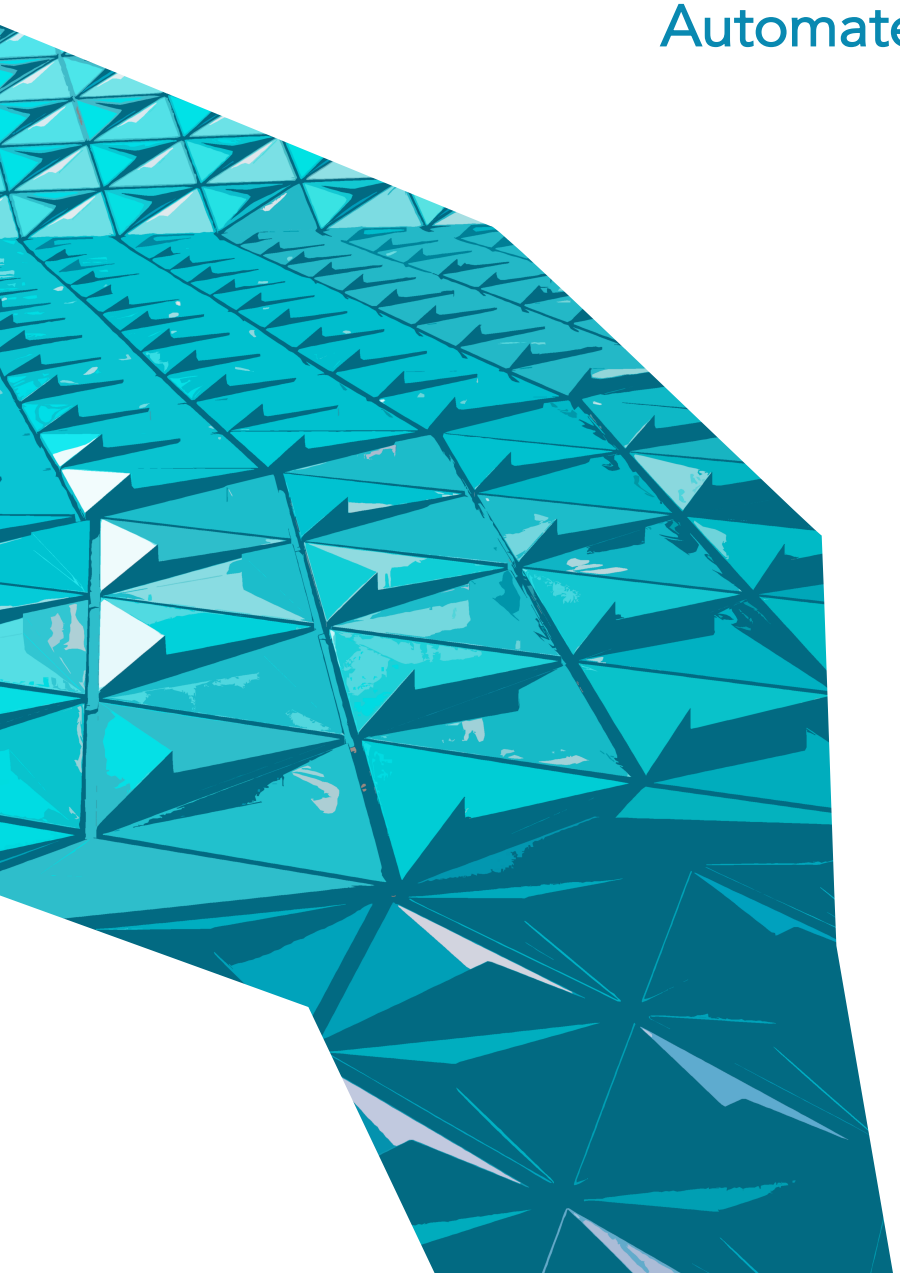
UNIVERSIDAD POLITÉCNICA DE MADRID

Escuela Técnica Superior de
Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Blockchain y Contratos Inteligentes. Oráculo de Precios para Automated Market Makers



Autor: Álvaro Jiménez Mesa
Tutor: Sergio Ríos Aguilar

Madrid, junio 2022

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática
Título: Blockchain y Contratos Inteligentes. Oráculo de Precios para Automated Market Makers.
Junio 2022

Autor: Álvaro Jiménez Mesa

Tutor: Sergio Ríos
Department of Organization Engineering, Business Administration and Statistics
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

¿Qué sabemos acerca de las tecnologías Blockchain? Resulta fácil hablar de la revolución de Bitcoin sin conocer realmente lo que está sucediendo detrás de esta innovación, y es que no podemos separar el fenómeno financiero de la tecnología que lo soporta. Por este camino, descubrimos que la red Ethereum es la plataforma que dio a luz a las famosas Finanzas Descentralizadas. Este soporte distribuido ha generado un entorno seguro, democrático y eficiente que, con la automatización de los Contratos Inteligentes, ha sustentado el desarrollo de los mercados descentralizados.

Frente a tanta transformación, debemos ser conscientes de los riesgos, limitaciones y debilidades de las tecnologías Blockchain, especialmente en los inicios de una propuesta que aún no ha alcanzado madurez. Se recorrerán los vectores fundamentales de innovación, desde las bases de la arquitectura Blockchain hasta los Contratos Inteligentes en Ethereum, plateando casuísticas que permitan concebir el valor de esta innovación de manera pragmática.

El ecosistema de las Finanzas Descentralizadas ha propiciado un Comercio de Alta Frecuencia que juega con la alta volatilidad de los valores que se operan en estos mercados. Para los agentes involucrados en estas actividades, resulta un reto conocer información de precios de las diferentes plataformas del ecosistema. Se consumará el proyecto con el desarrollo práctico de un Oráculo de Precios, una aplicación que proveerá información fiable, inmediata, actualizada y exacta sobre los precios ofertados para los activos soportados en diferentes protocolos del entorno. La utilidad que ofrece esta API podrá ser potencialmente integrada con estrategias como el arbitraje en DeFi.

Palabras Clave: Blockchain, Ethereum, DeFi (Decentralized Finance), AMM (Automated Market Maker), Contratos Inteligentes, Oráculo, Solidity.

Abstract

What do we *really* know about Blockchain? Talking about the Bitcoin revolution without really knowing what is happening behind this innovation is common. Truth is that we cannot separate the financial phenomenon from the technology that supports it. Ethereum is the network that has given birth to the well-renowned system of Decentralized Finance. The creation of a secure, democratic and efficient environment for decentralized markets has only been possible thanks to the automation of Smart Contracts.

In the face of so much transformation, we must be aware of the risks, limitations and weaknesses of Blockchain technologies, especially in the early stages of a proposal that has not yet reached its maturity. We will go through the fundamental vectors of innovation, from the foundations of Blockchain architecture to Smart Contracts in Ethereum, presenting case studies that allow us to conceive the value of this innovation in a pragmatic way.

The Decentralized Finance ecosystem has fostered High Frequency Trading that exploits with the high volatility of the securities traded in these markets. For the agents involved in these activities, it is a challenge to keep up with price information from the different platforms of the framework. We will conclude the project with the development of a *Price Oracle*, an application that will provide reliable, immediate, updated and accurate information of prices disposed in different protocols of the environment for numerous assets. The utility offered by this API could potentially be integrated with strategies such as arbitrage in *DeFi*.

***Key Words**: Blockchain, Ethereum, DeFi (Decentralized Finance), AMM (Automated Market Maker), Smart Contracts, Oracle, Solidity.*

Índice de Contenidos

1 Introducción 6

1.1 Contexto y Motivación 6

1.2 Objetivo y Alcance 8

1.3 Estructura del documento 9

2 Comprendamos la Tecnología Blockchain 10

2.1 Web 3, punto, ¿qué? 10

2.2 Fundamentos del Blockchain 12

2.3 Arquitectura Esencial 13

I. Transacciones 13
II. Cifrado Asimétrico 13
III. Registro 14
IV. Funciones Hash 14
V. Bloques y encadenamiento 15
VI. Tipos de Blockchain 16
VII. Modelos de Consenso 17
VIII. Conflictos en el Blockchain y Forking 19

2.4 Aplicaciones de la Tecnología Blockchain 22

2.5 Debilidades, Limitaciones y Retos del Blockchain 28

I. ¿Inmutabilidad? 28
II. Asimetría de poder en la red 29
III. La desunión entre lo real y digital 29
IV. Actuaciones maliciosas y Ciberseguridad 30

3 Contratos Inteligentes 32

3.1 "Smart" + "Contracts" 32

3.2 Ethereum y otras redes 34

I. Bitcoin 34
II. Hyperledger Fabric 35
III. Corda 35
IV. Ethereum y el inicio de las DeFi 35

3.3 Los Contratos Inteligentes también están limitados 40

4 Desarrollo de la Contribución: Creación de Oráculo de Precios de *Automated Market Makers* con Contratos Inteligentes 42

4.1 ¿Qué se propone? 42

4.2 Recursos y herramientas 45

I. Apoyo en los primeros pasos 45
II. Entornos de desarrollo 49
III. Despliegue en la red 51

4.3 Implementación 54
I. Análisis de requisitos 54
II. Diseño del sistema 55
III. Solución en Contratos Inteligentes 57
4.4 Testeo y Simulación 67
Tests unitarios 67
Tests e2e 67
Resultados del test y simulación 70
5 Conclusión y trabajos futuros 72
6 Análisis de impacto 75
7 Referencias 77
8 Anexos 81
8.1 Anexo 1: Tests Unitarios 81
8.2 Anexo 2: Procedimiento de Instalación 87

Índice de Figuras

Figura 1: Esquema simplificado de red en Web 3.0 versus modelos previos [1]. 11
Figura 2: Encadenamiento de Bloques 16
Figura 3: Conflicto entre dos versiones de Blockchain. Se decanta por B porque es la cadena más larga. 20
Figura 4: Perspectiva general de estructura del Blockchain [14]. 21
Figura 5: Representación de transacciones en Bitcoin. La T1 dirige a otras cuentas 5 BTC y retiene los otros 5 BTC que suman la entrada de 10 BTC [25]. 34
Figura 6: Valor total bloqueado en Contratos Inteligentes en la red Ethereum para DeFi [21]. 38
Figura 7: La pila DeFi [21]. 39
Figura 8: Interfaz de Remix IDE [38]. 48
Figura 9: Interfaz de la extensión web de Metamask en Google Chrome. 49
Figura 10: Punto de acceso generado por Infura para la mainnet de Ethereum. 53
Figura 11: Interfaz de Etherscan. 53
Figura 12: Visión general del diseño del sistema. 56
Figura 13: Código de Oracle.sol 60
Figura 14: WrapperUniswapV2.sol 61
Figura 15: WrapperSushiswap.sol 62
Figura 16: WrapperUniswapV3.sol 62
Figura 17: WrapperCurveFi.sol 63
Figura 18: Wrapper1Inch.sol 63
Figura 19: WrapperChainlink.sol 64
Figura 20: IOracle.sol 64
Figura 21: IEvents.sol 64
Figura 22: UniV2Pair.sol 66
Figura 23: IQuoter.sol 66
Figura 24: IRegistry.sol 66
Figura 25: ICryptoPool.sol 66
Figura 26: I1Split.sol 66
Figura 27: e2e/oracle.specs.ts 69
Figura 28: unit/oracle.spec.ts 82

1 Introducción

Para introducir este proyecto, se hace un breve acercamiento para que el lector pueda entender la dirección y el alcance de este trabajo. Se hablará de la motivación, objetivos y estructura del documento.

1.1 Contexto y Motivación

Se comienza ubicando el proyecto en un marco específico: está bastante extendido el desconocimiento acerca de la tecnología Blockchain. Yo mismo, como estudiante de Ingeniería Informática, desconocía el potencial de esta tecnología. ¿Cómo va a conocer el público promedio acerca de ello si yo, como estudiante del sector tecnológico, no tengo ninguna noción al respecto?

Esta desinformación general se debe a que Blockchain es bastante novedoso, y aunque su origen se remonta a hace más de una década, es ahora cuando estamos comenzando a advertir el boom de una tecnología disruptiva como esta. Términos como criptomonedas o *NFTs* están en boga; todo el mundo habla de ello o, al menos, le suena de algo, pero ¿alguien entiende realmente de la naturaleza de los activos de los que están hablando?

Es cierto que todo el mercado que ha surgido en torno al Blockchain parece ser un imán de inversión. No obstante, como suele ocurrir con el nacimiento de una nueva tecnología, existe mucha especulación a la vez que muy poca información, junto con un ritmo de cambio y actualización acelerado. Más aún, la inestabilidad de este mercado ha quedado más que demostrada en los últimos años.

En definitiva, no son tantos los perfiles que tienen un conocimiento amplio y profundo sobre la materia. No se habla simplemente al conocimiento financiero necesario para invertir de manera inteligente en criptomonedas. Todos los fondos que se están desplazando hacia las finanzas descentralizadas se asientan en la tecnología subyacente. Las redes Blockchain tienen más que convencidos a miles de inversores que apuestan por la potencial revolución que puede generar.

En esta línea, es bien conocido que, hasta el momento, el campo de aplicación en el que las tecnologías Blockchain se han asentado con más capacidad son las finanzas. En concreto, este nuevo nicho ha pasado a llamarse Finanzas Descentralizadas, desvinculándose significativamente de las Finanzas Centralizadas; las finanzas convencionales ya conocidas.

Las redes, protocolos, plataformas, instrumentos, mecanismos y múltiples sistemas fundamentados en Blockchain son la razón de su atractiva y latente rentabilidad. No parece que esta innovación vaya a quedar limitada a las finanzas, sino que propone interesantes mejoras en numerosos campos. Incluso, se está valorando Blockchain

como un medio para generar nuevos paradigmas sociales. ¿Qué mejor que conocer esta tecnología para entender la especulación que hay en torno a ella?

Ante la expansión de los mercados financieros descentralizados, los *Decentalized Exchanges* son unas de las plataformas más populares dado a su significativa utilidad para los agentes que operan en el contexto. Su aportación es bastante sencilla a la vez que elemental en los mercados financieros y no es otra que proporcionar una plataforma de comercio de criptoactivos. *Uniswap V2* o *Curve* son algunos de los protocolos más conocidos que ofrecen esta funcionalidad.

La volatilidad de los tipos de cambio y precios a los que se sujetan los valores de estas plataformas provocan una inestabilidad evidente. Por ello, obtener información fiable, fresca y exacta directamente desde la red descentralizados puede suponer una importante ventaja para los operadores de este entorno. Es más, precisamente esta variación de los precios de los criptoactivos genera un potencial margen de beneficio del que se aprovechan diferentes estrategias de *High Frequency Trading* como el arbitraje.

Aprovechando mis estudios en Doble Grado de Ingeniería Informática y Administración y Dirección de Empresas, veía muy conveniente el trabajar las materias del Blockchain y Finanzas Descentralizadas desde cada uno de estos ámbitos académicos. Se anima al lector a leer el trabajo "Finanzas Descentralizadas en Blockchain: Automated Market Makers, Swaps y Arbitraje" para contar con una visión más amplia del potencial del Blockchain en Finanzas.

1.2 Objetivo y Alcance

En primer lugar, se desea mencionar que este proyecto no aspira a ofrecer mayor protagonismo a la implementación práctica, sino que, conociendo las bases de *DeFi* en Blockchain, este trabajo funciona como una perspectiva técnica de la tecnología y una expansión práctica de la utilidad financiera de las tecnologías Blockchain. De esta forma, se encuentra un punto común entre ambos trabajos a la vez que se desarrolla desde cada uno los aspectos tecnológicos y financieros que convergen en la materia.

Por otra parte, aunque las ramas técnicas que ofrece el Blockchain son muy amplias, se desea conjugar este trabajo con las Finanzas Descentralizadas. Los Contratos Inteligentes son el software esencial en el que se apoyan las *DeFi*. Por ello una implementación orientada a Contratos Inteligentes es un buen acercamiento del estudio, que además proporciona servicio al ecosistema *DeFi*. En concreto, se desarrollará un Oráculo de Precios que interaccionará con otras plataformas descentralizadas. Específicamente, esta API incorporará diversas fuentes para otorgar información sobre tipos de cambio de *swaps* o precios de tokens solicitados.

La herramienta diseñada permitirá ofrecer información de sumo interés para comerciar en mercados descentralizados, haciendo énfasis en las posibles diferencias de precios que se puedan dar entre cada uno de los mercados. Así, la aplicación también será capaz de establecer cuál es el mejor tipo de cambio de entre todos los ofrecidos por cada uno de los protocolos. Adelantamos que, los contratos inteligentes que se diseñarán servirán para la red Ethereum y se programarán en Solidity. El desarrollo de la aplicación resultará muy expositivo para consolidar el recorrido que se hará desde la competencia técnica a la vez que pondrá en manifiesto, de manera más funcional, los aspectos financieros de las *DeFi*.

Este trabajo pretende servir como un acercamiento a las tecnologías Blockchain, con el foco puesto sobre los Contratos Inteligentes en *DeFi*. Se pretende contribuir a la literatura académica para orientar y guiar a todos aquellos que se suman a la ola del Blockchain y las *DeFi* y no ven un claro camino por dónde comenzar.

El objetivo principal de este Trabajo Fin de Grado es: **Comprender la tecnología subyacente a las criptomonedas.**

Se sintetizan más objetivos en la siguiente lista ordenada por prioridades:

1. Implementar una herramienta para obtener información sobre precios en mercados descentralizados y que pueda ser útil para estrategias de comercio de alta frecuencia en mercados descentralizados como el arbitraje.
2. Estudiar el uso de los Contratos Inteligentes a nivel teórico y ofrecer una perspectiva más cercana de su aplicación en *DeFi* desde el desarrollo del Oráculo de Precios.
3. Demostrar la atractiva base técnica que Blockchain ofrece a las Finanzas y exponer el potencial con el que cuenta esta tecnología más allá de las finanzas.

1.3 Estructura del documento

Para realizar el recorrido propuesto, este documento se estructurará en diferentes módulos que cubrirán el tema desde una visión más superficial o general hasta un mayor nivel de especificación:

- ◊ En la siguiente sección, 2, se descubrirá en qué consiste la tecnología Blockchain y cuál es el gran valor que aporta. Primero se comenzará a hablar de la transición al nuevo modelo de Internet al que contribuye la descentralización del Blockchain. A continuación, se expondrán las principales propiedades de la tecnología, que se entenderá en mayor profundidad con el apartado consecutivo, en el que se analizan las características más definitorias de su arquitectura. Para finalizar, se verán los principales campos de aplicación en los que Blockchain tiene algo que ofrecer y se expondrá las limitaciones y debilidades más incomprendidas de esta innovación.
- ◊ En el punto 3 se comenzará estudiando el concepto de Contrato Inteligente. Como ya se ha mencionado, esto es solo una de las ramas de desarrollo de la tecnología Blockchain, de hecho, se descubrirá que hay importantes redes de cadenas de bloques sobre las cuales los Contratos Inteligentes no tienen utilidad significativa. Esto conducirá a *Ethereum*, la red más popular para el despliegue de este software y donde el mayor grueso de los mercados descentralizados se asienta. Se terminará hablando de las barreras que existen en el desarrollo de Contratos Inteligentes dado el precoz estado del software.
- ◊ En la sección 4 se abordará el desarrollo práctico del Oráculo de Precios mediante contratos inteligentes en Ethereum. En primer lugar, se expondrá el objetivo la implementación. A continuación, se presentarán las herramientas y recursos de los que se ha valido el montaje de la aplicación. Seguidamente se planteará la producción del software, desde la definición de requisitos hasta la disposición del código programado. Por último, se mostrará el plan de pruebas llevado a cabo y la ejecución del sistema final.
- ◊ Al igual que esta sección funciona de manera introductoria, se cerrará el proyecto en el punto 5 con una conclusión que sintetice y ultime todo el proyecto y en el cuál puedan presentarse las resoluciones extraídas de todo el trabajo. Como añadido, se ofrecerá un apartado de "Análisis de Impacto" desde dónde reflexionamos acerca de la aportación de este proyecto en diferentes ámbitos.

2 Comprendamos la Tecnología Blockchain

Nuestro trabajo parte de las bases y motivaciones más elementales de la tecnología. Se expone la nueva propuesta del modelo de internet junto con los recursos técnicos de Blockchain que la hacen viable.

2.1 Web 3, punto, ¿qué?

La Web 2.0, el modelo de consumo de internet actual, ha alcanzado la madurez. Esto se puede intuir en diversos factores del entorno digital. Por una parte, la expansión masiva del Big Data ha generado una sobreinformación que ha repercutido negativamente en la credibilidad y valor que se otorga al contenido consumido en internet. Por otro lado, somos conscientes de la regulación, control y monopolio de los gigantes tecnológicos que ofrecen servicios en la red (como las RRSS) o incluso la censura, que se impone de manera estricta en países con "Autoritarismo Digital" como China [1].

No pasa desapercibido todo el valor que ha generado una *World Wide Web* interactiva e inclusiva: ha potenciado la voz de los individuos, ha creado nuevas vías de comunicación en una plataforma global colectiva y ha dado luz a iniciativas y negocios con un impacto muy positivo para los consumidores, la sociedad y el mundo. No obstante, existen otros costes que quedan en la sombra [2].

Uno de los mayores conflictos de la era digital es el control sobre los datos personales: quién los posee o los usa, cómo y para qué. La Web 2.0 supone una amenaza para la privacidad: los datos que generamos en la red se convierten automáticamente en una fuente de beneficio para las empresas que los reciben. Nuestra "identidad virtual" es un reflejo de nuestro comportamiento como consumidores en internet en todas las formas: desde el sitio en el que estuvimos un 18 de marzo a las 17:05h, hasta la música que escuchamos ayer o el artículo que estamos ojeando ahora mismo.

Los usuarios de las plataformas son la fuente de un valor del que solo se ven beneficiados económicamente las grandes corporaciones, además, de manera muy significativa. Por otra parte, esto supone la pérdida fundamental del derecho a la privacidad y control sobre tu identidad. El usuario acepta unos Términos y Condiciones, pero esto no justifica que, en la práctica, la ignorancia generalizada de los usuarios en términos de tecnología o legalidad y la ambigüedad que ofrecen las empresas sobre el uso de sus datos termine por dificultar un contexto transparente, simétrico y justo.

Ante este estancamiento y todo un conjunto de problemáticas en el sistema de Internet, la Web 3.0 es un fenómeno que está ganando tracción. Chris Dixon define la transición Web 1.0, 2.0 y 3.0 como [3] :

Read-Only -> Read-Write -> Read-Write-Own

La Web 1.0 presenta unos protocolos abiertos descentralizados que busca exclusivamente ofrecer información al usuario. El flujo es unidireccional de extremo a extremo, de desarrolladores a usuarios, y colocaba el poder de la Web 1.0 en la comunidad, que posee la infraestructura.

En la Web 2.0 se propone un cambio radical en el que el usuario final puede interaccionar en la red y convertirse en un "agente modulador". Es decir, ya no solo los desarrolladores alteran e impactan lo que hay en internet; todos los participantes de la red tienen capacidad para alterarla. No obstante, toma forma un modelo de centralización acaparado por los dueños de protocolos y servicios, que actúan como intermediaros.

La Web 3.0 pretende quedarse con lo mejor de sus modelos predecesores: una gobernanza de internet por parte de los usuarios (como se conocía en la Web 1.0) con las atractivas posibilidades de interacción (dispuestas en el modelo Web 2.0). La Web 3.0 no es solo una promesa: parece inevitable y se propone como el camino más viable para la democratización real de Internet. La Web 3.0 aboga por un internet gobernado por los usuarios, para los usuarios.

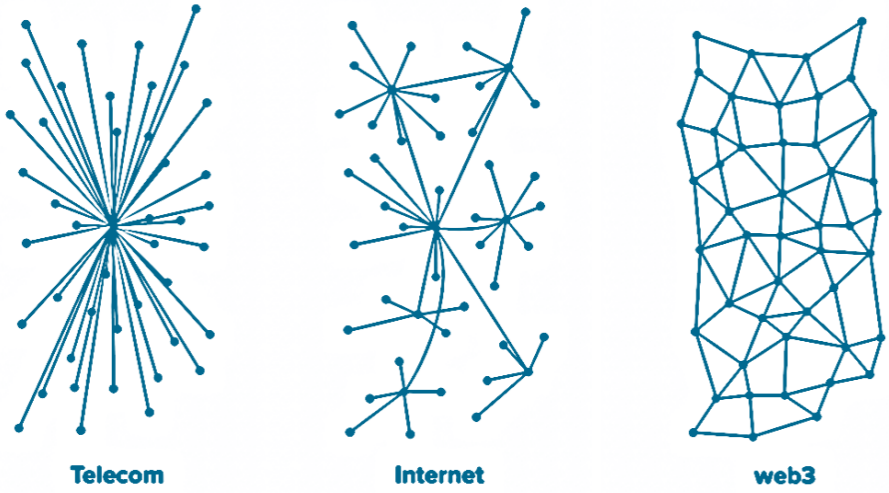


Figura 1: Esquema simplificado de red en Web 3.0 versus modelos previos [1].

2.2 Fundamentos del Blockchain

La base teórica de la tecnología Blockchain se desarrolla en el protocolo *Paxos* y se formaliza en el artículo "The Part-Time Parliament", presentado por Leslie Lamport para la *Association for Computing Machinery* en 1998 [4]. En este artículo propone un modelo de consenso en una red distribuida de ordenadores.

El valor de este algoritmo era la tolerancia a fallos que presenta frente a la participación de miembros "no fiables". Esta teorización evoluciona hasta una el famoso Bitcoin, presentado en 2008 por Satoshi Nakamoto en el artículo "Bitcoin: A Peer to Peer Electronic Cash System" [5].

A pesar de que se abordaron implementaciones de diferentes modelos sistemas electrónicos monetarios previas al Bitcoin, ésta última fue la que logró mayor aceptación. Además, asentaba la distribución del sistema como característica primordial para que no existiera un único punto de fallo mientras se permitían transacciones de extremo a extremo. Por otro lado, proponía una emisión automatizada de la criptomoneda subyacente (Bitcoin) que servía como incentivo para los mineros, que mantienen y expanden la red. Otras de las claves de este sistema son [6]:

- **Persistencia e integridad**. La red Blockchain actúa como un libro mayor en el que se van registrando las transacciones o cambios de estado que tienen lugar en la red, manteniendo el historial hasta el origen. Además, los datos almacenados en el Blockchain son inalterables. Esta característica difiere de los modelos convencionales en los que los valores de los datos que apuntan al estado actual se actualizan o sobrescriben.
- **Seguridad, distribución y transparencia**. Bitcoin proponía desde su origen un sistema distribuido en el que ningún usuario estuviera en posesión del control y en el que no existieran puntos de ataque o fallo. Asimismo, esta tecnología garantiza la validez de los datos que soporta mediante protocolos criptográficos de seguridad que evitan la alteración maliciosa. Estos protocolos están integrados sólidamente en el sistema, de manera que se aplican sobre cualquier participante que desea unirse a la red: no es opcional. Es más, la naturaleza compartida y transparente del sistema otorga una simetría y permite identificar unívocamente a aquellos usuarios que llevan a cabo operaciones ilícitas dentro de la red.
  Por otro lado, la distribución contribuye a la escalabilidad del sistema incorporando nuevos miembros. Esto, a su vez, hace que el impacto de un ataque sobre el modelo de consenso sea menor a medida que la red sigue creciendo.
- **Privacidad.** El modelo de Blockchain hace posible la participación en la red sin la identificación personal de los miembros, sino un alta en el sistema asociado a un pseudónimo. Este *pseudoanonimato* legitima la visibilidad de las transacciones realizadas por los usuarios para todos los integrantes del Blockchain, contribuyendo más aún a los principios de seguridad y transparencia ya mencionados.

