Universidad
Rey Juan Carlos

Escuela Técnica Superior de Ingeniería Informática

Organización descentralizada autoorganizada para gestión de fondos sanitarios

Memoria del Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Iñaki Murillo del Prado

Tutor: Roberto Gallardo Cava

Octubre 2023

Agradecimientos

Me gustaría agradecer a mis padres por el apoyo que me han brindado desde que comencé en el Grado de Ingeniería Informática, sin el cuál no habría llegado hasta aquí.

También quiero expresar mi agradecimiento a mi tutor por su apoyo y confianza durante la realización de mi Trabajo de Fin de Grado.

Resumen

La descentralización y el sistema de salud son dos elementos cruciales para el desarrollo de un país y para la realización de los proyectos de vida de sus ciudadanos. A menudo, se asume erróneamente que en los países más desarrollados se dispone de un sistema de salud público de alta calidad y de acceso universal gratuito, lo cuál no siempre se cumple.

Un ejemplo de esta situación son los Estados Unidos, que a pesar de contar con el mayor gasto en salud en porcentaje del PIB (Producto Interior Bruto) entre países desarrollados, cuenta con un sistema sanitario con altos costos de atención médica y la falta de cobertura universal [26].

El objetivo central de este trabajo consiste en la creación de una organización que proporcione acceso real e igualitario para todos sus ciudadanos, además de brindar ayuda a aquellos individuos con dificultades para acceder a tratamientos debido a su posición económica.

Para alcanzar este propósito, se ha empleado una tecnología de registro distribuido o Blockchain. En concreto, se ha desarrollado un contrato inteligente en la red de Ethereum, en el cual los miembros de la red tienen la capacidad de gestionar los fondos de forma transparente y descentralizada. Todos los miembros poseen un NFT (Token No Fungible) que les habilita para participar en la toma de decisiones de la organización, además del acceso a tratamientos médicos a través del uso de los fondos. Adicionalmente, se ha diseñado una plataforma web para interactuar con el contrato inteligente y las distintas entidades, facilitando el uso y la accesibilidad a todos los interesados, teniendo como único requisito la posesión de una cartera de Ethereum.

Índice general

Índice de tablas

Índice de figuras

Nomenclatura

DAO	Organización Descentralizada Autoorganizada
DApp	Aplicación Descentralizadas
EVM	Ethereum Virtual Machine
HIPAA	Ley de Responsabilidad y Portabilidad del Seguro de Salud
HTML	Lenguaje de Marcas de Hipertexto
IDE	Entorno de Desarrollo Integrado
IoT	Internet of Things
NFT	Token No Fungible
npm	Node Package Manager
ODAFS	Organización Descentralizada Auto-organizada para Fondos Sanitarios
PIB	Producto Interior Bruto
PoW	Proof of Work

Capítulo 1

Introducción

En este primer capítulo, se explorará las motivaciones que han dado origen a este proyecto. También se profundizara en el estado del arte de la tecnología Blockchain, con un enfoque específico en la red de Ethereum, donde se llevarán a cabo la creación de los contratos inteligentes y Tokens No Fungibles (NFT) de esta organización. Por último, delinearemos los principales objetivos que queremos cumplir, se explicará la planificación del proyecto y se proporcionará una visión general de la estructura que seguirá esta memoria.

1.1. Motivación

Una de las principales inquietudes que las personas tienen en relación con sus proyectos de vida es la atención médica. En la sociedad moderna, esta suele ser proporcionada o gestionada por los estados y suele representar uno de los mayores beneficiarios de los presupuestos públicos. Sin embargo, en muchos casos, la gestión del presupuesto tiende a ser ineficiente y, en ocasiones, resultar en unos inadecuados servicios al paciente. Entre las alternativas más comunes se encuentran los seguros médicos privados y la mutualización de los riesgos laborales por parte de los empleados.

Un ejemplo de esta situación se observa en Estados Unidos, que, a pesar de contar con uno de los mayores presupuestos en el ámbito de la salud, enfrenta enormes costos que recaen directamente en los pacientes. Los seguros médicos son muy comunes entre personas con ingresos elevados, mientras que aquellos que enfrentan dificultades económicas a menudo quedan excluidos de tratamientos médicos adecuados. Esto, a su vez, otorga un poder significativo tanto a las empresas farmacéuticas como a las aseguradoras [25].

Como posible solución, se plantea la creación de una organización descentralizada autoorganizada que proporcione a los usuarios la posibilidad de gestionar los fondos de manera directa. Además, se ofrecerá acceso a personas en riesgos de pobreza a través de un fondo de donaciones destinado a tratamientos y medicamentos.

Para llevar a cabo este proyecto, se utilizarán contratos inteligentes en la red blockchain de Ethereum, ya que esta tecnología proporciona una transparencia total en la gestión de fondos, una seguridad sin igual en comparación con empresas u organizaciones centralizadas, además de ser incorruptible. También se desarrollará un prototipo web de fácil uso para facilitar su acceso y utilización por parte de los usuarios.

1.2. Estado del arte

En la actualidad existen multitud de proyectos sanitarios que usan como pilar principal la tecnología blockchain. Estos proyectos buscan mejorar la trazabilidad y cadena de suministro de los medicamentos, además de mejorar y optimizar los servicios proporcionados por los seguros médicos. En este sector se pueden destacar tres proyectos por encima de los demás, Chronicled, HealthVerity y Patientory, siendo empresas que buscan la innovación en el sector de la salud a través del uso de blockchain y contratos inteligentes

Chronicled[20] es una empresa la cual quiere mejorar la trazabilidad y optimizar las cadenas de suministro de diversos productos. Utiliza la tecnología blockchain para controlar la cadena de suministro y proporcionar a sus clientes la garantía de que los productos llegan en su estado original. Siendo de gran utilidad para productos de alto valor, como son joyas, dispositivos electrónicos o vehículos.

HealthVerity[16] es una plataforma que busca facilitar la comunicación entre distintas entidades del sector de la salud, como son hospitales, farmacias y aseguradoras. Su principal objetivo consiste en la gestión de datos, transacciones y pagos a través de los contratos inteligentes, reduciendo la burocracia y acelerando los tratamientos y servicios, además de garantizar la privacidad y seguridad de la información sobre sus pacientes.

Patientory[22] es una aplicación distribuida basada en blockchain que revoluciona la manera en la que los usuarios acceden y gestionan sus datos médicos de manera segura y eficiente. También cabe destacar la capacidad de Patientory de combinar la descentralización innata de la blockchain con la integración de distintos sistemas centralizados de datos, todo en una misma plataforma. Emplea los contratos inteligentes para el servicio al paciente, como pueden ser tratamientos o atención médica, manteniendo y cumpliendo todos los requisitos HIPAA (Ley de Responsabilidad y Portabilidad del Seguro de Salud)[3].

Como podemos observar el propósito de este trabajo dista bastante de los proyectos mencionados, ya que se centran más en la calidad de los servicios y trazabilidad de la información. Nuestra meta es crear una organización que proporcione un acceso asequible y descentralizado a la atención médica, manteniendo siempre las características antes mencionadas.

1.3. Objetivos

El objetivo primordial de este proyecto es la creación de un contrato inteligente que represente una DAO (Organización Descentralizada Autoorganizada) que permita a sus usuarios la capacidad de administrar recursos destinados a la atención médica. Para poder interactuar con el contrato, los usuarios deberán poseer un NFT (Token No Fungible) que les identifique como miembros. Con el fin de alcanzar este objetivo, se han establecido las siguientes metas:

- Realizar una investigación sobre la tecnología blockchain, su funcionamiento y la formación de redes blockchain de primera, segunda y tercera generación.
- Evaluar las distintas redes blockchain que admiten contratos inteligentes y decidir la más adecuada para nuestro propósito.

- Definir la estructura jerárquica de la organización y las consecuentes relaciones con entidades externas. Se llevarán a cabo votaciones para la toma de decisiones importantes.

- Diseñar e implementar un contrato inteligente que constituya la DAO, así como los NFTs que representaran a sus miembros.

- Diseñar e implementar la interfaz necesaria para que las organizaciones externas puedan interactuar eficazmente con la DAO.

- Desarrollar un prototipo web para interactuar con el contrato inteligente. Deberá ser visual y fácil de usar.

1.4. Planificación de tareas

Título	Descripción	Duración
Investigación de la tecnología blockchain	Realizar un estudio sobre la tecnología blockchain, sus ventajas y distintas etapas.	15h
Investigación sobre contratos inteligentes	Realizar un estudio sobre los contratos inteligentes y como podrían ayudar al proyecto. Analizar el funcionamiento de los contratos y ventajas o características diferentes en comparación a otras tecnologías.	20h
Investigación de Ethereum	Realizar una investigación sobre la red blockchain de Ethereum donde se crearán y desplegarán los contratos inteligentes del proyecto.	10h
Análisis del proyecto	Realizar un análisis sobre el proyecto, decidir estructura y funcionamiento de la organización, además de los requisitos del proyecto.	25h
Elaborar casos de uso	Decidir que casos de uso tiene que cumplir la organización y todas las entidades involucradas en el proyecto, debe encontrarse un escenario totalmente funcional.	30h
Herramientas	Estudio y selección de las herramientas necesarias para el desarrollo del proyecto, también se tiene en cuenta la instalación y preparación del equipo.	8h
Implementación backend	Creación y desarrollo de los contratos inteligentes de los NFT, la organización y otras entidades.	40h
Implementación frontend	Desarrollo de la aplicación web en la que los usuarios interactuaran con los contratos inteligentes.	40h
Estructurar memoria	Realizar una estructuración de la memoria. Elaborar formato, capítulos y secciones de la memoria.	5h
Desarrollo memoria	Creación y redacción de la memoria.	30h

Tabla 1: Tareas de la planificación

En la Tabla 1 se identifican todas las tareas a realizar en este proyecto. Cada tarea estará desglosada en título, descripción y duración estimada.

1.4.1. Mapa de tareas

En la Figura 1 se muestra una representación gráfica del plan de trabajo del proyecto a lo largo del tiempo.

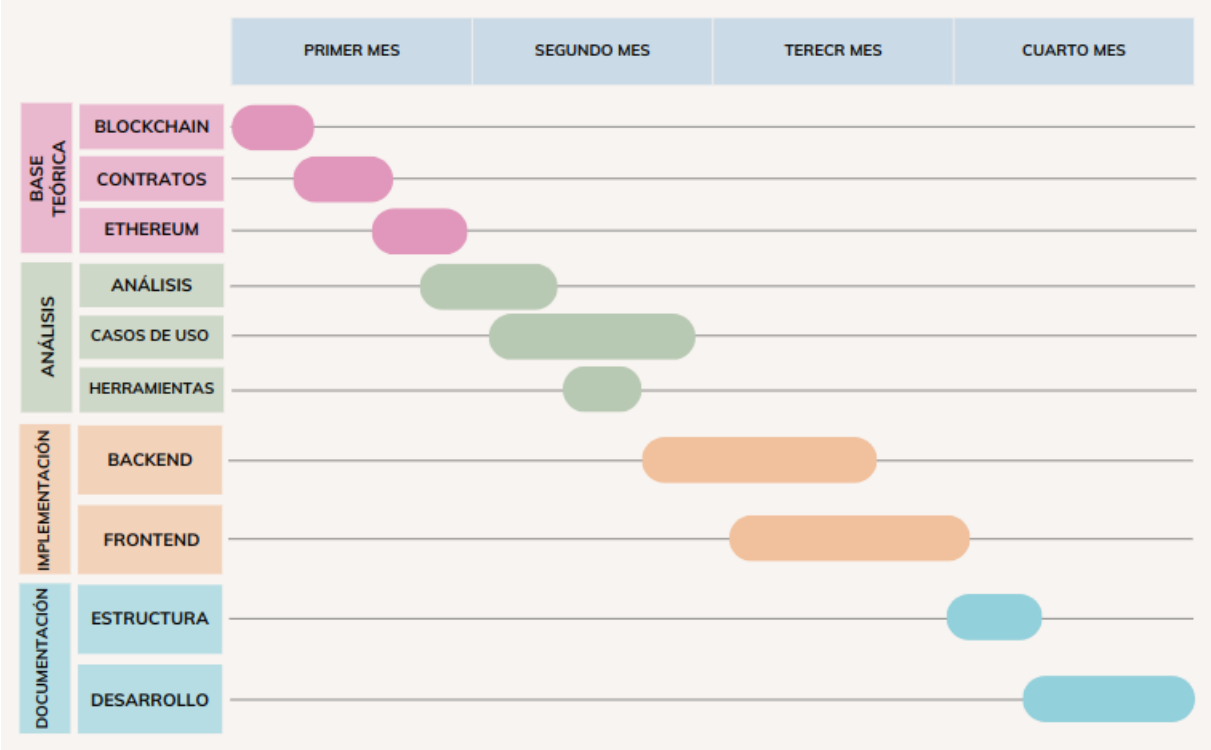


Figura 1: Diagrama de Gantt

1.5. Estructura de la memoria

Una vez acabado este capítulo introductorio, la memoria se estructura en seis capítulos. En el Capítulo 2, se explica la base teórica de Blockchain y contratos inteligentes. En el Capítulo 3, se realiza un análisis y se presentan los distintos casos de uso y las herramientas empleadas. A continuación, en los Capítulos 4 y 5 se explica la implementación del backend y frontend del proyecto. Seguidamente, en el Capítulo 6 se muestra la estructura y funcionamiento de las pruebas realizadas a los contratos inteligentes. Finalmente, se describen las conclusiones y los posibles futuros trabajos en el Capítulo 7.

Capítulo 2

Blockchain y Contratos Inteligentes

En este capítulo se abordan los conceptos básicos de la tecnología Blockchain y Contratos Inteligentes, incluyendo su evolución y las distintas redes que se han formando a lo largo de su historia. Se elegirá la red donde se desarrollará el proyecto y finalmente se expondrá el funcionamiento de los contratos inteligentes y las ventajas que proporcionan respecto a otras tecnologías.

2.1. Blockchain

En esta sección se definirá el concepto de Blockchain y se mostrará su estructura y funcionamiento usando un ejemplo gráfico. A continuación, se analizará la evolución de las distintas generaciones de Blockchain. Finalmente, se explicará la importancia de la descentralización en una organización o entidad.

2.1.1. ¿Qué es la Blockchain?

La blockchain, también conocida como cadena de bloques, es una tecnología que actúa como un libro de contabilidad digital e inmodificable, sin la necesidad de una entidad o individuo que actué como intermediario. La cadena de bloques consiste en una estructura de datos donde la información es almacenada en bloques conectados mediante metainformación, relativa al bloque anterior de la cadena. Esto implica que para modificar un determinado bloque en concreto sea necesario modificar toda la cadena posterior al mismo.

Entre los usos más populares de la blockchain se encuentran:

- Pagos digitales y Criptomonedas: la cadena de bloques actúa como verificador de transacciones, creando transacciones veloces y seguras.
- Bases de datos: forman un confiable e inalterable almacén de datos, además de permitir un mayor control a los usuarios sobre sus datos.
- Contratos inteligentes: permite la creación de contratos entre individuos de manera descentralizada donde las condiciones son controladas por la blockchain.

2.1.2. Tipos de Blockchain

Según quién tiene acceso y control sobre una red blockchain, se pueden identificar dos clases de redes, permisionadas y no permisionadas.

- No permisionadas, comúnmente llamadas públicas, son aquellas redes descentralizadas y abiertas al público. Cualquier usuario puede unirse a la red y verificar las transacciones. Todos los datos contenidos por la red son totalmente públicos. Ejemplos de estas redes son Bitcoin[24] y Ethereum[5].
- Permisionadas, también conocidas como privadas, son redes que cuentan con los mismos elementos que las redes permisionadas en las cuales una entidad central verifica y confirma todas las transacciones. Entre las principales características encontramos que el acceso es privado y debe ser aprobado por la entidad central, al igual que el acceso al libro de transacciones. Un ejemplo de estas redes es Corda[4].

