COMMITTEE MANAGER APP: UNA APLICACIÓN PARA LA GESTIÓN DE COMITÉS DENTRO DE ORGANIZACIONES AUTÓNOMAS DESCENTRALIZADAS

PAULO J. COLOMBO

GRADO EN INGENIERÍA DEL SOFTWARE

UNIVERSIDAD COMPLUTENSE DE MADRID

TRABAJO FIN DE GRADO

20/09/2019

Director: Samer Hassan Collado y David Llop Vila

Autorizo a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos mencionando expresamente a su autor, tanto la propia memoria, como el código, la documentación y/o el software desarrollado.

Paulo J. Colombo

AGRADECIMIENTOS

Quisiera agradecer a mi tutor Samer Hassan Collado por permitirme formar parte de este trabajo de fin de grado y a David Llop Vila por su excelente dirección del proyecto y por dar un 110% de sí y ayudarme en aquellos momentos en los que el desarrollo del proyecto parecía llegar a un impasse.

Por último, quisiera agradecer a la comunidad de desarrolladores de Aragon por resolver la gran cantidad de dudas que tuve a lo largo de estos 3 meses y por promover el desarrollo de un framework extraordinario.

ÍNDICE

RESUMEN

El presente trabajo describe y detalla el proceso de diseño y desarrollo de Committee figuraManager, una aplicación para DAOs (Organizaciones Autónomas Descentralizadas) desarrolladas utilizando el proyecto Aragon, cuyo propósito es la creación y gestión de comités dentro de estas, siendo posible crear, eliminar o configurar las propiedades de cada comité.

El objetivo de Committee Manager es facilitar la realización de los procesos, funciones y operaciones propias de una DAO, como por ejemplo: la admisión de nuevos miembros, el pago a proveedores o servicios ofrecidos por terceros, etc. Las responsabilidades se dividen en comités, que son un subgrupo de entidades (usuarios y contratos inteligentes) que pertenecen a la DAO, y poseen permisos especiales (ya sea a título individual o por acuerdo con los demás miembros del comité) para realizar ciertas acciones dentro de la DAO.

La aplicación ha sido desarrollada utilizando las herramientas ofrecidas por el proyecto de código abierto Aragon, una iniciativa que pretende facilitar el desarrollo y creación de DApps, principalmente DAOs. El fundamento tecnológico principal del proyecto es Ethereum, plataforma en la cual, mediante el desarrollo de contratos inteligentes con Solidity, podemos crear y publicar las DAOs a la blockchain. Por otro lado, Aragon utiliza la librería React para el diseño y creación de la UI de las aplicaciones o módulos que se desarrollen para las DAOs.

Palabras clave: Aragon, Blockchain, Organizaciones descentralizadas autónomas, Ethereum, React, Smart Contract, Solidity, DAO.

ABSTRACT

This paper describes and details the process of design and development of Committee Manager, an application that can only be used within the DAOs (Decentralized Autonomous Organizations) created using the Aragon project, whose purpose is the creation and management of committees within them. It is possible to create, delete or configure the properties of each committee.

The objective of the Committee Manager is to facilitate the realization of the processes, functions and operations of a DAO, such as: admission of new members, payment to suppliers or services offered by third parties, etc; through the division of responsibilities into committees, which can be defined as a subgroup of entities (users and smart contracts) that belong to the DAO and have special permits (either individually or by agreement with other committee members) to perform certain actions within the DAO.

The application has been developed using the tools offered by the Aragon open source project, an initiative that aims to facilitate the development and creation of DApps, mainly DAOs. The main technological foundation of the project is Ethereum, a platform on which, through the development of smart contracts with Solidity, we can create and publish DAOs to the blockchain. On the other hand, Aragon uses the React library for the design and creation of the UI of the applications or modules that are developed for DAOs.

Keywords: Aragon, Blockchain, Organizaciones descentralizadas autónomas, Ethereum, React, Smart Contract, Solidity, DAO.

INTRODUCCIÓN

La organización y agrupamiento en comunidades ha sido un rasgo característico del ser humano desde el principio de su existencia. La unión a un grupo favorecía la supervivencia del individuo frente a aquellos que eran solitarios, al ofrecer ventajas como: mayor seguridad y protección frente a otros depredadores, facilidad al acceso de alimentos o cuidados de las crías. Con el pasar del tiempo estas agrupaciones fueron creciendo tanto en número como en complejidad, y para gestionarlas adecuadamente se requería de modelos de gobernanza más complejos: conjuntos de normas, reglas y principios que regulan el proceso de toma de decisiones que afecta a la propia comunidad y establezcan la forma en que los miembros de esta interactúan entre sí.

A lo largo de la historia, muchos modelos de gobernanza han surgido y han contribuido en gran medida a las formas de organización de la actualidad. Tal es el caso de La Compañía Neerlandesa de las Indias Orientales, creada a principios del siglo XIV con el objetivo de contrarrestar al avance comercial británico, portugués y español. La compañía controlaba el monopolio comercial en Asia y necesitaba de una nueva forma de financiar los viajes y hacer frente a los riesgos que suponía navegar por las rutas comerciales. La solución fue recurrir a la emisión de acciones y bonos a los posibles inversores que se beneficiaban de los viajes en el caso de que estos fuesen exitosos. Por primera vez, se hace evidente una clara separación entre dos agentes: los inversores que aportaban el capital y no sufrían los peligros de los viajes en ultramar y los marineros y mercantes que formaban parte de los propios viajes. Esto sentó las bases de muchos modelos corporativos que darían lugar a variadas estructuras administrativas adoptada por muchas empresas a partir de ese momento hasta la actualidad.

Hoy en día, estas formas de gobernanza y modelos corporativos han tropezado con el creciente progreso tecnológico de los últimos años. La propuesta publicada por Satoshi Nakamoto a finales de 2008 relativa al desarrollo de una red P2P y el lanzamiento de Bitcoin al año siguiente lograron un decisivo avance en los campos de la criptografía, redes computacionales y la teoría de juegos. Propuso por primera vez la tecnología blockchain. Tecnologías subsiguientes han modificado el protocolo propuesto por Nakamoto con la finalidad de explorar nuevas posibles aplicaciones. Una de las más relevantes en la actualidad es Ethereum, una plataforma de código abierto que salió a mediados de 2015 y se caracteriza, entre muchas otras cosas, por desacoplar la capa de smart contracts de la capa de blockchain, estableciendo un entorno de desarrollo mucho más flexible que el de Bitcoin que permite el desarrollo de aplicaciones que usan el poder de cómputo que ofrece la red.

Ethereum amplió las posibilidades de los smart contract que originalmente se limitaban a la transferencia de dinero en el protocolo Bitcoin. Esto junto a las redes basadas en blockchain propiciaron la concepción de un nuevo tipo de organización denominada DAO (Organización Descentralizada Autónoma) que desafía los modelos de gobernanza existentes y promueve una forma de organización donde la toma de decisiones sea

enteramente cohesiva, puntual y efectiva; y los procesos de la organización y los esfuerzos conjuntos de los participantes puedan ser gestionados y automatizados hasta cierto punto a través del código (blockchain y smart contracts), reduciendo los costes burocráticos y la presencia de intermediarios.

Actualmente, muchas iniciativas han emergido propulsando el desarrollo de estas nuevas organizaciones. Entre ellas destaca Aragon, un proyecto de código abierto que tuvo sus inicios en 2016 y fue oficialmente fundado a mediados de 2017, y que busca crear una serie de herramientas y aplicaciones que faciliten el desarrollo y la gestión de nuevas DAOs, ofreciendo completa libertad a cualquiera de crear estructuras de gobernanza altamente configurables y aplicables a las mismas.

INTRODUCTION

The organization and grouping into communities has been a characteristic feature of the human being since the beginning of its existence. Any individual who was part of a group increased his survival chances as opposed to those who were lonely. Groups offers advantages such as: greater security and protection against other predators, ease of access to food or care of the offspring. With the passage of time these groups grew both in number and complexity, to manage them successfully a governance model was required: a set of rules, rules and principles that regulate the decision-making process that affects the community itself and establish the way in which members interact with each other.

Throughout history, many governance models have emerged and contributed greatly to today's forms of organization. Such is the case of the Dutch East India Company, created in the early fourteenth century with the aim of counteracting British, Portuguese and Spanish commercial progress. The company controlled the commercial monopoly in Asia and needed a new way of financing travel and addressing the risks involved in navigating trade routes. The company solution was the issuance of stocks and bonds to potential investors who benefit from any successful commercial travel. For the first time, a clear separation between two agents is evident: the investors who contributed the capital and did not suffer the dangers of overseas travel and the sailors and merchants who were part of the trips themselves. This laid the foundations of many corporate models that would create various administrative structures adopted by many companies from that moment to the present.

Today, these forms of governance and corporate models have stumbled upon the growing technological progress of recent years. The proposal published by Satoshi Nakamoto at the end of 2008 regarding the development of a P2P network and the launch of Bitcoin the following year achieves a decisive breakthrough in the fields of cryptography, computer networks and game theory. He proposed blockchain technology for the first time. Subsequent technologies have modified the protocol proposed by Nakamoto with the proposal to explore new possible applications. One of the most relevant today is Ethereum, an open source platform that came out in mid-2015 and is characterized, among many other things, by decoupling the smart contracts layer from the blockchain layer, establishing a development environment much more flexible than that of Bitcoin that allows the development of applications that use the computing power offered by the network.

Ethereum expanded the possibilities of smart contracts that were originally limited to the transfer of money in the Bitcoin protocol. This together with blockchain-based networks led to the conception of a new type of established DAO (Autonomous Decentralized Organization) organization that challenged recognized governance models and promoted a form of organization where decision making is entirely cohesive, timely and effective; and the processes of the organization and the joint efforts of the participants can be managed and automated to some extent through the code (blockchain and smart contracts), reducing bureaucratic costs and the presence of intermediaries.

Currently, many initiatives have emerged propelling the development of these new organizations. Among them, Aragón stands out, an open source project that had its beginnings in 2016 and was officially founded in mid-2017, which seeks to create a series of tools and applications that facilitate the development and management of new DAOs, complete freedom to anyone to create highly configurable governance structures appropriate to them.

MOTIVACIONES

Las posibilidades detrás de las DAOs son realmente prometedoras y más si pensamos que se apoyan en tecnologías innovadoras que tienen pocos años de existencia. Las DAOs tienen la capacidad de remodelar la sociedad en la que vivimos y la forma en que nos relacionamos y asociamos con otras personas, de allí la consideración de estudiar algunos de sus aspectos dentro de este trabajo.

Proyectos como Aragon, aunque grandes dentro de su nicho, no han recibido gran cobertura y siguen permaneciendo desconocidos para muchos desarrolladores. Los medios de comunicación siguen indefectiblemente asociando la blockchain, smart contracts y tecnologías derivadas como fuentes de especulación financiera atados al único propósito de crear y tratar criptomonedas, cuando sus aplicaciones claramente van más allá de esto. El orientar un trabajo de investigación a la exploración de las posibilidades de desarrollo que estas tecnologías esconden y mostrar a los lectores la profundidad de las aplicaciones que estas poseen se considera realmente necesario.

Siguiendo el desarrollo de Aragon a través de blogs y chats se encontró que la distribución de responsabilidades y cuestiones como la apatía al voto o delegación de obligaciones era un problema recurrente. Propuestas y bosquejos iniciales se publicaron en el blog, pero no encontraban respuestas reales de futuras implementaciones. Parte de la estructura de los comités desarrollados en este trabajo se vió influenciada por estas proposiciones.

¿Por qué Aragon?

Aragon ha mostrado un creciente desarrollo a lo largo de estos años, impulsado por una comunidad de desarrolladores activa y productiva. El framework está dotado de distintas herramientas con un amplio rango de funcionalidades ya completas que llegan a ofrecer al desarrollador interesado en utilizarlas un entorno de desarrollo y marco de trabajo bastante amplio. Y no solo esto, el proyecto está respaldado por una extensa documentación que aumenta cada día, e incluso ya existen iniciativas de internacionalizar el proyecto a otros lenguajes para hacerlos accesibles a otras comunidades.

En su momento, se contemplaron otras alternativas como Colony o DaoStack, pero estas se encontraban en una etapa de desarrollo muy temprana o no poseían una documentación lo suficientemente extensa o explicativa, factores que se consideraron negativos y que dificultarían el desarrollo del trabajo sobremanera.

OBJETIVOS

El principal objetivo de este trabajo es incorporar al repertorio de Aragon una aplicación que facilite la gestión de las operaciones de la DAO y permita flexibilizar la distribución de responsabilidades. Se busca crear una aplicación que complemente sinérgicamente a otras , como *Voting* o *TokenManager*, y amplíe los modelos de gobernanza que se puedan llegar a implementar.

No es una solución final ni acabada y está abierta a modificaciones, mejoras y retoques.

Otras finalidades que se esperaba cumplir con el desarrollo del presente trabajo son las siguientes:

1. Abrir las puertas a posibles futuros desarrollos utilizando Aragon.
2. Establecer un enfoque real al desarrollo de proyectos blockchain explorando las posibilidades de tecnologías innovadoras como Solidity, Ethereum o incluso React.
3. Desarrollar una aplicación que pueda ser utilizada o mejorada por la comunidad incluso después de haber terminado el TFG.

Se espera que el código desarrollado en este TFG constituya la piedra angular de una aplicación que permita segmentar las responsabilidades y distribuirlas entre una serie de grupos especializados.

ESTADO DEL ARTE

Bitcoin y el concepto de blockchain

En octubre de 2008, Satoshi Nakamoto publicó una propuesta donde plantea crear un protocolo llamado Bitcoin sobre una red P2P con la finalidad de transferir valor, representado por una criptomoneda llamada bitcoin, de un punto a otro de la red, sin necesidad de interactuar con terceras partes, como pueden ser bancos, plataformas de internet u otras instituciones. El protocolo propone una solución al antiguo problema humano de la confianza, al garantizar a todos los actores que forman parte de la red que pueden confiar en los resultados producidos por ésta sin necesidad de confiar unos en los otros o siquiera conocerse.

El protocolo Bitcoin es la convergencia de tres importantes campos que han experimentado un significativo desarrollo a lo largo de las últimas décadas:

1. Redes: El protocolo Bitcoin opera sobre redes P2P, conformada por nodos iguales entre sí en la que no hay servidores o clientes fijos.
2. Teoría de juegos: Los algoritmos que determinan la forma en que se recompensan, castigan y relacionan los participantes de la red se fundamentan en principios y conceptos de este campo de la matemática aplicada.
3. Criptografía: La seguridad de la red y el protocolo, desde el almacenamiento de los datos o el traspaso de paquetes de información entre nodos, se apoya completamente en técnicas y métodos criptográficos.

Como se ha comentado previamente, la red sirve de medio para transferir bitcoins entre los actores que forman parte de la misma. Normalmente, a estas transferencias de dinero se les suele llamar transacciones. La red lleva un registro de todas las transacciones que se han hecho en la red desde su creación. Para ello, Bitcoin se apoya fuertemente en la blockchain, una tecnología innovadora que ha presentado soluciones a grandes problemas en el campo de las redes P2P como el del doble gasto [5].

La tecnología blockchain se puede describir como una estructura de datos conformada por transacciones validadas por la red en algún punto del pasado, que se agrupan en bloques donde cada uno se conecta con el anterior de manera que se forma una lista enlazada comúnmente denominada cadena de bloques. Esta cadena se encuentra protegida criptográficamente de tal manera que ningún actor de la red puede modificar o corromper la cadena de bloques sin que el resto de participantes se den cuenta. Este hecho es el garantiza la completa confianza en el sistema sin tener que confiar en sus participantes.

Dentro de la red, cada nodo participante guarda una copia de la última versión de la cadena de bloques. Al no requerir de terceros, la red valida sus propias transacciones para ser

incorporadas a la cadena. La red emplea un algoritmo de consenso llamado *Proof of Work (PoW),* fruto de una mezcla entre incentivos económicos y criptografía, que ofrece incentivos económicos en forma de bitcoins para motivar a los nodos a validar e incorporar las nuevas transacciones a la cadena de bloques. Para seleccionar al próximo nodo validador se propone un juego matemático que deben resolver todos y cada uno de los nodo utilizando su poder de cómputo para encontrar la solución por fuerza bruta.

Los algoritmos de consenso mejoran la red, haciéndola tolerante a fallos o ataques y resistente a colusiones al proponer un contexto donde es económicamente inviable sabotear y engañar al sistema.

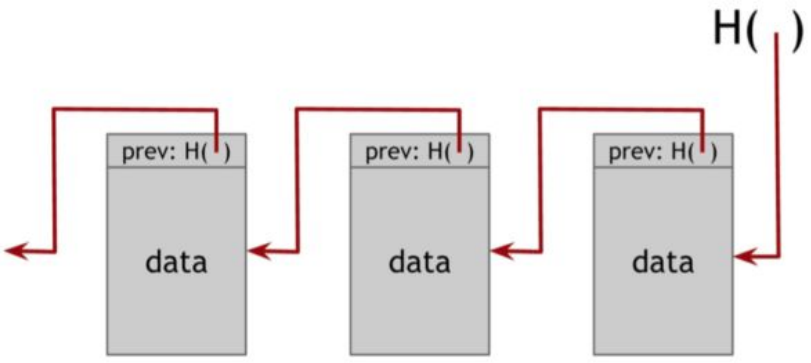

Figura 1. Cadena de bloques

Ethereum y la siguiente generación de blockchain

No tuvo que pasar mucho tiempo después de la publicación de Bitcoin para ver el potencial detrás de la tecnología blockchain, originalmente diseñada para trabajar en una red P2P con transacciones de criptomonedas. Los desarrolladores se dieron cuenta que las transacciones podían contener otro tipo de valor. Ademas, el lenguaje de scripting de Bitcoin era muy cerrado, poco expresivo y lento a la hora de incorporar nuevos cambios sintácticos.

A raíz de estas limitaciones nace el proyecto Ethereum, una plataforma de código abierto descentralizada desarrollada a mediados de 2014 y lanzado oficialmente el 30 de julio de 2015. La principal característica de Ethereum la podemos ver si prestamos atención al stack de tecnologías de este:

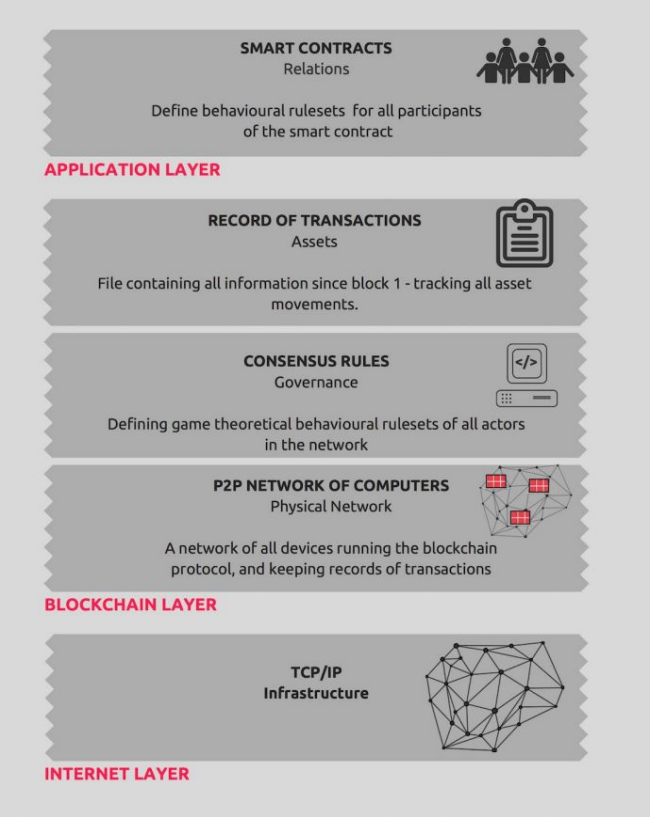

Figura 2. Stack de tecnologias de Ethereum [4]

Originalmente utilizados en la capa de blockchain para desencadenar transacciones automáticas sobre la cadena de bloques una vez ciertas condiciones se haya cumplido, los smart contracts son recolocados en la capa de aplicación de Ethereum desacoplándolos por completo de la blockchain con la finalidad de establecer un entorno de desarrollo para los smart contracts mucho más amplio y flexible que aquel definido en Bitcoin.

Los smart contracts se pueden definir como trozos de código autoejecutables que se despliegan sobre una red blockchain cuya finalidad es la de controlar activos digitales almacenados en la cadena y ejecutar transacciones si ciertas normas arbitrarias programadas se cumplen.

La principal ventaja de los contratos es que reduce el coste transaccional de :

1. Alcanzar un acuerdo entre las partes involucradas.
2. Formalizar el acuerdo entre ambas partes.
3. Aplicar el contrato una vez las condiciones definidas se hayan cumplido.

El rango de aplicaciones de los smart contracts puede ser bastante amplio, algunos de los casos de uso que podemos listar son:

- Ejecutar transacciones simples de transferencia de dinero entre una cuenta A y otra cuenta B.
- Registro de patentes. Un caso real es el de la plataforma Bernstein[1] que ofrece servicios blockchain para el registro propiedad intelectual
- Registro de documentos gubernamentales o semi gubernamentales[2], como pueden ser: certificado de nacimiento, documentos de propiedad, certificados académicos, etc.
- Promotor de la economía colaborativa, al permitir la representación criptográfica de bienes o servicios, en oposición a su análogo digital, que puedan ser intercambiados.
- Creación y gestión de tokens criptográficos que puedan representar activos, derechos de acceso o incentivos. Un ejemplo podría ser la plataforma Steemit, que ofrece tokens propios de la red para incentivar a los usuarios a contribuir con contenido

DAOs: Un uso de la tecnología blockchain con potenciales repercusiones

El avance de la tecnología blockchain, los contratos inteligentes y los tokens ha constituido un caldo de cultivo para todo tipo de desarrollos e iniciativas. Las DAOs conforman uno de los casos de uso más relevantes hoy en día, con un potencial disruptivo dentro de las estructuras organizativas los modelos de gobernanza tradicionales actuales.

A grandes rasgos, podemos definir una DAO como un tipo de organización donde la forma de gobernanza que implica actividades como la asignación y almacenamiento de recursos, interacción entre participantes, mecanismos de consenso y votación, etc; se codifican utilizando contratos inteligentes que residen en Ethereum, lo que comporta transparencia de los procesos de la organización y el acercamiento de participantes que no confían entre si.

Podemos definir las siglas de la siguiente manera:

1. **D**ecentralized: Al operar sobre una red apoyada en blockchain, ninguna entidad puede controlar la DAO, por lo que se evita cualquier punto central de fallo.
2. **A**utonomous: Se refiere al hecho de que la DAO puede tomar decisiones sin necesidad de que sea gestionada por entidades gestoras como la administración o la dirección.
3. **O**rganization: La tecnología opera sobre un conjunto de individuos o entidades con una finalidad común.

[1] https://www.bernstein.io
[2] El aspecto legal y regulatorio de este caso uso presenta serias dificultades para su implementación que deben ser evaluadas.

Las principales ventajas que ofrecen las DAOs son las siguientes:

1. Reducir los efectos del problema del agente-principal[3].
2. Desintermediar y reducir costes burocráticos y de gestión al incentivar a todos los participantes de la organización por medio de tokens nativos y alinear sus intereses personales con el interés global.
3. Sustituir el mecanismo reactivo de seguridad regulados por los sistemas legales actuales por mecanismos automáticos proactivos regulados por código.

Las DAOs presentan un gran contraste con las organizaciones tradicionales:

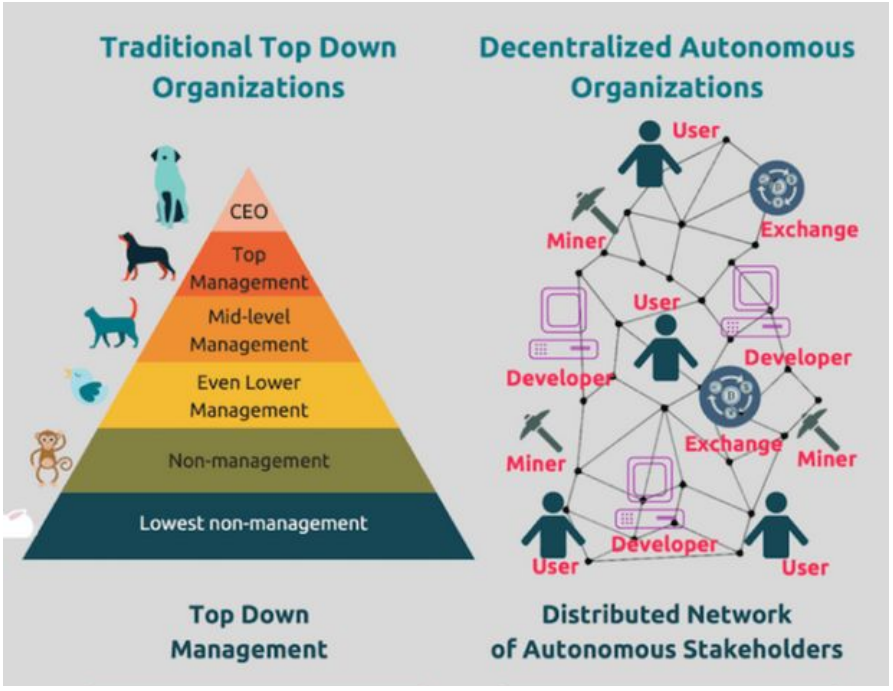

Figura 3. Organizaciones tradicionales verticales vs DAOs [3]

Las organizaciones tradicionales suelen estar divididas en múltiples capas de direccion y gestion que estan dispuestas de forma jerarquizada. Su objetivo es coordinar y fomentar la aplicación de los procesos y las operaciones de la organización. Para ello, gran parte del tiempo tienen que comunicarse bidireccionalmente con las capas adyacentes para transmitir información pertinente y tomar decisiones. El principal problema es que este proceso suele ser lento, costoso y puede degenerar en cuellos de botella en algunas partes de la organización.

[3] Un antiguo problema de gobernanza donde el agente de una organización tiene el poder de tomar decisiones en nombre del principal, otra persona o entidad de la misma organización , y que tienen consecuencias sobre estos. Existe toda clase de riesgos morales ya que el agente puede llegar a tomar decisiones a la ligera, sabiendo que las consecuencias de esta afectan a otras personas.

Por último, para tener una idea completa sobre las DAOs se deben comentar las inconveniencias y problemas que presentan a dia de hoy:

1. Actualmente, los aspectos legales de las DAOs siguen siendo materia de discusión. Todavía no hay una legislación que regularice esta clase de organizaciones.
2. Muchos aspectos relacionados con las DAOs siguen sin entenderse completamente y muchos la consideran una idea abstracta. Aunque durante los últimos años han surgido diversas iniciativas, todavía se considera que ha habido pocas aplicaciones de este tipo de organización.
3. Las DAOs son tan eficientes y seguras como lo permita el código hecho por los desarrolladores.

Uno de los ejemplos más representativos que ilustra los puntos previamente descritos es The DAO, de la cual hablaremos a continuación.

The DAO

The DAO se puede considerar uno de los acontecimientos más relevantes dentro de la historia de las DAOs. Consistió en un fondo de inversión de capital de riesgo creado en 2016 por miembros de la comunidad de Ethereum como Christoph Jentzsch, su hermano Simon Jentzsch o Stephan Tual; con la ayuda del equipo de desarrollo Slock.i. La idea era desarrollar una organización implementada con contratos inteligentes provista de una gestión autónoma de las inversiones y los fondos donde la participación de gestores de inversión fuera innecesaria.

The DAO abrió un periodo de venta de tokens que tuvo una duración de cuatro semanas, en la que los interesados podia transferir una cantidad de Ether a una cuenta determinada a cambio de tokens DAO. Se esperaba acumular un fondo común donde cada participante fuera copropietario de la DAO y tuviera derechos de votación en las decisiones de inversión proporcional a la cantidad de tokens poseídos. Cualquier participante podía presentar propuestas de proyectos al resto de la organización que eran sometidos a votación para decidir una posible financiación

La venta de tokens fue un rotundo éxito al llegar a acumular un total de 150 millones de dólares (alrededor de 12.7 millones de Ether en ese momento). Sin embargo, a mediados de junio de 2016, un hacker encontró un error en el código de la DAO gracias al cual llegó a robar alrededor de 70 millones de dólares de los fondos que transfirió a una cuenta personal.

Para sanear los daños, Ethereum reembolsó el dinero robado a los participantes realizando un hard fork [4] de la plataforma para transferir el dinero a una cuenta común a la que los participantes tuvieran acceso.

[4] Una bifurcación del código de la blockchain donde la nueva versión es incompatible con la antigua, siendo imposible comunicar ambas.

Este suceso supuso el final de The DAO a medida que plataformas de intercambios y servicios de trading como Poloniex o Kraken excluían los tokens DAO de sus listas de monedas digitales.

Otra consecuencia de gran importancia fue la división de la comunidad de Ethereum en dos grupos. Esto desembocó en la aparición de dos plataformas de Ethereum: aquella que incorporó el fork y la que se conoce como Ethereum Classic que no aceptó el fork por considerar que iba en contra de los principios de inmutabilidad de blockchain.

Plataformas y frameworks para el desarrollo de DAOs

En los últimos años, han surgido distintas iniciativas y proyectos que han intentado formalizar la metodología de desarrollo de las DAOs y modularizar su construcción. Veamos las más importantes.

