GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Estudio sobre la cancelación de deuda circular mediante el uso de Smart Contracts en Ethereum.

Autor: Iñigo Sagredo Ruiz

Director: Atilano Fernández-Pacheco Sánchez-Migallón

Director: José Luis Gahete Diaz

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Estudio sobre la cancelación de deuda circular mediante el uso de Smart Contracts en Ethereum.

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2018/19 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Iñigo Sagredo Ruiz Fecha: 14/ 06/ 2019

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Nombre del Director Fecha: ……/ ……/ ……

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D. Iñigo Sagredo Ruiz

DECLARA ser el titular de los derechos de propiedad intelectual de la obra: Estudio sobre la cancelación de deuda circular mediante el uso de Smart Contracts en Ethereum, que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente).*

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.
d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción

de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 14 de junio de 2019

ACEPTA

Fdo Iñigo Sagredo Ruiz

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Estudio sobre la cancelación de deuda circular mediante el uso de Smart Contracts en Ethereum.

Autor: Iñigo Sagredo Ruiz

Director: Atilano Fernández-Pacheco Sánchez-Migallón

Director: José Luis Gahete Diaz

Madrid

Agradecimientos

En primer lugar, agradecer este trabajo a mis padres. Su dirección a lo largo no solo de la carrera sino también de toda mi educación ha hecho posible la realización de un TFG que simboliza el final de un grado de mucha exigencia.

A mis directores, Atilano Fernández-Pacheco y José Luis Gahete, que, a pesar de tener muchas responsabilidades, siempre tienen la puerta abierta para resolver cualquier duda y facilitar material, ejemplos e ideas que han permitido mostrar, sin duda, una mejor versión de este trabajo. Gracias a ambos.

También me gustaría reconocer a la profesora Bina Ramamurthy de la universidad Estatal de Nueva York por los cursos impartidos sobre la materia y a Gregory McCubbin, fundador de Dapp University, un negocio online que trata a fondo el desarrollo de aplicaciones distribuidas y en el que me he inspirado especialmente para la parte práctica.

Una vez más, gracias a todos.

ESTUDIO SOBRE LA CANCELACIÓN DE DEUDA CIRCULAR MEDIANTE EL USO DE SMART CONTRACTS EN ETHEREUM.

Autor: Iñigo Sagredo Ruiz
Director: Atilano Fernández-Pacheco Sánchez-Migallón
Director: José Luis Gahete Diaz
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

La siguiente tesis gira en torno a la tecnología Blockchain, la red Ethereum, y la posibilidad de utilizarla para resolver el problema de la deuda circular entre las empresas, así como explorar otros beneficios que puede reportar. Además de entrar en detalles precisos sobre la tecnología, se desarrollará un prototipo para probar cómo podría funcionar la idea en un entorno de prueba.

Palabras clave: Blockchain, Ethereum, deuda circular, prototipo

1. Introducción

La tecnología de la cadena de bloques ha ido evolucionando desde su inicio en 2008, cuando Satoshi Nakamoto la introdujo al mundo. Desde entonces, la filosofía detrás de Blockchain no ha cambiado, pero las implementaciones y posibilidades que permitía se han expandido enormemente.

Un claro ejemplo de ello ha sido el desarrollo de los contratos inteligentes en Ethereum. Este aspecto de Blockchain constituye la piedra angular de este proyecto, ya que ha sido tratado de forma muy precisa desde un punto de vista teórico.

La idea principal que rodea el uso de los contratos inteligentes es la de poder cancelar la deuda circular y mantener una copia verificable de todas las transacciones que tienen lugar. La deuda circular es el tipo de deuda en la que todos los participantes son al mismo tiempo acreedores y deudores y, por lo tanto, puede cancelarse parte de la deuda. El caso más frecuente de deuda circular parece ocurrir en el sector energético, especialmente en el sector energético de Pakistán[1]. No obstante, en este trabajo se prestará especial atención a la deuda circular contraída por las empresas, predominantemente las Pymes (Pequeñas y Medianas Empresas).

Hay muchas razones para creer que la tecnología Blockchain es ideal para resolver este problema, y se tratara de hacer patente a lo largo del trabajo.

2. Objetivo del trabajo

En primer lugar, el objetivo principal de este proyecto es aprender, de manera profunda, todo lo que hay que saber sobre la tecnología que hay detrás de la idea; Blockchain. Esto se hará a través de varios cursos e investigaciones independientes con el fin de obtener una visión lo más profunda posible. Este profundo estudio ha sido documentado a lo largo de la tesis de una manera ordenada.

En segundo lugar, se desarrollará un prototipo de la idea. Los prototipos tienen la funcionalidad básica de lo que se necesita. Vale la pena mencionar el hecho de que hubo algunos conocimientos considerables que tuvieron que ser adquiridos para desarrollar un "login" y un registro en node.js, el lenguaje de servidor utilizado en este prototipo.

Finalmente, visto el trabajo en retrospectiva y habiendo aprendido todo lo que se necesitaba, se han propuesto algunas sugerencias e ideas personales para la mejora potencial o el desarrollo futuro de la red Blockchain y Ethereum.

3. Trabajo desarrollado

Para llevar a cabo esta tarea, los 3 primeros cursos de la siguiente especialización se realizaron en Coursera[2] . Las credenciales de la universidad del estado de Nueva York que prueban que el curso fue tomado serán proporcionadas con el trabajo final.

Estos cursos sirvieron de brújula para orientarse en el camino de dominar el conocimiento de la tecnología. Cualquier tema que no fuera lo suficientemente detallado se ha ido ampliando con las diferentes fuentes de información como foros, vídeos y cualquier artículo que haya resultado útil. En consecuencia, se hará referencia a las que hayan resultado pertinentes.

Una vez que los cursos fueron tomados y el aprendizaje fue terminado, hubo un proceso largo y reflexivo para decidir si el entorno de Blockchain era el más adecuado para el desarrollo de la idea. Hubo 2 factores principales que fueron considerados. Por un lado, la cadena de bloques proporcionaba un medio verificable e inmutable para contrastar cualquier información proporcionada por el App y, por otro, permitía verificar programáticamente todas las condiciones necesarias. Todo esto es realmente importante ya que se pretende dedicar en el futuro un apartado para puntuar a las empresas en función de su responsabilidad a la hora de pagar sus deudas. Esperamos que todo esto quede claro a lo largo del trabajo.

Finalmente, una vez adquiridos los conocimientos y comprendidos estos en profundidad, el documento fue redactado de manera ordenada, con la esperanza de condensar lo mejor posible toda la información clave relacionada con esta tecnología. Una vez hecho, el prototipo fue desarrollado y explicado brevemente.

4. Breve explicación del prototipo

La funcionalidad, que una potencial versión completa de la aplicación puede tener, será discutida en el cuarto capítulo de esta tesis. A continuación, se ofrece una breve definición del prototipo y de su funcionamiento interno. La estructura del fichero es la siguiente.

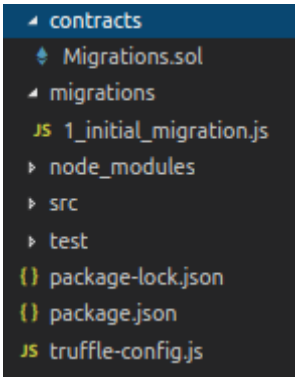

Ilustración 1. Estructura de los ficheros de una Dapp.

El prototipo tendrá la capacidad de registrar usuarios e iniciar sesión con esos mismos usuarios. Aunque esto puede considerarse un concepto trivial, se hizo con la esperanza de adquirir un mejor entendimiento sobre la forma en la que node.js, el lenguaje del lado del servidor utilizado, funciona. Esto cumplido con el propósito.

Luego permitirá a los usuarios crear deudas, que se mostrarán en el *Smart Contract* y, finalmente, podrán visualizarlas a través de la interfaz de la aplicación. Tenga en cuenta que en caso de que se pueda cancelar cualquier deuda, se hará por defecto, sin preguntar a ningún usuario. Esta funcionalidad se debate al final de este documento.

5. Conclusiones

En resumen, el proyecto se desarrolla de forma clara, comenzando con la introducción y explicación detallada de la tecnología en cuestión, y termina discutiendo la aplicación, su potencial y las posibilidades de Blockchain en el futuro.

Esta tesis resultó ser un tema desafiante, especialmente cuando se trataba de entender el funcionamiento interno de algunos conceptos de Blockchain. Para ello, se ha aprendido mucho y se espera que este documento ayude a cualquier otra persona que desee obtener una visión completa y amena de la tecnología.

6. Referencias

[1] Jorgic, D. (2019). Pakistan says curbing power sector debt, seeks energy investors. Obtenido de https://www.reuters.com/article/pakistan-power-minister/pakistan-says-curbing-power-sector-debt-seeks-energy-investors-idUSL5N1ZU0IK

[2] Blockchain | Coursera. (2019). Obtenido de https://www.coursera.org/specializations/blockchain

STUDY ON THE CANCELATION OF CIRCULAR DEBT BY MEANS OF A SMART CONTRACT IN THE ETHEREUM NETWORK.

Author: Iñigo Sagredo Ruiz
Supervisor: Atilano Fernández-Pacheco Sánchez-Migallón
Supervisor: José Luis Gahete Diaz
Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

The following thesis revolves around the blockchain technology, the Ethereum network, and the possibility of using it so as to solve the circular debt problem among companies, as well as exploring other benefits it may yield. Besides going into precise detail on the technology, a prototype will be developed to test out how the idea could work on a test environment.

Keywords: Blockchain, Ethereum, circular debt, prototype.

1. Introduction

Blockchain technology has been evolving ever since its beginning in 2008, when Satoshi Nakamoto introduced it to the world. Since then, the philosophy behind Blockchain hasn't changed, but the implementations and possibilities it allowed for have expanded greatly.

A clear example of this has been the development of Smart Contracts in Ethereum. This aspect of Blockchain constitutes the cornerstone of this project as it has been dealt with in a very precise fashion from a theoretical standpoint.

The main idea surrounding the use of smart contracts is that of both being able to cancel out circular debt and keeping a verifiable copy of all the transactions that take place. Circular debt is the sort of debt where all the participants are at the same time creditors and debtors, and hence some amount of the debt may be cancelled. The most frequent case of circular debt seems to take place in the energy sector, especially in Pakistan's energy sector [1]. Nevertheless, this work will emphasise in the circular debt incurred by business, predominantly SMEs (Small to Medium Enterprises).

There a plenty of reasons to believe that the blockchain technology is ideal to solve this problem, and it will be made as clear as possible throughout the thesis.

2. Thesis objectives

First and foremost, the main goal of this project is to learn, in a deep manner, everything there is to know about the technology behind the idea; Blockchain. This will be done through several courses and independent research in order to garner as profound an insight as possible. This deep study has been documented throughout the thesis in an orderly manner.

Secondly, a prototype of the idea will be developed. Prototypes have the basic functionality of what is needed. It is worth mentioning the fact that there was some

considerable knowledge that had to be acquired so as to develop a login and a register in node.js, the server-side language used in this prototype.

Finally, with the benefit of hindsight and having learned all of what was needed, some suggestions and personal ideas have been proposed for the potential improvement or future development of the Blockchain and Ethereum network.

3. Work undergone

In order to perform the task at hand, the first 3 courses from the following specialization were taken on Coursera [2]. The credentials from the New York state university that prove the fact that the course was taken will be provided with the final work.

Those courses served as a compass to orient oneself on the path of mastering the knowledge of the technology. Any topic that wasn't detailed enough has been extended by going the different sources of information such as forums, videos and any article that proved useful. The ones that proved relevant will be referenced accordingly.

Once the courses where taken and the learning was mostly finished, there was a long and thoughtful process to decide whether the Blockchain environment was the best suited for the development of the idea. There where 2 main factors that where most considered. On one hand, the blockchain provided a verifiable and immutable means by which to contrast any information given by the App, and on the other, it allowed for programmatically verify every condition needed. All of this is really important given the fact that it is intended to dedicate in the future a section to score the companies in the basis of their responsibility when paying their debts. All of this will hopefully be made clear throughout the thesis.

Finally, once the knowledge was acquired and profoundly understood the document was written in an orderly manner, in hopes of condensing as well as possible all the key information related with this technology. When done, the prototype was developed and explained briefly.

4. Prototype definition

The functionality that a potential full version of the app may have will be discussed in the fourth chapter of this thesis. Here, a brief definition of the prototype and its inner workings will be provided. The file structure is the following.

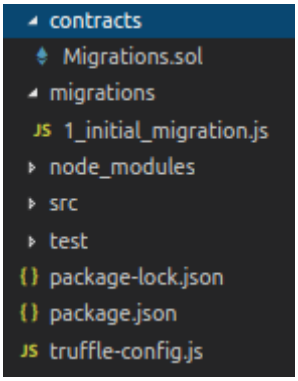

Ilustración 2. Dapp file structure

The prototype will have the ability to register users and login with those same users. Though this may be considered a trivial concept, it was done in hopes of getting a better grip into the way node.js, the server-side language used, works. It served its purpose.

Then it will allow users to create debt, it being shown in the Smart Contract and finally being able to watch it through the application's interface. Note that in case any debt can be cancelled, it will be done by default, without asking any user. This functionality is debated at the end of this paper.

5. Conclusion

To sum up, the project is developed in a clear way, starting with the introduction and thorough explanation of the technology at hand, and ends up discussing the application, its potential and the possibilities of Blockchain in the future.

This thesis proved to be a challenging subject, especially when it came to understand the low lever inner workings of some concepts in Blockchain. For this, a great deal has been learnt and this paper will hopefully help anyone else hoping to get a complete and enjoyable insight on the technology.

6. References

[1] Jorgic, D. (2019). Pakistan says curbing power sector debt, seeks energy investors. Retrieved from https://www.reuters.com/article/pakistan-power-minister/pakistan-says-curbing-power-sector-debt-seeks-energy-investors-idUSL5N1ZU0IK

[2] Blockchain | Coursera. (2019). Retrieved from https://www.coursera.org/specializations/blockchain

Índice de la memoria

Índice de figuras

Índice de Ilustraciones

Figura 3. Ejemplo de uso de UTXO

Capítulo 1. INTRODUCCIÓN

En el presente Trabajo de Fin de Carrera, se abordará la posibilidad de emplear las capacidades que ofrece la tecnología Blockchain para la cancelación de deuda circular.

Para ello, primero se introducirá el trabajo, explicando la motivación de este y un breve contexto histórico de la tecnología. A continuación, se hará un estudio exhaustivo del estado del arte de la tecnología, ya que, entre otras cosas, trata de manera cercana y practica números temas tratados a lo largo del grado. Esto posibilitara una mejor comprensión de las tecnologías empleadas para la resolución de este trabajo.

Posteriormente, se tratará a fondo la aplicación práctica en cuestión, mostrando la funcionalidad y la forma en la que se han integrado los conocimientos adquiridos en un prototipo funcional.

Para terminar, se discutirá la nueva variable programática que supone el gas a la hora de escribir código en solidity, se indagará en las perspectivas de futuro de Blockchain y se cerrará con una conclusión y la pertinente bibliografía. Nótese que junto con el trabajo se adjuntaran dos anexos: Un manual de usuario que facilitara al cliente el uso de la aplicación y un manual de instalación que dará una perspectiva sobre cómo se desarrolló la aplicación (instalación de programas, estructura del código…)

1.1 MOTIVACIÓN DEL PROYECTO

A día de hoy, la deuda es uno de los problemas más notorios de cualquier economía. Se estima que, en 2018, la deuda mundial ascendió a 247 billones de dólares [3]. Una considerable cantidad de esa deuda es de carácter circular, es decir, que todos los miembros de la cadena de deuda son a la vez acreedores y deudores, y se forman bucles de deuda. Por consiguiente, este proyecto trata de ofrecer una solución a través de la tecnología de Blockchain, haciendo hincapié en su carácter distribuido para paliar el problema.

A medida que la economía mundial se va globalizando aún más, es de esperar que esta deuda circular aumente. Se puede encontrar un claro ejemplo de esto en el sector energético de Pakistán, cuya deuda circular está produciendo graves problemas en el país, alcanzando cifras de 10,600 millones de dólares de deuda circular en el sector en abril de 2019, y aumentando a una vertiginosa velocidad de 6.4 millones de dólares diarios [4].

Existen soluciones al problema de la deuda circular en el ámbito privado de las personas, como la aplicación de viajes *Settle Up* que permite cancelar deudas circulares con tus acompañantes. Sin embargo, no es una aplicación óptima para utilizar a gran escala ya que se requiere entre otras cosas una gran confianza en una entidad centralizada.

Por lo tanto, en este trabajo se explorará la posibilidad de solucionar la deuda circular en el ámbito de las empresas. El incentivo de una empresa a inscribir su deuda con otras entidades en esta aplicación es que por un lado tienen la garantía de que es totalmente descentralizada y confiable, y, por otro lado, se ahorrarían pagar comisiones a una entidad bancaria para recibir préstamos que cancelen una deuda que tienen cuando puede ser cancelada perfectamente si es de carácter circular con la plataforma.

1.2 BREVE HISTORIA DE LA TECNOLOGÍA

Blockchain nació en el año 2008 de la mano de Satoshi Nakamoto. Este individuo (o grupo de individuos) es desconocido a día de hoy, pero es el nombre que se le da al creador ya que dicho nombre aparece en el *White Paper* de la tecnología. Satoshi implemento Blockchain en el protocolo Bitcoin, que actualmente sigue siendo el más relevante.

Sin embargo, en 2019, se encuentran disponibles varios protocolos además de Bitcoin como Ethereum o Ripple, todos ellos con rasgos distintivos sobre los demás. Este trabajo se centrará en el protocolo Ethereum, que permite el desarrollo de *Smart Contracts* que dotan al protocolo de una elevada flexibilidad y versatilidad, frente a los limitados *Scripts* que ofrece bitcoin.

Capítulo 2. ESTADO DEL ARTE

A continuación, se procede a una explicación exhaustiva de la tecnología Blockchain. Se tratará mas en detalle aquellos puntos que se consideren de mayor importancia.

2.1 IDEAS BÁSICAS DE BLOCKCHAIN

La red Blockchain permite establecer una relación de confianza incluso entre entidades que no tengan una relación entre sí. La forma en la que esto se consigue es a través de la validación (se guarda la información en un "libro mayor" descentralizado), la verificación (se crean requerimientos para la creación de un nuevo bloque mediante *Proof Of Work*, concepto que se explicara más adelante) y consenso (necesario para incluir un nuevo bloque en la cadena). Además, los mineros de los bloques están incentivados a ser honestos (no incluir transacciones fraudulentas en un bloque), ya que se les recompensa por la creación de un bloque, y si tratan de engañar a la red, el resto de los nodos detectaría el engaño y todo el trabajo (POW) realizado por el minero seria en vano. Por otro lado, también están fuertemente desincentivados los fraudes a gran escala, ya que, en caso de producirse, al ser muy volátil la moneda, bajaría de valor inmediatamente perdiendo así las potenciales ganancias de dicho fraude. Cabe destacar que gran parte de la seguridad de la Blockchain recae en el uso de funciones resumen (referidas de ahora en adelante como hashes) como SHA 256 y keccak256 en el caso de Ethereum, además de en el uso de claves privadas y públicas creadas con el algoritmo Eliptic Curve Criptography (ECC) y los arboles de Merkel, que permiten, entre otras cosas, correr nodos ligeros. Estos conceptos serán expandidos en la sección de bitcoin. Toda esta seguridad garantiza la inmutabilidad, la autenticación y el no repudio.

En esencia, existen 3 tipos diferenciados de protocolos en Blockchain en función de su nivel de lógica.

- Tipo 1. Poseen la condición de criptomoneda y escasa lógica. El mejor ejemplo es Bitcoin
- Tipo 2. Integran junto con la condición de criptomoneda una lógica más avanzada y *Turing complete*, es decir, que es capaz de simular una máquina de Turing (daría solución a cualquier problema si tuviese suficiente memoria y tiempo). Esta capacidad de lógica tiende a ser referida como lógica de negocio (*business logic*). Un claro ejemplo es Ethereum.
- Tipo 3: Solo posee la lógica de negocio y no tiene condición de criptomoneda. Un ejemplo que destacar es Hyperledger de IBM, gestionada por *The Linux Foundation.*

Por otro lado, en función de su privacidad existen tres tipos

- Blockchain Publica: Cualquiera puede participar y es *open source*.
- Blockchain Privada: Esta reservada para usuarios elegidos (por ejemplo, una organización)
- Blockchain permisionada: Permiten otorgar por consenso a una serie de usuarios la responsabilidad de crear bloques. Suelen darse en organizaciones de un mismo sector como puede ser el sanitario.

2.1.1 BITCOIN

El protocolo de Bitcoin consigue tres ideas clave: Validación, verificación y consenso. Esto lo hace a través de un libro mayor común para todos los integrantes. Se procede a explicar los conceptos claves que caracterizan a este protocolo. Se ha de mencionar que algunos de los conceptos que se van a tratar son comunes con otros protocolos como puede ser Ethereum. En tal caso, se mencionarán las diferencias con este protocolo, además de ampliarse las mismas en el apartado dedicado a Ethereum.

2.1.1.1 Nodos y arboles de Merkel

En primer lugar, se comienza explicando el concepto de nodo en Blockchain, los tipos de nodos y porque los árboles de Merkel resultan tan útiles. Un nodo constituye el elemento

esencial de cualquier protocolo Blockchain, y ha de guardar una copia de todas las transacciones realizadas en el protocolo en cuestión para asegurar la confiabilidad. Los nodos están repartidos por toda la red, y cuanto mayor sea el número de nodos más descentralizada será esta. Sin embargo, existe un problema. Para guardar todas las transacciones por ejemplo de Bitcoin, se requiere un mínimo de 145 GB de espacio y 2 GB de RAM [5], y a medida que aumenta la cadena de bloques, más espacio será requerido.

Satoshi Nakamoto fue consciente de esta dificultad, y delego la resolución del problema en la Ley de Moore, la cual sostiene que la capacidad de almacenamiento de los dispositivos aumentaría con el tiempo a mayor velocidad de lo que lo haría la red. La ley de Moore establece que cada dos años las capacidades de las computadoras se duplicarían. Sin embargo, esta ley no parece estar cumpliéndose a día de hoy, como afirma el CEO de NVIDIA [6]. Como ya se ha visto antes, a cualquier red pública que use la tecnología Blockchain le conviene tener el mayor número de usuarios (nodos) posibles. Pero a medida que pasa el tiempo y con el aumento del espacio requerido, parece ser que las barreras de entrada son mayores. Por ello se introduce el concepto de nodo ligero.

Un nodo ligero requiere un nodo completo para poder operar. El proceso de conexión entre ambos se denomina suscripción. Se dice que un nodo ligero se suscribe a un nodo completo. Pero ¿Cómo puede un nodo ligero realizar consultas verificables a un nodo completo? En principio, si el nodo ligero dependiese única y exclusivamente del nodo completo, esto podría llevar a fraudes por parte de un nodo completo comprometido que mandase información falsa al nodo ligero. Para conseguir la confiabilidad, los nodos ligeros hacen uso de un árbol de Merkel (*Merkel Tree*).

El árbol de Merkel en Bitcoin es sencillo comparado con el de Ethereum, que requiere 3 árboles de Merkel independientes. Bitcoin hace uso únicamente de un árbol de Merkel de transacciones. El caso de Ethereum se explicará en su propio apartado. En Bitcoin, se saca la función hash de todas las transacciones. Esto se conoce como un *leaf node*. El hash utilizado es el SHA-256 que es una función unidireccional capaz de transformar cualquier dato en 256 bits. Esta función no tiene inversa. Nótese que la probabilidad de colisión si se

tuviese mil millones de transacciones (por poner un ejemplo exagerado) sería de 4.3×10^{-60}. Esta probabilidad, es despreciable y, por consiguiente, se puede considerar un algoritmo seguro.

Tras esto, se juntan los hashes de las transacciones en pares, se concatenan y se saca el hash de dicha concatenación. Se continua con este proceso hasta que queda un único hash. Este hash se denomina el hash raíz. En la figura que se muestra a continuación se representa este concepto gráficamente.

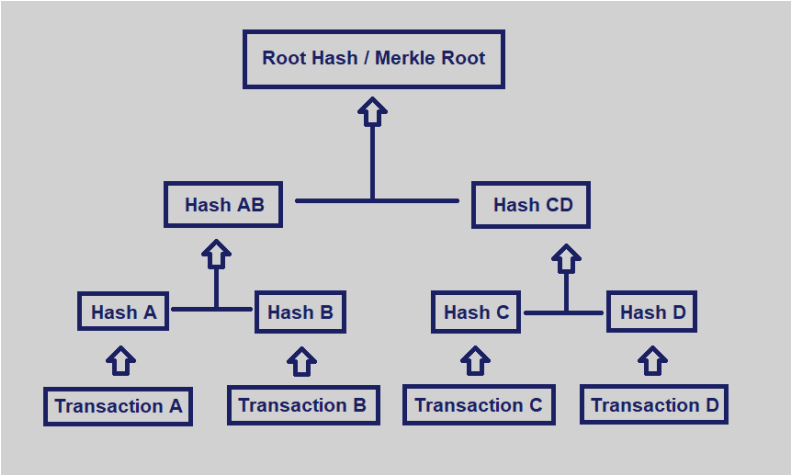

Figura 1. Árbol de Merkel de transacciones

Ahora bien, ¿cómo puede aprovecharse de los árboles de Merkel un nodo ligero? Un nodo ligero guarda únicamente las cabeceras de los bloques. Nada más. Esta cabecera incluye el hash raíz mencionado anteriormente. Por tanto, si un nodo ligero quiere comprobar cierta transacción, lo único que tiene que hacer es pedir la rama de dicha transacción, y comprobar la validez de los hashes recibidos. Nótese que solo requiere la rama de la transacción que está examinando y el hash colindante en cada punto de la rama. Se muestra una imagen a continuación para aclarar esta idea.

