DTrustManager
Sistema de confianza anónima aplicado a redes de blockchain

José María Franco Pérez

Grado de Ingeniería Informática

Trabajo Final de Grado

Aplicaciones y Sistemas Distribuidos - Blockchain

Félix Freitag

8 de Enero de 2020

FICHA DEL TRABAJO FINAL

Título del trabajo:	DTrustManager: Sistema de confianza anónima aplicado a redes de blockchain
Nombre del autor:	José María Franco Pérez
Nombre del consultor:	Félix Freitag
Fecha de entrega (mm/aaaa):	01/2020
Área del Trabajo Final:	Aplicaciones y Sistemas Distribuidos – Blockchain
Titulación:	*Grado de Ingeniería Informática*

Resumen del Trabajo (máximo 250 palabras):

Uno de los mayores problemas a los que se enfrentan las redes de blockchain hoy en día es la falta de confianza que existe entre dos usuarios que deciden interactuar entre ellos.

Generalmente, si dos personas quieren intercambiarse un token o valor, irán a un mercado secundario que intermedie por ellos ese intercambio. Además de los mercados secundarios, existen otras soluciones que solucionan el problema de confianza de otras maneras, pero siempre con un intermediario.

Las transacciones directamente entre pares, o peer-to-peer (P2P), requieren de una confianza mutua que sólo dos usuarios que se conocen "off-chain" suelen conseguir. La base del funcionamiento de las blockchain más comunes es la transmisión de tokens de un punto a otro sin posibilidad de "reembolsar" los tokens o "deshacer" la operación. Por lo tanto el usuario que realiza la transacción debe estar muy seguro antes de llevar a cabo la acción.

En este Trabajo de Fin de Grado se analiza e implementa una solución a este problema mediante la creación de un sistema de confianza, basado en redes de confianza (Web of Trust) y sistemas de reputación, que permita que los diferentes usuarios de una red puedan hacer pública su confianza en otros usuarios. De esta manera, cada usuario posee un Score de confianza que puede servir de base para futuras transacciones con otros usuarios desconocidos. Esta solución no requiere de la identificación real de un usuario, manteniendo así el anonimato de los usuarios dentro de las redes blockchain, característica fundamental de esta tecnología.

Abstract (in English, 250 words or less):

One of the biggest problems facing blockchain networks today is the lack of trust between two users who choose to interact with each other.

Generally, if two people want to exchange a token or security, they will go to a secondary

market that will broker that exchange for them. Besides secondary markets, there are other solutions that solve the trust problem in other ways, but always with an intermediary.

Transactions directly between peers, or peer-to-peer (P2P), require a mutual trust that only two users who know each other "off-chain" usually achieve. The basis of the operation of the most common blockchains is the transmission of tokens from one point to another without the possibility of "refunding" the tokens or "undoing" the operation. Therefore the user performing the transaction must be very sure before carrying out the action.

In this End-of-Degree Project, a solution to this problem is analyzed and implemented by creating a trust system, based on Web of Trust networks and reputation systems, that allows the different users of a network to make public their trust in other users. In this way, each user has a trust score that can serve as a basis for future transactions with other unknown users. This solution does not require the real identification of a user, thus maintaining the anonymity of users within blockchain networks, a fundamental characteristic of this technology.

Palabras clave (entre 4 y 8):

Blockchain, Ethereum, Sistema de Confianza, Reputación, Web of Trust

Índice

Lista de figuras

1 Introducción

1.1 Contexto y justificación del Trabajo

Uno de los mayores problemas a los que se enfrentan las redes de blockchain hoy en día es la falta de confianza que existe entre dos usuarios que deciden interactuar entre ellos.

Generalmente, si dos personas quieren intercambiarse un token o valor, irán a un mercado secundario que intermedie por ellos ese intercambio. Además de los mercados secundarios, existen otras soluciones que solucionan el problema de confianza de otras maneras, pero siempre con un intermediario.

Las transacciones directamente entre pares, o *peer-to-peer* (P2P), requieren de una confianza mutua que sólo dos usuarios que se conocen off-chain suelen conseguir. La base del funcionamiento de las blockchain más comunes es la transmisión de tokens de un punto a otro sin posibilidad de "reembolsar" los tokens o "deshacer" la operación. Por lo tanto el usuario que realiza la transacción debe estar muy seguro antes de llevar a cabo la acción.

En este Trabajo de Fin de Grado (TFG) se analiza e implementa una solución a este problema mediante la creación de un sistema de confianza, basado en redes de confianza (Web of Trust) y sistemas de reputación, que permita que los diferentes usuarios de una red puedan hacer pública su confianza en otros usuarios. De esta manera, cada usuario posee un Score de confianza que puede servir de base para futuras transacciones con otros usuarios desconocidos.

En un análisis previo que se ha realizado, se han encontrado algunas iniciativas que han trabajado en sistemas de reputación o confianza de diversa índole, pero siempre enlazando la reputación o confianza de un usuario con la verificación de su identidad real. En este proyecto se desarrolla una solución que no requiere de la identificación real de un usuario, manteniendo así el anonimato de los usuarios dentro de las redes blockchain, característica fundamental de esta tecnología.

1.2 Objetivos del Trabajo

El objetivo principal de este trabajo es el de realizar un análisis del estado del arte actual sobre las soluciones de confianza entre pares dentro de una red blockchain, y desarrollar y desplegar una herramienta que permita crear un sistema de confianza distribuida entre los usuarios de la red.

Se describen, a continuación, de manera más detallada, los objetivos específicos del trabajo.

Objetivo O1: Realizar un análisis del estado del arte sobre redes de confianza (Web of Trust) y de reputación sobre Blockchain

En primer lugar, y para formar una base sólida sobre la cual construir el resto de la investigación, se realizará un análisis del estado del arte en materia de redes de confianza (Web of Trust) y sistemas de reputación implementados sobre blockchain. Las principales redes sobre las que se basará el análisis son Ethereum y Hyperledger.

Para complementar el análisis, se realizará un estudio de los sistemas de confianza y reputación más comunes y de los cuales se hayan realizado implementaciones en entornos

fuera de blockchain. De esta manera se obtendrá un conjunto de soluciones validadas en otros entornos que pueden ser de utilidad para crear nuestra aplicación distribuida.

Objetivo O2: Diseñar una solución de Sistema de Confianza sobre blockchain, basada en el análisis previo, en forma de DApp

Tras realizar el análisis del estado del arte y los posibles algoritmos de confianza y reputación que se pueden implementar, se pasará a diseñar la solución para crear una DApp que implemente el sistema buscado. Este diseño incluirá un análisis de requisitos funcionales y no funcionales de la DApp, la arquitectura del sistema, el esqueleto del SmartContract que gestionará las operaciones necesarias, el Bridge para comunicarse con el SmartContract y el SmartContract y mockups del front-end con el que interactuarán los usuarios.

Objetivo O3: Desarrollar la solución diseñada en la red distribuida Ethereum

Con el diseño realizado, y optando por la blockchain Ethereum para desplegar la solución, se utilizará el lenguaje Solidity para desarrollar el SmartContract. Para el Bridge se utilizará Vanilla JavaScript junto con el empaquetador Webpack y para el front-end, se utilizarn las tecnologías web más comunes: HTML5, CSS, JavaScript, y las librerías Bootstrap y jQuery. Tras tener el desarrollo terminado en el entorno local, se realizarán los tests funcionales de la DApp y se desplegará la solución en la red de test de Ethereum, Rinkeby.

Objetivo O4: Analizar la viabilidad de la solución desde el punto de vista de varios casos de uso

Una vez terminado el desarrollo, se validará la solución para verificar, a un nivel más práctico, si la solución encontrada es válida para solucionar el problema que se está tratando en cada uno de los casos de uso a validar. Estos tests se desarrollarán utilizando herramientas de testing del framework utilizado para desarrollar el SmartContract, Truffle. Se crearán un número de nodos con un comportamiento predefinido y se simularán transacciones entre ellos.

Objetivo O5: Establecer las conclusiones y los próximos pasos a dar de la solución obtenida

Finalmente, se detallarán las conclusiones obtenidas sobre el trabajo realizado y, teniendo ya la visión completa del problema y su solución, se procederá a establecer los puntos en los que habría que seguir trabajando para mejorar la solución obtenida.

1.3 Enfoque y método seguido

La tecnología blockchain se sigue considerando una tecnología incipiente, dado que sigue recibiendo actualizaciones a nivel fundamental y aún no existen muchos casos de uso reales validados. Además, aunque existen herramientas de desarrollo cada vez más potentes, aún no hay soluciones que engloben todo el flujo de implementación de este tipo de soluciones. Por estas razones, este trabajo se ha enfocado desde el punto de vista de la investigación.

Así, se ha seguido un enfoque tradicional en este tipo de trabajos: análisis del sector, diseño conceptual de la implementación de la solución, desarrollo de un prototipo para validación, validación de la idea. A la hora de implementar, se valorarán las soluciones existentes y se escogerán aquellas que faciliten la labor de integración de los distintos componentes.

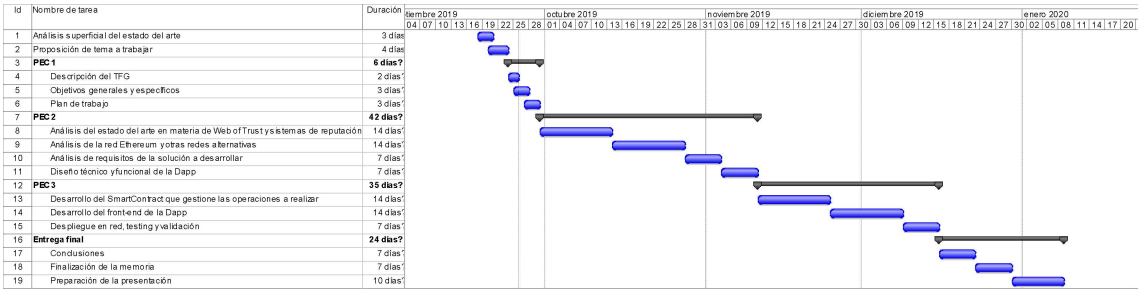
1.4 Planificación del Trabajo

A continuación se describen los principales hitos a cumplir en el desarrollo del TFG, además del resultado que se obtiene en cada uno de ellos:

1. **Realización del plan de trabajo:** Para cumplir este hito, se realiza una propuesta del tema en el que se trabajará y un análisis superficial del estado del arte para validarlo. Tras finalizarlo, se obtiene un documento que contiene la descripción a alto nivel del trabajo a realizar, los objetivos generales y específicos del proyecto, y el plan de trabajo a seguir.
2. **Análisis del estado del arte en materia de Web of Trust y sistemas de reputación:** Para cumplir este hito, se realizará un análisis más en profundidad del estado del arte, comprobando las iniciativas que se han emprendido en la aplicación del Web of Trust y sistemas de reputación en una red de blockchain. Al finalizar este hito, se obtendrá un documento resumen de la investigación realizada.
3. **Análisis de la red Ethereum y otras redes alternativas:** Con el estudio anterior, se obtendrán ejemplos de iniciativas similares a la que se intenta desarrollar, y las tecnologías utilizadas para ello. Para completar este hito, se realizará un análisis simple de las blockchain que mejor se ajusten a la solución buscada, entre ellas, Ethereum y Hyperledger. Tras cumplir este hito, se obtendrá un documento resumen con el análisis realizado y la decisión tomada.
4. **Análisis de requisitos de la solución a desarrollar:** Siguiendo el trabajo realizado hasta este punto, se realizará un análisis de requisitos funcionales y no funcionales de la solución a desarrollar. Además, se implementará el algoritmo de confianza que regirá los resultados del sistema. Este análisis dará como resultado un documento simple de requisitos que la DApp que se desarrollará deberá cumplir.
5. **Diseño técnico y funcional de la DApp:** Con los requisitos obtenidos, y para finalizar esta etapa de análisis y diseño, se realizará un diseño técnico y funcional de la DApp a desarrollar, que incluirá la arquitectura a construir, el esqueleto del SmartContract a implementar y mockups de las distintas interfaces con las que interactuará el usuario. Al finalizar este hito, se obtendrá una serie de esquemas representativos del diseño de la DApp (diagrama de la arquitectura, mockups, …).
6. **Desarrollo del SmartContract (o similar) que gestione las operaciones a realizar:** Para cumplir este hito, se desarrollará el SmartContract siguiendo el esquema obtenido en el hito anterior y utilizando el lenguaje necesario. En este hito se obtendrá uno o varios ficheros con el código del SmartContract.
7. **Desarrollo del Bridge y del front-end de la DApp:** Para completar este hito, se desarrollará el Bridge de comunicación con el SmartContract y las interfaces definidas en los hitos anteriores, de manera que permitan realizar las operaciones necesarias en la DApp diseñada. Tras finalizar, se obtendrá el código entero y el prototipo de la DApp planteada en este proyecto.
8. **Testing, validación y despliegue en red testnet pública**: Una vez la implementación ha finalizado en el entorno local, se pasará a realizar los tests funcionales finales y a validar la solución trabajada según los distintos casos de uso definidos. Seguidamente, se desplegará en una de las redes de test disponibles para Ethereum (como por ejemplo Rinkeby), o la blockchain elegida.

9. **Conclusiones y próximos pasos**: Con el trabajo de validación realizado, y para cumplir este hito, se obtendrán las conclusiones finales del proyecto realizado y se establecerán los próximos pasos a dar en caso de seguir con la investigación. La consecución de este hito resultará en un documento de conclusiones.
10. **Redacción de la memoria y preparación de la presentación:** Este último hito culmina con la concentración del trabajo realizado en una memoria final del TFG y la presentación de la investigación realizada.

Finalmente se muestra un diagrama tentativo de la ejecución de los trabajos definidos y la consecución de los principales hitos de este TFG:



Id	Nombre de tarea	Duración
1	Análisis superficial del estado del arte	3 días
2	Proposición de tema a trabajar	4 días
3	**PEC 1**	**6 días?**
4	Descripción del TFG	2 días?
5	Objetivos generales y específicos	3 días?
6	Plan de trabajo	3 días?
7	**PEC 2**	**42 días?**
8	Análisis del estado del arte en materia de Web of Trust y sistemas de reputación	14 días?
9	Análisis de la red Ethereum y otras redes alternativas	14 días?
10	Análisis de requisitos de la solución a desarrollar	7 días?
11	Diseño técnico y funcional de la Dapp	7 días?
12	**PEC 3**	**35 días?**
13	Desarrollo del SmartContract que gestione las operaciones a realizar	14 días?
14	Desarrollo del front-end de la Dapp	14 días?
15	Despliegue en red, testing y validación	7 días?
16	**Entrega final**	**24 días?**
17	Conclusiones	7 días?
18	Finalización de la memoria	7 días?
19	Preparación de la presentación	10 días?

1.5 Breve sumario de productos obtenidos

A continuación se presentan los productos obtenidos de este trabajo:

- **Memoria**: El documento actual. En él se presentan toda la investigación y el desarrollo realizados.
- **PECs entregadas**: Resultados parciales del Trabajo realizado. Cada PEC contiene el resultado del trabajo realizado hasta el punto de entrega y se detallan las conclusiones obtenidas hasta el momento y los cambios que ha ido sufriendo el proyecto. Además, cada PEC contiene como Anexo los documentos obtenidos de la consecución de los hitos planificados.
- **Código del SmartContract, Bridge y Front-end desarrollados**: Este código contiene toda la implementación de la solución. La parte más importante es el SmartContract, que contiene la lógica del sistema. El Bridge y el Front-end, además de ofrecer una interfaz para testing, definen también un ejemplo de integración de este tipo de aplicaciones.

1.6 Breve descripción de los otros capítulos de la memoria

A continuación se da una breve descripción de los capítulos incluidos en la memoria:

- **Análisis del Estado del Arte sobre redes de confianza (Web of Trust) y sistemas de reputación sobre Blockchain**: Este capítulo incluye el análisis del estado del arte sobre Web of Trust y sistemas de reputación, en el que se da una explicación de los diferentes conceptos que se tratan en el trabajo y se realiza un estudio de diferentes algoritmos que se han desarrollado en los campos de Web of Trust y Sistemas de reputación.
- **Análisis de la red Ethereum y otras redes alternativas**: Este capítulo entra en profundidad en la estructura de una blockchain básica. Se explica cómo funciona una

blockchain básica y se estudian tres ejemplos: Bitcoin, como blockchain de primera generación, y Ethereum y Hyperledger Fabric como ejemplos de blockchain de segunda generación.

- **Descripción y diseño de la solución a desarrollar**: En este capítulo se describe la solución a desarrollar y se establecen las funcionalidades que debe implementar así como la tipología de usuarios que interactuarán con ella. Por otro lado, se definen las tecnologías que se utilizan, además de hacer hincapié en el núcleo central de la solución a desarrollar: el algoritmo de confianza.
- **Desarrollo, despliegue y testing de la solución**: En este capítulo se detalla el proceso de desarrollo del Smart Contract utilizando Truffle y Ganache, así como el desarrollo del Bridge y de las interfaces necesarias con JavaScript y Webpack. También se incluye el resultado de las simulaciones realizadas en el entorno de simulación desarrollado.
- **Conclusiones y próximos pasos**: En este capítulo se describen las conclusiones obtenidas tras el trabajo realizado, así como la contribución realizada al sector de Blockchain y los próximos pasos a dar para continuar la investigación en esta línea.
- **Bibliografía**: Este capítulo incluye todas las referencias bibliográficas citadas en el trabajo.
- **Anexos**: Se incluye como Anexo el detalle del desarrollo realizado en la implementación del Sistema de Confianza.

2 Análisis del Estado del Arte sobre redes de confianza (Web of Trust) y sistemas de reputación sobre Blockchain

2.1 Preliminar

Este capítulo incluye el análisis del estado del arte sobre Web of Trust y sistemas de reputación, en el que se da una explicación de los diferentes conceptos que se tratan en el trabajo y se realiza un estudio de diferentes algoritmos que se han desarrollado en los campos de Web of Trust y Sistemas de reputación.

Antes de comenzar el análisis, definiremos brevemente las tecnologías y conceptos que se tratan en el documento: Blockchain, Web of Trust y Sistemas de reputación.

2.1.1 ¿Qué es Blockchain?

En 2009, apareció publicado el White paper[1] "Bitcoin: A Peer-to-Peer Electronic Cash System", firmado por Satoshi Nakamoto, definiendo por primera vez la tecnología Blockchain aplicada a la infraestructura de un sistema monetario electrónico distribuido y seguro, la red de Bitcoin (Nakamoto, 2009).

Sin embargo, aunque durante un tiempo la tecnología Blockchain pasó desapercibida, sin tener presencia por sí misma, sino únicamente como infraestructura de Bitcoin, hoy en día la realidad es muy diferente. Esta tecnología ya no está asociada únicamente a las criptofinanzas, y las palabras "Blockchain es el futuro" aparecen constantemente en los medios de comunicación (Parrondo, 2018), (García, 2018), (AMadrid, 2019).

Pero, ¿qué es Blockchain y por qué está revolucionando el entorno empresarial?

En pocas palabras, Blockchain, o "cadena de bloques", es una estructura de datos que agrupa el contenido en bloques enlazados entre sí utilizando técnicas criptográficas. Los bloques se enlazan entre sí mediante un hash que corresponde al contenido del bloque anterior, de manera que la información de un bloque no puede ser modificada sin modificar todos los hashes de los bloques sucesivos. Otra propiedad de la Blockchain es que se trata de un sistema distribuido, formado por un gran número de nodos que contienen la misma información, con lo que el sistema se asegura de que, si un nodo cae, la información sigue estando disponible en el resto de nodos. Finalmente, para confirmar las transacciones de la red, se aplican técnicas de consenso entre los distintos nodos del sistema (Cadena de bloques, 2015).

Aunque inicialmente se utilizó esta infraestructura para soportar sistemas electrónicos monetarios, podemos decir que una segunda generación de Blockchain vio la luz en el momento en el que empezó a utilizarse para albergar otro tipo de información, especialmente los contratos inteligentes, o *SmartContracts*.

En 2014, Vitalik Buterin publicó el trabajo que había desarrollado aplicando nuevas funcionalidades a la infraestructura Blockchain (Buterin, 2014), entre ellas, la capacidad de almacenar contratos inteligentes en una Blockchain que se ejecutaran utilizando cierta cantidad de una criptomoneda y almacenara los resultados en la propia Blockchain. De esta manera conseguía crear un sistema abierto, para que cualquier persona programara sus contratos inteligentes y pudiera llevar un registro transparente de las transacciones de ese contrato.

[1] Un White paper, o "Libro blanco" es un documento que generalmente ayuda a entender un tema en particular, a tomar decisiones frente a un problema concreto, o que propone soluciones a un problema.

Estos fueron los inicios de la red Ethereum, y esta es una de las razones por las que la tecnología Blockchain está ganando tanto impulso. Con la capacidad de almacenar transacciones de cualquier tipo en una red distribuida, segura y transparente, se abre un sinfín de posibilidades de aplicación de esta tecnología como puede ser el registro y monitorización de cualquier cadena productiva (Thomasson, Carrefour says blockchain tracking boosting sales of some products, 2019), o el registro de transacciones de cualquier activo (Premier, 2019).

2.1.2 ¿Qué son las redes de confianza (Web of Trust)?

El concepto de red de confianza, o Web of Trust, se utiliza principalmente en sistemas OpenPGP (*Open Pretty Good Privacy*, por sus siglas en inglés). Estos sistemas utilizan un algoritmo de cifrado de claves pública-privada mediante el cual se cifra una información con la clave privada que será, seguidamente, transmitida por un canal de comunicación no seguro y que podrá descifrarse tras su recepción utilizando la clave pública (Corporation, 2007).

Para poder utilizar un sistema como éste con cierta seguridad, es indispensable poder confiar en el autor del mensaje. En cada mensaje tendremos una clave pública que se utiliza para descifrar el contenido del mensaje, pero no podemos saber con seguridad si la clave pública pertenece al autor del mensaje, o se trata de una suplantación de identidad.

Para solucionar este problema, Philip Zimmermann, creador de PGP, estableció el concepto de Web of Trust: una persona X puede acreditar que una clave pública pertenece a una persona Y, cifrando y firmando la clave pública y los datos de Y con su clave privada (de X). De esa manera, X estará certificando la identidad de Y, hecho que puede ser verificado por cualquier persona que conozca a X. Si varias personas certifican la identidad de una persona, es muy probable que el destinatario de los mensajes de esa persona conozca al menos a una o dos de las personas que han certificado su identidad. A esta red de certificación de personas se le denomina "red de confianza" (Zimmerman, 1993).

Además, mediante estas redes de confianza podemos establecer una relación más entre pares: *la confianza indirecta*. Si A confía en B y B confía en C, podemos decir que A tiene una relación de confianza indirecta con C. Podemos ver un ejemplo de las relaciones entre los distintos nodos de una red en la Ilustración 1.

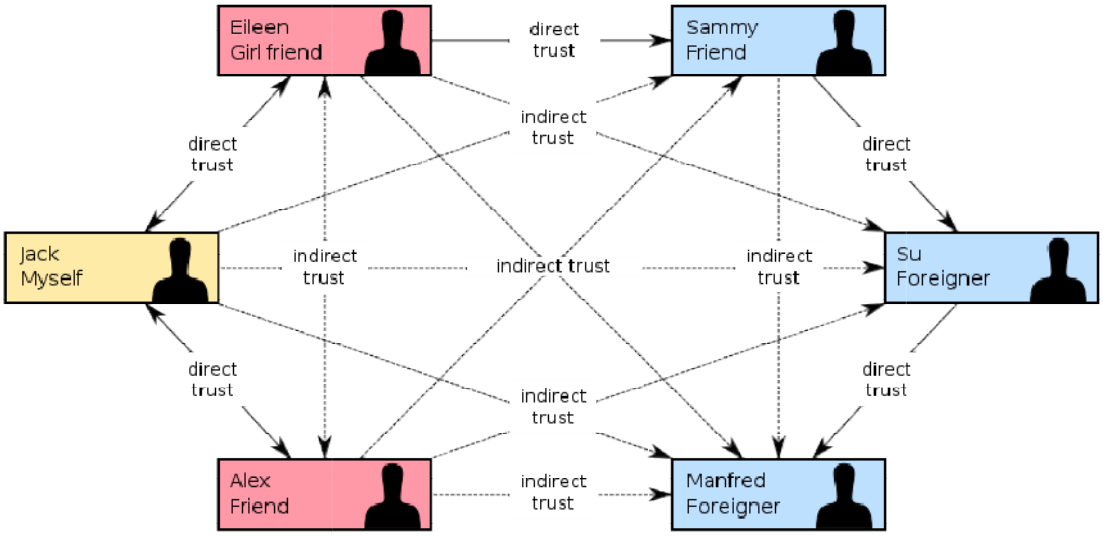


Ilustración 1: Khu. (2019). Web of Trust [Figura]. Obtenido de https://commons.wikimedia.org/wiki/File:Web_of_Trust-en.svg

La manera más común de certificar la identidad de una persona y su clave pública es mediante las fiestas de firmado de claves.

2.1.3 ¿Qué es un sistema de reputación?

Un sistema de reputación es aquel que permite medir el prestigio de una persona en la red en la que se aplica. El ejemplo más extendido es el sistema de puntuaciones entre usuarios de una plataforma, en el que un usuario, después de haber recibido un servicio o producto de otro usuario, puede "puntuar" dicho servicio/producto del 1 (puntuación más baja) al 5 (puntuación más alta) y, en ocasiones, dejar comentarios sobre la transacción. La media de todas las puntuaciones recibidas constituirá la reputación del usuario que ofrece esos servicios/productos. AirBnB (AirBnB) y Amazon (Amazon) son ejemplos de grandes portales que funcionan con un sistema de reputación como éste.

Sin embargo, no es el único algoritmo de medición de reputación que existe. Como veremos en el siguiente apartado, hay otros algoritmos que no se basan únicamente en las votaciones de otros usuarios. En contextos más amplios, como la medición de la reputación de una marca o empresa, se tienen en cuenta otros factores como, por ejemplo, la opinión de los contenidos de otros autores con respecto al elemento estudiado, o la aparición en medios de comunicación (Leiva-Aguilera, 2012).

2.2 Algoritmos

En este apartado, entramos ya en el análisis propiamente dicho y desarrollamos los algoritmos existentes de Web of Trust y Sistemas de reputación. Se analizarán tanto las soluciones existentes en el mercado como las investigaciones que se han realizado sobre ambos tipos de sistemas.

2.2.1 Red de confianza (Web of Trust)

Como se describió en la sección anterior, el término de red de confianza se refiere a la "confianza" existente entre los nodos de una red, entendiendo "confianza" como la acreditación que un usuario realiza de otro usuario.

Partiendo de un escenario en el que existe una red formada por un número indeterminado de nodos, un usuario A puede acreditar que confía en un usuario B, y este usuario B puede acreditar que confía en un usuario C. Se dice que A tiene una ***confianza directa*** con B y que B tiene una confianza directa con C. Se dice, también, que A tiene una ***confianza indirecta*** con C. Hay que tener en cuenta que las relaciones de confianza son unidireccionales. Es decir, el hecho de que A confíe en B, no necesariamente significa que B confía en A.

El primer algoritmo que se puede citar es el utilizado en PGP. Según este algoritmo, una persona X puede acreditar que una clave pública pertenece a una persona Y, cifrando y firmando la clave pública y los datos de Y con su clave privada (de X). De esa manera, X estará certificando la identidad de Y, hecho que puede ser verificado por cualquier persona que conozca a X (Zimmerman, 1993).

Otro algoritmo que podríamos incluir en el conjunto de "Web of Trust" es el algoritmo que utiliza Google para clasificar las páginas web en el buscador: PageRank. Según PageRank, Google otorga un Score a cada página web basado en la cantidad y calidad de enlaces que

apuntan a esa página. Así, una página web puede tener un Score alto si muchas páginas apuntan a ella, pero también, si pocas páginas que tienen un Score alto apuntan a ella (Page, 2006).

Por último, cabe destacar estudios como el de M. Bhattacharya en el que la red de confianza de un nodo de una red social se forma utilizando dos indicadores de confianza: una métrica global, basada en la interacción del nodo dentro de la red, y una métrica local basada en la confianza depositada por otros nodos en el nodo estudiado (Bhattacharya & Nesa, 2016).

Por otro lado, algunos estudios se basan en determinar la mejor forma de obtener el Score de confianza de un nodo con respecto a otro, buscando el camino de confianza que los une. Un ejemplo de este tipo de estudios es el de H. Zhang y A. van Moorsel, en el que analizan la eficiencia de 3 tipos de algoritmo de búsqueda en redes P2P: Inundación (Flooding), Consulta aleatoria (Random querying) y Consulta selectiva (Selective querying) (Zhang H., 2008).

2.2.2 Sistemas de reputación

Como se definió anteriormente, un sistema de reputación es aquel que permite medir el prestigio de una persona en la red en la que se aplica. Esta reputación viene dada, generalmente por otros usuarios de la misma red con los que el usuario ha interactuado. Tras una interacción, el usuario puntúa al otro usuario con un valor que añade a la lista de puntuaciones que ya ha recibido. Estas puntuaciones se pasan por una fórmula que permite obtener la reputación del usuario.

El algoritmo más común es el del cálculo de la reputación como la media de todas las reputaciones obtenidas. Es el algoritmo que se puede ver en AirBnB (AirBnB) y Amazon (Amazon), por ejemplo. En estos portales, después de haber recibido el servicio de hospedaje (AirBnB) o el producto comprado (Amazon), los usuarios pueden puntuarse recíprocamente con una valoración del 1 (muy mala) al 5 (muy buena), en base al servicio/producto recibido. En ambos casos, los portales obtienen más métricas (rapidez de respuesta, limpieza en el caso de AirBnB, estado del producto en el caso de Amazon,…) que se basan en el mismo sistema y que sirven de información adicional a futuros usuarios.

Otro algoritmo interesante es el presentado por Kamvar, Schlosser y Garcia-Molina, llamado **EigenTrust**, y utilizado en redes P2P mediante el cual la reputación de un usuario de la red es la ponderación de las valoraciones recibidas por ese usuario según las propias reputaciones de los emisores de las puntuaciones. Por ejemplo, si tenemos 3 usuarios en la red, Ana, Luis e Isabel, la reputación de Ana será igual a la suma de la puntuación que le ha dado Luis multiplicado por la reputación de Luis, y la puntuación que le ha dado Isabel multiplicado por la reputación de Isabel (Kamvar & Schlosser, 2003).

En el estudio de Xiong y Liu, se utilizan tres métricas para aplicar el algoritmo de reputación. Por un lado, la puntuación recibida de otros usuarios de la red, al igual que la mayoría de algoritmos de reputación. Por otro lado, el número de interacciones que un usuario ha tenido con otros usuarios. El tercer parámetro para aplicar el algoritmo es el factor de confianza del usuario. La reputación de un usuario viene dada, pues, por la ponderación de las reputaciones de los usuarios que lo han valorado multiplicado por sus factores de confianza, y todo ello dividido por la cantidad de interacciones que el usuario ha tenido (Xiong & Liu, 2002).

Finalmente, se destaca el algoritmo presentado por Mao y Shen, Web of Credit, en el cual se tienen en cuenta más factores a la hora de calcular la reputación de una red. La lógica detrás del algoritmo se puede resumir en que el algoritmo toma en cuenta cinco factores: Crédito (reputación total dada menos reputación total obtenida), Riesgo (asociado a un sector en particular y definido como una media entre las medias de reputación de un usuario en ese

sector y las del resto de usuarios), Inclinación (asociado también a un sector en particular, es la probabilidad de que el usuario tenga preferencia por ese sector según sus puntuaciones), Impedimento (métrica relativa a dos usuarios que representa cuán lejos están entre ellos en la red de confianza), y Confianza (también relativa a dos usuarios y que representa cuánta reputación ha recibido un usuario desde otro usuario a través de la red de confianza) (Mao & Shen, 2016).

3 Análisis de la red Ethereum y otras redes alternativas

Este capítulo entra en profundidad en la estructura de una blockchain básica. Se explica cómo funciona una blockchain básica y se estudian tres ejemplos: Bitcoin, como blockchain de primera generación, y Ethereum y Hyperledger Fabric como ejemplos de blockchain de segunda generación.

3.1 ¿Cómo funciona una Blockchain?

Una Blockchain es, esencialmente, una base de datos distribuida que contiene información sobre la propiedad de diversos "activos" y las transacciones realizadas entre los distintos participantes de la red.

Esta base de datos tiene la estructura de una cadena de bloques (de ahí el nombre "Blockchain"), en la que cada bloque contiene una lista de transacciones ordenadas cronológicamente que han ocurrido en la red y que han sido validadas por los usuarios correspondientes. Cada bloque, además, está enlazado cronológicamente con el bloque anterior, de manera que se mantiene un registro ordenado de transacciones que nos lleva a obtener los propietarios actuales de cada "activo" de la red.

3.1.1 Un bloque de la cadena de bloques

El elemento fundamental de la cadena de bloques es el bloque. ¿Qué guarda un bloque en su interior? Pongamos uno de los ejemplos más simples de bloque de una Blockchain: un bloque de la red Bitcoin, que podemos ver en la Ilustración 2.

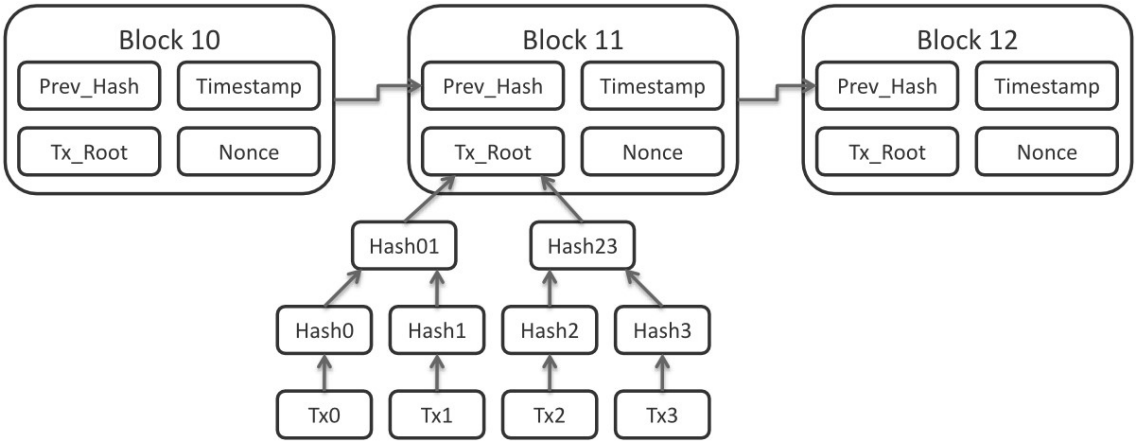


Ilustración 2: Matthäus Wander (2013). Graphic of data fields in Bitcoin block chain [Figura]. Obtenido de https://commons.wikimedia.org/wiki/File:Bitcoin_Block_Data.png

Podemos dividir el bloque en dos partes:

- **La cabecera (Block Header)**: contiene la información importante de cara a mantener la gestión de la Blockchain.
- **Lista de transacciones**: contiene las transacciones que han ocurrido en la red.

La cabecera, a su vez, contiene la siguiente información:

- **Versión:** La versión de la Blockchain a la cual corresponde este bloque

- **Timestamp:** La fecha y hora actual, en forma de Timestamp
- **Raíz de Merkle de las transacciones (Merkle Root):** Este hash nos sirve para comprobar rápidamente si una transacción está incluida en este bloque
- **Hash del bloque anterior:** Este hash nos permite enlazar los bloques entre sí, además de fortalecer la seguridad de la red. Este funcionamiento se explica a continuación.
- **Nonce:** El valor de este campo es el que permitirá al minero "cerrar" el bloque y establecer el "Proof of Work" necesario para obtener la recompensa de la red. Este concepto se explica en el apartado de minado de bloques. Este campo puede no estar presente si el método de minado es diferente al "Proof of Work".

