siguiendo este método, se plasman en el lienzo de Lean (**Lean Canvas** Figura A-0-3) las hipótesis o suposiciones acerca del modelo de negocio en cada uno de los bloques. Con esto conseguimos tener una visión global a cerca de nuestro proyecto. En la siguiente figura, se muestra el método Lean Canvas aplicado a *VIKYNGO*.

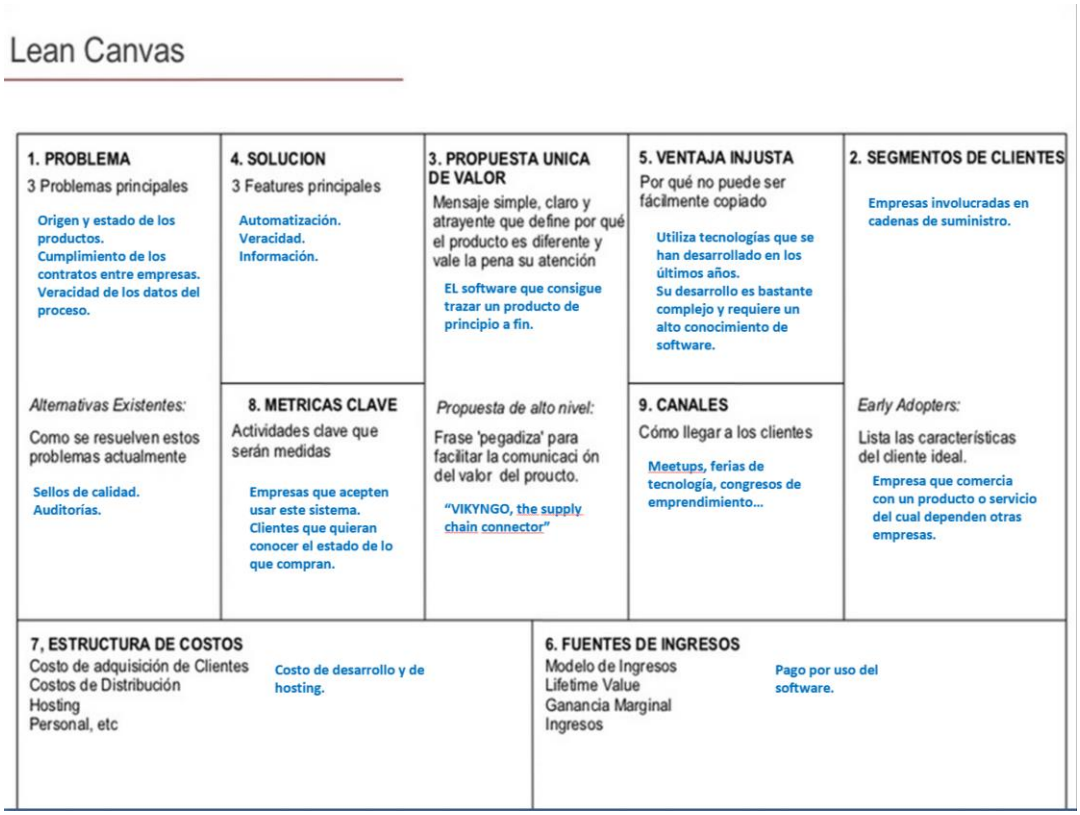

Figura A-0-3 – Lean Canvas

Antes de pasar a validar las hipótesis para convertirlas en certezas, se busca el encaje de la **propuesta de valor** (Figura A-0-4) con el segmento de clientes. Para ello, se utiliza el lienzo de la propuesta de valor. Esta herramienta pretende dar a entender si de verdad el producto/servicio les aporta valor a los clientes, cubre su necesidad o soluciona su problema proporcionando la satisfacción que buscan. La clave de este lienzo es empatizar con el cliente y entender qué le frustra y qué le hace feliz. Para rellenar este lienzo ha sido necesario hablar con posibles clientes interesados en el software.