DAOStack

DAOStack se puede definir como una plataforma de código abierto publicada en la primavera de 2018, que constituye un framework para el desarrollo de dApps, específicamente DAOs, cuyo objetivo principal se centra en la descentralización de la toma de decisiones de estas. De acuerdo a DAOStack, los problemas más importantes en el desarrollo de las DAOs actualmente radican

El ecosistema de DaoStack está conformado por múltiples DAOs interoperables que pueden llegar a interactuar entre sí, al poder comunicarse con otras organizaciones las DAOs pueden participar en votaciones de otras, transmitir datos de interés, ofrecer servicios, etc. Esta actividad maximiza el potencial beneficio procedente de la colaboración abierta y distribuida.

La creación, configuración y despliegue de estas organizaciones y sus modelos de gobernanza se apoya en una serie de herramientas fundamentales a disposición del desarrollador ideadas y desarrolladas por la plataforma.

DAOStack se organiza de la siguiente manera:

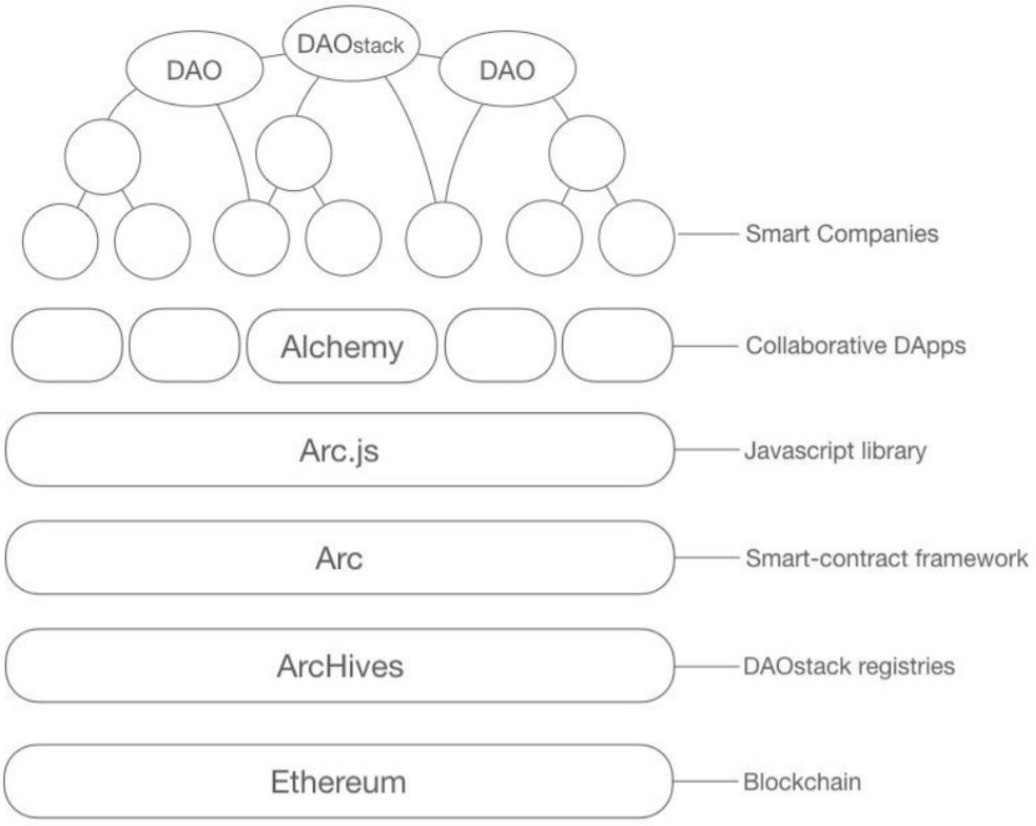

Figura 4. Stack de tecnologías de DAOStack [4]

1. Ethereum: Constituye la capa más baja. Sobre esta red se ejecutan todos los módulos de DaoStack implementados como contratos inteligentes.
2. ArcHives: Capa que contiene un conjunto de registros diseñados para mejorar la interoperabilidad entre DAOs. Se encuentran tres registros principales:
    a. Compendium: Estos registros se pueden ver como una especie de "app store" donde miembros de la comunidad de DAOStack pueden publicar nuevos módulos o elementos de gobernanza a costa de pagar una cantidad de tokens. Cada vez que una DAO utiliza el módulo publicado, el autor recibe un porcentaje de tokens como recompensa.
    b. The Hive: Contiene un listado de ofertas y peticiones de recursos o talento de las DAOs de la plataforma. Se busca reunir miembros de diferentes DAOs que puedan suplir las necesidades de otros de DAOs distintas.
    c. Mosaic: Contiene un listado de metadatos de todas las DAOs de la plataforma.
3. Arc: Framework de código abierto, modular y de uso general que permite el desarrollo de la estructura del modelo de gobernanza de la DAO. Ofrece una librería compuesta por módulos o elementos que podemos seleccionar y combinar para configurar el protocolo de gobernanza deseado para la DAO. También existe la posibilidad de crear nuevos módulos. A continuación, podemos observar una arquitectura convencional que puede adoptar el Arc dentro de una DAO:

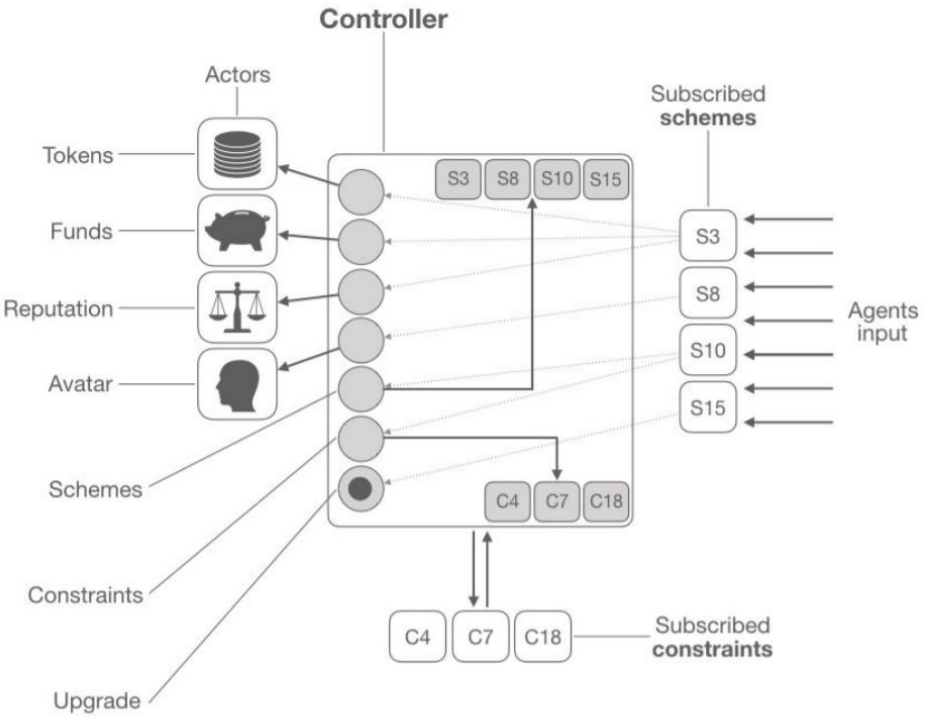

Figura 5. Arquitectura del framework Arc [4]

El núcleo del framework reside en el controlador, un contrato que se encarga de poner en contacto a los agentes (Agents), representados por direcciones de Ethereum, con los actores (Actors), que constituyen contratos que implementan algún componente del modelo de gobernanza, como por ejemplo el gestor de tokens (Tokens), fondos (Funds) o reputación (Reputation). La manera en que los agentes se comunican es por medio de los esquemas (schemes) suscritos al controlador, que actúan a modo de funciones que pueden ser invocadas a través de la emisión de transacciones que llegan al controlador y ocasionan el envío de órdenes a los actores por parte de este. Hay que tener en cuenta que para poder comunicarse con los esquemas, muchas veces una votación debe ser aprobada en la que participan otros agentes de la DAO.

Por último, el controlador también posee mecanismos para actualizarse al transferir el control sobre todos los actores que posee a una nueva dirección que puede ser un nuevo controlador actualizado que ha remodelado la arquitectura o uno completamente distinto.

4. Arc.js: Es una librería de JavaScript desarrollada sobre Web3.js[5] que abstrae al desarrollador de la estructura del framework Arc y del lenguaje Solidity en el que está implementado, permitiendo llamar a funcionalidades del framework desde el Front-End.

[5] Una API de Ethereum desarrollada en JavaScript.

5. DApps colaborativas: Esta capa está compuesta de DApps colaborativas a través de las cuales podemos interactuar con las DAOs del ecosistema. Esto nos permite acercar a usuarios promedios para que puedan interactuar con las DAOs a través de las DApps desarrolladas sobre Arc y Arc.js.
   Una de las DApps más importante actualmente es Alchemy[6], una intuitiva interfaz de usuario que permite crear DAOs de una manera intuitiva y amigable, crear tokens o coordinar los esfuerzos de todos los participantes de la organización para la asignacion y distribucion de recursos y fondos por medio de la creación, predicción y votación de propuestas.

Uno de los aspectos más característicos de DAOStack es el mecanismo criptoeconómico denominado consenso holográfico, qué busca de manera eficiente y lo más precisa posible tomar decisiones y procesar propuestas de tal manera que el resultado decidido se acerque a las decisiones grupales reales empleando una mayoría relativa en lugar de una mayoría absoluta del grupo, sin poner en peligro la resiliencia[7] ni la escalabilidad[8] de la organización.

Para emplear una mayoría relativa se debe consumir parte de lo que podemos llamar atención coletiva, un recurso escaso dentro de la organización que debemos gestionar de la mejor manera posible. Para ello, el consenso holográfico pone un precio en tokens a la publicación de propuestas, siendo necesario pagar una cantidad de los llamados tokens GEN si queremos que la propuesta sea considerada por los participantes de la organización.

A todo esto, se añade un último componente identificado como un “mercado de predicciones”. Las propuestas publicadas en la DAO son respaldadas por una red de predictores que apuestan una cantidad determinada de tokens GEN sobre el futuro de la propuesta, considerando que esta va a ser aprobada o rechazada. Si su predicción es acertada son compensados con tokens adicionales, en caso contrario, pierden los tokens apostados. Las propuestas respaldadas por una gran cantidad de predicciones favorables pueden ser susceptibles a cambios positivos en las condiciones de votación, por ejemplo, puede exigirse una menor cantidad de votos a favor en propuestas respaldadas por una gran cantidad de predicciones positivas.

Este mecanismo da lugar un modelo de toma de decisiones que hace uso de dos tokens complementarios pero bien diferenciados. Los tokens GEN transferibles y fungibles que actúa como “tokens de atención” que permiten filtrar y seleccionar propuestas en función de su probabilidad de aprobación determinada por las predicciones realizadas por la red, promoviendo así la escalabilidad dentro de la DAO. Por otro lado, tenemos los tokens de reputación no transferibles que permiten a sus dueños formar parte de las votaciones relacionadas con las propuestas.

[6] https://github.com/daostack/alchemy
[7] Resiliencia se refiere a la capacidad de una DAO de asegurar que todas las decisiones tomadas dentro de la organización se aproximan lo máximo posible a la opinión global de esta.
[8] Escalabilidad se refiere al número de decisiones que la organización puede realizar eficientemente en un periodo de tiempo.

Un ejemplo de aplicación de la estructura y protocolos desarrollados por DAOStack es The Genesis DAO, la primera DAO creada en la plataforma cuyo objetivo es la gestión y coordinación del desarrollo de DAOStack y su ecosistema, por medio de la asignación y distribución de sus recursos y fondos.

Colony

Es una colección emergente de tecnologías desarrolladas sobre Ethereum que el Protocolo Colony, un protocolo flexible y extensible que permite la posibilidad de incorporar a las aplicaciones formas descentralizadas y autorregulables de divisiones de trabajo, toma de decisiones y gestión financiera.

Colony nos permite crear lo que ellos denominan colonias, organizaciones descentralizadas que facilitan la colaboración grupal entre sus miembros y coordina los esfuerzos individuales para alcanzar objetivos comunes. Las colonias constituyen un medio para tratar uno de los problemas más relevantes dentro del protocolo de Colony, que es la división de trabajo. El trabajo se descompone en dos partes importantes:

1. Tareas: Constituyen la unidad estructural más pequeña dentro la colonia. Encapsulan una unidad de trabajo indivisible que no necesita ser delegada. Las tareas involucran tres roles:
    a. Un manager: Entidad encargada de crear y coordinar la tarea y su progreso. Se encarga de seleccionar el trabajador y el evaluador, la recompensa para parte una vez la tarea se haya completado (pueden ser tokens o criptomonedas de distinto tipo) y el plazo de entrega.
    b. Un trabajador: Entidad encargada de realizar la tarea.
    c. Un evaluador: Entidad encargada de comprobar que el trabajo realizado cumpla con los objetivos y requisitos de la tarea.

    La creación de tareas requiere de una cantidad determinada reputación y tokens.
2. Dominios: Constituye una unidad estructural que engloba subdominios o tareas parecidas. La finalidad de este componente es conceder cierta flexibilidad y modularidad a la colonia que permite abordar cualquier problema que surja dentro de la colonia de forma contextual, es decir, no se requiere de la participación de todos los miembros de una colonia sino sólo de una porción de participantes pertenecientes al dominio donde el problema ocurrió.

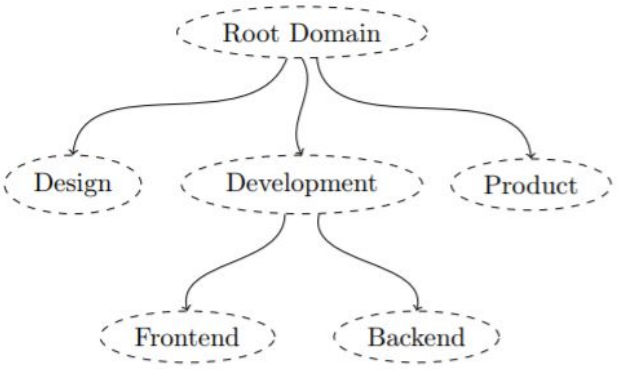

Figura 6. Ejemplo de dominios que conforman una colonia encargada del desarrollo de una web. [9]

De la misma forma que ocurre con las tareas, la creación de comités requiere de una cantidad determinada de tokens de reputación y la transferencia temporal de tokens que serán devueltas una vez el dominio haya sido creado.

Las colonias también poseen un token ERC20 propio, que se pueden obtener completando tareas. La configuración de los parámetros del token, como por ejemplo la cantidad máxima permitida, el ratio de emisión, el suministro inicial o incluso la usabilidad del propio token dependen completamente de la propia colonia que puede abrir votaciones para procesar proposiciones para tratar estos asuntos.

Todas las colonias del ecosistema de Colony son gestionadas por la entidad Colony Network, una colección de contratos desplegados en Ethereum entre los que destaca el contrato *Colony*, cuyas instancias conforman cada una de las colonias o el contrato central *ColonyNetwork*, que se encarga de la ejecución del proceso de minado de reputación y del control de la red general.

Una vez el contrato de la Colony Network haya sido desplegado, se crea la primera colonia denominada Meta Colony provista del token CLNY. Los dueños de este contrato tienen dos funciones principales:

1. Participar en el proceso de minado de reputación.
2. Gestionar la red Colony al regular, modificar y configurar parámetros propios de la red. Deben participar en votaciones para aprobar o rechazar las últimas actualizaciones disponibles del contrato Colony

Meta Colony es la beneficiaria de las comisiones que se pagan en la red. Se puede obtener reputación en la Meta Colony al participar en el proceso de minado de reputación o al adquirir nuevos tokens CLNY como resultado de completar tareas, obligaciones administrativas y evaluativas

Uno de los aspectos más destacables de Colony es la forma en que abordan la reputación dentro del protocolo.

La reputación se define como un valor numérico no transferible asociado a cada participante que pretende cuantificar el mérito de las contribuciones realizadas por este a la colonia reflejando el grado de aportación y compromiso general a la organización. Un alto nivel de reputación comporta un mayor peso en las decisiones tomadas y una mayor recompensa por parte de la colonia durante los periodos de reparto.

Colony granulariza la reputación de forma que la toma de decisiones en los distintos dominios de la colonia sea contextualizada, es decir, que pueda reunir a participantes, cuyas competencias sean afines al asunto sobre el que se quiere decidir. Se identifican dos tipos de reputación:

1. Reputación por dominio: Se cuantifica la reputación por dominio.Las ganancias o pérdidas de esta reputación ocurridas en un dominio determinado afectan en el mismo valor a los dominios padres y a los dominios hijos (este último solo en el caso de que se haya perdido reputación).
2. Reputacion por habilidad: Se puede considerar la forma más específica de reputación, donde se cuantifica la reputación para cada habilidad requerida en la realización de una tarea. Colony lleva un listado de etiquetas de habilidades universal off-chain, que es gestionado y mantenido por Meta Colony. De esta manera, al crear una tarea se le asigna una serie de etiquetas de habilidades relacionadas y al completar positiva o negativamente la tarea, se asigna uniformemente la reputación obtenida o pérdida entre cada habilidad

La reputación se puede adquirir de tres formas:

1. Formar parte de la realización de una tarea, ya sea como manager, evaluador o trabajador.
2. Formar parte de los procesos de disputa que involucra a dos o más participantes con opiniones opuestas.
3. Formar parte del proceso de creación y despliegue de una nueva colonia.

De igual manera, es posible perder reputación si alguna de las tareas descritas previamente se ejecutan pobremente. Además, con el paso del tiempo la reputación también va decrementando diariamente de forma que en el plazo de 90 días se reduce la totalidad de esta a la mitad. Para poder actualizar periódicamente la reputación se ideó un mecanismo de minado de reputación semejante al mecanismo de consenso PoW, en el cual puede participar cualquier miembro siempre y cuando ofrezca una cantidad de tokens.

Colony representa el estado de la reputación de cada usuario de cada colonia en un árbol Merkle [9], este incluye la reputación de cada habilidad que tenga el usuario. En la raiz del árbol, se encuentra el hash resultante que resume el contenido de toda la estructura.

[9] Estructura de datos en árbol binario donde cada nodo que no es una hoja representa el resultado de la función hash de la concatenación de los resultados hash de sus nodos hijos. Proporciona un método de verificación segura y rápida del contenido.

El mecanismo de minado consiste en producir un nuevo estado de todas las reputaciones partiendo del antiguo estado, añadiendo el conjunto de todos los cambios positivos y negativos en las reputaciones de todos los miembros de todas las colonias. Este proceso que se realiza off-chain produce un nuevo árbol Merkle del cual el minero calcula el hash de la raíz y lo envía, junto al número de hojas del árbol, al contrato *ColonyNetwork*. De todas las entregas válidas enviadas al contrato, solo se escoge una siguiendo una fórmula matemática. Una vez el nuevo estado se propague por toda la red, el minero seleccionado es recompensado con tokens de la colonia a la que pertenece y adquiere una cantidad de reputación en la Meta Colony.

Aragon

Aragon es un proyecto de código libre creado por Jorge Izquierdo y Luis Cuende enfocado al desarrollo de herramientas y módulos que constituyan un framework dedicado a la creación y gestión de DAOs altamente estructuradas de una manera fácil y sencilla. El framework se ha implementado en contratos inteligentes que se encuentran desplegados en la red Ethereum. Además, hace uso de IPFS como medio de almacenamiento principal

La iniciativa fue lanzada oficialmente en mayo de 2017 con la salida de una ICO (Initial Coin Offering) en la que se puso en venta los primeros tokens ANT (Aragon Network Token) del proyecto. Se llegó a recaudar alrededor de 25 millones de dólares procedentes de 2043 inversores en menos de 15 minutos. Es sin duda el proyecto de más alto perfil actualmente.

Como se acaba de comentar, cuenta con un conjunto de herramientas y módulos para el desarrollo de DAOs, los cuales se explicarán con mayor profundidad en posteriores capítulos.

Aragon cuenta con dos componentes de gran importancia:

1. Aragon Core: Consiste en una DApp orientada a la creación de todo tipo de DAOs (startups, corporaciones, organizaciones sin fines de lucro, proyectos de código abierto, etc) de manera intuitiva y sencilla que pueden ser desplegadas en la red principal de Ethereum. Permite incorporar aplicaciones, ya hechas por Aragon, a la nueva organización para extender la funcionalidad y modelar la estructura de la misma, como por ejemplo: un gestor de tokens, una aplicación para crear y procesar votaciones, una aplicación para controlar y gestionar las finanzas y pagos, etc.
2. Aragon Network: Es una organización de Aragon que provee servicios e infraestructura a los usuarios y DAOs que existen en la plataforma. Actúa como un marco jurisdiccional virtual gobernado por los poseedores de los tokens ANT, que participan en procesos de gobernanza denominados AGP (Aragon Governance Proposals) para mejorar el funcionamiento de la red y decidir cómo distribuir y asignar los recursos existentes para apoyar proyectos de desarrollo que beneficien la red.
   En caso de que haya disputas en torno a alguna AGP, esta puede ser procesada por el protocolo descentralizado Aragon Court en el que participan, por medio del pago

de una cantidad de tokens ANT determinada, una serie de miembros de la red que actúan como jurado para decidir si la disputa debe desestimarse o no. El jurado recibe una pequeña comisión por ofrecer sus servicios y dependiendo del resultado del veredicto el autor de la propuesta recibe o paga una cantidad de tokens a la contraparte si su propuesta es aprobada o rechazada.
El proceso general de las AGPs sería el siguiente:

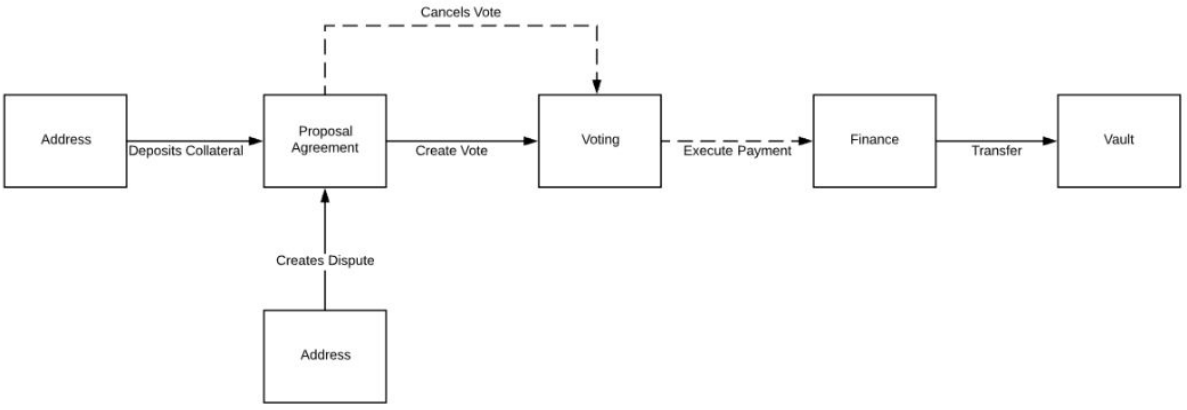

Figura 7. Proceso de las AGPs [10]

Los interesados en publicar una propuesta pagan un colateral conformado por tokens para que la propuesta se someta a votación a través de la aplicación *Voting*. En caso de aprobación se destina un pago utilizando la aplicación *Finance* que retira dinero almacenado en la aplicación *Vault* de la Aragon Network. En caso de originarse alguna disputa por considerarse que la propuesta no respeta los acuerdos y el formato establecido por la organización se llama a Aragon Court.

El proyecto más importante que utiliza el mecanismo de propuestas AGP es Aragon Nest [11], una serie de becas concedidas a equipos de desarrollo o pequeñas empresas para el desarrollo de proyectos que resultan en el beneficio de la comunidad y la adición de valor al ecosistema. Las becas pueden llegar a financiar hasta 150000 DAIs, divididos en trozos otorgados con cada hito del proyecto completado. Una vez que finaliza exitosamente el desarrollo, se concede hasta 30000 tokens ANTs al equipo de desarrollo [3].

Algunas organizaciones que han recibido la beca han sido: Autark (desarrollo de un conjunto de aplicaciones funcionales para DAOs), Prysmatic Labs (desarrollo de una implementación de un cliente sharding para Go-Ethereum), Pando (buscan desarrollar un Git repo descentralizado), etc. [12]

Un vistazo al panorama actual de las DAOs

dxDAO

dxDAO[10] es una DAO desarrollada utilizando el framework DAOStack cuyo lanzamiento ocurrió a finales de mayo de 2019. El objetivo de esta DAO consiste gestionar y coordinar el desarrollo y despliegue de protocolos o plataformas de comercio sobre Ethereum. Actualmente gestiona DutchX, protocolo enfocado al comercio e intercambio de tokens ERC20[11] empleado un mecanismo basado en la subasta holandesa[12], permitiendo comercializar tokens con baja liquidez y sin necesidad de involucrar a terceros.

dxDAO se encarga de coordinar la comunidad detrás del desarrollo de DutchX utilizando la infraestructura de DAOStack y el mecanismo de holografico de consenso de DAOStack facilitando la toma de decisiones en asuntos como:

- Actualizaciones de la lógica de los contratos inteligentes.
- Gestionar la lista de tokens permitidos para comercializar
- Cambios en la estructura organizativa y contributiva de la DAO.
- Efectuar cambios necesarios para cumplir con nuevos requisitos legales o regulatorios.

Los participantes de la DAO son recompensados con mayor poder de votación y reputación si comercian utilizando DutchX, bloquean los tokens ERC20 que posean o pujan por tokens GEN20 (token de DAOStack).

MakerDAO

MakerDAO es una DAO conformada por varios contratos inteligentes que gestiona y controla DAI Stablecoin, una criptomoneda estable que constituye una alternativa al uso de criptomonedas altamente volátiles como Ether o Bitcoin que se caracterizan por una fluctuación del precio desmesurada.

MakerDAO se encarga de respaldar y estabilizar DAI bajo el precio del dólar, utilizando sistemas dinámicos de deuda garantizada o CDP, mecanismos de retroalimentación autónoma y actores externos incentivados

Los CDP actúan como mecanismos financieros que nos permiten generar DAIs siempre y cuando ofrezcamos un activo colateral como garantía, que es Pooled Ether (PETH)[13]. Al generar las DAIs, creamos deuda que le indica al CDP que debe bloquear el acceso al activo hasta que presentemos una cantidad de DAI equivalente a la deuda inicial más una

[10] https://dxdao.daostack.io/
[11] Tipo de tokens que implementan la interfaz ERC20 que busca estandarizar el formato de los tokens utilizados dentro de la red Ethereum.
[12] Tipo de subasta en el que el precio del producto empieza a subastarse con un alto precio que es gradualmente rebajado hasta que algún participante compre el producto
[13] Un tipo de Ether que se recibe cuando se depositan Ether convencional en un contrato que recoge todo el Ether depositado por todos los usuarios de MakerDAO. Se utiliza como mecanismo preventivo en caso de que el precio del Ether colapse.

pequeña cuota de estabilidad que debemos pagar por los servicios ofrecidos. Una vez, la deuda esté saldada el usuario es libre de retirar tanto activo colateral como quiera.

MakerDAO hace uso de los tokens MKR, que actúan como tokens de gobernanza que se reparten entre los participantes de la DAO. Se utiliza para el pago de las cuotas de estabilidad y una vez utilizado se destruye

Los poseedores de tokens MKR tienen la posibilidad de participar en votaciones en las que se suelen tratar propuestas como :

- Modificación de variables de gestión interna de gobernanza de la DAO.
- Elección de los oráculos y nodos que se van a utilizar.
- Modificación y control de los parámetros de riesgo, como puede ser: ratio de liquidación, penalidad de liquidación, límite de deuda permitida, cuota de estabilidad o ratio de penalidad.
- Cierre de la DAO en caso de que ocurra un *flash crash*[14] del precio del Ether

MakerDAO es uno de los proyectos más exitosos construidos sobre la plataforma Ethereum. Actualmente posee 2% del Ether total de Ethereum, presenta un incremento del 20% en términos del DAI emitido mensualmente y alrededor de 71% de los usuarios devuelve el DAI que han adquirido en un corto periodo de tiempo, lo que indica que su uso especulativo es bajo [16].

Terra0

Terra0[15] es un proyecto en etapas muy tempranas de desarrollo creado por Paul Seidler, Paul Kolling y Max Hampshire en la Universidad de Artes de Berlín que busca explorar las posibilidades de que una entidad no humana provista de IA (Inteligencia Artificial), constituya una unidad económica autónoma capaz de producir capital por sí misma al funcionar como una DAO y sin necesidad de intervención humana, valiéndose de un sistema natural para sustentarse a sí mismo económicamente y expandirse.

La idea es desarrollar un contrato inteligente alojado en la red Ethereum que controle la entrada y salida de datos de una entidad, que llamaremos NHA (Non-human actor), que representa un pedazo de tierra. El contrato implementará modelos económicos y lógica que permita el análisis forestal para la explotación de la tierra.

Se pretende comprar un trozo de bosque en Alemania que pueda ser gestionado por el NHA . Cada 6 meses, un programa de análisis forestal se conecta con una API de algún proveedor para obtener imágenes satelitales del bosque. Utilizando librerías como OpenCV [16] se busca extraer información de estas imágenes para determinar el número, estado y

[14] Movimiento muy fuerte y repentino en la cotización de un activo concreto.
[15] https://terra0.org/
[16] Biblioteca libre de visión artificial utilizada para el procesamiento de imágenes. https://opencv.org/

edad de los árboles localizados dentro de la propiedad. El contrato accede a estos datos para determinar qué y cuántos árboles es posible vender por medio de la compra de licencias para el talado, sin poner en peligro el ratio de crecimiento de la población general de árboles. Las licencias se implementan utilizando un token llamado *Woodtoken* que puede ser adquirido a cambio de Ether. La cantidad de *Woodtoken* disponible y el precio de estos se recalcula dos veces al año de acuerdo a los cálculos de los árboles disponibles.

