ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

GRADO EN INGENIERÍA TELEMÁTICA

IMPLEMENTACIÓN Y DESPLIEGUE DE UNA SOLUCIÓN DE GAMIFICACIÓN EN LA EMPRESA SOBRE REDES PÚBLICAS Y PRIVADAS BLOCKCHAIN

Autor: María Pilar Hernández Bas

Director: David Contreras Bárcena

Madrid

Julio 2018

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

IMPLEMENTACIÓN Y DESPLIEGUE DE UNA SOLUCIÓN DE GAMIFICACIÓN EN LA EMPRESA SOBRE REDES PÚBLICAS Y PRIVADAS BLOCKCHAIN en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2017/18 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

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

tomada de otros documentos está debidamente referenciada.

Fdo.: María Pilar Hernández Bas Fecha: 11/ 7/ 2018

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: David Contreras Bárcena Fecha: 11/ 7/ 2018

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

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

El autor Dña. María Pilar Hernández Bas DECLARA ser el titular de los derechos de propiedad intelectual de la obra: "Implementación y despliegue de una solución de gamificación en la empresa sobre redes privadas y públicas blockchain" , que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

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

3º. Condiciones de la cesión y acceso

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

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

4º. Derechos del autor.

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

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

5º. Deberes del autor.

El autor se compromete a:

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

de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

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

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

Madrid, a 11 de julio de 2018

ACEPTA

Fdo……

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

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

GRADO EN INGENIERÍA TELEMÁTICA

IMPLEMENTACIÓN Y DESPLIEGUE DE UNA SOLUCIÓN DE GAMIFICACIÓN EN LA EMPRESA SOBRE REDES PUBLICAS Y PRIVADAS BLOCKCHAIN

Autor: María Pilar Hernández Bas

Director: David Contreras Bárcena

Madrid

Julio 2018

Agradecimientos

A mis padres, por enseñarme el valor del trabajo.

A mi familia, por el apoyo incondicional durante estos años y nunca dejar de confiar.

A David, por proponerme este proyecto y guiarme durante todo el camino.

A los amigos de verdad, los que están siempre ahí.

IMPLEMENTACIÓN Y DESPLIEGUE DE UNA SOLUCIÓN DE GAMIFICACIÓN EN LA EMPRESA SOBRE REDES PÚBLICAS Y PRIVADAS BLOCKCHAIN.

Autor: María Pilar Hernández Bas.
Director: David Contreras Bárcena.
Entidad Colaboradora: ICAI - Universidad Pontificia Comillas.

RESUMEN DEL PROYECTO

Aprovechando el auge y la popularidad que la tecnología Blockchain suscita en los entornos empresariales y de investigación, se propone un proyecto que busca conocer más a fondo esta nueva revolución tecnológica, así como implementar una nueva aplicación basada en Smart Contracts que pueda funcionar en una red real como la desarrollada por el consorcio Alastria.

Palabras clave: Blockchain, Smart Contracts, Ethereum, Quorum, Alastria

1. Introducción

Blockchain o cadena de bloques en español es una tecnología que busca eliminar los intermediarios, descentralizando la gestión de las transacciones. Cada nodo participante de la red es el encargado de la verificación y autenticación de las transacciones. Agilizar los procesos, gestionar la trazabilidad de las operaciones o la inmutabilidad de las transacciones son algunos de los atractivos que motivan a las empresas a buscar nuevos casos de uso que sirvan para mejorar los servicios que se ofrecen en la actualidad. [1]

Esta tecnología todavía está en fase de experimentación y desarrollo, con proyectos que nos permiten vislumbrar todas las posibilidades que podría traer consigo. El quid de la cuestión ahora es encontrar casos de uso reales y posibles que nos hagan sacar partido de los recursos disponibles beneficiando así a los usuarios.

2. Definición del proyecto

El proyecto se puede definir en tres etapas. La primera fase será realizar un estudio en profundidad de la tecnología blockchain y sus características y funcionamiento.

Aprovechando los conocimientos adquiridos durante la fase anterior, se explorará el concepto de contrato inteligente y se propondrá una solución empresarial basada en el concepto de gamificación para la recompensa de empleados a través de un sistema de objetivos. Se utilizarán diferentes redes tanto públicas como privadas para las pruebas y el desarrollo.

Por último, lanzaremos el proyecto a una red dónde podamos ver la interacción de los contratos inteligentes desarrollados con la red desplegada. Esta aplicación se alojará en un servidor. El objetivo final de este proyecto es poner en funcionamiento un caso de uso que podría implementarse en un ambiente empresarial con una red blockchain como la red del consorcio Alastria.

3. Descripción del modelo/sistema/herramienta

El caso de uso que se va a implementar consiste en una aplicación basada en el concepto gamificación por el que los empleados son motivados e incentivados por medio de un sistema de retos y recompensas a través de la emisión de tokens por parte de la empresa.

La funcionalidad de la aplicación sería la siguiente: la empresa genera una cantidad de tokens que el empleado conforme supera los objetivos propuesto va pudiendo reclamar y acumular en su cartera para después canjear esos puntos por recompensas o ventajas.

El sistema es un modelo sencillo que trabaja a nivel de aplicación en una red blockchain, migrando los contratos a un nodo capaz de desplegar aplicaciones, como el nodo general de la red Alastria. La aplicación estará corriendo sobre un servidor e interactuará con el cliente, en este caso los empleados o la empresa, a través del navegador. Metamask será la herramienta o intermediario entre nuestra aplicación y los contratos, a través de la librería Web3.js. Para el desarrollo y pruebas lanzaremos los contratos en la red local de Ganache que nos ofrece Truffle antes de migrar a una red real.

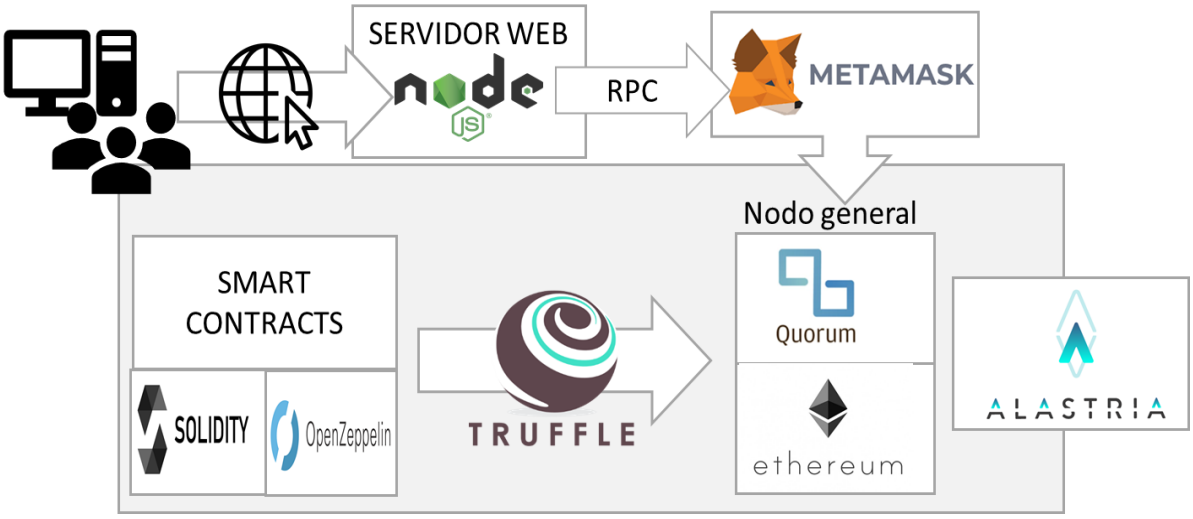


Ilustración 1. Diagrama de arquitectura del proyecto

4. Resultados

Los resultados obtenidos tras el diseño de la aplicación y su implementación cumplen con los objetivos propuestos al comienzo del proyecto. A través de la aplicación la empresa es capaz de crear los tokens con el estándar ERC20 y los empleados son capaces de reclamar los objetivos conseguidos y canjear esos puntos por recompensas. En la siguiente ilustración se muestra la página de retos donde los empleados pueden ver los objetivos que ya han conseguido lograr. Los empleados podrán reclamar sus puntos, a través de Metamask, y serán añadidos al balance total del empleado. La aplicación tiene un funcionamiento muy sencillo e intuitivo ya que el objetivo principal es poder observar la interacción de la aplicación con los contratos y la red.

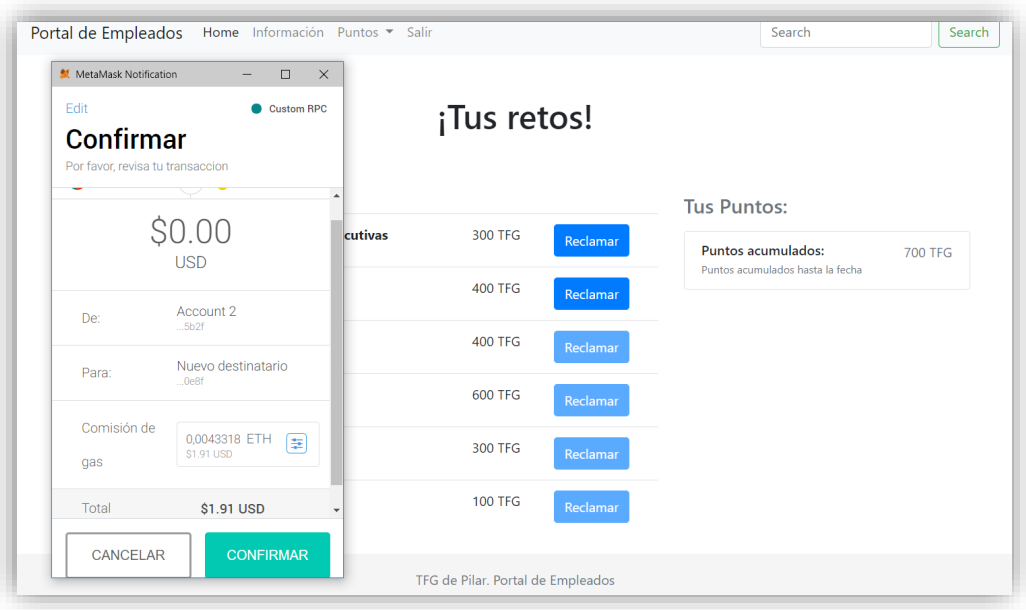


Ilustración 2. Transacción con Metamask al reclamar puntos

5. Conclusiones

La finalidad de este proyecto era adentrase en el conocimiento de esta tecnología y conseguir una visión más específica de las posibilidades que aporta al momento actual en el que nos encontramos y a un futuro no muy lejano.

Este estudio ha permitido adquirir una imagen más realista y profunda de lo que se puede conseguir con los recursos actuales para diseñar un caso de uso acorde a la tecnología y la red. La aplicación se ha desarrollado usando código auditado y estandarizado para los tokens ERC20, y para el contrato de la gestión de los tokens se ha intentado utilizar código simple sin redundancias ni bucles que consumen más recursos de la red.

La migración a la red real de Alastria ha sido el paso más complejo del trabajo, se ha trabajado con el nodo general de la universidad, situado en el clúster al que se ha accedido a de forma remota. Se ha configurado el proyecto según los requisitos de la red como el precio del gas y el límite de gas por contrato. Estas redes y herramientas están en constante desarrollo, por lo que a veces no se obtienen los resultados exactamente esperados. Sin embargo, proyectos como Alastria, con equipos formados por empresas de todos los sectores, realizan grandes avances en eficiencia y seguridad de las redes gracias a la comunidad de desarrolladores.

Blockchain supone un reto tecnológico al que le quedan unos años de madurez y estudio antes de ser totalmente implementado, pero ya nos permite hacernos una idea de las posibilidades que conlleva esta tecnología y todas las ventajas y beneficios que ofrece.

6. Referencias

[1] Observatorio Blockchain Sngular, Informe de noviembre de 2017. https://sngular.team/wp-content/uploads/2018/01/observatorio-blockchain-sngular.pdf

IMPLEMENTATION AND DEPLOYMENT OF A GAMIFICATION SOLUTION FOR COMPANIES ON PUBLIC AND PRIVATE BLOCKCHAIN NETWORKS

Author: Hernández Bas, María Pilar.
Supervisor: Contreras Bárcena, David.
Collaborating Entity: ICAI – Universidad Pontificia Comillas.

ABSTRACT

Taking advantage of the boom and popularity of Blockchain technology in business and research environments, a project is proposed that seeks to learn more about this new technological revolution, as well as implement a new application based on Smart Contracts that can work in a real network as developed by the Alastria consortium.

Keywords: Blockchain, Smart Contracts, Ethereum, Quorum, Alastria

1. Introduction

Blockchain is a technology that seeks to eliminate intermediaries by decentralizing the management of transactions. Each participating node of the network is responsible for the verification and authentication of transactions. Streamlining processes, managing the traceability of operations or the immutability of transactions are some of the attractions that motivate companies to look for new use cases that serve to improve the services currently offered.[1]

This technology is still in the experimentation and development phase, with projects that allow us to glimpse all the possibilities it could bring. The important matter now is to find real and potential use cases that will enable us to take advantage of available resources to benefit users.

2. Project definition

The project can be defined in three stages. The first phase will be an extensive study of the blockchain technology and its characteristics.

Based on the concept of smart contracts and all the knowledge from the previous stage, a business solution based on the concept of gamification will be proposed. It will be a rewarding system for employees who achieve their objectives. Different public and private networks will be used for testing and development purposes.

Finally, we will launch the application into a network where we can see the interaction of the smart contracts with the network. The application will be hosted on a server. The final objective of this project is to put into operation a use case that could be implemented in a business environment in a blockchain network such as the Alastria consortium network.

3. System description

The project to be implemented consists of an application based on the gamification concept. Employees are motivated to obtain rewards through the issuance of tokens by the company.

The functionality of the application would be as follows: the company generates several tokens and then the employee can claim and accumulate tokens in his or her wallet in order to later change those points for prizes or advantages.

The system is a simple model that works at the application level in a blockchain network. We would have to migrate the contracts to a node capable of deploying applications, such as the general node of the Alastria network. The application will be running on a server and will interact with the client through the browser. Metamask will be the tool or intermediary between our application and the contracts, through the Web3.js library. For development and testing we will launch the contracts on the local Ganache network offered by Truffle and then we will be ready to migrate to a real network.

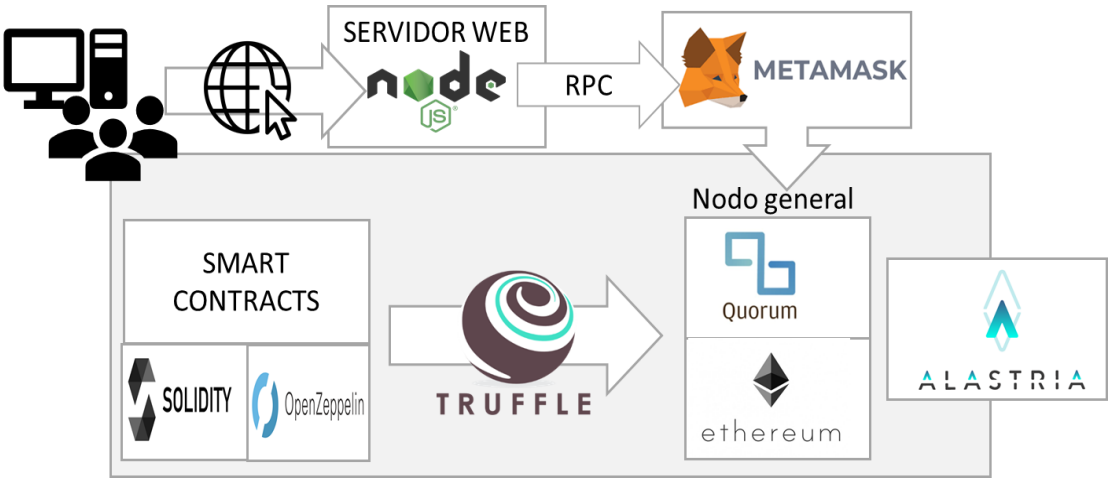


Ilustración 3. System´s architecture diagram

4. Results

The results obtained after the design of the application and its implementation comply with the objectives proposed at the beginning of the project. Through the application the company can create tokens with the ERC20 standard. Employees, on the other hand, can claim the objectives achieved and redeem those points for rewards. The following illustration shows an example of the screen where employees can see the challenges they have already achieved. Employees will be able to claim their goals with Metamask and the points will be added to the total balance. The application has a very simple and intuitive interface since the

main objective is to be able to observe the interaction between the application and contracts.

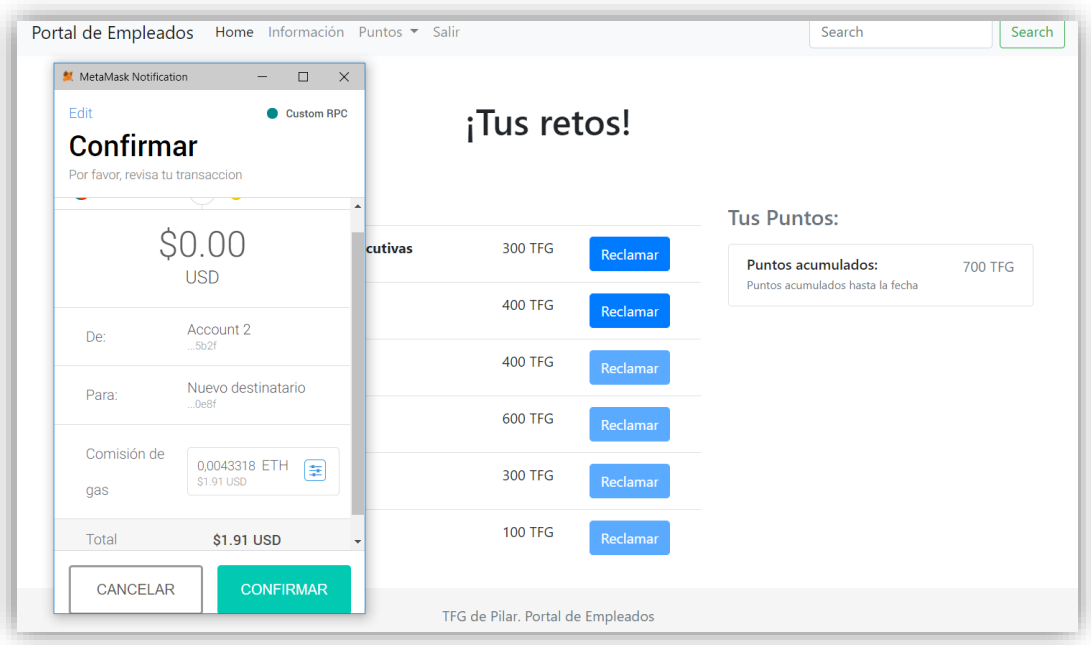


Ilustración 4. Metamask transaction when obtaining new points

5. Conclusions

The aim of this project was to get to know this technology and get a more specific vision of the possibilities it may bring to the current moment and in a nearly future.

This study has allowed us to acquire a more realistic image of what can be achieved with current resources. The application has been designed in accordance with the technology and the network. It has been designed using audited and standardized code for the ERC20 tokens. For the token management contract, we have tried to use simple code without redundancies to not consume more network resources than needed.

The migration to the real Alastria network has been the most complex step of the project. We have worked with the regular node of the university, located in the cluster that has been accessed remotely. The project has been configured according to network requirements such as gas price and gas limit. These networks and tools are constantly being developed, so that sometimes the results are not exactly as we expected. However, projects such as Alastria, with teams made of companies from all sectors, make great advance in network efficiency and security thanks to the developer community.

Blockchain is a technological challenge that needs a few years of maturity left before being fully implemented, but it already allows us to understand all the possibilities that this technology involved and all the advantages and benefits it offers us.

6. References

[1] Observatorio Blockchain Sngular, Informe de noviembre de 2017. https://sngular.team/wp-content/uploads/2018/01/observatorio-blockchain-sngular.pdf

Índice de la memoria

Índice de figuras

Índice de tablas

Capítulo 1. INTRODUCCIÓN

En la actualidad Blockchain supone una tecnología en plena fase de desarrollo e implementación. Numerosas empresas e instituciones ven en ésta una oportunidad para introducir elementos disruptivos en sus negocios, de manera que sean más seguros y fiables. La tecnología ya se encuentra en un estado bastante avanzado de desarrollo y pruebas, ahora el quid de la cuestión se encuentra en encontrar aplicaciones que se puedan aplicar al mundo de los negocios. Empresas de todos los sectores unen fuerzas para poder explotar esta tecnología en beneficio de las diferentes industrias.

Blockchain o cadena de bloques en español es una tecnología que busca eliminar los intermediarios, descentralizando la gestión de los intercambios y las operaciones. Agilizar los procesos, gestionar la trazabilidad de los procesos y la inmutabilidad de las transacciones son algunos de los atractivos que motivan a las empresas a buscar nuevos casos de uso que sirvan para mejorar los servicios que se ofrecen en la actualidad. A pesar del potencial más que obvio para los servicios y sistemas financieros, son muchas empresas de diferentes industrias y sectores que estudian esta tecnología, siendo de especial interés en aplicaciones de IoT o seguridad e identidad digital.

La naturaleza de Blockchain permite la descentralización y eliminación de forma considerable de las entidades reguladoras, haciendo que cada elemento o nodo participante de la red forme parte y se haga cargo de la veracidad y autenticación de las transacciones y operaciones que se realizan. No solo eso, permite crear redes públicas, accesibles por todos, además redes privadas que pueden beneficiar a la interconexión de empresas y servicios.

Blockchain reúne a todos los integrantes de procesos y les permite participar y ser protagonistas de las decisiones, de la veracidad, ya que son los mayores interesados en el éxito de la red. Además, incentiva y premia aquellos que más colaboran de forma que contribuye también a un mayor reparto de forma equitativa de los recursos de la red.

Este proyecto surge de la necesidad de comprender la utilidad y lo que hay detrás de esta nueva tecnología, no solo conocerla, si no ponerla en práctica, observar su comportamiento y proponer nuevas ideas y casos de uso para su implantación. La universidad especialmente implicada en proyectos sobre las cadenas de bloques supone el marco idóneo para realizar las pruebas de concepto y de ese modo estar al tanto de las últimas novedades y avances cuanto a tecnología se refiere. De ahí que sea importante conocer qué supone este concepto que está siendo estudiado por multitud de empresas, universidades e instituciones de manera que se puedan buscar aplicaciones que sirvan para implementar en el mundo actual.

Este trabajo supone una oportunidad para estar al tanto de una tecnología disruptiva y novedosa que podrá revolucionar las industrias en un futuro no muy lejano, por lo que es importante conocerla, entenderla y saber aplicarla. Blockchain es una tecnología con unas posibilidades muy amplias y dispares que necesita aplicaciones concretas al mundo real.

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

En este apartado se van a mencionar las diferentes tecnologías que se han usado para el desarrollo del proyecto en sus diferentes fases.

2.1 BLOCKCHAIN

2.1.1 DEFINICIÓN

De una forma muy simple podemos definir Blockchain como una base de datos compartida que contiene un registro de transacciones. Este registro realiza un seguimiento de la propiedad de una moneda. Cada participante posee una copia actualizada de ese registro de manera que puede verificar cada una de las cuentas. Cada dispositivo conectado que disponga de una copia del registro será lo que de aquí en adelante llamaremos nodo.

Blockchain elimina el problema de confianza que afecta a otras bases de datos de la siguiente manera:

- Descentralización completa: ninguna persona o grupo puede controlar una cadena de bloques, la lectura y escritura de información en la base de datos es descentralizada y segura.
- Tolerancia a fallos muy alta: al haber muchos participantes en esa red, y todos manejar la misma información actualizada, en caso de que falle uno o varios de los nodos, la comprobación y actualización de la red posibilita reparar o corregir los errores que se den por información corrupta, asegurando además la seguridad de la red.
- Verificación independiente: cualquier participante o incluso persona ajena a la red, en caso de que hablemos de una red pública, puede verificar las cuentas sin la necesidad de una entidad reguladora. No necesita intermediaros.

2.1.2 FUNCIONAMIENTO

Llamamos Blockchain o cadena de bloques a una tecnología que funciona como un libro de contabilidad digital el cual es distribuido de manera actualizada a todos los participantes que conforman la red para garantizar seguridad y confiabilidad eliminando la necesidad de una autoridad reguladora y central.

Las interacciones entre cuentas en una red son llamadas transacciones y son estas interacciones las que se registran dentro de la cadena de bloques. Pueden ser transacciones monetarias como por ejemplo enviar Ether, la criptomoneda usada en Ethereum. Sin embargo, pueden ser transmisiones de información como un comentario o un nombre de usuario. Cada paquete de transacciones se conoce como bloque. En la tecnología Blockchain cada bloque hace referencia al bloque anterior a través de lo que se conoce como "*Hash*" que se podría entender como la huella digital de nuestro bloque. Este proceso del *hash* es debido a un algoritmo matemático que toma todos los datos que contienen estos bloques y los convierte en una cadena de caracteres de longitud fija (el mencionado *hash*). El *hash* será marcado temporalmente para conocer la fecha exacta y el momento en el que se procesó un bloque, además de las transacciones que han sido incluidas en cada bloque. [1]

Un bloque es un conjunto de transacciones confirmadas que incluyen además información adicional. Cada bloque que forma parte de la cadena está compuesto por:

1. Un código alfanumérico que enlaza con el bloque anterior.
2. El paquete de transacciones confirmadas que incluye el bloque.
3. Otro código alfanumérico que enlazará con el siguiente bloque.

El nodo que está minado el bloque ,que se encuentra en progreso, lo que pretende averiguar a través de cálculos es el código alfanumérico que servirá para enlazar con el bloque que venga a continuación. Este código sigue unas determinadas reglas para ser válido y solo puede sacarse mediante fuerza bruta y capacidad de computación.

Cada cuenta en blockchain tiene una estructura única, lo que permite que todo el mundo sea capaz de identificar la cuenta o nodo que inició la transacción. En una cadena pública, todo el mundo puede leer o escribir información en la cadena de bloques. Leer información de

una cadena no tiene ningún coste, pero, en el caso de querer escribir, entonces hay un coste asociado. Este coste que es conocido como "gas" ayuda a disuadir el spam, fuerza a optimizar los códigos y paga la seguridad de la red.

Una de las características principales de esta tecnología es la confiabilidad y de esta manera cualquier nodo en la red puede tomar parte en la aseguración de la red a través de un proceso llamado "minería" (*mining*). Los nodos que optan a ser mineros compiten para resolver problemas matemáticos que protegen los contenidos del bloque. Como este proceso requiere energía de computación y coste de electricidad, los mineros pueden ser recompensados por su servicio. El ganador de la competición recibe dinero en forma de criptomonedas como premio y a su vez compensar el gasto de computación. Esto incentiva a los nodos para trabajar asegurando la red y previniendo que mucho poder acabe en manos de un simple minero. El concepto de minado se explica de una forma más extensa en el apartado: Concepto de minado.

Una vez ese bloque es minado, los otros mineros son notificados y comienzan verificando y añadiendo este nuevo bloque a sus copias de la cadena. Esto se hace mediante el concepto de "*hashing*". Un *hash* es un proceso en una sola dirección en el que se coge la información y se devuelve una cadena de caracteres de longitud fija representando esa información. Mientras la información original no puede ser obtenida a través de su hash, la misma información siempre producirá ese mismo hash. De esa forma, la información que no está verificada puede ser hasheada con la misma función y ser comparada con la original. Si son idénticas las cadenas, la información es validada. En el momento en el que más de la mitad de los mineros han validado ese nuevo bloque, y se llega a un consenso, entonces el bloque pasa a ser parte de la cadena de bloques de forma permanente. Llegados a este punto la información puede ser descargada y usada por todos y cada uno de los nodos, sabiendo que está validada.

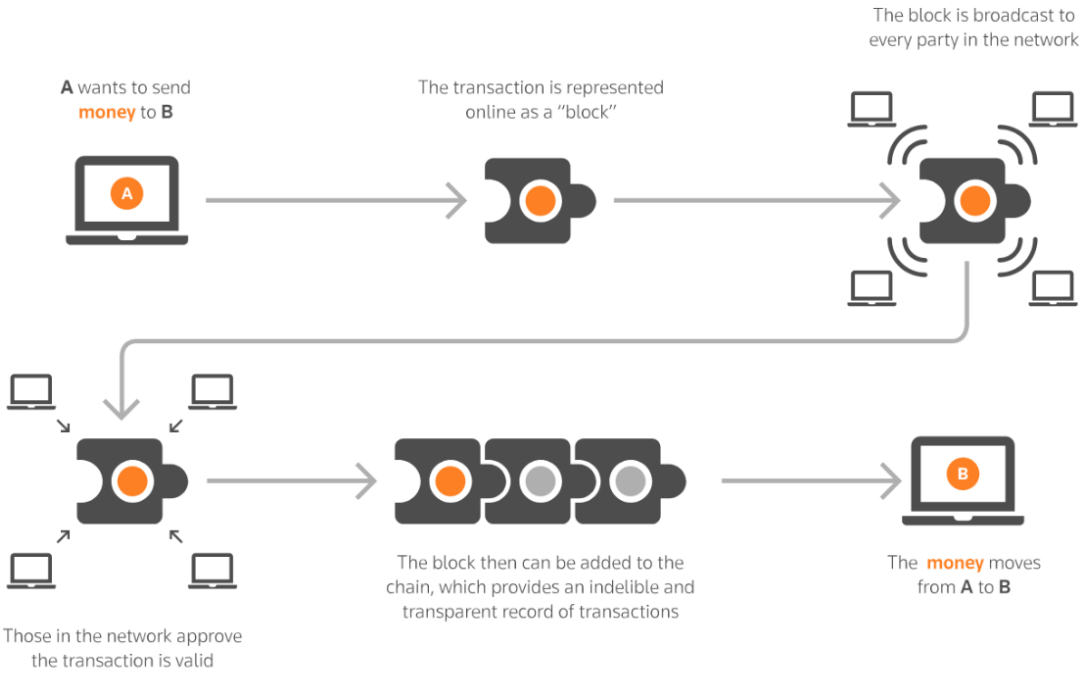


Figura 1. Transacción Blockchain. (Fuente: Financial Times)

2.1.3 ALGORITMOS DE CONSENSO

En un sistema descentralizado surge el problema de la confianza ya que no contamos con una autoridad central que regule y apruebe las transacciones. Son los propios nodos participantes de la red los que se van a encargar de verificar esas transacciones e incorporarlas a la cadena de bloques. Es por esta razón que se debe encontrar un mecanismo que se encargue de consensuar las transacciones y operaciones que se van a añadir en la cadena de bloques. Esto se consigue mediante algoritmos de consenso, los nodos deberán ponerse de acuerdo en cuales son los bloques de transacciones que realmente representan el histórico de transacciones de la red según unas reglas determinadas. A su vez, estos algoritmos deben ser lo suficientemente difíciles de manipular y engañar para mantener la integridad y la confiabilidad de la red. Además, cada tipo de red necesitará condiciones más específicas, con algoritmos que acentúen por ejemplo la seguridad frente a la rapidez o,

viceversa. Conforme va pasando el tiempo, se van desarrollando nuevos algoritmos que permiten que las transacciones se confirmen más rápido y se necesite menos gasto computacional y de energía. Se pueden catalogar en los siguientes grupos principales: [2]

- **Proof-of-Work**

Es hasta ahora el algoritmo más utilizado. Consiste en una serie de problemas matemáticos con una complejidad muy avanzada, con lo cual se requiere un gran consumo de electricidad y una gran capacidad de cómputo. Esta es la manera de proteger al sistema de nodos que quieran burlar la red ya que deberían resolver una gran cantidad de operaciones que requieren un gasto computacional tan elevado que resulta muy complicado y desalentador.

El principal problema de este método es la cantidad de recursos desperdiciados y de lo perjudicial que resulta para el medio ambiente al necesitar cantidades innecesarias de energía y electricidad. [3]

- **Proof-of-Stake (PoS)**

En este tipo de algoritmo se usa la cantidad de recursos que se posea. De manera que un miembro de la red solo podrá participar en el porcentaje de recursos que tenga en su poder. Este algoritmo resulta más seguro según la teoría de juegos y, además, utiliza menos recursos que el algoritmo PoW debido a que los cálculos que se realizan son más fáciles y por lo tanto se necesitan unos dispositivos menos especializados. En este tipo de algoritmo los mineros apuestan una parte de los recursos que poseen para verificar las transacciones. El elegido para acuñar el valor del bloque suele ser elegido en función del valor apostado en la red en comparación con el valor total de la red. Este tipo de algoritmo disuade al mar comportamiento puesto que la verificación la realizarán los que tienen más valor, lo que significa que son los más interesados en que las verificaciones se hagan de manera correcta.

- **Delegated Proof-of-Stake (DPoS)**

En este caso, los participantes de la red votan a un conjunto de nodos que serán los validadores. Este tipo de algoritmo sigue siendo descentralizado para la votación de los nodos, pero se vuelve centralizado en cuanto a que es un pequeño conjunto del sistema el que toma las decisiones de manera que se verifican y validan las transacciones de una forma más rápida.

- **Proof-of-Importance (PoI)**

Este nuevo mecanismo otorga una peso o puntuación a los nodos basado en su participación en el sistema, en vez de asignarlo solo en función del valor de los recursos que posee. Este incentivo resulta muy útil para evitar que los miembros se dediquen a acumular recursos. En PoI el participante que más dinero mueve es el que más peso recibe dentro de la red. [4]

- **Byzantine Fault Tolerance (BFT)**

Este tipo de algoritmo tiene por finalidad eliminar los fallos bizantinos. Permite a los nodos validadores gestionar el estado de la cadena de bloques y a su vez intercambiar mensajes para acordar el histórico de transacciones correcto y de esa forma garantizar la honestidad. Las principales ventajas de este tipo de consenso son la escalabilidad y las comisiones muy bajas.

2.1.4 CONCEPTO DE MINADO

El minado es la forma que tiene la red de asegurar la red por medio de crear, verificar, publicar o propagar los bloques de transacciones. Minar significa verificar las transacciones de criptomonedas dentro una red blockchain. [5]

Los mineros recopilan todas las solicitudes de transacciones que se produzcan durante un cierto tiempo y deben comprobar si los datos encajan. Cuando les llega una solicitud comprueban que la cantidad se encuentre en la cuenta que lanza la solicitud y que no haya gastado ya la cantidad. Una vez verificados esos datos, el minero añadirá a su bloque de transacciones esa solicitud que ha sido considerada como válida.

A través de una serie de cálculos muy complejos que requieren mucha capacidad de cómputo, los mineros encuentran un nuevo bloque y lo deben agregar a la cadena. Los nodos encargados de minar deben competir entre ellos para para dar solución a ese problema matemático. Una vez se encuentre la solución, el bloque se incorpora a la cadena cuando los nodos (la mayoría de ellos) lleguen al consenso de que las transacciones realizadas por el minero que ha llegado a la solución sean correctas y que éste ha adivinado correctamente en '*nonce'*. El *nonce* corresponde a un número especial que es la 'clave' para resolver el problema planteado a los mineros. Una vez llegados los nodos al consenso, el nodo que ha realizado este trabajo recibe una recompensa en forma de criptomoneda que varía en función de la blockchain que se esté utilizando, por ejemplo, Bitcoin recompensará de una forma distinta a Ethereum.

El minado favorece a la red puesto que crea una competición que hace muy difícil la inserción de nuevos bloques de transacciones en la red. Se asegura que, si se quiere revertir alguna transacción o modificarla, sea necesario cambiar la cadena entera, lo que previene el fraude. Además, el consenso de los nodos protege la neutralidad al no poder un individuo tomar ninguna decisión sin el consentimiento de la mayoría de los nodos.

2.1.5 BITCOIN. LA PRIMERA BLOCKCHAIN

Bitcoin fue creado por Satoshi Nakamoto en 2008. Fue la primera aplicación desarrollada sobre blockchain y fue diseñada como un sistema de pago descentralizado, seguro y rápido. Posibilita el intercambio de criptomonedas, en este caso bitcoins, un tipo de moneda virtual que funciona como moneda global y accesible por todo el mundo, compitiendo contra el dinero fiat o el oro. Sin embargo, Bitcoin tiene una cantidad finita de monedas con una cantidad de 21 millones, lo que hace que sea deflacionaria y a la larga perderá su valor .

La plataforma utiliza un algoritmo criptográfico de seguridad conocido como Sha256. Este algoritmo favorece la minería con un tipo de chips conocidos como ASIC. El número 256 hace referencia al valor de los bits del 'hash' que corresponde a 32 bytes.

El lenguaje de programación es C++ que dispone de 70 comandos lo que hace que sea un lenguaje simple y limitado, pero a la vez más seguro y más controlado. [6]

2.2 ETHEREUM

Ethereum es una plataforma blockchain que permite correr programas en un ambiente de confianza. Diseñada para ser flexible y adaptable, Ethereum se compone de una máquina virtual llamada EMV (Ethereum Virtual Machine) y esta permite verificar y ejecutar código[7]. Basada en la misma tecnología que Bitcoin son bastantes las características que difieren de ésta.