2.3 Arquitectura Esencial

A continuación, se van a exponer los principales elementos que caracterizan la tecnología Blockchain. Se expondrán los siguientes puntos, desde un nivel más elemental hasta uno más genérico [7] [8] [9] [10]:

I. Transacciones

Representa la forma más básica de interacción entre dos miembros del Blockchain. Estas transacciones se almacenan en los bloques y son fundamentales para permitir el crecimiento constante de la cadena de bloques y así dotar de mayor seguridad al sistema.

En función de la implementación de la red, el contenido de la transacción puede diferir, pero es normal que incluyan datos como la cantidad de activos digitales de entrada y salida que se dan en la transferencia. Las transacciones no se dan únicamente con activos digitales, sino que también se utilizan para compartir datos.

II. Cifrado Asimétrico

Blockchain está implementado bajo el principio de que las partes involucradas no tienen por qué contar con una confianza mutua previa. Por ello, las transacciones se cifran de manera asimétrica, dado a que es la mejor forma de garantiza la seguridad, eliminando la necesidad del secreto compartido (presente en el cifrado simétrico).

En el cifrado asimétrico existen dos claves: una pública y otra privada. Estas claves servirán para cifrar y descifrar la información, respectivamente. El cifrado asimétrico funciona como una firma digital, en la cual, la clave privada servirá para cifrar la información, mientras que la pública podrá ser usada por cualquier miembro que tenga acceso a ella para descifrar una información. Alternativamente, también puede cifrarse la información con la clave pública y descifrarla con la privada, aunque en la mayoría de redes Blockchain funciona no funciona así.

Este mecanismo permite verificar la integridad y autenticidad en transacciones, que permanecen públicas sin que emisores y receptores se conozcan o compartan confianza mutua. En otras palabras, la criptografía asimétrica permite verificar que el usuario emisor de valor está en posesión de la clave privada para poder ejecutar dicha operación.

La criptografía simétrica es significativamente más rápida que la asimétrica, sin embargo, no ofrece la misma seguridad que la criptografía asimétrica en el contexto de confianza distribuida. Para lograr beneficio de la velocidad de la criptografía simétrica con la seguridad de la asimétrica, se puede optar por un cifrado mixto: los datos se cifran simétricamente mientras que las claves se cifran asimétricamente.

III. Registro

Un libro mayor es un registro o historial de transacciones. Durante siglos, previo a los sistemas analógicos y digitales, este método de registro se ha llevado a cabo en diferentes ámbitos, con importante presencia en el ámbito financiero. En la era digital se ha seguido utilizando y se ha gestionado, normalmente, de manera centralizada.

¿Y si los libros mayores se mantienen como registros replicados y compartidos por todos los nodos de una red? Así, estos nodos contarían con acceso simultáneo, validación y actualización. Esta tecnología fue bautizada como *Distributed Ledger Technology* (DTL) y está en pleno auge tras la revolución de su precursor: Bitcoin. Blockchain no es más que una categoría de DLT [11].

Con esta innovación, el interés en libros mayores distribuidos ha ido creciendo por la seguridad, fiabilidad y confianza que ofrecen con respecto a un libro mayor en posesión centralizada:

- Un registro central puede ser destruido.
- Un registro central ofrece homogeneidad de hardware, software e infraestructura de la red, lo que reduce la resistencia a ataques.
- Un registro central suele estar asociado a una localización geográfica determinada, por lo que la información queda a merced del contexto geográfico.
- En un registro central, el usuario debe confiar que todas las transacciones están quedando reflejadas de manera completa y válida, y que las transacciones antiguas no están siendo alteradas.

En suma, un registro central es mucho más vulnerable a ataques debido a su naturaleza centralizada: un solo objetivo de ataque puede presentar riesgos de seguridad mucho mayores que un sistema distribuido compuesto por miles de nodos.

IV. Funciones Hash

Blockchain hace uso de funciones hash, específicamente *SHA-256* o *Secure Hash Algorithm* con una salida de 256 bits. Esta función produce *outputs* hash(x) y hash(y) diferentes para dos mensajes x e y de *input*. Además, puede aplicarse sobre datos de diversa naturaleza: desde texto hasta imágenes. La salida será siempre de 256 bits, independientemente de la longitud de la entrada. Cualquier cambio en la entrada producirá una salida totalmente diferente.

Existen $2^{256} \approx 10^{77}$, o 115,792,089,237,316,195,423,570,985,008,687,907,853,269,984,665,640,564,039,457,584,007,913,129,639,936 valores de salida. Esto significa que hay valores de salida finitos. Sin embargo, es altamente improbable que suceda una colisión tal que hash(x) = hash(y). Se estima que, para lograr encontrar una colisión, el algoritmo se debe ejecutar de media 340,282,366,920,938,463,463,374,607,431,768,211,456 veces.

V. Bloques y encadenamiento

Las transacciones que tienen lugar en la red se van propagando a todos los nodos y se van acumulando en un "pool" o repositorio de transacciones temporal, una especie de buffer replicado en cada nodo de la red. Cuando hay transacciones suficientes para formar un bloque, las contenidas en este repositorio se liberan para acoplarse en el nuevo bloque publicado por uno de los nodos de la red. El proceso de selección del nodo publicador se explica en próximos apartados.

Es en el momento de la generación del bloque cuando se verifica la validez y autenticidad de las transacciones: cada transacción tiene el formato adecuado y el proveedor de activos digitales ha firmado criptográficamente con su clave privada. En caso de que no se acepte cualquiera de las transacciones de la lista, el bloque completo se rechazará.

La cabecera del bloque contiene metadatos, mientras que el contenido del bloque almacena un conjunto de transacciones (u datos de otro tipo) validadas y autenticadas por el Blockchain. La cabecera de los bloques suele tener un formato:

- Número de bloque
- Hash de la cabecera del bloque anterior
- Una representación hash de los datos del bloque
- Marca de tiempo
- El tamaño del bloque
- El valor *nonce*

El valor del hash de los datos del bloque se puede hallar con diferentes mecanismos. Uno de los más comunes es el *Merkle Tree,* que aplica el algoritmo SHA-256 recursivamente en estructura de árbol hasta llegar a un único valor hash en la raíz [12].

El formato de la cabecera puede variar en función de la red y versión del Blockchain. Por ello, en los metadatos de la propia cabecera, se suele incluir un atributo que indique la versión del bloque para que la red comprenda bajo que protocolo funciona dicho bloque de la red.

Estas propiedades ayudarán a mantener la coherencia e integridad de la estructura y su contenido a medida que se va expandiendo. En concreto, se observa que el hash del contenido del bloque *A* en la cabecera permitirá actuará como un "resumen" unívoco del contenido de este bloque. A su vez, el bloque *B* tendrá un "resumen" hash de la cabecera del bloque *A*. Por consiguiente, en caso de cualquier tipo de vulneración a la integridad de los datos de *A*, el siguiente bloque *B* "notará" esta brecha.

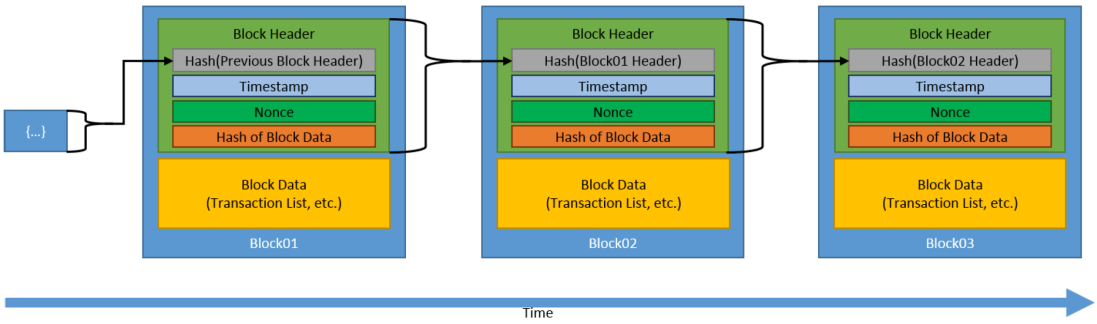


Figura 2: Encadenamiento de Bloques [7].

Así, sucesivamente, si cualquier información de un bloque previo se alterara, el hash de este bloque sería diferente y, por tanto, los bloques siguientes también se alterarían. Esto hace muy fácil la detección de bloques modificados y, por consiguiente, su rechazo.

Se debe recordar que, si se altera un bloque en un nodo, todos los demás nodos mantienen una copia del mismo bloque. El protocolo de consenso juega un papel fundamental en el "juego de hash", al que se enfrentarán los mineros, que se explicará más adelante.

VI. Tipos de Blockchain

La clasificación fundamental de las redes Blockchain se realiza según su modelo de permiso o, dicho de otra forma, en quién tiene la capacidad de mantener la red publicando bloques. Existen dos categorías: *permissioned* y *permissionless* o con y sin permiso. Utilizando un símil, las primeras actúan como una intranet corporativa controlada mientras que las segundas funcionan como el internet público, accesible a todos los usuarios. Desarrollando algo más:

- ***Permissionless***. Plataformas de registro descentralizadas en las que no se requiere el permiso de ninguna autoridad para participar. Este tipo de redes Blockchain funcionan como un software libre, a disposición de cualquiera que desea descargarlo. De esta forma, cualquier usuario involucrado en una red Blockchain de tipo *permissionless* puede publicar bloques y, por lo tanto, leer y escribir en el sistema, así como participar en transacciones como emisor o receptor.
  Ante esta abierta participación, las redes *permissionless* pueden ser más vulnerables a ataques u operaciones maliciosas por parte de algunos usuarios. Esto obliga a las redes de este tipo a integrar modelos de consenso lo suficientemente seguros para defenderse de este tipo de actuaciones.
- ***Permissioned***. En contraste con la categoría anterior, las redes Blockchain "permisionadas", "con permiso" o *permissioned*, solo permiten la publicación de bloques a los usuarios que cuentan con la autorización de algún tipo de entidad que, a su vez, puede ser a su vez centralizada o descentralizada.
  Una de las propiedades más definitorias en este tipo de sistema es la publicación y verificación de la identidad de cada usuario; de esta forma es posible otorgar autoridad, capacidades y competencias diferentes dentro del sistema. Esta

publicidad de la identidad repercute en la conservación del anonimato o privacidad, dispersando la confianza distribuida. Aunque este cambio en el modelo de confianza puede suponer una desventaja o impedimento para ciertos usos, aplicaciones o sistemas, también cuenta con puntos a su favor:

- El uso de recursos hardware y energéticos se ve sustancialmente reducido. Esto se debe a que los modelos de consenso en este tipo de redes requieren, generalmente, mucha menor complejidad computacional, ya que la legitimidad de los usuarios se verifica a través de su identidad conocida. Este modelo conlleva también mayor velocidad.
- Las redes Blockchain permissioned pueden ser utilizadas en organizaciones que buscan un control mayor sobre sus registros por motivos de seguridad o privacidad. Esto juega en contra del principio de descentralización que persigue la tecnología Blockchain, ya que los usuarios la red deben confiar en esa entidad, y si es una entidad centralizada, con mayor motivo. Por ello, en estas redes, el modelo de consenso utilizado se puede determinar en base a la confianza que se tenga la entidad controladora.
- La revelación de la identidad de los usuarios sirve para personalizar o individualizar la autoridad o capacidades que se le otorgan a cada usuario. De esta manera, se le puede permitir a un usuario leer la red, pero no generar bloques, revelar selectivamente información de las operaciones o definir cuál puede ser su papel como involucrado en una transacción.
- Por otro lado, este requisito de identificación genera un desincentivo natural para el fraude o cualquier otro tipo de actuaciones con fines maliciosos, dado a que la identidad del usuario transgresor puede ser conocida y, por consiguiente, penalizada.

VII. Modelos de Consenso

En las redes Blockchain es fundamental el mecanismo encargado de seleccionar el nodo que publica el nuevo bloque. En las redes *permissioned* se suelen aplicar algoritmos que están sujetos a un mayor determinismo para definir el nodo publicador, mientras que en las *permissionless,* los nodos deben competir por publicar el nuevo nodo. A continuación, se presentan los dos modelos más extendidos para este último caso:

A. Modelo de consenso *Proof of Work*

Este modelo de consenso consiste en que el nodo encargado de publicar el nuevo bloque es aquel que resuelve antes un problema computacional y, por lo tanto, la solución que presenta al problema queda como "prueba de su trabajo". El problema está formulado de tal manera que hallar la solución es difícil, pero validar la solución es fácil. Así, los demás nodos pueden verificar con facilidad que el bloque publicado por el nodo ganador es válido.

El reto o problema planteado más común consiste en lograr que el hash de la cabecera del nuevo bloque sea menor que un valor objetivo. Los miembros participantes en el reto logran la alteración del *hash* de la cabecera para dar con el

objetivo a medida que modulan el valor de *nonce*, uno de los parámetros que se usa en el input del cálculo. A pesar de que hallar el hash de la cabecera no es computacionalmente muy pesado, realizarlo iterativamente por cada *nonce* diferente que se desea probar se vuelve tarea muy densa computacionalmente.

El valor objetivo es modificado con periodicidad para facilitar o dificultar más la tarea en nuevas ejecuciones del algoritmo de consenso. Entonces, la complejidad puede ajustarse a la cantidad de nodos integrantes que existen en la red, así como la frecuencia con la que se desea publicar un bloque. Cuanto menor sea el valor objetivo, más difícil será hallar la solución del problema.

Cuando un nodo dé con la solución, se convertirá en el nodo ganador y, por lo tanto, en el nodo publicador. A continuación, compartirá el nuevo bloque con el *nonce* calculado al resto de miembros de la red, que verificarán que la solución es efectivamente correcta: calcularán el mismo hash que el nodo ganador para verificar que se satisface la incógnita planteada. Entonces, el ganador añadirá el nuevo bloque a su cadena, y lo propagarán a sus nodos pares, permitiendo una rápida distribución del nodo por la red. En la red Blockchain de Bitcoin, se estima que se valida un nuevo bloque cada 10 minutos.

Un aspecto importante de este modelo de consenso es que el esfuerzo invertido en resolver el problema no incrementa la probabilidad de resolver el siguiente problema. Por ello, cuando algún nodo ha hallado la solución y se ha convertido en el nodo publicador, los otros nodos dejarán de intentar hallar la solución e integrarán el nuevo bloque. Esto distribuye la probabilidad de lograr ser el nodo publicador.

Además, este modelo previene ataques de seguridad que pretender lograr la influencia sobre la red a través del control de múltiples nodos o identidades. Para poder llevar a cabo ese ataque en Blockchain, es necesario contar con suficiente potencia computacional, que solo es alcanzable con una desmesurada inversión en hardware. Más aún, siguen teniendo que participar en un sistema de lotería: cuanta más capacidad computacional se tenga, más probabilidad hay de encontrar la solución, pero tampoco es una garantía.

Este modelo de consenso propone un modelo de incentivo, ya que los nodos ganadores recibirán una bonificación por el esfuerzo invertido en hallar la solución al problema. En las redes Bitcoin o Ethereum, la recompensa que reciben son criptomonedas de nueva creación. Por esta razón, la carrera por hallar la solución al problema también se conoce como "minería" y los competidores se hacen llamar "mineros".

B. Modelo de consenso *Proof of Stake*

Uno de los problemas más grandes que presenta el modelo *Proof of Work*, por el cual ya ha suscitado polémica y crítica, es su ineficiente consumo energético. Se estima que, solo en la red de Bitcoin, se consumen mil millones de vatios por segundo para producir un nuevo bloque mediante este modelo de consenso. Otras métricas estiman

que la red Bitcoin tiene el mismo gasto energético que el país entero de República de Irlanda [13].

La finalidad de la complejidad computacional de este modelo es dificultar el proceso de creación de bloques para limitar el acceso y capacidad de generación, y así distribuir la responsabilidad de publicarlos en la red. Aun así, de alguna forma, el gran consumo energético para este proceso computacional resulta excesivo para la meta tan elemental que cubre: seleccionar un nodo publicador.

El modelo *Proof of Stake* consiste en otorgar mayor probabilidad de convertirse en el nodo editor a aquellos cuya participación (o "Stake") sea mayor. La participación se mide normalmente en una cantidad de criptomonedas que el usuario invierte en el sistema. Por lo tanto, se elimina la inversión en tiempo, energía y computación que presenta *Proof of Work*. Además, dado que no se invierten los mismos recursos, las recompensas para los mineros no tienen por qué ser tan altas, sino que se reducen a unas mínimas tasas de transacción.

Además del significativo ahorro y eficiencia energética, el modelo *Proof of Stake*, también se fundamenta en el principio de que los usuarios con mayor capital en criptomonedas o "participaciones" sobre la propiedad de la red, son menos propensos a atacar o abusar de su posición de nodo publicador (en caso de resultar seleccionado).

En la otra cara, este modelo tiende a beneficiar más a aquellos candidatos que poseen mayor grueso de criptoactivos. Por ello, existen diferentes variaciones e implementaciones sobre este modelo que pretende hacer más aleatoria la elección del nodo.

VIII. Conflictos en el Blockchain y Forking

El término "fork" tiene dos acepciones diferentes dentro de las redes Blockchain. Ambas hacen referencia a la existencia de conflictos temporales en el registro de la cadena de bloques mantenida por diferentes nodos. No obstante, esta situación puede originarse en dos contextos diferentes.

Es probable que se dé la situación de múltiples bloques siendo publicados muy próximos en el tiempo, casi instantáneamente. Esto puede dar lugar a diferentes bloques añadiéndose al Blockchain en diversos nodos de la red y, como consecuencia, creando diferentes versiones del registro. Se recuerda que las transacciones que aún no se han añadido al Blockchain se encuentran a la espera de la generación de un nuevo bloque almacenadas en un repositorio temporal.

Por ejemplo, puede suceder que un nodo A publique un bloque A' con las transacciones 1, 2 y 3 mientras que el nodo B publica un bloque B' con las transacciones 1, 2 y 4. Ninguno de los bloques es erróneo, pero la existencia de ambos bloques puede violar el principio de la integridad, ya que una versión del Blockchain puede mostrar unos activos digitales como transferidos mientras que la otra versión puede mostrar los mismos activos como no transferidos.

Para solucionar este problema, normalmente se espera a la generación del siguiente bloque y se optará por la versión del Blockchain más larga. Las transacciones no contempladas en la versión elegida se volverán a enviar al repositorio temporal de transacciones para que sea añadido al próximo bloque. Es por esto que, normalmente, una transacción no se da por confirmada hasta que se han añadido varios bloques por encima del bloque que la contiene.

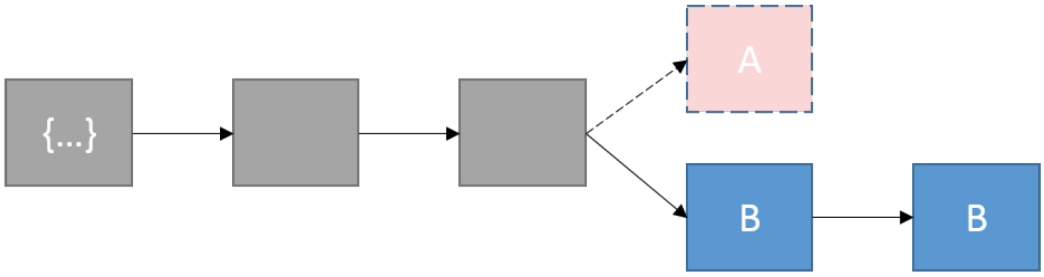


Figura 3: *Conflicto entre dos versiones de Blockchain [7].*

Los cambios y actualizaciones que se dan en los protocolos y estructuras de la tecnología Blockchain también se denominan *forks*. En este caso, también se dan conflictos entre las diferentes versiones del Blockchain en los nodos, pero está motivado por un cambio en el software más que una conflicto de publicación de bloques.

Es importante tener en cuenta que, cuanto mayor sea el tamaño de la red, más difícil se hace la propagación de las actualizaciones y el consenso entre todos los usuarios para la aceptación de dichos cambios.

Existen dos tipos de *forks*: *soft forks* y *hard forks*. En el primer caso, se permite la compatibilidad entre los nodos actualizados y no actualizados. Es decir, los nodos que no se hubieran actualizado no iban a encontrar ningún problema para realizar transacciones con los nodos que sí estuvieran actualizados. De hecho, en caso de que la actualización no fuera adoptada por un grupo importante de nodos, se revocarían los cambios, manteniendo la versión antigua.

Por el otro lado, los *hard forks* no permiten la compatibilidad "hacia atrás". Los nodos no actualizados no aceptarán el nuevo formato de los bloques, por lo que no podrán seguir creciendo junto a los demás integrantes de la red. Tarde o temprano deberán actualizarse para adoptar los cambios y poder seguir interaccionando con el resto de participantes de la red.

En la siguiente figura se ofrece una visión general de la arquitectura de la cadena de bloques:

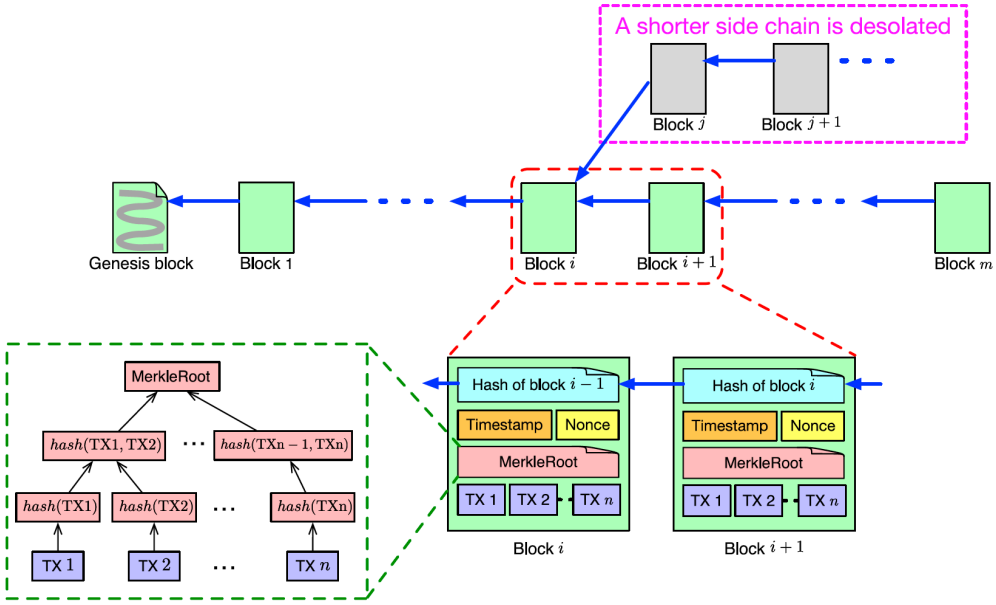


Figura 4: Perspectiva general de estructura del Blockchain *[14]*.

2.4 Aplicaciones de la Tecnología Blockchain

En este apartado se recorrerá de manera genérica las aplicaciones reales que tiene la tecnología Blockchain. Algunos de estos campos están en pleno apogeo con el enriquecimiento de las innovaciones Blockchain. El caso más sonado es el de las Finanzas. Otras áreas, como el Gobierno, se encuentran actualmente en un estadio de desarrollo prematuro. No obstante, se desea incluirlas y exponerlas dado al atractivo que surge de la unión con tecnología Blockchain y el enorme potencial que aún está por explorar.

El impacto del Blockchain abarca una gran variedad de campos: Finanzas, Gobierno, Identidad y Certificación, Registros Públicos y Privados, Activos Intangibles, Distribución de Contenido Digital, Seguimiento Logístico, Energía, *Internet of Things*, Turismo y Hostelería, Sanidad, *Smart Cities* y Producción Industrial [15] [10] [16] [17] [6].

Finanzas	Es el sector más explotado por este tipo de tecnologías actualmente. Algunos de los factores que han motivado la aparición de un mercado descentralizado han sido la creciente desconfianza hacia las instituciones financieras, que progresivamente acaparan mayor influencia y poder, o las barreras que dificultan el acceso a determinados mercados y operaciones financieras. La naturaleza distribuida de Blockchain permite la automatización y desintermediación en el contexto de las finanzas, lo que reduce los tiempos y costes a la vez que prioriza la seguridad y la transparencia. Las Finanzas Descentralizadas impulsan la interoperabilidad, inclusión y custodia de valor o derechos, además de otra infinidad de nuevas posibilidades que facilitan u democratizan el intercambio de valor. Criptomonedas, tokens, activos digitales, financiación, préstamos, derivados, seguros, pensiones, fondos de liquidez, mercados automatizados o arbitraje son algunas de los instrumentos y operaciones que se han reinventado o creado con el nacimiento de las DeFi, dando inicio de una nueva forma de finanzas más flexible y eficiente.
Gobierno	Existe mucha literatura académica en torno a las nuevas posibilidades de gobierno que puede ofrecer el Blockchain [18]. De hecho, uno de los atractivos fundamentales de esta tecnología radica en los mecanismos de consenso en los que se fundamentan los diferentes productos, sistemas y redes.