Para poner en marcha la DAO, se organizará una fase inicial de adquisición de fondos mediante la emisión de tokens *terra0* a cambio de Ether. El token actúa como una especie de obligación financiera capaz de revenderse a la DAO en el futuro. Esta etapa sirve para dotar a la DAO de Ether que pueda utilizar para empezar a operar. Una vez la DAO haya acumulado suficiente cantidad de Ether pagará a los inversores iniciales y se convertirá en una unidad económica independiente.

PolkaDAO

PolkaDAO es una DAO desarrollada utilizando DAOStack publicada a finales de abril de 2019, cuyo objetivo consiste en la gestión del protocolo Polkadot y la gestion y financiacion proyectos pequeños relativos a este que no superen una financiación mayor a 1000 dólares.

Polkadot es una tecnología heterogénea y multicadena que, por medio de una cadena de bloques central llamada *relay chain*, pretende permitir la comunicación entre estructuras de datos globalmente coherentes y validables denominadas *parachains* (cadenas paralelas). Se busca crear una espacio donde puedan coexistir cadenas de bloques que procesan transacciones con características potencialmente diferentes ya sean privadas propias de una o varias empresas o públicas como Bitcoin, Ethereum o Namecoin.

Dos factores claves de las cadenas de bloque que Polkadot pretenden abordar son: la interoperabilidad y la escabilidad. El framework busca abstraer dos componentes vitales de las cadenas de bloque: la transición de estado y el mecanismo de consenso. La transición de estado se refiere al mecanismo que emplea la cadena de bloque para pasar de un estado antiguo a uno nuevo y el mecanismo de consenso a la forma en que se propaga el nuevo estado a lo largo de los nodos de la red.

Polkadot también se vale de un token interno llamado DOT que sirve de incentivo para los actores que forman parte de la estructura interna del framework que llevan a cabo los procesos de validación y autenticación. Los poseedores también pueden participar en las votaciones relacionadas con la financiación de los proyectos de desarrollo o con aspectos del protocolo del framework.

METODOLOGÍA Y ORGANIZACIÓN

En el presente apartado se procederá a explicar y detallar la metodología de trabajo empleada, las formas de abordar y enfocar los objetos de estudio y las técnicas organizativas utilizadas en la elaboración de planificación, distribución de tareas y gestión de riesgos. También se describirán la tecnologías utilizadas en el proyecto, profundizando en aquellas que se consideren más importantes para entender mejor el trabajo.

Fases de trabajo

La realización del proyecto se puede segmentar en tres fases de desarrollo:

1. Fase de aprendizaje y acercamiento a las tecnologías: Esta fase tuvo una duración aproximada de dos meses. Consistía en un acercamiento a las tecnologías involucradas para adquirir cierta soltura con estas durante el desarrollo de software. Se realizaron dos cursos de aprendizaje de la plataforma Udemy: uno de React [18] y otro de Solidity [19]. Se organizaron sesiones teóricas que se realizaban tres veces a la semana y venían acompañadas de ejercicios y pequeñas prácticas donde se realizaban aplicaciones de juguete para asentar los conocimientos aprendidos. Durante esta fase no se concertaron reuniones con los tutores, y todo contacto con ellos ocurría a través de correo electrónico o Telegram para tratar alguna duda relativa a la tecnología estudiada durante ese momento.
2. Fase de diseño y planificación: La fase de diseño tuvo una duración aproximada de dos semanas. Se realizaron dos reuniones con el tutor, donde se discutió sobre el diseño tanto del Front-End como del Back-End. En cuanto al Front-End, se realizaron los primeros esbozos de la aplicación, se realizó un árbol de los componentes de React que se crearían, y el estado de la aplicación
3. Fase de desarrollo e implementación: Esta fase tuvo una duración de dos meses. Se realizaron reuniones semanales con el tutor para discutir los últimos avances y cualquier posible problema que había surgido. Durante esta fase se organizaban sesiones diarias para trabajar en el proyecto. A lo largo de esta fase hubo una comunicación continua con Aragon a través de los chats oficiales[17] creados para ayudar al desarrollador.

Metodología de desarrollo software

La metodología de desarrollo utilizada en este proyecto sigue los principios de las metodologías ágiles, especialmente la metodología Scrum. Se adaptó ciertos enfoques y técnicas al contexto específico del trabajo.

[17] https://aragon.chat/channel/dev-help

Se decidió apostar por un desarrollo iterativo e incremental, caracterizado por entregas constantes de software que eran evaluadas por el director.

Al ser un proyecto conformado por una única persona, se decidió excluir algunos aspectos de la metodología Scrum más enfocados a la gestión y trabajo en equipo, como pueden ser: las reuniones diarias de 15 minutos, las reuniones de retrospectiva, etc.

Se decidió utilizar la metodología Kanban para gestionar las tareas. Esto se vió motivado en gran parte a la experiencia en otros proyectos previa donde ya se había utilizado este tipo de técnicas.

La metodología Kanban divide el trabajo del proyecto en tareas representadas por tarjetas que son gestionadas y organizadas en tableros, donde las tarjetas se organizan en columnas que reflejan cada uno de las etapas de nuestro proyecto, por las cuales pueden pasar las tarjetas. De esta manera, logramos representar el flujo de trabajo general de nuestro proyecto y a medida que avanzamos en las tareas las colocamos en la siguiente etapa.

En este proyecto, se decidió utilizar la herramienta web gratuita Trello, que nos permite crear tableros virtuales provistos de tantas etapas como queramos. La herramienta es muy intuitiva, simple, interactiva y rápida.

Se decidió utilizar las tres etapas tradicionales de Kanban:

1. To Do: Esta etapa contiene las tareas creadas que todavian no han sido empezadas.
2. Doing: Esta etapa contiene aquellas tareas que se han iniciado pero todavía no han acabado.
3. Done: Contiene aquellas tareas que ya han culminado.

El número máximo de tareas en progreso se limitó a 3 para no sobrecargar la etapa y ralentizar el flujo de trabajo general.

Debido a la gran cantidad de tareas que podían surgir en el proyecto, se decidió organizarlas por temática, creando un total de tres tableros:

1. Tablero de diseño: Este tablero contiene todas aquellas tareas relativas al diseño del software, ya sea diagramas, esquemas o bosquejos del Front-End o de la parte de Ethereum.

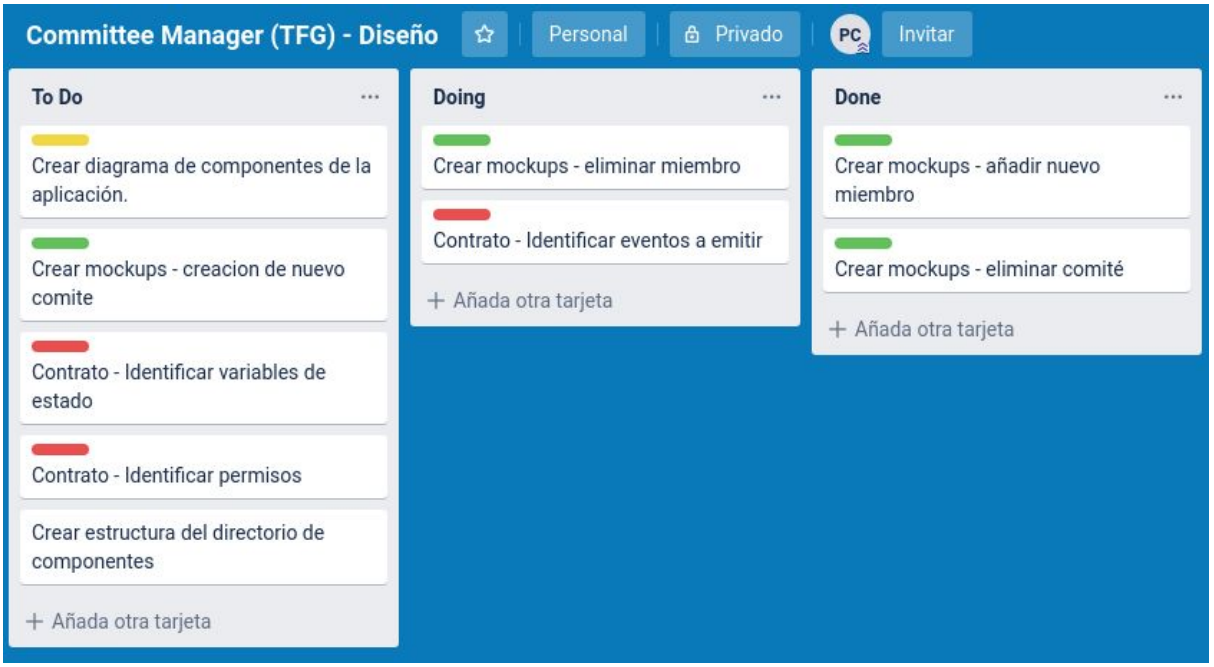

Figura 8. Ejemplo de tablero de diseño en Trello.

2. Tablero de implementación: Este tablero contiene todas aquellas tareas relativas al desarrollo del código de los componentes de React en el Front-End o el desarrollo de los contratos en Ethereum. También contiene cualquier otra actividad relativa a la implementación.

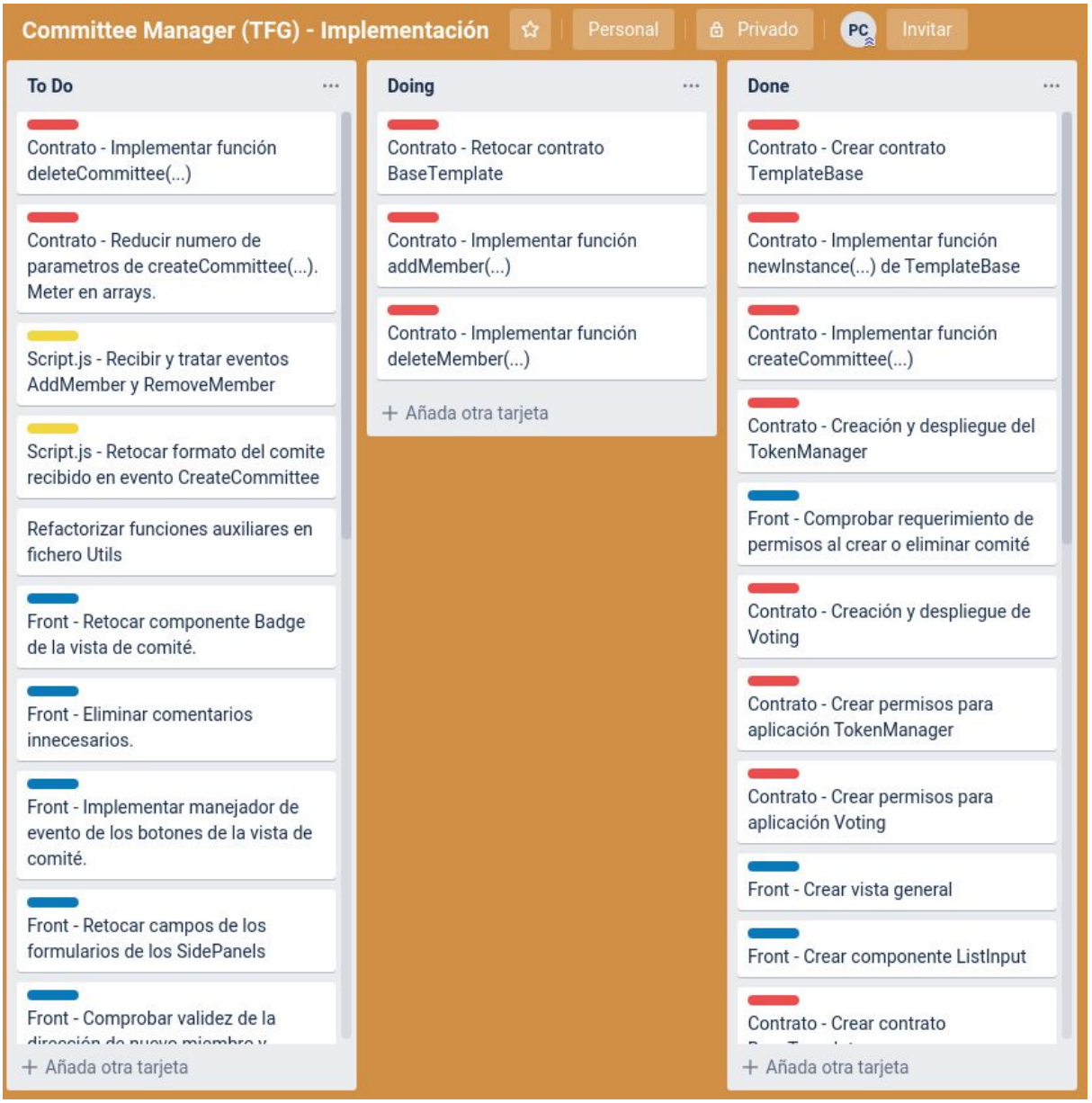

Figura 9. Ejemplo de tablero de implementación en Trello.

3. Tablero de memoria: Contiene todas la tareas relativas a la redacción, corrección y estructuración de la memoria.

Figura 10. Ejemplo de tablero de memoria en Trello.

Tecnologías utilizadas

A continuación se resumen las tecnologías utilizadas a lo largo del desarrollo del proyecto, téngase en cuenta que solo se listan aquellas tecnologías que se llegaron a utilizar directamente y no aquellas que utilizaba algún framework por debajo.

Tipo	Nombre	Uso

Frameworks	React	Creación de la interfaz de usuario de la aplicación
	AragonOS	Desarrollo de los contratos de la aplicación
Lenguajes de programación	JavaScript (ES6)	Escritura de código
	Solidity	Desarrollo de los contratos inteligentes
Lenguaje de diseño	CSS	Creación del estilo de cada componente de la UI desarrollado con React
Librerías	AragonUI	Uso de componentes y elementos UI
	AragonAPI	Comunicación entre los contratos y la interfaz de usuario y viceversa
Repositorios	Github	Control de versiones
IDEs	Webstorm	Escritura de código
	Visual Studio Code	Escritura de código
	Remix	Escritura y prueba de los contratos inteligentes
Administración de proyectos	Trello	Organización de tareas de desarrollo
Documentación	Google Docs	Escritura y redacción de la memoria
	LibreOffice Word	Escritura y redacción de la memoria
Diagramas y gráficas	Balsamiq Cloud	Creación de mockups
	Draw.io	Creación de diagramas
	Creately	Creación de diagramas
	sol2uml	Creación de diagramas UML de contratos

Aragon

La complejidad y tamaño de Aragon requiere dividir y estructurar el proyecto en distintos módulos con propósitos específicos. En total, existen cinco módulos:

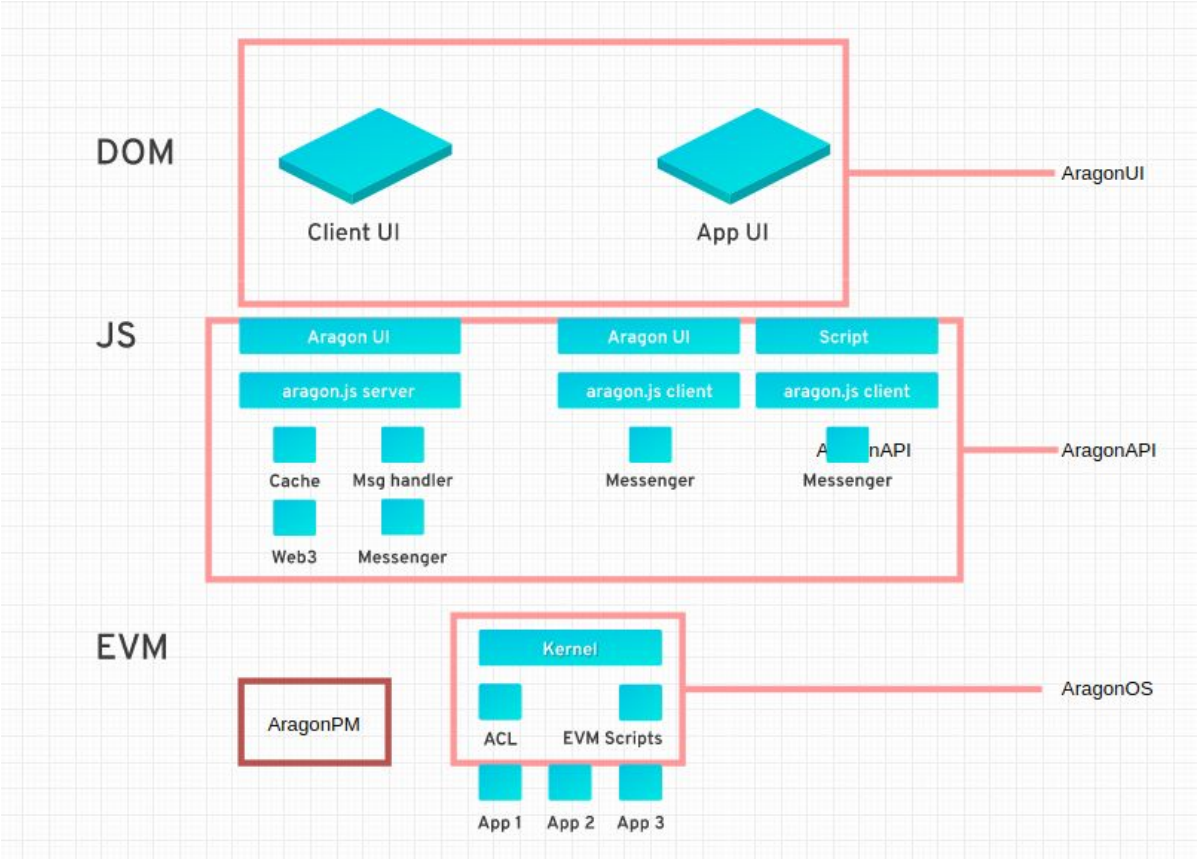

Figura 11. Stack de tecnologías de Aragon

AragonOS

Este módulo es un framework orientado al desarrollo de contratos. Contiene un contrato inteligente llamado Kernel que actúa como coordinador de los componentes que conforman una DAO. Se encarga de tres funciones principales:

1. La gestión de una estructura de datos llamada Lista de Control de Accesos (ACL), que establece la manera en que las aplicaciones de la DAO interactúan entre sí, definiendo los permisos que las entidades o aplicaciones tienen sobre otras, así como el gestor de esos permisos. Esto abstrae al desarrollador de tener que implementar la lógica de autenticación al programar una aplicación.

   La ACL se organiza de la siguiente manera:

Entity	App	Role	Manager

- Entity: Es la aplicación a la que se le concede el permiso.
- App: Aplicación que posee el permiso
- Role: Constituye el permiso.
- Manager: Es la entidad encargada de revocar o conceder el permiso

2. La gestión de las actualizaciones de cada una de las aplicaciones de la DAO, haciendo un seguimiento de las versiones de estas, específicamente de las referencias que apuntan a los contratos de estas aplicaciones. Esto permite incorporar funcionalidades futuras a la aplicación o la solución de bugs. Esta funcionalidad constituye una de las características más atractivas e innovadoras de Aragon ya que podemos versionar ficheros de código que en principio son inmutables una vez son desplegados en la red. La forma en que Aragon aborda este problema es a través delpatrón de diseño *DelegationCall*.

3. Llevar el control del escalado de la permisibilidad. La naturaleza de los permisos en Aragon difiere de los permisos que se gestionan en un sistema operativo convencional. El escalado de los permisos consiste en la concatenación de permisos que poseen varias entidades, logrando que estos sean transversales. Ampliando las posibilidades de diseño del modelo de gobernanza de la DAO al permitir un sistema de permisos más transversal.

Para ilustrar lo anteriormente mencionado, supongamos que nos encontramos en la siguiente situación:

Ejemplo de desarrollo

Dentro de nuestra DAO, tenemos dos aplicaciones: una llamada *Boveda* que se encarga de almacenar los fondos de la DAO y permite depositar o retirar Ether; otra llamada *GestorNomina* que se encarga del pago en Ether a participantes o entidades externas por servicios que hayan prestado. Para poder realizar el pago debemos primero poder retirar Ether de la *Boveda*, por lo que le asignamos el permiso SACAR_DINERO a *GestorNomina*.

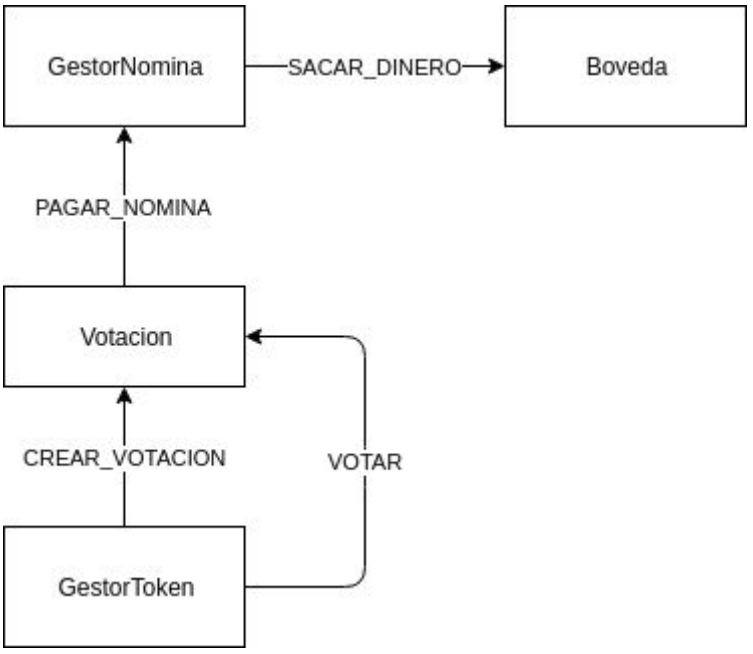

Figura 12. Ejemplo de desarrollo 2

Supongamos que el modelo de gobernanza de la DAO dicta que la determinación de los pagos sea sometida a la votación del comité de nóminas. A ese comité pertenecerán aquellos integrantes que posean el token NOM, creado para tal fin. Podemos conceder el permiso PAGAR_NOMINA del *GestorNomina* a la aplicación *Votacion* y, a su vez, el permiso CREAR_VOTACION y VOTAR, a la aplicación *GestorToken* que controla el token NOM.

Esto es un ejemplo de la escalabilidad que podemos otorgar a los permisos en una DAO, “encadenando” unos con otros. Ahora todos aquellos integrantes que tengan el token NOM pueden invocar las funcionalidades que requieran tener el permiso PAGAR_NOMINA a través de una aplicación intermedia, que en este caso es *Votacion*.

La Lista de Control de Accesos para esta DAO quedaría de la siguiente manera:

Entity	App	Role	Manager
GestorToken	Votacion	CREAR_VOTACION	Votacion
GestorToken	Votacion	VOTAR	Votacion
GestorNomina	Boveda	SACAR_DINERO	Votacion
Votacion	GestorNomina	PAGAR_NOMINA	Votacion

AragonPM (Package Manager)

Este módulo se desarrolló sobre AragonOS y consiste en un gestor de paquetes descentralizado que nos permite llevar un control sobre el versionado de los contratos inteligentes de las aplicaciones de las DAOs, de los propios componentes del framework de Aragon o de conjuntos de datos arbitrarios.

El APM contiene una serie de contratos desplegados en Ethereum que se denominan registros. Cada registro posee un dominio ENS (Ethereum Name Service) un sistema análogo al DNS que nos permite traducir las direcciones de Ethereum por un nombre de dominio de lectura fácil, por ejemplo: Aragon utiliza *aragonpm.eth* como dominio para almacenar los componentes base. Los registros están compuestos de repositorios que pueden estar registrados bajo subdominios del ENS del registro. Los repositorios se encargan de llevar registro de todas las versiones del contenido que alberga, ya sea la aplicación web de la aplicación, contratos, etc. El contenido se organiza en tuplas de datos compuestas por:

- Una referencia hash que apunta a un conjunto de datos que se encuentra guardado en IPFS (InterPlanetary File System), un sistema de archivos distribuidos que permite almacenar y compartir ficheros dispersos por todos los nodos de esta red P2P.
- Dirección de Ethereum que apunta al contrato relacionado con esa versión.

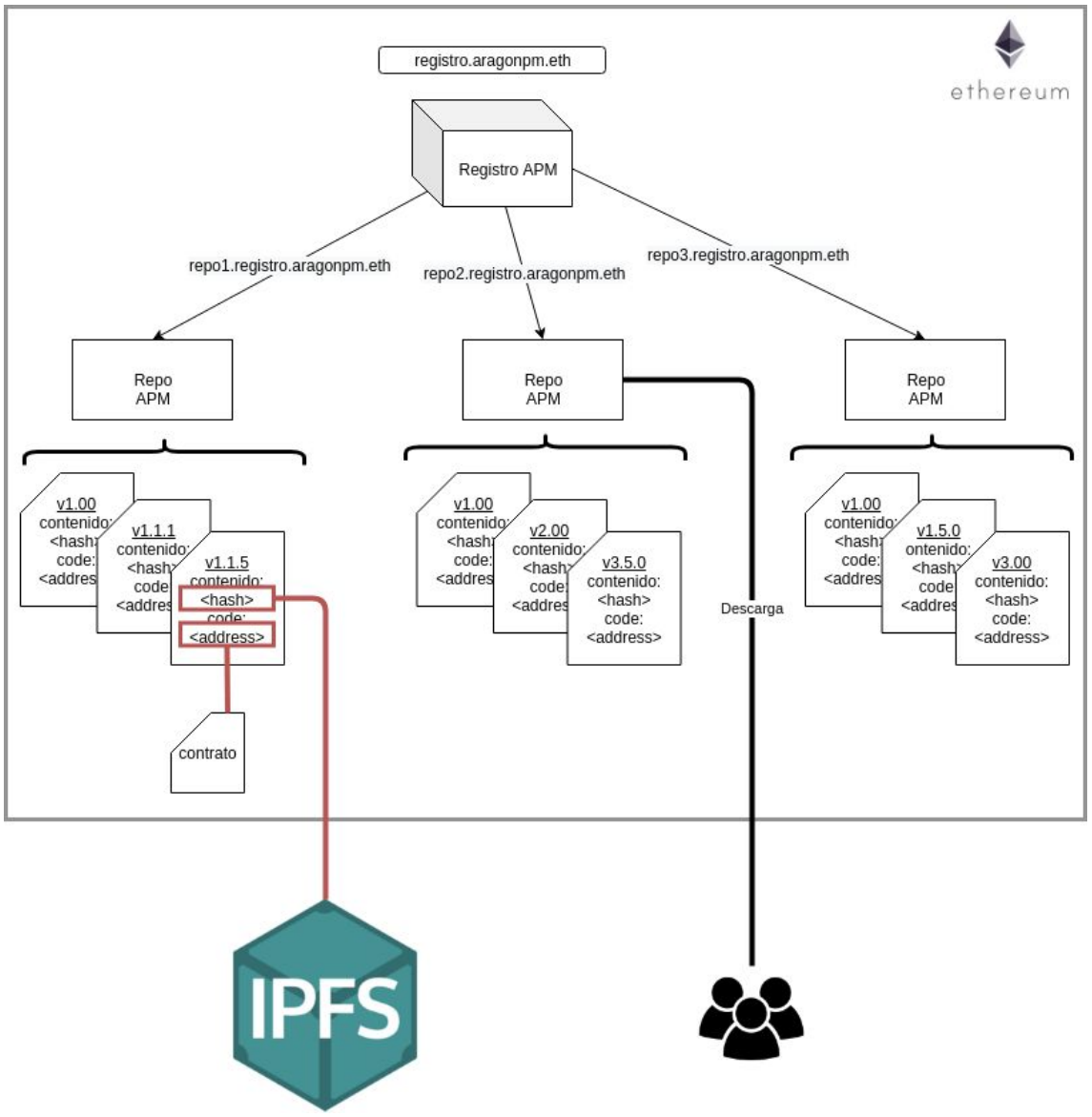

Figura 13. Estructura de AragonPM

AragonPM es altamente configurable, los repositorios de cada registro se pueden considerar como una DAO en sí, ya que podemos establecer un modelo de gobernanza que regule la forma en que se incentiva a los participantes a interactuar con el repositorio o como publican, auditan o modifican los paquetes dentro de estos.

AragonAPI

A grandes rasgos, este módulo se encarga de comunicar todos los componentes entre sí. Consiste en una librería de JavaScript que provee una interfaz a través de la cual:

1. Las aplicaciones desarrolladas en Aragon puedan comunicarse con los contratos de AragonOS.
2. Abstrae a las aplicaciones de crear, firmar y enviar transacciones a la red de Ethereum.
3. Conecta el Front-End de la aplicación y el cliente de Aragon con los contratos que residen en la red de Ethereum, al establecer un mecanismo de suscripción mediante el cual los cambios hechos en la blockchain actualizan y “renderizan” el estado de la interfaz de usuario por medio de la ejecución de scripts en segundo plano.
4. Permite realizar llamadas a los contratos de la aplicación desde el Front-End.

AragonCLI (Command Line Interface)

Es una linea de comandos que ofrece Aragon con la finalidad de gestionar y crear DAOs, abstrayendo al usuario de tareas relativas a la preparación de la infraestructura como por ejemplo: el despliegue de la blockchain local de Ethereum, configuración y montaje del nodo IPFS, despliegue de los contratos de la DAO o la publicación de las aplicaciones desarrolladas en el entorno local al APM.