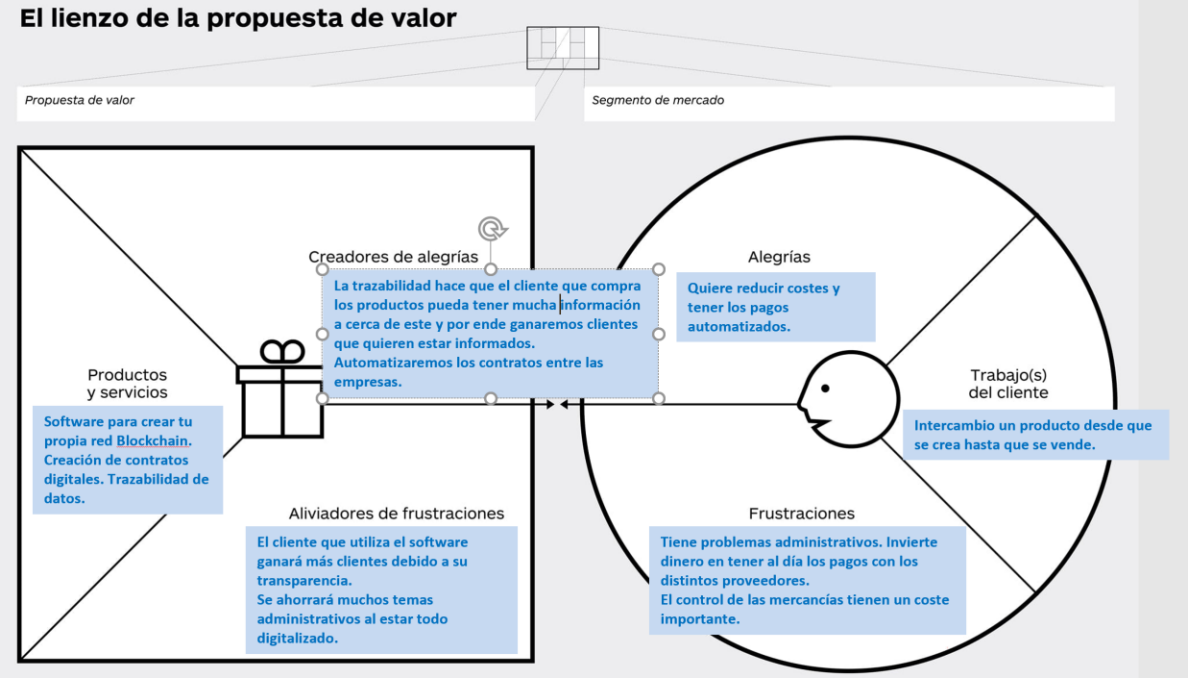

Figura A-0-4 – Lienzo de la propuesta de valor

El **tablero de experimentación Javelin (**Figura A-0-5**)** es una herramienta para validar rápidamente las ideas de una empresa emergente mediante la realización de pruebas y experimentos. El objetivo de utilizar esta herramienta es darse cuenta si estamos desarrollando un producto que nadie quiere. Para rellenar esta tabla primero se hacen ciertas hipótesis y después se habla con los posibles clientes para validarlas. Como se puede observar en la figura, se realizaron 3 iteraciones con los clientes.

Figura A-0-5 – Tablero de experimentación Javelin

Otro de los ejercicios fue diseñar y crear una **web corporativa** en la que mostrar información del producto a los clientes interesados. La web se puede encontrar en www.vikyngo.com.

Figura A-0-6 – Web corporativa VIKYNGO

- ***Organización del equipo***

Durante el programa se encontraron a 4 personas de distintas partes de España para colaborar en el desarrollo del proyecto. Al tratarse de un TFG, **se decidió esperar a la finalización y entrega del trabajo** para comenzar a colaborar en su desarrollo. De cara a la futura organización, se han estudiado e investigado técnicas para la organización en proyectos de estas características:

- **Repositorio:** Es un software que se utiliza para almacenar proyectos al que los programadores tienen acceso para poder contribuir a su desarrollo desde cualquier parte del mundo. Este programa, lleva un control de versiones con el que se puede ver los cambios desde el inicio del proyecto y las contribuciones de cada desarrollador. En este proyecto se ha utilizado la plataforma *Github*. Esta plataforma hace uso del software de control de versiones Git. Si en un futuro alguien quisiera colaborar, bastaría con darle acceso al repositorio.
- **Metodologías ágiles:** Por definición, las metodologías ágiles son aquellas que permiten adaptar la forma de trabajo a las condiciones del proyecto, consiguiendo flexibilidad e inmediatez en la respuesta para amoldar el proyecto y su desarrollo a las circunstancias específicas del entorno. Estas técnicas mejoran la satisfacción del cliente dado que se involucra a lo largo de todo el proyecto. En cada etapa se informa al cliente de los progresos del producto, con el objetivo de involucrarlo directamente y así optimizar las características del resultado final. Para obtener los conocimientos sobre estas metodologías se han seguido los cursos y libros que se facilitan en *ScrumStudy* [69].

- ***Divulgación del proyecto***

Para la divulgación del proyecto se realizarán charlas y artículos de investigación. El 26 de junio se convocó una charla en la que se habló sobre VIKYNGO y la tecnología *BigchainDB* aplicada a las cadenas de suministro.

Figura A-0-7 – Meetup BigchainDB

B. Creando la red Blockchain en IBM Cloud

El proceso para levantar una red Blockchain puede ser bastante complejo según con qué tecnología se trabaje. En el caso de *BigchainDB*, cuenta con un guion de comandos [64] que levanta 4 nodos automáticamente (es el mínimo de nodos que requiere el algoritmo de consenso de *Tendermint*).

Para levantar la red se ha creado una instancia en *IBM Cloud* con Ubuntu 18.04, 1CPU y 4GB de memoria RAM. Accedemos a la instancia utilizando SSH. Una vez dentro, descargamos el repositorio de *BigchainDB* **¡Error! No se encuentra el origen de la referencia.** con la herramienta *Git* y ejecutamos el guión de comandos.

```
1  diego@Diego:~$ ssh root@158.176.64.163
2  diego@Diego:~$ git clone https://github.com/bigchaindb/bigchaindb.git
3  diego@Diego:~$ cd bigchaindb/pkg/scripts/stack
4  diego@Diego:~$ ./stack
```