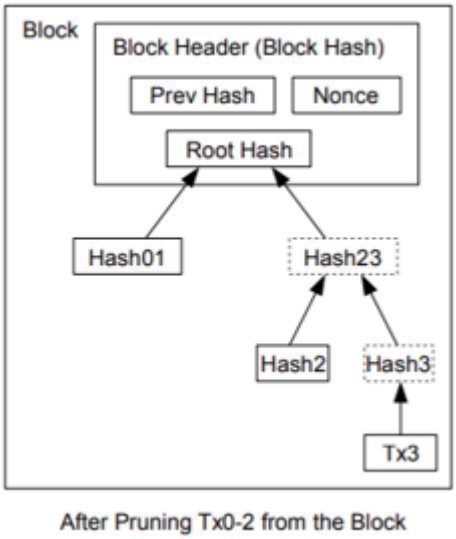

Figura 2 Ejemplificación de una petición de una rama de un nodo ligero

De esta forma los nodos ligeros tienen absoluta certeza de que la información que les proporciona el nodo completo al que están suscritos es veraz, debido a que la información de las cabeceras la reciben de la red directamente.

2.1.1.2 Creacion de una dirección pública.

Bitcoin utiliza para sus cuentas un par de claves publica privada. La clave publica se genera a partir de la privada (la clave privada no se puede recuperar conocida la clave publica). Este par de claves permiten cifrar cualquier transacción con una clave y descifrarla con su complementaria.

El procedimiento para crearse una dirección publica es el siguiente:

1. Se obtiene una clave privada, de 32 bytes, a través de un algoritmo ECDSA.
2. A partir de la clave privada se obtiene la clave publica, de 65 bytes.
3. Se aplica a dicha clave publica un hash mediante el algoritmo SHA-256 y se obtiene un resultado de 32 bytes.
4. Se toma dicho hash y se le aplica un algoritmo de hash RIPEMD-160. El resultado es un valor de 20 bytes.

5. A dicho valor se le concatena un byte de versión al comienzo. El byte de versión para la red principal es 0x00.
6. A dicho valor se le aplica el algoritmo SHA-256 2 veces.
7. A continuación, se toman los 4 primeros bytes y se concatenan con el valor del algoritmo RIPEMD-160 calculado en el paso 4.
8. El resultado obtenido es la dirección publica, que no ha de ser confundida con la clave publica, con un tamaño de 25 bytes.

2.1.1.3 Proceso de una transacción.

El proceso para completar una transacción en Bitcoin es relativamente sencillo. Antes de explicar los pasos que se siguen se ha de destacar la manera en la que una transacción es reconocida como válida por parte del resto de nodos. Bitcoin hace uso de UTXO (*Unspent Transaction Output*), es decir, son transacciones que una determinada cuenta posee el derecho de utilizar (ya que las ha recibido de otra cuenta o en una recompensa de bloque). Por tanto, las cuentas no tienen un balance como tal (este es el caso de Ethereum) sino que tienen "derechos de uso" de transacciones. Es importante mencionar que dichas transacciones no pueden ser utilizadas parcialmente, es decir, si un usuario tiene el derecho de uso de una transacciones cuyo valor es 1BTC, y desea enviar 0,5BTC a otro usuario, este deberá, en la sección de *outputs* devolver la parte que no manda al usuario al propio remitente. De no ser así, el minero que cree el próximo bloque tomará la diferencia entre el *input* y el *output* como su comisión. La siguiente imagen trata de aclarar esta idea:

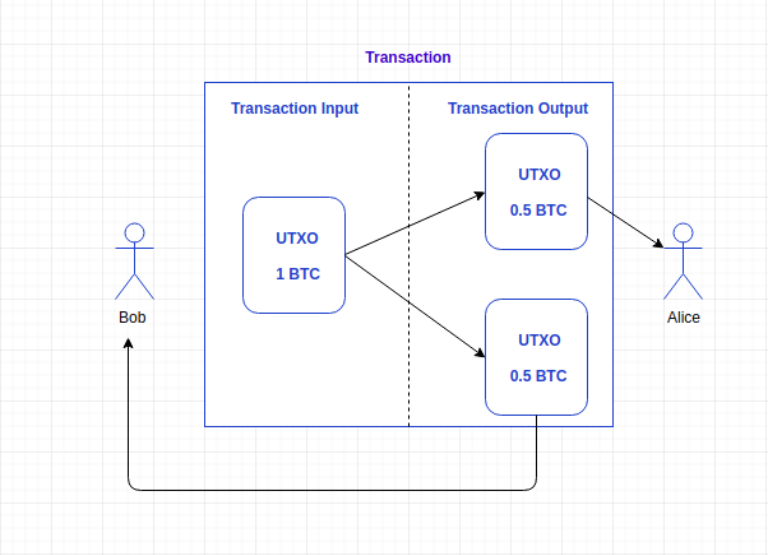

Figura 3. Ejemplo de uso de UTXO

Supóngase que el usuario A desea enviar bitcoins al usuario B. El usuario A tiene el id de transacción que recibió en el pasado (generado a partir de un hash sha-256 de la transacción recibida). El usuario A concatena ese id de transacción, cuyo tamaño es de 32 bytes con la dirección publica de B, cuyo tamaño es de 25 bytes (como se explicó antes) y computa el hash sha-256 de esa concatenación. Una duda razonable que puede surgir aquí es si puede darse un problema con la inyectividad (es decir, que el hash de ab+cd equivale al de abc+d). Sin embargo, teniendo en cuenta que los tamaños tanto del id de transacción como de la dirección publica son fijos, no supone ningún problema. Tras esto, se encripta dicho valor con la clave privada de A. A continuación, cualquier minero que desee validar la transacción tan solo debe aplicar la clave publica de A para desencriptar la transacción (garantizando así que viene de A), realizar el mismo proceso que realizo A y comprobar que los valores que obtiene coinciden con los enviados por A. Si este es el caso entonces la transacción es válida. Si no, se descarta.

Los pasos seguidos para que se complete una transacción son los siguientes:

1. Las transacciones validas se almacenan en una *Pool* de transacciones sin confirmar. Esta Pool contiene muchas transacciones con dos campos: La cantidad que se envía y la cantidad que se recibe y por quien. La diferencia entre ambas cantidades constituye la comisión que se llevara el minero al minar dicha transacción. Es, por tanto, que los mineros a la hora de formar un bloque con transacciones elegirán aquellas transacciones que les ofrezcan una mayor comisión.
2. Los mineros resuelven con las transacciones que ofrecen mayor comisión un bloque a través de un *Proof Of Work* (concepto que se detallara más adelante).
3. Una vez se resuelve un bloque, se difunde a toda la red.
4. Cada nodo verifica la validez de ese bloque y procede a añadirlo a su propia cadena. De esta manera (salvo casos de *forking* que se detallaran posteriormente) se garantiza que la transacción ha sido completada con éxito. A continuación, se muestra un esquema del proceso.

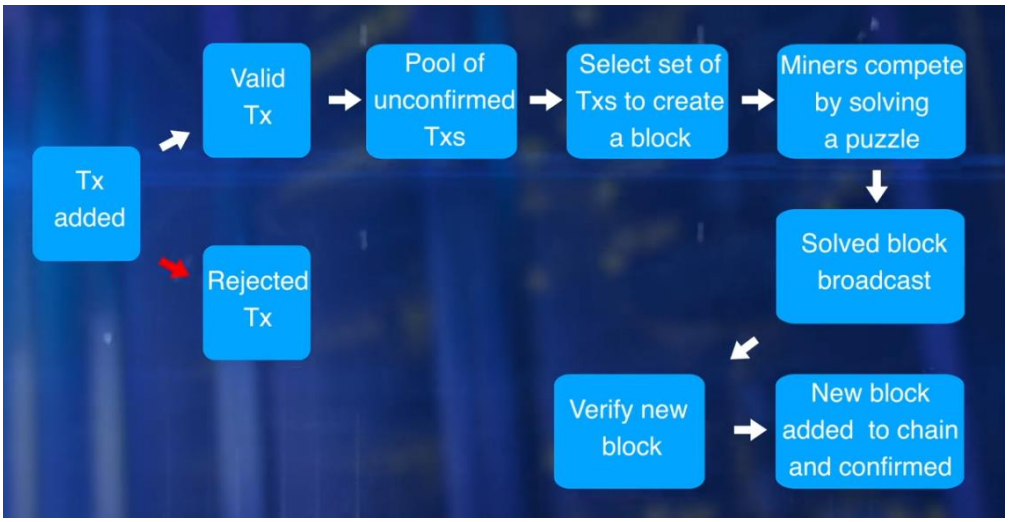

Figura 4. Proceso de confirmación de una transacción en Bitcoin

Todo este proceso puede parecer complejo, sin embargo, la existencia de Billeteras online consigue que el usuario medio no deba de preocuparse por todo esto. Estas billeteras facilitan el acercamiento de la tecnología media a la población que no comprende estos fenómenos a bajo nivel.

2.1.1.4 Proceso de creacion de un bloque. Mineros

Se denomina minero a cualquier nodo que participe activamente en la creacion de bloques, en busca de la recompensa que otorga la creación de este. Esta recompensa procede de las comisiones de las transacciones y de la propia recompensa del bloque, conocida de antemano.

En primer lugar, un minero elige el número de transacciones que prefiere minar (optara por aquellas que ofrezcan una mayor comisión). Tras esto, une estas transacciones a una cabecera que tiene una serie de campos relevantes:

- **hashPrevBlock**: Contiene el hash del bloque anterior.
- **hashmerlkleRoot**: Contiene el hash de las transacciones visto anteriormente
- **Time**: El segundo en el que se mina el bloque.
- **Nonce**: Un numero de 32 bits que se puede variar.

Una vez el minero cree el bloque, este realizara el hash de este (realmente de la cabecera del bloque), buscando que este hash comience por un numero de ceros determinado por la red. Por ejemplo, actualmente el número de bits a cero necesarios para la creacion de un bloque es de 76 (sobre los 256). El número de 0s necesario va variando a medida que más o menos mineros existan en la red, con el objetivo de mantener la media de tiempo de creacion entre bloques constante (unos 10 minutos). Por tanto, si hubiese menos mineros (menos potencia de minado) el número de ceros requerido sería menor que si hubiese más.

Se recuerda que el algoritmo de hash es totalmente impredecible, y por tanto la única manera de obtener un numero con ciertos parámetros (en este caso cierto número de 0s) es probar múltiples veces. La forma en la que el hash del bloque puede cambiar es variando dos campos: El campo *nonce* y el campo *time*, que va cambiando cada segundo. Con la variación de estos se pueden conseguir un numero lo suficientemente elevado de hashes para que entre ellos se encuentre uno de los deseados.

Todos los mineros de la red compiten para buscar dicho hash. Una vez encontrado por un minero, este lo difunde a toda la red. Cada nodo de la red comprueba que el cálculo es

correcto, y que todas las transacciones del bloque son válidas. Tras esto lo añaden a su propia cadena de bloques y se comienza una nueva carrera por encontrar el siguiente bloque. En el caso de que más de dos mineros encuentren un bloque cuyo hash cumpla los parámetros especificados, se produce una situación de forking, la cual se expandirá más adelante.

Finalmente se muestra una gráfica que expresa como a medida que se han ido introduciendo más potencia de minado en la red, para mantener la media de tiempo constante se ha aumentado la dificultad de minado de los bloques. En el eje de las ordenadas se muestra los PetaHashes/s (el número de hashes de la red por segundo) y la dificultad de los bloques, y en el eje de abscisas la fecha.

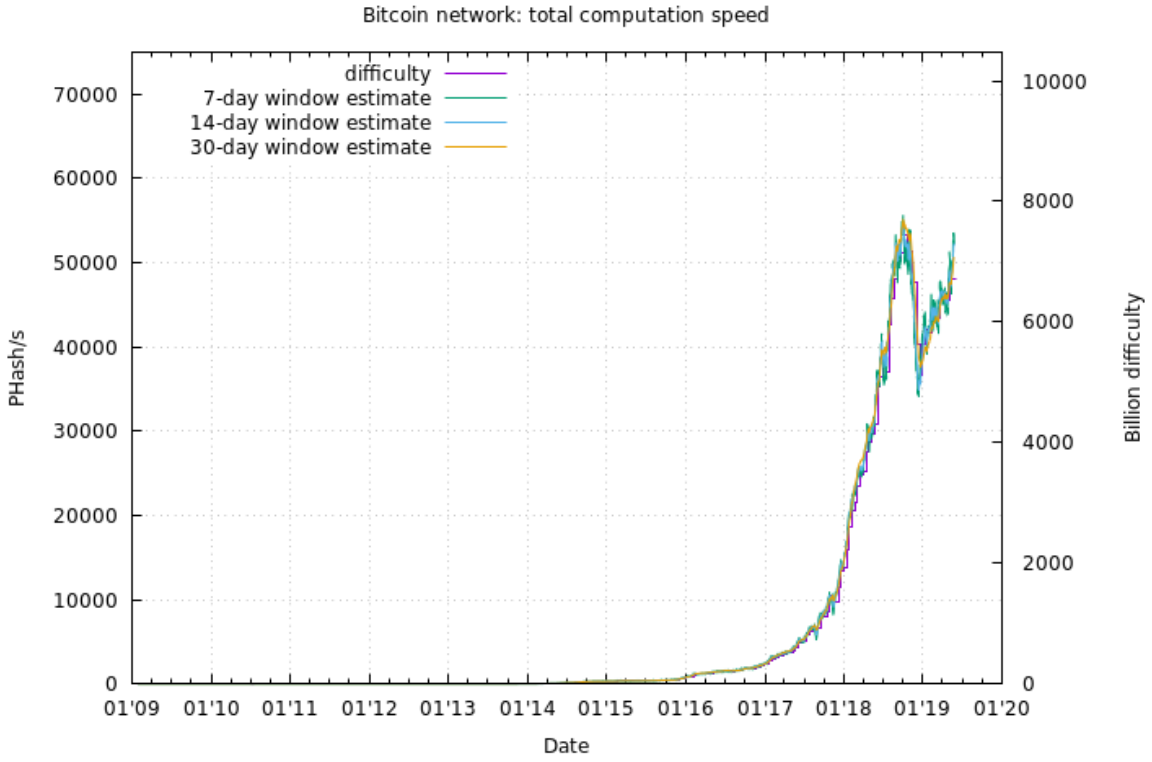

Figura 5. Variación en la dificultad de minado de bloque con el tiempo

2.1.2 ETHEREUM

Ethereum no es más que otro protocolo en Blockchain, creado por joven programador ruso-canadiense Vitalic Buterin. La intención de este protocolo es la de crear un ordenador descentralizado, a través del uso de *Smart Contracts*. La criptomoneda utilizada por Ethereum es el Ether. A continuación, se estudiarán las diferencias y componentes distintivas de este protocolo.

A grandes rasgos, cabe mencionar que, para el minado de bloques, Ethereum usa Ethash y para muchas de las operaciones como los Merkel Patricia Tries utiliza Kecckak-256 (una variante de SHA-3)

2.1.2.1 Cuentas y balances

Existen dos tipos de cuentas en Ethereum. Las cuentas de usuarios (*Externally Owned Accounts*) y las cuentas de contratos.

- **Cuentas de usuarios:** Son controladas por usuarios y tienen asociadas un balance de Ether. De forma similar que, en bitcoin, esta cuenta está construida a partir de una clave privada, derivando de la misma la clave publica y obteniendo la dirección de la cuenta. Estas cuentas son necesarias para participar en la red Ethereum.

- **Cuentas de contratos:** Una cuenta de contrato ha de ser desplegada por una cuenta de usuario. La forma en la que se determina la dirección de la cuenta de contrato es juntando la dirección de la cuenta de usuario con un nonce, y obteniendo el hash de esto. Es importante señalar que este nonce no es un valor insignificante como en la creacion de bloques, sino que es un valor que va aumentando de uno en uno a medida

que una cuenta de usuario vaya realizando transacciones. Esta cuenta contiene una balance de Ether y un código asociado a la cuenta.

La interacción entre cuenta de usuarios y cuenta de contratos es muy sencilla, y se explicara en detalle en la sección de *Smart Contracts*.

En lo que concierne a los balances, Ethereum guarda en cada bloque los cambios de balance de cada uno de los usuarios involucrados en las transacciones del bloque. Los balances de todos los usuarios están representados en cada bloque a través del *State Root* que se comentara más adelante junto con el Merkel Patricia Trie. De esta manera, se puede saber el balance de todas las cuentas en la red, en vez de hacer uso de las UTXOs.

2.1.2.2 Diferencias entre Ether, Wei y Gas.

El Ether es la criptomoneda que utiliza la red Ethereum. En 2019, el equivalente del Ether al dólar esta alrededor de los $200. Es por tanto que para transacciones más pequeñas se tiende a utilizar una unidad de medida menor, denominada Wei. (También es común encontrar el termino Giga Wei como Gwei, ya que es más manejable). La conversión es:

$$1 \text{ ETH} = 10^{18} \text{ Wei}$$

Por lo general es bastante común usar el termino Wei en especial para las micro transacciones.

Por otro lado, el concepto de Gas es algo más sutil. Para comenzar a explicarlo, se ha de entender que el código que existe en una cuenta de contrato ha de ejecutarse. Esta ejecución, requiere procesamiento por parte del minero y de los nodos. Este procesamiento tiene un precio. No es lo mismo ejecutar un simple código condicional que guarde un solo valor en almacenamiento, que código con varios bucles y que guarde numerosas variables en almacenamiento. El segundo código requiere mucho más procesamiento. Es por todo esto que la ejecución de código ha de ser cobrada. Pero esta cifra no se puede dar ni en Ether ni en Gwei, ya que estos valores cambian con el mercado. Se utiliza por tanto un valor llamado Gas, que tiene una conversión a Ether que varía con las propias variaciones del Ether con el

mercado. Se puede decir que el Gas es constante ya que una operación cuesta lo que cuesta, y no depende del valor de mercado de la moneda.

Además, los bloques poseen dos campos relacionados con el Gas. El límite de gas por bloque, del que depende el número de transacciones que se incluyan en el mismo y el mínimo de gas por bloque. El mínimo establecido es de 1,5 pi millones de Gas (4,712,388) en la versión de Ethereum *Homestead*. El límite de gas es votado en cada bloque. El minero de dicho bloque podrá decidir variar el límite de gas por un factor de 0.0976%.

El concepto del Gas es de gran importancia, y al final de este trabajo, se procederá a considerar el Gas como una nueva variable programática a la hora de escribir el código de un *Smart Contract*.

2.1.2.3 Tiempo de convergencia en la creacion de bloques. Consecuencias

Como ya se mencionó anteriormente, en la red Bitcoin un bloque tarda de media en crearse 10 minutos, ajustándose la dificultad de creacion en función de la potencia de la red. En la red Ethereum, sin embargo, un bloque tarda de media en crearse entre 10 y 19 segundos.

También se discutió anteriormente que en caso de que dos mineros distintos creasen un bloque valido en un periodo de tiempo cercano, se daría un problema de forking. Este problema es mucho más común en Ethereum ya que hay menos tiempo para crear un bloque, y el tiempo de difusión del bloque a la red en su totalidad es similar (en torno a los 12 segundos para llegar al 50% de la red). Por esto, Ethereum creo un sistema de recompensa para aquellos bloques que, aunque no formasen parte de la cadena principal habían sido creados cumpliendo con los requisitos de dificultad establecidos.

Un minero que consiga crear un bloque en Ethereum recibe una recompensa de 3 Ether y las comisiones pertinentes de las transacciones. El minero que también resuelve el bloque, pero dicho bloque no pertenece a la cadena principal se denomina *Ommer* (significa “repetición” en danés). El bloque que se crea se denomina bloque Ommer o *Uncle Block*. También puede darse que un nodo creé, por el problema del *forking*, un bloque a partir del bloque Ommer.

Este bloque recibe el nombre de *Nephew block*. Para dar solución al problema del *forking* en Ethereum se introdujo, en 2013 el protocolo GHOST (Greedy Heaviest Observed Subtree)

Protocolo GHOST: Este protocolo, que entró en vigor en 2013, estipula una serie de recompensas para los creadores de *Uncle Blocks* y *Nephew Blocks*. De estos bloques, las transacciones son ignoradas y únicamente se tiene en cuenta las cabeceras. El creador de un Uncle Block recibirá un 87.5% de la recompensa de un bloque (no recibirá las comisiones de las transacciones, solo la parte de la recompensa del bloque) y un *Nephew Block* un 12.5% de la recompensa.

2.1.2.4 Merkel Patricia Trie.

Como se ha mencionado antes, Ethereum toma el concepto de Árbol de Merkel y lo lleva un paso más allá. para garantizar, entre otras cosas, la confiabilidad de los nodos ligeros. El Merkel Patricia Trie no es más que el uso de tres arboles de Merkel separados. Son los siguientes:

- ***Transaction Tree:*** El árbol de transacción en Ethereum otorga a la red la misma funcionalidad que el árbol de Merkel en bitcoin Crea hash raíz con todas las transacciones. El procedimiento para conseguir este hash raíz ya fue explicado en el apartado de Bitcoin y por tanto se omite.
- ***State Tree:*** El árbol de estado consiste en esencia en un *mapping* clave-valor. La clave está formada por la dirección de una cuenta y el valor está comprendido por 4 parámetros:
    - *Nonce:* Este valor sirve para mantener la cuenta del número de transacciones que lleva esa cuenta de usuario. Esto evita que se realicen transacciones en desorden, ya que, para que una transacción sea válida, ha de tener un nonce cuyo valor sea uno más que el nonce de la transacción anterior.
    - *Balance*: El Ether total que tiene esa cuenta de usuario o cuenta de contrato en Wei.
    - *Raíz de almacenamiento:* Este valor es relevante únicamente para las cuentas de contratos y representan las variables almacenadas por el contrato.

- *Hash del código:* De nuevo solo es necesario tener en cuenta este valor en las cuentas de contratos. Se realiza un hash de todo el código y se almacena.

Por tanto, en cada bloque, cuando cambia el valor de una cuenta, se hacen las modificaciones necesarias en el mapping, relacionándolo con las ramas de bloques anteriores. Por tanto, para conocer el estado de Ethereum se ha de tener acceso a toda la cadena de bloques. La siguiente imagen representa perfectamente la explicación mencionada.

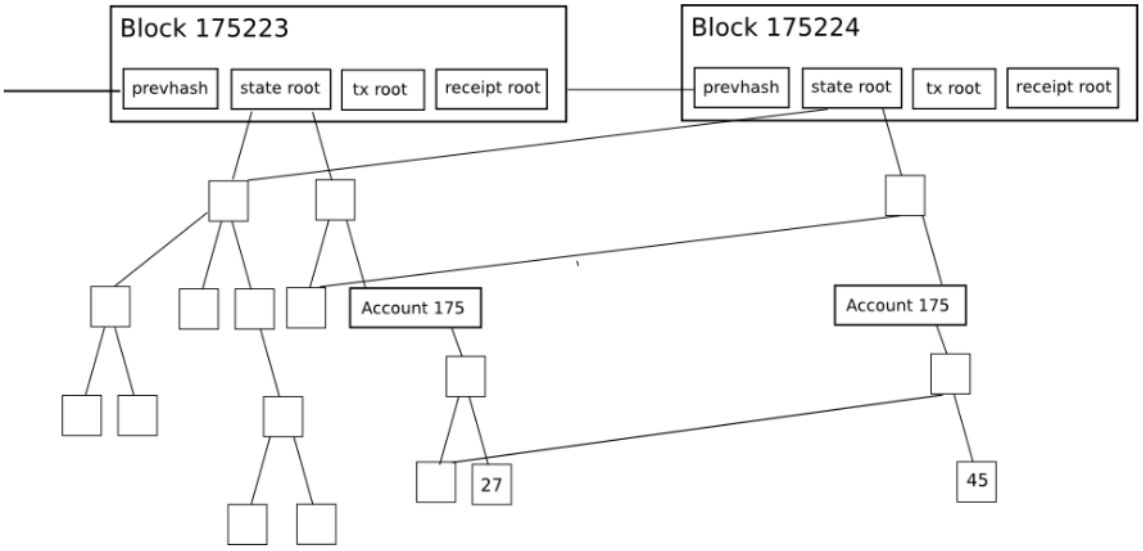

Figura 6. Ejemplo de modificación del árbol de estado entre dos bloques

- ***Receipt Tree:*** Cada transacción que se produce en la red produce un recibo con información relevante de la misma. Se construye un árbol de Merkel con los recibos de la transacción de la misma forma que con el árbol de transacciones y el ultimo hash es el recibo raíz.

