



**UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA**

GRADO EN INGENIERÍA INFORMÁTICA

Tecnologías de la información

TRABAJO FIN DE GRADO

Plataforma Blockchain de gestión de activos digitales mediante tokens no fungibles (NFTs)

Juan Manuel Porrero Almansa

junio, 2022



**UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA**

**Dept. del Tutor
(cont.)**

Tecnologías de la Información

TRABAJO FIN DE GRADO

**Plataforma Blockchain de gestión de activos digitales
mediante tokens no fungibles (NFTs)**

Autor: Juan Manuel Porrero Almansa

Tutor académico: Félix Jesús Villanueva Molina

Tutor FORTE: Manuel Hervás Ortega

junio, 2022

Plataforma Blockchain de gestión de NFTs
© Juan Manuel Porrero Almansa, 2022

Este documento se distribuye con licencia CC BY-NC-SA 4.0. El texto completo de la licencia puede obtenerse en <https://creativecommons.org/licenses/by-nc-sa/4.0/>.

La copia y distribución de esta obra está permitida en todo el mundo, sin regalías y por cualquier medio, siempre que esta nota sea preservada. Se concede permiso para copiar y distribuir traducciones de este libro desde el español original a otro idioma, siempre que la traducción sea aprobada por el autor del libro y tanto el aviso de copyright como esta nota de permiso, sean preservados en todas las copias.

Este texto ha sido preparado con la plantilla L^AT_EX de TFG para la UCLM publicada por [Jesús Salido](#) en GitHub¹ y Overleaf² como parte del curso «[L^AT_EX esencial para preparación de TFG, Tesis y otros documentos académicos](#) impartido en la Escuela Superior de Informática de la Universidad de Castilla-La Mancha.



¹https://github.com/JesusSalido/TFG_ESI_UCLM, DOI: 10.5281/zenodo.4574562

²<https://www.overleaf.com/latex/templates/plantilla-de-tfg-escuela-superior-de-informatica-uclm/phjgscmfqtsw>

TRIBUNAL:

Presidente: _____

Vocal: _____

Secretario(a): _____

FECHA DE DEFENSA: _____

CALIFICACIÓN: _____

PRESIDENTE

VOCAL

SECRETARIO(A)

Fdo.:

Fdo.:

Fdo.:

*A mi tío Félix
Por ser mi mayor ejemplo de niño y guiarme en mi etapa adulta
De ti viene mi motivación y pasión por la informática*

Plataforma Blockchain de gestión de NFTs

Juan Manuel Porrero Almansa
Ciudad Real, junio 2022

Resumen

La tecnología Blockchain ha cobrado especial importancia en los últimos años ya sea por el impacto de las criptomonedas, las colecciones de arte de los NFTs. Es una realidad que a día de hoy es mucho mayor el número de personas que al menos han oído hablar de ella que hace unos años.

La tecnología de cadena de bloques es famosa por su seguridad, de modo que tiene otras aplicaciones a parte de las mayormente conocidas, como, por ejemplo, un sistema de voto en el que cada voto es único y se almacena en una red Blockchain. Cada voto sería un NFT único. Así es como surge este TFG, como una utilidad de las muchas que tiene esta tecnología.

Un **Marketplace** es un sitio web donde se puede comprar un producto de nicho de los muchos que se ofertan y aprovechando la tecnología Blockchain, se generará un token que demuestre la autoría del comprador. La blockchain desarrollada es una red privada construida en un *cloud* y al utilizar **Hyperledger Fabric**, es **compatible** con los contratos y tokens de **Ethereum**.

El Marketplace desarrollado permite la diferenciación de roles: administrador y usuario. Como administrador, se podrá generar un token, estableciendo su precio y principales características y también eliminarlo o “quemarlo”. Como usuario, se podrá comprar un NFT, consultar los NFTs de su colección, pujar por un NFT y regalar un NFT.

Blockchain platform for the management of NFTs

Juan Manuel Porrero Almansa
Ciudad Real, June 2022

Abstract

Blockchain technology has become particularly important in recent years, whether it is the impact of cryptocurrencies, art collections or NFTs. It is a reality that today far more people have at least heard of it than a few years ago.

Blockchain technology is famous for its security, so it has other applications besides the mostly applications, such as a voting system in which each vote is unique and stored in the blockchain. is unique and stored in a Blockchain network. Each vote would be a unique NFT. This is how this This is how this TFG arises, as one of the many uses of this technology.

A Marketplace is a website where you can buy a niche product of the many that are offered and, taking advantage of Blockchain technology, a token will be generated to prove the buyer's authorship. The blockchain developed is a private network built in a cloud and by using Hyperledger Fabric, it is compatible with Ethereum contracts and tokens.

The developed Marketplace allows the differentiation of roles: administrator and user. As an administrator, you can generate a token, setting its price and main characteristics and also delete or "burn" it. As a user, you can buy an NFT, consult the NFTs in your collection, bid for an NFT and give an NFT as a gift.

Agradecimientos

Lo primero agradecer a mis padres y mi hermano por haberme dado los valores correctos y los recursos necesarios para llegar aquí, también acordarme de mi perrita Dona, que siempre me alegra los días. (Pilar, no llores mucho leyendo esto)

Mencionar a mis abuelos Teo y Félix, unas de las personas que más se alegran por todo lo bueno que me pase, quizá algún día pueda comprarte un coche bien grande, ¿no?

Agradecer a los amigos que me ha dado la carrera como Pablo, Gualo y Santi los buenos ratos en clase y fuera de esta que hemos pasado, y también a los que ya formaban parte de mi vida como Desi, Ernesto y Alonso. No me olvido de LV, por estar siempre ahí y apoyarme.

También me acuerdo de aquellos de profesores que hacían de clase un lugar mejor, que me motivaron a asistir a clase y tutoría por voluntad propia y no tener problema en preguntarles dudas como fueron Isidro, Ramón, Pepe, María Isabel y Macario.

Agradecer enormemente a la empresa Alpinia y a todos los compañeros con los que he trabajado, especialmente a Manu, que me ha ayudado todo lo posible y más y ha hecho que no pueda estar más cómodo y contento de haber realizado mi TFG con ellos.

Por último, dar las gracias al Juanma de primero de carrera por haber seguido adelante.

Gracias a todos.

*Juan Manuel Porrero Almansa
Ciudad Real, 2022*

Acrónimos

LISTA DE ACRÓNIMOS

Ejemplo de lista con los acrónimos empleados en el texto.

TFG	:	Trabajo Fin de Grado
NFT	:	Non-Fungible Token
FT	:	Fungible Token
P2P	:	Peer-to-Peer
DES	:	Data Encryption Standard
AES	:	Advanced Encryption Standard
ECB	:	Electronic Codebook
CBC	:	Cipher Block Chaining
CFB	:	Cipher Feedback
CTR	:	Counter Mode
SHA	:	Secure Hash Algorithm
NIST	:	Instituto Nacional de Estándares y Tecnología
FIPS	:	Estándares Federales de Procesamiento de la Información
MD5	:	Message-Digest Algorithm 5
RSA	:	Rivest, Shamir y Adleman
DSS	:	Data Security Standard
DSA	:	Digital Signature Algorithm
ECDSA	:	Elliptic Curve Digital Signature Algorithm
CA	:	Certificate Authority
ESI	:	Escuela Superior de Informática
ERC	:	Ethereum Requests for Comments
USB	:	Universal Serial Bus
MP3	:	MPEG Audio Layer III
JPG	:	Joint Photographic Experts Group
STO	:	Secure Token Offering
IDE	:	Integrated Development Environment
API	:	Application Programming Interfaces
REST	:	Representational State Transfer
J2EE	:	Java™ 2 Platform, Enterprise Edition
TLS	:	Transport Layer Security

Índice general

Resumen	V
Abstract	VII
Agradecimientos	IX
Acrónimos	XI
Índice de figuras	XVII
Índice de tablas	XIX
Índice de listados	XXI
1. Introducción	1
1.1. Antecedentes históricos	1
1.2. Redes descentralizadas y P2P	2
1.2.1. Redes descentralizadas	2
1.2.2. Redes P2P	3
1.3. Proyecto Bitcoin	3
1.3.1. Transacciones en Bitcoin	4
1.3.2. Servidor de sellado de tiempo	5
1.3.3. Proof-of-work	5
1.3.4. Red	6
1.4. Impacto Socio-Económico	6
1.5. Motivación	7
1.6. Estructura del documento	7
2. Antecedentes	9
2.1. Criptografía	9
2.1.1. Criptografía simétrica	9
2.1.2. Autenticación de mensajes y funciones Hash	10
2.1.3. Cifrado de clave pública	13
2.1.4. Firmas digitales	15
2.2. Blockchain	16
2.2.1. Funcionamiento	16
2.2.2. Tipos de Blockchain	18
2.3. Smart-Contract	19
2.4. Proyecto Ethereum	20
2.4.1. Ether	20
2.5. Tokens	20

2.5.1. Token Fungible (FT)	21
2.5.2. Token no fungible (NFT)	21
2.5.3. Cómo se compran NFTs	21
2.6. El estándar ERC-721	22
2.6.1. Otros ERC de interés	22
2.7. Hyperledger Fabric	23
3. Objetivo	25
3.1. Objetivo general	25
3.2. Objetivos específicos	25
3.2.1. Construcción de una red Blockchain	25
3.2.2. Creación de un contrato inteligente	25
3.2.3. Interfaz para acceder a los servicios	26
3.2.4. Custodia de credenciales y certificados de usuario	27
3.2.5. Arquitectura desacoplada	27
3.2.6. Se deben utilizar tecnologías libres	27
3.2.7. Despliegue fácil, rápido y eficaz	27
4. Metodología	29
4.1. Metodología de trabajo: metodologías ágiles	29
4.2. Scrum	30
4.3. Aplicación de la metodología de trabajo	31
4.4. Diseño	32
4.4.1. Historias de usuario	32
4.4.2. Requisitos no funcionales	36
4.4.3. Definición de los sprints	36
4.5. Marco tecnológico	37
4.5.1. Hardware	37
4.5.2. Software	37
5. Arquitectura	39
5.1. Módulo Marketplace	39
5.1.1. Desarrollo con patrón MVC: Modelo-Vista-Controlador	39
5.1.2. Desarrollo con Java Spring Boot	40
5.2. Módulo API-gateway	44
5.2.1. Gateway	44
5.2.2. Event-bus	44
5.2.3. Transaction-manager	45
5.2.4. Event-listener	45
5.3. Módulo Hyperledger Fabric	45
5.3.1. Peers	45
5.3.2. CAs	46
5.3.3. Channel y Organizations	46
5.3.4. Contrato inteligente	46
5.3.5. Docker y Kubernetes	47
6. Resultados	49
6.1. Estadísticas del repositorio	49
6.2. Prototipo construido	50
6.2.1. Explicación del archivo Docker-compose	55
6.3. Costes	57
6.3.1. Coste de hardware	57

6.3.2. Coste de desarrollo	57
6.3.3. Coste de software	58
6.3.4. Aspecto de la aplicación	58
7. Conclusiones	63
7.1. Objetivos cumplidos	63
7.2. Competencias adquiridas	64
7.3. Trabajos futuros	64
Referencias	65
A. COMPETENCIAS DE TECNOLOGÍAS DE LA INFORMACIÓN	69

Índice de figuras

1.1.	Diferencias entre redes centralizadas, descentralizadas y distribuidas [21]	2
1.2.	Conexión entre usuarios: cliente-servidor y redes P2P [8]	3
1.3.	Precio de Bitcoin desde octubre de 2013 hasta enero de 2022, en dólares [17]	4
1.4.	Funcionamiento de transacciones en Bitcoin [14]	5
1.5.	Ejemplo de la cadena de sellado de tiempo [14]	5
1.6.	Funcionamiento de proof-of-work [14]	6
2.1.	Esquema de cifrado con criptografía simétrica [20]	10
2.2.	Autenticación de mensajes usando un código de autenticación de mensaje [20]	12
2.3.	Función Hash; $h = H(M); P, L = padding$ más longitud [20]	13
2.4.	Cifrado con clave pública [20]	14
2.5.	Cifrado con clave privada [20]	14
2.6.	Ejemplo simplificado de flujo esencial para el proceso de firma digital [20]	16
2.7.	Ejemplo de uso de certificado de clave pública [20]	17
2.8.	Ejemplo de Blockchain y sus bloques sucesivamente unidos [22]	17
2.9.	Ejemplo de bloque de Blockchain: cabecera y body. [22]	18
2.10.	Logo de ETH [10]	20
2.11.	En rojo, ejemplo de utilización token de seguridad en URL	20
4.1.	Metodologías ágiles más usadas. [1]	30
4.2.	Proceso de desarrollo en Scrum [6]	31
5.1.	Diagrama del módulo Marketplace aislado	40
5.2.	Estructura básica del patrón MVC	40
5.3.	Diagrama de comunicación (desacoplada) con MVC de la aplicación.	41
5.4.	Diagrama del módulo API-gateway aislado	44
5.5.	Diagrama del módulo Blockchain aislado	46
5.6.	Arquitectura completa del proyecto	48
6.1.	Estadísticas repositorio API-Gateway 1/2	49
6.2.	Estadísticas repositorio Event-Bus	49
6.3.	Estadísticas repositorio API-Gateway 2/2	50
6.4.	Estadísticas repositorio Event-Listener	50
6.5.	Estadísticas repositorio Transaction-Manager	50
6.6.	Estadísticas repositorio Smart Contract	50
6.7.	Estadísticas repositorio BackEnd aplicación web	51
6.8.	Página principal del proyecto	58
6.9.	Página y formulario de registro	59
6.10.	Página y formulario de login	59
6.11.	Edición de perfil	60
6.12.	Formulario de creación de un NFT	60

6.13. Características de un NFT	61
6.14. Pantalla de puja por un NFT	61

Índice de tablas

2.1.	Comparativa de diferentes aspectos de DES, tripe DES y AES.	10
2.2.	Tiempo medio requerido para que un ataque de fuerza bruta tenga éxito. [20]	11
2.3.	Aplicaciones para el cifrado de clave pública y sus algoritmos.	15
6.1.	Coste hardware.	57
6.2.	Coste hardware.	57
7.1.	Competencias adquiridas y justificación.	64

Índice de listados

5.1.	Clase principal de un proyecto Spring Boot	41
5.2.	Creación de un repositorio en Java Spring Boot	42
5.3.	Ejemplo de uso de @Autowired en Java Spring Boot	42
5.4.	Respuesta de error 400, login fallido del servidor	43
5.5.	Respuesta de login correcto por parte del servidor	43
6.1.	Dockerfile api-gateway node	51
6.2.	Dockerfile event-listener	51
6.3.	Dockerfile transaction manager	52
6.4.	Dockerfile Backend	52
6.5.	Archivo de configuración de la Blockchain	52
6.6.	Docker-compose que despliega la aplicación completa	53

CAPÍTULO 1

Introducción

La tecnología Blockchain^[21] se dio a conocer en 2008, momento en el cual Bitcoin usó esta tecnología. Desde entonces, su popularidad se puede decir que ha crecido de forma exponencial. Siempre que se escucha la palabra Blockchain, se suele relacionar con el término criptomonedas, pero lo cierto, es que existen otros intereses que despiertan esta tecnología que pueden ser una solución eficiente y robusta en otros campos o ámbitos.

Muchas empresas, en los últimos años, han ido acercándose a este tipo de tecnología y gran parte de ellas han optado por incorporarla gradualmente a sus negocios. Es por ello, que, debido a sus características, se le ha calificado en numerosas ocasiones como la *Tecnología del futuro*.

Se ha realizado un Marketplace de NFTs, en el que los usuarios pueden comprar NFTs, pujar por ellos, regalarlos, ver su colección, así como toda la información relevante del activo que poseen. El administrador del Marketplace será el encargado de la creación del NFT y de la quema del token asociado si así se requiere. Con el uso de la tecnología Blockchain junto con el concepto de NFTs se pretende demostrar la autoría del activo comprado de manera segura, utilizando para ello el estándar ERC-721 de la red **Ethereum** y **Hyperledger Fabric** utilizado para crear redes Blockchain privadas.

En este documento, se pretende dejar constancia del TFG realizado, que aborda una de esas utilizadas de la tecnología Blockchain que no son tan conocidas, además se espera que sirva de apoyo para comprender una mejor manera como funciona esta tecnología.

1.1. ANTECEDENTES HISTÓRICOS

La tecnología Blockchain se populariza en 2008 como se ha mencionado anteriormente, sin embargo, nos tenemos que remontar a 1991 si queremos encontrar la primera referencia¹ a esta tecnología. Se describía entonces un sistema de registro digital de archivos de diferentes tipos como vídeo, imagen, audio o texto que proporcionara información sobre su autor y fecha de creación.

Casi 20 años después, en 2008 surgió **Bitcoin**^[14] en una famosa nota técnica que se publicaba bajo el pseudónimo de Satoshi Nakamoto y que utilizaba Blockchain como plataforma sobre la que se asentaba. Al estar tan ligados estos dos términos, a menudo se suele confundir Bitcoin con Blockchain, creyendo que son lo mismo, lo que es un error, ya que Bitcoin es una de las muchas aplicaciones de Blockchain.

La primera generación de la revolución digital nos ofreció el internet de la información. La segunda generación —impulsada por la tecnología de cadena de bloques— nos trae ahora el internet del valor: una nueva plataforma destinada a reconfigurar el mundo empresarial y a transformar, para mejor, el viejo orden de los negocios.

(Don Tapscott, director ejecutivo del Tapscott Group y exitoso autor de Wikinomics, The Digital Economy y Blockchain Revolution)

¹S. Habber, W.S. Stornetta – How to time-stamp a digital document, 1991.

Bitcoin se basaba en el uso de la cadena de bloques para registrar transacciones entre usuarios en una red P2P. Una de las características de Blockchain es que las transacciones son públicas, por lo tanto, podemos definirla como un *libro contable público descentralizado diseñado para registrar las transacciones en un entorno protegido*[9]. Su funcionamiento es semejante al de una base de datos, en la que se registran todas las transacciones que ocurren en esa red, y que es “copiada” en todos los nodos (generalmente computadores) que conforman la red.

1.2. REDES DESCENTRALIZADAS Y P2P

La descentralización es uno de los diferentes enfoques que sirven para gestionar una red. A modo de explicación nos centraremos en qué consisten, donde convienen y se pueden utilizar así como ventajas y desventajas de este tipo de arquitectura para después centrarnos en un tipo en concreto de redes descentralizadas como es P2P.

1.2.1. Redes descentralizadas

Las redes descentralizadas[7] son aquellas en las que diferentes y numerosos nodos van a actuar, debido a la ausencia de un servidor central, es decir, todas las conexiones no van a pasar de forma obligatoria por un ordenador central, conectándose en su lugar todos los nodos entre sí.

Al ser descentralizadas, la carga de trabajo se va a dividir entre estos nodos, algo a lo que llamamos **procesamiento distribuido** en el que la información va pasando de nodo en nodo hasta llegar a su destino final. Otra característica interesante de estas redes, es que cada nodo tiene control de forma independiente y por lo tanto, puede establecer sus **propias reglas**. Un aspecto muy importante es la capacidad de escalabilidad de estas redes, a los que se pueden ir sumando nodos sin que esto suponga un problema para la red. De igual forma, si uno de estos nodos falla, no compromete la disponibilidad o funcionamiento de la red entera, ya que cada uno de ellos es independiente, no como en una red centralizada, en la que si el servidor central falla, es atacado, o cae, toda la red se ve comprometida.

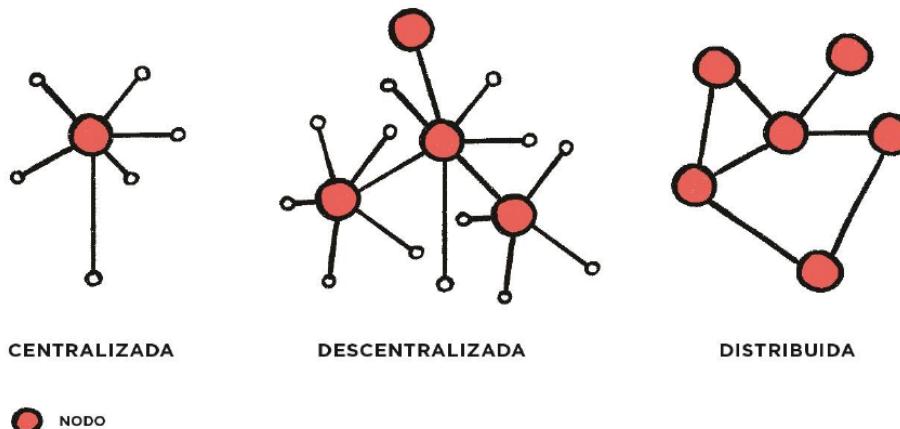


Figura 1.1: Diferencias entre redes centralizadas, descentralizadas y distribuidas [21]

Ventajas

A continuación, se destacan las principales ventajas de las redes descentralizadas:

- Ausencia de nodo central, lo que otorga disponibilidad y fiabilidad a la red.
- Tiene una mayor escalabilidad, ya que agregar nodos a la red es relativamente sencillo.
- Se goza de mayor privacidad, al no tener que pasar toda la información por un servidor principal. La información es más difícil de rastrear al pasar por diferentes puntos hasta llegar al destino.

Desventajas

Por otra parte, presentamos algunos inconvenientes:

- Si queremos un rendimiento aceptable, vamos a tener que conectar un mayor número de dispositivos, que se traduce en mayores costes de mantenimiento o sobrecarga de los recursos.
- Son más complejas de desarrollar o construir que una red cliente-servidor, ya que todos los nodos tienen que estar actualizados y correctamente configurados para establecer la conexión de forma correcta.
- En cuanto a consumo de energía, suelen ser menos eficientes y también, al tener que pasar la información por varios nodos, se va a tener mayor latencia que en otras estructuras de redes.

1.2.2. Redes P2P

También conocidas como redes de pares en castellano, forman parte de las redes descentralizadas en las que no hay clientes ni servidores fijos ni definidos, todos los nodos se adquieren el rol requerido por igual (pueden actuar como servidores o clientes). Este tipo de redes permiten el intercambio directo de información entre los nodos que la conforman.

Estas redes suelen tener un gran número de usuarios conectados mutuamente, y cada uno tiene la intención de acceder o requerir la información que tiene alojado en sí el nodo que la almacena. Es decir, en el momento en el que un nodo se conecta a la red P2P gana acceso a lo que tiene los demás nodos, y, los demás nodos, tienen acceso igualmente a lo que almacena el nodo que se ha unido.

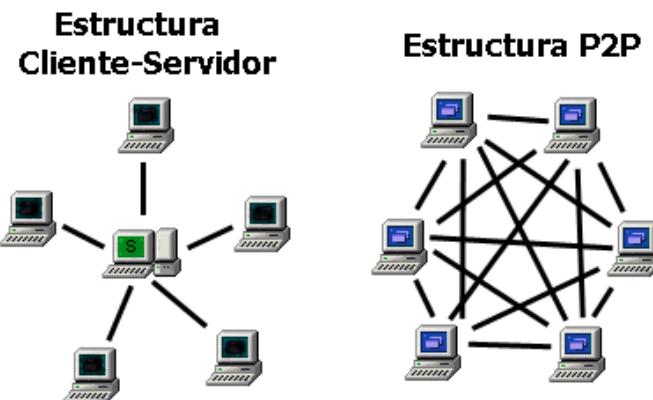


Figura 1.2: Conexión entre usuarios: cliente-servidor y redes P2P [8]

Su aplicación a las redes Blockchain puede entenderse con el siguiente ejemplo: imaginemos que un nodo quiere que el resto de nodos se enteren de una transacción, es decir, quiere comunicarla. Ese nodo enviaría la información a los nodos con los que tiene conexión directa, y estos a su vez, con los que tienen ellos. La información se va replicando poco a poco por toda la red.

Esto ocurrirá así en caso de que la transacción sea válida, si un usuario por ejemplo, crea una transacción con dinero falso, la propia red Blockchain con sus mecanismos de seguridad y validación, rechazará esta transacción, y cuando los nodos la “escuchen” simplemente la ignorarán.

1.3. PROYECTO BITCOIN

Bitcoin apareció en 2008, con la mencionada nota bajo el pseudónimo de Satoshi Nakamoto. En aquel momento, se presentó como un sistema alternativo de pago, pero, la realidad hoy en día, es muy diferente, ya que es la **criptomoneda más popular y exitosa del mundo**, llegando en noviembre de 2021 a costar 66.000 dólares o lo que es lo mismo, 58.300 euros. Como ya sabemos, bitcoin utiliza como plataforma base la tecnología Blockchain. En 2009, se publicó el primer cliente Bitcoin, que era

de código abierto y ello supuso el principio de lo que conocemos hoy en día, junto con una base de datos que se conoce con el nombre de *ledger* (libro de registros en español).