Tabla B-1 – Proceso de ejecución del guón de comandos

Cada nodo requiere de 3 componentes de software (una instancia de *BigchainDB*, una estancia de *Tendermint* y una instancia de *MongoDB*). Al tener 4 nodos este script genera 12 contenedores Docker:

```
1  root@bigchaindb-core:~# sudo docker ps
2  CONTAINER ID            IMAGE                                          COMMAND
3  CREATED                              STATUS                                PORTS
4  NAMES
5  80fdbbd8a0e6         bigchaindb/bigchaindb:develop   "bigchaindb -l DEBUG…"
6  3 months ago            Up 3 months              0.0.0.0:32815->9984/tcp,
7  0.0.0.0:32814->9985/tcp, 0.0.0.0:32813->45558/tcp   bigchaindb4
8  83d89d361fcd         bigchaindb/bigchaindb:develop   "bigchaindb -l DEBUG…"
9  3 months ago            Up 3 months              0.0.0.0:32809->9984/tcp,
10 0.0.0.0:32808->9985/tcp, 0.0.0.0:32807->45558/tcp   bigchaindb3
11 55c832d32fda         bigchaindb/bigchaindb:develop   "bigchaindb -l DEBUG…"
12 3 months ago            Up 3 months              0.0.0.0:32803->9984/tcp,
13 0.0.0.0:32802->9985/tcp, 0.0.0.0:32801->45558/tcp   bigchaindb2
14 0190437df1e6         bigchaindb/bigchaindb:develop   "bigchaindb -l DEBUG…"
15 3 months ago            Up 3 months              0.0.0.0:32797->9984/tcp,
16 0.0.0.0:32796->9985/tcp, 0.0.0.0:32795->45558/tcp   bigchaindb1
17 c6dc43726887         bigchaindb/tendermint:develop   "bash -c 'cp /tender…"
16 3 months ago          Up 3 months           46656-46657/tcp, 0.0.0.0:32791-
17 >26656/tcp, 0.0.0.0:32790->26657/tcp          tendermint4
18 c9eb9f34038d         bigchaindb/tendermint:develop   "bash -c 'cp /tender…"
19 3 months ago          Up 3 months           46656-46657/tcp, 0.0.0.0:32787-
20 >26656/tcp, 0.0.0.0:32786->26657/tcp          tendermint3
21 ea6d9a3661ba         bigchaindb/tendermint:develop   "bash -c 'cp /tender…"
22 3 months ago          Up 3 months           46656-46657/tcp, 0.0.0.0:32783-
23 >26656/tcp, 0.0.0.0:32782->26657/tcp          tendermint2
24 feaf92da7954         bigchaindb/tendermint:develop   "bash -c 'cp /tender…"
25 3 months ago          Up 3 months           46656-46657/tcp, 0.0.0.0:32779-
26 >26656/tcp, 0.0.0.0:32778->26657/tcp          tendermint1
27 ca0969de9541         mongo:3.6                         "docker-entrypoint.s…"
28 3 months ago            Up 3 months              0.0.0.0:32775->27017/tcp
29 mongodb4
30 50dd5f080d61         mongo:3.6                         "docker-entrypoint.s…"
31 3 months ago            Up 3 months              0.0.0.0:32773->27017/tcp
```

```
mongodb3
959fac755cec        mongo:3.6                          "docker-entrypoint.s…"
3 months ago             Up 3 months              0.0.0.0:32771->27017/tcp
mongodb2
9427101f8d7a        mongo:3.6                          "docker-entrypoint.s…"
3 months ago        Up 3 months          0.0.0.0:32769->27017/tcp
```

Tabla B-2 - Contenedores Docker

Para comprobar que la red funciona correctamente hacemos una petición HTTP, a uno de los nodos de *BigchainDB*, con la aplicación *Postman* [65] que es una aplicación muy útil para hacer peticiones HTTP. Por ejemplo, hacemos un GET al nodo que se encuentra en el puerto 32815 (http://158.176.64.163:32815). La respuesta que nos devuelve el nodo es la que se muestra a continuación:

```
{
   "api": {
       "v1": {
           "assets": "/api/v1/assets/",
           "blocks": "/api/v1/blocks/",
           "docs":
"https://docs.bigchaindb.com/projects/server/en/v2.0.0b9/http-client-
server-api.html",
           "metadata": "/api/v1/metadata/",
           "outputs": "/api/v1/outputs/",
           "streams":
"ws://0.0.0.0:9985/api/v1/streams/valid_transactions",
           "transactions": "/api/v1/transactions/",
           "validators": "/api/v1/validators"
       }
   },
   "docs": "https://docs.bigchaindb.com/projects/server/en/v2.0.0b9/",
   "software": "BigchainDB",
   "version": "2.0.0b9"
}

```

Tabla B-3 – GET 158.176.64.163:32815

Analizando la información que nos ha devuelto, podemos acceder a ciertos datos de nuestra Blockchain. Por ejemplo, si quisiésemos saber cuántos nodos validadores existen en nuestra red (http:// 158.176.64.163:32815/api/v1/validators). Nos devuelve información de los 4 nodos que acabamos de arrancar con el script. En esta respuesta podemos observar ciertos campos:

- **Public_key**: Es el campo donde se muestra la clave pública del nodo
- **Voting_power**: Es el poder de voto que tienen los nodos. Recordemos que para que una transacción sea aceptada, 2/3 partes del poder de voto tienen que haber aceptado esta transacción.

```
[
   {
       "public_key": {
           "type": "ed25519-base64",
           "value": "k6r5vKd+F2bC6SOVX6lYfs9LrIOBtZ5ecqU8VuheZWo="
```

```
        },
        "voting_power": 10
    },
…
…
…
…
  {
        "public_key": {
            "type": "ed25519-base64",
            "value": "xui6nxqjEq2cBEzHpeHzLY9SThNerctsJnEngWj8t9g="
        },
        "voting_power": 10
    }
]
```