2.1.3 TÉCNICAS QUE GARANTIZAN LA SEGURIDAD

En este apartado se exploran una serie de ideas que garantizan la seguridad de la red. La integridad de la cadena de bloques ya se ha visto que se mantiene ya que cada bloque en su

cabecera ha de incorporar el hash del bloque anterior. El no repudio en las transacciones se ha visto con el concepto de firma de la transacción (cifrado) a través de la clave privada.

Se tratarán una serie de temas que se han dejado para el final de este apartado como el forking, el concepto de *Proof of Work* detallado…

2.1.3.1 El concepto de Forking

El concepto de *Fork* se puede resumir como la situación que se produce cuando una cadena de bloques diverge en dos caminos potenciales. Esto se puede producir porque dos mineros han obtenido un bloque valido en un tiempo cercano y han difundido a la red su bloque. Parte de la red ha recibido un bloque antes que otro y viceversa. La siguiente figura se muestra para ilustrar del concepto.

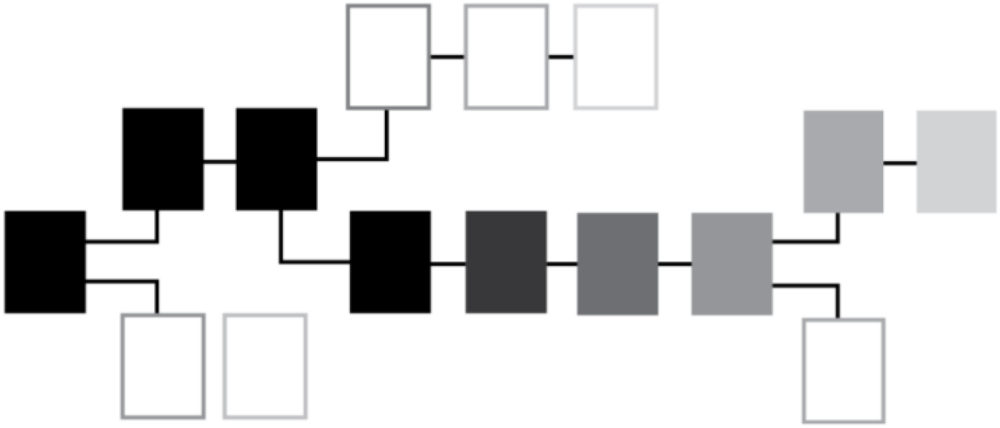

Figura 7. Concepto de forking

Nótese que, en la imagen, los bloques más oscuros son los que son considerados bloques más fiables. Esto se debe a que hay suficientes bloques posteriores para garantizar con mucha certeza que esa cadena es válida. Los nodos tienen en cuenta cual es la cadena más larga. Si se observa el final de la cadena de la Figura 6, la rama superior es más fiables que la inferior, pero si se produjesen dos bloques seguidos en la inferior, entonces la cadena principal pasaría a ser esta. La razón principal del forking es que existe un tiempo de propagación del bloque a la red que no es despreciable.

Los bloques blancos son los comentados anteriormente (*Ommer Blocks* y sus "hijos").

Por otro lado, un forking de la cadena de bloques puede ser realizado por la comunidad de manera voluntaria para introducir modificaciones y actualizaciones que otorgan mayor robustez al protocolo Blockchain en cuestión. Estos pueden ser por cuestiones de seguridad, o para variar parámetros como la recompensa del bloque, su tamaño máximo…En este caso diferenciamos entre dos tipos de *forking* con significados muy distintos; *Soft Fork* y *Hard Fork.*

- **Soft Fork:** Se produce cuando las modificaciones sobre el protocolo permiten la compatibilidad retrospectiva (es decir, la nueva cadena de bloques puede existir con la antigua). Suelen ser condiciones más restrictivas que las preexistentes (por ejemplo, reducir el tamaño máximo de bloque)
- **Hard Fork:** Se produce cuando las nuevas condiciones propuestas son más permisivas que las prexistentes. En este caso no es posible la compatibilidad retrospectiva. Un ejemplo de esto es la separación de bitcoin y bitcoin cash. Bitcoin Cash permite que el tamaño máximo de un bloque sea de 8Mb frente a los 1-2 Mb de Bitcoin. En la siguiente imagen se muestran algunos de los *Hard Forks* que ha tenido Bitcoin.

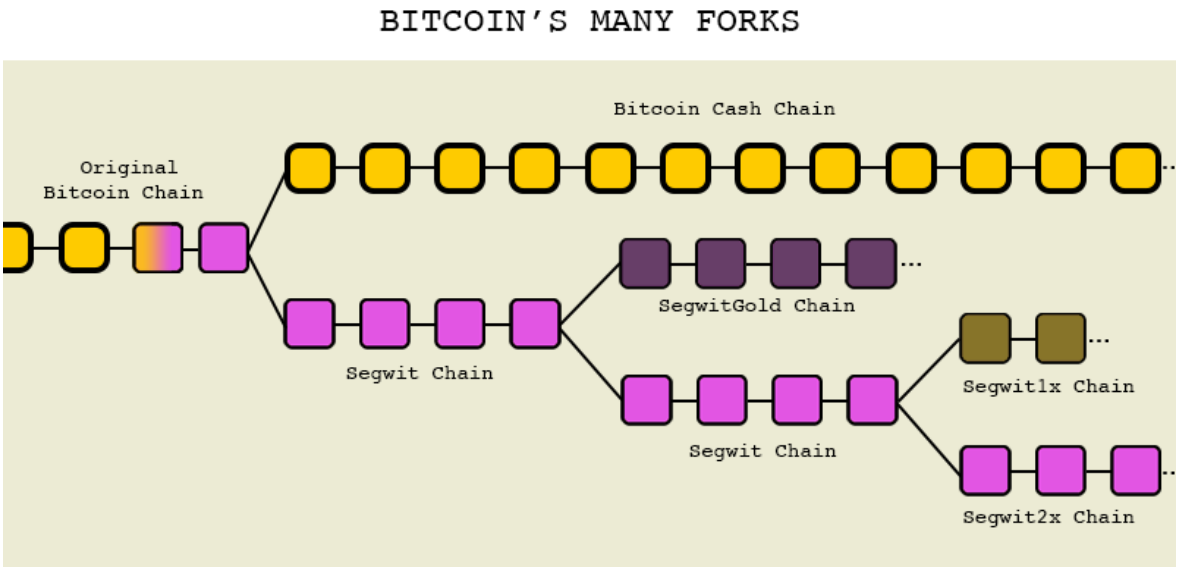

Figura 8. Hard Forks de Bitcoin

2.1.3.2 Tipos de Proofs para la validación de un bloque.

Proof Of Work (POW) es el tipo de validación de bloques más conocido y utilizado. Se ha explicado su funcionamiento en apartados anteriores, pero en esencia consiste en buscar un hash de la cabecera del bloque que cumpla con los parámetros pedidos por la red en ese momento. El mayor problema que presenta este tipo de validación es que requiere una enorme capacidad de procesamiento. Se estima que, solo el protocolo de Bitcoin utiliza en torno a un 0,3% de la electricidad del mundo de manera anual [7]. Por otro lado, este método de validación favorece a aquellos mineros con más equipamiento y fomenta la creacion de *Pools* de minería, es decir, situaciones en las que un conjunto de mineros se pone de acuerdo para minar juntando su capacidad de procesamiento y dividiendo la recompensa en función de la potencia suministrada a la *Pool*. Téngase en cuenta que si los 3 principales pools de minería se pusiesen de acuerdo podrían realizar un "ataque del 51%", es decir, que serían capaz de controlar más de la mitad de la red y por tanto aprobar los bloques que ellos deseen de manera fraudulenta. Todo esto es peligroso ya que puede llevar a la centralización.

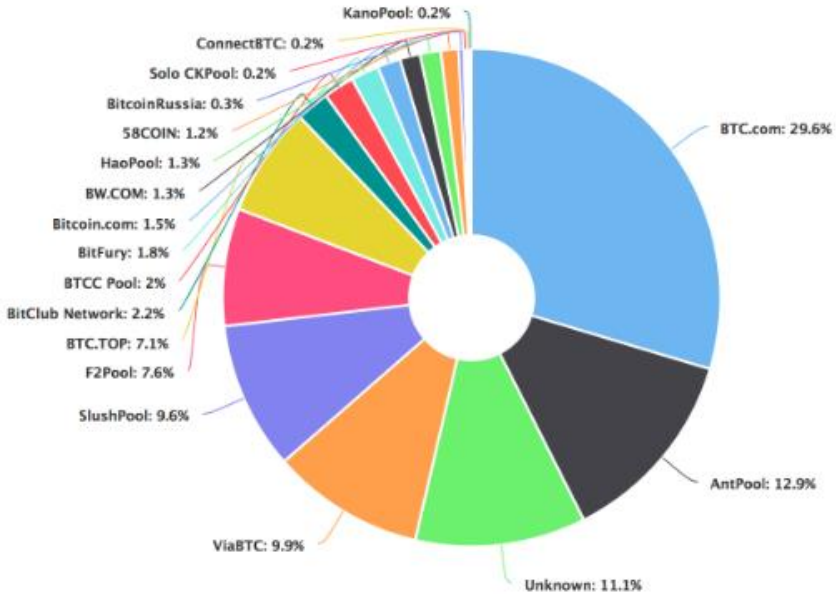

Figura 9. Principales Pools de minería

Estos problemas supusieron que se propusiesen nuevos tipos de validación de bloques, tratando de mantener la naturaleza descentralizada de la red. Estos son solo algunos de los tipos de validaciones que han sido propuestos y que son utilizados, normalmente en combinación con POW, en numerosos protocolos. Se enfatiza POS (Proof Of Stake) ya que es el que más importancia está teniendo después de POW y el que con el que combina Ethereum POW.

- ***Proof of Stake (POS):*** La idea fue introducida por primera vez por un usuario en el foro *bitcointalk* [8]. Para evitar el desperdicio eléctrico que supone que todos los mineros estén independientemente buscando el próximo bloque, se establece un sistema de elecciones para decidir quién acuñara el bloque. Nótese que se ha empleado el termino acuñar en vez de minar, ya que esta es la manera en la que se denomina a la creacion de bloques en POS. El procedimiento es muy sencillo. Todos los que deseen participar en el proceso de la creacion del próximo bloque han de depositar una inversión (*Stake)* en la red. Tras esto, la red establece una probabilidad a cada usuario, en función de la inversión depositada, de ser el que acuñe el próximo

bloque. La función de probabilidad empleada es lineal (protege mejor a los usuarios con menos capital ya que los que tienen mas no disponen de la economía de escala que les proporciona POW, teniendo la cuenta que el precio de la electricidad por unidad disminuye con la cantidad.) La inversión original no será devuelta al creador del bloque hasta que el resto de los nodos verifiquen la integridad y validez del bloque creado. En caso de que algunos nodos también estén comprometidos, estos perderán parte de la inversión que ellos depositaron si toman como valido un bloque invalido. De esa manera, ni el acuñador ni los nodos validadores tienen incentivos para aprobar un bloque fraudulento. Se estima que en junio de 2019 se publicara el código para el algoritmo de Ethereum POS, conocido como Casper [9].

- ***Proof of activity:*** Este protocolo combina POW y POS. Un ejemplo de criptomoneda que emplea este tipo de validación es *Decred* (DCR).

- ***Proof of burn:*** Consiste en la demostración por parte de los mineros de que han "quemado" criptomonedas minadas mediante POW. Esto se hace mandándolas a una dirección que no está asignada a ningún usuario, haciendo que sea imposible recuperarlas. El principio detrás de esta idea es que los usuarios en vez de consumir recursos (que equivalen a dinero) consuman directamente las criptomonedas. No se profundizará más en detalle sobre este método ya que escapa al objetivo del presente trabajo, pero se ha de mencionar que existen varias variantes de este proceso de verificación y que tiende a combinarse con POW y POS. Un ejemplo de una criptomoneda que lo emplea es *SlimCoin*.

- ***Proof of capacity:*** Este método de verificación se basa en la capacidad de disco duro de los mineros para guardar "plots". A mayor capacidad de almacenamiento mayor es la probabilidad de que se le otorgue al usuario la responsabilidad de crear el siguiente bloque. De nuevo, entrar en profundidad en este método de validación escapa al objetivo del trabajo. Un ejemplo de criptomoneda que lo usa es *BurstCoin*.

2.2 *SMART CONTRACTS*

Una vez comprendidas las principales ideas y herramientas que ofrece la tecnología Blockchain, se centrara el trabajo en el uso de *Smart Contracts* del protocolo Ethereum.

Una sucinta definición de *Smart Contract* es la de un protocolo que garantiza, de manera digital, el cumplimiento de una serie de puntos de un contrato en el que participa una número de usuarios. Se hace uso de líneas de código y una vez ejecutados, al ser parte de la red Blockchain, son irreversibles. Se puede interactuar con ellos a través de transacciones.

Este apartado será dividido en tres partes. Una introducción a las ideas básicas de los *Smart Contracts*, un apartado de Solidity, el lenguaje primordial en el que se escriben estos contratos y una serie de conceptos y mejores prácticas a la hora de programar.

2.2.1 IDEAS PRINCIPALES

El concepto de *Smart Contract* fue considerado por primera vez por el criptógrafo Nick Szabo en 1994 [10]. Actualmente, son utilizados en numerosos protocolos de Blockchain. Se pondrá atención en particular a su uso en el protocolo Ethereum.

Ethereum hace uso de su EVM (*Ethereum Virtual Machine*). Esta máquina es la encargada de ejecutar el código que se escribe en el *Smart Contract*, tras ser traducido de un lenguaje a alto nivel como puede ser Solidity o Viper a lo que se denomina *byte code*. La EVM se puede considerar como el sistema operativo de la tecnología. Todos los nodos de Ethereum corren la misma implementación de EVM, por lo que elimina la necesidad de que los desarrolladores de DApps (más sobre esto en apartados posteriores) tengan que interactuar directamente con la red Blockchain lo cual requiere extensos conocimientos a bajo nivel y constituye un proceso muy tedioso.

Se explicarán las principales ideas sobre como diseñar, desplegar y como podría eliminarse un Smart Contract. También se comentará sucintamente acerca del precio a pagar al minero que ejecuta una llamada al *Smart Contract* y como interactúan con elementos fuera de la red a través de oráculos.

2.2.1.1 Proceso de despliegue y eliminación.

Una vez diseñado el *Smart Contract* en el lenguaje deseado, se procede a su despliegue en la red. Como se ha comentado antes, se requiere una cuenta de usuario (EOA) para desplegar una cuenta de contratos. Esta cuenta de contratos será la que tenga el código del contrato, y se podrá interactuar con ella a través de transacciones.

A continuación, se proceden a explicar los fenómenos que ocurren de fondo en el despliegue de un contrato.

En primer lugar, se ha de recordar que la manera en la que se interactúa con la red Ethereum es únicamente a través de transacciones. Por ello, para desplegar el contrato a la red Ethereum, se ha de enviar una transacción. A continuación, se muestra la estructura de los campos de una transacción en Ethereum para comprender mejor que campos que se emplean en el despliegue [11]:

- **Nonce**: A diferencia del nonce que se usa por parte de los mineros para modificar el valor del hash de la cabecera del bloque, este nonce sirve para indicar el número de transacción enviadas por el usuario. Cuando un minero recibe la transacción, procede a introducir en el árbol de estados de la red Ethereum la dirección del remitente de la transacción como clave del *mapping*, obteniendo así el valor, que consiste en una serie de campos incluyendo el número de transacciones que lleva efectuada hasta ese momento el usuario. Si el numero en la transacción no es un valor inmediatamente superior al nonce el árbol de estados, entonces se descarta la transacción. Esto puede darse ya que un usuario puede mandar más de una transacción a la *pool* de transacciones debiéndose ejecutar en el orden correcto para evitar numerosos problemas como la falta de fondos o el fraude. Este nonce se combina con la dirección pública del remitente para crear la nueva dirección de la cuenta de contratos. Esto se consigue con el codificador de bajo nivel utilizado por Ethereum, RLP. Tras codificar ambos valores con este algoritmo se procede a obtener el hash a

través del algoritmo Keccak-256. En el caso de las cuentas de usuarios, el nonce de su primera transacción comienza con valor 0, mientras que el nonce de un contrato comienza con valor 1 desde la especificación EIP 161 [12], y aumenta con el despliegue de otros contratos de uno en uno.

- **Precio de gas:** Valor en Wei a pagar por unidad de Gas. Si se está dispuesto a pagar más gas por Wei, tu transacción tardara menos en ejecutarse, ya que los mineros la priorizaran para obtener esa recompensa. La siguiente fuente [13] es muy útil para poder calcular el tiempo de manera aproximada que tardaría en producirse tu transacción en función del precio del gas que se introduzca, además de indicar un mínimo seguro.

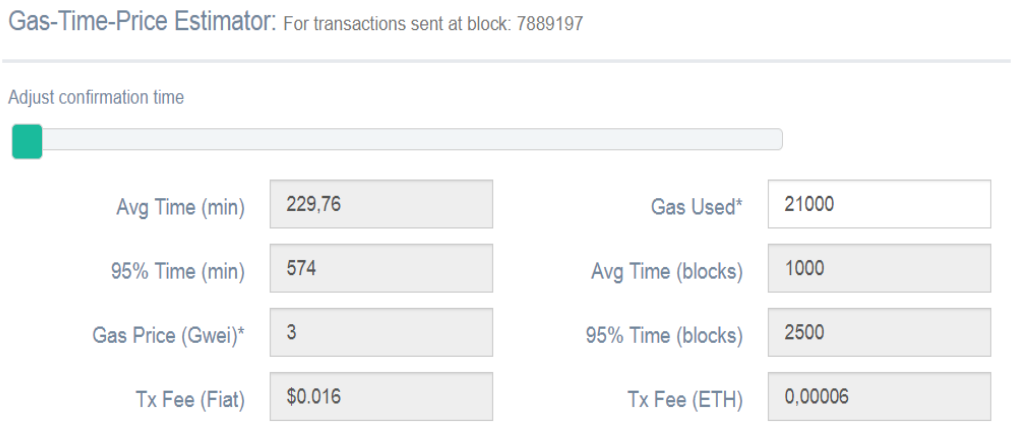

Figura 10. Tiempo medio de confirmación de la transacción a 1 Gwei/Gas para el bloque 7889197

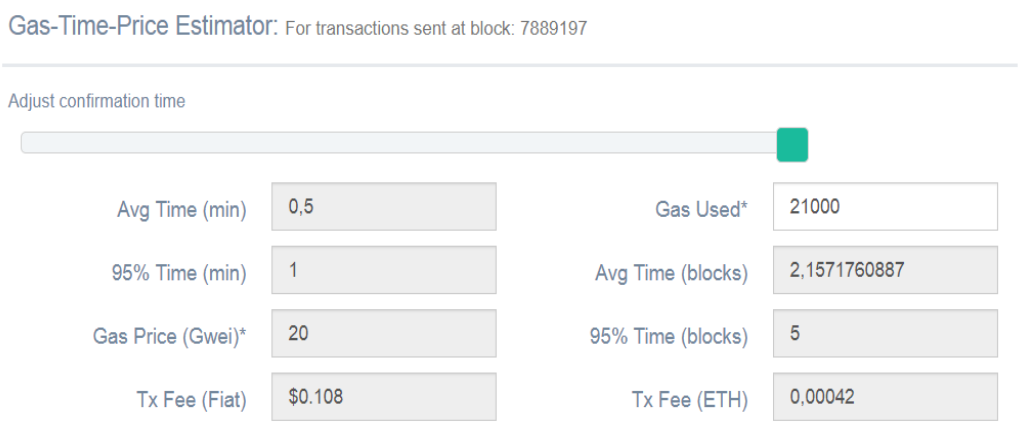

Figura 11. Tiempo medio de confirmación de la transacción a 20 Gwei/Gas para el bloque 7889197

- **Límite de Gas:** El límite establecido de Gas para realizar la transacción. Este valor se paga de antemano, aunque puede no ser usado por completo, y en tal caso se devolverá al remitente.
- **Destinatario**: Aquel que recibe la transacción. En el caso de que se desee desplegar un contrato, el destinatario contendrá el valor 0x00.
- **Valor**: La cantidad en Wei a enviar al destinatario. En caso de que el destinatario sea la creacion de un contrato, este valor le será añadido al contrato. Se ha de recordar que, ya que los contratos también son cuentas, disponen de la opción de tener un saldo.
- **Init/data**: Es un campo de tamaño no definido que contiene el código *bytecode* para ser interpretado por la EVM. En el caso en el que se esté desplegando un contrato, el código que se encuentra en este campo no es el código como tal sino el código que permite la inicialización del contrato.
- **v, r, s**: Estos campos tienen comprobaciones sobre la firma de la transacción por el remitente y garantizan que este sea quien la ha realizado.

En lo que respecta a la eliminación de *Smart Contracts*, se ha de tener en cuenta que los contratos en la red son inmutables. Sin embargo, existe una única forma de borrar un contrato de la red. Para ello, se debe haber incorporado una función *selfdestruct()* al contrato. Lo correcto es introducir esta función en otra función que solo pueda ser llamada por el creador del contrato. La función *selfdestruct()* acepta como argumentos una dirección a la que enviara todo el Ether que tiene esta cuenta antes de quedar inutilizable.

2.2.1.2 Oráculos. Interacción con el exterior de la red.

Los oráculos son empleados para consultar información que se encuentra fuera de la red, a través de fuentes consideradas como fiables. Por ejemplo, si el contrato necesita saber el valor en bolsa de un índice financiero en un determinado momento, este debe consultar fuera de la red Ethereum. Esto se consigue a través de una función *oraclize_query()* que invoca una función *callback()* ya que la búsqueda puede tomar un tiempo. A continuación, se muestra este proceso con un diagrama.

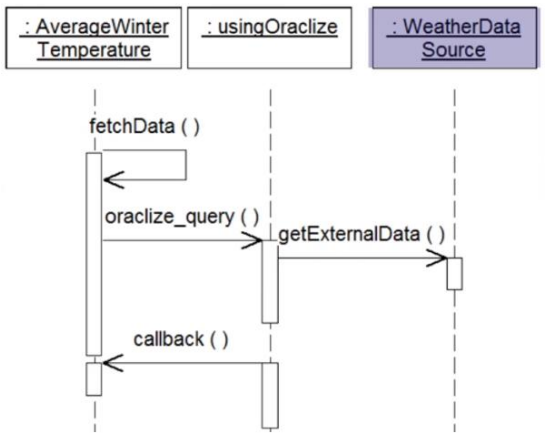

Figura 12. Diagrama simplificado de la consulta con un oráculo

2.2.2 SOLIDITY

La sintaxis de solidity se basa en JavaScript, haciendo uso de algunos conceptos de C++ y Python. El objetivo de este lenguaje es el de actuar contra la EVM [14].

En este apartado se analizará el lenguaje en cuestión, estudiándose los tipos de funciones, de datos y las variables globales, los eventos y los modificadores y la principal alternativa a este lenguaje, Viper. Únicamente se comentarán los tipos de datos y funciones consideradas distintivas, ya que se entiende que el resto son comunes con otros lenguajes más frecuentes como Java o C.

2.2.2.1 Tipos de funciones, de datos y variables globales.

Los principales tipos de funciones son los siguientes:

- **Función *Fallback***: Aquella función que se llama por defecto si se le envía Ether a un *Smart Contract* sin especificar una función a la que llamar.
- **Función *view***: Es una función que promete no cambiar el valor del almacenamiento de variables en Blockchain. Estas funciones se pueden llamar en local ya que lo único que hacen es leer cierto parámetro de la red Ethereum sin modificar ni crear ningún valor.
- **Función *pure***: Este tipo de función, además de no escribir nada en el almacenamiento de Ethereum, ni siquiera lee ninguna información. Es por tanto que la pure función es un caso particular de la View función.
- **Función privada/publica**: La función privada es accesible únicamente desde dentro del contrato mientras que la publica puede ser accedida desde el exterior. Por defecto, las funciones son privadas.
- **Función interna/externa**: Una función interna solo puede accederse desde el contrato o desde los contratos que hereden de este. Una función externa permite el acceso desde el exterior.
- **Función pagable**: Es una función que tiene la cualidad de poder recibir Ether.

- **Función constante**: Una función se denomina como constante cuando tiene la cualidad de ser llamada sin gastar Gas. Para ello no debe de alterar el estado de la red Ethereum, por lo que ha de ser *View* o *Pure*.
- **Función *require, revert y assert***: Las tres sirven para captar excepciones, y se emplean en lugar de la función *throw*, que está actualmente deprecada. Sin embargo, existen sutiles diferencias entre las tres [15]. Serán comentadas a continuación:
    - Require: Si se llama compila a la dirección 0xfd. Deshace todos los cambios y devuelve el Gas restante. Se ha de utilizar para la validación, al comienzo de las funciones, de argumentos.
    - Revert: Devuelve el gas restante al usuario y permite mandar un mensaje con el error que se ha producido. Se ha de utilizar para resolver las mismas situaciones que require, pero que necesiten más lógica.
    - Assert: Si se llama, compila a la dirección 0xfe, deshaciendo todos los cambios hechos hasta el momento y, a diferencia de *require*, gastando todo el gas. Estas funciones se han de utilizar idealmente menos veces que *require*, y al final de las funciones o para errores en tiempo de ejecución.