En este proyecto se ha optado por emplear redes no permisionadas, concretamente Ethereum.

2.1.3. Estructura y Funcionamiento de una Blockchain

Para poder comprender mejor el funcionamiento de una cadena de bloques se propone el siguiente ejemplo basado en el modelo Blockchain propuesto por Satoshi Nakamoto en el White Paper de Bitcoin[24].

El principal elemento criptográfico utilizado es el hash, el algoritmo empleado para producirlo puede variar según la red, sus requisitos o condiciones, en este ejemplo usaremos SHA256 empleado por Bitcoin. El algoritmo SHA256 produce una salida única de 256 bits de longitud a partir de los datos o conjuntos de datos que recibe como entrada. Un ejemplo de la generación del algoritmo SHA-256 se puede encontrar en la Figura 2.

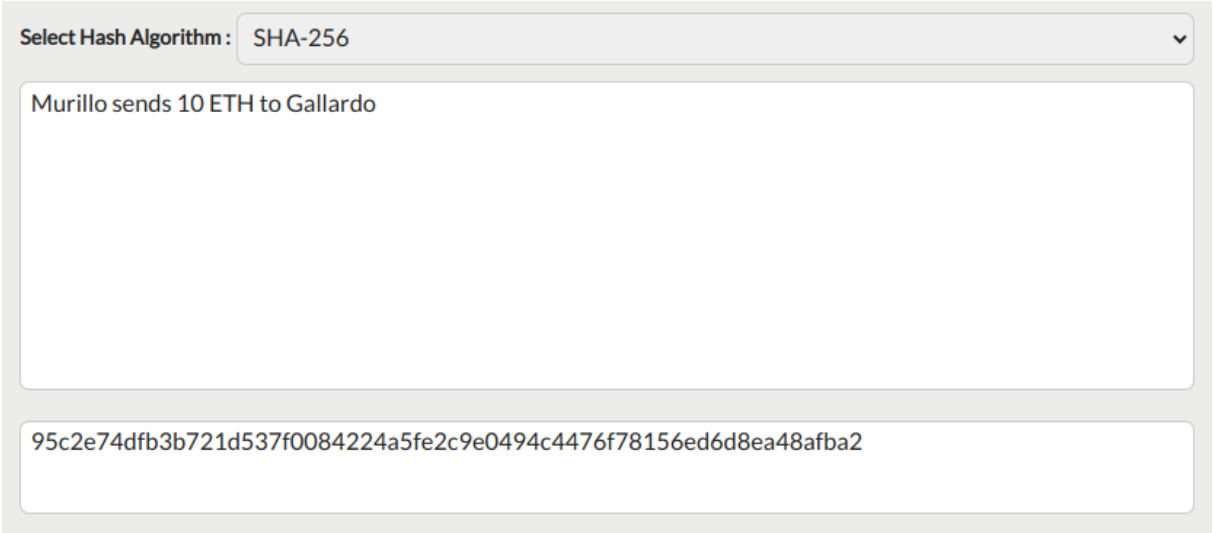


Figura 2: Función Hash

Como se ha mencionado con anterioridad, la blockchain esta formado por distintos bloques, cada uno formado por 5 elementos:

- Número de bloque: posición del bloque en la cadena. El primer bloque de la cadena se llama bloque génesis.
- Nonce: número arbitrario usado para alterar el hash producido por el conjunto de elementos del bloque.
- Información: puede contener cualquier tipo de información de valor, por ejemplo todas las transacciones realizadas durante un periodo de tiempo.
- Hash del bloque: creado por el algoritmo hash usando como valores el resto de elementos.
- Hash anterior: hash del bloque anterior, este es considerado la unión o enlace entre los bloques.

El objetivo principal de esta estructura es poder verificar de una manera rápida y sencilla si algún bloque ha sido modificado. En la Figura 3 se puede observar una cadena que se considera legítima, todos los hash coinciden con el del bloque anterior. Sin embargo, cuando se realiza un cambio en el bloque 2, por ejemplo se lleva a cabo una transacción fraudulenta, se ven modificados todos los valores hash de los posteriores bloques, demostrando su manipulación. En la Figura 4 se pueden observar los cambios.

En este modelo, se añade a la cadena, cada 10 minutos, un bloque con las transacciones realizadas en ese periodo. Satoshi Nakamoto lo que propone en su modelo es emplear el protocolo de consenso Proof of Work (PoW). Su objetivo principal es decidir que participante de la red propone el siguiente bloque, el cual incluye todas las transacciones realizadas en ese periodo de tiempo. El vencedor de este proceso será el minero que halle el nonce que altere el resultado del algoritmo hash haciendo que cumpla ciertas condiciones. Los mineros son participantes de la red cuyo propósito es hallar dicho nonce para recibir una recompensa por añadir un bloque a la red. La premisa principal es que el hash resultante del bloque empiece por un número concreto de ceros, determinado por la red. El número de ceros se considera la dificultad del minado. La finalidad de este proceso es verificar la integridad de manera eficiente y evitar bloques fraudulentos. Cualquier nodo de la red puede comprobar la cadena realizando una serie de algoritmos SHA-256.

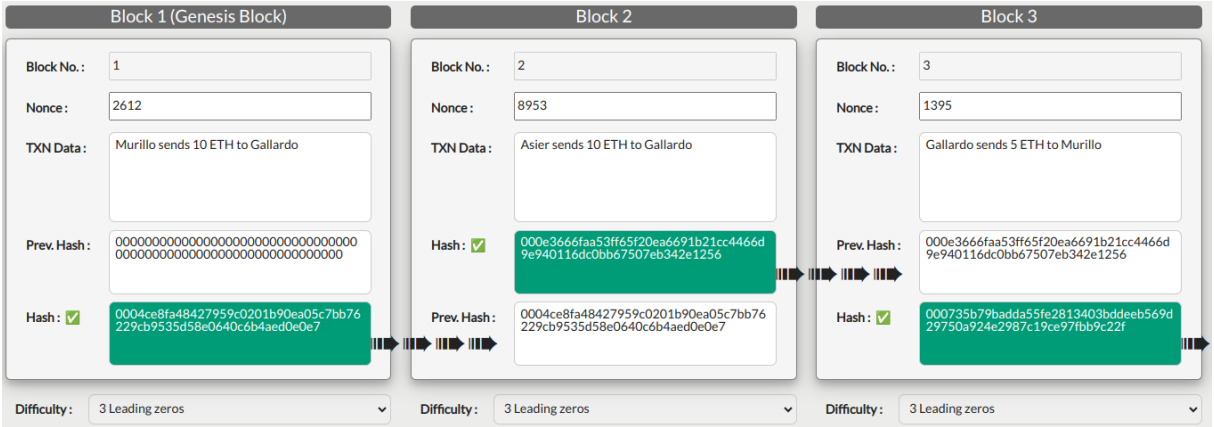


Figura 3: Ejemplo cadena de bloques

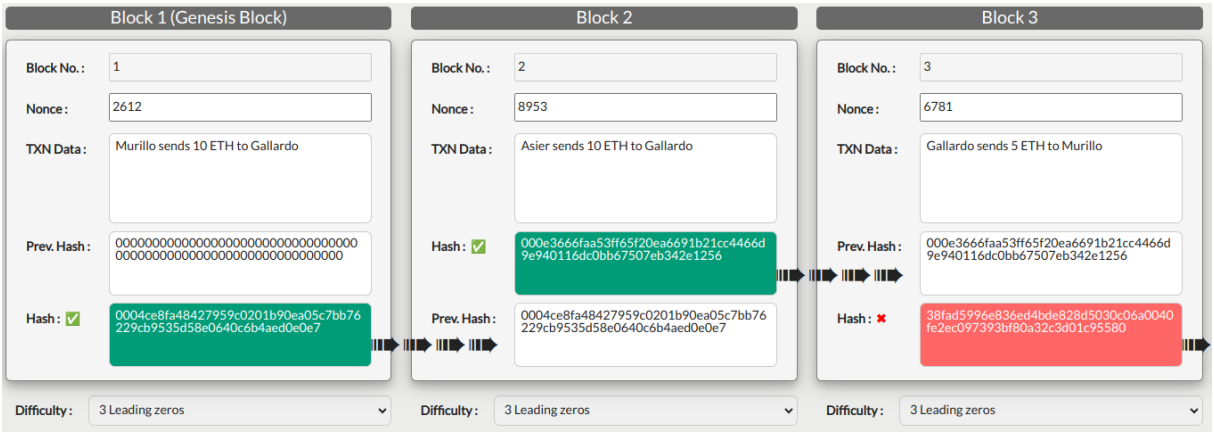


Figura 4: Ejemplo bloque no verificado

2.1.4. Evolución de la Blockchain

Es bastante común la confusión entre los términos Bitcoin y blockchain. La blockchain se considera la tecnología basada en la cadena de bloques, y Bitcoin es la criptomoneda basada en la tecnología blockchain.

La historia de la blockchain comienza en 1991 cuando Stuart Haber y W. Scott Stornetta trabajaron en una cadena de bloques protegida criptográficamente en la que nadie podía manipular las marcas de tiempo de los documentos[15]. Pero no fue hasta 2008 cuando Satoshi Nakamoto, seudónimo de un individuo o grupo desconocido, publicó el white paper de Bitcoin[24]. Las criptomonedas, como Bitcoin, nacen de la necesidad de realizar transacciones seguras sin intermediarios, logrando un sistema financiero descentralizado. Se considera que en este momento empezó la primera etapa, también conocida como Blockchain 1.0. Este periodo fue dominado por Bitcoin, centrado en el desarrollo de un sistema electrónico de transacciones entre individuos sin la necesidad de intermediarios (peer-to-peer). Un avance notorio fue la inclusión de árboles de Merkle, los cuales mejoraban la seguridad y eficiencia de la red al no ser necesario descargar el bloque completo[8].

En 2013 un importante colaborador en el desarrollo de Bitcoin, Vitalik Buterin, junto a otros desarrolladores creó Ethereum[5]. Ethereum nació de la necesidad de eliminar las limitaciones de Bitcoin. A diferencia de Bitcoin, limitada a transacciones, Ethereum dispone la posibilidad de crear contratos inteligentes y aplicaciones descentralizadas. Esta etapa es llamada Blockchain 2.0[31].

Desde 2018 hasta la actualidad se considera que nos encontramos en la etapa 3.0. En este periodo los desarrolladores se han centrado en aumentar las capacidades de los contratos inteligentes y resolver el desafío del triángulo blockchain[23]. Este dilema consiste en que al incremento de escalabilidad, seguridad o descentralización las restantes se ven reducidas. También cabe mencionar el cambio en los métodos de consenso, sistema de decisión sobre el creador de los nuevos bloques, reduciendo consumo energético y emisiones de CO_2[33].

El futuro de las criptomonedas se ve brillante, teniendo infinidad de nuevas innovaciones.

2.1.5. Importancia de la descentralización

La descentralización se ha convertido en un concepto clave para el desarrollo social y político de la sociedad[14]. Se fundamenta en la separación de poder de una entidad u organización central hacía un grupo dividido o separado de nodos. Su principal objetivo consiste en mejorar el nivel de vida de los ciudadanos y es universalmente considerado un pilar fundamental para la democracia.

Los principales beneficios de la descentralización basada en blockchain son:

- Eliminación de intermediarios: la blockchain permite la realización de procesos y transacciones sin la necesidad de intermediarios (peer-to-peer), reduciendo tiempos y costes.
- Transparencia: las redes blockchain son públicas y permiten observar transacciones e información con total libertad.
- Seguridad: la información está distribuida y replicada entre todos los nodos que forman la red, cuanto mayor sea la red mayor será la descentralización.
- Resistencia a la censura: la descentralización elimina la posibilidad de que una entidad autoritaria centralice el poder y ejerza censura. La blockchain permite a particulares acceder y compartir información libremente.

2.2. Contratos Inteligentes

En esta sección se explicará que son los contratos inteligentes y como funcionan. También se estudiarán sus posibles casos de uso y aplicaciones. Finalmente, se analizará la Ethereum Virtual Machine, esencial para el funcionamiento de la red de Ethereum.

2.2.1. ¿Qué son los contratos inteligentes?

Un contrato inteligente es un programa que se ejecuta en una red de blockchain. Al igual que un programa convencional, está formado por funciones. Las funciones pueden desde calcular y almacenar datos, hasta realizar transacciones con su propio balance. Para interactuar con los contratos, los usuarios deberán usar las funciones definidas en su código. En la red de Ethereum, los contratos tienen direcciones únicas, similares a las de los usuarios, y al igual que estas, tienen un balance sujeto a transacciones. Los contratos son permanentes y sus acciones irreversibles.

2.2.2. Aplicaciones

Los contratos inteligentes pueden usarse en gran variedad de sectores. A continuación, se presentan seis de los casos de uso más comunes e importantes:

- DAO y votaciones: Una DAO (Decentralized Autonomous Organisation) son comunidades controladas, total o parcialmente, por sus miembros a través de contratos

inteligentes, realizando votaciones y transacciones reguladas en la blockchain. Los miembros que forman estas organizaciones pueden comprar su entrada en la organización a través del uso de token de gobernanza. Se realizan votaciones para la toma de decisiones de la organización, limitadas a los poseedores de los tokens. El objetivo final de la DAO es ser completamente descentralizada y operar totalmente sin depender del control e intervención de un moderador o intermediario.

- Crowdfunding: Crowdfunding es el método más común para una empresa en crecimiento de obtener fondos para sus proyectos, planes de negocio e ideas. También puede ser una eficaz manera de formar una comunidad detrás de tu producto, además de traer nuevos clientes. Los contratos inteligentes pueden programarse para acumular los fondos de los clientes y, una vez recaudado el capital deseado, realizar las acciones previstas con anterioridad[27]. Los colaboradores también podrán ser recompensados a través de beneficios subsecuentes del proyecto incluso otros privilegios como pueden ser coleccionables o acceso anticipado a los servicios.

- Registro de la propiedad: La blockchain permite almacenar información inalterable de manera sencilla, rápida y económica. Un ejemplo de uso podría ser la eliminación de la necesidad de emplear notarios o abogados a la hora de identificar y registrar la propiedad de cualquier bien, como pueden ser bienes raíz, joyas o obras de arte[17]. Un método de registro de propietario de un bien o producto sería generar un token identificativo, también conocido como tokenizar. El proceso consiste en la creación de un contrato en el cuál se registra el autor de manera segura. Esta nueva tecnología también significa que por primera vez, el vendedor tiene la capacidad de manejar la transacción completamente por sí mismo. El sector bancario también puede verse beneficiado de préstamos más accesibles, rápidos y seguros.

- Seguros: La industria de seguros se gasta decenas de millones en los procesos de reclamación, sin contar además con los problemas ocasionados a clientes. Muchos usuarios protestan que las aseguradoras no siempre cumplen con lo acordado. Los contratos inteligentes protegen los acuerdos y aceleran las transacciones, protegiendo tanto a empresas y clientes además de reducir costes. En el largo plazo también se verá beneficiado del IoT (Internet of Things), como puede ser conectando coches y contratos.

- Inventario y cadena de suministro: El IoT también puede ser beneficiado por el uso de los contratos inteligentes, controlando todas las fases de las cadenas de producción como del transporte y la entrega. Otro gran beneficio sería la reducción de robos o desapariciones de productos, al detectar el momento en el que ocurran los hechos y poder garantizar al consumidor el origen de los productos.