La Blockchain es semi-anónima, lo que quiere decir que en Bitcoin, cada transacción también lo es, pues los usuarios se **identifican con claves públicas** (ver sección [2.1.3]) que actúan como pseudónimos sustituyendo a una identidad real. La cadena de bloques podríamos decir que es una “base de datos” a la que varias entidades tienen acceso ya que se comparte en redes P2P y en la que la información se almacena de forma inmutable y ordenada.

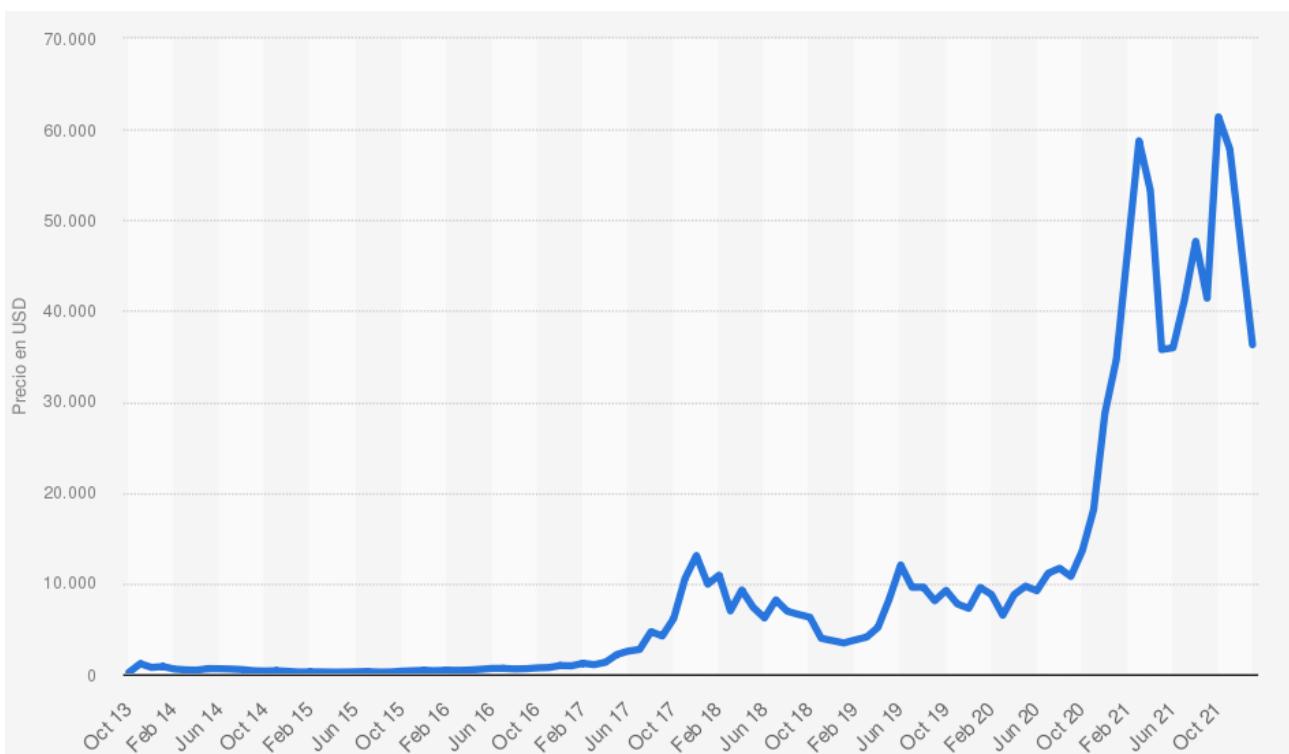


Figura 1.3: Precio de Bitcoin desde octubre de 2013 hasta enero de 2022, en dólares [17]

1.3.1. Transacciones en Bitcoin

Si tenemos que definir qué es una criptomoneda de forma clara y concisa lo podríamos hacer como: **una cadena de firmas digitales**. La moneda va pasando de propietario en propietario y para ello se firma un Hash (ver sección [2.1.2]) de forma digital de la transacción anterior junto con la clave pública del siguiente propietario. Estos dos elementos se añaden ambos al final de la moneda y el destinatario o nuevo dueño de la moneda puede verificar en cualquier momento las firmas para comprobar la cadena de propiedad.

Un interrogante que surgió es que el beneficiario no podía verificar que uno de los propietarios no hubiera gastado dos veces la misma moneda. Se necesita entonces una forma de que el destinatario sepa que los anteriores dueños de la moneda no han firmado transacciones previas. Para este objetivo, la transacción más temprana es la importante, y nos olvidamos de la de los posibles intentos de fraude que se produzcan de forma posterior. Una posible técnica para lograr esto es la de conocer absolutamente todas las transacciones que se han producido, y Bitcoin, para proporcionar este conocimiento anuncia todas las transacciones de forma pública. Entonces surge la necesidad de un sistema que ponga de acuerdo a los participantes en un historial único del orden en que esas transacciones se produjeron, el llamado *Servidor de sellado de tiempo*.

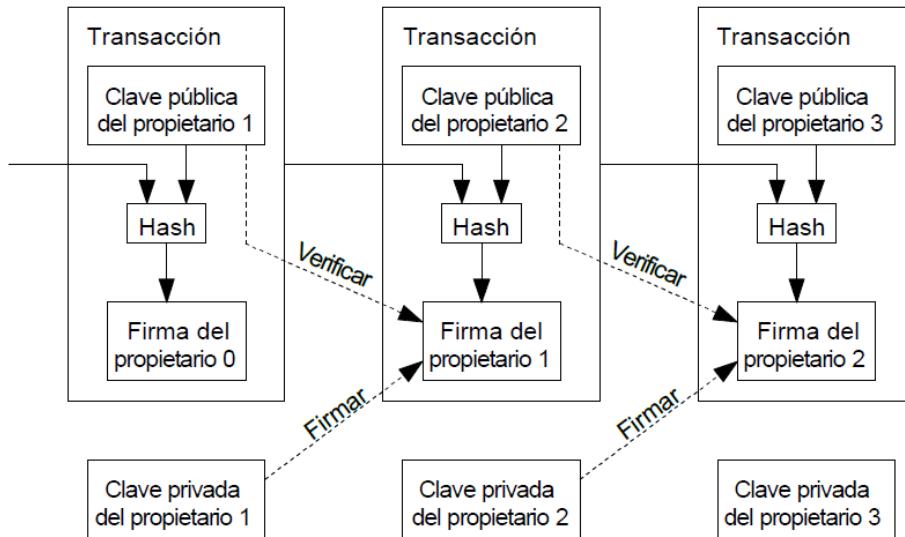


Figura 1.4: Funcionamiento de transacciones en Bitcoin [14]

1.3.2. Servidor de sellado de tiempo

Este servidor funciona cogiendo el Hash de un bloque de elementos para “sellarlo en el tiempo” y su Hash se emite públicamente, como si de un informativo se tratara o un comunicado oficial. Cada sellado de tiempo tiene el sellado de tiempo previo anterior en su Hash, de esta forma se crea una cadena a la que se van agregando nuevos Hash.

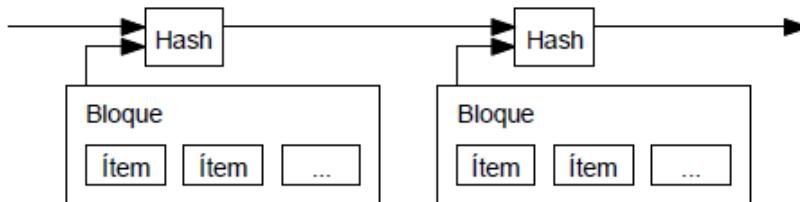


Figura 1.5: Ejemplo de la cadena de sellado de tiempo [14]

1.3.3. Proof-of-work

Para implementar el sellado de tiempo es requisito tener un sistema de *proof-of-work*. Este último consiste en buscar un valor que cuando fue *Hasheado*, al igual que con SHA-256, el Hash comienza con un número de certo bits.

Bitcoin implementa el proof-of-work incrementando un nonce² en el bloque hasta que se encuentra un valor que proporcione al has los cero bits que se necesitan.

La proof-of-work sirve también para resolver el interrogante de la representación en la toma de decisiones. Si utilizáramos un voto por IP, podríamos tener apuros ya que cualquiera podría controlarla por la facilidad de asignar muchas IPs. La proof-of-work es en realidad un **voto por CPU** en la que la decisión de la mayoría está interpretada por la cadena de longitud mayor.

Para que un atacante modificara un bloque de la blockchain, tendría que realizar esta proof-of-work de todos los bloques anteriores a él y alcanzar el trabajo de los nodos “verdaderos”. La probabilidad de que esto pase es ínfima, ya que disminuye conforme se van añadiendo bloques.

²El nonce o “número de un solo uso”, es simplemente un número. Un número aleatorio y de características únicas que tiene como finalidad ser usado en sistemas criptográficos.

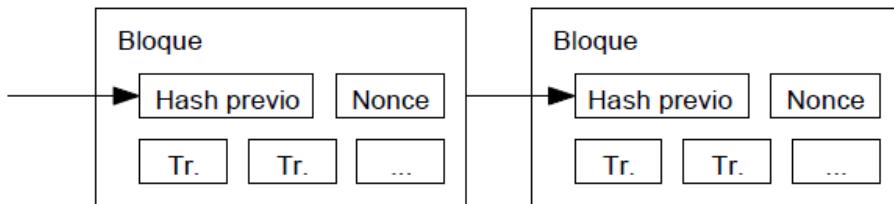


Figura 1.6: Funcionamiento de proof-of-work [14]

1.3.4. Red

Por último, hablaremos de como ejecutar la red de Bitcoin y sus pasos:

- (1) Las transacciones nuevas se transmiten a todos los nodos.
- (2) Cada nodo recoge todas las transacciones en un bloque.
- (3) Cada nodo trabaja en resolver una proof-of-work compleja para su bloque.
- (4) Cuando un nodo resuelve una proof-of-work, transmite el bloque a todos los nodos.
- (5) Los nodos aceptan el bloque si todas las transacciones en él son válidas y no se han gastado con anterioridad.
- (6) Los nodos expresan su aceptación del bloque al trabajar en crear el siguiente bloque en la cadena, usando el hash del bloque aceptado como hash previo.

La cadena más larga siempre es considerada la correcta por los nodos y estos trabajarán con el objetivo de extenderla. La transmisión de nuevas transacciones no precisa alcanzar todos los nodos. Con alcanzar a la mayoría de los nodos, entrarán en un bloque en poco tiempo. Las transmisiones de nodos también toleran mensajes perdidos. Si un nodo no recibe un bloque, lo reclamará cuando reciba el siguiente bloque y se dé cuenta de que falta uno.

1.4. IMPACTO SOCIO-ECONÓMICO

La tecnología y Bitcoin han traído un importante cambio en el pensamiento social. Hay gente que las considera el futuro, una inversión segura; por otra parte, tenemos otro bando que las considera una estafa y que no tiene valor real, más que el que marca la oferta y demanda.

Dejando opiniones subjetivas a un lado, lo cierto es que estos dos términos han revolucionado el mundo de los negocios y las finanzas, y cada vez más empresas apuestan por incorporar la tecnología Blockchain a su infraestructura y modelo de negocio. En el caso de Bitcoin, tenemos el reciente caso del El Salvador, que estableció esta criptomoneda como moneda nacional de pago y propuso el desarrollo de una ciudad financiada enteramente con Bitcoin, bajo el nombre de *Bitcoin City*³.

A partir de Bitcoin, surgieron más criptomonedas, las llamadas *Altcoins*⁴. Cada Altcoin tiene un propósito, al que pretende añadir valor, es decir, cuenta con un proyecto detrás que lo sostiene y lo hace atractivo para los inversores, algunos ejemplos son:

- | | |
|---------------|-----------------|
| ▪ Cardano [4] | ▪ Litecoin [12] |
| ▪ Solana [15] | ▪ XRP [19] |
| ▪ BNB [3] | ▪ Ethereum [10] |

Es tal el impacto que ha tenido, que desde 2021, los proveedores de servicios de criptomonedas tienen que constar en el Banco de España, es decir, tienen que registrarse en sus listado. Esta medida

³https://cincodias.elpais.com/cincodias/2022/05/10/mercados/1652184500_469588.html

⁴Palabra formada a partir de las palabras “alternativa” y “coin”. Es un término que se utiliza para referirse a las criptomonedas que no son Bitcoin.

tiene como fin evitar la evasión de impuestos, financiación de causas ilegales como terrorismo. No cumplir con este requerimiento conlleva multas económicas que pueden ascender hasta millones de euros.

La tecnología del libro contable descentralizado tiene el potencial de cambiar los servicios financieros de una manera muy profunda, similar al cambio producido por el internet en los medios de comunicación y en el entretenimiento.

(<https://r3.com/>)

Actualmente, hay más de 800 criptomonedas alternativas. Pero el impacto de la blockchain no queda solo a su aplicación a la banca, que es donde más influencia ha tenido las aseguradoras, telecomunicaciones, sector energético, salud, pymes, juegos en línea, industria 4.0, etc. Así como también otros sectores como la música, las Smart Cities, la participación ciudadana, entre otros. La Blockchain también se ha aplicado en los últimos años a la forma en la que se colecciona arte o se quiere demostrar la autoría de algún bien mediante los llamados NFT, motivación principal de este TFG.

1.5. MOTIVACIÓN

Como se ha mencionado en varios puntos de este capítulo de introducción, la tecnología Blockchain no solo sirve para crear criptomonedas, si no que también tiene otros usos. La creciente popularidad de la tecnología ha hecho que muchas empresas se interesen por incluirla, atraídos por su seguridad, rapidez y demás características.

Este TFG surge como una de esas muchas aplicaciones posibles, que consiste en la elaboración de un Marketplace, en la que las empresas interesadas pueden ofrecer sus productos (ya sea arte en forma de imagen, objetos físicos que se van a tokenizar, etc) en la que los usuarios finales podrán hacerse con la autoría del bien tokenizado, quedando constancia de ello en la cadena de bloques.

Esto supone una revolucionaria forma de gestionar bienes por parte de las empresas, un sistema al que se puede acceder desde cualquier parte del mundo, en la que cualquiera que lo desee puede comprar los bienes. De esta forma, el Marketplace se puede utilizar también como espacio de inversión utilizando la **compra-venta** de los NFTs entre los distintos usuarios de la plataforma.

1.6. ESTRUCTURA DEL DOCUMENTO

El presente documento se ha redactado siguiendo la guía de estilo y el formato requerido por la ESI de Ciudad Real, en la que se incluyen los siguientes capítulos:

- **Capítulo 1. Introducción:** donde se explica el contexto del proyecto y la motivación que lleva a desarrollarlo
- **Capítulo 2. Antecedentes:** en este capítulo se explican conceptos clave para la correcta comprensión del proyecto y en qué consiste.
- **Capítulo 3. Objetivo:** se describe el objetivo general del proyecto y los objetivos específicos que surgen a partir de este.
- **Capítulo 4. Metodología:** en ella se detalla la metodología de trabajo que se ha seguido así como su justificación.
- **Capítulo 5. Arquitectura:** en dicho capítulo se explica la arquitectura del proyecto y se explican en detalle cada uno de los módulos que lo componen.
- **Capítulo 6. Resultados:** se analizan los resultados obtenidos tras la realización del trabajo.
- **Capítulo 7. Conclusiones:** apartado donde se realizará un juicio crítico y discusión del proyecto, y se detallan los trabajos futuros.

CAPÍTULO 2

Antecedentes

La tecnología Blockchain o cadenas de bloques es una tecnología relativamente nueva, que ha experimentado crecimiento en popularidad y uso en los últimos años. Es por ello que es necesario poner en contexto a los lectores de este TFG que puedan estar más desinformados y hacer más fácil la compresión del proyecto. Es por ello que, a continuación, se explicarán algunos conceptos clave a tener en cuenta.

2.1. CRIPTOGRAFÍA

Si hablamos de Blockchain, tenemos que tener en cuenta unas nociones básicas de criptografía, ya que esta juega un papel muy importante en el funcionamiento de estas redes. Para entender mejor este concepto, explicaremos algunos conceptos que deben tenerse en cuenta, como los factores por los que se clasifican los algoritmos de cifrado:

- **El tipo de operación empleada para transformar el texto plano en texto cifrado.** Todos los algoritmos están basados en dos principios fundamentales **sustitución**, etapa en la que cada elemento del texto plano, como puede ser un bit, una letra, un grupo de letras o bytes es sustituido por otro elemento diferente, y **transposición**, que consiste en reordenar o alterar el orden del texto.
- **El número de claves usadas.** Cuando el emisor y el receptor usan la misma clave, estamos hablando de criptografía **simétrica** también conocida como de **clave única o de clave secreta**. Por otra parte, si el emisor y el receptor utilizan cada uno un par de claves diferentes, estamos ante criptografía **asimétrica o de clave pública**.
- **La forma de procesar el texto plano.** El **cifrador de bloque** procesa un bloque de elementos cada vez, de esta forma se produce un bloque cifrado por cada bloque de texto plano entrante. Un **cifrador de flujo** procesa los elementos de forma continua, uno a uno, por cada elemento entrante se produce un elemento de salida.

2.1.1. Criptografía simétrica

El cifrado simétrico también llamado de clave secreta o de clave única, es una técnica utilizada para proporcionar **confidencialidad** a los datos que se quieren almacenar o enviar. El texto se cifra y se descifra con la **misma clave**. Los elementos que forman el cifrado simétrico son:

- Texto plano
- Algoritmo de cifrado
- Clave secreta, las sustituciones y transformaciones realizadas por el algoritmo dependen de ella.
- Texto cifrado
- Algoritmo de descifrado

Hay dos principios importantes para el uso de cifrado simétrico:

- (1) **Se necesita un algoritmo de cifrado robusto** ya que el atacante no debería poder descifrar la clave o el texto, aunque el atacante estuviera en posesión de una serie de texto cifrados junto con sus textos originales.
- (2) El emisor y el receptor deben de haber obtenido **copias de la clave privada de forma segura**.

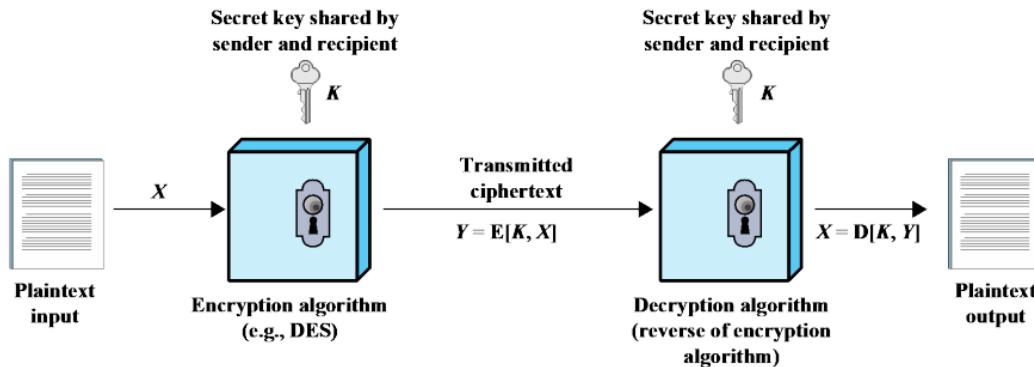


Figura 2.1: Esquema de cifrado con criptografía simétrica [20]

En cuanto a los algoritmos de cifrado simétrico podemos destacar tres, sobre los que no entraremos en mucho detalle en este informe. Es de importancia recordar que los algoritmos de cifrado simétrico están expuestos a dos tipos de ataques:

Tabla 2.1: Comparativa de diferentes aspectos de DES, triple DES y AES.

	DES	Triple DES	AES
Tamaño del texto plano (bits)	64	64	128
Tamaño del texto cifrado (bits)	64	64	128
Tamaño de clave (bits)	56	112 o 168	128, 192 o 256

- **Criptoanálisis.** Es el proceso por el cual se intenta descubrir una clave privada o texto plano basándonos en la naturaleza del cifrado, alguna información que tenemos de otro textos cifrados o planos.

Los algoritmos DES (creado en los años 70) y AES (creado en los años 90) no han podido romperse siguiendo esta estrategia. Si este ataque hubiera tenido éxito, todo lo que se ha cifrado anterior a ese punto, y todo lo cifrado posteriormente, estaría en riesgo de ser descifrado.

- **Fuerza bruta.** Este ataque consiste en probar un número muy elevado de claves contra un texto relativamente corto hasta obtener un texto legible o plano. Si hacemos caso a estudios estadísticos, haría falta probar al menos la mitad de todas las combinaciones para que el ataque prospere.

Cada uno de los algoritmos (DES y AES), puede trabajar en modos¹ diferentes como son:

- | | |
|-------------------|-----------------|
| (1) ECB (bloques) | (3) CFB (flujo) |
| (2) CBC (bloques) | (4) CTR (flujo) |

2.1.2. Autenticación de mensajes y funciones Hash

El cifrado simétrico protege ante ataques pasivos, pero no contra la falsificación de datos y transacciones, lo que conocemos como **ataques activos**. Para cubrir esta importante necesidad, surge la **autenticación de mensajes**.

¹Para más información sobre los modos de operación de los cifradores de bloques y flujo consultar el libro [20]

Tabla 2.2: Tiempo medio requerido para que un ataque de fuerza bruta tenga éxito. [20]

Tamaño de clave (bits)	Cifrado	Numero de claves posibles	Tiempo requerido para 10^9 intentos	Tiempo requerido para 10^{13} intentos
56	DES	$2^{56} \approx 7,2 \times 10^{16}$	$2^{55} ns = 1,125$ años	1 hora
128	AES	$2^{128} \approx 3,4 \times 10^{38}$	$2^{127} ns = 5,3 \times 10^{21}$ años	$5,3 \times 10^{17}$ años
168	Triple DES	$2^{168} \approx 3,7 \times 10^{50}$	$2^{167} ns = 5,8 \times 10^{21}$ años	$5,8 \times 10^{29}$ años
192	AES	$2^{192} \approx 6,3 \times 10^{57}$	$2^{191} ns = 9,8 \times 10^{21}$ años	$9,8 \times 10^{36}$ años
256	AES	$2^{256} \approx 1,2 \times 10^{77}$	$2^{255} ns = 1,8 \times 10^{21}$ años	$1,8 \times 10^{56}$ años

Cuando estamos ante un archivo, documento o mensaje podemos decir que es auténtico cuando es **genuino** y su remitente es la fuente original o deseada. La autenticación de mensajes es un procedimiento mediante el cual podemos saber y verificar que el mensaje recibido durante una comunicación es auténtico. Algunos aspectos a tener en cuenta en la verificación de mensajes son:

- (1) Comprobar que el contenido **no ha sido modificado**
- (2) Comprobar que el origen es **auténtico**
- (3) **Verificar la puntualidad** del mensaje
- (4) CTR (flujo)

La autenticación de mensajes se puede realizar con el cifrado convencional, solo si el emisor y el receptor conocen y comparten una clave secreta ya que solo el auténtico emisor sería capaz de cifrar ese mensaje de forma correcta para que el receptor pudiera hacerlo legible. De forma adicional, para reforzar la seguridad, se puede incluir un **código de detección de errores** y un **número de secuencia**. De esta forma, el receptor estaría seguro de que ese mensaje no ha sido alterado y la secuencia es adecuada.

Autenticación de mensajes sin cifrado

Hay tres situaciones en las que es conveniente la autenticación de mensajes sin cifrar:

- Hay un gran número de aplicaciones en las que el mismo mensaje se emite a muchos destinatarios.
- Una parte de la comunicación tiene una gran carga y no tiene capacidad para estar descifrando un número elevado de mensajes.
- La autenticación de un programa informático en texto plano es un servicio atractivo.

Autenticación de mensajes utilizando un mensaje

Implica el uso de una **clave secreta** para generar un bloque de datos pequeño, conocido como **código de autenticación del mensaje**, que se añade al final del propio mensaje. Cuando las dos partes se comunican, supongamos A y B , comparten entre ellas una clave secreta común, K_{AB} . De esta forma, cuando A quiere enviar un mensaje a B , calcula el código de autenticación como una función del mensaje y la clave.

Destacar que en la Figura 2.2, MAC no es una dirección física, es el código de autenticación de mensajes.