2.2.1 BITCOIN VS ETHEREUM

Aunque se puedan llegar a confundir estas dos plataformas Ethereum y Bitcoin puesto que tienen la misma base tecnología que las sustenta, blockchain, ambas plataformas responden a funciones diferentes y fueron creadas con distintos propósitos. Para empezar, las separan unos años de diferencia puesto que Bitcoin fue creada en 2008, mientras que Ethereum nació en 2013. Bitcoin nació como una plataforma de pago descentralizado mientras que Ethereum buscaba ser una plataforma donde ejecutar código, contratos inteligentes. Al ser más moderna Ethereum usa un algoritmo criptográfico llamado Ethash mientras que Bitcoin usa Sha256. [6]

Por supuesto cada una de estas plataformas tiene su propia criptomoneda, Bitcoin posee una criptodivisa de igual nombre, bitcoin, y Ethereum a su vez genera ether, cada una de estas monedas tiene sus propios decimales. Sin embargo, la cantidad de Bitcoin es finita y corresponde a 21 millones, lo cual la convierte en una moneda deflacionaria que perderá valor, frente a los 18 millones que genera Ethereum cada año para operar en la red.

	Bitcoin	Ethereum
Nacimiento	Agosto de 2008	Diciembre 2013
Función principal	Sistema de pago descentralizado	Plataforma para ejecutar smart contracts y dapps
Tecnología usada	Blockchain	Blockchain
Algoritmo criptográfico	Sha256	Ethash
Criptomoneda	Bitcoin (BTC)	Ether (ETH)
Creación de criptomoneda	21 millones en total	18 millones por año
Sistema de minado	Proof of Work (PoW)	Proof of Work (PoW)
Tiempo de procesamiento	Cada 10 minutos	Cada 16 segundos
Calculo de la dificultad de minado	Cada 2016 bloques minados	Cada bloque minado
Coste de las transacciones	Todas por igual	Depende del gas

Tabla 1. Comparativa Bitcoin VS Ethereum

Más diferencias que podemos apreciar es la velocidad de minado. Mientras Bitcoin tarda 10 minutos en procesar una transacción, Ethereum lo consigue en un tiempo aproximado de 16 segundos, calculando la dificultad del minado en cada bloque, de manera que se va ajustando a las necesidades y requerimientos de la red, para alcanzar siempre esas velocidades.

Sin embargo y de momento los algoritmos de consenso son del mismo tipo, puesto que utilizan la capacidad de cómputo para llegar al acuerdo de la mayoría de los nodos. Esto cambiará en unos meses con la bifurcación de Ethereum para implementar el algoritmo Casper.

Por último y una de las grandes diferencias con Ethereum es el coste por transacción. Mientras que Bitcoin cobra todas las transacciones por igual, Ethereum al ser un lenguaje de Turing completo y usar cálculos más complejos, modifica el coste de la transacción según el gasto de computación, buscando recompensar a los mineros por gastar más recursos y 'penalizar' en cierto modo por consumir más recursos de la red.

En definitiva, Ethereum busca ser una plataforma donde se desplieguen aplicaciones, que tenga la capacidad suficiente para las necesidades de sus usuarios y que proporcione una rapidez mayor en las transacciones.

2.2.2 ETHEREUM VIRTUAL MACHINE

Ethereum es una plataforma programable. A diferencia de Bitcoin que establece las operaciones que se pueden realizar de forma predeterminada, Ethereum permite al usuario y desarrollar sus propias operaciones de cualquier complejidad. Esto hace que las posibilidades se amplíen de una forma muy amplia y no solo haga referencia a criptomonedas. [7]

De una manera general podríamos definir Ethereum como los protocolos necesarios para que corran aplicaciones descentralizadas. En el núcleo se encuentra la Ethereum Virtual Machine (EMV) que permite ejecutar código sin importar la complejidad. Ethereum se considera como un sistema Turing Completo y se refiere a la capacidad que tiene el lenguaje de programación de Ethereum, Solidity, para aplicarse y poder resolver cualquier tipo de problema computacional y posibilitar el uso de estructuras complejas como los bucles.

Como cualquier blockchain, Ethereum incorpora un protocolo de red peer-to-peer. La base de datos de Ethereum es mantenida y actualizada por los nodos que están conectados en la red. Cada nodo corre la EMV y ejecuta las mismas instrucciones, de manera que se mantenga el consenso a través de la red. Este consenso descentralizado permite que sea muy tolerante respecto a los fallos, mantenga inmutable la cadena de bloques y asegure la disponibilidad de la red.

2.2.3 ALGORITMOS DE CONSENSO – ETHASH Y CASPER

Ethereum igual que Bitcoin ha estado utilizando un algoritmo de consenso PoW. Este algoritmo que usa Ethereum se llama Ethash y se basa en descubrir el 'nonce' para que el resultado esté por debajo de un determinado umbral de dificultad que marca la red.

Sin embargo, al consumir demasiados recursos no resulta un algoritmo eficaz y se vio la necesidad de encontrar un algoritmo basado en PoS . De ahí surgió Casper, que implementa una lógica más compleja y analiza los nodos de manera que permite identificar cuál es la cadena más larga y beneficiosa para la red. Sus dos características principales son la estabilidad y la información inequívoca. Esto impide que se genere información contradictoria y garantiza la capacidad de generar un nuevo bloque si al menos dos terceras partes de los validadores sigan el protocolo. Esto generará una mayor eficiencia puesto que el tiempo de generación de los bloques se reducirá de manera significativa al no tener que comprobar quién posee mayor potencia de cálculo si no simplemente quién posee más recursos. [8]

Además, el algoritmo incorporará un sistema de sanciones, en el que el importe de las penalizaciones dependerá de la gravedad de los fallos y la frecuencia de ausencia en la red. Casper aún está en fase de pruebas, y se espera su implementación en otoño de 2018 y la actualización se implementará por medio de una bifurcación o fork puesto que será incompatible con versiones anteriores de Ethereum.

2.2.4 CLIENTES ETHEREUM

Desde el principio del proyecto Ethereum se han desarrollado numerosos clientes para esta plataforma en diferentes lenguajes de programación y para los diferentes sistemas operativos. Esto supone una variedad muy amplia que no hace más que probar que el protocolo funciona en cualquier ambiente y supone una puerta abierta a la innovación y a nuevas mejoras en un futuro cercano. [9]

Los más conocidos actualmente son los siguientes:

Cliente	Lenguaje	Desarrollador
Go Ethereum (Geth)	Go	Ethereum Foundation
Parity	Rust	Ethcore
Cpp-ethereum	C++	Ethereum Foundation
Pyethapp	Phython	Ethereum Foundation
Ethereumjs-lib	Javascript	Ethereum Foundation
Ethereum(J)	Java	<Ether.camp>
Ruby-Ethereum	Ruby	Jan Xie

Tabla 2. Clientes de Ethereum más conocidos (Ethereum Homestead)

2.2.5 CONECTAR CLIENTES ETHEREUM

Los clientes Ethereum para conectarse a través de aplicaciones cuentan con varios métodos JSON-RPC. Sin embargo, interactuar directamente con estos métodos conllevan mucha carga de información para los desarrolladores. Es por eso por lo que se han desarrollado librerías que permiten una cierta abstracción de la conexión entre las aplicaciones y

Ethereum. En el caso de este proyecto vamos a utilizar la librería Web3.js utilizada en Javascript que nos va a servir para enlazar nuestro cliente con nuestro caso de uso.

2.2.6 ETHER

Ether es el nombre de la criptomoneda asociada a Ethereum. El ether es usado para pagar los costes de computación de la máquina virtual de Ethereum.

Etheruem emplea su propio sistema de unidades para el ether. Cada denominación tiene un nombre único siendo el Wei la unidad más pequeña de medida. La tabla de medidas es la siguiente: [10]

Unit	Wei Value	Wei
wei	1 wei	1
Kwei (Babbage)	1e3 wei	1.000
Mwei (Lovelace)	1e6 wei	1.000.000
Gwei (Shannon)	1e9 wei	1.000.000.000
microether (Szabo)	1e12 wei	1.000.000.000.000
milliether (Finney)	1e15 wei	1.000.000.000.000.000
ether	1e18 wei	1.000.000.000.000.000.000

Tabla 3. Unidades de medida del ether (Ethereum Homestead)

Hay varias maneras de conseguir Ether: convertirse en minero, intercambiando otras monedas por ether o simplemente comprándolo.

2.2.7 GAS

El concepto de gas se refiere al precio que se paga por realizar ciertas operaciones dentro del red, es la unidad de medida para medir el trabajo que se realiza en Ethereum. Cada operación realizada en la red tendrá un coste asociado según la complejidad y el gasto computacional.

El gas tiene varias funcionalidades asociadas:[11]

- Asignar un coste a la ejecución de las operaciones. Cada operación exige una capacidad de cómputo diferente, de manera que según la complejidad se asignará un coste u otro dependiendo de los recursos que consuma.
- Asegurar el sistema. En el momento que se exige un gasto al realizar cualquier operación fuerzas a los participantes a realizar solo las operaciones imprescindibles. Esto fuerza a que la cadena de bloques sea más ligera sin información inútil, evitamos el spam y el uso de bucles infinitos que podrían hacer inservible el sistema.
- Recompensar a los mineros por el gasto computacional. Este gasto que aceptamos al operar en la red servirá para recompensar a los mineros que han puesto sus recursos a disposición de la red para validar las transacciones. Por lo que resulta un sistema de recompensa e incentivo para mantener la estabilidad de la red.

El gas no es una criptomoneda debido a que en general son muy volátiles y no tienen un valor fijo, va cambiando constantemente. El gas debería tener un precio constante para usar los recursos de la red, puesto que se debería conocer cuánto nos va a costar realizar transacciones o cómputos en nuestra red. El gas en sí mismo no tiene un valor, es un parámetro independiente al ether y el principio sobre el que se basa es que debe ser estable para evitar que, si el precio del ether subiera demasiado, el precio de las transacciones se disparara, o viceversa. Si el precio del ether subiera, el coste del gas sería menor para mantener el mismo coste de operaciones. [10]La solución consiste en implementar un precio del gas que se ajuste en función de unos parámetros de la red pero que sea independiente para evitar otro mercado paralelo y la especulación.

Relacionamos muchos términos con el gas. Dentro de una transacción Ethereum existen varios parámetros que dependen del gas como: límite de gas, gas usado por la transacción, precio del gas, comisión y gas usado acumulado.

- Precio del gas: corresponde a un valor constante de Ether.
- Límite de gas: corresponde a la cantidad máxima de gasque una transacción puede consumir para llegar a ser validada.
- Gas usado; corresponde al gas que realmente se ha utilizado por cada transacción que se lleva a cabo en la red. Si el gasto de computación ha sido mayor que el límite de gas, la transacción no se podrá realizar y aparecerá como fallida.
- Gas acumulado: este dato permite hacer una estimación del gasto de gas que se ha usado por bloque.
- Comisión: se refiere al sobrecargo que se debe pagar por procesar una transacción. La fórmula que sigue la transacción sería aproximadamente la siguiente:
  *Coste Actual de la Transacción – Comisión = Gas usado * precio del Gas*

El funcionamiento del gas dentro de un bloque es el siguiente. Un bloque está formado por muchas transacciones. Dentro de este bloque se repiten los campos de gas usado, que se refiere al gas usado en total por todas las transacciones, y límite de gas, cantidad de gas que tiene el bloque para ser validado. Este último concepto determina un espacio máximo en el bloque para las transacciones. Los mineros deben decidir como rellenar los bloques con las transacciones sin sobrepasar el límite de gas. Normalmente se busca la combinación que genere una mayor rentabilidad en base a las comisiones que tenga cada transacción.

2.3 SMART CONTRACTS

Un Smart Contract o contrato inteligente es un código que ejecuta reglas y acuerdos establecidos entre varias partes de modo que cuando se cumplan ciertas condiciones específicas sucedan ciertas acciones. Cuando se da una condición programada con anterioridad se ejecuta la cláusula correspondiente de forma automática. Estos contratos inteligentes se dan en un ambiente no controlado por ninguna de las partes implicadas en el contrato puesto que corren en un ambiente descentralizado.

A diferencia de los contratos convencionales que están en escritos en lenguaje natural, en lenguaje legal, los contratos inteligentes están escritos en un lenguaje de programación de manera que elimina las posibles interpretaciones que cada participante del acuerdo podría hacer. Las condiciones del contrato se ejecutan de forma automática cuando se cumplen los requisitos y requerimientos.

Los smart contracts presentan una serie de beneficios frente a los contratos actuales. Autonomía. Este tipo de contratos se dan entre varias entidades, sin ningún intermediario de por medio. No es necesario que por ejemplo un abogado valide el acuerdo. De esta manera se reducen de manera considerable el número de agentes externos que estén implicados en el contrato. Costes. Los costes son más reducidos puesto que requieren una menor intervención humana e incluso se pueden estandarizar. Confianza. Al utilizar la tecnología blockchain todos estos contratos están replicados en cada nodo y van directos a la cadena de bloques. Esto significa que los contratos están encriptados de modo que solo las personas que formen parte del acuerdo puedan leerlo y además permite la interacción entre los nodos que no se conocen entre sí eliminando el problema de la estafa. Velocidad. Se aumenta la velocidad en los procesos de negocio. Los smart contracts no dejan ser código software que automatiza tareas que de manera contraria se realizarían de una forma manual que genera más errores. Seguridad. Estos contratos se basan en la cadena de bloques, de manera que no pueden perderse y no pueden modificarse sin alterar toda la cadena, todo queda registrado de manera inmutable, accesible y consensuada por los nodos participantes de la red. Nuevos

modelos de negocio. Los contratos inteligentes, permiten nuevos tipos de negocios que abren nuevas vías de innovación y emprendimiento.

2.3.1 COMUNICACIÓN ENTRE ETHEREUM Y SMART CONTRACTS

Un nodo Ethereum ofrece una interface RPC. Esta interface permite el acceso de las aplicaciones a la blockchain de Ethereum y la funcionalidad que el propio nodo provee, como, por ejemplo, compilar el contrato. Usa un subconjunto de la especificación JSON RPC 2.0 como el protocolo serializable y se encuentra disponible sobre HTTP y IPC.

Esta interface usa una serie de convenciones que conviene conocer:

- Los números están codificados en hexadecimal. Esto es debido a que algunos lenguajes de programación no soportan trabajar con números muy grande y es una manera de evitar errores.
- Número de bloque por defecto.

2.3.2 GAS EN UN SMART CONTRACT

Los contratos inteligentes cuando se compilan se transforman en una serie de códigos de operación 0 también llamados '*opcodes'*. Estos códigos se muestran bajo nombres nemotécnicos en inglés que nos permiten entenderlos de una forma fácil. En el Yellow Paper de Ethereum podemos encontrar los *opcodes*. [11]

Name	Value	Description*
G_{zero}	0	Nothing paid for operations of the set W_{zero}.
G_{base}	2	Amount of gas to pay for operations of the set W_{base}.
$G_{verylow}$	3	Amount of gas to pay for operations of the set $W_{verylow}$.
G_{low}	5	Amount of gas to pay for operations of the set W_{low}.
G_{mid}	8	Amount of gas to pay for operations of the set W_{mid}.
G_{high}	10	Amount of gas to pay for operations of the set W_{high}.
$G_{extcode}$	700	Amount of gas to pay for operations of the set $W_{extcode}$.
$G_{balance}$	400	Amount of gas to pay for a BALANCE operation.
G_{sload}	200	Paid for a SLOAD operation.
$G_{jumpdest}$	1	Paid for a JUMPDEST operation.
G_{sset}	20000	Paid for an SSTORE operation when the storage value is set to non-zero from zero.
G_{sreset}	5000	Paid for an SSTORE operation when the storage value's zeroness remains unchanged or is set to zero.
R_{sclear}	15000	Refund given (added into refund counter) when the storage value is set to zero from non-zero.
$R_{selfdestruct}$	24000	Refund given (added into refund counter) for self-destructing an account.
$G_{selfdestruct}$	5000	Amount of gas to pay for a SELFDESTRUCT operation.
G_{create}	32000	Paid for a CREATE operation.
$G_{codedeposit}$	200	Paid per byte for a CREATE operation to succeed in placing code into state.
G_{call}	700	Paid for a CALL operation.
$G_{callvalue}$	9000	Paid for a non-zero value transfer as part of the CALL operation.
$G_{callstipend}$	2300	A stipend for the called contract subtracted from $G_{callvalue}$ for a non-zero value transfer.
$G_{newaccount}$	25000	Paid for a CALL or SELFDESTRUCT operation which creates an account.
G_{exp}	10	Partial payment for an EXP operation.
$G_{expbyte}$	50	Partial payment when multiplied by $\lceil \log_{256}(exponent) \rceil$ for the EXP operation.
G_{memory}	3	Paid for every additional word when expanding memory.
$G_{txcreate}$	32000	Paid by all contract-creating transactions after the *Homestead* transition.
$G_{txdatazero}$	4	Paid for every zero byte of data or code for a transaction.
$G_{txdatanonzero}$	68	Paid for every non-zero byte of data or code for a transaction.
$G_{transaction}$	21000	Paid for every transaction.
G_{log}	375	Partial payment for a LOG operation.
$G_{logdata}$	8	Paid for each byte in a LOG operation's data.
$G_{logtopic}$	375	Paid for each topic of a LOG operation.
G_{sha3}	30	Paid for each SHA3 operation.
$G_{sha3word}$	6	Paid for each word (rounded up) for input data to a SHA3 operation.
G_{copy}	3	Partial payment for *COPY operations, multiplied by words copied, rounded up.
$G_{blockhash}$	20	Payment for BLOCKHASH operation.
$G_{quaddivisor}$	100	The quadratic coefficient of the input sizes of the exponation-over-modulo precompiled contract.

Figura 2. Códigos de operación de Ethereum [Ethereum Yellow Paper]

Se puede observar en la imagen superior que además del nombre de la operación encontramos una columna con una descripción de cada operación y una columna de valor. Este valor representa el consumo en unidades de gas para cada '*opcode'*. Cada vez que un contrato usa una de estas operaciones se le asigna un coste. Según se va actualizando la plataforma de Ethereum estos valores se pueden ir modificando, de manera que es necesario comprobar de forma periódica que los valores no hayan cambiado.

2.4 QUORUM

Quorum es un protocolo de libro mayor distribuido orientado a entornos empresariales y basado en Ethereum. Ha sido desarrollado para proveer un sistema de permisionado para las redes de nodos y soporta transacciones y contratos privados. Esto permite que una red sea privada lo que implica que necesite algún tipo de permiso para leer la información de la cadena de bloques, limite el número de nodos que pueden participar en las transacciones o quién puede escribir en la cadena. [12]

Quorum es una versión de Ethereum a la que se le han añadido nuevas funcionalidades. Específicamente, Quorum te permite crear redes privadas entre participantes seleccionados y, además, añadir una capa de privacidad adicional por encima de las transacciones normales de Ethereum.

Las características principales de Quorum son:

- La posibilidad de realizar transacciones y contratos privados
- La posibilidad de elegir tu propio mecanismo de consenso
- Permisos de gestión para la red y los nodos
- Un alto rendimiento y desempeño.

2.4.1 COMPONENTES

Actualmente, Quorum incorpora los siguientes componentes:[12]

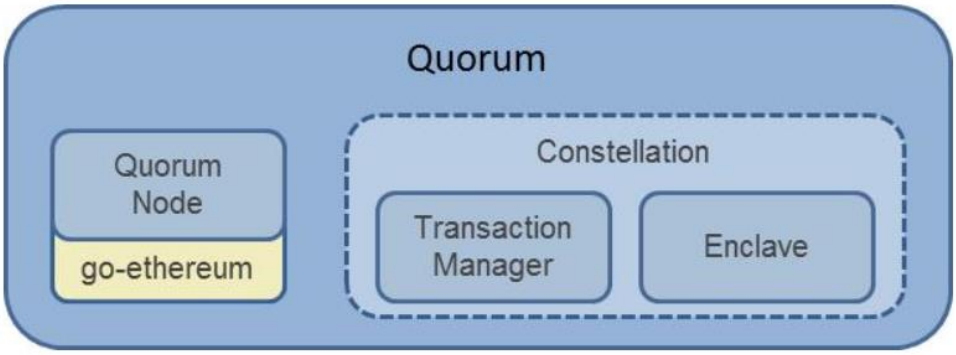


Figura 3. Arquitectura Quorum

QUORUM NODE

Quorum Node es una bifurcación del cliente Geth de Ethereum, de manera que se pueda aprovechar todos los avances e investigación que se realiza en la comunidad Ethereum. De esa manera Quorum se va actualizando de una manera continua con las diferentes versiones de Geth. El nodo Quorum incluye una serie de modificaciones respecto a su homólogo en Ethereum. Alguno de ellos sería:

- El algoritmo de consenso que se usa en Ethereum es uno basado en Proof-of-Work aunque a partir de este año, se actualizará al algoritmo Casper que se basa en Proof-of-Stake para realizar transacciones de una forma mucho más eficiente. Sin embargo, Quorum permite elegir su algoritmo de consenso y encontramos dos diferentes: Raft e Istanbul-BFT.
- La capa peer-to-peer ha sido modificada para poder admitir transacciones desde nodos y hacia nodos que hayan obtenido permiso para acceder a la red.
- Se ha modificado la manera de validación de los bloques para permitir las transacciones privadas.

- Se ha eliminado el precio del gas, aunque se mantiene el concepto de gas para evitar el spam y los códigos ineficientes que saturen la red. Quorum fuerza a la red al que el precio del Gas sea 0.

CONSTELLATION

Constellation es un sistema para enviar información de manera segura. No es un sistema específico para Blockchain y es aplicable a multitud de casos de uso. El módulo de Constellation se compone de dos partes: el nodo Constellation Transaction Manager y Enclave.

- Transaction Manager

Es responsable de las transacciones privadas. Se encarga de almacenar y permitir el acceso a información sensible y encriptada.SE encarga de intercambiar la información ecnriptada con los otros Transaction Managers participantes de a transacción privada. Sin embargo, no tiene acceso a ninguna clave privada, esa función la delega en Enclave

- Enclave

Para mejorar el rendimiento y especialmente la seguridad, Enclave se encarga de la mayor parte de los procesos criptográficos. Trabaja estrechamente relacionado con el Transaction Manager y se encarga por ejemplo de la generación de claves simétricas. Es el bloque encargado de reforzar la seguridad realizando la encriptación/desencriptación de una manera aislada del resto del proceso.

2.4.2 ALGORITMOS DE CONSENSO

Actualmente para la versión 2 de Quorum, se pueden elegir dos tipos diferentes de algoritmos de consenso, en función de los resultados y la distribución de nuestra red. Estos dos algoritmos son: Raft e Istanbul-BFT. Estos dos tipos diferentes de algoritmos buscan ser alternativas al método de consenso Proof-of-Work que utiliza Ethereum.

RAFT

Este algoritmo de consenso es útil para aquellas redes en las que la tolerancia a fallos bizantinos sea necesaria, si no que se prima la velocidad a la hora de confirmar transacciones con velocidades de milisegundos. Raft y Ethereum conciben el nodo de manera diferente. Raft considera dos tipos de nodo: un líder único (*leader*) para todo el clúster y seguidores (*follower*). Durante la elección del líder existirá otro tipo llamado candidato. Sin embargo, en Ethereum los nodos no se distinguen, cualquier nodo puede minar un nuevo bloque a la red, lo que equivaldría a ser el líder en esa ronda (*minter*) y el resto de los nodos verifican la transacción (*verifiers*) [13]

En este algoritmo cada nodo tiene una correspondencia entre Raft y Ethereum. Cada nodo Ethereum será también un nodo Raft y por convención, el líder elegido por Raft es el único nodo Ethereum que podrá minar o acuñar un nuevo bloque. De esta manera quedaría una tabla de relaciones como esta:

Ethereum	**Raft**
Minter ⟷	Leader
Verifier ⟷	Follower

Tabla 4. Relación entre Ethereum y Raft

De esta manera se asegura que solo hay un nodo minando el bloque en cada ronda. El nodo líder podrá perder su estatus y en ese caso dejará de minar. Hay un breve período en el que varios nodos pueden ejercer de líderes durante el cambio de liderazgo, y esto se resuelve a través de una carrera, el nodo que consiga extender el bloque antes a la cadena será el que se considere como nuevo líder, el otro bloque será rechazado.

En este método las transacciones se comunican mediante la capa de transporte de la red P2P [Redes Peer-To-Peer] de Ethereum, pero los bloques se comunican a través de la capa de transporte de Raft. Estos bloques los crea únicamente el líder y lo envía al resto de nodos seguidores por este protocolo de transporte. Desde el punto de vista de Ethereum, Raft se implementa a través de la interface *Service*.

El ciclo típico de una transacción sería el siguiente:

1. La transacción se envía por medio de una llamada RPC a Geth. A través de la capa de transporte de Ethereum P2P, la transacción es enviada a todos los peers.
2. La transacción llegará al nodo que ha sido elegido acuñador o líder. El bloque se minará y se disparará un evento que anuncie al protocolo Raft que hay un nuevo bloque y este será propuesto.
3. Cuando ya ha sido enviado a través de la capa HTTP del protocolo Raft se convertirá en el principio de la cadena de bloques en todos los nodos.
4. En este punto, Raft ha llegado un consenso y la transmite al resto de nods
5. La transacción ha llegado a todos los nodos. Raft garantiza un solo pedido de entradas almacenadas en su registro, y todo lo que se ha comprometido está garantizado por lo que no se producirá ninguna bifurcación en la cadena.

Por defecto los bloques se minan cada 50ms. Cuando llega una nueva transacción se mina un nuevo bloque inmediatamente si han pasado más de 50 ms desde el último bloque para no saturar y conseguir equilibrio entre rendimiento y latencia, siendo configurable con el flag - - *raftblocktime* en geth.

ISTANBUL-BFT

Este algoritmo es una modificación del algoritmo original PBFT para poder trabajar con las cadenas de bloques. Istanbul BFT usa tres fases para el consenso: *pre-prepare, prepare y commit.* El sistema puede tolerar como mucho un número F de fallos en los nodos de manera que la red debería estar formada por N=3F+1. Antes de cada ronda los nodos validadores elegirán uno de ellos como '*proposer'*, por defecto, siguiendo el método *round-robin* por defecto. El nodo elegido hará una propuesta de bloque y mandará esta con el mensaje "*pre-prepare*". Después se mandará el mensaje "*prepare*". Este paso sirve para verificar que todos los nodos validadores están en funcionamiento y trabajando en la misma ronda. Si se reciben 2F +1 mensajes de preparado, las validadores entran en la fase de preparados y lanza a toda la red el mensaje de "*commit*". Este paso sirve para informar a los *peers* que se acepta el bloque propuesto por el '*proposer'* y que va a introducir el bloque en la cadena. Por último, los validadores esperan a que les lleguen 2F+1 de mensajes "*commit*" para entrar en el estado *'committed'* y añadir el bloque a la cadena. [14]

Los bloques en el protocolo Istanbul son finales, lo que significa que no hay desviaciones y todos los bloques válidos deben estar en la cadena principal. Para prevenir que un nodo fallido cree una cadena totalmente diferente, cada validador adjunta 2F+1 mensajes de "*commit*" firmados en el campo *extraData* en la cabecera antes de insertarlo el bloque en la cadena. Sin embargo, esto puede crear un problema en el cálculo del *hash*, ya que cada bloque puede incluir un número diferente de firmas. Es por eso por lo que el cálculo se realiza sin contar con los mensajes "*commit*". De ese modo se mantiene la consistencia del bloque y del hash a la vez que se puede incluir el consenso en la cabecera.

La forma de elegir al nodo '*proposer'* puede variar según dos criterios:

- *Round robin:* el nodo propuesto va cambiando en cada bloque y cambio de ronda.
- *Sticky proposer*: el nodo irá cambiando solo cuando se produce un cambio de ronda.

Un cambio de ronda se puede dar debido a tres situaciones, si el contador de cambio de ronda expira, si se produce un mensaje no válido de *"pre-prepare"* o una inserción de bloque fallida. Si un nodo validador detecta alguna de estas situaciones entonces lanzará un mensaje a toda la red pidiendo un cambio de ronda incluyendo el número de ronda que propone. El número de ronda es elegido según dos condiciones diferentes: si recibe mensajes de cambio de ronda elige el número mayor que haya tenido F+1 mensajes, o escoge el 1+ nº actual de ronda.

Este algoritmo funciona como una máquina de estados A continuación se puede apreciar los estados y las variables que son responsables de los cambios de un estado a otro. [14]

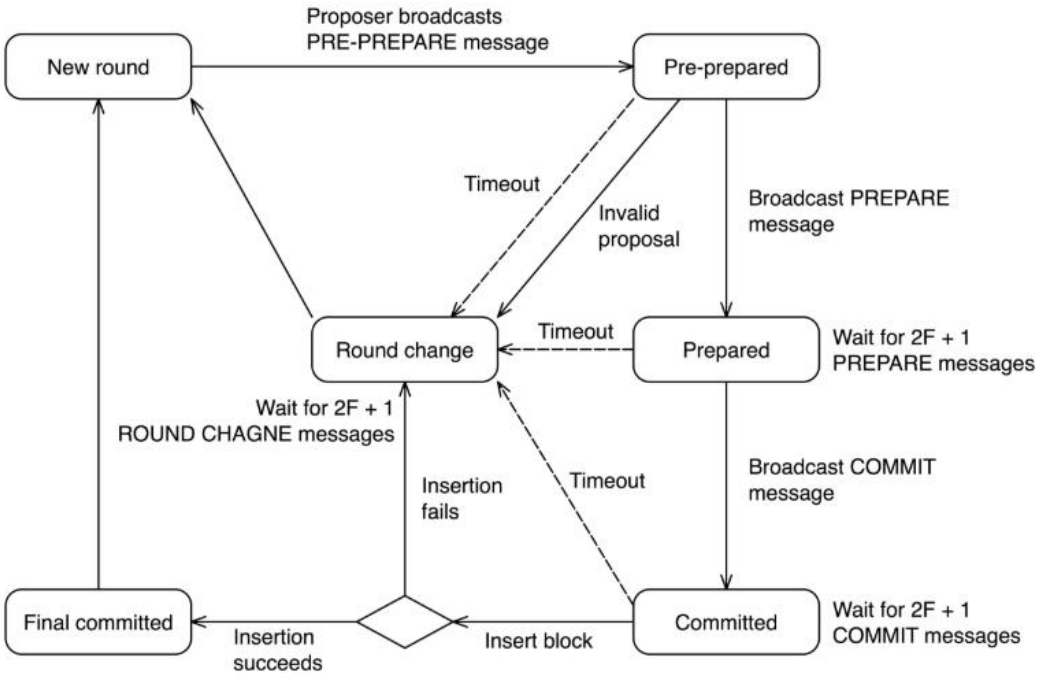


Figura 4. Máquina de estados. Algoritmo Instanbul BFT (JP Morgan GitHub Instanbul BFT)

2.4.3 TRANSACCIONES

En Quorum se introduce el concepto de transacciones privadas. Esto quiere decir que se añade un nuevo parámetro opcional '*privateFor'* en las transacciones que hace que se traten de forma privada.

Las llamadas transacciones públicas, son transacciones creadas de la manera estándar que se usa en Ethereum. Estas transacciones serán visibles a todos los participantes de la red Quorum. Es necesario añadir que las transacciones que se consideran públicas en Quorum no son transacciones de la red pública Ethereum, son independientes.

Existe otro tipo de transacciones como se ha mencionado anteriormente que son llamadas transacciones privadas. La información que contienen solo será visible para los nodos cuyas claves estén especificadas en el parámetro '*privateFor'* de la transacción. En el momento en que un nodo Quorum encuentra una transacción con el valor del parámetro '*privateFor'* diferente a *null*, entonces modificará el valor de la firma de la transacción a '37' o '38', en contraposición a las transacciones públicas cuyos valores pueden ser '27' o '28'.

Las transacciones públicas se manejan de la misma forma que en Ethereum, de manera que, si una transacción es enviada a una cuenta que soporta un smart contract, entonces cada participante ejecutará el mismo código y sus bases de datos subyacentes será actualizadas.

Sin embargo, una transacción privada no cumple con el estándar. Antes de que se propague la transacción al resto de nodos, se reemplaza la carga de la transacción original, o sea, su información con un hash del contenido encriptado, que proporciona el módulo de Constellation. Aquellos nodos que sean parte de la transacción serán capaces de recuperar la información original a través de Constellation, mientras que los nodos no participantes solo podrán visualizar el hash. Esto resulta en que, si la transacción se envía a una cuenta que contenta código, esos participantes de la red que no sean parte de la transacción simplemente no correrán el código, lo obviarán. Los participantes de la transacción reemplazarán el hash con la información original antes de llamar a la máquina virtual de Ethereum (EMV) para la ejecución de su código. En este momento se podría pensar que esto provocaría un error en

la cadena, puesto que algunos nodos tendrían una información diferente y no serían capaces de llegar a un consenso. Para evitar este problema Quorum guarda el estado de los contratos en un Public State Trie que está sincronizado de manera global, y un Private State Trie que no se sincroniza de forma global. [15]

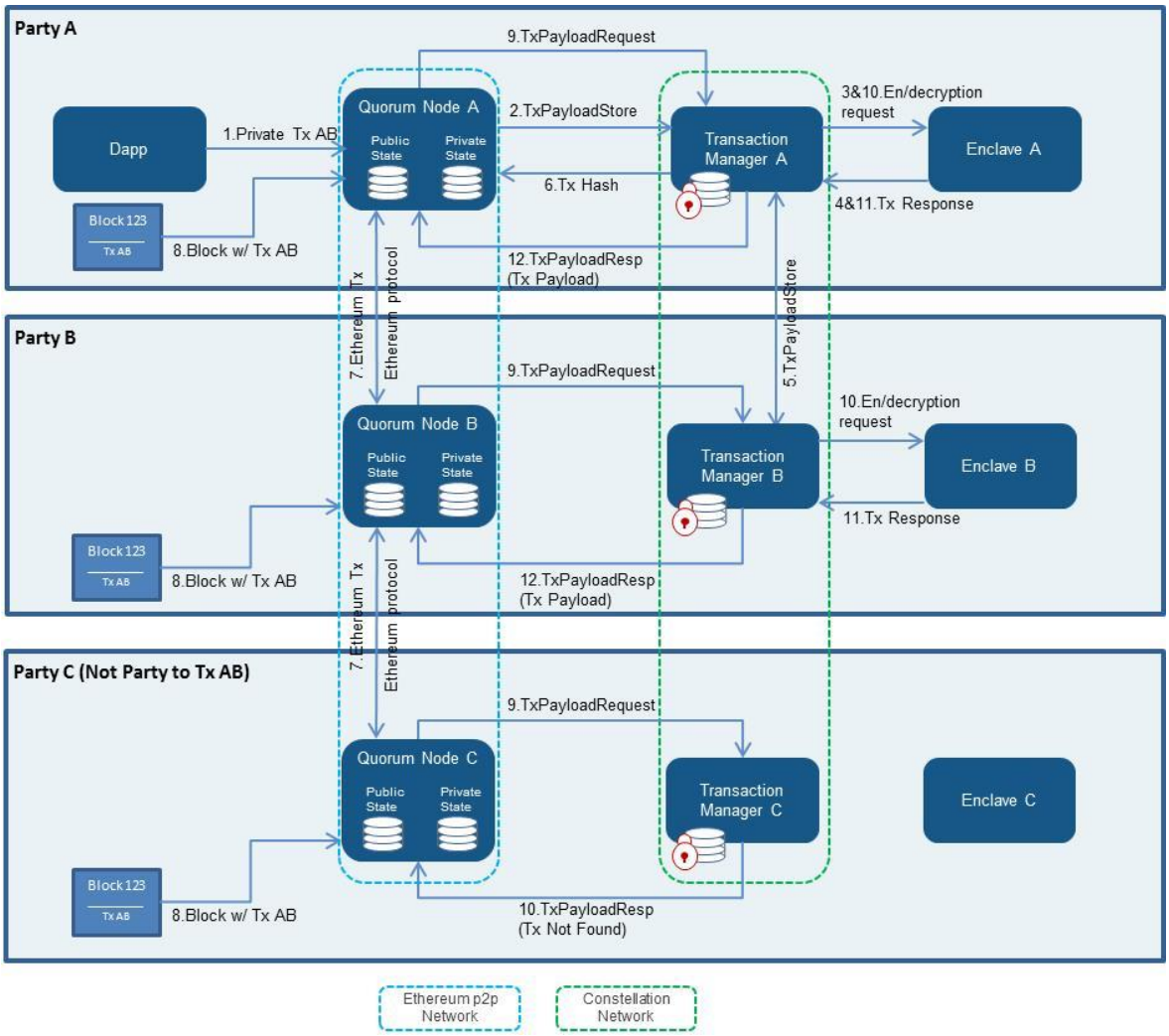


Figura 5. Proceso de una transacción privada (JPMorgan)

En la imagen superior podemos ver el flujo que sigue una transacción privada desde el nodo A al nodo B, participantes de la transacción privada, mientras que el nodo C estaría excluido de esta transacción.