2.2.3. Ethereum Virtual Machine

La red de Ethereum se puso en funcionamiento el 30 de julio de 2015 sacudiendo el mundo blockchain y las criptomonedas. Hasta ese momento las monedas más relevantes eran Bitcoin[24] y Litecoin[6], enfocadas en las transacciones peer-to-peer, Ethereum introdujo Ethereum Virtual Machine (EVM) proporcionándole la capacidad de ejecutar código en la red. EVM supuso una gran innovación, la cuál fue copiada por una gran cantidad de redes, actuando como modelo de referencia.

EVM es la pieza central de la red de Ethereum y los contratos inteligentes, además de colaborar en la descentralización y adopción. La EVM se encuentra dentro del software empleado en los nodos de la red. Al igual que en Bitcoin los nodos contienen una copia del libro contable, registro donde se almacenan todas las transacciones realizadas en la red. Las funciones ejecutadas se interpretan en un bajo nivel. Sin embargo, la información es transcrita a bytecode empleando Solidity[28]. Este bytecode es ejecutado por la EVM actuando como si la red fuera un ordenador. En la Figura 5 se muestra la estructura de la EVM. Solidity es el lenguaje de programación en el cuál se implementan los contratos. Otro lenguaje menos empleado es Vyper[32].

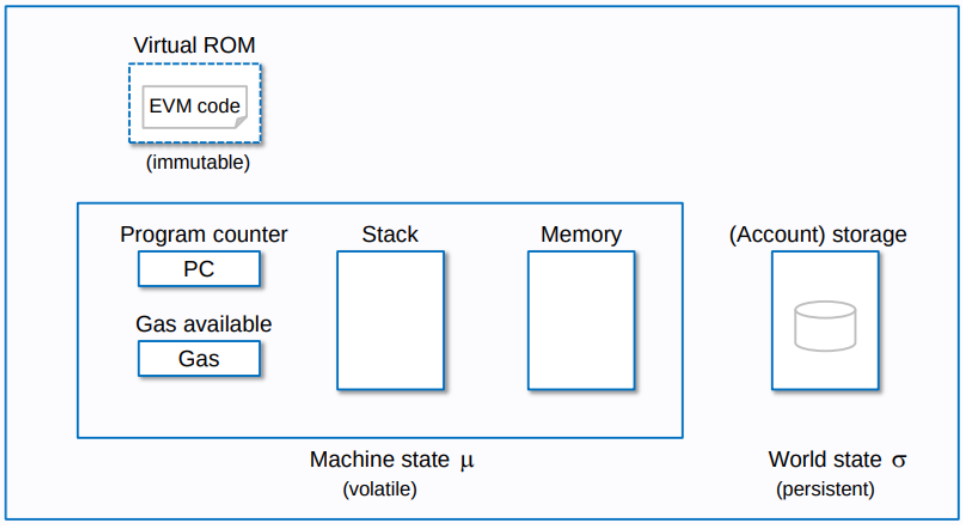


Figura 5: Ethereum EVM Ilustrado[29]

Ethereum cuenta con un token nativo con el cual se realizan transacciones. Cuando se ejecuta una función, el minero que añade el nuevo bloque emplea poder computacional para ejecutar las funciones y transacciones. Como recompensa se cobra una pequeña tarifa de ether (moneda de Ethereum) llamada gas. También es usada para evitar el sobreuso de la red. El precio del gas a día 8 de Noviembre de 2023 se encuentra a 30 gwei, o lo que sería lo mismo que 1.07$. La tarifa se verá afectada por el poder computacional requerido por la transacción.

Capítulo 3

Análisis, casos de uso y herramientas utilizadas

En este capítulo, se realizará un análisis y casos de uso del proyecto a realizar. Se establecerá la estructura de la organización y las relaciones y privilegios de los miembros, además de concebir los vínculos con las empresas e instituciones externas. Finalmente, se terminará mostrando todas las herramientas necesarias para el desarrollo del trabajo.

3.1. Análisis

En esta sección se realizará una visión general del proyecto, mostrando la estructura de los contratos inteligentes y como los usuarios emplearán una aplicación web para interactuar con los mismos. También se enumerarán todos los requisitos del trabajo. Finalmente, se explicarán los distintos casos de uso.

3.1.1. Visión General

El objetivo principal del proyecto consiste en crear una DAO para mutualizar los gastos personales en salud. En este sistema cualquier persona podrá ser miembro pagando una comisión de entrada. La organización se desarrollará haciendo uso de un contrato inteligente de la red de Ethereum, lo que permite un sencillo acceso y una disponibilidad total. El nombre de la asociación sera ODAFS (Organización descentralizada Auto-organizada para Fondos Sanitarios).

ODAFS funcionará igual que cualquier seguro médico, los miembros deberán pagar cuotas en forma de mensualidades. Los integrantes podrán usar los fondos para pagar sus tratamientos médicos o productos farmacéuticos en los centros habilitados.

Al gestionar grandes cantidades de fondos la descentralización se convierte en esencial. Muchos seguros o empresas públicas han sufrido retrasos o problemas a la hora de realizar los pagos a sus usuarios. La descentralización de blockchain permite la creación de una organización incorruptible gracias a su tipo de arquitectura. La organización será administrada por los propios miembros, lo que empoderará al usuario. Se emplearán votaciones para la toma de decisiones, en las cuales solo pueden intervenir miembros. Al

unirse a ODAFS se creará un NFT, necesario para poder participar en las distintas decisiones. ODAFS interactuará con instituciones externas, hospitales y farmacias, con las que llegará a acuerdos para ofrecer consultas, tratamientos y medicamentos a través de contratos inteligentes. Los miembros podrán pagar sus servicios utilizando los fondos de la organización.

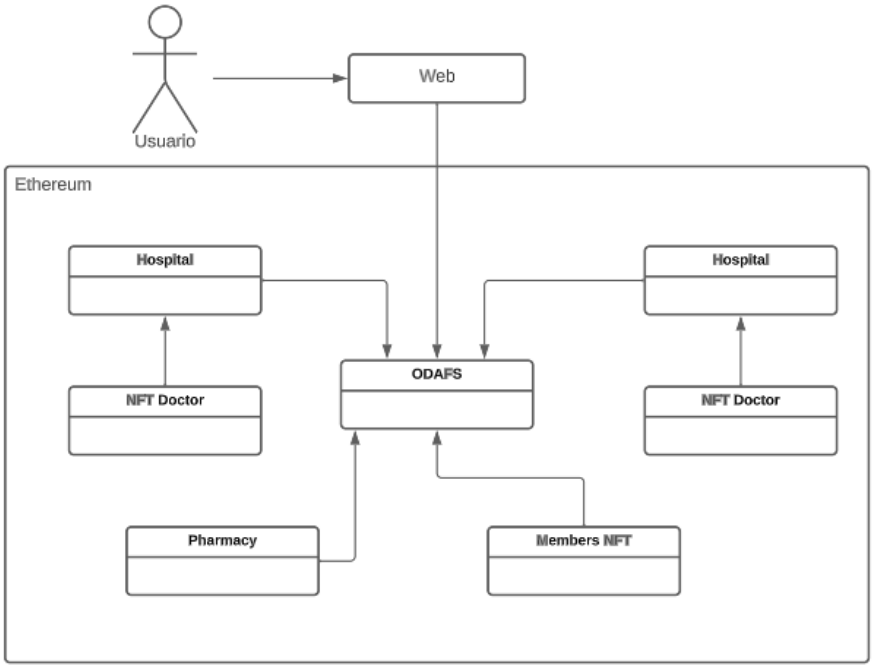


Figura 6: Diagrama UML contratos inteligentes

En la Figura 6 se muestra la estructura de los contratos del proyecto. Para interactuar con la blockchain el usuario deberá usar un prototipo web, simplificando y acelerando las tareas. Como se puede observar, el contrato ODAFS funciona como enlace entre las farmacias, hospitales y miembros. El usuario podrá elegir que entidad o contrato le brindará los servicios que necesite.

3.1.2. Análisis de requisitos

- Requisitos funcionales: describen funcionamientos o servicios que el software debe proporcionar para su correcto uso. En la Tabla 2 se identifican una lista de requisitos de ODAFS. El proyecto también creará una serie de contratos representando farmacias y hospitales, cuyos requisitos no serán mencionados.

- Requisitos no funcionales: describen una colección de características de calidad y rendimiento necesarias en el diseño e implementación. En la Tabla 3 se identifican y explican los distintos requisitos no funcionales de ODAFS.

- Requisitos específicos del sistema: estos requisitos son detalles de la tecnología de desarrollo necesarios para su funcionamiento. En la Tabla 4 se recogen los requisitos específicos de ODAFS.

Requisito	Nombre	Descripción
RF1	Unión a la organización	Cuando un miembro se una a la organización se debe crear un NFT el cuál actuará como tarjeta de identificación.
RF2	Renovación membresía	Si un miembro posee la tarjeta de identificación podrá renovar su membresia realizando un pago, los meses renovados dependerán de la cantidad aportada.
RF3	Visualización de estado de membresía	Se deberá mostrar a cualquier individuo si pertenece a ODAFS y la fecha de finalización de afiliación.
RF4	Realización de donaciones	Cualquier individuo podrá realizar donaciones a los fondos solidarios.
RF5	Visualización de fondos	Se deberá mostrar la cantidad de fondos reservados a los miembros y aquellos con fines solidarios.
RF6	Creación de votación	Los miembros podrán comenzar votaciones en cualquier momento usando una breve descripción de la propuesta. Sera considerará exitosa si un numero mayor al 50 % del número de miembros en el comienzo de la votación.
RF7	Envió de voto	Los miembros podrán realizar sus votos tanto a favor como en contra de las propuestas.
RF8	Visualización de propuestas y resultado	Se mostrarán los resultados de las votaciones, tanto de las concluidas como las pendientes.
RF9	Tratamientos en hospitales	Los miembros podrán solicitar atención médica en un hospital, el tratamiento será pagado por ODAFS. Se creará un servicio para aquellas personas que no puedan costearse el servicio.
RF10	Comercio en farmacias	Los miembros podrán solicitar el pago de medicamentos en farmacias a ODAFS. Se creará un servicio para aquellas personas que no puedan costearse el servicio.

Tabla 2: Requisitos funcionales

Requisito	Nombre	Descripción
RNF1	Total Seguridad	La organización podrá disponer de grandes cantidades de fondos, por lo que las transferencias y tomas de decisiones deberán ser fiables y confiables.
RNF2	Descentralización	La gestión de la organización deberá estar distribuida entre todos sus miembros.
RNF3	Disponibilidad continua	La organización y uso de sus fondos deberá ser accesible a tiempo completo.
RNF4	Transparencia	Todas las transacciones, votaciones y tomas de decisiones deben ser totalmente transparentes al público.
RNF5	Aplicación web sencilla e intuitiva	Para facilitar el uso de las distintas funciones del contrato, la aplicación web deberá ser sencilla y facilitar el uso de las funcionalidades de ODAFS.

Tabla 3: Requisitos no funcionales

Requisito	Nombre	Descripción
RE1	Red Ethereum	Los contratos se ejecutaran en la red blockchain de Ethereum.
RE2	Smart Contracts usando Solidity	Para implementar los contratos inteligentes utilizará el lenguaje de alto nivel Solidity.
RE3	Aplicación web	Se creará una aplicación web mediante los lenguajes de HTML y Javascript.

Tabla 4: Requisitos específicos del sistema

3.2. Casos de uso

En esta sección se examinarán los casos de uso más importantes del proyecto, proceso de registro y renovación de membresía, compra de medicamentos y pagos de tratamientos médicos. Estos casos de uso representarán, describirán y especificarán los procesos de interacción desde el punto vista de un usuario. Por cada escenario de interacción se creará una tabla descriptiva y un diagrama UML mostrando el flujo.

3.2.1. Registro y renovación de membresía

Este caso de uso representa ambos procesos de registro y renovación, ya que ambas situaciones son resueltas por las mismas funciones. Los datos de este caso de uso están recogidos en la Tabla 5 y representados en la Figura 7.

	Descripción
Nombre	Registro y renovación de membresía en ODAFS
Actores	Usuario que quiere entrar en ODAFS o renovar su membresía
Descripción	El caso de uso comienza cuando el usuario accede a la sección de Inicio de la aplicación web. Una vez el usuario ha accedido, se comprueba si ya pertenece a la organización. A continuación, se mostraráN dos opciones, renovar o unirse, dependiendo de si ya es miembro o no. En el caso que el usuario ya sea miembro, se mostrara la fecha de caducidad de su membresía. Finalmente, el usuario podrá comprar o renovar la membresía por el periodo que desee.
Flujo principal	1. El usuario accede a la sección de Inicio. 2. Se comprueba si el usuario es miembro de ODAFS. 3. Se muestra el botón de renovar o unirse, dependiendo si es miembro o no. 4. El usuario renueva o registra en la organización.

Tabla 5: Caso de uso - Registro y renovación de membresía en ODAFS

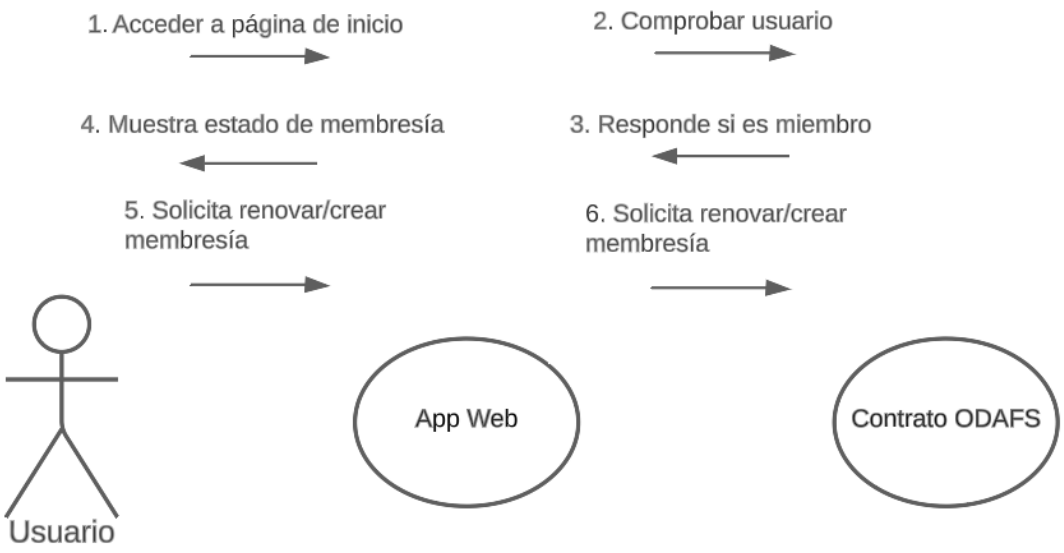


Figura 7: Caso de uso - Registro y renovación de membresía en ODAFS

3.2.2. Compra de medicamentos

Los datos de este caso de uso están recogidos en la Tabla 6 y representados en la Figura 8.

