

Sistema de gestión descentralizado de la financiación de proyectos



Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:

Elvi Mihai Sabau Sabau

Tutor/es:

Mora Mora, Higinio

Adsuar Abaldea, Victor

Mayo 2023

Sistema de gestión descentralizado de la financiación de proyectos

Autor

Elvi Mihai Sabau Sabau

Tutor/es

Mora Mora, Higinio

B148 Dpto. Tecnología Informática y Computación

Adsuar Abaldea, Victor

B148 Dpto. Tecnología Informática y Computación



Grado en Ingeniería Informática



Escuela
Politécnica
Superior



Universitat d'Alacant
Universidad de Alicante

Alicante, Mayo 2023

Preámbulo

Hoy en día existen muchas plataformas y organizaciones orientadas a la gestión de la subvención de proyectos, como por ejemplo las plataformas de crowdfunding. Durante los últimos años estas plataformas han conseguido una gran popularidad debido a que gracias a estas plataformas se han podido poner en marcha tanto proyectos como pequeñas startups, pero también ha atraído la atención de otros actores fraudulentos, provocando así un surgimiento cada vez mayor de estafas.

Las estafas, en concreto en las plataformas de crowdfunding abarcan muchos campos, y estas se dan cuando una campaña de crowdfunding solicita y acepta donaciones a través de falsos pretextos, engañando a las personas sobre el resultado, la naturaleza del proyecto y/o la causa solicitada.

También se dan casos de malversación de fondos, o el uso del proyecto para el blanqueo de capital, ya sea modificando el valor de las donaciones recibidas, o mediante una idea ilusoria de responsabilidad ecológica (greenwashing).

En este trabajo se investiga, desarrolla y propone una solución a este tipo de problemas mediante un sistema de gestión descentralizado de la financiación de proyectos.

Agradecimientos

Me gustaría agradecer a Higinio Mora, Marcelo Saval y Victor Adsuar, quiénes fueron mis tutores durante este último año de carrera. Ellos me enseñaron lo que se podría decir "la parte más excitante" de la carrera, abriéndome camino a un conjunto de tecnologías que no conocía, y así guiándome durante esta etapa final de mi carrera.

Agradezco a Higinio por su apoyo y tutoría durante este trabajo, además de haberme brindado la oportunidad de aprender más en detalle la tecnología blockchain. Agradezco tambien a Marcelo por su ayuda y excelentes clases en IMCR, donde no solo aprendimos a realizar un proyecto desde cero, sino también formalizarlo y enseñarnos a como redactar un documento académico.

También me gustaría agradecer a: Francisco Antonio Ferrandez, Jose Maria Oncina, Fidel Aznar Gregori, Jose Joaquin Rodes y Miguel Angel Albeza. Puedo decir por experiencia que estos profesores son expertos en su campo, además la pasión que poseen por sus respectivas asignaturas y los temas que imparten facilita muchísimo la compresión de estas, además son unos profesores dedicados y muy retroactivos que mejoran la asignatura y los recursos gracias al feedback de los alumnos, y no les importa quedarse horas de más para explicar conceptos que a lo mejor durante la clase no se han entendido.

Por ultimo y más importante me gustaría agradecer a mis padres Livia Elena Sabau y Emil Ioan Sabau por sus colosales esfuerzos y sacrificios, y a mi abuela Dodu Aurica que de pequeño me llevaba siempre a la escuela. Ellos me apoyaron, inspiraron y me dieron la oportunidad no solo de estudiar y trabajar en un campo que me gusta, sino de poder realizar mis estudios, ya que mi carrera académica procede desde lo más bajo, y gracias a ellos logré salir a flote, por ello reconozco sus esfuerzos realizados y les agradezco todo lo que han hecho por mi.

A ellos les dedico este trabajo, ya que sin ellos no estaría aquí.

A mis amigos y familia, que me apoyaron durante este largo y turbulento trayecto.

Love with your heart, use your head for everything else.
- Alan Melikdjanian.



Índice de Contenidos

1. Introducción	21
1.1. Motivación	21
1.2. Sobre este trabajo	21
1.3. Herramientas empleadas para la elaboración de este documento	22
1.4. Objetivos	22
1.4.1. Objetivos Específicos	22
1.5. Solución Propuesta	23
1.6. Abstracción del problema abordado	25
1.7. Metodología	26
1.7.1. Metodología Tecnológica	26
1.7.2. Metodología de Desarrollo	26
2. Estado del Arte	27
2.1. Plataformas de Financiación Actuales	27
2.1.1. Kickstarter	27
2.1.2. Indiegogo	28
2.1.3. CrowdFunder	28
2.1.4. GitCoin	29
2.1.5. CoinStarter	30
2.1.6. Tecra Space	31
2.2. Conclusiones respecto al estado del arte	33
3. Contexto Tecnológico	34
3.1. Las redes Blockchain	34
3.1.1. Inmutabilidad y Trazabilidad	35
3.1.2. Consenso y tipos de consensos	36
3.1.2.1. Proof-of-Work (PoW)	36
3.1.2.2. Proof-of-Stake (PoS)	36
3.1.2.3. Delegated Proof-of-Stake (DPoS)	37
3.1.2.4. Practical Byzantine Fault Tolerance (PBFT)	38
3.1.3. Nodos y tipos de Nodos	39
3.1.3.1. Nodo Validador	39
3.1.3.2. Nodo Completo	39
3.1.3.3. Nodo Parcial	39
3.1.3.4. Nodo Archivo	39
3.1.3.5. Nodo de Autoridad	40
3.1.4. Cuentas y Carteras	40
3.1.5. Accesos y tipos de acceso	41
3.1.5.1. Pública	41
3.1.5.2. Privada	41
3.1.5.3. Híbrida	41
3.1.5.4. Consorcio	41
3.1.6. Blockchains monolíticas y modulares	42
3.1.6.1. Criptomonedas	43
3.1.7. Smart Contracts - Chain Code	43



3.2.	Ethereum	44
3.2.1.	El protocolo Ethereum	44
3.2.2.	Ethereum Virtual Machine	44
3.2.3.	Funcionamiento de la EVM	45
3.2.4.	Ejecución de la EVM	46
3.2.5.	Gas	46
3.2.6.	Redes	47
3.2.7.	Sobre Polygon	48
3.2.8.	Cuentas Externas y Cuentas de Contratos	48
3.2.8.1.	Externas	48
3.2.8.2.	Contratos	48
3.2.9.	Transacciones en Ethereum	49
3.2.10.	Solidity: Lenguaje Orientado a Smart Contracts	49
3.2.11.	Comportamiento de un Smart Contract en Ethereum	49
3.2.12.	Estructura de un smart contract con Solidity	50
3.2.13.	Estandards	66
3.2.14.	OpenZeppelin	66
3.2.14.1.	ERC-20 - Tokens	67
3.2.15.	ABIs	67
3.2.16.	Dapps	67
3.2.17.	Sistemas	67
3.2.17.1.	ICO	67
3.2.17.2.	DAO	68
3.2.17.3.	DEX	68
3.3.	IPFS	69
3.4.	EtherJS	70
3.5.	Svelte	70
3.6.	HardHat	71
3.7.	ThirdWeb	72
3.8.	Bootstrap	72
3.9.	MetaMask	73
3.10.	Docker	74
4.	Modelo del Sistema	75
4.1.	Agentes	75
4.1.1.	Proponente	76
4.1.2.	Votantes	76
4.1.3.	Inversor	76
4.1.4.	Perito	77
4.2.	Requerimientos	78
4.2.1.	Requerimientos Funcionales	78
4.2.2.	Requerimientos no Funcionales	79
4.2.3.	Requisitos excluidos	80
4.3.	Gobernanza	80
4.4.	Diseños	81
4.4.1.	Diagrama de Flujo	81
4.4.1.1.	Proceso de contribución en un proyecto	81
4.4.1.2.	Proceso de propuesta de un nuevo proyecto	82



4.4.2.	Mockups	83
4.4.2.1.	Pantalla de Registro	83
4.4.2.2.	Pantalla de notificación de red equivocada	84
4.4.2.3.	Pantalla de notificación de cartera inexistente	85
4.4.2.4.	Pantalla de la lista de Proyectos	86
4.4.2.5.	Pantalla del formulario para proponer un proyecto	87
4.4.2.6.	Pantalla de detalles de un proyecto	88
4.4.3.	Modelo del smart contract	89
4.4.3.1.	Project	90
4.4.3.2.	Milestone	90
4.4.3.3.	Comment	90
4.4.3.4.	Investment	90
4.4.3.5.	Media	90
4.4.3.6.	Document	90
4.4.4.	Arquitectura del smart contract	91
4.4.4.1.	ProjectState	91
4.4.4.2.	Comment	91
4.4.4.3.	Investment	91
4.4.4.4.	Project	91
4.4.4.5.	getProjectsCount	93
4.4.4.6.	proposeProject	93
4.4.4.7.	getProject	93
4.4.4.8.	getCommentsCount	94
4.4.4.9.	addComment	94
4.4.4.10.	getComment	94
4.4.4.11.	getComments	94
4.4.4.12.	voteProject	94
4.4.4.13.	hasVoted	95
4.4.4.14.	checkProjectApproval	95
4.4.4.15.	investInProject	95
4.4.4.16.	checkProjectFunding	95
4.4.4.17.	withdrawInvestment	95
5.	Implementación y Evaluación de la prueba de concepto	96
5.1.	Implementaciones	96
5.1.1.	Front-End (Interfaz Web)	96
5.1.1.1.	Pantalla de Registro	97
5.1.1.2.	Pantalla de notificación de red equivocada	98
5.1.1.3.	Pantalla de notificación de cartera inexistente	99
5.1.1.4.	Pantalla de la lista de Proyectos	100
5.1.1.5.	Pantalla del formulario para proponer un proyecto	101
5.1.1.6.	Pantalla de detalles de un proyecto	102
5.1.2.	Back-End (DAO)	103
5.1.2.1.	Implementación del Smart Contract	103
5.2.	Despliegue del sistema	114
5.2.1.	Código fuente del proyecto	114
5.2.2.	Despliegue de la interfaz web	114
5.2.2.1.	Sobre la metodología del despliegue	114



5.2.2.2. Encapsulando la aplicación web	114
5.2.3. Despliegue del Smart Contract	117
5.2.3.1. Verificación del contrato	118
5.3. Evaluación de la prueba de concepto	119
6. Aspectos Éticos, Económicos y Sociales	120
6.1. Aspectos Éticos	120
6.2. Aspectos Económicos	120
6.3. Aspectos Sociales	120
7. Presupuesto	121
7.1. Infraestructura Web	121
7.1.1. Dominio	121
7.1.2. Alojamiento Web	122
7.2. Smart Contract	123
7.2.1. Despliegue	123
7.3. Tabla de totales	124
8. Conclusiones	125
8.1. Conclusión sobre el trabajo	125
8.2. Beneficios personales	125
8.3. Sobre la tecnología	125
8.4. Trabajo a futuro	126



Índice de Figuras

1.	Representación del proceso de inversión	24
2.	Comparativa entre la arquitectura común Cliente-Servidor-DB y Cliente-Smart Contract.	26
3.	Estructura de una Blockchain.	34
4.	Funcionamiento de una Blockchain - Business Insider.	35
5.	Ventajas y desventajas de los tipos de blockchain basado en su acceso - Tech Target Media.	41
6.	Diagrama representativo de las blockchains monolíticas y modules - Médium.	42
7.	Representación de las layers del 0 al 2 de una blockchain.	42
8.	Comparación entre web2 y web3 del proceso de interacción con aplicaciones.	43
9.	Diagrama que representa la EVM - Ethereum.org	45
10.	Diagrama que representa el reembolso del gas - Ethereum.org	47
11.	Panel de control del Cliente IPFS.	69
12.	Console del cliente de HardHat.	71
13.	Logotipo y slogan de Bootstrap.	72
14.	Captura de la cartera de MetaMask.	73
15.	Captura del panel de control de Portainer.	74
16.	Diagrama de Agentes - Interacción de los agentes en el sistema a lo largo de un proyecto.	75
17.	Diagrama de Agentes - Interacción de un proponente con un proyecto.	76
18.	Diagrama de Agentes - Interacción de un votante con un proyecto.	76
19.	Diagrama de Agentes - Interacción de un inversor con un proyecto.	76
20.	Diagrama de Agentes - Interacción de un perito con un proyecto.	77
21.	Diagrama de Flujo - Proceso de contribución con un proyecto.	81
22.	Diagrama de Flujo - Proceso de propuesta de proyecto.	82
23.	Mockup - Pantalla de Registro.	83
24.	Mockup - Notificación de red equivocada.	84
25.	Mockup - Notificación de que la cartera no se ha detectado.	85
26.	Mockup - Lista de Proyectos.	86
27.	Mockup - Formulario para proponer un proyecto.	87
28.	Mockup - Detalles de un proyecto.	88
29.	Modelo - Estructura de los datos del smart contract.	89
30.	Interfaz Web - Pantalla de Registro.	97
31.	Interfaz Web - Notificación de red equivocada.	98
32.	Interfaz Web - Notificación de cartera inexistente.	99
33.	Interfaz Web - Lista de Proyectos.	100
34.	Interfaz Web - Formulario para proponer un proyecto.	101
35.	Interfaz Web - Detalles de un proyecto.	102
36.	Dockerfile - Dockerfile del Front-End.	115
37.	Portainer - Interfaz de Portainer, gestor de contenedores de Docker.	116
38.	Front-End - Muestra de la aplicación desplegada.	116
39.	Verificación del contrato - Captura de la terminal, HardHat notificando de la correcta validación del contrato en la red Mumbai.	117
40.	Polyscan - Captura del contrato desplegado y verificado en la red de Mumbai.	118
41.	Presupuesto - Coste anual del dominio.	121
42.	Presupuesto - Coste mensual del VPS.	122
43.	Presupuesto - Coste del despliegue del contrato en Euro.	123



Índice de Tablas

1.	Estado del Arte - Tabla comparativa de plataformas de crowdfunding	32
2.	Análisis y Diseño - Requerimientos funcionales del sistema.	78
3.	Análisis y Diseño - Requerimientos no funcionales del sistema.	79
4.	Análisis y Diseño - Requerimientos no implementados en la prueba de concepto. . .	80
5.	Presupuesto - Tabla de totales.	124



Índice de Códigos

1.	Ejemplo de un Smart Contract en Solidity.	50
2.	Ejemplo de los diferentes espacios para alojar variables en Solidity.	51
3.	Ejemplo de los diferentes tipos de variables en Solidity.	52
4.	Ejemplo de variables inmutables en Solidity.	53
5.	Ejemplo de pragma y licencia en los archivos .sol.	53
6.	Ejemplo de un constructor y herencias en Solidity.	54
7.	Ejemplo de visibilidades en funciones en Solidity.	55
8.	Ejemplo de visibilidades en variables en Solidity.	55
9.	Ejemplo de mutabilidades en Solidity.	56
10.	Ejemplos de returns en Solidity.	57
11.	Ejemplo de un contrato usando Payable en Solidity.	58
12.	Ejemplo de sentencias require en Solidity.	59
13.	Ejemplo del uso de modifiers en Solidity.	60
14.	Ejemplo del uso de modifiers con parámetros en Solidity.	60
15.	Ejemplo del uso de funciones por defecto en Solidity.	61
16.	Ejemplo de transferencia de Ether en Solidity.	62
17.	Ejemplo de llamada externa usando el código del contrato destinatario en Solidity.	63
18.	Ejemplo de llamada externa usando el identificador de la función destinataria en Solidity.	64
19.	Ejemplo del uso y definición de eventos en Solidity.	65
20.	Ejemplo del uso de la librería SafeMath en Solidity.	66
21.	Ejemplo de conexión a una cartera Metamask usando Ethers.	70
22.	Hello World en Svelte.	70
23.	Ejemplo de la conexión y subida de un archivo a IPFS.	72
24.	Smart Contract de la prueba de concepto.	103



1. Introducción

1.1. Motivación

La motivación de este trabajo viene de la pasión por el aprendizaje de la tecnología blockchain, una tecnología emergente y disruptiva, capaz de proveer de características novedosas a Internet.

En concreto, todo empezó en el verano del 2020, cuando BitCoin hizo su boom económico. El entusiasmo provocado por este evento hizo que muchos llegarán a conocer esta tecnología que hasta ahora no tenía tanta fama, entre ellos yo (Elvi). Tuve curiosidad con respecto a la tecnología blockchain, concretamente el protocolo Ethereum, y dediqué el verano a estudiar sus utilidades y la metodología de programación empleada para crear aplicaciones sobre estos protocolos.

Así, durante el verano aprendí a programar usando estos protocolos, llegué a desarrollar pequeños prototipos y criptomonedas, pero nada realmente digno de mencionar. Un año más tarde, a principios del 2022, Higinio Mora que en ese momento fue mi profesor de la asignatura de Sistemas Embedidos publicó varias convocatorias de prácticas extracurriculares en la Universidad, curiosamente relacionadas con la investigación de estas nuevas tecnologías.