1. El nodo A manda una transacción a su nodo Quorum, especificando que la información que lleva y estableciendo el parámetro *'privateFor'* con la clave pública de A y B.
2. El nodo Quorum de A pasa la transacción al Transaction Manager de Constellation, solicitando que se almacene la carga de la transacción.
3. El módulo Transaction Manager del nodo A llama a su Enclave asociado para validar el emisor y encriptar la información.
4. El Enclave de A revisa la clave privada del mismo y una vez que la ha validado procede a convertir la transacción. Esto conlleva que se genere una clave simétrica y un *nomce* aleatorio, que se encripte el *payload* de la transacción y el *nomce* con la clave simétrica. Calcula el *hash* con SHA3-512 de la información encriptada y devuelve toda esta información (el *payload* cifrado, el hash y la clave simétrica cifrada con cada una de las claves públicas de los participantes) al Transaction Manager.
5. El Transaction Manager del participante A almacena la información encriptada y la clave simétrica también encriptada usando el hash como índice, y de esta manera utilizando HTTPs transfiere de manera segura toda esta información al participante B. El receptor de la transacción devuelve un *Ack/Nack* confirmando o denegando la recepción. Si no se recibe un mensaje de confirmación, entonces no se propaga por la red la transacción.
6. Una vez la información transmitida al Transaction Manager de B se ha completado con éxito, el Transaction Manager del usuario A devuelve el *hash* al Quorum Node que reemplaza la transacción original con ese *hash* y cambia el valor de la transacción a 37 o 38 o que indica a otros nodos que ese hash representa a una transacción privada.

7. La transacción entonces es propagada al resto de la red usando el protocolo estándar Ethereum P2P
8. Se crea un bloque que contenga la transacción AB y es distribuido a cada participante de la red.
9. Al procesar el bloque, todos los participantes intentarán procesar la transacción. Cada nodo reconocerá el valor 37 o 38 identificando esa transacción como una que requiere que se desencripte su *payload* y harán una llamada a sus Transaction Managers correspondientes para determinar, a través del hash como índice, si la deben retener.
10. Como C no es un participante de esa transacción recibirá un mensaje *'NotARecipient'* y saltará esa transacción, de manera que no actualizará su base de datos *'Private StateDB'*. En cambio, A y B mirarán en su Transaction Manager a través del hash que funciona como un índice e identificarán que deben retener esa transacción. Cada uno de ellos llamará a Enclave y le pasarán toda la información que se ha estado almacenando, el *payload* encriptado, la clave cifrada y la firma.
11. Enclave valida la firma y desencripta la clave simétrica usando la clave privada que se mantiene en Enclave y desencripta el *payload* de la transacción usando la clave simétrica que ha sido revelada. Acto seguido lo devuelve al Transaction Manager.
12. Los Transaction Manager de A y B envían el *payload* a la EVM para ejecutar el código del contrato. Esta ejecución actualizará el estado del *'Private StateDB'*. Una vez que el código se ha ejecutado, se descarta para que nunca sea posible leer la información sin pasar por el proceso previo.

    De manera general podemos ver el conjunto de la arquitectura de Quorum como se muestra en la siguiente imagen

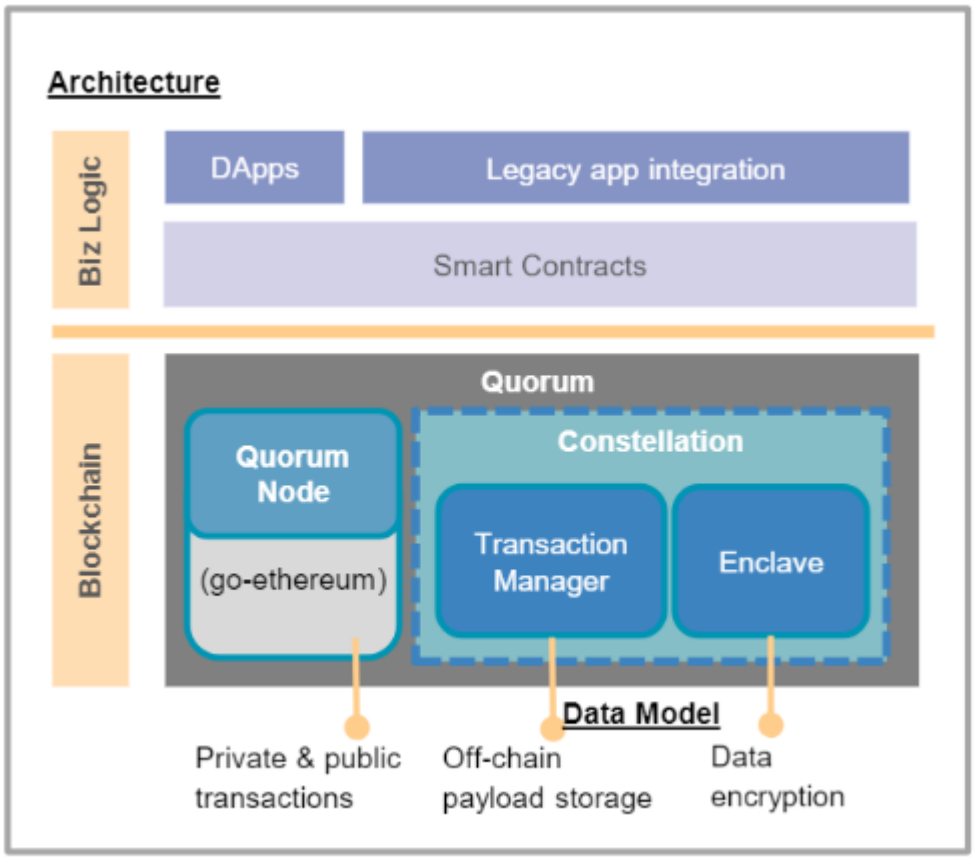


Figura 6. Arquitectura Quorum. (Quorum-docs GitHub)

En resumen, podemos comprobar como Quorum funciona como una red blockchain basada en Ethereum, pero con un añadido extra, la posibilidad de enviar transacciones privadas entre nodos participantes sin que el resto de la red sea partícipe de la operación, lo que podría tener multitud de aplicaciones, especialmente financieras.

2.5 OTRAS TECNOLOGÍAS UTILIZADAS

2.5.1 INFURA

Infura es un cluster de Ethereum que permite a los usuarios correr sus aplicaciones sin tener que instalar previamente su propio nodo Ethereum o *wallet*. Infura, por motivos de seguridad no maneja las llaves privadas del cliente, lo que significa que no puede firmar transacciones en tu nombre.

2.5.2 OPPENZEPPELIN

Los smart contracts que se despliegan en la red principal de Ethereum pueden manejar dinero por lo que es esencial que nuestro código esté libre de errores. Zeppelin Solutions es un servicio de auditoría, que cuenta con una serie de contratos que han sido probados y que se consideran seguros, esto es OppenZeppelin. Esta herramienta se usa para crear contratos seguros en menos tiempo y tener la certeza de que han sido probados y auditados, de manera que nuestro código funcione sin problemas.

2.5.3 GANACHE

Ganache lanza de una manera muy rápida una bockchain de Ethereum que nos permite correr pruebas, ejecutar comandos e inspeccionar el estado de la red mientras se controla como está operando la cadena. No es necesario instalar ningún cliente de Ethereum. Esta aplicación incorpora varias características como poder controlar el estado de todas las cuentas de la red incluidas las direcciones, las claves privadas, las transacciones o los balances. Además, permite configurar como se realiza el minado de los bloques de la manera que convenga a las necesidades del desarrollo.

2.5.4 TRUFFLE

Truffle es un ambiente de desarrollo para Ethereum en que el que además puede desplegar tus proyectos en su propia red de desarrollo. Truffle es una forma fácil de desarrollar dapps y realizar las pruebas pertinentes de una forma simple, que cuenta con un compilador y herramientas para depurar los desarrollos.

2.5.5 SOLIDITY

Solidity es un lenguaje de programación utilizado en Ethereum para el desarrollo de Smart Contracts. Es un lenguaje de alto nivel parecido a Javascript diseñado para correr en la máquina virtual de Ethereum (EVM). Solidity es un lenguaje de tipo 'Turing Complete' que se refiere a la capacidad del lenguaje para resolver cualquier tipo de cómputo y ser capaz de añadir código más complejo como por ejemplo bucles.

2.5.6 VAGRANT

Vagrant es una herramienta de creación y configuración de entornos de desarrollo virtualizados de una manera rápida y sencilla. Vagrant proporciona un ambiente de desarrollo fácilmente configurable, reproducible y portable que trabaja con programas de virtualización como VirtualBox o VMWare.

2.5.7 DOCKER

Docker es un proyecto *open source* que permite desplegar aplicaciones en contenedores virtuales más ligeros que puedan ejecutarse en cualquier ordenador que tenga Docker instalado, independientemente del sistema operativo que se esté usando. Docker se diferencia de máquinas virtuales en tanto que no necesita un sistema operativo y sin embargo permite meter en ese contenedor todos los paquetes, librerías, programas o aquello que se necesite en el proyecto. Docker elimina el problema de la incompatibilidad de sistemas operativos o versiones ya que en el contenedor se encuentran todos y cada uno de los elementos que se pueden necesitar para ejecutar la aplicación.

2.5.8 METAMASK

Metamask es una extensión de Chrome que funciona como puente entre las aplicaciones descentralizadas y el navegador de una forma segura y la posibilidad de usar varias cuentas. Metamask nos permite firmar transacciones, administrar tokens y monederos a través de una interfaz fácil que agrupa todos los servicios Ethereum. Esto es posible gracias a la integración de la API Web3 en el JavaScript de cada sitio web. Metamask permite que corran aplicaciones basadas en Ethereum sin la necesidad de tener un nodo Ethereum. [17]

Metamask además, proporciona una capa extra de seguridad puesto que posee su propia llave privada, de manera que no maneja las claves de las cuentas y maneja los tokens propios, de esa manera las transacciones nunca tocan la configuración del servidor, reduciendo riesgos.

2.5.9 CAKESHOP

Cakeshop es un conjunto de herramientas y APIs para el desarrollo de entornos Ethereum. Se despliega como una aplicación Java e incluye los paquetes de Geth, Quorum y Constellation y un compilador de Solidity para los contratos inteligentes.

Esta aplicación nos permite manejar una red local blockchain, montar el clúster, explorar la cadena de bloques o trabajar con los contratos.

2.5.10 GIT

Git es un software para el control de versiones. Git gestiona los cambios que se realizan en los proyectos de desarrollo de aplicaciones y de código. Permite usar herramientas para modificar nuestro código y facilita la administración de las distintas versiones de cada producto. Además, Git posee un repositorio de proyectos y código conocido como GitHub al que se puede acceder para encontrar código para nuestros nuevos desarrollos o utilizar los componentes y herramientas que se encuentran accesibles al resto de usuarios.

2.5.11 NODEJS

NodeJs es un entorno de ejecución para JavaScript orientado a eventos asíncronos. Node está orientado a eventos lo que hace que sea más ligero y eficiente. Además, con la instalación de Node.js viene NPM, un sistema para trabajar con paquetes y librerías de código abierto.

2.5.12 BOOTSTRAP

Bootstrap es un *framework* de diseño web. Contiene numerosas plantillas con elementos de navegación basados en HTML, CSS y JavaScript. Será necesario introducir en el código de la aplicación las referencias a los archivos de Bootstrap para poder hacer uso de todas las fuentes, botones, menús y demás funcionalidades que vienen incorporadas.

Capítulo 3. ESTADO DE LA CUESTIÓN

A raíz de Bitcoin la tecnología Blockchain ha ido despertando un interés muy importante en grandes empresas, gobiernos e instituciones debido a su gran potencial y a su enorme popularidad.

Según el análisis de la empresa Gartner a Blockchain aún le quedan entre 5 y 10 años para alcanzar la madurez y convertirse en una tecnología implementada en la sociedad y en el mercado. Si observamos bien el gráfico que tenemos a continuación podemos ver que, aunque va decayendo la expectación por esta tecnología es muy grande y está produciendo una gran revolución de la que se esperan cambios en la forma de concebir las redes y los intercambios de transacciones.

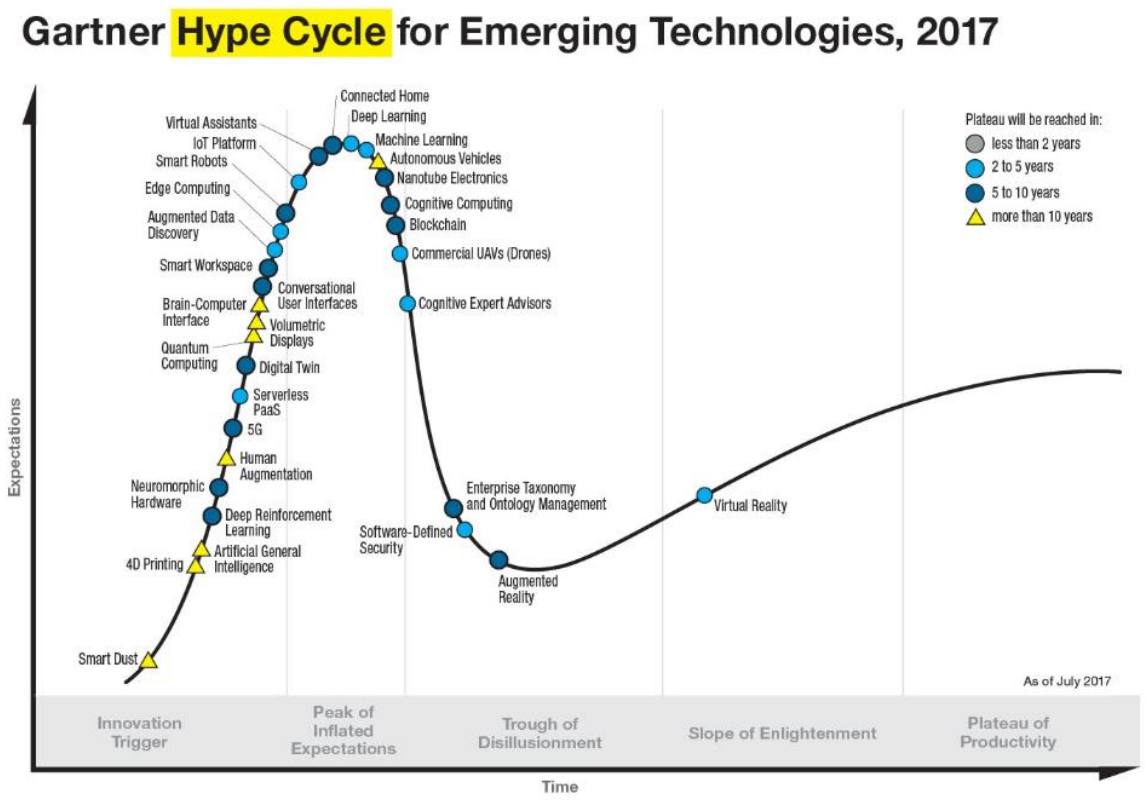


Figura 7. Ciclo de Garnet de nuevas tecnologías en 2017 (Garnet)

Desde que en 2009 surgiera Bitcoin, el mercado de criptodivisas no ha dejado de creer y también el número de usuarios que se están apuntando a esta nueva tendencia. Esta tendencia creciente se ha visto muy reforzada por la introducción de nuevas formas de financiación como las ICOs (Initial Coin Offering) y la emisión de tokens como activos digitales.

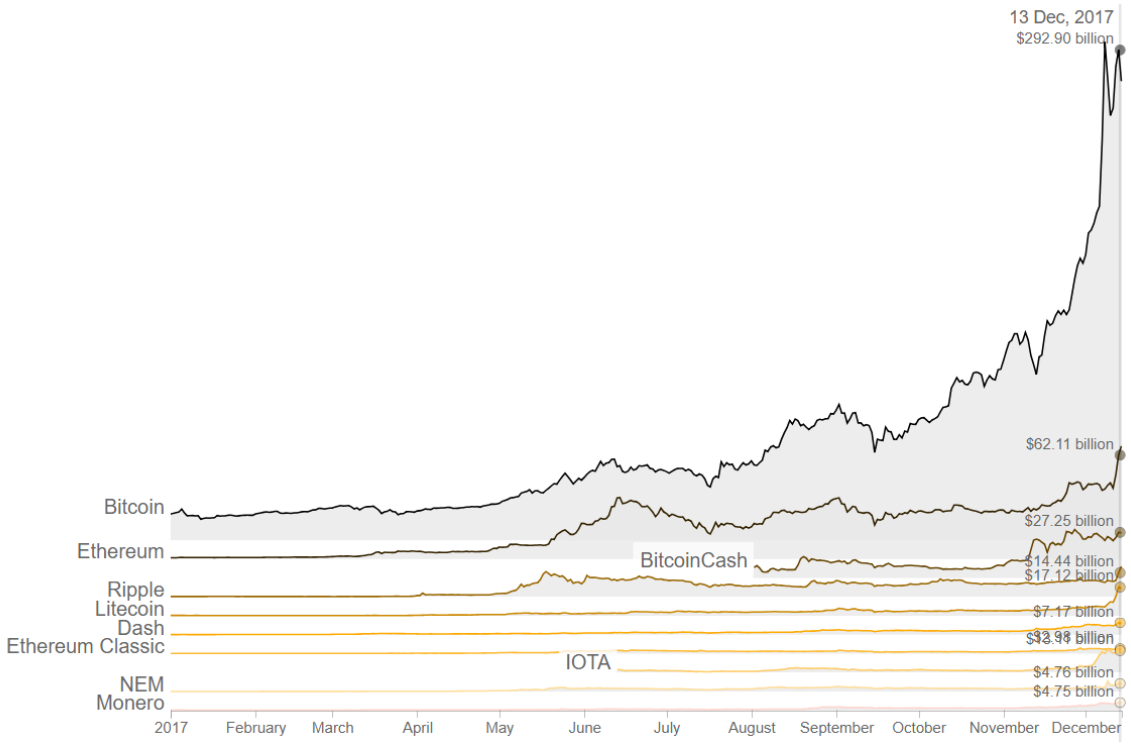


Figura 8:Top 5 Criptomonedas según su capitalización bursátil (Fuente: Thomson Reuters)

Sin embargo, la adaptación de esta tecnología es más lenta de lo que se podría esperar ya que se tienen dudas sobre la seguridad en la capa de usuario, los fines ilícitos y robos, la oposición de algunos gobiernos y sobre todo se encuentra el factor de la complejidad técnica que hace que muchos posibles usuarios no se encuentren cómodos con una tecnología que no consiguen entender. Además, la mayor parte de las personas no ven un problema en el dinero convencional y en las entidades reguladoras. [18]

A pesar del interés general que se ha generado en el ambiente empresarial los gobiernos y organismos reguladores no son tan proclives a aceptar los nuevos cambios. No todos los gobiernos reaccionan de la misma manera ante esta nueva tecnología y sus posibilidades. Países de todo el mundo están tomando posiciones respecto a la tecnología y los gobiernos respectivos buscan como aceptar y abrazar los nuevos cambios o simplemente cierran las puertas al uso de estas criptomonedas pudiendo llegar los usuarios a ser penalizados y multados. En la Figura 9. Adaptación de Blockchain en el mundo (Thomson Reuters) podemos ver de un vistazo cómo se posicionan los gobiernos alrededor del mundo y el nivel de tolerancia siendo en algunos casos nula.

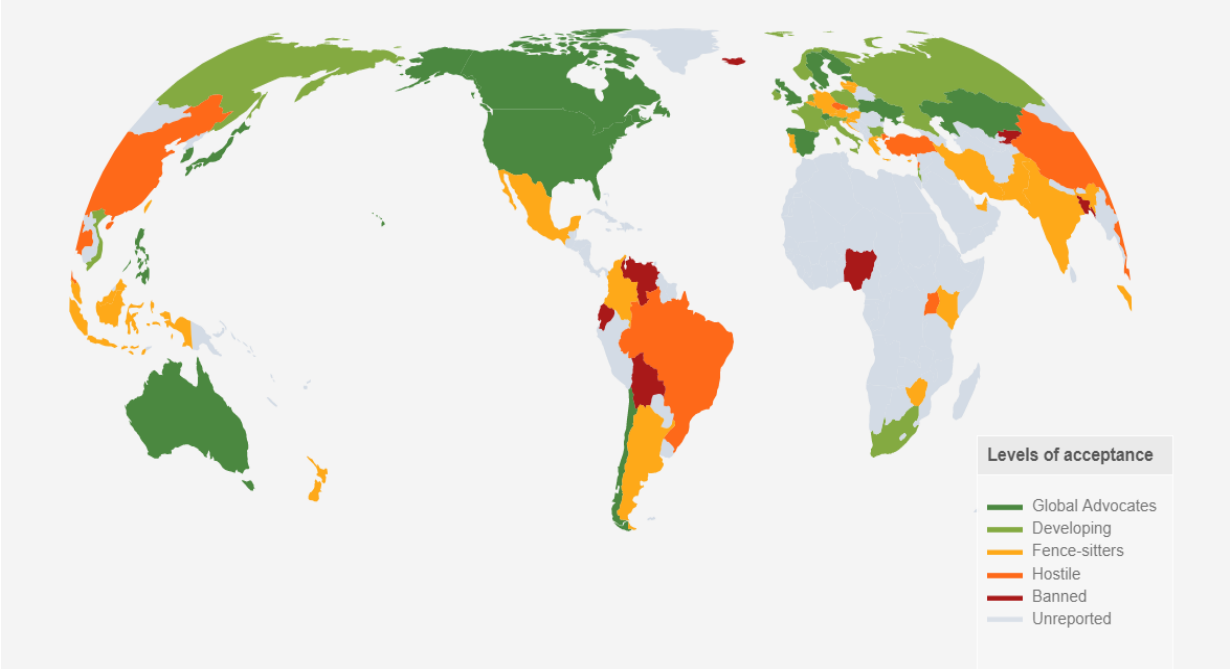


Figura 9. Adaptación de Blockchain en el mundo (Thomson Reuters)

El observatorio Blockchain Sngular [18] publicó en noviembre del año pasado un informe en el que se recoge la actividad relacionada con la tecnología Blockchain en España y a nivel mundial y recoge una serie de cifras y argumentos que nos pueden formar una idea bastante clara de la situación actual de la tecnología y sus aplicaciones. Algunos datos muy significativos son:

- El año pasado se registraron un total de 1.244 criptodivisas.
- Capitalización total de todas las criptomonedas: 184.014.248.248. 705 de dólares
- Recaudación total en forma de ICOs durante la primera parte de 2017: 1.270.000.000 de dólares
- Número de empresas que forman parte de Hyperledger: 172

3.1 SITUACIÓN DE BLOCKCHAIN EN ESPAÑA

En España, las grandes empresas financieras Santander, BBVA, Bankinter han decidido entrar a formar parte de grandes proyectos de Bitcoin como Coinbase y Ripple, lo que ha permitido de primera mano conocer la situación de la tecnología, los pros y contras que existen en todas las nuevas ideas y empezar a desarrollar proyectos basados en la cadena de bloques.

Las empresas españolas de todos los sectores no se están quedando al margen en el estudio y desarrollo de la tecnología de bloques. Una de las mayores representaciones de este interés es la puesta en marcha de la red Alastria, una red semipública permisionada, de la que forman parte más de 100 empresas españolas de diferentes sectores como financiero, telecomunicaciones y energético entre otros.

Actualmente hay aproximadamente 48 startups españolas que se dedican a las criptmonedas y a blockchain. Y sin olvidarnos de las universidades española como uno de los focos principales de innovación y estudio.

Como se puede ver en España hay un ecosistema preparado para desarrollar nuevas propuestas y avanzar hacia la implantación de esta nueva tecnología gracias al apoyo no solo de las grandes instituciones si no también gracias a startups que son capaces, de un manera muy ágil, de adaptarse a los nuevos cambios y son muy propensas a la innovación.

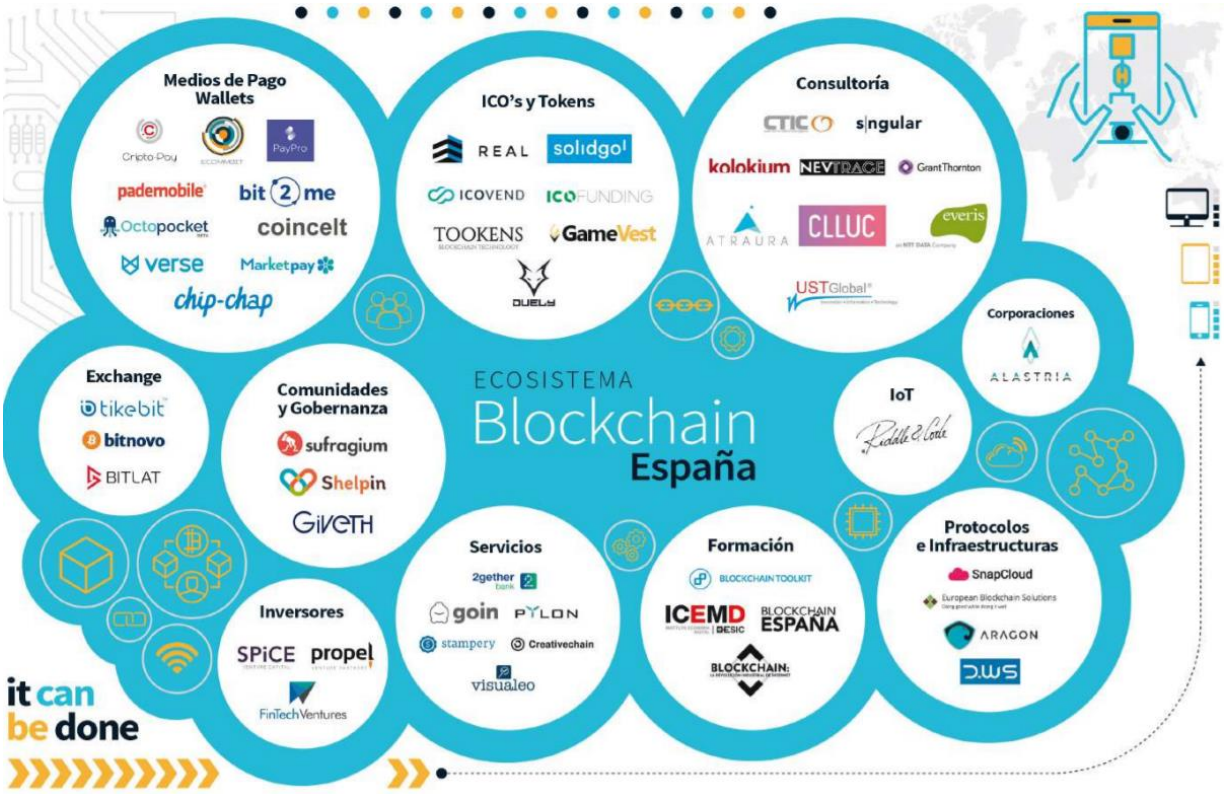


Figura 10. Ecosistema Blockchain España (Observatorio Sngular)

3.2 TECNOLOGÍAS BLOCKCHAIN PARA SMART CONTRACTS

Hay diferentes tecnologías que se pueden usar para el desarrollo de Smart Contracts. La más común en la actualidad es Ethereum y a partir de ella, surgen otras tecnologías que aportan una capa nueva de funcionalidad como Monax y Quorum. Otras que se encuentran aún en fases más tempranas de maduración son Hyperledger y Corda, soluciones que funcionan con otro tipo de blockchain diferente a Ethereum.

Solución	Madurez	Comodidad desarrollo	Permisionado	Tiempo	Privadacidad
Ethereum	4	3	Ninguno	~16 seg	No
Monax/Burrow	3	3	Fino	Inmediato	No
Quorum	3	3	Solo validadores	Inmediato	Si
Hyperledger	1	2	Fino	Inmediato	Si
Corda	1	3	Grueso	Inmediato	Si

Tabla 5. Comparativa de las distintas tecnologías Blockchain para Smart Contracts. (El nivel de madurez de la tecnología va del 1 al 5, de menor a mayor madurez).

Dependiendo de la finalidad de nuestra red, y del nivel de complejidad y madurez que necesitemos usaremos una u otra plataforma.

3.3 LA UNIVERSIDAD Y LA RED ALASTRIA

La universidad inauguró este 13 de junio de 2018 el primer nodo universitario en España de la red Alastria, de la que es socio fundador. Este nodo ya está conectado a la red, siendo de las primeras universidades que está experimentando con blockchain.

Este nodo está basado en Quorum y Ethereum. La implementación de este nodo supone un avance en el futuro intercambio eficiente de la información entre universidades y centros educativos. Además, aprovechando la infraestructura que posee la universidad con el clúster supone un ambiente idóneo para realizar pruebas de concepto e investigación que generarán grandes oportunidades para fomentar la innovación y el emprendimiento dentro de la universidad. [19]

3.4 CASOS DE USO

La tecnología que nos ofrece Blockchain nos abre una puerta a una amplia gama de posibles casos de uso y aplicaciones. Como se viene hablando en capítulos anteriores, al ser un sistema descentralizado y sin necesidad de una autoridad reguladora se facilita mucho la escalabilidad y el crecimiento, permitiendo que muchos nodos o participantes puedan disfrutar de los beneficios que aporta la cadena de bloques.

SERVICIOS FINANCIEROS

- Préstamos: si la persona que contrata el préstamo no realiza el pago en el tiempo estipulado, se ejecutaría el contrato para retirarle las garantías.

- Liquidación de operaciones: los contratos calculan importes de liquidación y transfieren fondos automáticamente.

- Pagos de cupones y bonos: los contratos calculan y pagan automáticamente de forma periódica los cupones y devuelve el capital al vencimiento de los bonos.

- Depósito en garantía en el registro de la propiedad: el contrato supervisa la información externa a la cadena de bloques y una vez transferida la propiedad de un vendedor a un comprador, el contrato ingresa automáticamente los fondos al vendedor.

- Herencias: una vez que el contrato puede verificar el fallecimiento de la persona, automáticamente las propiedades quedan repartidas y asignadas entre los herederos.

- Automatización de pagos y donaciones: se pueden acordar pagos o donaciones periódicas o puntuales a personas o entidades.

SERVICIOS DE LA SALUD

- Expedientes médicos electrónicos: los contratos proporcionan accesos a los historiales de cada paciente.

- Acceso a los datos sanitarios de la población: se podrían conceder a las organizaciones de investigaciones sanitarias el acceso a determinada información sanitaria personal.

- Seguimiento de la salud personal: se realiza un seguimiento de las acciones relacionadas con la salud de los pacientes a través de dispositivos IoT (Internet of Things)

SERVICIOS DE PROPIEDAD INTELECTUAL

- Distribución de royalties: el contrato calcula y distribuye los pagos de royalties a los artistas y a todas las otras partes involucradas en el contrato según los términos acordados.

SERVICIOS ENERGÉTICOS

- Estaciones autónomas de recarga para vehículos eléctricos: el contrato procesa un depósito, habilita la estación de recarga y devuelve los fondos restantes una vez completados.

SERVICIOS DEL SECTOR PÚBLICO

- Votación: valida los criterios del votante, registra el voto en la cadena de bloques e inicia acciones específicas como resultado del voto mayoritario.

- Apuestas: las partes interesadas pueden apostar de forma segura y sin necesidad de un tercero estableciendo las condiciones de dicha apuesta.

- Propiedades inteligentes: cualquier objeto que pueda conectarse a Internet se considera una propiedad inteligente de manera que a través de contratos podrían ser gestionadas su compra o venta o alquiler.

Capítulo 4. DEFINICIÓN DEL TRABAJO

4.1 JUSTIFICACIÓN

Como se ha podido comprobar en el capítulo anterior, Blockchain supone una tecnología que aún está en fase de estudio e implantación. Con un futuro prometedor a la vista, esta tecnología todavía debe enfrentarse a los retos de encontrar casos de uso y empresas y gobiernos dispuestos a invertir y adaptarlos a sus negocios.

Esta es la razón que impulsa el desarrollo de este proyecto. Un estudio de la tecnología a fondo, que permita obtener un conocimiento más técnico y riguroso. Además, aplicar este conocimiento para levantar una red con la que podamos realizar pruebas de concepto y probar nuestros desarrollos.

Por último y para cubrir todos los estratos posibles, se busca un caso de uso, orientado a las empresas y a descentralizar la gestión, de manera que no solo se conoce la tecnología si no que se da un paso más y se proponen soluciones que buscan mejorar la situación laboral implementando una tecnología puntera y que en un futuro no muy lejano será implementada en multitud de sectores y servicios.

4.1.1 USO DE LA TECNOLOGÍA BLOCKCHAIN

Se ha decidido trabajar con esta tecnología debido a las múltiples incógnitas que todavía no se han resuelto. La tecnología cada vez está más desarrollada y aún sigue en fase de experimentación y pruebas, pero ya se empieza a vislumbrar en algunos proyectos el potencial que esconden las cadenas de bloques. Supone un reto trabajar con las blockchains y escoger la que mejor vaya a encajar con nuestra definición del proyecto. Existen diferentes plataformas que responden a diferentes utilidades, por ejemplo, Bitcoin es una plataforma de pagos descentralizados, pero en este caso queremos ir un paso más. En este proyecto se

quiere profundizar en el desarrollo de contratos inteligentes y automáticos que se ejecuten cuando se den las condiciones programadas. Esto se puede desarrollar en la plataforma Ethereum, creada para este tipo de desarrollos y aplicaciones. Además, podemos configurar la red sobre la que trabajamos para que se adapte a nuestra situación y a los parámetros que necesitamos. Todas estas posibilidades son las que nos permite la tecnología blockchain, eliminando además todas las entidades reguladoras y controladoras. Es la propia red y los participantes los que tomen las decisiones que sean mejores para la red y nosotros podemos configurar qué reglas del juego vamos a emplear para llegar al consenso.

Blockchain nos aporta una flexibilidad de configuración y de trabajo que muy pocas infraestructuras nos lo van a permitir de esa manera. Además, y de modo personal, supone un reto nuevo y apasionante. Este trabajo estudia una tecnología en pleno desarrollo y experimentación, es pura innovación.

4.1.2 ELECCIÓN DE LA TECNOLOGÍA QUORUM

Se ha elegido Quorum debido a tres factores muy importantes: la confianza, la comunidad, y la madurez. Quorum aporta una capa más en las redes anónimas, añadiendo firmas y una red permisionada, sin embargo, al estar basada en Ethereum se aprovecha todo el desarrollo de una red popular y con mucho recorrido. Quorum evoluciona de la mano con Ethereum ya que solo se modifica el núcleo de forma mínima, de manera que se actualiza de una forma rápida y sin problemas.

La tendencia actual por el código abierto y libre hace que Quorum sea una plataforma con una licencia GPL/ LGPL que hace que los desarrolladores sean libres de experimentar y mantener la perpetuidad de la red mientras que se va renovando y ampliando.

Nos interesa además conocer la tecnología que están usando grandes consorcios de empresas como por ejemplo Alastria. Alastria es el primer consorcio formado por una amplia gama de empresas e instituciones que promueven el establecimiento de una estructura semipública basada en Blockchain, que soporte servicios con eficacia y validez legal en España y acorde con la regulación europea. Este consorcio es abierto y cualquier organización que lo desee

puede disponer de las herramientas fundamentales para desarrollar su estrategia Blockchain en el mercado español. Su objetivo principal consiste en habilitar un espacio digital común para el desarrollo de productos Blockchain. Habilita una estructura tecnológica que soporta modelos de Smart Contract con un enfoque multisectorial y privado. Además, es una red independiente y neutral sin modelo de negocio alguno.

4.2 OBJETIVOS

Este trabajo se va a estructurar en torno a cuatro objetivos fundamentales y los cuales nos servirán para evaluar si nuestro proyecto se ha realizado con éxito:

1. Análisis y estudio de la tecnología Blockchain. Conocer cuáles son las bases fundamentales de la tecnología y su funcionamiento. Es importante conocer el estado de esta tecnología en la actualidad y qué podríamos hacer con ella.

2. Despliegue de una red basada en Ethereum y Quorum. Ser capaz de implementar una red Blockchain y sus correspondientes nodos. Conocer la infraestructura de la red que lo soporta y sus comunicaciones.

3. Entender qué es un Smart Contract y los conceptos asociados a su utilización y programación. Se busca tener una visión más clara y concisa de qué significa este término y cuáles son los requisitos que se piden en función de la aplicación.

4. Desarrollo de una demostración real del funcionamiento de la tecnología. Se implementará un caso de uso en el que se aplique todo lo estudiado en los puntos anteriores. Se lanzará a una red para la interacción de la aplicación con los contratos de la aplicación a través de un servidor. Supone el culmen del estudio y aúna toda la información recopilada a lo largo del proyecto.

4.3 METODOLOGÍA

El proyecto por desarrollar se puede dividir principalmente en dos fases. Por un lado, un estudio y análisis detallado de las características de la tecnología y, por otro, una aplicación de esta para un caso de uso que refleje todo lo aprendido sobre la tecnología. Para ello, y dependiendo de la fase del proyecto se usará una metodología diferente.

Una vez se haya realizado un estudio intensivo de la tecnología y sus características será necesario desarrollar un caso de uso concreto para probar aquello que hemos estudiado de forma teórica. El desarrollo ágil de software se basa en un desarrollo iterativo e incremental, en el que los requisitos y soluciones evolucionan con el tiempo según la necesidad del proyecto. Cada iteración de este método incluye una serie de fases: planificación, análisis de requisitos, diseño, codificación, pruebas y documentación. De esta manera se va incrementando el valor del software asegurando que funciona mientras se van agregando funcionalidades.

El primer método que vamos a usar es Scrum. Es un proceso en el que se aplican una serie de buenas prácticas para obtener el mejor resultado posible de un proyecto. Se realizan entregas parciales de una manera constante priorizadas por el beneficio que aportan al éxito del proyecto. Scrum es ideal para proyectos como este en el que necesitamos obtener resultados pronto y de ahí se puede ir iterando para ir aportando más funciones a nuestro caso de uso de una forma más completa. Scrum es muy flexible para proyectos donde la innovación y la productividad son muy importantes.