Además de esta información, los bloques pueden contener otro tipo de datos que sea necesario para la gestión de la red, según la Blockchain que se esté utilizando.

3.1.2 Minado de bloques

En Blockchain, el minado de bloques se refiere a la operación de agregar nuevos bloques a la cadena de bloques mediante la resolución de un problema matemático de alta dificultad. Este proceso es realizado por nodos especiales de la red, llamados **mineros**.

El proceso de minado de bloques es el siguiente:

1. Se eligen las transacciones a añadir a la Blockchain de una "pool" de transacciones, un espacio de la red reservado para las transacciones que aún no han sido añadidas a la cadena. Las transacciones a añadir deben ser transacciones válidas.
2. Con las transacciones escogidas, se genera el nuevo bloque completando los datos que vimos en la sección anterior, a excepción del "nonce".
3. Se calcula el "nonce" del bloque mediante una operación matemática de alta dificultad cuya solución sirva como prueba de minado o "Proof of Work". Este paso se explica a continuación.
4. Se envía el nuevo bloque completo a la red.

Como ejemplo, en la red de Bitcoin, la operación utilizada en el paso 3 consiste en encontrar una cadena de caracteres alfanumérica tal que, al calcular el hash del bloque que se está minando, éste comience por una cantidad determinada de ceros (0). El hash es una operación que sólo se puede realizar probando diferentes combinaciones de caracteres, sin poder interactuar con otra variable de la función. Por lo tanto, el coste computacional de esta operación es lo suficientemente alto como para que el proceso de minado de bloques suponga un reto a la altura de su recompensa: ser el nodo elegido para añadir el siguiente bloque de la cadena y obtener como pago las tasas de todas las transacciones incluidas en ese bloque.

Cabe destacar que este proceso de consenso no es el que se sigue en todas las Blockchain. Además del "Proof of Work" aquí explicado, existen otros algoritmos de consenso como el "Proof of Stake", mediante el cual, el siguiente bloque de la cadena se elige aplicando varios criterios de selección aleatorias, o el "Proof of Authority", mediante el cual, el siguiente bloque de la cadena sólo puede ser validado por ciertos nodos validadores.

3.1.3 Arquitectura de una red Blockchain

Como se comentó en la introducción de este apartado, una Blockchain es una base de datos distribuida. Esto quiere decir que cada nodo de la red contiene toda la información disponible de la Blockchain.

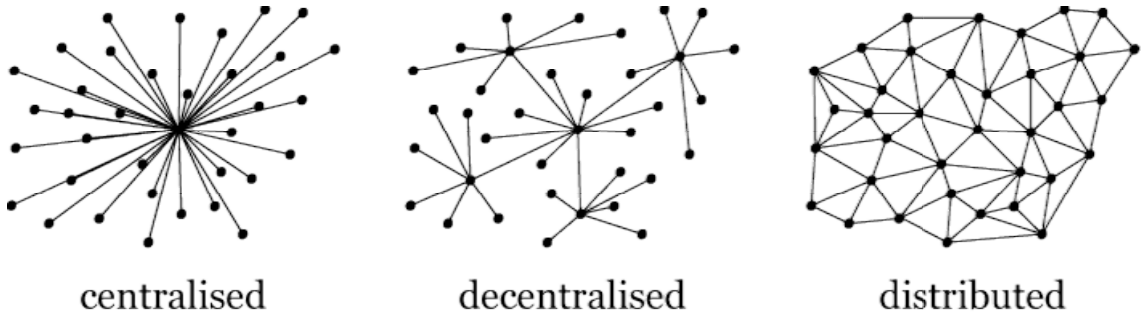


Ilustración 3: 1983~enwiki (2007). Centralised, decentralised, distributed [Figura]. Obtenido en https://commons.wikimedia.org/wiki/File:Centralised-decentralised-distributed.png

La capacidad de ser una red distribuida permite que la información esté disponible prácticamente a todas horas y que, si un nodo se desconecta, la red no se vea afectada ya que toda la información se encuentra en el resto de nodos.

Sin embargo, la dificultad que presenta esta ventaja es que se ha de llevar a cabo un consenso entre todos los nodos para añadir nueva información a la red. En una Blockchain, la nueva información a introducir aparece en forma de bloques, es decir, se seleccionan las nuevas transacciones a agregar a la red y se agrupan completando un bloque. Este bloque, tras realizar las operaciones necesarias, las cuales se han detallado en la sección de "Minado de bloques", es agregado a la cadena de bloques y puesto a disposición del resto de nodos para su verificación.

Otra cualidad de la red distribuida es que, como todos los nodos contienen la información de la red, ésta es completamente **transparente** puesto que cualquier nodo puede acceder a la información que contiene.

Por último, cabe destacar que, aunque originalmente la arquitectura de la Blockchain es distribuida, existen otras Blockchain con una arquitectura descentralizada e incluso en ocasiones, centralizada. Podemos ver representaciones de estas arquitecturas en la Ilustración 3.

3.1.4 Participar en la red de blockchain

Se analiza ahora el proceso de participación en una red de blockchain. En el proceso de alta en la red, el nuevo usuario crea un par de claves necesarias para interactuar en ella: una clave privada y una clave pública.

La clave privada es una cadena de caracteres utilizada para firmar digitalmente las transacciones a realizar, las cuales pueden ser verificadas utilizando la clave pública. De esta manera, cuando el usuario realiza una transacción y la firma digitalmente, cualquier usuario de la red puede verificar que el usuario que la ha realizado es efectivamente quien dice ser utilizando la clave pública para comprobar su contenido. Por lo tanto, la clave privada es la verificación última de la propiedad de un "activo" en la red. Si un usuario pierde su clave privada, no será capaz de firmar ninguna transacción y, consecuentemente, perderá todo acceso a los activos que pudiera tener.

La clave pública es el identificador del usuario en toda la red y es única para cada usuario. Se trata también de una cadena de caracteres que, además de verificar que un usuario es el emisor real de una transacción, es la base que se utiliza para obtener la dirección del receptor de una transacción. Esta dirección se conoce también como la dirección de la "cartera" del usuario en la red.

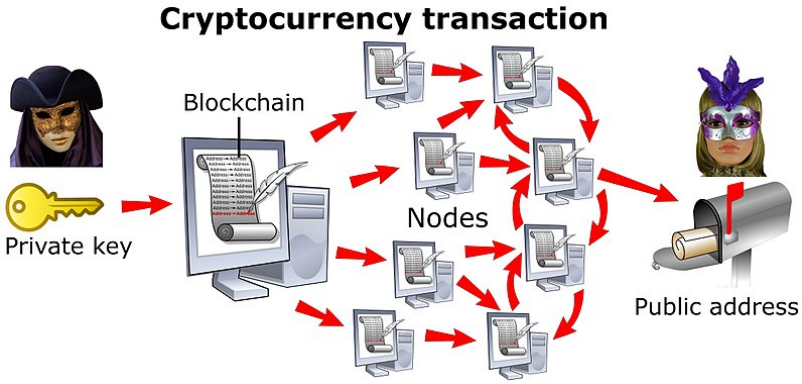


Ilustración 4: Mikael Häggström (2018). Simplified figurative process of a Cryptocurrency transaction [Figure]. Obtenido de https://en.wikipedia.org/wiki/File:Cryptocurrency_transaction.jpg

Pongamos ahora un ejemplo de cómo se realiza una transacción en la red. Supongamos que el usuario A quiere enviar ciertos activos al usuario B. El usuario A define los parámetros de la transacción:

- Cantidad de activos
- Dirección a la que enviarla, en este caso la dirección de la cartera del usuario B
- Tasas de la transferencia. Éstas son las tasas que se transfieren al minero que añadirá el bloque que contenga esta transferencia a la cadena de bloques.

Tras definir los parámetros, el usuario A firma digitalmente la transacción utilizando su clave privada, y ésta es enviada a la "pool" de transacciones que se mencionó en la sección de "Minado de bloques".

Una vez la transacción es escogida para introducirse en un bloque, ésta es verificada por el minero que la esté tratando mediante la clave pública del usuario A. Tras la verificación, la transacción es finalmente introducida en el bloque y éste en la cadena de bloques, pasando el usuario B a ser el nuevo poseedor oficial de los activos transferidos.

Por último, cabe destacar que la participación en la red de blockchain puede ser completamente **anónima**, ya que el resto de participantes solamente necesita conocer la clave pública del usuario.

3.1.5 Seguridad de la blockchain

Las redes blockchain contemplan varios niveles de seguridad que permiten que las operaciones que se realicen en ella se ejecuten correctamente y que la información contenida en toda la red sea fiable y verificable.

En primer lugar, como se comentaba anteriormente, los bloques están enlazados entre sí cronológicamente. Este enlace se realiza incluyendo en el bloque el hash[2] del bloque anterior, lo cual, además, aumenta la seguridad de la cadena de bloques ya que evita que se modifiquen los registros de bloques anteriores puesto que esto modificaría el hash resultante del bloque, modificando a su vez, los hashes de los bloques posteriores.

Es decir, si se modifica el contenido del bloque n-3, su hash cambiaría, con lo que habría que cambiar el hash enlazado que se ha añadido en el bloque n-2. Esto provocaría que el hash del propio bloque n-2 cambiara, con lo que habría que realizar la misma operación en el bloque n-1. Y así sucesivamente hasta el bloque actual.

Dado que la Blockchain está distribuida en una red con numerosos nodos, esta operación se tendría que efectuar en todos los nodos, lo que resulta prácticamente imposible. Ésta es una de las medidas de seguridad que garantiza la **inmutabilidad de la información** de la blockchain.

En segundo lugar, todas las transacciones que se añaden a los bloques deben ser verificadas un número determinado de veces antes de darse por válidas. Estas verificaciones se realizan añadiendo más bloques a la cadena. Es decir, si se añade una transacción al nuevo bloque n, ésta tendrá una confirmación. Al añadir el bloque n+1, la transacción tendrá 2 confirmaciones. Y así sucesivamente. Ya que cada minero debe validar las transacciones de su bloque, y el minero del siguiente bloque tiene en cuenta las transacciones del bloque anterior, se hace imposible utilizar ataques de doble gasto o incluir transacciones no válidas en la cadena.

3.2 Blockchain de primera generación: Bitcoin

En 2009, apareció publicado el White paper[3] "Bitcoin: A Peer-to-Peer Electronic Cash System", firmado por Satoshi Nakamoto, definiendo por primera vez la tecnología Blockchain aplicada a la infraestructura de un sistema monetario electrónico distribuido y seguro, la red de Bitcoin (Nakamoto, 2009).

El objetivo principal de la red de Bitcoin es el de ofrecer un sistema electrónico de pago, seguro y transparente, que no necesita de una autoridad central para ser regulado y que mantiene el anonimato de los usuarios. Todas estas características han sido ya descritas en el apartado anterior.

[2] El "hash" es el resultado de aplicar una función específica a toda la información contenida en el bloque, de manera que se obtiene una serie de caracteres alfanuméricos de un tamaño predeterminado.

[3] Un White paper, o "Libro blanco" es un documento que generalmente ayuda a entender un tema en particular, a tomar decisiones frente a un problema concreto, o que propone soluciones a un problema.

Ilustración 5: Credit: BeatingBetting.co.uk (2017). Bitcoin vs Banks & Centralization [Figure]. Obtenido en https://www.flickr.com/photos/159526894@N02/37499031504

A nivel técnico, la red de Bitcoin funciona tal y como se ha descrito anteriormente, formando los bloques con la estructura predeterminada, almacenando las operaciones de transferencia de Bitcoins entre los distintos participantes de la red y utilizando el algoritmo de consenso "Proof-of-work".

Las dos maneras principales de obtener Bitcoins son:

- Minado de bloques: tras minar un bloque, se obtiene, como recompensa, las tasas en Bitcoin de todas las transacciones incluidas en el bloque.
- Compra en un mercado secundario: existe una gran cantidad de mercados secundarios en los que se puede comprar y vender todo tipo de criptoactivos, entre ellos, Bitcoin. Ejemplos de este tipo de mercados son Coinbase (Coinbase) o Binance (Binance).

Dado que para minar bloques de la red de Bitcoin es necesario poseer una gran potencia computacional (Hayes, 2019), la mayoría de los nuevos usuarios acceden a la red mediante un mercado secundario. Para garantizar la seguridad de las transacciones y, en ocasiones, a efectos regulatorios (Hidalgo, 2019), estos mercados necesitan identificar a los usuarios que van a participar en la red a través de ellos. Esto provoca que se pierda uno de los aspectos fundamentales de la blockchain: el anonimato. Explicaremos más en detalle este problema en la sección "La confianza y el anonimato en la red de Blockchain".

Con todo, Bitcoin ha supuesto un antes y un después a nivel tecnológico y a nivel financiero. La tecnología subyacente es actualmente objeto de estudio en numerosos sectores, y permitirá mejorar numerosos procesos industriales. A nivel financiero, ha supuesto un buen medio de inversión y ha sentado las bases de una alternativa al sistema económico tradicional.

3.3 Blockchain de segunda generación: Ethereum

En 2014, Vitalik Buterin publicó el trabajo que había desarrollado aplicando nuevas funcionalidades a la infraestructura Blockchain (Buterin, 2014), entre ellas, la capacidad de almacenar contratos inteligentes en una Blockchain que se ejecutaran utilizando cierta cantidad de una criptomoneda y almacenara los resultados en la propia Blockchain. De esta manera conseguía crear un sistema abierto, para que cualquier persona programara sus contratos inteligentes y pudiera llevar un registro transparente de las transacciones de ese contrato.

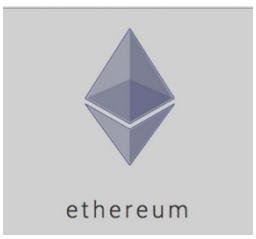


Ilustración 6: Marc van der Chijs (2014). Ethereum logo [Figura]. Obtenido en https://www.flickr.com/photos/chijs/14552266437

3.3.1 Estructura de bloques de Ethereum

Buterin cambió la estructura de la blockchain definida por Nakamoto permitiendo almacenar, no sólo transacciones entre usuarios, sino también estados de los distintos participantes de la red. Además, añadió el concepto de "Smart Contract" como participante de la red, de manera que los desarrolladores pudieran crear pequeños trozos de código que ejecutaran ciertas acciones al recibir transacciones con determinados parámetros.

Por lo tanto, con su nueva estructura, la red Ethereum distingue dos tipos de participantes en la red. Por un lado, las **Cuentas de Usuario** (Externally Owned Accounts), que funcionan de manera similar a una cuenta en la red de Bitcoin, formada por una clave pública para identificar la cuenta y una clave privada para firmar las transacciones realizadas.

Por otro lado, Ethereum permite alojar trozos de código, denominados **Smart Contracts**, desarrollados por cualquier usuario, en un espacio similar al de los Usuarios en la red. Los Smart Contract son un tipo especial de participante que ofrece cierto comportamiento adicional cuando recibe transacciones.

Ambos tipos de participante comparten el mismo espacio en la red y son gestionados de manera muy similar. Los estados de las Cuentas de Usuario determinan el saldo de la cuenta entre otros, y los estados de un Smart Contract determinan la estructura actual del contenido que pueda alojar.

Toda esta información se almacena en la blockchain utilizando raíces de árboles de Merkle, específicamente cuatro:

- **State Root**: es la raíz de Merkle del estado global de todos los participantes de la red, entendiendo por estado lo que se ha definido en los párrafos anteriores.
- **Storage Root**: se almacena en el "estado" de un Smart Contract y contiene la raíz del árbol de Merkle que almacena el código del contrato y las estructuras de datos necesarias.
- **Transaction Root**: es la raíz de Merkle de todas las transacciones almacenadas en el bloque.
- **Receipt Root**: es la raíz de Merkle del resultado de la ejecución de las transacciones almacenadas en el bloque.

3.3.2 Funcionamiento de la red y ejecución de los Smart Contracts

En la red Ethereum, el minado de bloques se hace de manera similar a la red Bitcoin, utilizando el algoritmo de consenso "Proof-of-work". Sin embargo, El tamaño del bloque es mucho menor en la red, por lo que el proceso de minado se realiza mucho más rápidamente que en la red Bitcoin: 1 bloque cada 14 segundos aproximadamente frente a los 10 minutos de Bitcoin.

Ethereum, además, añade un procedimiento más al minado de bloques. Los mineros no sólo tienen que obtener las transacciones del espacio de transacciones y empaquetarlas en el bloque, sino que además tienen que ejecutar el código correspondiente a las transacciones que llaman a un Smart Contract.

La ejecución de un Smart Contract conlleva, por un lado, un cambio en el estado del Smart Contract, que se almacena en el bloque de la misma manera que el cambio de estado de una cuenta de Usuario. Por otro lado, la ejecución también conlleva una utilización de recursos computacionales para determinar el resultado de la función llamada. Estos recursos se añaden a los que se consumen para realizar la "Proof-of-work", con lo que supone un gasto extra para el minero. Para paliar esta situación, la red Ethereum realiza un cálculo de los recursos computacionales a utilizar por la ejecución de cada funcionalidad del Smart Contract y

determina el resultado en unidades de **Gas**. Este Gas debe ser pagado por el usuario que realiza la transacción, de manera que el minero tenga un incentivo extra para ejecutar el código correspondiente.

3.3.3 Dapps y la versatilidad de la red Ethereum

Con la aparición de los Smart Contracts, la red Ethereum ofrece un espacio abierto para la creación de aplicaciones distribuidas o Dapps, por sus siglas en inglés. Estas Dapps ofrecen una funcionalidad determinada según el Smart Contract desarrollado, pudiendo ser aplicado a una inmensa cantidad de escenarios.

Un ejemplo de puede ser la aplicación del Smart Contract para el seguimiento y monitorización de la producción y distribución de materias primas de alimentación. Cada "materia prima" puede representarse en el Smart Contract con un **token** específico e inmutable, que va pasando por diferentes estados según el lugar en el que se encuentre en la vida real: creado, recolectado, en almacén, en ruta 1, en ruta 2, etc. Ya que las operaciones almacenadas son inmutables, en todo momento se puede saber en qué estado está un token en particular, para determinar si ha ocurrido algo con la materia prima que representa, o para mejorar los flujos internos de producción/distribución (Thomasson, Carrefour says blockchain tracking boosting sales of some products, 2019).

Otro ejemplo puede ser la intermediación en el sector de los seguros. Axa desarrolló una Dapp mediante la cual, los usuarios que compraran un billete de avión y contrataran este seguro, podían, automáticamente, ser indemnizados si el vuelo se retrasaba un determinado tiempo. El Smart Contract desarrollado permitía crear un token con la información de cada usuario y vuelo y, en tiempo real, determinar si un vuelo se había retrasado el tiempo indicado, indemnizando así los tokens afectados (Fintech, 2017).

En definitiva, la ventaja principal de utilizar la tecnología de Ethereum y, en particular, la creación de Dapps es que, de manera muy personalizada, se puede hacer uso de una infraestructura distribuida, muy barata, a modo de base de datos que ofrece las características de toda Blockchain: transparencia, inmutabilidad y seguridad.

3.4 Blockchain de segunda generación: Hyperledger Fabric

Hyperledger Fabric nació en 2016 como resultado del trabajo de la Linux Foundation junto a un grupo de 30 empresas, bajo el proyecto Hyperledger. Este proyecto estaba enfocado en desarrollar herramientas relacionadas con la tecnología blockchain para un ámbito más empresarial. Empresas como IBM, Intel y Accenture fueron las primeras en formar parte de este proyecto Hyperledger y participar en la creación de su primera blockchain: Hyperledger Fabric (Hyperledger, 2016).

La filosofía de Hyperledger Fabric es diferente a la de Ethereum. En este caso, la blockchain se estructura en nodos con diferentes roles y permisos, y está pensada para ser implantada en un entorno cerrado y corporativo, generalmente como parte de un grupo de empresas que les permita compartir una base de datos compartida con las ventajas de blockchain (inmutable, transparente y segura).

Una instancia de Hyperledger Fabric está formada por **miembros**, los cuales son responsables de **dar de alta sus nodos** para participar en la red. Cada nodo debe configurarse mediante los certificados apropiados y se les debe asignar los permisos correspondientes a la responsabilidad que tendrán en la blockchain: qué acceso tendrán a qué funcionalidades, qué información son capaces de ver, etc. Además, los miembros de la misma red pueden definir

como públicos sólo los elementos que ellos deseen, manteniendo así la privacidad de ciertas informaciones que no deseen exponer al resto de miembros de la red, o definir sólo qué nodos tienen acceso a cierta información.

Mientras que la unidad principal que se gestiona en Bitcoin es el propio Bitcoin y en Ethereum, el Ether, en Hyperledger Fabric no existe una unidad principal común, sino que se representan directamente activos que se pueden intercambiar. Estos activos también son definidos por los miembros de la red.

Para interactuar con la blockchain, se utilizan Smart Contracts escritos en Chaincode, código implementado en Go o Node, que permiten definir activos y funciones para manejar esos activos dentro de la blockchain.

Otra de las principales diferencias entre Ethereum y Hyperledger Fabric es que, mientras que en el primero todos los nodos de la red son iguales, en el segundo existen dos tipos distintos de nodos: **Nodos Peer** y **Nodos de Órdenes**. Los Nodos Peer son los encargados de ejecutar y verificar las transacciones, mientras que los Nodos de Órdenes son los responsables de enviar las órdenes de ejecución a la red y de propagar las transacciones entre el resto de nodos. De esta manera se dividen las principales tareas que se llevan a cabo en la Blockchain. Por ejemplo, el algoritmo de consenso se aplica únicamente entre los nodos de órdenes.

Además, en Hyperledger Fabric se distinguen dos estructuras de datos diferentes: el **Log de Blockchain**, que almacena la secuencia de registros de transacciones en los bloques (la propia blockchain), y una **base de datos de estado**, que almacena el estado actual de la blockchain. De esta manera, se utiliza la eficiencia de una base de datos para obtener el estado de un activo en particular, pero a la vez se aprovecha la ventaja de la blockchain para conocer el histórico de un activo.

3.5 La confianza y el anonimato en la red de Blockchain

Tras haber estudiado la tecnología de Blockchain y las principales redes existentes, en esta sección se detalla el problema existente con la confianza entre usuarios de la red. Dejaremos la red de Hyperledger Fabric fuera de la discusión, ya que se trata de una blockchain de un tipo especial en el que todos sus nodos son identificados al crearse. Por lo tanto, hablaremos únicamente de redes públicas.

Como se ha visto en el análisis, el único requisito para participar en una red pública de Blockchain es poseer un par de claves privada y pública. Esta característica permite que cualquier usuario pueda participar en una red de Blockchain y, además, conserve su anonimato. En redes de criptodivisas, como puede ser Bitcoin, la capacidad de guardar el anonimato de un usuario de la red es más que deseable, ya que hasta ahora no ha existido ningún sistema de pago que no obligue a los usuarios a identificarse a partir de cierto límite.

Sin embargo, hay que tener en cuenta que la razón principal por la cual se identifican a los usuarios de una red de pagos es para controlar el flujo de dinero en la red y evitar la estafa y el fraude.

Ilustración 7: Aaron Olson (2018). Dark Blockchain Crypto Digital [Figura]. Obtenido en https://pixabay.com/photos/dark-blockchain-crypto-digital-3123582/

Este mismo problema se da, fundamentalmente, en las redes de Blockchain. Al garantizar el anonimato de los usuarios, también se está abriendo la puerta a usuarios maliciosos que puedan desarrollar diferentes técnicas para engañar al resto de usuarios y conseguir que les transfieran sus activos.

Para solucionar este problema, generalmente se utilizan terceros que intermedien la transacción. Un ejemplo de intermediarios pueden ser los mercados secundarios de criptoactivos (Coinbase, Binance,…). Estos mercados obligan a los usuarios a identificarse de alguna manera para poder participar en la red a través de ellos, con lo que los participantes cuentan con una confianza extra a la hora de transferir sus activos a otro usuario. Otro ejemplo de intermediarios, ya utilizando Smart Contracts, pueden ser las entidades que hacen de escrow y capturan los activos a transferir hasta que la otra parte haya enviado el pago necesario (EscrowMyEther, 2019).

Por lo tanto, podemos concluir que, aunque el anonimato constituye una característica fundamental de la Blockchain, a nivel práctico se pierde por la falta de confianza entre los usuarios de la red.

4 Descripción y diseño de la solución a desarrollar

En este capítulo se describe la solución a desarrollar y se establecen las funcionalidades que debe implementar así como la tipología de usuarios que interactuará con ella. Por otro lado, se definen las tecnologías que se utilizan, además de hacer hincapié en el núcleo central de la solución a desarrollar: el algoritmo de confianza.

4.1 Análisis funcional

El elemento principal del Sistema a desarrollar es un Smart Contract que ofrece las funcionalidades de gestión de confianza de los usuarios de la red. Como elemento de apoyo, existirá un front-end que permitirá visualizar el contenido del Smart Contract.

Se detalla a continuación la lista de funcionalidades que debe permitir el Smart Contract:

- **Valorar a un usuario:** cualquier usuario debe poder llamar a una función del Smart Contract que, pasando como parámetros la dirección de otro usuario y una valoración entre 1 (muy baja) y 3 (muy alta), otorgue esa valoración al usuario definido. El Smart Contract debe guardar tanto la valoración como el usuario que la ha emitido.
- **Añadir información extra de un usuario:** un operador del Smart Contract debe poder llamar a una o varias funciones que, pasando como parámetro la dirección del usuario y cierta información extra, incluya esa información en el Smart Contract de manera que sirva para mejorar el algoritmo de confianza de un usuario. Esta información extra puede ser, por ejemplo, el número de transacciones que un usuario ha realizado/recibido. El Smart Contract debe guardar esta información extra.
- **Obtener el Score de Confianza general de un usuario**: cualquier usuario o Smart Contract debe poder llamar a una función que, pasando como parámetro la dirección del usuario del que se quiere realizar la consulta, devuelva el resultado de aplicar el algoritmo de confianza al usuario.
- **Obtener el Score de Confianza relativo de un usuario:** cualquier usuario debe poder llamar a una función que, pasando como parámetro la dirección del usuario del que se quiere realizar la consulta, y teniendo en cuenta su propia dirección, devuelva el Score de Confianza basado en su valoración anterior.
- **Gestionar los operadores del Smart Contract**: Tanto el propietario como los operadores del Smart Contract deben ser capaces de llamar a varias funciones que, pasando como parámetro la dirección de un usuario o Smart Contract, añadan o eliminen al usuario como operador del Smart Contract.

Se describe a continuación el tipo de usuarios que utilizará la solución:

- **Usuarios de la red:** Estos son todos los usuarios de la red donde será desplegada la solución. Estos usuarios serán capaces de valorar y de obtener el Score de Confianza general y relativo de otros usuarios.
- **Smart Contracts:** Otros Smart Contracts de la red donde será desplegada la solución, podrán obtener el Score de confianza general de los usuarios de la red.
- **Operadores del Smart Contract:** Estos son usuarios o Smart Contracts de la red donde será desplegada la solución y que tendrán acceso a funciones para añadir información extra útil en el cálculo del Score de Confianza de un usuario: número de transacciones

realizadas, número de transacciones recibidas, etc. Además, podrán gestionar los Operadores del Smart Contract, añadiendo y eliminando Operadores del mismo.

- **Propietario del Smart Contract:** Es el usuario que despliega el Smart Contract en la red. Es el primer usuario en ser declarado como Operador del Smart Contract.

4.2 El algoritmo de confianza

El núcleo de la solución a desarrollar es el algoritmo que calcula el Score de confianza de un usuario. Este algoritmo se basa en múltiples parámetros que permiten, de manera objetiva, determinar cuán fiable es un usuario de la red. A la hora de realizar una transacción, esta información ayuda a que ambos usuarios tengan más información sobre el otro y disminuya la barrera de desconfianza que existe en este tipo de transacciones. Además, este algoritmo debe ser capaz de detectar y "castigar" los nodos maliciosos de la red, otorgándoles un Score de Confianza lo más bajo posible.

Como los desarrollos en Ethereum no permiten realizar cálculos muy complejos dado el coste en gas que eso conlleva, definiremos un algoritmo sencillo que sirva como primera solución para el problema establecido, pero con margen de mejora para ir evolucionando el algoritmo según vaya avanzando la capacidad tecnológica de Blockchain en general.

4.2.1 Descripción

En esta primera versión del algoritmo, el Smart Contract cuenta con tres estructuras de datos:

- Valoraciones (ratings) que cada usuario a otorgado a otros usuarios
- Red de confianza
- Información extra de los usuarios

Los usuarios son capaces de valorar otro usuario otorgándoles una puntuación entre 1 (muy mala) y 3 (muy buena). Estas valoraciones se van almacenando por separado en el Smart Contract.

Si un usuario valora a otro con una puntuación de 3, este usuario entra a formar parte de la red de confianza del usuario valorador, información que se almacena en forma de grafo en el Smart Contract.

Si un usuario valora a otro con una puntuación de 1, se penaliza a la red de confianza del usuario valorado, otorgándoles otra puntuación de 1 a los usuarios de máxima confianza del usuario valorado, los de primer nivel en su red de confianza.

Así, para obtener el Score de Confianza general de un usuario, se obtiene la media de todas sus valoraciones, y para obtener el Score de Confianza relativo de un usuario, se obtiene la valoración que un usuario haya dado a otro.

La función principal que devuelve el Score de Confianza de un usuario es una mezcla de las dos operaciones anteriores, de manera que primero se busca el score de confianza relativo, si no existe, se busca si el usuario del cual se está haciendo la consulta forma parte de la red de confianza del usuario que realiza la consulta y, si tampoco se encuentra en esa red, se obtiene el score de confianza general del usuario objetivo.

Como se puede observar, en esta primera prueba de concepto utilizaremos únicamente las dos primeras estructuras de datos para calcular el Score de Confianza de un usuario. Dejaremos la información extra de los usuarios para futuras versiones del algoritmo. Además, la búsqueda

en la red de confianza la limitaremos a una búsqueda recursiva de dos niveles para aligerar la carga de procesamiento del Smart Contract.

4.2.2 Justificación

La razón por la que se ha elegido una puntuación de 1, 2 y 3 es simplificar el proceso de valoración para no entrar en cálculos muy complejos y evitar que el coste en gas del Smart Contract se incremente demasiado. Así, el significado de cada valoración y el impacto que tiene en el usuario es diferente en cada caso:

- **Valoración 1:** ha habido un problema con la transacción. El usuario que ha recibido esta puntuación no ha cumplido su parte del trato y debe ser penalizado en la red. El usuario que ha dado esta valoración probablemente no vuelva a confiar en el otro usuario para realizar otra transacción.
- **Valoración 2:** la transacción se ha realizado correctamente, aunque no en los términos deseados. Ningún usuario es penalizado, pero el usuario valorador probablemente sea reticente a realizar otra transacción con el usuario valorado.
- **Valoración 3:** la transacción se ha realizado en los mejores términos, y el usuario valorado entra a formar parte de la red de confianza del usuario valorador. Probablemente ambos usuarios realicen nuevas transacciones entre ellos en el futuro.

4.2.3 Las alternativas a este sistema de valoración eran:

- **Mantener un sistema binario (0, 1)**: aunque este sistema puede ser más eficiente, puede resultar muy agresivo para los usuarios que, aunque no en los mejores términos, hayan realizado su parte del trato. En un entorno en el que el resultado de las transacciones depende de terceros (los mineros), el usuario que haya sido perjudicado debido a la red y no a él mismo podría ver descender su Score. Asimismo, el usuario valorador podría no querer confiar plenamente en este usuario. El resultado sería, probablemente, la no-valoración de la transacción, escenario que se desea evitar lo máximo posible. Introduciendo la valoración media en la ecuación solucionamos este tipo de casos.
- **Ampliar el rango de valoración (por ejemplo, entre 1 y 5)**: este sistema complica los cálculos y añade nuevos interrogantes al algoritmo, esencialmente, ¿qué significan las puntuaciones 2 y 4? Encontrar la respuesta a esta pregunta puede ser complicada y, aunque se encuentre y se establezca un buen sistema, la labor de comunicación al usuario se hace difícil, con lo que pueden llegar a plantearse muchas ambigüedades desde el punto de vista del usuario valorador. Para evitar este comportamiento, se simplifica el rango de valoración para que cada valoración sea fácil de entender.

Finalmente, la razón por la que se penaliza la red de confianza cuando un usuario recibe una valoración de 1, es evitar que se creen redes de nodos maliciosos que se mejoren la puntuación entre ellos. En la red de Ethereum es muy sencillo generar un número n de usuarios que se realicen transacciones entre ellos y que mejoren su puntuación. Penalizando su red de confianza se va desarmando la red poco a poco, hasta que todos los nodos que han participado en la subida de las valoraciones de los nodos maliciosos reciben las valoraciones negativas.

4.3 Tecnologías a utilizar

Se describen a continuación las tecnologías que se utilizarán en el desarrollo de la solución.

4.3.1 Blockchain Ethereum

La blockchain sobre la que desplegaremos la solución será la de Ethereum. Ethereum permite llegar a un gran número de usuarios que pueden beneficiarse de esta solución (a día de hoy, en la red principal, existen casi 80 millones de cuentas creadas) y cuenta con muchísima actividad (unas 750.000 transacciones diarias).

Además, Ethereum es la base de un gran número de Smart Contracts que definen sus propios tokens, los cuales representan activos de diferente tipo. La comunidad de desarrolladores de Ethereum sigue desarrollando estándares de Smart Contract que ayudan a definir nuevos y diversos comportamientos para poder aplicarlos en diferentes escenarios.

Por último, la blockchain de Ethereum se utiliza en diferentes entornos: desde la red principal, o mainnet, hasta entornos de testing más conocidos como Rinkeby, Ropsten o Kovan.

Nuestra solución puede representar una buena oportunidad para mejorar las transacciones de todos los usuarios de la red, ya estén comerciando con Ether directamente o con cualquier otro token definido en un Smart Contract.

4.3.2 Solidity

Solidity es el lenguaje de programación utilizado para desarrollar el Smart Contract sobre Ethereum. Aunque existen otras alternativas, Solidity es el que mejor trayectoria lleva de entre los lenguajes disponibles: Serpent y Mutan, ambos desfasados; LLL, que trabaja a muy bajo nivel; y Vyper, que aún está en desarrollo.

Solidity es orientado a objetos, de tipado estático y soporta herencia y tipos de datos complejos (como *structs*). Cuando se compila, produce un *bytecode* que se puede ejecutar en la Ethereum Virtual Machine (EVM).

Finalmente, una de las mayores ventajas de utilizar Solidity es que existen numerosas herramientas desarrolladas para trabajar con Solidity como Truffle, la cual veremos a continuación.

4.3.3 Framework Truffle

Truffle es un entorno de desarrollo para Smart Contracts en Solidity que permite, entre otras funcionalidades, construir rápidamente directorios de desarrollo para Dapps, utilizar herramientas de testing para el Smart Contract, mejorar la compilación del Smart Contract y desplegarlo en la red de Ethereum elegida.

Truffle forma parte del conjunto de herramientas Truffle Suite, el cual contiene otras herramientas que facilitan la labor de crear Smart Contracts y Dapps, y de construir equipos para trabajar en conjunto en este tipo de desarrollos.