Gracias a estas prácticas, tuve la oportunidad de poner en uso lo aprendido por mi cuenta, además de instruirme en otros protocolos blockchain como Hyperledger Fabric y IoTa. Durante estas prácticas desarrolle varios prototipos en la red Polygon, tuve el impulso para cursar y graduarme en la plataforma de LW3¹ así formalizando lo aprendido durante todo este tiempo, y además contribuyendo a la publicación de varios artículos[17][24] en el congreso de RiiForum 2022.

Gracias a estas experiencias, y a la pasión por nuevas tecnologías, he decidido enfocar mi trabajo fin de grado a la investigación de un sistema de gestión descentralizado de la financiación de proyectos basado en Blockchain, ya que esta tecnología es adecuada para la gestión de activos de forma electrónica.

1.2. Sobre este trabajo

Para solventar el problema descrito en el preámbulo, el objetivo de este trabajo y solución se que propone en este documento es el uso de la tecnología blockchain para proveer de trazabilidad, confianza y inmutabilidad a los activos y eventos que se realizan durante el proceso de desarrollo de dichos proyectos.

Para ello, en este trabajo se estudiará las tecnologías necesarias para desarrollar un sistema descentralizado, y la integración de esta con una blockchain, además de detallar las tecnologías necesarias para desplegar y desarrollar un sistema capaz de gestionar un activo en la blockchain, y el desarrollo de la propia.

¹learnweb3.io



1.3. Herramientas empleadas para la elaboración de este documento

Para la realización de esta memoria se han utilizado las siguientes herramientas:

- **Tex**²: Sistema software para la preparación de documentos.
- **Draw.io**³: Herramienta web para la creación y diseño de diagramas y dibujos.
- **Overleaf**⁴: Herramienta web para el desarrollo cómodo de documentos en LaTeX.

1.4. Objetivos

El objetivo de este trabajo es el desarrollo conceptual y el de una prueba de concepto de un sistema descentralizado para la gestión de la financiación de proyectos, similar al funcionamiento de las plataformas crowdfunding^[8] actuales, pero solventando varios de los mayores problemas a los que se enfrenta este tipo de plataformas, la detección de proyectos fraudulentos y la aplicación de contra-medidas.

1.4.1. Objetivos Específicos

Para determinar mejor el objetivo del trabajo, este se ha dividido en los siguientes puntos:

- Estado del arte de las plataformas de financiamiento colectivo de proyectos centralizadas y descentralizadas: comparativa entre ambas, ventajas y desventajas de cada tipo, desafíos a los que se enfrentan estas plataformas y reflexión acerca de la aplicación de la tecnología Blockchain en este ámbito.
- Análisis y Investigación de las tecnologías blockchain para aprovechar la trazabilidad y inmutabilidad que aporta esta tecnología. En concreto analizar el Protocolo Ethereum sobre como funciona en detalle, y como se puede desarrollar una aplicación usando esta blockchain.
- Diseñar un sistema prototipo reducido de una Organización Autónoma Descentralizada en la Blockchain que implemente las funcionalidades necesarias orientadas a una solución a los problemas actuales los cuales afrontan estas plataformas, y que servirá como Back-End⁵.
- Desarrollo de un Front-End⁶ que permita interactuar con la DAO a través de una página web, usando tecnologías de vanguardia basadas en JavaScript.

²<https://en.wikipedia.org/wiki/TeX>

³<https://app.diagrams.net/>

⁴<https://es.overleaf.com>

⁵Back-End: Sistema software que soporta la funcionalidad de una aplicación y no es visible para el usuario final.

⁶Front-End: Interfaz de usuario basada en una aplicación web, se encarga de la interacción con el Back-End y la presentación visual de los datos.



1.5. Solución Propuesta

La solución que se propone (como se ha mencionado anteriormente), es el desarrollo de un sistema de gestión descentralizado de la financiación de proyectos, usando los beneficios de la tecnología blockchain.

La plataforma a desarrollar se regirá mediante un mecanismo de gestión autónomo para evitar que un administrador u otra entidad pueda malversar los fondos de la plataforma.

La comunidad decidirá qué proyectos se aprobarán y cuáles no. Los proyectos pasarán por un proceso de votación antes de ser aceptados donde cada usuario podrá votar una única vez en un proyecto, para verificar la integridad de la idea, la veracidad, la viabilidad, y para detectar cualquier incongruencia en la descripción, los hitos propuestos y los fondos requeridos para su desarrollo.

En concreto, la propuesta de proyecto, tendrá (entre otros), título, descripción, naturaleza / tipo, cantidad de fondos necesarios para su formalización, tiempo total para su desarrollo, lista de hitos, y por cada hito: título, descripción y % del tiempo total del desarrollo asignado para dicho hito.

Esta fase tendrá una duración límite de 30 días desde la propuesta del proyecto, y los votos positivos necesarios para aprobar la propuesta deben ser igual o superior al 85 % de los votos presentados, en caso de no alcanzar dicho umbral, se denegará la propuesta.

Se permitirá a los usuarios comentar los proyectos propuestos, lo que fomentará la interacción en la comunidad al compartir ideas, debatir sobre cada proyecto y comunicarse con el proponente. De este modo, se mantendrá una dinámica activa y se compartirá el progreso de cada proyecto.

Una vez aprobado por la comunidad, empieza la fase de financiación. Los usuarios pueden invertir en el proyecto durante un periodo de 30 días y una vez se recauden los fondos necesarios empezará la fase de desarrollo, dónde los fondos se desbloquearán de forma secuencial conforme se vayan cumpliendo y validando los hitos del desarrollo. En caso de no alcanzar la cuota necesaria definida en la propuesta en estos 30 días, la propuesta se denegará, y los fondos recaudados se reembolsarán a cada inversor.

El proponente podrá solicitar la revisión de cada hito para su validación. Los hitos serán validados por la comunidad, en concreto usuarios “peritos” con cierta especialización en la naturaleza del proyecto. Esta especialización será validada mediante un modelo de verificación de credenciales[31]. Estos se encargarán de validar que los hitos se han alcanzado correctamente.

El hito se dará por cumplido cuando por votación, la mayoría absoluta de los votos recibidos por los peritos aprueban el hito, en caso contrario, el perito discrepante deberá adjuntar el motivo de su negación. Si el hito no se cumple, se le proporcionará al proponente el motivo de la denegación (motivos de los votos negativos), y un tiempo limitante de 10 días para poder solventar el hito, al cabo de este tiempo el proponente podrá nuevamente volver a solicitar la validación del hito.

En un último acontecimiento en el cual el proyecto muestra claras señales de estancamiento, como por ejemplo: haber excedido el plazo para la solicitud de revisión del avance de los hitos en un plazo estimable al doble del tiempo asociado a cada hito, la propuesta del proyecto se cancelará

automáticamente, y el resto de los fondos asignados al los hitos sin alcanzar se reembolsará a los inversores.

Para cumplir con esta solución, es esencial garantizar la trazabilidad y el control de los fondos y activos de la plataforma. Además, para evitar que el proponente o un miembro malintencionado de la comunidad pueda eliminar o cambiar un proyecto durante su proceso los datos del proyecto deben ser inmutables.

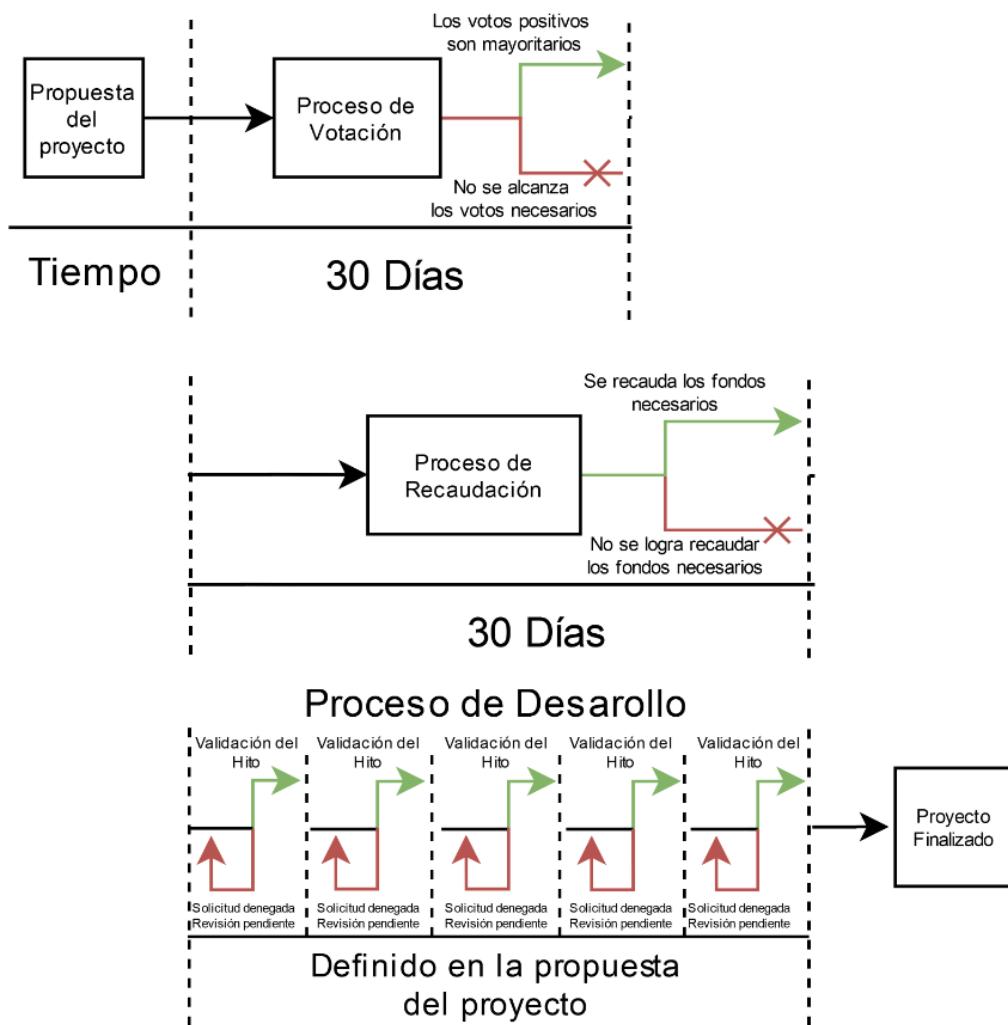


Figura 1: Representación del proceso de inversión.

El uso de una blockchain para este trabajo es una aportación perfecta para desarrollar el trabajo, ya que a través de la trazabilidad que nos proporciona podremos ver en que se gastan los activos y además tener un control sobre estos, y un transparencia total con los usuarios.

De esta forma pudiendo así implementar una solución la cual desbloqueará parte de lo recaudado a medida que se van completando ciertos hitos asociados al desarrollo del proyecto.



1.6. Abstracción del problema aborado

La solución propuesta, un sistema de gestión descentralizado de la financiación de proyectos basado en la tecnología blockchain, puede ser aplicable a una serie de contextos más allá de las plataformas de crowdfunding.

Estos pueden incluir la gestión de proyectos en organizaciones, la trazabilidad de la financiación de proyectos en instituciones financieras y la verificación del cumplimiento de las regulaciones en el transporte de mercancías, etc.

El conjunto de problemas presentado puede categorizarse como "Gestión y trazabilidad de proyectos financiados colectivamente". Dichos problemas son intrínsecos a cualquier plataforma o sistema que maneje la financiación y el desarrollo de proyectos. Los problemas pueden subdividirse en los siguientes aspectos clave:

- **Autenticidad y Veracidad:** Corresponde a las dificultades para verificar la autenticidad de un proyecto y la honestidad de los creadores del mismo. Incluye problemas como fraudes, estafas y malversación de fondos, donde los creadores del proyecto pueden engañar a los inversores acerca de la naturaleza, el resultado y/o la causa solicitada del proyecto.
- **Gestión de la Financiación:** Se refiere a la complejidad de rastrear y manejar adecuadamente los fondos recaudados. Entra en juego cuando se enfrentan problemas de malversación de fondos, lavado de dinero y falta de transparencia en la distribución y uso de los fondos recaudados.
- **Validación del Progreso:** Se refiere a las dificultades para verificar el cumplimiento de los hitos propuestos y para validar el progreso del proyecto. Los creadores del proyecto pueden engañar a los inversores y otros interesados acerca del progreso real y los logros del proyecto.
- **Participación y Decisión:** Se refiere a las dificultades para permitir y manejar la participación de la comunidad en la toma de decisiones, la validación del proyecto y la revisión de los hitos. Situaciones donde la comunidad se siente excluida del proceso de toma de decisiones y la falta de transparencia y control pueden provocar una disminución de la confianza y la participación.

1.7. Metodología

1.7.1. Metodología Tecnológica

La metodología usada para el desarrollo se ha basado en la investigación de las soluciones descentralizadas ya que el objetivo es crear una infraestructura que promoviera la descentralización y la trazabilidad de los activos.

También se destaca la Stack tecnológica,⁷ ya que esta varía drásticamente de la común, esto se debe a que los Smart Contracts eliminan la necesidad de tener una arquitectura Back-end clásica, mientras que la blockchain con la que interactúan reemplaza el uso de una base de datos.

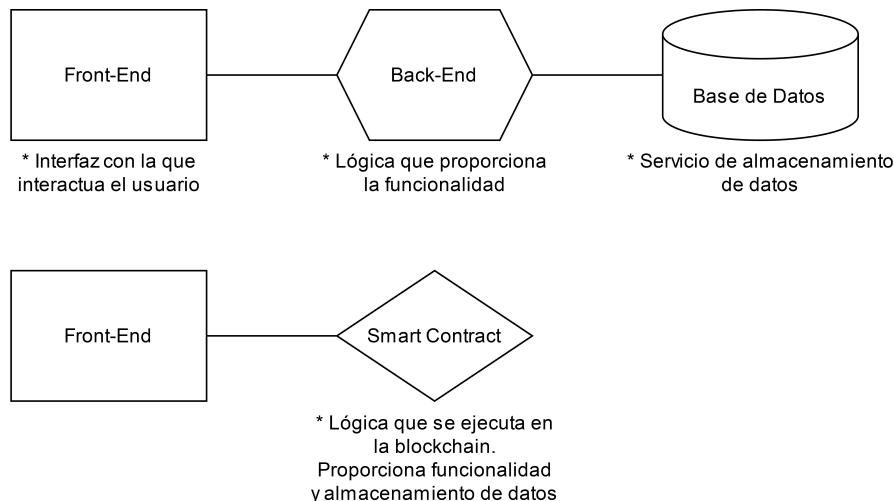


Figura 2: Comparativa entre la arquitectura común Cliente-Servidor-DB y Cliente-Smart Contract.

Para el desarrollo de este sistema, se hace uso de Hardhat, un framework que permite el despliegue de una red blockchain local simulada usando el protocolo Ethereum para el desarrollo, testeo y despliegue de los smart contracts a la red de Polygon, Svelte para el desarrollo ágil de un front-end usando JavaScript, Bootstrap para dotar de un estilo user-friendly⁸ al front-end, la librería EthersJS para interconectar el front-end con los smart contracts, y la librería ThirdWeb para interconectarnos con la blockchain IPFS para el almacenamiento de imágenes, vídeos y otros archivos.

1.7.2. Metodología de Desarrollo

Durante la implementación se ha usado la metodología ágil para planificar la implementación de la aplicación por partes, pero con una gran flexibilidad ya que el equipo solo está formado por una persona.

Se usa CI mediante **Github Actions**⁹ para poder realizar los tests en el Smart Contract (mediante MochaJS) y desplegar de forma automática mediante DockerHub la interfaz web.

⁷Conjunto de herramientas y arquitecturas usadas para el desarrollo de software.

⁸Diseño o sistema que es fácil de usar y entender por los usuarios, lo que facilita su interacción y mejora su experiencia.

⁹<https://github.com/features/actions>



2. Estado del Arte

En este apartado se expone una investigación las plataformas de crowdfunding actuales donde se destaca que métodos se aplican para evitar proyectos fraudulentos, y plantear una solución para mejorar el sistema actual que usan dichas plataformas.

También se explica en detalle que es una Blockchain y como funciona, además entrar en profundidad en el funcionamiento del Protocolo Ethereum y otras tecnologías que se usarán para desarrollar el trabajo.

2.1. Plataformas de Financiación Actuales

A continuación se describe un ejemplo representativo de las plataformas de crowdfunding y gestión de fondos, tanto en web2¹⁰ como en web3¹¹.

2.1.1. Kickstarter

Kickstarter[10] es una corporación de beneficio público estadounidense con sede en Brooklyn, Nueva York, que mantiene una plataforma global de financiación colectiva centrada en la creatividad. La misión declarada de la empresa es “ayudar a dar vida a proyectos creativos”.