Los tipos de datos más significativos son:

- **Address:** Guarda un valor de 20 bytes que espera ser una dirección de una cuenta de usuario o de contrato. Si se desea enviar Ether a esa cuenta ha de especificarse como *payable*.
- **Mapping:** Es un tipo de datos que permite, en función de una clave, obtener un valor. La forma en la que se guardan valores es la siguiente. Se introduce una clave que puede ser por ejemplo un *address*. Se produce el hash de esa clave a través de un algoritmo Sha3. El resultado será la dirección de memoria donde se guardará el dato. Es por ello que este tipo de dato no puede ser iterado de la misma forma que un array, ya que a menos que conozcas la clave, no hay forma en la que puedes obtener la dirección de memoria donde se encuentra este.

Finalmente hacer una breve mención a las variables globales [16]. Una variable global es accesible desde cualquier parte del programa. Un ejemplo de variable global muy utilizada es msg.sender (devuelve el *address* del remitente de la transacción), msg.value (devuelve el valor en Wei de la transacción), now (variable temporal cuyo valor es el timestamp del bloque en el que se resolverá la transacción), etc.

2.2.2.2 Eventos y modificadores

Los **eventos** sirven como registros de lo que va sucediendo a medida que avanza la vida del contrato. Son opcionales. Por ejemplo, cada vez que se llame a cierta función se produce un evento, que queda registrado en la red Ethereum. Permite mantener una idea de las ciertos hitos o acontecimientos que van sucediendo y cuya información puede ser relevante.

Existen 3 parámetros que permiten catalogar los eventos. Todo esto hace que los eventos sean mucho más eficientes y útiles que realizar un proceso de *polling* constante a la red Blockchain. Se pueden programar *listeners* para eventos específicos y actuar ante ellos en consecuencia.

Por otro lado, los **modificadores** sirven para alterar el comportamiento de funciones y se anexan a estas en su cabecera. Se ejecutan antes de que lo haga la función, lo cual puede ser muy útil para evitar malgastar gas. Los modificadores se crean con una serie de condiciones. Si la condición falla, la transacción puede ser revertida con la función *revert*, vista anteriormente. Nótese que para que una función sea de los tipos que se han mencionado anteriormente, se ha de incluir el modificador pertinente en la cabecera de dicha función. Por ejemplo, con una función view:

```
    function viewState() public view returns(string) {
        //read the contract storage
        return state;
    }
}
```

Figura 13. Ejemplo del modificador view en la cabecera de la función

Los modificadores pueden ser predeterminados como es el caso de *view* o pueden ser personalizados y definidos en el código.

2.2.2.3 Alternativa (Vyper).

Vyper es un lenguaje para el desarrollo de Smart Contracts con una comunidad mucho más pequeña. Sin embargo, sus innegables ventajas a la hora de escribir código sencillo y auditable no pueden ser ignoradas.

Este lenguaje permite escribir *Smart Contracts* sencillos, legibles y auditables. Actualmente este lenguaje está en estado de desarrollo, pero no se han de perder de vista las incontables ventajas que traerá una vez comience a tener más usuarios.

2.2.3 MEJORES PRÁCTICAS

Se deben tener en cuenta una serie de consideraciones, más allá del código, a la hora de escribir Smart Contracts. A continuación, se comentarán.

En primer lugar, se debe tratar siempre de **encajar los tipos** de datos en función de lo que se espera guardar en esa variable. Es decir, si una variable va a guardar valores con un máximo de 8 bits, no tiene sentido declarar esa variable como uint32, ya que se perdería mucho espacio y se estaría gastando Gas de manera innecesaria.

En segundo lugar, es importante mantener un **orden en las funciones** para mejorar la legibilidad. Esto se puede encontrar en la guía de estilos de la documentación oficial de Solidity [17]. El orden sería el siguiente:

1. Constructor: Es único a diferencia de otros lenguajes como Java.
2. Funciones *fallback*
3. Funciones externas
4. Funciones publicas
5. Funciones internas
6. Funciones privadas.

En tercer lugar, se ha de tener en cuenta la opción de utilizar **funciones hash** para aumentar la seguridad en el *Smart Contract*. De esta forma se puede limitar la visibilidad de las variables. Existen 3 funciones predefinidas en *Solidity* para realizar hashes:

1. Keccak256(“ “): Devuelve un valor de 32 bytes.
2. Sha256(“ “): Devuelve un valor de 32 bytes.
3. Ripemd160(“ “): Devuelve un valor de 20 bytes.

Los tres reciben como argumento aquello de lo que se desea sacar el hash.

Finalmente, es de extrema importancia saber cómo usar las 3 formas en las que se pueden **guardar variables**, para optimizar al máximo el gas utilizado. Son las siguientes:

1. Almacenamiento: Es la forma más cara de almacenamiento. Es persistente entre llamadas a funciones. El almacenamiento se verá reflejado en el Merkel Patricia Trie mencionado en apartados anteriores. Por lo general, guardar una nueva variable tiene un coste de 20000 Gas, mientras que modificar una variable ya existente cuesta 5000 gas. Téngase en cuenta que las variables *struct* y *array* serán guardadas, por defecto, en almacenamiento. Por lo que se debe estar atento al definir estas variables dentro de funciones.
2. Memoria: Guarda variables temporales. Mas barato de usar, pero es eliminada una vez que se termina la ejecución de la función, es decir, ese valor solo existe dentro de la función. El uso de la memoria por variable tiene un coste de entre 1 y 3 Gas, drásticamente inferior al uso del almacenamiento.

3. Pila: Prácticamente gratuita de usar. Guarda variables locales. Se necesita para las operaciones intermedias y tiene 1024 espacios. Sin embargo, si se queda sin espacio, entonces la transacción será terminada de manera abrupta. Por ello se suele recomendar, a menos que se tengan extensos conocimientos a bajo nivel de la pila, que el compilador sea quien la maneje como mejor considere.

2.3 APLICACIONES DESCENTRALIZADAS (DAPPS)

En este apartado, se tratará en detalle el concepto de aplicación descentralizada. Una aplicación descentralizada es que aquella que tiene total o parcialmente, el *back-end* en la red Blockchain.

En este tema se estudiará cual es la estructura que debe tener una Dapp, las APIs que proporciona Ethereum, la importancia de probar nuestras aplicaciones los modelos y estándares de las aplicaciones descentralizadas.

Finalmente se concluirá con un tema de debate muy presente en la comunidad; ¿Cuál debería ser el ratio ideal de descentralización en una aplicación descentralizada?

2.3.1 ESTRUCTURA DE UNA DAPP

Para comenzar el desarrollo de una Dapp, ha de estar muy claro la diferenciación entre cliente y servidor. La parte de cliente es aquella parte de una aplicación que corre en el dispositivo del usuario final. Por otro lado, la parte de servidor se refiere a aquellos procesos y servicios que corren en un servidor, fuera del dispositivo del usuario final. Ambas partes se comunican entre ellas para conseguir que la aplicación funcione.

Se introduce el concepto de servidor Blockchain. Es decir, que, en mayor o menor medida, la red Blockchain, es este caso Ethereum, actúe como un servidor, proporcionando información al cliente, que podrá interactuar con ella. Por tanto, el cliente, a través un nodo

propio o de Metamask (se explorará más adelante este puente entre la red Blockchain y el navegador) podrá obtener información de la red Blockchain. Se explicará brevemente el proceso y a continuación se mencionará cual es la forma en la que se debe dividir un proyecto para construir una Dapp.

2.3.1.1 Proceso de comunicación entre el navegador y Ethereum

En primer lugar, se supondrá que no se dispone de ningún puente de comunicación entre el navegador y la red Ethereum, como puede ser Metamask. Esto en la práctica no es cierto, ya que la vasta mayoría de usuarios hacen uso de estos puentes de comunicación. Sin embargo, para la mayor comprensión de la tecnología y con intención de entrar en detalles sobre cómo opera el funcionamiento de trasfondo, se procederá a suponer que no se dispone de este puente.

En primer lugar, se procede a la creacion de un nodo. Para ello, se ha de especificar la red a la que se ha de conectar el nodo a través del campo *Network ID*, además del bloque génesis, es decir, el primer bloque creado en esa red. Recuérdese que Ethereum tiene más de una red. Se tratará en más detalle esto en la sección referente a Metamask. Existe una implementación para nodos en el lenguaje de programación Go, llamado geth (Go-Ethereum). Esta implementación es la más utilizada y se encarga de interactuar con los datos no procesados que vienen de la red Ethereum. Una vez implementado un nodo, es posible interactuar con otros nodos en la red.

A continuación, se muestra una imagen que explica la interacción de los nodos entre sí. En la CLI que proporciona geth se introducen una serie de comandos predefinidos para la comunicación entre nodos. Una instancia de geth podrá interactuar con otra, por ejemplo, a través de la función web3.eth.sendTransaction, que se muestra a continuación.

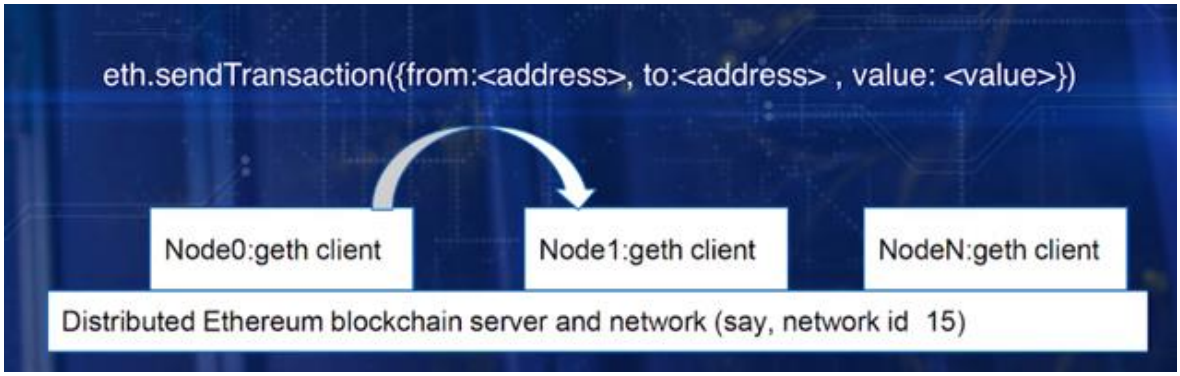

Figura 14. Comunicación entre nodos geth.

El nodo geth, interactúa con otros clientes a través de *listeners* mediante el protocolo TCP (Transport Control Potocol) y descubren otros nodos a través de UDP (User Datagram Protocol). Ambos protocolos son de nivel de transporte (según el modelo OSI) y utilizan el puerto 30303 [18].Dependiendo de donde se encuentre el nodo geth, existen dos protocolos de comunicación entre cualquier proceso (por ejemplo, web3) y el nodo geth.

Si el nodo geth corre en la misma maquina desde la que se desea acceder a través de web3 a la red Ethereum, entonces de utiliza el protocolo IPC (Inter-process Communication). Geth creara una vía IPC para comunicarse con el proceso en cuestión (mostrada en el fichero $HOME/.ethereum/geth.ipc) [19].Esto crea un conducto bidireccional de conexión. También se puede emplear RPC (Remote Procedure Calls) para comunicarse dentro de la misma maquina a través del puerto TCP 8545. Al ser un protocolo remoto se ha de establecer una dirección localhost:8545, solo accesible desde el mismo cliente.

Por otro lado, si se desea conectar a un nodo geth que este en un cliente remoto, se ha de utilizar necesariamente RPC, indicando la IP en la que existirá el punto final (endpoint) RPC, y el puerto. Por ejemplo, se puede indicar en una red privada 192.168.1.123:8545. Se ha de tener especial cuidado ya que según la política de trafico de tu Router, es posible que cualquier usuario de internet pueda acceder a tu punto final RPC como se explica en el siguiente post [20]. En esencia, si se conecta a través de RPC a otra máquina, este protocolo da la impresión de que existe una conexión directa entre el proceso y el nodo.

Existe una librería en JavaScript, llamada web3 que se encarga de encapsular las funcionalidades principales del protocolo JSON/RPC (peticiones RPC codificadas en JSON) y las hace mucho más accesible al usuario medio, bajando de esta forma las barreras de entrada para los desarrolladores.

El uso de la línea de comandos puede ser tedioso para interactuar con la red Ethereum. Para ello, se puede utilizar una aplicación, que actuara de la siguiente manera: La aplicación, a través de la librería web3.js, enviara peticiones al nodo. Esta librería, transformara dichas peticiones del usuario en peticiones JSON/RPC. El nodo geth escucha por defecto peticiones en el puerto TCP 8545, por lo que web3 las enviara allí. Tras esto el nodo geth será el encargado de interactuar con la red Ethereum. A continuación, se muestra un diagrama de funcionamiento.

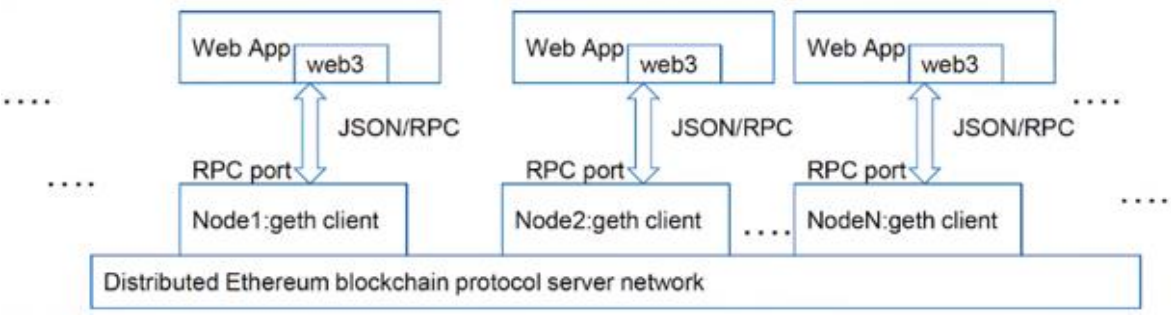

Figura 15. Esquema de funcionamiento completo de comunicación con el servidor Blockchain

2.3.1.2 Estructura para desarrollar una Dapp.

Para realizar una Dapp, se debe seguir la siguiente estructura (o similar) a la hora de ordenar los ficheros y el código.

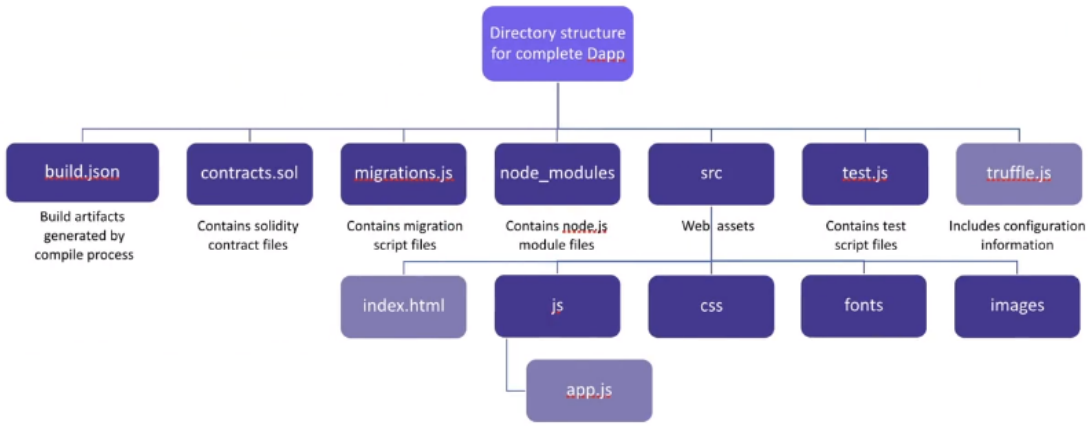

Figura 16. Ficheros y carpetas en una Dapp.

Se comentarán únicamente las carpetas distintivas sobre otras aplicaciones web tradicionales. La carpeta de contratos guardara los contratos escritos en solidity. La carpeta "build" almacena los contratos una vez han sido compilados. Estos están guardados en ficheros JSON con información importante de la que se hablara más adelante como el bytecode, el ABI… La carpeta de migraciones guarda los archivos que permiten migrar los contratos a la Blockchain. Estos ficheros se enumeran, indicando de esta forma el orden de migración en la red Blockchain. Finalmente mencionar el fichero truffle.js que contiene información del entorno de desarrollo Truffle (se expandirá sobre esto en la explicación sobre la tecnología) y la carpeta de test, que permite escribir pruebas en JavaScript para asegurar el correcto funcionamiento de los contratos antes de migrarlos.

2.3.2 ETHEREUM APIS

Existen dos categorías que provee geth:

1. APIs de administración: Estas APIs vienen integradas con la ventana de comandos de geth.
    - Admin: Otorga un mejor control y acceso a la instancia geth.
    - Debug: Permite la inspección y la depuración durante la ejecución de geth
    - Miner: Permite controlar las operaciones de minería

- Personal: Permite controlar las claves privadas
- Txpool: Dota al usuario de métodos para inspeccionar la pool de transacciones.

Los métodos de cada API se pueden encontrar en la documentación [21].

2. Web3: Se explicará más de la librería web3.js en el apartado de tecnologías implementadas.

2.3.3 IMPORTANCIA DE PROBAR LOS SMART CONTRACTS

En el año 1994, el profesor Thomas Nicely de la Universidad de Lynchburg, encontró un error en la unidad de hardware Intel Pentium. Esto llevo a Intel a tener que sustituir todos los microprocesadores con este defecto, lo que represento un tremendo coste. Este ejemplo es muy relevante en el desarrollo de DApps.

A efectos prácticos, un *Smart Contract*, al ser inmutable, no acepta cambios una vez desplegado, a menos que se hayan creado de antemano las previsiones para los mismos. Es por ello, que se puede crear una analogía valida entre un *Smart Contract* y una pieza de hardware desarrollada por una compañía. Una vez sale al mercado, cualquier cambio es extremadamente difícil.

Por esto, es muy importante hacer las pruebas necesarias para garantizar, antes de desplegar el contrato en la red, que funciona correctamente. Existe dependencias de desarrollo (devdependancies), es decir, dependencias únicamente necesarias durante el desarrollo de la aplicación y no durante la ejecución, como chai y mocha que permite el desarrollo de pruebas. Si bien es cierto que no se utilizaran estas dependencias para este proyecto, debido a que no se va a desplegar en la red principal, han de tenerse en cuenta para un proyecto a mayor escala que se desee realizar en la red principal de Ethereum.

2.3.4 OTROS CONCEPTOS EN LAS DAPPS

En este apartado se comentarán las diferencias entre EIP (Ethereum Improvement Proposals) y ERC (Ethereum Request for Comments), las distinciones entre ITO (Initial Token Offering) e ICO (Initial Coin Offering) y se tratara brevemente la razón por la que Ethereum tuve que realizar un *Hard Fork* respecto de Ethereum Classic.

2.3.4.1 ERC y EIP

En primer lugar, se comienza explicando la idea detrás de EIP. Las EIP sirven para que la comunidad Ethereum tenga la posibilidad de participar en el proceso de describir nuevas funcionalidades, proponer cambios o simplemente informarse de los mismos. Es un documento de carácter técnico. Por otro lado, la ERC es un documento focalizado a nivel de aplicación, que trata temas como los estándares necesarios en un Smart Contract o en los tokens.

Un token es similar a una moneda, pero su demanda tiende a estar asociada con un activo. Utilizan la tecnología de la red a la que pertenecen para actuar de forma similar a la que lo hace la moneda nativa, el Ether. El valor de los tokens varía en función de la oferta y la demanda que hay por los mismos. Un token se crea a través de un *Smart Contract*, que tras recibir cierta cantidad de Ether, los produce y los manda a las distintas cuentas.

Como ya se ha visto anteriormente, cualquiera con los fondos necesarios puede desplegar un *Smart Contract* en la red, y por tanto crear tokens, sin necesidad de atenerse a ningún estándar. Sin embargo, si se desea desarrollar una Dapp profesional, atenerse a estos estándares recogidos en la ERC dota de confianza a la aplicación de cara a los clientes, además de que permite a las billeteras añadir nuevos tokens, que utilicen este estándar, rápidamente. Los principales estándares son ERC20 y ERC721.

Si se busca en EtherScan los tokens que soportan ERC20, se puede observar que el número se encuentra cercano a los doscientos mil [22].

2.3.4.2 Diferencias entre ICO e ITO

Ambos términos son utilizados de manera análoga para referirse a los tokens que se utilizan en startups para, a través de la tecnología de Blockchain, financiar sus operaciones. Son de extremado riesgo, pero sus potenciales beneficios tienden a ser muy elevados. En esencia, una ICO se diferencia de una ITO en cuanto a que la ICO tiene su propia red Blockchain, y la ITO se establece sobre una Blockchain existente.

Especialmente en Ethereum, donde se despliega la mayoría de las ITOs, se tiende a usar de manera errónea el termino ICO para referirse a ITO.

2.3.4.3 Ataque DAO

DAO (Decentralized Autonomous Organization) fue un *smart contract* que tenía la intención de ser una entidad de capital riesgo descentralizada. La idea tuvo muchísimo éxito consiguiendo millones de dólares en Ether. La inversión recibida constituía un 15% de todo el Ether en circulación. Sin embargo, este contrato tenía una opción que fue utilizada como puerta trasera par un hackeo a gran escala.

En el contrato existía una provisión que permitía a los usuarios abandonar una inversión que previamente habían tomado, lo que les permitiría recuperar el Ether invertido en DAO, tras un periodo de espera de 28 días. Sin embargo, esto se utilizó por parte de múltiples hackers como una puerta trasera para extraer millones de dólares en Ether de este contrato.

Los desarrolladores y la comunidad tenían 28 días para decidir cómo proceder. Por un lado, una amplia mayoría de la comunidad propuso un *Hard Fork* para que los hackers no pudiesen sacar su dinero (es decir, si la mayoría de la cadena está de acuerdo invalidad una operación que ya ha sucedido lo único que deben hacer es proseguir con su minado desde el bloque anterior a aquel en el que se produjo dicha operación). Esta propuesta obtuvo fue tomada, por aquellos más afines a la filosofía de Blockchain sobre la inmutabilidad, de manera reacia.

Aquellos que se opusieron se mantuvieron en la cadena de bloques antigua, que paso a llamarse Ethereum Classic, mientras que la nueva, Ethereum, está apoyada por los pesos pesados de la comunidad.

2.3.5 RATIO DE DESCENTRALIZACIÓN IDEAL

Desde que aparecieron las primeras aplicaciones descentralizadas, ha existido debate sobre qué porcentaje de estas deben serlo. Es decir, ¿hay lugar para un servidor centralizado y una base de datos centralizada en una Dapp? La respuesta es que sí. La red Blockchain únicamente debería ser usada para guardar datos que se desea que sean inmutables, ya que el coste de guardar los mismos es muy alto. Este artículo de médium [23] explica que el porcentaje correcto de información descentralizada en una Dapp ronda en torno al 1%, y que muchas nuevas aplicaciones descentralizadas cometen el error de descentralizar muchos más datos. Esto resulta un problema no solo por el coste sino también por la accesibilidad del usuario medio a dicha información.

En el trabajo propuesto, se estudiará cual es la información que se debe mantener en la red Ethereum y cual se puede guardar a través de un servidor centralizado en una base de datos tradicional, además de las interacciones entre ambas bases de datos (la tradicional y la base de datos que represente la red Ethereum).

Capítulo 3. DESCRIPCIÓN DE LAS TECNOLOGÍAS

El desarrollo del prototipo practico de este trabajo de Fin de grado se ha desarrollado con la ayuda de varios programas informáticos, librerías y lenguajes que se mencionaran a continuación. Se ha optado por omitir el uso de lenguajes que no son específicos de este trabajo como HTML, CSS o JavaScript, mientras que se tratan en más detalle el resto.

3.1 REMIX IDE

Remix es un entorno de desarrollo web que permite el desarrollo y despliegue de *Smart Contracts*. Este entorno tiene 3 modos de funcionamiento a la hora de desplegar un contrato ya compilado [24]:

1. **Máquina virtual de JavaScript**: Esta opción permite comprobar el funcionamiento de un *Smart Contract* a través de una máquina virtual JavaScript que simula el funcionamiento de la máquina virtual de Ethereum (EVM). Se recomienda su uso para pruebas simples ya que no permite a nadie más que el propio usuario interactuar con este contrato, aunque este a través del uso de varias cuentas puede simular otros usuarios. Es el único modo que no requiere de software externo más allá del navegador web.
2. **Web3 inyectado**: Tratara de buscar un proveedor de Web3 dentro del navegador (como puede ser Metamask). Tras esto se realizarán las configuraciones necesarias desde Metamask (para elegir la red a la que conectarse…).
3. **Proveedor de Web3**: Este modo se conecta directamente a un nodo Ethereum a través del protocolo HTTP. Se debe indicar la dirección RPC a la que conectarse. También permite la opción de conectarse a un nodo propio, o a un cliente como Ganache.