4.3.4 Ganache

Ganache es otra herramienta que forma parte del conjunto de herramientas Truffle Suite. El papel de Ganache es el de ofrecer la estructura de una blockchain para poder testear el Smart Contract o Dapp que se esté desarrollando, en un entorno lo más similar posible al entorno final donde se desplegará la solución.

Ganache inicia una blockchain Ethereum en local y crea un número determinado de direcciones con Ether, con las que interactuar en la blockchain. Además, lleva un registro de todas las transacciones que ocurren en la blockchain, mientras realiza un proceso simplificado del minado de bloques.

Las pruebas de test que realiza Truffle suelen llevarse a cabo en una instancia de Ganache.

4.3.5 Node Package Manager (npm)

Node Package Manager (npm) es un gestor de paquetes de Javascript, que permite gestionar los diferentes paquetes a utilizar en una aplicación con base de Javascript. La gran mayoría de Dapps que basan su front-end en Javascript utilizan NPM para instalar todas las dependencias necesarias en el desarrollo: Truffle, react, web3,…

En este proyecto, NPM se utiliza para gestionar las dependencias de la Dapp a crear, además de para instalar el entorno inicial de desarrollo: Truffle.

4.3.6 Web3.js y Metamask

Web3.js es un conjunto de librerías que permiten conectar una interfaz en Javascript con un Smart Contract de la red Ethereum, utilizando una conexión HTTP o IPC.

Metamask es una extensión del navegador que permite conectar una cuenta de la red Ethereum (importando una clave privada o creando una nueva cuenta) para poder utilizarla en una Dapp accesible desde el navegador.

En la construcción de una Dapp, Web3.js se suele utilizar para crear un Bridge que permita llamar a las funciones públicas del Smart Contract desde el front-end de la Dapp, teniendo la cuenta conectada en Metamask como cuenta desde la cual se hace la llamada al Smart Contract.

4.4 Estructura del Smart Contract y funciones disponibles

A continuación se describe cómo será el Smart Contract y las funciones que debe tener disponibles para el correcto funcionamiento de la solución a desarrollar.

```
contract DTrustManager
{
    //
    // Eventos
    //
    event Rating(address indexed from, address indexed to, uint8 indexed score);

    //
    // Estructuras de datos
    //
    // Valoraciones de un usuario
    mapping(address => uint8[]) private _ratings;

    // Valoraciones otorgadas por cada usuario
    mapping(address => mapping(address => uint8)) private _scores;
```

```
    // Grafo representativo de la red de confianza
    mapping(address => address[]) private _trust_matrix;

    // Información extra (esta estructura podría alojar más información)
    struct extraInformation
    {
        uint256 network_transactions;
    }
    mapping(address => extraInformation) _user_extra_information;

    //
    // Funciones de gestión de operadores
    //
    /**
     * @dev Devuelve true si el usuario que llama a la función es Operador del
SmartContract
     */
    function isOperator() internal view returns (bool);

    /**
     * @dev Función de comprobación de si el usuario que llama a la función es
Operador del SmartContract
     */
    modifier onlyOperator();

    /**
     * @dev Añade un nuevo Operador al SmartContract
     */
    function addOperator(address operator) public onlyOperator;

    /**
     * @dev Elimina un Operador del SmartContract
     */
    function removeOperator(address operator) public onlyOperator;

    /**
    * @dev Añade el número de transacciones de un usuario como información extra que
puede ser utilizada en el algoritmo (Esta función sólo es accesible para los
Operadores del SmartContract)
    */
    function addUserExtraInformation(address user, uint256 number_of_transactions)
public onlyOperator;

    //
    // Funciones de gestión de la confianza
    //
    /**
    * @dev Otorga una valoración al usuario `user`
    */
    function rateUser(address user, uint8 score) public;

    /**
    * @dev Devuelve el Score de confianza general del usuario `user` de la red
    */
    function getGeneralTrustScore(address user) public view returns (uint8);

    /**
    * @dev Devuelve el Score de confianza relativo del usuario `user` según el
usuario `from`
    */
    function getRelativeTrustScore(address from, address user) public view returns
(uint8);

    /**
    * @dev Devuelve el Score de confianza relativo del usuario `user` según el
```

```
usuario `from` o el Score general si no existe relación entre ellos (La función
sobrecargada de un parámetro permite obtener la funcionalidad, suponiendo que `from`
es el usuario que llama a la función)
    */
    function getTrustScore(address from, address user) public view returns (uint8);
    function getTrustScore(address user) public view returns (uint8);

    /**
    * @dev Busca al usuario `target` en la red de confianza del usuario `user`
    */
    function _userIsInTrustNetwork(address user, address target) internal view
returns(bool);

    /**
    * @dev Castiga la red de confianza del usuario `user` (otorga una valoración de 0
a los usuarios en los que confía el usuario `user`)
    */
    function _punishNetwork(address) internal;
}
```

Como se especificó anteriormente, el Smart Contract almacenará cuatro estructuras de datos:

- Las **valoraciones (ratings) de cada usuario**, necesarias para obtener el Score de Confianza general del usuario.
- Una **matriz con cada valoración otorgada** por cada usuario a otros usuarios, necesaria para obtener el Score de confianza relativo de un usuario.
- Un **grafo de la red de confianza** de cada usuario, necesario para obtener la red de confianza de un usuario.
- La **información extra de cada usuario**, que podrá ser utilizada en el algoritmo en próximas versiones.

Las funciones que tendrá disponible para gestionar la confianza de los usuarios son:

- **rateUser**: que recibe como parámetros la dirección de un usuario y la valoración a otorgar. La función especificará como usuario valorador la dirección que ha llamado a la función. Además, emitirá el evento Rating.
- **getGeneralTrustScore**: que recibe como parámetro la dirección de un usuario. La función devolverá el Score de Confianza General que resulta de aplicar el algoritmo de confianza al usuario especificado.
- **getRelativeTrustScore**: que recibe como parámetros la dirección de dos usuarios, el usuario de quien se quiere obtener el Score de Confianza y el usuario de quien partimos para hacerlo. La función devolverá la valoración que el usuario haya otorgado al usuario de quien se quiere obtener el Score de Confianza o, en caso de no haberlo valorado anteriormente, una confianza máxima si el usuario forma parte de la red de confianza del usuario del cual partimos.
- **getTrustScore**: que recibe como parámetros la dirección de dos usuarios, el usuario de quien se quiere obtener el Score de Confianza y el usuario de quien partimos para hacerlo. La función devolverá el Score de Confianza relativo del usuario objetivo o, en caso de no haberlo valorado anteriormente, una confianza máxima si el usuario forma parte de la red de confianza del usuario del cual partimos. Si no existe relación entre ambos, esta función devuelve el Score de Confianza General del usuario objetivo.

Además, las funciones que tendrá disponible para los Operadores del SmartContract son:

- **addOperator**: que recibe como parámetro la dirección de un usuario y añade este usuario al conjunto de Operadores del SmartContract.
- **removeOperator**: que recibe como parámetro la dirección de un usuario y elimina este usuario del conjunto de Operadores del SmartContract.
- **addUserExtraInformation**: que recibe como parámetros la dirección de un usuario y el número de transacciones que ese usuario ha realizado en la red. Esta función añade el número de transacciones a la información extra del usuario, de manera que esta información pueda ser utilizada posteriormente en el algoritmo de confianza.

Las funciones a nivel interno son:

- **_punishNetwork**: que recibe como parámetro la dirección de un usuario. La función castiga la red de confianza del usuario especificado otorgándoles una valoración de 0 por parte del usuario que ha llamado a la función que llamará a ésta. Esta función se llama internamente desde rateUser.
- **_userIsInTrustNetork**: que recibe como parámetros la dirección de dos usuarios, el usuario cuya red de confianza se va a explorar, y el usuario que estamos buscando. Esta función recorre la red de confianza del usuario de base buscando el usuario objetivo para determinar si éste se encuentra en la red de confianza del primero.

Por último, el Smart Contract emite un evento **Rating** cada vez que un usuario otorga una valoración a otro.

4.5 Flujo de procesos del Sistema

En el gráfico siguiente se muestra el flujo que sigue el Sistema a la hora de realizar las diferentes acciones.

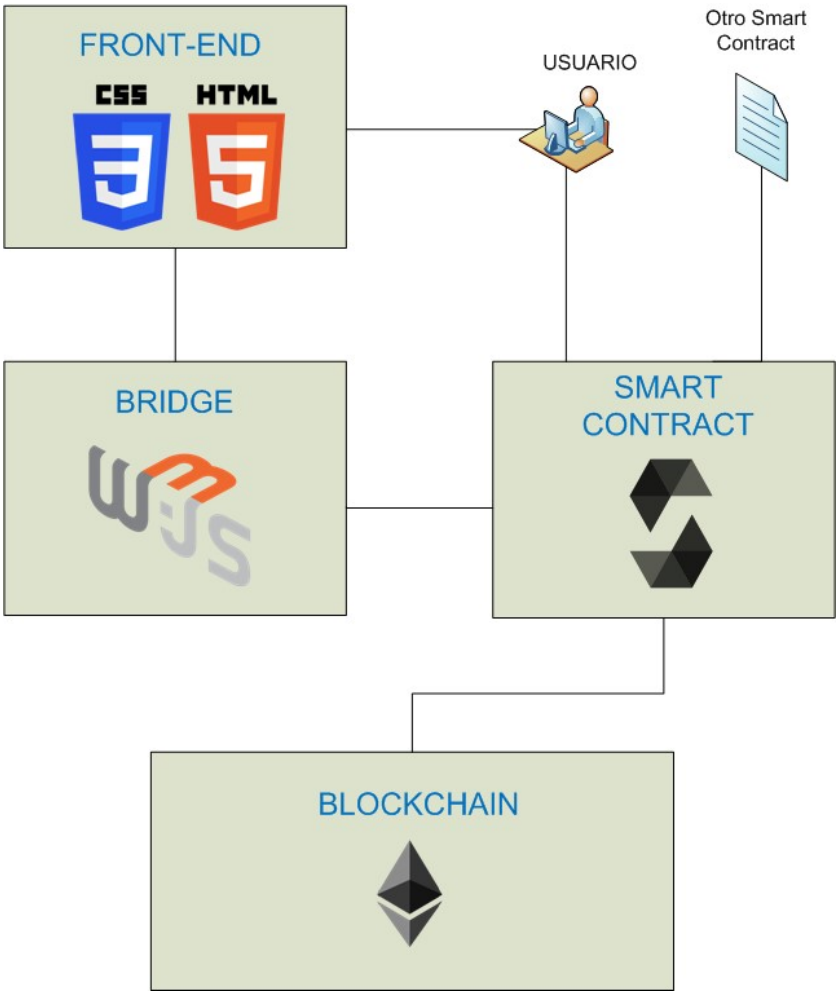


Ilustración 8: Arquitectura de la solución

Aunque los componentes ya han sido explicados anteriormente, cabe destacar cómo el usuario y otros Smart Contracts interactuarán con el sistema. El usuario podrá utilizar algún front-end disponible (como los que desarrollaremos en este proyecto), o llamar directamente a las funciones disponibles en el Smart Contract. Por otro lado, otros Smart Contracts podrán llamar directamente a las funciones disponibles en el Smart Contract a desarrollar.

5 Desarrollo, despliegue y testing de la solución

El contenido de este capítulo se ha resumido para mejorar la lectura del documento. El detalle completo de los pasos seguidos en la implementación de los diferentes componentes se puede encontrar en el "Anexo 1. Desarrollo, despliegue y testing de la solución".

5.1 Descripción del entorno de trabajo y la estructura del proyecto

Dada la complejidad que conlleva cualquier proyecto actual de desarrollo de Dapps, el entorno de trabajo debe estar claramente definido y estructurado para asegurar un eficiente proceso de desarrollo y la buena conexión entre los diferentes componentes.

Para el sistema de confianza y reputación, objeto de este proyecto, se han identificado 4 componentes principales en su estructura:

- **Truffle framework**: Este entorno de trabajo ofrece las herramientas necesarias para programar el Smart Contract del sistema utilizando el lenguaje de programación Solidity, y para compilarlo en *bytecode*, estructura necesaria para ser interpretado por la *EVM* (Ethereum Virtual Machine). Además, Truffle ofrece herramientas de testing unitario para el Smart Contract y permite desplegarlo en una red de Ethereum. El primer entorno de desarrollo que crearemos será utilizando Truffle para generar el Smart Contract.
- **Node y Webpack**: Estas dos tecnologías juntas facilitan el desarrollo de cualquier aplicación web basada en JavaScript. Node permite crear un entorno de trabajo para la aplicación web y gestionar los paquetes JS necesarios para el desarrollo de la funcionalidad dinámica de la aplicación. Webpack, que necesita Node para poder ser utilizada, permite compilar y optimizar la aplicación para mejorar el rendimiento en su versión final. Ambas tecnologías serán necesarias para desarrollar tanto los front-ends como el Bridge. Generaremos un segundo entorno de desarrollo utilizando Node y Webpack para desarrollar ambos componentes del sistema.
- **Metamask**: Esta aplicación permite comunicar una cuenta en una red de Ethereum (ya sea en "Producción", en una red de test pública, o en local) con una Dapp a través de una interfaz web. Utilizaremos Metamask para realizar las pruebas desde el front-end desarrollado.
- **Ganache**: Esta aplicación facilita la labor de desarrollo y testing de un Smart Contract en un entorno de red de blockchain similar al de "Producción". Esencialmente, Ganache crea un nodo de blockchain Ethereum en local y genera varias cuentas aleatorias, permitiendo desplegar Smart Contracts compilados a este nodo y realizar las operaciones necesarias para realizar las tareas de testing. Utilizaremos Ganache para generar un nodo de blockchain Ethereum en local, desplegar el Smart Contract compilado en Truffle sobre ese nodo, y realizar las pruebas necesarias desde el front-end junto con Metamask o desde la consola de Truffle.

En las siguientes secciones, se detalla el proceso seguido con cada uno de estos componentes para obtener el resultado final del sistema desarrollado.

5.2 Desarrollo del Smart Contract con Truffle Framework

Una vez instaladas las herramientas necesarias, creamos el Smart Contract que gestione el sistema de reputación y confianza. Lo llamaremos DTrustManager. Para desarrollarlo, se ha

seguido la estructura definida en el capítulo “Descripción y diseño de la solución a desarrollar”. El resultado final obtenido es el siguiente:

```
pragma solidity ^0.5.0;

contract DTrustManager
{
    ////////////
    // Events //
    ////////////
    event Rating(address indexed from, address indexed to, uint8 indexed score);

    ///////////////
    // Constants //
    ///////////////
    // Score constants
    uint8 constant LOW_SCORE = 1;
    uint8 constant MED_SCORE = 2;
    uint8 constant HIGH_SCORE = 3;

    // Trust matrix constants
    uint8 constant MAX_TRUST_DEPTH = 2;

    ////////////////////////
    // Managers variables //
    ////////////////////////
    address private _owner;
    mapping(address => bool) _operators;

    ////////////////////////
    // Storage variables //
    ////////////////////////
    // Received ratings
    mapping(address => uint8[]) private _ratings;

    // Ratings given by users
    mapping(address => mapping(address => uint8)) private _scores;

    // Trust graph
    mapping(address => address[]) private _trust_matrix;

    // Extra information (this struct might hold more information)
    struct extraInformation
    {
        uint256 network_transactions;
    }
    mapping(address => extraInformation) _user_extra_information;

    /////////////
    // Helpers //
    /////////////
    /**
     * @dev Returns true if the caller is an operator
     */
    function isOperator() internal view returns (bool)
    {
        return _operators[msg.sender];
    }

    /**
     * @dev Throws if called by any account that is not an operator
     */
    modifier onlyOperator()
    {
        require(isOperator(), "DTrustManager: caller is not the owner");
        _;
```

```
    }

    /**
     * @dev Checks whether a score is a valid score
     */
    function _isValidScore(uint8 score) internal pure returns (bool)
    {
        return
        (
            (score == LOW_SCORE) ||   // Lowest score (Possible scam)
            (score == MED_SCORE) ||   // Medium score
            (score == HIGH_SCORE)     // Highest score (maximum trust)
        );
    }


    ////////////////////////
    // Private functions //
    ////////////////////////
    /**
    * @dev Finds whether a target (user) is in the trust matrix of a user
    *       (A one parameter overriden function is provided to find whether a user is
in the trust network of the user calling the function)
    */
    function  _userIsInTrustNetwork(address  user,  address  target,  uint8  level)
internal view returns(bool)
    {
        if (level > MAX_TRUST_DEPTH) return false;

        bool userIsInTrustNetwork = false;
        for (uint16 i = 0; i < _trust_matrix[user].length; i++)
        {
            address trusted_node = _trust_matrix[user][i];
            if  (  (trusted_node  ==  target)  ||  _userIsInTrustNetwork(trusted_node,
target, level+1) )
            {
                userIsInTrustNetwork = true;
            }
        }

        return userIsInTrustNetwork;
    }
    function  _userIsInTrustNetwork(address  user,  address  target)  internal  view
returns(bool) { return _userIsInTrustNetwork(user, target, 1); }

    /**
    * @dev Punishing the trust network of `user`
    *    (gives an extra score of LOW_SCORE to all users that `user` trusts)
    */
    function _punishNetwork(address user) internal
    {
        for (uint16 i = 0; i < _trust_matrix[user].length; i++)
        {
            address trusted_node = _trust_matrix[user][i];
            _ratings[trusted_node].push(LOW_SCORE);
            _scores[msg.sender][trusted_node] = LOW_SCORE;
        }
    }

    ////////////////////////////////////
    // Public Operator-only functions //
    ////////////////////////////////////
    /**
    * @dev Adds a user's number of transactions as extra information to be used in
the algorithm
    *       (Function only accessible by the operators)
```

```
    */
    function addUserExtraInformation(address user, uint256 number_of_transactions)
public onlyOperator
    {
        extraInformation memory extra_information = extraInformation(
            {
                network_transactions: number_of_transactions
            }
        );
        _user_extra_information[user] = extra_information;
    }


    //////////////////////
    // Public functions //
    //////////////////////
    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor () public
    {
        _owner = msg.sender;
        _operators[_owner] = true;
    }

    /**
     * @dev Adds a new operator
     */
    function addOperator(address operator) public onlyOperator
    {
        _operators[operator] = true;
    }

    /**
     * @dev Removes an operator
     */
    function removeOperator(address operator) public onlyOperator
    {
        _operators[operator] = false;
    }

    /**
    * @dev Rates `user`
    */
    function rateUser(address user, uint8 score) public
    {
        require(user != address(0), "DTrustManager: Rating the zero address");
        require(_isValidScore(score), "DTrustManager: Score is not valid");

        // Saving the score in the proper matrices
        _ratings[user].push(score);
        _scores[msg.sender][user] = score;

        // Punishing network, if we have the lowest score
        if (score == LOW_SCORE)
        {
            _punishNetwork(user);
        }

        // Updating the trust graph
        if (score == HIGH_SCORE)
        {
            _trust_matrix[msg.sender].push(user);
        }

        // Emitting the appropriate event
```

```
        emit Rating(msg.sender, user, score);
    }

    /**
    * @dev Returns the general Trust Score of `user`
    */
    function getGeneralTrustScore(address user) public view returns (uint8)
    {
        uint16 n_ratings = 0;
        uint256 total_ratings = 0;

        for (uint16 i = 0; i < _ratings[user].length; i++)
        {
            total_ratings += _ratings[user][i];
            n_ratings++;
        }

        // If no ratings have been made, the medium score is returned (not good nor
bad)
        if (n_ratings == 0)
        {
            return MED_SCORE;
        }

        return uint8(total_ratings/n_ratings);
    }

    /**
    * @dev Returns the Trust Score relative to `user` as per user `from`
    *        (A one parameter overriden function is provided to get the trust score
relative to the user calling the function)
    */
    function getRelativeTrustScore(address from, address user) public view returns
(uint8)
    {
        // If user "from" already rated this user, we return that value
        if (_scores[from][user] > 0) return _scores[from][user];

        // Otherwise, we return the medium score (not good nor bad)
        return MED_SCORE;
    }
    function getRelativeTrustScore(address user) public view returns (uint8) { return
getRelativeTrustScore(msg.sender, user); }

    /**
    * @dev Returns the Trust Score relative to `user` as per user `from` or the
general score if there is no relation between them
    *        (A one parameter overriden function is provided to get the trust score
relative to the user calling the function)
    */
    function getTrustScore(address from, address user) public view returns (uint8)
    {
        // If user "from" already rated this user, we return that value
        if (_scores[from][user] > 0) return _scores[from][user];

        // If user is in the trust network of user "from", we return the high score
        if (_userIsInTrustNetwork(from, user)) return HIGH_SCORE;

        // Otherwise, we return the general trust score
        return getGeneralTrustScore(user);
    }
    function getTrustScore(address user) public view returns (uint8) { return
getTrustScore(msg.sender, user); }
}
```

Finalmente, compilamos el Smart Contract para obtener el *bytecode* necesario para migrarlo a la blockchain.

5.2.1 Inicialización del nodo local de Ethereum y despliegue del Smart Contract

Ahora que tenemos el Smart Contract desarrollado y compilado, lo desplegaremos en un nodo local en el que realizar las diferentes pruebas con el mismo. Utilizaremos el programa Ganache (https://www.trufflesuite.com/ganache) para crear nuestro nodo local.

Con la información que nos brinda el programa, modificamos el fichero de configuración de Truffle para indicar dónde queremos desplegar el Smart Contract. Seguidamente, procedemos a ejecutar el comando que realiza la migración del contracto DTrustManager a la blockchain local.

```
$ truffle migrate

Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.


Starting migrations...
======================
> Network name:    'development'
> Network id:      5777
> Block gas limit: 0x6691b7


1_dtrust_manager.js
===================

   Deploying 'DTrustManager'
   -------------------------
   >                               transaction                                hash:
0x58ad16ca989aeae7b0caa7d35bd06943c11c1a4c76d61d1f7000001d8980d58c
   > Blocks: 0            Seconds: 0
   > contract address:    0x4B6c8d10BbF57ffE36F8D2fe28beb332d824572C
   > block number:        6
   > block timestamp:     1575736345
   > account:             0x93B459B630Be5D1563A9B898c092F86f02939314
   > balance:             99.95404548
   > gas used:            702140
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.0140428 ETH

   > Saving artifacts
   -------------------------------------
   > Total cost:           0.0140428 ETH


Summary
=======
> Total deployments:   1
> Final cost:          0.0140428 ETH
```

Los resultados de la ejecución de este comando nos dan mucha información:

- Propietario del contrato: valor del parámetro “account”, que coincide con lo que habíamos definido en la configuración.
- Dirección del contrato: valor del parámetro “Contract address”.
- Gas utilizado: valor del parámetro “Gas used”. Esta información nos es importante si queremos calcular cuánto nos costará desplegar nuestro contrato en producción.

5.2.2 Desarrollo del Front-end con HTML y JS

Teniendo el Smart Contract desarrollado y desplegado en el nodo local, pasamos ahora a crear el front-end con el que nos conectaremos al Smart Contract a través del Bridge (que desarrollaremos en la siguiente sección).

Para este front-end, crearemos los ficheros directamente en una carpeta de la raíz del servidor local Apache. De esta manera, los ficheros estarán disponibles a través de nuestro navegador como si se tratara de un servidor web en Internet.

El objetivo de esta interfaz no es otro que el de probar el Smart Contract. Por lo tanto, crearemos una interfaz sencilla, que nos permita acceder a las funcionalidades necesarias como lo haría un usuario del Smart Contract.

Tras crear el fichero HTML y algunos estilos CSS básicos, el resultado que obtenemos es el siguiente:

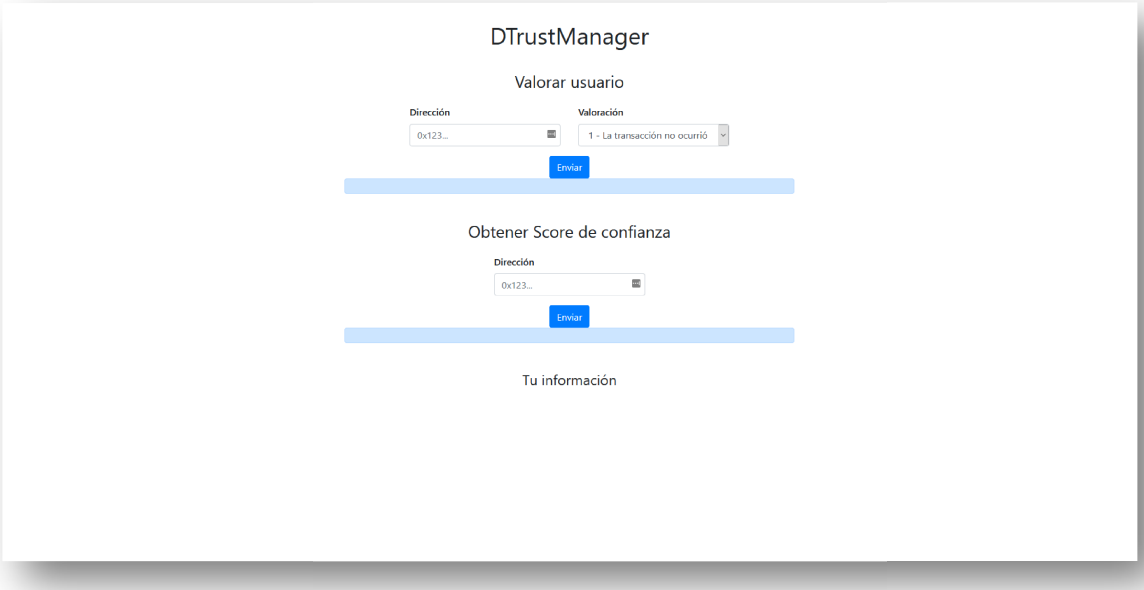


Ilustración 9: Front-end inicial. Imagen propia.

5.2.3 Desarrollo del Bridge Front- SmartContract con Node y Web3JS

La última pieza que nos falta para finalizar el sistema de reputación y confianza es el vínculo entre el front-end creado y el Smart Contract desplegado. Este vínculo lo realizaremos mediante un Bridge que permitirá que ciertas acciones realizadas en la interfaz, ejecuten funcionalidades del Smart Contract.

En la interfaz que hemos desarrollado, estas acciones serán dos:

- Valorar usuario, que llamará a la función rateUser() del Smart Contract
- Obtener Score de confianza, que llamará a la función getTrustScore() del Smart Contract.

El Bridge a desarrollar será sencillo, sólo necesitaremos un fichero JS con la especificación de todas las funciones necesarias para operar.

Para poder conectarse con Ethereum, necesitaremos la librería Web3js (https://web3js.readthedocs.io/en/v1.2.4/), que nos ofrece todo lo necesario para conectar cuentas a través del navegador y poder llamar a cualquier Smart Contract disponible en la red de blockchain, además de las funciones de la propia blockchain. Además, necesitamos especificar qué operaciones están disponibles de nuestro Smart Contract y esto se hará con el ABI (Application Binary Interface) de nuestro contrato, que no es otra cosa que un resumen de las operaciones existentes en *bytecode*, pero en un formato legible para JS. Este ABI se encuentra en el fichero JSON compilado con Truffle.

Mostramos el código del Bridge resultante.

```
///////////////
// Libraries //
///////////////
import Web3 from "web3";

///////////////
// Constants //
///////////////
const CONTRACT_ABI = require('./DTrustManager.json')['abi'];
const CONTRACT_ADDRESS = '0x4B6c8d10BbF57ffE36F8D2fe28beb332d824572C';

//////////////////////
// Global variables //
//////////////////////
var web3;
var connected_account;
var connected_account_trust_score;
var contract_handler;

//////////////////////
// Private Functions //
//////////////////////
async function _initialize_web3_provider()
{
    // Modern dapp browsers
    if (window.ethereum)
    {
        window.web3 = new Web3(ethereum);
        try
        {
            // Request account access if needed
            await ethereum.enable();
        }
        catch (error)
        {
            // User denied account access
            console.log(error);
        }
    }

    // Legacy dapp browsers
    else if (web3)
    {
        window.web3 = new Web3(web3.currentProvider);
    }
```

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

    // We have a connected provider
    if (window.web3 !== 'undefined')
    {
        await window.web3.eth.getAccounts().then(
            function(connected_accounts)
            {
                connected_account = connected_accounts[0];
                window.document.dispatchEvent(new CustomEvent('account_connected'));
            }
        );
    }
}

//
// Execute after web3 initialization
//
async function _initialize_blockchain_contract()
{
    var contract_instance = new window.web3.eth.Contract(CONTRACT_ABI, CONTRACT_ADDRE
SS);
    contract_handler = contract_instance.methods;

    await contract_handler.getGeneralTrustScore(connected_account).call().then(
        function(trust_score)
        {
            connected_account_trust_score = trust_score;
            window.document.dispatchEvent(new CustomEvent('contract_linked'));
        }
    );
}

async function _get_user_trust_score(user_address)
{
    return new Promise(
        async function(resolve)
        {
            return resolve(await contract_handler.getTrustScore(user_address).call(
                {
                    from: connected_account
                }
            ));
        }
    );
}

async function _rate_user(user_address, user_rating)
{
    return new Promise(
        async function(resolve)
        {
            return resolve(await contract_handler.rateUser(user_address, user_rating)
.send(
                {
                    from: connected_account
                }
            ));
        }
    );
```

```
}

//////////////////////
// Public functions //
//////////////////////
export function get_connected_account()
{
    return connected_account;
}

export function get_connected_account_trust_score()
{
    return connected_account_trust_score;
}

export async function get_user_trust_score(user_address)
{
    return new Promise(
        async function(resolve)
        {
            return resolve(await _get_user_trust_score(user_address));
        }
    );
}

export async function rate_user(user_address, user_rating)
{
    return new Promise(
        async function(resolve)
        {
            return resolve(await _rate_user(user_address, user_rating));
        }
    );
}