Kickstarter lleva desde el 2013 recaudando fondos para proyectos de todo tipo, y por ello una de las plataformas que más experiencia tienen a la hora de enfrentarse a fraudes y vulnerabilidades similares.

La plataforma de Kickstarter funciona de la siguiente forma, cuando un proyecto se publica, kickstarter verifica la identidad del proponente del proyecto, además del fondo mínimo definido y los hitos propuestos para el proyectos. Una vez alcanzado el fondo mínimo definido, todos los fondos se traspasan al proponente, teniendo este total control sobre los fondos, y lo mismo ocurre para los hitos.

En este sistema, se deposita confianza en el recaudador, lo cual representa una debilidad, ya que si el proyecto resulta fraudulento o no se completa el recaudador es responsable de reembolsar lo recaudado a los inversores, aunque no exista una obligación legal. Para crear conciencia e informar a los inversores al respecto, Kickstarter publicó un artículo[20] detallando el proceso de validación y verificación de los proyectos permitidos en su plataforma web. Sin embargo, se insinúa que la responsabilidad final de los fondos recae en el inversor y en su juicio personal sobre el proyecto en el que decida invertir.

Este articulo, y además el ToS de la plataforma confirma que la única forma de recibir un reembolso después de que un proyecto haya recaudado el mínimo acordado es mediante una solicitud formal al propio proponente.

Kickstarter no se hace responsable de los fondos recaudados en caso de que el proyecto no pueda desarrollarse, o si algún hito no se pueda llegar a alcanzar. Esto incluye situaciones como por ejemplo que el proponente deje de comunicarse con sus inversores a través de la plataforma entre otros.

¹⁰Terminología que referencia la arquitectura tecnológica web centralizada, más común conocida como arquitectura cliente - servidor.

¹¹Terminología que referencia la arquitectura tecnológica web descentralizada, también conocidas como dapps: aplicaciones descentralizadas desplegadas en una blockchain.



Esto permite que agentes maliciosos puedan proponer proyectos que mediante una campaña publicitaria logre convencer a los inversores de que el proyecto en el que están invirtiendo es factible, y al recaudar la cantidad acordada ignorar el desarrollo del proyecto y quedarse con lo recaudado.

Esto es un hecho ya que KickStarter tiene un amplio historial de proyectos fraudulentos, estafas, proyectos inviables, los cuales al final ninguno de los inversores recibieron el reembolso.[6][11][1][15][5]

Podemos asegurar que toda plataforma de recaudación de fondos que siga los mismo principios que esta tendrá las mismas flaquezas y se verá afectada de forma similar con respecto a la aparición de proyectos fraudulentos.

2.1.2. Indiegogo

Indiegogo es un sitio web estadounidense de crowdfunding fundado en 2008 por Danae Ringelmann, Slava Rubin y Eric Schell. Su sede se encuentra en San Francisco, California. El sitio es uno de los primeros sitios en ofrecer financiación colectiva. Indiegogo permite a las personas solicitar fondos para una idea, una organización benéfica o una empresa nueva. [9]

Indiegogo, al igual que Kickstarter, es una plataforma web2 de financiamiento colectivo en línea que permite a los emprendedores, creadores y organizaciones recaudar fondos para sus proyectos, productos o ideasm, y la cual también posee un FAQ[34] describiendo en detalle los pasos los cuales debería seguir un inversor si el proyecto en el que ha invertido no logra desarollarse.

En este documento se confirma la misma situación que en la plataforma Kickstarter. El curso de acciones a realizar es par con el inversor y el proponente únicamente, la plataforma no se responsabiliza de los fondos ni realiza un seguimiento de los hitos del proyecto.

La fiabilidad del proyecto recae sobre los comentarios y opiniones propuestas por anteriores inversores, por ello todo el proceso de reputación del proyecto recae sobre la comunidad.

2.1.3. Crowdfunder

Crowdfunder es una plataforma web2 de recaudación de fondos orientada a caridades, comunidades, empresas y individuales. Localizada en UK, esta plataforma tiene reputación por ser una de las plataformas más usadas sin ámbito de desarrollo de proyectos, donde sus fondos se usan para financiar todo tipo eventos y trabajos.

Esta plataforma usa un sistema idéntico al de IndieGogo y Kickstarter, incluido el relevo de la responsabilidad de la veracidad de los proyectos a la propia comunidad [21]. La plataforma no se hace cargo de las perdidas ocasionadas en caso de haber financiado un proyecto fraudulento, y recae en la comunidad reportar y revisar este tipo de proyectos.



2.1.4. GitCoin

Gitcoin[27] es una plataforma web3 que utiliza la tecnología blockchain para conectar a desarrolladores y creadores de contenido con oportunidades de financiamiento y trabajo en proyectos de código abierto. Esta plataforma se enfoca en fomentar la colaboración y el desarrollo sostenible de la comunidad de código abierto.

A través de esta plataforma, los desarrolladores pueden encontrar oportunidades de trabajo y financiamiento para sus proyectos de código abierto, mientras que los creadores de contenido pueden ganar dinero por la producción de tutoriales y documentación de proyectos. La plataforma utiliza un token llamado GTC (Gitcoin Coin) como medio de pago para las contribuciones de la comunidad.

Esta también ofrece herramientas y recursos para la gestión de proyectos de código abierto, incluyendo un tablero de tareas y un sistema de seguimiento de problemas. Esto ayuda a mantener la transparencia y la responsabilidad en el desarrollo de proyectos de código abierto y fomenta una cultura de colaboración y aprendizaje continuo.

Aunque Gitcoin proporciona un espacio para financiar y colaborar en proyectos de código abierto, la adopción y el conocimiento de la plataforma aún pueden ser limitados en comparación con otras soluciones de financiamiento. Además, la naturaleza descentralizada de las criptomonedas y la falta de regulación pueden generar cierto grado de incertidumbre y volatilidad en las recompensas recibidas por los desarrolladores.

El sistema sobre el que se ejecuta la plataforma utiliza el protocolo blockchain Ethereum para gestionar sus activos y contratos inteligentes. La plataforma se basa en la tecnología de blockchain para crear un sistema seguro, transparente y descentralizado para la gestión de fondos y la distribución de recompensas.

Los contratos inteligentes de Gitcoin están desplegados en la red principal de Ethereum (también conocida como Ethereum Mainnet). Esto significa que todas las transacciones y operaciones que se realizan en la plataforma se registran en la cadena de bloques de Ethereum, esto garantiza la inmutabilidad y la transparencia de los datos.



2.1.5. CoinStarter

CoinStarter[28], fundada en 2017 por Cornell Holloway, es una de las plataformas web3 de financiamiento colectivo más grandes basadas en criptomonedas. La estructura de esta plataforma, impulsada por la comunidad, ofrece una excelente manera de conectar a los donantes con los proyectos en los que creen de forma más intrínseca.

Para comenzar a invertir en CoinStarter se requiere una membresía. Esta membresía permite a las causas generar sus propios modelos de suscripción para obtener ingresos. Esto implica que los usuarios pagan una tarifa mensual para acceder a la extensa comunidad de la plataforma. No obstante, las donaciones también están sujetas a tarifas de transacción.

Uno de los aspectos destacados de CoinStarter es su diseño simplificado que permite a los usuarios, incluso aquellos que no tienen experiencia en criptomonedas, crear sus propias campañas de financiamiento colectivo. La plataforma es flexible y cuenta con salas de chat estilo Discord, que facilitan a los proyectos la identificación de audiencias objetivo y la comprensión de sus necesidades.

Los contratos inteligentes de CoinStarter están desplegados en la red Ethereum. Los beneficios de utilizar CoinStarter incluyen la posibilidad de acceder a una amplia comunidad de inversores y donantes interesados en proyectos basados en criptomonedas, la facilidad de uso para usuarios con poca experiencia en criptomonedas y la seguridad y confianza proporcionadas por la red Ethereum y sus contratos inteligentes. Además, la plataforma fomenta una mayor conexión entre donantes y proyectos, permitiendo a los creadores adaptar sus campañas a las necesidades de su público objetivo.

Los negativo por otro lado es el propio requerimiento de una membresía que incluye una tarifa mensual, lo que podría disuadir a algunos usuarios. Además, las donaciones están sujetas a tarifas de transacción, lo que podría reducir la cantidad de fondos que llegan a los proyectos.



2.1.6. Tecra Space

Al igual que CoinStarter, Tecra Space es una plataforma web3 de financiamiento colectivo que adopta un enfoque más tradicional. Este espacio descentralizado permite la inclusión de proyectos diversos que involucran patentes, activos digitales y otros derechos de propiedad intelectual, los cuales se desglosan y tokenizan.

La plataforma facilita a los desarrolladores la inclusión de sus proyectos y la oferta de plazos variados para los tokens y sus recompras. Además, Tecra Space permite que los proyectos se incluyan con diferentes niveles de inversión y ventajas para los inversores, como acceso a productos o servicios digitales, menciones, créditos, ilustraciones temáticas, entre otros.

Tecra Space incluye una amplia gama de proyectos, como videojuegos, aplicaciones descentralizadas, operaciones de minería y servicios de energía renovable. Los contratos inteligentes de Tecra Space están desplegados en la red Ethereum, garantizando seguridad y confiabilidad en las transacciones y procesos automatizados.

En cuanto a los beneficios de Tecra Space, se destacan la versatilidad en la inclusión de proyectos, la posibilidad de ofrecer ventajas personalizadas a los inversores y la descentralización a través de la tecnología blockchain. Esto permite una mayor flexibilidad y alcance para los creadores de proyectos y los inversores.

Por otro lado, los aspectos negativos incluyen la complejidad asociada a la tecnología blockchain y criptomonedas para usuarios sin experiencia en el área, y la incertidumbre regulatoria en torno a la tokenización de activos intangibles. Estos aspectos pueden representar desafíos tanto para los creadores de proyectos como para los inversores en la plataforma.



Para detallar las características estas plataformas, a continuación se muestra una tabla comparativa entre las plataformas mencionadas previamente, basándose en el modelo de financiamiento (A), moneda utilizada (B), industria principal (C), Comisión (D), plazo de financiamiento (E), distribución (F).

	Kickstarter	Indiegogo	Crowdfunder	GitCoin	CoinStarter	Tecra Space
A	All-or-nothing	Equidad	Flexible	Recompensas	ICO / STO	ICO / STO
B	Fiat	Fiat / Crypto	Fiat	Crypto	Crypto	Crypto
C	Tecnología	Tecnología	StarUps	Software	Blockchain	Ciencia
D	5 %	5 %	Varía	0 %	Varía	Varía
E	Hasta 60 días	Flexible	Flexible	Flexible	Flexible	Flexible
F	Centralizado	Centralizado	Centralizado	Varía	Descentralizado	Descentralizado

Cuadro 1: Estado del Arte - Tabla comparativa de plataformas de crowdfunding



2.2. Conclusiones respecto al estado del arte

De esta investigación se puede sacar en claro que las plataformas de crowdfunding actuales desplegadas que no son descentralizadas relevan su responsabilidad con el inversor, ya que en caso de tomar dicha responsabilidad la propia plataforma debería emprender acciones legales en contra del propONENTE en los casos de haber ejecutado una estafa o fraude.[18]

También podemos ver que la mayoría de las plataformas están reguladas por la comunidad, aunque la decisión final esté en manos de un administrador que posee la capacidad de alterar cualquier dato.

El mismo administrador en un futuro podría suponer una flaqueza para el sistema, ya que dependiendo de como esté desarrollado el sistema, este puede poseer la autoridad de enmascarar cualquier estafa publicada en la plataforma.

Para que nuestro sistema no sufra dichas flaquezas necesitamos que los activos recaudados puedan ser congelables¹² y que la plataforma esté totalmente controlada por la comunidad de una forma balanceada.

Además, una blockchain descentralizada, esta al ser inmutable no posee los riesgos externos comparados con el uso de un sistema centralizado, como por ejemplo que el recaudador o un administrador malicioso pueda modificar o eliminar una propuesta de proyecto sin dejar rastro de este.

También, a través de los Smart Contracts, se puede crear un sistema de gobernanza que no necesite ningún administrador, y que esté totalmente regulado por la comunidad.

Para este trabajo es importante que la blockchain sea de acceso público, debido a que queremos que la comunidad tenga el control sobre los activos, y cualquier inversor pueda trazar los activos que se han invertido en un proyecto. Además, la red debe ser modular, para evitar los problemas de escalado que tienen las redes blockchain monolíticas a largo plazo.

El consenso de la red blockchain debe ser Proof of Stake, ya que es la forma más cómoda para que los usuarios puedan participar en la red, sin que haya discriminación de recursos.

La discriminación de recursos en las redes blockchains se da cuando estas usan el consenso POW. Este consenso da pie a que un nodo minero pueda verificar transacciones más rápido que el resto de nodos, así obteniendo un beneficio mayor y todo debido a que los recursos computacionales del nodo son mayores. Esto afecta a la red, ya que a largo plazo provoca que el resto de nodos con menos recursos obtengan un beneficio cada vez menor.

En el siguiente capítulo se describe más en detalle la tecnología y la configuración a usar óptima para la propuesta de solución.

¹²Actividad que paraliza la transferencia de fondos.

3. Contexto Tecnológico

En este apartado se describe y explica en profundidad las tecnologías blockchain, como funcionan y que beneficios y contras nos provee.

3.1. Las redes Blockchain

Una blockchain[12] no es más que una red de ordenadores que funcionan de forma descentralizada, esto significa que los ordenadores no dependen del resto de los ordenadores de la red para seguir con el correcto funcionamiento, pero dicha descentralización fortalece el objetivo de la red.

Aunque hayan diferentes tipos de blockchain orientadas a diferentes objetivos, todas tienen una función común, y es la de ejercer como una base de datos. Mediante la criptografía, la propia red blockchain se fortalece con la descentralización, proveyendo de trazabilidad, inmutabilidad y privacidad.

Los datos guardados se replican en la red, y se almacenan de forma consecutiva y secuencial en cada nodo. Al almacenar un nuevo dato se crea una transacción en la blockchain firmada por el usuario que añade dicho dato, así ejerciendo como un libro mayor descentralizado en el cual se guarda que transacción ha emitido cada usuario.

Esta transacción es un mero registro con varios datos: un identificativo del usuario que ha insertado el dato, la firma del usuario asociado al dato a insertar y un hash¹³ de la combinación del dato insertado anteriormente y el dato actual así formando una Merkle Tree[4]. Y una vez que se han insertado varios datos, estos se agrupan en un bloque y se archivan para facilitar su acceso.

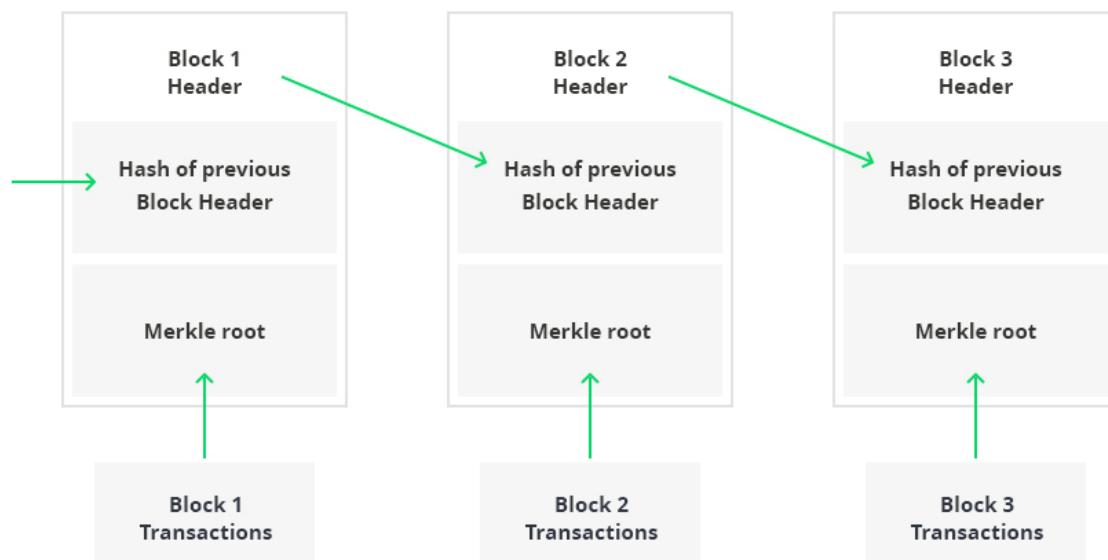


Figura 3: Estructura de una Blockchain.

¹³Algoritmo matemático que transforma el conjunto de datos de entrada en una expresión alfanumérica que tiene una longitud predeterminada.

De esta forma, se genera una arquitectura basada en bloques de datos que enlaza (mediante el hash) cada transacción sucesivamente (Blockchain).