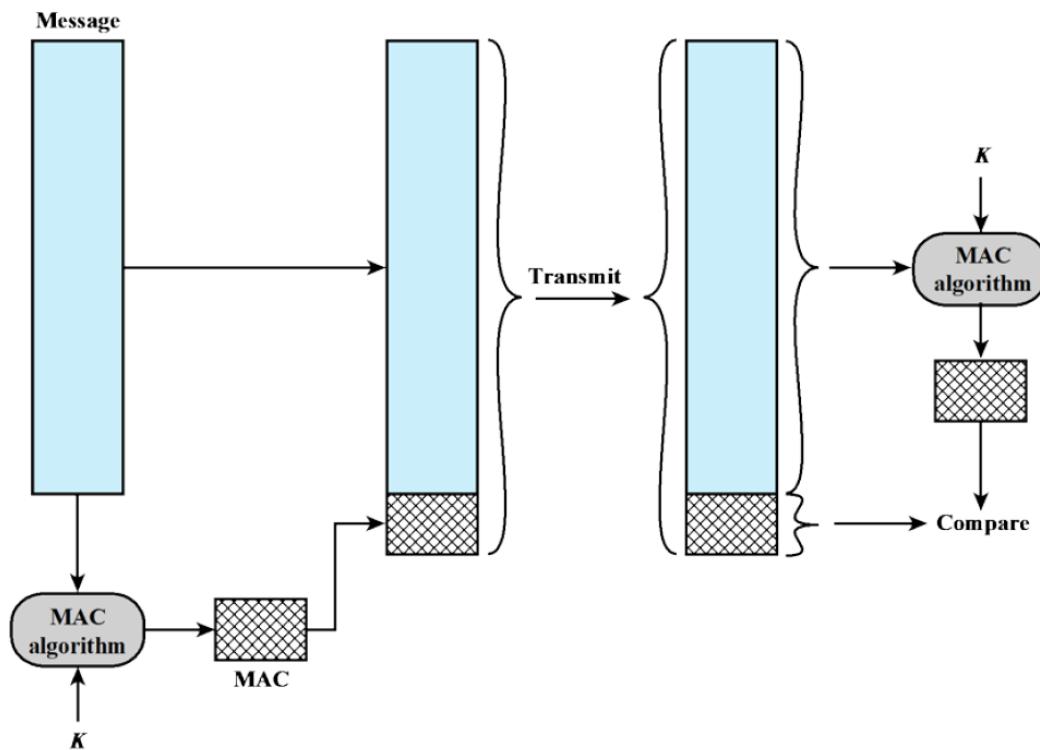


Figura 2.2: Autenticación de mensajes usando un código de autenticación de mensaje [20]

Función Hash

Al igual que en el enfoque anterior, una función Hash acepta un mensaje de longitud variable M como entrada (al que se le puede añadir *padding*²) y como resultado o salida produce lo que llamamos un resumen del mensaje de tamaño fijo $H(M)$. A diferencia de MAC, una función Hash no acepta una clave secreta como entrada, por lo que eliminamos la parte que hacía inseguro al enfoque comentado anteriormente.

La finalidad de una función Hash es la de crear una **huella de archivo**, mensaje u otro tipo de datos. Para que esto sirva como autenticación de mensajes, una función Hash debe tener las siguientes características:

- (1) H puede aplicarse a un bloque de datos de cualquier tamaño.
- (2) H produce una salida de tamaño fijo.
- (3) $H(x)$ es relativamente fácil de computar para cualquier x dado, haciendo que tanto las implementaciones de hardware como de software sean prácticas.
- (4) Para cualquier valor h dado, es imposible desde el punto de vista computacional encontrar un x tal que $H(x) = h$, lo cual, con frecuencia, se conoce como **propiedad unidireccional**.
- (5) Para cualquier bloque dado x , es imposible desde el punto de vista computacional, encontrar un $y \neq x$ con $H(y) = H(x)$, lo que se conoce como **resistencia débil a la colisión**.
- (6) Es imposible desde el punto de vista computacional encontrar un par (x, y) tal que $H(x) = H(y)$, lo que normalmente se conoce como **resistencia fuerte a la colisión**.

A partir de la función Hash surgió **SHA** desarrollado por el **NIST** y fue publicado en 1992 como un estándar **FIPS PUB 180**, más tarde, en 1995 surgió otra versión conocida como SHA-1 publicada

²El padding o relleno es una técnica que consiste en añadir datos al principio, medio o final de un mensaje antes de ser cifrado, con el fin de cubrir un tamaño mínimo de mensaje y para ocultar que varios mensajes terminan de manera predecible o parecida, y así evitar que se descifre información.

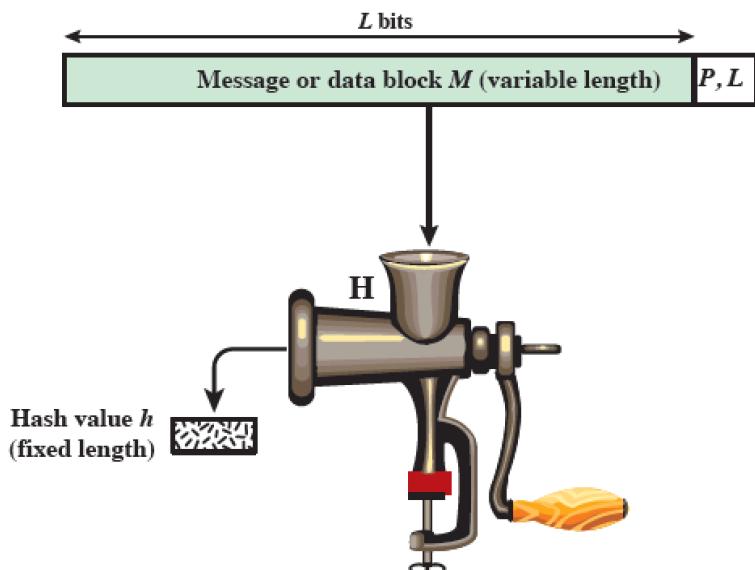


Figura 2.3: Función Hash; $h = H(M)$; $P, L = \text{padding más longitud}$ [20]

en FIPS PUB 180-1. El algoritmo tiene como entrada un mensaje de longitud máxima 2^{64} bits y produce como salida un resumen de mensaje de 160 bits.

Hasta que el ataque de fuerza bruta tuvo popularidad y los atacantes empezaron a interesarse por él, el algoritmo **MD5** fue el más usado. Este algoritmo tenía como entrada un mensaje de longitud indefinida y producía un resumen de mensaje de 128 bits.

2.1.3. Cifrado de clave pública

El cifrado de clave pública, propuesto por primera vez por **Diffie y Hellman en 1976**, es el primer avance realmente revolucionario en el cifrado de miles de años. El motivo es que los algoritmos de clave pública están **basados en funciones matemáticas** y no en simples operaciones sobre los patrones de bits. Además, la criptografía de clave pública es asimétrica, lo que implica el uso de **dos claves separadas**: pública y privada. Algunas cuestiones comunes relativas al cifrado de clave pública:

- La primera es la creencia de que el cifrado de clave pública es más seguro ante el criptoanálisis que el cifrado convencional. **No hay nada que haga uno superior al otro en lo que respecta al criptoanálisis.**
- Otra equivocación la hallamos en la idea de que **el algoritmo de cifrado asimétrico ha dejado desfasado al cifrado convencional**. Por el contrario, debido al coste computacional de los esquemas actualices de cifrado de clave pública, no parece que el cifrado convencional vaya a abandonarse.
- Por último, se piensa que la distribución de claves no es importante cuando se usa el cifrado de clave pública.

Un ejemplo de utilización de cifrado de clave sería el siguiente:

- (1) Cada usuario genera una pareja de claves para el cifrado y descifrado de mensajes.
- (2) Cada usuario localiza una de las dos claves en un registro público u otro archivo accesible. Esta es la clave pública.
- (3) Si Bob quiere enviar un mensaje privado a Alice, cifra el mensaje usando la clave pública de Alice.
- (4) Cuando Alicia recibe el mensaje, lo descifra con su clave privada. Ningún otro receptor puede descifrar el mensaje porque sólo Alice conoce su clave privada.

A continuación, se dejan unos diagramas que indican el funcionamiento de cifrado utilizando clave pública (Figura 2.4) y clave privada (Figura 2.5). El cifrar con clave pública otorga **confidencialidad** al mensaje ya que solo se puede descifrar con la clave privada correspondiente. El cifrar con clave privada otorga **autoría** pero no confidencialidad, ya que todo el mundo puede descifrar el mensaje con la clave pública adecuada.

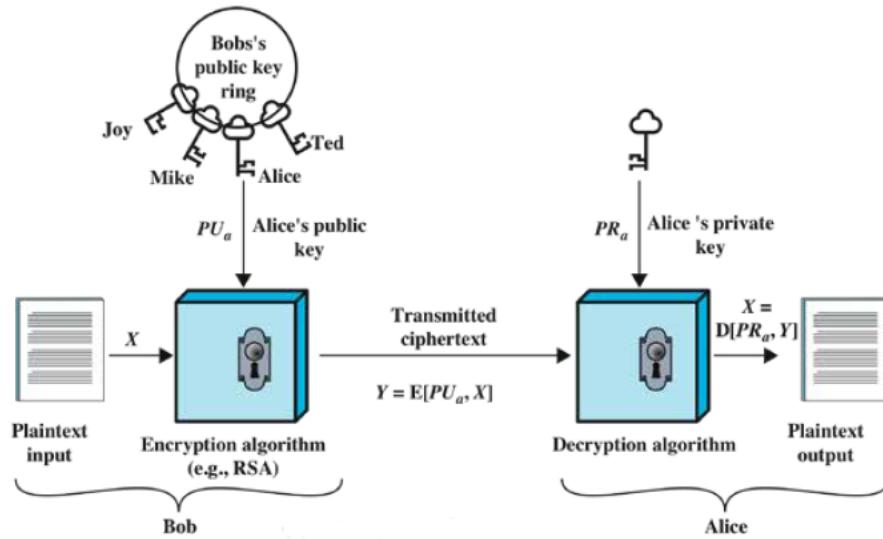


Figura 2.4: Cifrado con clave pública [20]

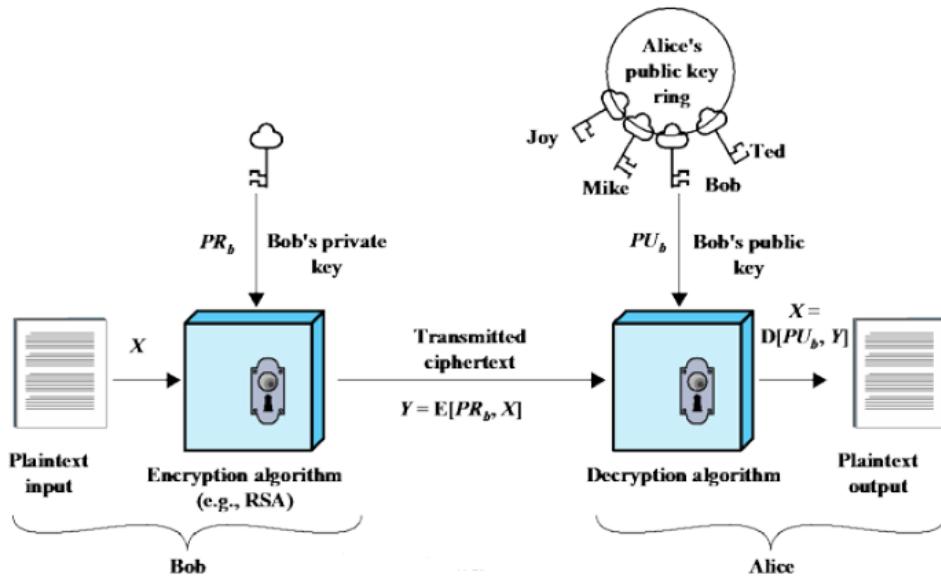


Figura 2.5: Cifrado con clave privada [20]

Dependiendo de la aplicación, el emisor usa su clave privada o la clave pública del receptor, o las dos. En términos generales, podemos clasificar el uso de cifrado de clave pública en tres categorías, que se muestran en la Tabla 2.3:

Tabla 2.3: Aplicaciones para el cifrado de clave pública y sus algoritmos.

Algoritmo \ Uso	Firma digital	Distribución de clave simétricas	Encriptación de claves secretas
RSA	Sí	Sí	Sí
Diffie-Hellman	No	Sí	No
DSS	Sí	No	No
Elliptic Curve	Sí	Sí	Sí

2.1.4. Firmas digitales

El **NIST FIPS PUB 186-4** define las firmas digitales como un patrón de bits generados por un agente a partir de un bloque de datos, de forma que otro agente puede acceder a la firma para garantizar que el bloque de datos no se ha modificado y que proviene de la fuente que dice provenir. Este mismo estándar especifica que se pueden utilizar 3 algoritmos para firmas digitales:

- DSA
- RSA
- ECDSA

Pongamos un ejemplo (ver Fig. 2.6): supongamos que Bob quiere enviar un mensaje a Alice, y aunque no es necesario que el mensaje se mantenga en secreto, quiere que Alice se asegure de que el mensaje proviene de él. En este caso, Bob usa su propia clave privada para cifrar el mensaje. Cuando Alice recibe el texto cifrado, se encuentra con que puede descifrarlo con la clave pública de Benito, demostrando así que, el mensaje ha debido ser cifrado por él. Nadie más tiene la clave privada de Benito, y, por tanto, nadie más ha podido crear un texto cifrado que pueda ser descifrado con su clave pública.

Además, es imposible alterar el mensaje sin acceso a la clave privada de Bob, así que el mensaje queda autenticado tanto en lo que respecta a la fuente como a la **integridad** de los datos.

Certificados de clave pública

La base del cifrado de clave pública se encuentra en el hecho de que la clave pública es pública. Aunque el enfoque es conveniente, presenta una **debilidad fundamental**: cualquier persona puede falsificar esa clave pública. Es decir, un usuario podría hacerse pasar por el usuario *A* y enviar una clave pública a otro participante o difundirla.

La solución a este problema es el certificado de clave pública. Básicamente, un certificado consiste en una clave pública y un identificador o nombre de usuario del dueño de la clave, con todo el bloque firmado por una tercera **parte confiable: una CA**. De esta forma, un usuario puede presentar su clave pública a la autoridad de forma segura, obtener un certificado y luego publicarlo.

Cualquier persona que necesite la clave pública de este usuario, puede obtener el certificado y verificar que es válida por medio de la firma fiable adjunta. Ejemplo y pasos de cómo funcionaría en un caso real (ver Fig. 2.7):

- (1) El software de usuario (navegador) genera un par de claves. La descarga por tanto, la tenemos que hacer desde el mismo ordenador y el mismo navegador.
- (2) El cliente prepara un certificado que incluye la clave del usuario.
- (3) El usuario proporciona el certificado sin firmar a la CA, de manera segura. Esto puede requerir una reunión cara a cara.
- (4) La CA crea una firma de la siguiente forma: utiliza una función Hash para calcular el código del certificado sin firmar, y genera una firma creando la clave privada de la propia CA y un algoritmo de generación de firmas.

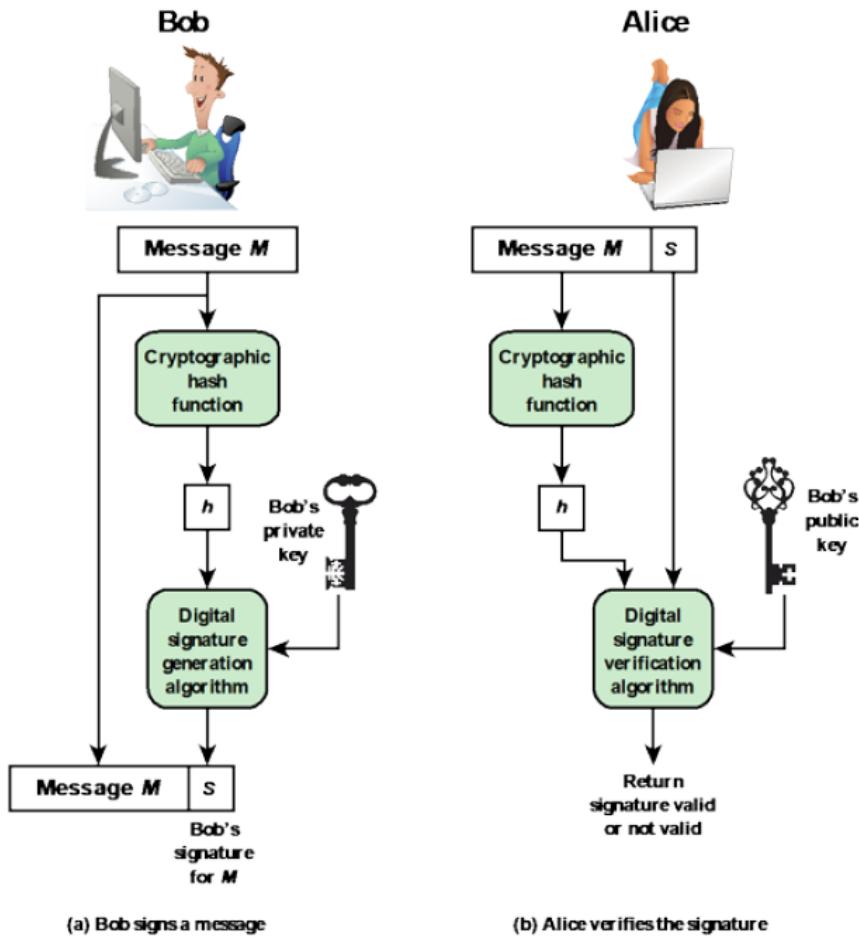


Figura 2.6: Ejemplo simplificado de flujo esencial para el proceso de firma digital [20]

- (5) Una vez que la CA hace esto, lo adjunta a todo lo anterior y genera el certificado firmado. Lo descargamos desde el mismo navegador que lo pedimos.

2.2. BLOCKCHAIN

En el anterior capítulo hemos hablado de las tecnologías sobre las que se apoyaba la Blockchain a modo de introducción, ahora, se busca profundizar más en su funcionamiento y los tipos de red que existen.

2.2.1. Funcionamiento

La Blockchain se asemeja a libro de contabilidad, en el que consta el saldo que tiene cada titular de cada cuenta. Es una secuencia de bloques los cuales tienen datos almacenados dentro, es una base de datos inmensa. Estos bloques están conectados unos a otros de manera secuencial, y, por lo tanto, un bloque depende del anterior, y así sucesivamente (al primer bloque de todos se le conoce como *Genesis Block*). La información que se hereda se encuentra en la **cabecera de bloque**, y el valor que se genera a partir de la cabecera se llama Hash, que como ya hemos visto, es único e irrepetible. Solo los bloques que sean capaces de relacionar su información con la del antecesor serán añadidos, lo que se conoce como **bloques honestos**.

La cabecera de bloque contiene la siguiente información:

- **Versión de bloque.** Se trata de un código que mezcla caracteres alfabéticos y numéricos en la que se indican las reglas a seguir para validar el bloque.

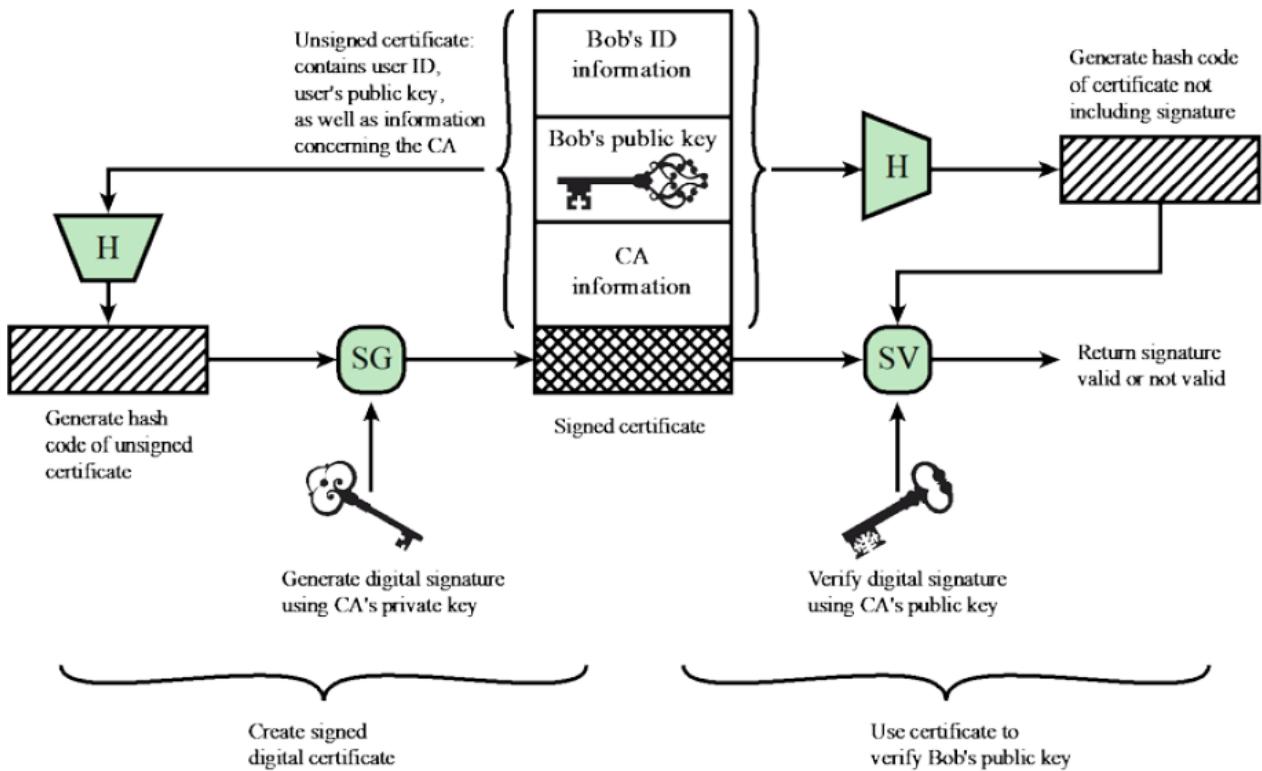


Figura 2.7: Ejemplo de uso de certificado de clave pública [20]

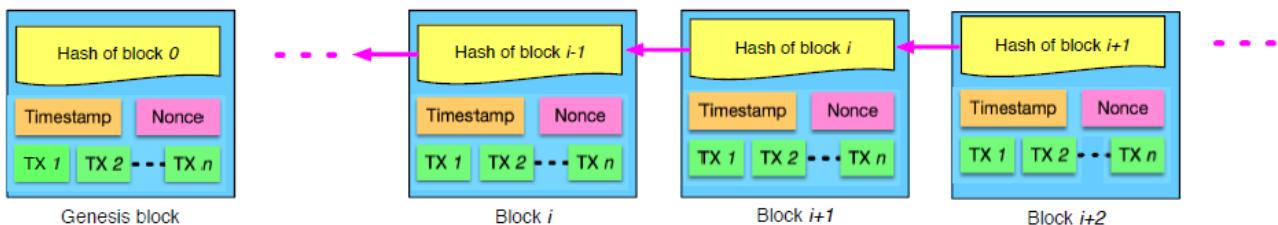


Figura 2.8: Ejemplo de Blockchain y sus bloques sucesivamente unidos [22]

- **Hash del árbol de Merkle.** Hace referencia a todas las transacciones cabidas en el bloque.
- **Time stamp.** Hace referencia a cuando se encontró el Hash.
- **nBits.** Valor del hash de forma compacta.
- **Nonce.** Valor numérico que se aumenta a medida que los bloques se van sumando
- **Bloque padre.** Permite la unión de todos los bloques a este. Se recogerán en él de forma compacta los 256 valores del Hash del bloque anterior.

En cuanto a las características más importantes de Blockchain son:

- **Descentralización.** Al estar basada en redes P2P (ver Sec. 1.2)
- **Inmutabilidad.** La probabilidad de que un atacante modifique un bloque es matemáticamente muy pequeña, esta probabilidad disminuye de forma exponencial a medida que se añaden bloques.
- **Anonimidad.** Todos los usuarios se identifican con claves públicas (ver Sec. 2.1.3)
- **Verifiable.** Todas las transacciones están verificadas por medio de la marca de tiempo o *time stamp* que indica cuando se produjo la validación y quienes eran los usuarios participantes. Después estas validaciones quedan registradas en el Árbol de Merkle y relacionadas entre sí.