////////////////////
// Initialization //
////////////////////
window.addEventListener('load', async function()
{
    await _initialize_web3_provider();
    await _initialize_blockchain_contract()
});
```

Como vemos, web3 es el objeto encargado de gestionar la conexión con la blockchain, y CONTRACT_ABI y CONTRACT_ADDRESS las constantes que contienen la información sobre el ABI y la dirección de nuestro contrato.

La primera función que se ejecuta, al finalizar la carga del DOM, es _initialize_web3_provider(), seguido de _initialize_blockchain_contract().

La primera función inicializa el objeto web3, que nos permitirá conectar con la blockchain, y al inicializarse, comprobará si existe alguna cuenta disponible a través de Metamask (que comentaremos en la siguiente sección). Si la hay, ejecutará las acciones en la blockchain utilizando esa cuenta.

La segunda función inicializa nuestro Smart Contract y llama a la función getGeneralTrustScore() del contrato, pasando como parámetro la cuenta que pueda estar conectada a través de Metamask.

Tras esto, el Bridge espera las acciones realizadas desde el front-end. El Bridge pone a disposición 4 acciones que pueden ser llamadas desde el front:

- Get_connected_account() : Obtiene la cuenta conectada a través de Metamask
- Get_connected_account_trust_score() : Obtiene el Score de confianza de la cuenta conectada a través de Metamask
- Get_user_trust_score() : Obtiene el Score de confianza de un usuario
- Rate_user() : Otorga una puntuación a un usuario

Ahora que tenemos el Bridge en código, lo compilamos para que nos sea fácilmente exportable. Con el fichero compilado, ya podemos hacer uso de estas funciones desde el front-end.

5.2.4 Ejecución y testing del sistema a través del Front-end con Metamask

Con todo el sistema integrado, podemos proceder a ejecutar y realizar pruebas desde el front-end, conectándonos con Metamask (https://metamask.io/).

Procedemos a acceder al front-end de nuestra Dapp para comprobar su funcionamiento.

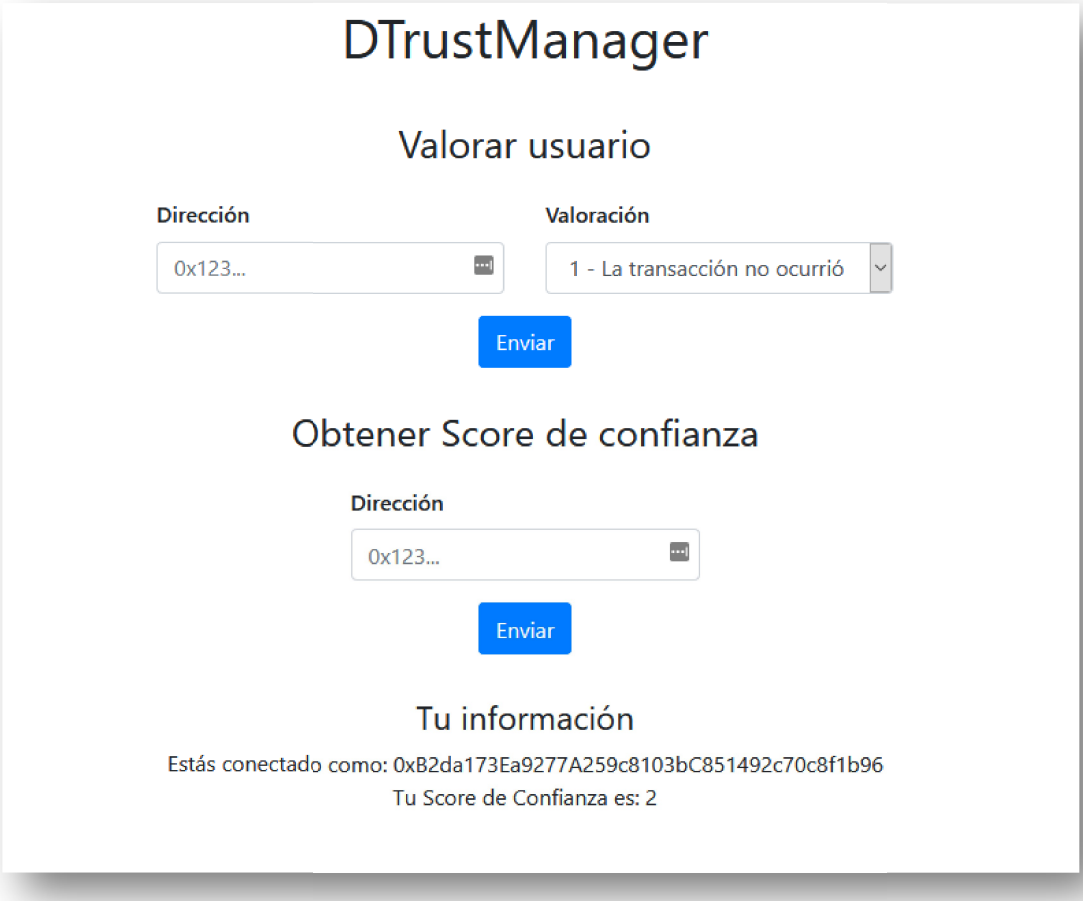


Ilustración 10: Front-end conectado con Metamask. Imagen propia.

Como podemos observar en el bloque de "Tu información", estamos efectivamente conectados con la cuenta deseada y nuestro Score de confianza es "2" (Medio). Esto se debe a

que no hemos recibido ninguna valoración aún, con lo que el sistema no se decanta por un lado ni por otro.

En primer lugar, obtengamos el Score de confianza de otro usuario de la red, proporcionado por Ganache, el “0xC7C854Cd30af6638ec2B0A2489c8d3D752AF71e1”.

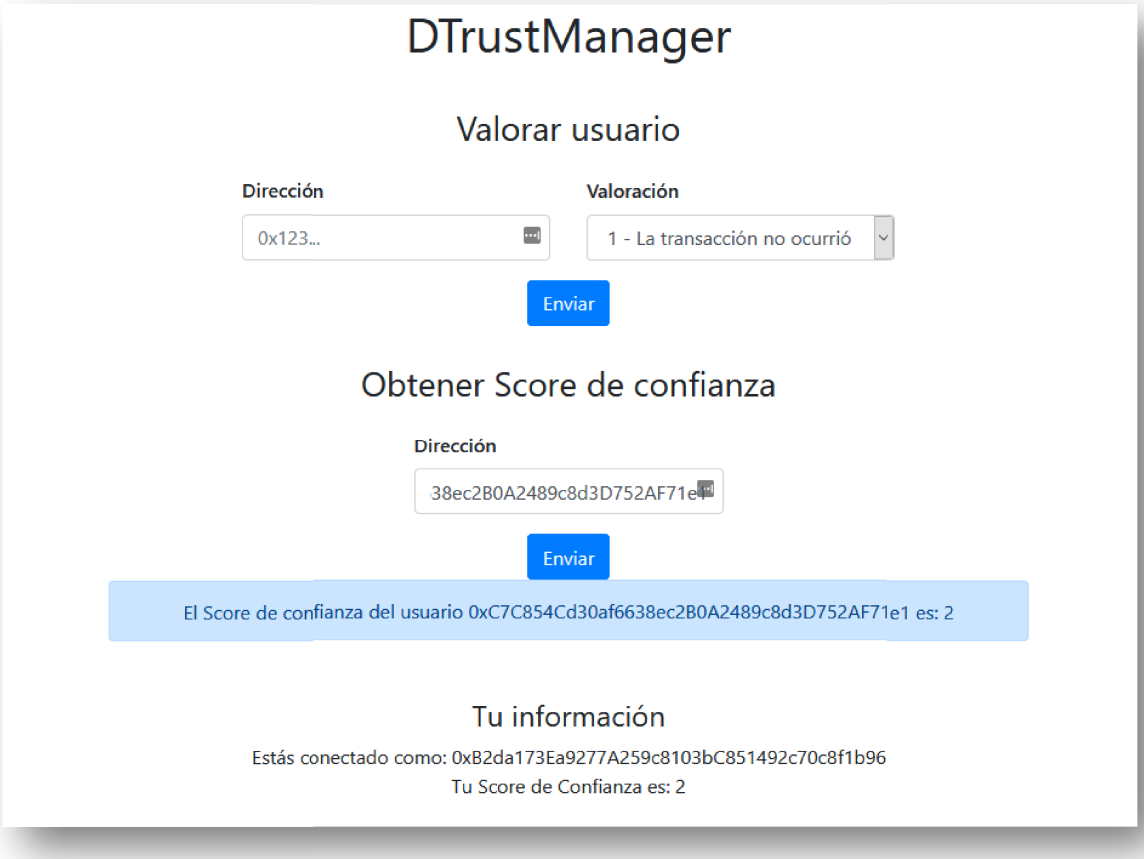


Ilustración 11: Obtener valoración de un usuario. Imagen propia.

Al igual que nuestro Score de confianza, el resultado es 2 (Medio), ya que aún no ha recibido ninguna valoración.

Procedemos ahora a otorgar una valoración buena, 3, a ese mismo usuario, y volvemos a llamar a la función de “Obtener Score de confianza”.

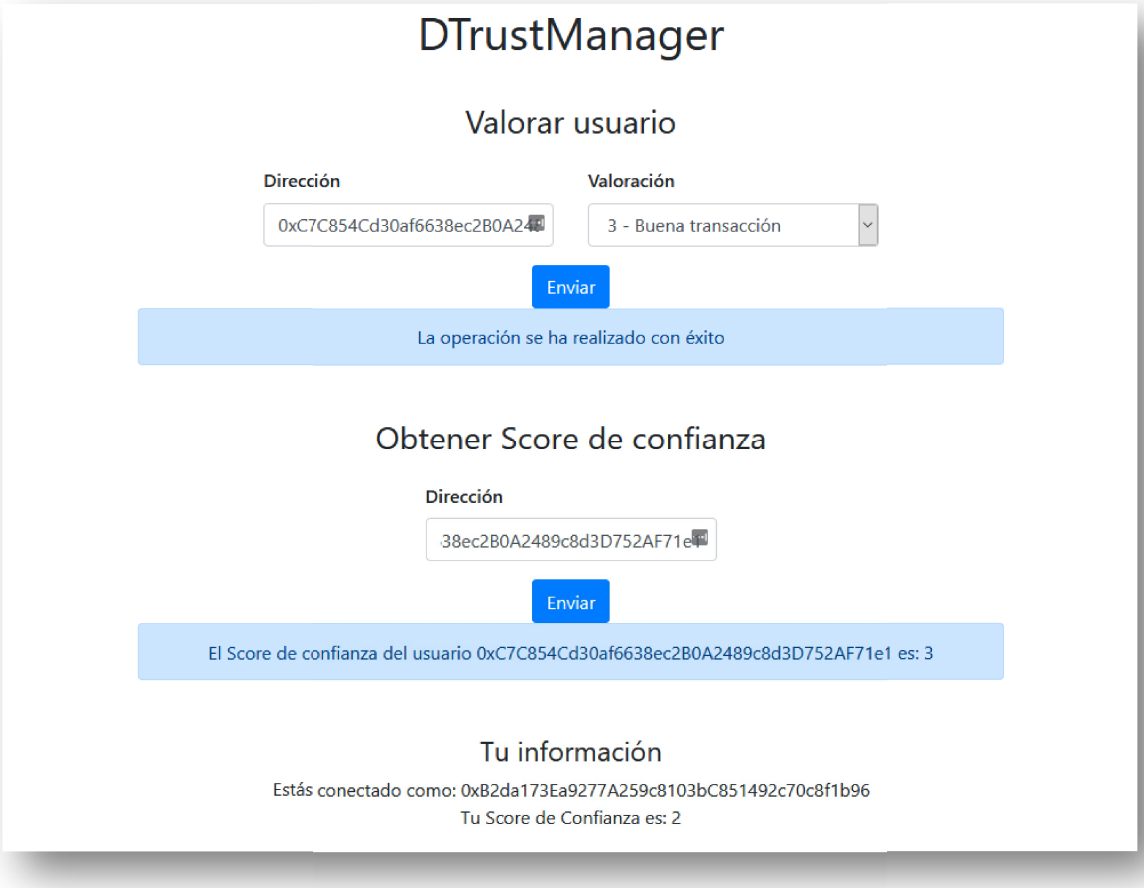


Ilustración 12: Valorar usuario y obtener su score de confianza. Imagen propia.

Efectivamente, podemos ver que el usuario valorado tiene ahora un Score de confianza diferente.

5.3 Testing del sistema mediante simulación de transacciones en la red

En el "Anexo 1. Desarrollo, despliegue y testing de la solución" se puede encontrar, también, el proceso seguido para el desarrollo del simulador de transacciones en la red. Sin embargo, en esta sección contemplamos el escenario establecido y los resultados obtenidos.

Para poder validar el sistema, es necesario realizar una simulación de cómo se comportaría nuestra solución en un entorno donde interactúan un número indeterminado de nodos realizando una gran cantidad de transacciones entre ellos.

En nuestro caso, simularemos este comportamiento creando veinte cuentas, asignándoles un comportamiento por defecto y ejecutando 200 transacciones entre ellas. Utilizaremos la herramienta de testing que viene integrada en Truffle.

Para verificar la mejora que supone la utilización del sistema desarrollado, realizaremos una primera simulación de transacciones sin utilizar el algoritmo de confianza y, seguidamente, 4 simulaciones más utilizando el algoritmo de confianza.

Antes de mostrar el proceso, sin embargo, explicamos los comportamientos que hemos elegido para permitir a los nodos tomar las decisiones dentro de la red. Hemos elegido lo que creemos que podría representar los 4 comportamientos más básicos dentro de una red:

- **SCAM**: Nodo malicioso. Su objetivo en la red es beneficiarse del resto de nodos, engañándoles y no cumpliendo su parte del trato. Estos nodos siempre quieren realizar las transacciones y, en caso de realizarlas, no puntúan al nodo con el que han interactuado, a no ser que sea otro nodo malicioso, al cual le darán una valoración alta. Suponemos que los nodos maliciosos forman un grupo entre ellos para hacerse más fuertes.
- **WARY**: Nodo precavido. Este nodo sólo realiza transacciones con nodos que tienen un Score de Confianza alto. Si la transacción ocurre correctamente, este nodo otorga una valoración alta.
- **NORMAL**: Nodo corriente. Este nodo sólo realiza transacciones con nodos que tienen un Score de Confianza medio o alto. Si la transacción ocurre correctamente, este nodo otorga una valoración alta.
- **RISKY**: Nodo atrevido. Este nodo realiza transacciones con todos los nodos, sin importar su Score de Confianza. Si la transacción ocurre correctamente, este nodo otorga una valoración alta.

Tras desarrollar nuestro simulador, procedemos a ejecutar el test. Los resultados que obtenemos son los siguientes.

```
josef@DESKTOP-8LVS0OR Z:\Dropbox\Development\Ethereum\dtrust-manager\truffle
$ truffle compile

Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.


josef@DESKTOP-8LVS0OR Z:\Dropbox\Development\Ethereum\dtrust-manager\truffle
$ truffle migrate

Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.


Starting migrations...
======================
> Network name:    'development'
> Network id:      5777
> Block gas limit: 0x6691b7


1_dtrust_manager.js
===================

   Replacing 'DTrustManager'
   -------------------------
   >                               transaction                                    hash:
0x1d3329619f0009f0aa5e80f684022a8a7c895430fdc32bcb3a2dd0dd410b6db5
   > Blocks: 0            Seconds: 0
   > contract address:    0xde3524F1e246aE87c5Eb8c0fA5dAbb06364Dd113
   > block number:        1
   > block timestamp:     1576431739
   > account:             0x40e837FE16f1482EC20F1BbA7d543e706E134FEc
   > balance:             99.99191516
   > gas used:            404242
```

```
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.00808484 ETH

   > Saving artifacts
   -------------------------------------
   > Total cost:          0.00808484 ETH


Summary
=======
> Total deployments:   1
> Final cost:          0.00808484 ETH

josef@DESKTOP-8LVS0OR Z:\Dropbox\Development\Ethereum\dtrust-manager\truffle
$ truffle test .\test\simulations.js
Using network 'development'.


Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.



  Contract: DTrustManager
    Simulation
**************************************
First simulation: Transactions without DTrustManager
**************************************

------------------------
| Perform transactions |
------------------------
[1] 8 - WARY -- [2] 13 - NORMAL
Transaction is performed.
[1] 19 - RISKY -- [2] 11 - NORMAL
Transaction is performed.
[1] 19 - RISKY -- [2] 2 - SCAM
Transaction is performed.
[1] 5 - WARY -- [2] 8 - WARY
Transaction is performed.
[1] 17 - RISKY -- [2] 18 - RISKY
Transaction is performed.
[1] 0 - SCAM -- [2] 12 - NORMAL
Transaction is performed.
[1] 0 - SCAM -- [2] 16 - NORMAL
Transaction is performed.
[1] 6 - WARY -- [2] 0 - SCAM
Transaction is performed.
[1] 2 - SCAM -- [2] 18 - RISKY
Transaction is performed.
[1] 2 - SCAM -- [2] 11 - NORMAL
Transaction is performed.
[1] 4 - WARY -- [2] 10 - NORMAL
Transaction is performed.
[1] 9 - WARY -- [2] 14 - NORMAL
Transaction is performed.

[.......]
NO MOSTRAMOS TODO EL RESULTADO, YA QUE ES MUY LARGO
[.......]
```

```
-----------------
| Final results |
-----------------
[Account 0 - SCAM] Transactions: 17; SCAMs: 0
[Account 1 - SCAM] Transactions: 20; SCAMs: 0
[Account 2 - SCAM] Transactions: 38; SCAMs: 0
[Account 3 - WARY] Transactions: 20; SCAMs: 4
[Account 4 - WARY] Transactions: 14; SCAMs: 1
[Account 5 - WARY] Transactions: 20; SCAMs: 2
[Account 6 - WARY] Transactions: 18; SCAMs: 5
[Account 7 - WARY] Transactions: 18; SCAMs: 3
[Account 8 - WARY] Transactions: 18; SCAMs: 4
[Account 9 - WARY] Transactions: 23; SCAMs: 8
[Account 10 - NORMAL] Transactions: 15; SCAMs: 2
[Account 11 - NORMAL] Transactions: 16; SCAMs: 4
[Account 12 - NORMAL] Transactions: 19; SCAMs: 5
[Account 13 - NORMAL] Transactions: 19; SCAMs: 3
[Account 14 - NORMAL] Transactions: 27; SCAMs: 4
[Account 15 - NORMAL] Transactions: 17; SCAMs: 3
[Account 16 - NORMAL] Transactions: 18; SCAMs: 5
[Account 17 - RISKY] Transactions: 20; SCAMs: 6
[Account 18 - RISKY] Transactions: 21; SCAMs: 4
[Account 19 - RISKY] Transactions: 22; SCAMs: 6
**************************************
Total Transactions: 200
Total Scams: 69
SCAM percentage: 34.5%
**************************************

      √ performs first simulation: Transactions without DTrustManager (137ms)

**************************************
Second simulation: Transactions with DTrustManager (I)
**************************************

------------------------
| Perform transactions |
------------------------
[1] 8 - WARY (2) -- [2] 14 - NORMAL (2)
Transaction is NOT performed.
[1] 12 - NORMAL (2) -- [2] 10 - NORMAL (2)
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]
[1] 1 - SCAM (2) -- [2] 6 - WARY (2)
Transaction is NOT performed.
[1] 16 - NORMAL (2) -- [2] 1 - SCAM (2)
Transaction is performed.
[1] gives score 1 to [2]
[1] 10 - NORMAL (3) -- [2] 0 - SCAM (2)
Transaction is performed.
[1] gives score 1 to [2]
[1] 0 - SCAM (1) -- [2] 1 - SCAM (1)
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]
[1] 18 - RISKY (2) -- [2] 19 - RISKY (2)
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]
[1] 16 - NORMAL (2) -- [2] 4 - WARY (2)
Transaction is NOT performed.
[1] 12 - NORMAL (3) -- [2] 6 - WARY (2)
```

```
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]

[.......]
NO MOSTRAMOS TODO EL RESULTADO, YA QUE ES MUY LARGO
[.......]

-----------------
| Final results |
-----------------
[Account 0 - SCAM] Transactions: 6/21 (28.57%); SCAMs: 0/0 (0%); Trust Score: 1
[Account 1 - SCAM] Transactions: 9/26 (34.62%); SCAMs: 0/0 (0%); Trust Score: 1
[Account 2 - SCAM] Transactions: 7/16 (43.75%); SCAMs: 0/0 (0%); Trust Score: 1
[Account 3 - WARY] Transactions: 11/11 (100%); SCAMs: 0/0 (0%); Trust Score: 3
[Account 4 - WARY] Transactions: 10/16 (62.5%); SCAMs: 0/2 (0%); Trust Score: 3
[Account 5 - WARY] Transactions: 13/17 (76.47%); SCAMs: 0/3 (0%); Trust Score: 3
[Account 6 - WARY] Transactions: 14/18 (77.78%); SCAMs: 0/3 (0%); Trust Score: 3
[Account 7 - WARY] Transactions: 19/25 (76%); SCAMs: 0/5 (0%); Trust Score: 3
[Account 8 - WARY] Transactions: 10/17 (58.82%); SCAMs: 0/5 (0%); Trust Score: 3
[Account 9 - WARY] Transactions: 13/15 (86.67%); SCAMs: 0/1 (0%); Trust Score: 3
[Account 10 - NORMAL] Transactions: 19/21 (90.48%); SCAMs: 1/3 (33.33%); Trust
Score: 3
[Account 11 - NORMAL] Transactions: 22/26 (84.62%); SCAMs: 1/5 (20%); Trust Score:
3
[Account 12 - NORMAL] Transactions: 19/20 (95%); SCAMs: 0/1 (0%); Trust Score: 3
[Account 13 - NORMAL] Transactions: 12/17 (70.59%); SCAMs: 0/5 (0%); Trust Score: 3
[Account 14 - NORMAL] Transactions: 13/19 (68.42%); SCAMs: 0/4 (0%); Trust Score: 3
[Account 15 - NORMAL] Transactions: 15/18 (83.33%); SCAMs: 0/3 (0%); Trust Score: 3
[Account 16 - NORMAL] Transactions: 24/29 (82.76%); SCAMs: 1/4 (25%); Trust Score:
3
[Account 17 - RISKY] Transactions: 16/16 (100%); SCAMs: 4/4 (100%); Trust Score: 3
[Account 18 - RISKY] Transactions: 23/23 (100%); SCAMs: 1/1 (100%); Trust Score: 3
[Account 19 - RISKY] Transactions: 29/29 (100%); SCAMs: 4/4 (100%); Trust Score: 3
Total Transactions: 152
Total Scams: 12
SCAM percentage: 7.89%
      √ performs second simulation: Transactions with DTrustManager (74162ms)

************************************
Third simulation: Transactions with DTrustManager (II)
************************************

------------------------
| Perform transactions |
------------------------
[1] 1 - SCAM (2) -- [2] 3 - WARY (2)
Transaction is NOT performed.
[1] 9 - WARY (2) -- [2] 4 - WARY (2)
Transaction is NOT performed.
[1] 0 - SCAM (2) -- [2] 1 - SCAM (2)
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]
[1] 2 - SCAM (2) -- [2] 14 - NORMAL (2)
Transaction is performed.
[2] gives score 1 to [1]
[1] 16 - NORMAL (2) -- [2] 1 - SCAM (3)
Transaction is performed.
[1] gives score 1 to [2]
[1] 10 - NORMAL (2) -- [2] 1 - SCAM (2)
Transaction is performed.
[1] gives score 1 to [2]
```

```
[1] 9 - WARY (2) -- [2] 6 - WARY (2)
Transaction is NOT performed.
[1] 13 - NORMAL (2) -- [2] 0 - SCAM (1)
Transaction is NOT performed.
[1] 4 - WARY (2) -- [2] 15 - NORMAL (2)
Transaction is NOT performed.
[1] 12 - NORMAL (2) -- [2] 2 - SCAM (1)
Transaction is NOT performed.
[1] 10 - NORMAL (2) -- [2] 19 - RISKY (2)
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]
[1] 16 - NORMAL (2) -- [2] 17 - RISKY (2)
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]
[1] 4 - WARY (2) -- [2] 19 - RISKY (3)
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]

[.......]
NO MOSTRAMOS TODO EL RESULTADO, YA QUE ES MUY LARGO
[.......]

-----------------
| Final results |
-----------------
[Account 0 - SCAM] Transactions: 4/17 (23.53%); SCAMs: 0/0 (0%); Trust Score: 1
[Account 1 - SCAM] Transactions: 6/27 (22.22%); SCAMs: 0/0 (0%); Trust Score: 1
[Account 2 - SCAM] Transactions: 3/14 (21.43%); SCAMs: 0/0 (0%); Trust Score: 1
[Account 3 - WARY] Transactions: 13/18 (72.22%); SCAMs: 0/4 (0%); Trust Score: 3
[Account 4 - WARY] Transactions: 15/22 (68.18%); SCAMs: 0/3 (0%); Trust Score: 3
[Account 5 - WARY] Transactions: 21/28 (75%); SCAMs: 0/6 (0%); Trust Score: 3
[Account 6 - WARY] Transactions: 14/23 (60.87%); SCAMs: 0/5 (0%); Trust Score: 3
[Account 7 - WARY] Transactions: 10/15 (66.67%); SCAMs: 0/5 (0%); Trust Score: 3
[Account 8 - WARY] Transactions: 12/20 (60%); SCAMs: 0/4 (0%); Trust Score: 3
[Account 9 - WARY] Transactions: 13/18 (72.22%); SCAMs: 0/2 (0%); Trust Score: 3
[Account 10 - NORMAL] Transactions: 25/28 (89.29%); SCAMs: 1/4 (25%); Trust Score:
3
[Account 11 - NORMAL] Transactions: 13/16 (81.25%); SCAMs: 0/3 (0%); Trust Score: 3
[Account 12 - NORMAL] Transactions: 13/16 (81.25%); SCAMs: 0/3 (0%); Trust Score: 3
[Account 13 - NORMAL] Transactions: 10/13 (76.92%); SCAMs: 0/3 (0%); Trust Score: 3
[Account 14 - NORMAL] Transactions: 25/25 (100%); SCAMs: 1/1 (100%); Trust Score: 3
[Account 15 - NORMAL] Transactions: 14/18 (77.78%); SCAMs: 0/3 (0%); Trust Score: 3
[Account 16 - NORMAL] Transactions: 20/21 (95.24%); SCAMs: 1/2 (50%); Trust Score:
3
[Account 17 - RISKY] Transactions: 18/18 (100%); SCAMs: 1/1 (100%); Trust Score: 3
[Account 18 - RISKY] Transactions: 17/19 (89.47%); SCAMs: 1/1 (100%); Trust Score:
3
[Account 19 - RISKY] Transactions: 24/24 (100%); SCAMs: 6/6 (100%); Trust Score: 3
Total Transactions: 145
Total Scams: 11
SCAM percentage: 7.59%
      √ performs third simulation: Transactions with DTrustManager (72244ms)

************************************
Fourth simulation: Transactions with DTrustManager (III)
************************************

------------------------
| Perform transactions |
------------------------
```

```
[1] 19 - RISKY (2) -- [2] 13 - NORMAL (2)
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]
[1] 6 - WARY (2) -- [2] 7 - WARY (2)
Transaction is NOT performed.
[1] 0 - SCAM (2) -- [2] 6 - WARY (2)
Transaction is NOT performed.
[1] 4 - WARY (2) -- [2] 8 - WARY (2)
Transaction is NOT performed.
[1] 10 - NORMAL (2) -- [2] 13 - NORMAL (3)
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]
[1] 6 - WARY (2) -- [2] 4 - WARY (2)
Transaction is NOT performed.
[1] 14 - NORMAL (2) -- [2] 19 - RISKY (3)
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]
[1] 18 - RISKY (2) -- [2] 2 - SCAM (2)
Transaction is performed.
[1] gives score 1 to [2]
[1] 10 - NORMAL (3) -- [2] 3 - WARY (2)
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]
[1] 5 - WARY (2) -- [2] 4 - WARY (2)
Transaction is NOT performed.
[1] 18 - RISKY (2) -- [2] 13 - NORMAL (3)
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]

[.......]
NO MOSTRAMOS TODO EL RESULTADO, YA QUE ES MUY LARGO
[.......]

-----------------
| Final results |
-----------------
[Account 0 - SCAM] Transactions: 6/13 (46.15%); SCAMs: 0/0 (0%); Trust Score: 1
[Account 1 - SCAM] Transactions: 5/20 (25%); SCAMs: 0/0 (0%); Trust Score: 1
[Account 2 - SCAM] Transactions: 5/17 (29.41%); SCAMs: 0/0 (0%); Trust Score: 1
[Account 3 - WARY] Transactions: 16/17 (94.12%); SCAMs: 0/0 (0%); Trust Score: 3
[Account 4 - WARY] Transactions: 16/24 (66.67%); SCAMs: 0/4 (0%); Trust Score: 3
[Account 5 - WARY] Transactions: 24/27 (88.89%); SCAMs: 0/1 (0%); Trust Score: 3
[Account 6 - WARY] Transactions: 23/29 (79.31%); SCAMs: 0/3 (0%); Trust Score: 3
[Account 7 - WARY] Transactions: 20/26 (76.92%); SCAMs: 0/5 (0%); Trust Score: 3
[Account 8 - WARY] Transactions: 17/19 (89.47%); SCAMs: 0/1 (0%); Trust Score: 3
[Account 9 - WARY] Transactions: 12/16 (75%); SCAMs: 0/3 (0%); Trust Score: 3
[Account 10 - NORMAL] Transactions: 20/24 (83.33%); SCAMs: 0/4 (0%); Trust Score: 3
[Account 11 - NORMAL] Transactions: 15/17 (88.24%); SCAMs: 0/2 (0%); Trust Score: 3
[Account 12 - NORMAL] Transactions: 20/24 (83.33%); SCAMs: 1/4 (25%); Trust Score:
3
[Account 13 - NORMAL] Transactions: 14/17 (82.35%); SCAMs: 1/4 (25%); Trust Score:
3
[Account 14 - NORMAL] Transactions: 20/22 (90.91%); SCAMs: 0/2 (0%); Trust Score: 3
[Account 15 - NORMAL] Transactions: 17/17 (100%); SCAMs: 1/1 (100%); Trust Score: 3
[Account 16 - NORMAL] Transactions: 13/17 (76.47%); SCAMs: 0/3 (0%); Trust Score: 3
[Account 17 - RISKY] Transactions: 15/18 (83.33%); SCAMs: 1/1 (100%); Trust Score:
3
[Account 18 - RISKY] Transactions: 16/16 (100%); SCAMs: 3/3 (100%); Trust Score: 3
```

```
[Account 19 - RISKY] Transactions: 20/20 (100%); SCAMs: 5/5 (100%); Trust Score: 3
Total Transactions: 157
Total Scams: 12
SCAM percentage: 7.64%
      √ performs fourth simulation: Transactions with DTrustManager (75063ms)

**************************************
Fifth simulation: Transactions with DTrustManager (IV)
**************************************

------------------------
| Perform transactions |
------------------------
[1] 10 - NORMAL (2) -- [2] 1 - SCAM (2)
Transaction is performed.
[1] gives score 1 to [2]
[1] 4 - WARY (2) -- [2] 12 - NORMAL (2)
Transaction is NOT performed.
[1] 11 - NORMAL (2) -- [2] 6 - WARY (2)
Transaction is NOT performed.
[1] 16 - NORMAL (2) -- [2] 11 - NORMAL (2)
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]
[1] 8 - WARY (2) -- [2] 1 - SCAM (1)
Transaction is NOT performed.
[1] 1 - SCAM (1) -- [2] 6 - WARY (2)
Transaction is NOT performed.
[1] 12 - NORMAL (2) -- [2] 7 - WARY (2)
Transaction is NOT performed.
[1] 3 - WARY (2) -- [2] 13 - NORMAL (2)
Transaction is NOT performed.
[1] 13 - NORMAL (2) -- [2] 16 - NORMAL (3)
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]
[1] 13 - NORMAL (3) -- [2] 2 - SCAM (2)
Transaction is performed.
[1] gives score 1 to [2]
[1] 13 - NORMAL (3) -- [2] 1 - SCAM (1)
Transaction is NOT performed.
[1] 3 - WARY (2) -- [2] 10 - NORMAL (2)
Transaction is NOT performed.
[1] 19 - RISKY (2) -- [2] 16 - NORMAL (3)
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]
[1] 6 - WARY (2) -- [2] 5 - WARY (2)
Transaction is NOT performed.

[.......]
NO MOSTRAMOS TODO EL RESULTADO, YA QUE ES MUY LARGO
[.......]

-----------------
| Final results |
-----------------
[Account 0 - SCAM] Transactions: 4/15 (26.67%); SCAMs: 0/0 (0%); Trust Score: 1
[Account 1 - SCAM] Transactions: 11/29 (37.93%); SCAMs: 0/0 (0%); Trust Score: 1
[Account 2 - SCAM] Transactions: 8/20 (40%); SCAMs: 0/0 (0%); Trust Score: 1
[Account 3 - WARY] Transactions: 12/21 (57.14%); SCAMs: 0/5 (0%); Trust Score: 3
[Account 4 - WARY] Transactions: 12/16 (75%); SCAMs: 0/0 (0%); Trust Score: 3
[Account 5 - WARY] Transactions: 17/24 (70.83%); SCAMs: 0/2 (0%); Trust Score: 3
```

```
[Account 6 - WARY] Transactions: 19/26 (73.08%); SCAMs: 0/2 (0%); Trust Score: 3
[Account 7 - WARY] Transactions: 16/25 (64%); SCAMs: 0/6 (0%); Trust Score: 3
[Account 8 - WARY] Transactions: 14/17 (82.35%); SCAMs: 0/2 (0%); Trust Score: 3
[Account 9 - WARY] Transactions: 10/21 (47.62%); SCAMs: 0/7 (0%); Trust Score: 3
[Account 10 - NORMAL] Transactions: 10/12 (83.33%); SCAMs: 1/2 (50%); Trust Score:
3
[Account 11 - NORMAL] Transactions: 12/15 (80%); SCAMs: 1/3 (33.33%); Trust Score:
3
[Account 12 - NORMAL] Transactions: 17/24 (70.83%); SCAMs: 0/1 (0%); Trust Score: 3
[Account 13 - NORMAL] Transactions: 10/17 (58.82%); SCAMs: 1/7 (14.29%); Trust
Score: 3
[Account 14 - NORMAL] Transactions: 17/21 (80.95%); SCAMs: 1/4 (25%); Trust Score:
3
[Account 15 - NORMAL] Transactions: 13/15 (86.67%); SCAMs: 0/2 (0%); Trust Score: 3
[Account 16 - NORMAL] Transactions: 19/21 (90.48%); SCAMs: 0/2 (0%); Trust Score: 3
[Account 17 - RISKY] Transactions: 17/17 (100%); SCAMs: 5/5 (100%); Trust Score: 3
[Account 18 - RISKY] Transactions: 22/22 (100%); SCAMs: 4/4 (100%); Trust Score: 3
[Account 19 - RISKY] Transactions: 22/22 (100%); SCAMs: 6/6 (100%); Trust Score: 3
Total Transactions: 141
Total Scams: 19
SCAM percentage: 13.48%
      ✓ performs fifth simulation: Transactions with DTrustManager (66142ms)