	Partiendo desde un ejemplo más básico, estas tecnologías podrían servir como plataforma pública verificable para el recuento de votos en una elección. Más adelante, la descentralización podría permitir una administración pública más funcional que se puede valer de la trazabilidad de los ciudadanos en los sistemas del Estado. También puede posicionarse como plataforma para dar voz al pueblo sin ser objeto de censura. En un nivel superior, se menciona el término "Decentralized Autonomous Organization" o DAO. Tal y como indica su nombre, son organizaciones que carecen de dirección y gestión humana. En su lugar están codificadas a través de contratos inteligentes. Simultáneamente, son entidades que representarán de manera mucho más fiel a la comunidad, ya que su estrategia estará definida por el consenso de todos los miembros integrantes. Este tipo de sistema puede operar como organismo público o privado. Por ello, también se le alude con el nombre de "Decentralized Autonomous Corporation" (DAC). Este concepto es fuertemente referenciado y estudiado por los potenciales usos que idealmente podría ofrecer y existen ambiciosas promesas en torno a este concepto. No obstante, carece de sólido recorrido o significativas implementaciones macroscópicas en las que sustentar su validez. Prueba de la inmadurez del DAO es que, en 2016, el mayor proyecto desplegado hasta la fecha en torno a este concepto, sufrió un ataque que sustrajo 60 millones de dólares en fondos de *Ether* [19].
Identidad y certificación, Registros públicos y privados, Activos Intangibles y Distribución de contenido digital	La seguridad criptográfica de las tecnologías Blockchain garantiza la unicidad, integridad e irrevocabilidad de las claves privadas. Bajo el supuesto de una custodia responsable de estas claves, los beneficiarios pueden ser capaces de atestiguar su identidad o cualquier tipo de certificaciones vinculadas a ella de manera veraz. Esta forma tan efectiva de sellar los datos puede ser trasladada a prácticamente cualquier ámbito en el que se deseen sustituir el almacenamiento tradicional de BBDD por una verificación y soporte más transparente en redes Blockchain. De manera anexa, el respeto a la certificación de patentes, marcas, pruebas de autoría o autenticidad, licencias y otra multitud de activos intangibles o derechos digitales, , están

	amenazados por la naturaleza centralizada de los sistemas de seguridad tradicionales. Aprovechando las ventajas de la descentralización, Blockchain otorga a los creadores y propietarios una nueva capacidad de supervisión y control sobre este tipo de activos, garantizando una protección mucho mayor sobre los derechos de esta información sensible sujeta a la legalidad.
Seguimiento logístico	Blockchain podría resolver el reto de auditoría, transparencia y trazabilidad que presentan multitud de productos en la cadena logística. Esto se puede aplicar tanto en el campo de los productos alimenticios como para mercaderías de diferente naturaleza en las que interesa especialmente el origen y seguimiento. En el ámbito de la seguridad alimenticia, por ejemplo, es fundamental conocer la gestión de producción, venta al por mayor, distribución, certificaciones, estándares de producción u otros factores involucrados en las diferentes fases de la cadena; desde la generación del producto hasta su recepción por parte del consumidor. Una aplicación real ajena a la industria alimenticia es la propuesta del proyecto *Everledger*, que certifica el origen de los diamantes a través de Blockchain para evitar la distribución de diamantes de sangre.
Energía	Los Sistemas de Energía Distribuidos componen una propuesta muy interesante que pretende acabar con muchas de las limitaciones de las redes de energía centralizada y funcionar como alternativa que maximice la eficiencia energética. Apoyándose en otras tecnologías como *Internet of Things*, Inteligencia Artificial y *Machine Learning*, se pretende monitorizar y registrar el proceso en redes Blockchain, desde su generación hasta el consumo energético, pasando por todas las fases y procesos operativos de este dominio. La capacidad distribuida y la confianza que genera la implementación de este sistema de energía en Blockchain da lugar a un flujo de información veraz entre emisores y receptores de energía. Por otro lado, pretende impulsar el uso de las fuentes renovables incorporando nuevas entidades en la red que

	funcionen como emisores de energía a pequeña escala, y que pueden valerse de las redes Blockchain como plataforma segura para el intercambio, comercio y distribución energética. De esta forma, un particular con placas solares podría comercializar el excedente de su energía. Algunas iniciativas de este ámbito pueden ser *PowerLedger*, *Share&Charge* o *NRGcoin*.
Internet of Things (IoT)	*Internet of Things* es una tecnología con la que ya estamos familiarizados. Los dispositivos conectados a través de internet para compartir datos específicos asociados principalmente a entornos físicos: sensibilidad a la temperatura, humedad, movimiento, iluminación… *IoT* es una tecnología que cuenta con un recorrido más amplio y sólido que el Blockchain: existen numerosas aplicaciones comercializadas y dispositivos basados en la propuesta de *IoT* en nuestras casas. No obstante, aún cuenta con un importante potencial que está por explorar y explotar, así como retos y fisuras que se deben superar. Algunas de las limitaciones de la tecnología *IoT* están ligadas a las capacidades de red, almacenamiento o computación. Estos obstáculos pueden derivar en vulnerabilidades de seguridad. Estas mismas trabas pueden ser resueltas o reforzadas con la firme infraestructura distribuida del Blockchain.
Turismo y Hostelería	Conociendo el importante componente de digitalización que ha potenciado la industria de la hostelería y servicios turísticos, bien asentada y actualizada en la era de Internet, no es sorprendente que apuesten por la integración en las redes Blockchain. Muchos de los servicios de este campo se centran en relaciones de cliente a cliente y, es por ello que su adaptación a las redes descentralizadas puede ser especialmente conveniente y beneficiosa. Por ejemplo, un modelo de negocio como el de *airbnb* podría implementarse sobre redes Blockchain ofreciendo una potencial mejora en sus prestaciones. Primero, prescindiendo de la intermediación de una plataforma centralizada que puede ser fuente de demoras, ineficiencias y mayores costes de transacción. Como consecuencia, la descentralización favorece la seguridad, privacidad, verificación y autenticación de las identidades, comunicaciones o pagos; llegando incluso hasta la cobertura de riesgo o acceso a las viviendas. Muchas vulnerabilidades más

	pueden ser superadas evitando una intermediación centralizada corporativa para trasladando el servicio de extremo a extremo; entre propietarios y huéspedes. Innovaciones como el pago en criptomonedas están sacando partido de las propuestas de Blockchain, junto a otras funcionalidades que pretende conjugar a clientes, mercado y conocimiento disponible.
Sanidad	El campo de *Smart Healthcare* o Sanidad Inteligente se encuentra en plena expansión y evolución. El principal planteamiento de esta área es monitorizar la salud de los pacientes a través de dispositivos y sensores postizos. Este proceso genera flujos masivos de información. Como consecuencia, grandes cantidades de datos asociadas a diagnósticos y pruebas médicas, entre otros, deben ser almacenados en Registros Electrónicos Sanitarios. Actualmente, este almacenamiento de datos suele diseñarse con una arquitectura centralizada. La tecnología Blockchain puede gestionar y depositar esta información de manera distribuida para otorgar la privacidad, seguridad, disponibilidad y certificación de estos datos de naturaleza tan sensible. Además, puede facilitar la accesibilidad de esta información hacia otros emisores que autoricen los propietarios de los datos. Todo ello sin la necesidad de una BD sanitaria común. Algunas propuestas existentes son *Healthcare Data Gateway* o *MedRec*.
Smart Cities y Producción Industrial	Con el crecimiento de la población urbana y la expansión de las ciudades, la integración de sensores, *IoT* y otro tipo de tecnologías son algunas de las medidas que derivan de este fenómeno de digitalización en las zonas metropolitanas. Este concepto ha sido muy explorado, estudiado e incluso implementado a pequeña escala. La gestión del tráfico, transporte público, residuos, suministro de agua o energía, instalaciones públicas, servicios comunitarios y plantas energéticas, entre otras, son algunas de las materias que pueden abordarse desde la visión de *Smart Cities*. Además de las cualidades ya mencionadas de seguridad, privacidad y autenticidad que otorga esta tecnología, es especialmente oportuno contar con un soporte en redes Blockchain para asegurar la validez de la información ofrecida por

	todos los nodos de la red *IoT* que componen la infraestructura inteligente de la ciudad. En el mismo hilo, el uso de *IoT* en la producción industrial está incrementándose significativamente. Se ha probado la eficiencia que otorga la integración de este tipo de tecnología sobre diferentes ámbitos del proceso productivo: plantas de producción, gestión de inventarios y almacenes o la observación de ciclos de desarrollo. En definitiva, el sistema industrial se vuelve increíblemente óptimo gracias a la mayor automatización que ofrecen las herramientas *IoT*. El mantenimiento predictivo resultando en una mayor eficiencia energética, la calidad y velocidad productiva son algunas de las posibilidades. La infraestructura distribuida puede actuar como una plataforma de coordinación entre los diferentes terminales *IoT* involucradas en la cadena del proceso. Más aún, no solo puede suponer una mejora en la seguridad y eficiencia del proceso, sino que además puede otorgarle una autonomía absoluta a través de los contratos inteligentes, permitiendo prescindir de cualquier tipo de intervención humana para organizar el proceso productivo.

2.5 Debilidades, Limitaciones y Retos del Blockchain

Es fundamental entender que, Blockchain, como cualquier otra tecnología, presenta limitaciones y puntos débiles. Algunas adversidades como el excesivo consumo energético o coste de los recursos hardware ya han sido mencionados.

La descentralización, escalabilidad o confianza distribuida son propiedades que se han presentado como hazañas de esta tecnología, pero se debe recalcar que pueden terminar convirtiéndose en fuentes de problemas para algunos sistemas o aplicaciones específicas. Se debe entender que esta tecnología no es apta para cualquier implementación.

Es más, en una fase tan temprana de evolución, promesas como la seguridad o transparencia no están tan garantizadas como se puede proponer a nivel teórico. Desde la introducción ya se hizo referencia a la creciente ola de *hype* que rodea esta innovación. En esta sección se apuntaron algunas de las desventajas más notables de la tecnología Blockchain [7]:

I. ¿Inmutabilidad?

A pesar del principio teórico de inmutabilidad bajo el que se implementa y se promociona el Blockchain, no se puede hablar de una inmutabilidad absoluta e inmediata.

En el apartado VIII de la sección 1.3 se hablaba del concepto de *forking* y cómo, cuando existe una bifurcación en la cadena de bloques, se adhiere la más larga y se trasladan aquellos bloques que se encuentran en la cola "desechada" a ese "buffer" para más adelante incluirlos en la cadena principal. Ya se adelantaba la latencia con la que se validan las transacciones mientras se ubican en su posición definitiva dentro de la cadena de bloques. Este fenómeno hace tambalear el concepto de inmutabilidad.

Existe una forma de ataque a la red denominada popularmente como "El Ataque del 51%" por el que, poseyendo la mayoría de los recursos de la red (>50%), se podría alcanzar el dominio de la producción de bloques. Con este control, sería factible generar bloques de manera muy acelerada con el consenso corrupto de la mayoría de nodos de la red.

Esta flaqueza funcional hace posible que un usuario malicioso fuerce al sistema a decantarse por una cola concreta: se anexionan bloques a una de ellas para hacerla más larga. Con ello, se podrían llegar a introducir en la cadena de bloques determinadas transacciones emitidas a posteriori antes de las transacciones que queden en "espera", provenientes de la cola original y legítima que se revocó por *forking*. En otras palabras: una transacción original *A* se podría manipular introduciendo otra transacción *B* en la cola más larga. Consecuentemente, *A* quedará como transacción posterior a *B*.

Este ataque, aunque no es técnicamente muy complejo, requiere una exagerada inversión en recursos computacionales, ya que necesita hacerse con el control de la red; simplemente es tan extremadamente caro que llega a ser imposible. Sin embargo, la

dificultad de realizar el ataque dependerá de la dimensión de la red en la que se hace y el nivel de retrospección con el que se desea hacer el cambio. Es decir, conforme más se desee remontar en la cadena de bloques para introducir la transacción maliciosa, mayor ratio de generación de bloques se necesitará. Por otra parte, este tipo de ataque puede ser mitigado sin mayor dificultad en las redes Blockchain de tipo *permissioned*.

II. Asimetría de poder en la red

La naturaleza distribuida del Blockchain genera una falsa apariencia de que "nadie controla o posee el Blockchain". Lo cierto es que, para posibilitar la creación, funcionamiento, mantenimiento y desarrollo de esta tecnología, deben existir agentes que controlen o posean el sistema. En las redes *permissionless* esta autoridad se deposita de manera más distribuida y uniforme sobre los usuarios de la red, nodos publicadores y desarrolladores. Sin embargo, en las redes *permissioned*, el poder se concentra en una entidad o consorcio.

Siguiendo en esta línea, es importante resaltar la posición de influencia en la que se encuentran los desarrolladores del software. A pesar de que los sistemas son, en su mayoría, de código abierto, pocos son los usuarios con las capacidades y recursos para analizarlo o compilarlo, y menos aún para desplegarlo como protocolo operativo.

Además, en algunas ocasiones, los desarrolladores pueden actuar en nombre de toda la comunidad sin previo consenso. Por ejemplo, en 2013, los desarrolladores de la red Bitcoin lanzaron una nueva versión que resultó ser deficiente. En proceso de transición hacia esta nueva versión, advirtieron el fallo y tuvieron que optar entre adoptarla definitivamente o volver a la versión anterior. De cualquier de las maneras, muchos de los usuarios se iban a ver afectados al revocarse la versión de la red que utilizasen.

En las redes Blockchain *permissionless*, los usuarios no suelen están obligados a adoptar las nuevas actualizaciones del software. Sin embargo, los nodos publicadores pueden definir la tendencia de la transición hacia una nueva versión. En caso de existir un *hard fork* (explicado en el apartado VIII de la sección 1.3), los usuarios de la red que no hayan adoptado la versión de los nodos publicadores no podrán incorporar los nuevos bloques generados y por lo tanto quedarían marginados de los beneficios que éstos propongan. Inversamente, los miembros de la red no se verán motivados a adherirse a la nueva versión si los nodos publicadores no lo hacen.

También en las redes *permissioned*, en caso de que el consorcio o administrador al mando quiera actuar ilegítimamente, tendría a su alcance capacidades de tipo: excluir a usuarios de ciertas funciones o beneficios, tomar el control de la producción de bloques o alterar los datos de la cadena.

III. La desunión entre lo real y digital

Las redes Blockchain trabajan muy bien con los datos originarios del ecosistema digital. No obstante, muchos de los datos con los que trabajan provienen del mundo real; por ejemplo, un sensor de temperatura. La captación de estos datos se hace a

través de los denominados "Oráculos". Estos elementos, en su acepción más genérica, no son más que cualquier tipo de dispositivo o software integrado en el mundo *off-chain* para nutrir de datos a las redes Blockchain. Es totalmente posible que un oráculo esté aportando de información corrupta o errónea (intencionadamente o no) a la red Blockchain.

Existen iniciativas como *Chainlink* que ofrecen alternativas para cubrir esta vulnerabilidad, pero las soluciones dirigidas a este fin no están tan extendidas y asentadas. A pesar de que este riesgo no es exclusivo de las redes Blockchain sino de una amplia variedad de sistemas digitales, puede llegar a ser un problema especialmente grande en plataformas donde se trabaja con anonimato y activos digitales. Por ejemplo, si se hace un pedido utilizando pago en criptomonedas, no existe manera efectiva de saber si el pedido se ha llegado a enviar sin algún tipo de sensor o intervención humana.

IV. Actuaciones maliciosas y Ciberseguridad

Al igual que en cualquier plataforma o sistema digital implementado hoy en día, se debe garantizar un programa de ciberseguridad robusto que sujete un diseño y mantenimiento del producto a prueba de ataques. Esto es especialmente relevante a medida que la tecnología Blockchain se desarrolla en paralelo al conocimiento de los hackers. En esta carrera de la industria y el mercado por unirse a la ola de innovación del Blockchain, las soluciones desplegadas pueden ofrecer nuevas vulnerabilidades (o más de las ya conocidas) que se convertirán en vectores de ataque de las que, tarde o temprano, un usuario malicioso se aprovechará.

Una red donde no se puede imponer un código de conducta no es inmune a actores malintencionados. Este factor multiplica su presión en la categoría de redes *permissionless*, en la que la identidad real es desconocida. Aunque estos sistemas hayan diseñado mecanismos de incentivo para el buen comportamiento como la percepción de criptomonedas, potenciales malhechores pueden ver un nicho en aquellos ataques que les reporte mayor beneficio.

La tecnología Blockchain ofrece una seguridad intrínseca que se convertido en uno de sus principales valores. No obstante, su resistencia a la alteración no es infalible. Ya se introducía en el apartado I de esta sección algunas de las posibles fracturas de esta inmutabilidad. Otro ejemplo de vulnerabilidad viene de las redes Blockchain que se valen de un código de tiempo transaccional: falsear la hora (o *timestamp*) puede ser una vía para alterar la integridad del registro.

Como se ha expuesto, la mayor barrera para estos ataques es disponer de recursos suficientes para lanzar el ataque de manera efectiva, pero una vez logrado ese control, se pueden dar otras formas de actuación maliciosas que alteren la distribución de la información como ignorar transacciones de emisores concretos o no emitir bloques a nodos anexos.

En contra de estos potenciales riesgos, es importante mencionar que, como respuesta a un ataque, los desarrolladores, usuarios o administradores de la red

Blockchain afectada, pueden llevar a cabo un *hard fork* si existe el consenso. Esto podría "restaurar" o "revertir" los daños causados por el ataque. Esto se llevó a cabo en el famoso "Ataque al DAO" de 2016, presentado en el punto -o. Ante el evidente perjuicio causado, se llevó a cabo un *hard fork* que dividió la red *Ethereum* en *Ethereum Classic*. Esta última reflejaría las consecuencias de la malversación de fondos mientras que *Ethereum* se reconstruyó como cadena principal desde el momento previo al ataque. Este procedimiento permitió "recuperar" el capital robado de ETH valorado 60 millones de dólares [20] [19].

3 Contratos Inteligentes

Una vez se ha concluido el recorrido por la tecnología Blockchain y se ha entendido sus fundamentos, se cuenta con una base lo suficientemente sólida para entender el concepto de Contrato Inteligente. En varias secciones anteriores se ha mencionado el término "Contratos Inteligentes" y, probablemente el lector se haya preguntado "¿qué es esto?". De la manera más simplificada posible, los contratos inteligentes no son más que código construido encima de la tecnología Blockchain. Se desarrolla en los siguientes apartados [21] [14] [22]:

3.1 "Smart" + "Contracts"

El término lo propone *Nick Szabo* en los años 90 [23]. Su aportación consistía en trasladar las cláusulas de un contrato convencional a un software, soportado en un hardware que permitiera una ejecución automática cuando unas condiciones predefinidas se diesen. De esta manera, los procesos se podrían ejecutar sin necesidad de intermediación humana a la vez que los costes y riesgos de la transacción se verían minimizados. Esta idea, que en un principio no parece tan descabellada, quedo abordada simplemente de manera teórica ante el escaso sustento técnico con el que se contaba en el momento.

Puede que esa filosofía de eliminar costes de la transacción resulte familiar. Es precisamente porque, desde el inicio de este trabajo, se ha incidido bastante en uno de los puntos cimentales de Blockchain: eliminar la intermediación y los costes asociados a ellos. Y es que, a pesar de que las estructuras centralizadas, jerarquías, intermediación, externalización y otros sistemas corporativos pretenden ser una alternativa para optimizar la industria o mercado [24], contar con una automatización de procesos que involucran relaciones contractuales supone un mayor avance para minimizar costes en ciertos contextos.

Es importante puntualizar que cuando se habla de costes de intermediación o transacción no se limitama estrictamente al coste económico explícito asociado al servicio, sino que también se puede hablar de los riesgos u otro tipo de costes indirectos (de oportunidad, de agencia, operativos etc.). En definitiva, se podrá maximizar la eficiencia de los procesos con menor riesgo y coste. Con toda la infraestructura y posibilidades que surgen en base a la tecnología Blockchain, este concepto de Contratos Inteligentes (CI) que había quedado estancado ya puede tomar forma en aplicaciones reales.

Los contratos inteligentes se implementan por encima del Blockchain. En esencia, el código y los estados por los que transita se almacenan de manera distribuida y replicada en todos los nodos de la red. Así, se asegura la validez de la ejecución del contrato al contar con el consenso paralelo de todos los nodos, sin recurrir a una tercera parte confiable. Además, tal y como promete la tecnología Blockchain, ofrece transparencia y minimiza el riesgo de manipulación arbitraria o malintencionada de agentes involucrados.

Se puede condensar estos atributos de la siguiente forma: contrato inteligente es un programa ejecutado de manera distribuida en una red Blockchain. Tienen un comportamiento determinista en contraste con la tergiversación o desviación en la que pueden incurrir las relaciones contractuales tradicionales ya que formalizan una relación contractual entre partes diferentes, garantizando la compensación o retribución del valor como contrapartida de la transacción, pactado con anterioridad al desarrollo y despliegue del contrato. El cumplimiento del acuerdo es irrevocable y el procedimiento subyacente que se programa se ejecuta de manera automática, autónoma y atómica.

En este punto cabe mencionar que el término "Contrato Inteligente" cuenta con más de una acepción. Generalmente, se acepta Contrato Inteligente como lo que ya ya expuesto: un programa. No obstante, también existe una aproximación más legal a este concepto: un Contrato Inteligente también puede considerarse un software que, en mayor o menor medida, formaliza las condiciones o cláusulas de un contrato y que puede requerir intervención humana [22]. Se centra el estudio en la primera significación.

Por otro lado, también es oportuno resaltar que los CI no tienen por qué ser estrictamente implementados sobre el Blockchain. Sin embargo, sí es cierto que este tipo de DLT es la tecnología subyacente extendida para el desarrollo de este formato software, ya que ha aportado la infraestructura técnica necesaria para implementar este concepto de manera efectiva y eficaz.

3.2 Ethereum y otras redes

Ya es sabido que *Bitcoin* fue la primera red Blockchain; la precursora. Esta red estableció las bases para todas las otras que se implementaron más tarde. Por ello, se comenzará por los orígenes para establecer unas bases. A continuación, se tratarán muy superficialmente las redes *Corda* y *Hyperledger Fabric*. Por último, se extenderá la exposición en la red que más interesa para este trabajo: Ethereum.

I. Bitcoin

Bitcoin introduce el modelo transaccional. Estas transacciones se realizan entre diferentes cuentas. Las cuentas no atesoran el activo (Bitcoin) como se podría esperar de una cuenta bancaria convencional. En contraste, las transacciones se realizan sobre el historial de transacciones anteriores. Entonces, el registro del balance de cada miembro de la red no se lleva a cabo actualizando los valores ni calculando el capital del que dispone en su cuenta, sino que se extrae a través de las cifras previas almacenadas inmutablemente en el Blockchain [25].

Este modelo se bautizó como UTXO en el planteamiento original del *Bitcoin* [5], iniciales que vienen de *Unspent Transaction Output*. Este nombrado hace referencia a la imposibilidad de transferir Bitcoins que ya han sido utilizados para un propósito; no se puede gastar un activo que ya se ha gastado. En caso de que se de esta última situación, la transacción sería revocada por la red.

Es importante volver a poner el foco en la importancia de las claves privadas en este proceso. Lo único que da poder para utilizar esos fondos es la clave privada que sella la autenticidad del activo. Por consiguiente, perder la clave implica perder los valores en *Bitcoin* asociados a ella.

Una transacción recibe una o múltiples entradas de UTXO que se derivarán a una o múltiples salidas (que a su vez generarán nuevos flujos de UTXO). Si el sumatorio de la entrada es mayor que el de la salida, se generará un nuevo flujo de salida devuelto al usuario. Se ejemplifica en la siguiente figura:

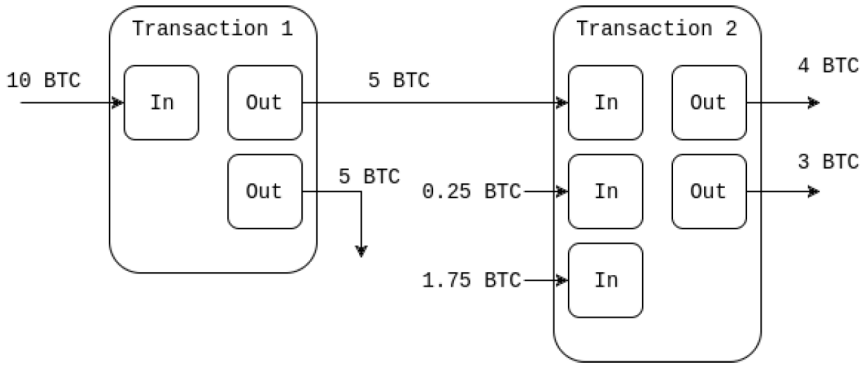


Figura 5: Representación de transacciones en Bitcoin. La T1 dirige a otras cuentas 5 BTC y retiene los otros 5 BTC que suman la entrada de 10 BTC [25].

II. Hyperledger Fabric

A diferencia de las redes *Bitcoin* y *Ethereum*, *Hyperledger Fabric* es una plataforma *permissioned.* Referenciando lo ya expuesto en el apartado VI de la sección 1.3, las redes *permissioned* implicaba una identificación, autenticación y autorización expresa para poder asignar los diferentes permisos con los que contará cada nodo. Por ello, existirán diferentes categorías de nodos en función a sus roles, capacidades o permisos.

La aportación diferencial de *Hyperledger Fabric* son los *channels* o canales. Estos canales son subconjuntos de usuarios de la red que comparten un libro mayor (o registro) para su uso exclusivo, asentado sobre un Blockchain o cadena de bloques. Por consiguiente, en este canal se da un intercambio de transacciones y activos disponibles exclusivo para sus miembros [25].

Otros de los atributos diferenciales de Hyperledger Fabric (en adelante, HF) [14]:

- A diferencia de Ethereum, que ejecuta los contratos inteligentes en una máquina virtual (EVM), *HF* se vale de contenedores en Docker, lo que tiende a minimizar la sobrecarga en la ejecución.
- HF acepta lenguajes de programación de alto nivel como Java.
- De igual manera que Ethereum y en contraste con Bitcoin, HF adopta un sistema *Turing-complete*.
- HF utiliza un modelo de estructura de datos de par clave-valor.

Se entenderá en más profundidad algunos de estos puntos cuando sea desarrollada la sección de Ethereum.

III. Corda

Corda también es una red Blockchain de tipo *permissioned*. Esta plataforma se caracteriza por su enfoque en servicios financieros sobre activos digitales. En *Corda*, los usuarios de la red están *mapeados* a identidades legales reales.

De manera parecida a *HF*, *Corda* da soporte a diferentes grupos o plataformas de usuarios autorizados que pueden intercambiar activos y acceder al historial de transacciones de manera privada.

Este sistema también acepta lenguajes de programación de alto nivel bien conocidos como Java. Sin embargo, estos se ejecutarán sobre la Máquina Virtual de Java (JVM). Por otro lado, esta plataforma no es *Turing-complete*. Su paradigma de datos se fundamenta en transacciones [14].

IV. Ethereum y el inicio de las DeFi

Ethereum es la red Blockchain que más interesa analizar en este trabajo. La red Ethereum es operativa desde julio de 2015 y actualmente es la plataforma más popular para el desarrollo de contratos inteligentes [22]. Esto se debe fundamentalmente a todas las prestaciones que ofrece en materia de Contratos Inteligentes en contraste con las otras redes.

Ethereum se vale del planteamiento original de *Bitcoin* para llevarlo más allá. Para empezar, esta plataforma presenta un sistema de Turing completo que difiere con el sistema de Turing incompleto basado en lenguaje script de tipo pila que ofrecía Bitcoin. Esto implica que operaciones computacionales como los bucles no podían llevarse a cabo en la red de *Bitcoin*, pero sí en la de *Ethereum*. Para poder emular una ejecución iterativa en Bitcoin sería necesario repetir el código, lo cual resulta extremadamente ineficiente computacionalmente [26]. Retener objetos y datos o interaccionar con otros contratos también son nuevas funcionalidades que se adquieren en *Ethereum*.