El CLI está provisto de muchos comandos que nos permiten interactuar con la DAO, siendo posible realizar acciones como: crear nuevos proyectos, ejecutar las DAOs, instalar e inspeccionar las aplicaciones de la DAO, crear tokens, listar los permisos de las aplicaciones, etc.

También permite gestionar los repositorios de AragonAPM, siendo posible publicar nuevas versiones de las aplicaciones en el APM.

AragonUI

Es una biblioteca de UX que contiene una gran variedad de componentes gráficos hechos con React que facilita la creación de interfaces de usuario. Podemos encontrar: botones, gráficas, paneles laterales, tablas, etc.

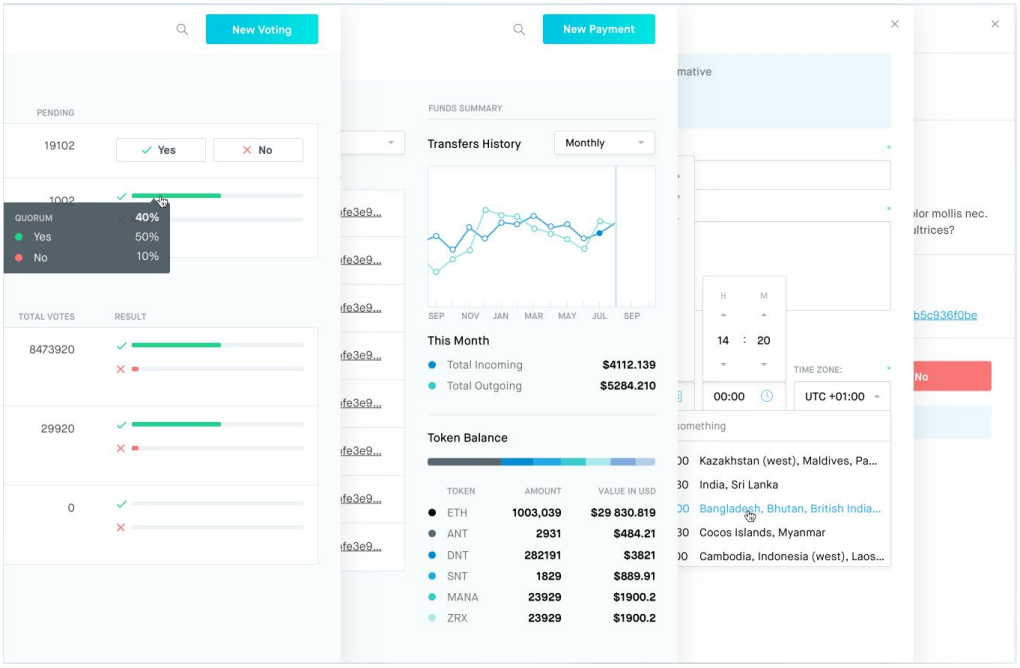

Figura 14. Biblioteca AragonUI

Aparte de que simplificó el trabajo relativo al desarrollo del Front-End, una de las razones por las que se utilizó esta biblioteca fue para obtener una interfaz de usuario coherente con el estilo del resto de aplicaciones de Aragon, para lograr una armonia en cuanto al diseño.

Templates de Aragon

Son scripts de despliegue que se ejecutan dentro de Ethereum para crear DAOs de manera atómica y en una única transacción evitando incurrir en los grandes costes de gas que implica emitir múltiples transacciones.

Los scripts son altamente configurables, permitiendo crear DAOs que respondan a modelos de gobernanza y estructuras organizativas detalladas.

Aragon ofrece múltiples *templates* por defecto, entre ellos podemos encontrar:

1. *Company*
2. *Company-Board*
3. *Membership*
4. *Reputation*
5. *Democracy*
6. *Multisig*

Metamask

Es una extensión para el navegador que te permite gestionar una cartera de Ethereum e interactuar con la red, permitiendo firmar transacciones, recibir ether, llamar a funciones de contratos, etc. Todo esto sin necesidad de usar un cliente de Ethereum y configurar un nodo en tu ordenador local, que puede llegar a ser bastante engorroso y costoso ya que se tiene que almacenar localmente la blockchain de Ethereum.

Metamask te permite configurar el nodo remoto al que te quieres conectar, que por lo defecto suele ser Infura.io[18], y lo hace por medio del uso de la librería de JavaScript Web3 que ofrece una interfaz para utilizar el protocolo JSON-RPC[19], empleado por Ethereum para definir la comunicación con los nodos.

Figura 15. Vista inicial de Metamask

Solidity

Es un lenguaje de programación orientado a objetos de alto nivel que nos permite implementar los contratos inteligentes en plataformas compatibles con la Ethereum Virtual Machine (EVM), como por ejemplo la propia Ethereum, Qtum, Dfinity, Rootstock, Ubiq, etc.

[18] Nodo de Ethereum público al que los usuarios pueden acceder para ejecutar DApps sin necesidad de que tengan que configurar o instalar su propio nodo.

[19]

Influenciado por otros lenguajes como Python, C++ o JavaScript (al cual se asemeja mucho), Solidity se ha convertido en el lenguaje principal para el desarrollo de los contratos dentro de Ethereum.

Solidity comparte muchas similaridades con otros lenguajes. Posee las estructuras de control más comunes, como por ejemplo: *for, while, if, else, return,* etc. De la misma forma, posee los tipos de variables típicos, por ejemplo: *int, boolean, string*, etc; añade otras nuevas como puede ser *address* que refleja una dirección de Ethereum y está provista de campos y métodos útiles como puede ser *balance* que nos permite obtener el balance de la cuenta a la que apunta la dirección o *transfer* que nos permite pasar una cantidad de Ether determinada a una cuenta. Otro tipo de variable que vale la pena mencionar es *mapping*, una especie de hashtable en donde podemos almacenar una clave y el valor al que esta apunta. Por último, Solidity también incorpora las funciones junto a modificadores de acceso conocidos como *public* o *private*, y otros propios del lenguaje como por ejemplo: *view, pure, payable*, etc.

A continuación se muestra un contrato de ejemplo sacado de la documentación oficial de Solidity que nos permite crear y transferir “monedas”.

```
contract Coin {

    address public minter;
    mapping (address => uint) public balances;

    event Sent(address from, address to, uint amount);

    constructor() public {
        minter = msg.sender;
    }

    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        balances[receiver] += amount;
    }

    function send(address receiver, uint amount) public {
        require(amount <= balances[msg.sender], "Insufficient balance.");
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```

FIgura 16. Contrato de ejemplo hecho en Solidity

A primera vista, podemos observar que la estructura del contrato guarda una gran similitud con la de una clase. Declaramos atributos, un constructor y métodos. Quien “instancie” el contrato tiene permisos de crear nuevas monedas y cualquier usuario tiene la posibilidad de transferir sus monedas a otra persona. También existen variables globales como es el caso aquí de *msg* que nos permite acceder a información adicional como el emisor de la transacción que ha creado el contrato o ha invocado alguno de sus métodos o datos relativos a la transacción .

Se puede observar que Solidity también nos permite crear eventos que al emitirse pueden ser capturados en la interfaz de la aplicación . Esto nos permite ejecutar lógica pertinente en el Front-End tras llamar al contrato.

Remix

Es un IDE para el navegador de código abierto que nos permite escribir y depurar nuestros Smart Contracts utilizando Solidity, Viper u otros lenguajes. La gran ventaja de utilizar Remix es que es altamente configurable y nos abstrae de la compilación y despliegue de nuestros contratos, por lo que nos facilita enormemente la tarea de depuración.

Entre las características más destacables de Remix que han facilitado el desarrollo de los contratos de este proyecto se encuentran:

1. Incorpora un listado de compiladores de distintas versiones de Solidity entre los que podemos seleccionar el que más nos interese.
2. Podemos seleccionar el entorno donde queremos que se desplieguen los contratos y se ejecuten las transacciones. Por defecto, Remix los despliega en una blockchain interna del navegador, pero podemos también decirle que se conecte a nodos remotos utilizando un cliente de Ethereum o a proveedores de Web3 como Metamask o Mist.
3. Posee un explorador de ficheros interno donde se encuentran todos nuestros contratos. Nos permite añadir, editar y eliminar nuevos ficheros.
4. Una de la secciones de la interfaz muestra un listado de todas las funciones de nuestro contrato con campos por cada parámetro que podemos rellenar para luego invocar la función. Es bastante intuitivo y simplifica el proceso de depuración.

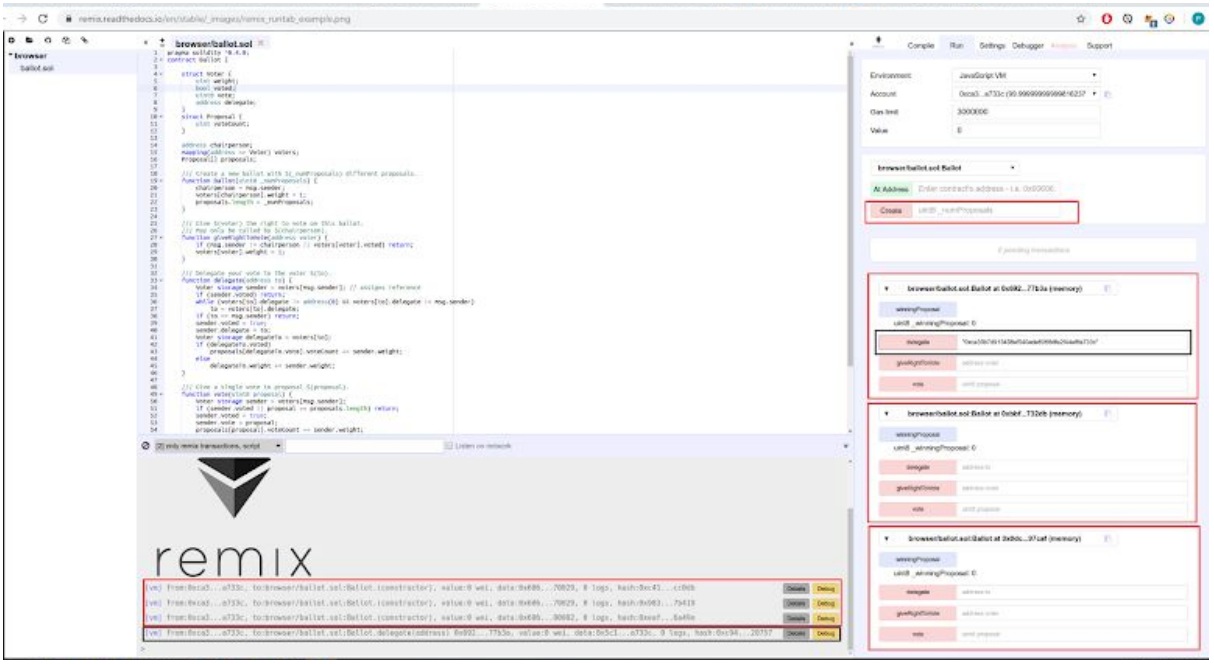

Figura 17.Vista principal del IDE Remix.

React

React es una librería de JavaScript de código abierto creada por Facebook para el diseño y desarrollo de interfaces de usuario dentro de aplicaciones de una sola página (SPA) por medio de la creación y desarrollo de componentes, trozos de código organizados en módulos que admiten una entrada de datos y renderizan otros como salida. Estos componentes son enteramente dinámicos por lo que cambios en los datos de entrada modifican el estado del componente y acarrean cambios en los elementos de salida renderizados.

Parafraseando a los desarrolladores de Aragon, las razones por las que se escogió React como framework del proyecto sobre otros como Vue o Angular fueron:

1. ***Popularidad**: React es el framework más popular entre los desarrolladores. Como plataforma, evidentemente queremos llegar a tantos usuarios como sea posible.*
2. ***Animaciones de buena calidad**: Nada parecido a react-motion existe en Vue, y tener una buena librería de animación es una necesidad absoluta si queremos alcanzar los niveles de calidad de la UI que nos proponemos*
3. ***Facilidad a la hora de dar estilo a los componentes**: Paquetes como styled-components no existen para otros frameworks (al momento de escribir esto), que facilitan el uso de CSS para estilizar los componentes.*

React opera sobre la idea de composición (y no herencia), es decir, los componentes pueden contener otros componentes de forma que se puedan construir elementos de la interfaz (botones, campos de formularios, formularios, barras de búsqueda, etc) que se agrupan en componentes de orden mayor que constituyen la estructura básica de nuestra aplicación.

Para ilustrar lo anteriormente mencionado, observemos a modo de ejemplo el componente de formulario del proyecto:

```
import React from 'react'
import { Button, Text, theme } from '@aragon/ui'

const Form = ({ children, onSubmit, submitText, heading, subHeading })
=> {
   return (
       <React.Fragment>
            {heading && <Text size="xxlarge">{heading}</Text>}
            {subHeading && <Text>{subHeading}</Text>}
            <div style={{ height: '1rem' }} />
            {children}
            <Button
                style={{ userSelect: 'none' }}
                mode="strong"
                wide
                onClick={onSubmit}
            >
                {submitText}
            </Button>
        </React.Fragment>
   )
}