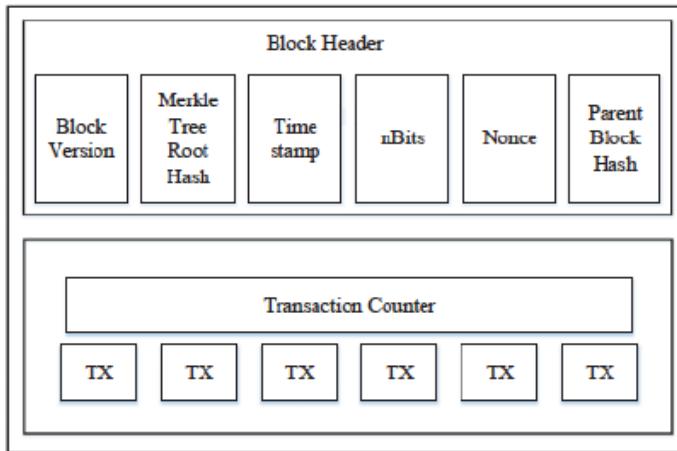


Figura 2.9: Ejemplo de bloque de Blockchain: cabecera y body. [22]

2.2.2. Tipos de Blockchain

En función de quien pueda formar parte de ellas y participar, las redes se clasifican en públicas y privadas. También existe un tercer tipo llamado “híbrido” que es una combinación de ambas.

Públicas

Se caracterizan porque cualquier usuario puede formar parte de ellas en el momento que desee, es decir, el usuario crea un nodo que añade y verifica los bloques de la red. Las redes públicas suelen utilizar lo que se conoce como Proof-of-Work, como el caso de Bitcoin comentado en el capítulo anterior (ver Sec. 1.3) que consiste en resolver una o varias operaciones matemáticas y que supone un gasto considerable de energía y consumo y hace que en la práctica no cualquier usuario pueda añadir bloques a la red. El Proof-of-Work es necesario ya que si no, un usuario con intenciones delictivas o negativas podría añadir bloques para lucrarse o sacar algún tipo de beneficio. Estas medidas pueden variar dependiendo del **algoritmo de consenso**³. Algunos ejemplos de redes blockchain públicas son **Bitcoin**[14] y **Ethereum**[10].

Privadas

A estas redes solo pueden añadir y verificar nuevos bloques los usuarios que han sido previamente autorizados por la entidad que la red y que hace de **autoridad** que puede ser centralizada o no. Cabe destacar que los usuarios no autorizados pueden tener vetado el acceso incluso a la lectura.

Puede que estas redes usen también un algoritmo de consenso, pero no es necesario que tengan un coste computacional tan elevado como en las redes públicas, ya que se da por hecho que si están dentro de la red es porque han sido autorizados previamente, y hay confianza total o parcial en ellos. Estas redes al no tener un algoritmo de consenso tan “fuerte” gozan de mayor rapidez en las transacciones y menor consumo de recursos. Las redes privadas se suelen usar en dos casos:

- Una empresa u organización quiere tener control total sobre la red Blockchain que va a usar así como de los datos que se van a manejar en ella, y restringir el acceso a ellos. Implica que los usuarios que la usen confíen plenamente en la organización
- Varias organizaciones quieren colaborar pero no confían del todo unas en otras y la red proporciona esa confianza

Un ejemplo de esta red sería Corda[5]

³Es un mecanismo que permite a los usuarios o máquinas coordinarse en un entorno distribuido. Debe garantizar que todos los agentes del sistema puedan ponerse de acuerdo respecto a una fuente única de verdad, incluso en el caso de que algunos de ellos fallen. <https://academy.binance.com/es/articles/what-is-a-blockchain-consensus-algorithm>

Híbridas

Las redes híbridas son aquellas que tienen características de las dos anteriores. Este tipo de redes no están controladas ni por sus participantes ni por una entidad, sino que la controla un conjunto de nodos que forman un **consorcio**.

En cuanto a su acceso, podría ser público o solo limitarlo a unos pocos usuarios. Una de las principales ventajas que tienen, es que al ser una mezcla de red pública y privada, pueden coger lo mejor de cada una y evitar sus inconvenientes. Esto es muy atractivo desde el punto de vista de las empresas, ya que se configura un entorno de trabajo creado a medida para la interacción con sus usuarios.

Un claro beneficio lo encontramos en que estas redes se protegen de forma implícita contra ataques como el **ataque del 51 %⁴**. Otra ventaja es la posibilidad de esconder ciertos datos a la misma vez que se comunica con el exterior.

Un ejemplo de redes híbridas son todas aquellas que se han creado bajo el framework **Hyperledger Fabric** (ver Sec. 2.7)

2.3. SMART-CONTRACT

La primera definición que se dio de contrato inteligente es que son un protocolo de transacciones que ejecuta los términos de un contrato. En el contexto de Blockchain, un contrato inteligente o smart-contract es un conjunto de datos y código que se despliega en una red por medio del uso de transacciones. En su esencia, no es más que un programa informático, como cualquier otro que podemos conocer, y que su código es ejecutado por los nodos y que **siempre es determinista**: lo que se traduce en que para una entrada siempre se produce la misma salida. Esta salida se almacenará en la Blockchain.

Los smart-contracts permiten que los procesos que tengan como objetivo se realicen de manera automatizada y verificando que se cumplen las condiciones que se establecen en ellos. Tienen diversas aplicaciones, se pueden usar para compartir bienes (dinero, propiedades, etc) entre las partes implicadas transparentemente, ya que el código del contrato debe ser público, de esta manera se ahorra el hecho de tener que confiar en una tercera parte, que de lugar a conflictos de intereses, manipulación... Esto implica también que la programación de cierto contrato debe realizarse de forma minuciosa para evitar errores de los que puedan aprovecharse las partes o terceros.

Las características más importantes de un contrato inteligente son:

- **Eficiencia temporal.** En cuanto a eficiencia temporal nos referimos a que hoy en día un proceso burocrático se puede demorar semanas, días, meses ya que conlleva mucho tiempo revisar minuciosamente todos los documentos y aprobarlos. Con un contrato inteligente estos tiempos se reducen de forma considerable, ya que se comprueba según lo programado en ellos, y cuando se realiza la comprobación se produce una salida.
- **Seguridad.** De forma implícita los smart-contract tienen la misma seguridad que la red Blockchain, esto se traduce en que no se puede modificar el contrato. Las posibles vulnerabilidades están asociadas a errores humanos tales como fallos en el proceso de programación.
- **Acuerdos en grupo.** Cuando se alcanza un porcentaje de acuerdo se pueden ejecutar ciertas cláusulas. Un ejemplo sería el de un grupo de amigos que realiza una apuesta deportiva, donde cuando el 70 % de las personas envíen el nombre de la persona que ha acertado esa apuesta, se pague el dinero a dicha persona.

En las redes públicas, como por ejemplo Ethereum[10] cada vez que un contrato inteligente es llamado a ejecución, se cobra un porcentaje que es proporcional al número de instrucciones

⁴El ataque del 51 % es el nombre que reciben los ataques en los que un cibercriminal o persona con malas intenciones intenta hacerse con el control de la red poseyendo al menos el 51 % de los nodos

ejecutadas, es decir, se le asocia un gasto económico. De hecho, la primera red Blockchain que los implementó fue Ethereum[10] y desarrolló y lenguaje de programación para crearlos llamado **Solidity**[16].

2.4. PROYECTO ETHEREUM

Cuando Ethereum se presentó lo hicieron bajo es eslogan “*Ethereum: a next generation Smart contract and decentralized application Platform*”. Su creador es **Vitalik Buterin** y lo hizo en 2015. Antes era divulgador tecnológico en varios blogs y desarrollaba artículos para **Bitcoin Magazine** de la que es co-fundador. Tras dejar este rol de divulgador, se asoció con un grupo de personas y fundó Ethereum.

Ethereum se traduce como una plataforma programable, pública y distribuida, que da soporte a la creación de aplicaciones descentralizadas basadas en la Blockchain y que es capaz de ejecutar programas en cualquier código (C, C++, JavaScript...).

Muchas personas en este punto estarán haciendo la pregunta *¿Cuál es la diferencia entre Bitcoin y Ethereum?* La primera la encontramos en que Bitcoin presenta limitaciones a la hora de los lenguajes de programación que interpreta, por que es *Incompleto*. La segunda es que Bitcoin **no** utiliza contratos inteligentes al contrario que Ethereum, lo que proporciona a Ethereum la cualidad de realizar transacciones más sofisticadas que Bitcoin, que se remite al mero pago.

Ethereum está también llamando la atención de los gigantes de las finanzas y de la tecnología, como JPMorgan Chase, Microsoft e IBM...

(The New York Times)

2.4.1. Ether

Las criptomonedas que están alojadas en la red Ethereum se denominan Ether (ETH) y no hay que confundir los términos Ethereum (red) y Ether (criptomoneda) ya que a veces se les suele llamar Ethereum a los dos conceptos de forma errónea. En el momento de escribir este TFG, ETH es la criptomoneda que más capitalización de mercado tiene después de Bitcoin habiendo llegado a alcanzar un precio máximo de 4.891,70\$ en noviembre de 2021.



Figura 2.10: Logo de ETH [10]

2.5. TOKENS

Un token es una cadena de caracteres alfanumérica, que se utiliza para algún fin (8893-dd67ds-8973dj4, por ejemplo). En el contexto de la seguridad informática, normalmente consiste en la sustitución de datos sensibles por una cadena equivalente (que lo representa) llamado token, que no tiene significado o valor explotable. Es decir, estos tokens sirven para hacer comprobaciones y validar (como por ejemplo, cuando queremos cambiar nuestra contraseña en un sistema, nos suelen mandar un link que lleva un token en la URL, Fig. 2.11).

Si tienes problemas dando click al botón "Cambiar contraseña", copia y pega el siguiente enlace en tu navegador: <http://localhost/password/reset/>
 190056dd6789eb32f42ead92a72f63b98af684d7170421bed426be84e8e05cb6?
 email=rogeliotoba%40gmail.com&guard=api

Figura 2.11: En rojo, ejemplo de utilización token de seguridad en URL

En el contexto de Blockchain, cuando hablamos de tokenizar algún bien, nos referimos a estas mismas representaciones, en cambio, estos tokens sí tienen un valor asociado, y se guardan en la cadena de bloques. Representan un bien, el que se quiera digitalizar (una obra de arte, un objeto de colección como botas de fútbol, un video, etc.) Hoy en día son diversos los bienes físicos y digitales que se tokenizan. Tenemos varios tipos de token que hay que diferenciar bien: Token Fungible (FT) y Token No Fungible (NFT).

2.5.1. Token Fungible (FT)

Los FT son aquellos tokens que pueden repetirse, es decir, puede haber un número indefinido de ellos y todos son iguales, poseen la cualidad de fraccionarse. Un ejemplo de ello serían las criptomonedas, son tokens que tienen todos el mismo valor y de los que hay muchos, y todos iguales.

Uno de los estándares más famosos de los FT es el ERC-20 de Ethereum (a partir del cuál ha habido adaptaciones y han surgido otros basados en él) y su uso es el de emitir o crear criptomonedas. Litecoin[12] y Ethereum[10] están basados en este estándar.

2.5.2. Token no fungible (NFT)

Los NFT por su parte, no pueden fraccionarse, tampoco puede haber dos iguales ni pueden cambiarse entre sí. Un NFT por lo tanto **es único**. Al ser únicos, se almacenan en la cadena de bloques, para dejar constancia de la propiedad con una transparencia total. Esto, sumado a la seguridad que ya aporta de por sí la cadena de bloques hace que no puedan ser manipulados o destruidos, asegurando así la **integridad** de los datos.

Una de sus aplicaciones por tanto puede ser la de verificar o registrar la propiedad de algún objeto valioso, como por ejemplo de una obra de arte. Otro de los usos que puede tener, y, que cada vez se habla más de ello, es el de accesorios que se pueden conseguir en los videojuegos, pongamos de ejemplo una “skin”⁵, un arma especial, que solo se pueda conseguir y utilizar si se tiene ese NFT, y, que además se puede vender en la vida real, adicionalmente su precio fluctúa y puede sacarse un beneficio (o pérdida) de ese NFT. También se puede aplicar a sistemas de votación o de entradas a un espectáculo deportivo, concierto... donde cada NFT proporciona acceso al evento.

2.5.3. Cómo se compran NFTs

Para comprar NFTs a día de hoy, se necesitan básicamente solo dos cosas: un medio de pago y una wallet (billetera).

En cuanto al medio de pago, este se puede hacer por tarjeta de crédito con dinero FIAT o por criptomoneda, nos centraremos en este último. La divisa que utilicemos puede variar dependiendo del sitio donde compramos los NFTs, aunque por lo general se suele hacer utilizando Ethereum y, últimamente se está popularizando la red de Solana. Ethereum es con diferencia el más utilizado ya que es la mayor plataforma de contratos inteligentes y es la red que cuenta con mayor número de proyectos de blockchain. También, lo más usual es que a este precio se sume un pequeño porcentaje, por ejemplo, del 1% del precio como gastos de gestión. Las criptomonedas se pueden comprar en plataformas como [Binance](#), [Crypto](#) y [Kraken](#).

Además, como hemos comentado anteriormente, necesitamos una Wallet o billetera virtual para guardar nuestros NFTs una vez los hemos adquirido. La billetera más famosa, debido a su simplicidad e intuitividad a la hora de crearnos una es [Metamask](#)[13]. También tenemos otra opción como puede ser un monedero hardware, en este caso el más famoso es el de [Ledger](#). Esta opción es claramente más segura, se les suele llamar **Cold Wallet** y se trata de dispositivos parecidos a un USB convencional

⁵skin cuyo significado literal en español es *piel* se utiliza en el contexto de los videojuegos para referirse a la apariencia física de un personaje, arma, vehículo, etc.

donde almacenamos nuestros tokens, y aunque una billetera en Internet también es fiable, la física está protegida contra ciber-ataques.

2.6. EL ESTÁNDAR ERC-721

Los estándares de tipo de ERC son aquellos que son utilizados por la red Ethereum y que sirven para proporcionar **interoperabilidad** en distintas plataformas que tiene esta blockchain como pueden ser OpenSea, SuperRare o Decentraland. Este concepto, para que sea más fácil de entender, podríamos asociarlo a un estándar MP3 para audio, al JPG de las imágenes... Generamos así la capacidad de exponer nuestro NFT en Decentraland (es como un museo o galería de NFTs de cada usuario) y venderlo al mismo tiempo en OpenSea.

Este estándar fue el primero que surgió en la red Ethereum para poder crear NFTs y es uno de los más usados a nivel mundial actualmente. Si se utiliza para crear NFTs, **cada token ERC-721 es único**, solo existe uno y jamás se podrá modificar e intercambiar con otro (ya que es no fungible). Esto se consigue con un asignador único para cada token y conseguimos que, aunque dos imágenes sean aparentemente iguales, en realidad son distintas ya que su identificador no es igual. Un ejemplo típico, es, las coordenadas geográficas: si seleccionamos dos puntos muy próximos, con una variación de unos segundos, nos parecerá el mismo sitio e iguales, pero en realidad son diferentes.

Al no ser fungibles, no se pueden intercambiar, sería como sustituir un cuadro de Da Vinci original por uno falso, el falso no tiene valor. Este estándar es perfecto cuando queremos generar piezas únicas, que gozan de exclusividad, solo hay 1/1. Sin embargo, este estándar no es tan óptimo cuando lo que se pretende es lanzar 1000 ediciones de una misma obra, por ejemplo, *skins* de un juego donde el mismo personaje tiene una bufanda de un color, un gorro, unos zapatos... esto generaría muchos costes adicionales debido a los gas fees⁶. Para solucionar este problema, se creó el ERC-1155.

2.6.1. Otros ERC de interés

El estándar que más interesa en este TFG es el ERC-721, no obstante hay otros ERC que pueden resultar de interés.

- **ERC-20.** El estándar ERC-20 es el más común de todos los estándares para tokens digitales y el más funcional de todos. En este token se basa la criptomoneda de Ethereum (Ether) y este estándar permite el intercambio de ella, que suele tener valor de algún tipo.

Cuando hablamos de un token ERC-20 estamos refiriéndonos a que es un **token fungible**, es decir, se pueden intercambiar entre sí ya que todos son iguales. Es como los euros físicos, da igual que moneda tengas, todas tienen el mismo valor. Estos token pueden almacenarse sin problema alguno en una billetera Ethereum.

- **ERC-1155.** Este estándar surge como combinación del token fungible con el token no fungible. Permite la generación de varios tipos de NFTs con un solo Smart Contract. Un ejemplo sería el contado anteriormente de las *skins* de los juegos.
- **ERC-2309.** Este estándar es una expansión del ERC-721 y la principal utilidad que tiene es que permite la creación de múltiples ERC-721 de una vez, lo que supone un ahorro considerable de gas fees. Podríamos entenderlo como copiar una carpeta a un disco duro en vez de transferir uno a uno los archivos.
- **ERC-223.** Sirve para proteger al cliente o usuario de transferencias accidentales.
- **ERC-827.** Permite autorizar a un tercero a gastar el token.
- **ERC-777.** Permite al usuario rechazar un token que proviene de una cartera que no es deseada.

⁶Los gas fees o tarifas de gas son una cuantía económica que los usuarios de la Blockchain pagan a los mineros para que la transacción se incluya en un bloque, normalmente supone un pequeño porcentaje, entre el 1 % y el 5 % de la transacción.

La principal diferencia que existe entre los estándares ERC-721 y ERC-1155 es la forma en la que se registra la propiedad. Si se tienen 20 NFTs de una chaqueta (ERC-1155, chaqueta en el contexto de *skin* de videojuego), no existiría la posibilidad de distinguir unos de otros, dentro la wallet aparecerían diez chaquetas que tienen el mismo ID. Es decir, estos token tienen fungibilidad parcial, ya que son solo intercambiables los del mismo tipo.

Los ERC-721 se utilizan cuando se quiere crear un token totalmente único, 1/1. Los ERC-1155 cobran sentido cuando se tienen más ediciones, es decir 1/20000, ya que de otra forma sería inviable. Una de las causas es que transferir token ERC-1155 es menos costoso respecto a ERC-721 ya que solo se trata de actualizar un valor que es la cantidad, y en el otro caso es transferir uno a uno. En cuanto a valor, los ERC-721 tiene más, ya que son únicos.

2.7. HYPERLEDGER FABRIC

Hyperledger Fabric es un framework de código abierto usado para la creación de redes Blockchain privadas (o híbridas) que tiene como fin apoyar el avance de la tecnología de cadena de bloques en sus diferentes aplicaciones, industrias, áreas, etc. Hyperleger Fabric surge como una colaboración a escala mundial de la cual forman parte empresas punteras de las finanzas, banca, internet de las cosas, tecnología, cadenas de suministro, etc.

Se utiliza mayormente para desarrollar Blochian de entorno empresarial, ya que tiene controles de seguridad y privacidad, donde el/los administradores puedes determinar quien tiene acceso a los recursos. Las transacciones de las cadenas de bloques creadas con este framework son naturalmente rastreables e irreversibles, mecanismo que aporta **seguridad**.

Algunos de los beneficios [11] que remarcan en cuanto a la utilización de Hyperledger Fabric son:

- **Red con permisos.** Se otorga permisos a quien se autoriza, en lugar de usuarios anónimos como en una red pública.
- **Transacciones confidenciales.** Solo se comparten los datos que se desean con quien se desea.
- **Arquitectura conectable.** La red Blockchain se puede adaptar a las necesidades de negocio, debido a su arquitectura conectable.
- **Fácil inicio.** Los contratos inteligentes se pueden programar en varios lenguajes, utilizando el que más convenga al equipo de desarrollo o a la empresa.

Lo más frecuente es utilizar Hyperledger Fabric junto con IBM Blockchain Platform, una distribución comercial que da soporte a Hyperledger Fabric y de la que incluye herramientas para crear, gestionar y operar la Blockchain que se cree. La red Blockchain de este TFG se ha desarrollado utilizando este framework y la extensión de IBM Blockchain Platform para Visual Studio.

CAPÍTULO 3

Objetivo

En este capítulo se explica el objetivo general así como los objetivos específicos de una forma más detallada:

3.1. OBJETIVO GENERAL

El objetivo general del proyecto **es la creación de una plataforma Blockchain de gestión de activos digitales mediante tokens no fungibles (NFT)**. La virtualización se realizará en una red Blockchain de forma segura, STO (Secure Token Offering).

La red Blockchain estará alojada en un clúster de Kubernetes en la que existirá un Smart Contract basado en Hyperledger Fabric que define la generación y las reglas de *trading* de los activos digitales. Dichas reglas seguirán el estándar de tokenización de activos ERC-721 (ver Sec. 2.6).

3.2. OBJETIVOS ESPECÍFICOS

Del objetivo general, surgen otros objetivos específicos o subobjetivos que se describen a continuación.

3.2.1. Construcción de una red Blockchain

Se procede a la creación de una red Blockchain alojada en un clúster de kubernetes garantizando la estabilidad del sistema. La red Blockchain se creará con Hyperledger Fabric (ver Sec. 2.7)

3.2.2. Creación de un contrato inteligente

Se creará un Smart Contract que define la generación y las reglas del trading. El Smart Contract debe poder gestionar:

En cuanto al **token ERC-721**:

- Aprobar cambios o reafirmar el cliente aprobado para un NFT.
- Habilitar o deshabilitar la aprobación de un tercero para administrar.
- Devolver el cliente aprobado para un único NFT.
- Devolver si un cliente está aprobado para operar con un NFT.
- Devolver si un cliente es un operador autorizado de otro cliente.
- Crear un nuevo NFT.
- Transferir la propiedad de un NFT de un cliente a otro cliente.
- Eliminar un NFT.

En cuanto al **custodio**:

- Crear solicitud de consumo del NFT.
- Aceptar solicitud de consumo de NFT.
- Rechazar solicitud de consumo de NFT.
- Devolver información de solicitud de consumo de NFT.
- Devolver una lista de todas las solicitudes de consumo.
- Devolver una lista de todos los NFTs que tiene un custodio.
- Eliminar una solicitud de consumo.

En cuanto al **trading**:

- Crear solicitud de compra.
- Aceptar solicitud de compra.
- Rechazar solicitud de compra.
- Devolver información de solicitud de compra de NFT.
- Devolver una lista de todas las solicitudes de compra.
- Devolver una lista de todos los NFTs que tiene un compra.
- Eliminar una solicitud de compra.

En cuanto al **usuario**:

- Devolver información del cliente.
- Devolver todos los tokens de un cliente.
- Devolver el histórico de transacciones de un cliente.
- Devolver el balance de un cliente.
- Devolver todos los tokens de un cliente.
- Actualizar el saldo de un cliente.

En cuanto al **propio objeto NFT**:

- Devolver información del NFT.
- Devolver el historial de transacciones de un NFT.
- Devolver una lista con todos los NFTs.
- Devolver una lista con todos los NFTs filtrados por estado..
- Devolver una lista con todos los NFTs filtrados por precio.

3.2.3. Interfaz para acceder a los servicios

Se dotará de una aplicación web cliente para el acceso a los servicios. Se necesita un servidor para interaccionar con la red Blockchain. Se debe modelar:

- Arquitectura cliente-servidor que permita la comunicación entre ambos.
- Las aplicaciones no tienen porque estar escritas en el mismo lenguaje y aún se deben poder comunicar las partes.

La aplicación web se debe comunicar con los diferentes módulos y debe permitir:

- **Registrar un usuario.** Registra un usuario en la red Blockchain.
- **Importar identidad.** Importa la identidad y el certificado y clave privada del usuario.
- **Llamar a una transacción.** Llama a una transacción del contrato digital en la red Blockchain.
- **Llamar a varias transacciones en bucle.** Llama a varias transacciones en bucle del contrato digital en la red Blockchain.
- **Visualizar la lista de NFTs disponibles.** El usuario debe poder visualizar los NFTs que están disponible para su compra
- **Permitir el login de usuario.** Un usuario debe ser capaz de iniciar sesión con su cuenta.
- **Diferenciación de interfaces entre usuario y administrador.** El administrador deberá visualizar opciones que un usuario base no debería de poder realizar, tales como la creación de un NFT, la eliminación de un NFT, etc.

3.2.4. Custodia de credenciales y certificados de usuario

Mediante un módulo SoftHSM se guardarán de forma segura las claves privadas de los participantes de la Blockchain (requeridas por esta misma para las transacciones e identificación), así mismo, y de forma automática, la Blockchain requerirá cuando sea necesario las claves al módulo SoftHSM. Su implementación se realizará mediante una librería específica para SoftHSM en Hyperledger Fabric.

3.2.5. Arquitectura desacoplada

La arquitectura del sistema debería desarrollarse pensando en la reutilización del código, con un enfoque que facilite su despliegue y su mantenimiento, cuya recuperación en caso de error sea fácil de gestionar y de solucionar.