Como se ha mencionado, cada transacción posee un hash que representa la integridad de todos los datos anteriores. Mediante el hash de la ultima transacción de esta cadena, se puede validar que ningún dato ha sido alterado.

3.1.1. Inmutabilidad y Trazabilidad

La inmutabilidad se logra a través de un proceso llamado “consenso” usando el hash de cada dato. En este proceso, un grupo de nodos verifica y valida cada transacción antes de agregarla a la blockchain. Una vez que se agrega un bloque, no puede ser modificado o eliminado.

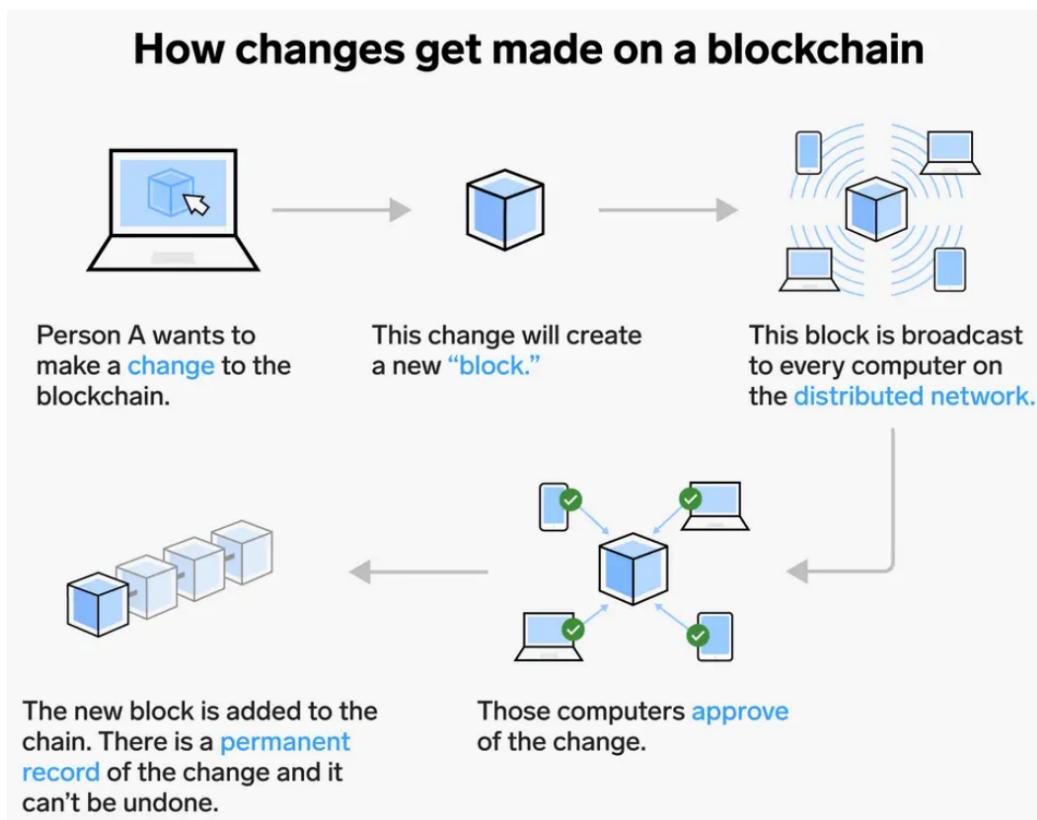


Figura 4: Funcionamiento de una Blockchain - Business Insider.

La trazabilidad se logra porque cada transacción en la blockchain está vinculada al usuario que la ha emitido, lo que permite seguir la pista de quién hizo qué y cuándo.

Mientras más nodos posea una red, más difícil será atacarla, ya que cada nodo de la red posee una copia exacta de los datos, de esta forma se valida la integridad de los datos entre los nodos de la red.



3.1.2. Consenso y tipos de consensos

El consenso[7] en una blockchain es el proceso por el cual los nodos de la red acuerdan sobre la veracidad de las transacciones y la creación de nuevos bloques en la cadena. Es esencial para garantizar la integridad y seguridad de la red y evitar varias formas de fraude.

Existen varios tipos de consenso en las blockchains, algunos de los más comunes incluyen:

3.1.2.1 Proof-of-Work (PoW)

El consenso PoW es un sistema en el que los nodos mineros compiten para resolver un problema matemático complejo y crear un nuevo bloque. El primer nodo que lo resuelve recibe una recompensa en criptoactivos (critomonedas / tokens).

Pros:

- Seguridad: El sistema Proof of Work proporciona una forma robusta de proteger una red blockchain contra ataques y manipulaciones.
- Decentralización: Al requerir que los mineros compitan entre sí para validar bloques, la red no está controlada por una sola entidad, sino por un grupo descentralizado de participantes.
- Immutabilidad: Una vez que un bloque es validado y agregado a la cadena, no se puede modificar sin la necesidad de recrear toda la cadena desde el bloque cambiado, lo que resulta en una inalterabilidad más alta.

Cons:

- Consumo de energía: El proceso de minería en sí consume una gran cantidad de energía, lo que resulta en un impacto ambiental negativo.
- Lentitud: El tiempo requerido para validar un bloque puede ser demasiado largo, lo que resulta en una menor escalabilidad y eficiencia.
- Centralización: A medida que los costos de minería aumentan, solo las grandes corporaciones con acceso a grandes cantidades de hardware y energía pueden competir en la minería, lo que puede resultar en una centralización y una menor descentralización de la red.

3.1.2.2 Proof-of-Stake (PoS)

El consenso PoS es un sistema en el que los nodos validadores son seleccionados aleatoriamente para validar las transacciones y insertar nuevos bloques basados en la cantidad de criptoactivos que poseen y están dispuestos a “apostar”.

Pros:

- Reducción de la energía: El sistema Proof of Stake consume significativamente menos energía que Proof of Work, lo que resulta en un impacto ambiental significativamente menor.
- Mayor escalabilidad: Al requerir menos recursos para validar bloques, la red es más eficiente y escalable.



- Descentralización mejorada: Al permitir que los validadores seleccionados sean elegidos por la cantidad de criptomonedas que poseen, el sistema Proof of Stake puede resultar en una mayor descentralización y una menor centralización en comparación con Proof of Work.

Cons:

- Vulnerabilidad a ataques: Si un validador malintencionado tiene una participación significativa en la red, puede controlar la validación de los bloques y llevar a cabo ataques.
- Falta de inmunidad ante manipulaciones: Al no requerir un esfuerzo computacional para validar bloques, el sistema Proof of Stake es más vulnerable a manipulaciones y ataques.
- Dificultad para participar: Al requerir que los validadores posean una cantidad significativa de criptomonedas, el sistema Proof of Stake puede resultar en una barrera para la participación de nuevos usuarios en la red.

3.1.2.3 Delegated Proof-of-Stake (DPoS)

El consenso DPoS es un sistema en el que los nodos validadores son elegidos por la comunidad de usuarios a través de un sistema de votación.

- Mayor escalabilidad: Al limitar el número de validadores en la red, el sistema Delegated Proof of Stake permite una mayor velocidad de procesamiento y una mejor escalabilidad en comparación con otros sistemas de consenso.
- Reducción de costos: Al requerir menos recursos para validar bloques, DPoS resulta en una reducción de los costos de hardware y electricidad en comparación con otros sistemas de consenso.
- Descentralización mejorada: Al permitir a los usuarios elegir a los validadores, DPoS puede resultar en una descentralización más efectiva y una menor centralización en comparación con sistemas de consenso centralizados.

Cons:

- Vulnerabilidad a la manipulación: Al permitir a los usuarios elegir a los validadores, DPoS es más vulnerable a la manipulación y a la influencia de grupos con intereses específicos.
- Centralización potencial: Si un pequeño grupo de validadores controla una gran cantidad de la red, puede resultar en una centralización y una falta de descentralización.
- Falta de inmunidad ante ataques: Al tener un pequeño número de validadores, la red puede ser más vulnerable a ataques y manipulaciones en comparación con otros sistemas de consenso.



3.1.2.4 Practical Byzantine Fault Tolerance (PBFT)

El consenso PBFT es un algoritmo en el que los nodos validadores se comunican entre sí para acordar la veracidad de las transacciones y la creación de nuevos bloques.

Pros:

- Alta disponibilidad: El algoritmo Practical Byzantine Fault Tolerance (PBFT) se diseñó específicamente para manejar la falla de nodos y garantizar la disponibilidad de la red incluso en presencia de fallas.
- Velocidad y escalabilidad: PBFT permite una velocidad de procesamiento rápida y una mejor escalabilidad en comparación con otros sistemas de consenso basados en pruebas.
- Seguridad mejorada: Al requerir un consenso entre los nodos en la red antes de validar un bloque, PBFT mejora la seguridad y reduce la vulnerabilidad a ataques en comparación con otros sistemas de consenso.

Cons:

- Centralización potencial: Al requerir un consenso entre los nodos en la red, PBFT puede resultar en una centralización si un pequeño número de nodos controla una gran cantidad de la red.
- Dificultad para implementar: El algoritmo PBFT es más complejo que otros sistemas de consenso y puede ser más difícil de implementar y mantener.
- Requerimiento de recursos: PBFT requiere una cantidad significativa de recursos para funcionar de manera efectiva, lo que puede resultar en una barrera para la participación de nuevos usuarios en la red.

Cada tipo de consenso tiene sus propios pros y contras en términos de seguridad, escalabilidad, eficiencia y costo, y la elección de un sistema de consenso puede afectar significativamente el funcionamiento y desempeño de una blockchain.



3.1.3. Nodos y tipos de Nodos

Los nodos[33] son los ordenadores que forman parte de la red blockchain, cada nodo puede tener una responsabilidad, y dependiendo del tipo de blockchain y para lo que esté orientada, este tipo de nodos pueden ser diferentes. Basada en esta responsabilidad los nodos más comunes son los siguiente:

3.1.3.1 Nodo Validador

Un nodo validador o nodo minero en una blockchain es un participante especial que se encarga de realizar la función de consenso y crear nuevos bloques en la cadena.

Los nodos validadores o mineros verifican las transacciones y las incluyen en un nuevo bloque siguiendo un protocolo de consenso específico, como Proof-of-Work (PoW) o Proof-of-Stake (PoS).

Al crear nuevos bloques y agregarlos a la cadena, estos nodos ayudan a mantener la integridad y seguridad de la red y son recompensados con criptomonedas o tokens.

3.1.3.2 Nodo Completo

Un nodo completo en una blockchain es un participante que mantiene una copia completa de la blockchain y verifica todas las transacciones y bloques antes de aceptarlos y agregarlos a la cadena.

Estos nodos cumplen un papel importante en mantener la integridad y seguridad de la red, ya que verifican que todas las transacciones cumplan con las reglas y protocolos establecidos de la red.

3.1.3.3 Nodo Parcial

Un nodo parcial en una blockchain es un participante que mantiene solo una parte de la información de la blockchain, en comparación con un nodo completo que mantiene una copia completa de la cadena.

Estos nodos pueden ser menos seguros y confiables que los nodos completos, ya que dependen de la información proporcionada por otros nodos para validar las transacciones y agregar bloques a la cadena.

Sin embargo, también pueden ser más eficientes en términos de recursos y pueden cumplir un papel útil en la escalabilidad de la red.

3.1.3.4 Nodo Archivo

Un nodo archivo en una blockchain es un nodo que mantiene una copia completa de la historia de todas las transacciones y bloques en la cadena. Este tipo de nodo está diseñado para proporcionar acceso a información histórica y auditável de la red a los usuarios y aplicaciones que la utilizan.

Los nodos archivo son diferentes a los nodos completos, ya que los nodos completos están activamente involucrados en la verificación y el procesamiento de transacciones, mientras que los nodos archivo están diseñados principalmente para proporcionar acceso a la información histórica de la blockchain.

Estos nodos son útiles para aplicaciones que requieren acceder a información detallada sobre transacciones y eventos pasados en la blockchain, como auditorías financieras, análisis de datos y



supervisión reguladora. Sin embargo, debido a la gran cantidad de datos que deben almacenar, los nodos archivo pueden requerir una gran cantidad de recursos de hardware y ancho de banda.

3.1.3.5 Nodo de Autoridad

Un nodo de autoridad en una blockchain es un nodo especial que se confía explícitamente para proporcionar información veraz y actualizada sobre la blockchain. Estos nodos suelen ser administrados por organizaciones confiables, como bancos, gobiernos u organizaciones reguladoras, y están diseñados para proporcionar información sobre la blockchain a terceros que necesitan verificar transacciones o eventos en la red.

Este tipo de nodos se usan en entornos regulados o en los que se requiere una mayor transparencia y confianza en la información de la blockchain, como en el caso de aplicaciones financieras o de seguimiento de suministro. Sin embargo, también pueden ser criticados por ser un punto centralizado de falla y por socavar la descentralización y la confianza en la red.

3.1.4. Cuentas y Carteras

Para que un usuario pueda interactuar con una blockchain, este deberá usar una cuenta. Las cuentas en una blockchain están formadas por 2 claves, una pública y otra privada, la clave pública se usa para identificar la cuenta, y la privada se usa para acceder y firmar las transacciones.

Las cuentas sirven para tener un registro público de transacciones y saldo. Para interactuar con una cuenta se requiere una “cartera”, que es el nombre que se le da a un software que permite a un usuario controlar sus cuentas y realizar transacciones. La cartera puede ser un software en línea o un dispositivo físico que almacena las claves privadas asociadas con las cuentas del usuario en la blockchain.



3.1.5. Accesos y tipos de acceso

Dependiendo del objetivo y tipo de blockchain (empresarial, gubernamental, publica, etc), el acceso a ésta puede clasificarse en 4 tipos generales.

4 main types of blockchain technology

	Public (permissionless)	Private (permissioned)	Hybrid	Consortium
ADVANTAGES	+ Independence + Transparency + Trust	+ Access control + Performance	+ Access control + Performance + Scalability	+ Access control + Scalability + Security
DISADVANTAGES	- Performance - Scalability - Security	- Trust - Auditability	- Transparency - Upgrading	- Transparency
USE CASES	■ Cryptocurrency ■ Document validation	■ Supply chain ■ Asset ownership	■ Medical records ■ Real estate	■ Banking ■ Research ■ Supply chain

Figura 5: Ventajas y desventajas de los tipos de blockchain basado en su acceso - Tech Target Media.

3.1.5.1 Pública

Una blockchain pública es una blockchain abierta y accesible para cualquier persona. Todos pueden leer, escribir y participar en la red. Ejemplos incluyen Bitcoin y Ethereum.

3.1.5.2 Privada

Una blockchain privada es una blockchain que restringe el acceso a un grupo específico de participantes. Este tipo de blockchain se utiliza en aplicaciones empresariales y puede ser administrados por una organización o un grupo de organizaciones.

3.1.5.3 Híbrida

Una blockchain híbrida es una combinación de una blockchain pública y privada. En una blockchain híbrida, un grupo de nodos es responsable de validar las transacciones y mantener la integridad de la red, mientras que otros nodos tienen acceso de lectura.

3.1.5.4 Consorcio

Una blockchain de consorcio es similar a una blockchain federada, pero está controlada por un grupo de organizaciones que trabajan juntas. Este tipo de blockchain se utiliza en aplicaciones empresariales y permite un mayor control y privacidad de las transacciones de la red.

3.1.6. Blockchains monolíticas y modulares

También, dependiendo del tipo de arquitectura existen 2 clasificaciones de blockchains: Monolítico y Modular.

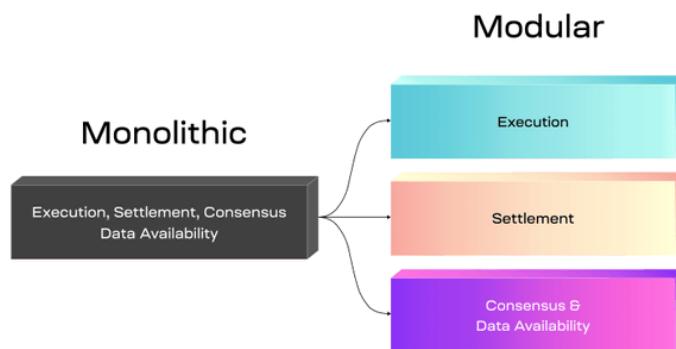


Figura 6: Diagrama representativo de las blockchains monolíticas y modulares - Médium.

La diferencia entre monolíticas y modulares es la forma en que están diseñados y organizados los componentes. Las blockchains monolíticas tienen todas las funciones en un solo módulo, mientras que las blockchains modulares tienen diferentes funciones en módulos separados.

Las blockchains modulares son las más usadas hoy en día, porque permiten relevar la carga de trabajo de un módulo a otras blockchains. A esta práctica se le conoce como Layer 2[3], que describe un set específico de soluciones de escalado para la blockchain.