  5 passing (5m)
```

Como podemos observar, todas las simulaciones utilizando el algoritmo de confianza mejoran los resultados de la simulación sin el algoritmo de confianza. La simulación sin el algoritmo arrojó un porcentaje de SCAM del 34,5%, mientras que la media del resto de pruebas arroja un porcentaje del 9%.

En todas las simulaciones con el algoritmo, los nodos maliciosos han sido rápidamente identificados y el resto de nodos ha tomado como decisión no interactuar con ellos. Los nodos maliciosos sólo han ejecutado alrededor del 25-30% de las transacciones, que probablemente incluyan bastantes transacciones entre ellos. El resto de nodos han ejecutado más transacciones entre ellos, teniendo siempre en cuenta su comportamiento por defecto.

Este resultado nos indica que la utilización del sistema de confianza reduce las transacciones desde nodos maliciosos y, por lo tanto, mejora la salud global de la red, permitiendo que se ejecuten más transacciones de forma segura directamente entre nodos. Con esto podemos afirmar que hemos logrado el objetivo que pretendíamos en este trabajo.

5.4 Experimentación con el algoritmo

Antes de finalizar este capítulo, cabe recalcar que este algoritmo se presta a realizar más simulaciones para comprobar su comportamiento en una red blockchain. Por limitaciones de tiempo, no ha sido posible incluir el resto de experimentos en este trabajo, pero sí queremos hacer mención a varios posibles experimentos adicionales incluidos a posteriori en la planificación de las pruebas:

- **Progresión temporal de transacciones con los nodos maliciosos**: Sería interesante investigar en qué momento ocurren exactamente las transacciones con los nodos maliciosos. ¿Ocurren todas al principio de la simulación? ¿Están repartidas de forma equitativa en el tiempo?

- **Pruebas específicas con estrategias de grupos de nodos maliciosos**: Conociendo cómo funciona el algoritmo de confianza, los grupos de nodos maliciosos podrían formar estrategias para intentar sobreponerse a él. Por ejemplo, ¿qué ocurre si antes de comenzar los nodos maliciosos realizan un gran número de transacciones entre ellos y se otorgan la máxima puntuación?
- **Relación entre el número de transacciones de la simulación y los resultados obtenidos**: Sería interesante conocer qué ocurre si se realizan pocas transacciones en la simulación. ¿El porcentaje de transacciones con nodos maliciosos aumenta? ¿Qué ocurre con los nodos precavidos?
- **Pruebas con redes que contengan más nodos de cada uno de los tipos definidos**: En nuestra simulación, hemos compensado más o menos cada uno de los tipos de nodos definidos pero, ¿qué ocurriría en una red en la que un 75% de los nodos es malicioso? ¿Y si fueran precavidos?
- **Impacto de la adición de nodos maliciosos en distintos puntos de la simulación**: Sería interesante comprobar el comportamiento del algoritmo cuando se añaden nodos maliciosos a mitad de la simulación, o hacia el final. ¿Cuál es el impacto en la red?

6 Conclusiones y próximos pasos

En este capítulo se presentan las conclusiones del trabajo realizado. En primer lugar se estudia la viabilidad del sistema desarrollado, así como las limitaciones encontradas. A continuación, se mencionan los objetivos conseguidos en base a los detallados al inicio del Trabajo. Seguidamente se pasan a enumerar las contribuciones realizadas con este trabajo en el ámbito tecnológico de blockchain. Por último, se detallan los posibles próximos pasos a trabajar para mejorar la solución propuesta.

6.1 Viabilidad del sistema desarrollado y limitaciones identificadas

El sistema que se ha desarrollado en este trabajo aporta un valor añadido a las redes de blockchain actuales al establecer una solución para el problema de confianza existente en las transacciones entre pares.

Mediante la utilización de una red de confianza (Web of Trust) y un sistema de reputación, los usuarios son capaces de obtener información sobre la confianza que la red tiene en un usuario determinado antes de realizar una transacción con él.

Con este sistema, además, los usuarios maliciosos son rápidamente identificados por poseer un Score de Confianza bajo. Además, los grupos de nodos maliciosos que implementen estrategias de cooperación entre ellos son también contrarrestados gracias a las medidas de propagación de los Score bajos.

Estos resultados han quedado plasmados en la simulación de comportamiento del sistema llevada a cabo en la fase final de la implementación del mismo, y que podemos observar en el capítulo “Desarrollo, despliegue y testing de la solución”.

Sin embargo, se han detectado ciertas limitaciones en nuestro sistema que podrían abordarse en el futuro con vistas a mejorar la solución:

- En primer lugar, existe un problema de escalabilidad y coste con la estructura planteada. Ya que cada interacción entre dos nodos se va almacenando en el Smart Contract, aplicar el algoritmo de confianza se irá haciendo cada vez más caro. Esto podría reducir la participación de los usuarios. Este problema se puede intentar solucionar de varias maneras:
    - Intentar añadir la nueva variable de Confianza dentro de la propia blockchain, como parte de la estructura de una Cuenta de usuario. Esta solución, sin embargo, conllevaría generar un tipo de blockchain nuevo, diferente de Ethereum o como un hard-fork[4] de la misma.
    - Modificar la estructura de almacenamiento de los datos que se necesitan para el algoritmo de confianza. Se podría intentar utilizar algún sistema externo a la blockchain, por ejemplo, IPFS para almacenar los valores que se necesitan para aplicar el algoritmo de confianza.
    - Ejecutar los procesos computacionalmente complejos del Smart Contract en algún entorno off-chain. A día de hoy existen pocas alternativas a este

[4] Un hard-fork de una blockchain representa la separación de la blockchain principal en dos blockchain a partir de un bloque concreto. Estas dos blockchain tendrán la misma estructura que la blockchain principal a excepción de ciertos cambios que las harán diferentes entre ellas. Estas diferencias suelen ser de tipo estructural: tamaño del bloque, estructura de la información almacenada,…

respecto, pero si en un futuro aparecen nuevas soluciones, se podría evitar el problema de la escalabilidad y coste.

- La necesidad de hacer uso de un SmartContract externo para ofrecer el Sistema de Confianza supone una adaptación que puede ser costosa para las Dapps existentes, ya que necesitarían modificar su código para integrar las llamadas a nuestro Smart Contract.
- En ciertos escenarios, hacer pagar al usuario que emite la valoración puede no ser interesante. En ciertas ocasiones puede resultar más lógico que sea el usuario que desea recibir la valoración el que pague la transacción.
- Aunque se logra detectar a los nodos maliciosos rápidamente, es preciso que existan valoraciones negativas hacia estos nodos antes de poder ser detectados. Por lo tanto, se puede decir que alguien será víctima al menos una vez antes de poder identificar estos nodos.
- Por último, el algoritmo no se comporta muy bien con las redes de nodos maliciosos. El algoritmo permite "castigar" estas redes de nodos al replicar las valoraciones negativas a través de la red de confianza de un nodo. Sin embargo, la valoración final de un nodo se obtiene aplicando una media de las valoraciones recibidas, con lo que el impacto que esta medida tiene en el Score final de confianza de un nodo, puede resultar muy pequeño. Cuantas más transacciones se realicen entre los nodos de la red maliciosa, menos afectarán las puntuaciones negativas recibidas. La solución a este problema pasa por hacer evolucionar el algoritmo para que tenga en cuenta este comportamiento.

6.2 Objetivos conseguidos

El objetivo principal del Trabajo consistió en realizar un análisis en profundidad del estado del arte actual sobre las soluciones de confianza entre pares dentro de una red blockchain, y desarrollar y desplegar una herramienta que permitiera crear una red de confianza distribuida entre los usuarios de la red.

Este objetivo principal se dividió en 5 objetivos más específicos que utilizamos ahora para medir la consecución del trabajo.

Objetivo O1: Realizar un análisis del estado del arte sobre redes de confianza (Web of Trust) y de reputación sobre Blockchain

En el capítulo "Análisis del Estado del Arte sobre redes de confianza (Web of Trust) y sistemas de reputación sobre Blockchain" se realiza un análisis sobre los algoritmos existentes sobre redes de confianza y sistemas de reputación. Este análisis sirvió para obtener conocimientos sobre el algoritmo a desarrollar en base a investigaciones previas.

Objetivo O2: Diseñar una solución de Web of Trust sobre blockchain, basada en el análisis previo, en forma de DApp

Tras realizar el análisis del estado del arte y el análisis sobre redes de blockchain existentes, en el documento "Descripción y diseño de la solución a desarrollar" se diseña el sistema propuesto en el Trabajo. En primer lugar se definen los requisitos funcionales que debe

cumplir el sistema y seguidamente se diseña tanto la estructura del Smart Contract que gestionaría las funcionalidades del sistema, como el Bridge de comunicación con el mismo y las interfaces que nos ayudan a probar la solución.

Objetivo O3: Desarrollar la solución diseñada en la red distribuida Ethereum

Tras obtener el diseño del sistema, se desarrolla el mismo utilizando las herramientas de desarrollo existentes para Dapps: Truffle, Ganache, Metamask, Web3,… Este proceso está detallado en el "Anexo 1. Desarrollo, despliegue y testing de la solución".

Objetivo O4: Analizar la viabilidad de la solución desde el punto de vista de varios casos de uso y Objetivo O5: Establecer los próximos pasos a dar de la solución obtenida

En este capítulo se detalla tanto la viabilidad del sistema desarrollado como los próximos pasos que sería interesante dar para mejorar la solución obtenida. Se ha realizado una simulación de cómo se comportaría el sistema en un entorno real, obteniendo resultados positivos. Esta simulación ha quedado plasmada en el capítulo "Desarrollo, despliegue y testing de la solución".

Por lo tanto, podemos decir que se han cumplido los objetivos específicos establecidos para este proyecto. En cuando al objetivo general, según el análisis de viabilidad realizado en la sección anterior, se constata que este sistema podría aportar un valor añadido a las redes de blockchain existentes. En la siguiente sección se mencionan las contribuciones específicas que este proyecto ha hecho en el ámbito tecnológico de blockchain.

6.3 Contribuciones realizadas al ámbito tecnológico de blockchain

Este trabajo surgió para buscar una solución al problema que existe actualmente de falta de confianza entre dos usuarios que deciden interactuar entre ellos dentro de una red de Blockchain. Como se mencionaba al principio, cuando dos usuarios desean realizar un intercambio de valor entre ellos, generalmente acuden a un tercero para gestionar este intercambio. Estos terceros, sin embargo, necesitan que los usuarios se identifiquen apropiadamente como medida de seguridad de su servicio. Al realizar esta acción, los usuarios pierden la anonimidad que les otorga por defecto la red de blockchain.

El sistema desarrollado propone una primera solución para este problema. Es la actividad que los usuarios tienen en la red la que les otorga la reputación necesaria para ser confiables de cara a otros usuarios con los que interactuar. Esto mejora directamente la salud de la red, ya que los nodos que se comportan de manera adecuada son vistos como usuarios confiables, mientras que los nodos que no lo hacen, son vistos como usuarios peligrosos.

Relacionado con lo anterior, este sistema permite recuperar uno de los pilares básicos de blockchain: el anonimato. No es necesario que un usuario tenga que identificarse para poder realizar transacciones con usuarios desconocidos de manera segura, sino que puede basarse en la puntuación de confianza de un usuario para decidir si desea operar con él.

Por último, esta solución se ha desarrollado para que pueda ser utilizada desde otros Smart Contracts, con lo que es fácilmente integrable en otras Dapps que deseen hacer uso del mismo para mejorar el servicio ofrecido.

6.4 Próximos pasos para la mejora de la solución propuesta

Tal y como se ha mencionado, el sistema propuesto es un primer paso que soluciona el problema existente, pero existen bastantes mejoras que se pueden plantear para obtener una solución más robusta y que reduzca las limitaciones detectadas anteriormente.

Mejorar el algoritmo de confianza obteniendo más información con la que obtener un Score de confianza más representativo

El algoritmo de confianza actual es bastante sencillo y se basa únicamente en las valoraciones otorgadas por los usuarios. Existen otras variables que se pueden utilizar para mejorar el algoritmo y calcular un Score de confianza más representativo de la actividad del usuario en la red. Algunos ejemplos son el número de transacciones efectuadas por el usuario en la red, o el número de usuarios con los que ha interactuado.

Reducir el coste utilizando un sistema de almacenamiento (por ejemplo, IPFS) y una estructura en árbol Merkle para almacenar la matriz de confianza

Uno de los factores que más incrementa el coste del Smart Contract es el almacenamiento de la red de confianza dentro del propio contrato. Se podría utilizar un sistema de almacenamiento externo, como puede ser IPFS (https://ipfs.io/), para liberar el contrato de las tareas de almacenamiento y reducir, así, el coste de las operaciones disponibles.

Ofrecer otros escenarios de pago de las acciones del Smart Contract

En la solución desarrollada, el usuario valorador es el encargado de pagar la operación de Valoración. En ciertos escenarios, sin embargo, puede ser interesante que sea el usuario que quiere ser valorado el que pague la operación, o incluso ofrecer la posibilidad a un Smart Contract que integre nuestra solución, de que asuma el cargo.

Incorporar el propio sistema de reputación y confianza dentro de la estructura de la blockchain.

Tal y como se almacenan transacciones y estados de las cuentas, se podrían añadir las valoraciones realizadas y la matriz de confianza como parte del estado de una cuenta, dentro de los bloques de la Blockchain. La matriz de confianza podría estructurarse como un árbol de Merkle para ahorrar espacio.

7 Bibliografía

AirBnB. (s.f.). *¿Cómo funcionan las valoraciones por estrellas?* Obtenido de AirBnB: https://www.airbnb.es/help/article/1257/c%C3%B3mo-funcionan-las-valoraciones-por-estrellas

AMadrid. (2019). Blockchain determinará el futuro de los sistemas empresariales y las grandes firmas mundiales así lo confirman. Madrid.

Amazon. (s.f.). *Acerca de las opiniones*. Obtenido de Amazon: https://www.amazon.es/gp/help/customer/display.html?nodeId=201967050&qid=1571568379

Bhattacharya, M., & Nesa, N. (2016). An Algorithm for Predicting Local Trust based on Trust Propagation in Online Social Networks. *156* (7).

Binance. (s.f.). *Binance*. Obtenido de https://www.binance.com/es

Buterin, V. (Septiembre de 2014). *A Next-Generation Smart Contract and Decentralized Application Platform.* Obtenido de GitHub: https://github.com/ethereum/wiki/wiki/White-Paper

Cadena de Bloques. (25 de Febrero de 2015). Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Cadena_de_bloques

Coinbase. (s.f.). *Coinbase*. Obtenido de https://www.coinbase.com/

Corporation, P. (Noviembre de 2007). *RFC4880: OpenPGP Message Format.* Obtenido de IETF: https://tools.ietf.org/html/rfc4880

EscrowMyEther. (14 de Mayo de 2019). *Homepage*. Obtenido de EscrowMyEther: http://escrowmyether.com/

Fintech, O. (28 de Septiembre de 2017). *Axa presenta Fizzy, un seguro de vuelo basado en la 'blockchain' de Ethereum.* Obtenido de FinTech Observatorio: https://www.fin-tech.es/2017/09/axa-fizzy-blockchain-ethereum.html

García, C. C. (2018). 'Blockchain', la tecnología que revolucionará el futuro.

Hayes, A. (25 de Junio de 2019). *How does Bitcoin Mining work?* Obtenido de Investopedia: https://www.investopedia.com/tech/how-does-bitcoin-mining-work/

Hidalgo, K. (29 de Julio de 2019). *Marco legal de las criptomonedas en el mundo.* Obtenido de Mente Diamante: https://mentediamante.com/blog/marco-legal-criptomonedas

Hyperledger. (9 de Febrero de 2016). *Linux Foundation's Hyperledger Project Announces 30 Founding Members and Code Proposals To Advance Blockchain Technology.* Obtenido de Hyperledger: https://www.hyperledger.org/announcements/2016/02/09/linux-foundations-hyperledger-project-announces-30-founding-members-and-code-proposals-to-advance-blockchain-technology

Kamvar, S., & Schlosser, M. G.-M. (2003). The EigenTrust Algorithm for Reputation Management in P2P Networks. *I* (1).

Leiva-Aguilera, J. (2012). *Gesión de la reputación online.* Editorial UOC.

Mao, Y., & Shen, H. (2016). Web of Credit: Adaptive Personalized Trust Network Inference From Online Rating Data. *3* (4).

Nakamoto, S. (9 de Enero de 2009). *Bitcoin: A Peer-to-Peer Electronic Cash System.* Obtenido de Bitcoin.org: https://bitcoin.org/bitcoin.pdf

Page, L. (2006). *Patente nº 09/895.174.* Estados Unidos.

Parrondo, L. (2018). El Blockchain, ¿la tecnología del futuro? Barcelona.

Premier, E. (14 de Octubre de 2019). *How will blockchain transform the stock market?* Obtenido de Hackernoon: https://hackernoon.com/how-will-blockchain-transform-the-stock-market-cd41c79c51be

Thomasson, E. (3 de Junio de 2019). *Carrefour says blockchain tracking boosting sales of some products.* Obtenido de Reuters: https://www.reuters.com/article/us-carrefour-blockchain/carrefour-says-blockchain-tracking-boosting-sales-of-some-products-idUSKCN1T42A5

Thomasson, E. (3 de Junio de 2019). *Carrefour says blockchain tracking boosting sales of some products.* Obtenido de Reuters: https://www.reuters.com/article/us-carrefour-blockchain/carrefour-says-blockchain-tracking-boosting-sales-of-some-products-idUSKCN1T42A5

Xiong, L., & Liu, L. (2002). Building Trust in Decentralized Peer-to-Peer Electronic Communities.

Zhang H., v. M. (2008). Evaluation of P2P Algorithms for Probabilistic Trust Inference in a Web of Trust. En J. C. Thomas N., *Computer Performance Engineering.* Berlin: Springer.

Zimmerman, P. (6 de Marzo de 1993). *Pretty Good Privacy: Public Key Encryption for the Masses.* Obtenido de Repositorio MIT: http://www.mit.edu/afs.new/sipb/project/pgp/doc/pgpdoc1.txt

8 Anexos

8.1 Anexo 1. Desarrollo, despliegue y testing de la solución

En este anexo, se describe el entorno de trabajo utilizado y la estructura global del proyecto de desarrollo del Sistema. Seguidamente, se describe el desarrollo realizado para obtener los diferentes componentes del sistema. Además, se incluye el desarrollo de un sistema de simulación de transacciones y votaciones aleatorias para validar la solución. Por último, se explicará el proceso seguido para configurar y desplegar el sistema en una red de test pública.

Como información adicional, para el desarrollo de este proyecto se ha utilizado una máquina con el Sistema Operativo Windows 10, el servidor web Apache HTTP (https://httpd.apache.org/) disponible y el Node Package Manager (NPM) (https://www.npmjs.com/), previamente instalado.

8.1.1 Descripción del entorno de trabajo y la estructura del proyecto

Dada la complejidad que conlleva cualquier proyecto actual de desarrollo de Dapps, el entorno de trabajo debe estar claramente definido y estructurado para asegurar un eficiente proceso de desarrollo y la buena conexión entre los diferentes componentes.

Para el sistema de confianza y reputación, objeto de este proyecto, se han identificado 4 componentes principales en su estructura:

- **Truffle framework**: Este entorno de trabajo ofrece las herramientas necesarias para programar el Smart Contract del sistema utilizando el lenguaje de programación Solidity, y para compilarlo en *bytecode*, estructura necesaria para ser interpretado por la *EVM* (Ethereum Virtual Machine). Además, Truffle ofrece herramientas de testing unitario para el Smart Contract y permite desplegarlo en una red de Ethereum. El primer entorno de desarrollo que crearemos será utilizando Truffle para generar el Smart Contract.
- **Node y Webpack**: Estas dos tecnologías juntas facilitan el desarrollo de cualquier aplicación web basada en JavaScript. Node permite crear un entorno de trabajo para la aplicación web y gestionar los paquetes JS necesarios para el desarrollo de la funcionalidad dinámica de la aplicación. Webpack, que necesita Node para poder ser utilizada, permite compilar y optimizar la aplicación para mejorar el rendimiento en su versión final. Ambas tecnologías serán necesarias para desarrollar tanto los front-ends como el Bridge. Generaremos un segundo entorno de desarrollo utilizando Node y Webpack para desarrollar ambos componentes del sistema.
- **Metamask**: Esta aplicación permite comunicar una cuenta en una red de Ethereum (ya sea en “Producción”, en una red de test pública, o en local) con una Dapp a través de una interfaz web. Utilizaremos Metamask para realizar las pruebas desde el front-end desarrollado.
- **Ganache**: Esta aplicación facilita la labor de desarrollo y testing de un Smart Contract en un entorno de red de blockchain similar al de “Producción”. Esencialmente, Ganache crea un nodo de blockchain Ethereum en local y genera varias cuentas aleatorias, permitiendo desplegar Smart Contracts compilados a este nodo y realizar las operaciones necesarias para realizar las tareas de testing. Utilizaremos Ganache para generar un nodo de blockchain Ethereum en local, desplegar el Smart Contract

compilado en Truffle sobre ese nodo, y realizar las pruebas necesarias desde el front-end junto con Metamask o desde la consola de Truffle.

En las siguientes secciones, se detalla el proceso seguido con cada uno de estos componentes para obtener el resultado final del sistema desarrollado.

8.1.2 Desarrollo del Smart Contract con Truffle Framework

Se comienza el desarrollo del sistema por el Smart Contract, que contiene las funciones necesarias para generar el sistema de confianza y reputación. Para ello, se cuenta con la ayuda del entorno de trabajo Truffle Framework (https://www.trufflesuite.com/truffle), el cual se instala utilizando NPM con el comando siguiente:

```
npm install truffle -g
```

Seguidamente, se crea la carpeta del proyecto Truffle que contendrá el código del Smart Contract y nos permitirá utilizar las herramientas de Truffle, y se inicia el proyecto Truffle:

```
josef@DESKTOP-8LVS0OR Z:\Informatics\Development\Ethereum
$ mkdir dtrustmanager

josef@DESKTOP-8LVS0OR Z:\Informatics\Development\Ethereum
$ cd dtrustmanager

josef@DESKTOP-8LVS0OR Z:\Informatics\Development\Ethereum\dtrustmanager
$ truffle init

√ Preparing to download
√ Downloading
√ Cleaning up temporary files
√ Setting up box

Unbox successful. Sweet!

Commands:

  Compile:        truffle compile
  Migrate:        truffle migrate
  Test contracts: truffle test

josef@DESKTOP-8LVS0OR Z:\Informatics\Development\Ethereum\dtrustmanager
$ dir
 El volumen de la unidad Z es Data
 El número de serie del volumen es: A0B9-5BDF

 Directorio de Z:\Informatics\Development\Ethereum\dtrustmanager

07/12/2019  13:56    <DIR>          .
07/12/2019  13:56    <DIR>          ..
07/12/2019  13:56    <DIR>          contracts
07/12/2019  13:56    <DIR>          migrations
07/12/2019  13:56    <DIR>          test
07/12/2019  13:56             4.233 truffle-config.js
               1 archivos          4.233 bytes
               5 dirs  179.729.932.288 bytes libres
```

Tenemos la siguiente estructura de archivos:

- **Contracts**: Donde se aloja el código en Solidity de los Smart Contract
- **Migrations**: Donde se alojan los ficheros de configuración del despliegue de los Smart Contract en la red de Ethereum correspondiente
- **Test**: Donde se alojan los ficheros de testing unitario
- **Truffle-config.js** : Fichero que contiene la configuración del proyecto y de las herramientas disponibles por Truffle

Creamos ahora, el Smart Contract que gestione el sistema de reputación y confianza. Lo llamaremos DTrustManager y lo almacenaremos en la carpeta Contracts. Para desarrollarlo, se ha seguido la estructura definida en el capítulo "Descripción y diseño de la solución a desarrollar". El resultado final obtenido es el siguiente:

```
pragma solidity ^0.5.0;

contract DTrustManager
{
    ////////////
    // Events //
    ////////////
    event Rating(address indexed from, address indexed to, uint8 indexed score);

    ///////////////
    // Constants //
    ///////////////
    // Score constants
    uint8 constant LOW_SCORE = 1;
    uint8 constant MED_SCORE = 2;
    uint8 constant HIGH_SCORE = 3;

    // Trust matrix constants
    uint8 constant MAX_TRUST_DEPTH = 2;

    ////////////////////////
    // Managers variables //
    ////////////////////////
    address private _owner;
    mapping(address => bool) _operators;

    ///////////////////////
    // Storage variables //
    ///////////////////////
    // Received ratings
    mapping(address => uint8[]) private _ratings;

    // Ratings given by users
    mapping(address => mapping(address => uint8)) private _scores;

    // Trust graph
    mapping(address => address[]) private _trust_matrix;

    // Extra information (this struct might hold more information)
    struct extraInformation
    {
        uint256 network_transactions;
    }
    mapping(address => extraInformation) _user_extra_information;
```

```
    //////////////
    // Helpers //
    //////////////
    /**
     * @dev Returns true if the caller is an operator
     */
    function isOperator() internal view returns (bool)
    {
        return _operators[msg.sender];
    }

    /**
     * @dev Throws if called by any account that is not an operator
     */
    modifier onlyOperator()
    {
        require(isOperator(), "DTrustManager: caller is not the owner");
        _;
    }

    /**
     * @dev Checks whether a score is a valid score
     */
    function _isValidScore(uint8 score) internal pure returns (bool)
    {
        return
        (
            (score == LOW_SCORE) ||   // Lowest score (Possible scam)
            (score == MED_SCORE) ||   // Medium score
            (score == HIGH_SCORE)     // Highest score (maximum trust)
        );
    }


    ///////////////////////
    // Private functions //
    ///////////////////////
    /**
    * @dev Finds whether a target (user) is in the trust matrix of a user
    *       (A one parameter overriden function is provided to find whether a user is
in the trust network of the user calling the function)
    */
    function  _userIsInTrustNetwork(address  user,  address  target,  uint8  level)
internal view returns(bool)
    {
        if (level > MAX_TRUST_DEPTH) return false;

        bool userIsInTrustNetwork = false;
        for (uint16 i = 0; i < _trust_matrix[user].length; i++)
        {
            address trusted_node = _trust_matrix[user][i];
            if  (  (trusted_node  ==  target)  ||  _userIsInTrustNetwork(trusted_node,
target, level+1) )
            {
                userIsInTrustNetwork = true;
            }
        }

        return userIsInTrustNetwork;
    }
    function  _userIsInTrustNetwork(address  user,  address  target)  internal  view
returns(bool) { return _userIsInTrustNetwork(user, target, 1); }

    /**
    * @dev Punishing the trust network of `user`
```

```
    *    (gives an extra score of LOW_SCORE to all users that `user` trusts)
    */
    function _punishNetwork(address user) internal
    {
        for (uint16 i = 0; i < _trust_matrix[user].length; i++)
        {
            address trusted_node = _trust_matrix[user][i];
            _ratings[trusted_node].push(LOW_SCORE);
            _scores[msg.sender][trusted_node] = LOW_SCORE;
        }
    }

    ////////////////////////////////////
    // Public Operator-only functions //
    ////////////////////////////////////
    /**
    * @dev Adds a user's number of transactions as extra information to be used in
the algorithm
    *       (Function only accessible by the operators)
    */
    function addUserExtraInformation(address user, uint256 number_of_transactions)
public onlyOperator
    {
        extraInformation memory extra_information = extraInformation(
            {
                network_transactions: number_of_transactions
            }
        );
        _user_extra_information[user] = extra_information;
    }


    //////////////////////
    // Public functions //
    //////////////////////
    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor () public
    {
        _owner = msg.sender;
        _operators[_owner] = true;
    }

    /**
     * @dev Adds a new operator
     */
    function addOperator(address operator) public onlyOperator
    {
        _operators[operator] = true;
    }

    /**
     * @dev Removes an operator
     */
    function removeOperator(address operator) public onlyOperator
    {
        _operators[operator] = false;
    }

    /**
    * @dev Rates `user`
    */
    function rateUser(address user, uint8 score) public
    {
        require(user != address(0), "DTrustManager: Rating the zero address");
```

```
        require(_isValidScore(score), "DTrustManager: Score is not valid");

        // Saving the score in the proper matrices
        _ratings[user].push(score);
        _scores[msg.sender][user] = score;

        // Punishing network, if we have the lowest score
        if (score == LOW_SCORE)
        {
            _punishNetwork(user);
        }

        // Updating the trust graph
        if (score == HIGH_SCORE)
        {
            _trust_matrix[msg.sender].push(user);
        }

        // Emitting the appropriate event
        emit Rating(msg.sender, user, score);
    }

    /**
    * @dev Returns the general Trust Score of `user`
    */
    function getGeneralTrustScore(address user) public view returns (uint8)
    {
        uint16 n_ratings = 0;
        uint256 total_ratings = 0;

        for (uint16 i = 0; i < _ratings[user].length; i++)
        {
            total_ratings += _ratings[user][i];
            n_ratings++;
        }

        // If no ratings have been made, the medium score is returned (not good nor
bad)
        if (n_ratings == 0)
        {
            return MED_SCORE;
        }

        return uint8(total_ratings/n_ratings);
    }

    /**
    * @dev Returns the Trust Score relative to `user` as per user `from`
    *        (A one parameter overriden function is provided to get the trust score
relative to the user calling the function)
    */
    function getRelativeTrustScore(address from, address user) public view returns
(uint8)
    {
        // If user "from" already rated this user, we return that value
        if (_scores[from][user] > 0) return _scores[from][user];

        // Otherwise, we return the medium score (not good nor bad)
        return MED_SCORE;
    }
    function getRelativeTrustScore(address user) public view returns (uint8) { return
getRelativeTrustScore(msg.sender, user); }

    /**
    * @dev Returns the Trust Score relative to `user` as per user `from` or the
general score if there is no relation between them
```

```
    *        (A one parameter overriden function is provided to get the trust score
relative to the user calling the function)
    */
    function getTrustScore(address from, address user) public view returns (uint8)
    {
        // If user "from" already rated this user, we return that value
        if (_scores[from][user] > 0) return _scores[from][user];

        // If user is in the trust network of user "from", we return the high score
        if (_userIsInTrustNetwork(from, user)) return HIGH_SCORE;

        // Otherwise, we return the general trust score
        return getGeneralTrustScore(user);
    }
    function  getTrustScore(address  user)  public  view  returns  (uint8)  {  return
getTrustScore(msg.sender, user); }
}
```

Finalmente, compilamos el Smart Contract para obtener el *bytecode* necesario para migrarlo a la blockchain.

```
josef@DESKTOP-8LVS0OR Z:\Informatics\Development\Ethereum\dtrustmanager
$ truffle compile

Compiling your contracts...
===========================
> Compiling .\contracts\DTrustManager.sol
> Compiling .\contracts\Migrations.sol
>                     Artifacts                     written                     to
Z:\Informatics\Development\Ethereum\dtrustmanager\build\contracts
> Compiled successfully using:
   - solc: 0.5.8+commit.23d335f2.Emscripten.clang
```

8.1.3 Inicialización del nodo local de Ethereum y despliegue del Smart Contract

Ahora que tenemos el Smart Contract desarrollado y compilado, lo desplegaremos en un nodo local en el que realizar las diferentes pruebas con el mismo.

Comenzamos descargando e instalando Ganache (https://www.trufflesuite.com/ganache). Al iniciarlo, podemos crear un Espacio de trabajo (Workspace) para este proyecto.

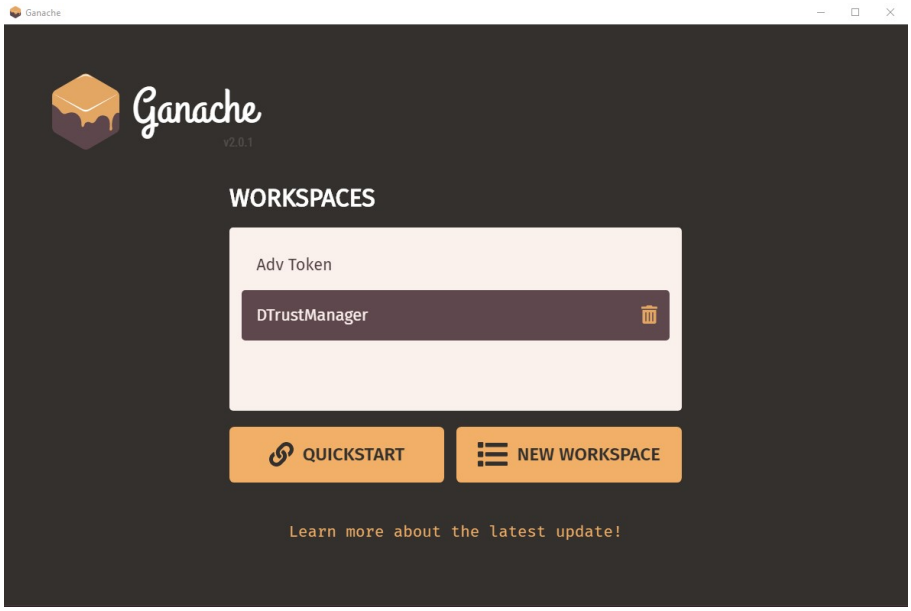


Ilustración 13: Creación de un espacio de trabajo en Ganache. Imagen propia.

Al hacer esto, Ganache inicializará un nodo de Ethereum en local y generará 10 direcciones con 100 ETH cada una.

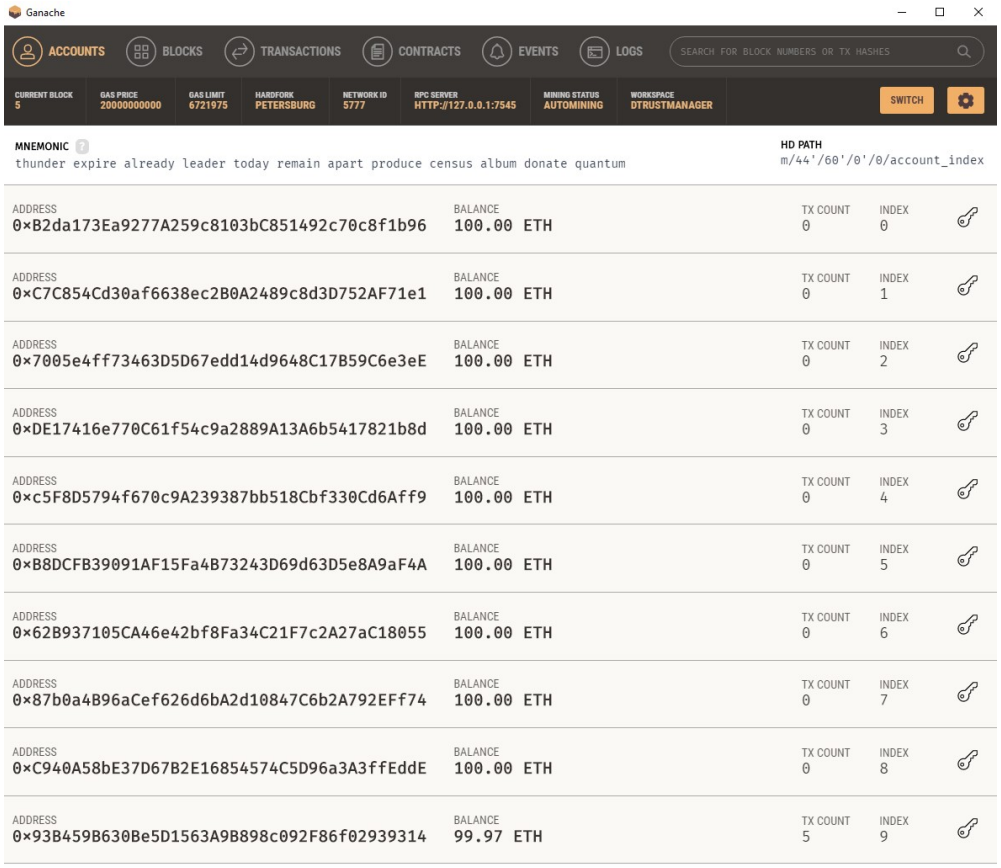


Ilustración 14: Lista de cuentas disponibles en Ganache. Imagen propia.

Desde esta ventana, podemos ver las direcciones que tenemos disponibles y en qué URL podemos encontrar el servidor para desplegar nuestro Smart Contract, en este caso http://127.0.0.1:7545 .

Con esta información, modificamos el fichero Truffle-config.js para indicar dónde queremos desplegar el Smart Contract. Este fichero se compone de varios bloques de configuración, pero el que nos interesa es "networks". En este bloque, podemos definir las diferentes redes con las que interactuaremos. Para empezar, crearemos la red "development".

```
module.exports = {
  networks: {
    // Useful for testing. The `development` name is special -
 truffle uses it by default
    // if it's defined here and no other network is specified at the command line.
    // You should run a client (like ganache-
cli, geth or parity) in a separate terminal
    // tab if you use this network and you must also set the `host`, `port` and `netw
ork_id`
    // options below to some value.
    //
    development: {
      host: "127.0.0.1",     // Localhost (default: none)
      port: 7545,            // Standard Ethereum port (default: none)
      network_id: "*",       // Any network (default: none)
      from: "0x93B459B630Be5D1563A9B898c092F86f02939314"
    },
  },
}
```

Cabe destacar, que hemos incluido una de las direcciones ofrecidas por Ganache como la cuenta desde la cual realizar la migración. Al realizar la migración, esta cuenta "0x93B459B630Be5D1563A9B898c092F86f02939314" será la propietaria del Smart Contract a desplegar, además de que será la cuenta que asumirá los costes del despliegue.

Con esta configuración, procedemos a migrar el Smart Contract a nuestro nodo local. Para ello, necesitaremos primero definir un fichero de migración en la carpeta "migrations".

```
const DTrustManager = artifacts.require("DTrustManager");

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

Ahora, procedemos a ejecutar el comando que realiza la migración del contracto DTrustManager a la blockchain local.

```
josef@DESKTOP-8LVS0OR Z:\Informatics\Development\Ethereum\dtrustmanager
$ truffle migrate

Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.


Starting migrations...
======================
> Network name:    'development'
> Network id:      5777
```

```
> Block gas limit: 0x6691b7


1_dtrust_manager.js
===================

   Deploying 'DTrustManager'
   -------------------------
   >                              transaction                                    hash:
0x58ad16ca989aeae7b0caa7d35bd06943c11c1a4c76d61d1f7000001d8980d58c
   > Blocks: 0            Seconds: 0
   > contract address:    0x4B6c8d10BbF57ffE36F8D2fe28beb332d824572C
   > block number:        6
   > block timestamp:     1575736345
   > account:             0x93B459B630Be5D1563A9B898c092F86f02939314
   > balance:             99.95404548
   > gas used:            702140
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.0140428 ETH

   > Saving artifacts
   -------------------------------------
   > Total cost:           0.0140428 ETH


Summary
=======
> Total deployments:   1
> Final cost:          0.0140428 ETH
```

Los resultados de la ejecución de este comando nos dan mucha información:

- Propietario del contrato: valor del parámetro “account”, que coincide con lo que habíamos definido en la configuración.
- Dirección del contrato: valor del parámetro “Contract address”.
- Gas utilizado: valor del parámetro “Gas used”. Esta información nos es importante si queremos calcular cuánto nos costará desplegar nuestro contrato en producción.

8.1.4 Desarrollo del Front-end con HTML y JS

Teniendo el Smart Contract desarrollado y desplegado en el nodo local, pasamos ahora a crear el front-end con el que nos conectaremos al Smart Contract a través del Bridge (que desarrollaremos en la siguiente sección).

Para este front-end, crearemos los ficheros directamente en una carpeta de la raíz del servidor local Apache. De esta manera, los ficheros estarán disponibles a través de nuestro navegador como si se tratara de un servidor web en Internet.

El objetivo de esta interfaz no es otro que el de probar el Smart Contract. Por lo tanto, crearemos una interfaz sencilla, que nos permita acceder a las funcionalidades necesarias como lo haría un usuario del Smart Contract.

En primer lugar, creamos un fichero HTML con el código de la interfaz que queremos pintar.

```
<!DOCTYPE html>
```

```
<html lang="es">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1">

        <title>DTrustManager</title>

        <!-- Bootstrap -->
        <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4
.1/css/bootstrap.min.css" integrity="sha384-
Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin="anonym
ous" />
    </head>
    <body>
        <div class="container">
            <div class="row justify-content-center">
                <div class="col12">
                    <h1 class="text-center">DTrustManager</h1>
                </div>
            </div>
        </div>

        <div class="container" id="rate-user">
            <div class="row justify-content-center block-title">
                <div class="col-12">
                    <h3 class="text-center">Valorar usuario</h3>
                </div>
            </div>
            <div class="row justify-content-center">
                <div class="col-3 form-group">
                    <label for="user-address">Dirección</label>
                    <input type="text" name="user-address" id="rate-user-
address" class="form-control" placeholder="0x123..." />
                </div>
                <div class="col-3 form-group">
                    <label for="rate-user-rating">Valoración</label>
                    <select name="rate-user-rating" id="rate-user-
rating" class="form-control">
                        <option value="1">1 - La transacción no ocurrió</option>
                        <option value="2">2 - Tuve algún problema</option>
                        <option value="3">3 - Buena transacción</option>
                    </select>
                </div>
            </div>
            <div class="row justify-content-center">
                <div class="col-12 text-center">
                    <button class="btn btn-primary" id="rate-user-
trigger">Enviar</button>
                </div>
            </div>
            <div class="row justify-content-center result-block">
                <div class="col-8 alert alert-primary text-center">
                    <span></span>
                </div>
            </div>
        </div>

        <div class="container" id="get-user-score">
            <div class="row justify-content-center block-title">
                <div class="col-12">
                    <h3 class="text-center">Obtener Score de confianza</h3>
                </div>
            </div>
            <div class="row justify-content-center">
                <div class="col-3 form-group">
```

```
                    <label for="user-address">Dirección</label>
                    <input type="text" name="user-address" id="get-user-score-
address" class="form-control" placeholder="0x123..." />
                </div>
            </div>
            <div class="row justify-content-center">
                <div class="col-12 text-center">
                    <button class="btn btn-primary" id="get-user-score-
trigger">Enviar</button>
                </div>
            </div>
            <div class="row justify-content-center result-block">
                <div class="col-8 alert alert-primary text-center">
                    <span></span>
                </div>
            </div>
        </div>

        <div class="container metamask-information">
            <div class="row">
                <div class="col-12">
                    <h4 class="text-center">Tu información</h4>
                    <p class="text-center">
                        <span id="connected-address"></span>
                        <br />
                        <span id="connected-trust-score"></span>
                    </p>
                </div>
            </div>
        </div>

        <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
        <script src="https://code.jquery.com/jquery-
3.4.1.slim.min.js" integrity="sha384-
J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n" crossorigin="anonym
ous"></script>
        <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.mi
n.js" integrity="sha384-
Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo" crossorigin="anonym
ous"></script>
        <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.
min.js" integrity="sha384-
wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6" crossorigin="anonym
ous"></script>
    </body>
</html>
```

Hemos aprovechado para integrar la librería Bootstrap (https://getbootstrap.com/), la cual nos ofrece muchos elementos de diseño predefinidos, y la librería jQuery (https://jquery.com/) de JavaScript, para permitirnos manejar más fácilmente el DOM de nuestra aplicación.