- Arquitectura dividida en módulos, para facilitar su desarrollo.
- A su vez, cada módulo debería estar basado en **microservicios** que aportan desacople entre las partes facilitando su desarrollo y la detección y corrección de errores.
- Se utilizará el patrón MVC (Modelo-Vista-Controlador) en la parte Backend.

3.2.6. Se deben utilizar tecnologías libres

El hardware y software con el que se desarrolle el proyecto debería ser de libre uso (salvo en modo producción), por lo que:

- La herramientas utilizadas no deben de conllevar el pago de una suscripción u otro coste económico asociado.
- Las bibliotecas externas utilizadas durante la escritura del código usadas que sean de código abierto en la medida de lo posible así como sus licencias.

3.2.7. Despliegue fácil, rápido y eficaz

El desplegar la aplicación no debe suponer un esfuerzo mayúsculo para el cliente, desarrollador y posteriores usuarios que puedan intervenir durante el ciclo de vida del proyecto. Para ello:

- Se hará uso de Docker, para la contenerización de la aplicación, de esta forma se dispondrá de todo lo necesario para su ejecución en los contenedores de software.
- Clúster de kubernetes, para la gestión y orquestación de dichos contenedores Docker.

CAPÍTULO 4

Metodología

4.1. METODOLOGÍA DE TRABAJO: METODOLOGÍAS ÁGILES

Una metodología de trabajo se puede definir como “*conjunto de filosofías, fases, procedimientos, reglas, técnicas, herramientas, documentación y aspectos de formación para los desarrolladores de Sistemas de Información*”[18]. Una metodología define un camino para desarrollar software de una manera procedural y sistemática, hace la función de **estándar de trabajo** para la organización.

Este proyecto se ha desarrollado siguiendo una metodología ágil, cuya principal característica es que se centra en los factores humanos y en el producto software. Se da mayor relevancia a las personas y la interacción y comunicación con el cliente, a su vez, el software se desarrolla de manera incremental y con iteraciones de poca duración.

Las metodologías ágiles son otra opción de desarrollo frente a las metodologías tradicionales. En las metodologías ágiles se encaminan a las personas y se pueden adaptar, mientras que las tradicionales tienen una planificación severa y se orientan a procesos. Un proyecto ágil se lleva a la iteración al extremo:

- Las tareas del proyecto son divididas en incrementos que llevan una pequeña planificación a seguir y son de corta duración (entre 1 semana y 4).
- Una iteración termina con un **prototipo operativo** y se genera un producto entregable que es mostrado y revisado junto al cliente

De igual forma, los desarrollos ágiles se basan en los siguientes principios[2]:

- **Individuos e interacciones sobre procesos y herramientas.** El proyecto se construye en torno a individuos motivados y se les proporciona el entorno adecuado así como el apoyo necesario para confiar en ellos.
- **Software funcionando sobre documentación extensiva.** La simplicidad es esencial, las mejores arquitecturas, requisitos y diseños surgen de los equipos que se organizan por sí mismos. El software que funciona es la principal medida de progreso, y este se debe entregar con el menor intervalo posible.
- **Colaboración con el cliente sobre negociación contractual.** El diálogo cara suele ser el método más eficiente y efectivo para comunicar información, el cliente está informado en todo momento del estado del proyecto, equipo de desarrollo y cliente se reúnen de forma constante y trabajan conjuntamente. Se debe satisfacer al cliente con entregas tempranas y continuas.
- **Respuesta ante el cambio sobre seguir un plan.** Se capturan los cambios para que el cliente tenga una ventaja competitiva. El equipo reflexiona sobre cómo puede llegar a ser más efectivo y según ello, ajusta su comportamiento.

El hecho de utilizar la palabra *sobre* no es para despreciar lo que hay a la derecha de ella, sino para dar importancia y potenciar las de la izquierda.

Durante los primeros años de aparición de metodologías ágiles, surgieron otras como:

- | | |
|---|-------------------------------------|
| (1) Adaptative Software Development (ASD) | (4) Lean Software Development (LSD) |
| (2) Agile Unified Process (AUP) | (5) Programación Extrema (XP) |
| (3) Scrum | (6) Open Unified Process (OpenUP) |

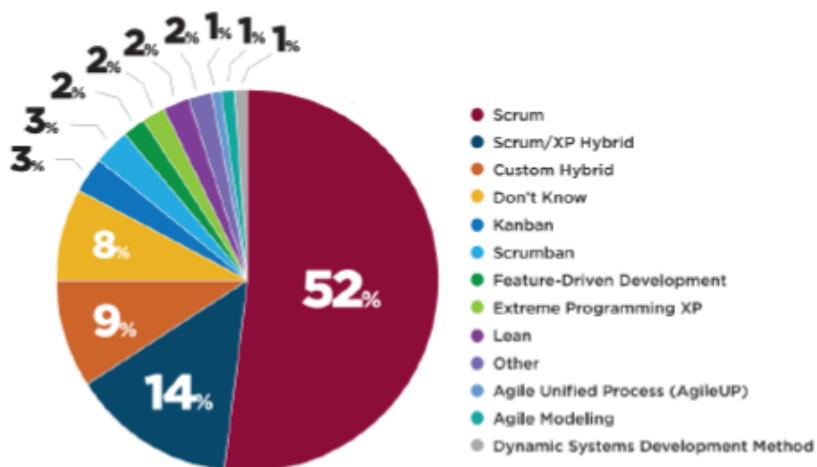


Figura 4.1: Metodologías ágiles más usadas. [1]

Este proyecto se ha desarrollado siguiendo Scrum, que es la metodología ágil más usada actualmente y que se explica en la siguiente sección.

4.2. SCRUM

Lo primero que hay que destacar es que Scrum es una metodología de proyectos que se puede aplicar a cualquier ámbito, no solo a proyecto de desarrollo Software. Consiste en un framework cuya adaptación es diferente dependiendo de la organización en la que se aplique y cuyo objetivo es obtener resultado de una forma rápida y eficaz permitiendo la adaptación y modificación de lo planificado en función de las necesidades que marquen los clientes. Scrum tiene dos características principales: el desarrollo software se realiza por medio de **iteraciones incrementales** y se hacen reuniones a modo de coordinación y *feedback* durante el proyecto que se conocen como *daily*.

A su vez, Scrum define un ciclo de vida que es iterativo e incremental, en la que se intentan mejorar y reducir los riesgo y se fortalece la comunicación. Los tres pilares fundamentales son:

- **Inspección.** Se debe de revisar que lo que se está desarrollando se está realizando de forma correcta para evitar variaciones que no serían aceptadas por el cliente.
- **Transparencia.** Todos lo referente a la gestión del proyecto es visible para quienes gestionan el resultado.
- **Revisión.** El proyecto debe cumplir unos requisitos y límites aceptables.

En Scrum[6], a cada iteración se le conoce como Sprint, y tienen un duración de entre 1 y 4 semanas. Primero, el equipo debe de investigar las tareas, después se estimará el tiempo empleado en realizar cada una de ellas y por último, se le asignará a un miembro del equipo de desarrollo una tarea. El trabajo se realiza de forma independiente por cada miembro, y se van realizando las reuniones (dailies) que se consideren oportunas para poner datos en común. Al final de cada Sprint, el equipo se encarga de conformar el incremento y mostrarlo al cliente para que este lo revise y pueda aportar su opinión y presente las variaciones oportunas.

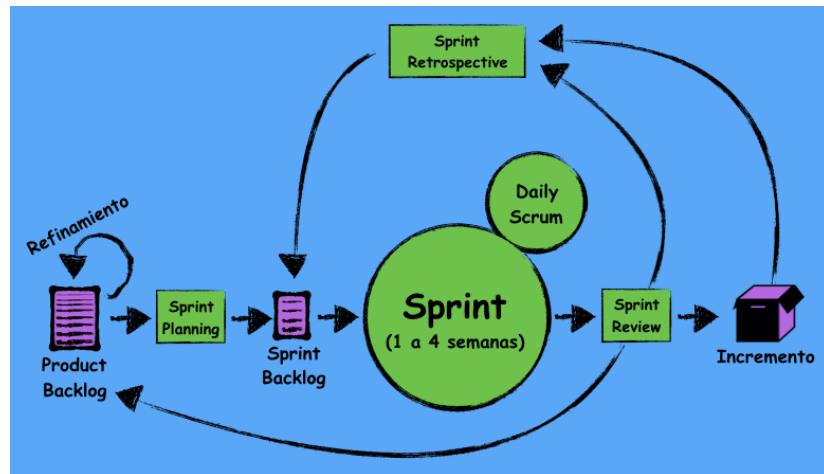


Figura 4.2: Proceso de desarrollo en Scrum [6]

El *Product Backlog* es una lista donde recogen las historias de usuario. A partir del *Product Backlog* surge el *Sprint Backlog* que es una lista de tareas, la que hemos mencionado anteriormente.

En proyecto Scrum existen diferentes roles que interpretan responsabilidades:

- **Product Owner:** Las funciones del Product Owner son la de definir las características del producto, es responsable de asegurar el retorno de la inversión, es la persona que conoce las necesidades del cliente y el que las plasma en el Backlog del Producto. Destaca que tiene la posibilidad de financiar el proyecto desde su inicio hasta su final y es el que acepta o no los resultados del trabajo realizado.
- **Scrum Master:** Es el representante de la gestión del proyecto, se asegura de que el equipo funciona correctamente y es un entorno productivo, facilita la cooperación entre los diferentes roles y la comunicación de los mismos, es decir, es un facilitador.
Revisa e inspecciona los cambios y es el encargado de blindar al equipo de influencias externas y de comunicar en *feedback* que le dio el cliente
- **Development Team:** Su tarea es la de desarrollar el producto, son los responsables de que cada iteración cumpla con los establecido y que se progrese de manera adecuada. Son auto-gestionados, multifuncionales, no distribuidos y tienen un tamaño óptimo de acuerdo a la grandeza del proyecto. No se recomienda que el equipo cambie mientras se está dentro de un sprint.

4.3. APLICACIÓN DE LA METODOLOGÍA DE TRABAJO

Este proyecto se ha desarrollado siguiendo la metodología ágil Scrum (ver Sec. 4.2). De acuerdo a los roles de Scrum tratados en la sección anterior, en este proyecto, se definen de la siguiente forma:

- **Product Owner:** La empresa u organización interesada en el software que se ha desarrollado y que por motivos de confidencialidad no se puede añadir como tal.
- **Scrum Master:** Manuel Hervás Ortega
- **Development Team:** El equipo de desarrollo de Alpinia Technologies formado por Juan Manuel Porrero Almansa, María Isabel Ortiz Lizcano, Rubén Pérez Rubio y Diego Fernando Villanueva Penagos.

Se han seguido las prácticas de metodologías ágiles de Scrum tales como:

- **Dailies:** todos los días alrededor de las 12 horas de la mañana, el equipo de desarrollo se reunía con el Scrum Master para poner en común lo desarrollado hasta el momento, exponer

dificultades encontradas por cada miembro en su tarea, los miembros del equipo dialogaban acerca de la utilización y acople de los diferentes módulos desarrollados, etc.

- **Sprints:** el equipo avanzaba de acuerdo a los sprints establecidos, con capacidad de adaptación en base al *feedback* recibido por parte del cliente.

4.4. DISEÑO

En esta sección se han identificado y analizado las historias de usuario, para después hacer una planificación de los Sprints sobre los cuales se va a desarrollar y planificar el software.

Existen perfiles de usuario con diferentes roles y en función de ellos poseen credenciales para realizar unas acciones u otras (recordatorio de que los NFTs son generados a partir de activos reales):

- **Usuario:** persona genérica que utiliza el Marketplace. Puede realizar compras, ventas y transferencias de sus items adquiridos.
- **Custodio:** empresa, organización, persona que tiene los activos en custodia. Si un cliente reclama uno de los activos que tiene en custodia, tiene la capacidad de quemar el token asociado al activo.
- **Administrador:** administra la aplicación y por lo tanto, tiene la responsabilidad de configurar adecuadamente las opciones del sistema. Es quien puede crear los tokens asociados a los activos que se añaden al Marketplace.

4.4.1. Historias de usuario

Al inicio de cada iteración se realiza una lista de requisitos y las historias de usuario forman parte de ella. Describen una necesidad específica del usuario al utilizar el producto que se va a desarrollar. Tienen como fin la recolección de ideas para que más tarde se estime su duración y esfuerzo. En el proyecto desarrollado en este TFG se han identificado las siguientes historias de usuario:

ID	US_REGISTRATION
Rol	Como USUARIO
Funcionalidad	Podré registrarme en el Marketplace.
Resultado	Funcionalidades básicas del usuario.
Aceptación	Se podrá registrar un usuario en el Marketplace introduciendo dirección de correo electrónico y contraseña, así como datos personales entre los que se incluye DNI e imagen del DNI y la activación de la autenticación de doble factor.

ID	US_LOGIN
Rol	Como USUARIO
Funcionalidad	Podré iniciar sesión.
Resultado	Funcionalidades básicas del usuario.
Aceptación	Se podrá iniciar sesión introduciendo email y contraseña en un formulario. Se validará que ningún campo esté vacío, más el control del formato de usuario/contraseña indicados por el cliente. Si ocurre algún error se mostrará una alerta simple con la descripción del problema. Se ofrecerá la opción “Recuérdame” para que, si se activa, se almacenen los datos de sesión. Tras iniciar sesión de forma correcta, se le solicitará introducir la clave asociada al 2FA. Una vez validada la clave 2FA, se redirigirá a la pantalla principal del wallet del usuario

ID	US_RESETPASSWORD
Rol	Como USUARIO
Funcionalidad	Podré recuperar mi contraseña.
Resultado	Funcionalidades básicas del usuario.
Aceptación	Cuando el usuario no recuerde su contraseña podrá solicitar recuperarla mediante un enlace que se le enviará a su dirección de email. Al acceder al enlace recibido accederá a una pantalla donde se le brindará la opción de introducir una nueva contraseña. Se presentará un campo para la introducción de la nueva contraseña y otro para repetirla a modo de confirmación. Antes de cambiar la contraseña se validará que ningún campo esté vacío, más el control del formato de usuario/contraseña indicados por el cliente. Si ocurre algún error se mostrará una alerta simple con la descripción del problema.

ID	US_LOGOUT
Rol	Como USUARIO
Funcionalidad	Podré cerrar mi sesión.
Resultado	Funcionalidades básicas del usuario.
Aceptación	Se podrá cerrar sesión en cualquier pantalla del sistema. Tras cerrar sesión, el sistema se redirigirá hacia la pantalla de inicio de sesión.

ID	US_EDITPROFILE
Rol	Como USUARIO
Funcionalidad	Podré editar mi perfil.
Resultado	Funcionalidades básicas del usuario.
Aceptación	Se podrán editar los datos del perfil, excepto los datos personales, que estarán asociados a la imagen del DNI que se ha subido.

ID	US_CATALOGUE
Rol	Como USUARIO
Funcionalidad	Podré visualizar el catálogo de NFTs de la plataforma.
Resultado	Funcionalidades básicas del usuario.
Aceptación	Mediante un explorador se mostrarán los NFT que se encuentran disponibles y públicos en la plataforma. Se podrá filtrar en base a diferentes criterios.

ID	US_WALLET
Rol	Como USUARIO
Funcionalidad	Podré visualizar mi wallet.
Resultado	Funcionalidades básicas del usuario.
Aceptación	Se visualizarán: los ítems que el usuario haya adquirido en la plataforma, las solicitudes de ofertas recibidas por un determinado item, dinero FIAT disponible para utilizar en el marketplace, coste de comisiones de custodia

ID	US_VISIBILITY
Rol	Como USUARIO
Funcionalidad	Podré seleccionar la visibilidad de los NFTs adquiridos en el marketplace.
Resultado	Funcionalidades básicas del usuario.
Aceptación	En el wallet, los NFTs adquiridos dispondrán de un check para definir su visibilidad en búsquedas del Marketplace. De esta forma se tendrá la opción de indicar NFTs en “modo privado” que no serán visibles para el resto de usuarios de la plataforma.

ID	US_STORESERVICE
Rol	Como USUARIO
Funcionalidad	Podré disponer de un servicio de custodia del activo.
Resultado	Funcionalidades básicas del usuario.
Aceptación	Al adquirir el token de un activo real, se ofrecerá al comprador la custodia del activo en un lugar adecuado. Este servicio tiene asociado un coste recurrente que se informará al usuario en el momento de la contratación.

ID	US_LOADFIAT
Rol	Como USUARIO
Funcionalidad	Podré cargar dinero FIAT a mi wallet.
Resultado	Funcionalidades básicas del usuario.
Aceptación	Para realizar compras dentro del marketplace, previamente, el usuario debe disponer de saldo FIAT en su wallet. El usuario podrá indicar que cantidad añadir y mediante una pasarela de pago (stripe, redsys,...) cobrarla de la tarjeta de crédito.

ID	US_RECEIVEFIAT
Rol	Como USUARIO
Funcionalidad	Podré recibir dinero FIAT desde mi wallet.
Resultado	Funcionalidades básicas del usuario.
Aceptación	El saldo FIAT no consumido podrá ser transferido nuevamente al usuario mediante la pasarela de pago.

ID	US_BUY
Rol	Como USUARIO
Funcionalidad	Podré adquirir un activo que se encuentre en la plataforma.
Resultado	Funcionalidades básicas del usuario.
Aceptación	Se pueden comprar los NFTs disponibles en el Marketplace. La compra puede conllevar una comisión, aplicable al comprador o vendedor. Las NFTs tendrán un valor asociado y podré realizar la compra del NFT si dispongo de saldo FIAT disponible. Una vez finalizada la compra, el NFT se añadirá a mi wallet.

ID	US_SELL
Rol	Como USUARIO
Funcionalidad	Podré vender un NFT de mi propiedad.
Resultado	Funcionalidades básicas del usuario.
Aceptación	Sobre las botellas de mi propiedad podré poner disponible para la venta una o varias unidades de un vino para compra particular.

ID	US_SENDFIFT
Rol	Como USUARIO
Funcionalidad	Podré regalar un NFT a otro usuario de la plataforma.
Resultado	Funcionalidades básicas del usuario.
Aceptación	Se dispondrá de una opción de regalar un NFT a otro usuario de la plataforma. De esta forma, se realizará una transferencia de propiedad sin transferencia de saldo FIAT.

ID	US_RECEIVEGIFT
Rol	Como USUARIO
Funcionalidad	Podré recibir un NFT de otro usuario de la plataforma.
Resultado	Funcionalidades básicas del usuario.
Aceptación	Se podrá recibir un NFT de otro usuario de la plataforma. De esta forma, se realizará una transferencia de propiedad sin transferencia de saldo FIAT.

ID	US_TRANSACTIONS
Rol	Como USUARIO
Funcionalidad	Podré consultar el histórico de transacciones realizadas.
Resultado	Funcionalidades básicas del usuario.
Aceptación	Se visualizarán las transacciones históricas realizadas por el usuario a nivel general y/o filtradas para un NFT en concreto.
ID	CUST_MYITEMS
Rol	Como CUSTODIO
Funcionalidad	Podré visualizar el listado de NFTs que dispongo en custodia.
Resultado	Funcionalidades básicas del custodio.
Aceptación	Se podrá visualizar en el explorador la lista de los activos en custodia.
ID	CUST_DELETENFT
Rol	Como CUSTODIO
Funcionalidad	podré quemar el NFT asociado a un NFT.
Resultado	Funcionalidades básicas del custodio.
Aceptación	Una vez que un cliente solicita un NFT (consumo, envío o retirada) el custodio del activo quemará el token asociado. Al iniciar la quema, el usuario deberá aceptar el quemado.
ID	ADMIN_CREATE
Rol	Como ADMINISTRADOR
Funcionalidad	Podré crear un token ERC-721 asociado a una botella exclusiva.
Resultado	Funcionalidades básicas del administrador.
Aceptación	Se podrá dar de alta un token ERC-721 asociado a una botella de vino con las siguientes propiedades: nombre, imagen del activo, precio, etc.
ID	ADMIN_LISTUSERS
Rol	Como ADMINISTRADOR
Funcionalidad	Podré listar los usuarios del marketplace y realizar filtros por nombre, dirección o DNI.
Resultado	Funcionalidades básicas del administrador.
Aceptación	En el explorador se verá una lista con los usuarios.
ID	ADMIN_USERINFO
Rol	Como ADMINISTRADOR
Funcionalidad	Podré ver el detalle de un usuario (Datos Personales, Wallet y Facturación).
Resultado	Funcionalidades básicas del administrador.
Aceptación	Se mostrará por pantalla la información del usuario requerido por el administrador
ID	ADMIN_DELETENFT
Rol	Como ADMINISTRADOR
Funcionalidad	Podré quemar el NFT asociado a un activo.
Resultado	Funcionalidades básicas del administrador.
Aceptación	Se podrá quemar un NFT asociado al activo.
ID	ADMIN_SETTAXES
Rol	Como ADMINISTRADOR
Funcionalidad	Podré definir las comisiones de custodia y transferencia
Resultado	Funcionalidades básicas del administrador.
Aceptación	Como administrador se podrán fijar los costes de custodia y transferencia introduciendo estos valores en un formulario.

4.4.2. Requisitos no funcionales

Los requisitos no funcionales son aquellos que se refieren a características generales o restricciones del software que se está desarrollando. Los requisitos no funcionales de este proyecto son dos:

- Se utilizará la IBM Blockchain Platform y la IBM Cloud para alojar y construir el clúster.
- En cuanto a la seguridad de las claves, estas se almacenarán en unidades SoftHSM.

4.4.3. Definición de los sprints

Tras la identificación y análisis de las historias de usuario, se ha decidido dividir el proyecto en 8 sprints, que se realizarán de forma iterativa e incremental, quedando definidos de la siguiente forma:

SPRINT 1	
Funcionalidades	<ul style="list-style-type: none"> ■ Diseño de Pantallas para un Marketplace genérico en FIGMA según descripción de la oferta
Duración estimada	3 semanas

SPRINT 2	
Funcionalidades	<ul style="list-style-type: none"> ■ Diseño de Arquitectura en función de múltiples verticales ■ Adaptación de diseño FIGMA a la necesidad del cliente
Duración estimada	3 semanas

SPRINT 3	
Funcionalidades	<ul style="list-style-type: none"> ■ Construcción FRONTEND marketplace (Explorer, New Item, Login, Sign up, Item Details) ■ Construcción BACKEND marketplace (item details, explorer (sin filtros)) ■ Construcción del Transaction Manager con operaciones de registrar usuario, obtener la identidad y claves del usuario y llamar a las funciones del contrato
Duración estimada	3 semanas

SPRINT 4	
Funcionalidades	<ul style="list-style-type: none"> ■ Construcción del Event Listener y del Event Bus ■ Módulo de Authentication & HSM
Duración estimada	3 semanas

SPRINT 5	
Funcionalidades	<ul style="list-style-type: none"> ■ Despliegue en Kubernetes primera versión funcional NFT (Entrada de usuarios, mintado de tokens, listado de vinos (explorador), compra/venta, wallet, quema de tokens)
Duración estimada	3 semanas

SPRINT 6	
Funcionalidades	<ul style="list-style-type: none"> ■ Edición del perfil ■ Recuperar contraseña ■ Operaciones de transferencia de FIAT sobre el Wallet (operaciones de pasarela de pago) ■ Listados de custodios de activos ■ Listados de activos en un custodio
Duración estimada	3 semanas
SPRINT 7	
Funcionalidades	<ul style="list-style-type: none"> ■ Vista administrador del listado de usuarios ■ Facturación, comisiones de custodia y transferencia ■ Tracing, Logging y Monitoring del clúster
Duración estimada	3 semanas
SPRINT 8	
Funcionalidades	<ul style="list-style-type: none"> ■ Pruebas UAT y entrega ■ Documentación
Duración estimada	3 semanas

4.5. MARCO TECNOLÓGICO

El desarrollo del proyecto se ha llevado a cabo mediante la utilización de diferentes herramientas. En esta sección se va a explicar el hardware y software utilizado.

4.5.1. Hardware

Los elementos hardware que se han empleado para la realización del proyecto han sido:

- 1x Ordenador portátil
- Máquinas del servicio Cloud IBM (despliegue en producción solo)

4.5.2. Software

Los elementos software que se han utilizado han sido:

Lenguajes de programación

Durante el desarrollo del software del proyecto, se ha hecho uso de diversos lenguajes de programación como son:

- **NodeJS.** Utilizado para la programación dentro del framework Hyperledger Fabric.
- **Java.** Se ha utilizado Java para la parte BACKEND de la aplicación web. Es un lenguaje orientado a objetos y que es muy utilizado para el desarrollo web.
- **JavaScript.** Utilizado para la parte FRONTEND de la aplicación web. Es un lenguaje que se utiliza para la programación del cliente en la aplicación web.
- **CSS.** Hoja de estilos en cascada, sirve para personalizar los componentes del FRONTEND de la aplicación web.