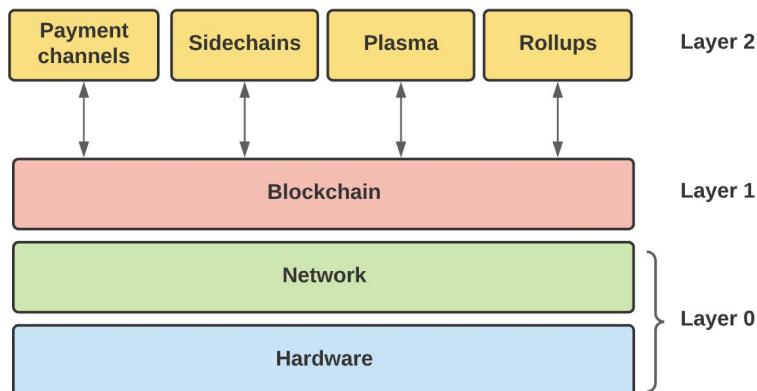


Figura 7: Representación de las layers del 0 al 2 de una blockchain.

3.1.6.1 Criptomonedas

Las criptomonedas, aunque no sean totalmente necesarias en una blockchain, sigue siendo una parte fundamental para muchos ecosistemas en los cuales se requiere de un incentivo para minar o validar transacciones.

Las criptomonedas surgieron originalmente del Bitcoin, una de las primeras redes blockchain. Esta red fue creada con el objetivo de crear el primer criptoactivo de Internet, una moneda la cual no se vería influenciada por empresas o gobiernos, y que fuese de libre uso para todas las personas.

Hoy en día existen blockchains orientadas a todo tipo de finalidades, pero la mayoría poseen una criptomoneda nativa a cada red. Esta criptomoneda se usa para recompensar y fomentar que las personas usen la red. Por ejemplo, la propia red recompensa a los mineros y validadores al prestar su capacidad de computo para verificar las transacciones de la red.

3.1.7. Smart Contracts - Chain Code

Algunas blockchains como Ethereum permiten aprovechar la capacidad computacional de los nodos de la red, de tal forma la blockchain además de solo almacenar datos también puede ejecutar programas en los nodos.

Los Smart Contracts o también conocido como Chain-Code son programas que se ejecutan en la blockchain y permiten automatizar el cumplimiento de acuerdos digitales. Estos programas una vez desplegados son inmutables, por lo tanto las funciones y lógica definidas son inalterables.

Estos contratos pueden almacenar y transferir valores, activos o información en una forma segura, transparente y sin intermediarios, gracias a la inmutabilidad y descentralización de la blockchain.

De esta manera, los smart contracts permiten crear una red de confianza entre sus participantes sin la necesidad de terceros confiables.

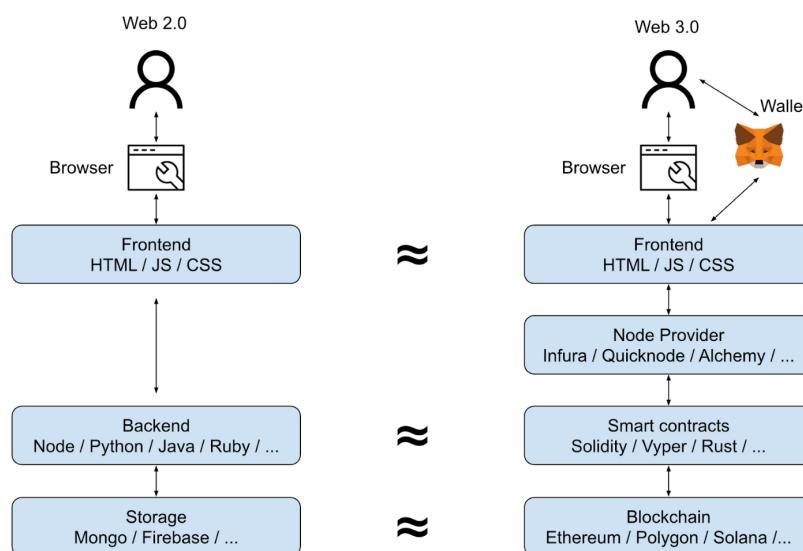


Figura 8: Comparación entre web2 y web3 del proceso de interacción con aplicaciones.



3.2. Ethereum

Ethereum[19] es una blockchain pública modular, que soporta diferentes consensos, y basada en DLT con su propia criptomoneda y con la capacidad de almacenar datos y ejecutar lógica en sus nodos.

Esta última característica es la que la diferencia de la mayoría de blockchains de hoy en día, además de la posibilidad de poder clonar el proyecto, y desplegar una versión propia de esta blockchain, ya que el protocolo Ethereum es un proyecto de código abierto.

3.2.1. El protocolo Ethereum

El protocolo Ethereum[13] es un proyecto desarrollado en C++, Python y Go, que permite desplegar una blockchain personalizable.

El protocolo despliega una red basa en una estructura de blockchain y utiliza que propia criptomoneda, Ether (ETH), para compensar a los nodos en la red por el trabajo que realizan al validar y ejecutar las transacciones y contratos en la red.

Además, este protocolo permite la creación de tokens[36] personalizados, lo que lo hace popular para la creación de proyectos de finanzas descentralizadas (DeFi) y aplicaciones descentralizadas (dApps). En otras palabras, el protocolo Ethereum es la base tecnológica sobre la cual se ejecuta la blockchain Ethereum.

Esto también significa que la blockchain Ethereum no es la única que ejecuta el protocolo Ethereum, por ejemplo Polygon, Telos, Cardano, Avalanche, TRON son redes blockchain con ciertas diferencias a la blockchain Ethereum pero que siguen ejecutando el mismo protocolo.

3.2.2. Ethereum Virtual Machine

La Máquina Virtual Ethereum (EVM)[14] es el motor de Ethereum, se encarga de almacenar todos los datos y de ejecutar las operaciones de los contratos inteligentes.

Además es entorno de ejecución de los contratos inteligentes. La EVM permite que los contratos se ejecuten de manera uniforme y segura en todos los nodos en la red, independientemente del hardware y del software que utilicen.

Mientras Ethereum tiene su propia criptomoneda nativa (Ether) que sigue casi exactamente las mismas reglas intuitivas, también permite una función mucho más poderosa: los contratos inteligentes. Para esta función más compleja, se requiere una analogía más sofisticada.

En lugar de un ledger distribuido[30], Ethereum es una máquina de estado distribuido. El estado de Ethereum es una estructura de datos grande que no solo contiene todas las cuentas y saldos, sino también un estado de la máquina, que puede cambiar de bloque a bloque de acuerdo a un conjunto de reglas preestablecidos y que puede ejecutar cualquier código de máquina.

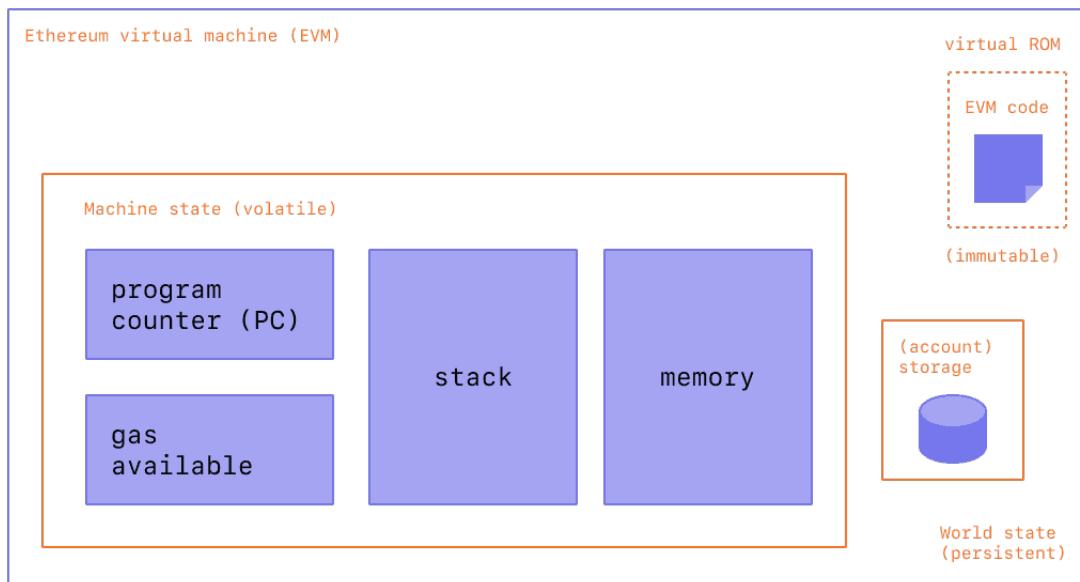


Figura 9: Diagrama que representa la EVM - Ethereum.org .

Las reglas específicas para cambiar el estado de bloque a bloque están definidas por la EVM. Los contratos inteligentes se escriben en un lenguaje de programación de alto nivel, ya sea usando un lenguaje orientado a objetos mediante el uso de alguna librería como Viper[32] para Python, o mediante Solidity[29], y luego se compilan a instrucciones bytecode que se puede ejecutar en la EVM. Una vez que un contrato se ha implementado en la EVM, los usuarios pueden interactuar con él mediante transacciones en la red.

En resumen, la EVM es un entorno de ejecución seguro y uniforme para contratos inteligentes en la red Ethereum, que permite que los contratos se ejecuten de manera confiable y consistente en todos los nodos en la red.

3.2.3. Funcionamiento de la EVM

El EVM se comporta como lo haría una función matemática, dada una entrada, produce una salida determinista. Por lo tanto, es muy útil describir de manera más formal a Ethereum como si fuese una función de transición de estados.

En el contexto de Ethereum, el “state” es una enorme estructura de datos Merkle Patricia Trie modificada[25], que mantiene todas las cuentas vinculadas por hashes y se puede reducir a un solo hash raíz almacenado en la blockchain.



3.2.4. Ejecución de la EVM

La EVM se encarga de ejecutar instrucciones definidas (OPCODES)[23], que se ejecutan en la blockchain.

Para evitar que hayan bucles infinitos u otras instrucciones capaces de agotar los recursos de los nodos de la blockchain y para incentivar a los mineros a validar las transacciones que realiza una instrucción, cada instrucción tiene un coste en ETH definido en una unidad llamada GAS.

3.2.5. Gas

El gas es una unidad de medida del ETH, que se usa para medir cuánto ETH cuesta ejecutar una operación.[16] Esta medida se rige por la siguiente formula:

$$\text{gasUsado} * \text{precioGas} = \text{tarifaDelGas}$$

Dado que cada transacción requiere recursos computacionales para ejecutarse, cada transacción requiere una tarifa. Gas se refiere a la tarifa requerida para realizar una operación en Ethereum con éxito.

Debido a la variación de la carga de trabajo de los nodos de la blockchain, el precio del gas suele oscilar durante el día. Cada año, debido a que la red siempre está en continua expansión, el gas se vuelve ligeramente más caro.

Hay páginas como [Etherscan](#)¹⁴ que muestran el estado de la red, sus transacciones, y el precio estimado del gas durante el día.

El cálculo de la tarifa de las transacciones en gas funciona de la siguiente manera:

$$\text{unidades de gas (límite)} * (\text{tarifa base} + \text{propina})$$

Cuando un usuario ejecuta una instrucción, éste puede definir el límite de gas a usar, de esta manera, si hay gas sobrante (ETH que no se quema al ejecutar las instrucciones) el usuario puede elegir solicitar la devolución del cambio, o usarlo para dar prioridad a la ejecución de dicha transacción, siendo este gas sobrante un incentivo extra para los mineros.

¹⁴<https://etherscan.io/>

Esto se puede representar en la siguiente fórmula:

$$\text{reembolso} = \text{tarifa máxima} - (\text{tarifa base} + \text{tarifa prioritaria})$$

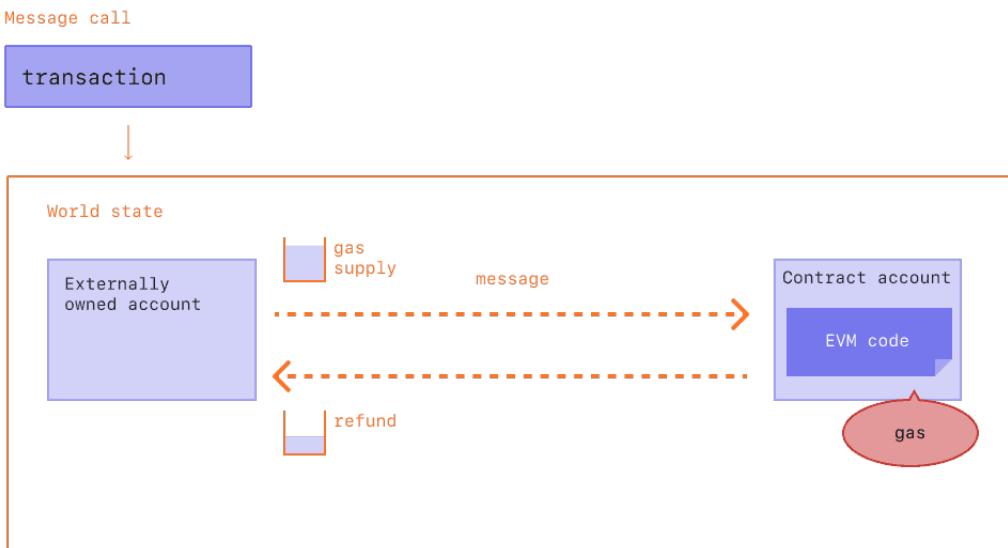


Figura 10: Diagrama que representa el reembolso del gas - Ethereum.org .

Por otro lado, cuando no hay gas suficiente, este gas se paga a los mineros por finalizar transacciones. Incluso si falla, los mineros deben validar y ejecutar la transacción.

3.2.6. Redes

Independientemente de la blockchain que usemos, si esta está basada en el protocolo Ethereum es probable que exista una red de pruebas[2]. Por cada Mainnet (blockchain principal) hay una testnet (red de pruebas).

La red Mainnet es la principal, y es donde se despliegan los contratos en producción. El coste del despliegue suele ser alto, y una vez desplegado el contrato ya no se puede modificar. Por lo tanto es muy arriesgado desplegar en la mainnet sin tener una red donde poder realizar pruebas sobre el smart contract.

Las testnets son redes donde los desarrolladores de smart contracts pueden desplegar sus contratos para probarlos. La criptomoneda de estas testnets no valen nada, y sirven únicamente para el propósito de probar el correcto funcionamiento del contrato a desplegar.

Los desarrolladores pueden obtener criptomonedas desde cuentas "grifo"[26] de forma gratuita para realizar las pruebas que crean oportunas sobre sus smart contracts.

Un ejemplo de mainnet y testnet son las redes Polygon y Mumbai, que se usarán para este trabajo.



3.2.7. Sobre Polygon

Polygon es una red blockchain escalable que se enfoca en mejorar la eficiencia, la velocidad y la accesibilidad de Ethereum. Fue fundada en 2017 bajo el nombre de Matic Network y ha sido una de las soluciones de blockchain más exitosas en términos de escalabilidad y adopción en la comunidad Ethereum.

Polygon ofrece una capa adicional en la blockchain de Ethereum que permite la realización de transacciones rápidas y eficientes a través de un sistema de confirmación de bloques descentralizado y validación de transacciones a nivel de nodo. Además, Polygon es compatible con las aplicaciones y los contratos inteligentes existentes de Ethereum, lo que facilita la migración a la red.

Pros de Polygon:

- Mejora en la escalabilidad: Polygon permite una mayor cantidad de transacciones por segundo y una confirmación de transacciones más rápida que Ethereum.
- Compatibilidad con Ethereum: Polygon es compatible con la mayoría de las aplicaciones y contratos inteligentes de Ethereum, lo que permite una migración sencilla a la red.

Contras de Polygon:

- Menor seguridad: Debido a la estructura de la red, Polygon presenta un nivel de seguridad algo inferior al de Ethereum, aunque se están llevando a cabo esfuerzos para mejorarlo.
- Menor adopción: Polygon aún no ha alcanzado el mismo nivel de adopción que Ethereum, lo que puede limitar la cantidad de aplicaciones y contratos inteligentes disponibles en la red.

3.2.8. Cuentas Externas y Cuentas de Contratos

Una cuenta de Ethereum es una entidad con un saldo de ether (ETH) que puede enviar transacciones en Ethereum. Las cuentas pueden ser controladas por el usuario o implementadas como smart contracts.

3.2.8.1 Externas

Estas cuentas representan una persona en la blockchain, su creación es gratuita y tienen claves públicas y privadas para poder acceder a ellas a través de una cartera, y firmar las transacciones.

3.2.8.2 Contratos

Una cuenta de contrato en Ethereum es una dirección especial en la blockchain que apunta a una instancia de un contrato inteligente. Los contratos también tienen cuentas, ya que pueden albergar Ether y Tokens, y también ejecutar transacciones en la red, pero solo si alguien externo llama a una de sus funciones, ya que los contratos no pueden ejecutarse por sí solos.