export default Form
```

Figura 18. Formulario hecho con React

Este componente se suele llamar en React “componente funcional” ya que se define a modo de función que exportamos para luego poder ser utilizada en cualquier parte de la aplicación que queramos. Aparte de este tipo de componentes, también existen los componentes de clase, que se definen como una clase utilizando el estándar de JavaScript ES6.

El componente recibe una serie de datos de entrada que especificamos como parámetros de la función. En este caso, recibe una variable children que representa un conjunto de componentes hijos que contendrá Form, esperando que sean los campos del formulario; recibimos onSubmit que será el manejador de evento que se ejecute cuando presionemos el botón “Submit”; y luego otras variables contienen el nombre del botón de de submit, la cabecera y subcabecera del formulario.

Para implementar el componente se utiliza JSX, una extensión de la sintaxis de JavaScript promovido por React que busca unificar lenguajes de marcado (HTML) y diseño (CSS) e incorporarlos a JavaScript. Si queremos incluir algún tipo de lógica con JavaScript escribimos el código entre corchetes, como es el caso de las líneas 7 y 8, donde utilizamos una propiedad condicional de JavaScript donde si el primer valor es verdadero asignamos el segundo, que en este caso es un componente.

Por último, cabe mencionar que el componente Form contiene y hace uso de otros dos componentes: *Text* y *Button*.

DESARROLLO DE PROTOTIPOS: UN PRIMER ACERCAMIENTO

Durante el diseño y planificación del proyecto se identificaron dos tecnologías críticas para el desarrollo de la aplicación: React y Solidity. Era necesario adquirir competencias y cierto nivel de experiencia en su uso si se quería poner en marcha el proyecto.

Para fomentar el aprendizaje, se decidió realizar dos cursos que se encontraban en la plataforma online Udemy [20]. A continuación, se listan algunas de las razones por las cuales se seleccionó estos cursos y no otros:

- Ambos cursos se caracterizan por un alto componente práctico: se desarrolla una pequeña aplicación a la par que se van explicando conceptos, técnicas y herramientas relativas a la tecnología; cada funcionalidad desarrollada afianza los conocimientos aprendidos.
- Ambos cursos se imparten por educadores certificados con muy buenas críticas en la plataforma.
- Los cursos son renovados cada vez que un cambio significativo en la tecnología tiene lugar.

BurgerBuilder: Una aplicación hecha con React

Esta curso representa un primer paso en el manejo de React, consiste en el desarrollo de una pequeña aplicación de compra de hamburguesas llamada BurgerBuilder.

Se ofrecen tres funcionalidades:

1. Crear tu propia hamburguesa seleccionando los ingredientes y su cantidad.
2. Realizar el pago de la compra y rellenar un formulario de contacto para enviar a domicilio el pedido.
3. Consultar todas tus órdenes de compra.

[20] https://www.udemy.com/

Implementación

El primer paso en el desarrollo del prototipo fue el esbozo de un diagrama estructural que delineara los componentes que conforman la aplicación:

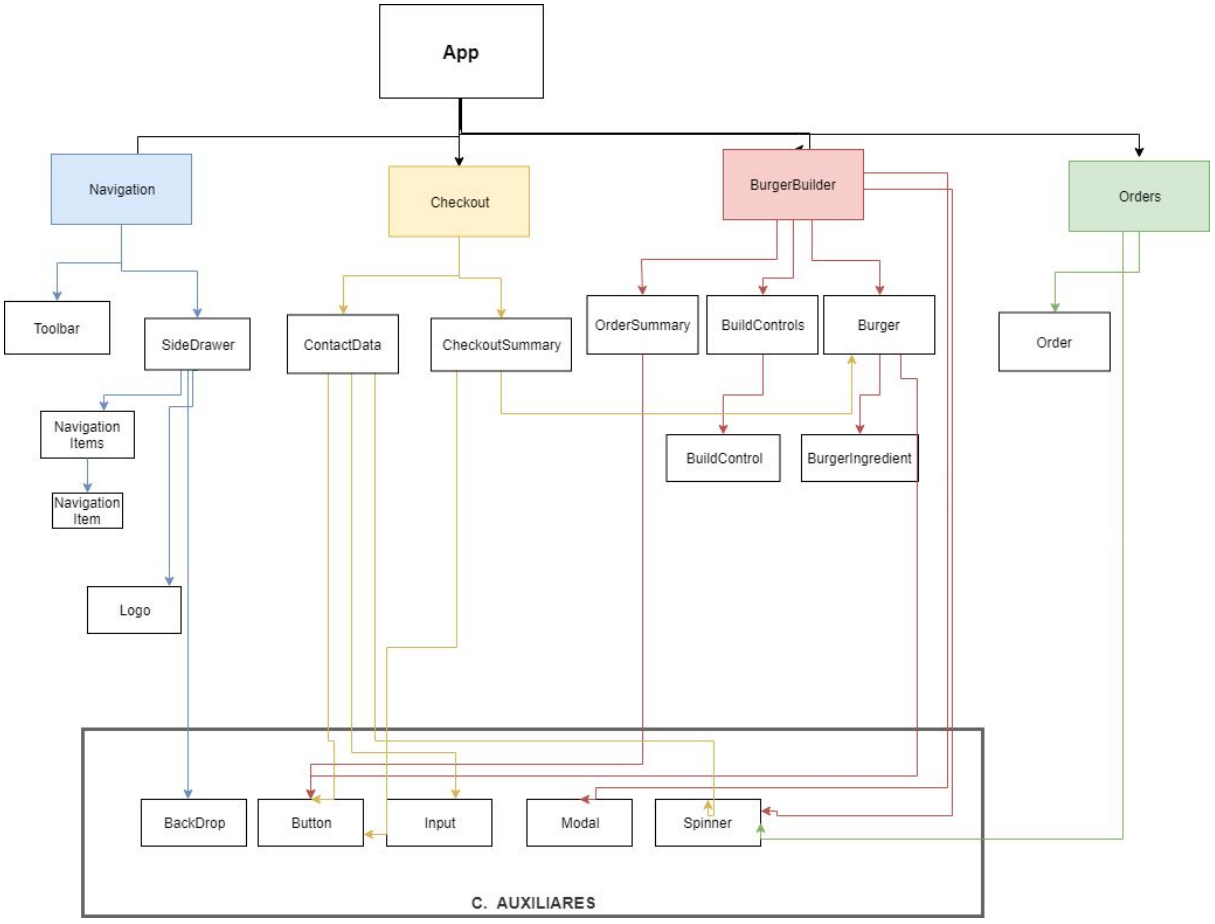

Figura 19. Diagrama de componentes de BurgerBuilder

Como se puede observar, cada componente está representado por una caja y las flechas representan una relación de composición, donde: si la flecha sale del componente A y llega al componente B, entonces A contiene y utiliza B.

El diagrama también muestra en la parte inferior los componentes auxiliares, una serie de componentes cuyo uso suele ser frecuente por lo que se han organizado en un directorio específico. Por ejemplo, los botones para aceptar o rechazar acciones; las ventanas modales para mostrar mensajes determinados; o los iconos de carga (spinners) que indican al usuario que su operación se está realizando.

La aplicación está compuesta por cuatro componentes principales:

1. Navigation: Representa la barra de navegación de la aplicación.
2. Checkout: Engloba el modal de confirmación de la compra y el formulario de contacto una vez la compra se haya realizado.
3. BurgerBuilder: Contiene la representación gráfica de la hamburguesa y el panel de ingredientes para personalizar el pedido.
4. Orders: Se encarga de renderizar las órdenes de compras y mostrar un listado de las mismas.

Para comunicar y conectar los componentes entre sí, se usó React.js [21], una libreria que nos permite definir el enrutamiento para nuestra aplicación, una manera de coordinar los componentes con las rutas URL.

Por último, se utilizó Redux para almacenar el estado de la aplicación e ir conociendo un poco la arquitectura Flux, de la cual hablaremos en capítulos posteriores.

La vista principal de la aplicación es la siguiente:

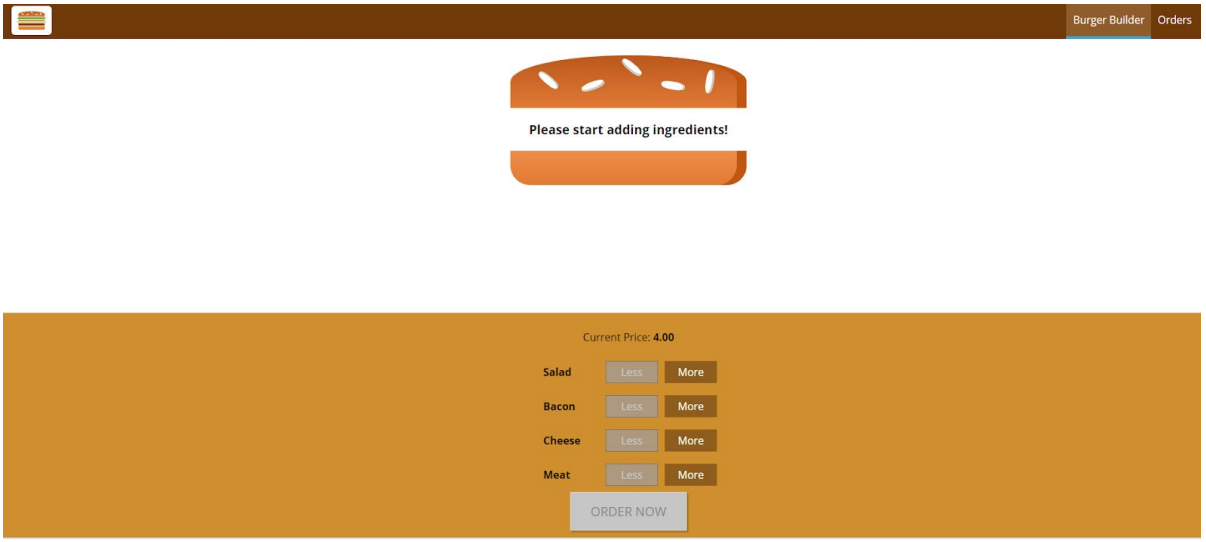

Figura 20. Vista principal de BurgerBuilder.

En el centro de la vista, se encuentra una representación visual de la hamburguesa que se va creando. Debajo, podemos observar un panel con selectores donde podemos añadir o eliminar un total de cuatro ingredientes; junto a estos se encuentra un botón inicialmente desactivado para procesar la orden de compra. Por último, en la parte superior podemos observar la barra de navegación donde se encuentra la sección “Orders” a la que podemos acceder para ver todas las órdenes de compra.

[21] https://reacttraining.com/react-router/web/guides/quick-start

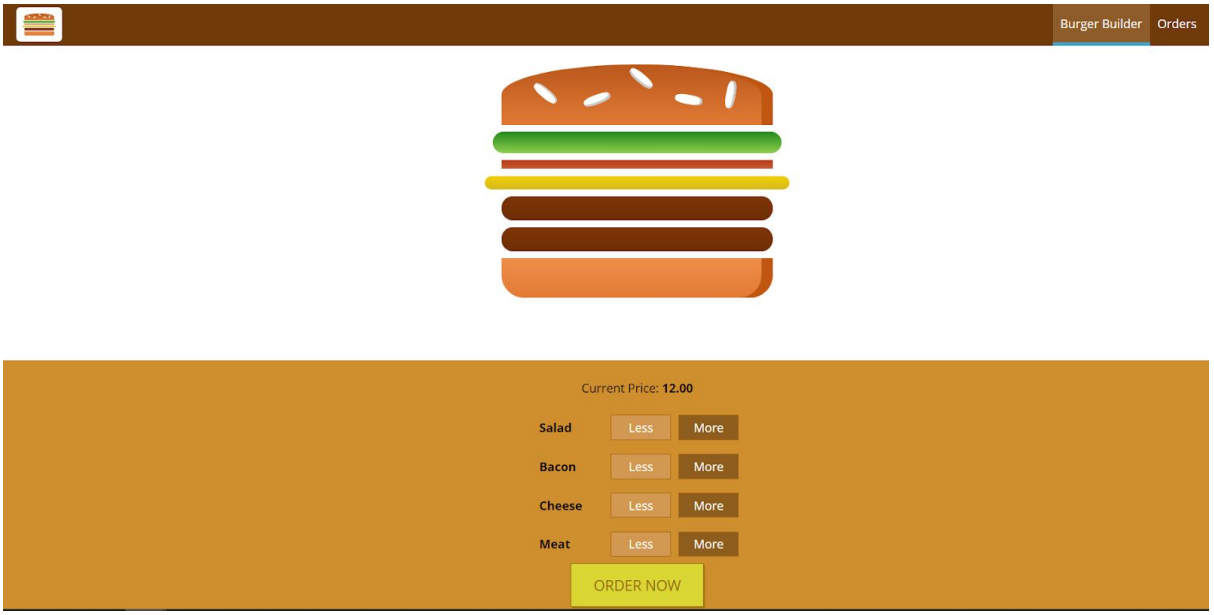

Figura 21. Vista de hamburguesa creada.

Cuando creamos la hamburguesa y hacemos clic en “Order Now”, se abrirá una ventana emergente para confirmar nuestra orden:

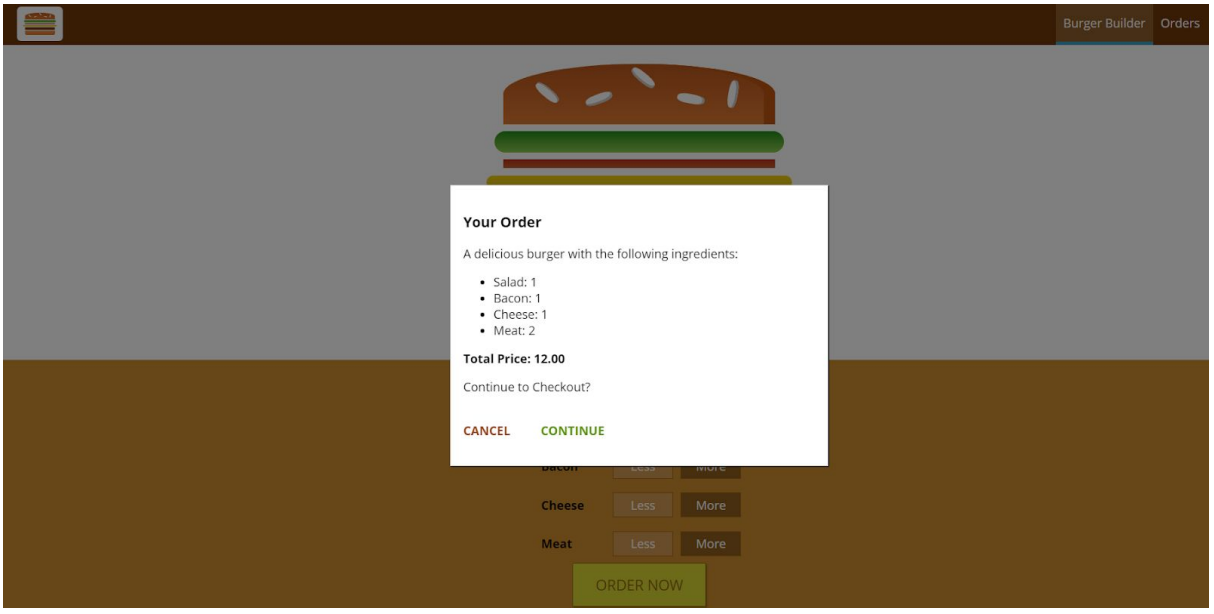

Figura 22. Ventana de confirmación de la orden.

Cuando confirmamos la compra, la aplicación nos pedirá rellenar un formulario de contacto:

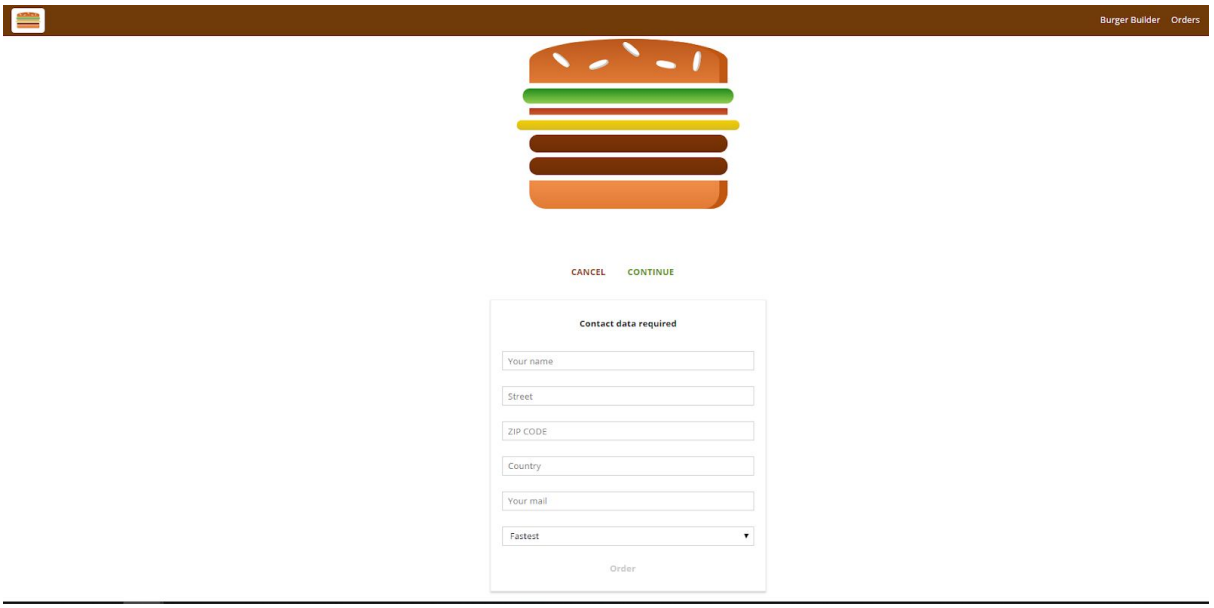

Figura 23. Vista del formulario de contacto.

Una vez hemos rellenado el formulario, se creará la orden a la cual podemos acceder desde la sección “Orders”:

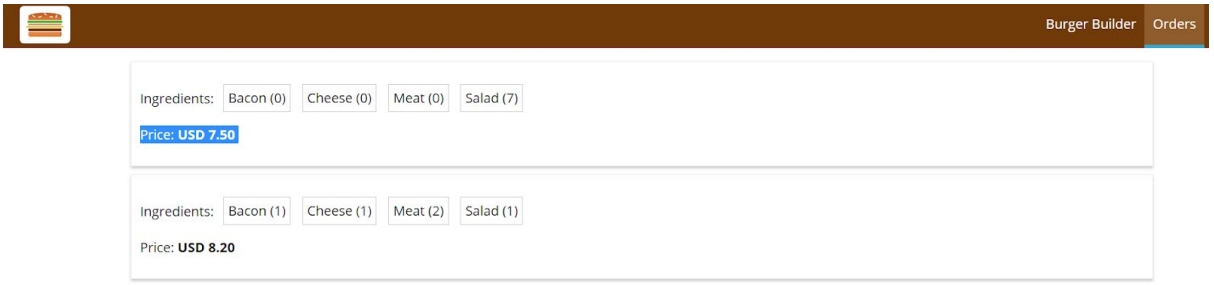

Figura 24. Vista del listado de ordenes de compra.

Conclusión

Al acabar este curso, se aprendió lo siguiente:

1. Facilidad de diseñar una aplicación en términos de componentes.
2. Estructurar adecuadamente los componentes del proyecto.
3. Aprender a desarrollar componentes, cómo gestionar su estado y ciclo de vida, recibir datos de entrada y emitir eventos.
4. Cómo conectar comunicar componentes entre sí y crear las rutas de direccionamiento (React.js).
5. Conocimientos sobre los Hooks de React [22].
6. Gestionar el estado de una aplicación utilizando Redux.

[22] Una nueva característica de React publicada en la versión 16.8 que nos permite manipular el estado y el ciclo de vida de los componentes de una manera distinta.

COMMITTEE MANAGER: UN GESTOR DE COMITÉS PARA DAOs CREADAS CON ARAGON

Dentro del contexto de las DAOs, un comité es un grupo conformado por integrantes de la DAO encargados de la gestión de operaciones específicas de la organización y con la capacidad de realizar dichas operaciones de manera ágil. Para la ejecución de estas tareas, cada comité está provisto de una serie de permisos sobre funcionalidades determinadas de algunas aplicaciones de la DAO que les permiten realizar su trabajo.

La idea detrás de estas entidades es segmentar y dividir las actividades que se desempeñan dentro de una DAO que gradualmente demandan más trabajo y atención a medida que esta crece en tamaño y complejidad, llegando a situaciones donde las responsabilidades y realización del trabajo por todos y cada uno de los integrantes es sencillamente inviable debido al coste en cuanto a tiempo y esfuerzo mental.

La aplicación debe seguir un modelo de gobernanza que plantea preguntas como:

- ¿Cómo se añaden o eliminan miembros a los comités?
- ¿Quién concede o revoca los permisos asignados al comité?
- ¿Quién puede crear o eliminar los comités?

Hay tantas respuestas a estas preguntas como la imaginación lo permita. Empezar con el desarrollo de la aplicación exigió establecer un contexto y normas específicas. Se optó por crear una DAO con un modelo de gobernanza democrático:

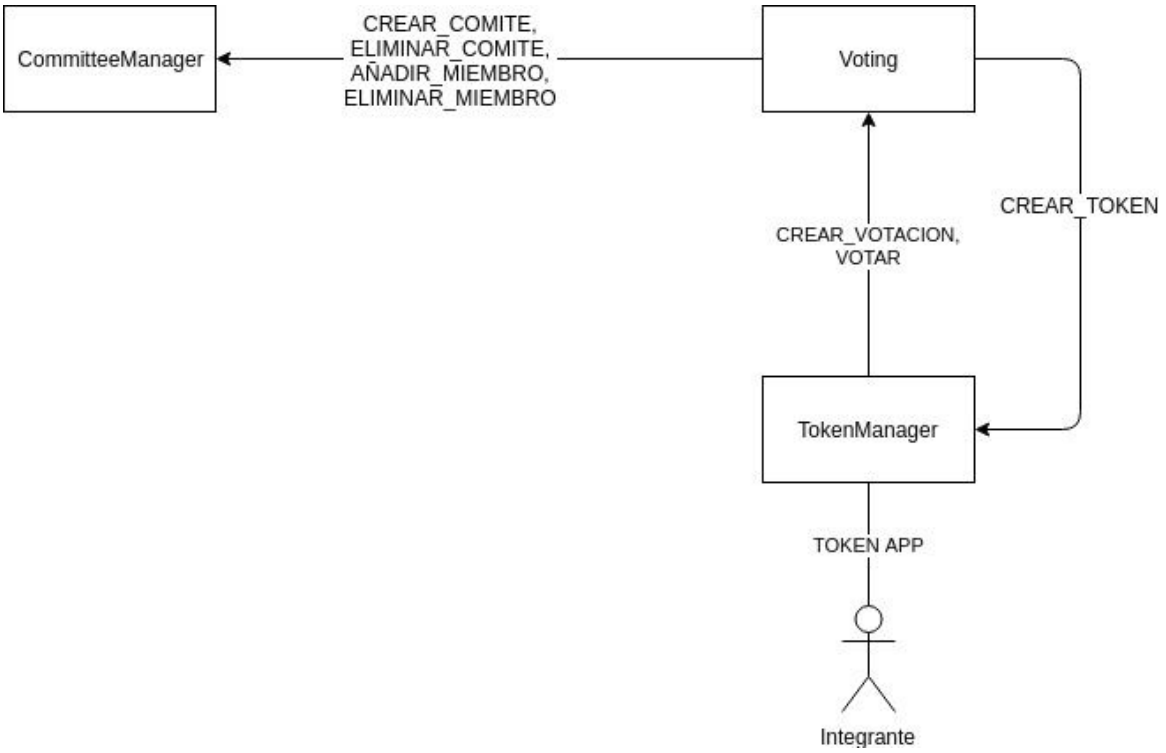

Figura 25. Esquema de gobernanza del proyecto

La organización tendrá la aplicación CommitteeManager y otras dos aplicaciones por defecto que ofrece Aragon:

1. TokenManager: App que permite crear y asignar tokens a integrantes de la DAO. Permite decidir si los tokens son transferibles y/o acumulables, y concede una serie de permisos a las entidades que los posean dependiendo de cómo esté configurada la app.
2. Voting: App que permite la creación de votaciones dentro de la DAO. Los integrantes puede abrir nuevas votaciones para decidir sobre asuntos de la DAO. Solo pueden votar aquellos que posean un token específico.

Los miembros de la DAO poseen un token APP que es único (no se puede acumular más de uno) y no transferible. Este token permite a los miembros a participar en cualquier votación que se cree en la app *Voting*.

El ingreso de nuevos miembros a la organización y por ende, la creación de nuevos tokens debe ser sometido a votación y se creará el token sólo en el caso de que la votación pase.

La creación/eliminación de comités o la adición/eliminación de los miembros deben pasar una votación general para que puedan realizarse.

La aplicación *CommitteeManager* permite crear/eliminar comités y añadir/eliminar miembros. Al crear un nuevo comité se instancia una nueva aplicación *TokenManager* y

Voting, creando un token propio del comité y un sistema de votación que solo pueden usar los dueños de ese token, es decir, los miembros del comité. Estas dos aplicaciones pueden dar lugar a dos tipos de permisos:

1. Grupales: Permisos que se asignan a la aplicación *Voting* del comité. Todas las acciones aprobadas para estos permisos deben ser sometidas a una votación de los miembros del comité.
2. Individuales: Son permisos que se asignan al token que controla el *TokenManager* del comité. Todo miembro de este comité pueden ejecutar acciones aprobadas para estos permisos sin obligación de tener que pasar una votación del comité.

Especificación

En este apartado se describen los requisitos funcionales y no funcionales de la aplicación. Se omiten los requisitos de dominio debido a que no surgió ninguno durante la fase de diseño

Los requisitos han sido identificados utilizando el siguiente formato para las etiquetas:

<Iniciales aplicación> - **<Iniciales tipo de requisito>** - **<Número de requisito>**

Así , por ejemplo, el requisito CM-RF-7 se refiere al séptimo requisito funcional de la aplicación *Committee Manager*.

Requisitos funcionales

En el siguiente subapartado se detallan todos los requisitos funcionales del proyecto acompañados por un diagrama de actividad.

Se han definido estos requisitos dentro de una tabla definiendo una serie de propiedades:

1. Prioridad: Grado de importancia del requisito. Se mide en: *Alta, Media y Baja*
2. Estabilidad: Posibilidad de que cambie el requisito en el futuro. Se mide en: *Alta, Media y Baja.*
3. Descripción: Descripción del requisito.
4. Entrada: Datos y variables que recibe la acción.
5. Salida: Datos y variables que devuelve la acción.
6. Necesita: Requisitos ajenos al sistema que se necesitan para efectuar la acción.
7. Precondición: Condiciones necesarias que requiere la acción antes de efectuarse
8. Postcondición: Condiciones necesarias que deben cumplirse una vez se haya ejecutado la acción.
9. Efectos colaterales: Efectos generados a otros componentes o sistemas.

CM-RF-1	**Crear comité**
Prioridad	Alta
Estabilidad	Media
Descripción	Se crea un comité que se almacena en el contrato ComitteeManager.

Entrada	Nombre, descripción, tipo de comité, tipo de votación, símbolo del token del comité, miembros iniciales, porcentaje de aprobación, porcentaje de quorum y duración de la votación.
Salida	-
Necesita	Conexión a la red de Ethereum. Gas.
Precondición	Los datos introducidos por el usuario cumplen el formato adecuado y son válidos. No existe un comité con el mismo nombre o símbolo del token. El usuario posee un token APP. El usuario ha creado una votación en la aplicación *Voting* que ha concluido con un mayor número de votos a favor.
Postcondición	El comité es creado dentro de la red de Ethereum
Efectos colaterales	No hay.

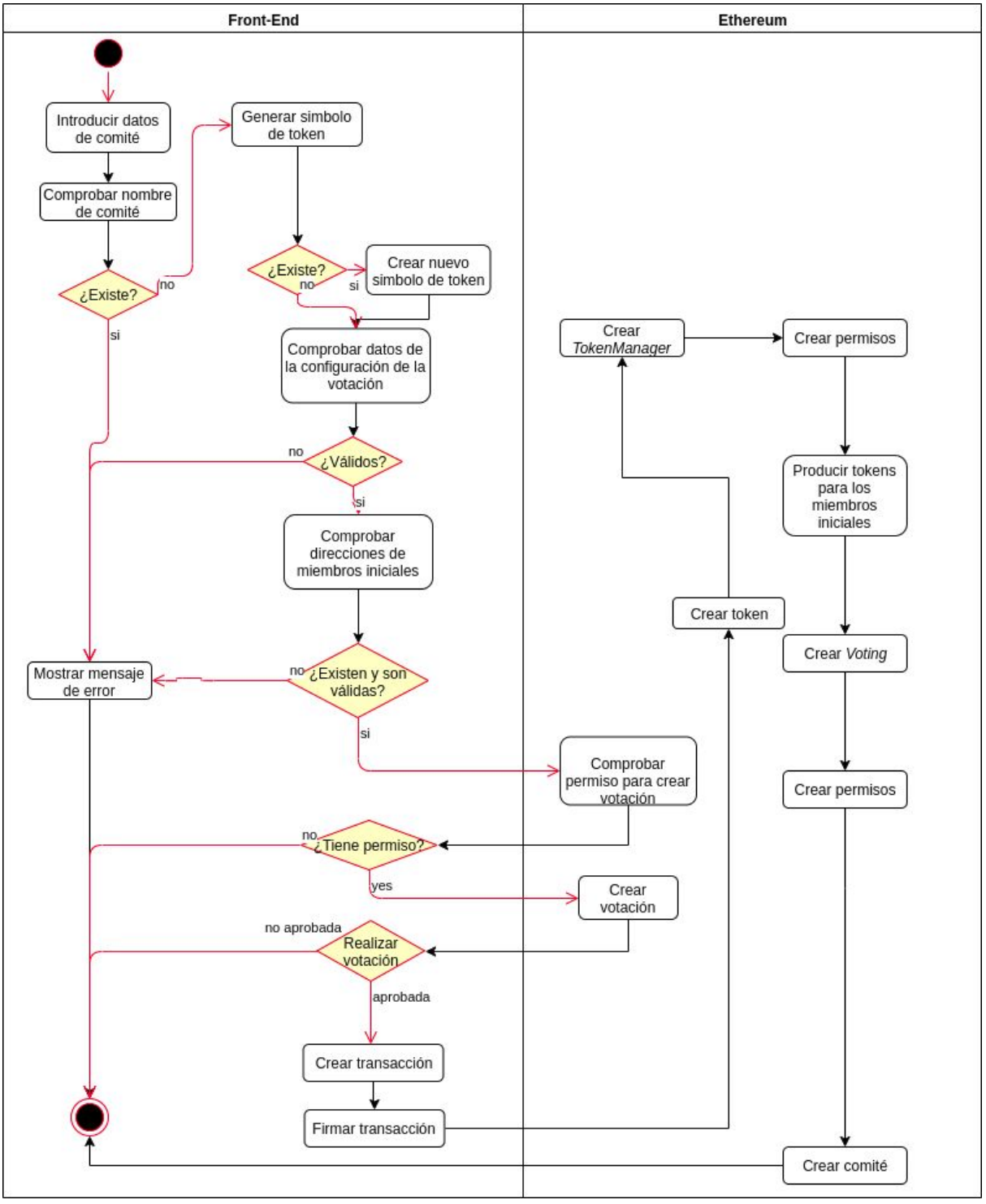

Figura 26. Diagrama de actividad de la creación de un comité.

CM-RF-2	**Eliminar comité**
Prioridad	Alta.

Estabilidad	Alta.
Descripción	Se elimina un comité del contrato CommitteeManager.
Entrada	Clave hash del comité
Salida	-
Necesita	Conexión a la red de Ethereum. Gas.
Precondición	El usuario posee un token APP. El usuario ha creado una votación en la aplicación *Voting* que ha concluido con un mayor número de votos a favor. La clave hash del comité existe.
Postcondición	El comité queda eliminado del contrato CommitteeManager.
Efectos colaterales	No hay.

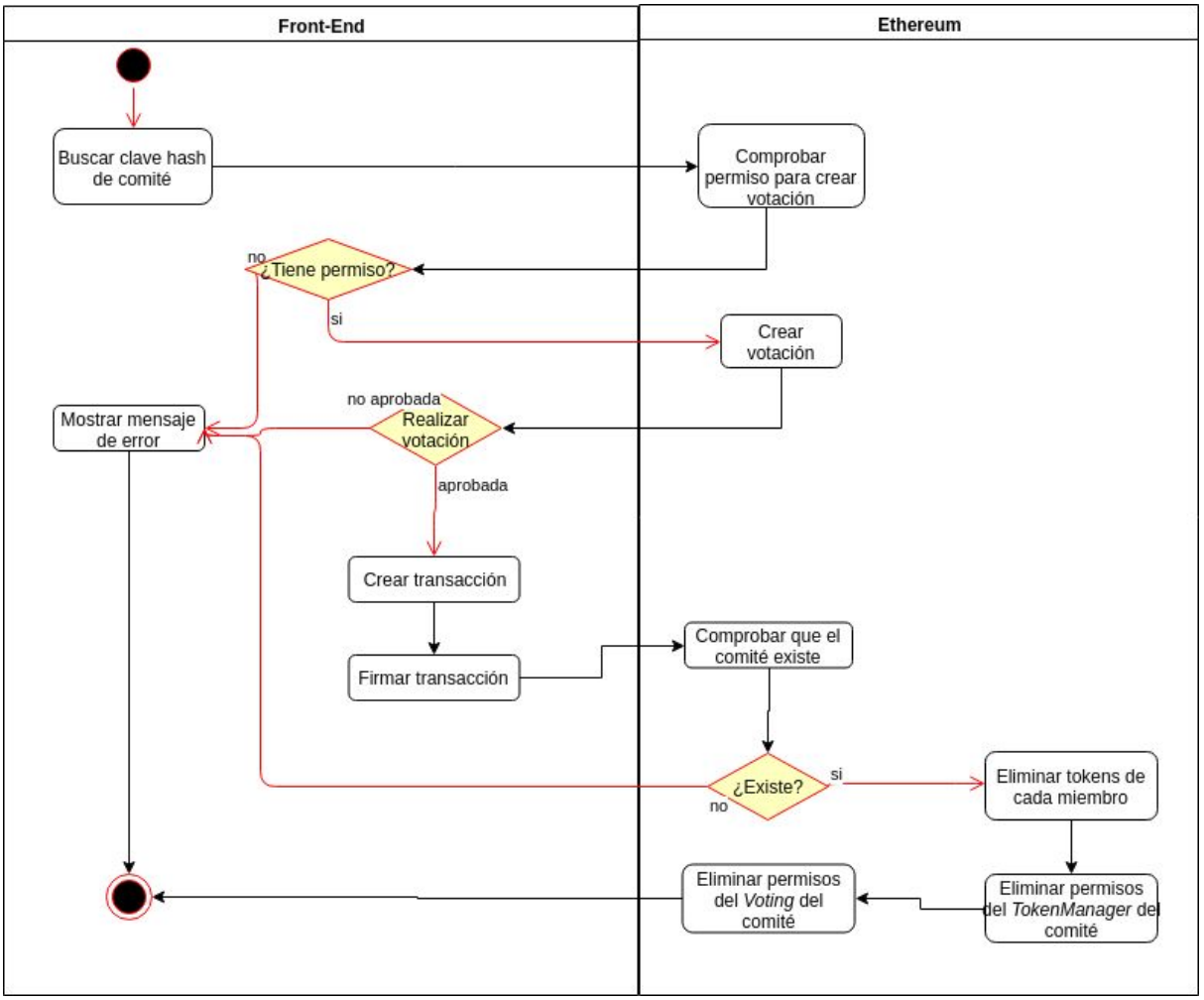

Figura 27. Diagrama de actividad de la eliminación de un comité.

CM-RF-3	**Añadir miembro a comité**
Prioridad	Alta.
Estabilidad	Alta.
Descripción	Se añade un nuevo miembro a un comité determinado.
Entrada	Clave hash del comité y dirección de la cuenta de Ethereum del nuevo miembro
Salida	-
Necesita	Conexión a la red de Ethereum. Gas.
Precondición	El usuario posee un token APP. El usuario ha creado una votación en la aplicación *Voting* que ha concluido con un mayor número de votos a favor.

	La clave hash del comité existe. La dirección del nuevo miembro es válida y existe.
Postcondición	El nuevo miembro posee un token gestionado por el *TokenManager* del comité al que ha sido añadido.
Efectos colaterales	No hay.

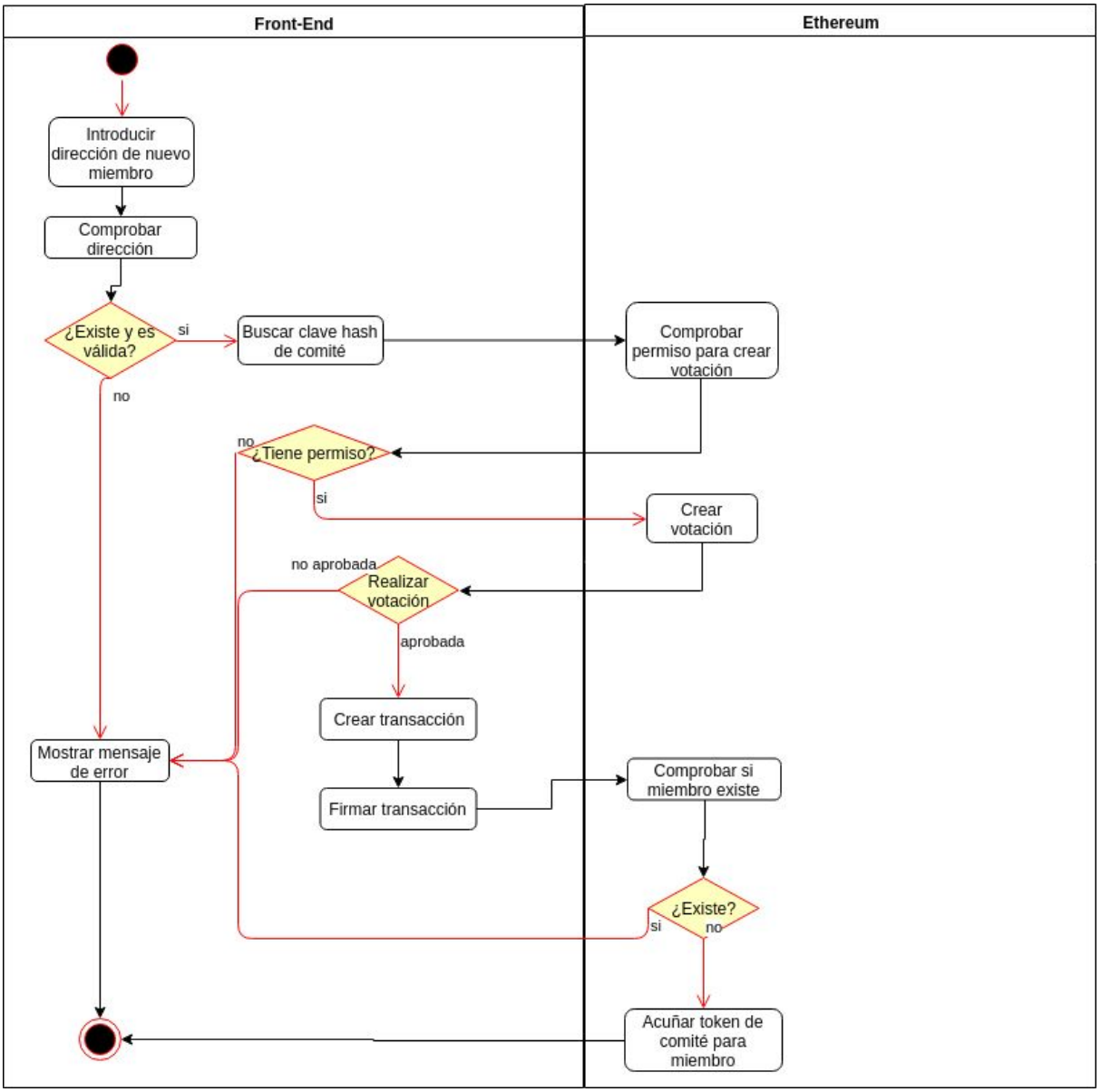

Figura 28. Diagrama de actividad de añadir miembro.

CM-RF-4	**Eliminar un miembro de un comité**
Prioridad	Alta
Estabilidad	Alta
Descripción	Se elimina un comité de la red Ethereum
Entrada	Clave hash del comité y dirección del miembro
Salida	-
Necesita	Conexión a la red de Ethereum. Ether.
Precondición	El usuario posee un token APP. El usuario ha creado una votación en la aplicación *Voting* que ha concluido con un mayor número de votos a favor. La clave hash del comité existe. La dirección del miembro es válida y existe.
Postcondición	El miembro ya no posee un token del *TokenManager* del comité.
Efectos colaterales	No hay.

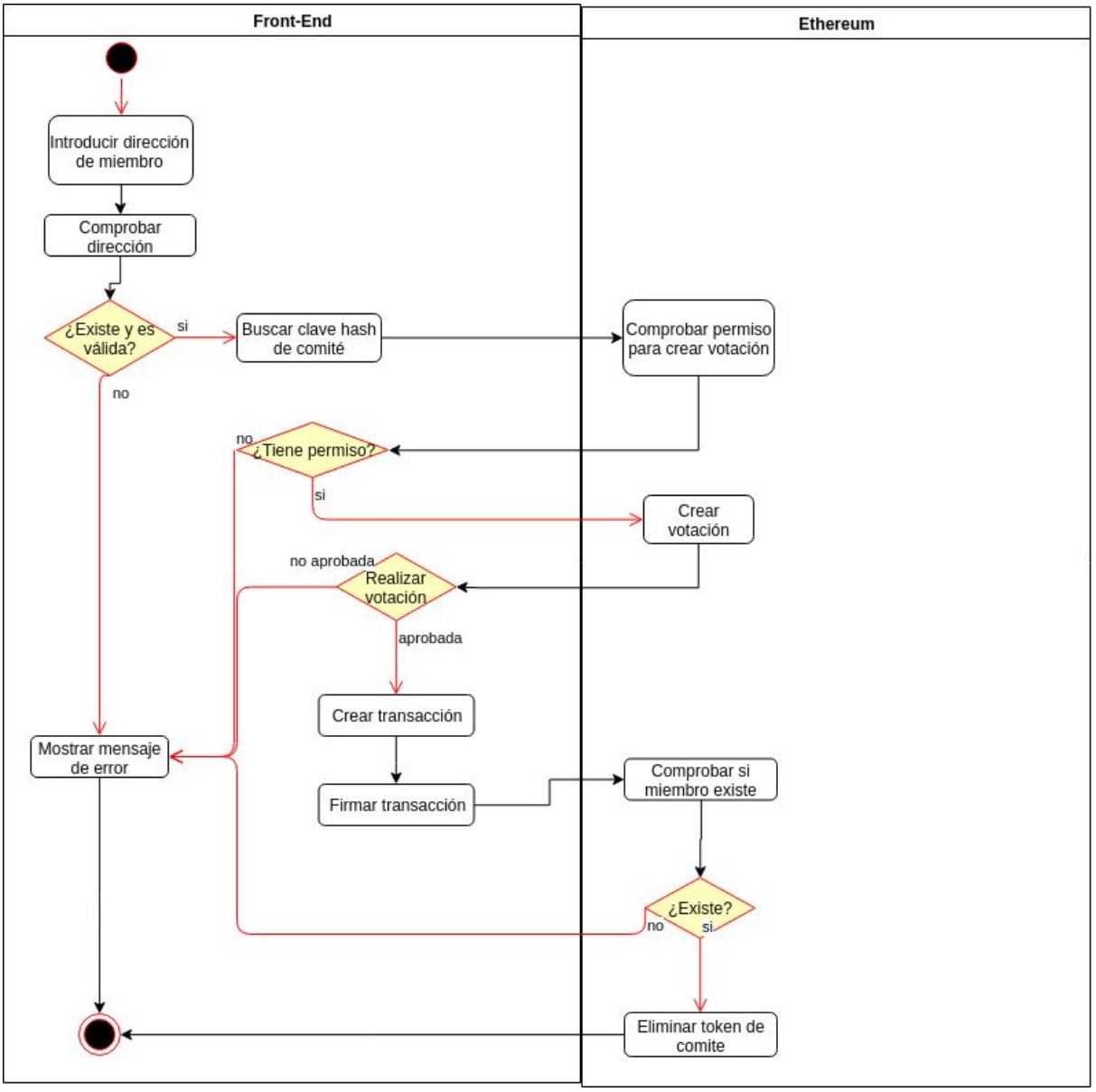

Figura 29. Diagrama de actividad de eliminar miembro.

Requisitos no funcionales

1. **Limitar el tamaño de los contratos:** Ethereum establece un límite máximo de consumo de gas para la creación de cualquier contrato. Esto restringe el número de operaciones y cálculos computacionales que se pueden realizar. Una forma de abordar este problema es modularizando la estructura de la aplicación en la red blockchain, descomponiendo contratos que crecen en tamaño en otros más pequeños que contengan lógica más específica. El desarrollo de los contratos se realizó de tal manera que se reutilizara el máximo codigo posible sin necesidad de incurrir en operaciones redundantes.
2. **Minimizar los costes de gas:** Ejecutar la lógica contenida en los contratos exige gas para pagar a los mineros la propagación del nuevo estado a lo largo de la red. El usuario tiene que desembolsar ether de su cartera para hacer frente a los gastos de gas. Es de gran importancia que el código desarrollado sea simple y eficiente para evitar, en la medida de lo posible, utilizar un alto nivel de poder de cómputo que se traduzca en un alto coste de ether.
   Cualquier tipo de lógica que involucra el procesamiento y cálculo de datos que no requiera de la blockchain, se ha hecho en el Front-End para reducir aún más los costes de gas.
3. **Minimizar almacenamiento on-chain:** El coste de guardar datos en la blockchain es muy alto. Almacenar un valor en la blockhain puede llegar a costar alrededor de 20.000 gas y actualizarlo 5.000 gas [43] (a 0,033€ y 0,008€ respectivamente al precio actual de 10 gwei/gas y 162,763€/ether). Debido a esto, el modelo de datos está conformado por tipos de variables de poco tamaño. Cualquier tipo de dato que requiera un espacio mayor puede ser almacenado en otras plataformas descentralizadas y distribuidas como IPFS, guardando en la blockchain el hash que apunta a ese dato. En versiones posteriores, se tiene pensado trasladar el almacenamiento de la descripción del comité a IPFS.
4. **Usabilidad intuitiva:** La interfaz de usuario debe caracterizarse por un diseño que facilite al usuario el aprendizaje del sistema en muy poco tiempo.

Tipos de Usuario

Los permisos de esta aplicación se han configurado para que aprovechen la escalabilidad de permisos que ofrece Aragon. Siguiendo el modelo de gobernanza democrático establecido para la DAO, todas las funcionalidades que ofrece la aplicación *CommitteeManager* están supeditadas a la aplicación *Voting* general de forma que toda acción tenga que pasar una votación previamente. Por lo tanto, el único actor dentro del contexto de la aplicación *CommitteeManager* es la aplicación de *Voting*

Casos de uso

A continuación se detallan los casos de uso de la aplicación que recoge todos los requisitos funcionales anteriormente expuestos. Como se puede observar, se sigue el modelo de gobernanza al permitir ejecutar los procesos sólo por la aplicación *Voting* general de la DAO, y gracias al transversalidad y escalado de los permisos todo usuario con un token APP puede crear votaciones en *Voting* para invocar alguna de las operaciones del *Committee Manager.*

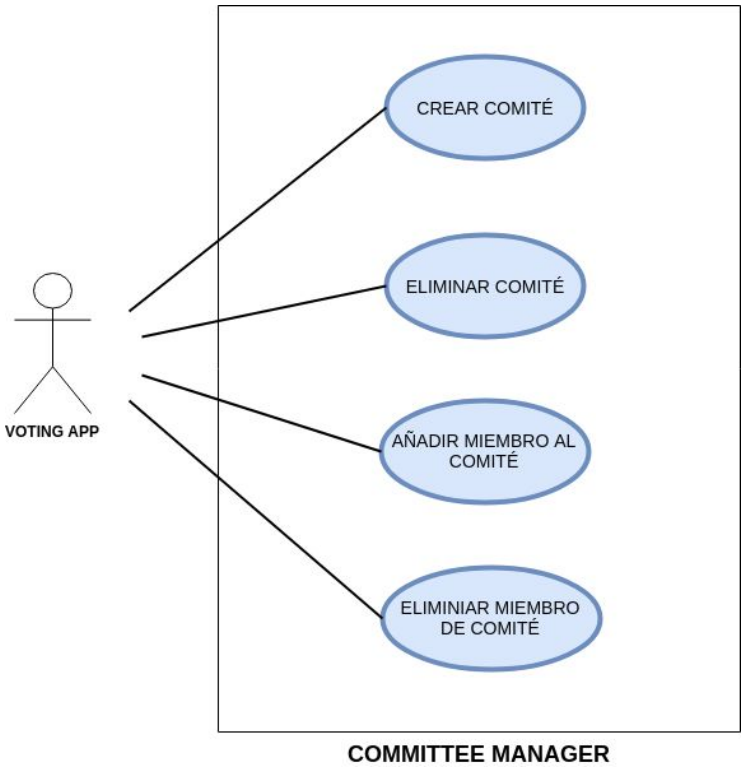

Figura 30. Diagrama de casos de uso.

Diseño general de la aplicación: Mockups

El primer paso en el desarrollo de la interfaz de usuario consistió en crear un diseño inicial de las vistas de la aplicación. Para ello, se recurrió a la herramienta web Balsamiq Cloud, que ofrecía diseños estéticamente superiores a los de otras herramientas, una interfaz intuitiva y un fácil acceso a la aplicación.

Se establecieron una serie de pautas principales a la hora de elaborar el diseño:

1. El diseño tenía que ser coherente con el estilo que Aragon había seguido en sus aplicaciones, tanto en el uso de los colores y su tonalidad como en el estilo de los componentes utilizados o creados.
2. La interfaz debe ser lo más intuitiva posible y no abrumar al usuario con una excesiva cantidad de botones o elementos de control.
3. Debido a que el contacto con los contratos en la red de Ethereum se realiza por medio de llamadas asíncronas que pueden tener una duración considerable hasta devolver el control de la aplicación al usuario, era importante mantener informado al usuario durante la ejecución de las transacciones de alguna forma.

A continuación se muestran los mockups elaborados:

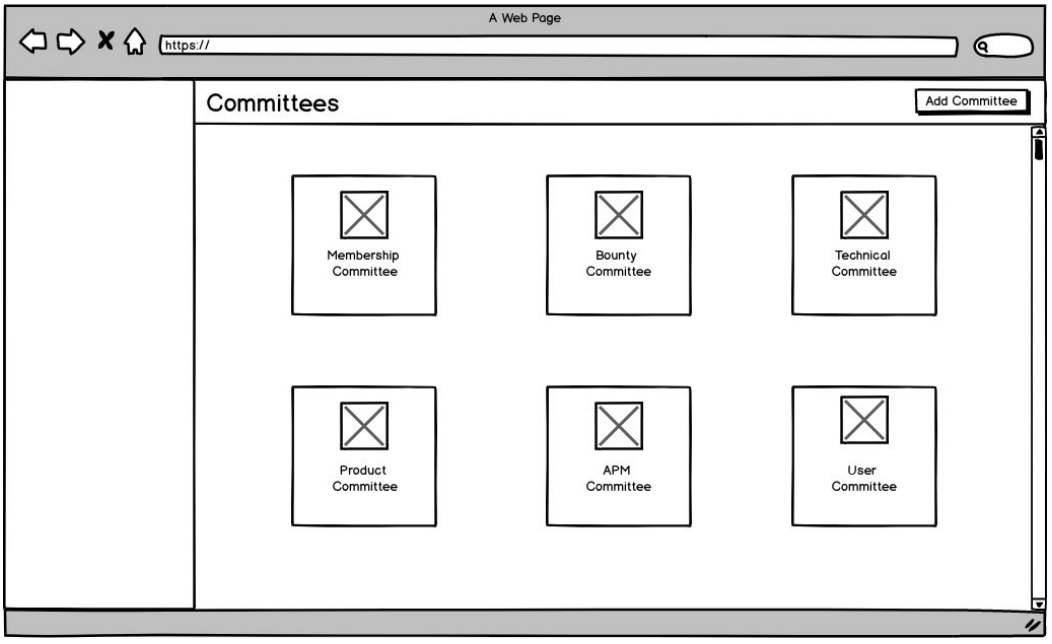

Figura 31. Vista principal de Comittee Manager

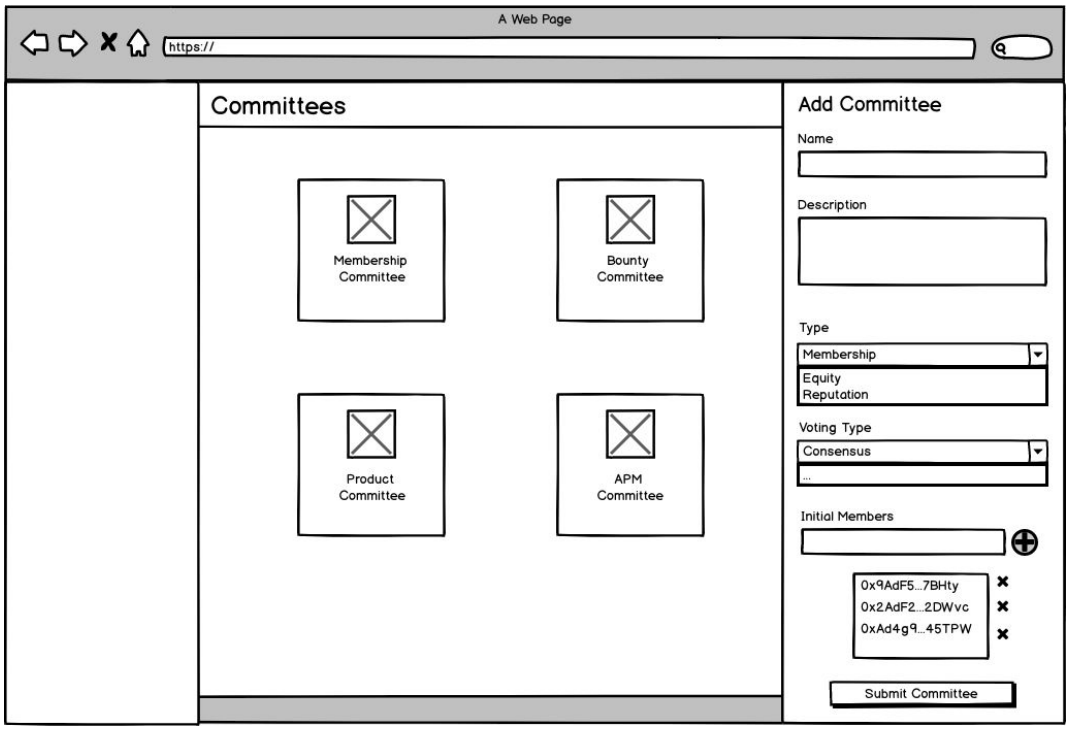

Figura 32. Vista del panel de creación de un comité

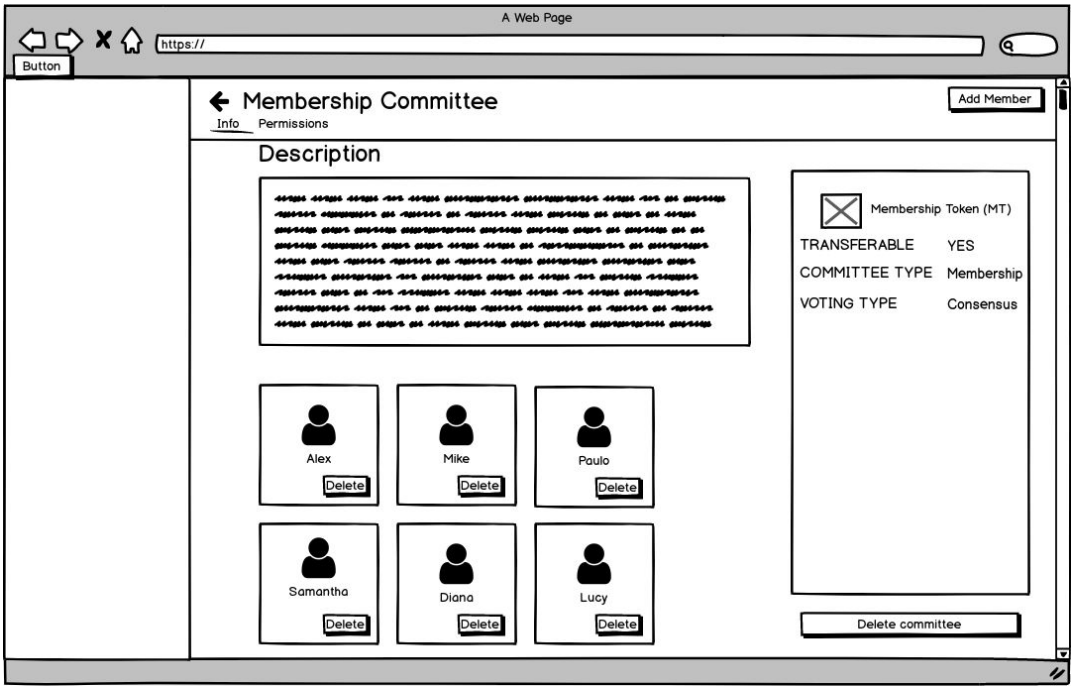

Figura 33. Vista del comité

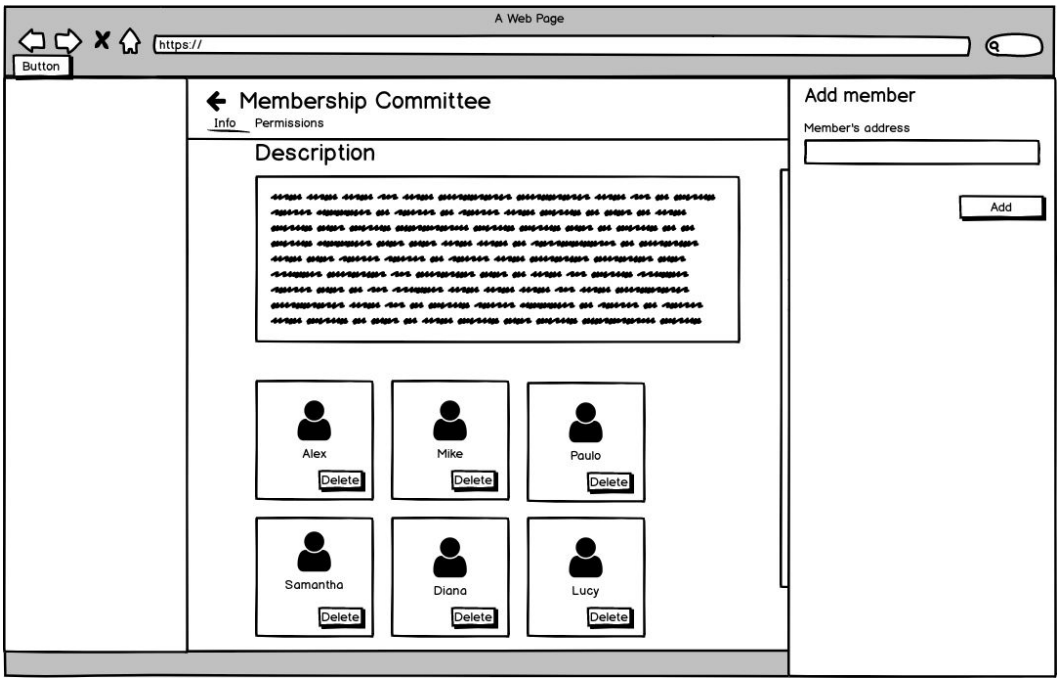

Figura 34. Vista del panel de nuevo miembro

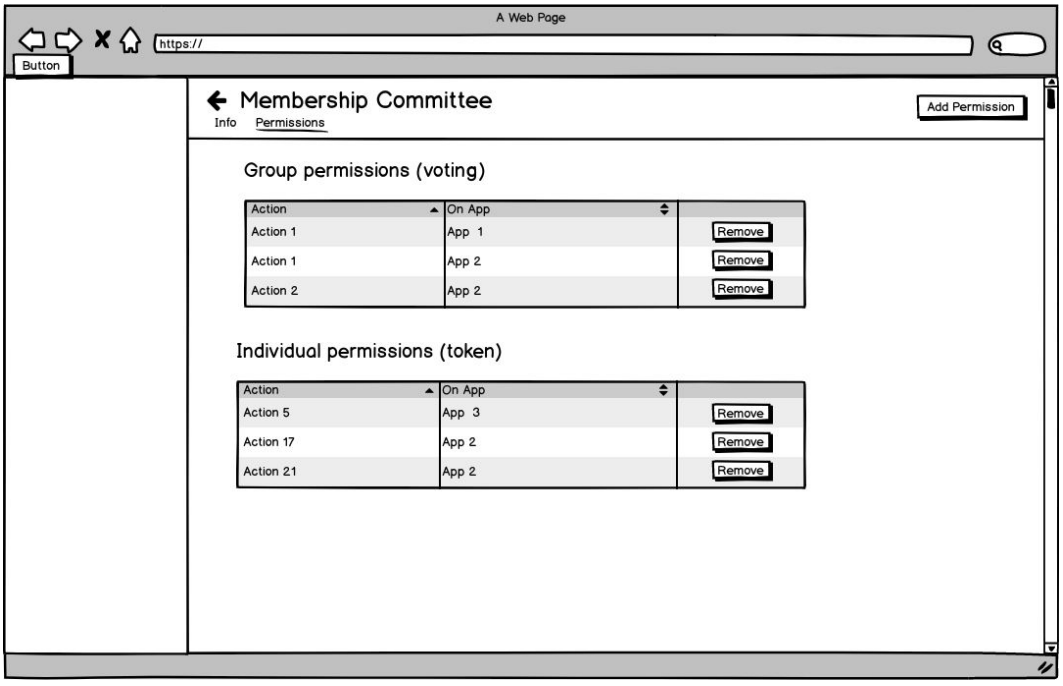

Figura 35. Vista de los permisos del comité

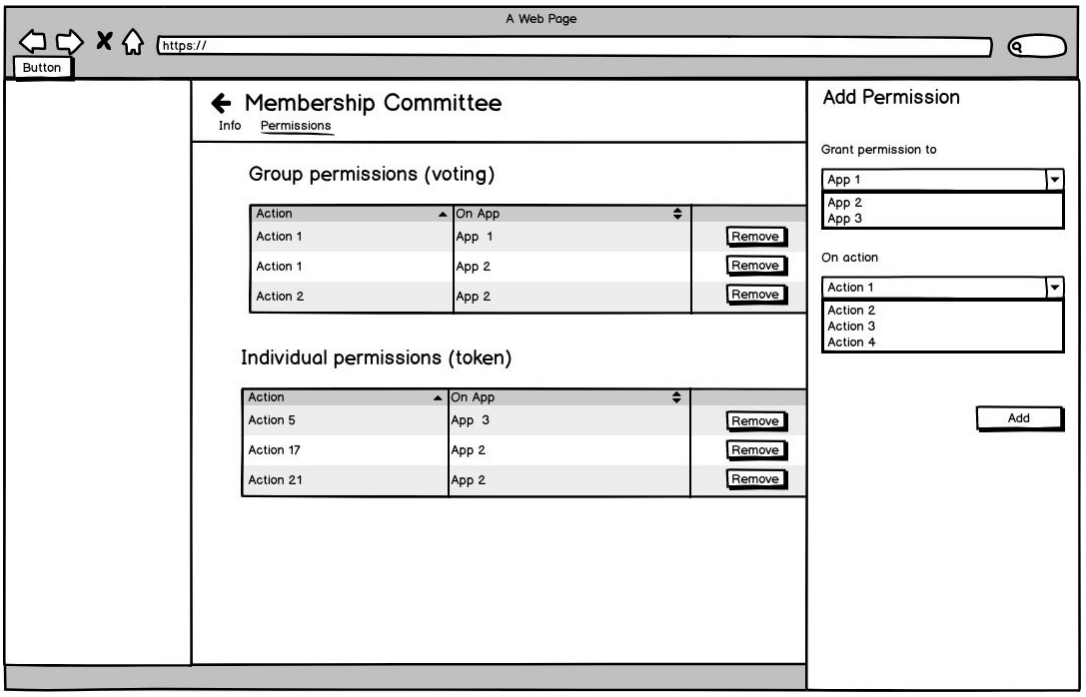

Figura 36. Vista del panel de nuevo permiso

Implementación

A continuación se describe la puesta en marcha de la aplicación una vez ya preparados los diseños y establecida la metodología de desarrollo y gestión del trabajo. Se expone la estructura principal de la aplicación profundizando en cada uno de los elementos que la componen.

Estructura principal

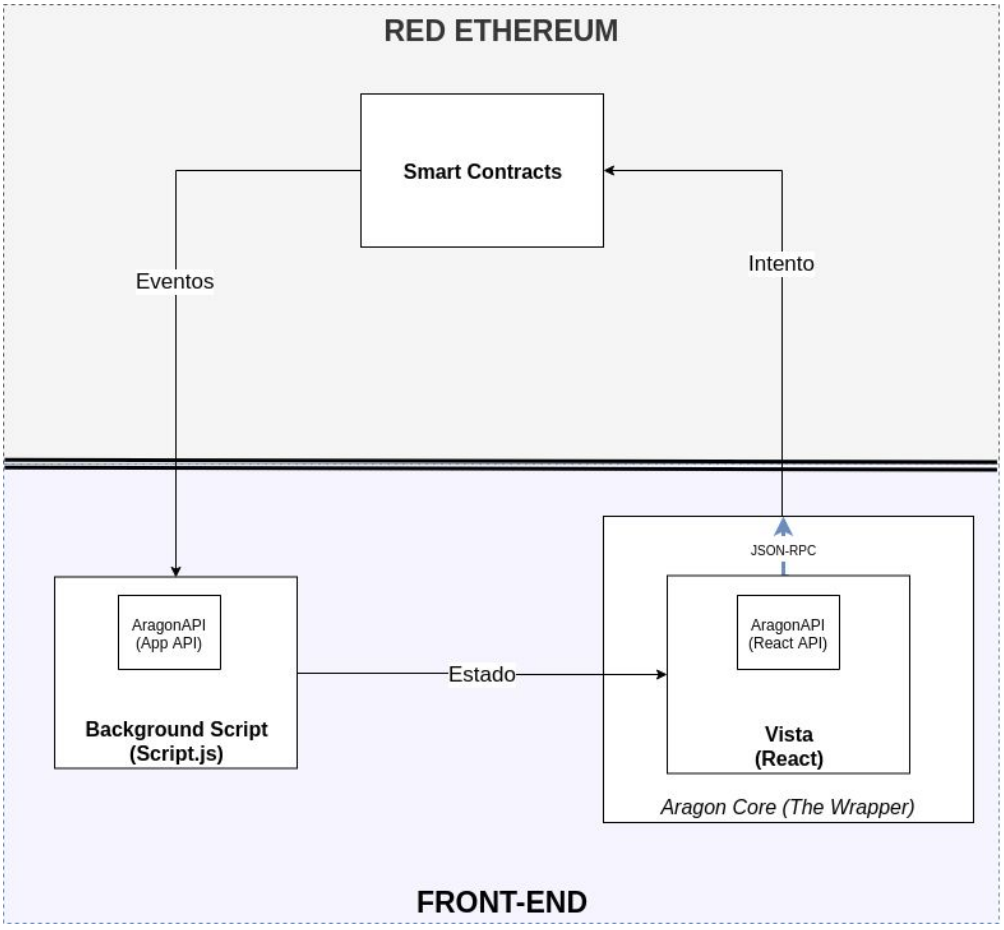

Figura 37. Estructura principal del *Committee Manager*

A grandes rasgos, podríamos organizar la aplicación en tres componentes principales: Smart Contracts, Background Script y la Vista.

El flujo de información principal de la aplicación se define de la siguiente forma: la vista se comunica con los contratos por medio del módulo AragonAPI, específicamente utilizando React API, que emite una petición por medio del protocolo JSON-RPC a Aragon Core, que se comunica con el contrato de la aplicación para invocar algunos de sus métodos, a esto se le llama “intento”. Una vez el contrato haya terminado de ejecutar el método, emite un evento que es capturado por el *Background Script*

Smart Contracts

En total, la aplicación contiene cuatro contratos distribuidos en dos ficheros .sol principales:

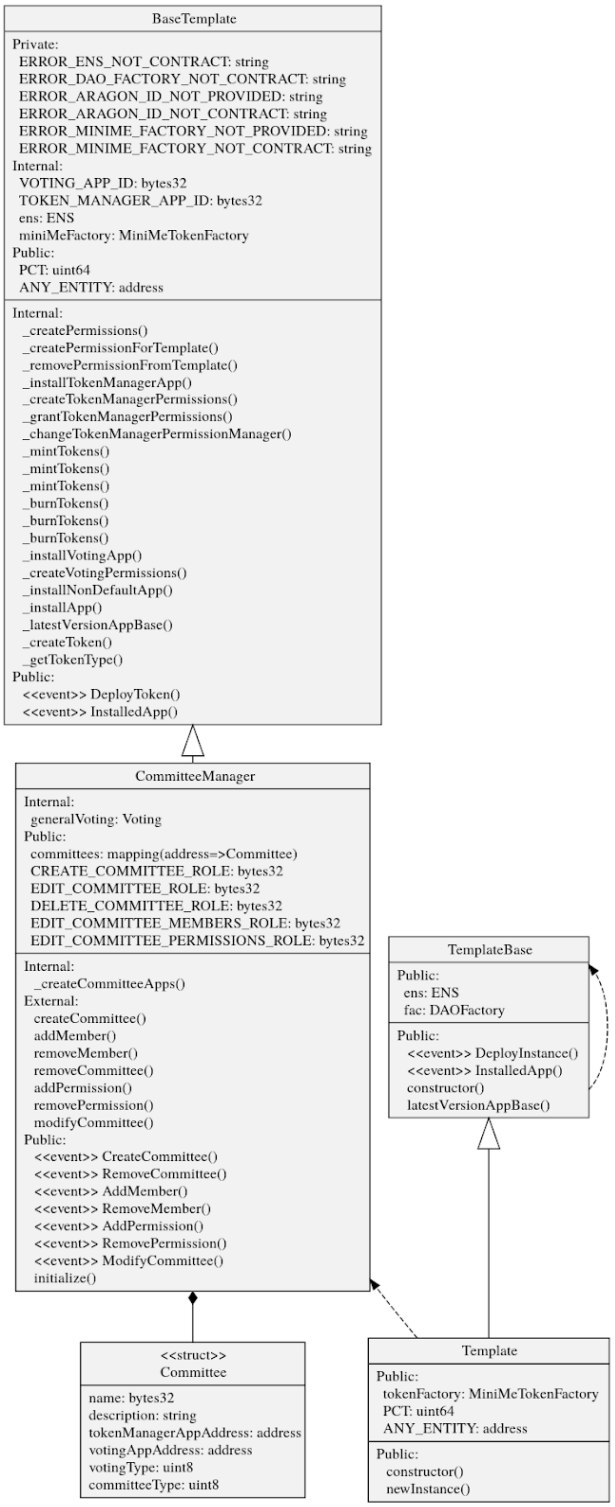

Figura 38. Diagrama UML de los contratos de la aplicación

Template.sol

Este fichero viene por defecto en todo proyecto de Aragon que utilice *templates* para crear las DAOs. Dentro de este, se implementan dos contratos:

TemplateBase: Este contrato ofrece tres funcionalidades básicas:

1. Acceder al DAOFactory, un contrato que abstrae al desarrollador del mecanismo de creación de una DAO, desde instanciar el Kernel y el ACL hasta configurar sus respectivos permisos.
2. Buscar y devolver la dirección del contrato de la última versión de cualquier aplicación que se le solicite. Para ello se le pasa el nombre hasheado de la aplicación, por ejemplo: *committee-manager.aragonid.eth*.

   Esto se utiliza para obtener las aplicaciones que queremos instalar en nuestra DAO.

   Por último, cabe mencionar que para obtener las versiones de las aplicaciones, el contrato utiliza el ENS (Ethereum Name Service) que comentamos en capítulos previos.

Template: Este contrato hereda de TemplateBase, por lo que podemos acceder a las funcionalidades previamente descritas. Se encarga de crear y configurar la DAO, contiene la lógica principal de este proceso. El mecanismo de creación se puede resumir en los siguientes pasos:

1) Crea una nueva DAO utilizando DAOFactory
2) Asigna al contrato Template dos permisos necesarios para configurar la DAO:
   a) APP_MANAGER_ROLE: Permite instanciar y asignar nuevas aplicaciones a la DAO.
   b) CREATE_PERMISSIONS_ROLE: Permite crear nuevos permisos en el ACL de la DAO.
3) Crea instancias de las aplicaciones *CommitteeManager, Voting*, *TokenManager* y el token APP. Las aplicaciones *Voting* y *TokenManager* regulan los permisos grupales e individuales (a través del token APP) que se tengan sobre cada uno de los comités creados con *CommitteeManager*.
4) Inicializa las aplicaciones anteriormente creadas.
5) Crea y configura los permisos para cada una de las aplicaciones, así como el manager de cada permiso. Todas las operaciones sobre *CommitteeManager* (crear, eliminar comités, etc) solo pueden ser invocadas por la aplicación de *Voting* por medio de votaciones en las que pueden participar todos aquellos con un token APP.
6) Revoca los permisos asignados al contrato Template en el paso 2 y reasignalos al *CommitteeManager* para que pueda crear una nueva instancia y configurar los permisos de las aplicaciones *Voting* y *TokenManager* para cada comité nuevo que se cree.

CommitteeManager.sol

Este fichero contiene los contratos propios de la aplicación. Para optimizar el consumo de gas se procuró utilizar en la medida de lo posible variables de tipo bytes.

BaseTemplate: El procesos de creación y despliegue de las aplicaciones de *Voting* y *TokenManager* puede ser costoso en términos de gas y ocupar muchas líneas de código,

algo que se tiene que tener en cuenta ya que Ethereum limita la máxima cantidad de gas que se puede consumir, el tamaño de los contratos o incluso las variables locales que utilicemos dentro de una función (permite un máximo de 16). Para solventar estos problemas, se decidió crear un contrato que permitiese organizar mejor el código y abstraer al ComitteeManager de todas las operaciones relativas a la configuración de las otras aplicaciones, y que este solo se limitase a llamar a los métodos de BaseTemplate al crear un nuevo comité. El contrato también ofrece funciones auxiliares para comunicarse con el Kernel o ACL para facilitar, asignar o eliminar permisos a los comités creados.

El contrato necesita de otros dos contratos para poder operar:

1. ENS: Lo necesitamos para poder mapear los nombres de las aplicaciones de *Voting* y *TokenManager* a las direcciones de sus contratos para poder instanciar ambos cuando creemos un nuevo comité.
2. MiniMeTokenFactory: Un contrato que actúa como factoría que permite crear cualquier token para la DAO. En nuestro caso, lo utilizaremos para crear el token APP. Originalmente esta factoría se iba a desplegar en conjunto con el contrato CommitteeManager, pero esto demandaba un gran coste de gas y aumentaba el tamaño del código del contrato más allá de los límites permitidos en Ethereum, por lo que se consideró que era preferible pasar el contrato desde Template al instanciar el CommitteManager.

CommitteeManager: Se podría decir que es el contrato más importante de la aplicación al encerrar la lógica principal de la misma. Hereda de BaseTemplate para poder acceder a todas las funciones anteriormente descritas y de AragonApp debido a que es obligatorio para el contrato de cualquier aplicación de Aragon.

Uno de los primeros puntos de discusión acerca del diseño de la aplicación fue el de decidir qué permisos debe tener esta. En un principio, se consideraron tres permisos: crear, modificar y eliminar un comité. No obstante, se observó que estos permisos no eran lo suficientemente granulares y quizás habría que dividirlos en otros. Tras un estudio más minucioso, se llegó a los siguientes permisos:

1. Crear comité.
2. Editar comité.
3. Eliminar comité.
4. Editar los miembros de un comité.
5. Editar los permisos de un comité.

A modo de ejemplo, a continuación se muestra como se declara uno de estos permisos en el contrato:

```
bytes32 constant public CREATE_COMMITTEE_ROLE = keccak256("CREATE_COMMITTEE_ROLE");
bytes32 constant public EDIT_COMMITTEE_ROLE = keccak256("EDIT_COMMITTEE_ROLE");
bytes32 constant public DELETE_COMMITTEE_ROLE = keccak256("DELETE_COMMITTEE_ROLE");
bytes32 constant public EDIT_COMMITTEE_MEMBERS_ROLE = keccak256("EDIT_COMMITTEE_MEMBERS_ROLE");
bytes32 constant public EDIT_COMMITTEE_PERMISSIONS_ROLE = keccak256("EDIT_COMMITTEE_PERMISSIONS_ROLE");
```

Figura 39 . Declaración de permisos del contrato *CommitteeManager.*

Podemos observar que los permisos se declaran de tipo *bytes32* en vez de *string* para que ocupen menos espacio en la blockchain. La idea es “hashear” el string que describe el permiso, es decir, pasar como parámetro el permiso a una función hash (en este caso keccak256) que nos devolverá una cadena de 32 bytes fija que es mucho más concisa y fácil de manipular.

Dentro del contrato se define el comité de la siguiente manera:

Committee: Es un *struct* que representa el comité. Se definen en él una serie de datos que caracterizan un comité:

a. Name: Contiene el nombre del comité. Es de tipo *bytes32*.
b. Description: Contiene la descripción del comité, es decir, una breve explicación de los objetivos y funciones del comité. Es de tipo *string* debido a que es muy probable que contenga cadenas de texto de más de 32 bytes.
c. VotingType: Es un número que especifica el tipo de votación que posee el comité. El tipo de votación determina tres valores:
    i. El porcentaje de aprobación necesario para que pase la votación.
    ii. El porcentaje de quórum.
    iii. La duración de la votación (en días).

    Se han creado cuatro tipos de votación:

Tipo	Aprobación (%)	Quórum (%)	Duración (días)	Codificación
Consenso	99	99	30	0
Mayoría absoluta	51	51	30	1
Mayoría simple	51	15	30	2
Personalizada	-	-	-	3

    La última columna indica el valor que representa el tipo de votación dentro de la variable. VotingType. La última votación es configurable completamente por el usuario.

    Sabiendo que el rango de valores para este campo no es muy amplio y que se necesita reducir lo máximo posible el uso de memoria en la

blockchain, se acordó utilizar *uint8,* el entero más pequeño que permite Solidity.

d. CommitteeType: Es un número que especifica el tipo de comité . Se utiliza para configurar el tipo de token que el comité posee. Los tipos de token se caracterizan por tres aspectos:
   i. Transferible: El token puede ser traspasado a otra persona.
   ii. Único: Determina si el token es acumulable.
   iii. Decimales: Determina el número de decimales.

   Se han creado tres tipos de tokens:

Tipo	Transferible	Acumulable	Codificación
Membership	NO	NO	0
Equity	SI	SI	1
Reputation	NO	SI	2

   El tipo de este campo es *uint8*, por las mismas razones expuestas en el campo previo. En el código, estos campos se han declarado uno después otro, ya que la EVM (Máquina Virtual de Ethereum) empaqueta mejor los campos de un struct del mismo tipo, reduciendo el tamaño que debe reservar para estos en la blockchain.

e. TokenManagerAppAddress: Contiene la dirección del contrato de la aplicación *TokenManager* del comité. Necesario a la hora de conceder o revocar futuros permisos individuales al comité.
f. VotingAppAddress: Contiene la dirección del contrato de la aplicación *Voting* del comité. Necesario a la hora de conceder o revocar futuros permisos grupales al comité.

El contrato posee únicamente dos atributos:

1. Committees: Mapping que contiene los comités. Tiene como clave la dirección del contrato de la aplicación *TokenManager* del comité al que apunta, y como valor el *struct* Committee. Se ha escogido la dirección del *TokenManager* ya que no pueden existir dos iguales, aunque tambien se podia haber elegido la dirección del contrato de la aplicación *Voting*.
2. GeneralVoting: Referencia al contrato de la aplicación *Voting* general de la DAO. Es necesario tener a disposición este contrato para concederle a la votación general permisos de crear o quemar tokens sobre la aplicación *TokenManager* y crear votos o cambiar la configuración de la aplicación *Voting* (porcentaje de aprobación, duración de la votación y porcentaje de quórum) de cada comité. De esta forma,

queda a elección de todos los miembros decidir quienes conforman el comité en todo momento y cómo deben votar estos para aprobar o rechazar propuestas.

Otro aspecto importante del contrato son los eventos, que constituye la principal forma de comunicación con el Front-End de la aplicación para notificar qué ha cambiado y qué acciones han afectado a la blockchain. Debido a que la emisión de eventos supone un coste muy bajo en términos de gas, se decidió emitir uno por cada funcionalidad del contrato. Por lo tanto, tendríamos los siguientes:

1. CreateCommittee, RemoveCommittee y ModifyCommittee: Estos eventos se disparán al crear, modificar o eliminar un comité.
2. AddMember y RemoveMember: Estos eventos se disparan al añadir o eliminar miembros.
3. AddPermission y RemovePermission: Estos eventos se disparan al añadir o eliminar permisos al comité.

La idea es que una vez el contrato haya acabo de ejecutar la lógica de la funcionalidad que se haya invocado, se emite el evento correspondiente.

Podría decirse que la lógica más importante del contrato se encuentra en la función *_createCommitteeApps(...)*, la cual se encarga de poner en marcha la creación de las aplicaciones de *Voting* y *TokenManager* que requieren los comités para funcionar adecuadamente, invocando a los métodos de BaseTemplate necesarios para configurar y desplegar las aplicaciones. El número de parámetros que se le pasaban a esta función eran tantos que sobrepasan el límite permitido por Solidity, se tuvo que agrupar parámetros del mismo tipo dentro de arrays de tamaño fijo para reducir el número de variables