Por ello, se ha puesto al alcance de los programadores en *Ethereum* diferentes lenguajes de Turing completo mucho más expresivos como *Solidity* o *Serpent* que permiten desarrollar Contratos Inteligentes complejos que van más allá de aplicaciones del ámbito financiero. Estos contratos se compilan en formato *bytecode* y se ejecutan en la máquina virtual descentralizada de *Ethereum* (EVM).

Uno de los atributos fundamentales que caracteriza la red *Ethereum* es el *gas*. El *gas* no es más que un precio o tasa impuesto por la ejecución de transacciones. La ejecución de los Contratos Inteligentes se realiza de manera paralela en todos los nodos de la red (por la naturaleza distribuida del Blockchain) y esto implica un gasto computacional muy elevado. Como consecuencia, este mecanismo se implementa como forma de retribuir el coste que soporta la red, así como limitar la cantidad de recursos que puede utilizar la ejecución de un Contrato Inteligente.

Asociado al *gas*, existen los siguientes parámetros:

- start_gas/gas_limit: máximo número de "pasos computacionales" permitidos para la transacción.
- gas_price: tasa valorada en *Ether* (ETH) que debe pagar el emisor de la transacción por "paso computacional".

En base a esto, el coste de una transacción es start_gas x gas_price.

Además, con este mecanismo cortafuegos, la red Ethereum logra lo siguiente [22] [14] [25]:

- Mitigar el spam.
- Permitir a la red Ethereum conocer dónde reasignar recursos computacionales.
- Regular la inestabilidad del valor de ETH en base a la oferta de la criptomoneda.
- Evitar bucles infinitos (intencionados o no).
- Limitar la capacidad computacional de operaciones maliciosas y hacer pagar a los malhechores por los recursos invertidos en el ataque.
- Motivar el desarrollo de programas eficientes, evitando el lanzamiento y ejecución de código "basura".
- En definitiva: maximizar la eficiencia de los Contratos Inteligentes que se ejecutan en EVM.

En oposición al modelo de datos basados en UTXO con el que opera *Bitcoin*, la red *Ethereum* se fundamenta en un modelo de cuentas con el que cada participante es

identificado por su cartera digital, una dirección de 20 bytes que le permite realizar operaciones *peer-to-peer*. Estas operaciones se formalizan como transiciones de estado.

Las cuentas pueden ser relativas a un usuario o a un CI. En términos formales, son prácticamente iguales. Contienen los siguientes campos: un valor *nonce* que evita que una transacción se pueda ejecutar más de una vez, el balance de la cuenta y el almacenamiento de datos de ésta. En caso de ser una cuenta vinculada a un CI incluirá otro campo para el código fuente.

Por otra parte, las ya conocidas transacciones cuentan con los siguientes campos:

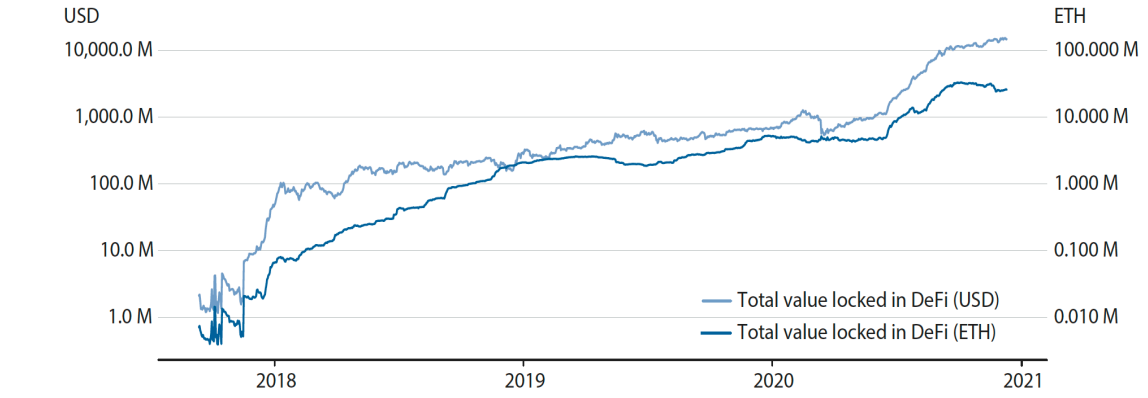
- Dirección del destinatario.
- Valor de *Ether* a transferir. Medido en *Wei* = 10^{-18} ETH.
- gas_price
- start_gas/gas_limit
- La firma criptográfica del emisor.

En este sistema existen mensajes que replican el comportamiento de las transacciones. La diferencia radica en que los mensajes son generados por los propios contratos con la finalidad de invocar código de otros CI (asociados a otras cuentas en otras direcciones).

Estas propiedades que se definen convierten a *Ethereum* en una plataforma muy atractiva para el soporte de software más allá de meras transacciones. Esencialmente, el lenguaje de Turing completo posibilita el desarrollo de programas dirigidos a diferentes aspectos; el más explotado es el ámbito de las Finanzas, pero pueden enfocarse a otro tipo de funcionalidades.

Gran parte de las plataformas y servicios que han surgido en la red Ethereum se valen de *Tokens*. Un *token* es un criptoactivo construido por encima del protocolo de la red Blockchain, su naturaleza es muy parecida a la de las criptomonedas nativas a la red. Los tokens no representan un valor líquido, una reserva de ahorro o medio de cambio como sí sucede con las criptomonedas. En diferencia, los tokens reflejan activos intangibles como derechos, propiedad o licencias [25] [27] [28] [29].

Las Finanzas Descentralizadas son uno de los campos de mayor expansión en el mundo de Blockchain. Se podrá el foco de los Contratos Inteligentes en las *DeFi*, ya que, una vez marcada la amplia utilidad de este software, se considera que el campo de las Finanzas Descentralizadas puede resultar especialmente representativo para completar recorrido que lo precede. En la siguiente figura se aprecia el valor de la coyuntura de las DeFi x Contratos Inteligentes:



NOTE: M, million.

SOURCE: DeFi Pulse.

Figura 6: *Valor total bloqueado en Contratos Inteligentes en la red Ethereum para DeFi [21].*

Se aproxima la arquitectura de la red Blockchain de Ethereum con dirección en *DeFi*. Se comienza indicando que, al igual que otros sistemas digitales bien asentados, cuenta con un diseño de capas múltiples. En concreto, de manera similar a otras redes de computadores, es de esperar que, cuanta más descentralización y más terminales abarca el sistema, se requiere una mayor abstracción del protocolo (o software) sobre el hardware en el que se apoya.

Tal y como lo propone Fabian Shär [21], se puede hablar de los siguientes niveles o capas:

1. **Asentamiento**. Red Blockchain con su estándar y criptomoneda nativa. En este nivel se asegura la propiedad de los valores y la integridad del historial de transacciones o transición de estados.
2. **Capa de criptoactivos**. La criptomoneda nativa se extiende hasta esta capa y se alinea con los nuevos tokens construidos en la red en base a contratos inteligentes.
3. **Capa de protocolos.** En este nivel se montan los protocolos diseñados para las plataformas, mercados y servicios *DeFi*. La compartición de esta capa por parte de las diferentes entidades promueve interoperabilidad.
4. **Capa de aplicación.** Este nivel está orientado al usuario final del servicio. Desde aquí, se logra una abstracción del sistema para hacerlo usable para el cliente.
5. **Agregación.** Actúa como una extensión de la capa de aplicación. La diferencia de este nivel con el anterior radica en que, desde aquí se pretende conectar y combinar diferentes aplicaciones para ofrecer servicios específicos para el usuario que faciliten la interconexión.

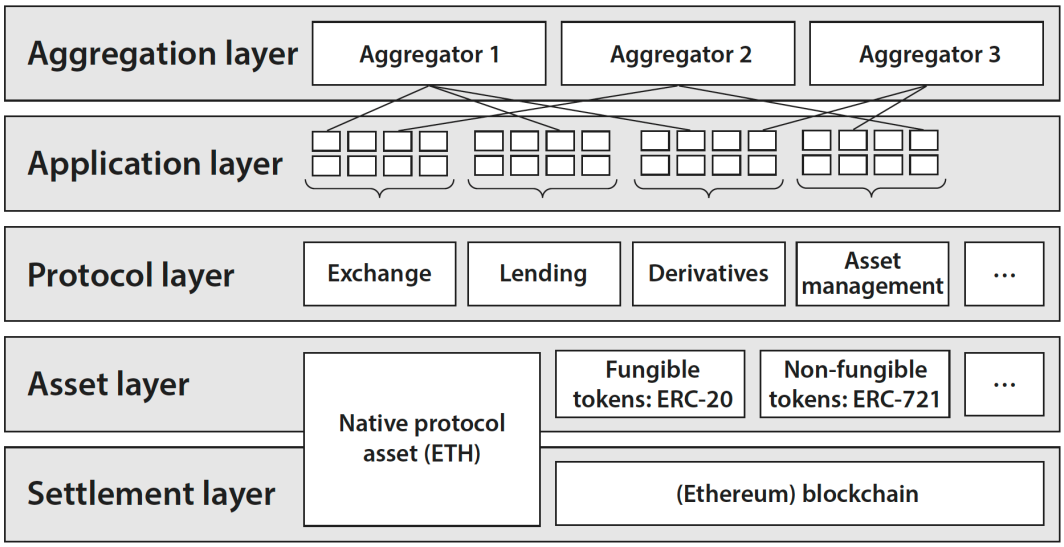


Figura 7: La pila DeFi [21].

Con esta información y poniendo el foco en la red Ethereum, se cierra este apartado exponiendo el ciclo de vida de los Contratos Inteligentes conlleva las siguientes fases [14]:

1. Creación. Las partes involucradas establecen las bases del contrato que los desarrolladores traducirán del Lenguaje Natural al lenguaje de programación pertinente (como Solidity). De la misma forma que la programación ordinaria de software, se llevará a cabo el diseño, implementación y testeo. Este proceso se puede dar de manera iterativa.
2. Despliegue en la red. El contrato inteligente (o contratos inteligentes) se compila en la EVM para montarse por encima de la red Ethereum en la capa de aplicación o agregación (mencionadas previamente) en formato *bytecode*. Una vez el CI se encuentra establecido en la red, no es posible modificarlo de ninguna manera.
3. Ejecución. En caso de que las condiciones lógicas del contrato se satisfagan, los procedimientos determinados en el programa se ejecutarán automáticamente. Las transacciones ejecutadas y las actualizaciones de estado se almacenarán en el Blockchain.
4. Finalización. El Contrato Inteligente ha completado el ciclo completo cuando la transición de estados del programa ha quedado reflejado en la red y, en caso de que hubiera activos digitales involucrados, los valores se han desbloqueado del contrato para dirigirse al destinatario (o destinatarios) correspondiente.

3.3 Los Contratos Inteligentes también están limitados

No se quería cerrar esta sección sin antes mencionar algunos de los obstáculos que surgen en el desarrollo de Contratos Inteligentes. Esta problemática se debe, fundamentalmente, del estadio prematuro de la tecnología. Las herramientas y recursos ofrecidos son insuficientes, limitados e incluso defectuosos. Además, el consenso, formalización y normalización en torno al desarrollo de CI también es escaso dado al reducido recorrido y asentamiento con el que cuentan los productos implementados con contratos inteligentes.

A continuación, se van a enunciar algunos de los retos a los que se enfrentan los desarrolladores de CI. Estos inconvenientes son expuestas por desarrolladores expertos que han sido entrevistados y encuestados en el trabajo de "Smart Contract Development: Challenges and Opportunities" [22]:

- **Seguridad**. La seguridad en el desarrollo de CI debe trabajarse con una integración mucho más certera que en desarrollo de software tradicional debido a:
  - Gestión de información sensible como activos financieros.
  - Irreversibilidad de transacciones dado a la inmutabilidad del Blockchain.
  - Incapacidad para modificar el código una vez se lanza en la red.
  - Código de acceso público, que permite a los malhechores estudiar sus ataques previamente con mayor capacidad.
  - Compiladores deficientes que inducen errores.
  - Carencia de prácticas para escribir código seguro.
  - Falta de herramientas y técnicas para verificar la corrección del código.
  - A pesar del pseudoanonimato, todas las transacciones de contratos inteligentes son visibles en Blockchain, lo que conlleva un reto a la hora de preservar la privacidad y gestionar el ocultamiento de información [14].
- **Depuración**. En su mayoría, los desarrolladores encuentran mucha dificultad para depurar el código.
  - Escasez de herramientas potentes para depuración interactiva, que permita visualizar el estado de la memoria y variables o depurar línea por línea.
  - Mensajes de error ambiguos o poco informativos.
- **Limitaciones del lenguaje de programación.** Según un propio desarrollador [22], el único lenguaje de programación que está listo para producción es Solidity. Aún así, Solidity tiene un trayecto breve que lo revela como aún inmaduro:
  - Pocas librerías de propósito general.
  - Poca transparencia para la visualización de errores o informes.
  - Carencia de estándares o reglas de programación.
  - Inconveniencia para llamar a funciones externas; Solidity no acepta abstracciones de estructuras de datos como parámetros.
  - Inaccesibilidad al desarrollador para gestionar la memoria.
  - Limitación en número de variables locales, lo que lleva a los programadores a utilizar variables de estado, más pesadas.

    - Solidity es un lenguaje procedimental, que implica la indicación explícita por parte del programador la secuencia de pasos en la que se ejecuta el código [26].
- Ethereum Virtual Machine (EVM).
    - Soporte muy reducido para depuración.
    - Rechazo de lenguajes de programación tradicionales.
    - Ineficiencia de la ejecución del *bytecode* (o CI ejecutable), debido al diseño de hilo único. Es significativo mencionar que es un reto lograr una ejecución concurrente debido a la compartición de datos por parte de los programas [14]. Esta circunstancia espera derivar en un problema de mayor peso con la expansión de la red y la creciente necesidad de capacidad ejecución para ejecutar más programas [26].
- Gas. Uno de los puntos diferenciales más grandes de la programación de contratos inteligentes con respecto al desarrollo software tradicional es el requisito de pagar tasas de gas, que van sujetas a la complejidad del código a ejecutar. Por ello, los desarrolladores deben prestar especial atención a la consumición de gas:
    - El gas es dinero, por lo que los recursos que necesiten de la red se traducirán en costes directos sobre el usuario o entidad que los requiera.
    - Aborto de transacción al quedar sin fuente de gas a medida que se va ejecutando.
    - No existen herramientas exactas para estimar el coste de gas a nivel de código fuente, lo que impide optimizar el código en términos de consumo de gas.
    - La optimización de gas implica en muchas ocasiones obstaculizar la legibilidad del código.
- Apoyo online y de la comunidad. Acostumbrados al extenso soporte disponible para el desarrollo del software bien conocido, *Solidity* y otros lenguajes de la esfera cuenta con recursos de dominio público mucho más reducidos debido a la novedad de la tecnología.
    - Escasez de código de referencia, que obliga a los desarrolladores a construir los Contratos Inteligentes desde cero.
    - Falta de conocimiento estandarizado: hay descohesión entre las pocas fuentes de información que existen.
    - Documentación disponible desactualizada debido a la rápida evolución de la tecnología.
    - Aunque el apoyo de la comunidad en torno a la programación de Contratos Inteligentes está creciendo, aún sigue siendo muy limitado.

4 Desarrollo de la Contribución: Creación de Oráculo de Precios de *Automated Market Makers* con Contratos Inteligentes

Habiendo alcanzado esta sección, se pretende que el lector tenga una noción lo suficientemente cercana a lo que significa Blockchain y Contratos Inteligentes como para entender o, al menos, aproximarse a la exposición de esta parte del proyecto.

Una implementación práctica es la mejor manera de consumar el trabajo de investigación realizado en el proyecto y con ella, ofrecer una perspectiva mucho más funcional de lo que sucede en el innovador entorno Blockchain. Con esta aplicación no solo se verán reflejadas algunas de las particularidades y atributos tan valorados de esta tecnología, sino que también se proporcinará una API plenamente funcional y útil al ecosistema *DeFi*.

4.1 ¿Qué se propone?

Tal y como se adelantaba previamente, se pretene orientar el tramo final de este trabajo hacia las Finanzas Descentralizadas. Aunque ya se ha expuesto todo el potencial que el Blockchain tiene por explotar en áreas tan diversas como el desarrollo urbano o la sanidad, las *DeFi* ofrecen un nicho suficientemente consolidado y extenso para asentar nuestra aplicación. Además, resulta de especial interés expandir el área que mayor atracción ofrece en términos de cotización e inversión de capital.

Dentro del mundo de las *DeFi*, se centrará el alcance de nuestro sistema en la comunicación con los *Decentralized Exchanges* (DEX) implementados como *Automated Market Makers* (AMM).

Los *Exchanges* convencionales son plataformas de intercambio de valores bien conocidas en las finanzas centralizadas ya que ofrecen una amplia plataforma de comercio para todos los agentes del mercado que desean adquirir o cambiar activos. La arquitectura paradigmática de estos sistemas es centralizada. Aunque los *Centralized Exchanges* fueron la implementación primitiva de estos sistemas para el comercio de criptoactivos, era de esperar que una tecnología que fomenta los valores de descentralización, desintermediación y confianza distribuida diera a la luz unos *Decentralized Exchanges*.

Trasladar la concepción de *Exchanges* al entorno de las Finanzas Descentralizadas necesitó la imprescindible aportación de los Contratos Inteligentes, que permitirían la ejecución autónoma de un código de manera distribuida en la red. Pese a las diferentes modelos que se podían abordar para los DEXes, los *Automated Market Makers* han probado ser el mecanismo más eficiente; tanto a nivel operacional como financiero.

Los *AMM* ofrecen un mercado automático para el intercambio de criptoactivos en el que compradores y vendedores satisfacen óptimamente su oferta y demanda. Sin

entrar en extensa especificación, se sintetizan los puntos clave de los *AMM* que interesan a nuestro trabajo:

- Los *AMM* sostienen reservas de pares de valores en las que se retienen dos criptoactivos diferentes.
- La liquidez de ambos valores de la reserva la inyectan los *Liquidity Providers*, que reciben intereses por ceder su capital.
- Los corredores que deseen cambiar sus criptomonedas o tokens deberán recurrir a la reserva A ⇌ B correspondiente al *swap* que desean realizar.
- El precio de los valores A y B bloqueados en la reserva A ⇌ B definirá el tipo de cambio A/B o la cantidad de criptoactivos A que se pueden obtener en base a una unidad de B (o viceversa).
- Los precios de A y B estarán condicionados por la relación de magnitudes contenidas en la reserva de ambos criptoactivos. Por lo tanto, los swaps ejecutados por los usuarios alterarán el tipo de cambio de la reserva A ⇌ B, ya que introducen y extraen valores del fondo al realzar sus operaciones de cambio.
- Una mayor provisión de liquidez dotará de más solidez al fondo y, como consecuencia, una menor oscilación del tipo de cambio A/B.

Con estas pautas explicadas en los puntos anteriores, se puede intuir que el precio de oferta de swaps varía en cuestión de segundos. Teniendo en cuenta que existen decenas de protocolos de AMM [30] (probablemente cientos[1]), y miles de criptoactivos [31] , es imaginable todas las alternativas que existen de *swaps* a través de los mercados descentralizados y la variedad de precios bajo los que se pueden operar.

Ya se ha introducido el concepto de Oráculo; se recuerda que funciona como un sistema que aporta información agnóstica a su alcance. La aplicación que se propone es un Oráculo de Precios; un sistema que ofrecerá datos de precios. Estos precios serán los tipos de cambio ofrecidos para *swaps* de un par específico A ⇌ B o tokens en las plataformas elegidas. Con ello, se podrá obtener orientación de los precios del entorno, no solo para determinar la plataforma que ofrece mejor *swap*, sino para advertir las diferencias entre las ofertas ofrecidas desde cada mercado.

Los *AMM* o fuentes seleccionadas para nutrir el Oráculo son: *Uniswap V2*, *Uniswap V3*, *Sushiswap*, *1inch*, Curve y *Chainlink*. Los cinco primeros son AMM que pueden proporcionar servicio de *swap*, mientras que Chainlink es una red de oráculos que suministrará información de precios unitarios de tokens. Los precios se expresan en términos de *Wei* = 10^{-18} ETH.

[1] En la fuente se presenta la cotización de todos los tokens conocidos emitidos por protocolos de AMM, pero no es estrictamente necesario que un AMM comercialice un token propio para operar, por lo que pueden existir muchos más AMM además de los considerados en la lista.

La implementación de esta aplicación se llevará a cabo mediante la programación de diferentes contratos inteligentes que compondrán el sistema. Como ya se sabe, estos CI se desarrollarán en Solidity y se desplegarán en la red Ethereum. Esto último es una verdad a medias, más adelante se comprenderá por qué. Además del desarrollo, también se tratarán los recursos, herramientas y factores involucrados en el proceso.

En síntesis, se permitirá a un usuario conocer, en un momento preciso, información sobre precios actualizada, segura y exacta suministrada directamente desde los mercados descentralizados. Con ello, un operador en *DeFi* tendrá referencia sobre el valor de los criptoactivos que se comercian en los múltiples mercados, pudiendo obtener una ventaja competitiva de los márgenes que se producen en la fluctuación de valores.

4.2 Recursos y herramientas

Para comprender cómo se ha llevado a cabo la implementación, se deben conocer los recursos y herramientas de necesarios para desarrollar estos contratos inteligentes.

I. Apoyo en los primeros pasos

Las siguientes herramientas han servido como recursos de referencia en los que apoyar lo que será la implementación definitiva. Es decir, en las fases iniciales, entender la dinámica de la red Ethereum o manipular contratos inteligentes con los que interaccionaría el sistema han sido requisitos primordiales para poder crear la aplicación final y simularla desde un entorno específico.

Se considera que la exposición de estos recursos es pertinente para el trabajo dado a que sirve como marco para contextualizar el entorno en el que se despliega nuestra aplicación. Además, puede servir como referencia para programadores que quieran adentrarse en este ámbito.

Open Source

Uniswap, como muchas otras plataformas o aplicaciones *DeFi*, facilita su código fuente en Internet desde GitHub [32]. Esto permite a los demás participantes del contexto Web 3.0 comprender la dinámica de un sistema de esta naturaleza, servir como referencia a otras aplicaciones o incluso apoyarse en la comunidad de desarrolladores para optimizar los protocolos que implementan.

Más iimportante aún, los desarrolladores se nutren de este código fuente para producir aplicaciones vinculadas a Uniswap, promoviendo la interoperabilidad en este ecosistema. No todas las funcionalidades de Uniswap son orientadas al usuario desde una interfaz *off-chain*, sino que otros aspectos del sistema como los *Flash Swaps*, el oráculo de precios que construye el sistema y demás mecanismos que integran los servicios de Uniswap dependen de que los desarrolladores generen nuevos contratos inteligentes en base a las funcionalidades proporcionadas desde el *open source* por Uniswap.

Por otra parte, Uniswap también ofrece una documentación muy accesible e inteligible para todos aquellos que quieran conocer más acerca de los protocolos que implementan [33].Centrándonos en el protocolo Uniswap V2, se expone el diseño del sistema:

- Core [34]. Es el núcleo del sistema, encargado de crear y gestionar los pares, así como los fondos de tokens almacenados en reservas, con su respectiva vinculación a los usuarios participantes en la plataforma:
  - UniswapV2Pair. Funciones elementales de emitir, intercambiar o retirar tokens.
  - UniswapV2Factory. Crea y controla los pares existentes.
  - UniswapV2ERC20. Gestiona los *Liquidity Tokens* que reciben los Proveedores de Liquidez a cambio de su inyección de fondos en los *Swap*

Pools. Estos tokens llevan asociados el derecho de propiedad y participación en el sistema [35].

- **Periphery** [36]. La periferia de Uniswap V2 pretende servir como API o *front-end* del sistema. La mayor parte de sus funciones cubren esencialmente las mismas prestaciones que las implementadas en el Core, pero desde la periferia el sistema se vuelve más usable para los usuarios y actúa como envoltorio para amortiguar los posibles errores que cometan los agentes que interaccionen con Uniswap [37].

Además, existen librerías en ambos módulos, que pretenden optimizar el diseño del sistema en funciones como la recuperación de datos o el establecimiento de precios. Por ejemplo, la librería *SafeMath* está orientada al *overflow/underflow* de los datos y la librería UQ112x112 está implementada para que Uniswap V2 pueda soportar números en coma flotante.

Uniswap V2 sirve como la documentación y código abierto modélico. Pero se debe considerar también la información facilitada por las otras plataformas que se incorporan en el desarrollo [2]:

- Uniswap V3: documentación [38], *open source* de *core* [39] y *periphery* [40].
- Sushiswap es un *fork* de Uniswap V2, por lo que, aunque el protocolo presenta algunas diferencias, a los efectos funcionales que conciernen en la implementación, se puede operar de la misma forma que con *Uniswap* V2.
- Curve: documentación [41] y *open source* [42] de *Registry*.
- 1inch: documentación y *open source* [43].
- Chainlink: documentación [44] y open source [45].

Test Networks

Las aplicaciones funcionales de Ethereum, donde las *DeFi* mueven criptoactivos con valor real, se asientan en la red principal de Ethereum, la *mainnet*. Sin embargo, tal y como ya se ha explicado, las tasas de gas se aplican sobre cualquier transacción que tenga lugar en la red. El lanzamiento de contratos inteligentes también implica un coste transaccional.

Los desarrolladores de sistemas basados contratos inteligentes en la red Ethereum utilizan redes públicas de prueba para construir actualizaciones del sistema o nuevos productos asentados en el protocolo ERC20. Estas redes de prueba no son más que un entorno de desarrollo que replica el comportamiento de la red principal sin establecer ningún tipo de tasas.

[2] Es fundamental entender que estos protocolos están sujetos a la naturaleza cambiante de un entorno en el que el avance de la innovación sigue un ritmo muy acelerado. Por ello, estos protocolos son susceptibles a actualizaciones futuras. No obstante, para lo que se desea demostrar en este proyecto en fecha actual, esto no supone ningún impedimento.

La arquitectura de las redes de prueba es paralela a la red principal, es decir, se apoya en Blockchain. Sin embargo, en estas redes no existe minería, sino que unos pocos nodos serán los encargados de publicar bloques. Esto sucede porque el valor del *ETH* en este entorno es falso, por lo que no se dispone ningún modelo de incentivo. *Ropsten*, *Kovan*, *Goreli* o *Rinkbeby* son algunas de las redes de prueba públicas que existen [46].

Existen distintas aplicaciones denominadas *Faucets* para inyectar ETH en las cuentas con las que se trabajan en las redes de prueba. En el desarrollo de las pruebas, se ha utilizado la ofrecida por *Chainlink* [47].

Remix IDE

Remix IDE es un entorno de desarrollo online específico para la programación de contratos inteligentes en Ethereum [48]. El sistema ofrece una interfaz muy usable que permite al usuario interaccionar con los contratos de manera muy clara y flexible. Remix IDE es una herramienta especialmente enfocada fundamentalmente desarrolladores principiantes o que desean probar contratos inteligentes desde una interfaz muy intuitiva.