3.2.9. Transacciones en Ethereum

En Ethereum, una transacción representa una acción en la blockchain, como el envío de ETH o de un token basado en Ethereum de una cuenta a otra, la interacción con un contrato inteligente, o la modificación de los datos almacenados en la blockchain.

Cada transacción en Ethereum requiere una cantidad de gas para ser procesada y validada por los mineros de la red. Una vez que se realiza una transacción, se registra en la blockchain y se considera irrevocable.

3.2.10. Solidity: Lenguaje Orientado a Smart Contracts

En el caso de Ethereum, hay varias librerías para lenguajes como JS, Python o incluso C++ que nos permiten describir un smart contract y que compilan a bytecode que ejecuta la EVM (Ethereum Virtual Machine), aún así, se ha desarrollado un lenguaje específico para el desarrollo de smart contracts que compila a bytecode de EVM llamado Solidity.

Solidity[29] es un lenguaje que nos permite desarrollar Smart Contracts de forma cómoda, además Solidity posee sintaxis y definiciones que son similares a C++ y JS, aún así es importante no confundir Solidity como un lenguaje orientado a objetos, ya que este no entra en este paradigma.

3.2.11. Comportamiento de un Smart Contract en Ethereum

Un smart contract, al menos en Ethereum, se comporta muy similar a una API REST[35]. Cada contrato tiene un conjunto de funciones que se ejecutan sólo cuando se las llaman, estas pueden recibir, procesar, y devolver datos, además de alterar el estado de la blockchain, o incluso llamar a funciones de otros contratos.

Una vez desplegado un contrato, se genera una instancia de este en la blockchain en todos los nodos de la red, su código fuente no puede ser modificado y cada instancia es independiente.

Debido a que la EVM es una máquina de estados, al cambiar el valor de una variable de estado en nuestro contrato, lo que estamos haciendo en realidad es cambiar el estado de la blockchain.

Esto no significa que se esté sobreescribiendo dicho valor, en cambio se actualiza el valor más reciente para dicho espacio de almacenamiento en la blockchain.

En cualquier momento podemos ver los valores previos de estas variables a través del explorador de bloques, ya que cada cambio de estado es provocado por una transacción que debe ser minada, validada y finalmente archivada en un bloque.

Además, para cada contrato desplegado, se generan una cuenta de contrato, esto significa que el contrato en la blockchain tiene una dirección y una cartera para mantener Ether.

Una vez desplegado un contrato, para llamar a sus funciones deberemos saber la dirección del contrato, y la firma de la función (identificador de función).

A continuación se muestra ejemplo simple que de un smart contract hecho en Solidity que puede obtener, incrementar y decrementar un contador.



```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.17;
3
4 contract Counter {
5     uint public count;
6
7     // Funcion para obtener el valor de "count"
8     function get() public view returns (uint) {
9         return count;
10    }
11
12    // Funcion para incrementar count
13    function inc() public {
14        count += 1;
15    }
16
17    // Funcion para decrementar count
18    function dec() public {
19        // Esta funcion fallara si count = 0, esto se debe al tipo de count
20        count -= 1;
21    }
22 }
```

Bloque de Código 1: Ejemplo de un Smart Contract en Solidity.

3.2.12. Estructura de un smart contract con Solidity

La estructura de un Smart Contract es bastante sencilla, los Smart Contracts en Solidity suelen tener principalmente un conjunto de variables globales (variables de estado), un constructor y funciones.

A este programa se le puede añadir eventos, modificadores y librerías que se necesite importar, etc, pero al fin y al cabo cuando se interactua con un contrato, lo que se ejecutará será una llamada a una función, o la lectura de un valor de una variable de estado.

Variables - Almacenamiento

Las variables de un contrato se guardan en 3 ubicaciones diferentes en la blockchain.

- Storage: Persistente, se guarda en la blockchain.
- Memory: Variables locales / temporales, arrays.
- Calldata: Valores recibidos por parámetros de funciones, acceso externo.

Al igual que la RAM, Memory es un espacio temporal para almacenar datos durante la ejecución de un smart contract, pero se borra por completo una vez que la ejecución termina.

Storage, por otro lado, es persistente y los datos previamente almacenados en él están disponibles para cada ejecución del contrato. Sin embargo, su uso es costoso en términos de gas y se recomienda minimizar el número de accesos a las variables de estado. Las variables de estado solo se pueden declarar a nivel de contrato.



Calldata es similar a Memory y contiene los argumentos de la función, pero solo está disponible para los parámetros de llamadas a funciones externas.

En resumen, solo se debe especificar la zona de almacenamiento cuando se realizan transferencias de datos o al declarar variables dinámicas o secuenciales como arrays. Las variables en Memory solo se pueden declarar a nivel de función.

```
1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity >=0.7.0 <0.9.0;
3 contract Storage {
4
5     /// @dev numero es una variable de estado almacenada en Storage.
6     /// Por defecto, las variables declaradas a nivel de contrato se guardan en
7     /// Storage.
8     uint256 numero;
9
10    /// @dev _numero es una variable recibida por parametro almacenada en calldata.
11    function store(uint[] calldata _numero) public {
12        // Declaramos un array de enteros sin signo de tamaño 5.
13        // Esta variable es local, y guardada en Memory.
14        uint[] memory misNumeros = new uint[](5);
15
16        // Guardamos en Storage el valor de la posición del array _numero.
17        numero = _numero[0];
18
19        // Variable local, guardada en Memory.
20        uint8 n = 4;
21
22    }
23 }
```

Bloque de Código 2: Ejemplo de los diferentes espacios para alojar variables en Solidity.



Variables - Tipos de datos

En Solidity, existen diferentes tipos de variables que se pueden utilizar en un smart contract, como string, int, uint, arrays, etc. Además, es posible especificar el tamaño de la variable, por ejemplo: uint8, uint32, uint64, lo que permite reducir el costo de gas asociado con la asignación de memoria al interactuar con las variables.

Cuando no se especifica un tamaño, como en el caso de uint, se define con su tamaño máximo, es decir, uint = uint256.

Es importante tener en cuenta que el tamaño de la variable afecta la eficiencia de la asignación de memoria y, por lo tanto, también influye en el costo de gas asociado con la interacción con las variables.

```
1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity >=0.7.0 <0.9.0;
3 contract Variables {
4
5     uint numeroSinSigno;
6     int numeroConSigno;
7     string nombre;
8     bytes2 dosChars;
9     address direccionCuenta;
10
11    // Mapping es la equivalencia a un diccionario en Solidity.
12    mapping(address => string) nombres;
13
14    struct persona {
15        string nombre;
16        uint edad;
17    }
18
19    function setNombre (address _direccion, string memory _nombre) public {
20        nombres[_direccion] = _nombre;
21    }
22 }
```

Bloque de Código 3: Ejemplo de los diferentes tipos de variables en Solidity.



Variables - Inmutables

Las variables inmutables son aquellas que no pueden ser modificadas una vez se han establecido. Se pueden establecer valores para las variables inmutables dentro del constructor, pero no se pueden cambiar después. Las variables inmutables funcionan como constantes y se utilizan para almacenar valores que no deben ser modificados a lo largo del tiempo.

Al usar variables inmutables, se pueden asegurar que ciertos valores permanezcan constantes y sean confiables a lo largo de la ejecución del contrato. Esto es útil para establecer valores que se usan repetidamente en el código o para proporcionar una capa adicional de seguridad y prevenir errores involuntarios.

```
1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity >-0.7.0 <0.9.0;
3 contract Immutable {
4
5     address immutable MY_ADDRESS;
6     uint immutable MY_UINT;
7
8     constructor(uint _myUint) {
9         MY_ADDRESS = msg.sender;
10        MY_UINT = _myUint;
11    }
12 }
13 }
```

Bloque de Código 4: Ejemplo de variables inmutables en Solidity.

Pragma y Licencias

Las primeras 2 líneas de cada archivo .sol deben ser:

- La licencia del código del archivo.
- La versión de compilador de solidity a usar.

Si no se especifica el compilador esperado en el smart contract, se generará un aviso. Esto es un recordatorio para el desarrollador para especificar la versión de compilador adecuada. Si se especifica un compilador y se compila con una versión diferente o incompatible, el proceso de compilación será abortado.

Esto es importante para garantizar la compatibilidad y evitar errores debido a cambios en la sintaxis o en las funciones disponibles en diferentes versiones del compilador de Solidity. Especificar la versión de compilador correcta garantiza que el contrato se compilará correctamente y funcionará como se esperaba.

```
1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity >=0.7.0 <0.9.0;
```

Bloque de Código 5: Ejemplo de pragma y licencia en los archivos .sol.



En el ejemplo anterior se especifica que la licencia es GPL-3, y la versión del compilador debe ser superior o igual a 0.7.0 y inferior a la 0.9.0.

Constructores

El constructor de un smart contract es una función especial que se ejecuta automáticamente una sola vez, cuando se despliega el contrato en la blockchain. El constructor puede recibir argumentos o parámetros y utilizarlos para inicializar o configurar el estado inicial del contrato.

También es posible que un contrato herede de otro contrato y llame al constructor padre para inicializar algunos valores antes de continuar con su propia inicialización. Esto puede ser útil cuando se quiere reutilizar la lógica o los valores iniciales de otro contrato en un nuevo contrato.

```
1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity >=0.7.0 <0.9.0;
3
4
5 contract Acontract {
6
7     string name;
8
9     // Constructor.
10    constructor(string memory _name) {
11        name = _name;
12    }
13 }
14
15
16 // Herencia.
17 contract Constructor is Acontract {
18
19     uint number;
20
21     // Acontract() Es la llamada al constructor padre desde el constructor del mismo.
22     constructor(string memory _name, uint _number) Acontract(_name) {
23         number = _number;
24     }
25 }
```

Bloque de Código 6: Ejemplo de un constructor y herencias en Solidity.

En la herencia de contratos no se hereda el estado actual del contrato padre, solo sus variables, funciones, valores iniciales, etc. No es posible heredar de un contrato ya desplegado, solo del código fuente. Además, también es posible crear interfaces en Solidity.



Visibilidad

La visibilidad de una función o variable restringe su accesibilidad:

- Public: Cualquiera puede llamar a la función, (usuario, otro contrato, función interna, contrato heredado).
- External: Solo se puede llamar desde fuera del contrato actual, (usuario, otro contrato).
- Private: Solo contratos heredados o funciones internas pueden llamar a la función (función interna, contrato heredado).
- Internal: Solo funciones internas del contrato actual pueden llamar a la función (función interna).

```
1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity >=0.7.0 <0.9.0;
3 contract Funciones {
4
5     // Cualquier puede llamar a esta funcion f1.
6     function f1() public { }
7
8     // Solo cuentas externas puede a f2.
9     function f2() external { }
10
11    // Solo el contrato actual y heredados pueden llamar a f3.
12    function f3() private { }
13
14    // Solo el contrato actual puede llamar a f4.
15    function f4() internal { }
16 }
```

Bloque de Código 7: Ejemplo de visibilidades en funciones en Solidity.

Cuando una variable es declarada como pública, el compilador genera un getter implícito, lo que significa que no es necesario crear un getter explícito para acceder a su valor. Sin embargo, aunque se utilice private o internal para restringir el acceso, esto no garantiza la ocultación del valor.

La información almacenada en los contratos de Ethereum es pública y puede ser leída por cualquier persona en la blockchain mediante un block explorer. Es importante tener en cuenta que Ethereum es una blockchain pública y toda la información almacenada en los contratos es accesible para todos.

```
1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity >=0.7.0 <0.9.0;
3 contract Variables {
4
5     address public immutable MI_DIRECCION;
6     uint private immutable MI_UNIDAD;
7     string external immutable MI_STRING;
8     bytes8 internal immutable B8;
9
10 }
```

Bloque de Código 8: Ejemplo de visibilidades en variables en Solidity.



Funciones

Las funciones son el punto principal de interacción de un smart contract. A través de estas se proporciona la funcionalidad de la blockchain.

Funciones - Mutabilidad

La mutabilidad define si la función actual va a cambiar / acceder a alguna variable de estado, esto influye en el gasto de gas por el contrato, ya que si una función no cambia el estado de la blockchain tampoco habrá propagación, y por lo tanto se notifica a la EVM que no se necesita realizar una transacción para la ejecución de dicha función, esto hace que la llamada a dicha función sea gratuito. Las operaciones lógicas se ejecutarán sólo en un nodo.

- View: Especifica que la función no va a modificar las variables de estado (no se altera la blockchain).
- Pure: Igual que la anterior pero que además especifica que no se va a leer de las variables de estado (no se accede a la blockchain).

```
1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity >=0.7.0 <0.9.0;
3
4 contract Funciones {
5     uint256 numero;
6
7     function setNumero (uint256 _numero) public {
8         numero = _numero;
9     }
10
11    function getNumero() public view returns (uint256) {
12        return numero;
13    }
14
15    function suma (uint256 a, uint256 b) public pure returns (uint256) {
16        return a + b;
17    }
18 }
```

Bloque de Código 9: Ejemplo de mutabilidades en Solidity.



Funciones - Returns

Como se ha visto en el ejemplo anterior de mutabilidad, si una función devuelve un valor se debe especificar el tipo, si no devuelve nada no se requiere especificar nada.

Se pueden declarar directamente las variable a devolver como un “contenedor”, la cual la función al acabar su ejecución devolverá automáticamente su valor. También se puede devolver múltiples variables.

```
1  contract Funciones {
2      // M todo A
3      function f1() public pure returns (uint){
4          uint numero= 1;
5          return numero;
6      }
7
8      // M todo B
9      function f2() public pure returns (uint numero){
10         numero = 1;
11     }
12
13
14     // Devuelve varias variables
15     function devuelve2() internal pure returns (uint a, string b) {
16         a = 1;
17         b = "holo";
18     }
19
20     // Captura las variables
21     function captura2() public pure {
22         (uint numero, string memory palabra) = devuelve2();
23     }
24 }
```

Bloque de Código 10: Ejemplos de returns en Solidity.



Funciones - Payable

Es posible habilitar que un contrato tenga un balance y pueda recibir transferencias de Ether, al igual que una cuenta normal.

Se puede acceder al balance actual del contrato utilizando la sintaxis “address(this).balance”. Además, es posible permitir que una función del contrato reciba Ether a través de la adición de la palabra "payable".

Esta misma palabra también se puede utilizar para transferir Ether desde el contrato a una cuenta.

La dirección de la cuenta que ha llamado a la función se puede acceder a través de “msg.sender”, y la cantidad de Ether enviada a través de “msg.value”.

```
1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity >=0.7.0 <0.9.0;
3
4 // Contrato para depositar Ether.
5 contract Deposito {
6
7     mapping(address => uint) balances;
8
9     function depositar() public payable {
10         balances[msg.sender] += msg.value;
11     }
12
13     function retirar (uint256 cantidad) public {
14         balances[msg.sender] -= cantidad;
15         (bool success, ) = msg.sender.call{value: cantidad}("");
16         require(success, "Fallo en la transferencia, revertiendo estado...");
17     }
18
19     function totalDepositadoPorTodos() public view returns (uint256) {
20         return address(this).balance;
21     }
22 }
```

Bloque de Código 11: Ejemplo de un contrato usando Payable en Solidity.



Requires

Durante la ejecución de una función se puede realizar comprobaciones llamadas "requires". En caso de fallar estas comprobaciones, se revierte la transacción con un mensaje de error personalizado, lo que significa que se deshacen todos los cambios realizados hasta ese momento.

El consumo de gas para la transacción solo ocurre hasta que se ejecuta la reversión. Esto se debe a que, aunque se reviertan todos los cambios, se requiere capacidad computacional para realizar estas operaciones.

```
1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity >=0.7.0 <0.9.0;
3
4 contract Deposito {
5     address propietario;
6     mapping(address => uint) balances;
7
8     // Cuando el contrato se despliega en la red, se setea "propietario" con la
9     // Dirección de la cuenta ( usuario ) que ha desplegado el contrato. propietario
10    msg.sender;
11    constructor() {
12        propietario = msg.sender;
13    }
14
15    function depositar() public payable {
16        // ...
17    }
18
19    function retirar (uint cantidad) public {
20        // ...
21    }
22
23    // Función que vacía el contrato, solo puede ser ejecutada por el propietario.
24    function vaciar() public {
25        // Si la condición falla, se revierte con el mensaje de error.
26        require(msg.sender == propietario, "El usuario no es el propietario!");
27
28        // Se envía el Ether al propietario del contrato. Devuelve un booleano que
29        // indica si la transacción fue exitosa.
30        (bool success,) = msg.sender.call{value: address(this).balance}("");
31
32        // Si la transacción no fue exitosa, se revierte con el mensaje de error.
33        require(success, "Error en la transferencia de Ether");
34    }
35}
```

Bloque de Código 12: Ejemplo de sentencias require en Solidity.