En este trabajo, solo se ha empleado el funcionamiento Javascript VM, ya que el despliegue se ha realizado en el Cliente Ganache directamente a través de Truffle, el entorno de desarrollo.

El proceso de compilación de Remix produce los siguientes campos, que se comentaran a continuación:

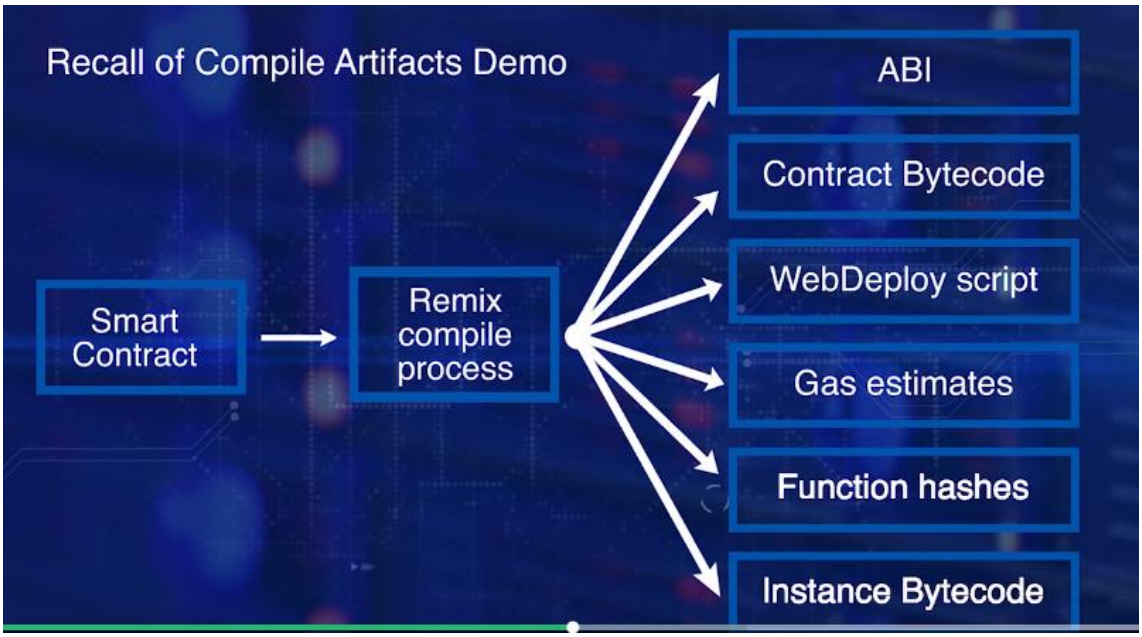

Figura 17. Principales campos creados por remix al compilar un contrato

- **ABI (Aplication Binary Interface):** Consiste en una lista de funciones y argumentos en formato JSON, que sirve de interfaz entre el código maquina (es decir el Ethereum Bytecode que comprende la EVM) y un usuario o contrato que desee llamar a una función o interactuar con el *Smart Contract*. El código utilizado por la EVM no puede ser utilizado directamente ya que no es comprensible para un usuario, y como se ha explicado en apartados anteriores, no se puede obtener mediante ingeniería inversa el código original. Por tanto, se puede entender la ABI como un

manual para poder operar de manera comprensiva con el contrato, una vez este ha sido desplegado a la red.

- **El *bytecode* del contrato:** Es el código, a nivel de ensamblador, que lee la EVM, a la hora de desplegar el contrato. Se obtiene a partir del código a alto nivel en Solidity o cualquier otro lenguaje que permita escribir *Smart Contracts*.
- **Scripts de despliegue de web3:** Es un código que permite desplegar un contrato programáticamente si se copia en una consola web3.
- **Estimaciones de Gas:** Estima el coste en gas que tendrá cada una de las funciones creadas en un contrato.
- **Funciones hash:** Se obtiene el hash de cada una de las funciones creadas, y se toman los 4 primeros bytes para denotar a cada función. Se entiende que la probabilidad de colisión tomados los 4 primeros bytes es prácticamente nula.
- **La instancia del Bytecode:** Es el bytecode que queda de una instancia en particular de ese contrato. Es decir, el bytecode del contrato es el necesario para desplegarlo, y el de instancia es el de un contrato desplegado.

3.2 *TRUFFLE IDE Y VISUAL STUDIO CODE*

Remix es un entorno de desarrollo que se utiliza para para la prueba y despliegue de *Smart Contracts*, mientras que **Truffle IDE** se utiliza para ensamblar varios componentes necesarios para la producción de una Dapp. Tiene numerosas funcionalidades que facilitan mucho el desarrollo de DApps, como la capacidad de hacer pruebas con mocha y chai, en vez de tener que una y otra vez introducir los datos que se desean probar en remix. Para comenzar a usar Truffle se inserta el comando “Truffle init”, que desplegara en la carpeta en la que se realice la estructura de ficheros para el desarrollo de una Dapp.

Sin embargo, Truffle no provee de elementos de desarrollo como subrayado de errores. Esto se puede conseguir a través de un editor de texto. El más empleado para combinar con Truffle IDE es **Visual Studio Code**. Una vez iniciado VS code, se instala la extensión de corrección

sintáctica en solidity, desarrollada por Juan Blanco [25]. Tras esto, haciendo uso de la línea de comandos que proporciona VS code se puede de manera muy sencilla combinar con Truffle y comenzar el desarrollo de la Dapp.

3.3 WEB3.JS

La librería web3.js ha sido mencionada varias veces a lo largo de este trabajo. Se tratará de explicar con más detalle su funcionamiento.

Web3.js consiste en una colección de librerías para actuar con un nodo local o remoto Ethereum. Como se ha mencionado ya, se puede utilizar una conexión IPC, para conectarse a un nodo local, o a través de HTTP o web sockets a uno remoto [26]. La actuación contra un nodo es suficiente para que se pueda conocer el estado de la red Ethereum o para realizar transacciones (se harían a través del nodo).

También ha sido mencionado que detrás de todo, web3 emplea llamadas RPC para interactuar con el nodo. Esto facilita muchísimo al desarrollador y al usuario interactuar con Ethereum. Para ser más precisos, si se desarrolla un cliente o una aplicación web, la acción de obtener y escribir datos en la red se hará a través de web3.js. Entre otras acciones, posibilita mandar Ether, leer/escribir datos de *Smart Contracts*, crear estos contratos, etc... Todo de manera muy sencilla y sin una necesidad de comprensión profunda de cómo opera la tecnología por detrás. El análogo a web3.js en desarrollo web seria JQuery trabajando contra un servidor web. La siguiente imagen representa adecuadamente el funcionamiento de web3.js.

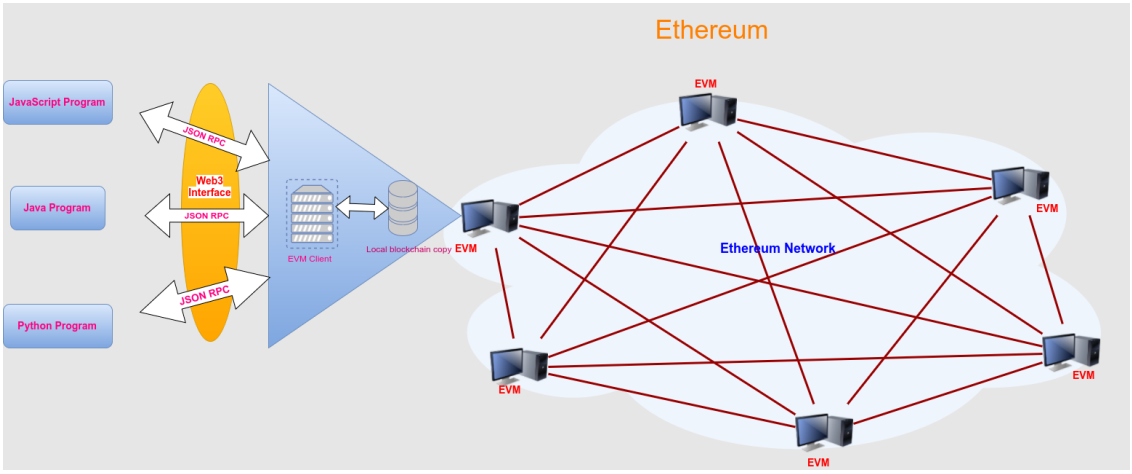

Figura 18. Esquema de uso entre nodos de Web3.js

La primera pregunta que surge es si se debe conectar a un nodo local o a uno remoto. Para ello se ha de tener en cuenta que, para conectarse a un nodo local, en primer lugar, se necesita correr un nodo local. Esto implica la descarga de una enorme cantidad de información y mantenerla actualizada, lo cual, a menos que se tenga un buen motivo para ello (como practicar minería), se debe evitar.

La alternativa es mucho mas atractiva; conectarse a un nodo remoto. Esto se puede conseguir a través de un gestor de nodos de Ethereum como puede ser *Infura*. Este gestor permite de manera sencilla y gratuita la conexión a un nodo externo. Se obtiene una "URL RPC" y se conecta a dicha dirección. Para realizar esta conexión, una vez descargada la librería web3, se debe instanciar un objeto web3, en el cual se debe indicar la URL mencionada anteriormente.

Se debe mencionar que este proceso se utiliza para conectarse a la red principal. Si se desea conectar a Ganache por ejemplo (una red de prueba que corre en local), se deberá introducir una URL localhost, e indicar a Ganache esa misa dirección. Tras esto ya se puede actuar contra la red Ethereum.

3.4 METAMASK

Metamask actúa como un puente entre el navegador y la red Ethereum. Permite, sin necesidad de correr un nodo Ethereum propio, hacer uso de DApps. Dado que la mayoría de los usuario no tienen un nodo propio, esta herramienta es imprescindible para la mayoría de los usuarios que deseen interactuar con Aplicaciones descentralizadas.

A día de hoy, existen navegadores que tienen esta opción de interactuar, a través de un objeto web3, con la red Ethereum, integrada. Un ejemplo puede ser *Reborn 3* de *Opera* [27]. Sin embargo, la mayoría de los navegadores incluyendo los mas utilizados como Google Chrome, Safari y Firefox deben utilizar una extensión. Esta extensión es Metamask.

La principal ventaja de Metamask es que, aunque no es la extensión mas intuitiva de usar para la mayoría de los usuarios sin conocimientos de Blockchain, guarda las claves privadas y publicas en el navegador, siendo de esta manera fiel al principio de descentralización de Blockchain.

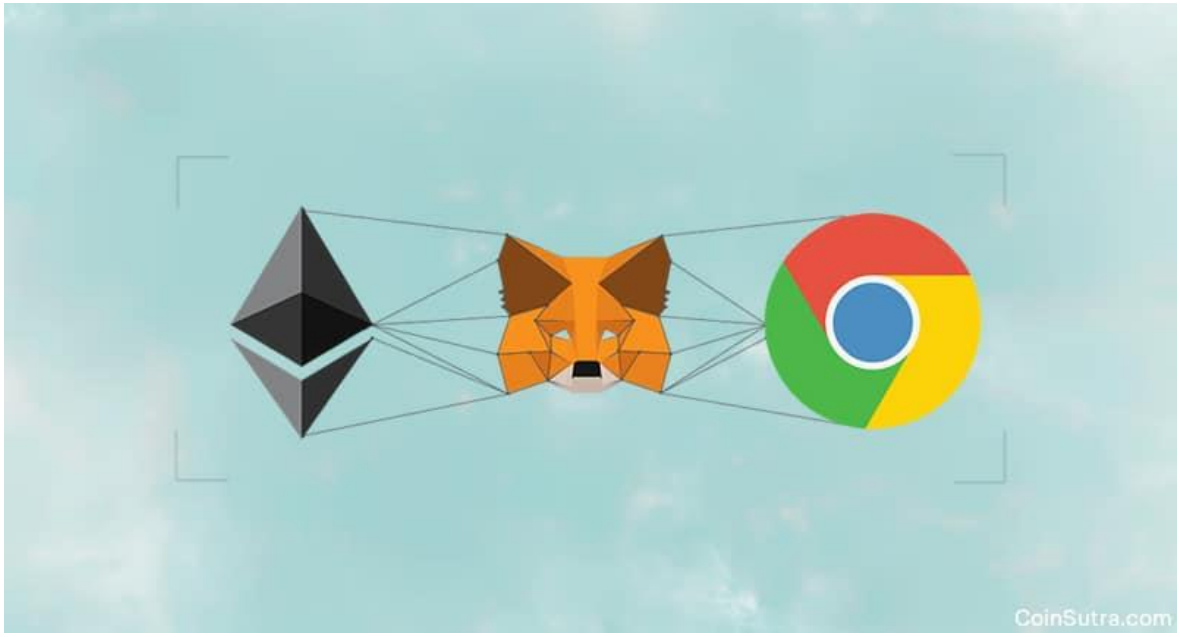

Figura 19. Metamask actúa como un puente entre el navegador y la red Ethereum

3.5 NODE.JS

Si bien es cierto que esta tecnología no es exclusiva del desarrollo de DApps, este lenguaje de servidor es muy útil para aplicaciones en tiempo real. Node.js utiliza el lenguaje JavaScript, normalmente reservado para la integración de la lógica a nivel de cliente, para el desarrollo del servidor. La forma en la que consigue esto es la siguiente:

En primer lugar, hace uso de motor V8 de Chrome. Este motor no es mas que un programa capaz de transformar código JavaScript a código máquina, es decir, el código a bajo nivel que interactúa con los microprocesadores de nuestros dispositivos. Otros motores como Rhino no son tan rápidos como este, ya que traducen el código JavaScript a un lenguaje intermedio (bytecode) y luego a código máquina. Esto hace que el motor V8 de Chrome sea el mas rápido. Este motor puede ser utilizado de manera independiente para escribir en JavaScript y que esto sea traducido a código máquina. Sin embargo, es posible integrar dicho motor con un programa de C++ propio que contenga una serie de definiciones que dicho motor de forma independiente no tendría.

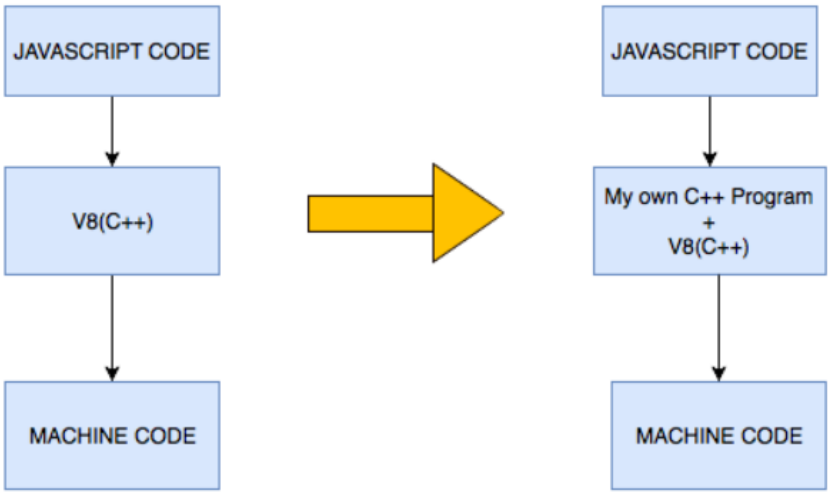

Figura 20. Motor V8 de Chrome incorporando un programa en C++

El motor V8 es de código abierto, por lo que fue utilizado para desarrollar node.js. Node.js simplemente coge este motor e implementa un código C++, destinándolo a que funcione como un lenguaje de servidor.

La principal ventaja de emplear este lenguaje a nivel de servidor es que funciona por defecto de manera asíncrona, es decir, cuando recibe una petición a la que sabe que va a tardar en dar una respuesta, devuelve un *callback*, y de esta forma puede atender otras peticiones hasta que la información que necesita está disponible. Un *callback* es una función que le indica al cliente que su petición se esta tramitando, y que en cuanto esté disponible podrá recibirla. Por esto, Node.js es muy útil para aplicaciones en tiempo real, y nefasta a la hora de operar con aplicaciones que requieren un uso intensivo de la CPU [28].

La filosofía de node.js se basa en los módulos. Si se ha de utilizar una analogía, los módulos son similares a los paquetes en java. Se utilizan para organizar cierta funcionalidad y están contenidos en un único fichero. Es por tanto que cada módulo se debería encontrar en un fichero distinto.

La funcionalidad de un modulo puede ser usada en otro a través de la palabra reservada “require()” en el módulo en el que se desee utilizar. Esto esta supeditado a que, en el módulo que se quiere exportar, se haya especificado la opción de ser exportado, a través de la expresión “module.exports“ Tras esto se tendrá acceso a la funcionalidad del modulo que se requiere. La manera mas practica de ver que módulos se están utilizando en la aplicación es a través del fichero “package.json”. Este fichero contiene todos los módulos que se han importo hasta el momento. La importación es sencilla de realizar a través de línea de comandos.

El resto de las ideas relevantes de este lenguaje serán comentadas en los apartados de explicación del proyecto.

3.6 GANACHE

Ganache es una herramienta para desarrolladores de DApps que se utiliza para simular una red Ethereum de manera local. Por tanto, todo lo que se realice en esta red de prueba no tendrá ramificaciones en la red real, y permite probar de manera más verídica las aplicaciones y contratos, permitiendo las mismas acciones que permitiría la red real sin el coste que estas supondrían. Ganache tiene dos componentes:

- **Ganache CLI:** Viene incluida con Truffle, y permite ejecutar desde línea de comandos las instrucciones necesarias para correr la red de prueba.
- **Ganache GUI:** Es una interfaz visual de Ganache CLI, que permite al usuario observar de manera mas sencilla lo que esta sucediendo en la cadena de bloques privada. Una vez descargada la aplicación, se ha de enlazar con Ganache CLI. A continuación, se muestra como es la interfaz gráfica.

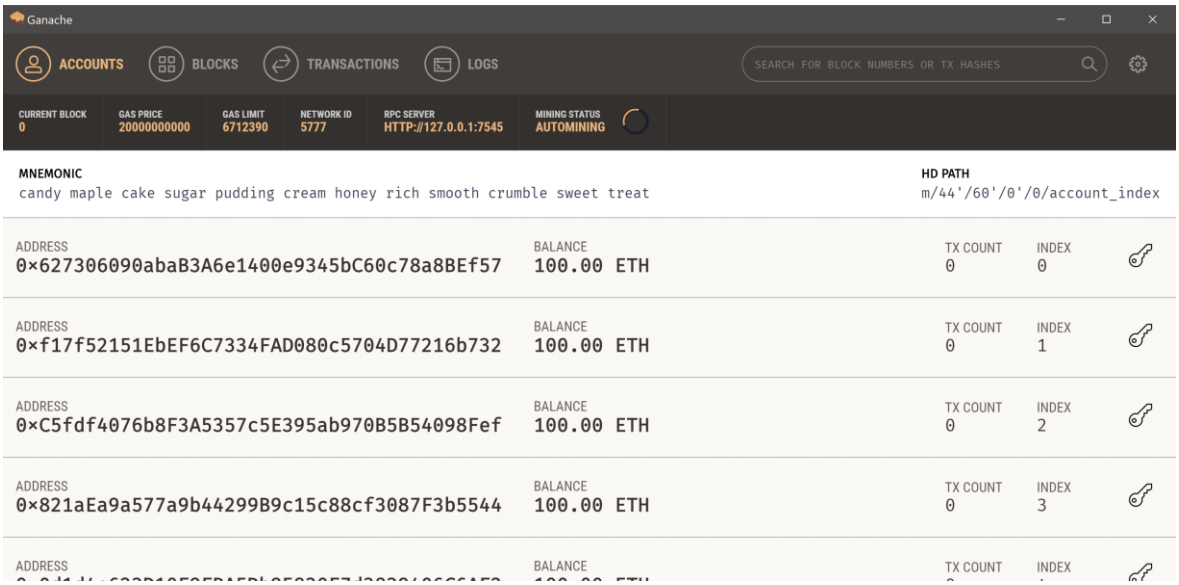

Figura 21. Ganache GUI

En los manuales sobre como se ha procedido con la instalación del entorno de desarrollo se tratará de expandir las funcionalidades de Ganache que se hayan empleado.

Capítulo 4. DEFINICIÓN DEL TRABAJO

4.1 JUSTIFICACIÓN

El motivo por el que se comienza este trabajo es por la falta de atención que se da al tema de la deuda circular. La premisa es que este problema no puede ser resuelto por una entidad centralizada, ya que implicaría que tuviese gran control de la información de un numero potencialmente elevadísimo de usuarios. Por otro lado, una entidad bancaria por sí sola, en principio, no podría ofrecer estos servicios ya que no tendría acceso a la información financiera requerida de otros clientes que tengan cuentas en otros bancos. Es por eso por lo que se opta por una solución descentralizada.

El incentivo principal que tendría un usuario que use dicha aplicación es el **ahorro de comisiones bancarias** que conllevarían tener que pagar una deuda de que podría haber sido cancelada si hubiese sido conocedor de la naturaleza circular de la deuda.

Existen una serie de posibilidades para desempeñar el proyecto en un entorno profesional. Se exponen a continuación:

1. El Smart Contract sea controlado y promovido por **el gobierno**, haciendo que las PYMES e incluso empresas de mayor tamaño la usen como una medida para acelerar la economía. Si se consiguiese demostrar que los resultados favorecen la actividad económica de una manera significativa esto podría llevar a su implementación rápidamente.
2. El Smart Contract está controlado por una **empresa privada**, que realiza y mantiene una aplicación para interactuar con este. Se podría añadir funcionalidad para incentivar el uso de la aplicación al comienzo, ya que la aplicación será más útil a medida que más usuarios la utilicen. Esta funcionalidad extra podría contener ideas más tradicionales como software de organización financiera, aplicaciones para administrar contratos y deudas… Además, se ha de posibilitar la opción de que, de

manera sencilla, el usuario pueda adquirir criptomonedas para poder interactuar con el Smart Contract. Existen dos maneras distintas de obtener beneficios para esta empresa.

- Un modelo de suscripción**.** El usuario paga para entrar en la lista de integrantes del Smart Contract, y se cobrara anualmente para pertenecer a esa lista. Este modelo no es el preferido ya que las barreras de entrada de pagar por software serian mayores y desincentivarían al usuario.
- Un modelo de comisión**.** El uso de la aplicación es totalmente gratuito. Sin embargo, se concreta una provisión en el Smart Contract que incluya comisiones a usuarios por cancelar su deuda. La comisión será un porcentaje en función de la deuda cancelada, y ha de ser determinado.

4.2 OBJETIVOS

Este trabajo de fin de carrera tiene como objetivo un estudio exhaustivo de la tecnología Blockchain y descubrir la posibilidad de que esta sea empleada para la cancelación de deuda circular mediante Smart Contracts. Se enfatiza la palabra "descubrir" ya que la idea del trabajo es propia, y postulada antes de estudiar la tecnología a fondo.

Tras el **estudio detallado de la tecnología**, reflejado en los folios anteriores, se puede afirmar con un alto nivel de certeza que la idea puede ser llevada a cabo mediante el uso de Smart Contracts en la red Ethereum. Además, se aprovechará el conocimiento adquirido para plantear posibles mejoras a la red Blockchain, en detrimento de que estas ya hayan sido consideradas y desestimadas en el pasado. Hay que mencionar que la estructura del conocimiento adquirido se debe en gran parte a los tres cursos de la universidad estatal de Nueva York, *University at Buffalo* tomados online e impartidos por la profesora Bina Ramamurthy.

Para comenzar, tras el estudio exhaustivo se desarrollará, en una máquina virtual, una **simulación de la aplicación**. Para ello, se hará uso de la red de prueba Ganache, y de las tecnologías mencionadas anteriormente. Se ha comprado una plantilla HTML para dotar a

la aplicación de una apariencia seria. Se pretende también reutilizar esta plantilla para futuros proyectos. Se desarrollará una página de autenticación y un inicio de sesión con intención de utilizar y explotar los conocimientos de node.js adquiridos (junto con una base de datos relacional mysql). Por otro lado, se ofrecerá una interfaz simple que muestre al usuario el proceso de la conexión con la red Blockchain. Cada uno de los pasos de la conexión estará acompañado por un mensaje en la interfaz gráfica para ir comprendiendo lo que sucede de fondo.

Finalmente se indagará acerca de las limitaciones que tiene el **Gas como variable programática en Solidity** y se **plantearan ideas para la posible mejora de la red Ethereum**.

4.3 METODOLOGÍA

La metodología de un proyecto se refiere al conjunto de procedimientos realizados para alcanzar un objetivo. Se considera que la metodología aplicada en este trabajo es una metodología proyectiva u holística. Este tipo de investigación consiste en proponer soluciones a una situación en base a un proceso de investigación. En este caso, se propone dar solución al problema de la deuda circular mediante la tecnología Blockchain a través de una profunda indagación sobre esta ciencia.