	Descripción
Nombre	Compra de medicamentos en farmacias
Actores	Usuario que quiere compra medicamentos a través de ODAFS
Descripción	El caso de uso comienza cuando el usuario accede a la sección de farmacia. El usuario podrá elegir, entre una lista, un producto que le interese. A continuación, se mostrará toda la información del producto, incluyendo el precio. Finalmente el usuario podrá comprar una determinada cantidad del medicamento.
Flujo principal	1. El usuario accede a la sección de farmacia. 2. El usuario elige un producto. 3. La web muestra información del producto. 4. Comprar producto con fondos de ODAFS

Tabla 6: Caso de uso - Compra de medicamentos en farmacias

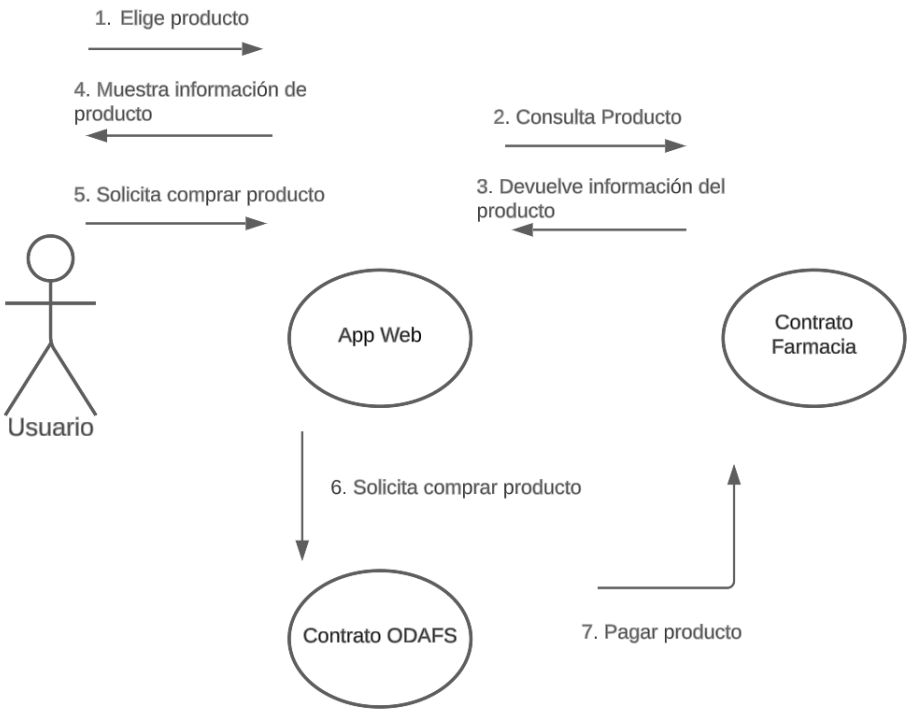


Figura 8: Caso de uso - Compra de medicamentos en farmacias

3.2.3. Atención Médica

Los datos de este caso de uso están recogidos en la Tabla 7 y representados en la Figura 9.

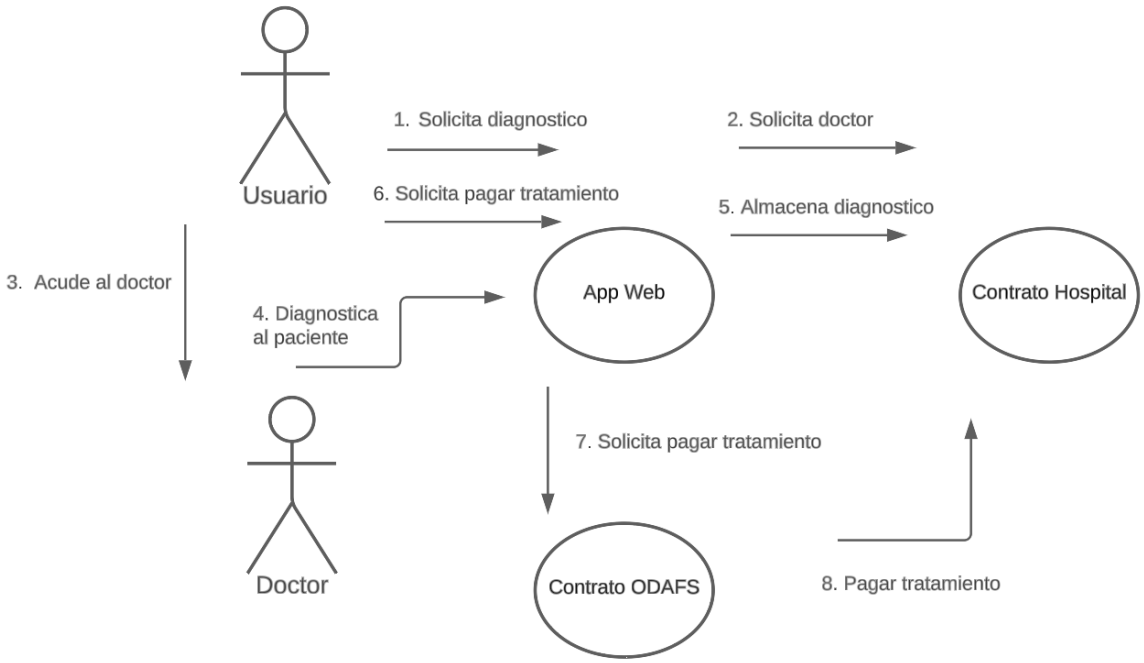


Figura 9: Caso de uso - Solicitar diagnostico y pago del tratamiento

	Descripción
Nombre	Solicitar diagnóstico y pago del tratamiento
Actores	El usuario que solicita el diagnóstico y el doctor que lo realiza
Descripción	El usuario accederá a la sección de pacientes y solicitará una cita con un doctor. El paciente recibirá un id, ese número deberá ser entregado cuando se lleve a cabo al cita. A continuación, el doctor rellenará un formulario con la información y coste del tratamiento. Finalmente, el paciente pagará el tratamiento con ODAFS.
Flujo principal	1. El usuario accede a la sección de paciente. 2. El usuario solicita una cita. 3. El usuario recibe un número de espera. 4. Cuando el doctor recibe al paciente, este debe entregarle su id de la cita. 5. El doctor rellenará un formulario con el diagnostico. 6. El paciente accede a la aplicación web y recibe su diagnostico. 7. El paciente paga el tratamiento con fondos de ODAFS.

Tabla 7: Caso de uso - Solicitar diagnostico y pago del tratamiento

3.3. Herramientas utilizadas

A continuación, se expondrán las tecnologías necesarias para la creación del proyecto. Entre ellas encontramos el entorno de desarrollo, administrador de paquetes, frameworks y lenguajes empleados.

3.3.1. Yarn

Un gestor o administrador de paquetes es una herramienta informática encargada de registrar el software instalado en un ordenador, facilitando la instalación, actualización o borrado. En este trabajo el administrador gestiona el software empleado en el proyecto. A la hora de migrar, trabajar en equipo o utilizar controles de versiones resulta esencial, permitiendo a los desarrolladores mantener el software de sus equipos actualizado.

El administrador más popular es Node Package Manager (npm)[1]. npm es un proyecto de código abierto desarrollado en JavaScript, ideal para integrar con Node.js. Sin embargo, se ha optado por Yarn[2], otro gestor de paquetes más moderno y con algunos importantes beneficios. Algunas de sus ventajas respecto a su predecesor son:

- Yarn contiene una mejor interfaz de usuario con una línea de comandos más sencilla, resultando en una experiencia de usuario más satisfactoria y acelerando el desarrollo.

- La eficiencia es mayor respecto a npm. Yarn guarda en cache todos los paquetes descargados eliminando la necesidad de descargar el paquete cada vez que se quieran añadir los archivos a un nuevo proyecto.
- El archivo "package-lock.json" de npm, donde se registran y archivan las versiones de los paquetes, es reemplazado por "yarn.lock" más detallado.
- Además yarn es compatible con npm facilitando el trabajo simultaneo en varios proyectos.

3.3.2. VSCode

Virtual Study Code[18], también conocido como VSCode, es el editor de texto empleado para el desarrollo del proyecto. Se ha elegido este Entorno de Desarrollo Integrado (IDE) debido a la facilidad para adaptar su configuración a la requerida por el proyecto. Al dividir la implementación en dos partes (backend y frontend) resulta esencial una correcta configuración. Dos extensiones utilizadas han sido Solidity, para implementar los contratos, y Live Server, el cuál crea un servidor local en el equipo para visualizar los cambios realizados a la aplicación web.

3.3.3. Hardhat

El desarrollo e implementación de Contratos Inteligentes es un proceso muy tedioso. A diferencia de otros campos de la informática los programas una vez ejecutados son inmodificables e irreversibles. Si al crear un contrato no se detecta un error, este permanece de manera indefinida en la red. Como apoyo se ha utilizado el framework Hardhat[13].

Hardhat es un entorno de desarrollo de Ethereum, y otras redes Blockchain de 2ª generación, que permite a los desarrolladores realizar tests, desplegar contratos en la red y simular redes de forma local.

3.3.4. Metamask

Metamask[12] es una extensión de navegador web empleado para almacenar las claves privadas del usuario de manera segura. Cuando se interactúa con una Aplicación Descentralizadas (DApp), Metamask actúa como intermediario entre la red y el usuario. Permite firmar transacciones, gestionar distintas carteras y conectar con DApp. Es compatible con redes tanto públicas como locales.

3.3.5. Node.js

Node.js[9] es un entorno de ejecución de código abierto usado para ejecutar código JavaScript. Node.js permite desarrollar aplicaciones en Javascript en el lado del servidor. Sus características clave encontramos:

- Node.js se basa en un modelo asíncrono de programación (event-driven y non-blocking), manejando simultáneamente varias solicitudes y eventos, convirtiéndolo en altamente escalable y eficiente.

- Cuenta con una gran velocidad de ejecución debido a su depurado motor V8 de Google Chrome.
- Node.js cuenta con un gran ecosistema de módulos y de frameworks, a través de npm y otros gestores de paquetes.
- Compatibilidad con multitud de plataformas: Windows, Mac, Linux...

3.3.6. Lenguajes de programación

En el proyecto se emplean 3 principales lenguajes de programación:

- Solidity[28]: propuesto por Gavin Wood y más tarde desarrollado por Christian Reitwiessner, consiste en un lenguaje orientado a objetos para el desarrollo de contratos inteligentes y DApps en redes Blockchain, principalmente Ethereum. Solidity cuenta con distintas versiones, en este proyecto se ha optado por emplear la versión 0.8, una versión estable sin fallos que puedan alterar el funcionamiento del proyecto.
- Lenguaje de Marcas de Hipertexto(HTML): conforma el componente más básico de una web. Usado para definir la estructura y el significado de la aplicación web del proyecto.
- JavaScript: lenguaje interpretado de programación que permite la mejora y creación de la interfaz de usuario y aplicación web del proyecto. Además de actuar como el intermediario entre los contratos inteligentes y el prototipo web usando la librería de ethers.js.

3.3.7. Control de versiones

En este proyecto se emplea un control de versiones para supervisar todos los cambios en el código fuente del desarrollo, concretamente dos repositorios (backend y frontend). Mantener un registro de las modificaciones de proyecto resulta esencial, ya que otorga la posibilidad a los desarrolladores de retroceder la versión ante un posible error y minimizar el daño o perdida de tiempo. Para trabajos en equipo donde el código es modificado simultáneamente, se estructuran las versiones formando un árbol de archivos y ayudando a la posterior unión del producto final. Github[7] es el software de control de versiones elegido para este proyecto. Siendo la plataforma más escogida por los desarrolladores, cuenta con una comunidad de 90 millones de colaboradores.

Capítulo 4

Implementación Backend

En este capítulo se presentan y explican las diferentes carácteristicas de los contratos inteligentes implementados. Se explicará el objetivo principal de cada contrato, sus funciones y componentes y se acabará mencionando sus relaciones con otros contratos.

4.1. NFTs

Como hemos visto la utilización de NFTs en el proyecto resulta vital para la organización y su gestión de usuarios. Su principal funcionalidad radica en la habilidad de verificar la autenticidad y el propietario de un activo, creando un documento identificativo inalterable.

4.1.1. ERC-721

En Ethereum existen una serie de estándares[11], que definen la estructura y implementación de ciertas funcionalidades. Estos estándares garantizan la interoperabilidad entre diferentes contratos. Los ejemplos más famosos y populares son: tokens fungibles (ERC-20)[30] y tokens no fungibles (ERC-721)[10]. Para realizar una correcta implementación de los NFT se emplea la interfaz IERC-721. Las funciones clave de la interfaz IERC-721 son:

- balanceOf: la función es empleada para determinar el número de tokens que una dirección posee. Como argumento utiliza una dirección y devuelve un número entero. En ODAFS esta función es emplea para verificar si un usuario es miembro o si en lo contrario es necesario que se cree un nuevo NFT.
- ownerOf: se utiliza para determinar el dueño de un NFT. Como argumento utiliza el identificador único del token y devuelve la dirección del propietario del token.

Otras funciones básicas serían approve, transferFrom y safeTransferFrom, sin embargo no serán empleadas en el proyecto, ya que los miembros no enviarán sus NFT a otras direcciones. En el proyecto se ha decidido crear una propia clase NFT, cumpliendo el estándar antes mencionado, empleada como clase padre en los NFT de doctores de hospitales y miembros de ODAFS.

4.1.2. Implentación de NFTs de miembros y doctores

Para el desarrollo de los NFT se ha decidido crear una clase padre, llamada NFT, en el contrato NFT.sol el cual tenga las funciones y elemento básicos del estándar ERC721. Esta clase es un modelo básico propio, adaptado a las necesidades del proyecto. Aunque el proyecto no se realicen transacciones de NFTs, las funciones han sido implementadas. Otro elemento implementado pero no empleado serán los eventos o Events. Los eventos consisten en notificaciones sobre acciones o cambios en el contrato. Además se añadido dos variables para guardar los nombres y símbolos de los NFTs. Los NFT podrán ser usados por otras aplicaciones descentralizadas externas al proyecto. Cualquier clase hijo de NFT cumplirá con los requisitos del estándar ERC721. En la Figura 10 se puede observar el diagrama UML completo para implementar todos los NFTs.

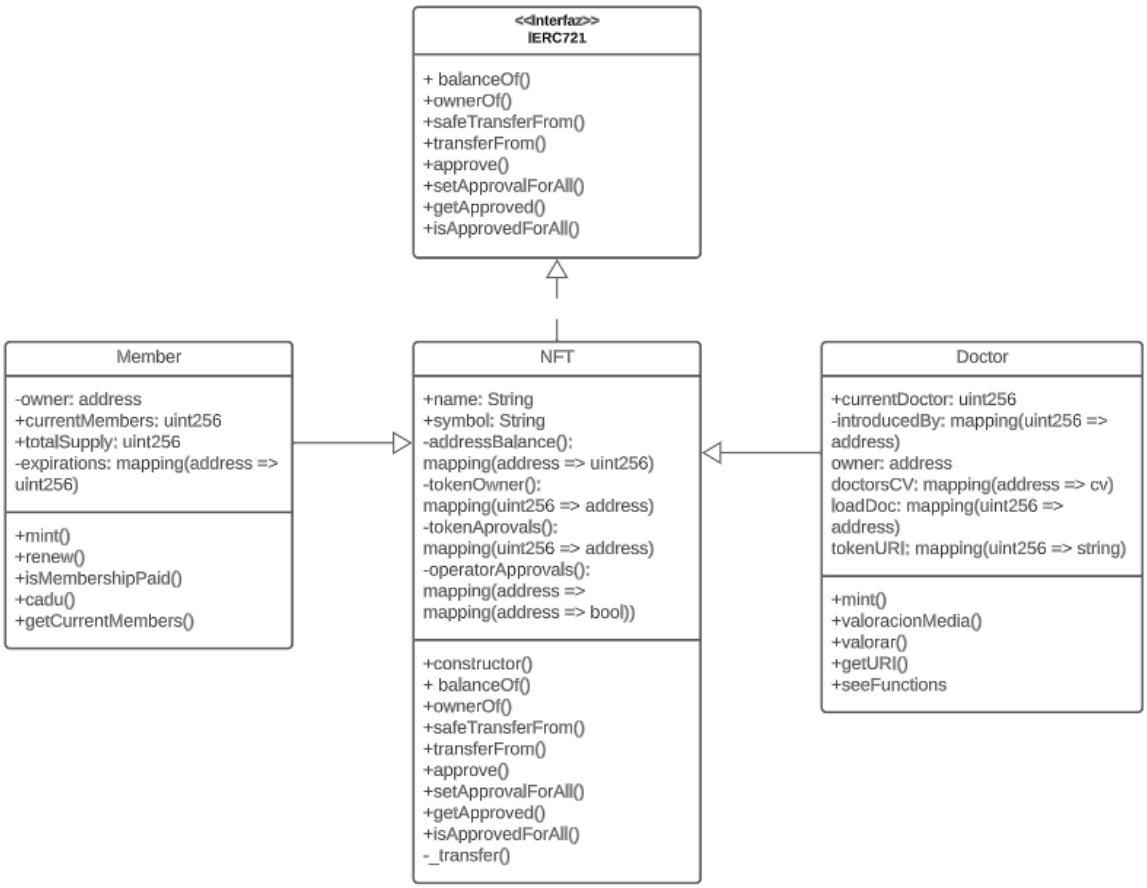


Figura 10: Diagrama UML de NFT

4.1.3. Members

Siendo hijo de la clase NFT, su principal finalidad consiste en gestionar los miembros de ODAFS y controlar el pago de las mensualidades de los mismos. Cada nuevo usuario deberá pagar una tarifa de entrada, una vez pagada se acuñará un nuevo NFT. Este token representará que el usuario es miembro de la organización. Cada NFT contendrá

una variable que indique la fecha de fin de participación en la organización, pudiendo renovarla pagando más mensualidades.