Esta introducción no pretende desvalorizar la herramienta sino todo lo contrario. *Remix IDE* va a permitir comprender el funcionamiento de contratos inteligentes con los que se pretende interactuar y probar el código que se desarrolla. Para comenzar es una herramienta de la que, hasta los desarrolladores más profesionales pueden sacar partido [49].

En adición a la compilación necesaria para poder probar nuestro código, *Remix IDE* también ofrece una prestación de depuración de código. Por otro lado, se puede ejecutar la simulación en diferentes entornos: en local, en redes de prueba o con la red de otros proveedores asociados. En la siguiente figura se ofrece una captura de la interfaz del sistema. Se ha lanzado el contrato de UniswapV2Factory.sol (que se conoce del apartado anterior) a la red de prueba *Rinkeby*. Desde este entorno, se pueden ejecutar las funciones implementadas en la instancia del contrato desplegada y observar los valores que devuelven.

Se obtiene utilidad de *Remix IDE* para probar funciones de contractos con los que interaccionará la aplicación, así como comenzar a desarrollar el código, compilarlo o manipular ciertas funcionalidades. No obstante, la implementación final se llevará a cabo de manera local, siguiendo el modelo de estructura de ficheros de las aplicaciones de contratos inteligentes.

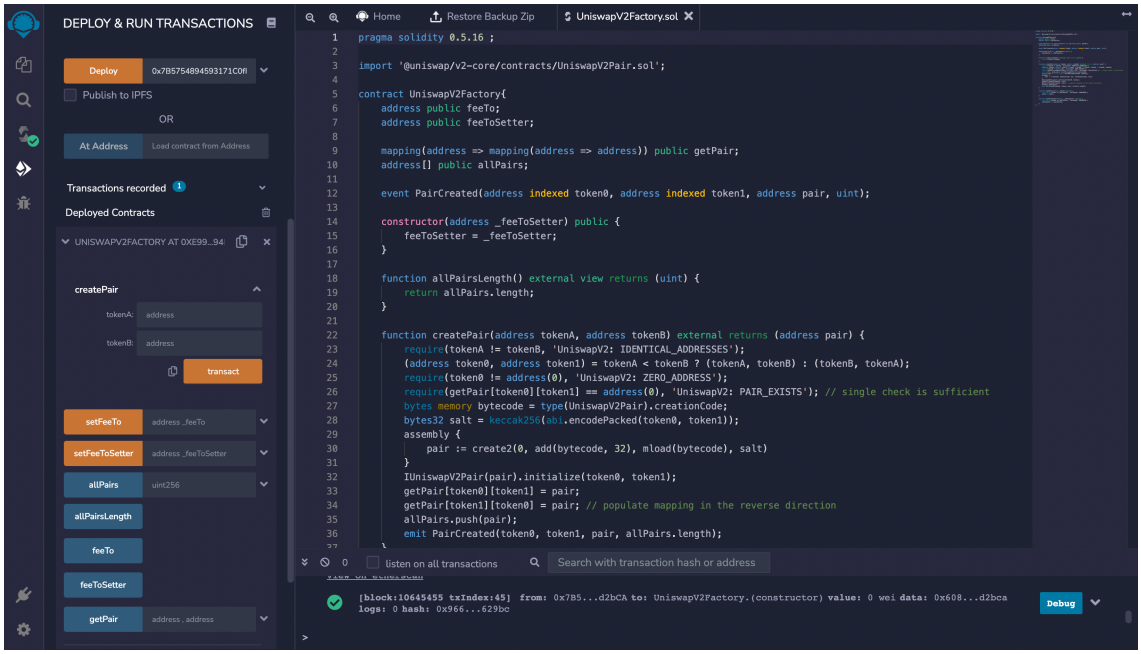


Figura 8: *Interfaz de Remix IDE [38].*

Metamask

Metamask es una cartera virtual que gestiona tokens del protocolo ERC20 (Ethereum). Como cualquier *hot wallet*, funciona como una aplicación que administra los tokens del usuario en línea. No obstante, este monedero no custodia los valores como cabe esperar por su nombre, sino que retiene las claves privadas que permite el acceso a dichos fondos [50].

En términos de seguridad, los *hot wallets* son aplicaciones relativamente vulnerables a ataques ya que operan en línea y, cualquier agente malicioso de la red que logre acceder a las claves que guardan la cartera, dispone de total acceso a los criptoactivos asociados a la cuenta de la cartera.

Por otra parte, la extensión web de *Metamask* [51] la convierte en una herramienta óptima para interaccionar con las *dApps* (Decentralized Apps). De esta forma, *Metamask* funciona como un puente entre las aplicaciones descentralizadas y los usuarios, facilitando la interacción entre ambos [52] [53].

En la siguiente captura se muestra la sencilla interfaz que presenta la extensión web de *Metamask* para *Google Chrome*. Se aprecia que la cartera está conectada a la red *Rinkeby*, aunque ofrece conexión a otras redes incluyendo la *mainnet* de Ethereum. Además de la dirección de nuestra cuenta y las prestaciones visualizadas, se observan los fondos con los que cuenta la cartera. Estos fondos han sido inyectados por el *Faucet* de *Chainlink* [47] mencionado con anterioridad.

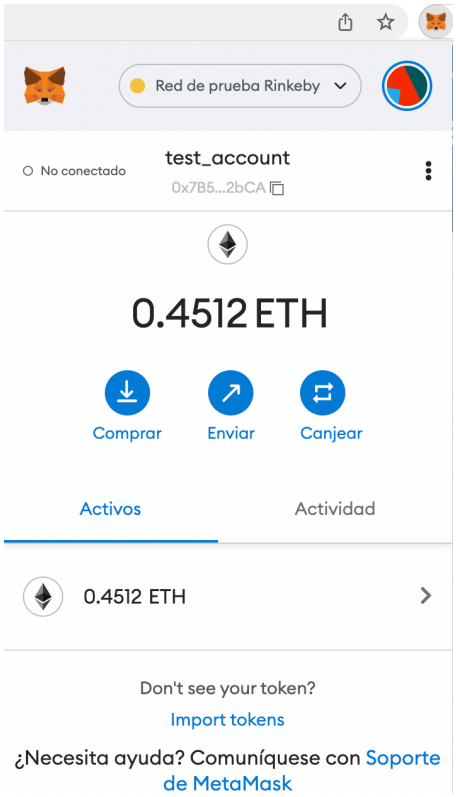


Figura 9: Interfaz de la extensión web de Metamask en Google Chrome.

II. Entornos de desarrollo

A continuación, se presentan los entornos de ejecución y desarrollo en los que se ha desplegado el sistema final y las utilidades complementarias de las que se ha servido.

Partiendo del paquete npm

Node.js es uno entorno en tiempo de ejecución que permite a los desarrolladores de *JavaScript* trabajar en producción sobre el motor V8 que articula a *Google*, sin necesidad de ejecutar la aplicación directamente sobre la web. Resulta atractivo para el desarrollo de proyectos de diversa índole orientados a la Web 2.0. Además, es un entorno código abierto y multiplataforma, motivando la interoperabilidad y simbiosis del contexto. En general Node.js permite gran velocidad y escalabilidad para el prototipado y desarrollo de proyectos agile [54] [55].

npm es un gestor de paquetes diseñado para servir a *Node.js* y funciona como uno de los principales puntos de apoyo de la comunidad *JavaScript*, propiciando uno de los ecosistemas de desarrollo más grandes del mundo. Esto lo logra de la siguiente manera [56] [57]:

1. Es un repositorio online *open source* en el que todos los desarrolladores pueden publicar sus proyectos modularizados en forma de paquetes para su uso compartido.

2. Integra una línea de comando que permite interaccionar con el repositorio de forma que permite, de forma rápida y sencilla, la gestión de paquetes, dependencias o versiones.

El extendido uso de *npm* en la comunidad de desarrolladores, unido a su eficacia y eficiencia en la gestión de paquetes, ha derivado en que posteriores entornos de ejecución dirigidos a contratos inteligentes en *Solidity* se valgan de su utilidad. Por ejemplo, *Truffle* o *Hardhat* se implementa por encima de npm.

Truffle

Truffle es un marco de desarrollo orientado a programadores de la red Ethereum [58]. Esta herramienta facilita notablemente la implementación de smart contracts, ya que agiliza muchos de os factores involucrados en el proceso de producción abstrayendo la responsabilidad o gestión explícita del desarrollador [59]. Por ejemplo:

1. Inicialización de un proyecto *truffle* con una estructura modular de directorios y ficheros organizada según el estándar *npm*, de acuerdo con las necesidades de una aplicación descentralizada implementada con Solidity.
2. Testeo de contratos en base a pruebas implementadas en JavaScript o Solidity.
3. Migración o despliegue del proyecto en la red.
4. Los contratos compilados y/o desplegados en la red se almacenan como "artefacto" en formato .json en el proyecto *truffle* y contendrá información de utilidad como el *ABI*, *bytecode* del contrato o datos asociados al despliegue del contrato como la red en la que se encuentra. Estos metadatos serán importantes para interaccionar con el contrato compilado desde el entorno *truffle*.
5. Este marco de desarrollo hace el proceso de construcción más dinámico desde la consola, ya que ofrece una amplia variedad de comandos intuitivos y flexibles. Además, también es posible utilizar la propia consola de *truffle*, que permite interactuar directamente con el contrato desplegado.

Hardhat

El estudio "Solidity Developer Survey" de 2021 concluyó que *Hardhat* es el entorno de desarrollo más común para los programadores de Ethereum, creciendo muy significativamente con respecto al año anterior, en el que *Truffle* se posicionaba como la plataforma predilecta [60].

Hardhat ofrece unas herramientas muy similares a las de *Truffle*, pero presenta las siguientes ventajas [61]:

- Trabaja con *plugins* más modularizados que los paquetes de Truffle, lo que permite mayor flexibilidad para instalar sólo aquellos complementos que se van a utilizar. Esto da lugar a un proyecto más ligero, evitando la sobrecarga del entorno de ejecución.
- La comunidad que apoya *Hardhat* presenta mayor tasa de crecimiento que la de *Truffle* [62]. Esto es una ventaja para los desarrolladores, a efectos de encontrar

soporte online por parte de la comunidad o documentación precisa, actualizada y de calidad.
- La integración de *plugins* como Ethers.js o *Waffle* hacen que testear contratos inteligentes en *Hardhat* sea más sencillo que en *Truffle*, que requiere una invocación más explícita por parte del desarrollador en *scripts*.

Boilerplate

Uno de los recursos que más facilita la tarea para los desarrolladores de contratos inteligentes o aplicaciones descentralizadas son los *boilerplates*. Son plantillas de proyectos que disponen una estructura de directorios y ficheros predefinidas que ayudan a la modularización y estandarización de los proyectos y al uso de buenas prácticas para la legibilidad del diseño del sistema.

Por otro lado, los *boilerplates* ya contienen múltiples scripts implementados o incluso paquetes que facilitan la resolución de dependencias, las tareas de testing o despliegue en la red. Estos scripts pueden ser modelos fácilmente adaptables y personalizables a las necesidades del proyecto. De igual manera, los paquetes dan apoyo a la tarea de desarrollo con funciones como la exportación de tipos más complejos o la configuración de módulos concretos.

Se ha utilizado un *boilerplate* (de Solidity) ofrecido por *DeFi Wonderland* en GitHub [63]. Este *boilerplate* está construido para una fácil integración en Hardhat e incluye complementos como el SDK *eth-sdk* de *@dethcrypto* [64], Chai [65] o ethers.js [66], que harán mucho más fácil, flexible, legible e intuitiva la tarea de testeo y simulación.

III. Despliegue en la red

Como es de esperar, desplegar el sistema desarrollado en la red Ethereum es una parte fundamental del proceso. Una implementación de contratos inteligentes carece de cualquier valor si no se establece sobre en una red Blockchain. Por ello, bien sea para probar nuestra implementación o para ponerla al servicio de otros usuarios del entorno, interesará ejecutar los contratos de manera descentralizada.

Interacción con los contratos

Ya se mencionó previamente en la sección IV del punto 2 el formato *bytecode* en el que se exporta el código ejecutable en la EVM. No obstante, el compilador de *Solidity* también produce como salida un ABI o *Application Binary Interface*. La salida en *bytecode* queda como una interpretación a bajo nivel del lenguaje de Solidity fuertemente tipada, por lo que la interacción con estos contratos se vuelve sencillamente imposible.

Por el otro lado, la salida en ABI compone una capa intermedia de interpretación o una interfaz que permite al usuario (u otros contratos) interaccionar con el *bytecode* de los contratos lanzados en la red Ethereum. La ABI está definido en formato JSON. De esta manera, este fichero define un esquema o estándar con el que llamar a las funciones de los contratos y recibir sus [67].

Proveedor

Conectarse a la cadena de bloques de la red Ethereum de manera directa solo es posible como nodo participante de la red. Aunque estos nodos tienen capacidad computacional para albergar una copia de la cadena de bloques, el proceso de desplegar contratos en la red, interaccionar con ella o simplemente actualizar la información que alberga puede ser especialmente lento. Más aún, como podría esperarse de lo visto en secciones anteriores, esto puede resultar muy costoso, dado que requiere una inversión en hardware y consumo energético para el dispositivo. Además, el proceso de sincronización como nodo de la red es especialmente lento y complejo. Por tanto, existe una importante limitación para poder desplegar contratos inteligentes en cualquiera de las redes Blockchain que componen la web 3.0.

Como es natural, no todos los desarrolladores de contratos inteligentes cuentan con acceso directo a la red (probablemente casi ninguno). Ante este impedimento, existen herramientas como *Infura* o *Alchemy* que pretenden servir como punto de acceso a la red descentralizada [68].

Infura [69]proporciona la infraestructura y herramientas necesarias a los desarrolladores de aplicaciones descentralizadas para conectarse a la red principal de Ethereum, redes de prueba o IPFS (InterPlanetary File System; otra red descentralizada). Esto no solo permite prescindir del equipo necesario para componer un nodo de Ethereum, sino que además, concede una acceso mucho más rápido a la red y una mayor capacidad de escalar el sistema (dado que no es necesario sobredimensionar la infraestructura) [70]. Más aún, para el uso mínimo que se le dará, el soporte de *Infura* resulta totalmente gratuito.

Hay un aspecto negativo del uso de *Providers*, y es que, valerse de cualquiera de estos puntos de acceso a la red constituye un perjuicio contra el componente de descentralización del sistema, ya que se delega parte de la operatividad del sistema en una tercera parte. Sin embargo, a efectos de lo que se quiere mostrar desde este trabajo, los recursos disponibles y el estado temprano de desarrollo del proyecto, el uso de un *Provider* es lo más conveniente y adecuado. Una mejora de este aspecto puede ser considerada ante una nueva proyección de la implementación [71].

Haciendo una analogía para comprender mejor la aportación de esta API, *Infura* actúa en la red descentralizada de manera similar a *Amazon Web Services* (Cloud Computing) en la red centralizada. *Infura* proporciona acceso a la Web 3.0 construyendo como puente HTTPS y WebSockets (de la Web 2.0) [72].

En la siguiente figura se muestran los puntos de acceso ofrecidos por *Infura* para la *mainnet* de Ethereum:

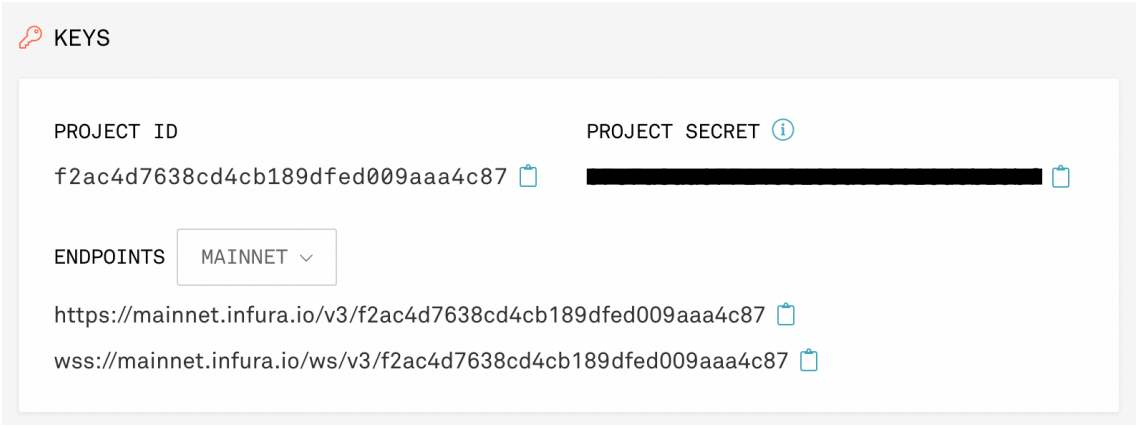


Figura 10: Punto de acceso generado por Infura para la mainnet de Ethereum.

Etherscan

Etherscan [73] es una plataforma web que opera como explorador de la red Blockchain de *Ethereum*. De hecho, fue fundada en los orígenes de la propia red en 2015. Además, cuenta con otras utilidades integradas como una API o *dashboard* analítica. La función principal para la que se originó y para la que se le saca partido mayormente, es que ofrece datos actualizados sobre lo que sucede en esa red *Ethereum*.

Al fin y al cabo, esta herramienta no debe quedar como una utilidad más, sino que es uno de los pilares en los que se sustenta la red *Ethereum* ya que, la transparencia prometida de la cadena de bloques se dispone a través de esta plataforma. Precisamente por ello, *Etherscan* es uno de los primeros proyectos de la red *Ethereum* [74].

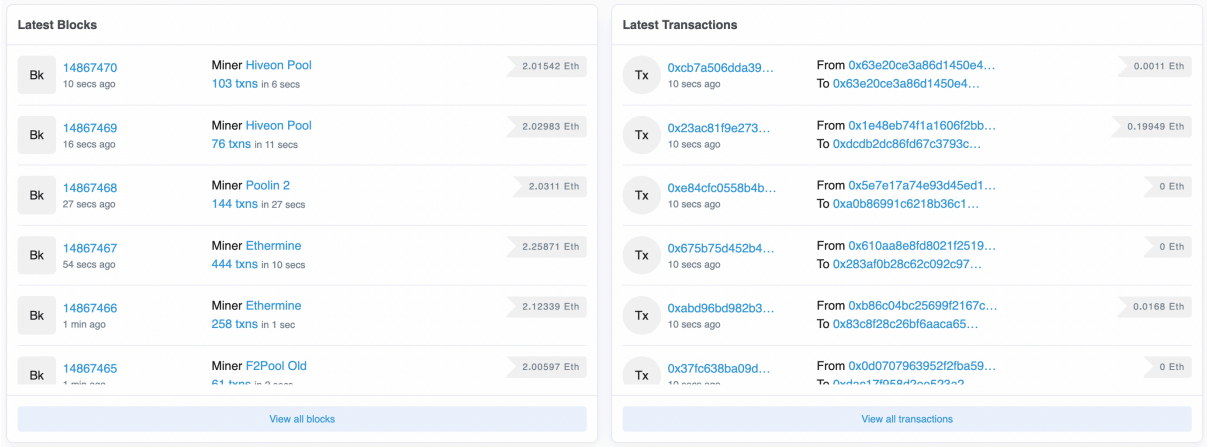


Figura 11: Interfaz de Etherscan.

4.3 Implementación

En esta sección se plasmará el proceso de desarrollo del sistema en contratos inteligentes. Primero se conocerán los requisitos del sistema, se obtendrá una visión general, se verá la propia implementación en Solidity y por último, el testeo y simulación.

I. Análisis de requisitos

Comenzando la explicación el desarrollo llevado a cabo se abordará meramente desde el servicio de los contratos inteligentes, es decir, no se implementará ninguna interfaz gráfica con la que dotar al sistema de gran accesibilidad, sino que para hacer uso de la aplicación se deberá invocar explícitamente el programa. Esta decisión se justifica con que, en definitiva, este sistema no va dirigido a un usuario promedio para el cual adaptar una interfaz sino que:

1. Las interfaces gráficas de las aplicaciones descentralizadas en *DeFi* han surgido de manera más tardía que la implementación del protocolo a bajo niel. La reciente irrupción de todos estos sistemas implica que, en sus primeros momentos de vida, su servicio fuera exclusivamente accesible desde un *script* o programación expresa del usuario. Es decir, se orientaba a usuarios altamente cualificados o, al menos, con unos conocimientos técnicos que le permitiera interaccionar con las plataformas. Con el tiempo, muchos de estos protocolos han acabado construyendo una interfaz gráfica a usuarios menos técnicos para algunos de sus servicios más elementales y populares.
2. Este sistema, al igual que otras tantas implementaciones de soluciones *DeFi*, están proyectados para una integración total con otros contratos inteligentes que hagan viable la ejecución de ciertas operaciones financieras. Es decir, es estrictamente necesario el ensamblaje y despliegue de CIs para realizar operaciones financieras en el entorno actual; al menos para realizarlas de manera eficiente. Por ejemplo, el arbitraje en los mercados descentralizados implica generalmente el uso de *flash loans*; que son exclusivamente programables desde contratos inteligentes.
3. El alcance de este trabajo nunca se consideró el desarrollo de interfaces de usuario que, de partida, plantean unos recursos y necesidades totalmente alejados de la orientación del proyecto. A efectos de lo que se intenta analizar y mostrar desde esta materia, construir una interfaz gráfica resulta irrelevante y disociado. No obstante, se menciona que diseñar un *front-end* que ofrezca servicio desde la Web 2.0 no difiere mucho de las implementaciones bien conocidas de interfaces orientadas a usuarios, y se puede abordar desde herramientas tan genéricas y accesibles como *React*.

Volviendo al sistema desarrollado desde este trabajo para reafirmar que es resulta de lo más corriente que un usuario se beneficie del servicio que se plantea a través de una invocación directa de los contratos inteligentes. Es más, sacar partido del comercio en mercados descentralizados requiere conocer el software subyacente de los CI, Solidity y los atributos técnicos que definen las bases del ecosistema.

Una vez asumido que se prescinde de una interfaz y conociendo la funcionalidad elemental del sistema, se definen los requisitos.

R1. **Devolver el tipo de cambio para una operación de *swap* o precio de un token.** El tipo de cambio devuelto no será más que el ratio A/B, siendo *A* y *B* los dos criptoactivos que componen el par a cambiar, tal que *address*(A) < *address*(B). Se deberán definir cada token de este par, identificados unívocamente en un nivel más bajo por la dirección del contrato inteligente que los implementa. Aprovechando este diseño y con expectativas de ampliar el alcance del Oráculo de Precios, también se devolverá el precio unitario de un token. Los precios se miden en términos de *wei*.

R2. **Ofrecer el mejor tipo de cambio para un swap de entre todas las fuentes del sistema.** El usuario podrá conocer el mejor cambio ofrecido para un swap de entre todas las fuentes de las que se vale el Oráculo.

Para cubrir las funciones básicas del sistema, se establecen los siguientes requisitos.

R3. **Añadir fuente de datos con la que nutrir al Oráculo.** Como es natural, el oráculo sólo funcionará si cuenta con fuentes de las que alimentarse. El usuario podrá asignar una plataforma de preferencia para la consulta de un par de tokens o un único token. Paralelamente, estos recursos se irán anexionando al registro general de fuentes del sistema. En nuestro sistema, las fuentes utilizadas las componen: *Uniswap V2, Uniswap V3, Sushiswap, 1inch, Curve* y *Chainlink*.

R4. **Eliminar fuente.** De la misma forma, también se pueden eliminar "raíces" de las que el Oráculo se nutre.

R5. **Obtener el precio para el swap o token de una de las fuentes añadidas.** Para que el sistema pueda ofrecer información sobre los precios, es necesario que pueda obtenerlo de cada una de las plataformas suministradas.

R6. **Establecer una fuente de datos por defecto.** Se considera una buena práctica que el Oráculo tenga una fuente por defecto, que será una de las opciones provistas. De esta forma, cualquier solicitud al sistema siempre devolverá un valor de precio.

II. Diseño del sistema

El sistema implementado constará de 3 módulos básicos:

1. **Oráculo**. Es el contrato inteligente principal. Ofrece la funcionalidad más básica e inmediata del sistema, es decir, facilitar datos sobre precios. Desde aquí también se podrán añadir o eliminar fuentes, definir el protocolo de referencia por defecto y solicitar el mejor tipo de cambio para un swap específico de entre todos los suministros del Oráculo.
2. **Wrappers**. Estos contratos inteligentes complementarios implementan el método *quote* que se invoca desde el oráculo. De esta forma, la lógica de cada uno de los protocolos de las que se vale el Oráculo se adapta en los *wrappers* para que la mecánica de *quote* sea funcional desde sus respectivas invocaciones a cada una de estas plataformas.

3. **Interfaces**. Las interfaces servirán para abstraer el polimorfismo de todos los *wrappers* a través de la definición de *quote*. Por otro lado, declara métodos específicos de los *source codes* de cada AMM de los que se harán uso. Por último, también se definen eventos.

En este primer acercamiento, se puede visualizar el diseño del sistema en la siguiente figura:

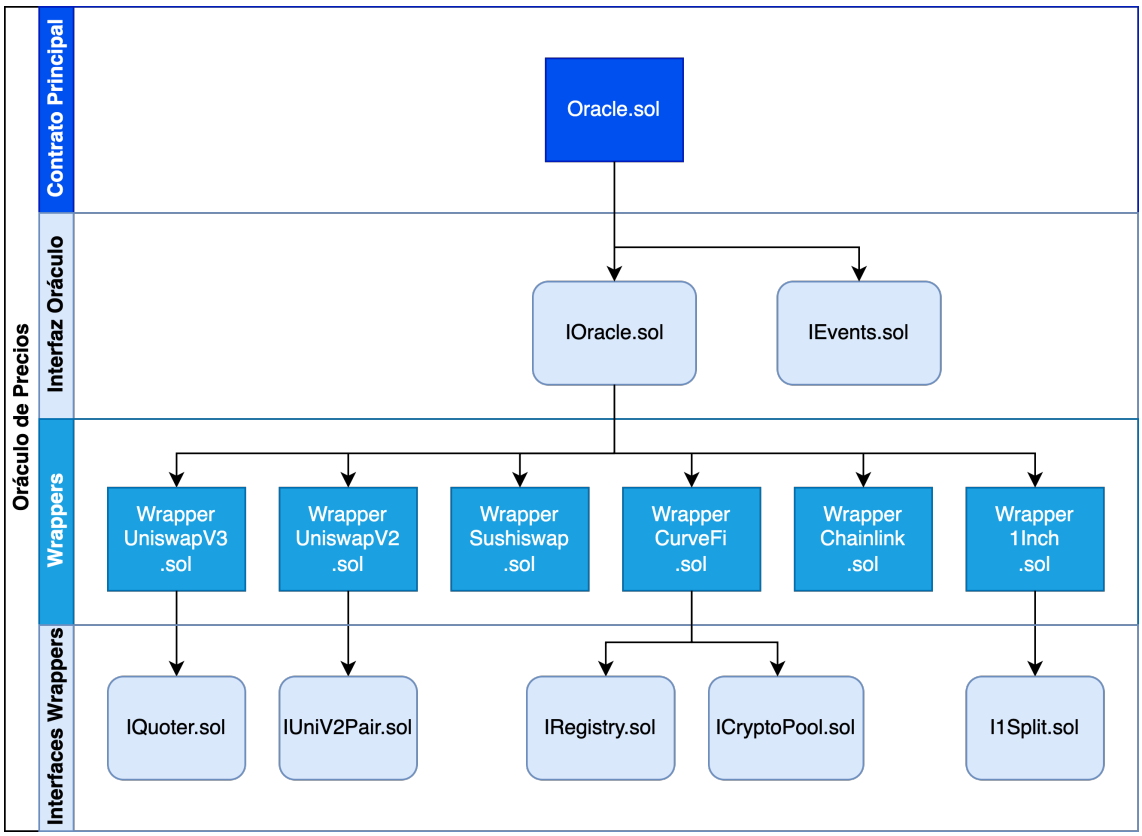


Figura 12: *Visión general del diseño del sistema.*

Desde este momento ya se advierten ciertas propiedades del diseño que se van a adelantar:

1. *Curve* implementa dos interfaces diferentes.
2. *Sushiswap* y *Chainlink* no implementa ninguna interfaz.