4.4 PLANIFICACIÓN

En primer lugar, se muestra el diagrama de Gantt que especifica a grandes rasgos la planificación que se ha llevado para este trabajo.

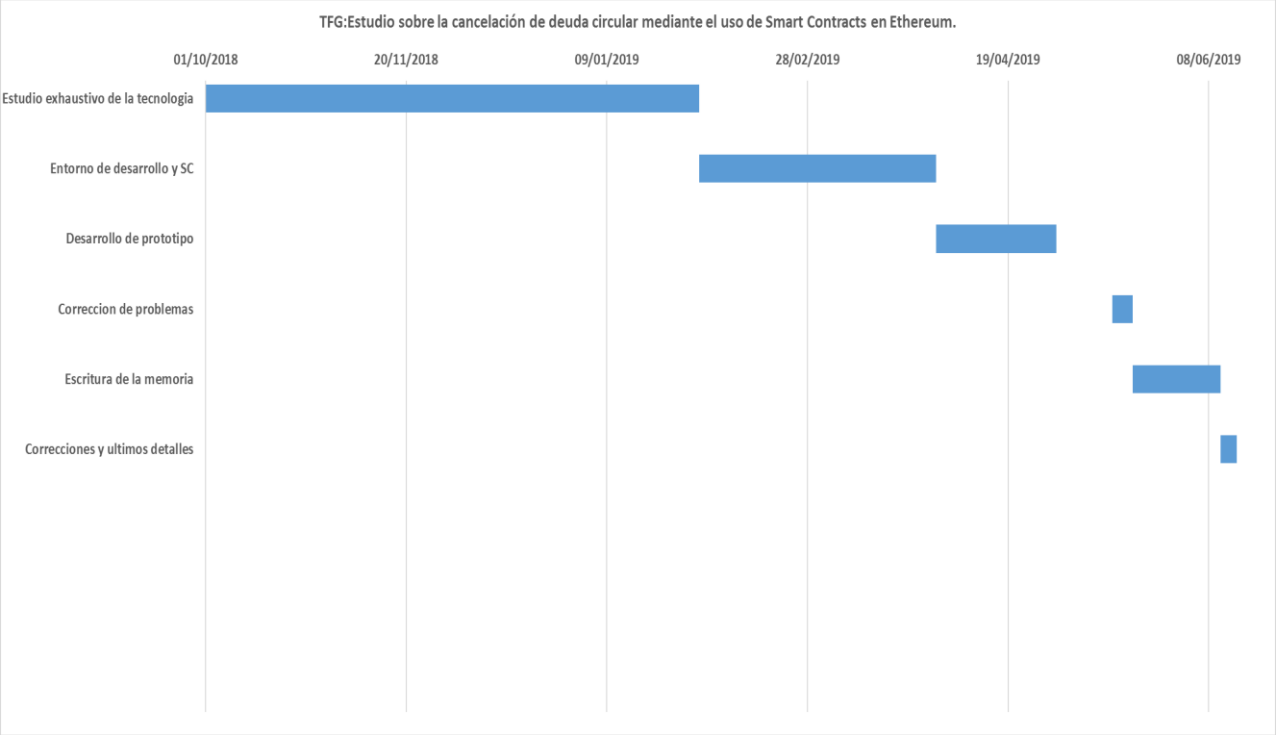

Figura 22. Diagrama de Gantt para el desarrollo del trabajo

Nótese que en etapa de mayor carga académica como puede ser la época de exámenes, no se tiene tanta actividad y por tanto puede llevar a que tareas que a priori no deberían llevar tanto tiempo se prorroguen hasta después de dichos periodos. Estos periodos comprenden el mes de diciembre y el mes de mayo.

4.5 POTENCIAL IMPLEMENTACIÓN ECONÓMICA

Para la discusión de este apartado se tomará la perspectiva de una iniciativa privada frente a la posibilidad de que sea un gobierno quien administre esta aplicación.

En primer lugar, se ha de estudiar si esta aplicación es o no necesaria, y la mejor forma, en caso de que lo sea, de implementarla a gran escala, ya que a mayor número de usuarios mayor beneficio podrán recibir los usuarios de esta.

La mejor manera de ver si esta aplicación tendría cabida en el mercado es a través de una entidad bancaria. Esta entidad guarda las cuentas de empresas que tienen su dinero en la

entidad. De esta forma se puede aplicar un algoritmo para ver qué cantidad de deuda es de carácter circular, fijándose en las transacciones realizadas por las empresas en un periodo de tiempo y el destinatario de estas, y extrapolar los beneficios que esto podría tener para los usuarios que, en vez de pedir a entidades bancarias dinero para poder hacer frente a sus deudas, cancelen parte de ella con la aplicación. De esta forma, la entidad bancaria puede usar esta aplicación como una forma de reclamo a empresas que tengan sus cuentas con otros bancos, señalando que estas empresas podrían tener una mejor gestión de su deuda e incluso un ahorro económico relevante si cambian de banco.

La alternativa es realizar esta aplicación de forma independiente, es decir, sin el respaldo de ninguna entidad bancaria. El principal problema con esta propuesta es la difícil implementación. Los usuarios no tendrían motivos para comenzar a introducir sus deudas si no hay otros usuarios con las que cancelarlas.

Además, se ha de atender a la confidencialidad. Una empresa puede ofrecer un precio por un servicio a una empresa y un precio muy distinto a otra diferente. Por ello se ha de buscar una manera de realizar estas transacciones de manera privada, pero a la vez confiable. Una alternativa es utilizar una Zero Knowledge Proof en Ethereum (Vitalik Buterin, fundador de Ethereum, ha sido un gran impulsor de esta idea [29]) o haciendo uso de una red permisionada como Hyperledger.

4.6 INFORMACIÓN ADICIONAL SOBRE LA APLICACIÓN

En este apartado se profundizará un poco en los ejemplos de uso de la aplicación y las ventajas que esta podría aportar a las empresas.

Pendiente de un estudio económico exhaustivo (podría constituir un TFG de Administración de Empresas) sobre el impacto económico que este tipo de deuda produce en una entidad económica a lo largo del año, se presupone que por lo menos podría incentivar a una empresa a considerar el uso de la aplicación.

Sin embargo, se entiende la posibilidad de que la cancelación de deuda no sea el motivo principal del uso de la aplicación, ya que los beneficios que esto otorgaría serian demasiado a largo plazo como para conseguir una temprana implementación con un alto número de usuarios. Por ello, se plantea que la aplicación pueda servir, en especial, a pequeñas y medias empresas (PYMES) a obtener cierto grado de confiabilidad a la hora de tratar con otras PYMES.

El problema con las PYMES es que es frecuente que no se conozca la reputación de esta. En esencia, si paga o no sus deudas. Por ello, se plantea un sistema de *Scoring*. El *Scoring* no es más que la palabra que se emplea para hablar de *rating* (valoración) en el entorno de las empresas. De esta manera, las PYMES que tengan la aplicación no requieren de dinero, sino que pueden usarla para pagar directamente con deuda. Este paradigma innovador se explica a continuación:

Supóngase que la aplicación gozase de una implementación prácticamente completa entre las PYMES. De esta manera, las transacciones entre ellas se realizarían a través de la aplicación. Por ejemplo, un despacho de abogados (se le llamará D) lleva un caso jurídico a una pequeña fábrica de muebles (se llamará F) por valor de 600 €. Esta fábrica le vende muebles a una empresa que se dedica a renovar mobiliario (se le llamará M) por valor de 500 €. Da la casualidad de que el despacho de abogados inicial requiere de una renovación de muebles en sus oficinas, y por ello contrata a esta empresa que se dedica a renovarlo por valor de 800 €. Se es consciente de que este ejemplo es una simple reducción hipotética y, aunque bien es cierto que es improbable que se dé una situación similar, se postula que a medida que los integrantes de una interacción económica aumenten tarde o temprano se llegara a un circulo de deuda. Se muestra en una simple figura la situación:

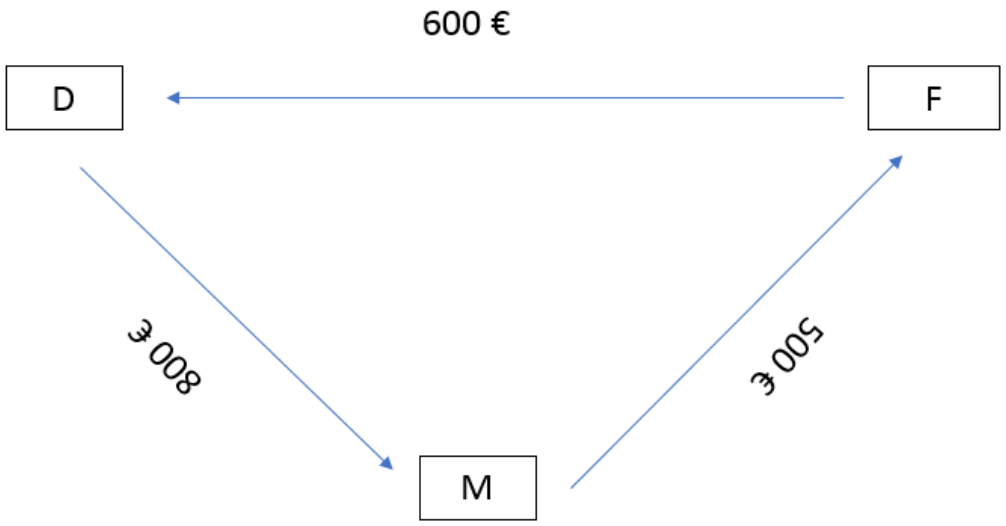

Figura 23. Ejemplo trivial de deuda circular.

En esta figura se muestra la siguiente idea. F le entrega a D un reconocimiento de deuda por valor de 600 €. Lo misma idea sucede en el resto de los casos. Hasta este punto todo igual que en una deuda registrada en una contabilidad de una empresa bajo el nombre de "deudores" o "deudores efectos comerciales a pagar". La principal diferencia, es que esto queda registrado en la Blockchain junto con el supuesto momento en el que, si la deuda no ha sido cancelada gracias a un bucle circular, se debe efectuar el pago. Se sigue insistiendo en la idea de que la deuda, con la existencia de un numero suficientemente alto de integrantes, debería poderse cancelar al menos de manera parcial.

Para extender el ejemplo, se tienen las siguientes hipótesis:

- F debe pagar a D el 10 de Julio de 2019. El reconocimiento de la deuda se produce el día 1.
- M debe pagar a F el 20 de Julio de 2019. El reconocimiento de la deuda se produce el día 10.
- D debe pagar a M el 30 de Julio de 2019. El reconocimiento de la deuda se produce el día 20.

Por tanto, a día 20 de Julio, el círculo de deuda es conocido, y se puede reducir según la siguiente figura.

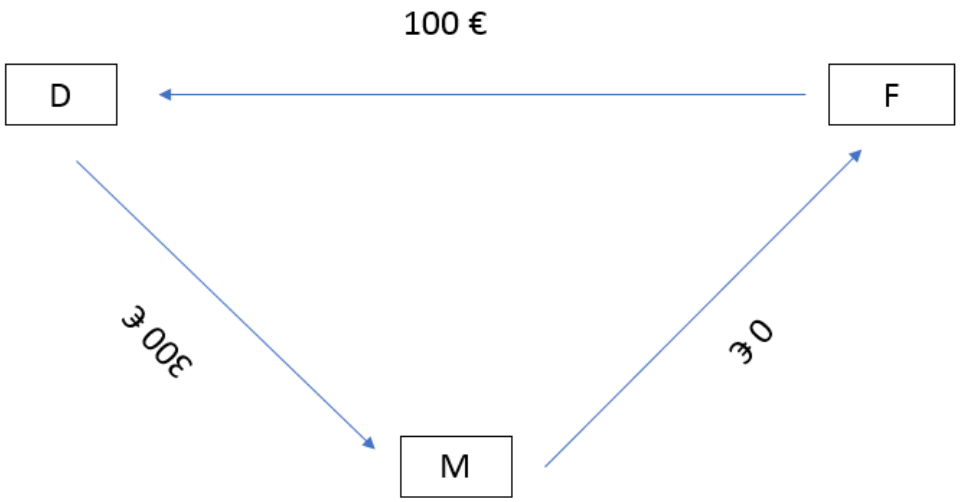

Figura 24. Simplificación de la deuda de la figura anterior.

Una vez se ha reducido la deuda, todos los integrantes salen beneficiados ante uno de los escenarios mas comunes que se explica a continuación. Imagínese que, debido a que D, a día 30 de Julio, no ha cobrado de F y, por consiguiente, no puede hacer frente a sus deudas con M. Por ello, al querer hacer frente al pago en fecha (comportamiento positivo), se ve obligado a pedir un préstamo bancario de 800 €. En aras del ejemplo, se estipula una comisión por parte del banco del 5%. Por ello, en un futuro, D debe pagar a su banco:

$$800 * 1.05 = 840 €$$

Supóngase la misma situación ahora haciendo uso de la aplicación. Esta sería capaz de reducir la cantidad que debe pagar D a M sustancialmente. D se puede seguir encontrando en la situación de necesitar pedir un préstamo bancario pero esta vez será de 300 €. Por tanto, aplicando la misma comisión que antes, D deberá devolver a su banco:

$$300 * 1.05 = 315 €$$

Según este ejemplo, esto constituye un ahorro por parte de D, gracias al uso de la aplicación, de:

$$100 * \left(1 - \frac{15}{40}\right) = 62'5\%$$

En este pequeño ejemplo, 25 euros (esta cifra viene de la resta de 40-15, que constituye la diferencia en comisiones pagadas sobre el principal) puede parecer una cantidad insignificante, pero si esto mismo se manejase con cifras mucho más elevadas, sería un ahorro significativo. De nuevo, la cifra a la que se ha de prestar atención es al porcentaje (65%), que nos da una idea del ahorro.

Finalmente, existirán usuarios que sean más responsables a la hora de pagar su deuda que otros. Este comportamiento ha de ser recompensado. Por ello se propone un sistema de *Scoring*. Este sistema permitiría a los usuarios tener mayor o menor calificación (por ejemplo, de un usuario de tipo A, que sería muy responsable a la hora de realizar sus pagos, a un usuario de tipo F, que sería muy poco responsable). Existen dos formas de pagar. Una seria a través de criptomonedas (ya que el Smart Contract es consciente de lo que se debe) o a través de medios tradicionales, en tal caso tendrá que dar fe de ello el usuario que es pagado. Ambas acciones son equivalentes y generarían un evento que de fe de que la deuda ha sido cerrada.

De esta forma, los usuarios a través de la aplicación (y la certeza a través de Blockchain de que esa información no ha sido adulterada de manera alguna), podrán elegir con quien hacer negocios y con quien, a riesgo de que sean irresponsables en el pago de sus deudas, no deberían. Permitiría además un sistema a través del cual cuantificar el riesgo. Si una empresa es muy responsable en el pago se le podrá cobrar un precio inferior a otra que se le podrá pedir mas dinero debido al riesgo que ello conlleva.

Por tanto, lo más importante es que el buen comportamiento y la responsabilidad en el pago de deudas serán recompensadas a través de un sistema totalmente confiable, ya que puede comprobarse a través de la Blockchain que la información ofrecida es inequívocamente veraz.

Capítulo 5. SOLIDITY Y EL GAS, UNA NUEVA VARIABLE PROGRAMÁTICA

A diferencia de otros lenguajes de programación, Solidity está diseñado para que el código sea desplegado en la EVM. Este código tiene un coste, no solo de despliegue sino también de uso posteriormente. Aunque ya se ha tratado el tema del lenguaje en apartados anteriores, en este se tendrá en cuenta únicamente los problemas del Gas. Siempre que se realiza una labor de programación, se trata de optimizar el código lo máximo posible, sobre todo si uno de los parámetros a tener en cuenta es el tiempo de ejecución. Esto en solidity es de mayor importancia aun, ya que no es únicamente el tiempo de ejecución que se ve afectado, sino también el precio de despliegue del contrato y de interacción con las funciones. Se resumen una serie de "elementos a tener en cuenta".

Para comenzar, se muestra el **coste de Gas por instrucción**. Esta obtenida del Yellow Paper de Ethereum.

Name	Value	Description*
G_{zero}	0	Nothing paid for operations of the set W_{zero}.
G_{base}	2	Amount of gas to pay for operations of the set W_{base}.
$G_{verylow}$	3	Amount of gas to pay for operations of the set $W_{verylow}$.
G_{low}	5	Amount of gas to pay for operations of the set W_{low}.
G_{mid}	8	Amount of gas to pay for operations of the set W_{mid}.
G_{high}	10	Amount of gas to pay for operations of the set W_{high}.
$G_{extcode}$	700	Amount of gas to pay for operations of the set $W_{extcode}$.
$G_{balance}$	400	Amount of gas to pay for a BALANCE operation.
G_{sload}	200	Paid for a SLOAD operation.
$G_{jumpdest}$	1	Paid for a JUMPDEST operation.
G_{sset}	20000	Paid for an SSTORE operation when the storage value is set to non-zero from zero.
G_{sreset}	5000	Paid for an SSTORE operation when the storage value's zeroness remains unchanged or is set to zero.
R_{sclear}	15000	Refund given (added into refund counter) when the storage value is set to zero from non-zero.
$R_{suicide}$	24000	Refund given (added into refund counter) for suiciding an account.
$G_{suicide}$	5000	Amount of gas to pay for a SUICIDE operation.
G_{create}	32000	Paid for a CREATE operation.
$G_{codedeposit}$	200	Paid per byte for a CREATE operation to succeed in placing code into state.
G_{call}	700	Paid for a CALL operation.
$G_{callvalue}$	9000	Paid for a non-zero value transfer as part of the CALL operation.
$G_{callstipend}$	2300	A stipend for the called contract subtracted from $G_{callvalue}$ for a non-zero value transfer.
$G_{newaccount}$	25000	Paid for a CALL or SUICIDE operation which creates an account.
G_{exp}	10	Partial payment for an EXP operation.
$G_{expbyte}$	10	Partial payment when multiplied by $\lceil \log_{256}(exponent) \rceil$ for the EXP operation.
G_{memory}	3	Paid for every additional word when expanding memory.
$G_{txcreate}$	32000	Paid by all contract-creating transactions after the *Homestead transition*.
$G_{txdatazero}$	4	Paid for every zero byte of data or code for a transaction.
$G_{txdatanonzero}$	68	Paid for every non-zero byte of data or code for a transaction.
$G_{transaction}$	21000	Paid for every transaction.
G_{log}	375	Partial payment for a LOG operation.
$G_{logdata}$	8	Paid for each byte in a LOG operation's data.
$G_{logtopic}$	375	Paid for each topic of a LOG operation.
G_{sha3}	30	Paid for each SHA3 operation.
$G_{sha3word}$	6	Paid for each word (rounded up) for input data to a SHA3 operation.
G_{copy}	3	Partial payment for *COPY operations, multiplied by words copied, rounded up.
$G_{blockhash}$	20	Payment for BLOCKHASH operation.

Figura 25. Precio de gas por instrucción

Siendo el subconjunto de instrucciones mencionado en la figura el siguiente

$W_{zero} = \{\text{STOP, RETURN, REVERT}\}$
$W_{base} = \{\text{ADDRESS, ORIGIN, CALLER, CALLVALUE, CALLDATASIZE, CODESIZE, GASPRICE, COINBASE, TIMESTAMP, NUMBER, DIFFICULTY, GASLIMIT, RETURNDATASIZE, POP, PC, MSIZE, GAS}\}$
$W_{verylow} = \{\text{ADD, SUB, NOT, LT, GT, SLT, SGT, EQ, ISZERO, AND, OR, XOR, BYTE, CALLDATALOAD, MLOAD, MSTORE, MSTORE8, PUSH*, DUP*, SWAP*}\}$
$W_{low} = \{\text{MUL, DIV, SDIV, MOD, SMOD, SIGNEXTEND}\}$
$W_{mid} = \{\text{ADDMOD, MULMOD, JUMP}\}$
$W_{high} = \{\text{JUMPI}\}$
$W_{extcode} = \{\text{EXTCODESIZE}\}$

Figura 26.Subconjunto de instrucciones en relación a la figura anterior.

Por tanto, una operación de suma tiene un coste de 3 mientras que una multiplicación un coste de 5.

Otro elemento a tener en cuenta es el concepto de **rembolso**. Nótese que siempre que se pueda vaciar un valor en el almacenamiento de un contrato, es decir, eliminarlo, o eliminar una cuenta de contrato mediante *selfdestruct()* se ha de hacer, ya que implica un rembolso de 15000 gas en caso de eliminar la variable y 24000 en caso de destruir el contrato. Sin embargo, existe un matiz relevante. El concepto de rembolso no implica que el minero te entregue a ti Gas por realizar la operación. Esto no tendría sentido ya que el incentivo del minero desaparecería. Lo que implica es que se devuelve un máximo de un 50% del coste de la transacción. Es decir, dado que el coste intrínseco en Gas de una transacción es 21000 y suponiendo que el resto de la transacción cuesta 79000 Gas (sumando un total de 100000), entonces cualquier tipo de rembolso no podrá exceder el 50% del valor de la transacción, es decir, 50000. Por tanto, una transacción que cuesta 100000 puede potencialmente quedar en un gasto de la mitad siempre y cuando se escriba el contrato con la intención de aprovechar este incentivo.

Además, se puede observar que el principal factor que consume Gas es el **almacenamiento**, ya que lo que pueda gastar una suma (3 Gas) frente a lo que cuesta un almacenamiento (20000 Gas) es muy significativo. Ya se ha diferenciado en apartados anteriores las distintas formas de guardar variables (almacenamiento, memoria y pila) y las ventajas y desventajas de estas.

Otro elemento a tener en cuenta es el **tamaño del contrato**, que afecta de la siguiente manera. A la hora de desplegar el contrato, el código en Solidity se transforma a Bytecode. La tendencia es, que cuanto mas largo sea el contrato en solidity, mayor será el bytecode. Teniendo en cuenta que el coste de despliegue de un contrato es de 200 Gas/bytecode, se ha de optimizar el código todo lo posible. Además, el máximo tamaño (en lo que se refiere al bytecode) de un contrato es de 24576 como se muestra en el fichero *protocol_params.go* [30] y el límite de tamaño en las transacciones es de 32KB como se muestra en el fichero *tx_pool.go* [31].

Siempre que sea posible, se ha de elegir el **tipo de dato** correctamente. Se clarifican los siguientes casos:

- El tamaño de la maquina Ethereum (EVM) es de 256 bits. Esto significa que el tamaño de cada hueco de almacenamiento es de 256 bits. Por tanto, definir una variable como uint128 implicara que de todas formas se use un espacio de 256 bits. Por lo que es interesante considerar la opción de, si existen dos variables de 128, en vez de definirlas por separado, hacerlo de manera conjunta una detrás de otra, y tener esto en cuenta en el código. A este procedimiento se le llama *packing* [32]. No solo eso, sino que el coste de cargar una variable de almacenamiento a memoria es de 200 Gas (por una palabra de 256 bits) por lo que si se guardan dos variables distintas habrá que hacer el doble de llamadas. El doble de llamadas y el doble de almacenamiento implicara el doble de gas en cada interacción con esas variables.
- Almacenamiento de tamaño fijo es más eficiente y más barato en lo que respecta al Gas. Por tanto, siempre que pueda especificarse el tamaño de un array o de un String se debe hacer.
- Guardar los datos en un mapping es mejor que hacerlo en un array excepto en casos en los que se requiera iteración. Aun así, se ha de fijar de antemano un tamaño máximo para dicho array para evitar ataques DOS (Denial Of Service), que se podrían dar si alguien llena un array tanto que el coste de iteración supere el máximo de gas permitido en la transacción.
- No inicializar variables antes de que sea necesario, ya que esto tendrá un coste de Gas por inicialización que se podría evitar. Se puede definir, y en un futuro ya se inicializará, si es necesaria.
- En los *require* comentados en secciones anteriores se pueden añadir Strings con el error en cuestión. Estas son preferibles que estén fijas a 32 bytes.

En ocasiones puede ser deseable guardar información en **eventos** en vez de almacenamiento con el motivo de ahorrar Gas. Los eventos como se ha mencionado anteriormente guardan su información en los bloques en los que se ejecuta la transacción que los acciono. Por ello, no siempre es recomendable usar esta técnica, ya que, aunque reduce la cantidad de gas necesaria, aumenta el tiempo de búsqueda de la información hasta puntos que quizás dejen de ser rentables y se conviertan en un problema.

Finalmente, hay que mencionar que Remix tiene una herramienta que permite observar una estimación del Gas que costara cada una de las funciones. Se ha de programar siempre tratando de minimizar estos valores, aplicando todas estas ideas.

Capítulo 6. FUTURO DE BLOCKCHAIN

En este apartado se tratarán ideas en lo que respecta al futuro y presente de la tecnología estudiada. Se explicarán las previsiones más prometedoras de la tecnología a través de la curva de Gartner, se indagará en el concepto de red permisionada, se explicará el concepto de IPFS y lo que esto implica y se finalizará tratando una serie de ideas propias sobre la tecnología.