Las funciones fundamentales del contrato son:

- Función constructora: esta función se ejecuta al crear el contrato por primera vez. Almacena la dirección del creador del contrato, ODAFS, y el número máximo de miembros. En la Figura 11 se muestra la función.

```
constructor(string memory name, string memory symbol, uint256 _totalSupply) NFT(name, symbol) {
    totalSupply = _totalSupply;
    owner = msg.sender;
}
```

Figura 11: Member - Función constructor

- onlyOwner: verifica que la dirección que llama a una función sea realmente el creador del contrato. En la Figura 12 se muestra la función.

```
modifier onlyOwner(){
    require(msg.sender == owner,"Not the owner");
    _;
}
```

Figura 12: Member - Función onlyOwner

- mint: si la dirección no es un miembro se crea un nuevo NFT y se crea una fecha de caducidad de su participación en la organización. En la Figura 13 se muestra la función.

```
function mint(address _newMember, uint256 _months) public onlyOwner {
    require(addressBalance[_newMember]==0,"Already a member");
    require(_months > 0,"Not posible");
    require(totalSupply > currentMembers, "Limit of members exceeded");
    addressBalance[_newMember]++;
    tokenOwner[currentMembers] = _newMember;
    currentMembers++;
    expirations[_newMember] = block.number + (_months * 214500);
    //1 day = 7150 blocks  //1 month = 214500 blocks
}
```

Figura 13: Member - Función mint

- renew: renueva o aumenta la fecha de participación de una dirección una determinada cantidad de meses. En la Figura 14 se muestra la función.

```
function renew(address _newMember, uint256 _months) public onlyOwner {
    require(_months > 0,"Not posible");
    if (expirations[_newMember] > block.number) {
        expirations[_newMember] += (_months * 214500);
    } else {
        expirations[_newMember] = block.number + (_months * 214500);
    }
}
```

Figura 14: Member - Función renew

Ethereum no cuenta con ninguna funcionalidad para obtener la hora, esto se debe al que ser formado por sistema descentralizado no podría confiar en un único nodo para determinar la hora. Un método alternativo consiste en utilizar el número del bloque actual y multiplicarlo por la duración media de minado de un bloque, creando una estimación del tiempo en segundos. Se estima que de media se mina un total de 7150 bloques al día, 214500 al mes.

Además de las funciones mencionadas, nos encontramos una serie de funciones de solo lectura, llamadas view, las cuales se utilizan para extraer porciones de información del contrato sin gasto ninguno. Entre ellas encontramos: getCurrentMembers y isMembershipPaid.

4.1.4. Doctors

Siendo el segundo tipo de NFT en el proyecto, su finalidad consiste en funcionar como la licencia de los doctores de los distintos hospitales con los que puedan interactuar los miembros de ODAFS. El contrato Doctor, al igual que Member, es hijo de NFT, cumpliendo con el estándar ERC721. Sin embargo la estructura interna del contrato difiere mucho con el anterior.

En members la única información que se almacenaba de los miembros era su identificador, ahora cada NFT de Doctor almacena una imagen y el currículum de su dueño. Para guardar la imagen se almacena un string con un enlace a una imagen del doctor. Cualquier usuario o aplicación, que desee obtener la imagen, simplemente tendrá que usar la función getURI y emplear el enlace otorgado.

Para almacenar el currículum se ha optado por crear una estructura de datos que almacene toda la información de valor de un doctor, Figura 15. Sus campos son: número de licencia, nombre y apellidos, casos tratados y valoración. Toda la información sera accesible a través de funciones View.

```
struct cv {
    uint256 numLicencia;
    string nombre;
    string apellidos;
    uint256 casosTratados;
    uint256 valoracion;
}
```

Figura 15: Doctor - Currículum

Las funciones constructora y onlyOwner tiene la misma funcionalidad que en Members, en este caso su dueño o creador sera el hospital. La función mint, Figura 16, es solo accesible por el dueño del Hospital. Recibe como argumentos la dirección del nuevo doctor, su número de licencia, nombre, apellidos y foto,en forma de enlace. Al igual que en members el NFT puede conectarse con otros contratos externos a nuestro proyecto, proporcionando multitud de funcionalidades extra si fuera necesario ampliar o aumentar el tamaño del proyecto.

```
function mint(address doctor, uint256 _numLicencia, string memory _name,
              string memory _apellidos, string memory _photo) public onlyOwner{
    require(addressBalance[doctor]==0,"Already a doctor");
    introducedBy[currentDoctors] = msg.sender;
    addressBalance[doctor]++;
    tokenOwner[currentDoctors] = doctor;
    doctorsCV[doctor].numLicencia = _numLicencia;
    doctorsCV[doctor].nombre = _name;
    doctorsCV[doctor].apellidos = _apellidos;
    currentDoctors++;
    loadDoc[currentDoctors] = doctor;
    //Image
    tokenURI[currentDoctors] = _photo;
}
```

Figura 16: Doctor - Función mint

Las funciones de lectura de este contrato tienen el mismo comportamiento, con la excepción de valoraciónMedia que calcula la valoración media de los servicios de un doctor con los datos almacenados en su currículum. La función valorar actualiza los casos tratados y sus valoraciones. En la Figura 17 se muestra la función valorar y una función de lectura.

```
function valorar(address doctor, uint256 nota) public onlyOwner {
    // Nota debe ser entre el 0 y 5
    require((nota >= 0 && nota <= 5), "Nota debe ser entre 0 y 5");
    doctorsCV[doctor].valoracion += nota;
    doctorsCV[doctor].casosTratados++;
}

function numLicenciaSee(uint256 _num) public view returns (uint256) {
    return doctorsCV[loadDoc[_num]].numLicencia;
}
```

Figura 17: Doctor - Función valorar

4.2. Organización Descentralizada

En esta sección se documentará la estructura de la organización (ODAFS), mostrando el implementación desarrollada para las votaciones para la toma de decisiones de la organización, las funciones para la gestión de los fondos y el contrato central donde se interconectan todas las fracciones del contrato. Cada subsección de este apartado representa un contrato.

4.2.1. Vault

Vault es una clase encargada de la gestión de fondos. Su principal función consiste en el control del saldo disponible. La clase podrá distinguir entre dos fondos, saldo (balance) y saldo solidario (solidaryBalance). Las funciones del contrato permitirán a cualquier usuario realizar pagos o donaciones. En la Figura 18 podemos observar el diagrama UML de la clase.

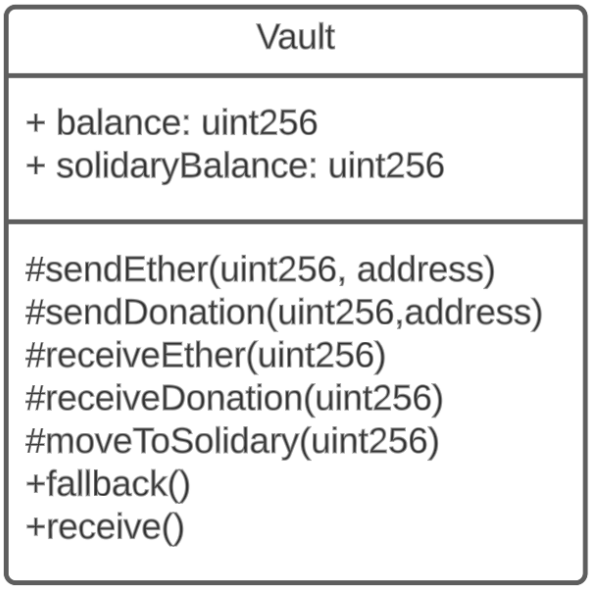


Figura 18: Diagrama UML Vault

El contrato contiene 3 tipos principales de funciones: recibir fondos, mandar fondos y mover fondos a donaciones.

- Recibir fondos: estas funciones son empleadas cuando un nuevo miembro entra o renueva su suscripción en la organización y paga las cuotas. Las funciones deberán comprobar que hay los suficientes fondos para la transacción, realizar la transacción y comprobar la misma, si todo se ha completado con éxito se reducirá los fondos de la cuenta apropiada. En la Figura 19 se muestra un ejemplo de estas funciones.

```
function receiveEther(uint256 _amount) internal {
    require(_amount > 0, "No ether");
    balance += _amount;
}
```

Figura 19: Función para recibir fondos

- Mandar fondos: estas funciones son empleadas para realizar los pagos de los tratamientos y medicamentos, además de las propuestas que resulten en un movimiento de fondos. Estas funciones comprobarán que la cantidad recibida es mayor que cero y reducirán el balance. En la Figura 20 se muestra un ejemplo de estas funciones.

```
function sendDonation(uint _amount, address _receiver) internal {
    require(_amount <= solidaryBalance, "No ether");
    (bool sent, ) = _receiver.call{value: _amount}("");
    require(sent, "Failed to send Ether");
    solidaryBalance -= _amount;
}
```

Figura 20: Función para realizar transacciones

- Mover fondos a donaciones: solo se utilizará si una propuesta propone aumentar los fondos solidarios de ODAFS. La función comprobará que haya suficientes fondos disponibles y reduce la cantidad del balance normal y aumenta los fondos solidarios. En la Figura 21 se muestra la función.

```
function moveToSolidary(uint256 _amount) internal {
    require(_amount < balance, "Not enough ether");
    balance -= _amount;
    solidaryBalance += _amount;
}
```

Figura 21: Función para aumentar fondos solidarios

Si la dirección del contrato recibe una transacción sin llamar ninguna función específica se ejecutan las funciones fallback() y receive(), Figura 22. fallback es llamada cuando un usuario intenta ejecutar una función que no existe en el contrato y receive se ejecuta cuando el contrato recibe una transferencia sin especificar una función. En el contrato toda las transacciones recibidas por estos medios se considerarán donaciones.

```
fallback() external payable {
    solidaryBalance += msg.value;
}

receive() external payable {
    solidaryBalance += msg.value;
}
```

Figura 22: Funciones fallback y receive

4.2.2. DAO

DAO consiste en un contrato donde se han desarrollado todas las funciones necesarias para el control de una DAO. Su objetivo principal es crear propuestas en las cuales se pueda votar y una vez realizadas se tomen decisiones. Se han implementado 2 tipos de votaciones dependiendo de las necesidades de las proposiciones, votaciones sencillas y votaciones para transacciones, las cuales almacenarán una dirección destinataria y una cantidad de fondos. En la figura 23 se muestra el diagrama UML de la clase.

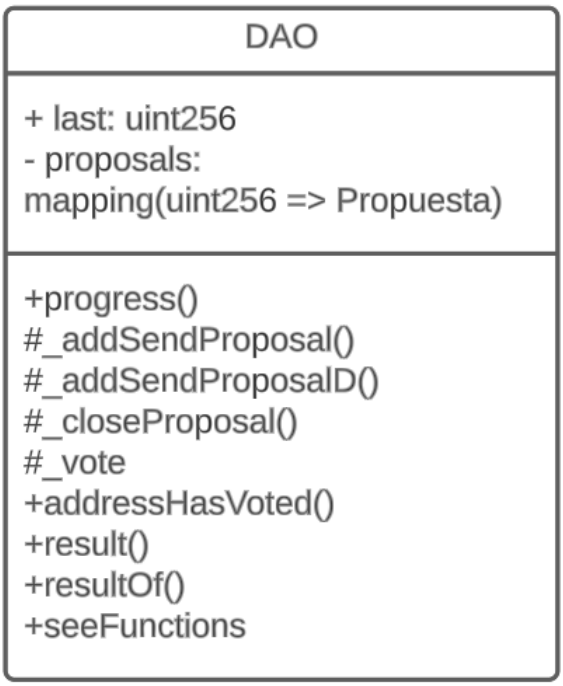


Figura 23: UML de DAO

Para almacenar la información de las votaciones se optado por emplear un mapping de propuestas. Mapping consiste en una estructura de datos muy parecida a un mapa o un diccionario de lenguajes de programación como Java o Python. Cada propuesta estará representada por una estructura de datos formada por información como: si esta activa, votos en contra y a favor, quienes han votado, descripción y, en el caso de ser una propuesta para realizar una transacción, destinatario y cantidad. En la Figura 24 se muestra la estructura de datos.

```
struct Propuesta {
    bool open;
    uint256 inFavor;
    uint256 against;
    mapping(address => bool) hasVoted;
    address fundReceiver;
    uint256 amount;
    string description;
}
```

Figura 24: Estructura de datos de Propuestas

El proceso de votación se separa en tres etapas: creación de propuesta, votación y cierre de propuesta.

- Se puede crear dos tipos de propuestas, aquellas las cuales al ser aprobadas realizan una transacción o en el contrario otras que solo buscan conocer las opiniones de los integrantes. Ambas votaciones tienen el mismo funcionamiento y las funciones implementadas para iniciar propuestas son muy similares, con la excepción de dejar las variables de la cantidad y destinatario de la transacción en valores por defecto. Como se puede observar en la Figura 25, la función añade la información otorgada por el usuario a una nueva clave del diccionario, se ha optado que el índice o clave represente la posición de la propuesta.

```
function _addSendProposalD(address _receiver, string memory _description, uint256 _amount) internal {
    ++last;
    proposals[last].open = true;
    proposals[last].fundReceiver = _receiver;
    proposals[last].amount = _amount;
    proposals[last].description = _description;
}
```

Figura 25: Función para crear propuestas

- Para realizar un voto el usuario deberá indicar el número de la propuesta y indicar si esta a favor o en contra con una variable booleana. La función además comprobará si la votación sigue activa. En la Figura 26 se muestra la función.

```
function _vote(bool _ballot, uint256 _proposalNum) internal hasntVoted isOpenProposal(_proposalNum) {
    proposals[_proposalNum].hasVoted[msg.sender] = true;
    if (_ballot) {
        ++proposals[_proposalNum].inFavor;
    } else {
        ++proposals[_proposalNum].against;
    }
}
```

Figura 26: Función Votar

- En la etapa final de la votación se cambia el parámetro que indica si la votación sigue activa y se devuelve el valor de la transacción si la propuesta ha sido aceptada. En el caso que una propuesta no tenga que realizar ninguna transacción, el valor devuelto por la función sera 0, ya que el valor por defecto de uint256 en Solidity es 0. En la Figura 27 se muestra la función.

```
function _closeProposal(uint256 _num) internal returns(uint256) {
    proposals[_num].open = false;
    return proposals[_num].amount;
}
```

Figura 27: función para cerrar propuestas

Además de las funciones mencionadas, se han implementado funciones para obtener información de las diferentes votaciones.