- **HTML5.** Lenguaje de etiquetas utilizado para el desarrollo de páginas web. Actúa como “esqueleto” de la página y se ha utilizado en la parte FRONTEND de la aplicación web.

Sistemas operativos

El sistema operativo que ha sido utilizado para el desarrollo del proyecto ha sido **Windows 10 Pro**.

Aplicaciones y herramientas de desarrollo

En cuanto al software de desarrollo utilizado, estos han sido:

- **Google Chrome:** utilizado para la visualización y desarrollo de la parte FRONTEND.
- **Visual Studio:** utilizado para el desarrollo del contrato inteligente. Es un IDE que tiene soporte para muchos lenguajes de programación.
- **IntelliJ IDEA:** es un IDE optimizado para JAVA y utilizado durante el desarrollo del BACKEND.
- **Gradle:** Es una herramienta que sirve para compilar código abierto, utilizado para la compilación de la parte BACKEND.
- **Docker:** Software que sirve para automatizar el despliegue de aplicaciones dentro de los llamados contenedores de software, y ha sido utilizado para la ejecución del proyecto.
- **NPM:** es un sistema de manejo de paquetes para NodeJS.
- **Git:** sistema de control de versiones utilizado durante el desarrollo.
- **Fork:** software que proporciona una interfaz más amigable en cuanto a la gestión de control de versiones basado en Git.
- **MongoDB:** base de datos noSQL, en la que los datos son almacenados en formato JSON utilizada para funciones como la de guardar usuarios.
- **LATEX:** lenguaje utilizado para la creación de documentos, en nuestro caso científico, de una calidad alta y ha sido utilizado para la elaboración de este documento.

CAPÍTULO 5

Arquitectura

El Marketplace de NFTs desarrollado en este proyecto cuenta con una arquitectura modularizada, en el que cada módulo se comunica entre sí a través de la red. Cada uno de esos módulos está desarrollado de una forma diferente e independiente en cuanto a software se refiere, funcionando como sistemas independientes. Destacar que se pretende tener **una única red Blockchain**, y que la utilicen distintos Marketplace verticales. De esta forma, hemos podido dividir el problema en unos más pequeños, dando como resultado los siguientes módulos:

- **Módulo Marketplace.** Constituye el backend del vertical específico.
- **Módulo Api-gateway.** Es la interfaz de la Blockchain.
- **Módulo Hyperledger Fabric.** Es la red Blockchain construida con el framework Hyperledger Fabric.

A continuación, se deja un diagrama (ver Fig. 5.6) que muestra el diseño y arquitectura del proyecto, que se ha dividido en microservicios independientes que pueden ser acoplados junto a otros independientemente de cual sea su uso final, favoreciendo así la reutilización del código y minimizando el tiempo empleado en la corrección de posibles errores.

5.1. MÓDULO MARKETPLACE

Este módulo representa el backend del vertical específico y su función principal es la encargarse de **transformar las peticiones** de frontend a llamadas del gateway de la Blockchain.

Está construido usando el framework Java Spring Boot, bajo un patrón de diseño MVC (Modelo-Vista-Controlador) y se ha utilizado una base de datos MongoDB para la persistencia de los datos. La comunicación entre el front y gateway se realiza mediante API REST.

5.1.1. Desarrollo con patrón MVC: Modelo-Vista-Controlador

Se trata de un patrón de desarrollo software en el que el dominio (*Modelo*) se separa de la *Vista* mediante un elemento que se sitúa en medio que recibe el nombre de *Controlador*.

El usuario interactúa con el sistema por medio de una interfaz gráfica que forma parte de la vista y decide que es lo que quiere hacer, visualizar, etc. Cuando el usuario decide, la vista es la encargada de comunicarse con el controlador, y le reenviará el mensaje con la información al modelo, que procesará la solicitud y servirá la funcionalidad accediendo a un servicio de un tercero, consultando una base de datos, modificando variables, haciendo operaciones matemáticas, etc. Cuando se ha servido la funcionalidad, el modelo lo regresa al Controlador y este de nuevo a la vista para que el usuario final vea el resultado.

El proyecto desarrollado utiliza este patrón: el usuario por medio de la aplicación web, interacciona con el sistema, pidiendo un recurso, como el de ver todos los NFTs disponibles para comprar, después se envía un mensaje por HTTP al servidor, a los diferentes *endpoints*, que conformaría el Controlador.

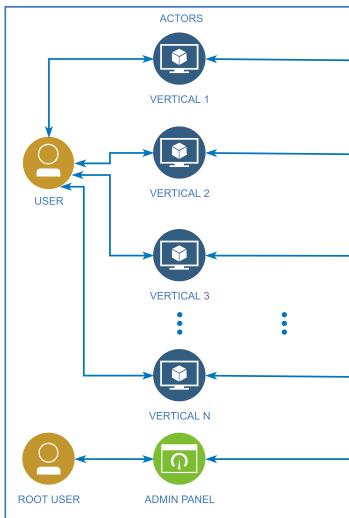


Figura 5.1: Diagrama del módulo Marketplace aislado

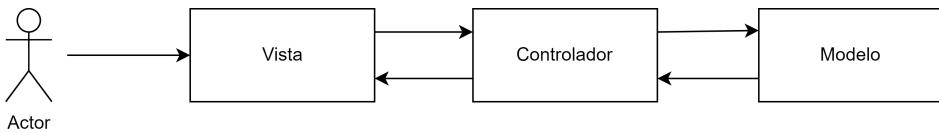


Figura 5.2: Estructura básica del patrón MVC

Después, las clases de la capa de dominio de la estructura del servidor, serían las que hacen la función de Modelo, que son las que se comunican por ejemplo con la base de datos MongoDB, quedando algo parecido a la estructura definida en la figura 5.3

5.1.2. Desarrollo con Java Spring Boot

Java Spring Boot es un framework de desarrollo de aplicaciones J2EE que se basa en la Inyección de dependencias¹. Está basado en una infraestructura no muy compleja, fácil de entender. Las aplicaciones desarrolladas son **autocontenidas**, lo que se traduce en mayor facilidad para desarrollar, teniendo que despreocuparse el programador de la arquitectura y focalizarse en el desarrollo. Una de las principales ventajas es que Spring Boot cuenta con un servidor de aplicaciones embedido, y por defecto es Tomcat aunque también se pueden utilizar otros.

Otra ventaja que tiene es la integración con los gestores de dependencias como Maven o **Gradle**, este último ha sido el utilizado en este proyecto. Para desplegar una aplicación Spring Boot, se puede hacer en tres sencillos pasos:

- (1) Crear un proyecto Maven o Gradle, y para ello, Spring nos facilita un sitio web (<https://start.spring.io/>) que se encarga de hacerlo por nosotros, pudiendo añadir desde el principio todas las dependencias que se deseen.
- (2) Se escribe el código de la aplicación.
- (3) Se ejecuta la aplicación y se despliega en un servidor.

En Spring, es muy frecuente el uso de notaciones, hay muchas y muy variadas, cada una con una función específica, pero nosotros nos centraremos en las 4 que considero más importantes y básicas.

¹En Java, la inyección de dependencias es un principio que nos permite usar las dependencias solo cuando se necesiten, en vez de inicializarlas dentro de la clase que las utiliza.

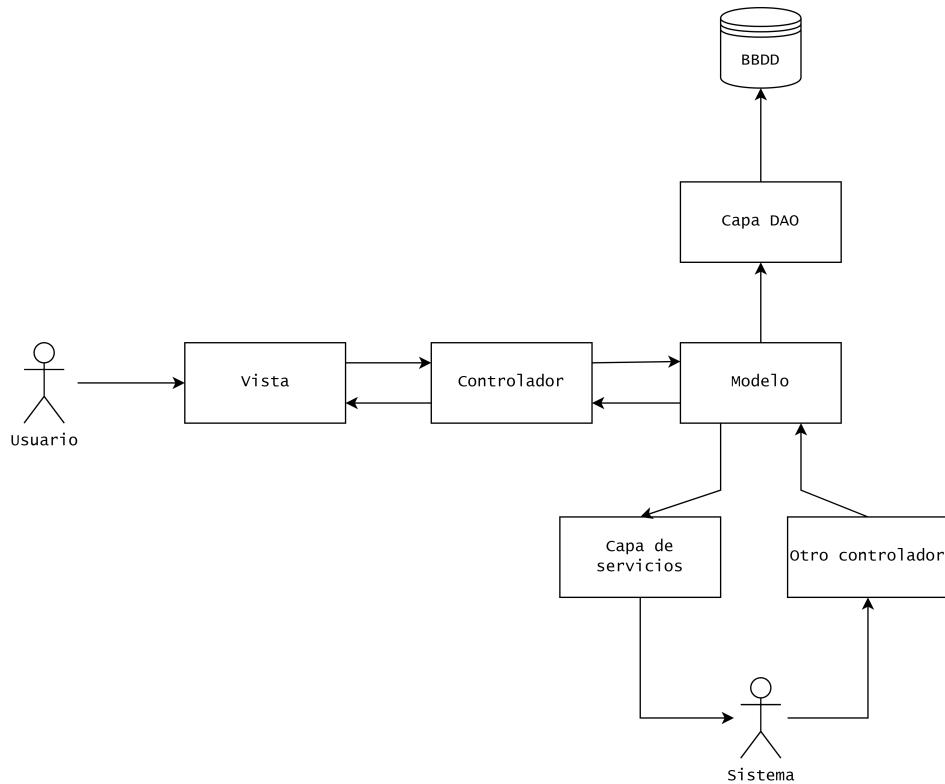


Figura 5.3: Diagrama de comunicación (desacoplada) con MVC de la aplicación.

@SpringBootApplication

Esta notación es la necesaria para arrancar una aplicación Spring, como se puede observar en el listado 5.1 la notación precede en este caso a la clase principal y dentro encontramos el método main, que llama al método estático run. Como primer parámetro se recibe la propia clase, que es **obligatorio** que esté notada con `@SpringBootApplication`.

Listado 5.1: Clase principal de un proyecto Spring Boot

```

1 import org.springframework.boot.SpringApplication;
2 import org.springframework.boot.autoconfigure.SpringBootApplication;
3
4 @SpringBootApplication //Notacion
5 public class WorldApplication {
6
7     public static void main(String[] args) {
8         SpringApplication.run(WorldApplication.class, args);
9     }
10 }
```

Si la ejecución ha sido exitosa, el método `run` devuelve un objeto del tipo `ConfigurableApplicationContext` el cual Spring utiliza para gestionar el ciclo de vida de la aplicación.

@Component

Las clases notadas con `@Component` son aquellas que Spring utilizará de forma automática cuando se requiera la inyección de dependencias o autocreación. Las clases `component` deben de tener un constructor vacío para que Spring pueda instanciarlas, de lo contrario, se nos lanzará una excepción y la aplicación se detendrá de ejecutar

@Autowired

Cuando a un elemento de una clase se le aplica esta notación, Spring recibe la orden de que ese campo se va a inicializar de forma automática mediante su constructor sin parámetros. Es una de las notaciones más utilizadas a la hora de trabajar con Spring.

Su uso más frecuente es de cuando tenemos un servicio y un repositorio, y queremos instanciar el repositorio en una o varias clases. A continuación se deja un ejemplo de utilización:

Listado 5.2: Creación de un repositorio en Java Spring Boot

```

1 import java.util.ArrayList;
2 import java.util.List;
3
4 import org.springframework.stereotype.Component;
5 import org.springframework.stereotype.Repository;
6
7 @Repository //Notación utilizada para declarar un repositorio
8 public class LibroRepository {
9     public List<Libro> buscarTodos() {
10         List<Libro> lista= new ArrayList<Libro>();
11         lista.add(new Libro ("1","java","Juanma"));
12         lista.add(new Libro ("2","python","Pablo"));
13
14     return lista;
15
16 }
17 }
```

Listado 5.3: Ejemplo de uso de @autowired en Java Spring Boot

```

1 import java.util.List;
2
3 import ↵
4     ↵ org.springframework.beans.factory.annotation.Autowired;
5 import ↵
6     ↵ org.springframework.web.bind.annotation.RequestMapping;
7 import ↵
8     ↵ org.springframework.web.bind.annotation.RestController;
9
10 @RestController //Notación para indicar que es un servicio REST
11 public class LibroRestService {
12     @Autowired //Notación @Autowired, se instancia el repositorio
13     LibroRepository repositorio;
14     @GetMapping("/libros") //Notación para peticiones GET de
15     HTTP
16     public List<Libro> buscarTodos() {
17
18         return repositorio.buscarTodos();
19
20     }
21 }
```

Como se ha podido observar en los listados 6.6 y 5.5 el repositorio es inicializado de forma automática debido a la notación `@Autowired`. Esto, en la práctica, se traduce como un ahorro de

tiempo y código para el desarrollar y para el proyecto.

@Controller y @RestController

La notación `@RestController` deriva de `Controller`. Este último dota a la clase de las características propias de un Controlador en el patrón MVC, la de comunicar el modelo con la vista. Los métodos que se anoten con esta notación van a poder escuchar peticiones HTTP.

Si utilizamos `@RestController` estamos indicando que ese controlador no va devolver una página, sino que va a **devolver objetos**. Un ejemplo de utilización lo podemos encontrar en el listado de la sección anterior (ver listado 5.5).

Nuestra aplicación funciona del mismo modo, el controlador está notado con `@RestController` y devuelve objetos. Cada *endpoint* está notado con una notación adicional, que marca el tipo de petición HTTP:

- `@GetMapping`: atiende única y exclusivamente peticiones GET.
- `@PostMapping`: atiende única y exclusivamente peticiones POST.
- `@DeleteMapping`: atiende única y exclusivamente peticiones DELETE.

Las respuestas de estos *endpoints* se darán en formato JSON:

Listado 5.4: Respuesta de error 400, login fallido del servidor

```

1  {
2      "response": {
3          "headers": {},
4          "body": "Bad credentials", -->Object body
5          "statusCodeValue": 400, -->Response code
6          "statusCode": "BAD_REQUEST" -->Response type
7      },
8      "code": null, -->Wrapper code
9      "message": null -->Message in case of error
10 }

```

Listado 5.5: Respuesta de login correcto por parte del servidor

```

1  {
2      "response": {
3          "headers": {
4              "Set-Cookie": [
5                  "cookie=eyJhbGciOiJIUzUxMiJ9.eyJpYXQiOjpbNSN8w; ↵
6                  ↵ Path=/api; Max-Age=86400; ↵
7                  ↵ Expires=Wed, 15 Jun 2022 ↵
8                  ↵ 11:17:17 GMT; HttpOnly"
9              ]
10         },
11         "body": {
12             "userId": "12345678M",
13             "name": "Prueba2Nombre",
14             "surname": "Prueba2Apellidos",
15             "email": "prueba2@prueba.com",
16             "postalCode": "13002",
17             "address": "C/ CallePrueba",
18             "city": "Madrid",
19             "province": "Madrid",
20         }
21     }
22 }

```

```

17     "country": "Spain",
18     "role": "USER",
19     "userImg": "",
20     "idImg": ""
21   },
22   "statusCodeValue": 200,
23   "statusCode": "OK"
24 },
25   "code": null,
26   "message": null
27 }

```

5.2. MÓDULO API-GATEWAY

Representa la interfaz de la red Blockchain. Su función principal es la de centralizar la entrada de peticiones a la cadena de bloques y gestionar su ejecución. Ha sido construido dividiendo el módulo en microservicios en lenguaje Node.js y la comunicación entre los microservicios es **asíncrona** habiendo utilizado para ello un bus de datos de mensajes con *Apache Kafka*.

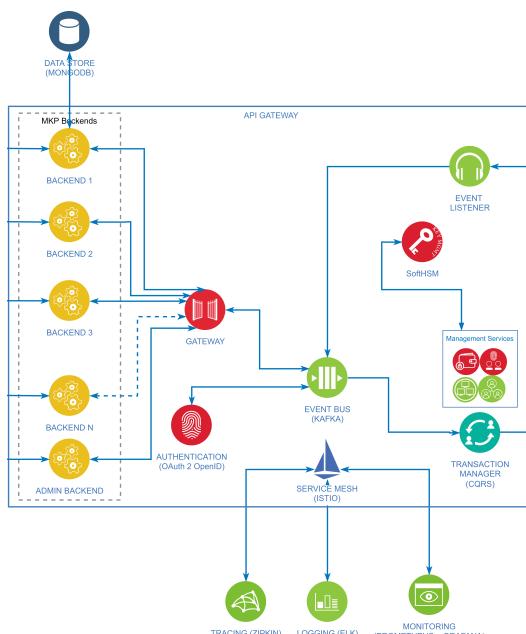


Figura 5.4: Diagrama del módulo API-gateway aislado

5.2.1. Gateway

Es un módulo de peticiones que se realiza mediante API REST y se encarga de centralizar todas las llamadas. Como se puede observar en la Figura 5.5 todas las peticiones pasan por este módulo.

5.2.2. Event-bus

Bus de mensajes implementado con Kafka para la comunicación entre los microservicios.

Apache Kafka es una plataforma distribuida utilizada para la transmisión de datos que permite publicar, almacenar y crear flujos de registro, además, los interesados se pueden suscribir a estos flujos de manera instantánea. Su estructura permite que se agrupen los mensajes que provienen de distintos orígenes o fuentes para después entregarlos a un destino o varios, es decir, hacer llegar ese mensaje a las partes interesadas, los suscriptores.

Nuestro proyecto utiliza microservicios que se comunican entre ellos por medio de Kafka y lo hacen de forma **asíncrona**. Kafka nos proporciona este tipo de integración necesaria para que los microservicios comparten los datos. Es una opción muy viable y rápida para integración asíncrona basada en eventos.

Se crea un *topic* por productor y consumidor, evitando que un servicio tenga rol de productor y consumidor a la vez, teniendo así tres diferentes *topics* o colas de mensajes que son:

- EventListenerWriteGatewayRead: el servicio de event-listener escribe en este *topic* y el servicio de gateway lo consume.
- TxManagerWriteGatewayRead: el servicio de transaction-manager escribe y la gateway lo consume.
- GatewayWriteTxManagerRead: mismo funcionamiento que los anteriores, el servicio de gateway escribe y el transaction-manager lo consume.

5.2.3. Transaction-manager

Encargado de gestionar las identidades de los usuarios, acceso al HSM y de enviar al Blockchain las peticiones de registro de usuarios en la red y la invocación de transacciones. Es capaz de invocar transacciones de consulta, recibir la respuesta de forma inmediata y enviarlas al *topic* correspondiente del servidor Kafka. También es capaz de agrupar y mandar un grupo de transacciones para que se ejecuten según la capacidad de la red blockchain (por ejemplo, 30 operaciones por segundo). En las llamadas de modificación del estado, al ser comunicación asíncrona, una vez enviada la petición continua con la siguiente, no se espera a la respuesta.

5.2.4. Event-listener

Encargado de recibir todas las respuestas a través de eventos de las operaciones que modifican el estado de la red y las envía al *topic* correspondiente del servidor de Kafka.

5.3. MÓDULO HYPERLEDGER FABRIC

Red Blockchain construida con Hyperledger Fabric sobre un clúster de kubernetes. Dentro de ella podemos destacar varios elementos como:

- Peers
- CAs
- Channel
- Organizations

A continuación, se explica con más detalle qué es cada componente.

5.3.1. Peers

Una red Blockchain se compone en su esencia de un conjunto de nodos. Estos nodos tienen asignados los contratos inteligentes y la información de la red. En Hyperdledger Fabric tenemos dos tipos de nodos: el **endorser peer** y el nodo **comitting peer**.

El endorser peer es el encargado de procesar las solicitudes de transacciones y manda el resultado firmado al sistema solicitante. El committing peer es el que cuando recibe un bloque de transacciones valida que que se han los requisitos de ese canal, y si es correcto, actualiza ese bloque en la red Blockchain.

Lo conveniente entonces es utilizar un número par de nodos, uno de cada tipo al menos, aunque depende de la organización que gestiona la Blockchain el número de nodos que se va a desplegar. ya

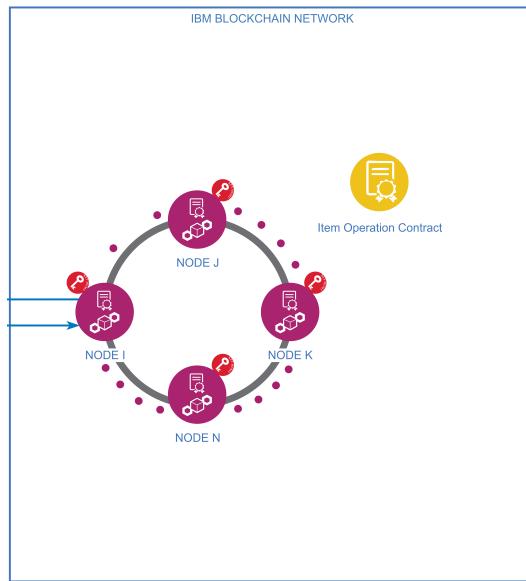


Figura 5.5: Diagrama del módulo Blockchain aislado

que en caso de que un *peer* falle y solo tenemos un par, podría suponer una pérdida de datos a la organización.

5.3.2. CAs

Son la autoridad de certificación por defecto de Hyperledger Fabric, es la encargada de emitir certificados a los participantes de la Blockchain. Primero emite un certificado raíz llamado *rootCert* a cada miembro y por último, se hace entrega de un certificado de inscripción *ECert* a cada usuario autorizado.

5.3.3. Channel y Organizations

Las organizaciones o miembros son entidades que tienen acceso a los canales (a los ledgers) y que dota de identidad a los participantes para que el origen o fuente de cada transacción sea identificable.

En el contexto de Hyperledger Fabric, channel o canal, se refiere a una comunicación privada entre dos o más participantes concretos de la Blockchain, podríamos decir que es una “subred” que se crea con el objetivo de realizar transacciones privadas y confidenciales. Un canal está definido por los siguientes elementos:

- Los miembros u organizaciones
- Los peers de cada miembro
- El ledger o libro compartido
- La Blockchain y su código

Cada transacción en la red se va a ejecutar sobre un canal diferente, en la que las partes implicadas deben estar autenticadas, autorizadas y verificadas para el canal concreto bajo el que se va a realizar la transacción.

5.3.4. Contrato inteligente

El contrato inteligente se ha desarrollado de acuerdo al estándar ERC-721 (ver Sec. 2.6), en Node.js e implementa las funciones definidas en los objetivos tales como:

- Aprobar cambios, devolver si un usuario posee autorización para un tarea, etc.

- Mintear (crear) un NFT, características del NFT, eliminar ese NFT, etc.
- Devolver información de un usuario, su saldo, los NFT que posee, historial de transacciones, etc.
- Devolver lista de todos los NFTs, filtrar en base a diferentes criterios como precio y estado, etc.

5.3.5. Docker y Kubernetes

La técnica de meter aplicaciones en contenedores ha ido en aumento, ya que nos permite desplegar de una forma rápida, sencilla y eficaz aplicaciones autocontenidoas. *¿Pero qué es una aplicación autocontenidoa?*

Cuando nosotros desplegamos de forma tradicional una aplicación, esta se despliega en una máquina como un ordenador físico, ya sea uno propio o uno de terceros como por ejemplo un servidor de un tercero. Cuando queremos ejecutar por ejemplo un programa en Python, necesitamos tener instalado Python en la versión correcta en nuestro ordenador así como todas las bibliotecas externas que hagan falta. Una aplicación autocontenidoa es aquella que ya cuenta con todos los requisitos que hacen falta para desplegarla y ejecutarla, por lo que se podría desplegar en cualquier máquina.

Docker es una plataforma de contenedores, en los que podemos añadir nuestra aplicación y desplegarla desde esos contenedores, junto con todas sus dependencias y requisitos, teniendo la posibilidad de convertir cada aplicación “tradicional” en una aplicación autocontenidoa.

La blockchain se desplegará en un clúster de kubernetes. Kubernetes es lo que se conoce como un orquestador de contenedores (por ejemplo contenedores Docker, usados en este proyecto). Fue creado por Google y actualmente es de código abierto.

El principal problema que resuelve es la gestión de un número elevado de contenedores, ofrece una elevada disponibilidad y es relativamente rápida.