Añadimos también algunos estilos básicos en CSS para mejorar la estructura de la página.

```
body { margin-top: 30px; }

.container { margin-bottom: 30px; }

.block-title { margin-bottom: 15px; }

label
{
    display: block;
    font-weight: 600;
}
```

El resultado final es el siguiente:

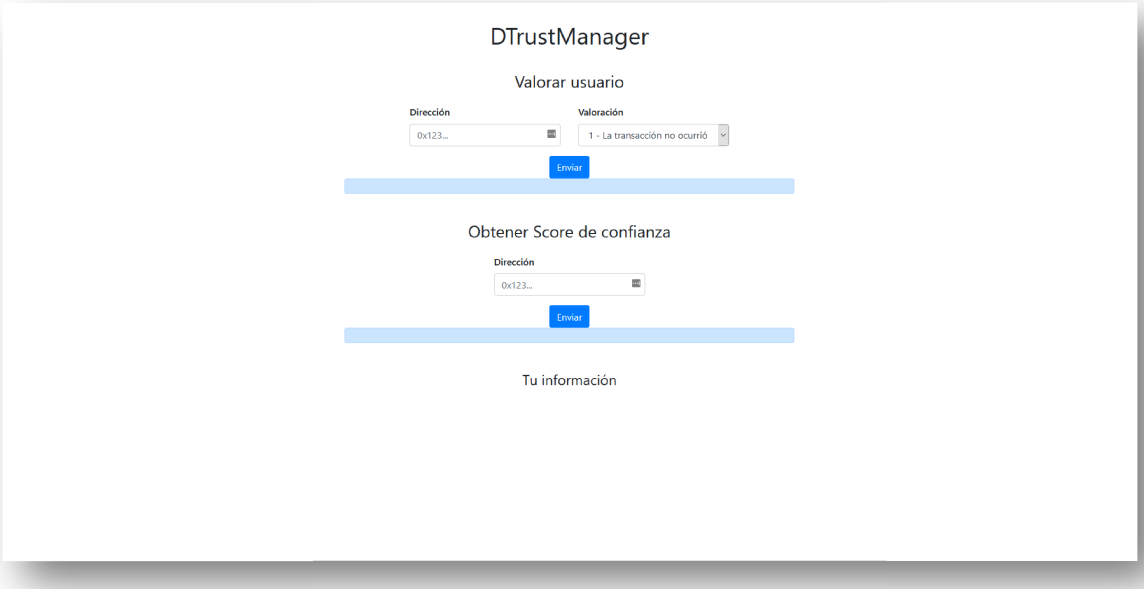


Ilustración 15: Front-end inicial. Imagen propia.

8.1.5 Desarrollo del Bridge Front- SmartContract con Node y Web3JS

La última pieza que nos falta para finalizar el sistema de reputación y confianza es el vínculo entre el front-end creado y el Smart Contract desplegado. Este vínculo lo realizaremos mediante un Bridge que permitirá que ciertas acciones realizadas en la interfaz, ejecuten funcionalidades del Smart Contract.

En la interfaz que hemos desarrollado, estas acciones serán dos:

- Valorar usuario, que llamará a la función rateUser() del Smart Contract
- Obtener Score de confianza, que llamará a la función getTrustScore() del Smart Contract.

Para desarrollar este Bridge, crearemos un proyecto utilizando Node y Webpack. Comenzamos, pues, creando una carpeta para este proyecto e instalando webpack.

```
josef@DESKTOP-8LVS0OR Z:\Informatics\Development\PHP
$ mkdir dtrustbridge

josef@DESKTOP-8LVS0OR Z:\Informatics\Development\PHP
$ cd dtrustbridge

josef@DESKTOP-8LVS0OR Z:\Informatics\Development\PHP\dtrustbridge
$ npm install webpack webpack-cli --save-dev
npm WARN deprecated fsevents@1.2.9: One of your dependencies needs to upgrade to
fsevents v2: 1) Proper nodejs v10+ support 2) No more fetching binaries from AWS,
smaller package size
npm WARN saveError ENOENT: no such file or directory, open
'Z:\Informatics\Development\PHP\dtrustbridge\package.json'
npm notice created a lockfile as package-lock.json. You should commit this file.
```

```
npm WARN enoent ENOENT: no such file or directory, open
'Z:\Informatics\Development\PHP\dtrustbridge\package.json'
npm WARN dtrustbridge No description
npm WARN dtrustbridge No repository field.
npm WARN dtrustbridge No README data
npm WARN dtrustbridge No license field.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9
(node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for
fsevents@1.2.9: wanted {"os":"darwin","arch":"any"} (current:
{"os":"win32","arch":"x64"})

+ webpack-cli@3.3.10
+ webpack@4.41.2
added 386 packages from 217 contributors and audited 5285 packages in 38.233s
found 0 vulnerabilities


josef@DESKTOP-8LVS0OR Z:\Informatics\Development\PHP\dtrustbridge
$ dir
 El volumen de la unidad Z es Data
 El número de serie del volumen es: A0B9-5BDF

 Directorio de Z:\Informatics\Development\PHP\dtrustbridge

07/12/2019  17:00    <DIR>          .
07/12/2019  17:00    <DIR>          ..
07/12/2019  17:00    <DIR>          node_modules
07/12/2019  17:00           142.482 package-lock.json
               1 archivos        142.482 bytes
               3 dirs  179.703.115.776 bytes libres
```

El Bridge a desarrollar será sencillo, sólo necesitaremos un fichero JS con la especificación de todas las funciones necesarias para operar.

Para poder conectarse con Ethereum, necesitaremos la librería Web3js (https://web3js.readthedocs.io/en/v1.2.4/), que nos ofrece todo lo necesario para conectar cuentas a través del navegador y poder llamar a cualquier Smart Contract disponible en la red de blockchain, además de las funciones de la propia blockchain. Instalaremos esta librería con el comando siguiente:

```
josef@DESKTOP-8LVS0OR Z:\Informatics\Development\PHP\dtrustbridge
$ npm install web3
```

Por último, necesitamos especificar qué operaciones están disponibles de nuestro Smart Contract y esto se hará con el ABI (Application Binary Interface) de nuestro contrato, que no es otra cosa que un resumen de las operaciones existentes en *bytecode*, pero en un formato legible para JS. Este ABI se encuentra en el fichero JSON compilado con Truffle.

Mostramos el código resultante.

```
///////////////
// Libraries //
///////////////
import Web3 from "web3";
```

```
///////////////
// Constants //
///////////////
const CONTRACT_ABI = require('./DTrustManager.json')['abi'];
const CONTRACT_ADDRESS = '0x4B6c8d10BbF57ffE36F8D2fe28beb332d824572C';

//////////////////////
// Global variables //
//////////////////////
var web3;
var connected_account;
var connected_account_trust_score;
var contract_handler;

//////////////////////
// Private Functions //
//////////////////////
async function _initialize_web3_provider()
{
    // Modern dapp browsers
    if (window.ethereum)
    {
        window.web3 = new Web3(ethereum);
        try
        {
            // Request account access if needed
            await ethereum.enable();
        }
        catch (error)
        {
            // User denied account access
            console.log(error);
        }
    }

    // Legacy dapp browsers
    else if (web3)
    {
        window.web3 = new Web3(web3.currentProvider);
    }

    // Non-dapp browsers
    else
    {
        console.log('Non-
Ethereum browser detected. You should consider trying MetaMask!');
    }

    // We have a connected provider
    if (window.web3 !== 'undefined')
    {
        await window.web3.eth.getAccounts().then(
            function(connected_accounts)
            {
                connected_account = connected_accounts[0];
                window.document.dispatchEvent(new CustomEvent('account_connected'));
            }
        );
    }
}

//
// Execute after web3 initialization
//
async function _initialize_blockchain_contract()
{
```

```
    var contract_instance = new window.web3.eth.Contract(CONTRACT_ABI, CONTRACT_ADDRE
SS);
    contract_handler = contract_instance.methods;

    await contract_handler.getGeneralTrustScore(connected_account).call().then(
        function(trust_score)
        {
            connected_account_trust_score = trust_score;
            window.document.dispatchEvent(new CustomEvent('contract_linked'));
        }
    );
}

async function _get_user_trust_score(user_address)
{
    return new Promise(
        async function(resolve)
        {
            return resolve(await contract_handler.getTrustScore(user_address).call(
                {
                    from: connected_account
                }
            ));
        }
    );
}

async function _rate_user(user_address, user_rating)
{
    return new Promise(
        async function(resolve)
        {
            return resolve(await contract_handler.rateUser(user_address, user_rating)
.send(
                {
                    from: connected_account
                }
            ));
        }
    );
}

//////////////////////
// Public functions //
//////////////////////
export function get_connected_account()
{
    return connected_account;
}

export function get_connected_account_trust_score()
{
    return connected_account_trust_score;
}

export async function get_user_trust_score(user_address)
{
    return new Promise(
        async function(resolve)
        {
            return resolve(await _get_user_trust_score(user_address));
        }
    );
}

export async function rate_user(user_address, user_rating)
```

```
{
    return new Promise(
        async function(resolve)
        {
            return resolve(await _rate_user(user_address, user_rating));
        }
    );
}

////////////////////
// Initialization //
////////////////////
window.addEventListener('load', async function()
{
    await _initialize_web3_provider();
    await _initialize_blockchain_contract()
});
```

Como vemos, web3 es el objeto encargado de gestionar la conexión con la blockchain, y CONTRACT_ABI y CONTRACT_ADDRESS las constantes que contienen la información sobre el ABI y la dirección de nuestro contrato.

La primera función que se ejecuta, al finalizar la carga del DOM, es _initialize_web3_provider(), seguido de _initialize_blockchain_contract().

La primera función inicializa el objeto web3, que nos permitirá conectar con la blockchain, y al inicializarse, comprobará si existe alguna cuenta disponible a través de Metamask (que comentaremos en la siguiente sección). Si la hay, ejecutará las acciones en la blockchain utilizando esa cuenta.

La segunda función inicializa nuestro Smart Contract y llama a la función getTrustScore() del contrato, pasando como parámetro la cuenta que pueda estar conectada a través de Metamask.

Tras esto, el Bridge espera las acciones realizadas desde el front-end. El Bridge pone a disposición 4 acciones que pueden ser llamadas desde el front:

- Get_connected_account() : Obtiene la cuenta conectada a través de Metamask
- Get_connected_account_trust_score() : Obtiene el Score de confianza de la cuenta conectada a través de Metamask
- Get_user_trust_score() : Obtiene el Score de confianza de un usuario
- Rate_user() : Otorga una puntuación a un usuario

Ahora que tenemos el Bridge en código, lo compilamos con Webpack para que nos sea fácilmente exportable. Definimos en primer lugar la configuración de Webpack, mediante un fichero webpack.config.js para que nos exporte la librería de manera que podamos llamarla como una variable BRIDGE.

```
const path = require('path');

module.exports = {
  entry: './src/jsbridge.js',
  mode: 'production',
  output: {
    filename: 'jsbridge.min.js',
    path: path.resolve(__dirname, 'build'),
    libraryTarget: 'var',
```

```
    library: 'BRIDGE'
  },
};
```

Y compilamos el proyecto.

```
josef@DESKTOP-8LVS0OR Z:\Informatics\Development\PHP\dtrustbridge
$ npx webpack
Hash: efd3ea5d0b46c42ca6c5
Version: webpack 4.41.2
Time: 7102ms
Built at: 2019-12-07 17:24:36
          Asset   Size  Chunks                    Chunk Names
jsbridge.min.js  1 MiB       0  [emitted]  [big]  main
Entrypoint main [big] = jsbridge.min.js
  [4] (webpack)/buildin/global.js 472 bytes {0} [built]
 [21] (webpack)/buildin/module.js 497 bytes {0} [built]
[132] ./src/jsbridge.js 3.5 KiB {0} [built]
[142] buffer (ignored) 15 bytes {0} [optional] [built]
[148] (webpack)/buildin/amd-options.js 80 bytes {0} [built]
[165] util (ignored) 15 bytes {0} [built]
[167] util (ignored) 15 bytes {0} [built]
[232] crypto (ignored) 15 bytes {0} [optional] [built]
[340] ./src/DTrustManager.json 288 KiB {0} [built]
    + 332 hidden modules
```

En la carpeta “build” tendremos nuestro JS disponible para copiarlo a la carpeta de nuestro front y poder contar con sus funciones. El último paso que vamos a dar, es el de añadir el código necesario en el fichero del front-end para llamar a las funciones del Bridge.

```
<script src="jsbridge.min.js"></script>
<script>
    $(document).on(
        'account_connected',
        function()
        {
            $('#connected-
address').html('Estás conectado como: '+BRIDGE.get_connected_account());
        }
    ).on(
        'contract_linked',
        function()
        {
            $('#connected-trust-
score').html('Tu Score de Confianza es: '+BRIDGE.get_connected_account_trust_score())
;
        }
    );

    $(document).ready(function()
    {
        //
        // Rate User
        //
        $('#rate-user .result-block').hide();
        $('#rate-user-trigger').on('click', function()
        {
            var user_address = $('#rate-user-address').val();
            if (user_address == '')
            {
```

```
                alert('Debes especificar una dirección para valorar');
                return;
            }

            var user_rating = $('#rate-user-rating').val();
            if (![1,2,3].indexOf(user_rating))
            {
                alert('Debes especificar una valoración entre 1 y 3');
                return;
            }

            $('#rate-user .result-block').hide('fast');
            BRIDGE.rate_user(user_address, user_rating).then(
                function()
                {
                    $('#rate-user .result-
block span').html('La operación se ha realizado con éxito');
                    $('#rate-user .result-block').show('fast');
                }
            );
        });

        //
        // Get User Score
        //
        $('#get-user-score .result-block').hide();
        $('#get-user-score-trigger').on('click', function()
        {
            var user_address = $('#get-user-score-address').val();
            if (user_address == '')
            {
                alert('Debes especificar una dirección para obtener el Score de Confi
anza');
                return;
            }

            $('#get-user-score .result-block').hide('fast');
            BRIDGE.get_user_trust_score(user_address).then(
                function(user_trust_score)
                {
                    $('#get-user-score .result-
block span').html('El Score de confianza del usuario '+user_address+' es: '+user_trus
t_score);
                    $('#get-user-score .result-block').show('fast');
                }
            );
        });
    })
</script>
```

8.1.6 Ejecución y testing del sistema a través del Front-end con Metamask

Con todo el sistema integrado, podemos proceder a ejecutar y realizar pruebas desde el front-end. Para ello, empezaremos instalando la extensión Metamask (https://metamask.io/) del navegador correspondiente.

Metamask nos permite seleccionar la red a la que nos queremos conectar y cargar las direcciones que queramos. Por lo tanto, lo configuraremos para conectarnos a nuestra red local proporcionada por Ganache, y cargaremos la primera cuenta de la lista de Ganache.

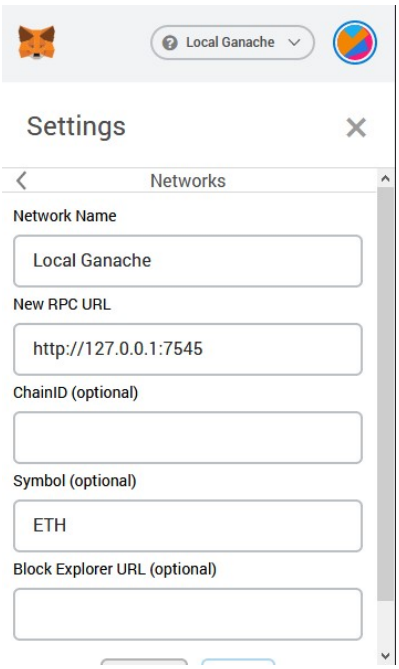


Ilustración 16: Configuración de la red local en Metamask. Imagen propia.

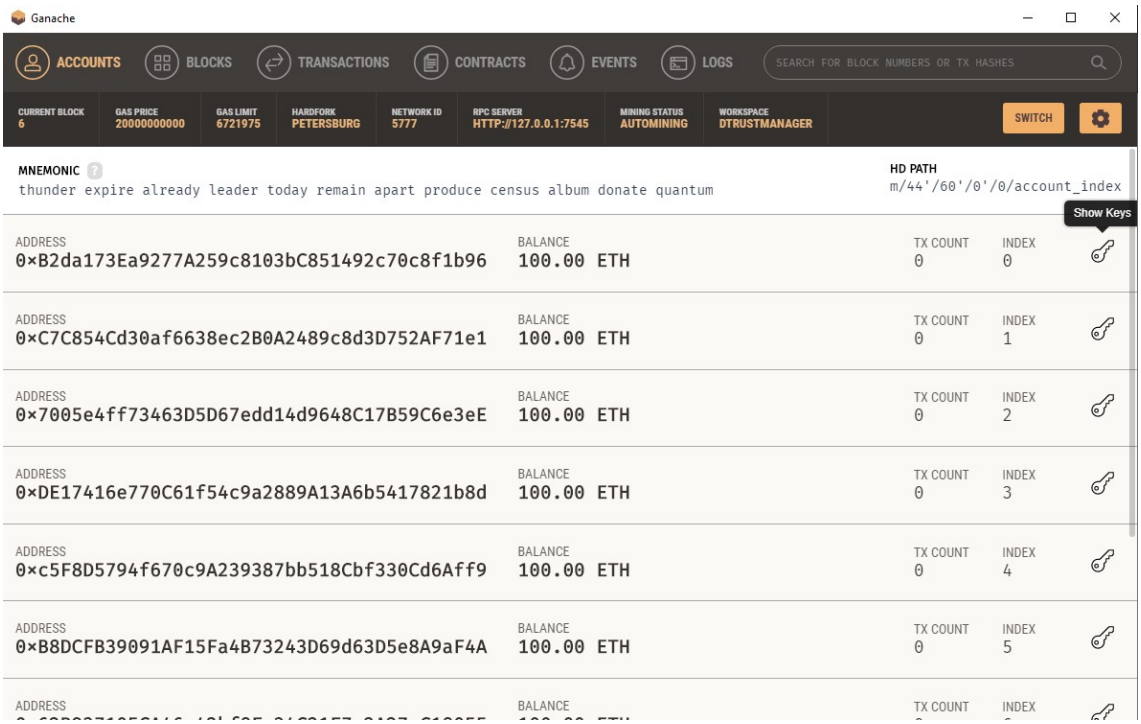


Ilustración 17: Obtener clave privada de una cuenta en Ganache (Paso 1). Imagen propia.

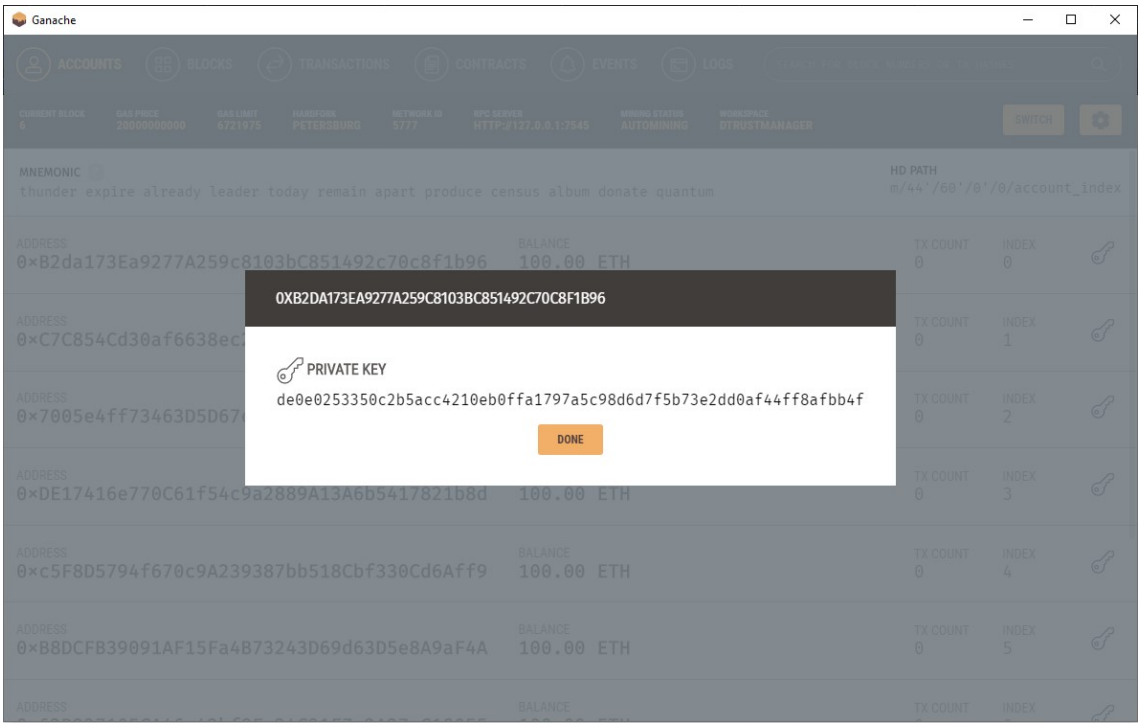


Ilustración 18: Obtener clave privada de una cuenta en Ganache (Paso 2). Imagen propia.

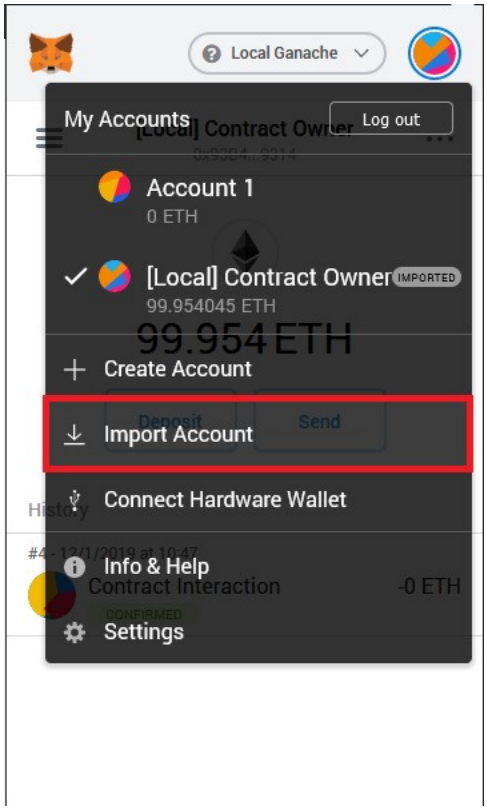


Ilustración 19: Importar cuenta en Metamask (Paso 1). Imagen propia.

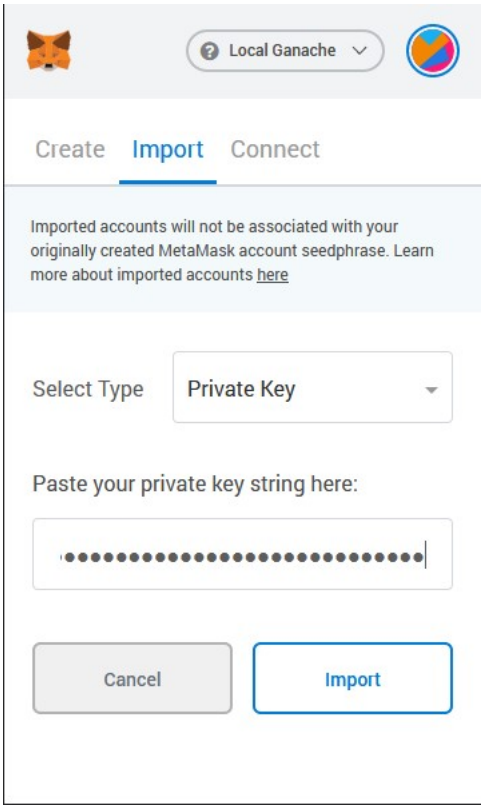


Ilustración 20: Importar cuenta en Metamask (Paso 2). Imagen propia.

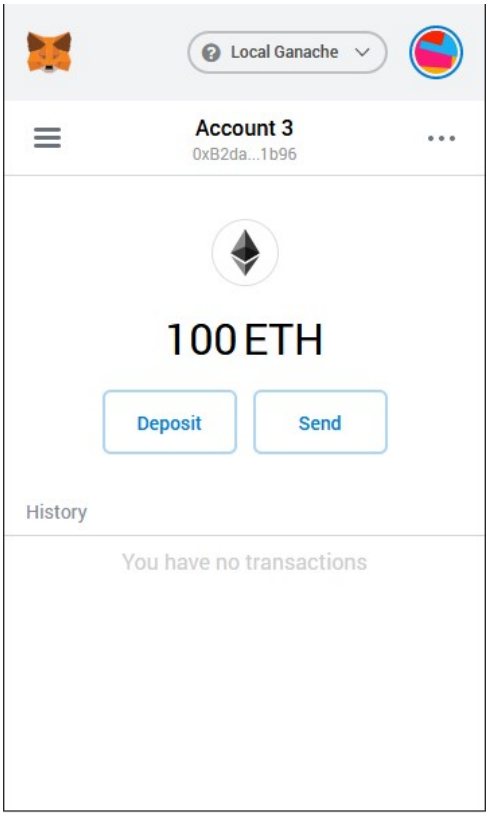


Ilustración 21: Importar cuenta en Metamask (Resultado). Imagen propia.

Hemos importado la cuenta "0xB2da173Ea9277A259c8103bC851492c70c8f1b96" en Metamask. Procedemos ahora a acceder al front-end de nuestra Dapp para comprobar su funcionamiento.

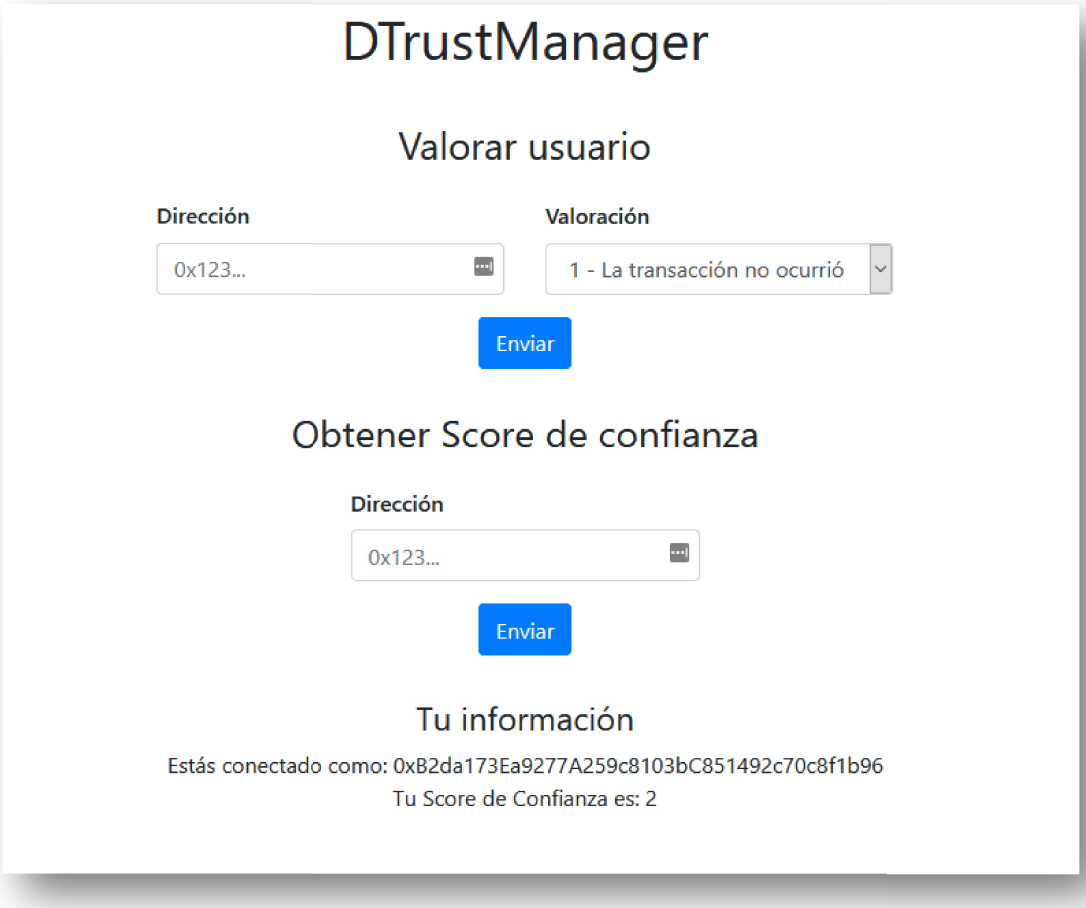


Ilustración 22: Front-end conectado con Metamask. Imagen propia.

Como podemos observar en el bloque de "Tu información", estamos efectivamente conectados con la cuenta deseada y nuestro Score de confianza es "2" (Medio). Esto se debe a que no hemos recibido ninguna valoración aún, con lo que el sistema no se decanta por un lado ni por otro.

En primer lugar, obtengamos el Score de confianza de otro usuario de la red, proporcionado por Ganache, el "0xC7C854Cd30af6638ec2B0A2489c8d3D752AF71e1".

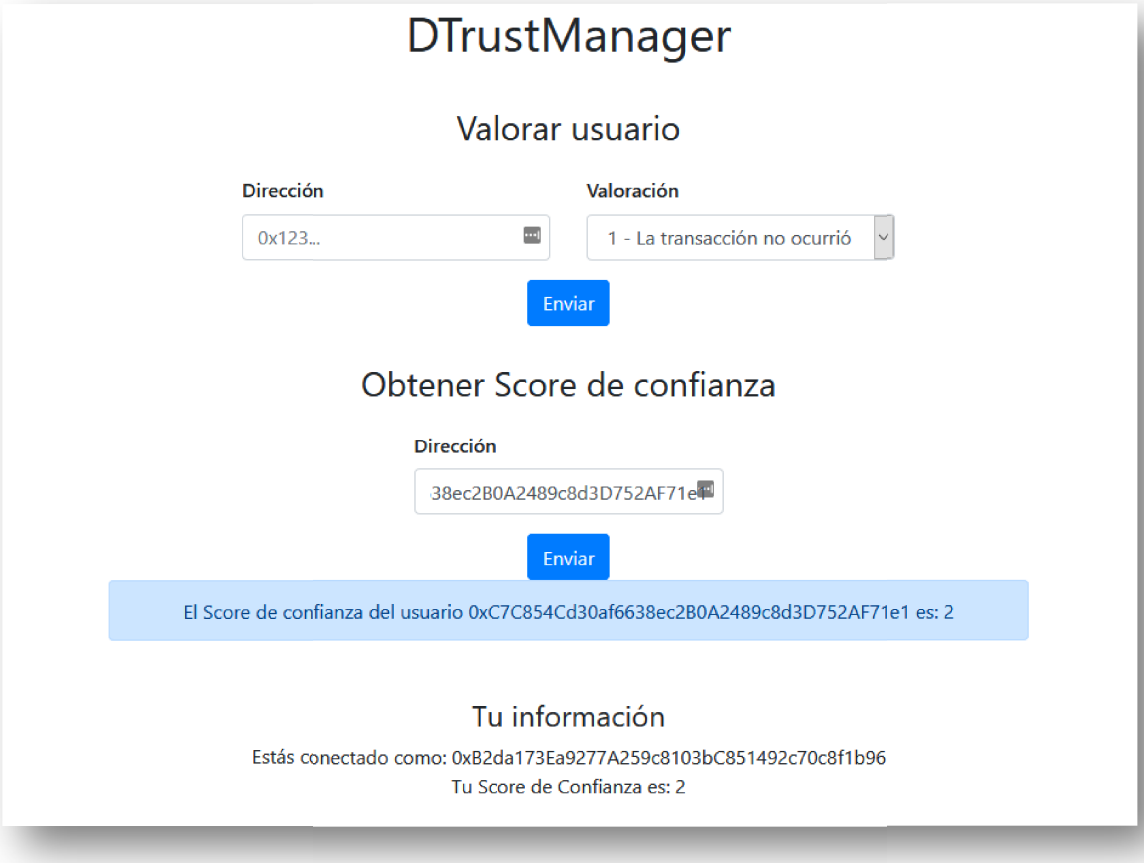


Ilustración 23: Obtener valoración de un usuario. Imagen propia.

Al igual que nuestro Score de confianza, el resultado es 2 (Medio), ya que aún no ha recibido ninguna valoración.

Procedemos ahora a otorgar una valoración buena, 3, a ese mismo usuario, y volvemos a llamar a la función de “Obtener Score de confianza”.

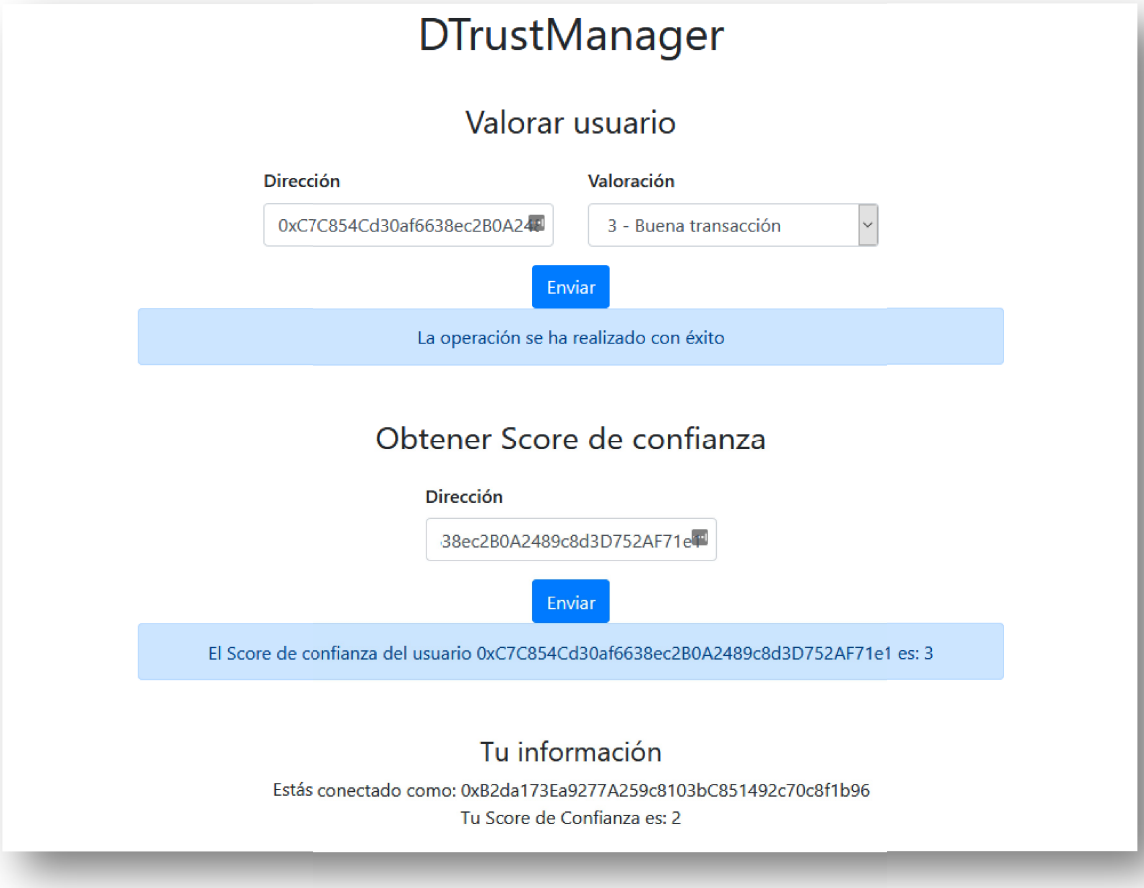


Ilustración 24: Valorar usuario y obtener su score de confianza. Imagen propia.