4.4 PLANIFICACIÓN Y ESTIMACIÓN ECONÓMICA

A la hora de realizar un proyecto es de vital importancia organizar cómo ha de estructurarse el trabajo y las tareas. En este proyecto encontramos tres actividades importantes: el estudio de la tecnología, la implantación del caso de uso y la redacción de toda la documentación necesaria. A lo largo del proyecto se tendrán reuniones de seguimiento con el director del proyecto de manera que se realice un seguimiento eficaz del estado del trabajo.

En la siguiente figura se aprecia un diagrama de Gantt con la planificación de las tareas de este proyecto:

	Nombre	Duracion	Inicio	Terminado	Predecesores
1	**DOCUMENTACIÓN**	164 days	**24/11/17 8:00**	**11/07/18 17:00**	
2	Entrega del Anexo A	1 day	24/11/17 8:00	24/11/17 17:00	
3	Entrega del Anexo B	1 day	5/03/18 8:00	5/03/18 17:00	2
4	Entregar borrador memoria	1 day	8/07/18 7:00	9/07/18 17:00	3
5	Entrega de la memoria	1 day	11/07/18 7:00	11/07/18 17:00	4
6	**ESTUDIO Y ANÁLISIS**	72 days	**6/03/18 8:00**	**13/06/18 17:00**	
7	Estudio blockchain	48 days	6/03/18 8:00	10/05/18 17:00	
8	Estudio smart contracts	24 days	11/05/18 8:00	13/06/18 17:00	7
9	**LEVANTAR INFRAESTRUCTURA**	14 days	**14/06/18 8:00**	**3/07/18 17:00**	**6**
10	Despliegue de red Quorun	7 days	14/06/18 8:00	22/06/18 17:00	
11	Despliegue de red Alastria Test	7 days	25/06/18 8:00	3/07/18 17:00	10
12	**DESARROLLO CASO DE USO**	14 days	**14/06/18 8:00**	**3/07/18 17:00**	**8**
13	Requisitos	1 day	14/06/18 8:00	14/06/18 17:00	
14	Diseño	2 days	15/06/18 8:00	18/06/18 17:00	13
15	Entrega 1	1 day	19/06/18 8:00	19/06/18 17:00	14
16	Revisión 1	1 day	20/06/18 8:00	20/06/18 17:00	15
17	Desarrollo	7 days	21/06/18 8:00	29/06/18 17:00	16
18	Entrega 2	1 day	2/07/18 8:00	2/07/18 17:00	17
19	Revisión	1 day	3/07/18 8:00	3/07/18 17:00	18
20	**DESPLIEGUE EN RED ALASTRIA**	5 days	**4/07/18 8:00**	**10/07/18 17:00**	
21	Probar contratos de prueba	3 days	4/07/18 8:00	6/07/18 17:00	18
22	Probar aplicación	2 days	9/07/18 8:00	10/07/18 17:00	21
23	**REVISIÓN Y VALIDACIÓN**	49 days	**11/05/18 8:00**	**18/07/18 17:00**	**6SS**
24	Tests parciales	42 days	11/05/18 8:00	9/07/18 17:00	7
25	Validación final	1 day	10/07/18 8:00	10/07/18 17:00	24
26	DEFENSA DEL PROYECTO	1 day	18/07/18 7:00	18/07/18 17:00	

Figura 11. Cronograma de actividades del proyecto

Además, se ha utilizado la herramienta Trello para organizar las tareas pendientes de una forma visual y rápida.

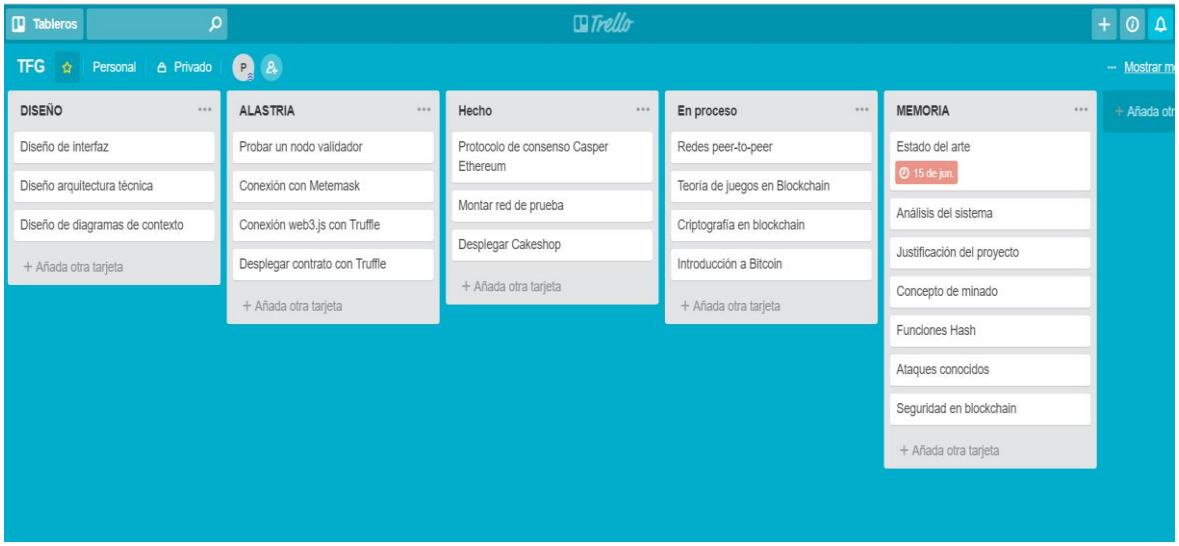


Figura 12. Tablero de Trello de actividades del proyecto

Atendiendo a la planificación que hemos mencionado en el párrafo anterior, los costes estimados se pueden recoger la siguiente tabla: Tabla 6. Estimación de costes para el proyecto

- Coste de infraestructura: Aprovechando las instalaciones y el clúster que se encuentra en la universidad, el caso de uso se desplegará sobre el nodo general que se ha levantado en la universidad, por lo que al estar instalada la infraestructura no tendremos que realizar ninguna nueva instalación.
- El equipo que necesitamos es un ordenador cualquiera con suficiente capacidad de procesamiento para realizar las pruebas pertinentes.
- Usaremos herramienta *Open Source*, que es la tendencia actual, aplicaciones con una gran comunidad que las soporta por lo que no necesitaremos ninguna licencia y no supondrá un coste extra a nuestra aplicación. Además, nos aseguraremos estar al día de las novedades y las actualizaciones.
- Se necesita un desarrollador con conocimientos en Solidity y que sea capaz de maquetar aplicaciones web con conocimientos de diseño. Un desarrollador *full-stack.*

De esta manera, podemos resumir los costes de este proyecto en la siguiente tabla:

	Justificación	**Coste**
Equipo	Ordenador portátil de 16 GB de memoria RAM	0 €
Herramientas	Herramientas Open Source	0 €
Infraestructura	Clúster de la universidad	0 €
Coste de la implementación	Coste de un desarrollador por hora (60€/h) por un promedio de dos meses a media jornada (160 horas)	9600 €
Total		9600 €

Tabla 6. Estimación de costes para el proyecto

Si este proyecto tuviera que ser desarrollado de forma profesional como se puede apreciar el único gasto real de la aplicación sería el tiempo de programación del desarrollador de la aplicación. Pero incluso así el desarrollo tendría un precio muy asequible puesto que la infraestructura ya está montada. En caso contrario este sería el gasto más importante debido a que los nodos requieren gran capacidad de computación, lo que incrementa mucho el precio.

Capítulo 5. ESTUDIO DE LA TECNOLOGÍA

En este apartado se va a proceder a explicar de una forma más específica que abarca todos los estratos de la tecnología y se va a dividir en tres niveles: arquitectura, red y aplicación. En la siguiente imagen podemos ver de forma gráfica las capas de lo que sería la arquitectura de nuestra red de una forma general. La capa de internet, con los protocolos TCP/IP y ya por encima lo que en la imagen se denomina capa blockchain que para hacerlo más sencillo vamos a dividir en la capa de la arquitectura y la capa de red. Por último, en el nivel superior, tendríamos la capa de aplicación con los smart contracts y las Dapps.

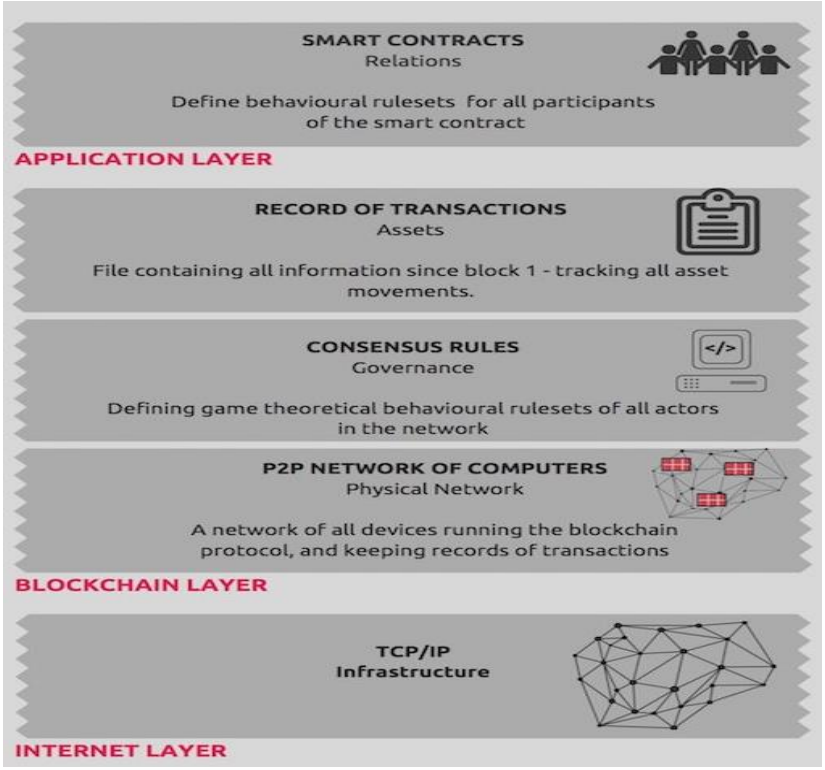


Figura 13: Arquitectura de Blockchain (Fuente:BlockcahinHub)

5.1 A NIVEL DE ARQUITECTURA

Blockchain surge como la combinación de tres conceptos: redes peer-to-peer, criptografía y la teoría de juegos. Todos los nodos de la red funcionan a su vez como cliente y servidor manteniendo una copia del estado de la red en cada nodo lo que permite la descentralización gracias a las redes P2P. El uso criptografía a través de un sistema de clave pública y de hash resulta esencial para la privacidad y la transparencia y, a su vez, estos nodos validan las transacciones mediante algoritmos de consenso con el apoyo de una mayoría y a través de incentivos económicos para la motivación y éxito de la red. [20]

5.1.1 REDES PEER-TO-PEER

El protocolo blockchain opera sobre Internet en una red peer-to-peer (P2P) de dispositivos en la que todo que mantienen el mismo registro de transacciones, permitiendo transacciones P2P sin una entidad mediadora.

Una red peer-to-peer o red de pares es una red en la que los dispositivos se comportan como iguales. En este tipo de red no hay clientes ni servidores, solo hay nodos que actúan de manera simultánea como clientes y servidores respecto a los demás nodos de la red. Es normal que este tipo de arquitectura se implemente por superpuesta sobre la capa de aplicación del protocolo de Internet[21]

Este tipo de tecnología hace referencia a un tipo de arquitectura de comunicación entre aplicaciones que permite compartir información sin que un servidor central facilite el intercambio. Una de las ventajas principales de este tipo de arquitectura es el máximo aprovechamiento de los recursos como el ancho de banda o la capacidad de almacenamiento de los nodos para ofrecer servicios de aplicación muy exigentes sin tener que confiar en un servidor central. Esta es una manera de evitar la congestión de la red y los cuellos de botella. Al no existir una única autoridad dota a la red de una alta tolerancia a fallos y una gran robustez.

5.1.2 CRIPTOGRAFÍA

La criptografía juega un papel fundamental puesto que es completamente necesario generar un sistema fiable y seguro que nos proporcione la confianza sin que haya un intermediario o entidad reguladora . En ese capítulo se van a mencionar los conceptos relacionados con los algoritmos 'hash' fundamentales en los procesos de verificación de la información y las transacciones dentro de la red.

5.1.2.1 Funciones Hash

Una función Hash es una función que asigna a un mensaje de longitud variable 'x' una clave de longitud fija 'y' independientemente de la longitud de la variable de entrada, de manera que H(x)= y. La salida de esta función representa un resumen de la información que se ha introducido. Para que esta función sea realmente útil y cumpla sus funciones en criptografía debe cumplir las siguientes propiedades:

- Resistencia a las colisiones: un buen algoritmo no debe producir dos funciones hash iguales de manera que no haya H(x) = H(y).
- Unidireccionalidad: se busca que sea fácil calcular la función hash que pero que sea muy difícil o imposible calcular la original a partir de la solución.

Las funciones hash están pensadas para que, al cambiar mínimamente la entrada de la función, cambie la salida drásticamente, de manera que sea imposible encontrar una correlación entre la entrada y la salida.

5.1.2.2 Puzles Hash

Todos los tipos de puzles tienen en común que se necesita un tiempo para resolverlos, tienen diferentes niveles de dificultad y casi todas las soluciones son obvias y fáciles de comprobar. Son una forma elegante de mantener la seguridad en las cadenas de bloques.

Los puzles hash no se pueden resolver si no es por el método de prueba y error usando la capacidad de computación. La tecnología blockchain utiliza este tipo de puzles para conseguir minar un bloque. Una información determinada produce una salida única y

diferente y por eso usan la información previa de la cadena y el hash del bloque anterior. Los mineros compiten por calcular números aleatorios hasta que uno de ellos produzca el resultado que coincide con el hash con un determinado nivel de dificultad asignado por el puzle. [22]

5.1.2.3 Árboles Hash

Un árbol hash también llamado árbol Merkle es una estructura de datos en árbol, en el que cada nodo que no es una hoja está etiquetado con el hash de la concatenación de etiquetas o valores de sus nodos hijo. Esto permite que gran cantidad de datos puedan estar ligados a un único valor de hash, el hash del nodo raíz del árbol, de manera que puedan ser verificados de una forma más eficiente

En Ethereum, cada uno de los bloques se representa como un árbol de Merkle para poder condensar todas las transacciones del bloque. Estas transacciones van a estar ligadas a un valor único del hash, el del nodo raíz que además irá firmado para asegurar que no ha sido alterado, que se recibe intacto y podemos verificar su fiabilidad.

De manera que, si un usuario quisiera modificar una transacción en cualquiera de las hojas del árbol, se modificaría también el hash de la raíz puesto que el hash sería diferente por la propiedad de las funciones hash que hemos visto en el apartado anterior. El protocolo detectaría que ese bloque es diferente y lo invalidaría. [23]

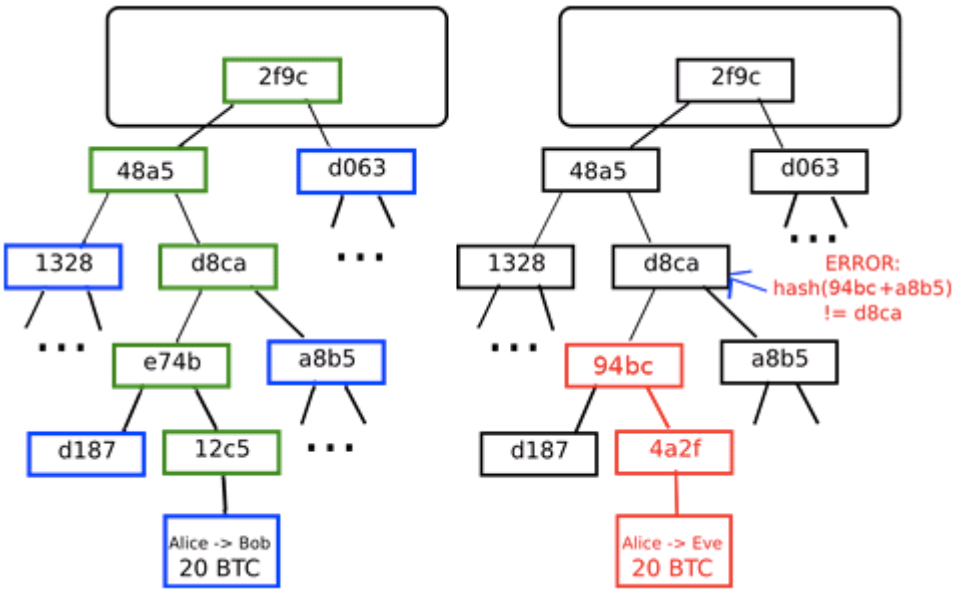


Figura 14. Ejemplo de árbol hash

5.1.3 TEORÍA DE JUEGOS

Blockchain devuelve al usuario el control de los recursos en lugar de confiarlo a una autoridad central a través de la teoría de juegos que constituye uno de los principales pilares sobre los que se sustenta esta tecnología. [24]

La teoría de juegos es el estudio sobre la toma de decisiones estratégicas. Un modelo de juego tiene al menos tres componentes: jugadores, estrategia y resultado. Además, se distinguen dos tipos de juego: juego de suma cero y juego sin suma cero.

El equilibrio de Cournot y Nash o también conocido como equilibrio del miedo, es una solución para juegos con dos o más participantes, en el que se asume que los jugadores han adoptado y conocen su mejor estrategia. De esa manera, cada jugador no cambia su estrategia si los otros jugadores no modifican la suya. De esta manera cada jugador ejecuta su mejor movimiento posible teniendo en cuanto los movimientos de los demás. Este concepto tiene un importante impacto sobre un sistema distribuido como Blockchain, que está basado en un sistema libre de trampas porque el protocolo está en equilibrio de Cournot y Nash.

Sin embargo, a veces lo que resulta más favorable para una serie de jugadores, no es beneficioso para la sociedad. Aquí entra el concepto de castigo y su implementación, al añadir este factor a la matriz de equilibrio, ésta cambia y obliga a los jugadores a comportarse de una manera honesta. Este sistema es el que encontramos en Blockchain, puesto que se trata de que los mineros se comporten honestamente.

El economista Tomas Schelling desarrolló una teoría llamada 'Punto focal' que consiste en una solución que las personas tenderán a usar si no hay comunicación entre ellos porque es especial, natural o relevante para ellos.

Un juego de coordinación fallará en el momento en que una minoría de participantes cambie su estado, mientras el resto lo mantiene, pero será un éxito si la mayoría consigue cambiar de estado. Si hay un par de participantes que hablan un lenguaje distinto no podrán comunicarse con el resto y serán rechazados con mucha probabilidad. Para la mayoría de los jugadores no merece la pena aprender ese lenguaje distinto, pero la minoría si estará interesada en adaptarse para evitar la exclusión.

Ahora, nos centramos de una forma más específica en la teoría de juegos asociada a Blockchain. De una forma simple y como ya se ha explicado en apartados anteriores una cadena de bloques está compuesta por una serie de bloques consecutivos por orden cronológico que contienen transacciones y, además, el hash del bloque anterior que vincula los bloques unos con otros formando la cadena. Cada bloque en la cadena tiene una puntuación. El bloque génesis, que es como se conoce al primer bloque de una cadena tiene una puntuación nula y el resto de los bloques puntuarán como la suma de la puntuación del bloque padre (aquel que precede al bloque) más el valor de la prueba de trabajo (PoW). El estado actual, que es el que se considera como verdadero, es el bloque con la puntuación más alta.

Los mineros tienen una capacidad muy importante para engañar al sistema o cometer fraudes y para mitigar este riesgo, la cadena de bloques usa modelos de la teoría de juegos que imponen la honestidad en el sistema. Los mineros pueden hacer trampas de diferentes formas como: incluir transacciones inválidas para ganar monedas adicionales, agregar bloques

aleatoriamente sin validarlos o minar bloques no válidos. Pero, la cadena de bloques actúa de manera que, si se introduce un bloque no válido en el sistema, automáticamente los que le preceden serán no válidos también, de manera que desalentará al resto de mineros a continuar la cadena, de manera que seguirán minando de la cadena que tiene una puntuación más alta, puesto que es la que se considera verdadera, es aquella que se considera un punto focal, a la que los usuarios tienden de manera natural.

5.2 A NIVEL DE RED

5.2.1 JSON RPC

JSON es un protocolo ligero para el intercambio de datos, basado en Javascript, es un formato de texto formado por una estructura tan simple como una lista de objetos en formato clave/valor.

RPC hace referencia a una llamada a un procedimiento remoto, es un programa que utiliza una computadora para establecer conexión y ejecutar código sin tener que preocuparse por las comunicaciones.

JSON-RPC es un protocolo simple y ligero de llamada de acceso remoto.

Para poder configurar nuestro nodo del que hablaremos en el siguiente apartado, es necesario utilizar una serie de parámetros que nos van a permitir habilitar las conexiones. Estan basados en el protocolo HTTP y pertenecen a la API JSON-RPC

```
--rpc Enable the HTTP-RPC server
--rpcaddr HTTP-RPC server listening interface (default: "localhost")

--rpcport HTTP-RPC server listening port (default: 8545)

--rpcapi API's offered over the HTTP-RPC interface (default: "eth,net,web3")

--rpccorsdomain Comma separated list of domains from which to accept cross origin
requests (browser enforced)
```

```
--ws Enable the WS-RPC server

--wsaddr WS-RPC server listening interface (default: "localhost")

--wsport WS-RPC server listening port (default: 8546)

--wsapi API's offered over the WS-RPC interface (default: "eth,net,web3")

--wsorigins Origins from which to accept websockets requests

--ipcdisable Disable the IPC-RPC server

--ipcapi API's offered over the IPC-RPC interface (default:
"admin,debug,eth,miner,net,personal,shh,txpool,web3")

--ipcpath Filename for IPC socket/pipe within the datadir (explicit paths escape
it)
```

5.2.2 GETH

Es una línea de comandos que nos permite correr e implementar las funciones de un nodo completo Ethereum en el lenguaje de programación Go. Geth permite minar ether, transferir fondos entre direcciones, crear contratos y enviar transferencias y explorar la cadena de bloques entre muchas otras funcionalidades.

Para la instalación de Geth es necesario instalar una versión de Go superior a la 1.9 y un compilador C. Una vez instaladas todas las dependencias utilizamos el comando

```
make geth
```

Y ya podremos acceder a todas las funcionalidades disponibles. A través de la consola Javascript con el modo Geth attach podemos acceder a nuevos comandos como los que se muestran en el ANEXO B – Comandos consola Geth

Fundamentalmente la opciones principal y más interesante que nos permite realizar Geth es manejar las cuentas disponibles. Te permitirá listar las cuentas, crear una nueva, actualizarlas o simplemente importarlas. Es muy importante que no se olvide la contraseña de una cuenta,

se necesita desbloquear las cuentas y sin ella será imposible acceder a ella. No es posible recuperar una cuenta sin la contraseña y se perderá todo el balance de las cuentas.

```
COMMANDS:
     list    Print summary of existing accounts
     new     Create a new account
     update  Update an existing account
     import  Import a private key into a new account
```

Las claves son guardadas bajo el parth <DataDir>/keystore en un formato:

```
 UTC--<created_at UTC ISO8601>-<address hex>
```

Y es bastante conveniente realizar una copia de seguridad de las claves de todas las cuentas que se encuentren en el nodo.

5.2.3 REDES PRIVADAS

Es posible crear redes privadas en Ethereum. Para ello es necesario configurar nuestra red con los parámetros necesarios:

- El archivo genesis-file
- El directorio de datos
- El ID de la red

- Genesis-file

Este archivo corresponde a la configuración del primer bloque de la cadena, el bloque 0, el cual es el único bloque que no tiene predecesor. El protocolo de la red se asegura que ningún otro nodo coincida con la versión de la red a menos que tenga el mismo archivo genesis-file.json. Esto permite crear todas las redes privadas que se deseen. [28]
Este es el archivo en el que se establecen

```
Genesis-file.json
{
  "alloc": {
    "0x58b8527743f89389b754c63489262fdfc9ba9db6": {
      "balance": "1000000000000000000000000000"
    }
  },
  "coinbase": "0x0000000000000000000000000000000000000000",
  "config": {
    "homesteadBlock": 1,
    "eip150Block": 2,
    "eip150Hash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
    "eip155Block": 3,
    "eip158Block": 3,
    "istanbul": {
      "epoch": 30000,
      "policy": 0
    },
    "isQuorum": true
  },
  "extraData":
"0x0000000000000000000000000000000000000000000000000000000000000000f85ad594b87dc3
49944cc47474775dde627a8a171fc94532b841000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000c0",
  "gasLimit": "0x2FEFD800",
```

```
  "difficulty": "0x1",
  "mixHash":
"0x63746963616c2062797a616e74696e65206661756c7420746f6c6572616e6365",
  "nonce": "0x0",
  "parentHash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
  "timestamp": "0x00"
}
```

- Nodos estáticos

En este archivo se guardan los nodos a los que quieres que se conecte tu nodo cada vez que se reinicie. El formato es el siguiente: se guarda el *enode* del nodo con su IP y el puerto RCP.

```
[

"enode://3905f943ba5446eba164c07ab5f53a84ce17d74ec4d7591f6ec54b9d7608f57cae7cfdf9
46616385f59cfb5b910161a1f8520cb6f992bcc0d1ab932601205e91@[::]:21000",
"enode://0dd
e7e76c5ea204b25c6e7473f3617ec5d01571b772ed0e271c3adcd6d78ef930decb27ea469db7be46d
c449c8f24d2bdc5e48105be7a58f953f9eaafd24582e@127.0.0.1:21000?discport=0",

"enode://0ff2a447bba5d48007b3b0822ce9e37ef31734388dbf35eef822c6cf7edd53f249e20bc8
b52f58bdb01d6383dad7ab946c366d4e395ac76cc22c02a8dce886dc@[::]:21001",

"enode://32c79d21c74986a84da09829dea104471d15b1b6d73c7dfb0f9b35d0f0b815bd115c01e0
e90ba7117719cb401604b1efc92eeef21eae87fe9fec5c486244590d@[::]:21002",

"enode://bedaedcbe9429ccd14d50b0f530a67ab60a03c8944621f35c4a04aaf4688802e88f237d4
69c4b2915e214ec8955da9cb1d0f39abedaa1df8edefe2abd4ad0540@[::]:21003",

"enode://e6d9ba20b71ed70dc72f17a19a18c441e04e46ed973ecca7be5aa2a5cc785a550530dcf1
2ed55f57f7ac566307c0fdedcc5e60fd31b8913486377ed8a2af0ef0@[::]:21004",

"enode://2b2da00dfe8431813a86f4f08637ea2edf3108d3505cd03bef30c791ac9236e02fa0ccee
569c275f7c1a417ff088f68cb619e97586154254424afe2f1e60843a@[::]:21005",

"enode://0dde7e76c5ea204b25c6e7473f3617ec5d01571b772ed0e271c3adcd6d78ef930decb27e
a469db7be46dc449c8f24d2bdc5e48105be7a58f953f9eaafd24582e@[::]:21006"
]
~
```

- Nodos permisionados

En las redes que se permite el permisionado es necesario añadir también la configuración apropiada para estos nodos. El fichero de configuración sería igual que el de nodos estáticos, pero con los nodos a los que permites en tu red.

```
[

"enode://3905f943ba5446eba164c07ab5f53a84ce17d74ec4d7591f6ec54b9d7608f57cae7cfdf9
46616385f59cfb5b910161a1f8520cb6f992bcc0d1ab932601205e91@[::]:21000",
"enode://0d
e7e76c5ea204b25c6e7473f3617ec5d01571b772ed0e271c3adcd6d78ef930decb27ea469db7be46d
c449c8f24d2bdc5e48105be7a58f953f9eaafd24582e@127.0.0.1:21000?discport=0",

"enode://0ff2a447bba5d48007b3b0822ce9e37ef31734388dbf35eef822c6cf7edd53f249e20bc8
b52f58bdb01d6383dad7ab946c366d4e395ac76cc22c02a8dce886dc@[::]:21001",

"enode://32c79d21c74986a84da09829dea104471d15b1b6d73c7dfb0f9b35d0f0b815bd115c01e0
e90ba7117719cb401604b1efc92eeef21eae87fe9fec5c486244590d@[::]:21002",

"enode://bedaedcbe9429ccd14d50b0f530a67ab60a03c8944621f35c4a04aaf4688802e88f237d4
69c4b2915e214ec8955da9cb1d0f39abedaa1df8edefe2abd4ad0540@[::]:21003",

"enode://e6d9ba20b71ed70dc72f17a19a18c441e04e46ed973ecca7be5aa2a5cc785a550530dcf1
2ed55f57f7ac566307c0fdedcc5e60fd31b8913486377ed8a2af0ef0@[::]:21004",

"enode://2b2da00dfe8431813a86f4f08637ea2edf3108d3505cd03bef30c791ac9236e02fa0ccee
569c275f7c1a417ff088f68cb619e97586154254424afe2f1e60843a@[::]:21005",

"enode://0dde7e76c5ea204b25c6e7473f3617ec5d01571b772ed0e271c3adcd6d78ef930decb27e
a469db7be46dc449c8f24d2bdc5e48105be7a58f953f9eaafd24582e@[::]:21006"
]
```

5.3 A NIVEL DE APLICACIÓN

Las Dapps son aplicaciones descentralizadas, desarrolladas a través de un software que utiliza un token para realizar la gestión de transacciones en una blockchain. Para ser considerada de tal forma, una aplicación debe cumplir con una serie de características concretas. La primera, debe ser código abierto, operar de forma autónoma sin necesidad de una entidad controladora. Las Dapps utilizan los tokens como combustible para funcionar.

5.3.1 WEB3.JS

Web3.js es una colección de librerías que permite la conexión local o remota con un nodo Ethereum usando una conexión HTTP o IPC. Además, nos permite compilar, desplegar e interactuar con los contratos inteligentes que hayamos desarrollado.

Algunos de los módulos que se incluyen en esta API contienen funcionalidades específicas para el ecosistema Ethereum. Por ejemplo:

- Web-eth: nos sirve para la blockchain de Ethereum y los smart contracts
- Web3-utils: contiene funciones de ayuda para los desarrolladores de Dapps.

Lo primera parte necesaria es la instalación para poder usar las librerías en nuestro proyecto. Podemos hacerlo mediante el siguiente comando:

```
npm install web3
```

Después necesitamos crear una instancia de web3 y asignar un proveedor. Ethereum soporta navegadores como Mist o MetaMask, este último será el que se ha escogido para nuestro desarrollo. Para web3.js debemos comprobar el Web3.givenProvider. Si es null, deberá conectarse a un nodo remoto o local.

```
// in node.js use: var Web3 = require('web3');

var web3 = new Web3(Web3.givenProvider || "ws://localhost:8546");
```

Ahora ya podemos usar el objeto web3.

- JSON INTERFACE

La interface json es un objeto json que describe la ABI (Application Binary Interface) para un smart contract de Ethereum. A través de esta interface, web3.js es capaz de crear un objeto JavaScript representando el contrato y todos sus métodos usando web3.eth.Contract.

Por eso cuando se conecte una aplicación javascript con nuestro smart contract tendremos que establecer el siguiente código en el que pasemos nuestro contrato en formato JSON

```
  initContract: function() {
    $.getJSON('/js/json/GestionarTokens.json', function(data) {
      // Get the necessary contract artifact file and instantiate it with
truffle-contract
      var GestionarTokensArtifact = data;
      App.contracts.GestionarTokens = TruffleContract(GestionarTokensArtifact);
      // Set the provider for our contract
      App.contracts.GestionarTokens.setProvider(App.web3Provider);
      return App.bindEvents();
    });

  },
```

En este caso, estamos diciendo a nuestra aplicación que el contrato que le estamos pasando es GestionarTokens en un formato que puede entender e interpretar como nuestro contracto con extensión .sol. [25]

5.3.2 TOKEN ERC20

Un token no es una criptomoneda, es una unidad de valor, que puede representar desde una moneda a un servicio. No es la moneda sobre la que se sustenta una plataforma blockchain, si no que corre por encima a nivel de aplicación, creada a través de un contrato.

Existen dos tipos de token, los tokens de uso, que actúan como divisa en las Dapps y tienen un valor monetario, pero no te confieren priviliegios en la propia red, y los tokens de trabajo que te hacen propietario de la Dapp y te hacen partícipe de las decisiones que se toman en cuanto al camino que sigue la aplicación.

Frente a la cantidad de tokens que se pueden crear en el ecosistema Ethereum, muchas veces es necesario interactuar con otros tokens, importarlos a tu cartera o realizar operaciones con ellos. Es por esta razón que se vio conveniente desarrollar un estándar para que los desarrolladores de contratos pudieran crear tokens compatibles de una forma fácil y auditada. De ahí surge la especificación del Token ERC20.

Esta especificación explica que el token debe estar formada por los siguientes métodos y eventos:

- Métodos
  - Name (opcional) – Nombre del token.
  - Symbol (opcional) – Símbolo del token.
  - Decimals (opcional) – El número de decimales que utiliza el token.
  - TotalSupply – Suministro total de tokens que existirán.
  - BalanceOf – Saldo de la cuenta del propietario.
  - Transfer(From) – Transferencia a…(desde)
  - Approve – Permite la retirada de fondos.
  - Allowance – Devuelve la cantidad que se puede retirar.
- Eventos
  - Transfer – Activado cuando se transfieren los tokens.
  - Approval – Activado siempre que se aprueba la transferencia.

De manera que al final, un token es un contrato inteligente que corre sobre Ethereum y que se comporta según la utilidad que nosotros le queramos dar en nuestra aplicación, por lo que el valor dependerá de la cantidad, de la red y de la confianza que genere el token.

Capítulo 6. MODELO DESARROLLADO

En este capítulo es donde se va a desarrollar el caso de uso para nuestra red. Este caso de uso ha sido elegido con una clara orientación al mundo empresarial y a crear un mejor entorno laboral en el lugar de trabajo. Por eso se va a desarrollar una aplicación basada en el concepto de gamificación, aplicando los nuevos conceptos adquiridos de blockchain y contratos inteligentes.

6.1 ANÁLISIS DEL SISTEMA

Actualmente las empresas apuestan por un modelo de trabajo que mejore la productividad de los trabajadores y cree un mejor entorno, un lugar en el que la persona se sienta cómoda y aproveche su máximo potencial. Este recurso conocido como gamificación, busca por medio de un sistema de objetivos y recompensas motivar a los trabajadores como si de un juego se tratase.

Por esto, y para mostrar como interactúan los contratos con los nodos de una red blockchain, se ha diseñado una aplicación sencilla y clara en la que se muestran las interacciones entre empleados y empresa por medio de tokens.

La aplicación consiste en lo siguiente: una empresa crea sus propios tokens y los asocia a su cuenta, los empleados, a medida que van consiguiendo los objetivos que se les establecen, se desbloquea el botón que les permite reclamar los tokens o puntos que cada objetivo les proporciona. Posteriormente podrían canjear esos puntos por productos o regalos que la empresa propone, y esos tokens volverían a la empresa.

El sistema diseñado consistirá en una aplicación web conectada a un servidor web que se conecta con un cliente y una red con la tecnología blockchain, este caso la red Alastria.

6.2 ESQUEMA DE LAS TECNOLOGÍAS

Para el desarrollo del caso de uso es necesario contar con una serie de aplicaciones externas que nos van a permitir conectar la aplicación con las cuentas de cada usuario, de manera que las transacciones queden firmadas y sean confirmadas.

A través de diferentes herramientas, el objetivo es conectar nuestra red blockchain, basada en Quorum y Ethereum a un servidor y a los clientes desde el navegador. Para ello lo primero y una vez desarrollado el código del contrato usando aplicaciones de auditoría como OpenZeppelin utilizaremos Truffle para desplegar nuestros contratos en un nodo general, conectándolo a la IP que se considere, ya sea local, si es una red de prueba o de desarrollo, o a una IP pública si es por ejemplo la red Alastria.

Necesitaremos un servidor donde poder colgar nuestra aplicación de manera que esté disponible y ahí colgaremos la aplicación en sí, es decir, los ficheros, las imágenes, los estilos, Bootstrap y demás que necesitemos para que nuestra aplicación no solo funcione si no que sea agradable para el usuario. Para ello crearemos un pequeño servidor Nodejs que podrá interactuar con nuestra red y el contrato a través de Metamask. Con Metamask, después de crearnos una cuenta y guardar esas 12 palabras que actúan como medida de seguridad, nos conectaremos por RPC al host y puerto del nodo al que apuntamos.

Una vez llegados a ese punto ya podemos empezar a trabajar con el contrato y nuestra aplicación. La aplicación se desarrollará en HTML, CSS y JavaScript y usaremos las librerías de web3.js para facilitar las conexiones JSON- RPC entre el contrato y la app.

Antes de lanzar nuestro proyecto a una red real conviene realizar las pruebas pertinentes en una red de desarrollo, y para ello utilizaremos Ganache, Ganache nos permitirá desplegar una red y las cuentas que necesitemos para probarlo en un entorno controlado.