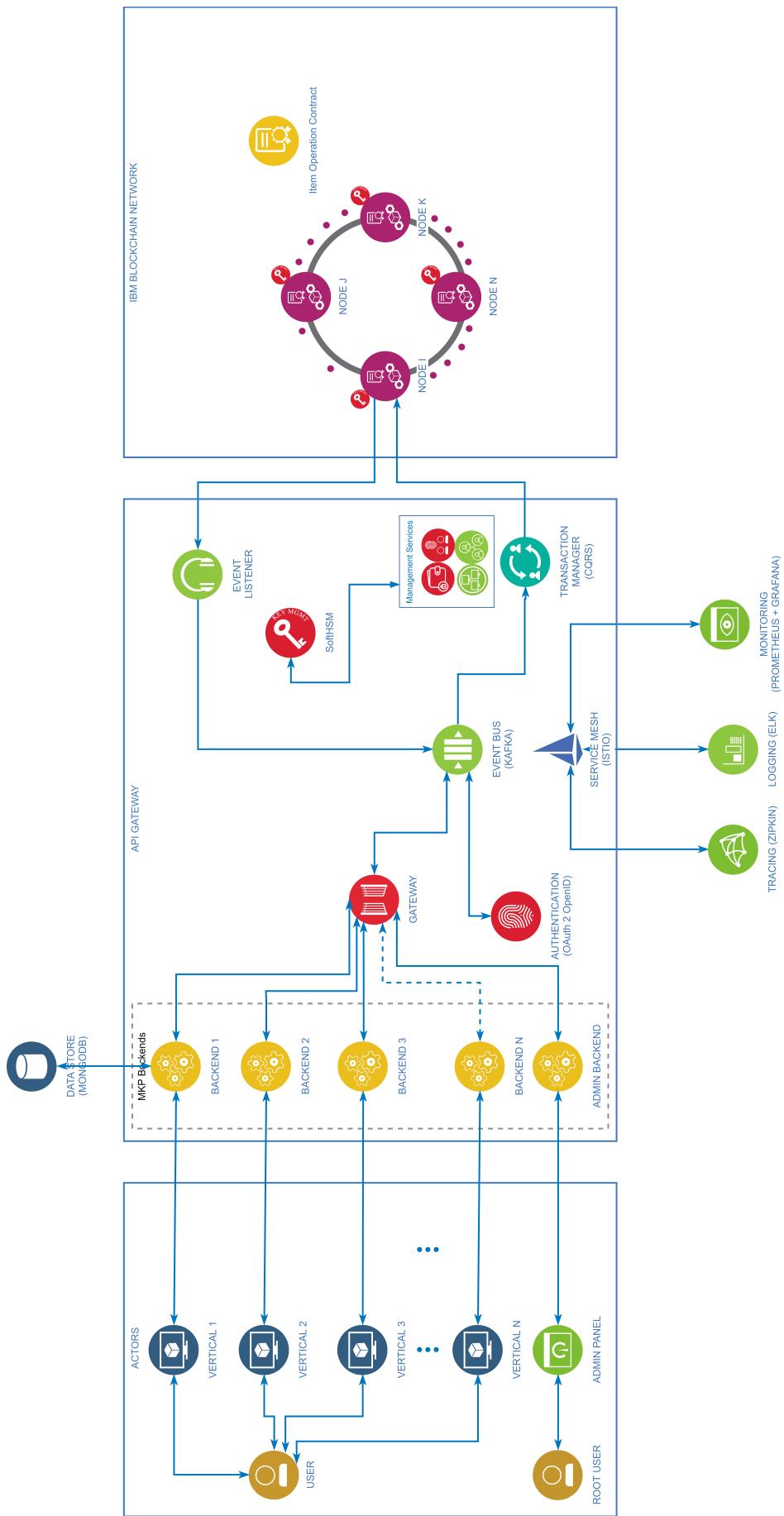


Figura 5.6: Arquitectura completa del proyecto

CAPÍTULO 6

Resultados

En este capítulo se introducirán los recursos y costes empleados. Se presentarán estadísticas del proyecto y algunas medidas de rendimiento que se han calculado junto a sus resultados. Para finalizar, se muestra el funcionamiento del sistema desarrollado dentro del proyecto mediante varios casos de usos.

6.1. ESTADÍSTICAS DEL REPOSITORIO

El código ha sido desarrollado utilizando una herramienta de control de versiones como es Git, que nos permite tener copias de seguridad del código de manera remota y local, pudiendo volver a un estado anterior en caso de ser necesario así como el seguimiento del código.

Se ha utilizado Fork como plataforma que aloja este código, haciendo uso de la consola de comandos de Windows para subir el código y realizar los *commits*. Teniendo el proyecto presentado de esta forma, se hace uso de la herramienta Cloc, para sacar diversas estadísticas de los repositorios, como número de líneas, número de archivos, líneas de código, lenguajes utilizados, etc.

Se ha creado un total de 7 repositorios, separados por función o servicios, cuyas estadísticas se muestran a continuación:

8 text files.				
8 unique files.				
2 files ignored.				

github.com/AlDanial/cloc v 1.82 T=0.10 s (69.2 files/s, 22645.1 lines/s)				

Language	files	blank	comment	code
JSON	3	0	0	2100
JavaScript	3	28	18	134
Dockerfile	1	2	0	10
SUM:	7	30	18	2244

Figura 6.1: Estadísticas repositorio API-Gateway 1/2

2 text files.				
2 unique files.				
2 files ignored.				

github.com/AlDanial/cloc v 1.82 T=0.03 s (38.1 files/s, 951.3 lines/s)				

Language	files	blank	comment	code
YAML	1	1	0	24

Figura 6.2: Estadísticas repositorio Event-Bus

```

9 text files.
9 unique files.
4 files ignored.

github.com/AlDanial/cloc v 1.82 T=0.16 s (37.1 files/s, 2355.2 lines/s)
-----
Language      files    blank   comment    code
-----
Bourne Shell      1       27     108       99
DOS Batch        1       21      2         66
Gradle           2       5      0         27
Java              2       8      0         18
-----
SUM:             6       61     110      210
-----
```

Figura 6.3: Estadísticas repositorio API-Gateway 2/2

```

13 text files.
13 unique files.
2 files ignored.

github.com/AlDanial/cloc v 1.82 T=0.16 s (74.1 files/s, 45744.8 lines/s)
-----
Language      files    blank   comment    code
-----
JSON            6       1      0       7084
JavaScript      5       59     25      202
Dockerfile      1       20      2       13
-----
SUM:            12      80     27      7299
-----
```

Figura 6.4: Estadísticas repositorio Event-Listener

```

20 text files.
20 unique files.
3 files ignored.

github.com/AlDanial/cloc v 1.82 T=0.31 s (60.6 files/s, 25194.9 lines/s)
-----
Language      files    blank   comment    code
-----
JSON            6       1      0       7136
JavaScript      12      174    171      409
Dockerfile      1       2      0       10
-----
SUM:            19      177    171      7555
-----
```

Figura 6.5: Estadísticas repositorio Transaction-Manager

```

21 text files.
21 unique files.
6 files ignored.

github.com/AlDanial/cloc v 1.82 T=0.21 s (77.8 files/s, 28879.8 lines/s)
-----
Language      files    blank   comment    code
-----
JSON            4       0      0       3158
JavaScript      12      827    159      1797
-----
SUM:            16      827    159      4955
-----
```

Figura 6.6: Estadísticas repositorio Smart Contract

6.2. PROTOTIPO CONSTRUIDO

Como se ha comentado en la sección anterior, se hará el despliegue de la aplicación mediante Docker, construyendo contenedores, para facilitar el despliegue y la “autocontención” de la aplicación. A

72 text files.				
72 unique files.				
4 files ignored.				
github.com/AlDanial/cloc v 1.82 T=1.29 s (53.3 files/s, 1878.6 lines/s)				

Language	files	blank	comment	code
Java	64	430	53	1854
Gradle	2	5	0	35
Markdown	1	7	0	17
Dockerfile	1	1	4	12
YAML	1	2	0	11
SUM:	69	445	57	1929

Figura 6.7: Estadísticas repositorio BackEnd aplicación web

continuación, se adjuntan los Dockerfile de cada una de las partes, después el archivo Docker-compose para el despliegue completo así como un archivo .env para configurar la Blockchain:

Listado 6.1: Dockerfile api-gateway node

```

1  FROM node:12.15.0
2  RUN mkdir apigateway-node
3  COPY ./apigateway-node
4  WORKDIR ./apigateway-node
5
6  RUN npm install --target=12.15.0 ←
    ↪ --target_platform=linux --target_arch=x64 ←
    ↪ --target_libc=glibc
7  ENV NODE_ENV staging
8  ENV PORT 8083
9  EXPOSE 8083
10
11 CMD npm rebuild
12 CMD npm run start

```

Listado 6.2: Dockerfile event-listener

```

1  FROM node:12.15.0
2  RUN apt-get update && apt-get install -y opensc openssl
3
4  # install libpkcs11-proxy.so
5  COPY ←
    ↪ --from=vegardit/softhsm2-pkcs11-proxy:develop-debian ←
    ↪ /usr/local/lib/libpkcs11-proxy* /usr/local/lib/
6
7  # install test TLS Pre-Shared Key
8  COPY ←
    ↪ --from=vegardit/softhsm2-pkcs11-proxy:develop-debian ←
    ↪ /opt/test.tls.psk /opt/test.tls.psk
9  RUN mkdir event-listener
10 COPY ./event-listener
11 WORKDIR ./event-listener
12 RUN npm install
13 ENV NODE_ENV development
14 ENV PORT 8082

```

```

15 EXPOSE 8082
16 RUN npm rebuild
17 CMD npm run development

```

Listado 6.3: Dockerfile transaction manager

```

1 FROM node:12.15.0
2 RUN mkdir transaction-manager
3 COPY ./transaction-manager
4 WORKDIR ./transaction-manager
5
6 RUN npm install --target=12.15.0 ←
    ↪ --target_platform=linux --target_arch=x64 ←
    ↪ --target_libc=glibc
7 ENV NODE_ENV development
8 ENV PORT 8081
9 EXPOSE 8081
10
11 RUN npm rebuild
12 CMD npm run development

```

Listado 6.4: Dockerfile Backend

```

1 FROM gradle:7.4.2-jdk17-alpine as build
2 # Only copy dependency-related files
3 COPY build.gradle settings.gradle /home/gradle/src/
4 WORKDIR /home/gradle/src
5 # Only download dependencies
6 RUN gradle --no-daemon clean build > /dev/null 2>&1 ←
    ↪ || true
7 # Copy all files
8 COPY --chown=gradle:gradle . /home/gradle/src
9 # Do the actual build
10 RUN gradle --no-daemon build
11
12 FROM amazoncorretto:17.0.3-alpine
13 WORKDIR /home/app
14 ENV TZ=Europe/Madrid
15 COPY --from=build ←
    ↪ /home/gradle/src/build/libs/nft-0.0.1-SNAPSHOT.jar ←
    ↪ /home/app/application.jar
16 EXPOSE 8080
17 ENTRYPOINT ["java", "-jar", "/home/app/application.jar"]

```

Listado 6.5: Archivo de configuración de la Blockchain

```

1 MICROFAB_CONFIG='{
2     "port":8443,
3     "endorsing_organizations": [ {"name": ←
        ↪ "Org1"}, {"name": "Org2"} ],
4     "channels": [ {"name": ←
        ↪ "mychannel", "endorsing_organizations": [ ←
        ↪ "Org1", "Org2"] } ]
5 }'

```

Listado 6.6: Docker-compose que despliega la aplicación completa

```
1  version: '3.8'
2
3  services:
4    blockchain:
5      image: ibmcom/ibp-microfab:0.0.11
6      container_name: "blockchain-network"
7      env_file:
8        - microfab-config.env
9      labels:
10        fabric-environment-name: "local-ibp Microfab"
11      ports:
12        - "8443:8443"
13      networks:
14        - mydockernet
15    nft-frontend:
16      container_name: "nft-frontend"
17      build: marketplace-nft-frontend/
18      ports:
19        - "3000:80"
20      networks:
21        - mydockernet
22    nft-backend:
23      container_name: "nft-backend"
24      build: marketplace-nft-backend/
25      ports:
26        - "8080:8080"
27      networks:
28        - mydockernet
29    zookeeper:
30      image: wurstmeister/zookeeper
31      container_name: zookeeper
32      ports:
33        - "2181:2181"
34      environment:
35        ZOOKEEPER_CLIENT_PORT: 2181
36      networks:
37        - mydockernet
38    kafka:
39      image: wurstmeister/kafka
40      depends_on:
41        - zookeeper
42      container_name: kafka
43      ports:
44        - "9092:9092"
45      environment:
46        KAFKA_ZOOKEEPER_CONNECT: 'zookeeper:2181'
47        KAFKA_LISTENERS: 'INTERNAL://:9092'
48        KAFKA_ADVERTISED_LISTENERS: ←
          ↵ 'INTERNAL://kafka:9092'
          ↵ KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: ←
```

```

      ↳ 'INTERNAL:PLAINTEXT',
50   KAFKA_INTER_BROKER_LISTENER_NAME: INTERNAL
51   KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: '1',
52   KAFKA_CREATE_TOPICS: ↳
      ↳ "EventListenerWriteGatewayRead:1:1,
53     TxManagerWriteGatewayRead:1:1,
54     GatewayWriteTxManagerRead:1:1"
55   KAFKA_ADVERTISED_HOST_NAME: 172.17.0.1 # change to
      172.17.0.1 if running on Ubuntu
56   KAFKA_MIN_INSYNC_REPLICAS: '1'
57   healthcheck:
58     test: ["CMD", "kafka-topics.sh", ↳
      ↳ "--bootstrap-server", "kafka:9092", ↳
      ↳ "--list"]
59     interval: 30s
60     timeout: 10s
61     retries: 10
62   networks:
63     - mydockernet
64   softhsm-server:
65     image: vegardit/softhsm2-pkcs11-proxy:latest
66     environment:
67       TOKEN_LABEL: securestack # define a custom token name
68       TOKEN_IMPORT_TEST_DATA: 0 # don't import test data
69       PKCS11_PROXY_TLS_PSK_FILE: ↳
          ↳ /mnt/config/pkcs11_proxy.psk # use a custom TLS
          pre-shared key
70     ports:
71       - 2345:2345
72     volumes:
73       - ./volumes/hsm-config:/mnt/config:ro      # mount
          config directory readonly
74       - ./volumes/hsm-data:/var/lib/softhsm:rw # mount
          data directory writable
75   deploy:
76     restart_policy:
77       condition: on-failure
78       delay: 5s
79   networks:
80     - mydockernet
81   api-gateway:
82     container_name: "api-gateway"
83     build: marketplace-apigateway-node/
84     environment:
85       PKCS11_PROXY_SOCKET: tcp://softhsm-server:2345
86     ports:
87       - "8083:8083"
88     depends_on:
89       kafka:
90         condition: service_healthy
91       softhsm-server:

```

```

92         condition: service_started
93     networks:
94         - mydockernet
95     transaction-manager:
96         container_name: "transaction-manager"
97     build: marketplace-transaction-manager/
98     environment:
99         PKCS11_PROXY_SOCKET: tcp://softthsm-server:2345
100        PKCS11_PROXY_TLS_PSK_FILE: /opt/test.tls.psk
101    ports:
102        - "8081:8081"
103    depends_on:
104        kafka:
105            condition: service_healthy
106        softthsm-server:
107            condition: service_started
108        networks:
109            - mydockernet
110    event-listener:
111        container_name: "event-listener"
112    build: marketplace-eventlistener-service/
113    environment:
114        PKCS11_PROXY_SOCKET: tcp://softthsm-server:2345
115        PKCS11_PROXY_TLS_PSK_FILE: /opt/test.tls.psk
116    ports:
117        - "8082:8082"
118    depends_on:
119        kafka:
120            condition: service_healthy
121        softthsm-server:
122            condition: service_started
123        networks:
124            - mydockernet
125    networks:
126        mydockernet:
127            driver: bridge

```

6.2.1. Explicación del archivo Docker-compose

El archivo más importante a comentar es el Docker-Compose, archivo en el que se recoge toda la configuración necesaria para automatizar el despliegue de la aplicación. Destacar que la aplicación está preparada para ser desplegada en un **entorno de producción**, pero para la presentación de este TFG se desplegará en una **red local**.

Como primer apunte, destacar que no se incluye un apartado de despliegue para la base de datos MongoDB, ya que esta es un servicio Cloud que no necesita ser desplegada mediante Docker porque ya es accesible por sí misma.

En todos los campos (Blockchain, Kafka, SoftHSM, etc) se define el atributo *network*, que sirve para desplegar los servicios en una red específica. En este caso nuestra red se llama **mydockernet** y todos los servicios se desplegarán en ella.

Blockchain

Lo primero que se despliega es la red Blockchain, encontramos su código en la línea 4, en la que se contiene una imagen de Microfab (que proporciona las funciones de API REST para interactuar con Hyperledger Fabric). En la línea 7 definimos el archivo de configuración (Listado 6.5) donde definimos **dos organizaciones y un canal**. Por último se le coloca una *label* y se definen los puertos.

Frontend

Lo segundo es el Frontend de la aplicación en la que únicamente se definen los puertos como parte destacable.

Backend

En el Backend tenemos lo mismo que en el Frontend, lo destacable son los puertos.

Api-Gateway

Lo más destacable es el uso de `depends_on` en la línea 88, la cual nos indica que este servicio depende de Kafka, y que debe de ser ejecutado cuando Kafka esté completamente operativo. También depende del servicio de SoftHSM, que cuenta con la condición `service_started` ya que no hay que esperar un tiempo adicional a que esté operativo, solo a que esté desplegado, SoftHSM es un servicio que se despliega de forma casi instatánea.

Zookeeper

Sirve para gestionar instancias de Kafka y nos permite levantar varias a la vez. Se define su imagen, desde la que se lanza, la variable de entorno del puerto por el que escucha y el puerto, así como el nombre del contenedor.

Kafka

Ya hemos hablado de Kafka en la parte de arquitectura, pero ahora podemos hacerlo de una manera más profunda viendo como podemos inicializar el servicio. A continuación se explican los atributos más importantes:

- `KAFKA_ZOOKEEPER_CONNECT`: El nombre del servicio zookeeper al que se conecta y el puerto.
- `KAFKA_LISTENERS`: Nombre del *listener* del servicio de Kafka y su puerto, y acepta cualquier host que se conecte, por la forma en la que está definida la dirección.
- `KAFKA_CREATE_TOPICS`: Los *topics* son colas de mensajes que se van a crear, en este caso hemos creado 3.
- `KAFKA_ADVERTISED_LISTENERS` : Los *listeners* accesibles.
- `KAFKA_LISTENER_SECURITY_PROTOCOL_MAP` : Se define el protocolo que se va utilizar para la conexiones del *listerner* que se indica, el cual es *plaintext*.
- `KAFKA_INTER_BROKER_LISTENER_NAME` : Nombre del *listener* del broker.
- `KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR` : se va a crear una única instancia del topic (dependiendo del valor que se le de, en este caso 1).
- `KAFKA_MIN_INSYNC_REPLICAS` : número de replicas necesarias del *topic* para que funcione correctamente.
- `KAFKA_CREATE_TOPICS` : Los *topics* son colas de mensajes que se van a crear, en este caso hemos creado 3, con factor de réplica de 1.

- **healthcheck**: cuando ejecutamos un archivo Docker-Compose, por defecto, todo se despliega a la vez. Esto puede suponer un problema si los servicios requieren de otros para funcionar correctamente, como en nuestro caso. Docker da por desplazado un servicio desde que lo ejecuta, pero a veces, estos servicios tardan un tiempo en estar operativos, es por ello que se define este **healthcheck**, que lo dará por válido cuando realmente lo esté. Dentro se define un intervalo de 30 segundos para levantar el servicio, con un tiempo límite de 10 segundos y 10 intentos.

SoftHSM

El servicio de SoftHSM cuenta con su imagen, TOKEN_LABEL define el nombre del token y PKCS11_PROXY_TLS_PSK_FILE el uso de un TLS personalizado y por último el puerto. Destacar también el uso de volúmenes, líneas 73 y 74, que hacen posible que si el servidor Docker se apaga, los datos de las claves privadas almacenadas sigan estando disponibles al encenderlo de nuevo.

Transaction Manager

Ubicado en la línea 95, se definen los puertos y su dependencia con el servicio de Kafka y SoftHSM.

Event Listener

Mismo procedimiento que en Transaction Manager.

6.3. COSTES

En esta sección se procede a realizar una estimación de costes del proyecto. El equipo de desarrollo está constituido por cuatro personas.

6.3.1. Coste de hardware

En la Tabla 6.1 se especifica el coste del Hardware empleado.

Tabla 6.1: Coste hardware.

COSTE HARDWARE			
Producto	Coste/unidad	Cantidad	Total
Ordenador portátil	350€	4	1400€
Cable de red	10€	4	40€
TOTAL: 1440€			

6.3.2. Coste de desarrollo

En la Tabla 6.2 se especifica el coste del Hardware empleado. Cada hora de trabajo cuesta 50€ y se ha realizado el cómputo de horas totales teniendo en cuenta que el horario laboral es de lunes a viernes, 8 horas al día, y excluyendo festivos. Desde febrero hasta junio, ambos incluidos.

Tabla 6.2: Coste hardware.

COSTE DESARROLLO			
Puesto	Cantidad	Horas	Total
Desarrollador backend	3(2 blockchain y 1 marketplace)	2496	124.800€
Desarrollador frontend	1 (marketplace)	832	41.600€
TOTAL: 166.400€			

6.3.3. Coste de software

Se detallan a continuación los gastos relacionados con el software, destacar que se computan como si la aplicación fuera desplegada en un entorno de producción y no local. En el entorno de producción el módulo SoftHSM se sustituirá por un módulo HSM físico de IBM Cloud, que lleva costes asociados. Además se usa el servicio de clúster de kubernetes de IMB Cloud.

A continuación se describe las características del servicio Cloud de IBM y su coste¹:

- 2 CAs
- 2 Peers
- 9.6 CPUs con 14.2GB de RAM y una capacidad de almacenamiento de 4040GB

El coste aproximado de estos servicios por mes es de: **3.591,78€**

6.3.4. Aspecto de la aplicación

A continuación, se adjuntan varias capturas de pantalla del aspecto que tendría la aplicación desarrollada. Recordar que por motivos de privacidad con el cliente, no se puede revelar el activo que se ha tokenizado, debido a que es un modelo de negocio nuevo y quieren preservar el factor “sorpresa” para el público. Lo mostrado a continuación es un aspecto aproximado, salvando características específicas del Marketplace.