Se entenderán estas características del diseño y muchas otras en el siguiente apartado, donde se abordarán la implementación del software.

III. Solución en Contratos Inteligentes

A continuación, se verá detenidamente la programación de los contratos inteligentes en Solidity. El editor utilizado ha sido *Visual Studio Code*, donde se han integrado las herramientas de desarrollo ya mencionadas previamente. El formato mostrado para el código será el mismo que el presentado en *VS Code*. Se agruparán los contratos inteligentes de igual forma que en el diseño del sistema visto en el punto anterior, siguiendo el orden de: Oráculo, *Wrappers* e Interfaces.

Oráculo

Oracle.sol es el contrato inteligente principal, desde dónde se satisfacen de manera más visible todos los requisitos definidos para el sistema a partir de cada una de las funciones definidas. Para ello se hará uso de las siguientes variables a nivel contrato:

- ◊ ***wrappers**: mapping* tridimensional, o lo que es lo mismo, una estructura de datos como *hash table* clave-clave-valor, A-B-C de tipos *address-address-address* para registrar los *wrappers* asociados a cada par.
  A siempre será el criptoactivo cuya dirección (en hexadecimal) sea menor. Este criterio para ordenar el par es una convención en la implementación de diversos AMM para evitar la duplicidad de los pools o registros para un mismo par. Es decir, aunque los *swaps* A $\rightarrow$ B y B $\rightarrow$ A sean operaciones diferentes, el registro de los pares se hace de forma unitaria para generar una matriz triangular, en lugar de una matriz completa con duplicidad de registros.

Siendo conscientes de cómo la ejecución del código implica un coste computacional de EVM que finalmente se traduce en coste económico, medido en términos de gas. De hecho, en el apartado 0, se exponía que incluso los aspectos de legibilidad, lógica o estética se supeditaban a la optimización suprema del código, lo que resultaba en que los contratos inteligentes pueden llegar a ser difíciles de interpretar. Por esta misma razón, en el desarrollo de nuestro sistema se tomarán numerosas elecciones basadas en un criterio de optimización de código. Desde este mismo momento ya se puede mencionar una de ellas.

A efectos formales, la estructura de *mapping* elegida podría haberse constituido en más dimensiones para, por ejemplo, dar capacidad a más direcciones de *wrappers* asociados a cada par. También podría haberse optado por una matriz completa en lugar de una triangular para considerar el sentido de la operación de swaps, vinculando diferentes *wrappers* a cada una de ellas. No obstante, el coste computacional que esto supone puede derivar en un gasto de gas desproporcionado que termina por encarecer el uso de este servicio.

Como ya se sabe, nuestro sistema se va a desplegar desde un *Provider*, lo que permitirá una ejecución totalmente gratuita. Aun así, se considera una buena práctica el priorizar la optimización del software ante todo, con expectativas de integrarlo en sistemas desplegados sobre la red *Ethereum*.

- ◊ ***defaultWrapper****: address* de wrapper por defecto. Con esta variable, se garantiza que cualquier consulta al sistema suministre información de interés y, como consecuencia, ninguna invocación al Oráculo resulte en vano.
- ◊ ***allWrappers***: conjunto de todos los *wrappers* implementado a través de *EnumerableSet* de la librería *openzeppelin* [75]. Desde esta estructura de datos se podrá albergar las direcciones de todos los *wrappers* sin ninguna vinculación a un par o token específico.

Por otro lado, se utilizará la librería *SafeMath* de *openzeppelin* para evitar el *overflow* del tipo uint256 [76].

Las funciones implementadas en este contrato son:

- constructor, donde se define el *wrapper* por defecto.
- ***getAmountOut*** devuelve el tipo de cambio para un par A ⇌ B específico. Desde esta función se invoca a *quote*, declarada por la interfaz *IOracle* e implementada por cada uno de los *wrappers*. Se le da prioridad al tipo de cambio de un par, en caso de que no exista dicho par en el *mapping*, se busca un *wrapper* que pueda devolver el precio del token y en última instancia se consulta el *default wrapper*.
- ***addWrapper*** para añadir un nuevo *wrapper* al Oráculo, tanto a la estructura de preferencias del *mapping* como al conjunto total de *allWrappers*. La definición del tipo *EnumerableSet* es tal que garantiza que no se agregue ningún valor duplicado al conjunto.
- ***setDefaultWrapper*** para definir el *wrapper* por defecto.
- ***removeWrapper*** para eliminar un *wrapper*.
- ***getBestPrice*** concede el mejor tipo de cambio de un par (o precio en caso de ser un solo token) entre todos los *wrappers* añadidos, considerados en *allWrappers*. La cantidad de referencia para establecer el mejor tipo de cambio es 1 wei.

Se menciona que, en cada una de las funciones, se han tenido en cuenta precondiciones que, en caso de no ejecutarse, abortan la ejecución de la función lanzando uno de los eventos definidos en *IEvents*. Se muestra el código del contrato a continuación:

```solidity
//SPDX-License-Identifier: Unlicense
pragma solidity >=0.8.0 <0.9.0;

import 'solidity/interfaces/IOracle.sol';
import 'solidity/interfaces/IEvents.sol';
import '@openzeppelin/contracts/utils/math/SafeMath.sol';
import '@openzeppelin/contracts/utils/structs/EnumerableSet.sol';
import '@openzeppelin/contracts/access/Ownable.sol';

contract Oracle is Ownable, IEvents {
  mapping(address => mapping(address => address)) public wrappers;
  address public defaultWrapper;

  using EnumerableSet for EnumerableSet.AddressSet;
  EnumerableSet.AddressSet private allWrappers;

  using SafeMath for uint256;

  constructor(address _defaultWrapper) {
    defaultWrapper = _defaultWrapper;
  }

  function getAmountOut(
    address _tokenIn,
    uint256 _amount,
    address _tokenOut
  ) external view returns (uint256 _amountOut) {
    if (_amount == 0) revert NotEnoughAmount();
    if (_tokenIn == address(0) || _tokenOut == address(0)) revert InvalidToken();

    (address _token0, address _token1) =
       _tokenIn < _tokenOut ? (_tokenIn, _tokenOut) : (_tokenOut, _tokenIn);

    //Pairs have priority over tokens, and tokens over default
    if (wrappers[_token0][_token1] != address(0)) {
      _amountOut = IOracle(wrappers[_token0][_token1]).quote(_tokenIn, _amount, _tokenOut);
    } else if (wrappers[address(0)][_tokenIn] != address(0)) {
      _amountOut = IOracle(wrappers[address(0)][_tokenIn]).quote(_tokenIn, _amount, _tokenOut);
    } else {
      _amountOut = IOracle(defaultWrapper).quote(_tokenIn, _amount, _tokenOut);
    }

    return _amountOut;
  }

  function addWrapper(
    address _wrapper,
    address _tokenIn,
    address _tokenOut
  ) external onlyOwner {
    (address _token0, address _token1) =
       _tokenIn < _tokenOut ? (_tokenIn, _tokenOut) : (_tokenOut, _tokenIn);
    if (wrappers[_token0][_token1] != address(0)) revert WrapperExists();
    wrappers[_token0][_token1] = _wrapper;
    //Adds only if doesn´t already exists
    allWrappers.add(_wrapper);
    emit WrapperAdded(_wrapper);
  }
```

```
  function setDefaultWrapper(address _wrapper) external onlyOwner {
    if (_wrapper == address(0)) revert WrapperZero();
    defaultWrapper = _wrapper;
    emit WrapperDefaultChanged(_wrapper);
  }

  function removeWrapper(
    address _wrapper,
    address _tokenIn,
    address _tokenOut
  ) external onlyOwner {
    if (wrappers[_tokenIn][_tokenOut] == address(0)) revert InvalidWrapper();
    delete wrappers[_tokenIn][_tokenOut];
    allWrappers.remove(_wrapper);
    emit RemovedWrapper(_wrapper);
  }

  function getBestPrice(address _tokenIn, address _tokenOut) external view returns (uint256
bestAmountOut) {
    if (_tokenIn == address(0) || _tokenOut == address(0)) revert InvalidToken();
    for (uint256 i = 0; i < allWrappers.length(); i++) {
      uint256 amount = IOracle(allWrappers.at(i)).quote(_tokenIn, 1 wei, _tokenOut);
      if (amount > bestAmountOut) {
        bestAmountOut = amount;
      }
    }
    return bestAmountOut;
  }
}
```

Figura 13: Código de Oracle.sol

Wrappers

Desde el apartado del diseño, ya se anunciaba la funcionalidad que ofrecerían los *wrappers*. Estos contratos tendrán el único cometido de implementar la función *quote*, definida en *IOracle.sol*, para devolver el tipo de cambio de un par de criptoactivos específicos desde cada uno de los protocolos considerados como fuentes. Para ello, los *wrappers* se valdrán de las funciones ofrecidas por las librerías de cada una de estas plataformas.

Uno de los aspectos más resaltados en estos contratos es lo siguiente: la consulta del cambio del swap debe considerar el descuento de las tasas del sistema. Es decir, se obtendrá una cantidad *B* de cambio por un ingreso de (*A*-comisión) a partir de la magnitud total de *A* de la que se dispone. Por ello, en caso de que las funciones de cada plataforma a las que se llamen desde la API no considere la reducción del cambio por la comisión, se debe gestionar explícitamente desde el código.

Nuestro sistema especifica cinco contratos inteligentes de *wrapper* para *Uniswap V2*, *Uniswap V3*, *Sushiswap*, *Curve*, *1inch* y *Chainlink*. Se muestran en ese orden en las siguientes figuras:

```solidity
//SPDX-License-Identifier: Unlicense
pragma solidity >=0.8.0 <0.9.0;

import 'solidity/interfaces/IUniV2Pair.sol';

contract WrapperUniswapV2 {
  address constant UNISWAP_FACTORY = 0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f;

  function quote(
    address _tokenIn,
    uint256 _amount,
    address _tokenOut
  ) external view returns (uint256 amountOut) {
    //Sort tokens
    (address _token0, address _token1) = _tokenIn < _tokenOut ? (_tokenIn, _tokenOut) : (_tokenOut,
_tokenIn);

    //PairFor using the UniswapV2 factory
    address _pair = address(
      uint160(
        uint256(
          keccak256(
            abi.encodePacked(
              hex'ff',
              UNISWAP_FACTORY,
              keccak256(abi.encodePacked(_token0, _token1)),
              hex'96e8ac4277198ff8b6f785478aa9a39f403cb768dd02cbee326c3e7da348845f' // init code hash
            )
          )
        )
      )
    );

    //GetReserves with the previous pair
    (uint112 _reserve0, uint112 _reserve1, ) = IUniV2Pair(_pair).getReserves();

    //GetAmountOut
    uint256 _amountInWithFee = _amount * (997);
    uint256 _numerator = _amountInWithFee * (_reserve1);
    uint256 _denominator = _reserve0 * (1000) + (_amountInWithFee);
    return amountOut = _numerator / _denominator;
  }
}
```

Figura 14: *WrapperUniswapV2.sol*

Una de las particularidades del contrato *WrapperUniswapV2.sol* es visible en el código destacado en el recuadro. Este código es una réplica de la función *pairFor* dispuesta desde *UniswapV2Library.sol* [77]. La razón por la que se ha extraído esta función en este contrato viene motivada por problemas de compatibilidad con la versión de Solidity utilizada en este proyecto (>=0.8.0 <0.9.0). Concretamente la conversión de tipos *uint* es el origen del conflicto; esta conversión está soportada en antiguas versiones de Solidity, pero no en las más recientes. Por ello, desde este contrato se realiza la conversión conforme al estándar impuesto en la versión utilizada.

Es significativo subrayar que *WrapperSushiswap.sol* se hace evidente que *Sushiswap* es un *fork* de *Uniswap V2*: el primero reutiliza las herramientas dispuestas en el protocolo original. Se ve a continuación:

```
//SPDX-License-Identifier: Unlicense
pragma solidity >=0.8.0 <0.9.0;

import '@sushiswap/core/contracts/uniswapv2/interfaces/IUniswapV2Pair.sol';
import '@sushiswap/core/contracts/uniswapv2/interfaces/IUniswapV2Factory.sol';

contract WrapperSushiswap {
  IUniswapV2Factory public constant SUSHIFACTORY =
IUniswapV2Factory(0xC0AEe478e3658e2610c5F7A4A2E1777cE9e4f2Ac);

  function quote(
    address _tokenIn,
    uint256 _amount,
    address _tokenOut
  ) external view returns (uint256 amountOut) {
    //Sort tokens
    (address _token0, address _token1) = _tokenIn < _tokenOut ? (_tokenIn, _tokenOut) : (_tokenOut,
_tokenIn);

    IUniswapV2Pair pair = IUniswapV2Pair(SUSHIFACTORY.getPair(_token0, _token1));
    //GetReserves with the previous pair
    (uint112 _reserve0, uint112 _reserve1, ) = IUniswapV2Pair(pair).getReserves();

    //GetAmountOut
    uint256 _amountInWithFee = _amount * (997);
    uint256 _numerator = _amountInWithFee * (_reserve1);
    uint256 _denominator = _reserve0 * (1000) + (_amountInWithFee);
    return amountOut = _numerator / _denominator;
  }
}
```

Figura 15: WrapperSushiswap.sol

```
//SPDX-License-Identifier: Unlicense
pragma solidity >=0.8.0 <0.9.0;

import '@uniswap/v3-periphery/contracts/libraries/PoolAddress.sol';
import '@uniswap/v3-core/contracts/interfaces/IUniswapV3Pool.sol';
import 'solidity/interfaces/IQuoter.sol';

contract WrapperUniswapV3 {
  address immutable quoter;

  constructor(address _quoter) {
    quoter = _quoter;
  }

  function quote(
    address _tokenIn,
    uint256 _amount,
    address _tokenOut
  ) external view returns (uint256 amountOut) {
    (address _token0, address _token1) = _tokenIn < _tokenOut ? (_tokenIn, _tokenOut) : (_tokenOut,
_tokenIn);
    IUniswapV3Pool _pool = IUniswapV3Pool(
      PoolAddress.computeAddress(0x1F98431c8aD98523631AE4a59f267346ea31F984,
PoolAddress.getPoolKey(_token0, _token1, 3))
    );
    bytes memory _path = abi.encodePacked(_pool);
    uint256 _amountOut = IQuoter(quoter).quoteExactInput(_path, _amount);
    return _amountOut;
  }
```

Figura 16: WrapperUniswapV3.sol

```
//SPDX-License-Identifier: Unlicense
pragma solidity >=0.8.0 <0.9.0;

import 'solidity/interfaces/IRegistry.sol';
import 'solidity/interfaces/ICryptoPool.sol';

contract WrapperCurveFi {
  address immutable registry;

  constructor(address _registry) {
    registry = _registry;
  }

  function quote(
    address _tokenIn,
    uint256 _amount,
    address _tokenOut
  ) external view returns (uint256 amountOut) {
    address poolCoins = IRegistry(registry).find_pool_for_coins(_tokenIn, _tokenOut, 0);
    (int128 fromIndex, int128 toIndex, ) = IRegistry(registry).get_coin_indices(poolCoins, _tokenIn,
_tokenOut);

    return ICryptoPool(poolCoins).get_dy(fromIndex, toIndex, _amount);
  }
}
```

Figura 17: WrapperCurveFi.sol

1inch implementa su protocolo internamente de tal manera que los criptoactivos que soporta tiene asignado un índice. Además, esta modularización se hace evidente en el uso de dos interfaces diferentes, razón por la cual se construyen dos contratos inteligentes disjuntos para definir cada una de estas interfaces.

```
//SPDX-License-Identifier: Unlicense
pragma solidity >=0.8.0 <0.9.0;

import 'solidity/interfaces/I1Split.sol';

contract Wrapper1Inch {
  address immutable oneSplit;

  constructor(address _oneSplit) {
    oneSplit = _oneSplit;
  }

  function quote(
    address _tokenIn,
    uint256 _amount,
    address _tokenOut
  ) external view returns (uint256 amountOut) {
    //Split volume in 1 part. Default flags value = 0
    (amountOut, ) = I1Split(oneSplit).getExpectedReturn(_tokenIn, _tokenOut, _amount, 1, 0);
  }
}
```

Figura 18: Wrapper1Inch.sol

```solidity
//SPDX-License-Identifier: Unlicense
pragma solidity >=0.8.0 <0.9.0;

import '@chainlink/contracts/src/v0.8/interfaces/FeedRegistryInterface.sol';

contract WrapperChainlink {
  FeedRegistryInterface internal registry;

  constructor(address _registry) {
    registry = FeedRegistryInterface(_registry);
  }

  function quote(
    address _tokenIn,
    uint256 _amount,
    address _tokenOut
  ) external view returns (uint256 amountOut) {
    (, int256 price, , , ) = registry.latestRoundData(_tokenIn, _tokenOut);
    return _amount * uint256(price);
  }
}
```

Figura 19: WrapperChainlink.sol

Interfaces

Las interfaces son contratos bastante sencillos y breves. Sobre todo, esto es así porque no se ofrece ninguna implementación completa de código desde estos módulos sino que, tal y como sucede en la naturaleza de las interfaces de otros lenguajes de programación, simplemente se declaran funciones o variables.

```solidity
//SPDX-License-Identifier: Unlicense
pragma solidity >=0.8.0 <0.9.0;

interface IOracle {
  function quote( address _tokenIn, uint256 _amount, address _tokenOut ) external view
    returns ( uint256 amountOut );
}
```

Figura 20: IOracle.sol

```solidity
//SPDX-License-Identifier: Unlicense
pragma solidity >=0.8.0 <0.9.0;

interface IEvents {
  error NotEnoughAmount();
  error WrapperExists();
  error InvalidToken();
  error WrapperZero();
  error InvalidWrapper();

  event WrapperAdded(address wrapper);
  event WrapperDefaultChanged(address wrapper);
  event RemoveWrapper(address wrapper);
}
```

Figura 21: IEvents.sol

Además de la función *quote*, utilizada por *Oracle.sol* e implementada por cada *wrapper*, se definen diferentes eventos, utilizados también por el contrato principal del oráculo. La incorporación de eventos específicos en el desarrollo software siempre compone una buena práctica para:

a) Orientar al usuario en caso de que invoque de manera errónea las funciones facilitadas desde nuestro sistema.
b) Facilitar la trazabilidad de la ejecución de los procesos de nuestra aplicación y el correcto tratamiento en caso de interrupción.

Por otro lado, y con otro cometido totalmente diferente, se crean interfaces para declarar los métodos utilizados por algunos de los *wrappers*. Estos métodos están implementados en el *open source* de cada una de estas plataformas y están a completa disposición de los desarrolladores para poder interaccionar con los respectivos AMM que constituyen. La declaración de estas funciones en estas interfaces permite la invocación desde el código que se ha desarrollado sin la necesidad de importar todas las de estos protocolos.

Llegados a este punto es importante tratar el criterio elegido para construir interfaces en nuestro sistema. En primer lugar, se debe entender que, a medida que nuestro software se valga de más dependencias externas o importaciones de librerías, se genera mayor sobrecarga de la ejecución del sistema. Se observa que en algunos contratos se importan directamente las librerías, mientras que en otros se importan las interfaces implementadas expresamente. Una vez más, las pauta a tener en cuenta es maximizar la optimización del código. Se explican las decisiones tomadas:

1. Si el *wrapper* se vale de pocas funciones (tres o menos) que son fácilmente declarables, se diseña una interfaz que permita al compilador de *Solidity* reconocer dichas invocaciones para que, en el momento de ejecución, se comunique adecuadamente con el protocolo que utiliza el wrapper. Esto sucede con *WrapperUniswapV2.sol* o *Wrapper1Inch.sol*.
2. Si el *wrapper* realiza numerosas llamadas a funciones de librerías o cuenta con muchas dependencias externas, resultará más compleja la implementación de interfaces propias y, por lo tanto, se procederá a importar las librerías correspondientes que definen dichas funciones.
3. Por último, en el caso *de WrapperChainlink.sol*, se hace uso de dos interfaces de programación propia: *IRegistry.sol* y *ICryptoPool.sol*. Esto atiende a la modularización del *open source* de 1inch ya mencionada previamente.

Con esta finalidad de maximizar la eficiencia del código, la librería utilizada de Uniswap V3 se vincula a través del *open source* facilitado por @Ibertenasco en github.com [78]. Esta implementación no es más que un *middleware* que optimiza las llamadas a *periphery* y *core* de Uniswap V3. Este diseño está sujeto al "bug bounty program" de Uniswap V3 que pretende resolver los errores e ineficiencias operacionales de su plataforma [79].

```solidity
// SPDX-License-Identifier: MIT
pragma solidity >=0.8.0 <0.9.0;

interface IUniV2Pair {
  function getReserves() external view
    returns ( uint112 reserve0, uint112 reserve1, uint32 blockTimestampLast );
}
```

Figura 22: UniV2Pair.sol

```solidity
//SPDX-License-Identifier: Unlicense
pragma solidity >=0.8.0 <0.9.0;

interface IQuoter {
  function quoteExactInput( bytes memory path, uint256 amountIn ) external view
    returns ( uint256 amountOut );
}
```

Figura 23: IQuoter.sol

```solidity
//SPDX-License-Identifier: Unlicense
pragma solidity >=0.8.0 <0.9.0;

interface IRegistry {
  function find_pool_for_coins( address _from, address _to, uint256 i ) external view
    returns ( address pool );

  function get_coin_indices( address pool, address _from, address _to ) external view
    returns ( int128, int128, bool );

  function get_dy( int128 i, int128 j, uint256 _dx ) external view
    returns (uint256 amountOut);
}
```

Figura 24: IRegistry.sol

```solidity
//SPDX-License-Identifier: Unlicense
pragma solidity >=0.8.0 <0.9.0;

interface ICryptoPool {
  function get_dy( int128 i, int128 j, uint256 _dx ) external view
    returns (uint256 amountOut);
}
```

Figura 25: ICryptoPool.sol

```solidity
//SPDX-License-Identifier: Unlicense
pragma solidity >=0.8.0 <0.9.0;

interface I1Split {
  function getExpectedReturn( address fromToken, address destToken, uint256 amount, uint256 parts,
uint256 flags )
    external view
    returns (uint256 returnAmount, uint256[] memory distribution);
}
```

Figura 26: I1Split.sol

4.4 Testeo y Simulación

Es imprescindible desplegar un plan de pruebas sobre el software para asegurarnos de su correcto funcionamiento. Se divien los tests en unitarios y e2e.

I. Tests unitarios

Desde los tests unitarios se comprobará el correcto funcionamiento del sistema a un nivel más elemental. No se comprueba tanto su ejecución adecuada en el entorno real dónde se desplegará, sino que se testean exclusivamente las funcionalidades más específicas de manera agnóstica el ecosistema Blockchain. Para ello, se falseará o simularán los valores esperados de los resultados con tal de que las llamadas a las funciones devuelvan resultados sin necesidad de sean fieles al entorno real. Se separan los tests unitarios en dos módulos. Por un lado, se hacen las siguientes pruebas para el contrato principal de *Oracle.sol*:

- Función constructor ha inicializado los parámetros correspondientes.
- Función *getAmountOut:*
  - ◊ Se aborta con *NotEnoughAmount()* si se le introduce una cantidad nula.
  - ◊ Se aborta con I*nvalidToken()* si la dirección *_tokenIn* o *_tokenOut* vale 0.
  - ◊ Se ejecuta con la entrada de un par de tokens y una cantidad como argumentos.
  - ◊ Se ejecuta con la entrada de un solo token y una cantidad como argumentos.
  - ◊ Se ejecuta con la entrada de ningún token y una cantidad.
- Función *addWrapper*:
  - ◊ Se aborta con *WrapperExists()* si ya existe un *wrapper* asociado a dicho par.
  - ◊ Se ejecuta si no existe ningún *wrapper* para dicho par.
- Función *setDefaultWrapper*:
  - ◊ Se aborta con *WrapperZero()* si añadiendo un *wrapper* con dirección nula.
  - ◊ Se ejecuta si se le pasa como argumento una dirección no nula.

Por otra parte, también se hacen tests unitarios sobre los wrappers para asegurarnos de que las llamadas a *quote* desde cada uno de ellos invocan las funciones asociadas al *open source* de cada una de las plataformas. Para ello, se deberán falsear las llamadas en base a interfaces "mocks".

Los tests unitarios se pueden ver en el Anexo 1: Tests Unitarios.