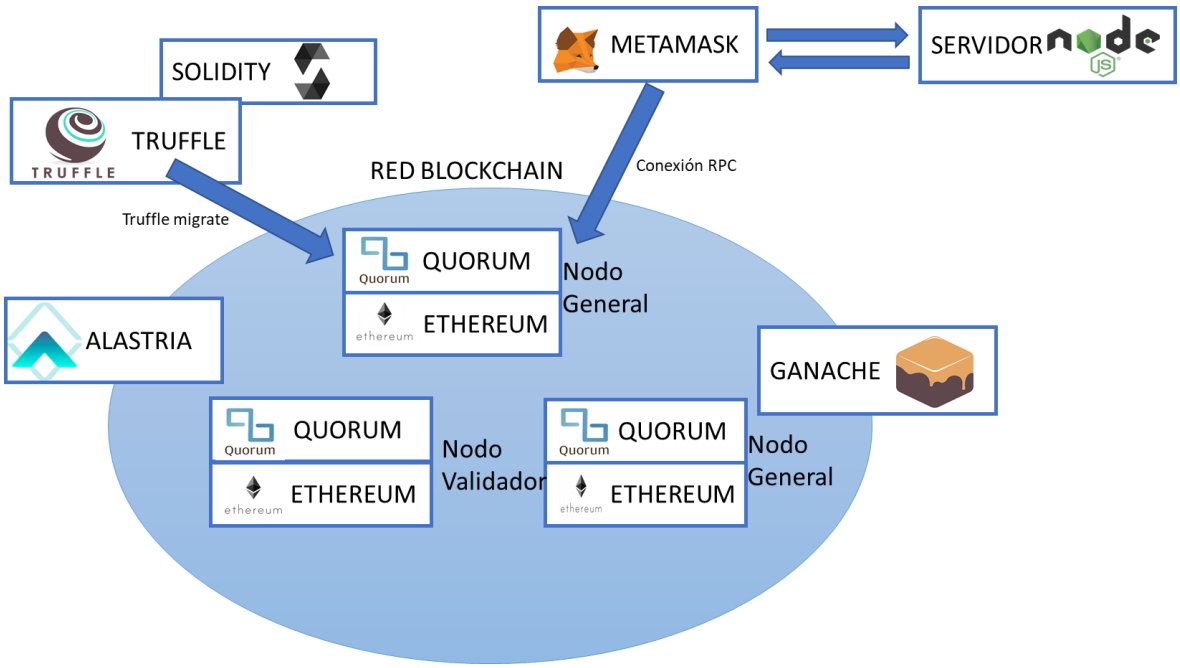


Figura 15. Esquema de las tecnologías para el desarrollo del proyecto

En la imagen Figura 15. Esquema de las tecnologías para el desarrollo del proyecto podemos comprobar la estructura que tendría la red y como se relacionan los elementos unos con otros, posibilitando así la buena ejecución de nuestra aplicación.

El usuario se conectará a la aplicación por medio del navegador y se podrá acceder a ella sin problemas, mientras que además el usuario estará usando sus cuentas y autenticándose también a través de Metamask.

6.3 DISEÑO

En este apartado se explica la estructura de la aplicación y los diagramas que han sido elaborados

6.3.1 MODELOS DE CONTEXTO

- **MODELO DE ARQUITECTURA**

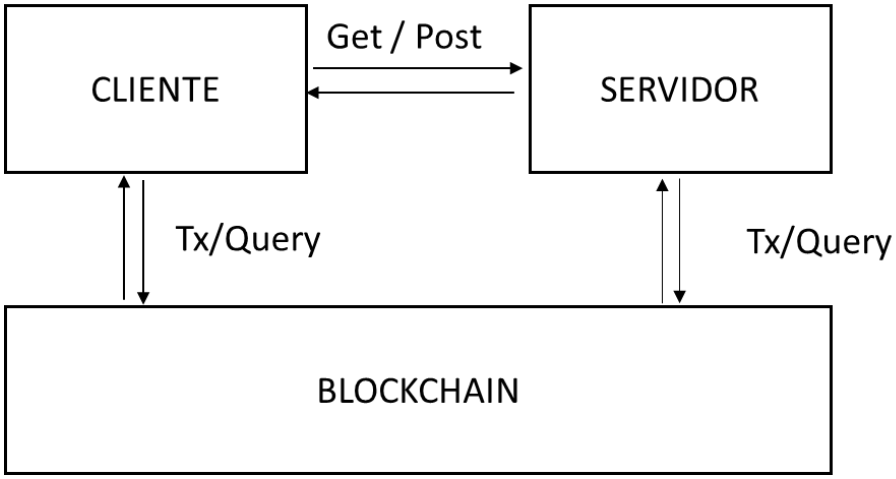


Figura 16. Modelo de arquitectura de la aplicación

Esta aplicación relaciona al servidor con la red P2P de blockchain y con el cliente a través de sus navegadores, pero además los usuarios se conectan también a la red con sus cuentas de Metamask que apuntan al nodo en el que se han desplegado los smart contracts. A través de un intermediario el servidor de aplicación es capaz de hablar con la red blockchain, en este caso Quorum y Ethereum, y a su vez acceder a los contratos a través de la ABI en formato json.

El modelo de arquitectura final será como la Figura 17, esta vez ya se han introducido todas las tecnologías necesarias para el diseño del sistema y en qué momento del proceso serán necesarias.

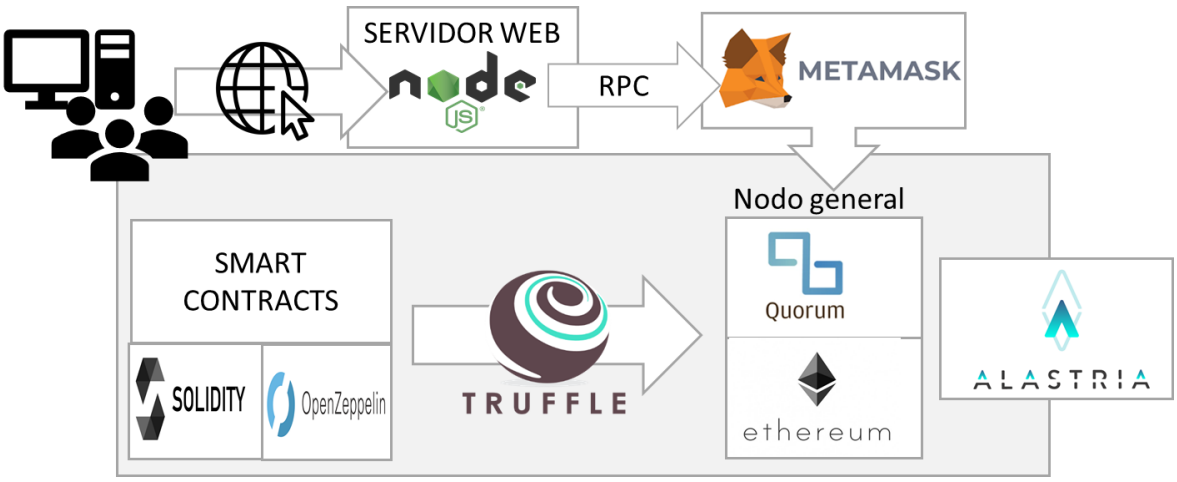


Figura 17. Diagrama de arquitectura del proyecto

6.3.2 DISEÑO DE COMPONENTES

- **DIAGRAMA DE SECUENCIA**

Para dejar más claro cómo debe compartarse la aplicación a continuación se muestran dos diagramas de secuencia para los casos en los que los empleados reclaman o canjean puntos siendo un caso de éxito.

Para el esceario donde el empleado desea reclamar sus puntos, la secuencia y la interacción serían las siguientes:

1. Nada más inicializar la página, la aplicación pregunta al contrato que el saldo de puntos que posee el empleado.
2. El contrato llama al método para conseguir el balance y le devuelve los puntos que tenga, el resultado podría ser igual 0 o superior.
3. Además la aplicación interactúa con el módulo de objetivos, que devolverá los objetivos y la situación de ellos, es decir si se han cumplido o no.
4. Si devuelve objetivos para reclamar, que vamos a suponer que es así, entonces el empleado ya puede solicitar los puntos de esos retos.

5. La aplicación llama al contrato para que se actualice el balance de puntos, llamando así a la cuenta, comprobando el balance y añadiendo los puntos a su cuenta.
6. Devolverá los puntos que tiene hasta ese momento el empleado y el botón de reclamar aparecerá inhabilitado.
7. El empleado ya tiene sus puntos actualizados y el saldo que podría canjear si quisiera.

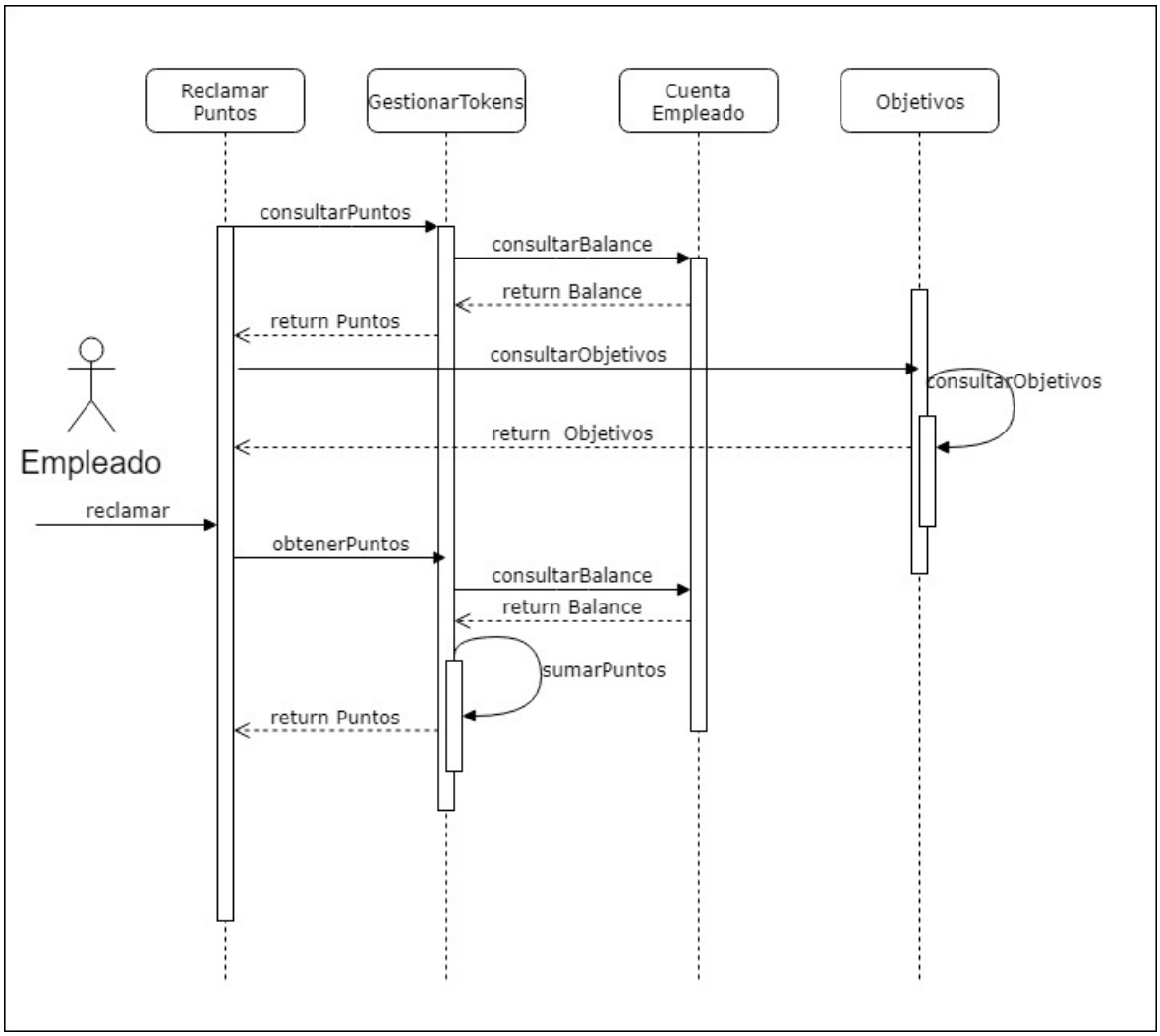


Figura 18. Diagrama de secuencia para solicitar puntos

En el escenario contrario, el empleado quiere consumir los puntos que ya reclamó anteriormente, en este caso el diagrama de secuencia sería el siguiente:

1. Al iniciarse la página, la aplicación llama al contrato para comprobar el saldo de puntos que tiene el usuario.
2. El contrato GestionarTokens consulta los puntos que tiene la cuenta sobre la que se está haciendo la consulta.
3. Se devuelven el saldo de la cuenta del empleado.
4. A continuación, la aplicación interactúa con el módulo de regalos para devolver todos los regalos disponibles.
5. El usuario decide qué regalo quiere canjear y la aplicación llama al método canjearPuntos del contrato para que se compruebe si el usuario tiene suficientes puntos y de ser así, eliminar puntos.
6. Se devolverá el saldo actualizado y un mensaje para que se sepa que ha sido confirmado y que la información llegará al email.

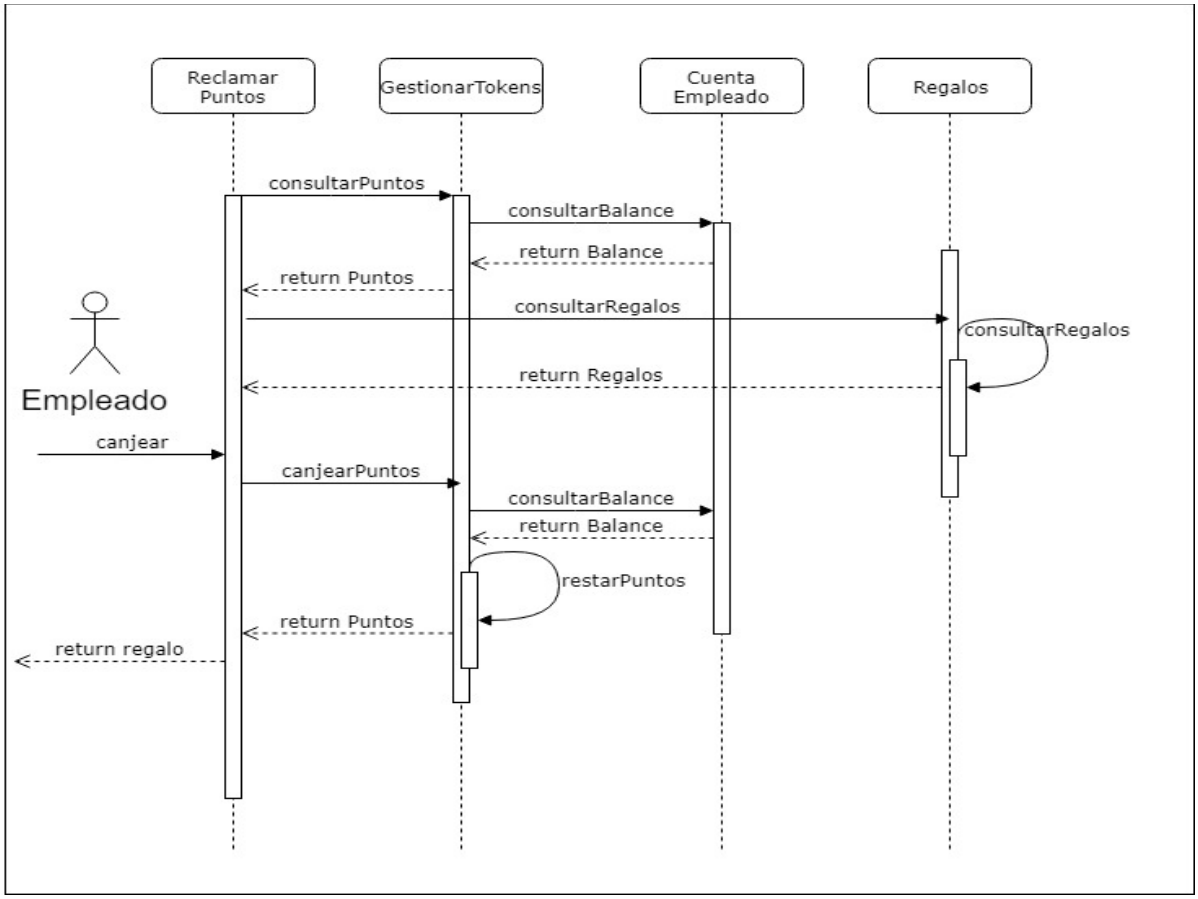


Figura 19. Diagrama de secuencia para canjear puntos

- **CASOS DE USO**

Un empleado de la empresa debe ser capaz de realizar las siguientes actividades

- Ver la información detallada de su perfil
- Debe ser capaz de reclamar los tokens de los objetivos ya cumplidos a través de su cuenta y de Metamask
- Debe ser capaz de canjear los tokens, siempre que tenga disponibles, y recibir la confirmación a través de su autenticación en Metamask.
- Deberá ser capaz de acceder a su cuenta de Metamask y a su cuenta de empleado para poder interactuar con la aplicación y poder ganar y canjear sus tokens.

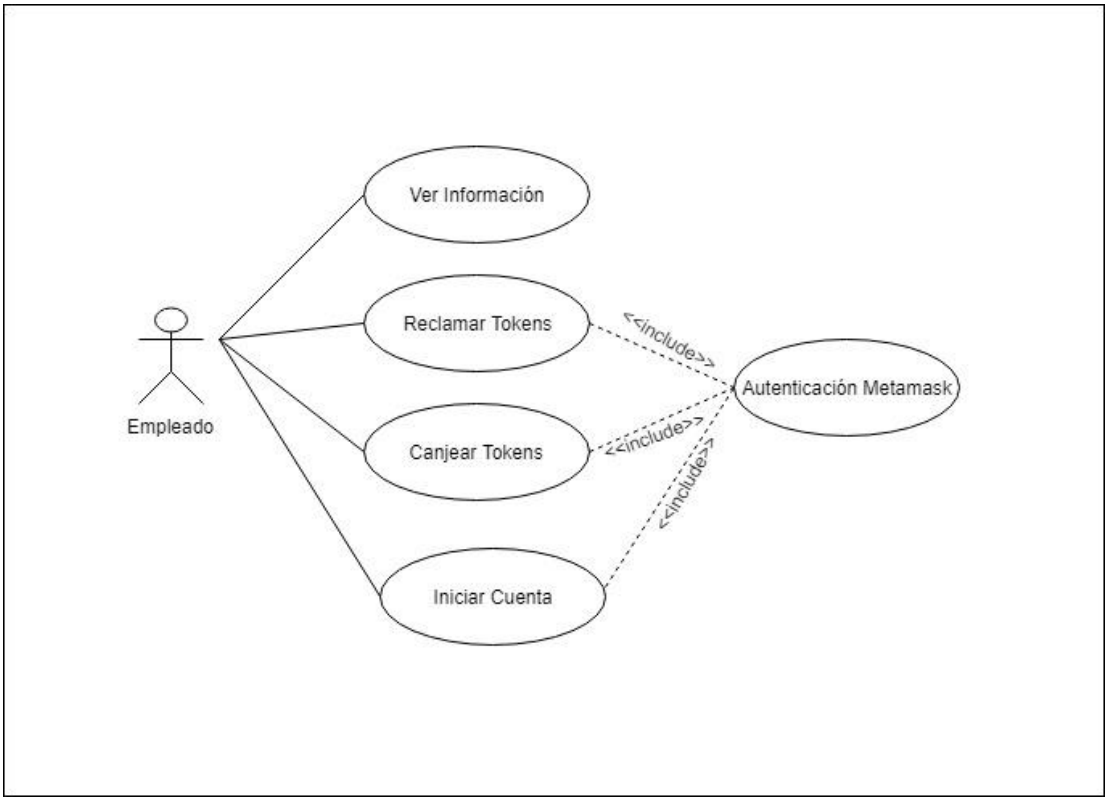


Figura 20. Diagrama de casos de uso para el empleado

A su vez, la empresa también podrá realizar unas operaciones parecidas al empleado, solo que su misión será la de crear los tokens.

- Podrá crear los tokens en una página especial para el administrador y a través de la confirmación en el navegador con Metamask.
- La empresa será capaz de ver los tokens que se han creado. Por defecto al pulsar el botón todos los tokens irán a la cuenta de la empresa.
- Cuando el empleado canjee los tokens, la empresa recuperará esos tokens que irán a pasar a la cuenta de la empresa, modificando el balance
- Por supuesto la empresa debe poder acceder a su cuenta con los tokens y gestionarla a través de las claves de Metamask.

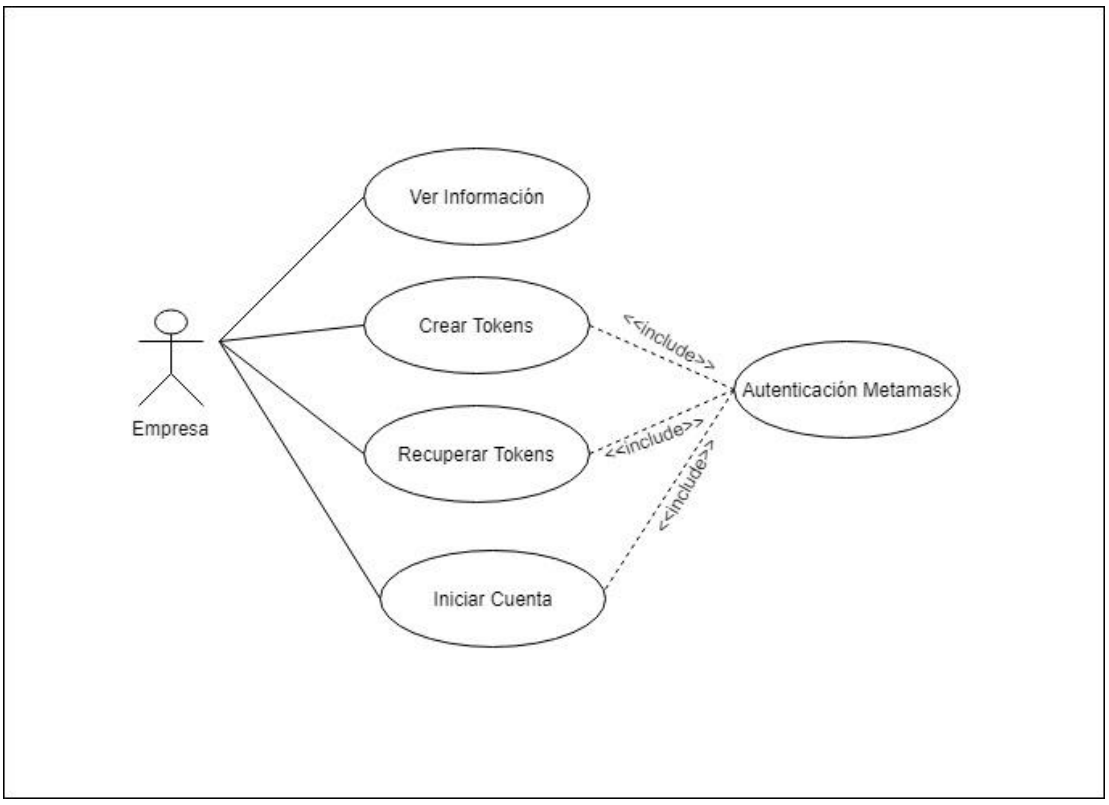


Figura 21. Diagrama de casos de uso para la empresa

6.3.3 DISEÑO DE INTERFAZ

- **DISEÑO DE NAVEGABILIDAD**

La aplicación contiene una interfaz muy sencilla y visual que permite tanto al empleado como a la empresa gestionar y visualizar la información. Centrándonos en el empleado, su página de inicio nos llevaría a su perfil en el que se podría ver la información general como su puesto o su correo de la empresa. Desde ahí puede acceder a las páginas de Obtener Puntos o Canjear Puntos donde se pueden ver los objetivos cumplidos o los productos a canjear.

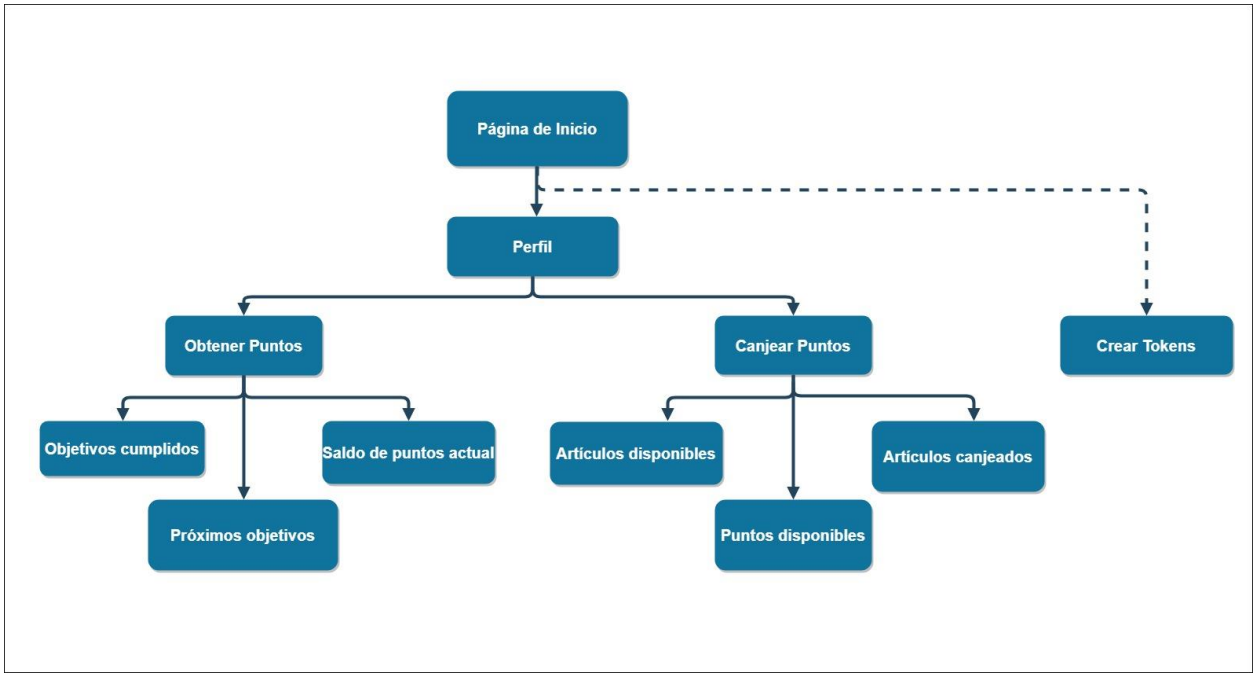


Figura 22. Diagrama de navegabilidad

6.4 IMPLEMENTACIÓN

6.4.1 CREACIÓN DE LOS CONTRATOS

Lo primero que vamos a hacer es personalizar los tokens con los siguientes atributos que debe tener nuestro Token ERC20 como hemos visto en apartados anteriores.

```
string public name= "TFG Tokens";
string public symbol= "TFG";
uint8 public decimal=2;
uint public INITIAL_SUPPLY=100000;
```

A la hora de desarrollar una aplicación, y especialmente cuando estás desarrollando un smart contract, es muy importante programar teniendo en cuenta la seguridad y la fiabilidad ya que se trabaja con activos y cuentas de usuario. Para ello se puede usar código que ha estado auditado y ya ha sido probado por expertos, como es el caso de OpenZeppelin, un repositorio de contratos seguros que podemos usar en nuestro desarrollo. En el anexo ANEXO D – OpenZeppelin podemos ver cómo instalarlo y usarlo en nuestro contrato.

En este caso se dejará el código de los tokens a cargo del repositorio seguro y en nuestro contrato nos dedicaremos a la gestión de los tokens, crearlos, añadirlos, transferirlos... utilizando los métodos que el contrato de tokens nos proporciona, de manera que nuestro contrato sería un Gestor de Tokens ERC20 compatible con las aplicaciones que necesitamos.

Las funciones más importantes de nuestro código serán las correspondientes a ‘obtenerPuntos’ y ‘canjearPuntos’

```
function canjearPuntos (uint256 cantidad, address empleado) public returns (int)
{

            int puntosRestantes;
            bool realizado;
            bool aprobado;
            uint256 balance;

            balance = balanceOf(empleado);

            aprobado = approve(this, cantidad);
```

```
            realizado = transfer(this, cantidad);
            revert('No se ha podido realizar la transacción');
            if(realizado) {
                  puntosRestantes = int(balance - cantidad);
            } else {
                  puntosRestantes = -1;
            }

            //devolvemos los puntos restantes que le quedan al empleado
            return puntosRestantes;

      }*/

function obtenerPuntos (address empresa, address empleado, uint256 cantidad)
public returns (int) {

            //el empleado reclama los puntos de sus obejtivos
            int puntos;
            //bool aprobado;
            bool realizado;
            uint256 balance;

            balance = balanceOf(empleado);

            if(realizado=realizarTransaccion(empresa, empleado, cantidad)) {
                  puntos = int(balance + cantidad);
            } else {
                  // -1 es que no se ha realizado la transacción
                  puntos = -1;

            }

            //devolvemos el total de puntos que le quedan
            return puntos;
      }
```

En ambas funciones lo importante es consultar el balance de tokens que posee el empleado en su cuenta y comprobar que en el caso de canjearPuntos, tiene saldo suficiente para poder gastar tokens y que no se produzca un fallo, gastando más de lo que tiene en la cuenta. Una vez comprobado, se realiza la transferencia, devolviendo un valor -1 en caso de que no se haya podido realizar la transacción, de manera que durante el desarrollo podamos depurar mejor el código y poder saber exactamente qué está pasando en la aplicación.

6.4.2 DESARROLLO DE UNA DAPP

En este apartado se va a desarrollar un breve tutorial sobre cómo desarrollar un smart contract para un caso de uso aplicado.

1. Configurar el entorno de desarrollo

Lo primero es instalar Truffle ya que es el entorno de desarrollo que vamos a usar. Es necesario además contar con Node.js y Git.

```
npm install -g truffle
```

2. Crear un proyecto en Truffle

Primero creamos el directorio sobre el que se va a trabajar.

```
mkdir nombre-proyecto

cd nombre-proyecto
```

A continuación, creamos un proyecto vacío con el comando:

```
truffle init
```

Un directorio en Truffle por defecto contendrá los siguientes ficheros:

- contracts/: contiene los ficheros fuente para nuestros contratos. Contiene uno muy importante por defecto 'Migrations.sol'
- migrations/: Truffle usa un sistema de migraciones para manejar los despliegues de los diferentes contratos. Una migración es un tipo de contrato especial que mantiene un registro de todos los cambios.
- test/: contiene los ficheros de Solidity y Javascript para las pruebas de nuestro contrato.
- truffle.js: es el archivo de configuración de Truffle.

3. Desarrollar el Smart Contract

En este caso el lenguaje de programación que vamos a utilizar es Solidity, por lo que los contratos irán en la carpeta contracts/ con la extensión '.sol'. Además, en la carpeta migrations/ se tendrá que configurar el despliegue de los contratos que estamos desarrollando.

Un ejemplo sencillo sería un contrato para almacenar una variable:

```
pragma solidity ^0.4.19; // información que solo utiliza el compilador

contract GestionarTokens {

      //variable
      uint myToken;

}
```

4. Configurar el archivo 'truffle.js'

Este es el paso en el que debemos decidir cuáles son los parámetros necesarios para nuestra red. Debemos pensar en el host, el puerto, el ID de la red, el precio del Gas… Es en este momento donde debemos configurar cómo interactúa el contrato con la red. El archivo tendrá una configuración parecida a esta:

```
module.exports = {
  // See <http://truffleframework.com/docs/advanced/configuration>
  // for more about customizing your Truffle configuration!
  networks: {
    development: {
      host: "127.0.0.1",
      port: 7545,
      network_id: "*" // Match any network id
    },
    alastria: {
       host: "130.206.64.6",
       port: 22000,
       network_id:"82584648528",
        gasPrice:0,
        gas:"0x2FEFD800",
    },
```

```
      proyecto: {
            host: "127.0.0.1",
            port: 22001,
            network_id: "9535753591",
        gasPrice: 0,
        gas: 4712388
       },

       general1: {
      host: "localhost",
      port: 22001,
      network_id: "*", // Match any network id
      gas: 0xfffff,
      gasPrice: 0x0,
      from: "0x74d4c56d8dcbc10a567341bfac6da0a8f04dc41d"
    }
  }
};
```

Podemos tener más de una red si queremos probar primero nuestro contrato en redes de desarrollo, o queremos probar en diferentes redes.

5. Compilar y migrar contratos

Usamos dos comandos para compilar y migrar nuestros contratos a la blockchain, en este caso a la propia que tiene Truffle para desarrollos.

```
truffle compile
truffle migrate -network <nombre_de_la_red>
```

Es posible que si se está trabajando en Windows haya un problema con los nombres de los ejecutables. Hay tres formas de evitar este error

- Cambiar el nombre del fichero de configuración 'truffle.js' a algo como 'truffle-config.js'
- Escribir en línea de comandos "truffle.cmd compile"
- Editar el Path del sistema y eliminar los archivos .js como ejecutables.

Para poder desplegar el contrato debemos configurar la migración. Este fichero que podemos llamar '2_deploy_contracts.js' en la carpeta migrations/ es el que nos permite desplegar nuestro contrato en la blockchain. El código sería el siguiente:

```
var GestionarTokens = artifacts.require("GestionarTokens");

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

6. Realizar pruebas y tests

Podemos utilizar esta funcionalidad que incopora Truffle en la que podemos crear funciones de prueba y comprobar si la salida de los métodos de nuestro contrato son los correctos antes de desplegarlos a la red.

```
truffle test
```

6.4.3 SERVIDOR NODEJS

Para albergar la aplicación y poder acceder a ella desde un navegador se ha creado un pequeño servidor que escuchará en el puerto 3030 de la dirección localhost.

El concepto sobre cómo funciona el servidor es el siguiente: tenemos un objeto 'App' al que le configuramos nuevas rutas. A cada ruta se le asigna una función de *callback*, que tiene un objeto 'req' que es una *request* que tiene los parámetros y las cabeceras además de todo lo que queremos que mande nuestra página, y un objeto 'res' que es una *response*, con el que se puede redirigir a sitios, enviar archivos, objetos *json*, en este caso mandamos el fichero con el código HTML de nuestras páginas,

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

const app = express();

app.use(express.static(__dirname + '/public'));

app.get('/', function(req, res) {
	console.log('Se ha recibido una peticion al root');
	res.sendFile(path.join(__dirname + '/public/html/index.html'));
});

app.get('/canjear', function(req, res) {
	console.log('Se ha recibido una peticion al root');
```

```
	res.sendFile(path.join(__dirname + '/public/html/canjearPuntos.html'));
});

app.get('/retos', function(req, res) {
	console.log('Se ha recibido una peticion al root');
	res.sendFile(path.join(__dirname + '/public/html/obtenerPuntos.html'));
});

app.get('/informacion', function(req, res) {
	console.log('Se ha recibido una peticion al root');
	res.sendFile(path.join(__dirname + '/public/html/informacion.html'));
});


const PORT = process.env.PORT || 3030;
app.listen(3030, function() {
	console.log('Listening on port ' + PORT + '...');
});
```

Cuando queramos acceder a la aplicación bastará con correr el siguiente comando

```
npm start
```

Y en cualquier navegador, por ejemplo, Chrome, escribimos en la barra de navegación: 'localhost:3030' y ya podemos acceder a nuestra aplicación que se conectará con Metamask.

Capítulo 7. CONFIGURACIÓN DE ALASTRIA

7.1 ARQUITECTURA

Alastria, como se ha mencionado en apartados anteriores, es un consorcio que ha desplegado una red blockchain privada basada en Ethereum y Quorum. Para poder operar en la red es necesario ser socio del consorcio y desplegar un nodo. Cada socio puede elegir que rol quiere jugar en la red y qué tipo de nodos puede aportar. [26]

Existen dos tipos de nodos:

- Nodos validadores: son los encargados de ejecutar el protocolo de consenso de Quorum para validar las transacciones que se producen en la red Alastria. Las escrituras en el libro mayor de la red se producen por este tipo de nodos. desplegar aplicaciones. Quorum constituye la base del cliente y Monitor para visualizar las transacciones. Para ello se debe dar conectividad a los siguientes puertos:

- 21000 TCP/UDP como entrada y salida para todos los nodos de la red.
- TDB, para la monitorización, además de necesitar el puerto RPC local de Geth.

- Nodos de aplicación o regulares: sobre este tipo de nodos se corren las aplicaciones y los smart contracts. Conforme va creciendo la red van siendo más necesarios para distribuir la carga de la blockchain y garantizar la seguridad y la privacidad. Un nodo regular debe desplegar tres procesos fundamentales. Por un lado, Quorum, que será nuestro cliente Ethereum base, Constellation para gestionar las transacciones privadas y por último Monitor, para poder monitorizar las transacciones. Para ello se debe permitir la conexión de los siguientes puertos:

- 21000 TCP/UDP como entrada de los nodos validadores para el protocolo Ethereum.
- 9000 TCP, como entrada y salida para Constellation
- TDB, para la monitorización, además de necesitar el puerto RPC local de Geth.