Modifiers

Los modificadores son funciones auxiliares que se declaran con la sentencia “modifier”, y se adjuntan a otras funciones, de tal manera que la lógica de la función adjunta se ejecuta antes, durante o después de la auxiliar.

El carácter “_” del modificador es reemplazado por la lógica de la función.

```
1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity >=0.7.0 <0.9.0;
3
4 contract Deposito {
5     address propietario;
6
7     constructor() {
8         propietario = msg.sender;
9     }
10
11     // Modificador que comprueba si el usuario que ha llamado a la función es el
12     // propietario.
13     modifier soloPropietario() {
14         require(msg.sender == propietario, "El usuario no es el propietario!");
15
16         // Ahora se ejecuta la función a la que se adjunta el modificador.
17        _;
18     }
19
20     // Adjuntamos el modificador a la función vaciar().
21     function vaciar() public soloPropietario {
22         (bool success, ) = msg.sender.call{value: address(this).balance}("");
23         require(success, "Error en la transferencia de Ether");
24     }
}
```

Bloque de Código 13: Ejemplo del uso de modifiers en Solidity.

Los modifiers también soportan atributos.

```
1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity >=0.7.0 <0.9.0;
3
4 contract Suma {
5     modifier numerosPares(uint numero1, uint numero2) {
6         require(
7             (numero1 % 2 == 0) &&
8             (numero2 % 2 == 0),
9             "Uno de los dos números no es par");
10       _;
11    }
12
13    function sumaPares (uint a, uint b) public pure numerosPares(a, b) returns (uint)
14    {
15        return a + b;
16    }
}
```

Bloque de Código 14: Ejemplo del uso de modifiers con parámetros en Solidity.



Funciones por defecto.

En Solidity hay 2 funciones que nos permiten definir el comportamiento del contrato ante ciertas circunstancias:

- **receive()** external payable: Se ejecuta si el contrato solo recibe ether mediante una transferencia normal, sin que se haya llamado a ninguna función payable (recordemos que un contrato al fin y al cabo es una cuenta, y puede recibir transferencias ordinarias).
- **fallback()** external payable: Si la función que se ha llamado no está definida en el contrato.

Por defecto, estas funciones no están definidas, sin embargo, si se requiere implementar alguna lógica específica en caso de que se produzca alguna de las situaciones mencionadas anteriormente, basta con definir la función con el nombre y los parámetros especificados previamente.

```
1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity >=0.7.0 <0.9.0;
3
4 contract Contrato {
5     // Se ejecuta cuando recibe ether a través de una transferencia ordinaria.
6     receive() external payable {
7     }
8
9     // Se ejecuta cuando se llama a una función inexistente.
10    fallback() external payable {
11    }
12 }
```

Bloque de Código 15: Ejemplo del uso de funciones por defecto en Solidity.



Transacciones

En Solidity tenemos 3 maneras de enviar ether desde nuestro contrato a un usuario u otro contrato.

- Send: devuelve false si falla, gas NO ajustable.
- Transfer: revierte si falla, gas NO ajustable.
- Call{value}(""): devuelve false si falla, gas ajustable.

```
1  function enviarEther() public {
2      bool success = payable(msg.sender).send(1 ether);
3      require(success);
4  }
5
6  function enviarEther() public {
7      payable(msg.sender).transfer (1 ether);
8  }
9
10 function enviarEther() public {
11     // Modo 1, especificando el gas m ximo a usar.
12     (bool success1, ) = msg.sender.call{value: 1 ether, gas: 500000}("");
13
14     // Modo 2, sin especificar el gas.
15     (bool success2, ) = msg.sender.call{value: 1 ether }("");
16
17     require(success1 && success2);
18 }
```

Bloque de Código 16: Ejemplo de transferencia de Ether en Solidity.



Llamadas Externas

Como ya se ha mencionado, un contrato puede llamar a otro contrato (de una función a otra). Para hacer esto tenemos 2 maneras:

- Usando el código del contrato, o una Interfaz.

```
1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity >=0.7.0 <0.9.0;
3
4 contract ContratoExterno {
5     function saludo (string memory nombre) external pure returns (string memory)
6     {
7         return string(abi.encodePacked("Hola", bytes(nombre)));
8     }
9 }
10
11 contract Contrato {
12     address direccionContratoExterno;
13     constructor(address _dirContExterno) {
14         direccionContratoExterno = _dirContExterno;
15     }
16     function saludar(string memory nombre) public payable returns (string memory)
17     {
18         return ContratoExterno(direccionContratoExterno).saludo(nombre);
19     }
}
```

Bloque de Código 17: Ejemplo de llamada externa usando el código del contrato destinatario en Solidity.



- Especificando el método en la llamada, al especificar el método, “abi.encodeWithSignature” genera el identificador de la función para el contrato a llamar.

```
1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity >=0.7.0 <0.9.0;
3
4 contract ContratoExterno {
5     function saludo(string memory nombre) external pure returns(string memory) {
6         return string (bytes.concat("Hola ", bytes (nombre)));
7     }
8 }
9
10 contract Contrato {
11     address direccionContratoExterno;
12     constructor(address _dirContExterno) {
13         direccionContratoExterno = _dirContExterno;
14     }
15
16     function saludar(string memory nombre) public payable returns (string memory)
17     {
18         // .call devuelve (bool, bytes).
19         // Si la llamada ha sido una transaccion ordinaria de ether, bool se debe
20         // comprobar, bytes son datos sobre la transferencia.
21         // Si la llamada ha sido una llamada a otra funcion de otro contrato,
22         // bytes almacenar los datos devueltos.
23         (bool success, bytes memory data) = direccionContratoExterno.call(abi.
24             encodeWithSignature("saludo(string)", nombre));
25     }
26 }
```

Bloque de Código 18: Ejemplo de llamada externa usando el identificador de la función destinataria en Solidity.



Eventos

Los eventos sirven para registrar un historial de acciones que suceden en el contrato para poder acceder a ellos posteriormente de forma cómoda. Se suele usar para indexar datos en la blockchain y poder leerlos sin tener que buscar bloque a bloque usando un explorador de bloques[22].

```
1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity >=0.7.0 <0.9.0;
3
4 contract Contrato {
5
6     address direccionContratoExterno;
7
8     constructor(address _dirContExterno) {
9         direccionContratoExterno = _dirContExterno;
10    }
11
12     // Definimos la estructura del evento ( los datos a loggear ).
13     event Tranferencia(address indexed destino, uint etherEnviado);
14
15     function enviarEther() public {
16         // Enviamos 1 ether del contrato al usuario que llama la función.
17         (bool success,) = msg.sender.call{value: 1 ether }("");
18         require(success);
19
20         // Emitimos el evento.
21         emit Tranferencia (msg.sender, 1 ether);
22     }
23 }
```

Bloque de Código 19: Ejemplo del uso y definición de eventos en Solidity.

Optimización del gas

Debido a que en la EVM, las variables de Storage se guardan en ranura de 32 bytes, al declarar las variables de estado se pueden empaquetar en ranuras de 32 bytes para optimizar el acceso a dicho pack, ya que cuando se lee o guarda una variable de estado, se lee la ranura entera de 32 bytes, y se accede a cada posición individualmente.

Empaquetar tus variables significa que empaquetas o juntas variables de menor tamaño para que en conjunto formen 32 bytes.

Por ejemplo, puede empaquetar 32 uint8 en una ranura de almacenamiento, pero para que eso suceda, es importante que se declaren consecutivamente porque el orden de declaración de las variables importa en Solidity.



Librerías

Las librerías en Solidity son bloques de código reutilizables que se pueden integrar en diferentes contratos inteligentes. Las bibliotecas son útiles porque permiten evitar la duplicación de código y mejoran la legibilidad y la organización del código.

SafeMath es una biblioteca popular en Solidity que proporciona funciones matemáticas seguras para evitar errores comunes, como desbordamientos y subdesbordamientos.

Aquí hay un ejemplo sencillo de un contrato inteligente que utiliza la biblioteca SafeMath:

```
1 pragma solidity ^0.8.0;
2 import "https://github.com/OpenZeppelin/openzeppelin-solidity/contracts/math/
3   SafeMath.sol";
4
5 contract SimpleContract {
6     using SafeMath for uint256;
7
8     uint256 public balance;
9
10    function deposit(uint256 _value) public {
11        balance = balance.add(_value);
12    }
13
14    function withdraw(uint256 _value) public {
15        require(balance >= _value, "Insufficient balance.");
16        balance = balance.sub(_value);
17    }
}
```

Bloque de Código 20: Ejemplo del uso de la librería SafeMath en Solidity.

En este ejemplo, importamos la biblioteca SafeMath desde GitHub y la hacemos disponible en el contrato inteligente utilizando la sentencia `using SafeMath for uint256;`. Luego, utilizamos las funciones `add` y `sub` de SafeMath para realizar operaciones matemáticas seguras en las funciones `deposit` y `withdraw`.

3.2.13. Estandards

Los estándares son convenios técnicos utilizados para definir la funcionalidad de los tokens en la cadena de bloques Ethereum. Estos estándares establecen un conjunto de reglas para la creación, emisión y transferencia de tokens en la red Ethereum. Los estándares más populares incluyen **ERC-20**¹⁵, **ERC-721**¹⁶ y **ERC-1155**¹⁷.

3.2.14. OpenZeppelin

OpenZeppelin¹⁸ es una organización que provee estandars y una biblioteca de contratos inteligentes de código abierto y seguro para Ethereum, que proporciona herramientas y soluciones comunes para desarrolladores de contratos inteligentes. OpenZeppelin incluye una amplia gama de contratos inteligentes preconstruidos y estandares de tokens, que se pueden utilizar como una base sólida para la construcción de nuevos contratos inteligentes.

¹⁵<https://eips.ethereum.org/EIPS/eip-20>

¹⁶<https://eips.ethereum.org/EIPS/eip-721>

¹⁷<https://eips.ethereum.org/EIPS/eip-1155>

¹⁸<https://www.openzeppelin.com/>



3.2.14.1 ERC-20 - Tokens

Un token ERC-20 es un tipo de token en la plataforma Ethereum que se adhiere al estándar ERC-20. ERC-20 es un estándar técnico que describe cómo se deben crear, emitir y transferir tokens en la red Ethereum.

El estándar ERC-20 incluye una serie de funciones y eventos que deben implementarse en un contrato inteligente para que sea considerado un token ERC-20. Algunos de los requisitos incluyen la implementación de funciones para obtener la información sobre el token, como el nombre, la simbología, el número total de tokens emitidos, etc.

3.2.15. ABIs

Los compiladores de Solidity también generan información adicional, como la ABI (Interfaz de Programación de Aplicaciones), que se utiliza para interactuar con el contrato inteligente a través de una aplicación o una interfaz.

3.2.16. Dapps

Las DAPPS (aplicaciones descentralizadas) son aplicaciones que funcionan con contratos inteligentes para llevar a cabo sus tareas.

Estas se construyen utilizando tecnologías blockchain como Ethereum, que les permiten tener una base de datos distribuida y un sistema de consenso descentralizado que asegura la integridad y seguridad de los datos.

Esto significa que las DAPPS son resistentes a la manipulación y la censura, y se les considera más seguras y privadas que las aplicaciones tradicionales que dependen de

3.2.17. Sistemas

En Ethereum, se pueden desarrollar diferentes tipos de sistemas usando, contratos inteligentes, estandares, librerias, etc ...

Los más comunes en la red suelen ser los siguientes:

3.2.17.1 ICO

Una ICO (Oferta Inicial de Monedas) es un tipo de financiación de proyectos en el mundo de las criptomonedas y la tecnología blockchain. Se trata de una oferta de venta de tokens digitales a inversores en una etapa temprana del proyecto, con el objetivo de recaudar fondos para su desarrollo.

Los inversores que participan en una ICO compran tokens con una criptomoneda establecida, como Bitcoin o Ethereum, o con dinero fiat. Luego, pueden mantener esos tokens o venderlos en un intercambio de criptomonedas cuando su valor aumente.

Las ICO se han utilizado para financiar una amplia variedad de proyectos en el mundo de las criptomonedas, desde nuevas criptomonedas hasta plataformas descentralizadas y aplicaciones descentralizadas (DAPPs).



3.2.17.2 DAO

Una DAO (Organización Descentralizada Autónoma) es un tipo de organización que se basa en la tecnología blockchain y utiliza contratos inteligentes para tomar decisiones y ejecutar acciones.

Estos sistemas compuestos por smart contracts son descentralizadas y autónomas, lo que significa que no tienen una estructura jerárquica centralizada y que sus decisiones y acciones son tomadas y ejecutadas automáticamente por sus contratos inteligentes, sin la necesidad de una intermediación humana.

Los participantes tienen derecho a voto en proporción a la cantidad de tokens que poseen. Estos tokens se adquieren mediante una inversión en la DAO y representan una participación en la organización.

3.2.17.3 DEX

Una DEX (Bolsa Descentralizada) es un tipo de plataforma de intercambio de criptomonedas que utiliza la tecnología blockchain y contratos inteligentes para permitir la negociación peer-to-peer de activos digitales.

A diferencia de las bolsas centralizadas, que tienen una entidad central que controla los activos y las operaciones de los usuarios, las DEX son descentralizadas y no dependen de intermediarios centralizados. Esto significa que los usuarios tienen un mayor control sobre sus activos y sus operaciones, y que no hay una sola entidad que tenga acceso a los fondos de los usuarios.

Las DEX funcionan mediante la utilización de contratos inteligentes en una blockchain, que se encargan de gestionar las órdenes de compra y venta y de ejecutar automáticamente las transacciones.

Estos contratos inteligentes también garantizan la seguridad y la transparencia de las operaciones, ya que se ejecutan en una blockchain descentralizada.



3.3. IPFS

IPFS¹⁹ es el acrónimo de Inter Planetary File System y es un sistema de archivos distribuido que utiliza la tecnología de blockchain. A diferencia de la tecnología de archivos convencional que utiliza servidores centralizados, IPFS utiliza una red descentralizada de nodos para almacenar y compartir archivos.

Esto significa que, en lugar de alojar archivos en un servidor centralizado, los archivos son fragmentados en pequeñas piezas y almacenados en múltiples nodos en la red IPFS.

Cuando alguien quiere acceder a un archivo, se descarga de los nodos que lo tienen, en lugar de un servidor central y cada archivo en la red se identifica mediante un hash único generado por una función de hash criptográfica, y a los cuales se puede acceder usando una dirección URL.

IPFS es una tecnología de blockchain porque utiliza la criptografía para garantizar la integridad y la autenticidad de los archivos en la red. Además, los archivos se identifican mediante una dirección única y permanente generada por un hash criptográfico, lo que significa que los archivos no pueden ser modificados o alterados sin ser detectados.

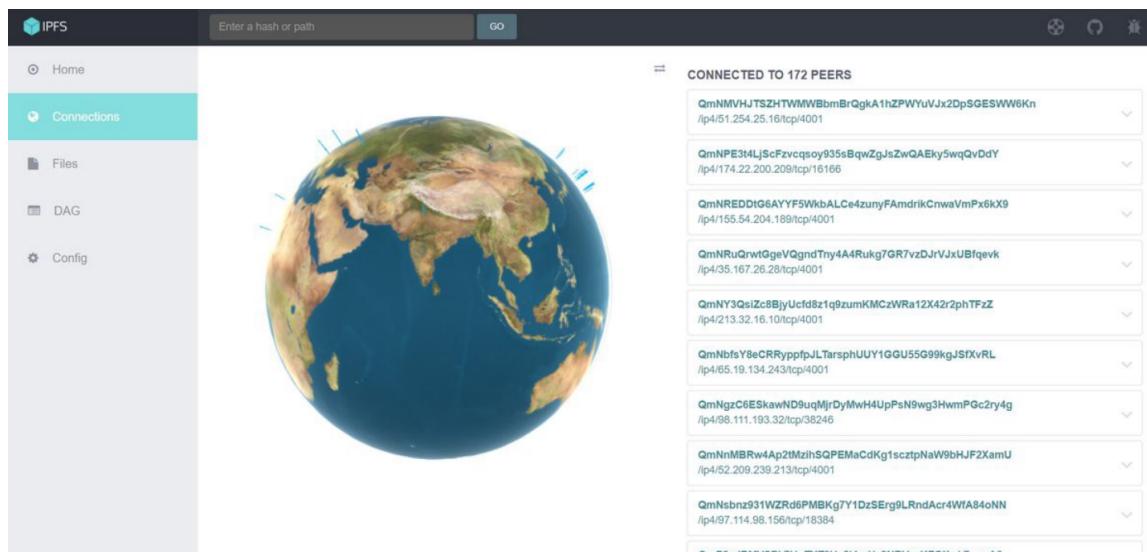


Figura 11: Panel de control del Cliente IPFS.