Tabla 0-4 - Nodos de la red

El siguiente paso es conectar un nodo local a la red que se acaba de desplegar con el fin de estudiar el proceso de adición de nodos a una red Blockchain. Para este trabajo, el programa encargado de administrar los nodos estará en nuestro servidor local, y por ello, levantaremos un nodo que será nuestra puerta de acceso a la Blockchain. El despliegue del nodo se ha hecho de forma manual, aunque de cara a trabajo futuro sería interesante crear un script que automatizara ese proceso. A continuación, se detallan los pasos que se han seguido basándose en la documentación oficial **¡Error! No se encuentra el origen de la referencia.**:

- Abrir los puertos necesarios para un correcto funcionamiento: TCP en el puerto 22 (SSH), TCP en el puerto 80 (HTTP) y cualquier protocolo en el puerto 26656 (Tendermint P2P). Para ello, se ha utilizado el cortafuegos UFW que viene con Ubuntu 18.04. Por ejemplo, si quisiésemos activar el puerto 80, deberíamos introducir el siguiente comando:

```
diego@Diego:~$ sudo ufw allow 26656
Regla añadida
Regla añadida (v6)

diego@Diego:~$ sudo ufw status
Estado: activo
Hasta                      Acción      Desde
-----                      ------      -----
22/tcp                     ALLOW       Anywhere
80/tcp                     ALLOW       Anywhere
26656                      ALLOW       Anywhere
22/tcp (v6)                ALLOW       Anywhere (v6)
80/tcp (v6)                ALLOW       Anywhere (v6)
26656 (v6)                 ALLOW       Anywhere (v6)
```

Tabla 0-5 - Configuración del cortafuegos

- Ejecutar guión de proceso de instalación de BigchainDB, Tendermint y MongoDB introduciendo el comando `make run` en el repositorio de BigchainDB

- Compartir la clave pública y el identificador del nodo con los administradores del resto de nodos. En nuestro caso, nosotros administramos todos los nodos por lo que bastará con obtener acceder a esta información.

- Uno de los nodos tiene que proponer la entrada de un nuevo nodo validador, introduciendo los datos del nodo desplegado siguiendo el siguiente comando:

  ```
  $ bigchaindb election new upsert-validator <public-key>
  <power> <node-id> --private-key <path-to-the-private-key>
  ```

```
root@bigchaindb-core:~# sudo docker exec -it 0190437df1e6 bigchaindb election new
upsert-validator 9FcaoJurrmKaKiCg4EDv1y0tjdeuczxwjT24RBDACuA= 9
5a46420a0c17ba1f54e3ace0e63f8fd49ec836aa --private-key tmdata/priv_validator.json

[SUCCESS] Submitted proposal with id:
04a067582cf03eba2b53b82e4adb5ece424474cbd4f7183780855a93ac5e3caa
```

Tabla 5-6 - Comando proponer nuevo nodo

- Aprobar la propuesta por 2/3 partes del total de poder de voto. Para ello 3 de los 4 nodos que tenemos instalados deberán introducir el comando `$ bigchaindb election approve <election-id> --private-key <path-to-the-private-key>`

- Una vez aprobado, el siguiente paso es enviarle al nuevo nodo el archivo genesis.json [Anexo **¡Error! No se encuentra el origen de la referencia.**] y configurar el archivo `$HOME/.tendermint/config/config.toml` para que se conecte al menos a uno de los nodos de la red.