6.1 CURVA DE GARTNER

Si bien es cierto que Blockchain como tecnología apareció en 2008 de la mano de Satoshi Nakamoto, a lo largo de los años ha ido introduciendo nuevas ideas a la tecnología. Sin ir más lejos, fue el 30 de Julio de 2015 cuando se introdujo el protocolo Ethereum, en el cual se basa en mayor medida este trabajo. Es por esto, que se estima que, aunque la idea principal que rodea a esta tecnología tiene más de diez años, las ideas que esta permite se renuevan y son totalmente actuales. A continuación, se mostrará la curva de Gartner de 2018. Esta curva, desarrollada por la consultora Gartner Inc., muestra los "ciclos de sobreexplotación" y por lo general la expectativa que tienen las tecnologías mostradas.

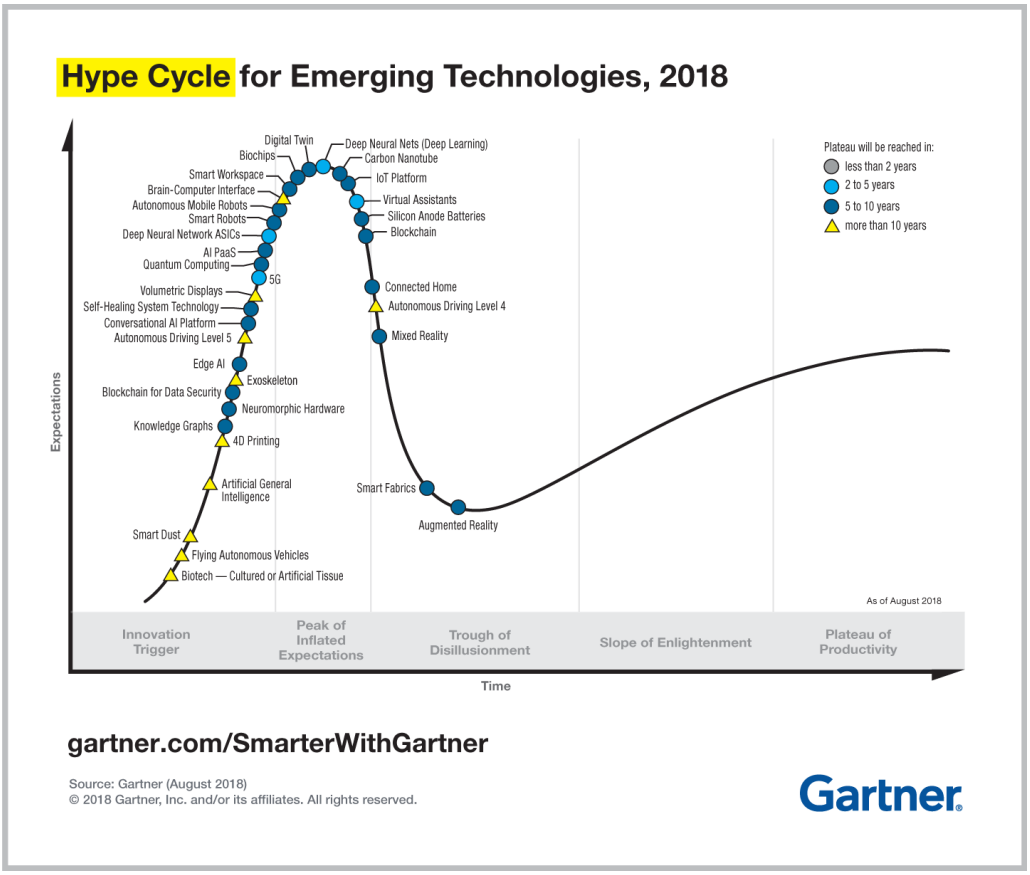

Figura 27. Ciclo de Gartner para nuevas tecnologías en 2018

Como se puede observar en la curva, la tecnología Blockchain como tal lleva entre 5 y 10 años existiendo (a día de hoy 11), y está en el rango de tecnologías con mayor expectativas, aunque esta esté disminuyendo. Sin embargo, a raíz de lo mencionado anteriormente, se destaca la solución de "Blockchain for data security". Es una nueva utilidad para la red Blockchain para guardar los datos más sensibles de una manera inmutable. Como se puede observar, nuevas ideas en relación con la tecnología están constantemente en desarrollo.

6.2 REDES PERMISIONADAS.

Esta sección no solo hace referencia al potencial futuro de Blockchain sino también la inmediato presente. Una Blockchain permisionada habilita la posibilidad de interacción

únicamente a nodos dotados del permiso para ello. Otro nombre que reciben es Blockchain de consorcio, ya que suelen usarse en sectores de manera diferenciada, como el sector del automóvil, el sector de salud…

El principal ejemplo de red permisionada que se va a tratar es Hyperledger de IBM. Hyperledger nace para dar solución a la privacidad en mercados donde existía la necesidad de proteger datos de alta sensibilidad. A diferencia de la mayoría de los protocolos Blockchain públicos que utilizan POW como mecanismo de consenso para crear bloques, Hyperledger utiliza pBFT (Practical Byzantine Fault Tolerance). La idea detrás de este algoritmo es llegar a un consenso teniendo en cuenta la posible intervención de agentes maliciosos. Para garantizar que pBFT funcione, el número de nodos maliciosos no debe superar un tercio de la red [33], situación que no se suele dar debido a las barreras de entrada que tienden a tener este tipo de redes.

Se enfatiza aun así que Hyperledger no tiene únicamente funcionalidad de Blockchain, sino que también proporciona ecosistemas y herramientas para negocios y partes interesadas en la empresa para colaborar. Sin embargo, se trata únicamente la funcionalidad que tiene relación con el trabajo.

6.3 IPFS

IPFS (Inter Planetary File System) propone la idea de un internet descentralizado para evitar, entre otras cosas, la censura. A día de hoy, la mayor parte de internet se encuentra centralizado. No hay manera de ver un video en YouTube si los servidores de esta empresa fallan o si la empresa bloquea contenido.

La principal diferencia en la manera de buscar contenido a través de un servidor centralizado o a través de un sistema de IPFS, es que para acceder a un servidor centralizado se utiliza una dirección en función de la localización del contenido. Con IPFS se especifica el contenido, a través del hash de este.

Se le pide a la red un fichero a través de su hash. Una vez que se recibe se comprueba que el fichero concuerda realizándose el mismo algoritmo de hash. La forma en la que se incentiva a los nodos para mantener la información en sus discos duros es a través de Filecoin. Filecoin es una red Blockchain sobre IPFS que otorga dinero a cambio de espacio en el disco duro, y se cerciora de que existan múltiples copias de un mismo fichero para que siempre esté disponible.

Se evita entrar en más detalle sobre esta tecnología ya que escapa al objetivo de este proyecto. Sin embargo, es un tipo de tecnología descentralizada que dado el clima actual de censura de algunas ideas ha de ser tenido en cuenta.

6.4 IDEAS PROPIAS POTENCIALES PARA EL FUTURO DE BLOCKCHAIN Y RELACIONADAS CON EL TRABAJO.

En este apartado se explicarán las ideas propias que han ido surgiendo a medida que se ha desarrollado este trabajo de fin de grado. Se organizarán en una pequeña lista y se ofrecerán consideraciones al respecto. Existe la posibilidad de que estas ideas ya hayan sido consideradas y desestimadas en el pasado, sin embargo, no se han encontrado evidencias de ello y por esto se exponen.

1. **El fracaso de la Ley de Moore**. Al comienzo de este trabajo se explicó que cuando la tecnología fue creada, Satoshi desestimo el problema de la constante y creciente necesidad de almacenamiento mediante la Ley de Moore que afirma que cada dos años el almacenamiento disponible se iría duplicando. Esto no se ha dado por lo que otra solución al problema del almacenamiento debe darse. Se utilizará como ejemplo de este problema a la red Ethereum. A continuación, se muestra una grafica con el crecimiento que ha tenido esta red en menos de dos años.

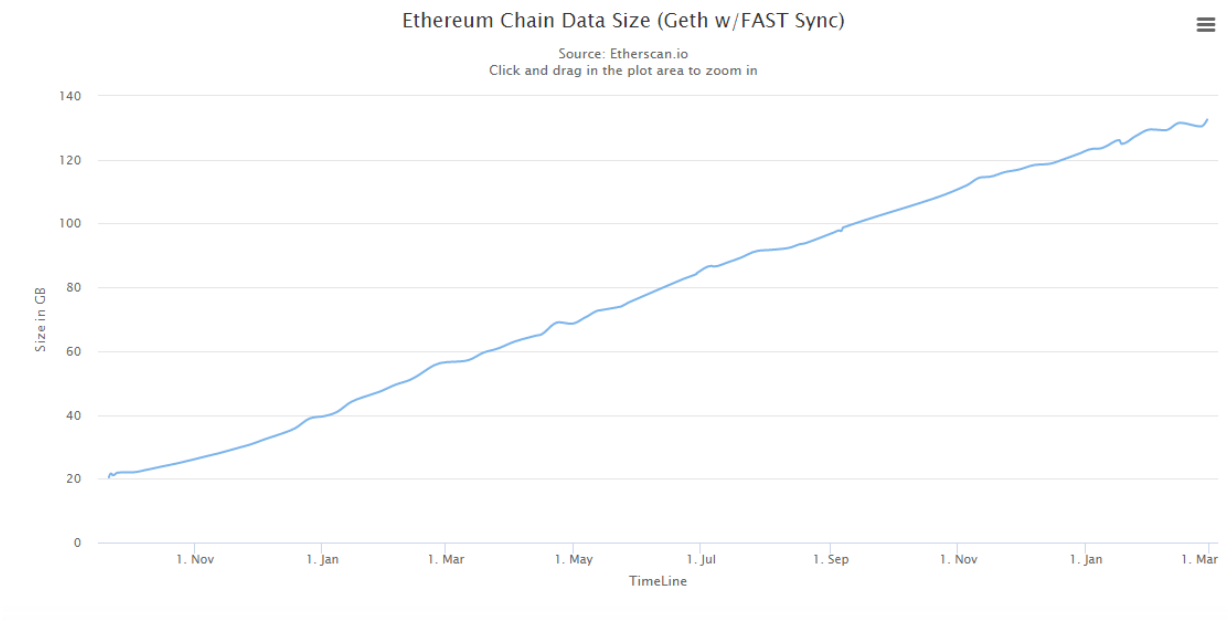

Figura 28. Almacenamiento de la red Ethereum en función del tiempo

Como se puede observar, los datos de la red han crecido desde septiembre de 2017 hasta marzo de 2019 de aproximadamente 20 GB a casi 140GB. Este crecimiento es aparentemente insostenible. Se postula la posibilidad de realizar un Hard Fork que incluya la posibilidad de truncar la red. Esto no se puede realizar a la fuerza eliminando una porción de la red ya que iría contra el principio de inmutabilidad de Blockchain. Lo que si parecería una opción es la designación de super nodos. Por tanto, habría tres categorías de nodos. Nodos ligeros, nodos completos y super nodos. Un super nodo deberá mantener una copia completa de la cadena de bloques. La forma en la que esto se incentivara es dedicando un porcentaje de los beneficios de la creacion de cada bloque a pagar una comisión a estos super nodos. Por ejemplo, un 0,5% de lo que le repercuta un bloque a un minero se destina a pagar a estos super nodos. La ley de oferta y demanda hará que se establezca un numero de nodos determinado en función de lo que se esté dispuesto a pagar por ellos. Nótese que la cantidad es totalmente arbitraria en este ejemplo y que se debe estudiar cual es la cantidad óptima. Estos nodos serán periódicamente encuestados por otros para asegurar que tengan la información. Existen muchos problemas que pueden ir

surgiendo a raíz de esta solución, sin embargo, parece una solución al problema del almacenamiento a tener en cuenta.

2. **Centralizar parte de la Dapp**. A medida que se ha ido desarrollando el trabajo, se ha descubierto que el uso de Metamask puede resultar complejo para un usuario con menos experiencia en la tecnología. Por ello se investiga la opción de que los usuarios accedan a través de usuario y contraseña únicamente, ya que es una forma que le es familiar a cualquier internauta. Sin embargo, no se dispone de la seguridad para mantener sus claves privadas (para interactuar con la red Ethereum) de manera segura y seria una forma de centralización, a fin de cuentas. Por tanto, se plantea la idea de utilizar el sistema de Microsoft Azure Key Vault. De esta manera, los usuarios accederían a sus claves privadas a través de este sistema, firmarían las transacciones y las mandarían para que el servidor de la aplicación sea el encargado de mandarlas a Ethereum. El servidor de la aplicación jamás entraría en contacto con las claves privadas y el usuario podría firmar transacciones solo a través de un correo y contraseña. Microsoft además de ser una entidad de confianza no tienen acceso a las propias claves que guardan por lo que si se olvida el usuario de su contraseña perderá el acceso a su cuenta Ethereum (igual que pasaría si perdiese acceso en un sistema tradicional de su clave privada)
3. **Sustituir los eventos por encuestas al SC**. Esto implicaría mayor lógica en el servidor, pero ahorraría el gas que producirían los eventos. Sin embargo, quizás merezca la pena gastar algo de gas para organizar lo mejor posible la información que produce el *Smart Contract*.

Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

El presente trabajo ha tenido como objetivo principal una instrucción profunda sobre los conceptos que rodean a la tecnología Blockchain. No hay motivos para pensar que esta tecnología no tendrá una notoria importancia en el futuro, y todo lo aprendido a lo largo de este trabajo resultara muy útil.

Por otro lado, se ha expandido en numerosos conocimientos adquiridos a lo largo de la carrera, como puede ser el concepto de hash o el concepto de clave pública-privada… Esto a permitido desarrollar la idea de investigación propia, es decir, el grado ha proporcionado las herramientas necesarias para desenvolverse en cualquier sector en relación con las telecomunicaciones, y estas han sido utilizadas para profundizar en un área totalmente nueva.

Quedan por determinar una serie de puntos sobre el trabajo, y se postulan para futuras investigaciones relacionadas con el tema. En primer lugar, quizás proceda hacer uso de la red permisionada Hyperledger en vez de Ethereum. Esta premisa se rechaza a priori debido a la barrera de entrada adicional que esto supondría para las pymes, ya que para pertenecer a una red permisionada, se debe ser aceptado por el resto de nodos de la red.

También existe la posibilidad de que la aplicación en general no sea necesaria. Por lo menos la funcionalidad de la deuda circular. La forma en la que se determinaría si la cancelación de la deuda circular es un prospecto de negocio viable (aunque solo sea parte de la funcionalidad de la app) se puede determinar a través de entidades bancarias de la forma especificada en el trabajo.

Existe otro dilema que surge de la posible (y probable) existencia de múltiples círculos de deuda. Esto implicaría que se deberá elegir que círculos priorizar en la cancelación (supóngase un nodo o más comunes a múltiples círculos). Los parámetros de elección se

dejan propuestos para futuros trabajos (Scoring, mayor cantidad neta de cancelación…). Todos los parámetros que se estipulen han de ser objetivos y verificables.

Se debe estudiar la posibilidad de eliminar la función del *Smart Contract* que obliga a que se confirme la cancelación de deuda a cada uno de los usuarios. Esto puede ser utilizado como un parámetro para elegir el círculo de cancelación. Sin embargo, se estima que ralentizara el proceso y dificultara la cancelación. Se entiende que un usuario que se registre en la aplicación desea que, ante la posibilidad de la cancelación de deuda, esta sea cancelada.

Si se da la situación en la que un usuario no confirma el pago de una deuda (uno de los mecanismos para dar fe de que la deuda ha sido saldada entre dos entidades), este problema ha de ser resuelto por la aplicación, reservándose el uso de una cuenta que pueda saldar cualquier deuda. Esta cuenta ha de explicar y dejar claros los motivos por la que es utilizada siempre y cuando se emplee.

Por último, puede añadirse la posibilidad de, aunque no se produzca un círculo cerrado de deuda, que esta sea simplificada, poniendo en contacto a usuarios que, aunque no se conozcan, puedan simplificar el saldado de muchas cuentas solamente con saldarlas entre ellos. Este comportamiento, si se emplea por parte del usuario, deberá ser recompensado por la aplicación (mejora de Scoring, por ejemplo).

Capítulo 8. BIBLIOGRAFÍA

[3] NatashaTurak. (2018, July 11). Global debt hits a new record at $247 trillion. Obtenido en: https://www.cnbc.com/2018/07/11/global-debt-hits-a-new-record-at-247-trillion.html

[4] Hashim, A., & Hashim, A. (2019). Lights out: 'Circular debt' cripples Pakistan's power sector. Obtenido de https://www.aljazeera.com/ajimpact/lights-circular-debt-cripples-pakistans-power-sector-190524055240222.html

[5] Running a Full Bitcoin Node for Investors. (2019). Obtenido de https://www.investopedia.com/news/running-full-bitcoin-node-investors/

[6] Tibken, S. (2019). CES 2019: Moore's Law is dead, says Nvidia's CEO. Obtenido de https://www.cnet.com/news/moores-law-is-dead-nvidias-ceo-jensen-huang-says-at-ces-2019/

[7] Bitcoin Energy Consumption Index - Digiconomist. (2019). Obtenido de https://digiconomist.net/bitcoin-energy-consumption

[8] Proof of stake instead of proof of work. (2019). Obtenido de https://bitcointalk.org/index.php?topic=27787.0

[9] Code For Ethereum's Proof-of-Stake Blockchain to Be Finalized Next Month - CoinDesk. (2019). Obtenido de https://www.coindesk.com/code-for-ethereums-proof-of-stake-blockchain-to-be-finalized-next-month

[10] Szabo, N. (1994). Smart Contracts. Obtenido de http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html

[11] Wood, D. (2019). Obtenido de https://ethereum.github.io/yellowpaper/paper.pdf

[12] ethereum/EIPs. (2019). Obtenido de https://github.com/ethereum/EIPs/blob/master/EIPS/eip-161.md#specification

[13] ETH Gas Station. (2019). Obtenido de https://ethgasstation.info/

[14] Chriest, J. (2019). From Javascript to Solidity. Obtenido de https://medium.com/coinmonks/from-javascript-to-solidity-12317e8965de

[15] McKie, S. (2019). Solidity Learning: Revert(), Assert(), and Require() in Solidity, and the New REVERT Opcode in the…. Obtenido de https://medium.com/blockchannel/the-use-of-revert-assert-and-require-in-solidity-and-the-new-revert-opcode-in-the-evm-1a3a7990e06e

[16] Units and Globally Available Variables — Solidity 0.4.24 documentation. (2019). Obtenido de https://solidity.readthedocs.io/en/v0.4.24/units-and-global-variables.html

[17] Style Guide — Solidity 0.4.24 documentation. (2019). Obtenido de https://solidity.readthedocs.io/en/v0.4.24/style-guide.html

[18] Which TCP and UDP ports are required to run an Ethereum client?. (2018). Obtenido de https://ethereum.stackexchange.com/questions/809/which-tcp-and-udp-ports-are-required-to-run-an-ethereum-client

[19] What are ipc and rpc?. (2016). Obtenido de https://ethereum.stackexchange.com/questions/10681/what-are-ipc-and-rpc

[20] How to reduce the chances of your Ethereum wallet getting hacked?. (2016). Obtenido de https://ethereum.stackexchange.com/questions/3887/how-to-reduce-the-chances-of-your-ethereum-wallet-getting-hacked

[21] Szilágyi, P. (2017). ethereum/go-ethereum. Obtenido de https://github.com/ethereum/go-ethereum/wiki/Management-APIs#miner

[22] (2019). Obtenido de https://etherscan.io/tokens

[23] DApps shouldn't be 100% decentralized. (2018). Obtenido de https://medium.com/mesg/dapps-shouldnt-be-100-decentralized-c3fb4d0d6888

[24] Marx, S. (2017). Testing and Deploying Smart Contracts with Remix. Obtenido de https://programtheblockchain.com/posts/2017/12/19/testing-and-deploying-smart-contracts-with-remix/

[25] Blanco, J. (2019). solidity - Visual Studio Marketplace. Obtenido de https://marketplace.visualstudio.com/items?itemName=JuanBlanco.solidity

[26] web3.js - Ethereum JavaScript API — web3.js 1.0.0 documentation. (2019). Obtenido de https://web3js.readthedocs.io/en/1.0/

[27] Opera R3: El nuevo estándar de navegación con W3 y Criptocartera | Navegador Opera para ordenadores | Opera. (2019). Obtenido de https://www.opera.com/es/reborn3

[28] Bangare, S., Gupta, S., Dalal, M., & Inamdar, A. (2016). Using Node.Js to Build High Speed and Scalable Backend Database Server. Obtenido de https://www.researchgate.net/profile/Sunil_Bangare/publication/301788361_Using_NodeJs_to_Build_High_Speed_and_Scalable_Backend_Database_Server/links/57285d6c08aee491cb416ad6/Using-NodeJs-to-Build-High-Speed-and-Scalable-Backend-Database-Server.pdf

[29] Buterin, V. (2017). Zk-SNARKs: Under the Hood. Obtenido de https://medium.com/@VitalikButerin/zk-snarks-under-the-hood-b33151a013f6

[30] Atilano

[30] ethereum/go-ethereum. (2019). Obtenido de https://github.com/ethereum/go-ethereum/blob/master/params/protocol_params.go

[31] ethereum/go-ethereum. (2019). Obtenido de https://github.com/ethereum/go-ethereum/blob/master/core/tx_pool.go

[32] Shahda, W. (2019). Gas Optimization in Solidity Part I: Variables. Obtenido de https://medium.com/coinmonks/gas-optimization-in-solidity-part-i-variables-9d5775e43dde

[33] What is Practical Byzantine Fault Tolerance (pBFT)? - Crush Crypto. Obtenido de https://crushcrypto.com/what-is-practical-byzantine-fault-tolerance/

[34] Obtenido de https://themeforest.net/

[35] Mhatre, S. (2017). Installing and setting up mysql with nodejs in ubuntu. Obtenido de https://medium.com/technoetics/installing-and-setting-up-mysql-with-nodejs-in-ubuntu-75e0c0a693ba

ANEXO A: MANUAL DE INSTALACIÓN Y USUARIO.

En el siguiente manual se detallarán brevemente los pasos seguidos para la instalación del prototipo.

En primer lugar, fue recomendado por uno de los directores, que se desarrollase el trabajo en una máquina virtual. Se instalo Ubuntu 18. Tras esto se procedió a instalar las dependencias mencionadas en la memoria. *VSCode* y a través de la línea de comandos el paquete *npm*. Con este paquete se pueden instalar todas las dependencias que se necesitaran de node.

Se crea una carpeta y se procede a introducir el comando "*npm install*" seguido por las dependencias requeridas. Estas dependencias se guardan en el paquete *package.json*. Se muestra el paquete a continuación.

```
"devDependencies": {
  "bootstrap": "4.1.3",
  "chai": "^4.1.2",
  "chai-as-promised": "^7.1.1",
  "chai-bignumber": "^2.0.2",
  "lite-server": "^2.3.0",
  "nodemon": "^1.17.3",
  "truffle": "5.0.2",
  "truffle-contract": "3.0.6"
},
"dependencies": {
  "balanced-match": "^1.0.0",
  "bcryptjs": "^2.4.3",
  "connect-flash": "^0.1.1",
  "ejs": "^2.6.1",
  "express": "^4.16.4",
  "express-flash": "0.0.2",
  "express-handlebars": "^3.1.0",
  "express-mysql-session": "^2.1.0",
  "express-session": "^1.16.1",
  "express-validator": "^5.3.1",
  "method-override": "^3.0.0",
  "mongoose": "^5.5.10",
  "morgan": "^1.9.1",
  "mysql": "^2.17.1",
  "passport": "^0.4.0",
  "passport-local": "^1.0.0",
  "random": "^2.1.1",
  "timeago.js": "^4.0.0-beta.2"
}
```

Figura 29. Dependencias en el fichero package.json

La diferencia entre devDependancies y dependencies es que las primeras solo son necesarias durante la etapa de desarrollo de la aplicación y las dependencias lo son tanto en la etapa de desarrollo como en la de ejecución. Algunas de las dependencias mostradas no se han llegado a usar. Esto se debe a que se hicieron muchas pruebas antes del desarrollo de la aplicación en su totalidad y algunas quedaron residuales. Si no se invocan no perjudican a la ejecución del prototipo.

Posteriormente se descargó una plantilla html css y javascript de la página web referenciada[34]. Tras esto se incorporaron las plantillas necesarias a la aplicación. Para juntar los diferentes elementos html y mantener la consistencia entre paginas (misma cabecera, mismo footer…) se hizo uso de EJS (Embeded JavaScript), que permite mandar información

por parte del servidor para realizar cambios de forma sencilla en el html. Tras esto, se montó un inicio de sesión sencillo a través de un registro y un login. Para ello se tuvo que descargar una base de datos. La base de datos utilizada es MySql. El proceso de integración de la base de datos con node.js se aprendió en esta fuente [35].Se muestra a continuación unas capturas del proceso.

```
inigo@inigo-VirtualBox:~/Dropbox/Iñigo/ICAI/TFG/1 Dapp DebtCrasher Mysql$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.7.26-0ubuntu0.18.04.1 (Ubuntu)

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Figura 30. Conexión con la base de datos

Tras esto se muestra un ejemplo del register y login. Actualmente la tabla con las cuentas solo tiene dos entradas.