```
function _createCommitteeApps(string _committeeTokenSymbol, address[] _initialMembers,
    uint64[3] _votingInfo, uint8 _tokenType) internal returns (address tmAddress, address vAddress)
```

Figura 40 .Declaración de la función *_createCommitteeApps*

Background Script

Aragon implementa una arquitectura Flux [45] para la gestión y control del flujo de datos a lo largo de la interfaz de usuario. Es una solución que busca facilitar el desarrollo de aplicaciones que cada vez van creciendo en complejidad y tamaño. Este componente, representado por el fichero Script.js en la aplicación, busca implementar esta arquitectura al establecer un flujo de datos unidireccional.

La estructura que propone flux es la siguiente:

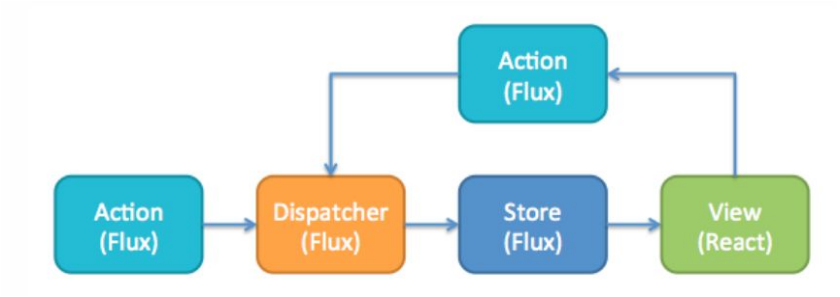

Figura 41. Flujo de datos en la arquitectura Flux. [45]

En el proyecto, tendríamos el siguiente flujo de datos:

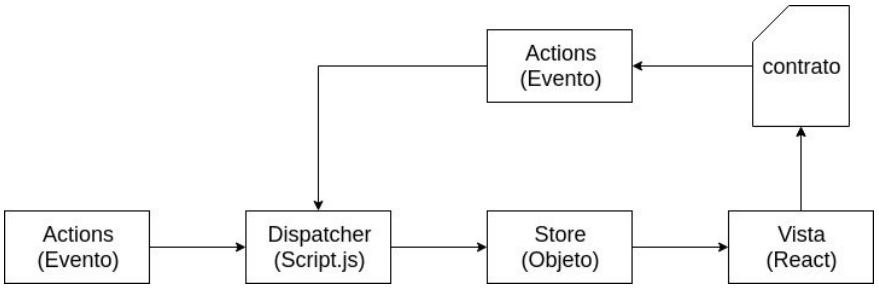

Figura 42. Flujo de datos en Aragon

La interacción del usuario con el contrato a través de alguna de las vistas resulta en la emisión de un evento que representaría una acción, ya sea crear/eliminar un comité o añadir/eliminar un miembro. El evento es captado por una función llamada reducer (reductor) que se encuentra en el fichero Script.js y se importa del módulo AragonAPI, cuyo objetivo es producir un nuevo estado de la aplicación dada una acción y el estado previo. El nuevo estado se guarda en el store. A continuación, la vista accede al store utilizando ciertas funciones de AragonAPI y actualiza la interfaz de usuario de acuerdo al nuevo estado.

Aragon ejecuta el fichero Script.js en un Web Worker[23] para que opere en segundo plano. La función reducer también puede recibir cambios que no provienen necesariamente del contrato, ajenos a la aplicación en sí.

Vista

[23] Los Web Workers constituyen una forma de ejecutar scripts en hilos separados del hilo principal de la aplicación web para no bloquear o ralentizar su ejecución.

La vista de la aplicación desarrollada con el framework React posee en gran medida una estructura semejante a la de un proyecto de React cualquiera.

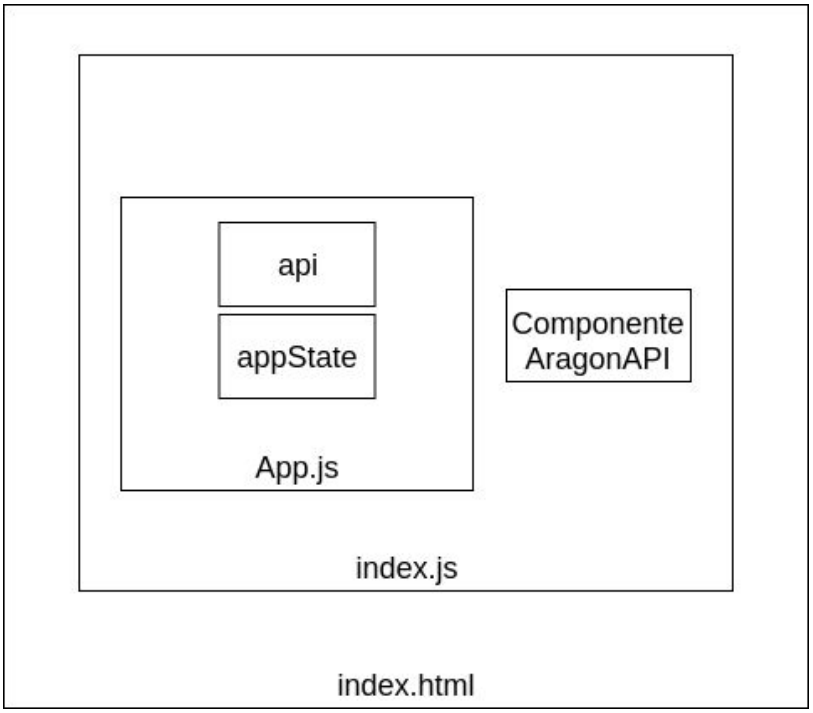

Figura 43. Estructura del Front-End

El punto de entrada de la aplicación es index.html, el fichero html principal que importa el fichero index.js, el cual se encarga de comunicar nuestra aplicación de React con los contratos utilizando la función reductora descrita previamente para crear el estado de la aplicación. La forma de conectar ambas partes es envolviendo el componente general de la aplicación *App* que proviene del fichero App.js en el componente *AragonAPI* que ofrece el módulo homónimo. A continuación, le indicamos a React que es posible renderizar la aplicación.

```
ReactDOM.render(
  <AragonApi reducer={reducer}>
    <App />
  </AragonApi>,
  document.getElementById('root')
)
```

Figura 44.Trozo de código del fichero index.js

El módulo AragonAPI ofrece dos objetos al Front-End, una vez ambos se hayan conectado:

1. *appState*: Objeto que contiene el estado de la aplicación que produce la función reductora del Background Script. Podemos acceder a los eventos o atributos públicos del contrato.
2. *api*: Objeto que nos permite invocar a las funciones del contrato de la aplicación. Por ejemplo, si queremos invocar al método que elimina un comité, lo haríamos de la siguiente manera:

```
api.removeCommittee(address, members).subscribe(() => {
  navigationBackHandler()
})
```

Figura 45. Ejemplo de uso del *api.*

Para el desarrollo de los componentes de React que conforman las interfaces de usuario de la aplicación se siguió el siguiente diagrama de componentes:

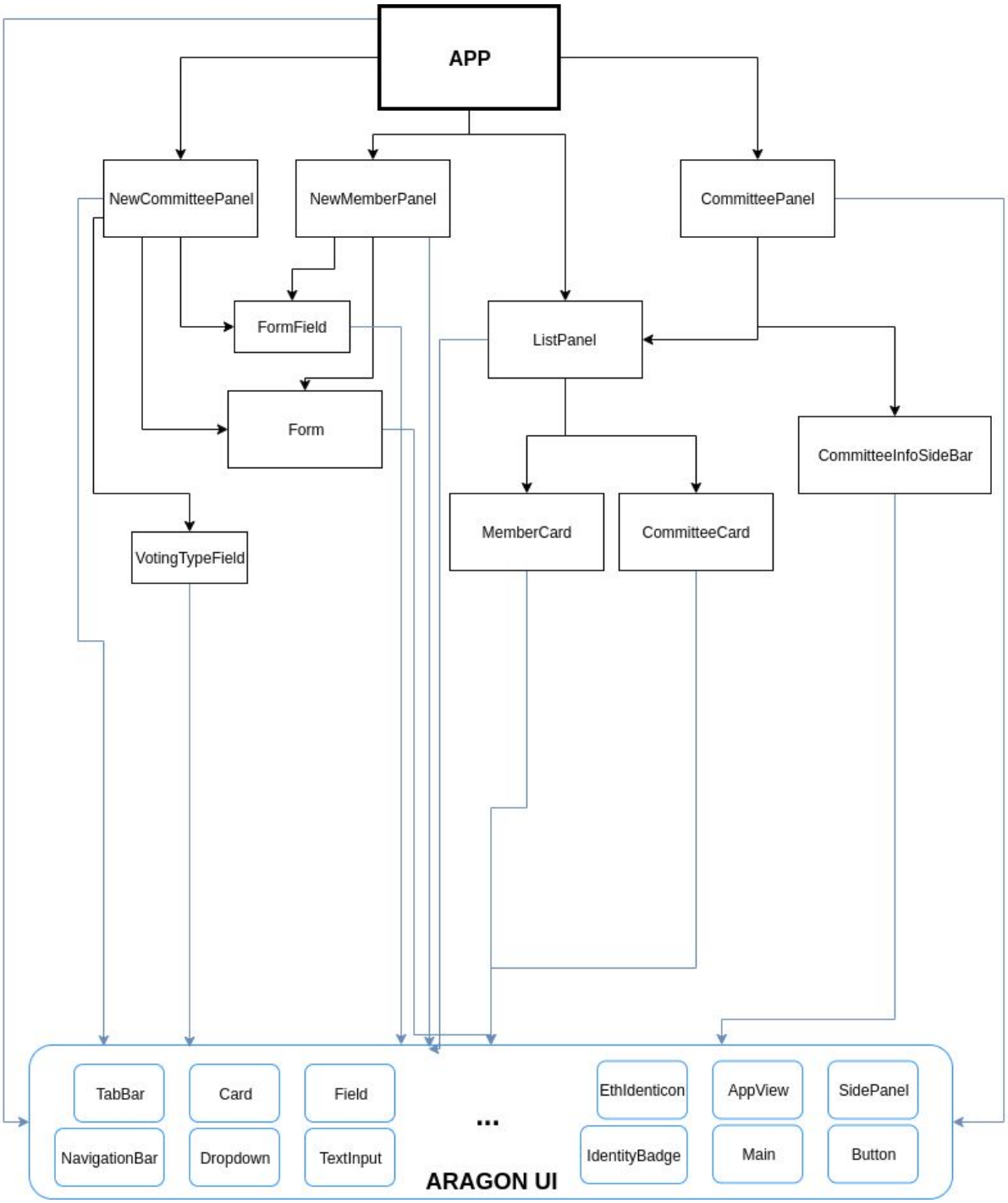

Figura 46 . Diagrama de componentes de la aplicación.

El componente central *App* hace uso de cuatro componentes principales:

1. *NewCommitteePanel*: Implementa el panel lateral que contiene el formulario para crear un nuevo comité.
2. *NewMemberPanel*: Implementa el panel lateral que contiene el formulario para añadir un nuevo miembro a un comité.

3. *ListPanel*: Implementa un panel que puede renderizar una lista de comités o miembros de un comité. Uno de los usos más importantes es la implementación de la vista general de la aplicación.
4. *CommitteePanel:* Implementa la vista de un comité seleccionado. Mostrando sus miembros e información general de interés.

Todos los componentes desarrollados utilizan la libreria AragonUI, que ofrece componentes para estructurar la aplicación como puede ser *Main, TabBar, AppView* o SidePanel que implementa el panel lateral característico de las aplicaciones de Aragon. También ofrece componentes para mostrar direcciones o cuentas de Ethereum de una forma más estética e interactiva, por ejemplo: *IdentityBadge* o *EthIdenticon*. Aparte de estos, AragonUI posee una serie de componentes que implementan distintos campos que frecuentemente se suelen utilizar en los formularios, entre ellos se utilizaron: *Dropdown, TextInput* y *Field.*

Ejemplos de uso

A continuación, se describe cada una de las funcionalidades que ofrece la aplicación. Ejemplificando su uso paso a paso.

Creación de un comité

Si se quiere crear un nuevo comité, hay hacer clic sobre el botón “New Committee”, el cual abrirá un panel lateral que contiene el siguiente formulario:

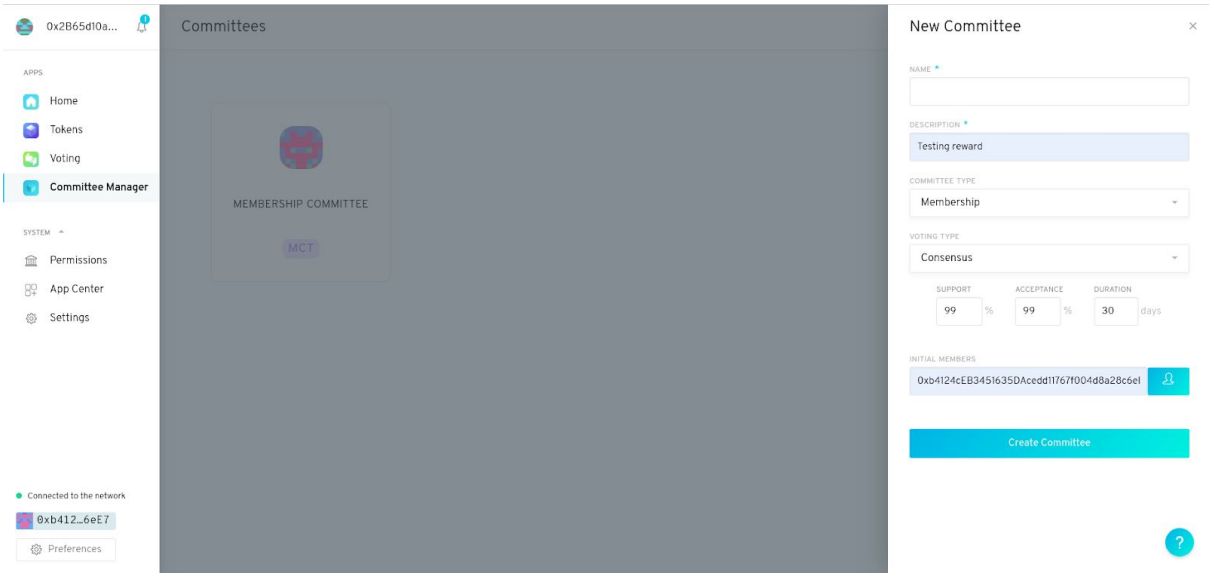

Figura 47. Vista del panel lateral para crear un comité.

Rellenamos los campos introduciendo: el nombre, descripción y el tipo de comité; el tipo de votación que queremos; y las direcciones de los miembros iniciales del comité. Cuando pulsemos el botón “Create Committee” se abrirá un nuevo panel lateral indicando que la acción tiene que someterse a votación para para poder realizarse:

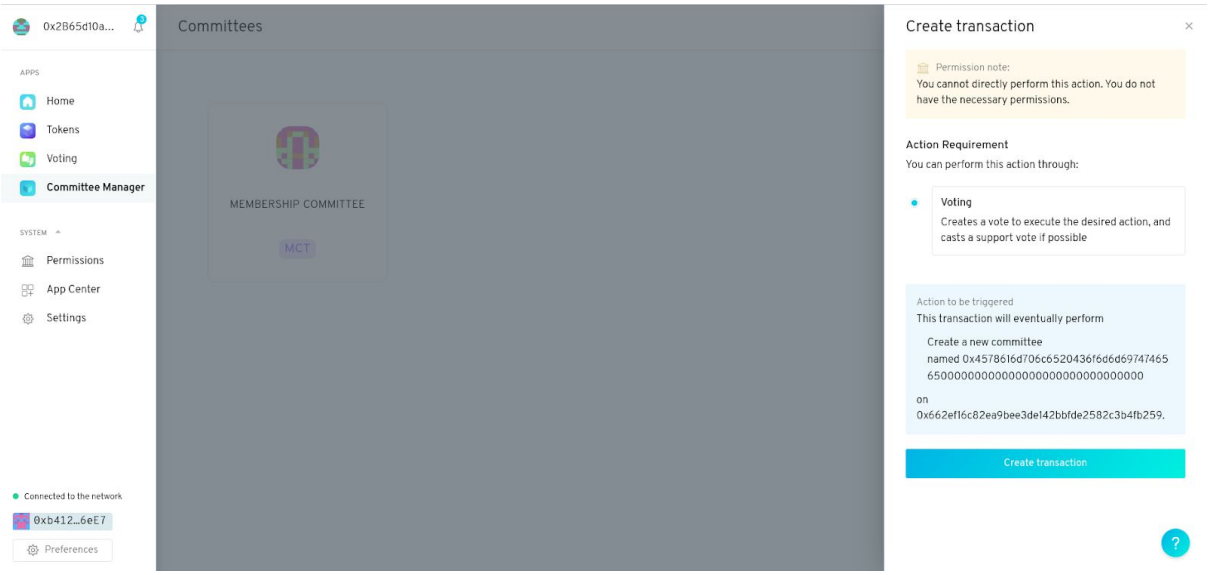

Figura 48. Panel lateral informativo.

Al presionar el botón “Create Transaction”, el cliente de Aragon creará una nueva transacción para la operación y notificará a Metamask para que abra una nueva ventana de pidiendo firmar la transacción:

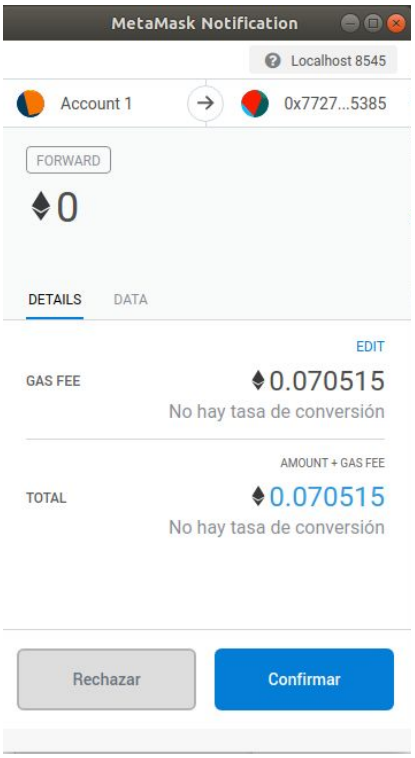

Figura 49. Ventana de Metamask

Una vez pulsado el botón “Confirmar”, se creará una nueva votación en la aplicación de *Voting* general donde todos los miembros de la DAO pueden votar:

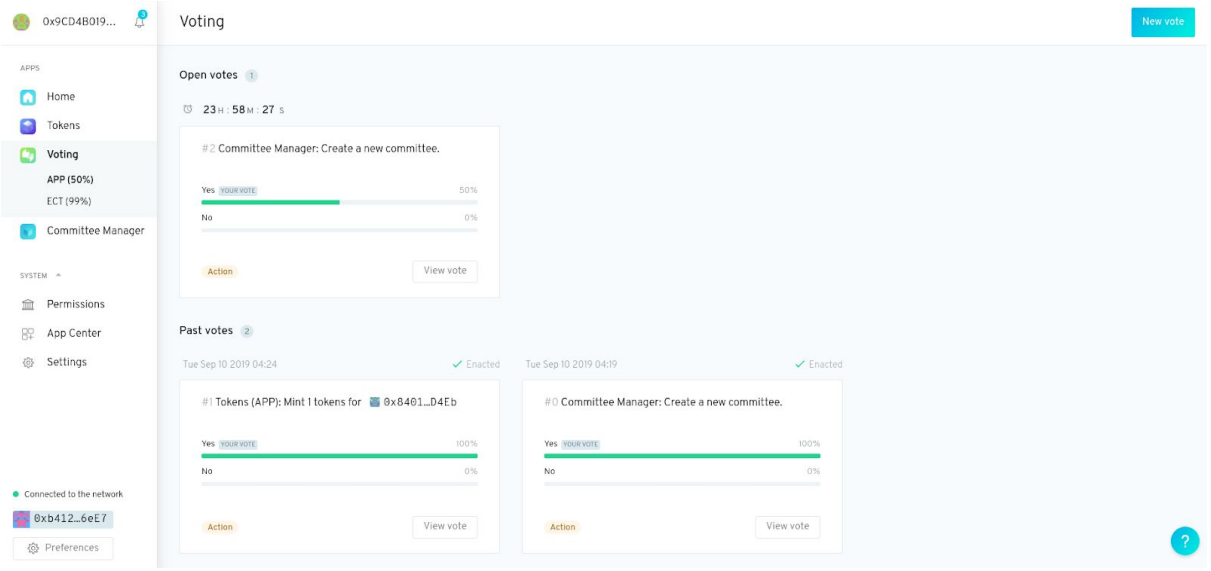

Figura 50. Vista de la aplicación *Voting*

Si la votación se aprueba, entonces se creará el comité y se añadirá a la vista general de comités de la aplicación:

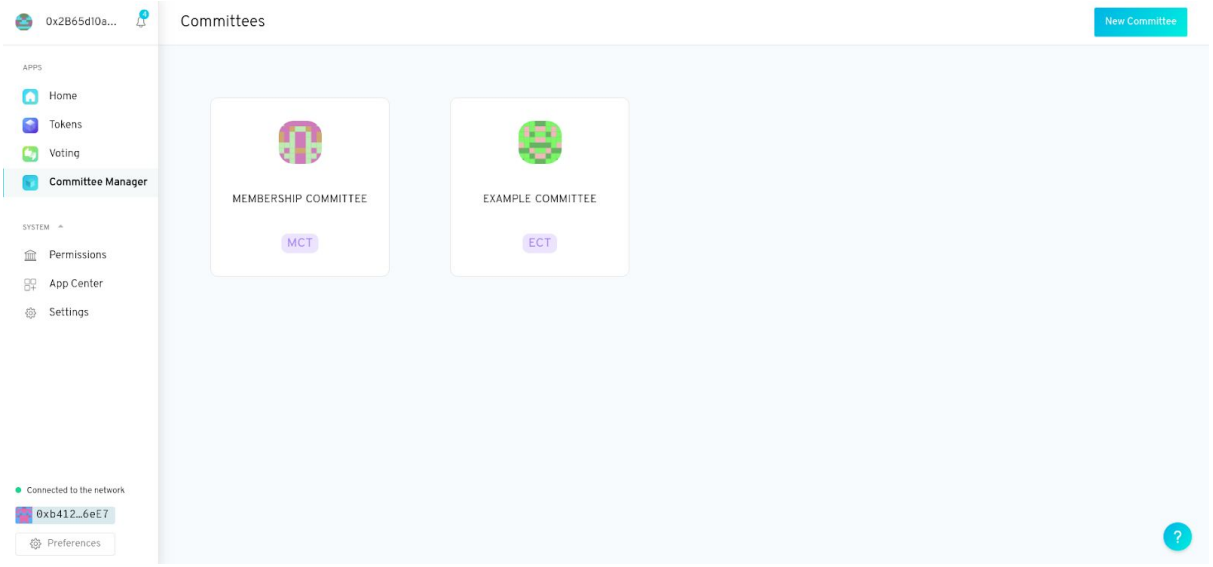

Figura 51.Vista mostrando el nuevo comité creado

Ver detalles de un comité

Si queremos ver un comité en específico, podemos seleccionarlo en la vista general. Se abrirá una vista con información detallada relativa al comité.

La vista está compuesta de dos pestañas:

1. Info: Esta pestaña muestra información general del comité. En la sección principal de la página se muestra la descripción del comité junto con todos los miembros actuales. En el panel derecho se muestra información sobre el token que posee el comité, el tipo de votación y la dirección, tipo e imagen del comité. En la esquina inferior derecha se puede observar el botón “Remove Committee” y en la esquina superior derecha el botón “Add Member”.

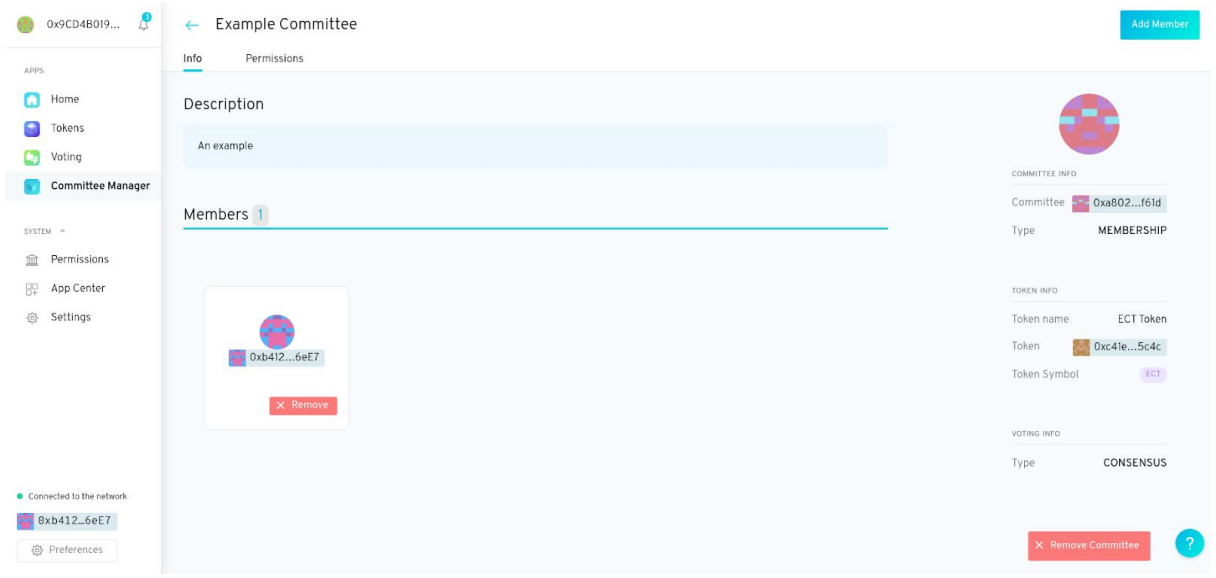

Figura 52. Vista pestaña Info

2. Permissions: Esta pestaña todavía está por desarrollar. Se espera que posea dos tablas: una con los permisos individuales del comité y otra con los permisos grupales; así como botones para poder añadir/eliminar permisos nuevos.

Eliminación de un comité

Tenemos que hacer clic sobre el comité que queremos eliminar desde la vista general de la aplicación. Una vez en la vista del comité, presionamos el botón rojo “Remove Committee” y debemos seguir el mismo proceso descrito en casos anteriores. Se abrirá un panel lateral para crear la transacción y nos informa que la acción debe pasar por votación antes de ser ejecutada, creándose una nueva votación en la aplicación de *Voting* general de la DAO.

Figura 53 .Panel lateral de eliminar comité.

Una vez se haya aprobado la votación, todos los miembros son eliminados del comité, al destruir cada uno de sus tokens. Luego, se elimina del comité todos los permisos que éste posea. Por último, el comité desaparece de la vista general de la aplicación.

Como se comentó anteriormente, en Aragon la funcionalidad de eliminar una aplicación todavía no está implementada. Lo único que se puede hacer es revocar todos los permisos a la aplicación y no permitir que nadie pueda conceder ninguno en el futuro. No obstante, la aplicación seguirá apareciendo en el panel izquierdo dentro del listado de Apps.

Añadir nuevo miembro a un comité

Para añadir un miembro al comité, debemos navegar hasta la vista del comité y hacer clic en el botón “Add Member”. A continuación, se abrirá el siguiente panel lateral con un formulario:

Figura 54. Panel lateral para añadir miembro.

El formulario posee un único campo donde debemos introducir la dirección del nuevo miembro. Una vez rellenado, se pulsa el botón “Submit Member” que abrirá el panel lateral para crear la transacción y la votación:

Figura 55. Panel lateral para crear transacción de nuevo miembro.

El resto del proceso es análogo al descrito en ejemplos de uso previos. Una vez completado, aparecerá un nuevo elemento en la sección de miembros.

Eliminar miembro de un comité

Para eliminar un miembro se debe navegar a la vista de comité y, dentro de la sección de miembros, pulsar el botón “Remove” del miembro que se quiera expulsar. El resto del proceso es igual al descrito en ejemplos de uso anteriores.

Eliminar un miembro supone destruir el token que este posee. Cuando se complete la operación, la carta del integrante dentro de la sección de miembros desaparecerá.

Descarga y ejecución del código

Necesitamos utilizar alguna distribución de Linux para poder ejecutar este proyecto.

Primero que nada, necesitaremos tener Node.js instalado, el cual podemos descargar desde el siguiente enlace https://nodejs.org/es/download/. Es necesario tener, como mínimo, la versión 10.5.3.

Una vez hayamos instalado Node.js podemos descargar e instalar globalmente el CLI de Aragon utilizando el gestor de paquetes NPM, que se instala implícitamente al instalar Node. Para descargar el CLI debemos abrir una consola de comandos e introducir la siguiente línea:

```
npm i -g @aragon/cli
```

Figura 56. Comando para instalar el CLI de Aragon.

Es necesario descargar la extensión para navegador Metamask, la cual podemos descargar desde el siguiente enlace: https://metamask.io/. Una vez hecho esto, debemos crear y configurar una cuenta siguiendo paso a paso las instrucciones que nos indica Metamask.

Un detalle a tener en cuenta es cambiar el tipo de red a la que Metamask apunta. Cuando ejecutemos el proyecto debemos seleccionar la red Localhost:8545, para ello hacemos clic sobre la lista desplegable ubicada en la parte superior de la ventana de la extensión:

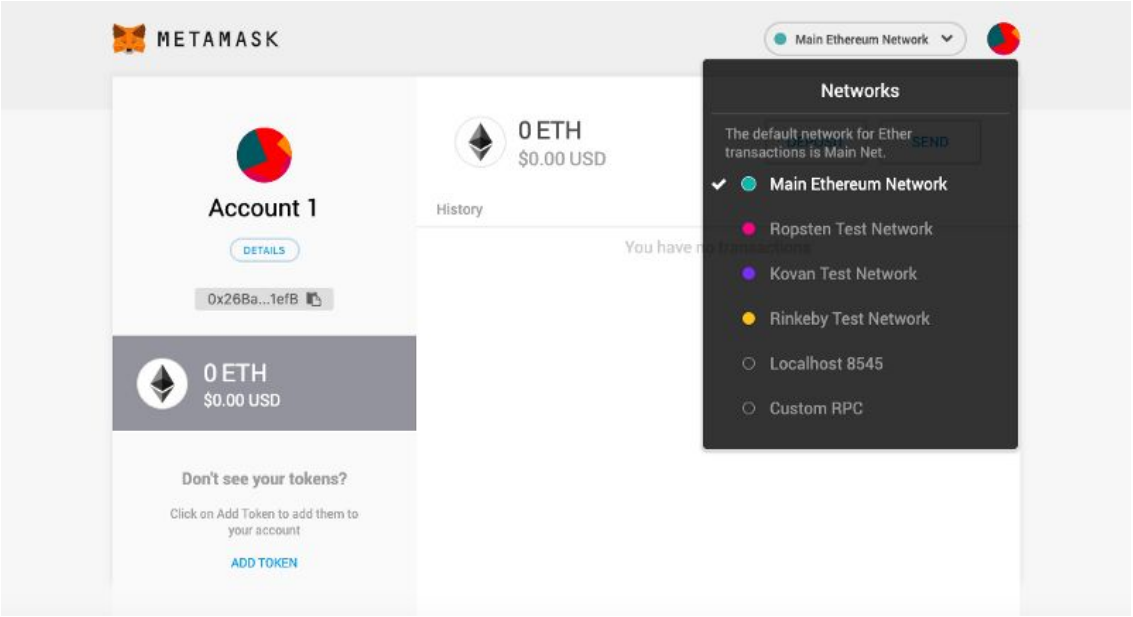

Figura 57. Selección de la red en Metamask.

El código del proyecto se encuentra en el siguiente repositorio de Github:

https://github.com/PJColombo/committee-manager-app

Debemos pulsar el botón “Clone or download” y hacer clic en “Download ZIP”

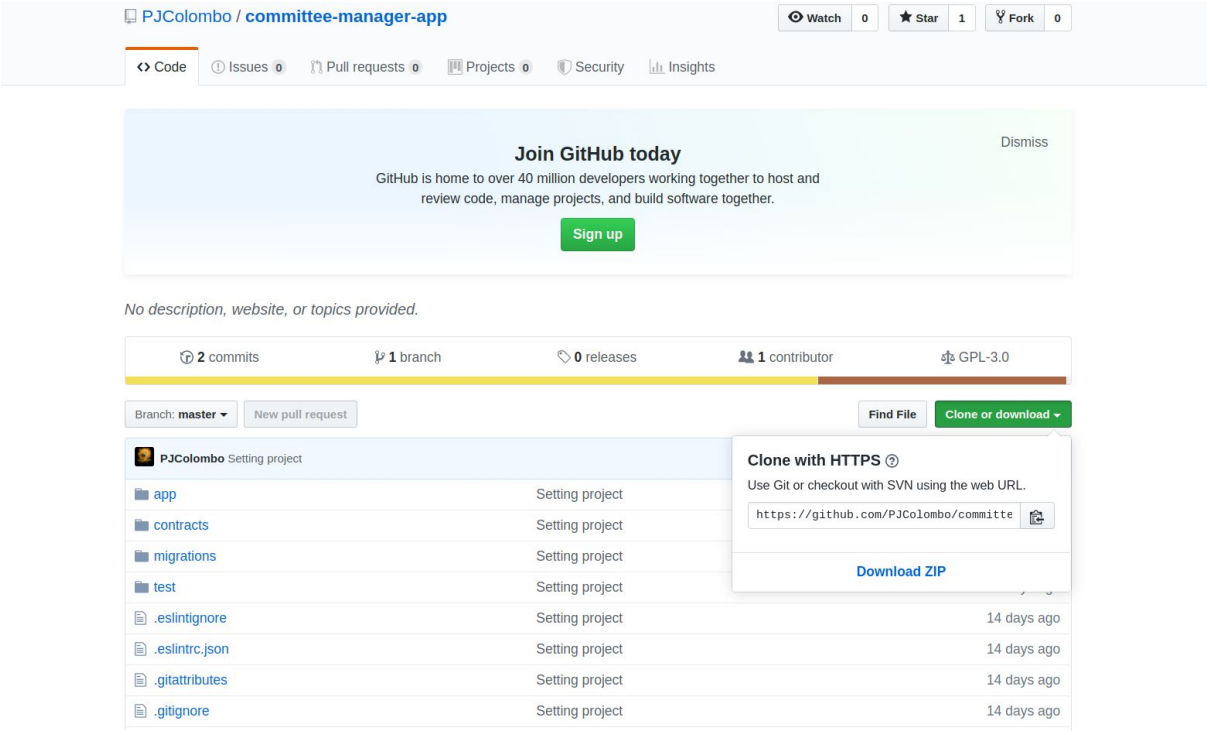

Figura 58. Repositorio del proyecto en Github.

Una vez descargado el ZIP, lo descomprimimos y abrimos una consola de comandos dentro del directorio del proyecto y ejecutamos el siguiente comando:

```
npm install
```

Figura 59. Comando para instalar todas las depedencias.

Este comando se encarga de descargar e instalar todas las dependencias que necesita el proyecto para funcionar del gestor de paquetes. Una vez hecho esto, ejecutamos la siguiente línea:

```
npm run start:ipfs:template
```

Figura 60. Comando para ejecutar el proyecto.

Esto preparará el entorno necesario para ejecutar nuestra aplicación

```
paulo@HAL9000:~/Documents/Uni/TFG/Julyproject/september/committee-manager-app$ npm run start:ipfs:template

> committee-manager-app@1.0.0 start:ipfs:template /home/paulo/Documents/Uni/TFG/Julyproject/september/committee-manager-app
> npm run start:ipfs -- --template Template --template-init @ARAGON_ENS


> committee-manager-app@1.0.0 start:ipfs /home/paulo/Documents/Uni/TFG/Julyproject/september/committee-manager-app
> aragon run "--template" "Template" "--template-init" "@ARAGON_ENS"

  ✔ Start a local Ethereum network
  ✔ Check IPFS
  ✔ Publish app to APM
  ✔ Deploy Template
  ✔ Create DAO
  ✔ Open DAO
 ⚠ Server already listening at port 3000, skipped starting Aragon
 ℹ You are now ready to open your app in Aragon.
 ℹ Here are some Ethereum accounts you can use.
  The first one will be used for all the actions the CLI performs.
  You can use your favorite Ethereum provider or wallet to import their private keys.

Address #1:  0xb4124cEB3451635DAcedd11767f004d8a28c6eE7 (this account is used to deploy DAOs, it has more permissions)
Private key: a8a54b2d8197bc0b19bb8a084031be71835580a01e70a45a13babd16c9bc1563
Address #2:  0x8401Eb5ff34cc943f096A32EF3d5113FEbE8D4Eb
Private key: ce8e3bda3b44269c147747a373646393b1504bfcbb73fc9564f5d753d8116608
 ℹ The accounts were generated from the following mnemonic phrase:
explain tackle mirror kit van hammer degree position ginger unfair soup bonus


ℹ This is the configuration for your development deployment:
   Ethereum Node: ws://localhost:8545
   ENS registry: 0x5f6f7e8cc7346a11ca2def8f827b7a0b612c56a1
   APM registry: open.aragonpm.eth
   DAO address: 0x8346Eb0Bf5A1603d3AC7E59714Ba4eD5CcF6520A

   Opening http://localhost:3000/#/0x8346Eb0Bf5A1603d3AC7E59714Ba4eD5CcF6520A to view your DAO
```

Figura 61. Resultado de ejecutar el comando.

Una vez se haya desplegado nuestra DAO, se abrirá automáticamente una nueva pestaña en nuestro navegador apuntando a ella. En caso de que no ocurriese esto, podemos abrirlo

manualmente accediendo a la url que se muestra en la última línea de código de la imagen previa.

CONCLUSIONES

A lo largo de los últimos años, la tecnología blockchain ha experimentado un progreso formidable dentro del contexto del desarrollo software, dando lugar a iniciativas y proyectos que utilizan todo tipo de software y herramientas fundamentadas en esta tecnología. Las grandes empresas ya están empezando crear sus propios departamentos de blockchain y a llevar a cabo proyectos de desarrollo propios, tal es el caso de Facebook que plantea lanzar Libra, su propia criptomoneda.

Aunque es cierto que la tecnología sigue cambiando a un ritmo acelerado, no cabe la menor duda de que no es la misma tecnología que la de hace cinco años, ya no es una mera prueba de concepto. El desarrollo software basado en blockchain se ha formalizado, ha creado su propio repertorio de patrones de diseño; nuevas técnicas, metodologías y enfoques para abordar problemas de diseño e implementación. Ha creado también una filosofía propia.

Este hecho lo podemos ver en la propia iniciativa de Aragon. El framework es gigante, depende de muchas tecnologías y paquetes subyacentes y el desarrollo de sus múltiples módulos han dado lugar a una serie de herramientas expuestas para el desarrollador que le permiten crear cosas potencialmente increíbles.

La finalidad de este trabajo era explorar las posibilidades que ofrecía Aragon a nivel de desarrollo. Es una iniciativa relativamente nueva que, en mi opinión, desconocen muchos desarrolladores. Se buscaba dar una explicación a su funcionamiento y cómo estaba estructurado y organizado; también el propósito de cada herramienta que ofrece, que se puede hacer con ellas y para qué contexto son más adecuadas. Todo esto a través del desarrollo de un proyecto con el objetivo de ofrecer un enfoque práctico y no puramente teórico al framework.

Se espera que el desarrollo del gestor de comité pueda formar parte de las aplicaciones disponibles de Aragon. Y si no es así, al menos que sirva de modelo y guía para futuros proyectos que quieran implementar un buen mecanismo de distribución de trabajo y tareas dentro de las DAOs.

CONCLUSION

Over the past few years, blockchain technology has experienced formidable progress within the context of software development, leading to initiatives and projects that use all types of software and tools based on this technology. Large companies are already starting to create their own blockchain departments and to carry out their own development projects, such is the case of Facebook that is launching Libra, its own cryptocurrency.

Although it is true that technology continues to change at an accelerated rate, there is no doubt that it is not the same technology as that of five years ago, it is no longer a mere proof of concept. The software development based on blockchain has been formalized, has created its own repertoire of design patterns; new techniques, methodologies and approaches to address design and implementation problems. He has also created his own philosophy.

We can see this fact in Aragon's own initiative. The framework is giant, it depends on many underlying technologies and packages and the development of its multiple modules has resulted in a series of tools exposed to the developer that allow you to create potentially incredible things.

The purpose of this work was to explore the possibilities offered by Aragon at the development level. It is a relatively new initiative that, in my opinion, many developers are unaware of. It was sought to give an explanation of its operation and how it was structured and organized; also the purpose of each tool it offers, what can be done with them and for what context they are more appropriate. All this through the development of a project with the objective of offering a practical and not purely theoretical approach to the framework.

It is expected that the development of the committee manager can be part of the available applications of Aragon. And if not, unless it serves as a model and guide for future projects that want to implement a good mechanism for distribution of work and tasks within DAOs.

TRABAJO FUTURO

Durante el desarrollo del proyecto, el equipo de desarrollo de Aragon todavia no habia implementado funciones para obtener el listado de aplicaciones y permisos de la DAO, algo que era necesario para poder programar la lógica de gestión de permisos. No obstante, durante la última semana del desarrollo del proyecto una nueva versión del cliente de Aragon fue publicada, la cual contenía las funcionalidades necesarias ya implementadas. Esto abre las puertas a un desarrollo futuro que implemente la lógica necesaria en el contrato y cree las vistas de los permisos en el Front-End.

Otra de las mejoras que se había considerado era la de personalizar en mayor medida los comités creados, por ejemplo:

- Permitir al usuario poder subir una imagen a la hora de crear uno nuevo. La imagen se guardaría en IPFS para evitar aumentar los costes de almacenamiento.
- Aparte de poder introducir un listado de miembros iniciales, se había considerado permitir introducir al usuario una cantidad de tokens iniciales a su elección para cada uno de ellos. Esta funcionalidad estaría restringida a los tokens Reputation y Equity.

En cuanto a funcionalidades, se había considerado añadir en algún futuro las siguientes:

- Implementar un chat para los miembros del comité utilizando algo parecido a Orbit Chat[24], donde se podrían discutir asuntos internos o relativos a las funciones del comité.
- Añadir una nueva vista donde se aparezcan gráficos históricos que muestren un historial de todas las acciones realizadas por los miembros utilizando los permisos individuales que provee el token del comité. Se podría intentar implementar leyendo los eventos históricos de las otras aplicaciones desplegadas en la DAO. Ademas, se habia pensando en añadir un grafico de lineas que mostrase un historial de todas las votaciones que han tenido lugar en el comité y el resultado de esta. Así, se podría tener una mejor percepción de las actividades desempeñadas por cada comité y de su nivel de rendimiento.

[24] https://orbit.chat/#/connect

BIBLIOGRAFÍA

[1] Shermin Voshmgir. *Token Economy: How blockchains and smart contracts revolutionize the economy*, 2019. ISBN-13: 978-3982103822

[2] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, Steven Goldfeder. *Bitcoin and Cryptocurrency Technologies*, 2017, Princeton University Press. ISBN-13: 978-0691171692

[3] *Traditional Organizations VS. DAOs* (10 ago. 2019) [En línea]. Available: https://blockchainhub.net/dao-decentralized-autonomous-organization/

[4] DAOStack Team (24 ago. 2019). DAOStack: An Operating System for Collective Intelligence, 2018 [en línea]. Available: https://daostack.io/wp/DAOstack-White-Paper-en.pdf

[5] Gavin Wood (23 ago. 2019) *PolkaDot: Vision for a heterogeneous multi-chain framework,* [en línea]. Available:https://polkadot.network/PolkaDotPaper.pdf

[6] Polkadot Team (23 ago. 2019) *Announcing PolkaDAO: Fund Your Project!*, 2019 [en línea]. Available: https://medium.com/polkadot-network/announcing-polkadao-fund-your-project-1891e6d895a

[7] Matan Field (24 ago. 2019). *Holographic consensus - part 1*, 2018 [en línea]. Available: https://medium.com/daostack/holographic-consensus-part-1-116a73ba1e1c

[8] Daniel Kronovet (25 ago. 2019). *Aragon, DAOStack, Colony, Moloch*, 2019 [en línea]. Available: http://kronosapiens.github.io/blog/2019/06/16/aragon-daostack-colony-moloch.html

[9] Alex Rea, Aron Fischer, Jack du Rose (26 ago. 2019). *Colony, Technical White Paper*, 2019 [en línea]. Available: https://colony.io/whitepaper.pdf

[10] Aragon Team (1 jul. 2019). *Aragon Whitepaper* [en línea]. Available: https://github.com/aragon/whitepaper

[11] Aragon Team (26 ago. 2019). *Introducing Aragon Nest*, 2017 [en línea]. Available: https://blog.aragon.org/introducing-aragon-nest-1aa8c91c0566/

[12] Aragon Team (26 ago. 2019). *Aragon Nest* [en línea]. Available: https://github.com/aragon/nest

[13] Luke Duncan, (18 ago. 2019) *Aragon Network and Token Primer*, 2019 [en línea]. Available: https://blog.aragon.one/aragon-network-and-token-primer/

[14] Luis Cuende (1 jul. 2019). *AGP-0: The Aragon Manifesto,* 2018 [en línea]. Available:https://github.com/aragon/AGPs/blob/master/AGPs/AGP-0.md

[15] Maker Team (22 ago. 2019). *The Dai Stablecoin System*, 2017 [en línea] Available: https://makerdao.com/whitepaper/Dai-Whitepaper-Dec17-en.pdf

[16] Kerman Kholi (22 ago. 2019). *What's MakerDAO and what's going on with it? Explained with pictures*, 2019 [en línea]. Available: https://hackernoon.com/whats-makerdao-and-what-s-going-on-with-it-explained-with-pictures-f7ebf774e9c2

[17] Paul Seidler, Paul Kolling y Max Hampshire. *terra0, Can an augmented forest own and utilise itself?*, 2016 [en línea]. Available: https://www.terra0.org/assets/pdf/terra0_white_paper_2016.pdf

[18] Maximilian Schwarzmüller. *Available: React - The Complete Guide (incl Hooks, React Router, Redux)* [en línea]. Available: https://www.udemy.com/react-the-complete-guide-incl-redux/

[19] Stephen Grider. *Ethereum and Solidity: The Complete Developer's Guide* [en línea]. Available: https://www.udemy.com/course/ethereum-and-solidity-the-complete-developers-guide/

[20] *Community Initiative: Aragon Cooperative* (4 jun. 2019) [en línea]. Available: https://forum.aragon.org/t/community-initiative-aragon-cooperative/356

[21] *Aragon Internationalization Effort* (27 may. 2019) [en línea]. Available: https://devpost.com/software/aragon-internationalization-effort-no0fr7

[22] Usman W. Chohan. *The Double Spending Problem and Cryptocurrencies,* 2017 (5 ago. 2019) [en línea]. Available:https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3090174

[23] Vitalik Buterin (2 jul. 2019). *An Introduction to Futarchy*, 2014 [en línea]. Available: https://blog.ethereum.org/2014/08/21/introduction-futarchy/

[24] Andrew Leonard (5 jul. 2019).*Can Aragon Build an Unstoppable Robotic Government?* [en línea]. Available: https://breakermag.com/can-aragon-make-decentralized-autonomous-governance-work/

[25] Menajem Benchimol (20 ago. 2019). *Decentralized Autonomous Organizations (DAOs) Will Enable Faster Decision Making Tools For Leaders*, 2019 [en línea]. Available: https://hackernoon.com/decentralized-autonomous-organizations-daos-will-enable-faster-decision-making-tools-for-leaders-49cfea72d086

[26] Antonio Madeira (19 ago. 2019). *The Dao, the Hack, the Soft Fork and the Hard Fork*, 2019 [en línea]. Available: https://www.cryptocompare.com/coins/guides/the-dao-the-hack-the-soft-fork-and-the-hard-fork/

[27] Samuel Falkon (19 ago. 2019). *The Story of the DAO — Its History and Consequences*, 2017 [en línea]. Available: https://medium.com/swlh/the-story-of-the-dao-its-history-and-consequences-71e6a8a551ee

[28] Eric Gorski (19 ago. 2019). *Introducing the dxDAO*, 2018 [en línea]. Available: https://blog.gnosis.pm/introducing-the-dxdao-27ec4301eced

[29] ConsenSys (19 ago. 2019). *The dxDAO Awakens: How to Participate and Become a dxDAO Stakeholder*, 2019 [en línea]. Available: https://media.consensys.net/the-dxdao-awakens-how-to-participate-and-become-a-dxdao-stakeholder-cc3ed713bf53

[30] Jorge Izquierdo, Ramon Recuero (6 ago. 2019). *The future of organizations*, 2018 [en línea]. Available: https://blog.aragon.one/the-future-of-organizations/

[31] Quiao Wang (6 ago. 2019) *Cryptonetworks and the theory of the firm,* 2018 [en línea]. Available: https://www.tokendaily.co/blog/cryptonetworks-and-the-theory-of-the-firm

[32] Jorge Izquierdo, Aragon One (17 ago. 2019). *Deploying and distributing the Aragon client*, 2018 [en línea]. Available: https://blog.aragon.org/deploying-and-distributing-aragon-core-11e70cbc9b50/

[33] Gorka Ludlow (16 ago. 2019). *Aragon Frontend Walkthrough*, 2019 [en línea]. Available: https://blog.aragon.one/aragon-frontend-walkthrough/

[34] Steven Mckie (6 ago. 2019). *The Year of the DAO Comeback*, 2019 [en línea]. Available: https://medium.com/amentum/the-year-of-the-dao-comeback-9c888b44980

[35] Luis Cuende, Aragon Team, Aragon One Team (15 ago. 2019). *Building a decentralized OS*, 2018 [en línea]. Available: https://blog.aragon.org/building-a-decentralized-os/

[36] Jorge Izquierdo, Aragon Team (15 ago. 2019).)Available:*Introducing aragonOS: Say hi to modular and extendable organizations*
[en línea]. Available: https://blog.aragon.org/introducing-aragonos-say-hi-to-modular-and-extendable-organizations-8555af1076f3/

[37] Jorge Izquierdo, Aragon Team (15 ago. 2019). *Introducing aragonOS 3.0 alpha, the new operating system for protocols and DApps* [en línea]. Available:

https://blog.aragon.org/introducing-aragonos-3-0-alpha-the-new-operating-system-for-protocols-and-dapps-348f7ac92cff/

[38] Brett Sun, Aragon Team, Aragon One Team (15 ago. 2019).)*Releasing aragonOS 4 – A refinement to our smart contract framework* [en línea]. Available: https://blog.aragon.org/releasing-aragonos-4/

[39] Luis Cuende (15 ago. 2019). *Using APM to replace NPM and other centralized package managers*, 2018 [en línea]. Available: https://blog.aragon.one/using-apm-to-replace-npm-and-other-centralized-package-managers/

[40] *Remix Docs* [en línea]. Available: https://remix.readthedocs.io/en/stable/index.html

[41]Glyn Holton (6 ago. 2019). *History of Corporations*, 2013 [en línea]. Available:https://www.glynholton.com/notes/corporation/#targetText=The%20word%20corporation%20derives%20from,certain%20community%20organizations%20called%20collegia.

[42] Luis Cuende, John Light (15 ago. 2019). *AGP-1: The Aragon Governance Proposal Process*, versión 1.2 [en línea]. Available: https://github.com/aragon/AGPs/blob/master/AGPs/AGP-1.md

[43] Nova Blitz (16 ago. 2019). *Storing Structs is costing you gas*, 2018 [en línea]. Available: https://medium.com/@novablitz/storing-structs-is-costing-you-gas-774da988895e

[44] *Motivación de Flux* (20 ago. 2019) [en línea]. Available: https://es.redux.js.org/docs/introduccion/motivacion.html

[45] *What is Flux?* (20 ago. 2019) [en línea]. Available: http://fluxxor.com/what-is-flux.html