Para comprobar que el nodo ha sido validado correctamente podemos hacer un GET al puerto de BigchainDB de nuestro nodo:

```
[
   {
       "public_key": {
           "type": "ed25519-base64",
           "value": "k6r5vKd+F2bC6SOVX6lYfs9LrIOBtZ5ecqU8VuheZWo="
       },
       "voting_power": 10
   },
…
…
  {
       "public_key": {
           "type": "ed25519-base64",
           "value": "9FcaoJurrmKaKiCg4EDv1y0tjdeuczxwjT24RBDACuA="
       },
       "voting_power": 9
   }
]
```

Tabla 0-7 – Resultado de nodos

Podemos observar que nuestro nodo nos devuelve una respuesta y además nos da la información de un nuevo nodo con poder de voto 9.

C. Código del proyecto

```
1   /**
2   * @api {post} /api/v1/login Request Token
3   * @apiName PostLogin
4   * @apiGroup Authentication
5   *
6   * @apiParam {String} username Username unique ID.
7   * @apiParam {String} password Password associated with the username
8   account.
9   *
10  * @apiSuccess {JSON} token Token access.
11  *
12  * @apiSuccessExample {json} Success-Response:
13  *     HTTP/1.1 200 OK
14  *{
15  *    "token":
16  "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImRpZWdvIi
17  wiaWF0IjoxNTYwMzQzMDk5fQ.9xHv04cJFLcOKRaDCwRP6FLKQj2rHfXKQskAMJx-NjA"
18  *}
19  *
20  * @apiError NotAuthorized Invalid password.
21  *
22  * @apiErrorExample Error-Response:
23  *     HTTP/1.1 401 Not Authorized
24  *     {
25  *       "error": "Invalid password"
26  *     }
27  */
28  router.get("", function (req, res, next) {
29     let username = req.body.username;
30     let password = req.body.password;
31
32     MongoClient.connect("mongodb://" + config.MONGODB.HOST + ":" +
33  config.MONGODB.PORT + "/",{ useNewUrlParser: true }, function(err, db) {
34         if (err) {
35            res.status(400).send(err);
36         }
37         else {
38             db.db(config.MONGODB.DB_NAME).collection("users")
39  .find({username: req.body.username}).project({"password":
40  true}).toArray(function(err, resp)  {
41                 if (err) {
42                     db.close();
43                     res.status(400).send(err);
44                  }
45                 else {
46                     db.close();
47                     if (resp[0].password !== password) {
48                         res.status(401).send({error: "Invalid password"})
49                     }
50                     else {
51                         const token = jwt.sign({"username": username},
52  process.env.SECRET_WORD);
53                         res.status(200).json({token});
54                     }
55                 }
56             });
57         }
```

Tabla C 1 – Método de autenticación

```
router.verifyToken = function(req, res, next) {
   var token = req.headers['authorization'];
   if(!token){
       res.status(401).send({
         error: "Es necesario el token de autenticación"
       });
       return;
   }
   token = token.replace('Bearer ', '');

   jwt.verify(token, process.env.SECRET_WORD, function(err, decoded) {
       if(err) {
           res.status(401).send({error: 'Token inválido'})
       }
       else {
           res.locals.username = decoded.username;
           next();
       }
   });
}
```

Tabla C 2 – Método de comprobación de token

```
router.post('/v1/keys', function(req, res) {
   MongoClient.connect("mongodb://" + config.MONGODB.HOST + ":" +
config.MONGODB.PORT + "/",{ useNewUrlParser: true }, function(err, db) {
       if (err) {
          res.status(400).send(err);
       }
       else {
           let keypair = new
BigchainDB.Ed25519Keypair(bip39.mnemonicToSeed(process.env.
SECRET_WORD+new Date()).slice(0, 32));
           db.db(config.MONGODB.DB_NAME).collection(collection).
insertOne({username: res.locals.username, _id: keypair.publicKey,
privateKey: keypair.privateKey, data: req.body.data}, function(err, resp) {
               db.close();
               if (err) {
                   res.status(400).send(err);
               }
               else {
                   res.status(200).send(resp);
               }
           });
       }
   });
});
```

Tabla C 3 – Método de creación de claves

```
router.post('/v1/blockchain/start', function (req, res) {
   exec("( cd ./software/bigchaindb/ && sudo make start )", { shell:
'/bin/bash' },
       (err, stdout, stderr) => {
           if (err !== null) {
               res.status(500).send(err);
           }
           else {
               res.sendStatus(201);
           }
       });
});
```