II. Tests e2e

Habiendo probado las especificaciones individuales a la funcionalidad de la aplicación en conjunto desde los tests e2e o *End to End*. Los tests unitarios son fundamentales, previos a los tests e2e, pero son estos últimos los que resultan más interesantes ya que, desde aquí se podrá simular el comportamiento del sistema para obtener su funcionamiento real. En la siguiente figura se muestra el código que permite simular el despliegue del contrato en la red:

```
import { getMainnetSdk } from '@dethcrypto/eth-sdk-client';
import { Dai, CurveRegistry, OneSplit, Usdt, Uni, Weth } from '@eth-sdk-types';
import { BigNumber } from 'ethers';
import { ethers } from 'hardhat';
import { evm } from '@utils';
import { expect } from 'chai';
import { SignerWithAddress } from '@nomiclabs/hardhat-ethers/dist/src/signer-with-address';
import { toUnit } from '@utils/bn';

import {
  Oracle,
  Oracle__factory,
  Wrapper1Inch,
  Wrapper1Inch__factory,
  WrapperCurveFi,
  WrapperCurveFi__factory,
  WrapperSushiswap,
  WrapperSushiswap__factory,
  WrapperUniswapV2,
  WrapperUniswapV2__factory,
  WrapperUniswapV3,
  WrapperUniswapV3__factory,
  WrapperChainlink,
  WrapperChainlink__factory
} from '@typechained';

const NOTOKEN = ethers.constants.AddressZero;

describe('Oracle', function () {
  let governor: SignerWithAddress;

  const FORK_BLOCK_NUMBER = 14133214;

  let curveRegistry: CurveRegistry;
  let oneSplit: OneSplit;
  let dai: Dai;
  let usdt: Usdt;
  let uni: Uni;
  let weth: Weth;
  let quoter: Quoter;
  let feedRegistry: FeedRegistry;

  let snapshotId: string;

  // All wrappers
  let oracleFactory: Oracle__factory;
  let oracle: Oracle;
  let wrapper1InchFactory: Wrapper1Inch__factory;
  let wrapper1Inch: Wrapper1Inch;
  let wrapperCurveFiFactory: WrapperCurveFi__factory;
  let wrapperCurveFi: WrapperCurveFi;
  let wrapperSushiFactory: WrapperSushiswap__factory;
  let wrapperSushi: WrapperSushiswap;
  let wrapperUniV2Factory: WrapperUniswapV2__factory;
  let wrapperUniV2: WrapperUniswapV2;
  let wrapperUniV3Factory: WrapperUniswapV3__factory;
  let wrapperUniV3: WrapperUniswapV3;
  let wrapperChainlinkFactory: WrapperChainlink__factory;
  let wrapperChainlink: WrapperChainlink;
```

```
    before(async () => {
     await evm.reset({
       jsonRpcUrl: process.env.NODE_URI_MAINNET,
       blockNumber: 14867676,
     });

     [governor] = await ethers.getSigners();
     const sdk = getMainnetSdk(governor);

     curveRegistry = sdk.curveRegistry;
     oneSplit = sdk.oneSplit;
     dai = sdk.dai;
     usdt = sdk.usdt;
     uni = sdk.uni;
     weth = sdk.weth;
     quoter = sdk.quoter;
     feedRegistry = sdk.feedRegistry;

     // Deploy oracle contract and all wrappers
     oracleFactory = await ethers.getContractFactory('Oracle');
     oracle = await oracleFactory.deploy(ethers.constants.AddressZero);

     wrapper1InchFactory = await ethers.getContractFactory('Wrapper1Inch');
     wrapper1Inch = await wrapper1InchFactory.deploy(sdk.oneSplit.address);

     wrapperCurveFiFactory = await ethers.getContractFactory('WrapperCurveFi');
     wrapperCurveFi = await wrapperCurveFiFactory.deploy(sdk.curveRegistry.address);

     wrapperSushiFactory = await ethers.getContractFactory('WrapperSushiswap');
     wrapperSushi = await wrapperSushiFactory.deploy();

     wrapperUniV2Factory = await ethers.getContractFactory('WrapperUniswapV2');
     wrapperUniV2 = await wrapperUniV2Factory.deploy();

     wrapperUniV3Factory = await ethers.getContractFactory('WrapperUniswapV3');
     wrapperUniV3 = await wrapperUniV3Factory.deploy(sdk.quoter.address);

     wrapperChainlinkFactory = await ethers.getContractFactory('WrapperChainlink');
     wrapperChainlink = await wrapperChainlinkFactory.deploy(sdk.feedRegistry.address);

     snapshotId = await evm.snapshot.take();
   });

   beforeEach(async () => {
     await evm.snapshot.revert(snapshotId);
     await oracle.connect(governor).addWrapper(wrapperSushi.address, weth.address, usdt.address);
     await oracle.connect(governor).setDefaultWrapper(wrapper1Inch.address);
   });

   describe('getTotalAmount', () => {
     let quote: BigNumber;
     it('Should quote USDT-WETH with sushiSwap', async () => {
       quote = await oracle.getAmountOut(weth.address, toUnit(1), usdt.address);
       console.log(quote.toNumber());
       expect(quote).to.not.be.eq(0);
     });

   it('Should quote USDT-WETH with UniswapV2', async () => {
       await oracle.connect(governor).removeWrapper(wrapperSushi.address, weth.address, usdt.address);
       await oracle.connect(governor).addWrapper(wrapperUniV2.address, weth.address, usdt.address);
       quote = await oracle.getAmountOut(weth.address, toUnit(1), usdt.address);
       console.log(quote.toNumber());
       expect(quote).to.not.be.eq(0);
     });

   });
});
```

Figura 27: e2e/oracle.specs.ts

III. Resultados del test y simulación

A continuación se visualizan las salidas de los tests unit y e2e por separado. Además de la confirmación satisfactoria de los tests, se observa para el segundo caso la salida de precios (en wei) programada para WETH/USDT en *sushiswap* y *Uniswap V2*.

```
alvarojimenez@MBP-de-Alvaro oracle_v4 % yarn test:unit
yarn run v1.22.18
$ yarn test 'test/unit/**/*.spec.ts'
$ yarn compile:test && cross-env TEST=true mocha 'test/unit/**/*.spec.ts'
$ cross-env TEST=true hardhat compile
No etherscan api key for ethereum
Nothing to compile
No need to generate any newer typings.
No etherscan api key for ethereum


  Oracle
    constructor
      ✔ should parameters are valid
    getAmountOut
      ✔ should revert if amount is 0
      ✔ should revert if token_in/out is 0 address (46ms)
      ✔ should return getAmountOut with a pair (39ms)
      ✔ should return getAmountOut with tokenIn (41ms)
      ✔ should return getAmountOut with default
    addWrapper
      ✔ should revert if wrapper exists
      ✔ should add wrapper (65ms)
    swapDefaultWrapper
      ✔ should revert is wrapper address is 0x0
      ✔ should change default wrapper (46ms)

  Wrapper1Inch
    oneInch quote
      ✔ should return correct quote using registry

  WrapperChainlink
    chainlink quote
      ✔ should return correct quote using registry

  WrapperCurve
    curveFi quote
      ✔ should return correct quote using registry (44ms)

  WrapperSushiswap
    sushiswap quote
      ✔ should return correct quote (46ms)

  WrapperUniswapV2
    uniswapV2 quote
      ✔ should return correct quote (45ms)

  WrapperUniswapV3
    uniswapV3 quote
      ✔ should return correct quote


  16 passing (2s)
```

Figura 28: Salida de tests unitarios.

```
alvarojimenez@MBP-de-Alvaro oracle_v4 % yarn test:e2e
yarn run v1.22.18
$ yarn test './test/e2e/**/*.spec.ts'
$ yarn compile:test && cross-env TEST=true mocha './test/e2e/**/*.spec.ts'
$ cross-env TEST=true hardhat compile
No etherscan api key for ethereum
Nothing to compile
No need to generate any newer typings.
No etherscan api key for ethereum


  Oracle
    getTotalAmount
1799123390
      ✔ Should quote USDT-WETH with sushiSwap
1797552333
      ✔ Should quote USDT-WETH with UniswapV2


  2 passing (3s)
```

Figura 29: Salida de tests e2e.

Se concluye que los precios devueltos por los AMM *Uniswap* V2 y *Sushiswap* para el swap USDT-WETH son 1799123390 y 179755233 respectivamente, ofreciendo *Uniswap* V2 un cambio más barato, con una diferencia de:

$$1799123390 - 179755233 = 1619368157 \text{ wei}$$

Desde este test, se pueden consultar explícitamente los precios de un par o token para cualquiera de las 6 plataformas implementadas en los wrappers [3].

[3] En la consulta, el usuario debe tener consciencia sobre los argumentos que dispone en la solicitud de precios. No todas las plataformas manejan cualquier token. Por ejemplo, una consulta de un token no estable en CurveFi devolvería error, porque este protocolo no contempla el comercio con dicho criptoactivo.

5 Conclusión y trabajos futuros

Como se ha podido observar, la contribución actual y potencial de Blockchain es amplia y variada. Muchas de las ramas se encuentran en pleno proceso de expansión, mientras que otras tantas se sitúan en un estadio mucho más precoz. Aunque no se puede prometer la supremacía futura de Blockchain, tal y como se conoce hoy, sí que somos optimistas para afirmar que esta tecnología disruptiva ha asentado las bases para un futuro tecnológico mucho más eficiente, democrático y seguro.

Viajando desde los fundamentos que capacitan a esta tecnología distribuida hasta sus posibles aplicaciones, se ha obtenido una visión elemental de su poder hasta visualizar su alcance en casos más prácticos. Además, se consideraba necesaria una exposición de objeciones, limitaciones o debilidades en las que cae Blockchain; más aun teniendo en cuenta la inmadurez que pesa sobre esta tecnología. Este recorrido ha introducido una tecnología que, como ya se adelantaba desde el inicio, es muy desconocida a la par que muy exaltada y cuestionada. En definitiva, sobra la exageración y la promoción de Blockchain a la vez que los intentos de menospreciar o envilecer su aportación.

En segunda instancia, se ha explorado el concepto de Contratos Inteligentes. Los contratos inteligentes son imprescindibles para la implementación de cualquier tipo de sistema, aplicación o producto descentralizado construido encima del protocolo y criptomoneda nativa a algunas redes como Ethereum. De hecho, prácticamente la totalidad de los casos ejemplificados requieren de contratos inteligentes para convertirse en utilidades reales.

Se ha incidido en cómo el software de los contratos inteligentes permite formalizar en código una relación contractual de manera inquebrantable. Este programa permitirá compensar a las partes involucradas en el contrato de manera automática, sin la intervención de la mano humana. Más aún, su ejecución se realiza de forma descentralizada, lo que favorece la seguridad y transparencia de la transacción pertinente. Por último, la atomicidad con la que se lleva a cabo el procedimiento considerado en el contrato evita problemas asociados a la latencia y otros riesgos operacionales. Como consecuencia, los contratos inteligentes son una manera de asegurar el cumplimiento de un contrato sin exponerse a los costes, obstáculos y amenazas asociados a la intermediación o gestión de terceras partes involucradas.

Además, se han estudiado diferentes redes Blockchain para llegar a la siguiente resolución: *Bitcoin* es la red pionera, precursora y paradigmática de esta tecnología; no obstante, resulta la menos adecuada para desarrollar aplicaciones descentralizadas. Por el contrario, *Ethereum* ofrece los atributos, recursos y herramientas idóneos para desplegar contratos inteligentes y, de hecho, es la red predilecta para este fin. Más aún, fue el entorno que propició el origen de las Finanzas Descentralizadas y, en el momento de realizar este trabajo, sigue siendo la plataforma donde las *DeFi* tienen mayor presencia.

Las mismas facultades que hacen que los CI sean tan interesantes pueden ser origen de trabas o contratiempos, por lo que es importante estar al tanto de los factores que caracterizan y condicionan este software, así como su alcance y consecuencias. No cualquier aplicación se puede optimizar al implementarse de manera descentralizada; al menos no en el contexto actual. Como prueba de ello se han constatado en el trabajo todas las limitaciones y problemáticas que existen en torno al desarrollo de contratos inteligentes en un contexto inicial y temprano de expansión, innovación y actualización.

El Oráculo de Precios expandido en el punto 4, sirve como ejemplo de software descentralizado, implementado a través de contratos inteligentes en la red Ethereum. Para empezar, los recursos presentados para abordar esta parte del trabajo han ayudado a ejemplificar la dinámica y partes involucradas en el despliegue de un sistema en la red Ethereum.

Por otro lado, ha quedado patente lo accesible que resulta la documentación y *open source* de los *Automated Market Makers* en los que se ha apoyado el oráculo mientras que se ha acercado a la arquitectura elemental de este tipo de plataformas. Con esta implementación no solo se ha aproximado al desarrollo de contratos inteligentes orientados a *DeFi*, sino que también ha quedado demostrada la interoperabilidad que caracteriza este ecosistema, lo que constituye uno de los atractivos principales de los mercados descentralizados.

Puede que al lector le haya impresionado la simplicidad de la implementación abordada. Lo cierto es que el sistema presentado no requiere de mayor infraestructura software que la construida para lograr cubrir los requerimientos que se habían planteado. Ya se sabe que la función de un Oráculo no es otra que reflejar información obtenida de fuentes externas, y teniendo en cuenta que gran parte de las funcionalidades de las que se vale la aplicación ya están integradas en el *open source* de los AMM de los que se alimenta, el trabajo de programación realizado no tiene por qué ir mucho más allá de lo expuesto. Al fin y al cabo, el sistema implementado es una API que incorpora diferentes DEXes o protocolos del entorno, para servir como un comparador de precios en mercados descentralizados. El valor de esta aportación radica en que se puede poner al alcance del usuario información de interés para realizar operaciones financieras. El Oráculo de Precios provee, desde una misma plataforma, información descentralizada, real, fresca, fiable y exacta de manera totalmente automatizada.

La sencillez de la construcción de la aplicación también es reflejo de la optimización que se ha intentado alcanzar en torno al desarrollo de contratos inteligentes. La expresividad del lenguaje de programación Solidity, los entornos de ejecución, la provisión de librerías o la compatibilidad con otras herramientas de desarrollo y sistemas del entorno son algunos de los factores que han propiciado esa comodidad y viabilidad a la hora de desplegar el sistema. Por otra parte, la simpleza del código busca facilitar la exposición del trabajo y ser coherente con el calibre que pretende abarcar. Dicho esto, es conveniente subrayar que el desarrollo de una aplicación mediante contratos inteligentes puede volverse una tarea mucho más compleja, técnica y avanzada. Además, tal y como se ha expuesto anteriormente, la novedad e inmadurez

de este ámbito se hace evidente en la escasez de código de referencia, documentación, apoyo de la comunidad o, en general, información extensa y actualizada accesible vía online.

La ejecución del Oráculo de Precios ha dado el resultado esperado sin mayor dilación: el tipo de cambio de un *swap* en plataformas diferentes. Esta operatividad ha quedado más que satisfecha. El servicio ofrecido desde aquí puede resultar de utilidad para realizar arbitraje en mercados descentralizados. Por ello, queda como trabajo pendiente el integrar el Oráculo de Precios en un sistema de contratos inteligentes más grande, que desarrolle operaciones necesarias, como *swaps* o solicitudes de *flash loans*, que permitan la ejecución de estrategias de arbitraje desde las cuales percibir rentabilidad. Estas estrategias se beneficiarían de las diferencias de precios expuestas desde el oráculo.

La creación del Oráculo de Precios también ha servido para revelar un claro punto en contra de Blockchain y contratos inteligentes como alternativa para desplegar aplicaciones: su latencia. Se ha presentado la atomicidad de la ejecución de los contratos inteligentes como una ventaja ante la latencia con la que se pueden completar ciertos procesos; esto no significa que todas las transacciones en el Blockchain se completen con agilidad, sino que un contrato inteligente puede acoplar múltiples transacciones en un solo proceso, minimizando la latencia entre ellas y actuando prácticamente como una sola. De esta forma se eliminan o reducen muchos costes y riesgos que surgen en los intervalos que transcurren entre cada transacción. Esto es especialmente conveniente en un contexto de *High Frequency Trading* o Comercio de Alta Frecuencia.

No obstante, la resolución de una sola transacción o la ejecución de un contrato inteligente requiere de un notable tiempo de espera hasta que queda reflejado de manera definitiva en la red Blockchain. Al menos, este tiempo de espera resulta especialmente significativo con respecto a la latencia de ciertos procesos que tienen lugar en los servicios ofrecidos por las plataformas centralizadas en la Web 2.0. Por otra parte, es concebible que una arquitectura distribuida que integra miles de nodos requiera de tiempo suficiente para actualizar los registros en cada uno de los terminales de la red. Además, es de esperar que el progreso de esta tecnología ofrezca en un futuro protocolos de comunicación en la red más ágiles que los que existen actualmente. En otras palabras, el modelo cliente-servidor convencional ofrece tiempos de respuesta muy superiores a los de una red Blockchain.

En conclusión, los contratos inteligentes componen una herramienta que permite automatizar múltiples procesos y prescindir la implicación de agentes intermediarios que pueden comprometer la seguridad e integridad de la información involucrada, garantizan el cumplimiento determinista de las cláusulas establecidas y optimizan el proceso, reduciendo tiempos, costes y riesgos. Todos estos atributos confieren una gran oportunidad para reinventar las finanzas y generar un entorno mucho más accesible, transparente, simbiótico, equitativo y distribuido. El Oráculo de Precios de AMM ha servido para ilustrar algunas de estas competencias de las que extraer un crucial valor.

6 Análisis de impacto

En este último apartado, se analizará el impacto que se espera de este Trabajo Fin de Grado y de la tecnología innovadora que se trata desde aquí. En paralelo, se mencionarán algunos de los Objetivos de Desarrollo Sostenibles propuestos en la Agenda 2030 [80] que se consideran pertinentes a la exposición desarrollada.

En este proyecto se han presentado muchos de los potenciales usos de esta tecnología. En la sección -o se hablaba de la aportación del Blockchain en áreas tan dispersas como gobierno, producción industrial, *smart cities* o sanidad.

En primer lugar, Blockchain puede dar un soporte positivo en materia de ecologismo o alivio del impacto medioambiental. Aunque ya se conoce el consumo computacional que implican algunas redes y dispositivos, en términos comparativos, creemos que esta tecnología puede optimizar multitud de procesos que maximicen la eficiencia energética. Soluciones ya comentadas como la compraventa de la energía, la optimización de la producción industrial o el soporte a *smart cities* son ejemplo de ello. Más aún, se espera que el avance de la tecnología Blockchain derive en implementaciones de algoritmos, mecanismos y protocolos que, sin minorar su eficacia o seguridad, necesiten de menor gasto energético. Estas preocupaciones quedan constatadas en los ODS: (7) Energía asequible y no contaminante, (11) Ciudades y comunidades sostenibles o (13) Acción por el clima.

Por otra parte, Blockchain parece la primera promesa más viable o, por lo menos más extendida, que democratiza las formas de gobierno y el uso de los datos. Para empezar, sistemas como los DAOs disponen capacidades de decisión sobre la dirección de una entidad a los usuarios que la componen, de manera simétrica y equitativa e inclusiva. Además, los sistemas de registros sanitarios, las certificaciones de identidades o cualquier forma de activo intangible (como patentes o derechos de autor) basados en Blockchain ratifican de manera mucho más firme la propiedad sobre datos o información sujeta a la propiedad. De esta manera, el tratamiento y distribución de este tipo de datos puede evitar o minimizar sustancialmente su uso ilegítimo, lucrativo o fraudulento por terceras partes, al quedar reconocida irrevocablemente su origen, autoría o inherente propiedad.

En la misma línea, se puede extrapolar al ámbito empresarial atributos ya mencionados de la tecnología. Se ha presentado el ejemplo de la industria del turismo, pero de la misma manera, otros sistemas, servicios o funcionalidades *Peer to Peer* pueden prescindir de la centralización y la intermediación gracias al Blockchain. Así se potencia la reducción de tiempos y costes, asegurando un cifrado de la información extremo a extremo para que los datos involucrados en la transacción sean conocidos exclusivamente por cada parte de la relación.

Tal y como se ha orientado el trabajo hacia la implementación práctica, el ámbito que más interesa es el de las finanzas. Aunque las finanzas descentralizadas definitivamente presentan objeciones, dilemas y contrariedades, ya se han planteado como una forma de finanzas mucho más justa y accesible que las convencionales. Los

mercados descentralizados se asientan en unos valores de desintermediación, transparencia e interoperabilidad que han quedado más que patentes en su recorrido y en este mismo trabajo.

Las DeFi eliminan barreras de entrada y requisitos asociados al capital o identidad de los agentes, a la vez que generan una plataforma compartida e igualitaria para todos los miembros del mercado. Esto promueve la inclusión de nuevos participantes de diferentes orígenes y condiciones. Simultáneamente, se limita el poder e influencia individual de cada uno de los agentes, desacelerando el alcance y crecimiento de aquellas entidades que dispongan de más recursos.

Todas estas mejoras dan respuesta a los puntos (10) Reducción de las desigualdades o (16) Paz, justicia e instituciones sólidas de las ODS. Incluso, la colaboración que se induce en este entorno desde su nivel más fundamental y que se refleja en una arquitectura computacional o protocolos comunes y cooperativos, puede constituir un vehículo para el punto (17) Alianzas para lograr objetivos.

Termino hablando en primera persona para manifestar mi agradecimiento hacia el interés ofrecido hacia mi trabajo y para hacer visible el impacto que ha tenido este trabajo a nivel personal. Transmito mi más sincero anhelo de que el lector haya podido aprender y extraer información para su interés de este trabajo. Ya sea como entusiasta de la materia, con el afán de profundizar su conocimiento técnico o con expectativas de orientar su esfuerzo hacia este ámbito, espero haber podido aportar el valor que se deseaba extraer de la lectura de este proyecto.

Por una parte, me siento realmente orgulloso y realizado por el trabajo elaborado y la contribución que haya podido suponer este proyecto en materia académica, técnica y literaria. Además, siento que este proyecto ha supuesto una de las tareas más fructuosas (si no la que más) realizadas en todo mi recorrido universitarios en términos de formación, desarrollo y enriquecimiento; tanto personal como académico.

Por último, considero muy recalcable expresar la motivación y estímulo que he ido experimentando a través del desarrollo de este Trabajo Fin de Grado. A medida que el proceso de investigación y desarrollo práctico me ha permitido ahondar en esta materia, he ido descubriendo un creciente interés por mi parte que ha servido como aliciente para continuar el trabajo con ilusión, satisfacción y predisposición. Prueba de ello es que estoy orientando mis expectativas profesionales hacia el área de programación de contratos inteligentes en *DeFi*. Espero que mi entusiasmo se haya hecho perceptible durante la lectura.

Por lo que a mí respecta, he terminado de cerrar este trabajo. Solo me queda alentar al lector a investigar más sobre las innumerables y sustanciosas posibilidades que ofrecen Blockchain y los contratos inteligentes. Tampoco quiero despedirme sin recordar una última vez que, para mayor comprensión de la Finanzas Descentralizadas, *Automated Market Makers* y *Swaps* se pone a total disposición el trabajo de "Finanzas Descentralizadas en Blockchain: Automated Market Makers, Swaps y Arbitraje" abordado desde una perspectiva teórico-financiera.

7 Referencias

[1] A. Horowitz, «a16z.com,» octubre 2021. [En línea]. Available: https://a16z.com/wp-content/uploads/2021/10/How-to-Win-the-Future-Deck.pdf. [Último acceso: marzo 2022].

[2] D. Vanas, «entrepreneur.com,» abril 2022. [En línea]. Available: https://www.entrepreneur.com/article/422311. [Último acceso: abril 2022].

[3] C. Dixon, «future.a16z.com,» octubre 2021. [En línea]. Available: https://future.a16z.com/why-web3-matters/. [Último acceso: 2022 marzo].

[4] L. Lamport, «The Part-Time Parliament,» *ACM Transactions on Computer Systems,* 1998.

[5] S. Nakamoto, «Bitcoin: A Peer-to-Peer Electronic Cash System,» 2008.

[6] D. &. A. Tapscott, Blockchain Revolution, Portfolio, 2016.

[7] D. Yaga, P. Mell, N. Roby y K. Scarfone, «Blockchain Technology Overview,» 2018.

[8] M. Nofer, P. Gomber, O. Hinz y D. Schiereck, «Blockchain,» 2017.

[9] Z. Zheng, S. Xie, H. Dai, X. Chen y H. Wang, «An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends,» 2017.

[10] B. Kewell, R. Adams y G. Parry, «Blockchain for good?,» *Wiley,* 2017.

[11] J. Frankenfield, «investopedia.com,» agosto 2021. [En línea]. Available: https://www.investopedia.com/terms/d/distributed-ledger-technology-dlt.asp. [Último acceso: abril 2022]

[12] A. P. Mohan, M. A. R. y A. Gladston, «Merkle Tree and BlockchainBased Cloud Data Auditing,» *International Journal of Cloud Applications and Computing,* vol. 10, n° 3, 2020.

[13] A. d. Vries, «Bitcoin's Growing Energy Problem,» *Joule,* vol. 2, n° 801-809, 2018.

[14] Z. Zheng, S. Xie, H.-N. Dai, W. Chen, X. Chen, J. Wengc y M. Imran, «An overview on smart contracts: Challenges, advances and platforms,» *Elsevier,* 2020.

[15] V. Gatteschi, F. Lamberti, C. Demartini, C. Pranteda y V. Santamar a, «To Blockchain or Not to Blockchain: That Is the Question,» *IEEE Computer Society,* 2018.

[16] A.-D. Popescu, «Transitions and Concepts Within Decentralized Finance Space,» 2020.

[17] U. Bodkhe, S. Tanwar, K. Parekh, P. Khanpara, S. Tyagi, N. Kumar y M. Alazab, «Blockchain for Industry 4.0: A Comprehensive Review,» *IEEE,* vol. 8, 2020.

[18] M. Xu, «A systematic review of blockchain,» *Financial Innovation,* vol. 5, n° 27, 2019.

[19] M. I. Mehar, C. L. Shier, A. Giambattista, E. Gong, G. Fletcher, R. Sanayhie, H. M. Kim y M. Laskowski, «Understanding a Revolutionary and Flawed Grand Experiment in Blockchain: The DAO Attack,» Jorunal of Cases on Information Technology, vol. 21, n° 1, 2019

[20] C. Staff, «gemini.com,» 17 marzo 2022. [En línea]. Available: https://www.gemini.com/cryptopedia/the-dao-hack-makerdao. [Último acceso: 2022 abril 2022].

[21] F. Schär, «Decentralized Finance: On Blockchain- and Smart Contract-Based Financial Markets,» pp. 160-163, 2021.

[22] W. Zou, D. Lo, P. S. Kochhar, X.-B. D. Le, X. Xia, Y. Feng, Z. Chen y B. Xu, «Smart Contract Development: Challenges and Opportunities,» *IEEE Transactions on Software Engineering* , vol. 47, nº 10, 2021.

[23] N. Szabo, «fon.hum.uva.nl,» 1996. [En línea]. Available: https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html. [Último acceso: abril 2022].

[24] E. S. C., «Teoría de los Costos de Transacción: Una Breve Reseña,» *Cuadernos de Administración*, vol. 16, nº 26, 2003.

[25] M. Scherer, «Performance and Scalability of Blockchain Networks and Smart Contracts,» 2017.

[26] M. Alharby y A. v. Moorsel, «Blockchain-Based Smart Contracts: A Systematic Mapping Study,» 2017.

[27] P. Schueffel, «DeFi: Decentralized Finance - An Introduction and Overview,» Vols. %1 de %29,3, 2021.

[28] V. P. Nartallo, «bbva.com,» Noviembre 2021. [En línea]. Available: https://www.bbva.com/es/que-diferencias-hay-entre-un-token-y-una-criptomoneda/. [Último acceso: Marzo 2022].