4.2.3. ODAFS

El contrato ODAFS constituye el centro del proyecto, debe disponer de todas las funcionalidades de los anteriores contratos, a través de herencia, y además poder interactuar con contratos externos, como son Farmacias y Hospitales. Para ello se han desarrollado dos interfaces que deben cumplir los centros de salud y puntos de venta como se puede ver en la Figura 28. Ambas interfaces tienen dos funciones, la primera para determinar el precio que se debe pagar y la segunda para pagar el servicio. En la Figura 29 se muestra el diagrama UML de la clase.

```
// Interface for interacting with pharmacies
interface IPharmacy {
    function buy(uint256 _id, uint _amount) external payable;
    function productPrice(uint256 _id) view external returns(uint256);
}

// Interface for interacting with hospitals
interface IHospital {
    function payTreatment(uint256 _id, uint256 _nota) external payable;
    function priceTreatment(uint256 _id) external returns(uint256);
}
```

Figura 28: Interfaces en ODAFS

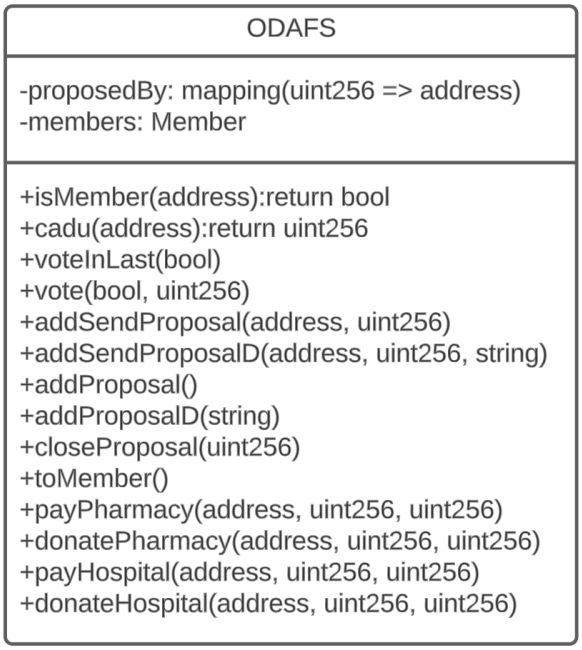


Figura 29: Diagrama UML de ODAFS

La función de pago a través de ODAFS se puede apreciar en la Figura 30, en la cual un usuario puede solicitar a la organización que pague un tratamiento. La función implementada pregunta el precio a la dirección del hospital y realiza una transacción para pagar el tratamiento. Si el pago del tratamiento es solicitado por un individuo externo de la organización se usará el balance solidario. El proceso es idéntico para compras en farmacias.

```
function payHospital(address hospital, uint256 _id, uint _nota) public onlyMember{
    uint256 price = IHospital(hospital).priceTreatment(_id);
    require(price <= balance, "No ether");
    IHospital(hospital).payTreatment{value: price + 1}(_id, _nota);
    balance -= price;
}
```

Figura 30: Pago de tratamiento por ODAFS

Como se ha mencionado con anterioridad, la gestión de miembros se realiza a través de NFTs. Cuando un usuario quiere unirse a la organización debe aportar una cantidad de ETHER determinada. Desde ODAFS también puede renovarse la subscripción aportando ETHER. Para ello se ha desarrollado una función que cumple ambas necesidades. La función determina si un miembro quiere renovar o inscribirse, verificando si un usuario ya es miembro o no, también aumentará los saldos con el ETHER recibido. En la Figura 31 se muestra la función para renovar.

```
function toMember() public payable {
    require(msg.value >= (4 ether), "Not enough ether");
    uint256 months = msg.value / 4000000000000000000;
    if (members.balanceOf(msg.sender) <= 0) {
        members.mint(msg.sender, months);
    } else {
        members.renew(msg.sender, months);
    }
    uint256 money = (msg.value - (msg.value / 20)); // 95%
    receiveEther(money);
    uint256 solidary = msg.value / 20; // 5%
    receiveDonation(solidary);
}
```

Figura 31: Función para unirse a ODAFS

Para verificar si un usuario es miembro, Figura 32, debe comprobarse si posee un NFT de miembro y si su subscripción esta al corriente de pago.

```
modifier onlyMember() {
    require(members.balanceOf(msg.sender)>0, "Not member");
    require(members.isMembershipPaid(msg.sender), "Membership not paid");
    _;
}
```

Figura 32: Función onlyMember

4.3. Hospital

El contrato Hospital representa una entidad privada que proporciona atención médica a sus clientes. Entre sus servicios se encuentra diagnósticos, tratamiento y cuidado de pacientes. Los doctores son los encargados de tratar a los pacientes, para lo que se ha implementado en el contrato los NFT de doctores, Figura 33. Estos doctores solo podrán ser contratados o añadidos por el dueño del hospital, este proceso se encuentra en la función de newDoctor, Figura 34.

```
// Contract's owner
address public owner;

// NFTs of hospital's doctors
Doctor public doctors = new Doctor("Doctor NFTs", "DOC");
```

Figura 33: Dueño y NFT de Doctores en Hospital

```
// Create NFT doctor
function newDoctor(address _newDoctor, uint256 _numLicencia,
          string memory _name, string memory _apellidos, string memory _photo) public payable onlyOwner {
    doctors.mint(_newDoctor, _numLicencia, _name, _apellidos, _photo);
}
```

Figura 34: Función para crear NFT de doctor

El principal caso de uso de Hospital son los tratamientos médicos. Los tratamientos pueden ser un proceso complejo, por lo que se ha optado por desarrollar un sistema de cola de espera. Los usuarios podrán entrar a la lista de espera y permitirá que cada tratamiento se pueda encontrar en una distinta fase que los demás, sin afectar unos a los otros. Este proceso se separa en tres fases:

- Solicitud de tratamiento: para solicitar un nuevo tratamiento el usuario no puede encontrarse en la lista de espera. Cuando se inicia el tratamiento se añade a la lista de espera el usuario. Este usuario recibirá un id, el cuál representa el número de su tratamiento.En la Figura 35 se muestra la función.

```
// Patien request to be diagnose
function requestTreatment() public {
    require(!isInWaitingList[msg.sender], "It is already in the waiting list");
    waitingList[index].patient = msg.sender;
    waitingList[index].status = state.WAITING;
    isInWaitingList[msg.sender] = true;
    waitingTicket[msg.sender] = index;
    index++;
}
```

Figura 35: Función para solicitar tratamiento

- Diagnosticar paciente: cuando el doctor reciba a su paciente deberá solicitarle su número de la lista de espera. Una vez diagnostique al paciente rellenará un formulario con el coste, la descripción y número del tratamiento. La función rellanará los datos y cambiará el estado del tratamiento. En la Figura 36 se muestra la función.

```
// Doctor diagnose patient and creates a budget for the treatment
function diagnosePatient(uint256 _id, uint256 _cost, string memory _diagnostic) public onlyDoctor {
    require(waitingList[_id].status == state.WAITING, "This treatment is not waiting diagnose");
    waitingList[_id].doctor = msg.sender;
    waitingList[_id].diagnostic = _diagnostic;
    waitingList[_id].cost = _cost;
    waitingList[_id].status = state.DIAGNOSED;
}
```

Figura 36: Función para diagnosticar paciente

- Pagar tratamiento: la última fase del procedimiento resulta en el pago y valoración del tratamiento. La función cerrará el expediente del procedimiento y valorará al doctor, cuya nota se añadirá al NFT del doctor. El pago del tratamiento se podrá ejercer de manera privada o a través de ODAFS o cualquier otro tipo de seguro.En la Figura 37 se muestra la función.

```
// Once is paid the treatment starts
function payTreatment(uint256 _id, uint256 _nota) public payable{
    require(waitingList[_id].status == state.DIAGNOSED, "This treatment is not diagnosed");
    require(waitingList[_id].cost < msg.value, "Not enought ether");
    waitingList[_id].status = state.COMPLETED;
    isInWaitingList[waitingList[_id].patient] = false;
    doctors.valorar(waitingList[_id].doctor, _nota);
}
```

Figura 37: Función para pagar tratamiento

Además de todas las funciones antes mencionados se han implementado todos los métodos necesarios para obtener la información de los pacientes y de las distintas fases de los tratamientos. En la Figura 38 se puede observar el diagrama UML de la clase.

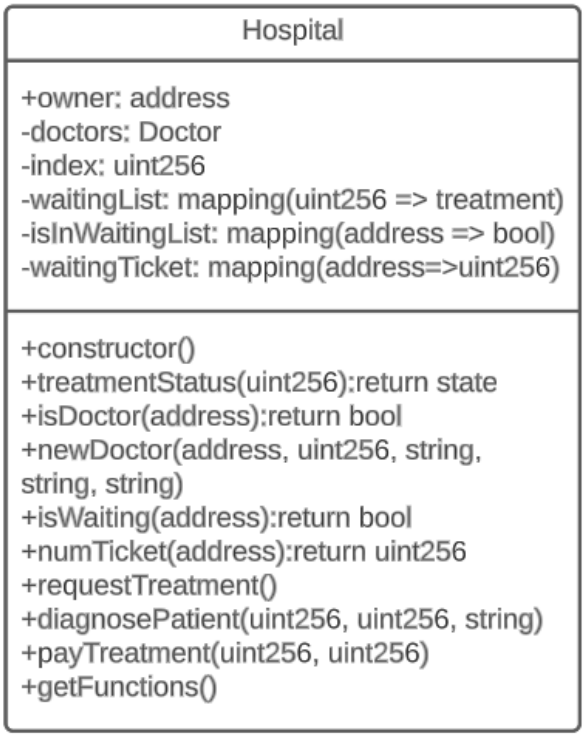


Figura 38: Diagrama UML Hospital

4.4. Farmacia

La otra entidad que proporciona atención a los pacientes son las Farmacias. Al igual que Hospitales, los productos se pueden pagar a través de ODAFS o de forma privada. El dueño del contrato es el encargado de gestionar el inventario de la farmacia, pudiendo agregar nuevos productos y añadir stock, Figura 39.

```
// Add new product to the inventory
function addProduct(string memory _name, string memory _description, uint256 _price) public onlyOwner{
    inventory[numOfProducts].name = _name;
    inventory[numOfProducts].description = _description;
    inventory[numOfProducts].price = _price;
    numOfProducts++;
}

// Add stock to existing product
function addStock(uint256 _id, uint256 _amount) public onlyOwner{
    require(_id < numOfProducts, "Product doesnt exist");
    inventory[_id].stock += _amount;
}
```

Figura 39: Funciones para añadir productos y stock

Para la implementación del inventario se almacena los productos en un diccionario o mapping, Figura 40. Los productos están formados por una estructura de datos que guarda

el nombre, descripción, precio y stock. Una vez se compra un producto, Figura 41, se reduce la cantidad del producto en el inventario.

```
// Pharmacy inventory
uint256 public numOfProducts;
mapping(uint256 => product) inventory;

// Product data type
struct product{
    string name;
    string description;
    uint256 price;
    uint256 stock;
}
```

Figura 40: Estructura de datos del inventario de farmacia

```
// Buy product
function buy(uint256 _id, uint _amount) public payable {
    require(_amount < inventory[_id].stock, "Not enought stock");
    require((inventory[_id].price * _amount) < msg.value, "Not enought ether");
    balance += msg.value;
    inventory[_id].stock -= _amount;
}
```

Figura 41: Función para comprar productos

En la Figura 42 se muestra el diagrama UML del contrato.

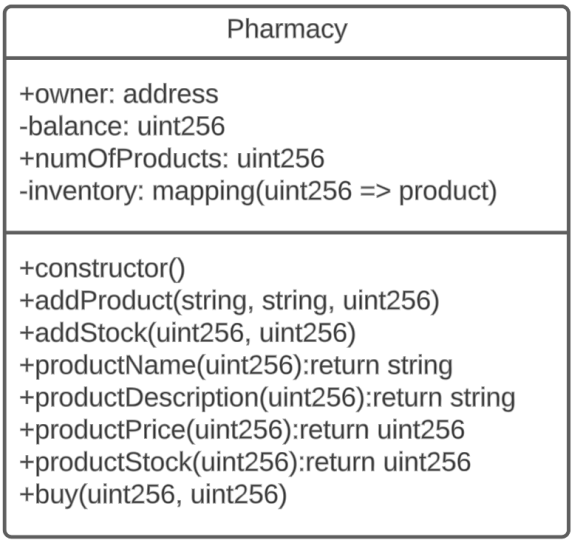


Figura 42: Diagrama UML Farmacia

Capítulo 5

Implementación Frontend

En este capítulo, se mostrará la implementación de la aplicación web y la conexión del usuario con la red de Ethereum. Para la ejecución de pruebas y ejemplos, se ha simulado una red local empleando el framework de Hardhat. El capítulo esta dividido en 7 secciones, se comenzará explicando como el usuario conectará su cartera a Metamask y como interactuará con los contratos de la entidad, continuará explicando la estructura de la cabecera de la aplicación web y las 5 páginas principales de la misma.

5.1. Conexión con Metamask y Contratos Inteligentes

El prototipo web nos permite interactuar con el backend, formado por los contratos inteligentes. Para comunicarse con los contratos es necesario tener una cartera digital con fondos para poder participar en la organización. Para ello se ha empleado la extensión de Metamask, la cuál nos permite gestionar y emplear las claves privadas. Cada vez que un usuario quiera realizar un pago o interactuar con los contratos, la extensión desplegará una ventana mostrando el costo e información de la transacción. En la Figura 43 se muestra un ejemplo de una transacción.

El primer paso, que debe realizar cualquier usuario, es conectar su cartera digital a la aplicación. Una cartera o billetera digital se refiere al software que almacena las claves privadas de un usuario y le permite mandar y recibir transacciones. En este proyecto se emplea Metamask. Se ha desarrollado un botón para que el usuario conecte Metamask, Figura 46a. En el caso que el usuario ya se encuentre conectado con la aplicación web, se mostrará la dirección de la cartera, Figura 46b.

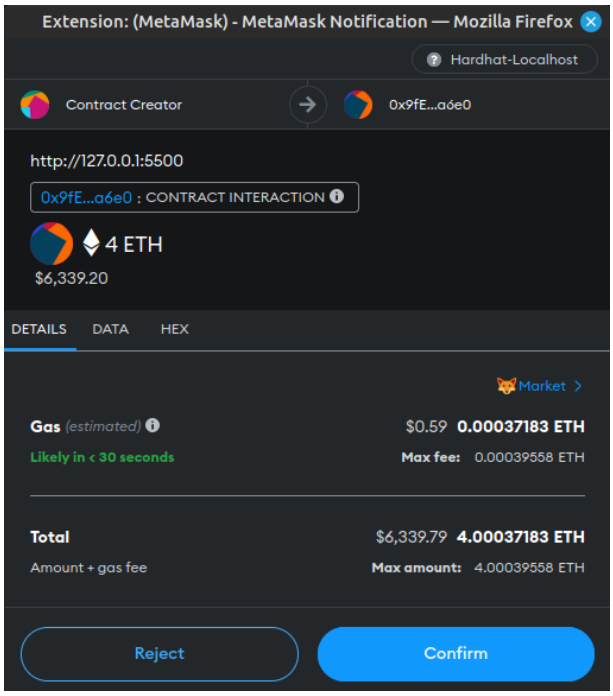


Figura 43: Extensión Metamask

(a) Cartera sin conectar (b) Cartera conectada

Figura 44: Botón para conectar cartera

Una vez la cartera esta conectada con la aplicación el siguiente paso es interactuar con el contrato empleando las funciones implementadas. Para poder acceder o interactuar con las funciones se han implementado una serie de scripts en JavaScript, los cuales proporcionan funcionalidades a diferentes elementos del código fuente. Se han utilizado la librería ethers y funciones asíncronas, aquellas que se ejecutan de manera independiente de otras tareas o eventos en el programa. Todos estas funciones tendrán una estructura en común en la que se conectan con la red de Ethereum y se emplean las claves privadas del usuario para firmar las transacciones, además de obtener y calcular valores auxiliares. En la Figura 45 se muestra un ejemplo de utilización de la librería ethers.

```
//toMember
const toMemberButton = document.getElementById("toMemberButton")
toMemberButton.onclick = toMember

async function toMember(){
    const months = document.getElementById("months").value
    const ethAmount = String((months * 4))
    console.log(`Member with ${ethAmount}...`)
    if (typeof window.ethereum !== "undefined") {
        const provider = new ethers.providers.Web3Provider(window.ethereum)
        const signer = provider.getSigner()
        const contract = new ethers.Contract(contractODAFS, abi, signer)
        const transactionResponse = await contract.toMember({
            value: ethers.utils.parseEther(ethAmount),
        })
    } else {
        console.log("No metamask")
    }
}
```

Figura 45: Ejemplo función que emplea la librería ethers

5.2. Cabecera

Como se ha mencionado con anterioridad la página web esta formada por 5 secciones principales. Para facilitar la navegación entre las diferentes áreas se optado por la implementación de una cabecera que facilite el desplazamiento por la aplicación.