```
mysql> SELECT * FROM accounts;
+----+----------+----------+--------------------+---------+
| id | username | password | email              | address |
+----+----------+----------+--------------------+---------+
|  1 | test     | test     | test@test.com      | 0x00    |
|  2 | Inigo    | Inigo    | isagredo@gmail.com | NULL    |
+----+----------+----------+--------------------+---------+
2 rows in set (0.00 sec)
```

Figura 31. Tabla accounts Mysql

Se procede a añadir un usuario. Nótese que se han empleado sesiones para que ningún usuario pueda acceder a la página principal sin estar identificado. A continuación, se muestra la pantalla de registro.

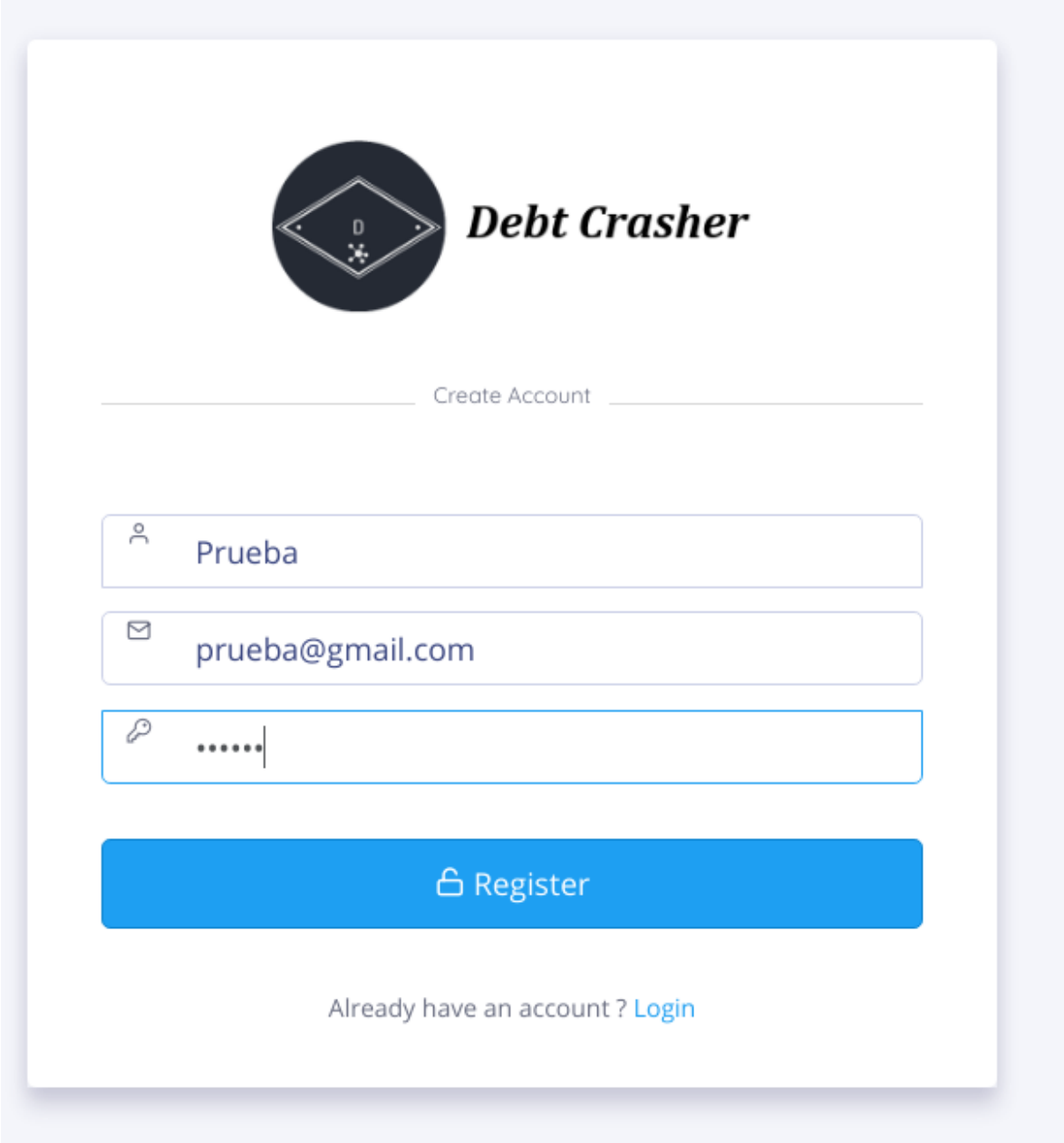

Figura 32. Registro del prototipo

Tras esto, el servidor redirige a través del método post (mediante Express.js) a la página de inicio de sesión.

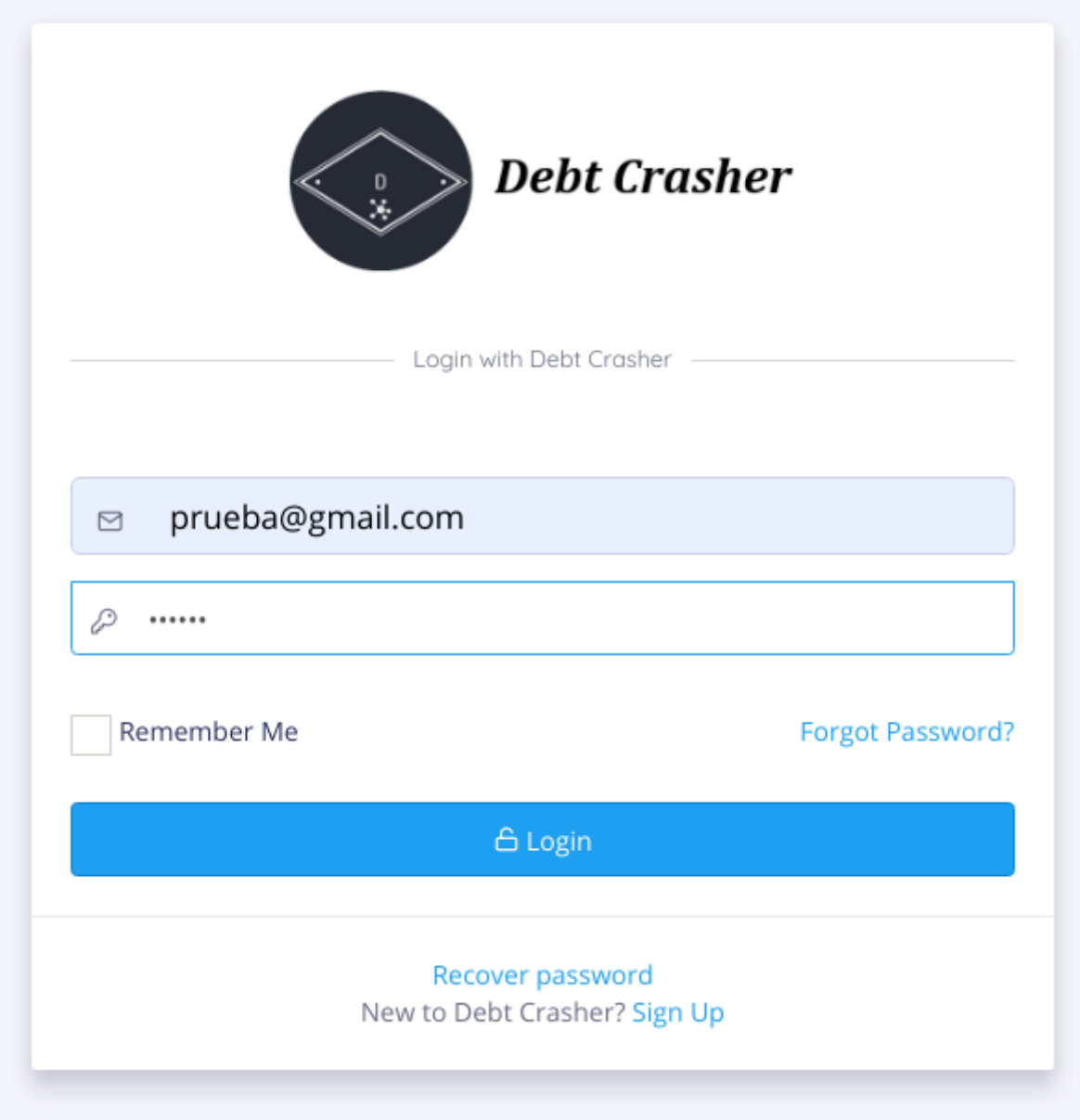

Figura 33. Inicio de sesión del prototipo

Se introducen los datos y se procede al menú principal de la aplicación.

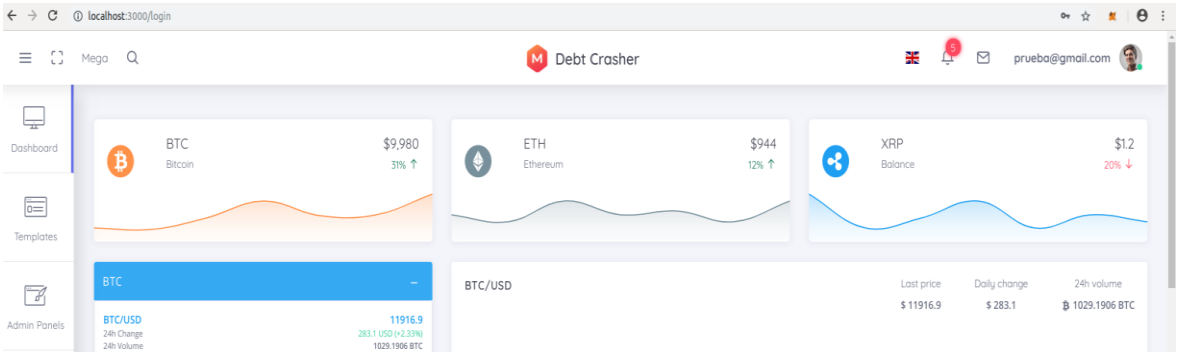

Figura 34. Página de inicio

Nótese como los cambios se hacen permanentes gracias a que se guardan en una base de datos y no en memoria.

```
mysql> SELECT * FROM accounts;
+----+----------+----------+--------------------+---------+
| id | username | password | email              | address |
+----+----------+----------+--------------------+---------+
|  1 | test     | test     | test@test.com      | 0x00    |
|  2 | Inigo    | Inigo    | isagredo@gmail.com | NULL    |
|  3 | Prueba   | prueba   | prueba@gmail.com   | NULL    |
+----+----------+----------+--------------------+---------+
3 rows in set (0.00 sec)
```

Figura 35. Base de datos con el nuevo valor

Tras esto se introduce la deuda. Para ello se navega a la siguiente ventana.

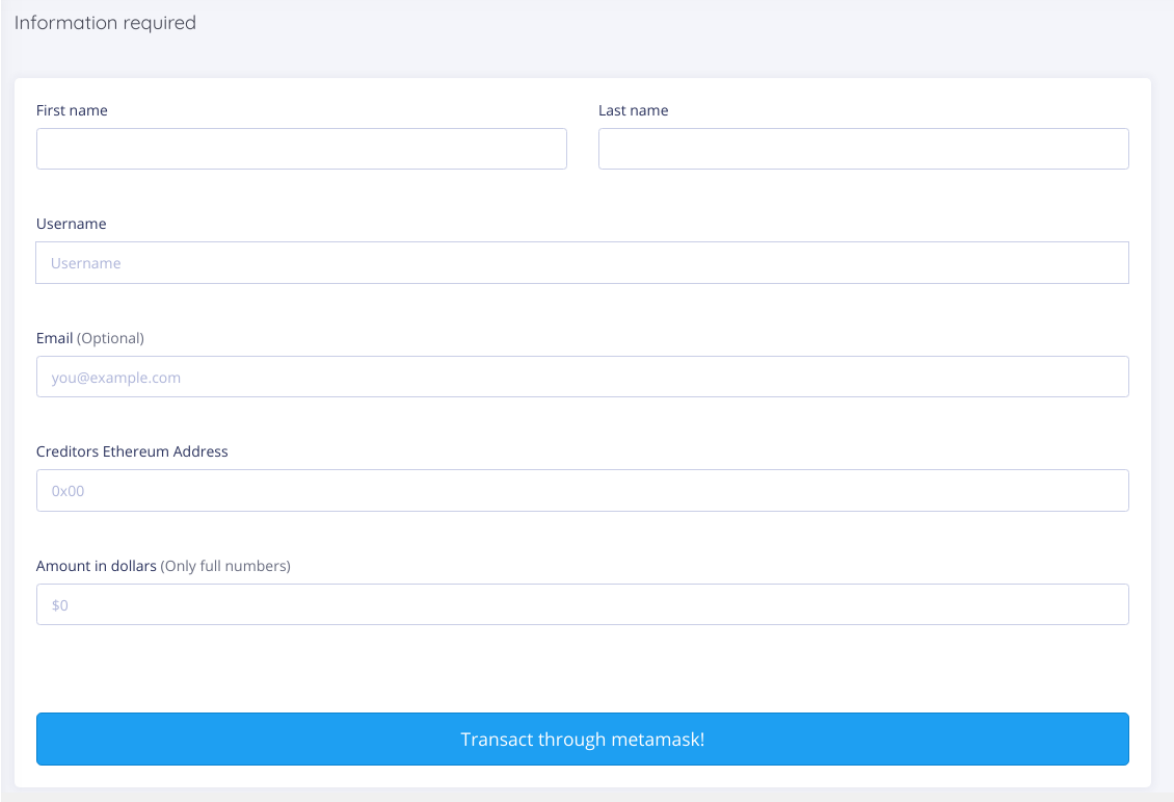

Figura 36. Ventana de creacion de deuda.

Estos datos son recogidos. El dato de la dirección Ethereum se recoge por Metamask. En Metamask se está usando la cuenta del usuario. Se debe pasar dicha cuenta y guardarla en la base de datos.

A continuación, se muestra la estructura de los ficheros empleados antes del despliegue del contrato.

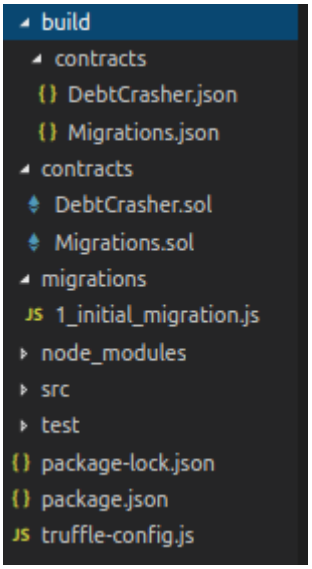

Figura 37. Estructura de los ficheros

El significado de cada uno de los directorios relacionados con Blockchain ha sido explicado en secciones anteriores. Tras esto se conecta la aplicación con Ganache. Esto se hace a través del fichero creado por Truffle; truffle-config.json. Se muestra a continuación.

```
module.exports = {
  networks: {
    development: {
      host: "127.0.0.1",
      port: 7545,
      network_id: "5778" // Match any network id
    }
  },
  solc: {
    optimizer: {
      enabled: true,
      runs: 200
    }
  }
}
```

Figura 38. Condigo del fichero truffle-config.js

Figura 39. Configuración en Ganache.

Véase como la configuración en ganache y en Truffle coinciden para permitir así la conexión.

Se despliega a la red de prueba Ganache el Smart Contract a tarves de una transacción. Para ello se crea el siguiente fichero:

```
var DebtCrasher = artifacts.require("./DebtCrasher.sol");

module.exports = function(deployer) {
  deployer.deploy(DebtCrasher);
};
```

Figura 40. Condigo de 2_deploy_contracts.js para desplegar el contrato.

Y se ejecuta el comando Truffle migrate. De esta forma se mira en la carpeta migrations y se ejecuta el código en orden. El 1_initial_migrations para la migración inicial de migrations.sol y el fichero 2_deploy_contracts para desplegar DebtCrasher. En la consola se muestra que el despliegue ha sido un éxito.

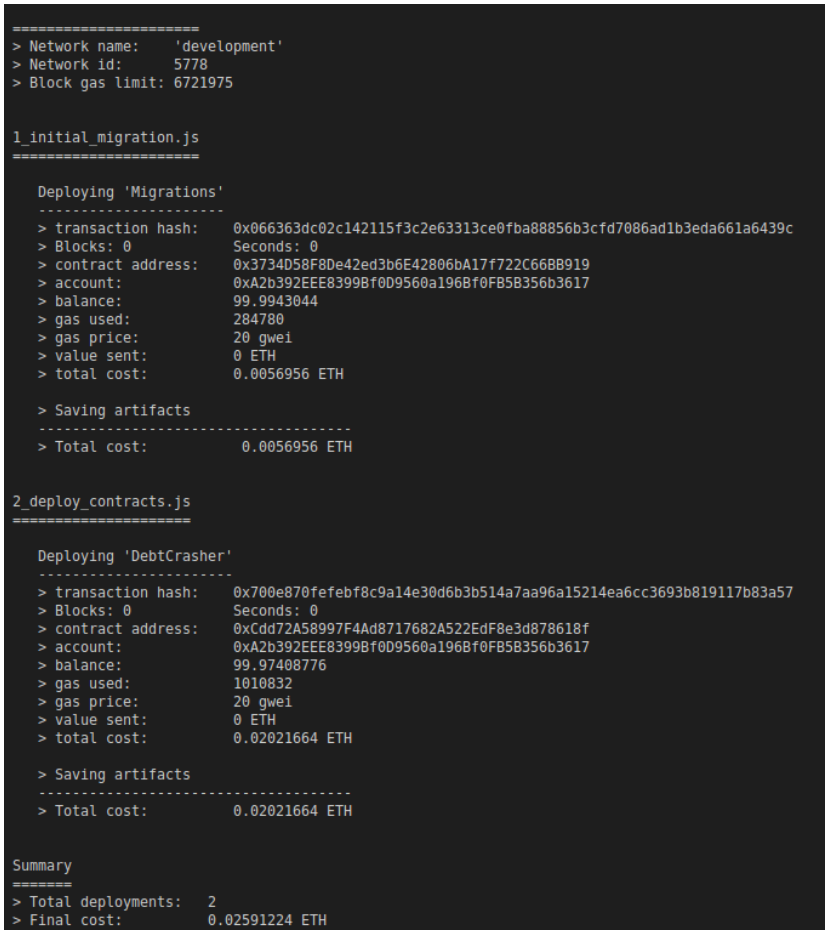

Figura 41. Despliegue del contrato exitoso mostrado por consola.

Tras esto se crea un archivo Javascript. Este archivo se referencia en los html's que se usen, y en el se incluye la lógica para la llamada al Smart Contract a través de web3. ´

```
loadWeb3: async () => {
  if (typeof web3 !== 'undefined') {
    App.web3Provider = web3.currentProvider
    web3 = new Web3(web3.currentProvider)
  } else {
    window.alert("Please connect to Metamask.")
  }

  if (window.ethereum) {
    window.web3 = new Web3(ethereum)
    try {

      await ethereum.enable()

      web3.eth.sendTransaction({/* ... */})
    } catch (error) {

    }
  }

  else if (window.web3) {
    App.web3Provider = web3.currentProvider
    window.web3 = new Web3(web3.currentProvider)

    web3.eth.sendTransaction({/* ... */})
  }

  else {
    console.log('Non-Ethereum browser detected. You should consider trying MetaMask!')
  }
},
```

Figura 42. Código para la conexión web 3.

Tras esto se manda la información necesaria (address y cantidad) con JavaScript al Smart Contract y esta queda registrada. Se mostrará el proceso en el prototipo de la presentación.

ANEXO B: CÓDIGO SMART CONTRACT SOLIDITY.

Si bien es cierto que se va a omitir la mayor parte del código realizado para el prototipo del trabajo, como la parte de servidor desarrollada en node.js o el resto de código en cliente, se desea mostrar el código realizado para el Smart Contract, ya que en el se muestran muchos de los principios en los que se ha hecho hincapié en el resto de la memoria. El código esta comentado, permitiéndose así una mejor comprensión de este.

```
pragma solidity ^0.5.0; //La version de solidity en la que se
programara.
contract DebtCrasher {

    // Data
    address payable owner;
    uint creationTime;

    struct Usuario {
        bool regist; // si el usuario esta registrado.
        address addr; // Quizas es redundante pero aclara la idea.
    }


    mapping(address => Usuario) usuarios;
    address[] users; //Un array con los address de los integrantes
    mapping(address => mapping(address=>int256)) desireToCancel;
//Indica la cantidad que desea cancelar de su deuda con la persona
pertinente.
    mapping(address => mapping(address=>int256)) relaciones; //La
mejor forma de resolver que una persona pueda tener muchas deudas con
muchas personas.
    mapping(address => address[]) contactos; //Los addresses con
relacion. Asi se pueden iterar. (a los que debe)


    //Eventos

    event debtCancelled(address[] usuariosInvolucrados,
                        int256 cantidadCancelada
    ); //Queda registrada la deuda que se cancela
```

```
//constructor

      constructor() public {
        owner = msg.sender; // reconoce al creador como el owner del
contrato
        creationTime = now;
    }


    //functions

    function()  payable external{} //Fallback function.

    function register() public onlyNew{ //Uso el modificador para que
esta funcion solo la puedan llamar nuevos usuarios.

            usuarios[msg.sender].regist=true;
            usuarios[msg.sender].addr =msg.sender;
            users.push(msg.sender);
    }


   function crearDeuda(address acreedor, int256 cantidad)
positiveAmount(cantidad) noSelfDebt(acreedor) public{ //msg.sender
sera el deudor.

       if(relaciones[msg.sender][acreedor]==0){ //Le agrego a su
lista de contactos que debe
       bool temp=false;
            for (uint i=0; i<contactos[msg.sender].length; i++){//Me
aseguro de que =0 no es una casualidad de que se hayan cancelado
deudas en el pasado
                if(contactos[msg.sender][i]==acreedor){
                    temp=true;
                }
            }
            if(temp==false){ //Si no esta en el array, lo meto, sino
no hace falta ya que ya esta metido.
                contactos[msg.sender].push(acreedor);
            }

       }
       relaciones[msg.sender][acreedor]+=cantidad;

   }
```

```
   function permisionToCancel(address addr, int256 cant) public{ //Yo
sere el encargado a traves del servidor de preparar esta trasaccion
para que el tenga que solo mandarla.
       require(
           relaciones[msg.sender][addr]>=cant
           );
       desireToCancel[msg.sender][addr]=cant;

   }

   function cancelarDeuda(address[] memory  addr, int256 cant) public
{ //En principio cualquiera es libre de llamar a esta funcion. Ver si
conviene que lo haga solo el owner.
       //Se usara un require para asegurar que lo que se va haciendo
es correcto

       require(
                relaciones[addr[addr.length-1]][addr[0]]>=cant //El
unico que no se comprueba en el for loop (Atento al -1 del array.
Sino desborda el GAS)
            );
        for (uint i=0; i<addr.length-1; i++){
            require(
                relaciones[addr[i]][addr[i+1]]>=cant
            ); //Si esto se cumple para todos los addresses, entonces
es que estamos en una situacion de deuda circular.
            require( //Todos los usuarios tienen que haber dado
consentimiento para la cancelacion de deuda. Sino no se podra. De
esto me debere de asegurar yo.
                desireToCancel[addr[i]][addr[i+1]]==cant
            );
            relaciones[addr[i]][addr[i+1]]-= cant; //Resto la
cantidad de la deuda. No hay problema si luego al final no es un
bucle cerrado pq revert lo manda al estado anterior.
        }
        //Si llega al final de este for, es que ha terminado con
exito. Lanzamos un evento. Para registrarlo Buena práctica.
        emit debtCancelled(addr,cant);

   }
```

```
  function viewDebt(address a, address b) view public returns
(int256){ //For debugging purposes. It shows the debt between two
people.
        return relaciones[a][b];
    }

    function kill() onlyBy(owner) public { //!!Quizas no sea buena
idea que si quiero destruir el contrato me quede con todo el dinero
del mismo.
        selfdestruct(owner);
    }

    modifier onlyNew{ //Para el registro
        require(usuarios[msg.sender].regist==false);
         _;
    }


  modifier positiveAmount(int256 cantidad) { //Solo tienen sentido
cantidades positivas de deuda.
     require(cantidad>0);
     _;
  }

  modifier noSelfDebt(address a){ //Evita que se cree deuda si el
acreedor es el mismo que quien envia el mensaje.
      require(msg.sender!=a);
      _;

  }

  modifier onlyBy(address a){
      require(msg.sender==a);
      _;
  }
}
```

ANEXO C: TÍTULOS COURSERA.

Se adjuntan los títulos obtenidos a lo largo del desarrollo del trabajo.

Ilustración 3. Titulo Coursera Blockchain Basics

Ilustración 4. Titulo Coursera Smart Contracts

Ilustración 5. Titulo Coursera Dapps.