Efectivamente, podemos ver que el usuario valorado tiene ahora un Score de confianza diferente.

8.1.7 Testing del sistema mediante simulación de transacciones en la red

Para poder validar el sistema, es necesario realizar una simulación de cómo se comportaría nuestra solución en un entorno donde interactúan un número indeterminado de nodos realizando una gran cantidad de transacciones entre ellos.

En nuestro caso, simularemos este comportamiento creando veinte cuentas, asignándoles un comportamiento por defecto y ejecutando 200 transacciones entre ellas. Utilizaremos la herramienta de testing que viene integrada en Truffle.

Para verificar la mejora que supone la utilización del sistema desarrollado, realizaremos una primera simulación de transacciones sin utilizar el algoritmo de confianza y, seguidamente, 4 simulaciones más utilizando el algoritmo de confianza.

Antes de mostrar el proceso, sin embargo, explicamos los comportamientos que hemos elegido para permitir a los nodos tomar las decisiones dentro de la red. Hemos elegido lo que creemos que podría representar los 4 comportamientos más básicos dentro de una red:

- **SCAM**: Nodo malicioso. Su objetivo en la red es beneficiarse del resto de nodos, engañándoles y no cumpliendo su parte del trato. Estos nodos siempre quieren realizar las transacciones y, en caso de realizarlas, no puntúan al nodo con el que han

interactuado, a no ser que sea otro nodo malicioso, al cual le darán una valoración alta. Suponemos que los nodos maliciosos forman un grupo entre ellos para hacerse más fuertes.

- **WARY**: Nodo precavido. Este nodo sólo realiza transacciones con nodos que tienen un Score de Confianza alto. Si la transacción ocurre correctamente, este nodo otorga una valoración alta.
- **NORMAL**: Nodo corriente. Este nodo sólo realiza transacciones con nodos que tienen un Score de Confianza medio o alto. Si la transacción ocurre correctamente, este nodo otorga una valoración alta.
- **RISKY**: Nodo atrevido. Este nodo realiza transacciones con todos los nodos, sin importar su Score de Confianza. Si la transacción ocurre correctamente, este nodo otorga una valoración alta.

Teniendo esto en cuenta, pasamos ahora a mostrar el proceso.

En primer lugar, vamos a obtener las veinte cuentas. Crearemos un nuevo Entorno de Trabajo en Ganache para poder configurar el número de cuentas que queremos (20 cuentas para testing + 1 que será la propietaria del contrato).

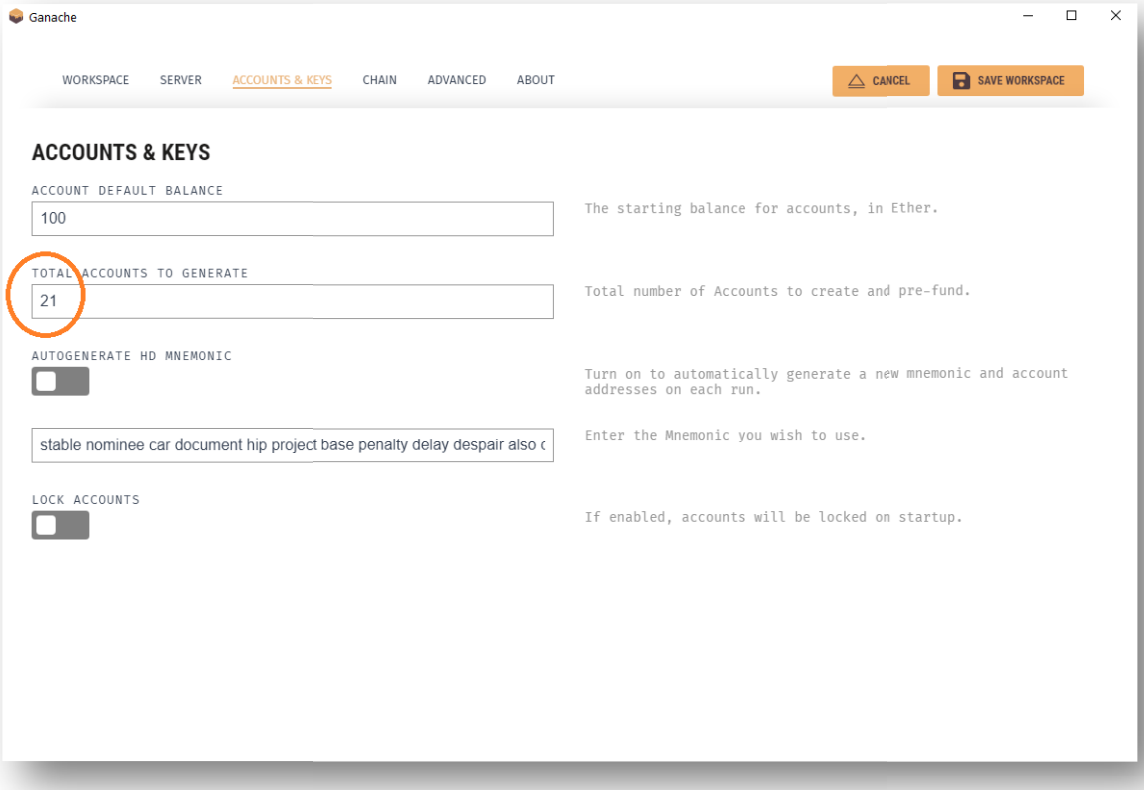


Ilustración 25: Configurar 20 cuentas en Ganache

A continuación, crearemos el script de simulación. Para agilizar el desarrollo, lo crearemos utilizando la arquitectura de testing integrada en Truffle. Creamos un fichero en la carpeta test, llamado simulation.js, que albergará nuestro código, escrito en JavaScript.

Mostramos el código a continuación.

```
// Configuration
const NUMBER_OF_TRANSACTIONS = 200;

// Constants
const DTrustManager = artifacts.require('DTrustManager');
const LOW_SCORE = 1;
const MED_SCORE = 2;
const HIGH_SCORE = 3;
const NODE_SCAM = 'SCAM';       // Trusts everybody and always gives High Score. Scam
mers are supposed to be in group (they support each other)
const NODE_WARY = 'WARY';       // Only trusts nodes with a High Score
const NODE_NORMAL = 'NORMAL';   // Trusts nodes with Medium and High Score
const NODE_RISKY = 'RISKY';     // Trusts everybody

// Variables
var contract_handler;
var account_matrix;
var transactions;

// Rules:
//--------
// 1. SCAM and RISKY nodes always want to make a transaction
// 2. WARY nodes only transact with nodes with a High Score
// 3. NORMAL nodes only transact with nodes with a Medium or High Score

// 4. SCAM nodes always give a High Score
// 5. The rest of the nodes give a High Score unless they transact with a SCAM node,
which receive a Low Score

contract('DTrustManager', function([
    _,
    account1, account2, account3, account4, account5,
    account6, account7, account8, account9, account10,
    account11, account12, account13, account14, account15,
    account16, account17, account18, account19, account20
])
{
    beforeEach(async function()
    {
        // Deploy Dapp
        contract_handler = await DTrustManager.new();

        // Creating the accounts matrix
        // SCAM, WARY, NORMAL, RISKY
        account_matrix = {
            0:  { 'address': account1, 'type': NODE_SCAM, 'transactions': 0, 'succeed
ed': 0, 'scam_attempts': 0, 'scam_success': 0 },
            1:  { 'address': account2, 'type': NODE_SCAM, 'transactions': 0, 'succeed
ed': 0, 'scam_attempts': 0, 'scam_success': 0 },
            2:  { 'address': account3, 'type': NODE_SCAM, 'transactions': 0, 'succeed
ed': 0, 'scam_attempts': 0, 'scam_success': 0 },
            3:  { 'address': account4, 'type': NODE_WARY, 'transactions': 0, 'succeed
ed': 0, 'scam_attempts': 0, 'scam_success': 0 },
            4:  { 'address': account5, 'type': NODE_WARY, 'transactions': 0, 'succeed
ed': 0, 'scam_attempts': 0, 'scam_success': 0 },
            5:  { 'address': account6, 'type': NODE_WARY, 'transactions': 0, 'succeed
ed': 0, 'scam_attempts': 0, 'scam_success': 0 },
            6:  { 'address': account7, 'type': NODE_WARY, 'transactions': 0, 'succeed
ed': 0, 'scam_attempts': 0, 'scam_success': 0 },
            7:  { 'address': account8, 'type': NODE_WARY, 'transactions': 0, 'succeed
ed': 0, 'scam_attempts': 0, 'scam_success': 0 },
            8:  { 'address': account9, 'type': NODE_WARY, 'transactions': 0, 'succeed
ed': 0, 'scam_attempts': 0, 'scam_success': 0 },
            9:  { 'address': account10, 'type': NODE_WARY, 'transactions': 0, 'succee
ded': 0, 'scam_attempts': 0, 'scam_success': 0 },
            10:  { 'address': account11, 'type': NODE_NORMAL, 'transactions': 0, 'suc
```

```
ceeded': 0, 'scam_attempts': 0, 'scam_success': 0 },
            11:  { 'address': account12, 'type': NODE_NORMAL, 'transactions': 0, 'suc
ceeded': 0, 'scam_attempts': 0, 'scam_success': 0 },
            12:  { 'address': account13, 'type': NODE_NORMAL, 'transactions': 0, 'suc
ceeded': 0, 'scam_attempts': 0, 'scam_success': 0 },
            13:  { 'address': account14, 'type': NODE_NORMAL, 'transactions': 0, 'suc
ceeded': 0, 'scam_attempts': 0, 'scam_success': 0 },
            14:  { 'address': account15, 'type': NODE_NORMAL, 'transactions': 0, 'suc
ceeded': 0, 'scam_attempts': 0, 'scam_success': 0 },
            15:  { 'address': account16, 'type': NODE_NORMAL, 'transactions': 0, 'suc
ceeded': 0, 'scam_attempts': 0, 'scam_success': 0 },
            16:  { 'address': account17, 'type': NODE_NORMAL, 'transactions': 0, 'suc
ceeded': 0, 'scam_attempts': 0, 'scam_success': 0 },
            17:  { 'address': account18, 'type': NODE_RISKY, 'transactions': 0, 'succ
eeded': 0, 'scam_attempts': 0, 'scam_success': 0 },
            18:  { 'address': account19, 'type': NODE_RISKY, 'transactions': 0, 'succ
eeded': 0, 'scam_attempts': 0, 'scam_success': 0 },
            19:  { 'address': account20, 'type': NODE_RISKY, 'transactions': 0, 'succ
eeded': 0, 'scam_attempts': 0, 'scam_success': 0 },
        };

        // Generating X random transactions
        transactions = [];
        for (var i = 1; i <= NUMBER_OF_TRANSACTIONS; i++)
        {
            var random_peer = Math.floor(Math.random()*Object.keys(account_matrix).le
ngth);
            var random_peer_2 = random_peer;

            while (random_peer_2 == random_peer)
            {
                random_peer_2 = Math.floor(Math.random()*Object.keys(account_matrix).
length);
            }
            transactions.push([random_peer, random_peer_2]);
        }
    });

    describe('Simulation', function()
    {
        it("performs first simulation: Transactions without DTrustManager", async fun
ction()
        {
            var total_transactions = 0;
            var total_scams = 0;
            var log_message;
            console.log(' ');
            console.log('****************************************');
            console.log('First simulation: Transactions without DTrustManager');
            console.log('****************************************');
            console.log(' ');

            console.log("------------------------");
            console.log("| Perform transactions |");
            console.log("------------------------");
            for (var i = 0; i < transactions.length; i++)
            {
                var peer1 = transactions[i][0];
                var peer2 = transactions[i][1];
                var scam_success = false;

                // Adding up the transactions
                account_matrix[peer1].transactions++;
                account_matrix[peer2].transactions++;

                // Summary of the situation
```

```
                console.log('[1] '+peer1+' - '+account_matrix[peer1].type+' --
 [2] '+peer2+' - '+account_matrix[peer2].type)

                // Peer1 SCAM assessment
                if (
                    (account_matrix[peer1].type != NODE_SCAM) &&
                    (account_matrix[peer2].type == NODE_SCAM)
                )
                {
                    account_matrix[peer1].scam_attempts++;
                    account_matrix[peer1].scam_success++;
                    scam_success = true;
                }

                // Peer2 SCAM assessment
                if (
                    (account_matrix[peer2].type != NODE_SCAM) &&
                    (account_matrix[peer1].type == NODE_SCAM)
                )
                {
                    account_matrix[peer2].scam_attempts++;
                    account_matrix[peer2].scam_success++;
                    scam_success = true;
                }

                // Performing transaction
                console.log('Transaction is performed.');

                // Adding up the succeeded transactions
                account_matrix[peer1].succeeded++;
                account_matrix[peer2].succeeded++;
                total_transactions++;
                if (scam_success) total_scams++;
            }

            console.log(" ");
            console.log("-----------------");
            console.log("| Final results |");
            console.log("-----------------");
            for (const [acc_key, acc_info] of Object.entries(account_matrix))
            {
                log_message = "";
                log_message += "[Account "+acc_key+" - "+acc_info.type+"] ";

                // Transactions
                log_message += "Transactions: "+acc_info.transactions+"; ";

                // SCAMs
                log_message += "SCAMs: "+acc_info.scam_success;

                console.log(log_message);
            }

            // Total Stats
            console.log('######################################');
            console.log('Total Transactions: '+total_transactions);
            console.log('Total Scams: '+total_scams);
            console.log('SCAM percentage: '+(Math.round((total_scams/total_transactio
ns) * 10000) / 100)+'%');
            console.log('######################################');
            console.log(' ');

            assert.equal(1, 1, "something is wrong");
        });

        it("performs second simulation: Transactions with DTrustManager", async funct
```

```
ion()
        {
            var total_transactions = 0;
            var total_scams = 0;
            var log_message;
            console.log(' ');
            console.log('**************************************');
            console.log('Second simulation: Transactions with DTrustManager (I)');
            console.log('**************************************');
            console.log(' ');

            console.log("------------------------");
            console.log("| Perform transactions |");
            console.log("------------------------");
            for (var i = 0; i < transactions.length; i++)
            {
                var peer1 = transactions[i][0];
                var peer2 = transactions[i][1];
                var peer1_wants_transaction = false;
                var peer2_wants_transaction = false;
                var score_from_to = false;
                var score_to_from = false;
                var scam_success = false;

                // Adding up the transactions
                account_matrix[peer1].transactions++;
                account_matrix[peer2].transactions++;

                // We calculate the Trust Score
                var peer1_trust_score = await contract_handler.getTrustScore(account_
matrix[peer1].address, {from: account_matrix[peer2].address});
                var peer2_trust_score = await contract_handler.getTrustScore(account_
matrix[peer2].address, {from: account_matrix[peer1].address});

                // Summary of the situation
                console.log('[1] '+peer1+' -
 '+account_matrix[peer1].type+' ('+peer1_trust_score+') -- [2] '+peer2+' -
 '+account_matrix[peer2].type+' ('+peer2_trust_score+')')

                // Does the first peer want to transact?
                if (
                    (account_matrix[peer1].type == NODE_SCAM) ||
                    (account_matrix[peer1].type == NODE_RISKY) ||
                    ( (account_matrix[peer1].type == NODE_WARY) && (peer2_trust_score
 == HIGH_SCORE) ) ||
                    ( (account_matrix[peer1].type == NODE_NORMAL) && (peer2_trust_sco
re != LOW_SCORE) )
                )
                {
                    peer1_wants_transaction = true;
                }

                // Does the second peer want to transact?
                if (
                    (account_matrix[peer2].type == NODE_SCAM) ||
                    (account_matrix[peer2].type == NODE_RISKY) ||
                    ( (account_matrix[peer2].type == NODE_WARY) && (peer1_trust_score
 == HIGH_SCORE) ) ||
                    ( (account_matrix[peer2].type == NODE_NORMAL) && (peer1_trust_sco
re != LOW_SCORE) )
                )
                {
                    peer2_wants_transaction = true;
                }

                // Peer1 SCAM assessment
```

```
                if (
                    (account_matrix[peer1].type != NODE_SCAM) &&
                    (account_matrix[peer2].type == NODE_SCAM)
                )
                {
                    account_matrix[peer1].scam_attempts++;
                    if (peer1_wants_transaction) { account_matrix[peer1].scam_success
++; scam_success = true; }
                }

                // Peer2 SCAM assessment
                if (
                    (account_matrix[peer2].type != NODE_SCAM) &&
                    (account_matrix[peer1].type == NODE_SCAM)
                )
                {
                    account_matrix[peer2].scam_attempts++;
                    if (peer2_wants_transaction) { account_matrix[peer2].scam_success
++; scam_success = true; }
                }

                if (peer1_wants_transaction && peer2_wants_transaction)
                {
                    console.log('Transaction is performed.');

                    // Adding up the succeeded transactions
                    account_matrix[peer1].succeeded++;
                    account_matrix[peer2].succeeded++;
                    total_transactions++;
                    if (scam_success) total_scams++;

                    // Calculating the score from the first peer to the second
                    if (account_matrix[peer1].type == NODE_SCAM)
                    {
                        if (account_matrix[peer2].type == NODE_SCAM)
                        {
                            score_from_to = HIGH_SCORE;
                        }
                    }
                    else
                    {
                        if (account_matrix[peer2].type == NODE_SCAM)
                        {
                            score_from_to = LOW_SCORE;
                        }
                        else
                        {
                            score_from_to = HIGH_SCORE;
                        }
                    }

                    // Calculating the score from the second peer to the first
                    if (account_matrix[peer2].type == NODE_SCAM)
                    {
                        if (account_matrix[peer1].type == NODE_SCAM)
                        {
                            score_to_from = HIGH_SCORE;
                        }
                    }
                    else
                    {
                        if (account_matrix[peer1].type == NODE_SCAM)
                        {
                            score_to_from = LOW_SCORE;
                        }
                        else
```

```
                        {
                            score_to_from = HIGH_SCORE;
                        }
                    }

                    // Giving Score from the first peer to the second
                    if (score_from_to)
                    {
                        console.log('[1] gives score '+score_from_to+' to [2]');
                        await contract_handler.rateUser(account_matrix[peer2].address
, score_from_to, {from: account_matrix[peer1].address});
                    }

                    // Giving Score from the second peer to the first
                    if (score_to_from)
                    {
                        console.log('[2] gives score '+score_to_from+' to [1]');
                        await contract_handler.rateUser(account_matrix[peer1].address
, score_to_from, {from: account_matrix[peer2].address});
                    }
                }
                else
                {
                    console.log('Transaction is NOT performed.');
                }
            }

            console.log(" ");
            console.log("-----------------");
            console.log("| Final results |");
            console.log("-----------------");
            for (const [acc_key, acc_info] of Object.entries(account_matrix))
            {
                var acc_score = await contract_handler.getGeneralTrustScore(acc_info.
address, {from: _});

                log_message = "";
                log_message += "[Account "+acc_key+" - "+acc_info.type+"] ";

                // Transactions
                log_message += "Transactions: "+acc_info.succeeded+"/"+acc_info.trans
actions+" ";
                if (acc_info.transactions > 0)
                {
                    log_message += "("+(Math.round((acc_info.succeeded/acc_info.trans
actions) * 10000) / 100)+"%)";
                }
                else
                {
                    log_message += "(0%)";
                }
                log_message += "; ";

                // SCAMs
                log_message += "SCAMs: "+acc_info.scam_success+"/"+acc_info.scam_atte
mpts+" ";
                if (acc_info.scam_attempts > 0)
                {
                    log_message += "("+(Math.round((acc_info.scam_success/acc_info.sc
am_attempts) * 10000) / 100)+"%)";
                }
                else
                {
                    log_message += "(0%)";
                }
                log_message += "; ";
```

```
                // Trust Score
                log_message += "Trust Score: "+acc_score;

                console.log(log_message);
            }

            // Total Stats
            console.log('Total Transactions: '+total_transactions);
            console.log('Total Scams: '+total_scams);
            console.log('SCAM percentage: '+(Math.round((total_scams/total_transactio
ns) * 10000) / 100)+'%');

            assert.equal(1, 1, "something is wrong");
        });

        it("performs third simulation: Transactions with DTrustManager", async functi
on()
        {
            var total_transactions = 0;
            var total_scams = 0;
            var log_message;
            console.log(' ');
            console.log('**************************************');
            console.log('Third simulation: Transactions with DTrustManager (II)');
            console.log('**************************************');
            console.log(' ');

            console.log("------------------------");
            console.log("| Perform transactions |");
            console.log("------------------------");
            for (var i = 0; i < transactions.length; i++)
            {
                var peer1 = transactions[i][0];
                var peer2 = transactions[i][1];
                var peer1_wants_transaction = false;
                var peer2_wants_transaction = false;
                var score_from_to = false;
                var score_to_from = false;
                var scam_success = false;

                // Adding up the transactions
                account_matrix[peer1].transactions++;
                account_matrix[peer2].transactions++;

                // We calculate the Trust Score
                var peer1_trust_score = await contract_handler.getTrustScore(account_
matrix[peer1].address, {from: account_matrix[peer2].address});
                var peer2_trust_score = await contract_handler.getTrustScore(account_
matrix[peer2].address, {from: account_matrix[peer1].address});

                // Summary of the situation
                console.log('[1] '+peer1+' -
 '+account_matrix[peer1].type+' ('+peer1_trust_score+') -- [2] '+peer2+' -
 '+account_matrix[peer2].type+' ('+peer2_trust_score+')')

                // Does the first peer want to transact?
                if (
                    (account_matrix[peer1].type == NODE_SCAM) ||
                    (account_matrix[peer1].type == NODE_RISKY) ||
                    ( (account_matrix[peer1].type == NODE_WARY) && (peer2_trust_score
 == HIGH_SCORE) ) ||
                    ( (account_matrix[peer1].type == NODE_NORMAL) && (peer2_trust_sco
re != LOW_SCORE) )
                )
                {
```

```
                    peer1_wants_transaction = true;
                }

                // Does the second peer want to transact?
                if (
                    (account_matrix[peer2].type == NODE_SCAM) ||
                    (account_matrix[peer2].type == NODE_RISKY) ||
                    ( (account_matrix[peer2].type == NODE_WARY) && (peer1_trust_score
 == HIGH_SCORE) ) ||
                    ( (account_matrix[peer2].type == NODE_NORMAL) && (peer1_trust_sco
re != LOW_SCORE) )
                )
                {
                    peer2_wants_transaction = true;
                }

                // Peer1 SCAM assessment
                if (
                    (account_matrix[peer1].type != NODE_SCAM) &&
                    (account_matrix[peer2].type == NODE_SCAM)
                )
                {
                    account_matrix[peer1].scam_attempts++;
                    if (peer1_wants_transaction) { account_matrix[peer1].scam_success
++; scam_success = true; }
                }

                // Peer2 SCAM assessment
                if (
                    (account_matrix[peer2].type != NODE_SCAM) &&
                    (account_matrix[peer1].type == NODE_SCAM)
                )
                {
                    account_matrix[peer2].scam_attempts++;
                    if (peer2_wants_transaction) { account_matrix[peer2].scam_success
++; scam_success = true; }
                }

                if (peer1_wants_transaction && peer2_wants_transaction)
                {
                    console.log('Transaction is performed.');

                    // Adding up the succeeded transactions
                    account_matrix[peer1].succeeded++;
                    account_matrix[peer2].succeeded++;
                    total_transactions++;
                    if (scam_success) total_scams++;

                    // Calculating the score from the first peer to the second
                    if (account_matrix[peer1].type == NODE_SCAM)
                    {
                        if (account_matrix[peer2].type == NODE_SCAM)
                        {
                            score_from_to = HIGH_SCORE;
                        }
                    }
                    else
                    {
                        if (account_matrix[peer2].type == NODE_SCAM)
                        {
                            score_from_to = LOW_SCORE;
                        }
                        else
                        {
                            score_from_to = HIGH_SCORE;
                        }
```

```
                    }

                    // Calculating the score from the second peer to the first
                    if (account_matrix[peer2].type == NODE_SCAM)
                    {
                        if (account_matrix[peer1].type == NODE_SCAM)
                        {
                            score_to_from = HIGH_SCORE;
                        }
                    }
                    else
                    {
                        if (account_matrix[peer1].type == NODE_SCAM)
                        {
                            score_to_from = LOW_SCORE;
                        }
                        else
                        {
                            score_to_from = HIGH_SCORE;
                        }
                    }

                    // Giving Score from the first peer to the second
                    if (score_from_to)
                    {
                        console.log('[1] gives score '+score_from_to+' to [2]');
                        await contract_handler.rateUser(account_matrix[peer2].address
, score_from_to, {from: account_matrix[peer1].address});
                    }

                    // Giving Score from the second peer to the first
                    if (score_to_from)
                    {
                        console.log('[2] gives score '+score_to_from+' to [1]');
                        await contract_handler.rateUser(account_matrix[peer1].address
, score_to_from, {from: account_matrix[peer2].address});
                    }
                }
                else
                {
                    console.log('Transaction is NOT performed.');
                }
            }

            console.log(" ");
            console.log("-----------------");
            console.log("| Final results |");
            console.log("-----------------");
            for (const [acc_key, acc_info] of Object.entries(account_matrix))
            {
                var acc_score = await contract_handler.getGeneralTrustScore(acc_info.
address, {from: _});

                log_message = "";
                log_message += "[Account "+acc_key+" - "+acc_info.type+"] ";

                // Transactions
                log_message += "Transactions: "+acc_info.succeeded+"/"+acc_info.trans
actions+" ";
                if (acc_info.transactions > 0)
                {
                    log_message += "("+(Math.round((acc_info.succeeded/acc_info.trans
actions) * 10000) / 100)+"%)";
                }
                else
                {
```

```
                    log_message += "(0%)";
                }
                log_message += "; ";

                // SCAMs
                log_message += "SCAMs: "+acc_info.scam_success+"/"+acc_info.scam_atte
mpts+" ";
                if (acc_info.scam_attempts > 0)
                {
                    log_message += "("+(Math.round((acc_info.scam_success/acc_info.sc
am_attempts) * 10000) / 100)+"%)";
                }
                else
                {
                    log_message += "(0%)";
                }
                log_message += "; ";

                // Trust Score
                log_message += "Trust Score: "+acc_score;

                console.log(log_message);
            }

            // Total Stats
            console.log('Total Transactions: '+total_transactions);
            console.log('Total Scams: '+total_scams);
            console.log('SCAM percentage: '+(Math.round((total_scams/total_transactio
ns) * 10000) / 100)+'%');

            assert.equal(1, 1, "something is wrong");
        });

        it("performs fourth simulation: Transactions with DTrustManager", async funct
ion()
        {
            var total_transactions = 0;
            var total_scams = 0;
            var log_message;
            console.log(' ');
            console.log('****************************************');
            console.log('Fourth simulation: Transactions with DTrustManager (III)');
            console.log('****************************************');
            console.log(' ');

            console.log("------------------------");
            console.log("| Perform transactions |");
            console.log("------------------------");
            for (var i = 0; i < transactions.length; i++)
            {
                var peer1 = transactions[i][0];
                var peer2 = transactions[i][1];
                var peer1_wants_transaction = false;
                var peer2_wants_transaction = false;
                var score_from_to = false;
                var score_to_from = false;
                var scam_success = false;

                // Adding up the transactions
                account_matrix[peer1].transactions++;
                account_matrix[peer2].transactions++;

                // We calculate the Trust Score
                var peer1_trust_score = await contract_handler.getTrustScore(account_
matrix[peer1].address, {from: account_matrix[peer2].address});
                var peer2_trust_score = await contract_handler.getTrustScore(account_
```

```
matrix[peer2].address, {from: account_matrix[peer1].address});

                // Summary of the situation
                console.log('[1] '+peer1+' -
 '+account_matrix[peer1].type+' ('+peer1_trust_score+') -- [2] '+peer2+' -
 '+account_matrix[peer2].type+' ('+peer2_trust_score+')')

                // Does the first peer want to transact?
                if (
                    (account_matrix[peer1].type == NODE_SCAM) ||
                    (account_matrix[peer1].type == NODE_RISKY) ||
                    ( (account_matrix[peer1].type == NODE_WARY) && (peer2_trust_score
 == HIGH_SCORE) ) ||
                    ( (account_matrix[peer1].type == NODE_NORMAL) && (peer2_trust_sco
re != LOW_SCORE) )
                )
                {
                    peer1_wants_transaction = true;
                }

                // Does the second peer want to transact?
                if (
                    (account_matrix[peer2].type == NODE_SCAM) ||
                    (account_matrix[peer2].type == NODE_RISKY) ||
                    ( (account_matrix[peer2].type == NODE_WARY) && (peer1_trust_score
 == HIGH_SCORE) ) ||
                    ( (account_matrix[peer2].type == NODE_NORMAL) && (peer1_trust_sco
re != LOW_SCORE) )
                )
                {
                    peer2_wants_transaction = true;
                }

                // Peer1 SCAM assessment
                if (
                    (account_matrix[peer1].type != NODE_SCAM) &&
                    (account_matrix[peer2].type == NODE_SCAM)
                )
                {
                    account_matrix[peer1].scam_attempts++;
                    if (peer1_wants_transaction) { account_matrix[peer1].scam_success
++; scam_success = true; }
                }

                // Peer2 SCAM assessment
                if (
                    (account_matrix[peer2].type != NODE_SCAM) &&
                    (account_matrix[peer1].type == NODE_SCAM)
                )
                {
                    account_matrix[peer2].scam_attempts++;
                    if (peer2_wants_transaction) { account_matrix[peer2].scam_success
++; scam_success = true; }
                }

                if (peer1_wants_transaction && peer2_wants_transaction)
                {
                    console.log('Transaction is performed.');

                    // Adding up the succeeded transactions
                    account_matrix[peer1].succeeded++;
                    account_matrix[peer2].succeeded++;
                    total_transactions++;
                    if (scam_success) total_scams++;

                    // Calculating the score from the first peer to the second
```

```
                    if (account_matrix[peer1].type == NODE_SCAM)
                    {
                        if (account_matrix[peer2].type == NODE_SCAM)
                        {
                            score_from_to = HIGH_SCORE;
                        }
                    }
                    else
                    {
                        if (account_matrix[peer2].type == NODE_SCAM)
                        {
                            score_from_to = LOW_SCORE;
                        }
                        else
                        {
                            score_from_to = HIGH_SCORE;
                        }
                    }

                    // Calculating the score from the second peer to the first
                    if (account_matrix[peer2].type == NODE_SCAM)
                    {
                        if (account_matrix[peer1].type == NODE_SCAM)
                        {
                            score_to_from = HIGH_SCORE;
                        }
                    }
                    else
                    {
                        if (account_matrix[peer1].type == NODE_SCAM)
                        {
                            score_to_from = LOW_SCORE;
                        }
                        else
                        {
                            score_to_from = HIGH_SCORE;
                        }
                    }

                    // Giving Score from the first peer to the second
                    if (score_from_to)
                    {
                        console.log('[1] gives score '+score_from_to+' to [2]');
                        await contract_handler.rateUser(account_matrix[peer2].address
, score_from_to, {from: account_matrix[peer1].address});
                    }

                    // Giving Score from the second peer to the first
                    if (score_to_from)
                    {
                        console.log('[2] gives score '+score_to_from+' to [1]');
                        await contract_handler.rateUser(account_matrix[peer1].address
, score_to_from, {from: account_matrix[peer2].address});
                    }
                }
                else
                {
                    console.log('Transaction is NOT performed.');
                }
            }

            console.log(" ");
            console.log("-----------------");
            console.log("| Final results |");
            console.log("-----------------");
            for (const [acc_key, acc_info] of Object.entries(account_matrix))
```

```
            {
                var acc_score = await contract_handler.getGeneralTrustScore(acc_info.
address, {from: _});

                log_message = "";
                log_message += "[Account "+acc_key+" - "+acc_info.type+"] ";

                // Transactions
                log_message += "Transactions: "+acc_info.succeeded+"/"+acc_info.trans
actions+" ";
                if (acc_info.transactions > 0)
                {
                    log_message += "("+(Math.round((acc_info.succeeded/acc_info.trans
actions) * 10000) / 100)+"%)";
                }
                else
                {
                    log_message += "(0%)";
                }
                log_message += "; ";

                // SCAMs
                log_message += "SCAMs: "+acc_info.scam_success+"/"+acc_info.scam_atte
mpts+" ";
                if (acc_info.scam_attempts > 0)
                {
                    log_message += "("+(Math.round((acc_info.scam_success/acc_info.sc
am_attempts) * 10000) / 100)+"%)";
                }
                else
                {
                    log_message += "(0%)";
                }
                log_message += "; ";

                // Trust Score
                log_message += "Trust Score: "+acc_score;

                console.log(log_message);
            }

            // Total Stats
            console.log('Total Transactions: '+total_transactions);
            console.log('Total Scams: '+total_scams);
            console.log('SCAM percentage: '+(Math.round((total_scams/total_transactio
ns) * 10000) / 100)+'%');

            assert.equal(1, 1, "something is wrong");
        });

        it("performs fifth simulation: Transactions with DTrustManager", async functi
on()
        {
            var total_transactions = 0;
            var total_scams = 0;
            var log_message;
            console.log(' ');
            console.log('****************************************');
            console.log('Fifth simulation: Transactions with DTrustManager (IV)');
            console.log('****************************************');
            console.log(' ');

            console.log("------------------------");
            console.log("| Perform transactions |");
            console.log("------------------------");
            for (var i = 0; i < transactions.length; i++)
```

```
            {
                var peer1 = transactions[i][0];
                var peer2 = transactions[i][1];
                var peer1_wants_transaction = false;
                var peer2_wants_transaction = false;
                var score_from_to = false;
                var score_to_from = false;
                var scam_success = false;

                // Adding up the transactions
                account_matrix[peer1].transactions++;
                account_matrix[peer2].transactions++;

                // We calculate the Trust Score
                var peer1_trust_score = await contract_handler.getTrustScore(account_
matrix[peer1].address, {from: account_matrix[peer2].address});
                var peer2_trust_score = await contract_handler.getTrustScore(account_
matrix[peer2].address, {from: account_matrix[peer1].address});

                // Summary of the situation
                console.log('[1] '+peer1+' -
 '+account_matrix[peer1].type+' ('+peer1_trust_score+') -- [2] '+peer2+' -
 '+account_matrix[peer2].type+' ('+peer2_trust_score+')')

                // Does the first peer want to transact?
                if (
                    (account_matrix[peer1].type == NODE_SCAM) ||
                    (account_matrix[peer1].type == NODE_RISKY) ||
                    ( (account_matrix[peer1].type == NODE_WARY) && (peer2_trust_score
 == HIGH_SCORE) ) ||
                    ( (account_matrix[peer1].type == NODE_NORMAL) && (peer2_trust_sco
re != LOW_SCORE) )
                )
                {
                    peer1_wants_transaction = true;
                }

                // Does the second peer want to transact?
                if (
                    (account_matrix[peer2].type == NODE_SCAM) ||
                    (account_matrix[peer2].type == NODE_RISKY) ||
                    ( (account_matrix[peer2].type == NODE_WARY) && (peer1_trust_score
 == HIGH_SCORE) ) ||
                    ( (account_matrix[peer2].type == NODE_NORMAL) && (peer1_trust_sco
re != LOW_SCORE) )
                )
                {
                    peer2_wants_transaction = true;
                }

                // Peer1 SCAM assessment
                if (
                    (account_matrix[peer1].type != NODE_SCAM) &&
                    (account_matrix[peer2].type == NODE_SCAM)
                )
                {
                    account_matrix[peer1].scam_attempts++;
                    if (peer1_wants_transaction) { account_matrix[peer1].scam_success
++; scam_success = true; }
                }

                // Peer2 SCAM assessment
                if (
                    (account_matrix[peer2].type != NODE_SCAM) &&
                    (account_matrix[peer1].type == NODE_SCAM)
                )
```

```
                {
                    account_matrix[peer2].scam_attempts++;
                    if (peer2_wants_transaction) { account_matrix[peer2].scam_success
++; scam_success = true; }
                }

                if (peer1_wants_transaction && peer2_wants_transaction)
                {
                    console.log('Transaction is performed.');

                    // Adding up the succeeded transactions
                    account_matrix[peer1].succeeded++;
                    account_matrix[peer2].succeeded++;
                    total_transactions++;
                    if (scam_success) total_scams++;

                    // Calculating the score from the first peer to the second
                    if (account_matrix[peer1].type == NODE_SCAM)
                    {
                        if (account_matrix[peer2].type == NODE_SCAM)
                        {
                            score_from_to = HIGH_SCORE;
                        }
                    }
                    else
                    {
                        if (account_matrix[peer2].type == NODE_SCAM)
                        {
                            score_from_to = LOW_SCORE;
                        }
                        else
                        {
                            score_from_to = HIGH_SCORE;
                        }
                    }

                    // Calculating the score from the second peer to the first
                    if (account_matrix[peer2].type == NODE_SCAM)
                    {
                        if (account_matrix[peer1].type == NODE_SCAM)
                        {
                            score_to_from = HIGH_SCORE;
                        }
                    }
                    else
                    {
                        if (account_matrix[peer1].type == NODE_SCAM)
                        {
                            score_to_from = LOW_SCORE;
                        }
                        else
                        {
                            score_to_from = HIGH_SCORE;
                        }
                    }

                    // Giving Score from the first peer to the second
                    if (score_from_to)
                    {
                        console.log('[1] gives score '+score_from_to+' to [2]');
                        await contract_handler.rateUser(account_matrix[peer2].address
, score_from_to, {from: account_matrix[peer1].address});
                    }

                    // Giving Score from the second peer to the first
                    if (score_to_from)
```

```
                    {
                        console.log('[2] gives score '+score_to_from+' to [1]');
                        await contract_handler.rateUser(account_matrix[peer1].address
, score_to_from, {from: account_matrix[peer2].address});
                    }
                }
                else
                {
                    console.log('Transaction is NOT performed.');
                }
            }

            console.log(" ");
            console.log("-----------------");
            console.log("| Final results |");
            console.log("-----------------");
            for (const [acc_key, acc_info] of Object.entries(account_matrix))
            {
                var acc_score = await contract_handler.getGeneralTrustScore(acc_info.
address, {from: _});

                log_message = "";
                log_message += "[Account "+acc_key+" - "+acc_info.type+"] ";

                // Transactions
                log_message += "Transactions: "+acc_info.succeeded+"/"+acc_info.trans
actions+" ";
                if (acc_info.transactions > 0)
                {
                    log_message += "("+(Math.round((acc_info.succeeded/acc_info.trans
actions) * 10000) / 100)+"%)";
                }
                else
                {
                    log_message += "(0%)";
                }
                log_message += "; ";

                // SCAMs
                log_message += "SCAMs: "+acc_info.scam_success+"/"+acc_info.scam_atte
mpts+" ";
                if (acc_info.scam_attempts > 0)
                {
                    log_message += "("+(Math.round((acc_info.scam_success/acc_info.sc
am_attempts) * 10000) / 100)+"%)";
                }
                else
                {
                    log_message += "(0%)";
                }
                log_message += "; ";

                // Trust Score
                log_message += "Trust Score: "+acc_score;

                console.log(log_message);
            }

            // Total Stats
            console.log('Total Transactions: '+total_transactions);
            console.log('Total Scams: '+total_scams);
            console.log('SCAM percentage: '+(Math.round((total_scams/total_transactio
ns) * 10000) / 100)+'%');

            assert.equal(1, 1, "something is wrong");
        });
```

```
    });
});
```