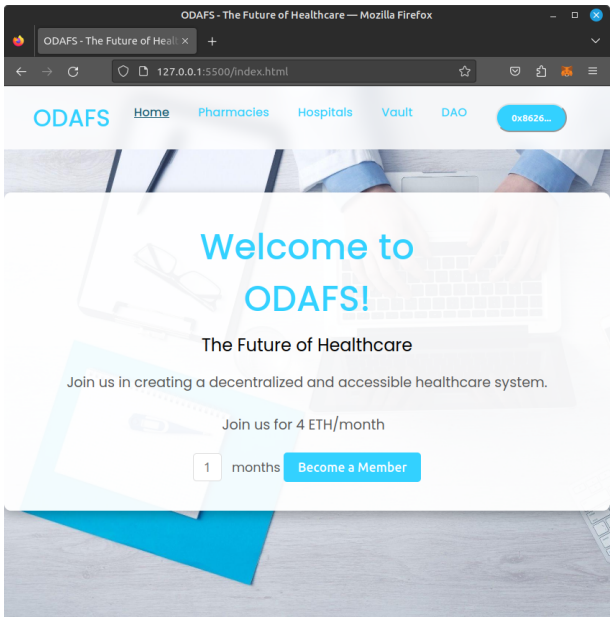
La cabecera tiene tres principales funcionalidades: indicar la sección actual, cambiar la página a la deseada y cargar la pantalla de inicio clickando el logo situado en el lado izquierdo.

5.3. Inicio

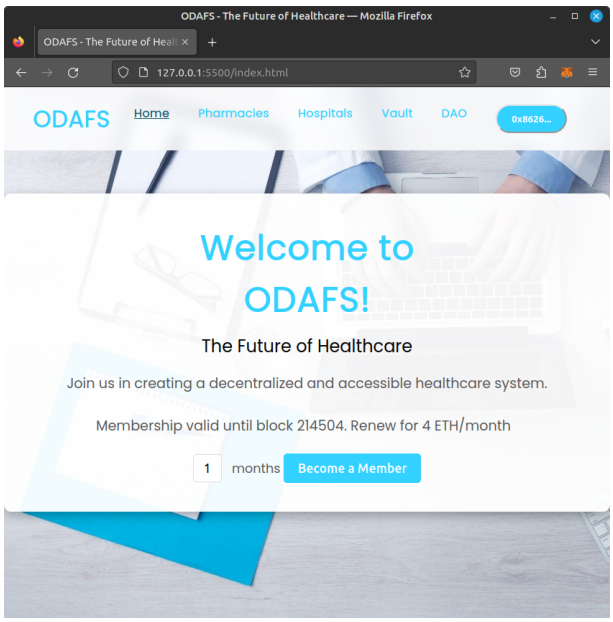
La primera página a la que accede el usuario es Inicio. Se muestra un mensaje de bienvenida y se indica el precio de la membresía. Una vez la billetera del usuario esta conectada a la aplicación, podremos encontrar dos resultados:

- El usuario no es miembro ODAFS: en esta situación se mostrará la opción de unirse y su precio.
- El usuario es miembro: en este caso se mostrará la fecha de caducidad de su afiliación y permitirá aumentar la participación del usuario.

En la Figura 46 se presentan ambas opciones.



(a) El usuario no es miembro



(b) El usuario es miembro

Figura 46: Inicio aplicación web

5.4. Farmacia

Esta sección lleva a cabo la función de un establecimiento virtual, en este caso una farmacia. Esta formado por dos elementos principales, el estante de los productos y el formulario para añadir nuevos artículos.

- En el apartado izquierdo encontramos Products el cual funciona como un estante, en cual se puede buscar productos y comprarlos. Para ver información de un producto

se deberá seleccionar el id y clickar el botón See product. Además de la opción de comprar estará disponible la opción de solicitar que ODAFS done el producto. Los productos serán pagados por la organización a los miembros.

- En el apartado derecho se ha implementado un formulario para que los farmacéuticos añadan productos. Para añadir un producto hay dos fases: añadir información y descripción del nuevo producto y añadir stock usando el id del medicamento.

En la Figura 47 se muestra la página completa de farmacia.

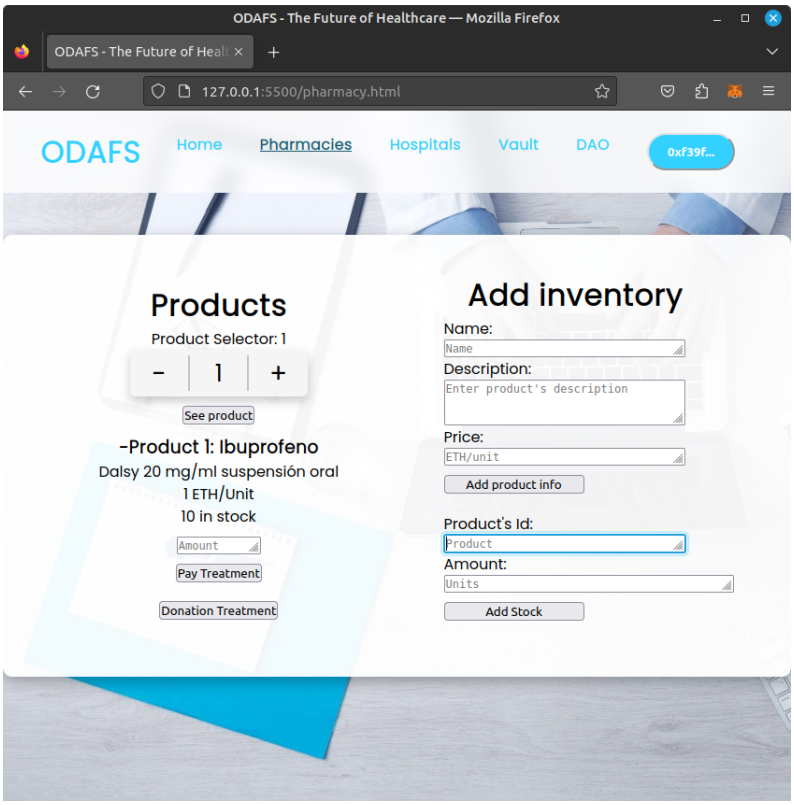


Figura 47: Farmacia aplicación web

5.5. Hospital

En hospital encontramos dos apartados: Doctor y Patient, cada una nos lleva a un apartado distinto.

5.5.1. Doctores

Esta primera subsección se encuentra destinada para ser usadas por doctores y personal del Hospital. Se han desarrollado dos casos de uso:

- Diagnostico de pacientes: cuando un doctor diagnostique un paciente deberá rellenar un formulario con el ID de la cita, diagnostico y precio del tratamiento.

- Gestión de doctores: se podrá observar los diferentes doctores trabajando en el hospital, se mostrará la imagen de perfil almacenada en el NFT además de toda la información del doctor. También se ha implementado un formulario para añadir nuevos doctores al hospital. Esta ultima función solo podrá ser empleada por los dueños del hospital.

En la Figura 48 se muestra el visualizador de doctores y el panel de diagnóstico empleado por los doctores.

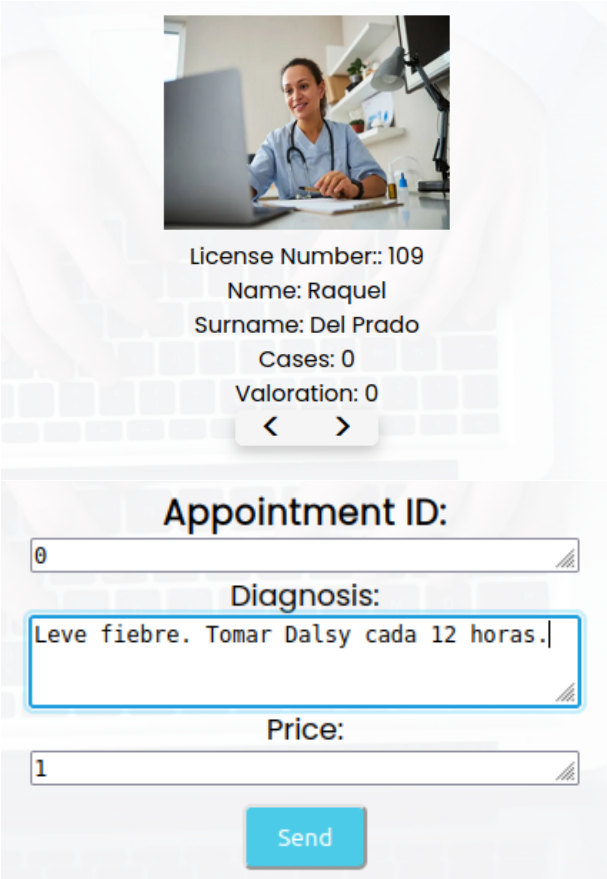


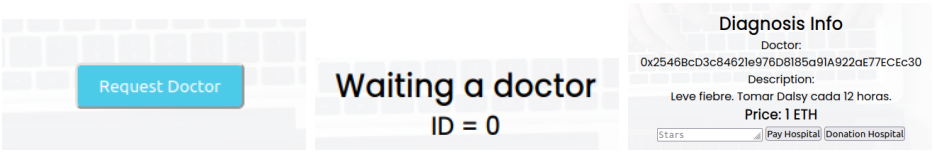
Figura 48: Doctores aplicación web

5.5.2. Pacientes

Cuando un paciente quiera realizar una consulta con un doctor se encontrará con el siguiente proceso en la aplicación.

- Primero deberá solicitar una cita con un doctor. La aplicación le pondrá en espera y se le asignará un número, el cuál deberá entregar a su doctor cuando comience la cita.
- Una vez el doctor realice la evaluació se mostrará el diagnostico al paciente y se indicará el precio del tratamiento.
- Finalmente el usuario decidirá si desea continuar con el tratamiento. En el caso que el usuario no pertenezca a ODAFS se podrá optar por utilizar los recursos solidarios de ODAFS.

Las tres fases se muestran en la Figura 49.



(a) Solicitar Doctor (b) Esperar Doctor (c) Pagar Tratamiento

Figura 49: Pacientes aplicación web

5.6. Caja fuerte

En esta sección, también llamada Vault, es la más sencilla del proyecto. Consiste principalmente en tres funciones o funcionalidades: donar fondos a ODAFS, consultar los fondos del contrato y crear una gráfica en forma de donut con los resultados. En la Figura 50 se muestra la página completa.

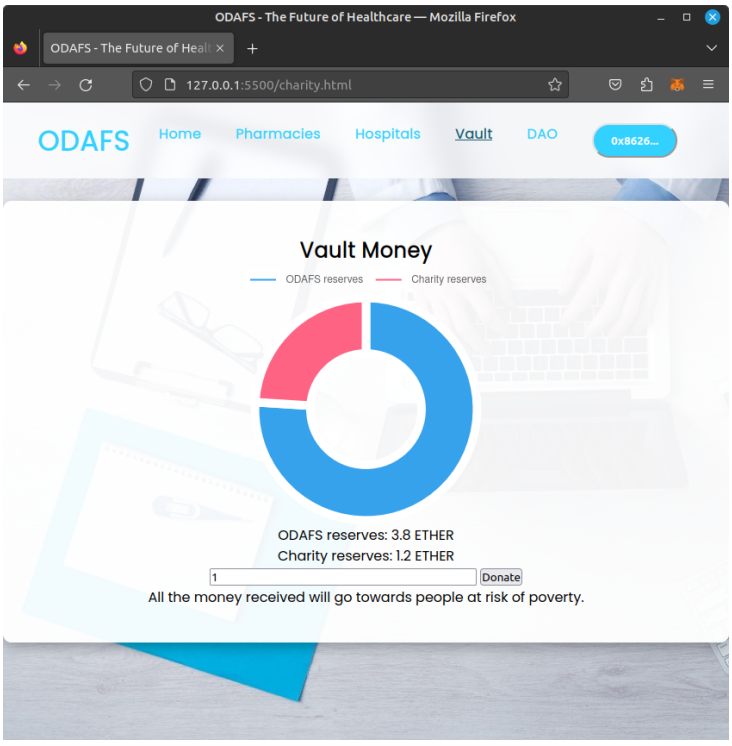


Figura 50: Caja fuerte aplicación web

5.7. DAO

Formado por dos elementos principales, visualizador de propuestas y el panel de votación, es una sección dedicada exclusivamente a la organización y control de ODAFS por parte de sus miembros.

El visualizador de propuestas consiste en una consola en la cuál seleccionando una propuesta existente en el contrato se muestra toda información relacionada con la misma, gráfica de las votaciones, descripción, receptor y cantidad.

En el panel de votación cualquier miembro puede crear o proponer cualquier propuesta, esta función solo podrá ser accedida por miembros de la organización, en cualquier otro caso el contrato lanzará un error. En la Figura 51 se muestra la página completa.

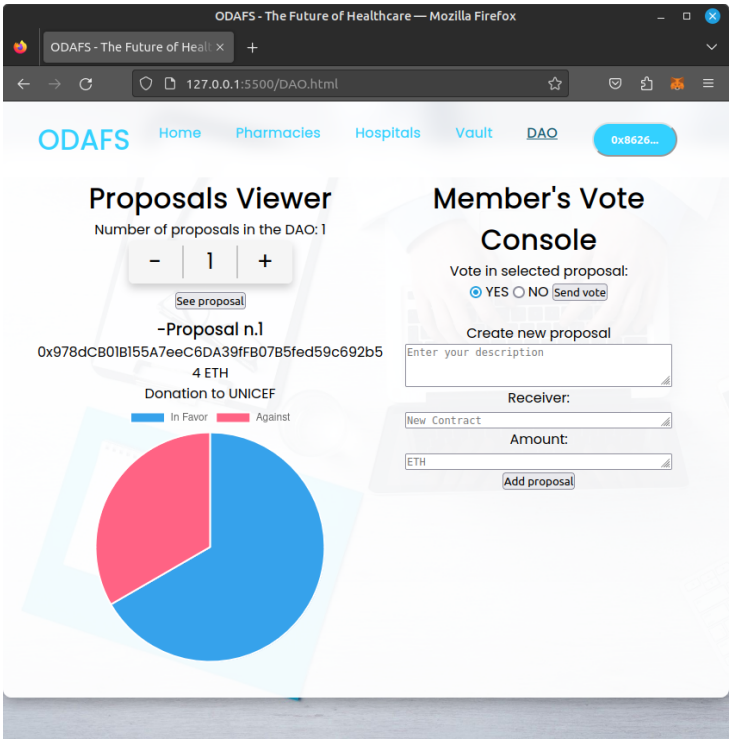


Figura 51: DAO aplicación web

Capítulo 6

Pruebas

En este capítulo, se explica la importancia de las pruebas en un proyecto que involucra contratos inteligentes. A continuación, se muestra la estructura de los tests. Finalmente, se analizarán los resultados de las pruebas realizadas en el proyecto.