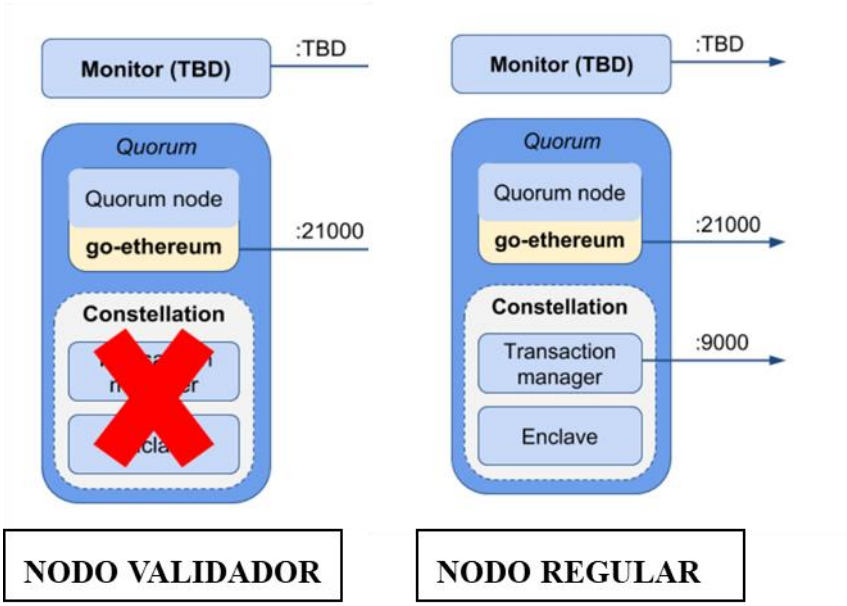


Figura 23. Anatomía nodos de Alastria (Alastria GitHub)

Cada nodo Quorum especifica los nodos que se pueden conectar a él a través del fichero: "permissioned-nodes.json". De esta manera los permisos de conexión en Quorum se gestionan mediante nodos, y no con direcciones. Un nodo se identifica por su '*enode'*, su IP pública y su puerto.

7.2 CONFIGURAR TESTNET

Para evitar hacer las pruebas en un entorno real, se recomienda desarrollar en la Testnet de Alastria y poder probar los códigos en local, desplegando los nodos validadores y generales que se permiten [27]. Como estamos trabajando en un entorno Windows y es necesario usar

Ubuntu, se ha activado el Subsistema de Windows para Linux y se ha descargado Ubuntu, una vez instalado, usaremos la línea de comandos de Ubuntu para instalar nuestra red de test y nuestro ejemplo.

Nota: es posible que al instalar Nodejs de nuevo en nuestro subsistema encontremos problemas de versiones y paquetes.

Esto nos pasa porque Ubuntu está usando nuestro PATH de Windows, en lugar del de Ubuntu. Es por eso que hay que cambiar el PATH usando el siguiente comando:

```
export
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/u
sr/local/games
```

Es necesario añadirlo al fichero de configuración home/user/. bashrc

Una vez modificado el PATH para que se corresponda a Ubuntu podemos empezar a instalar los paquetes necesarios para crear nuestra red.

```
sudo apt-get install git curl
```

Creamos una carpeta llamada 'tfg' que cuelgue de home/user/ y trabajamos desde ese directorio

```
mkdir tfg
cd tfg
```

Clonamos el repositorio que se encuentra en GitHub para pruebas de Alastria

```
git clone https://github.com/alastria/test-environment
```

Una vez descargado el repositorio, es importante eliminar del fichero bootstrap.sh la instalación de NPM ya que se producirá un error y nos saldrá un mensaje con avisándonos

que tenemos dependencias y paquetes rotos. Una vez modificado, ejecutamos el archivo de configuración ./bin/bootstrap.sh

```
cd test-environment/infrastructure/testnet
./bin/bootstrap.sh
```

Cuando se instalen todas las dependencias podemos continuar con el despliegue de la red.

En este fichero de Bootstrap podemos observar cómo se descarga el repositorio de Alastria-node para ejecutar la configuración de los nodos. El siguiente paso es levantar los nodos que queramos. En este caso vamos a crear una red con 2 nodos validadores: main y validator1 y tres nodos generales: general1, general2, general3. Para ello ejecutamos el siguiente comando: *./bin/start_network.sh clean <num-validators> <num-gws> -faulty_node <num_faulty>*

Hay dos posibilidades: la etiqueta clean lo que nos va a permitir es limpiar una configuración entera. Si queremos levantar una red existente usaremos la etiqueta restart. En este caso queremos levantar una red nueva desde cero.

```
./bin/start_network clean 2 3
```

Nos saldrá un mensaje de configuración tal como este:

```
[!!] Run this script from the directory test-environment/infrastructure/testnet/
[*] Cleaning previous environments
[!!] Run this script from the directory test-environment/infrastructure/testnet
[*] Preparing to clean the environment
[!!] Run this script from the directory test-environment/infrastructure/testnet/
[*] Spreading permissioned nodes config files
[*] Generating nodes in environment
[*] Starting validator nodes
[!!] Excecute from alastria test-environment/infrastructure/testnet/
[*] Starting main
Verify if /home/pilar/tfg/test-environment/infrastructure/testnet/logs/ have new
files.
[!!] Excecute from alastria test-environment/infrastructure/testnet/
[*] Starting validator1
Verify if /home/pilar/tfg/test-environment/infrastructure/testnet/logs/ have new
files.
```

```
null
[*] Starting gw nodes
[!!] Excecute from alastria test-environment/infrastructure/testnet/
[*] Starting general1
netcat: connect to localhost port 9001 (tcp) failed: Connection refused
netcat: connect to localhost port 9001 (tcp) failed: Connection refused
netcat: connect to localhost port 9001 (tcp) failed: Connection refused
Connection to localhost 9001 port [tcp/*] succeeded!
[*] constellation node at 9001 is now up.
Verify if /home/pilar/tfg/test-environment/infrastructure/testnet/logs/ have new
files.
[!!] Excecute from alastria test-environment/infrastructure/testnet/
[*] Starting general2
netcat: connect to localhost port 9002 (tcp) failed: Connection refused
netcat: connect to localhost port 9002 (tcp) failed: Connection refused
Connection to localhost 9002 port [tcp/*] succeeded!
[*] constellation node at 9002 is now up.
Verify if /home/pilar/tfg/test-environment/infrastructure/testnet/logs/ have new
files.
[!!] Excecute from alastria test-environment/infrastructure/testnet/
[*] Starting general3
netcat: connect to localhost port 9003 (tcp) failed: Connection refused
netcat: connect to localhost port 9003 (tcp) failed: Connection refused
Connection to localhost 9003 port [tcp/*] succeeded!
[*] constellation node at 9003 is now up.
Verify if /home/pilar/tfg/test-environment/infrastructure/testnet/logs/ have new
files.
```

Ahora, podemos monitorizar nuestra red a través del monitor que ofrece Alastria. Para ello abrimos una nueva ventana de línea de comandos y en la misma carpeta de test-environment/infrastructure/testnet ejecutamos el siguiente comando:

```
./bin/start_ethstats.sh
```

Una vez instalada la red ya podemos trabajar con nuestros smart contracts y nuestra configuración. Otros comandos importantes que nos van a servir serán los siguientes:

- Para parar la red usamos el siguiente comando: ./bin/stop_network.sh
- Para limpiar el ambiente : ./bin/clean_env.sh

Figura 24. Monitor Alastria para la testnet

En la imagen de arriba podemos ver cómo se visualizaría la red. Tenemos los nodos todos funcionando, los nodos que minan solo son los nodos validadores. El precio del gas es 0 puesto que en Alastria el gas tiene valor nulo. Se puede ver de un vistazo general que está ocurriendo en la red en tiempo real. Este será el mismo monitor que se usará para la red real de Alastria con muchos más datos.

7.3 CONFIGURAR CAKESHOP

Cakeshop es un paso más en la monitorización de nuestra red. Desarrollado por JPMorgan pensado para Quorum y para ir un paso más allá. No solo te permite visualizar lo que está pasando en la red a tiempo real si no que te permite interactuar con las cuentas, observar los bloques, mandar transacciones… Busca ser algo más que un mero monitor.

Lo primero que debemos hacer es crear una carpeta nueva que se llamará cakeshop en nuestro directorio desde el que estamos trabajando, en este caso el que creamos llamado tfg.

Descargamos el fichero de la página de releases de GitHub jpmorganchase/cakeshop. En este caso se ha elegido el de Linux ya que seguimos trabajando con el subsistema para Windows. Además, por comodidad se ha renombrado como cakeshop.war

```
mkdir cakeshop
cd cakeshop
wget -o cakeshop.war
https://github.com/jpmorganchase/cakeshop/releases/download/v0.10.0-
pre.2/cakeshop-0.10.0-x86_64-linux.war
```

Si no tenemos java configurado es necesario instalarlo en Linux. Basta escribir en la línea de comandos: java y comprobar si tenemos algo instalado. En caso de no estar configurado debemos instalarlo. Para eso usamos el paquete default-jre y escribimos el siguiente comando:

```
sudo apt-get install default-jre
```

Para una mayor comodidad a la hora de ejecutar Cakeshop renombramos el fichero a 'cakeshop.war'.Ahora ya podemos preceder a lanzar nuestro sistema.

```
mv cakeshop-0.10.0-x86_64-linux.war cakeshop.war
java java -jar cakeshop.war
```

Nota: Si el sistema operativo que usamos es Windows habrá que usar

```
java java -Dgeth.node=geth -jar cakeshop.war
```

A la hora de ejecutar Cakeshop se puede realizar de tres maneras diferentes. La primera será ejecutarlo por defecto, de manera que se crea un entorno para un nodo Ethereum normal en el que podemos realizar pruebas. La segunda forma nos va a permitir enlazar nuestro ambiente con uno de los nodos que forman nuestra red, este método se conoce como 'attach'. Este será el método que vamos a utilizar en nuestro proyecto. Y, por último, existe un tercer método que nos permite asociar nuestro Cakeshop a varios nodos a la vez.

Para configurar la herramienta en modo 'attach', aparecerá en el directorio una nueva carpeta llamada 'data'. En ella encontraremos dos nuevas carpetas 'geth' y 'local'. En concreto nos va a interesar un archivo de propiedades que se encuentra en la carpeta 'local' que se llama 'application.properties' que nos va a servir para configurar nuestro proyecto de manera que apunte a uno o varios de los nodos de nuestro proyecto. Debemos cambiar tres propiedades de nuestra herramienta y configurarlas de la siguiente manera:

```
geth.auto.start=false #Por defecto están en modo true
geth.auto.stop=false
geth.url=http\://127.0.0.1\:22001 #Dirección de nuestro nodo
```

Una vez modificado este fichero ya podemos volver a ejecutar el comando:

```
java -jar cakeshop.war
```

Y conectarnos a la dirección "localhost:8080/cakeshop"

Si salen estos errores:

- RPC request failed: invalid argument 1: json: cannot unmarshal hex string without 0x prefix into Go value of type hexutil.Bytes

  Después de mucho investigar parece que Cakeshop solo funciona bien con nodos exclusivamente pertenecientes a la *Main Net* de Ethereum, por lo que para redes privadas no nos va a aportar toda la información que promete la aplicación y no será el monitor que usemos.

7.4 INSTALAR UN NODO REGULAR

Configuración mínima recomendada:
- 4 CPU cores
- 32 Gb RAM
- 100 Gb SSD
- Red Hat >= 7.0

En este apartado, se va a instalar un nodo regular paso por paso. Se usará Docker para agilizar el proceso de creación, puesto que Alastria nos facilita ya el archivo. Lo primero que tenemos que hacer es clonar el repositorio de alastria-node en la carpeta en la que vamos a trabajar y ejecutar los archivos de configuración. [26]

Para ello introducimos los siguientes comandos:

```
$ git clone https://github.com/alastria/alastria-node.git
$ cd alastria-node/scripts/
$ ./bootstrap.sh
$ source ~/.bashrc
```

Ahora necesitamos generar la imagen de Docker, y para ello introducimos el siguiente comando:

```
sudo docker build -t alastria-node . --build-arg hostip=<host-ip> --build-arg
nodetype=<node-type> --build-arg nodename=<node-name>
```

En este caso vamos a usar nuestra dirección de localhost para nuestro nodo regular. El tipo de nodo que estamos instalando es general por lo que esto es lo que debemos indicar en <node-type> y el nombre del nodo será cualquiera que nos interese, en este caso se llamará test-tfg.

Una vez se haya creado la imagen podemos ejecutar el nodo, indicando que puertos vamos a necesitar abrir durante el proceso:

```
docker run -d --name alastria -p 9000:9000 -p 21000:21000 -p 22000:22000 -p
8443:8443 alastria-node
```

7.5 CONEXIÓN AL NODO REGULAR ALASTRIA DE LA UNIVERSIDAD

Para poder desplegar nuestros contratos en la red Alastria, debemos configurarlos de manera que apunten a un nodo regular. Como ya se ha mencionado existen dos tipos de nodos en Alastria y solo los regulares serán capaces de desplegar aplicaciones y contratos.

Para ello configuramos el archivo 'truffle.js' o 'truffle-config.js' de nuestro proyecto con los siguientes parámetros de la red:

```
module.exports = {
  // See <http://truffleframework.com/docs/advanced/configuration>
  // for more about customizing your Truffle configuration!
  networks: {
    alastria: {
       host: "130.206.64.6",
       port: 22000,
       network_id:"82584648528",
        gasPrice:0,
        gas:"0x2FEFD800",
    }
  }
};
```

De esta manera estamos imponiendo que el precio del gas sea 0, aunque Quorum por defecto fuerza a que sea 0 y estamos estableciendo la cantidad máxima de gas que puede usar el contrato. El ID de la red viene determinado por el de la red Alastria y la IP pública y el puerto son los parámetros a través de los cuales se conectará y se migrará nuestro contrato al nodo general de la universidad.

Ahora es necesario configurar nuestro Metamask y por ello usaremos la dirección IP y el puerto que hemos establecido en la configuración de nuestro contrato.

Nota: por defecto, y si no se indica lo contrario, Truffle migra el contrato a primera dirección del nodo, que se puede visualizar a través de la consola de geth mediante *eth.accounts[0]*. Al introducir el comando truffle migrate –network alastria, es probable que la cuenta esté

bloqueada y nos dé un mensaje de error que diga que se necesita una contraseña. Entonces será necesario utilizar el comando *personal.unlockAccount(DireccionCuenta)* también desde la consola, y a continuación introducir la contraseña de la cuenta. Una vez esté desbloqueada ya podremos migrar nuestro contrato al nodo sin problemas. En el ANEXO B – Comandos consola Geth, se explica detalladamente cómo acceder a la consola interactiva de Geth.

Capítulo 8. ANÁLISIS DE RESULTADOS

En este apartado se van a mostrar los resultados obtenidos a lo largo de todo el proyecto y el análisis que podemos sacar de ellos a través de las diferentes herramientas que se han empleado en el desarrollo.

8.1 APLICACIÓN DE GAMIFICACIÓN

En este apartado se analizará el caso de uso desarrollado en este proyecto. Para ello, usaremos la red de desarrollo de Ganache que nos proporciona Truffle. El motivo de usar esta red es la interfaz gráfica que nos va a permitir analizar las transacciones y ver de una manera más detalla qué es lo que está ocurriendo en nuestra transacción.

En primer lugar, para realizar una prueba vamos a crear 2 cuentas: empresa y empleado

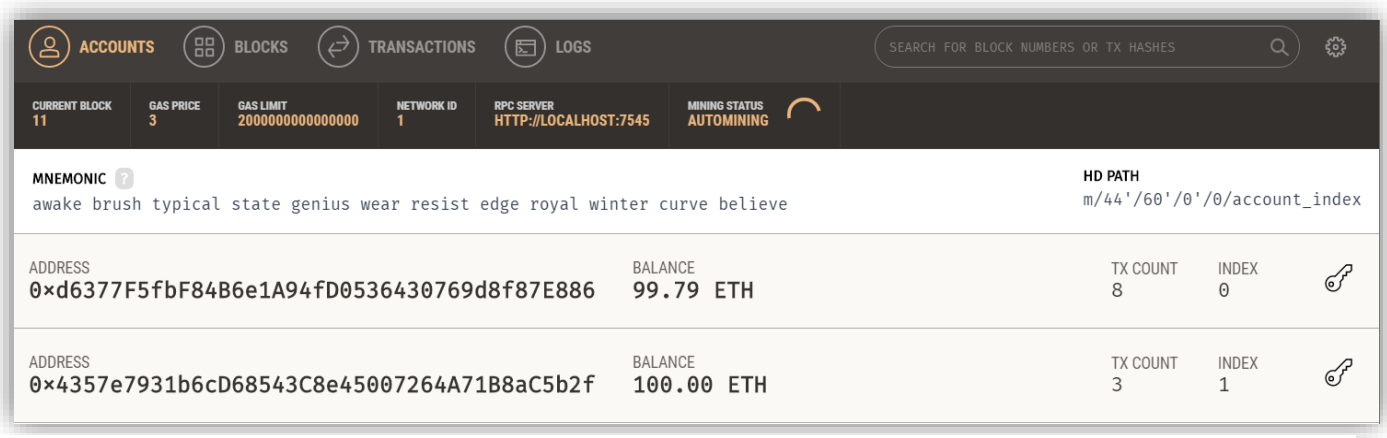


Figura 25. Pantalla principal de Ganache

La primera será la cuenta de la empresa y además será por defecto la cuenta a la que se migra nuestra aplicación con los contratos. La segunda cuenta será la que esté asociada con uno de los empleados. Ganache no permite que el precio del gas sea 0 por lo que como se puede ver en la imagen el precio sea igual a 3, lo que explica porque hemos consumido gas por la cuenta 1.

En la imagen también podemos apreciar que hay un límite de gas por transacción y que el ID de la red será 1. En Alastria tendremos un ID diferente que es único de cada red Ethereum. El servidor RPC que nos indica será la dirección IP que tendremos que poner en Metamask para poder interactuar con el contrato.

Si nos fijamos bien encontramos una serie de 12 palabras bajo el nombre de 'Mnemonic'. El mnemonic es la forma de acceder a nuestras cuentas desde Metamask en otro navegador diferente al que creamos, de manera que si tenemos esas 12 palabras podemos llevarnos nuestra cartera a cualquier parte y podemos importar nuestras cuentas siempre que tengamos accesible nuestra clave privada o el fichero json.

Por ejemplo en este caso la llave privada de nuestra cuenta de empleado sería esta:

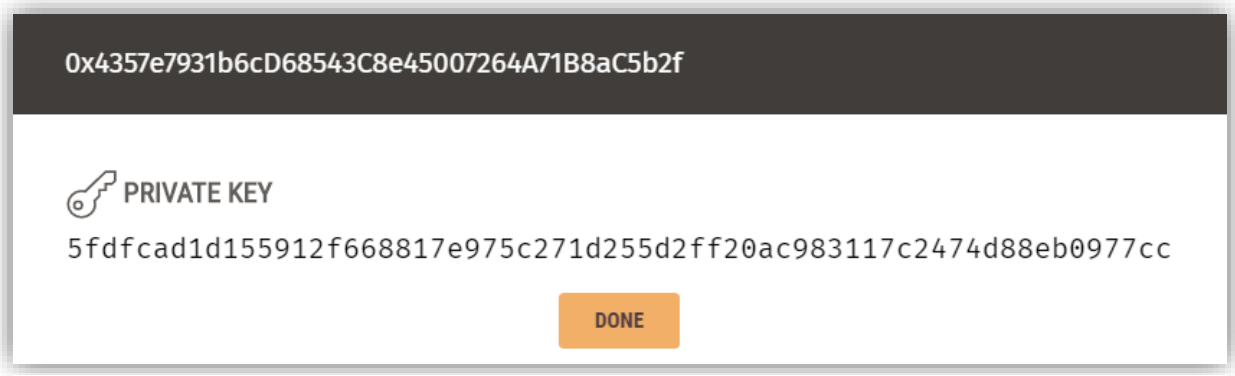


Figura 26. Clave privada de cuenta de Ganache

Además, se presenta cómo han quedado las pantallas de la aplicación, puesto que, aunque el resultado estético no es uno de los objetivos del proyecto, es importante que la interacción sea fácil e intuitiva para generar una buena experiencia en el usuario.

Si nos paramos un momento a mirar las transacciones en Ganache veremos cómo se han creado los contratos y la información de cada operación que nos sirve para saber si nuestra aplicación se está comportando correctamente.

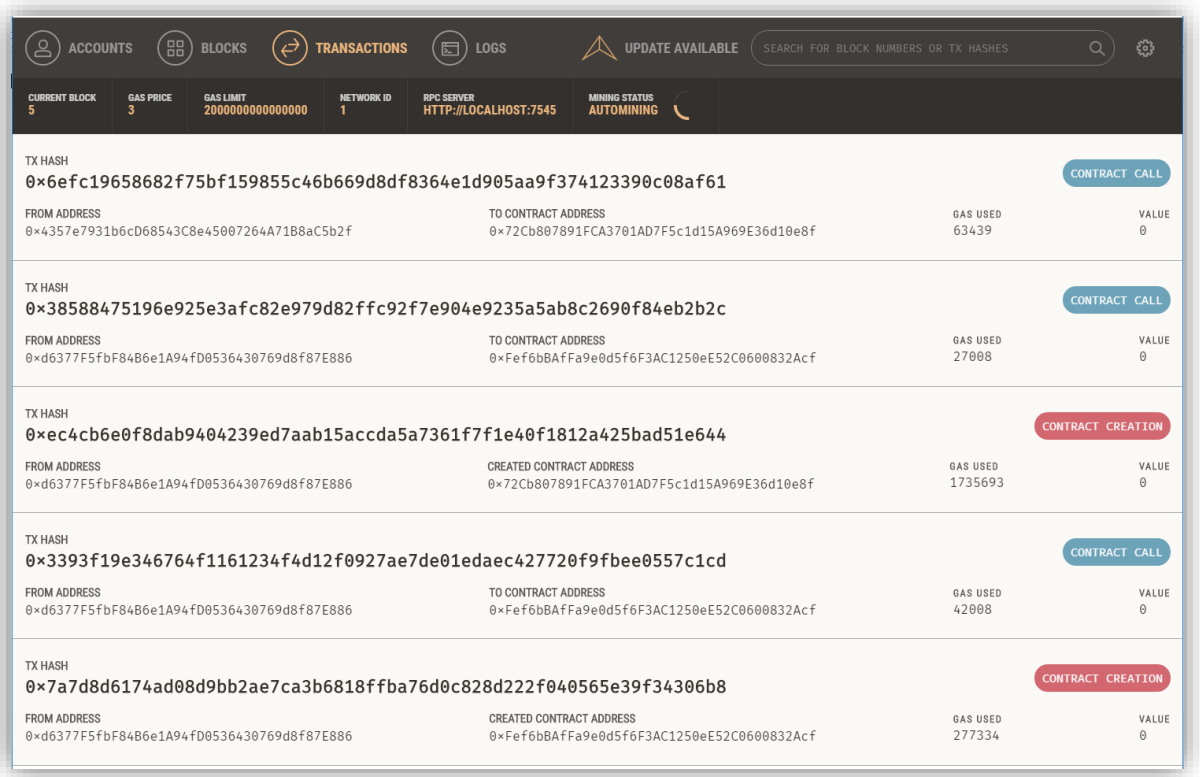


Figura 27. Transacciones con Ganache

Ahora ya podemos empezar a describir el flujo normal de la aplicación que hemos desarrollado sabiendo que podemos ir trazando la actividad del proyecto a través de Ganache, ya que todas las transacciones estarán recogidas.

Recordamos que esté proyecto tiene dos actores: la empresa y los empleados. Antes de empezar y que pueda entrar en acción el empleado, es la empresa desde su cuenta la encargada de crear los tokens, que se han llamado TFGs. Al iniciar la aplicación se encontrarán con la siguiente página:

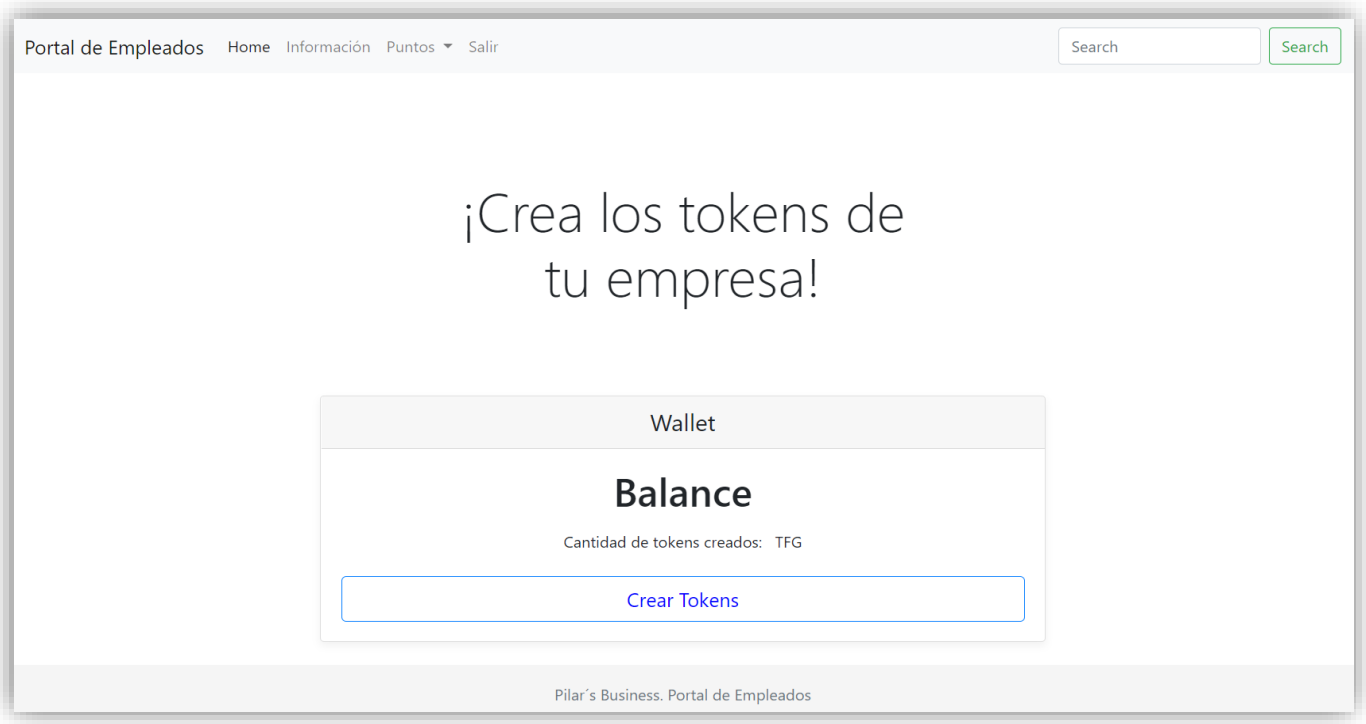


Figura 29. Pantalla de Crear Tokens

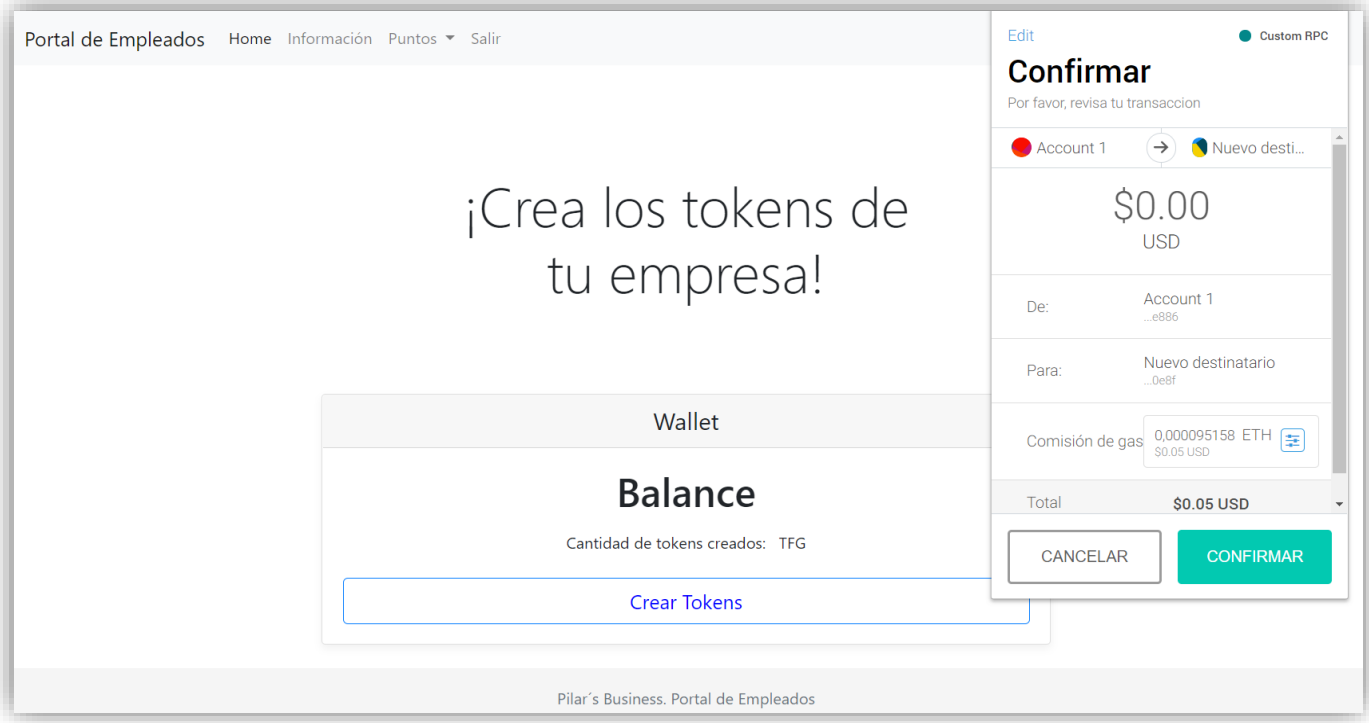


Figura 28. Confirmación Metmask para Crear Tokens

El siguiente paso sería pulsar el botón de crear Tokens y automáticamente nos saldrá la ventana de confirmación de Metamask en la que podremos confirmar que queremos realizar la operación.

Después de confirmar la transacción, esperando unos segundos, nos saldrá el valor de tokens creados que automáticamente se mandan a la cuenta de la empresa y serán el equivalente a 100000 TFGs. Estos serán los tokens que usarán los empleados y que se transferirán cuando reclamen sus objetivos. Es posible que tarde un poco de tiempo en confirmar el número de tokens creados ya que la operación debe ser minada y actualizada.

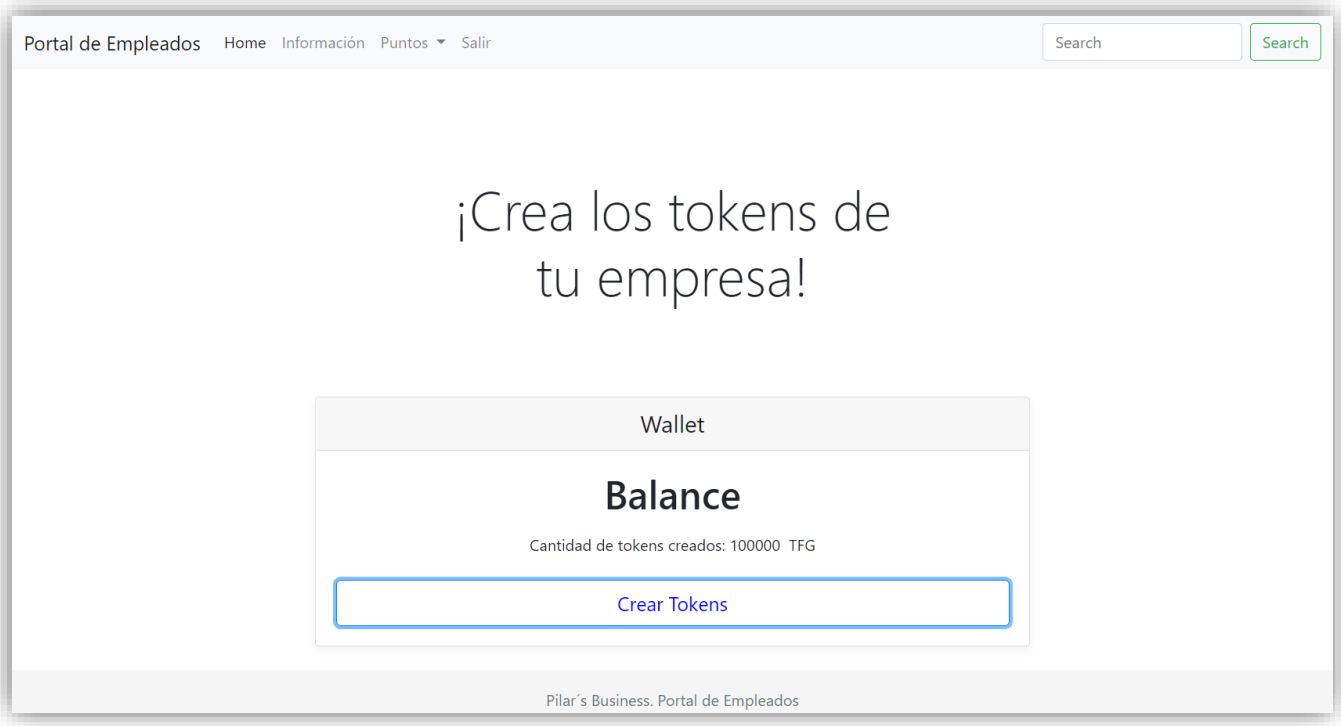


Figura 30: Pantalla con los tokens creados

Ahora, nos ponemos en la piel del empleado. En primer lugar, cuando se conecte a la aplicación y se autentique se encontrará con una página con toda la información de su perfil empresarial incluida una foto. Lo primero que debe acceder es a su cuenta de Metmask para poder gestionar los tokens.

Para configurar Metamask el empleado tendrá que utilizar el Mnemonic o la semilla de la cartera de su cuenta de Metamask y la contraseña. Para conectarnos a la red de la empresa tendremos que añadir una nueva conexión RPC con la dirección correcta, en este caso es 127.0.0.1 (localhost) y el puerto 7545. Guardamos la configuración y se conectará a la red que aparecerá como privada.

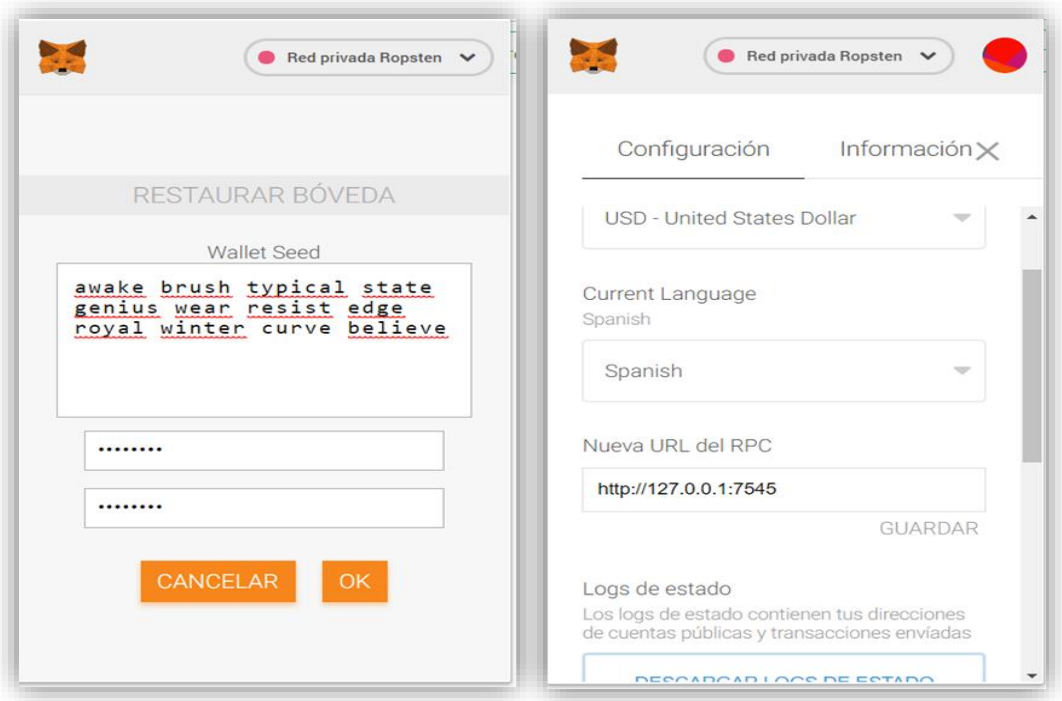


Figura 31. Configuración de la cuenta de Metamask

Figura 32. Página de información del empleado

A continuación, el empleado podría pasar a ver si ha conseguido alguno de los retos que se le habían planteado y ver cuáles le quedan por completar. Además, en el caso de que tenga puntos se mostrará su saldo actual y al darle a reclamar sus puntos se sumarán a su balance. Por ejemplo, este empleado ya tiene un saldo de 1000 TFGs y puede reclamar dos objetivos por lo que puede sumar aún más puntos a su balance actual.

Figura 33. Página de reclamar puntos

En este momento el empleado ya ha reclamado sus puntos 700 en este caso que se se sumarán al balance inicial que era 300 y los botones han pasado a estar deshabilitados puesto que los retos ya se han canjeado. Podrá ver los retos que le faltan por completar si quiere conseguir más puntos.