Compilamos y migramos el contrato una vez más a nuestra red local y ejecutamos el test. Los resultados que obtenemos son los siguientes.

```
josef@DESKTOP-8LVS0OR Z:\Dropbox\Development\Ethereum\dtrust-manager\truffle
$ truffle compile

Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.


josef@DESKTOP-8LVS0OR Z:\Dropbox\Development\Ethereum\dtrust-manager\truffle
$ truffle migrate

Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.


Starting migrations...
======================
> Network name:    'development'
> Network id:      5777
> Block gas limit: 0x6691b7


1_dtrust_manager.js
===================

   Replacing 'DTrustManager'
   -------------------------
   >                               transaction                                hash:
0x1d3329619f0009f0aa5e80f684022a8a7c895430fdc32bcb3a2dd0dd410b6db5
   > Blocks: 0            Seconds: 0
   > contract address:    0xde3524F1e246aE87c5Eb8c0fA5dAbb06364Dd113
   > block number:        1
   > block timestamp:     1576431739
   > account:             0x40e837FE16f1482EC20F1BbA7d543e706E134FEc
   > balance:             99.99191516
   > gas used:            404242
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.00808484 ETH

   > Saving artifacts
   -------------------------------------
   > Total cost:          0.00808484 ETH


Summary
=======
> Total deployments:   1
> Final cost:          0.00808484 ETH

josef@DESKTOP-8LVS0OR Z:\Dropbox\Development\Ethereum\dtrust-manager\truffle
$ truffle test .\test\simulations.js
Using network 'development'.
```

```
Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.



  Contract: DTrustManager
    Simulation
**************************************
First simulation: Transactions without DTrustManager
**************************************

------------------------
| Perform transactions |
------------------------
[1] 8 - WARY -- [2] 13 - NORMAL
Transaction is performed.
[1] 19 - RISKY -- [2] 11 - NORMAL
Transaction is performed.
[1] 19 - RISKY -- [2] 2 - SCAM
Transaction is performed.
[1] 5 - WARY -- [2] 8 - WARY
Transaction is performed.
[1] 17 - RISKY -- [2] 18 - RISKY
Transaction is performed.
[1] 0 - SCAM -- [2] 12 - NORMAL
Transaction is performed.
[1] 0 - SCAM -- [2] 16 - NORMAL
Transaction is performed.
[1] 6 - WARY -- [2] 0 - SCAM
Transaction is performed.
[1] 2 - SCAM -- [2] 18 - RISKY
Transaction is performed.
[1] 2 - SCAM -- [2] 11 - NORMAL
Transaction is performed.
[1] 4 - WARY -- [2] 10 - NORMAL
Transaction is performed.
[1] 9 - WARY -- [2] 14 - NORMAL
Transaction is performed.

[.......]
NO MOSTRAMOS TODO EL RESULTADO, YA QUE ES MUY LARGO
[.......]

-----------------
| Final results |
-----------------
[Account 0 - SCAM] Transactions: 17; SCAMs: 0
[Account 1 - SCAM] Transactions: 20; SCAMs: 0
[Account 2 - SCAM] Transactions: 38; SCAMs: 0
[Account 3 - WARY] Transactions: 20; SCAMs: 4
[Account 4 - WARY] Transactions: 14; SCAMs: 1
[Account 5 - WARY] Transactions: 20; SCAMs: 2
[Account 6 - WARY] Transactions: 18; SCAMs: 5
[Account 7 - WARY] Transactions: 18; SCAMs: 3
[Account 8 - WARY] Transactions: 18; SCAMs: 4
[Account 9 - WARY] Transactions: 23; SCAMs: 8
[Account 10 - NORMAL] Transactions: 15; SCAMs: 2
[Account 11 - NORMAL] Transactions: 16; SCAMs: 4
[Account 12 - NORMAL] Transactions: 19; SCAMs: 5
[Account 13 - NORMAL] Transactions: 19; SCAMs: 3
[Account 14 - NORMAL] Transactions: 27; SCAMs: 4
[Account 15 - NORMAL] Transactions: 17; SCAMs: 3
```

```
[Account 16 - NORMAL] Transactions: 18; SCAMs: 5
[Account 17 - RISKY] Transactions: 20; SCAMs: 6
[Account 18 - RISKY] Transactions: 21; SCAMs: 4
[Account 19 - RISKY] Transactions: 22; SCAMs: 6
**************************************
Total Transactions: 200
Total Scams: 69
SCAM percentage: 34.5%
**************************************

      √ performs first simulation: Transactions without DTrustManager (137ms)

**************************************
Second simulation: Transactions with DTrustManager (I)
**************************************

------------------------
| Perform transactions |
------------------------
[1] 8 - WARY (2) -- [2] 14 - NORMAL (2)
Transaction is NOT performed.
[1] 12 - NORMAL (2) -- [2] 10 - NORMAL (2)
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]
[1] 1 - SCAM (2) -- [2] 6 - WARY (2)
Transaction is NOT performed.
[1] 16 - NORMAL (2) -- [2] 1 - SCAM (2)
Transaction is performed.
[1] gives score 1 to [2]
[1] 10 - NORMAL (3) -- [2] 0 - SCAM (2)
Transaction is performed.
[1] gives score 1 to [2]
[1] 0 - SCAM (1) -- [2] 1 - SCAM (1)
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]
[1] 18 - RISKY (2) -- [2] 19 - RISKY (2)
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]
[1] 16 - NORMAL (2) -- [2] 4 - WARY (2)
Transaction is NOT performed.
[1] 12 - NORMAL (3) -- [2] 6 - WARY (2)
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]

[.......]
NO MOSTRAMOS TODO EL RESULTADO, YA QUE ES MUY LARGO
[.......]

-----------------
| Final results |
-----------------
[Account 0 - SCAM] Transactions: 6/21 (28.57%); SCAMs: 0/0 (0%); Trust Score: 1
[Account 1 - SCAM] Transactions: 9/26 (34.62%); SCAMs: 0/0 (0%); Trust Score: 1
[Account 2 - SCAM] Transactions: 7/16 (43.75%); SCAMs: 0/0 (0%); Trust Score: 1
[Account 3 - WARY] Transactions: 11/11 (100%); SCAMs: 0/0 (0%); Trust Score: 3
[Account 4 - WARY] Transactions: 10/16 (62.5%); SCAMs: 0/2 (0%); Trust Score: 3
[Account 5 - WARY] Transactions: 13/17 (76.47%); SCAMs: 0/3 (0%); Trust Score: 3
[Account 6 - WARY] Transactions: 14/18 (77.78%); SCAMs: 0/3 (0%); Trust Score: 3
[Account 7 - WARY] Transactions: 19/25 (76%); SCAMs: 0/5 (0%); Trust Score: 3
```

```
[Account 8 - WARY] Transactions: 10/17 (58.82%); SCAMs: 0/5 (0%); Trust Score: 3
[Account 9 - WARY] Transactions: 13/15 (86.67%); SCAMs: 0/1 (0%); Trust Score: 3
[Account 10 - NORMAL] Transactions: 19/21 (90.48%); SCAMs: 1/3 (33.33%); Trust
Score: 3
[Account 11 - NORMAL] Transactions: 22/26 (84.62%); SCAMs: 1/5 (20%); Trust Score:
3
[Account 12 - NORMAL] Transactions: 19/20 (95%); SCAMs: 0/1 (0%); Trust Score: 3
[Account 13 - NORMAL] Transactions: 12/17 (70.59%); SCAMs: 0/5 (0%); Trust Score: 3
[Account 14 - NORMAL] Transactions: 13/19 (68.42%); SCAMs: 0/4 (0%); Trust Score: 3
[Account 15 - NORMAL] Transactions: 15/18 (83.33%); SCAMs: 0/3 (0%); Trust Score: 3
[Account 16 - NORMAL] Transactions: 24/29 (82.76%); SCAMs: 1/4 (25%); Trust Score:
3
[Account 17 - RISKY] Transactions: 16/16 (100%); SCAMs: 4/4 (100%); Trust Score: 3
[Account 18 - RISKY] Transactions: 23/23 (100%); SCAMs: 1/1 (100%); Trust Score: 3
[Account 19 - RISKY] Transactions: 29/29 (100%); SCAMs: 4/4 (100%); Trust Score: 3
Total Transactions: 152
Total Scams: 12
SCAM percentage: 7.89%
      √ performs second simulation: Transactions with DTrustManager (74162ms)

**************************************
Third simulation: Transactions with DTrustManager (II)
**************************************

------------------------
| Perform transactions |
------------------------
[1] 1 - SCAM (2) -- [2] 3 - WARY (2)
Transaction is NOT performed.
[1] 9 - WARY (2) -- [2] 4 - WARY (2)
Transaction is NOT performed.
[1] 0 - SCAM (2) -- [2] 1 - SCAM (2)
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]
[1] 2 - SCAM (2) -- [2] 14 - NORMAL (2)
Transaction is performed.
[2] gives score 1 to [1]
[1] 16 - NORMAL (2) -- [2] 1 - SCAM (3)
Transaction is performed.
[1] gives score 1 to [2]
[1] 10 - NORMAL (2) -- [2] 1 - SCAM (2)
Transaction is performed.
[1] gives score 1 to [2]
[1] 9 - WARY (2) -- [2] 6 - WARY (2)
Transaction is NOT performed.
[1] 13 - NORMAL (2) -- [2] 0 - SCAM (1)
Transaction is NOT performed.
[1] 4 - WARY (2) -- [2] 15 - NORMAL (2)
Transaction is NOT performed.
[1] 12 - NORMAL (2) -- [2] 2 - SCAM (1)
Transaction is NOT performed.
[1] 10 - NORMAL (2) -- [2] 19 - RISKY (2)
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]
[1] 16 - NORMAL (2) -- [2] 17 - RISKY (2)
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]
[1] 4 - WARY (2) -- [2] 19 - RISKY (3)
Transaction is performed.
[1] gives score 3 to [2]
```

```
[2] gives score 3 to [1]

[.......]
NO MOSTRAMOS TODO EL RESULTADO, YA QUE ES MUY LARGO
[.......]

-----------------
| Final results |
-----------------
[Account 0 - SCAM] Transactions: 4/17 (23.53%); SCAMs: 0/0 (0%); Trust Score: 1
[Account 1 - SCAM] Transactions: 6/27 (22.22%); SCAMs: 0/0 (0%); Trust Score: 1
[Account 2 - SCAM] Transactions: 3/14 (21.43%); SCAMs: 0/0 (0%); Trust Score: 1
[Account 3 - WARY] Transactions: 13/18 (72.22%); SCAMs: 0/4 (0%); Trust Score: 3
[Account 4 - WARY] Transactions: 15/22 (68.18%); SCAMs: 0/3 (0%); Trust Score: 3
[Account 5 - WARY] Transactions: 21/28 (75%); SCAMs: 0/6 (0%); Trust Score: 3
[Account 6 - WARY] Transactions: 14/23 (60.87%); SCAMs: 0/5 (0%); Trust Score: 3
[Account 7 - WARY] Transactions: 10/15 (66.67%); SCAMs: 0/5 (0%); Trust Score: 3
[Account 8 - WARY] Transactions: 12/20 (60%); SCAMs: 0/4 (0%); Trust Score: 3
[Account 9 - WARY] Transactions: 13/18 (72.22%); SCAMs: 0/2 (0%); Trust Score: 3
[Account 10 - NORMAL] Transactions: 25/28 (89.29%); SCAMs: 1/4 (25%); Trust Score:
3
[Account 11 - NORMAL] Transactions: 13/16 (81.25%); SCAMs: 0/3 (0%); Trust Score: 3
[Account 12 - NORMAL] Transactions: 13/16 (81.25%); SCAMs: 0/3 (0%); Trust Score: 3
[Account 13 - NORMAL] Transactions: 10/13 (76.92%); SCAMs: 0/3 (0%); Trust Score: 3
[Account 14 - NORMAL] Transactions: 25/25 (100%); SCAMs: 1/1 (100%); Trust Score: 3
[Account 15 - NORMAL] Transactions: 14/18 (77.78%); SCAMs: 0/3 (0%); Trust Score: 3
[Account 16 - NORMAL] Transactions: 20/21 (95.24%); SCAMs: 1/2 (50%); Trust Score:
3
[Account 17 - RISKY] Transactions: 18/18 (100%); SCAMs: 1/1 (100%); Trust Score: 3
[Account 18 - RISKY] Transactions: 17/19 (89.47%); SCAMs: 1/1 (100%); Trust Score:
3
[Account 19 - RISKY] Transactions: 24/24 (100%); SCAMs: 6/6 (100%); Trust Score: 3
Total Transactions: 145
Total Scams: 11
SCAM percentage: 7.59%
      √ performs third simulation: Transactions with DTrustManager (72244ms)

**************************************
Fourth simulation: Transactions with DTrustManager (III)
**************************************

------------------------
| Perform transactions |
------------------------
[1] 19 - RISKY (2) -- [2] 13 - NORMAL (2)
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]
[1] 6 - WARY (2) -- [2] 7 - WARY (2)
Transaction is NOT performed.
[1] 0 - SCAM (2) -- [2] 6 - WARY (2)
Transaction is NOT performed.
[1] 4 - WARY (2) -- [2] 8 - WARY (2)
Transaction is NOT performed.
[1] 10 - NORMAL (2) -- [2] 13 - NORMAL (3)
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]
[1] 6 - WARY (2) -- [2] 4 - WARY (2)
Transaction is NOT performed.
[1] 14 - NORMAL (2) -- [2] 19 - RISKY (3)
Transaction is performed.
[1] gives score 3 to [2]
```

```
[2] gives score 3 to [1]
[1] 18 - RISKY (2) -- [2] 2 - SCAM (2)
Transaction is performed.
[1] gives score 1 to [2]
[1] 10 - NORMAL (3) -- [2] 3 - WARY (2)
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]
[1] 5 - WARY (2) -- [2] 4 - WARY (2)
Transaction is NOT performed.
[1] 18 - RISKY (2) -- [2] 13 - NORMAL (3)
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]

[.......]
NO MOSTRAMOS TODO EL RESULTADO, YA QUE ES MUY LARGO
[.......]

-----------------
| Final results |
-----------------
[Account 0 - SCAM] Transactions: 6/13 (46.15%); SCAMs: 0/0 (0%); Trust Score: 1
[Account 1 - SCAM] Transactions: 5/20 (25%); SCAMs: 0/0 (0%); Trust Score: 1
[Account 2 - SCAM] Transactions: 5/17 (29.41%); SCAMs: 0/0 (0%); Trust Score: 1
[Account 3 - WARY] Transactions: 16/17 (94.12%); SCAMs: 0/0 (0%); Trust Score: 3
[Account 4 - WARY] Transactions: 16/24 (66.67%); SCAMs: 0/4 (0%); Trust Score: 3
[Account 5 - WARY] Transactions: 24/27 (88.89%); SCAMs: 0/1 (0%); Trust Score: 3
[Account 6 - WARY] Transactions: 23/29 (79.31%); SCAMs: 0/3 (0%); Trust Score: 3
[Account 7 - WARY] Transactions: 20/26 (76.92%); SCAMs: 0/5 (0%); Trust Score: 3
[Account 8 - WARY] Transactions: 17/19 (89.47%); SCAMs: 0/1 (0%); Trust Score: 3
[Account 9 - WARY] Transactions: 12/16 (75%); SCAMs: 0/3 (0%); Trust Score: 3
[Account 10 - NORMAL] Transactions: 20/24 (83.33%); SCAMs: 0/4 (0%); Trust Score: 3
[Account 11 - NORMAL] Transactions: 15/17 (88.24%); SCAMs: 0/2 (0%); Trust Score: 3
[Account 12 - NORMAL] Transactions: 20/24 (83.33%); SCAMs: 1/4 (25%); Trust Score:
3
[Account 13 - NORMAL] Transactions: 14/17 (82.35%); SCAMs: 1/4 (25%); Trust Score:
3
[Account 14 - NORMAL] Transactions: 20/22 (90.91%); SCAMs: 0/2 (0%); Trust Score: 3
[Account 15 - NORMAL] Transactions: 17/17 (100%); SCAMs: 1/1 (100%); Trust Score: 3
[Account 16 - NORMAL] Transactions: 13/17 (76.47%); SCAMs: 0/3 (0%); Trust Score: 3
[Account 17 - RISKY] Transactions: 15/18 (83.33%); SCAMs: 1/1 (100%); Trust Score:
3
[Account 18 - RISKY] Transactions: 16/16 (100%); SCAMs: 3/3 (100%); Trust Score: 3
[Account 19 - RISKY] Transactions: 20/20 (100%); SCAMs: 5/5 (100%); Trust Score: 3
Total Transactions: 157
Total Scams: 12
SCAM percentage: 7.64%
      √ performs fourth simulation: Transactions with DTrustManager (75063ms)

**************************************
Fifth simulation: Transactions with DTrustManager (IV)
**************************************

------------------------
| Perform transactions |
------------------------
[1] 10 - NORMAL (2) -- [2] 1 - SCAM (2)
Transaction is performed.
[1] gives score 1 to [2]
[1] 4 - WARY (2) -- [2] 12 - NORMAL (2)
Transaction is NOT performed.
[1] 11 - NORMAL (2) -- [2] 6 - WARY (2)
```

```
Transaction is NOT performed.
[1] 16 - NORMAL (2) -- [2] 11 - NORMAL (2)
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]
[1] 8 - WARY (2) -- [2] 1 - SCAM (1)
Transaction is NOT performed.
[1] 1 - SCAM (1) -- [2] 6 - WARY (2)
Transaction is NOT performed.
[1] 12 - NORMAL (2) -- [2] 7 - WARY (2)
Transaction is NOT performed.
[1] 3 - WARY (2) -- [2] 13 - NORMAL (2)
Transaction is NOT performed.
[1] 13 - NORMAL (2) -- [2] 16 - NORMAL (3)
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]
[1] 13 - NORMAL (3) -- [2] 2 - SCAM (2)
Transaction is performed.
[1] gives score 1 to [2]
[1] 13 - NORMAL (3) -- [2] 1 - SCAM (1)
Transaction is NOT performed.
[1] 3 - WARY (2) -- [2] 10 - NORMAL (2)
Transaction is NOT performed.
[1] 19 - RISKY (2) -- [2] 16 - NORMAL (3)
Transaction is performed.
[1] gives score 3 to [2]
[2] gives score 3 to [1]
[1] 6 - WARY (2) -- [2] 5 - WARY (2)
Transaction is NOT performed.

[.......]
NO MOSTRAMOS TODO EL RESULTADO, YA QUE ES MUY LARGO
[.......]

-----------------
| Final results |
-----------------
[Account 0 - SCAM] Transactions: 4/15 (26.67%); SCAMs: 0/0 (0%); Trust Score: 1
[Account 1 - SCAM] Transactions: 11/29 (37.93%); SCAMs: 0/0 (0%); Trust Score: 1
[Account 2 - SCAM] Transactions: 8/20 (40%); SCAMs: 0/0 (0%); Trust Score: 1
[Account 3 - WARY] Transactions: 12/21 (57.14%); SCAMs: 0/5 (0%); Trust Score: 3
[Account 4 - WARY] Transactions: 12/16 (75%); SCAMs: 0/0 (0%); Trust Score: 3
[Account 5 - WARY] Transactions: 17/24 (70.83%); SCAMs: 0/2 (0%); Trust Score: 3
[Account 6 - WARY] Transactions: 19/26 (73.08%); SCAMs: 0/2 (0%); Trust Score: 3
[Account 7 - WARY] Transactions: 16/25 (64%); SCAMs: 0/6 (0%); Trust Score: 3
[Account 8 - WARY] Transactions: 14/17 (82.35%); SCAMs: 0/2 (0%); Trust Score: 3
[Account 9 - WARY] Transactions: 10/21 (47.62%); SCAMs: 0/7 (0%); Trust Score: 3
[Account 10 - NORMAL] Transactions: 10/12 (83.33%); SCAMs: 1/2 (50%); Trust Score:
3
[Account 11 - NORMAL] Transactions: 12/15 (80%); SCAMs: 1/3 (33.33%); Trust Score:
3
[Account 12 - NORMAL] Transactions: 17/24 (70.83%); SCAMs: 0/1 (0%); Trust Score: 3
[Account 13 - NORMAL] Transactions: 10/17 (58.82%); SCAMs: 1/7 (14.29%); Trust
Score: 3
[Account 14 - NORMAL] Transactions: 17/21 (80.95%); SCAMs: 1/4 (25%); Trust Score:
3
[Account 15 - NORMAL] Transactions: 13/15 (86.67%); SCAMs: 0/2 (0%); Trust Score: 3
[Account 16 - NORMAL] Transactions: 19/21 (90.48%); SCAMs: 0/2 (0%); Trust Score: 3
[Account 17 - RISKY] Transactions: 17/17 (100%); SCAMs: 5/5 (100%); Trust Score: 3
[Account 18 - RISKY] Transactions: 22/22 (100%); SCAMs: 4/4 (100%); Trust Score: 3
[Account 19 - RISKY] Transactions: 22/22 (100%); SCAMs: 6/6 (100%); Trust Score: 3
Total Transactions: 141
```

```
Total Scams: 19
SCAM percentage: 13.48%
      √ performs fifth simulation: Transactions with DTrustManager (66142ms)


  5 passing (5m)
```

Como podemos observar, todas las simulaciones utilizando el algoritmo de confianza mejoran los resultados de la simulación sin el algoritmo de confianza. La simulación sin el algoritmo arrojó un porcentaje de SCAM del 34,5%, mientras que la media del resto de pruebas arroja un porcentaje del 9%.

En todas las simulaciones con el algoritmo, los nodos maliciosos han sido rápidamente identificados y el resto de nodos ha tomado como decisión no interactuar con ellos. Los nodos maliciosos sólo han ejecutado alrededor del 25-30% de las transacciones, que probablemente incluyan bastantes transacciones entre ellos. El resto de nodos han ejecutado más transacciones entre ellos, teniendo siempre en cuenta su comportamiento por defecto.

Este resultado nos arroja una primera impresión positiva sobre el algoritmo implementado.

8.1.8 Configuración y despliegue en la red de testing pública Rinkeby

Para terminar, se muestra a continuación cómo desplegar el Smart Contract a una blockchain pública como puede ser la “mainnet”, la red principal de Ethereum, o una red de test como puede ser Rinkeby (https://www.rinkeby.io).

Podemos utilizar Rinkeby de manera gratuita, únicamente necesitamos un servicio de pasarela que nos permita conectar con Rinkeby, y una cuenta en la que añadir algunos ETH. El servicio de pasarela que utilizaremos será Infura (https://infura.io/). Al crear una cuenta en su web, podemos crear también un proyecto y obtener una clave para conectarnos a Rinkeby.

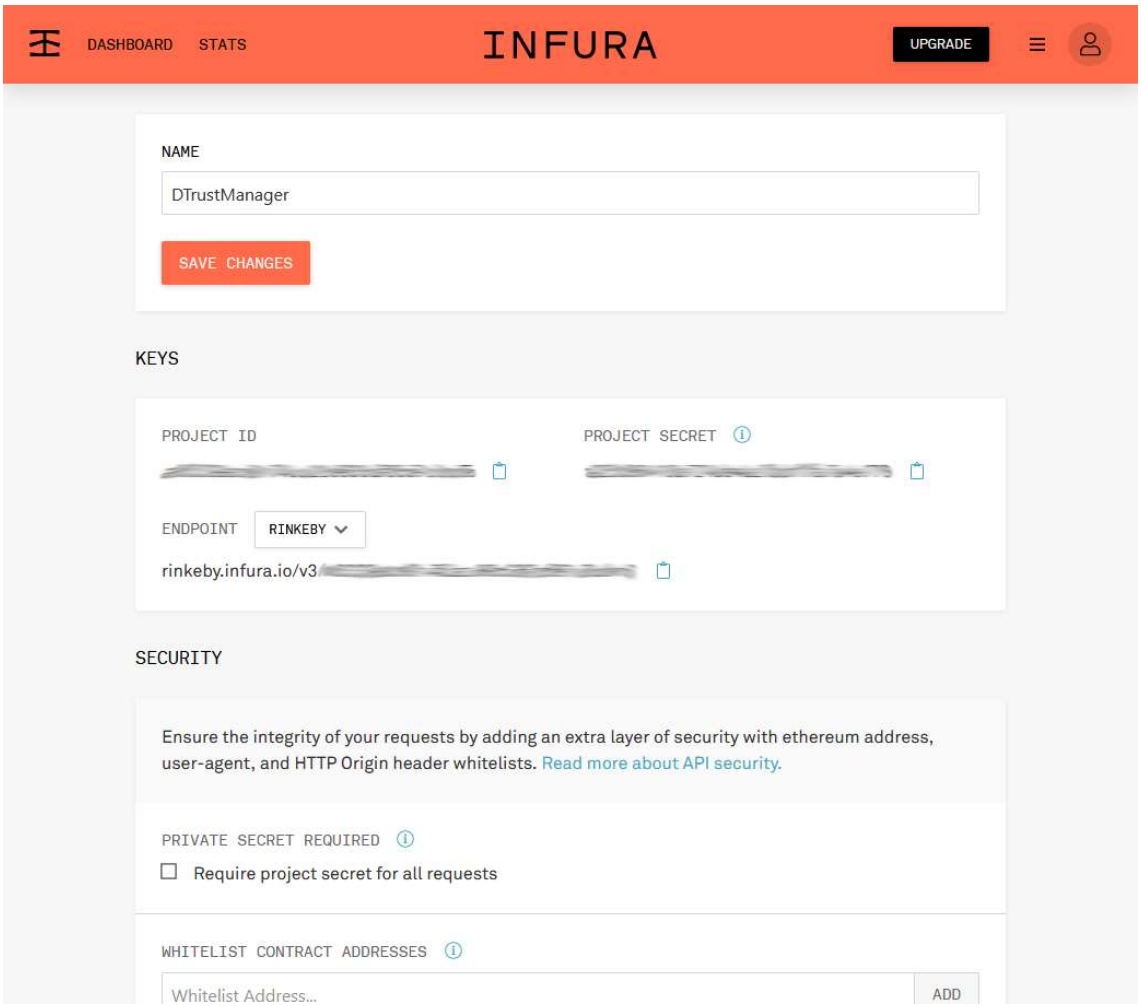


Ilustración 26: Panel de control de Infura. Imagen Propia.

Lo siguiente que necesitamos hacer es crear una cuenta en la red Rinkeby y obtener algunos ETH para poder trabajar en ella. Para ello, crearemos la cuenta directamente con Metamask, seleccionando en primer lugar la red Rinkeby. Podemos añadir ETH a una cuenta de Rinkeby a través de su servicio de emisión de ETH: https://faucet.rinkeby.io/.

Una vez tenemos la cuenta creada, procedemos a configurar Truffle para desplegar el Smart Contract en la red Rinkeby. Necesitaremos una librería más para lograr nuestro objetivo, la truffle-hdwallet-provider. Esta librería nos permitirá conectar una cuenta de blockchain mediante su clave privada, para poder firmar operaciones con esa cuenta, ya que de otra manera no podríamos desplegar el contrato en la red de Rinkeby. Instalamos, pues, la nueva librería.

```
josef@DESKTOP-8LVS0OR Z:\Informatics\Development\Ethereum\dtrustmanager
$ npm install truffle-hdwallet-provider
```

Seguidamente, modificaremos la configuración de Truffle para añadir la red de Rinkeby. Utilizaremos la cuenta creada con Metamask para desplegar el contrato.

```
var HDWalletProvider = require("truffle-hdwallet-provider");
var mnemonic = "XXXX fancy XXXX city XXXX tuition XXXX love XXXX reveal XXXX lucky";

module.exports = {
```

```
    networks: {
        // Useful for testing. The `development` name is special -
 truffle uses it by default
        // if it's defined here and no other network is specified at the command line
.
        // You should run a client (like ganache-
cli, geth or parity) in a separate terminal
        // tab if you use this network and you must also set the `host`, `port` and `
network_id`
        // options below to some value.
        //
        development: {
            host: "127.0.0.1",     // Localhost (default: none)
            port: 7545,            // Standard Ethereum port (default: none)
            network_id: "*",       // Any network (default: none)
            from: "0x93B459B630Be5D1563A9B898c092F86f02939314"
        },

        rinkeby: {
            provider: function()
            {
                return new HDWalletProvider(mnemonic, "https://rinkeby.infura.io/v3/a
6333ead9a7f4ca28e636af85e7bdbe5");
            },
            network_id: 4,
            gas: 4500000,           // 4.500.000
            gasPrice: 20000000000,  // 10.000.000.000
            from: "0x68c956Cae0b691Be4F270ce7A3Fd2CE7f281D434"
        }
    }
}
```

Finalmente, migramos el contrato con el mismo comando utilizado para migrar a local, pero especificando la red a la que queremos migrar.

```
josef@DESKTOP-8LVS0OR Z:\Informatics\Development\Ethereum\dtrustmanager
$ truffle migrate --network rinkeby

Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.


Migrations dry-run (simulation)
===============================
> Network name:    'rinkeby-fork'
> Network id:      4
> Block gas limit: 0x989680


1_dtrust_manager.js
===================

   Deploying 'DTrustManager'
   -------------------------
   > block number:        5573704
   > block timestamp:     1575744776
   > account:             0x68c956Cae0b691Be4F270ce7A3Fd2CE7f281D434
   > balance:             9.832581064
   > gas used:            702140
   > gas price:           20 gwei
   > value sent:          0 ETH
```

```
   > total cost:          0.0140428 ETH

   -------------------------------------
   > Total cost:           0.0140428 ETH


Summary
=======
> Total deployments:   1
> Final cost:          0.0140428 ETH


Starting migrations...
======================
> Network name:    'rinkeby'
> Network id:      4
> Block gas limit: 0x989680


1_dtrust_manager.js
===================

   Deploying 'DTrustManager'
   -------------------------
   >                              transaction                                hash:
0x0f084899ad399975be9f93533c4f4c54b9886e3f0e455a70d8676d117284fc4f
   > Blocks: 1            Seconds: 16
   > contract address:    0xa6776A857A4c0B21394eEF5628b60F9c29A050cA
   > block number:        5573705
   > block timestamp:     1575744800
   > account:             0x68c956Cae0b691Be4F270ce7A3Fd2CE7f281D434
   > balance:             9.835010504
   > gas used:            580668
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.01161336 ETH

   > Saving artifacts
   -------------------------------------
   > Total cost:          0.01161336 ETH


Summary
=======
> Total deployments:   1
> Final cost:          0.01161336 ETH
```