Tabla C 4 – Método de arranque del nodo Blockchain

```
router.post('/v1/entities/create', async function (req, res) {
   const assetOrm = new DID(req.body.publicKey
   assetOrm.define(req.body.modelEntity, req.body.entityInfo);
   const asset = await assetOrm.models[req.body.modelEntity].create(
       {
           keypair: req.body.ownerKeyPair,
           data: req.body.data
       });
   res.status(201).send(asset);
});
```

Tabla C 5 – Método de creación de entidades

```
client.on('connect', function (err) {
   client.subscribe("asset/#", function (err) {
       console.log(err);
   });
});
client.on('message', function (topic, message) {
   if (topic === 'asset/create') {
       console.log("Registrando activo");
       let data = JSON.parse(message);
       let keypair = new driver.Ed25519Keypair(
           bip39.mnemonicToSeed(data.BusinessID).slice(0, 32))
       bigchainDB.createAsset(data.AssetData, data.AssetMetadata,
           keypair.publicKey, keypair.privateKey);
   }
}
```

Tabla C 6 – Método para conectar con el Broker MQTT

```
router.post('/v1/entities/transfer', async function (req, res) {
       const transactionHistory = await
conn.listTransactions(req.body.assetId;
       console.log(transactionHistory)
       const txId = transactionHistory[transactionHistory.length - 1].id;
       const txCreated = await conn.getTransaction(txId);
       const createTranfer =
BigchainDB.Transaction.makeTransferTransaction(
           [{ tx: txCreated, output_index: 0 }],
           [BigchainDB.Transaction.makeOutput(BigchainDB.Transaction
.makeEd25519Condition(req.body.outputPublicKey))],
           req.body.metadata
```

```
        )
        const signedTransfer =
BigchainDB.Transaction.signTransaction(createTranfer,
req.body.privateKey);
        const response = await conn.postTransactionCommit(signedTransfer);
        res.status(200).send(response);
});
```

Tabla C 7 – Método de transferencia

```

var interval = setInterval(function () {
   // reset card
   mfrc522.reset();
   // Scan for cards
   let response = mfrc522.findCard();
   if (!response.status) {
       return;
   }
   console.log("Activo detectado");
   // Get the UID of the card
   response = mfrc522.getUid();
   if (!response.status) {
       console.log("UID Scan Error");
       return;
   }
   // If we have the UID, continue
   let uid = response.data;
   let ID = uid[0].toString(16) + uid[1].toString(16) + uid[2].toString(16)
+ uid[3].toString(16);
   console.log("ID: " + ID);
   // Stop
   mfrc522.stopCrypto();

   // Check broker connection
   if (client.connected == true) {
       if (process.env.ASSET_ACTION === "create") {
           client.publish('asset/create', '{"AssetData": {"AssetID": "' +
ID + '"}, "AssetMetadata": ' + null + '}');
       }
       if (process.env.ASSET_ACTION === "data") {
           let metadata;
           sensor.read(type, pin, function (err, temperature, humidity) {
               if (!err) {
                   metadata = {
                       temeprature: temperature.toFixed(1),
                       humidity: humidity.toFixed(1)
                   }
               }
               client.publish('asset/data', '{"AssetData": {"AssetID": "' +
ID + '"}, "AssetMetadata": ' + metadata + '}');
           });
       }
       if (process.env.ASSET_ACTION === "transact") {
           client.publish('asset/transact, '{ "AssetData": { "AssetID": "'
+ ID + '" }, "AssetMetadata": ' +  null + ' }, to: ' + pubKey + ', from:
keyPair + '}');
       }
   }
   else {
       console.log("No se ha podido conectar con el broker");
```

```
52          return;
53      }
54      // If create action then only 1 interval
55      if (process.env.ASSET_ACTION === "create") {
56          clearInterval(interval);
57      }
58  }, interval)
```

Tabla C 8 – Código Raspberry Pi