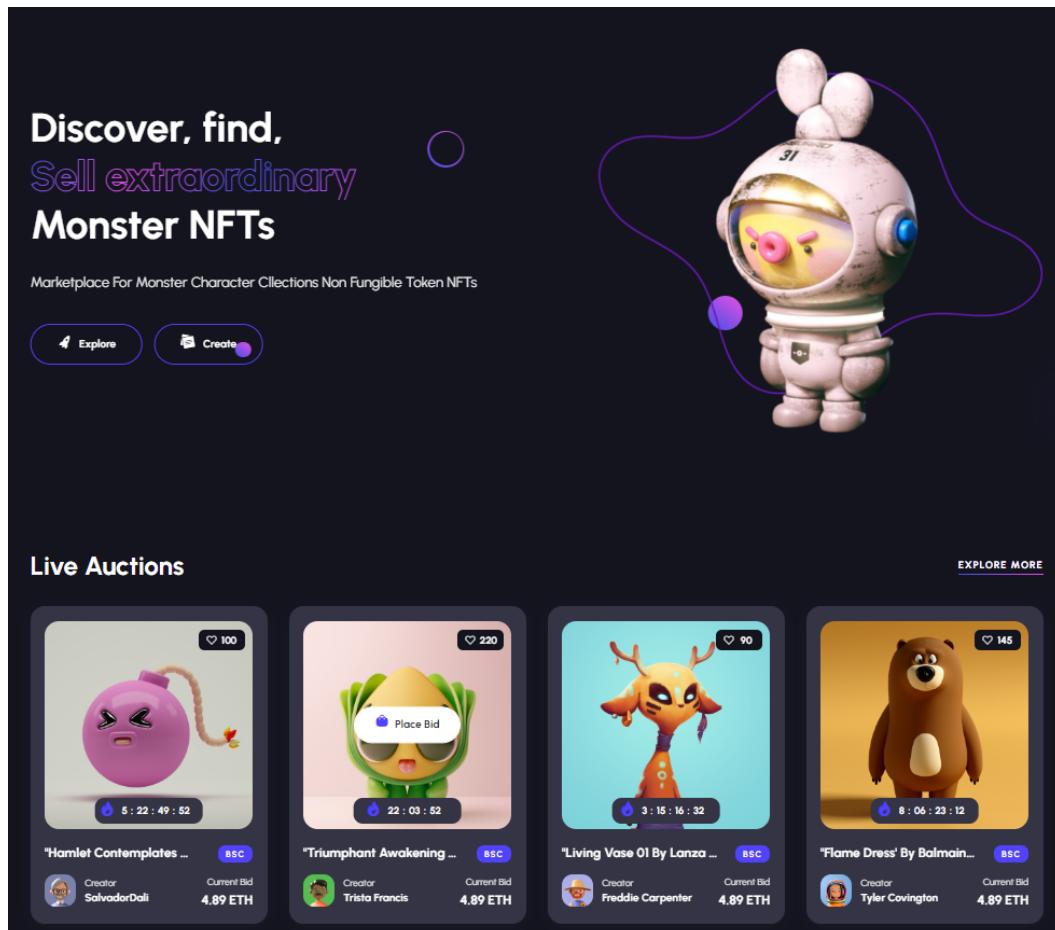


Figura 6.8: Página principal del proyecto

¹<https://cloud.ibm.com/docs/blockchain?topic=blockchain-ipb-detailed-pricing>

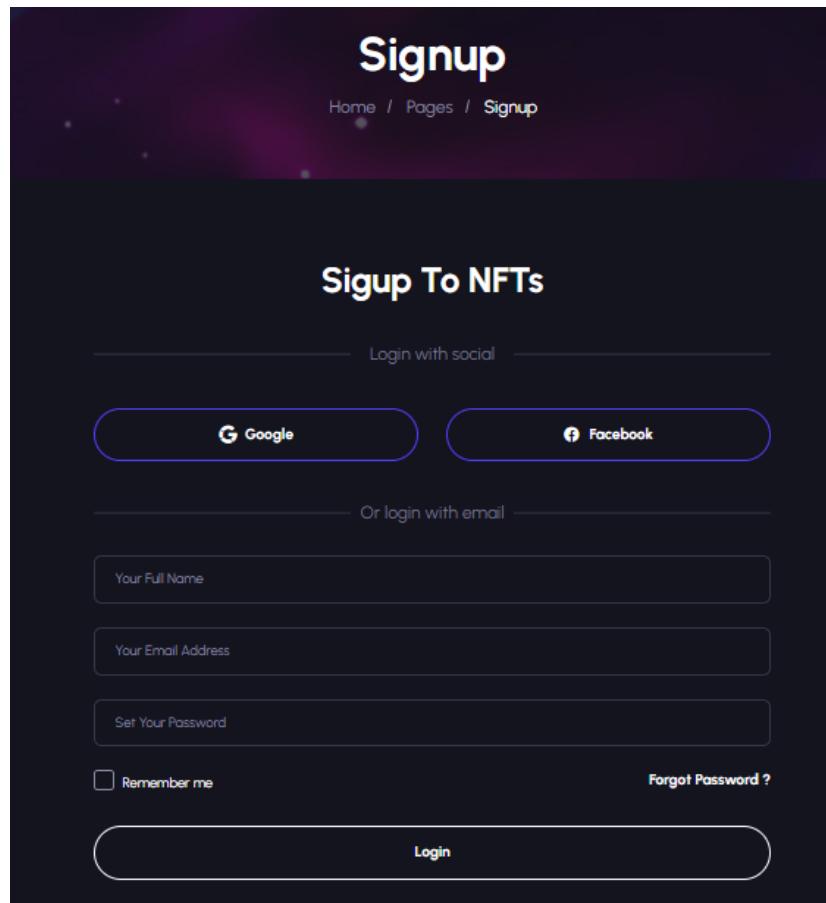


Figura 6.9: Página y formulario de registro

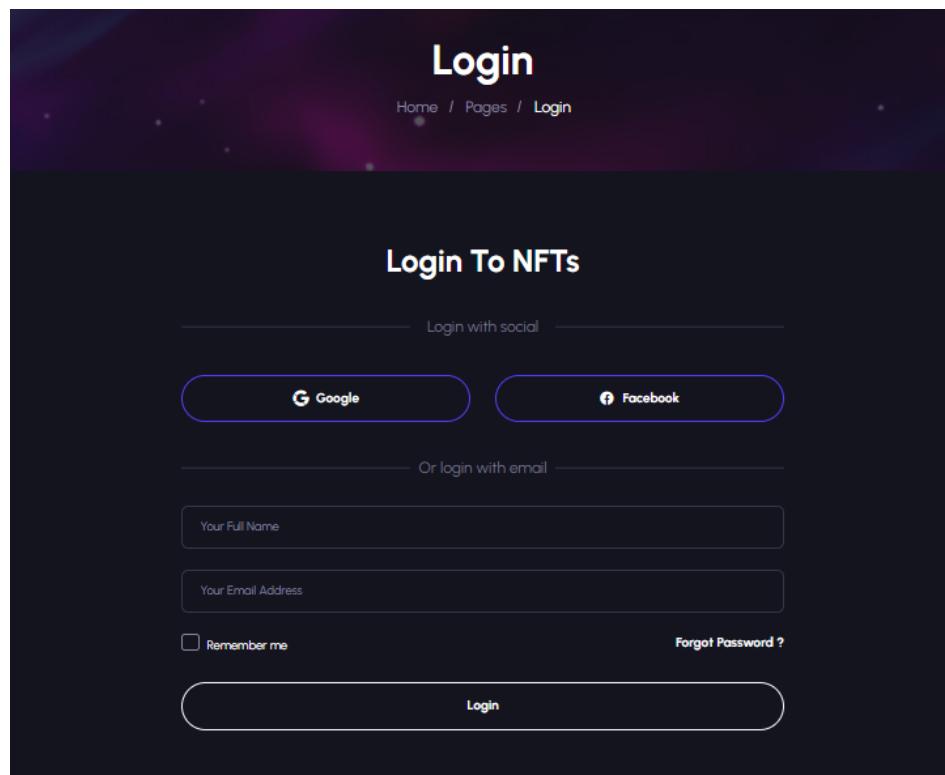


Figura 6.10: Página y formulario de login

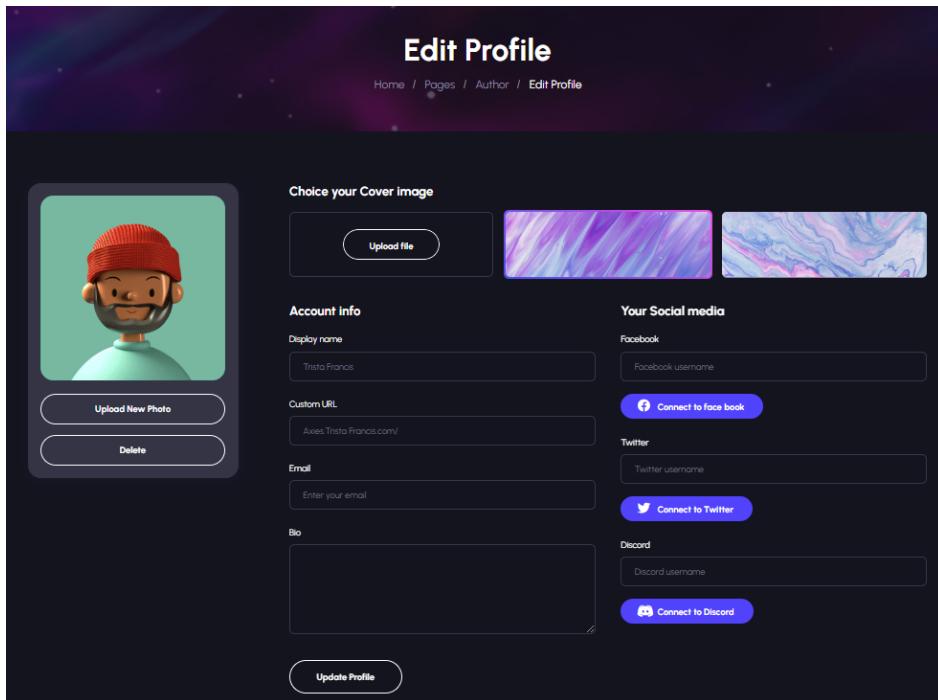


Figura 6.11: Edición de perfil

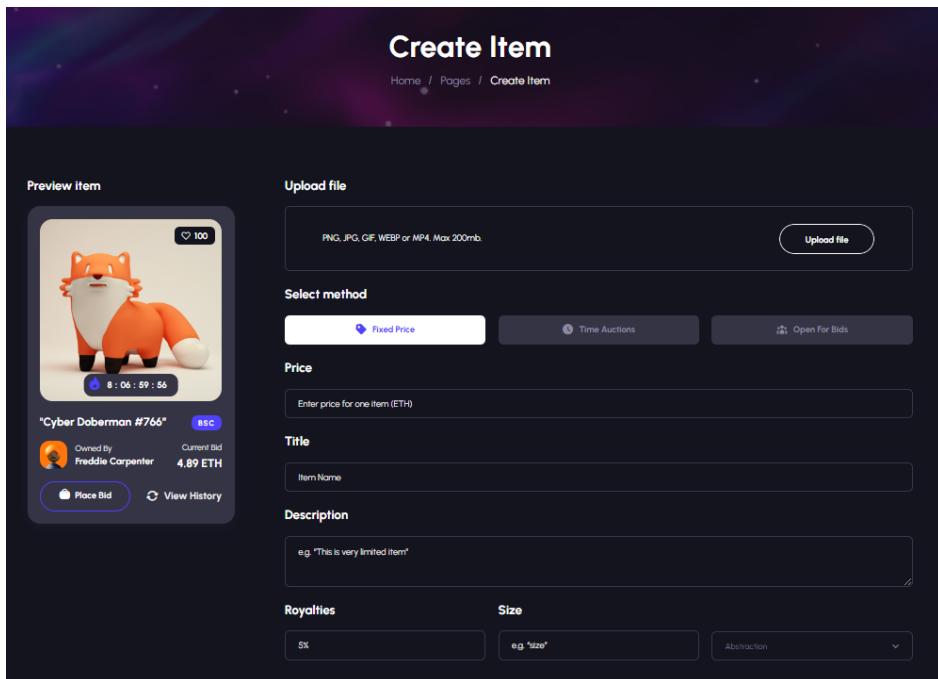


Figura 6.12: Formulario de creación de un NFT

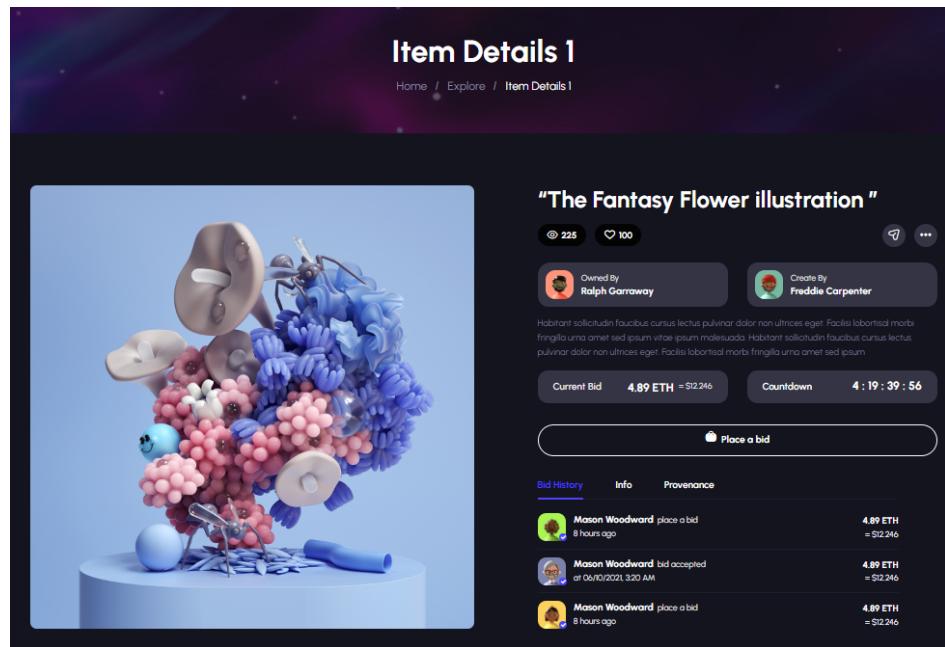


Figura 6.13: Características de un NFT

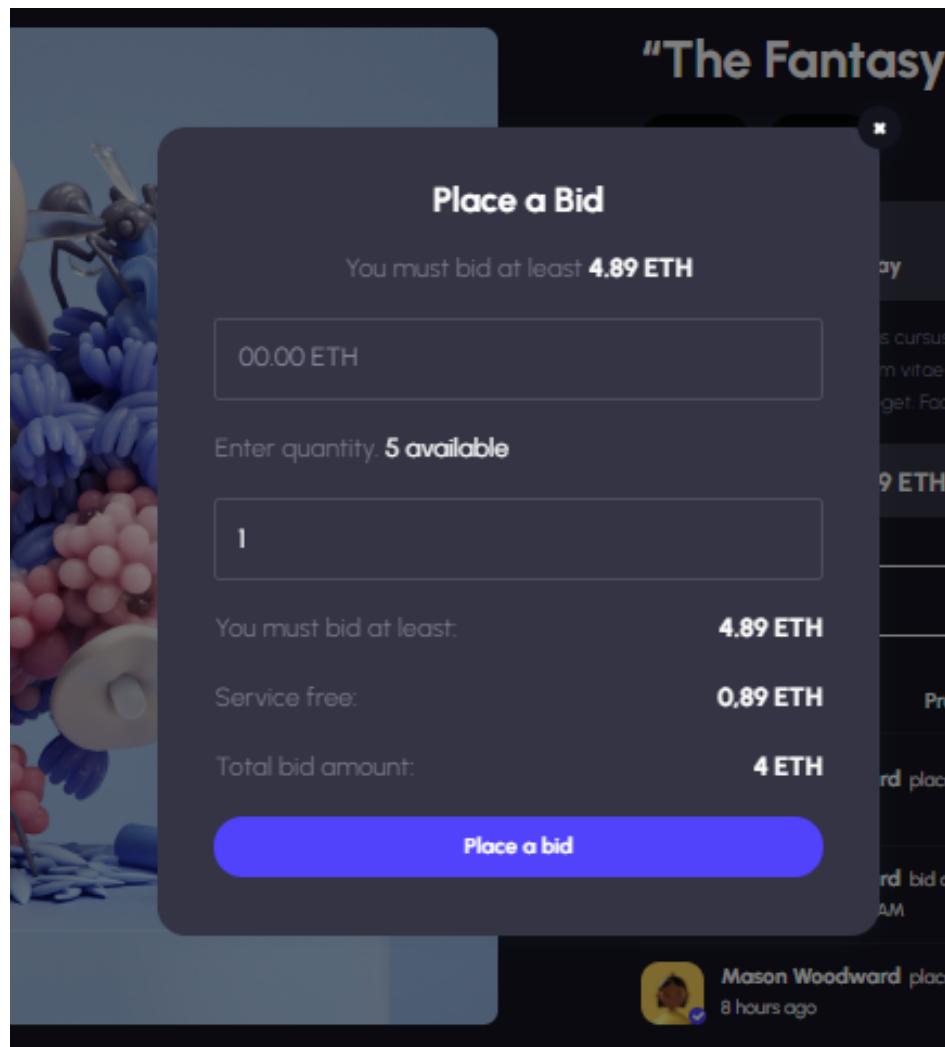


Figura 6.14: Pantalla de puja por un NFT

CAPÍTULO 7

Conclusiones

En este capítulo se realizará un juicio crítico y discusión sobre los resultados obtenidos. Primero se comentan los objetivos que se han logrado, después se comentan las competencias que se han adquirido y su correspondiente justificación, y por último se trata el aspecto de los trabajos futuros.

7.1. OBJETIVOS CUMPLIDOS

El objetivo general que se expuso en el Capítulo 3 se ha satisfecho. Se ha logrado la el desarrollo y construcción de un sistema Marketplace donde se gestionan bienes, en este caso bienes reales, que se tokenizan (NFT) y exponen para su compra-venta en el sitio.

Se ha cumplido por lo tanto con los siguientes objetivos específicos:

- El sistema permite al usuario visualizar el Marketplace por medio de una aplicación web, así como utilizar sus funcionalidades.
- Se ha creado una red Blockchain con Hyperledger Fabric que permite la creación segura y eficaz de bloques y transacciones, siendo esta una red híbrida, en la que participan miembros autenticados y autorizados.
- Se ha hecho uso del módulo SoftHSM para la custodia de forma segura de las claves privadas de los participantes de cadena de bloques, y se ha procedido a su integración mediante una librería para Hyperledger Fabric en lenguaje Node.js
- Se ha creado un contrato inteligente que permite realizar las acciones definidas en el Capítulo 3 en cuanto al propio token ERC-721, custodio, trading, usuario y el objeto NFT.
- La arquitectura sobre la que se ha basado el sistema está lo suficientemente modularizada y formada a su vez por microservicios que facilitan la gestión del proyecto a lo largo de todo el ciclo de vida.
- Las tecnologías utilizadas son de código libre y su uso no implica el pago adicional de ningún tipo (salvo en modo producción).
- Su despliegue es claro y conciso, es decir, todo lo necesario se aloja en contenedores Docker y el clúster de Kubernetes donde se encuentran todo lo necesario (bibliotecas, dependencias) en su versión correcta, haciendo del proyecto una aplicación autocontenido.

7.2. COMPETENCIAS ADQUIRIDAS

En esta parte se tratarán las competencias que se han adquirido durante el tiempo que el proyecto ha sido desarrollado. Se incluye el código de la competencia así como su justificación, en el Anexo A se puede obtener información acerca de estas competencias.

Tabla 7.1: Competencias adquiridas y justificación.

Competencia	Justificación
TI1	El desarrollo del proyecto ha llevado consigo la comprensión de lo que el cliente interesado pedía, para ello el tutor FORTE, Manuel Hervás Ortega, se encargaba de transmitir esa información, escuchando al interesado y elaborando los requisitos.
TI3	Durante el tiempo de desarrollo del TFG se ha usado una metodología ágil como Scrum, proporcionando así una relación cercana y continua con el cliente, desarrollando un sistema accesible, ergonómico y usable para los usuarios.
TI4	El despliegue de la aplicación se ha realizado en una red local, medio por la que los diferentes componentes del sistema se conectan o comunican entre sí. El autor de este TFG ha sido el responsable de diseñar, administrar y gestionar la red e infraestructuras de comunicaciones (como el despliegue en Docker y Kuberneete o la comunicación por Kafka)
TI6	El sistema desarrollado en este TFG es un módulo Marketplace al que se accede y funciona por medio de Internet en forma de página web que contiene archivos multimedia como imágenes de los NFTs y que permite la compra-venta de los mismos, cumpliendo con la función de comercio electrónico.

7.3. TRABAJOS FUTUROS

El sistema Marketplace es un sistema robusto, dividido en tres módulos, y cada uno de ellos dividido a su vez en microservicios independientes, desacoplados, que se comunican entre ellos independientemente de la arquitectura o lenguaje del microservicio destino. Esta arquitectura favorece la reutilización del código y minimiza los tiempos de detección de errores y su correspondiente rectificación.

Debido a esta arquitectura, se puede desarrollar un Marketplace a medida según las necesidades concretas del cliente. Al basarnos en una arquitectura desacoplada, se puede adaptar el Marketplace a todo tipo de productos, para elaborar un entorno de compra-venta de producto de nicho (por ejemplo, si el cliente es una tienda de arte digital, si el cliente es una empresa de zapatillas de colección) es decir, a partir del código base desarrollado en este TFG, es posible adaptar en un futuro este sitio así como la Blockchain para almacenar todo tipo de datos y desplegar varios Marketplace de nicho.

Referencias

- [1] Metodologías ágiles más usadas. URL: http://www.versionone.com/pdf/2011_State_of_Agile_Development_Survey_Results.pdf, 2022.
- [2] Kent Beck. Principios del manifiesto Ágil. URL: <http://agilemanifesto.org/iso/es/principles.html>, 2001.
- [3] BNB. Sitio web oficial. URL: <https://www.binance.com/es/bnbs>.
- [4] Cardano. Sitio web oficial. URL: <https://cardano.org/>.
- [5] Corda. Sitio web oficial. URL: <https://www.corda.net/>.
- [6] Eduardo Duarte. Introducción – scrum desde el sprint 1 (parte 1). URL: <https://eduardegallardo.medium.com/scrum-desde-el-sprint-1-parte-1-500c2d7abc11>, 2019.
- [7] Javier Jiménez en RedesZone. En qué consiste una red descentralizada y qué ventajas tiene. URL: <https://www.redeszone.net/tutoriales/redes-cable/que-son-redes-descentralizadas/>, 2022.
- [8] Paco Rodríguez en Xataka. Métodos para compartir archivos y contenidos en internet (iii): P2p, el auténtico espíritu del file sharing. URL: <https://www.xatakamovil.com/conectividad/metodos-para-compartir-archivos-y-contenidos-en-internet-iiip2p-el-autentico-espiritu-del-file-sharing>, 2012.
- [9] Equisoft. *La cadena de bloques una tecnología disruptiva con el poder de revolucionar el sector financiero*. Equisoft soluciones innovadoras, 2017.
- [10] Ethereum. Sitio web oficial. URL: <https://ethereum.org/es/>.
- [11] Hyperledger Fabric. ¿qué es hyperledger fabric? URL: <https://www.ibm.com/es-es/topics/hyperledger>.
- [12] Litecoin. Sitio web oficial. URL: <https://litecoin.org/es/>.
- [13] Metamask. Sitio web oficial. URL: <https://metamask.io/>.
- [14] Satoshi Nakamoto. *Bitcoin: un sistema de dinero en efectivo electrónico P2P*. First edition, 2008.
- [15] Solana. Sitio web oficial. URL: <https://solana.com/es>.
- [16] Solidity. Sitio web oficial. URL: <https://solidity-es.readthedocs.io/es/latest/introduction-to-smart-contracts.html>.
- [17] Statista.com. En qué consiste una red descentralizada y qué ventajas tiene. URL: <https://es.statista.com/estadisticas/1236504/bitcoin-historial-de-precios/>, 2022.
- [18] Mario G. Piattini Velthuis. *Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*. RA-MA S.A. Editorial y Publicaciones, first edition, 2007. ISBN: 8478977767.
- [19] XRP. Sitio web oficial. URL: <https://ripple.com/>.
- [20] William Stallings y Lawrie Brown. *Computer Security, principles and practice*. Financial Times Prentice Hall, second edition, 2014. ISBN: 0133773922.
- [21] Marcos Allende López y Vanessa Colina Unda. *Blockchain, como desarrollar confianza en entornos complejos para generar valor de impacto social*. Banco Interamericano de Desarrollo, 2018.

- [22] Hong-Ning Dai Zibin Zheng. *Blockchain challenges and opportunities: A survey*. International Journal of Web and Grid Services, 2018.

ANEXOS

ANEXO A

COMPETENCIAS DE TECNOLOGÍAS DE LA INFORMACIÓN

Competencias de la intensificación de Tecnologías de la Información, extraídas de la dirección:
<http://www.esi.uclm.es/www/josalido/downloads/CompetenciasIntensificaciones.pdf>

- TI1** Capacidad para comprender el entorno de una organización y sus necesidades en el ámbito de las tecnologías de la información y las comunicaciones.
- TI2** Capacidad para seleccionar, diseñar, desplegar, integrar, evaluar, construir, gestionar, explotar y mantener las tecnologías de hardware, software y redes, dentro de los parámetros de coste y calidad adecuados.
- TI3** Capacidad para emplear metodologías centradas en el usuario y la organización para el desarrollo, evaluación y gestión de aplicaciones y sistemas basados en tecnologías de la información que aseguren la accesibilidad, ergonomía y usabilidad de los sistemas.
- TI4** Capacidad para seleccionar, diseñar, desplegar, integrar y gestionar redes e infraestructuras de comunicaciones en una organización.
- TI5** Capacidad para seleccionar, desplegar, integrar y gestionar sistemas de información que satisfagan las necesidades de la organización, con los criterios de coste y calidad identificados.
- TI6** Capacidad de concebir sistemas, aplicaciones y servicios basados en tecnologías de red, incluyendo Internet, web, comercio electrónico, multimedia, servicios interactivos y computación móvil.
- TI7** Capacidad para comprender, aplicar y gestionar la garantía y seguridad de los sistemas informáticos.