Con los puntos que se han acumulado, el empleado puede irse a la página de canjear puntos en el menú superior en el apartado de Puntos. En la barra superior en la derecha podemos ver la cantidad de puntos que el empleado posee en este momento, podrá gastar solo aquellos que posea. Cuando pulse el botón canjear le saldrá un mensaje de Metamask pidiendo que confirme la transacción y se le restarán los puntos canjeados. Los puntos tardarán en actualizarse unos segundos debido a que las transacciones deben confirmarse dentro de la red.

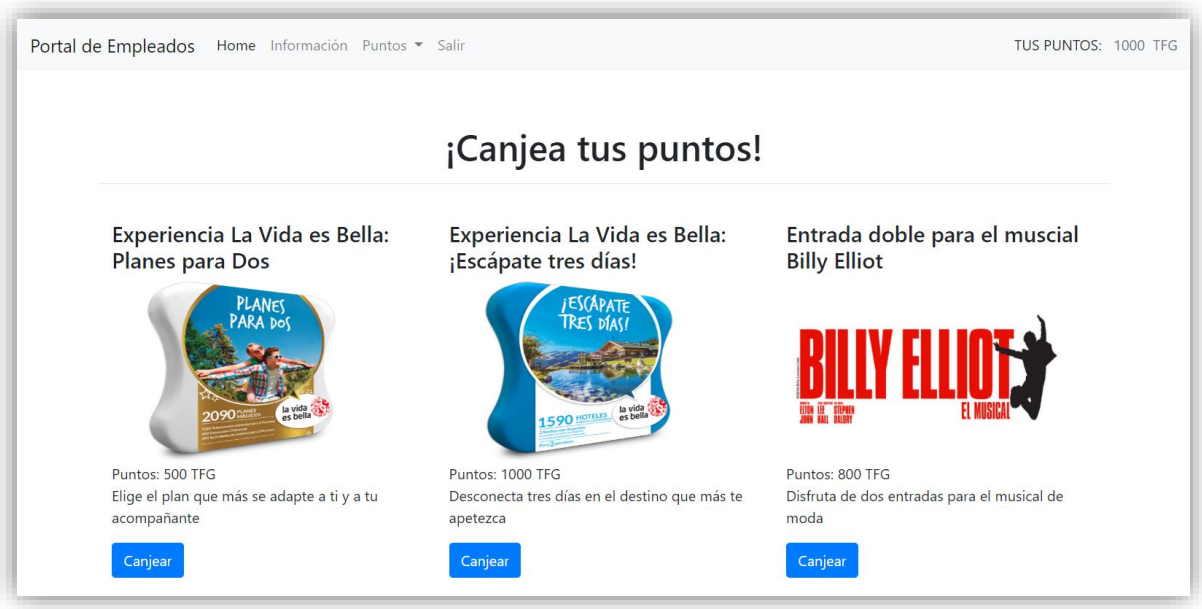


Figura 34. Página de canjear puntos

Cuando canjee una de las recompensas, saldrá un mensaje avisando al empleado de que toda la información sobre su regalo se ha enviado a su correo empresarial. Una vez acabado este ciclo, el empleado deberá esforzarse por conseguir más objetivos que le permitan conseguir puntos para intercambiar.

8.2 CREAR Y GESTIONAR TOKENS

Los tokens que hemos creado para nuestra aplicación se basan en el estándar ERC20 para poder usarlos con carteras y cuentas. Para probar que el código funcionaba a la perfección sin ningún fallo inesperado se utilizó Remix un IDE online de desarrollo de Smart Contracts.

Para probar una nueva red de prueba vamos a desplegar el contrato en la red de pruebas de Ethereum, Ropsten, que nos permite añadir Ether a través del Faucet de Metamask.

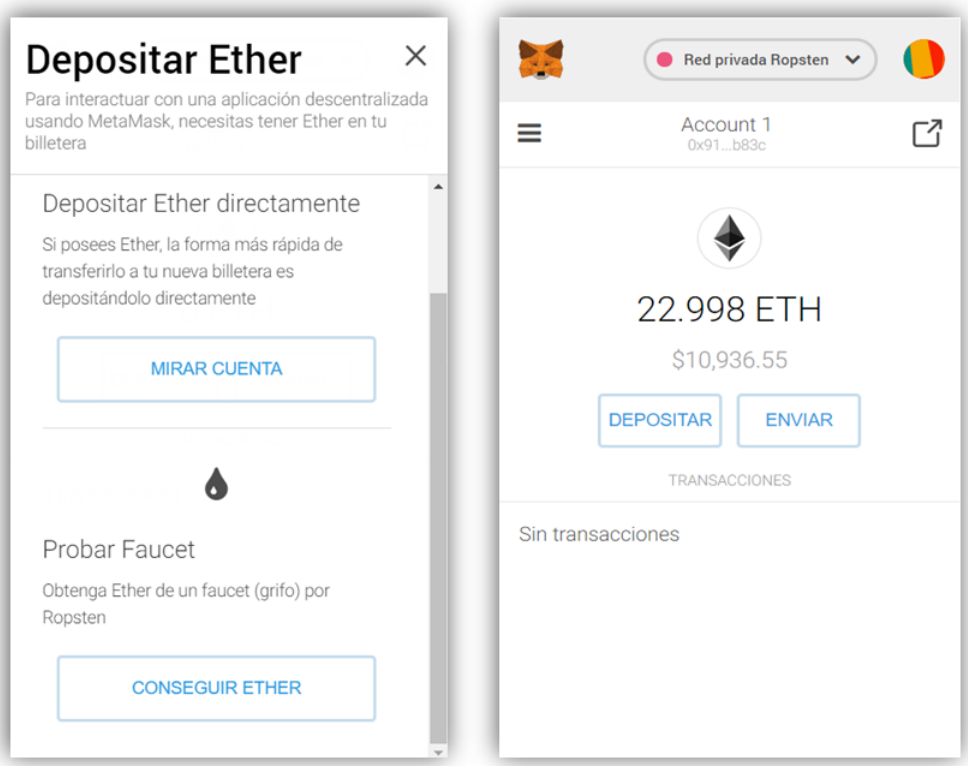


Figura 35. Añadir ether con el Faucet de Metamask

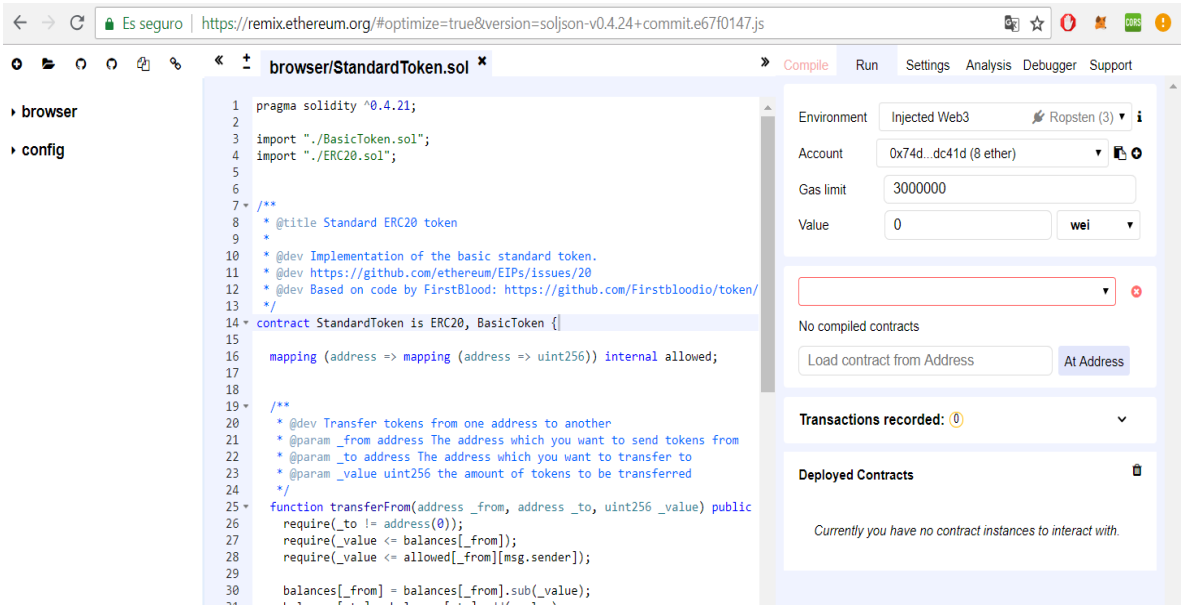


Figura 36. Remix, IDE para Ethereum

Ahora ya es el momento de desplegar nuestro contrato en la red. Desde Remix elegimos el contrato y pulsamos el botón de Run. El contrato puede tardar en crearse unos minutos, hay que recordar que estamos en una red blockchain en la que hay que minar para aceptar las transacción. Si ha sido migrado a la red con éxito entonces podremos ver nuestra transacción en Etherscan.

Podemos ver que el contrato se ha creado con éxito y nos da una serie de datos que pueden resultarnos muy interesantes para hacernos una idea del coste y el gasto.

Por ejemplo, en nuestro contrato hemos tenido un límite de gas de 1287206 y un precio de gas de 1 wei, el equivalente a 0.000000001 ether. Podemos ver el hash de la transacción y datos de la transacción y del bloque, como la fecha y la cuenta desde la que hemos lanzado el contrato. Una imagen de la información se puede ver en la Figura 37. Creación de Tokens ERC20 en Ropsten

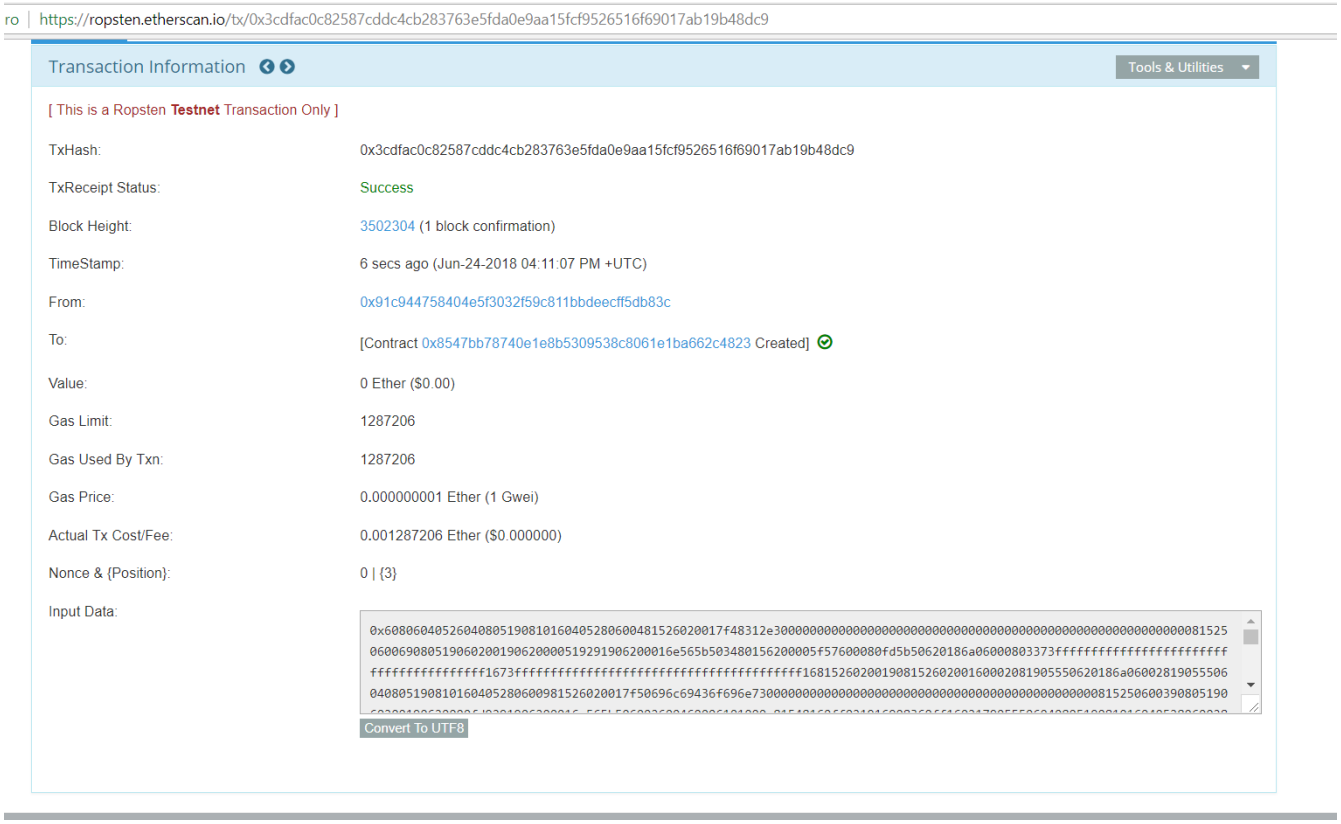


Figura 37. Creación de Tokens ERC20 en Ropsten

Una muy buena práctica es verificar nuestro código para que los usuarios sepan que no estamos haciendo ninguna actividad ilícita y generar más confianza para nuestros tokens. Esto no quiere decir que nuestro token no vaya a funcionar si no se verifica, pero nunca está de más y demostramos que el código es seguro.

Agregamos los tokens a nuestra cuenta desde Metamask, desde el apartado de agregar token. Solo necesitamos la dirección del contrato que acabamos de desplegar y ya podemos enviarlos de una cuenta a otra como en la siguiente imagen;

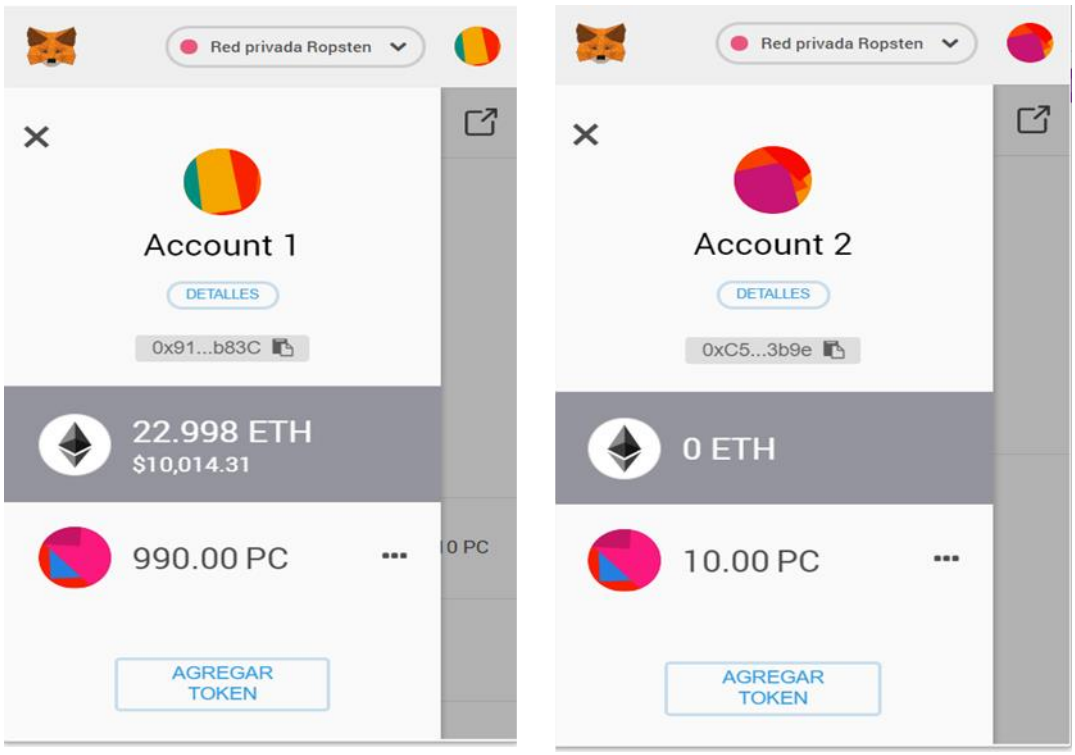


Figura 38. Intercambio de tokens entre cuentas con Metamask

8.3 RED ALASTRIA

La red Alastria es una red basada en Ethereum y Quorum y todavía está en fase de desarrollo y pruebas. En Alastria los nodos regulares o generales son los que pueden desplegar aplicaciones o dapps. Es por eso que tanto en la red tesnet de prueba que Alastria nos ofrece para hacer pruebas en local como en la red Alastria vamos a tener que lanzar nuestros contratos con truffle apuntando a un nodo general. En el caso de la red de prueba se ha escogido el nodo general 1 que trabaja en la dirección IP: 127.0.0.1:22001.

La red de prueba está configurada con los mismos parámetros que la red original, pero con la diferencia que en la red de prueba podemos modificar las configuraciones y hacer distintas pruebas con las cuentas.

Si queremos conectarnos a la red real tendremos que usar una IP pública en concreto **130.206.64.6** al puerto **22000** (que tendremos que tener habilitado) con ID**: 82584648528** y conectarnos por medio de SSH y una VPN para el acceso remoto. Es importante conectarse al nodo mediante conexión HTTP, de otra manera no funcionará puesto que no soporta la conexión HTTPS. También es conveniente tener activada la extensión de CORS para el navegador. CORS es una herramienta de Recursos de Origen Cruzado y utiliza encabezados adicionales para acceder a recursos desde un servidor.

También tenemos que hacer que el nodo sea capaz de entenderse con Metamask y vamos a añadir a la configuración del nodo algunas modificaciones para que cuando el nodo general arranque, sea capaz de detectar a nuestro proveedor de Web3.

En el archivo start.sh de la carpeta scripts de nuestro nodo alastria modificamos a la variable GLOBAL_ARGS que tiene este aspecto:

```
GLOBAL_ARGS="--networkid $NETID --identity $IDENTITY --permissioned --rpc --
rpcaddr 0.0.0.0 --rpcapi
admin,db,eth,debug,miner,net,shh,txpool,personal,web3,quorum,istanbul --rpcport
22000 --port 21000 --istanbul.requesttimeout 30000  --ethstats
$IDENTITY:bb98a0b6442386d0cdf8a31b267892c1@52.56.86.239:3000 --verbosity 3 --
vmdebug --emitcheckpoints --targetgaslimit 18446744073709551615 --syncmode full "
```

Añadimos la siguiente línea:

```
geth --rpc --rpccorsdomain="chrome-extension://nkbihfbeogaeaoehlefnkodbefgpgknn"
```

Ya estaría listo nuestro nodo para poder interactuar con Metamask. Ahora tendremos que configurar nuestro contrato con los parámetros adecuados. Para evitar un problema en el que la migración de los contratos exceda del límite de gas vamos a poner el valor de **20.000.000** que parece que es un número adecuado para la red según los desarrolladores de Alastria. Además, se indicará el valor del gas a 0, para que no haya confusión.

A la hora de desplegar contratos tenemos que desbloquear la cuenta sobre la que se va a migrar el contrato puesto que si la cuenta está bloqueada nos saldrá un mensaje que nos avise que no podemos lanzar el contrato, que se necesita una contraseña. A través de la consola geth introducimos la contraseña y ya no habría problema.

Se ha conseguido lanzar los contratos a la red de prueba y al nodo original, el problema es la conexión con Metamask puesto que se lee bien la información de las cuentas desde el contrato, pero cuando necesitas confirmar la transacción se queda suspendido indefinidamente y no responde. Este problema parece ser que se debe a un problema que tiene Metamask con los contratos cuyo precio del gas es 0, por lo que al ser ajeno a la red no podemos solucionarlo de manera inmediata.

Otro de los problemas con los que nos hemos encontrado es la herencia del gas que recibe Alastria por Ethereum. El gas en Alastria no afecta pues tiene un coste 0, pero en Ethereum da un problema relacionado también con el límite de gas que debemos especificar en el contrato.

La actividad de la red de Alastria tanto real como de pruebas se puede ver desde un monitor que levantamos desde línea de comandos. En este monitor podemos ver todos los nodos que tenemos, el tiempo de propagación, los pares de cada nodo y la información más relevante de la red. Para poder acceder al monitor real de la red debemos conectarnos a la siguiente dirección IP: **52.56.86.239:3000**

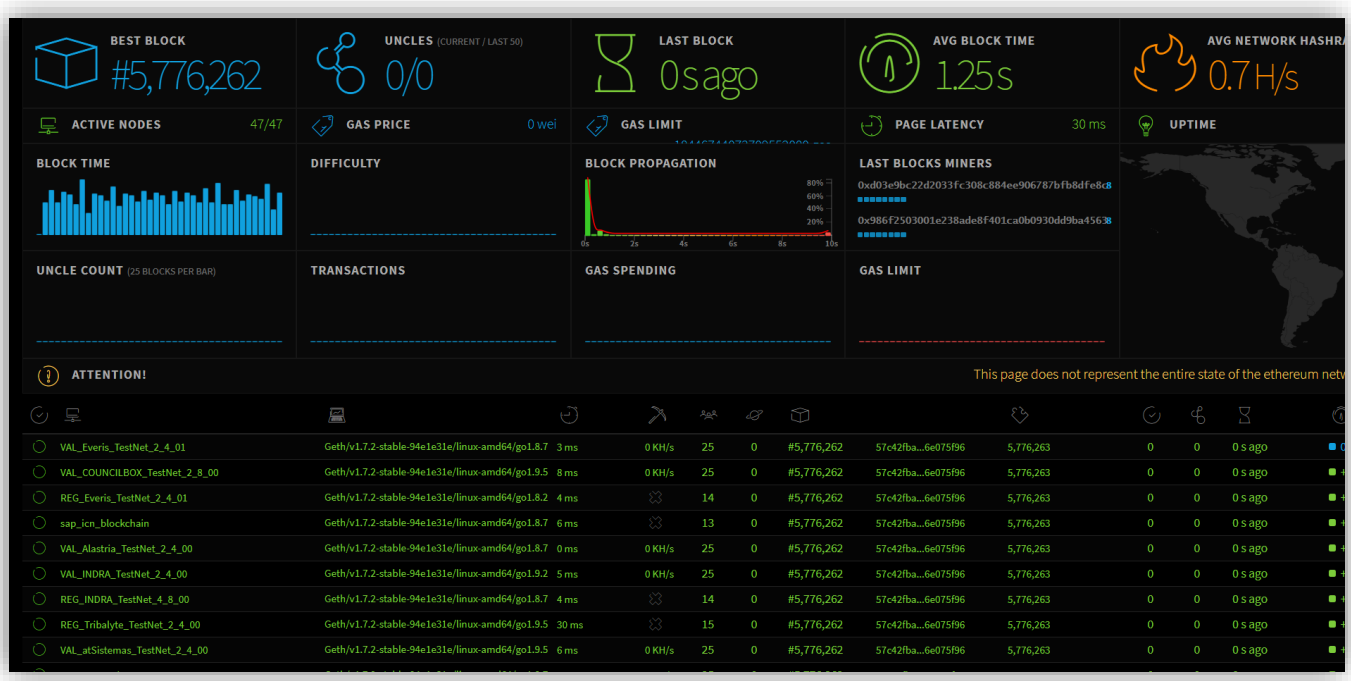


Figura 39. Monitor Alastria para la red real

Aunque el monitor sea el mismo que para la red de prueba que se montó para realizar pruebas, observamos diferencias significativas como el número de nodos conectados. La red Alastria tiene aproximadamente 47 nodos conectados, generales y validadores. Los nodos pertenecen a las empresas socias del consorcio. Concretamente podemos encontrar el nodo que tenemos en la universidad

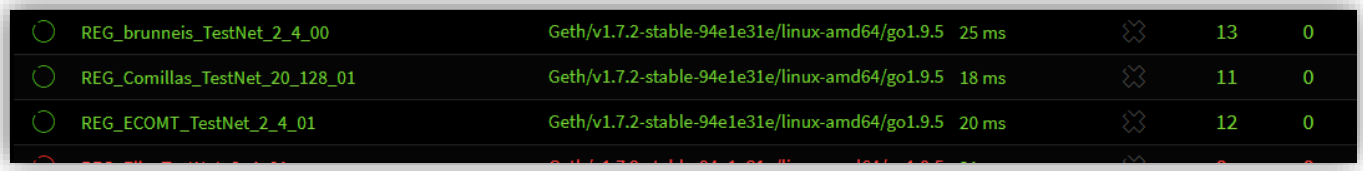


Figura 40. Nodo regular de la universidad

En el momento en el que se hizo esta captura, el nodo estaba conectado a otros 11 pares.

Las transacciones de la red las podremos ver desde el explorador Quorum para Alastria

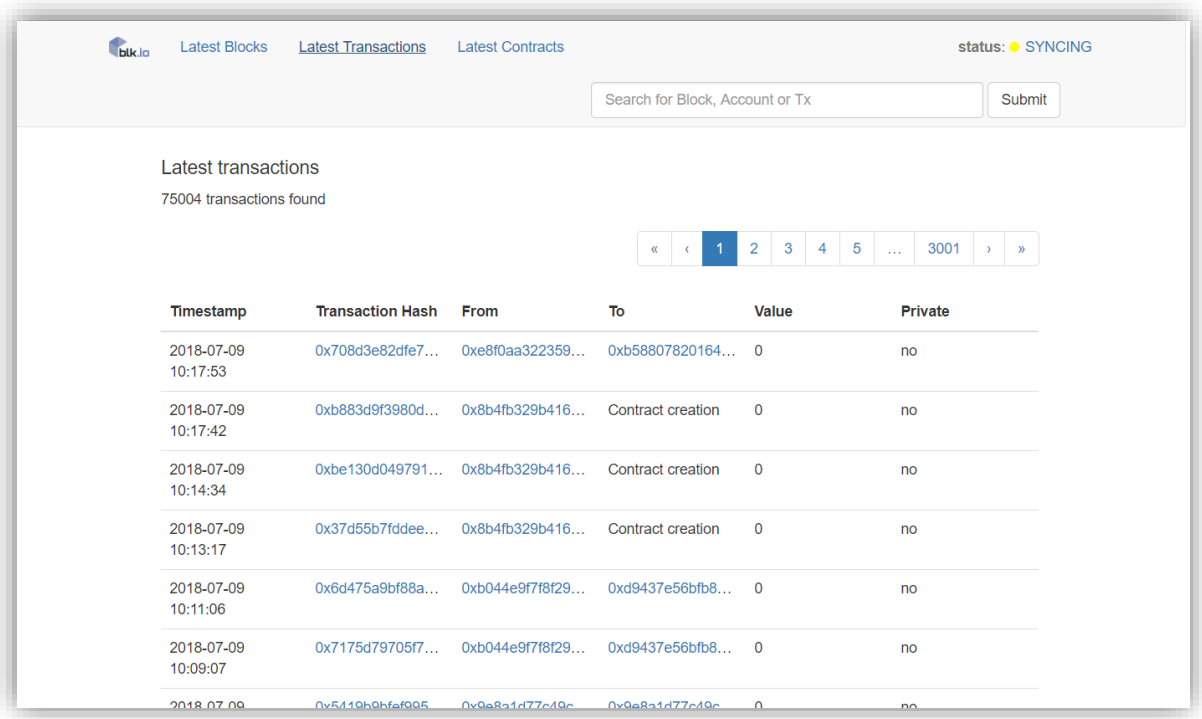


Figura 41. Visualizador de bloques de Quorum para Alastria

Y desde aquí podemos obtener más información de las transacciones que se están realizando en la red con la información parecida a lo que podríamos obtener en otro visualizador como Etherscan. Como podemos ver nos enseña el tiempo en el que fue confirmada la transacción, el hash, así como la dirección desde la que se envía, la dirección de envío y el valor. Además, nos indica si la transacción ha sido privada, ya que como hemos visto Quorum permite transacciones privadas entre nodos a través de Constellation.

Si pinchamos en una de las transacciones encontramos la información propia de la transacción, como el gas consumido, el coste de la transacción o el nonce que se ha calculado para minar y confirmar el bloque con esa transacción. La trazabilidad de las transacciones

es total para las transacciones no privadas, y con el número de transacción podremos localizarla fácilmente en este monitor de transacciones.

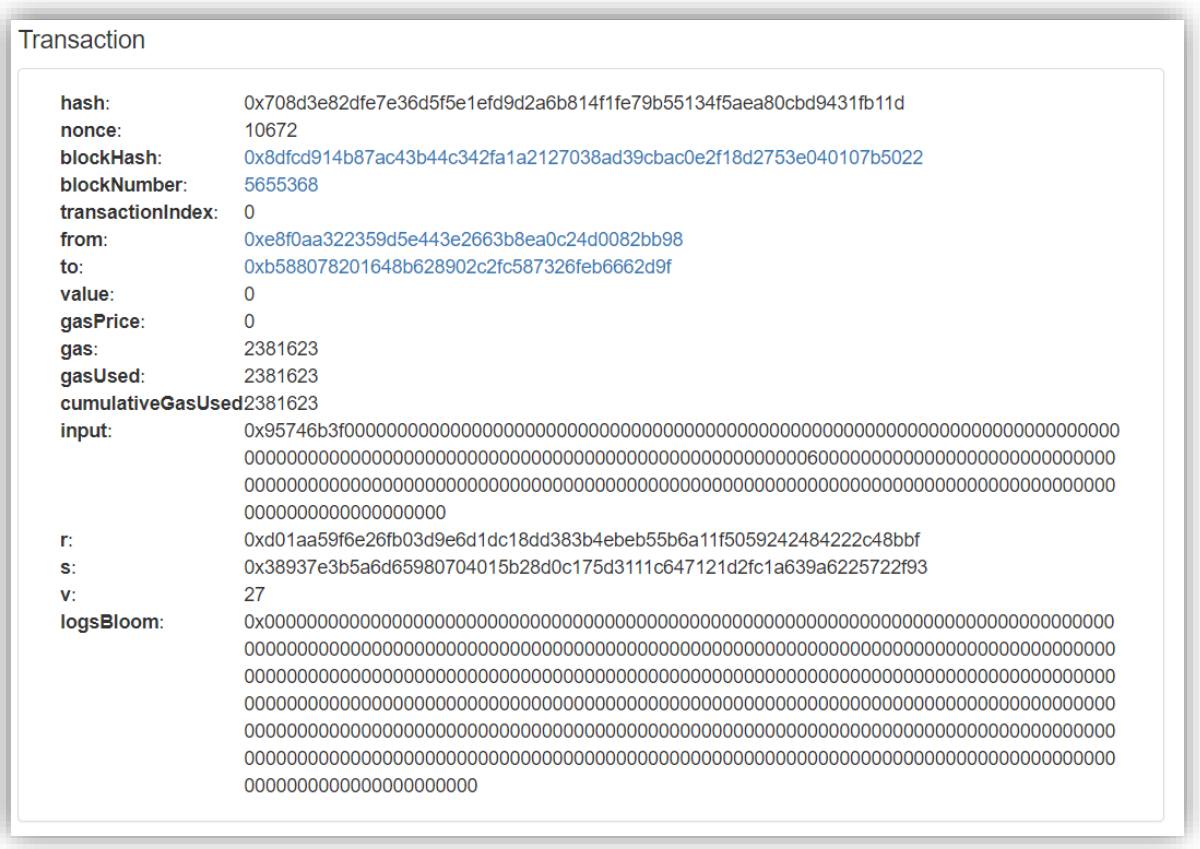


Figura 42. Información de una transacción completa en el visualizador de Alastria

Como conclusión, aunque la red Alastria se encuentre en fase de desarrollo y pruebas ya nos deja ver en qué se va a convertir y en gran gran potencial para comunicar a las empresas de diferentes sectores y trabajar en común por conocer e implementar una tecnología en España que esté probada.

Capítulo 9. CONCLUSIONES Y TRABAJOS FUTUROS

La popularidad de Blockchain aumenta considerablemente día a día y parece ser una tecnología que parece dispuesta a quedarse y a ser una gran revolución en el mundo empresarial.

La finalidad de este proyecto era adentrase en el conocimiento de esta tecnología y conseguir una visión más específica de las posibilidades que aporta al momento actual en el que nos encontramos y a un futuro muy próximo.

En cuanto al estudio de la tecnología se ha estudiado desde conceptos más generales como la definición de Blockchain y su funcionamiento, hasta adentrarnos en el diseño de Quorum y sus algoritmos y protocolos más concretos que marcan la diferencia respecto al resto de plataformas de la tecnología de cadena de bloques. De esta manera se ha conseguido entender de una manera más completa los procesos que iban a soportar el siguiente paso: la implementación de una red de pruebas para probar las comunicaciones y tener una infraestructura preparada para desplegar los contratos.

Se han desplegado varias redes a lo largo de este trabajo. Para empezar, con la herramienta Truffle por defecto puedes realizar tus pruebas en varias redes de desarrollo: Ganache o la propia que lleva integrada Truffle llamada 'development'. Ganache tiene una interfaz muy visual que permite monitorizar las transacciones, bloques y cuentas, de manera que para pruebas iniciales y saber qué está pasando en nuestros contratos es realmente útil. Esta es la primera red desplegada que nos ha permitido probar primero los contratos antes de llevarlos a otras redes. Además, para entender su funcionamiento y ver como interactuaban los nodos se ha desplegado la red de ejemplo de los 7 nodos de Quorum. Para poder probar nuestro código también lanzamos un contrato de tokens en la red de prueba de Ethereum, Ropsten, para además aprender a lanzar contratos a la *Main Net* de Ethereum. Finalmente, la red que hemos implementado ha sido la tesnet de Alastria con 2 nodos validadores y 3 generales para

probar todo definitivamente antes de lanzarlo al entorno real de la red Alastria, nuestro objetivo final.

Una vez se tuvo un conocimiento más amplio de las herramientas Truffle y del lenguaje de programación Solidity así como de las comunicaciones entre el navegador y los contratos, se pasó a la fase de desarrollo de un caso de uso que pudiera demostrar todo lo aprendido hasta este momento. Se escogió una aplicación simple y concisa la cual nos enseña cómo se puede combinar las necesidades de una empresa con la innovación. La aplicación, pensada para ejercer de refuerzo positivo a los trabajadores con los conceptos de gamificación, cumple con los objetivos de mostrar cómo la empresa genera los tokens y cómo en este caso van a funcionar como moneda de cambio cuando se vayan cumpliendo los objetivos de los empleados de la empresa para que se puedan intercambiar por recompensas al buen trabajo. Siendo una aplicación con una funcionalidad simple y moderada ya nos da una idea de todas las posibilidades que los contratos inteligentes y las redes blockchain pueden aportar a la industria y a las empresas. Lo más complicado fue encontrar las proporciones adecuadas de límite de gas y la configuración más apropiada para interactuar con el contrato puesto que cada red funciona con sus propios parámetros.

Por último, la migración a la red Alastria ha sido más complicada que lo que parecía en un principio debido a la configuración propia de la red. En las primeras pruebas conseguimos leer los contratos que habíamos desplegado, pero la aplicación de Metamask no era capaz de confirmar las transacciones que estábamos pidiendo realizar. Las aplicaciones y herramientas que se usan en este tipo de proyectos siguen en fase de desarrollo, de manera que avanzan en paralelo con las plataformas blockchain y se deben adaptar a los cambios constantes en cuanto a protocolos, seguridad y rendimiento. Gracias a proyectos como Alastria, formado por equipos multidisciplinares y de diferentes sectores, se hacen grandes avances a diario en materia de eficiencia y seguridad con una gran comunidad de desarrolladores. A Blockchain le quedan unos años de madurez para su total implementación, pero ya nos permite ser conscientes de las ventajas y beneficios que puede ofrecer.

9.1 TRABAJOS FUTUROS

Al ritmo que avanza la tecnología y el desarrollo de plataformas y aplicaciones es probable que en unos meses todo lo escrito en este trabajo haya sido modificado y aunque la base sea la misma, serán nuevos algoritmos de consenso, algoritmos criptográficos o nuevas redes privadas que se puedan implementar en nuestro desarrollo. De manera que esta es una tecnología que requiere una actualización constante para estar al día de las novedades y no quedarse atrás rápidamente.

Como trabajo futuro se podría implementar un monitor más específico que nos permita ver exactamente las transacciones una a una con la información intrínseca, independientemente de qué red se esté usando puesto que actualmente son muy concretas y en el caso de Alastria el monitor es demasiado general y nos termina de aportar toda la información que por ejemplo Ganache nos muestra para su red de pruebas.

En cuanto al caso de uso, a medida que se avance en soltura con el lenguaje de programación Solidity se podrá añadir lógica al contrato de manera que las interacciones del usuario con la aplicación sean más ricas e implementen mayor funcionalidad como la posibilidad de autenticación o plantear objetivos compartidos.

Aunque Web3.js es la librería que se usa ahora para evitar tener que realizar las tediosas conexiones JSON-RPC de forma manual dentro de no mucho estará en desuso por lo que sería interesante investigar en nuevas librerías que nos permitan realizar funciones similares a las que aporta esta librería actualmente.

A la vez, sería interesante investigar nuevas redes privadas y compararlas en seguridad, y rendimiento con la red Alastria actual para ver qué aspectos son los que se pueden mejorar y observar la dirección que las empresas están tomando para implantar esta tecnología en nuestro día a día.