[29] T. B. o. N. Y. M. Corporation, «https://www.bnymellon.com/,» Junio 2019. [En línea]. Available: https://www.bnymellon.com/us/en/insights/all-insights/tokenization-opening-illiquid-assets-to-investors.html. [Último acceso: Marzo 2022].

[30] «CoinGecko,» [En línea]. Available: https://www.coingecko.com/es/categories/automated-market-maker-amm. [Último acceso: mayo 2022].

[31] R. d. Best, «statista.com,» febrero 2022. [En línea]. Available: https://www.statista.com/statistics/863917/number-crypto-coins-tokens/#:~:text=How%20many%20cryptocurrencies%20are%20there,of%20digital%20coins%20in%202013. [Último acceso: 2022 marzo].

[32] «Uniswap en github.com,» Uniswap Labs, [En línea]. Available: https://github.com/Uniswap. [Último acceso: mayo 2022].

[33] Uniswap, «Uniswap Docs,» [En línea]. Available: https://docs.uniswap.org/. [Último acceso: mayo 2022].

[34] N. Zinsmeister, «github,» [En línea]. Available: https://github.com/Uniswap/v2-core/tree/master/contracts. [Último acceso: mayo 2022].

[35] N. Ilamanov, «medium.com,» febrero 2021. [En línea]. Available: https://betterprogramming.pub/uniswap-smart-contract-breakdown-ea20edf1a0ff. [Último acceso: mayo 2022].

[36] Moody Salem, «github,» [En línea]. Available: https://github.com/Uniswap/v2-periphery/tree/master/contracts. [Último acceso: mayo 2022].

[37] N. Ilamanov, «medium.com,» febrero 2021. [En línea]. Available: https://ilamanov.medium.com/uniswap-smart-contract-breakdown-part-2-b9ea2fca65d1. [Último acceso: mayo 2022].

[38] Uniswap Labs, «Uniswap Docs,» [En línea]. Available: https://docs.uniswap.org/protocol/reference/smart-contracts. [Último acceso: mayo 2022].

[39] «Contratos de Uniswap V3 Core en github.com,» Uniswap Labs, [En línea]. Available: https://github.com/Uniswap/v3-core.

[40] «Contratos de Uniswap V3 Periphery en github.som,» Uniswap Labs, [En línea]. Available: https://github.com/Uniswap/v3-periphery.

[41] «Curve Docs,» [En línea]. Available: https://curve.readthedocs.io/registry-overview.html. [Último acceso: mayo 2022].

[42] «Protocolo de CurveFi en github.com,» CurveFi, [En línea]. Available: https://github.com/curvefi/curve-pool-registry. [Último acceso: mayo 2022].

[43] «Protocolo 1inch en github.com,» 1inch, [En línea]. Available: https://github.com/1inch/1inchProtocol. [Último acceso: 2022 mayo].

[44] «Chainlink Docs,» [En línea]. Available: https://docs.chain.link/docs/price-feeds-api-reference/. [Último acceso: mayo 2022].

[45] «Protocolo de Chainlink en github.com,» Chainlink, [En línea]. Available: https://github.com/smartcontractkit/chainlink. [Último acceso: mayo 2022].

[46] «ethereum.org,» marzo 2022. [En línea]. Available: https://ethereum.org/en/developers/docs/networks/.

[47] «Chainlink Faucets,» [En línea]. Available: https://faucets.chain.link/rinkeby. [Último acceso: mayo 2022].

[48] Remix, «Remix IDE,» [En línea]. Available: https://remix.ethereum.org/. [Último acceso: mayo 2022].

[49] Remix, «Remix,» [En línea]. Available: https://remix-project.org/. [Último acceso: mayo 2022].

[50] «Metamask,» [En línea]. Available: https://metamask.io/. [Último acceso: mayo 2022].

[51] «chrome web store,» [En línea]. Available: https://chrome.google.com/webstore/detail/metamask/nkbihfbeogaeaoehlefnkodbefgpgknn?hl=es. [Último acceso: mayo 2022].

[52] bit2me Academy, «bit2me Academy,» [En línea]. Available: https://academy.bit2me.com/que-es-metamask-la-forma-mas-facil-de-usar-dapps/. [Último acceso: mayo 2022].

[53] bit2me Academy, bit2me Academy, [En línea]. Available: https://academy.bit2me.com/que-son-hot-wallets/#:~:text=Una%20Hot%20Wallet%2C%20es,y%20gesti%C3%B3n%20de%20las%20criptomonedas.. [Último acceso: mayo 2022].

[54] «node JS,» Node.js, [En línea]. Available: https://nodejs.org/en/about/. [Último acceso: mayo 2022].

[55] «Node JS dev,» Node JS, [En línea]. Available: https://nodejs.dev/learn. [Último acceso: mayo 2022].

[56] «About npm,» npm, [En línea]. Available: https://www.npmjs.com/about. [Último acceso: mayo 2022].

[57] node JS, «What is npm?,» node JS, [En línea]. Available: https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/. [Último acceso: mayo 2022].

[58] Truffle Suite, «Truffle Suite,» [En línea]. Available: https://trufflesuite.com/. [Último acceso: mayo 2022].

[59] Truffle Suite, «Truffle Suite,» [En línea]. Available: https://trufflesuite.com/docs/.

[60] F. Heintel, «Solidity Blog,» Solidity , febrero 2022. [En línea]. Available: https://blog.soliditylang.org/2022/02/07/solidity-developer-survey-2021-results/. [Último acceso: mayo 2022]

[61] H. Valdez, «TheBlockchainGuy,» octubre 2021. [En línea]. Available: https://theblockchainguy.dev/hardhat-vs-truffle-vs-remix. [Último acceso: mayo 2022].

[62] «LibHunt,» [En línea]. Available: https://www.libhunt.com/compare-hardhat-vs-truffle. [Último acceso: mayo 2022].

[63] «Boilerplate de Solidity provisto en github.com,» DeFi Wonderland, [En línea]. Available: https://github.com/defi-wonderland/solidity-boilerplate. [Último acceso: mayo 2022].

[64] «Paquete "eth-sdk" provisto en github.com,» dethcrypto, [En línea]. Available: https://github.com/dethcrypto/eth-sdk. [Último acceso: mayo 2022].

[65] «Chai Assertion library,» Chai Assertion library, [En línea]. Available: https://www.chaijs.com/. [Último acceso: mayo 2022].

[66] «Ethers Docs,» Ethers Docs, [En línea]. Available: https://docs.ethers.io/v5/. [Último acceso: mayo 2022].

[67] «Solidity Docs,» [En línea]. Available: https://docs.soliditylang.org/en/v0.4.24/abi-spec.html. [Último acceso: mayo 2022].

[68] @Ignjatovic, «Ethereum Docs,» 27 enero 2022. [En línea]. Available: https://ethereum.org/en/developers/docs/smart-contracts/deploying/. [Último acceso: mayo 2022].

[69] «Infura,» [En línea]. Available: https://infura.io/. [Último acceso: mayo 2022].

[70] Universidad de Alcalá, «Universidad de Alcalá,» [En línea]. Available: https://masterblockchain.net/infura-master-blockchain-online/. [Último acceso: mayo 2022].

[71] @dev-matthew, «Ethereum Docs,» 21 mayo 2022. [En línea]. Available: https://ethereum.org/en/developers/docs/nodes-and-clients/nodes-as-a-service/. [Último acceso: mayo 2022].

[72] Infura, «Infura,» [En línea]. Available: https://infura.io/. [Último acceso: mayo 2022].

[73] «Etherscan Dashboard,» Etherscan, [En línea]. Available: https://etherscan.io/. [Último acceso: mayo 2022].

[74] «Sobre Etherscan,» Etherscan, [En línea]. Available: https://etherscan.io/aboutus. [Último acceso: mayo 2022].

[75] «OpenZeppelin Docs,» [En línea]. Available: https://docs.openzeppelin.com/contracts/3.x/api/utils#EnumerableSet. [Último acceso: mayo 2022].

[76] «OpenZeppelin Docs,» [En línea]. Available: https://docs.openzeppelin.com/contracts/2.x/api/math. [Último acceso: mayo 2022].

[77] «UniswapV2Library.sol en github.com,» Uniswap Labs, [En línea]. Available: https://github.com/Uniswap/v2-periphery/blob/master/contracts/libraries/UniswapV2Library.sol. [Último acceso: mayo 2022].

[78] «@lbertenasco en github.com,» Github, [En línea]. Available: https://github.com/lbertenasco.

[79] «Uniswap V3 Bug Bounty en github.com,» Uniswap Labs, [En línea]. Available: https://github.com/lbertenasco/uniswap-v3-periphery/blob/main/bug-bounty.md.

[80] Naciones Unidas, «Objetivos de Desarrollo Sostenible,» [En línea]. Available: https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/. [Último acceso: mayo 2022].

[81] CoinMarketCap, «CoinMarketCap,» [En línea]. Available: https://coinmarketcap.com/charts/. [Último acceso: mayo 2022].

8 Anexos

8.1 Anexo 1: Tests Unitarios

```
import chai, { expect } from 'chai';
import { ethers } from 'hardhat';
import { Oracle, Oracle__factory, WrapperCurveFi, Wrapper1Inch } from '@typechained';
import { evm } from '@utils';
import { BigNumber } from 'ethers';
import { FakeContract, MockContract, smock } from '@defi-wonderland/smock';
import { SignerWithAddress } from '@nomiclabs/hardhat-ethers/dist/src/signer-with-address';

chai.use(smock.matchers);

describe('Oracle', function () {
  let oracleFactory;
  let oracle: MockContract<Oracle>;
  let wrapperCurve: FakeContract<WrapperCurveFi>;
  let wrapper1Inch: FakeContract<Wrapper1Inch>;

  let snapshotId: string;

  let governor: SignerWithAddress;

  const TOKEN_IN = '0xdAC17F958D2ee523a2206206994597C13D831ec7';
  const TOKEN_OUT = '0x6B175474E89094C44Da98b954EedeAC495271d0F';
  const ZERO = ethers.constants.AddressZero;
  const FIVE: BigNumber = ethers.BigNumber.from('5');

  before(async () => {
    [governor] = await ethers.getSigners();
    wrapper1Inch = await smock.fake('Wrapper1Inch');
    wrapperCurve = await smock.fake('WrapperCurveFi');
    oracleFactory = await smock.mock<Oracle__factory>('Oracle');
    oracle = await oracleFactory.deploy(wrapper1Inch.address);
    snapshotId = await evm.snapshot.take();
  });

  beforeEach(async () => {
    await evm.snapshot.revert(snapshotId);
  });

  describe('constructor', () => {
    it('should parameters are valid', async function () {
      expect(await oracle.defaultWrapper()).to.equal(wrapper1Inch.address);
    });
  });

  describe('getAmountOut', () => {
    it('should revert if amount is 0', async function () {
      await expect(oracle.getAmountOut(TOKEN_IN, 0,
TOKEN_OUT)).to.be.revertedWith('NotEnoughAmount()');
    });
    it('should revert if token_in/out is 0 address', async function () {
      await expect(oracle.getAmountOut(ZERO, FIVE, TOKEN_OUT)).to.be.revertedWith('InvalidToken()');
      await expect(oracle.getAmountOut(TOKEN_IN, FIVE, ZERO)).to.be.revertedWith('InvalidToken()');
    });

    it('should return getAmountOut with a pair', async function () {
      await oracle.setVariable('wrappers', {
        [TOKEN_IN]: {
          [TOKEN_OUT]: wrapperCurve.address,
        },
      });
```

```
      await oracle.getAmountOut(TOKEN_IN, FIVE, TOKEN_OUT);
      expect(wrapper1Inch.quote).to.have.been.calledWith(TOKEN_IN, FIVE, TOKEN_OUT);
    });

    it('should return getAmountOut with tokenIn', async function () {
      await oracle.setVariable('wrappers', {
        [TOKEN_IN]: {
          [ZERO]: wrapper1Inch.address,
        },
      });

      await oracle.getAmountOut(TOKEN_IN, FIVE, TOKEN_OUT);
      expect(wrapper1Inch.quote).to.have.been.calledWith(TOKEN_IN, FIVE, TOKEN_OUT);
    });

    it('should return getAmountOut with default', async function () {
      await oracle.getAmountOut(TOKEN_IN, FIVE, TOKEN_OUT);
      expect(wrapper1Inch.quote).to.have.been.calledWith(TOKEN_IN, FIVE, TOKEN_OUT);
    });
  });

  describe('addWrapper', () => {
    it('should revert if wrapper exists ', async function () {
      await oracle.setVariable('wrappers', {
        [TOKEN_OUT]: {
          [TOKEN_IN]: wrapperCurve.address,
        },
      });
      await expect(oracle.addWrapper(wrapperCurve.address, TOKEN_IN,
TOKEN_OUT)).to.be.revertedWith('WrapperExists()');
    });

    it('should add wrapper', async function () {
      await oracle.connect(governor).addWrapper(wrapper1Inch.address, TOKEN_IN, TOKEN_OUT);
      expect(await oracle.wrappers(TOKEN_OUT, TOKEN_IN)).to.be.equals(wrapper1Inch.address);
    });
  });

  describe('swapDefaultWrapper', () => {
    it('should revert is wrapper address is 0x0', async function () {
      await
expect(oracle.connect(governor).setDefaultWrapper(ZERO)).to.be.revertedWith('WrapperZero()');
    });
    it('should change default wrapper', async function () {
      await oracle.connect(governor).setDefaultWrapper(wrapperCurve.address);
      expect(await oracle.defaultWrapper()).equals(wrapperCurve.address);
    });
  });
});
```

Figura 30: unit/oracle.spec.ts

```
import chai, { expect } from 'chai';
import { ethers } from 'hardhat';
import { evm } from '@utils';
import { FakeContract, MockContract, smock } from '@defi-wonderland/smock';
import { BigNumber } from 'ethers';
import {
  Wrapper1Inch,
  Wrapper1Inch__factory,
  WrapperChainlink,
  WrapperChainlink__factory,
  WrapperCurveFi,
  WrapperCurveFi__factory,
  WrapperSushiswap,
  WrapperSushiswap__factory,
  WrapperUniswapV2,
  WrapperUniswapV2__factory,
  WrapperUniswapV3,
  WrapperUniswapV3__factory,
  I1Split,
  IQuoter,
  IRegistry,
  FeedRegistryInterface,
  ICryptoPool,
  IUniswapV2Pair,
  IUniswapV2Factory,
  IUniV2Pair,
} from '@typechained';

chai.use(smock.matchers);

describe('Wrapper1Inch', function () {
  let oneInch: MockContract<Wrapper1Inch>;
  let oneInchFactory;
  let mockSplit: FakeContract<I1Split>;

  let snapshotId: string;

  const AMOUNT_IN: BigNumber = ethers.BigNumber.from('5');
  const WETH = '0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2';
  const DAI = '0x6B175474E89094C44Da98b954EedeAC495271d0F';

  before(async () => {
    mockSplit = await smock.fake('I1Split');
    oneInchFactory = await smock.mock<Wrapper1Inch__factory>('Wrapper1Inch');
    oneInch = await oneInchFactory.deploy(mockSplit.address);

    snapshotId = await evm.snapshot.take();
  });

  beforeEach(async () => {
    await evm.snapshot.revert(snapshotId);
  });

  describe('oneInch quote', function () {
    it('should return correct quote using registry', async function () {
      mockSplit.getExpectedReturn.whenCalledWith(WETH, DAI, AMOUNT_IN, 1, 0).returns([10, [1]]);
      expect(await oneInch.quote(WETH, AMOUNT_IN, DAI)).to.be.equal(10);
    });
  });
});

describe('WrapperChainlink', function () {
  let chainlink: MockContract<WrapperChainlink>;
  let chainlinkFactory;
  let mockRegistry: FakeContract<FeedRegistryInterface>;

  let snapshotId: string;

  const AMOUNT_IN: BigNumber = ethers.BigNumber.from('5');
  const USDT = '0xdAC17F958D2ee523a2206206994597C13D831ec7';
  const DAI = '0x6B175474E89094C44Da98b954EedeAC495271d0F';
```

```
  before(async () => {
    mockRegistry = await smock.fake('FeedRegistryInterface');
    chainlinkFactory = await smock.mock<WrapperChainlink__factory>('WrapperChainlink');
    chainlink = await chainlinkFactory.deploy(mockRegistry.address);

    snapshotId = await evm.snapshot.take();
  });

  beforeEach(async () => {
    await evm.snapshot.revert(snapshotId);
  });

  describe('chainlink quote', function () {
    it('should return correct quote using registry', async function () {
      mockRegistry.latestRoundData.whenCalledWith(DAI, USDT).returns([0, ethers.BigNumber.from('5'),
0, 0, 0]);
      expect(await chainlink.quote(DAI, AMOUNT_IN, USDT)).to.be.equal(5 * 5);
    });
  });
});

describe('WrapperCurve', function () {
  let curve: MockContract<WrapperCurveFi>;
  let curveFactory;
  let mockRegistry: FakeContract<IRegistry>;
  let mockCrypto: FakeContract<ICryptoPool>;

  let snapshotId: string;

  const AMOUNT_IN: BigNumber = ethers.BigNumber.from('5');
  const USDT = '0xdAC17F958D2ee523a2206206994597C13D831ec7';
  const WETH = '0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2';

  before(async () => {
    mockRegistry = await smock.fake('IRegistry');
    mockCrypto = await smock.fake('ICryptoPool');
    curveFactory = await smock.mock<WrapperCurveFi__factory>('WrapperCurveFi');
    curve = await curveFactory.deploy(mockRegistry.address);

    snapshotId = await evm.snapshot.take();
  });

  beforeEach(async () => {
    await evm.snapshot.revert(snapshotId);
  });

  describe('curveFi quote', function () {
    it('should return correct quote using registry', async function () {
      mockRegistry.find_pool_for_coins.whenCalledWith(WETH, USDT, 0).returns(mockCrypto.address);
      mockRegistry.get_coin_indices.whenCalledWith(mockCrypto.address, WETH, USDT).returns([1, 1, 1]);
      mockCrypto.get_dy.whenCalledWith(1, 1, AMOUNT_IN).returns(5);
      expect(await curve.quote(WETH, AMOUNT_IN, USDT)).to.be.equal(5);
    });
  });
});

describe('WrapperSushiswap', function () {
  let sushi: MockContract<WrapperSushiswap>;
  let sushiFactory;
  let pairV2: FakeContract<IUniV2Pair>;
  let factoryV2: FakeContract<IUniswapV2Factory>;

  let snapshotId: string;

  const AMOUNT_IN: BigNumber = ethers.BigNumber.from('5');
  const WETH = '0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2';
  const USDT = '0xdAC17F958D2ee523a2206206994597C13D831ec7';
```

```
    before(async () => {
    pairV2 = await smock.fake('IUniV2Pair', { address: '0x06da0fd433C1A5d7a4faa01111c044910A184553' });
    factoryV2 = await smock.fake('IUniswapV2Factory', { address:
'0xC0AEe478e3658e2610c5F7A4A2E1777cE9e4f2Ac' });
    sushiFactory = await smock.mock<WrapperSushiswap__factory>('WrapperSushiswap');
    sushi = await sushiFactory.deploy();

    snapshotId = await evm.snapshot.take();
  });

  beforeEach(async () => {
    await evm.snapshot.revert(snapshotId);
  });

  describe('sushiswap quote', function () {
    it('should return correct quote', async function () {
      factoryV2.getPair.returns('0x06da0fd433C1A5d7a4faa01111c044910A184553');
      pairV2.getReserves.returns([5, 10, 0]);
      const NUMERATOR = AMOUNT_IN.mul(997).mul(10);
      const DENOMINATOR = AMOUNT_IN.mul(997).add(5000);
      expect(await sushi.quote(USDT, AMOUNT_IN, WETH)).to.be.equal(NUMERATOR.div(DENOMINATOR));
    });
  });
});

describe('WrapperUniswapV2', function () {
  let uniV2: MockContract<WrapperUniswapV2>;
  let uniV2Factory;
  let pairV2: FakeContract<IUniV2Pair>;

  let snapshotId: string;

  const AMOUNT_IN: BigNumber = ethers.BigNumber.from('5');
  const WETH = '0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2';
  const USDT = '0xdAC17F958D2ee523a2206206994597C13D831ec7';

  before(async () => {
    pairV2 = await smock.fake('IUniV2Pair', { address: '0x0d4a11d5eeaac28ec3f61d100daf4d40471f1852' });
    uniV2Factory = await smock.mock<WrapperUniswapV2__factory>('WrapperUniswapV2');
    uniV2 = await uniV2Factory.deploy();

    snapshotId = await evm.snapshot.take();
  });

  beforeEach(async () => {
    await evm.snapshot.revert(snapshotId);
  });

  describe('uniswapV2 quote', function () {
    it('should return correct quote', async function () {
      pairV2.getReserves.returns([5, 10, 0]);
      const NUMERATOR = AMOUNT_IN.mul(997).mul(10);
      const DENOMINATOR = AMOUNT_IN.mul(997).add(5000);
      expect(await uniV2.quote(USDT, AMOUNT_IN, WETH)).to.be.equal(NUMERATOR.div(DENOMINATOR));
    });
  });
});

describe('WrapperUniswapV3', function () {
  let uniV3: MockContract<WrapperUniswapV3>;
  let uniV3Factory;
  let mockQuoter: FakeContract<IQuoter>;

  let snapshotId: string;

  const AMOUNT_IN: BigNumber = ethers.BigNumber.from('5');
  const USDT = '0xdAC17F958D2ee523a2206206994597C13D831ec7';
  const WETH = '0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2';
```

```
  before(async () => {
    mockQuoter = await smock.fake('IQuoter');
    uniV3Factory = await smock.mock<WrapperUniswapV3__factory>('WrapperUniswapV3');
    uniV3 = await uniV3Factory.deploy(mockQuoter.address);

    snapshotId = await evm.snapshot.take();
  });

  beforeEach(async () => {
    await evm.snapshot.revert(snapshotId);
  });

  describe('uniswapV3 quote', function () {
    it('should return correct quote', async function () {
      mockQuoter.quoteExactInput.returns(5);
      expect(await uniV3.quote(USDT, AMOUNT_IN, WETH)).to.be.equal(5);
    });
  });
});
```

Figura 31: unit/wrappers.spec.ts

8.2 Anexo 2: Procedimiento de Instalación

En este apartado se muestra el procedimiento necesario para instalar la implementación en otro dispositivo y hacerla plenamente funcional. Partimos desde la premisa de que se cuenta exclusivamente con el código fuente de los contratos inteligentes y tests mostrados en este trabajo.

1. En primer lugar, se debe garantizar que el equipo cuenta con el paquete Node.js. En caso contrario, se deberá instalar: https://nodejs.org/es/download/
2. Elegimos un directorio raíz en el cual se quiere generar el repositorio del proyecto. Desde aquí se lanza el comando en la terminal que clonará el *boilerplate* en Solidity en el cual se ha apoyado el trabajo [4]. Este paquete ya incluye la instalación de *hardhat*.

   ```
   git clone https://github.com/defi-wonderland/solidity-boilerplate.git
   ```

3. Se instala *yarn* dentro del repositorio del *boilerplate*.

   ```
   yarn install
   ```

4. Se instalan las dependencias externas
   a) Librerías de Uniswap V3 dispuestas por @lbertenasco en github.com [78]:

   ```
   npm i https://github.com/lbertenasco/uniswap-v3-core.git
   ```

   ```
   npm i https://github.com/lbertenasco/uniswap-v3-periphery.git
   ```

   b) Librerías de Sushiswap:

   ```
   npm i @chainlink/contracts
   ```

   c) Librerías de Sushiswap core:

   ```
   npm i @sushiswap/core
   ```

5. Se genera el fichero ".env" como clon del que ya viene propuesto en ".env.example". Desde aquí, se configura el punto de acceso a la *mainnet* de *Ethereum* proporcionado por el *Provider* de preferencia. Tal y como se explicó en la sección III del punto 4.2, para el desarrollo del Oráculo de Precios, se ha utilizado la API de Infura. Es posible configurar un punto de acceso gratuito desde https://infura.io.
   La URI del punto de acceso creado deberá plasmarse en el fichero para que quede de la siguiente manera:

```
NODE_URI_MAINNET=https://mainnet.infura.io/v3/**********************
```

[4] Para utilizar el comando *git* es necesario instalarlo previamente: https://www.atlassian.com/git/tutorials/install-git

6. Se configura el complemento *eth-sdk*. Dentro del módulo "eth-sdk", creado dentro de nuestro directorio raíz, se encontrará un archivo "config.ts". En este archivo habrá una configuración por defecto que soportará únicamente la dirección del token Dai, se añadirán más direcciones asociadas a contratos de la *mainnet*. El fichero debe quedar como:

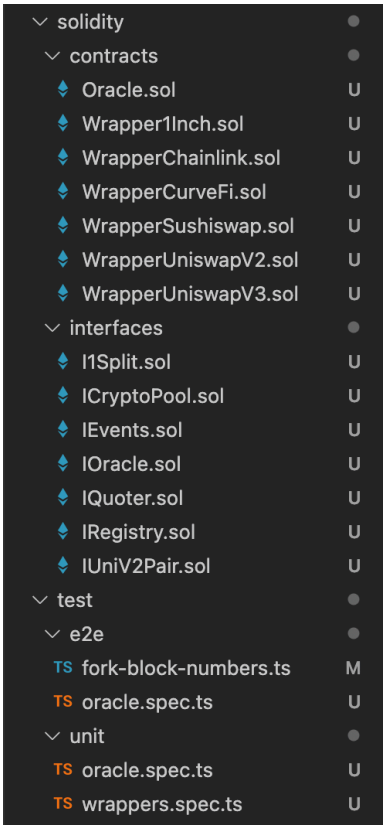
```
import { defineConfig } from '@dethcrypto/eth-sdk';

export default defineConfig({
  contracts: {
    mainnet: {
      dai: '0x6b175474e89094c44da98b954eedeac495271d0f',
      usdt: '0xdAC17F958D2ee523a2206206994597C13D831ec7',
      uni: '0x1f9840a85d5aF5bf1D1762F925BDADdC4201F984',
      curveRegistry: '0x90E00ACe148ca3b23Ac1bC8C240C2a7Dd9c2d7f5',
      oneSplit: '0xC586BeF4a0992C495Cf22e1aeEE4E446CECDee0E',
      quoter: '0xb27308f9F90D607463bb33eA1BeBb41C27CE5AB6',
      weth: '0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2',
      feedRegistry: '0x47Fb2585D2C56Fe188D0E6ec628a38b74fCeeeDf'
    },
  },
});
```

7. Una vez se encuentra configurada *eth-sdk*, se exportan los tipos con el comando:

   `yarn eth-sdk`

8. Copiamos los contratos inteligentes y los tests facilitados desde este trabajo en el módulo "solidity", en los directorios habilitados para ellos: contracts y test respectivamente. La estructura de ficheros debería quedar de la siguiente forma:



9. Con todo este recorrido completado, ya es posible compilar el proyecto con:

   `yarn compile`

   o bien ejecutar directamente los tests.

   `yarn test:unit`

   `yarn test:e2e`