6.1. Importancia de las pruebas

ODAFS pretende convertirse en un estándar para la gestión descentralizada de fondos sanitarios para miles de personas. Un aspecto esencial de la organización debe ser la seguridad de los fondos. A diferencia de empresas centralizadas o cualquier aplicación basada en otros paradigmas de programación, los contratos inteligentes son definitivos. Esto significa que cuando un contrato es ejecutado ya no se pueden modificar, arriesgando que los fondos se bloqueen o desaparezcan si hay algún fallo en el desarrollo. Las pruebas no solo deberán comprobar fallos en la ejecución de los contratos y sus funciones, sino también posibles vulnerabilidades que puedan ser explotadas por ciberdelincuentes.

6.2. Estructura

Para la realización de las pruebas, se han desarrollado una serie de test en JavaScript, muy parecidos a los tests unitarios de JUnit en Java. En el proyecto, además de emplear JavaScript, se ha utilizado el framework Hardhat, el cuál simula una red local de Ethereum, y la librería de aserciones Chai[21]. Chai proporciona una serie de funciones para facilitar la escritura de afirmaciones de manera más simple y clara. Las afirmaciones consisten en expresiones que evalúan si determinadas condiciones son ciertas. En nuestro proyecto son empleadas para comparar si los valores producidos por las funciones de los contratos son iguales a los valores esperados, comprobando el correcto funcionamiento del código.

En la implementación de las pruebas, se pueden observar varias funciones que se utilizan de manera repetida y resultan de gran importancia. Estas funciones incluyen:

- “describe”: define bloques de pruebas, llamados suites. Puede contener varias funciones “it” en su interior.
- “it”: define las pruebas individuales dentro del bloque “describe”.

- "expect": se emplea para las comparaciones entre resultados y valores esperados que determinan los resultados de las pruebas.
- "beforeEach": ejecuta el código de su interior antes de todas las pruebas individuales.

A la hora de ejecutar las pruebas de el proyecto es esencial comprobar el correcto funcionamiento de los rangos e interacciones. Para ello, Hardhat permite conectar diferentes cuentas con los contratos inteligentes usando claves privadas, estas cuentas contarán con cantidades de ETHER simuladas. En la Figura 52 se muestra el proceso completo. El primer paso consiste en almacenar dos direcciones de memoria, el contrato y su creador. A continuación, se conectan ambas variables permitiendo la ejecución de funciones por parte de un usuario ficticio.

```
beforeEach(async function () {
    deployer = (await getNamedAccounts()).deployer
    await deployments.fixture(["pharmacy"])
    pharmacy = await ethers.getContract("Pharmacy", deployer)
})
```

```
it("Add stock", async function () {
    await pharmacy.addStock(1,7)
    const products = await pharmacy.productStock(1)
    assert.equal(products.toString(), "7")
})
```

Figura 52: Conexión contrato y usuario

En la Figura 53 se muestra un ejemplo que comprueba el proceso de voto. Utilizando 10 cuentas, se suscriben 10 miembros a ODAFS y se realizan votos en una propuesta. Finalmente, se comprueba si el resultado de la votación es el deseado.

```
it("Proposal Accepted", async function () {
    const accounts = await ethers.getSigners()
    for (i = 0; i < 10; i++) {
        const ballotConnectedContract = await ballot.connect(
            accounts[i]
        )
        await ballotConnectedContract.toMember({ value: ethers.utils.parseEther("4") })
        if (i == 0) {
            await ballotConnectedContract.addProposal()
        }
        if (i < 7) {
            await ballotConnectedContract.voteInLast(true)
        } else {
            await ballotConnectedContract.voteInLast(false)
        }
    }
    const response = await ballot.result()
    assert.equal(response.toString(), "true")
})
```

Figura 53: Test contrato inteligente

6.3. Resultados

Al momento de realizar los tests se pueden emplear varios archivos a la vez, esto ayuda a la estructuración de las pruebas ya que permite una mayor modularización. Por ejemplo, creando un archivo Javascript para las pruebas de cada contrato. El objetivo de los tests consiste en automatizar diferentes pruebas de funcionamiento de los contratos inteligentes, acelerando el proceso de desarrollo y garantizando la seguridad de los contratos. En la Figura 54 se muestran los resultados de todas las pruebas realizadas a las funciones más importantes del contrato de ODAFS, de esta manera se puede comprobar que el contrato estaría listo para desplegar en la red principal de Ethereum.

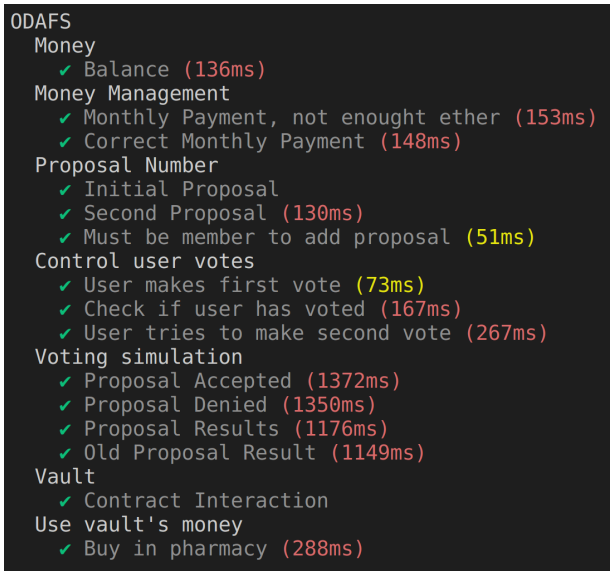


Figura 54: Resultado pruebas

Capítulo 7

Conclusiones

En este último capítulo, se resumen los resultados conseguidos a lo largo del proyecto y se mencionarán una lista de posibles mejoras y futuros trabajos.

7.1. Conclusiones y objetivos cumplidos

Al principio del trabajo se enumeraron los distintos objetivos que debía cumplir el proyecto, se ha conseguido implementar una serie de contratos inteligentes para la creación y funcionamiento de la organización descentralizada ODAFS, además de un prototipo web para interactuar con la aplicación descentralizada.

- Se ha realizado una investigación sobre la tecnología blockchain, analizando su funcionamiento e historia. Se han explicado las distintas generaciones y sus evoluciones. También se ha mostrado un ejemplo de como funciona una red.
- Una cuestión clave ha sido la elección de la red Ethereum. Su elección ha sido basada en su capacidad de ejecutar contratos, blockchain de segunda generación, y su seguridad. El lenguaje empleado para desarrollar los contratos inteligentes ha sido Solidity, el cual es empleado por la mayoría de redes, vital para una posible migración del proyecto a otra red.
- Se ha diseñado un sistema o ecosistema de contratos y usuarios alrededor de ODAFS. Los usuarios deben pagar mensualidades para pertenecer a la organización, obteniendo los privilegios de usar los fondos sanitarios y participar en la toma de decisiones. También se han creado interfaces que deben cumplir los organismos sanitarios, hospitales y farmacias, para interactuar con ODAFS.
- Finalmente, se han implementado todos los contratos y el prototipo web, que ejerce como intermediario entre los contratos y el usuario.

7.2. Futuros trabajos

A lo largo del desarrollo del proyecto, se han detectado una serie de posibles limitaciones o barreras para la adopción de la organización por el público. Se han identificado las siguientes posibles mejoras:

- Privacidad: uno de los principales atractivos de la tecnología blockchain es la transparencia en las transacciones y operaciones realizadas en la red, esto resulta ser una espada de doble filo. En una organización descentralizada como ODAFS, la transparencia es vital para la gestión de fondos, sin embargo puede suponer un problema ya que los votantes podrían sugestionarse en la decisión del voto al ser posible que cualquiera vea su decisión. Existen dos posibles soluciones para este problema. La primera consiste en utilizar otra red, la cuál de forma predeterminada pueda ocultar la información contenida en las transacciones de los usuarios, esta opción resulta la menos aceptable ya que en la actualidad esa privacidad causa una reducción de la seguridad de la red. La segunda, la que se considera la más acertada, consistiría en la encriptación de los votos en las funciones ya implementadas, un efecto secundario sería el aumento del poder computacional necesario para ejecutar las funciones.

- Costes: una de las principales preocupaciones de cualquier organización o empresa es la reducción de costes. Con el crecimiento de la red de Ethereum, los costes y comisiones al realizar transacciones ha ido aumentado de gran manera. Esto podría producir, en épocas de gran demanda de poder computacional o carga de trabajo en la red, que el uso de de ODAFS no sea sostenible por sus comisiones. La mejor alternativa sería migrar el proyecto a una red secundaria de Ethereum con menores comisiones, un ejemplo es Polygon[19]. Las redes secundarias, o Layer 2, son protocolos construidos una capa por encima de una red blockchain para mejorar la escalabilidad y eficiencia de la red, manteniendo su seguridad y capacidades. El proceso de migrar aplicaciones descentralizadas entre redes de distinta capa resulta sencillo debido a que ambas redes son compatibles con Solidity.

- Accesibilidad: las redes blockchain son relativamente modernas, por lo que algunos usuarios podrían encontrar grandes problemas para su uso. La creación de guías, soporte técnico y aplicaciones muy accesibles podría beneficiar a la comunidad de ODAFS en gran medida, aumentando el número de miembros y ayudando a una mayor cantidad de personas.

Otra posible evolución, no relacionada con el desarrollo del software, sería la incorporación de un mayor número de hospitales y farmacias, además de la inclusión de una mayor variedad de servicios cubiertos por ODAFS. Al igual que cualquier seguro u organización, la calidad del servicio entregado por ODAFS depende en gran medida por la calidad de la atención médica. Esto produce que un pilar fundamental de la organización sean los convenios y alianzas con agrupaciones médicas. La inclusión de una mayor variedad de servicios resultará un punto vital en el ciclo de vida de ODAFS. Gracias a la estructuración del proyecto, la inclusión de servicios como fisioterapia y parafarmacia podrán llevarse a cabo de manera sencilla mediante la implementación de nuevas interfaces para las entidades externas. Todas estas decisiones podrán ser llevadas a cabo por los miembros de ODAFS a través de votaciones.

Bibliografía

[1] *Administrador de paquetes npm*. URL: https://www.npmjs.com/ (visitado 08-11-2023).

[2] *Administrador de paquetes Yarn*. URL: https://yarnpkg.com/ (visitado 08-11-2023).

[3] HIPAA Compliance Assistance. «Summary of the hipaa privacy rule». En: *Office for Civil Rights* (2003).

[4] Richard Gendal Brown et al. «Corda: an introduction». En: *R3 CEV, August* 1.15 (2016), pág. 14.

[5] Vitalik Buterin et al. «A next-generation smart contract and decentralized application platform». En: *white paper* 3.37 (2014), págs. 2-1.

[6] Sebastian CLARKE, Iain CRAIG y Michael WYSZYNSKI. «Litecoin Cash: The best of all worlds SHA256 Cryptocurrency». En: *URL: https://litecoinca. sh/downloads/lcc whitepaper. pdf.[Last accessed on 2018 Sep 25]* (2018).

[7] *Control de versiones Github*. URL: https://github.com/ (visitado 08-11-2023).

[8] Saurabh Dhumwad et al. «A peer to peer money transfer using SHA256 and Merkle tree». En: *2017 23RD Annual International Conference in Advanced Computing and Communications (ADCOM)*. IEEE. 2017, págs. 40-43.

[9] *Entorno de ejecución para JavaScript Node.js*. URL: https://nodejs.org/en (visitado 08-11-2023).

[10] William Entriken et al. «Eip-721: Erc-721 non-fungible token standard». En: *Ethereum Improvement Proposals* 721 (2018).

[11] *Ethereum Improvement Proposals - ERC*. URL: https://eips.ethereum.org/erc.

[12] *Extensión Metamask*. URL: https://metamask.io/ (visitado 08-11-2023).

[13] *Framework Hardhat*. URL: https://hardhat.org/ (visitado 08-11-2023).

[14] César Rangel García. «Las Políticas de gobernanza, participación y descentralización en el éxito de la gestión local y su importancia para construir nuevos horizontes a la humanidad». En: *Prospectivas UTCRevista de Ciencias Administrativas y Económicas"* 1.2 (2018), págs. 16-28.

[15] Stuart Haber y W Scott Stornetta. *How to time-stamp a digital document*. Springer, 1991.

[16] *HealthVerity Project*. URL: https://healthverity.com/ (visitado 06-11-2023).

[17] Roger Heines et al. «The Tokenization of Everything: Towards a Framework for Understanding the Potentials of Tokenized Assets.» En: *PACIS*. 2021, pág. 40.

[18] *IDE Virtual Study Code*. URL: https://code.visualstudio.com/ (visitado 08-11-2023).

[19] Jaynti Kanani, Sandeep Nailwal y Anurag Arjun. «Matic whitepaper». En: *Polygon, Bengaluru, India, Tech. Rep., Sep* (2021).

[20] Ashley Lannquist. «Introducing Blockchain Impact Award Winner Chronicled». En: *Newsweek* 172.7 (2019).

[21] *Librería de aserciones Chai*. URL: https://www.chaijs.com/.

[22] Chrissa McFarlane et al. «Patientory: A healthcare peer-to-peer EMR storage network v1». En: *Entrust Inc.: Addison, TX, USA* 3 (2017), pág. 19.

[23] Gianmaria Del Monte, Diego Pennino y Maurizio Pizzonia. «Scaling blockchains without giving up decentralization and security: A solution to the blockchain scalability trilemma». En: *Proceedings of the 3rd Workshop on Cryptocurrencies and Blockchains for Distributed Systems*. 2020, págs. 71-76.

[24] Satoshi Nakamoto. «Bitcoin whitepaper». En: *URL: https://bitcoin. org/bitcoin. pdf-(: 17.07. 2019)* (2008).

[25] Vicenç Navarro. «El modelo sanitario liberal: EEUU». En: *Revista digital SISTEMA* 22 (), págs. 1-7.

[26] Vicente Navarro. «Luces y sombras de la reforma sanitaria de Obama». En: *Revista Digital Sistema* (2010).

[27] Shivansh Pandey et al. «Crowdfunding fraud prevention using blockchain». En: *2019 6th International Conference on Computing for Sustainable Global Development (INDIACom)*. IEEE. 2019, págs. 1028-1034.

[28] *Solidity Programming Language*. URL: https://soliditylang.org/ (visitado 08-11-2023).

[29] T Takenobu. «Ethereum EVM illustrated». En: *Github Pages* (2018).

[30] Fabian Vogelsteller y Vitalik Buterin. «Eip 20: Erc-20 token standard». En: *Ethereum Improvement Proposals* 20 (2015).

[31] Martin Von Haller Gronbaek. «Blockchain 2.0, smart contracts and challenges». En: *Comput. Law, SCL Mag* 1 (2016), págs. 1-5.

[32] *Vyper Programming Language*. URL: https://docs.vyperlang.org/en/stable/ (visitado 08-11-2023).

[33] Moritz Wendl, My Hanh Doan y Remmer Sassen. «The environmental impact of cryptocurrencies using proof of work and proof of stake consensus algorithms: A systematic review». En: *Journal of Environmental Management* 326 (2023), pág. 116530.