Capítulo 10. BIBLIOGRAFÍA

[1] ¿Qué es la cadena de bloques?, Blog Bit2me: https://blog.bit2me.com/es/que-es-cadena-de-bloques-blockchain/

[2] Del PoW al BFT: ¿Cuáles son los algoritmos para lograr el consenso?, InsiderPro: https://es.insider.pro/tutorials/2018-03-21/del-pow-al-bft-cuales-son-los-algoritmos-para-lograr-el-consenso/

[3] Consenso, Libro Blockchain: http://libroblockchain.com/consenso/

[4] Los diferentes tipos de consenso del Blockchain, Karl Booklover: https://www.karlbooklover.com/consensos-del-blockchain

[5] Mining, Ethereum Homestead: http://www.ethdocs.org/en/latest/mining.html

[6] Bitcoin VS Ethereum, Mi Ethereum: https://miethereum.com/ether/bitcoin-vs-ethereum/#toc13

[7] Ethereum Homestead: http://www.ethdocs.org/en/latest/

[8] Algoritmo Casper para Ethereum : https://es.cointelegraph.com/news/casper-what-is-known-about-the-new-ethereums-network-upgrade

[9] Clients Ethereum, Ethereum Homestead: http://www.ethdocs.org/en/latest/ethereum-clients/choosing-a-client.html

[10] Ether, Ethereum Homestead: http://www.ethdocs.org/en/latest/ether.html

[11] El gas en Ethereum: https://miethereum.com/ether/gas/#toc1

[12] Quorum Whitepaper, GitHub- JP Morgan: https://github.com/jpmorganchase/quorum-docs/blob/master/Quorum%20Whitepaper%20v0.1.pdf

[13] Raft-based consensus for Ethereum/Quorum, GitHub-Qorum: https://github.com/jpmorganchase/quorum/blob/master/raft/doc.md

[14] Istanbul Byzantine Fault Tolerance, GitHub- Ethereum: https://github.com/ethereum/EIPs/issues/650

[15] Transacciones Quorum, GitHub- JP Morgan: https://github.com/jpmorganchase/quorum-docs/raw/master/images/QuorumTransactionProcessing.JPG

[16] Arquitectura Quorum, GitHub: https://github.com/jpmorganchase/quorum-docs/blob/master/Quorum_Architecture_20171016.pdf

[17] Metamask, Criptonoticias: https://www.criptonoticias.com/aplicaciones/metamask-puente-ethereum-navegador/

[18] Observatorio Blockchain Sngular, Informe de noviembre de 2017. https://sngular.team/wp-content/uploads/2018/01/observatorio-blockchain-sngular.pdf

[19] Primer nodo universitario, El Economista: http://www.eleconomista.es/ecoaula/noticias/9204708/06/18/La-Universidad-Pontificia-Comillas-presenta-el-primer-nodo-universitario-blockchain.html

[20] What is Blockchain, Blockchain Hub: https://blockchainhub.net/blockchain-intro/

[21] Peer-to-peer, Wikipedia: https://en.wikipedia.org/wiki/Peer-to-peer

[22] Puzzles Hash, 3583 Bytes Free, Ready: https://3583bytesready.net/2016/09/06/hash-puzzes-proofs-work-bitcoin/

[23] Merkle Tree, MiEthereum: https://miethereum.com/mineria/#toggle-id-5

[24] Teoría de juegos, Steemit, Partes I-IV https://steemit.com/cryptocurrency/@juanfb/blockchain-y-la-teoria-de-juegos-parte-vi

[25] Consola JavaScript API y Web3, GitHub- Ethereum: https://github.com/ethereum/wiki/wiki/JavaScript-API#web3ethcoinbase

[26] Nodo Alastria, GitHub: https://github.com/alastria/alastria-node

[27] Alastria Test-Enviornment, GitHub: https://github.com/alastria/test-environment

[28] Configurar una red privada, Geth_faulty_nodes, GitHub: https://github.com/alastria/geth_faulty_nodes

ANEXO A – MIGRAR DAPP A LA RED ETHEREUM

En este tutorial se va a usar Truffle, que es el ambiente de desarrollo para Ethereum. La idea es migrar una aplicación existente al entorno Ethereum, en este caso va a ser migrada a una de las redes de test de Ethereum.

1. Instalar HDWalletProvider

```
npm install truffle-hdwallet-provider
```

2. Registro en Infura

Ahora, es necesario registrarse en Infura, para ello se necesita un token de acceso. El token te será enviado al correo electrónico, es importante mantener el token guardado y privado.

3. Configurar el proyecto en Truffle

Ahora es necesario configurar nuestro proyecto en Truffle. Debemos editar el fichero 'truffle.js' para poder usar HDWalletProvider. En la parte de arriba del fichero se añade la siguiente variable:

```
var HDWalletProvider = require("truffle-hdwallet-provider");
```

El siguiente paso es establecer la referencia de mnemonic para generar las cuentas. Es muy recomendable mantener este archivo secreto y separado, puesto que si se tiene acceso al mnemonic se tendrá acceso a todas las claves públicas y privadas de un usuario

```
var mnemonic = "apple orange banana ... ";
```

En este caso la red a la que se va a migrar es Ropsten, por lo que es necesario añadir la definición de esta red. Es el proveedor el que inicia el HDWalletProvider que usa como argumentos el mnemonic y la red deseada a la que nos queremos conectar.

```
module.exports = {
  networks: {
    ropsten: {
      provider: function() {
        return new HDWalletProvider(mnemonic,
"https://ropsten.infura.io/<INFURA_Access_Token>")
      },
      network_id: 3
    }
  }
};
```

Aunque en el código de arriba solo se ha definido una red, es posible definir múltiples redes, pero podemos especificar que red es a la que queremos acceder. Simplemente hay que cambiar el código, y pasarle el índice de la red, recordando que empieza en 0.

```
new HDWalletProvider(mnemonic, "https://ropsten.infura.io/<Infura_Access_Token>",
2);
```

4. Cómo adquirir Ether

Es necesario tener ether suficiente en la cuenta para poder desplegar nuestros contratos. Para la red Ropsten puedes adquirir ether a través de un servicio llamado en inglés “Faucet” o grifo. Hay multitud de páginas en las que se puede adquirir este servicio, uno de los más recomendados es EthTools. Lo primero es navegar hasta EthTools Ether Faucet. Es necesario introducir tu mnemonic y seleccionar la cantidad de ether que se necesita (con un máximo de 5). El faucet estará asociado con la cuenta. En un periodo breve de tiempo, la cuenta recibirá el ether solicitado.

Otra opción para obtener Ether podría ser a través de MetaMask. Simplemente hay que elegir la red Rospten en este caso y pulsar el botón de comprar. Automáticamente te llevará al

faucet de MetaMask para la red de prueba de Rospten, que funciona de manera muy parecida al descrito anteriormente.

5. Desplegar el contrato

Es necesario compilar el proyecto que queremos desplegar a través del siguiente comando:

```
truffle compile
```

Desplegamos el contrato en la red deseada, en este caso Ropsten

```
truffle migrate --network ropsten
```

Si todo sale bien nos saldrá un mensaje parecido a este:

```
Using network 'ropsten'.

Running migration: 1_initial_migration.js
  Deploying Migrations...
  ... 0xd79bc3c5a7d338a7f85db9f86febbee738ebdec9494f49bda8f9f4c90b649db7
  Migrations: 0x0c6c4fc8831755595eda4b5724a61ff989e2f8b9
Saving successful migration to network...
  ... 0xc37320561d0004dc149ea42d839375c3fc53752bae5776e4e7543ad16c1b06f0
Saving artifacts...
Running migration: 2_deploy_contracts.js
  Deploying MyContract...
  ... 0x7efbb3e4f028aa8834d0078293e0db7ff8aff88e72f33960fc806a618a6ce4d3
  MyContract: 0xda05d7bfa5b6af7feab7bd156e812b4e564ef2b1
Saving successful migration to network...
  ... 0x6257dd237eb8b120c8038b066e257baee03b9c447c3ba43f843d1856de1fe132
Saving artifacts...
```

Si se quiere verificar que el contrato fue desplegado de manera correcta, se puede ver en EtherScan en la sección dedicada a Rospten, simplemente se necesita el ID de la transacción, para realizar la comprobación.

De esta manera ya hemos conseguido desplegar un proyecto Truffle en una de las redes de Ethereum.

ANEXO B – COMANDOS CONSOLA GETH

Este apartado hace un breve resumen de las funcionalidades más útiles que nos permite Geth. Hay dos formas de acceder a Geth, mediante comandos utilizando 'geth' delante del comando que queremos usar o a través de una consola interactiva Javascript. Una descripción más detallada se puede encontrar en:

- https://github.com/ethereum/go-ethereum/wiki/Management-APIs
- https://github.com/ethereum/wiki/wiki/JSON-RPC

- Acceder a la consola interactiva Javascript:

```
$ geth attach ipc:/some/custom/path
$ geth attach http://191.168.1.1:8545
```

Geth proporciona las siguientes APIs:

ADMIN: ADMINISTRACIÓN DEL NODO

- addPeer: nos permite añadir un nuevo nodo a la lista de nodos estáticos a los que se intentará conectar cada vez que se restablezca la conexión. Devuelve un booleano.

```
>
admin.addPeer("enode://a979fb575495b8d6db44f750317d0f4622bf4c2aa3365d6af7c2843399
68eef29b69ad0dce72a4d8db5ebb4968de0e3bec910127f134779fbcb0cb6d3331163c@52.16.188.
185:30303")
True
```

- dataDir: permite saber cuál es el path absoluto en el que geth guarda toda su base de datos sobre el nodo.

```
> admin.datadir
"/home/usr/.ethereum"
```

- nodeInfo: este comando nos permite averiguar toda la información que se encuentra el nodo geth que está corriendo en ese momento.

```
> admin.nodeInfo
{
  enode:
"enode://44826a5d6a55f88a18298bca4773fca5749cdc3a5c9f308aa7d810e9b31123f3e7c5fba0
b1d70aac5308426f47df2a128a6747040a3815cc7dd7167d03be320d@[::]:30303",
  id:
"44826a5d6a55f88a18298bca4773fca5749cdc3a5c9f308aa7d810e9b31123f3e7c5fba0b1d70aac
5308426f47df2a128a6747040a3815cc7dd7167d03be320d",
  ip: "::",
  listenAddr: "[::]:30303",
  name: "Geth/v1.5.0-unstable/linux/go1.6",
  ports: {
    discovery: 30303,
    listener: 30303
  },
  protocols: {
    eth: {
      difficulty: 17334254859343145000,
      genesis:
"0xd4e56740f876aef8c010b86a40d5f56745a118d0906a34e69aec8c0db1cb8fa3",
      head: "0xb83f73fbe6220c111136aefd27b160bf4a34085c65ba89f24246b3162257c36a",
      network: 1
    }
  }
}
```

- peers: nos muestra una lista con todos los pares que actualmente están conectados a nuestro nodo y toda la información que se tiene de ellos

```
> admin.peers
[{
    caps: ["eth/61", "eth/62", "eth/63"],
    id:
"08a6b39263470c78d3e4f58e3c997cd2e7af623afce64656cfc56480babcea7a9138f3d09d7b9879
344c2d2e457679e3655d4b56eaff5fd4fd7f147bdb045124",
    name: "Geth/v1.5.0-unstable/linux/go1.5.1",
    network: {
      localAddress: "192.168.0.104:51068",
      remoteAddress: "71.62.31.72:30303"
    },
    protocols: {
      eth: {
        difficulty: 17334052235346465000,
        head: "5794b768dae6c6ee5366e6ca7662bdff2882576e09609bf778633e470e0e7852",
```

```
        version: 63
      }
    }
}, /* ... */ {
    caps: ["eth/61", "eth/62", "eth/63"],
    id:
"fcad9f6d3faf89a0908a11ddae9d4be3a1039108263b06c96171eb3b0f3ba85a7095a03bb65198c3
5a04829032d198759edfca9b63a8b69dc47a205d94fce7cc",
    name: "Geth/v1.3.5-506c9277/linux/go1.4.2",
    network: {
      localAddress: "192.168.0.104:55968",
      remoteAddress: "121.196.232.205:30303"
    },
    protocols: {
      eth: {
        difficulty: 17335165914080772000,
        head: "5794b768dae6c6ee5366e6ca7662bdff2882576e09609bf778633e470e0e7852",
        version: 63
      }
    }
}]
```

DEBUG: DEPURAR

- traceTransaction: este método intentará correr la transacción de la misma manera que si fuera ejecutada en la red.

```
>debug.traceTransaction("0x2059dd53ecac9827faad14d364f9e04b1d5fe5b506e3acc886eff7
a6f88a696a")
{
  gas: 85301,
  returnValue: "",
  structLogs: [{
      depth: 1,
      error: "",
      gas: 162106,
      gasCost: 3,
      memory: null,
      op: "PUSH1",
      pc: 0,
      stack: [],
      storage: {}
  },
    /* snip */
  {
      depth: 1,
      error: "",
      gas: 100000,
```

```
      gasCost: 0,
      memory:
["0000000000000000000000000000000000000000000000000000000000000006",
"0000000000000000000000000000000000000000000000000000000000000000",
"0000000000000000000000000000000000000000000000000000000000000060"],
      op: "STOP",
      pc: 120,
      stack:
["000000000000000000000000000000000000000000000000d67cbec9"],
      storage: {
        0000000000000000000000000000000000000000000000000000000000000004:
"8241fa522772837f0d05511f20caa6da1d5a3209000000000000000400000001",
        0000000000000000000000000000000000000000000000000000000000000006:
"0000000000000000000000000000000000000000000000000000000000000001",
        f652222313e28459528d920b65115c16c04f3efc82aaedc97be59f3f377c0d3f:
"0000000000000000000000002e816afc1b5c0f39852131959d946eb3b07b5ad"
      }
  }]
```

MINER: CONFIGURAR EL MINADO EN EL NODO

- setGasPrice: establece la cantidad mínima de gas aceptado cuando se minan las transacciones.

```
miner.setGasPrice(number)
```

- start: la CPU empieza el proceso de minado con el número de hilos que se establezca

```
miner.start(number)
```

- stop: acaba con el proceso de minado.

```
miner.stop()
```

- setEtherbase: establece la cuenta a la que irán las recompensas resultantes del proceso de minado.

```
miner.setEtherbase(address)
```

PERSONAL: MANEJO DE LAS CUENTAS

- listAccounts: devuelve una lista de todas las cuentas de las claves que estén almacenadas

```
> personal.listAccounts
["0x5e97870f263700f46aa00d967821199b9bc5a120",
"0x3d80b31a78c30fc628f20b2c89d7ddbf6e53cedc"]
```

- lockAccount: elimina la clave privada de la memoria, de manera que la cuenta no puede ser usada para enviar transacciones.

```
personal.lockAccount(address)
```

- newAccount: genera una nueva cuenta, posteriormente pedirá la contraseña si no se especifica como argumento

```
> personal.newAccount()
Passphrase:
Repeat passphrase:
"0x5e97870f263700f46aa00d967821199b9bc5a120"
```

- unlockAccount: desbloquea la cuenta cuando introduces la contraseña con la llave privada de la cuenta. Se puede establecer la contraseña y la duración de la cuenta desbloqueada como argumentos.

```
> personal.unlockAccount("0x5e97870f263700f46aa00d967821199b9bc5a120", null, 30)
Unlock account 0x5e97870f263700f46aa00d967821199b9bc5a120
Passphrase:
true
```

- sendTransaction: valida la contraseña y envía la transacción

```
> personal.sendTransaction(tx, "passphrase")
0x8474441674cdd47b35b875fd1a530b800b51a5264b9975fb21129eeb8c18582f
```

TXPOOL: MUY ÚTIL PARA INSPECCIONAR TRANSACCIONES

- content: nos muestra en detalle la lista de transacciones pendientes de ejecución y las que están pendientes de incluir en el bloque

```
> txpool.content
{
  pending: {
    0x0216d5032f356960cd3749c31ab34eeff21b3395: {
      806: [{
        blockHash:
"0x0000000000000000000000000000000000000000000000000000000000000000",
        blockNumber: null,
        from: "0x0216d5032f356960cd3749c31ab34eeff21b3395",
        gas: "0x5208",
        gasPrice: "0xba43b7400",
        hash:
"0xaf953a2d01f55cfe080c0c94150a60105e8ac3d51153058a1f03dd239dd08586",
        input: "0x",
        nonce: "0x326",
        to: "0x7f69a91a3cf4be60020fb58b893b7cbb65376db8",
        transactionIndex: null,
        value: "0x19a99f0cf456000"
      }]
    },
queued: {
    0x976a3fc5d6f7d259ebfb4cc2ae75115475e9867c: {
      3: [{
        blockHash:
"0x0000000000000000000000000000000000000000000000000000000000000000",
        blockNumber: null,
        from: "0x976a3fc5d6f7d259ebfb4cc2ae75115475e9867c",
        gas: "0x15f90",
        gasPrice: "0x4a817c800",
        hash:
"0x57b30c59fc39a50e1cba90e3099286dfa5aaf60294a629240b5bbec6e2e66576",
        input: "0x",
        nonce: "0x3",
        to: "0x346fb27de7e7370008f5da379f74dd49f5f2f80f",
        transactionIndex: null,
        value: "0x1f161421c8e0000"
      }]
    },
```

- inspect: este método lista todas las transacciones que están pendientes de incluir en el bloque, así como aquellas pendientes de ejecución. Es muy útil para visualizar la lista de transacciones y ver errores potenciales.

```
> txpool.inspect
{
  pending: {
    0x26588a9301b0428d95e6fc3a5024fce8bec12d51: {
      31813: ["0x3375ee30428b2a71c428afa5e89e427905f95f7e: 0 wei + 500000 ×
20000000000 gas"]
    },
    0x2a65aca4d5fc5b5c859090a6c34d164135398226: {
      563662: ["0x958c1fa64b34db746925c6f8a3dd81128e40355e: 1051546810000000000
wei + 90000 × 20000000000 gas"],
      563663: ["0x77517b1491a0299a44d668473411676f94e97e34: 1051190740000000000
wei + 90000 × 20000000000 gas"],
      563664: ["0x3e2a7fe169c8f8eee251bb00d9fb6d304ce07d3a: 1050828950000000000
wei + 90000 × 20000000000 gas"],
      563665: ["0xaf6c4695da477f8c663ea2d8b768ad82cb6a8522: 1050544770000000000
wei + 90000 × 20000000000 gas"],
      563666: ["0x139b148094c50f4d20b01caf21b85edb711574db: 1048598530000000000
wei + 90000 × 20000000000 gas"],
      563667: ["0x48b3bd66770b0d1eecefce090dafee36257538ae: 1048367260000000000
wei + 90000 × 20000000000 gas"],
      563668: ["0x468569500925d53e06dd0993014ad166fd7dd381: 1048126690000000000
wei + 90000 × 20000000000 gas"],
      563669: ["0x3dcb4c90477a4b8ff7190b79b524773cbe3be661: 1047965690000000000
wei + 90000 × 20000000000 gas"],
      563670: ["0x6dfef5bc94b031407ffe71ae8076ca0fbf190963: 1047859050000000000
wei + 90000 × 20000000000 gas"]
    },
    0x9174e688d7de157c5c0583df424eaab2676ac162: {
      3: ["0xbb9bc244d798123fde783fcc1c72d3bb8c189413: 30000000000000000000 wei +
85000 × 21000000000 gas"]
    }
}
```

- status: nos permite conocer cuántas transacciones deben ser añadidas al próximo bloque y cuáles están planeadas para la próxima ejecución

```
> txpool.status
{
  pending: 10,
  queued: 7
}
```

ANEXO C – LEVANTAR UNA RED QUORUM

Se usará el ejemplo de referencia de los 7 nodos que nos proporciona GitHub en el repositorio de Quorum. Este ejemplo nos permite hacer pruebas de consenso, privacidad, transacciones y todas las funcionalidades que además permite Ethereum. Este ejemplo configura cada nodo con su propia gestión para realizar transacciones privadas. Es importante tener instalado Git, Vagrant y VirtualBox.

Clonamos el repositorio en la carpeta desde la que vamos a trabajar:

```
git clone https://github.com/jpmorganchase/quorum-examples
```

Una vez instalado navegamos por las carpetas y inicializamos Vagrant utilizando la configuración que se encuentra en el archivo 'Vagrantfile'. En este caso estamos inicializando una máquina virtual con el sistema operativo Ubuntu.

```
cd quorum-examples
vagrant up
```

Nota: Es posible que al añadir el comando Vagrant Up nos salga un error que dice así "". Es necesario irse a los archivos de nuestro proyecto, al archivo Vagrantfile y de ahí debemos cambiar el path a "vagrant/bootstrap.sh.

Los comandos necesarios para levantar los nodos correspondientes si estamos usando el algoritmo de consenso Raft serán los siguientes:

```
raft-init.sh: Inicia las cuentas y las claves
raft-start.sh:  Lanza Constellation y los nodos geth y realiza una transacción
privada
stop.sh: Detiene Contesllation y todos los nodos geth
```

En el caso que se esté utilizando el algoritmo de consenso Istanbul BFT entonces los comandos serán los siguientes:

```
istanbul-init.sh
istanbul-start.sh
stop.sh
```

Todos los registros y datos temporales se escribirán en la carpeta qdata.

Una vez que tengamos la red configurada disponemos de 7 nodos Quorum que han sido configurados con diferentes roles para poder realizar pruebas en la red.

Nuestro siguiente paso consistirá en desplegar nuestro proyecto en la red. Vamos a sincronizar nuestro proyecto desarrollado en Truffle con nuestra plataforma. Lo primero es dejar correr la red y abrir una nueva consola.

Por ello debemos modificar el fichero de configuración ‘truflle-config.js’

```
module.exports = {
```

```
// ejemplo de los 7 nodos de Quorum, en este caso solo vamos a utulizar la red de
desarrollo
// podríamos añadir cualquiera de test de Ethereum
       networks: {
              development: {// esta es la red de Truffle para desarrollo
                     host: "127.0.0.1", //IP del localhost
                     port: 22000, // este puerto corresponde al nodo 1, como son 7
                                         van del 0 al 6
                     network_id: "*",
                     gasPrice: 0, // establecemos el precio del gas para nuestro
                                         proyecto
                     gas:4500000 //límite de gas que ponemos para el desarrollo
              }
       }
};
```

Ahora creamos un pequeño contrato de ejemplo para comentar un par de detalles

```
pragma solidity ^0.4.23;
// información que solo necesita el compilador
// indica que se necesitará esta versión de solidity o superior

contract SimpleStorage {

       //declaración de variables
       uint  public miVariable;

       function set(uint variable) public {
              miVariable=variable;
       }

       function get() constant public returns (uint variable)
       {
              return miVariable;
       }
}
```

Es posible que al ejecutar el comando truffle compile te salga un error que tiene que ver con el fichero 'Migrations.sol'. En general, este archivo no habrá que tocarlo, de manera que probablemente sea un problema de versiones. Lo mejor es desinstalar Truffle y volverlo a instalar a través de estos dos comandos

```
npm uninstall -g truffle
npm install -g truffle
```

A partir de ahí ya se puede continar con el desarrollo.

Para poder monitorizar nuestra red vamos a usar Cakeshop. Será necesario descargarse el archivo '.war'. Hay diferentes formas de inicializar el programa, pero en ese caso se va a utilizar a través del comando:

```
java -jar cakeshop.war
```

Si se está usando Windows es necesario escribir la siguiente información:

```
java -Dgeth.node=geth -jar cakeshop.war
```

Esta forma sería la correcta para inicializar Cakeshop para un nodo Ethereum regular, sin embargo, la idea es poder monitorizar nuestra red, aquella que estamos usando para nuestra dapp, de manera que tenemos que inicializar el programa en modo 'attach'. Y para ello debemos configurar de una manera especial el fichero 'application.properties' cambiando los siguientes parámetros:

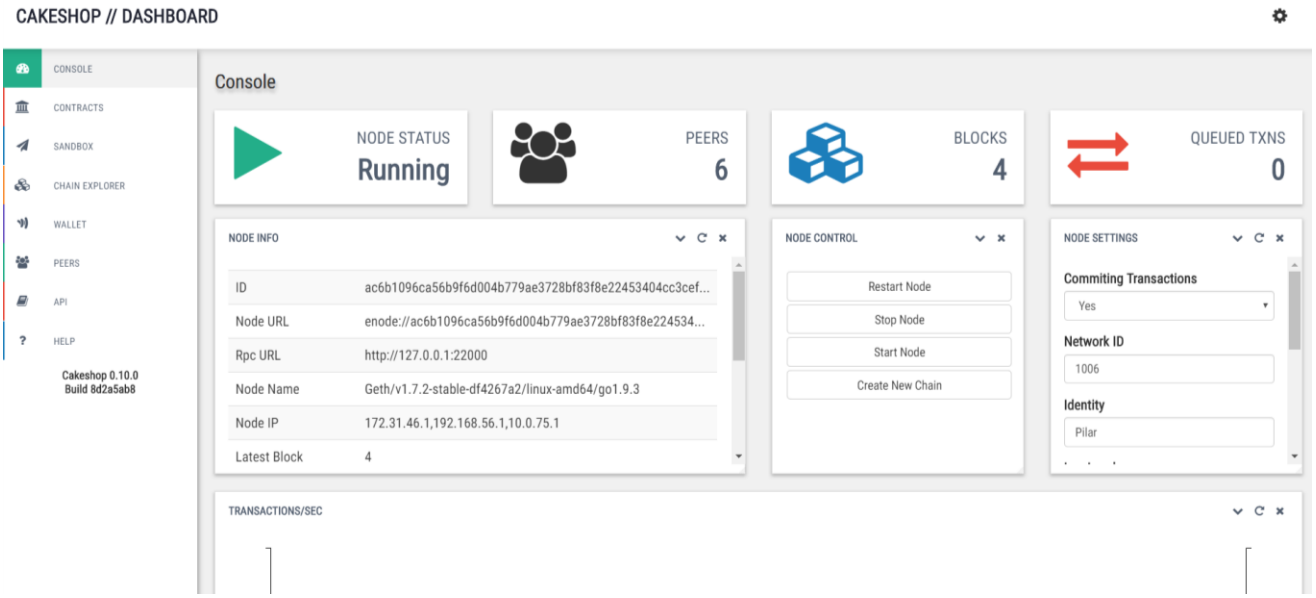
```
geth.auto.start=false

geth.auto.stop=false

 geth.url=http\://192.168.80.31\:22000
```

La url a la que debe apuntar el fichero es a la IP de nuestro nodo que queremos monitorizar. En este caso apuntamos al nodo 1 y sería: 127.0.0.1 con el puerto 22000

En este caso nos quedaría una pantalla como esta:



ANEXO D – OPENZEPPELIN

OpenZeppelin es una librería para contratos seguros en Ethereum en Solidity. Esta librería se integra con Truffle, el entorno de desarrollo que estamos usando en este proyecto. Lo primero que hacemos es iniciar npm y en el directorio raíz de nuestro proyecto debemos instalar la librería:

```
npm init -y
npm install -E openzeppelin-solidity
```

Después de esto, observamos en nuestro directorio que se ha creado una nueva carpeta ´node_modules´ que incluirá la carpeta ‘’openzeppelin-solidity/contracts’. Después de esto ya podemos incluir los contratos que se encuentran en la librería simplemente añadiendo el siguiente código en nuestro desarrollo. Como, por ejemplo:

```
import 'openzeppelin-solidity/contracts/ownership/Ownable.sol';

contract MyContract is Ownable {
  ...
}
```

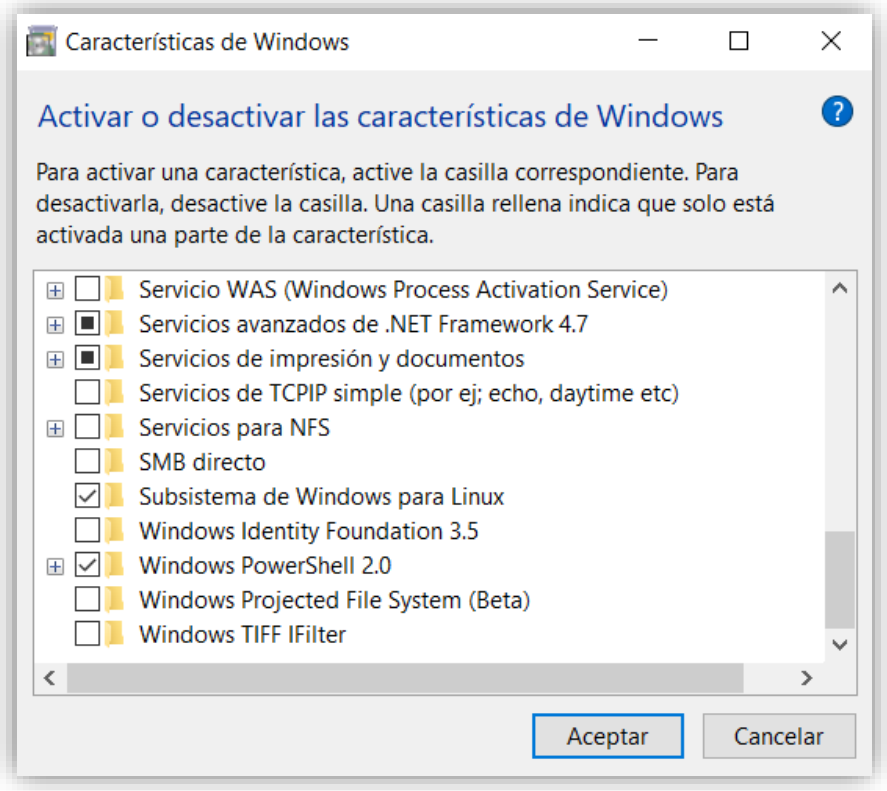
ANEXO E – MANUAL DE INSTALACIÓN

En este apartado se explicará cómo se instalan alguno de los paquetes que se necesitan para el desarrollo del proyecto.

SUBSISTEMA DE UBUNTU PARA WINDOWS

A la hora de realizar ciertos desarrollos es más cómo trabajar con Ubuntu que con Linux. Sin embargo, si nuestro sistema operativo es Windows, podemos aprovechar las ventajas de Ubuntu y usar las aplicaciones de nuestro sistema al mismo tiempo sin necesidad de usar contenedores o máquinas virtuales.

Para ello lo primero que debemos hacer es introducir: "Activar o desactivar características de Windows" en nuestra barra de búsqueda y automáticamente nos saldrá esta ventana:



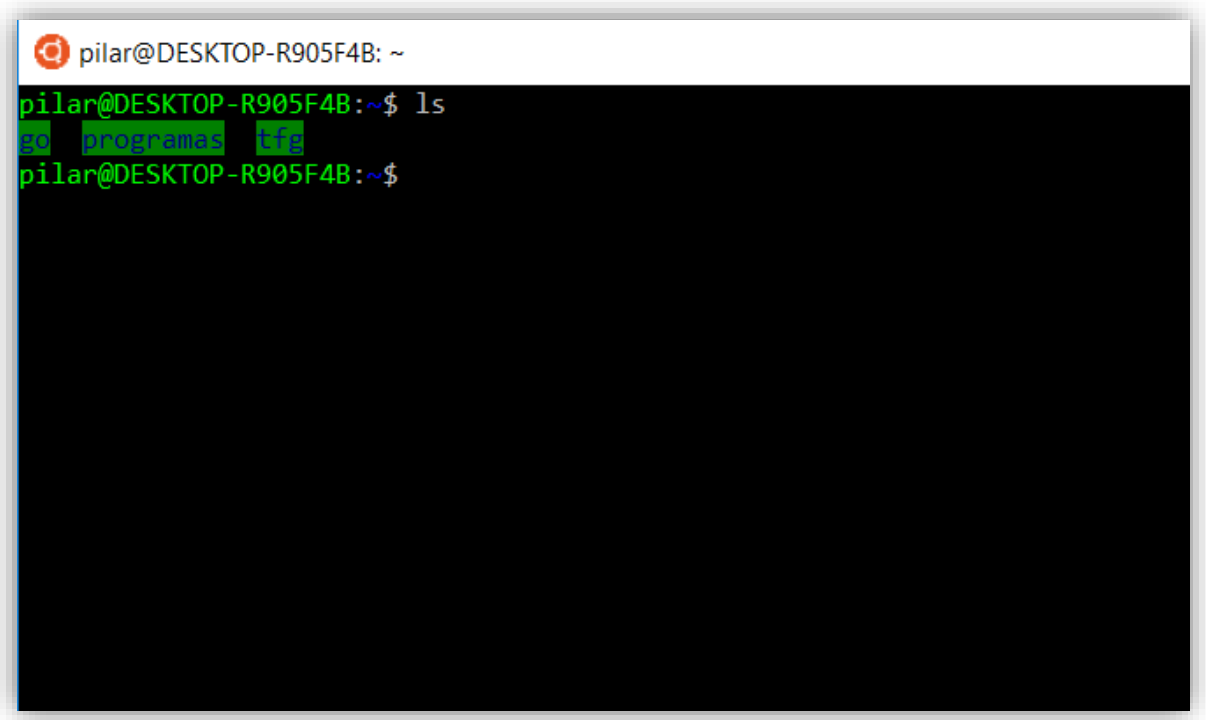
El siguiente paso es activar el apartado de 'Subistema de Windows para Linux' y aceptar. El sistema debe reiniciarse para aceptar las nuevas características.

Una vez reiniciado nos vamos a la tienda de Microsoft y buscamos 'Ubuntu'.

Nos saldrán tres tipos diferentes. En este caso se ha escogido la versión 16.04 que es la última versión estable.

Le damos a instalar y tendremos que esperar unos minutos hasta que se configure correctamente. Acto seguido nos pedirá un nombre de usuario y una contraseña. En ese momento ya tendremos nuestro subsistema instalado y listo para usar.

Podremos trabajar como si estuviéramos en un entorno Linux normal, pero sin interfaz gráfica.



Nota: es importante comprobar el PATH mediante el comando “echo $PATH” y que tenga una salida como esta:

```
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr
/local/games
```

En caso contrario debemos introducir en el fichero .bashrc en /home/usuario la siguiente configuración:

```
export
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr
/local/games
```

Si abrimos y cerramos la consola el PATH debería ser correcto.

NODEJS & NPM

Para que la instalación sea más fácil usaremos un gestor de paquetes. En Ubuntu instalaremos primero curl

```
sudo apt-get install curl
```

Ahora ya podemos introducir el comando para la descarga de Nodejs

```
curl -sL https://deb.nodesource.com/setup_10.x | sudo -E bash -

sudo apt-get install -y nodejs
```

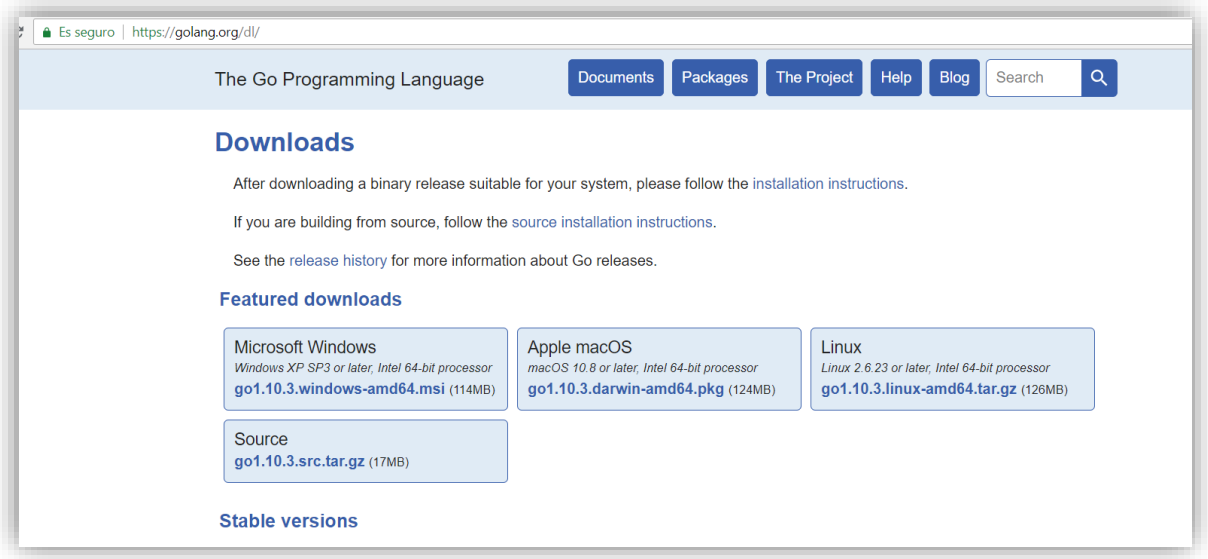
Una vez instalado comprobamos que las versiones de Nodejs y Node coinciden

```
nodejs –version
node --version
```

Por defecto Nodejs viene con NPM incorporado que nos será muy útil para la instalación de otros paquetes a lo largo del proyecto.

GO & GO-ETHEREUM

Lo primero es instalar la versión adecuada de Go. Para ello vamos a su página de descargas y escogemos la versión que vamos a utilizar, en este caso Ubuntu x64.



Podemos usar el comando wget para descargar el link y descomprimimos el fichero en la carpeta /usr/local. Por último, exportamos el *path.*

```
wget https://dl.google.com/go/go1.10.3.linux-amd64.tar.gz

tar -C /usr/local -xzf go1.10.3.linux-amd64.tar.gz

export PATH=$PATH:/usr/local/go/bin
```

Una vez tengamos Go instalado y el compilador C descargamos el repositorio de git en la carpeta que queramos y con el comando make terminamos la instalación de Geth.

```
git clone https://github.com/ethereum/go-ethereum.git
cd go-ethereum
make geth
```