En este trabajo se usa IPFS como un medio donde almacenar imágenes y videos asociados a las propuestas de proyectos.

¹⁹<https://ipfs.tech/>



3.4. EtherJS

Ethers²⁰ es una librería de JavaScript que permite la conexión de una aplicación desarrollada en JS a cualquier blockchain que esté usando el protocolo Ethereum.

```
1  /** Connecting to Ethereum: MetaMask
2
3      The quickest and easiest way to experiment and begin developing on Ethereum is
4      to use MetaMask, which is a browser extension that provides:
5
6          A connection to the Ethereum network (a Provider)
7          Holds your private key and can sign things (a Signer) */
8
9 // A Web3Provider wraps a standard Web3 provider, which is
10 // what MetaMask injects as window.ethereum into each page
11 const provider = new ethers.providers.Web3Provider(window.ethereum)
12
13 // MetaMask requires requesting permission to connect users accounts
14 await provider.send("eth_requestAccounts", []);
15
16 // The MetaMask plugin also allows signing transactions to
17 // send ether and pay to change state within the blockchain.
18 // For this, you need the account signer...
19 const signer = provider.getSigner()
```

Bloque de Código 21: Ejemplo de conexión a una cartera Metamask usando Ethers.

La librería provee de un conector dada la dirección de un nodo de la red, y de métodos para interactuar con los smart contracts mediante el uso de una ABI y un firmante mediante (cuenta de la blockchain) una cartera.

3.5. Svelte

Svelte²¹ es un framework de desarrollo web basado en JavaScript que facilita la creación de interfaces de usuario. A diferencia de otros frameworks populares como React o Vue, Svelte se enfoca en compilar los componentes durante el proceso de construcción en lugar de ejecutarlos en tiempo de ejecución en el navegador del cliente, esto resulta en un rendimiento superior y una menor carga en los recursos del navegador.

Los componentes de Svelte están escritos en un archivo “.svelte” que combina el HTML, CSS y JavaScript necesarios para el componente. Esto permite una mejor separación de responsabilidades y una mayor legibilidad del código.

```
1 <script>
2     let name = 'world';
3 </script>
4
5 <h1>Hello {name}!</h1>
```

Bloque de Código 22: Hello World en Svelte.

²⁰<https://docs.ethers.org/v5/>

²¹<https://svelte.dev/>



Para este trabajo se ha seleccionado este framework por su realmente reactividad, y sus patrón basado en componente para desarollar las vistas del front-end.

3.6. HardHat

Hardhat²² es un entorno de desarrollo de smart contracts desarrollado en JavaScript, este nos permite compilar código en Solidity, desplegar nodos con cuentas de prueba localmente, para testear nuestros contratos y finalmente desplegarlos.

```
$ npx hardhat
 888   888           888 888           888
 888   888           888 888           888
 888   888           888 888           888
888888888888 8888b. 888d888 .d88888 88888b. 8888b. 8888888
 888   888     "88b 888P" d88" 888 888 "88b     "88b 888
 888   888 .d888888 888   888 888 888 888 .d888888 888
 888   888 888 888   Y88b 888 888 888 888 888 888 Y88b.
 888   888 "Y888888 888     "Y888888 888 888 "Y888888  "Y888

 专员 Welcome to Hardhat v2.2.1 专员

? What do you want to do? ...
  Create a sample project
  > Create an empty hardhat.config.js
  Quit
```

Figura 12: Console del cliente de HardHat.

Es altamente configurable además de soportar tests unitarios para nuestros contratos. Se pueden programar “tasks” para automatizar ciertos procesos como la verificación de contratos en los block explorers como Etherscan.

²²<https://hardhat.org/>



3.7. ThirdWeb

ThirdWeb²³ es un framework de desarrollo de DApps, con un amplio abanico de librerías que permiten facilitar la integración de funcionalidades basadas en blockchain en aplicaciones desarrolladas en JavaScript.

De este framework se destaca la librería “thirdweb-dev/storage” que provee de la funcionalidad de conectarse mediante JavaScript a la blockchain IPFS, subir documentos, y obtener su hash único.

```
1 import { ThirdwebStorage } from "@thirdweb-dev/storage";
2
3 // First, instantiate the SDK
4 const storage = new ThirdwebStorage();
5
6 // Now we can upload a file and get the upload URI
7 const file = readFileSync("path/to/file.jpg");
8 const uri = await storage.upload(file);
9
10 // Finally we can download the file data again
11 const res = await storage.download(uri);
12 const data = await res.text();
```

Bloque de Código 23: Ejemplo de la conexión y subida de un archivo a IPFS.

3.8. Bootstrap

Bootstrap²⁴ es un framework de diseño web de código abierto que proporciona una colección de herramientas, componentes y estilos para ayudar a los desarrolladores a crear sitios y interfaces responsive²⁵.

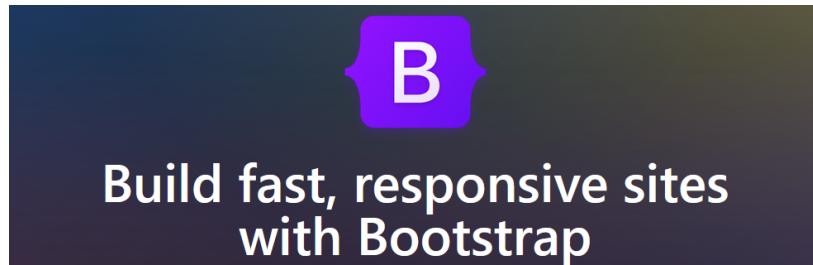


Figura 13: Logotipo y slogan de Bootstrap.

Bootstrap está basado en HTML, CSS y JavaScript y permite a los desarrolladores crear diseños personalizados, así como utilizar estilos predefinidos, también incluye una cuadrícula de diseño flexible que permite a los desarrolladores crear diseños complejos y alinear fácilmente.

²³<https://thirdweb.com/>

²⁴<https://getbootstrap.com/>

²⁵El término se refiere a la capacidad de un sitio web para adaptarse a diferentes tamaños de pantalla, desde pantallas pequeñas de dispositivos móviles hasta pantallas grandes de escritorio.

3.9. MetaMask

MetaMask²⁶ es una cartera que nos permite gestionar nuestra cuenta de Ethereum, pero que además existe en formato de extensión de navegador, esto hace más seguro y fácil desarrollar e interactuar con interfaces web que conectan con un contrato.

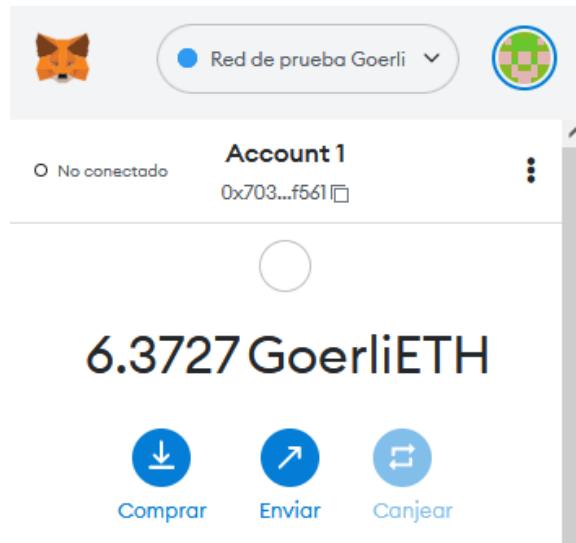


Figura 14: Captura de la cartera de MetaMask.

MetaMask provee al navegador de una variable en JS “window.ethereum”, la cual después es usada por librerías como Ethers para interactuar con la blockchain desde las DApps.

Esta variable, posee entre otros, un objeto que representa al firmante (la cuenta ethereum del usuario) y la conexión al nodo incluyendo la dirección JSONRPC, estos datos son fundamentales para poder interactuar con los smart contracts.

²⁶<https://metamask.io/>



3.10. Docker

Docker²⁷ es una plataforma de software que permite a los desarrolladores crear, ejecutar y distribuir aplicaciones en contenedores. Los contenedores son unidades de software portátiles y livianas que contienen todas las dependencias y configuraciones necesarias para que una aplicación se ejecute de manera eficiente y confiable, independientemente del entorno en el que se esté ejecutando.

The screenshot shows the Portainer.io interface. On the left, there's a sidebar with navigation links like Home, Dashboard, App Templates, Stacks, Containers (which is selected), Images, Networks, Volumes, Events, Host, Settings, Users, Environments, Registries, Authentication logs, Notifications, and Settings. The main area is titled 'Container list' and shows a table of running containers. The columns are: Name, State, Filter, Quick Actions, Stack, Image, Created, IP Address, GPUs, and Published Ports. The table contains the following data:

Name	State	Filter	Quick Actions	Stack	Image	Created	IP Address	GPUs	Published Ports
FAIL2BAN_MONITOR	exited			-	vexium/fail2banmonitor:latest	2022-12-19 10:31:46	-	-	
FTP_SERVER	running			-	defleratine/ftp-server:latest	2022-12-24 02:10:44	172.17.0.3	-	none 21004:21004 21008:21008 21121:21121 21000:21000 21001:21001 21002:21002
INFLUXDB_SERVER	exited			-	influxdb:latest	2022-12-16 23:49:54	-	-	
NGINX_PROXY_MANAGER	healthy			-	jc21/nginx-proxy-manager:latest	2022-12-22 12:47:14	172.17.0.2	-	none 443:443 80:80
NODERED_SERVER	exited			-	node-red/node-red:latest	2022-12-11 05:32:48	-	-	
OWNCLOUD_SERVER	running			-	nextcloud:latest	2022-01-25 01:06:05	172.17.0.4	-	
PLEX_SERVER	running			-	plex:latest	2022-12-24 02:41:21	172.17.0.5	-	
POCKETBASE_SERVER	healthy			-	ghcr.io/muchobrain/pocketbase:latest	2022-12-24 18:17:51	172.18.0.10	-	
PORTAINER_INTERFACE	running			-	portainer/portainer-ce:latest	2022-12-10 20:53:39	172.18.0.9	-	
VSCODE_SERVER	exited			-	codercom/code-server:latest	2022-12-16 23:49:58	-	-	

Figura 15: Captura del panel de control de Portainer.

En este trabajo, se usará Docker para desplegar el front-end en un servidor, una vez haya finalizado el proyecto. También usaremos Portainer²⁸, un contenedor que nos provee de una aplicación web para gestionar los contenedores de nuestro servidor.

²⁷<https://www.docker.com/>

²⁸<https://www.portainer.io/>

4. Modelo del Sistema

Previamente, en los capítulos anteriores se describe el problema que se intenta abordar, y una propuesta de solución así dando lugar a la propuesta del desarrollando un sistema de gestión descentralizado.

En el presente capítulo, se abordarán con mayor profundidad las funcionalidades que incorpora el sistema propuesto, además de detallar los requerimientos del sistema, diagramas representativos del flujo de trabajo del sistema, mockups de la interfaz que permitirá interactuar con los smart contracts y una interfaz del código del smart contract.

4.1. Agentes

En este apartado se muestra los criterios y características de los 4 roles que un usuario podría tomar dentro del sistema.

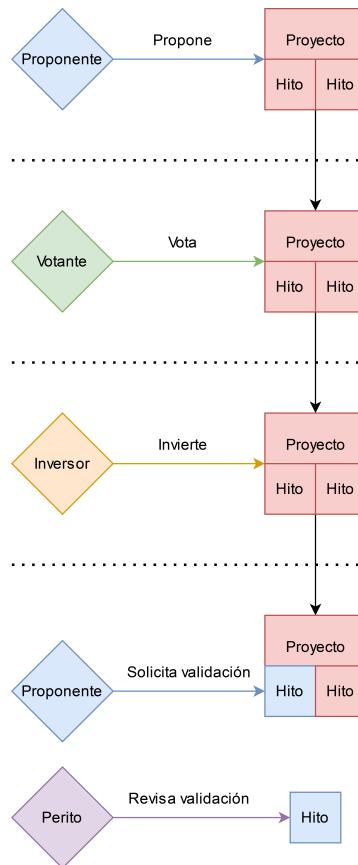


Figura 16: Diagrama de Agentes - Interacción de los agentes en el sistema a lo largo de un proyecto.

4.1.1. Proponente

Los “proponentes” en este sistema son aquellos que presentan una propuesta detallada de proyecto a la comunidad y se encargan de cumplir con los hitos definidos para que el proyecto sea validado por un perito, y reciba la financiación necesaria para su desarrollo. Para que un usuario pueda proponer un proyecto, este deberá identificarse con un documento legal valido.

Además, el Proponente debe cumplir con las normas y estándares de la plataforma, y trabajar en conjunto con la comunidad y el sistema de gestión autónomo para asegurar el éxito del proyecto.

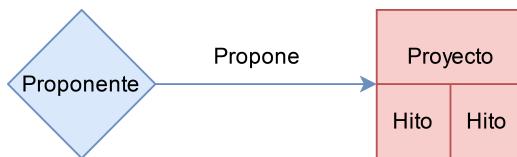


Figura 17: Diagrama de Agentes - Interacción de un proponente con un proyecto.

4.1.2. Votantes

Los “votantes” de este sistema es un participante activo en la toma de decisiones sobre qué proyectos se deberían financiar y cuáles no. Los votantes deciden si un proyecto debe pasar a la fase de recaudación.

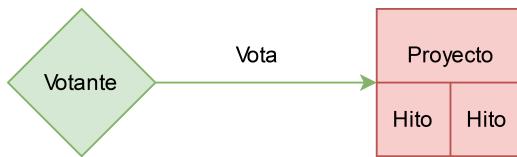


Figura 18: Diagrama de Agentes - Interacción de un votante con un proyecto.

4.1.3. Inversor

Los “Inversores” son los usuarios que financian los proyectos que han sido aprobados por la comunidad con la garantía de que parte de su inversión está protegido en caso de que el proyecto no se lleve a cabo total o parcialmente.



Figura 19: Diagrama de Agentes - Interacción de un inversor con un proyecto.

4.1.4. Perito

Los “Peritos” son usuarios expertos en ciertos campos relacionados con la naturaleza de los proyectos que se financian en la plataforma. Un usuario para ser perito debe poseer la acreditación necesaria de la especialización en el campo en el que dice especializarse además de una identificación personal. Los peritos se encargan de revisar y validar que un hito se ha cumplido realmente.

Para validar que una acreditación es real y está en conformidad con la normativa del centro que lo ha expedido, este deberá seguir el standard “Verifiable Credentials Data Model”[31] de W3C.

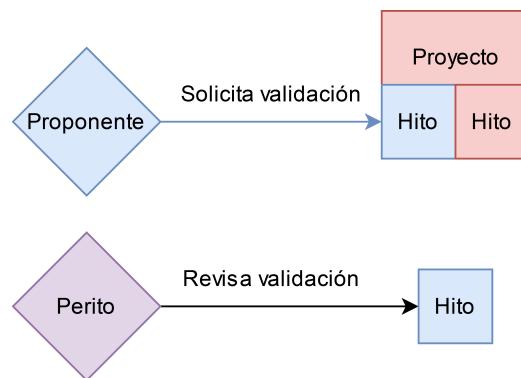


Figura 20: Diagrama de Agentes - Interacción de un perito con un proyecto.



4.2. Requerimientos

Estas especificaciones se detallan en 3 tablas, la primera para los requerimientos funcionales y segunda para los requerimientos no funcionales además de mencionar en una tercera tabla los requerimientos que se no se implementarán en la prueba de concepto.

4.2.1. Requerimientos Funcionales

ID	Requerimiento	Descripción
RF1	Autenticación	Los usuarios podrán autenticarse en el sistema mediante su cartera.
RF2	Notificación de red equivocada	Se le notificará al usuario si se ha conectado desde una red blockchain donde el contrato no está desplegado.
RF3	Notificación de cartera no detectada	Se le notificará al usuario de que no posee ninguna cartera conectada al navegador.
RF4	Propuesta de proyectos	Los usuarios pueden proponer nuevos proyectos.
RF5	Listado de proyectos	Los usuarios podrán ver todos los proyectos en la página principal de la web.
RF6	Añadir comentarios	Los usuarios pueden realizar comentarios sobre un proyecto en la página del proyecto.
RF7	Leer comentarios	Los usuarios pueden leer los comentarios sobre un proyecto en la página del proyecto.
RF8	Votación de proyectos	Los usuarios pueden votar para aprobar o rechazar proyectos propuestos.
RF9	Aprobación del proyecto	Un proyecto es aprobado si alcanza al menos el 85 % de votos positivos en un plazo de 30 días desde su propuesta. De lo contrario, se rechaza.
RF10	Financiación del proyecto	Los usuarios pueden invertir en proyectos aprobados.
RF11	Solicitud de Validación de hitos	El proponente solicitará a la comunidad validar el siguiente hito.
RF12	Validación de hitos	Los peritos validarán una solicitud, así confirmando o denegando si se ha alcanzado el objetivo del hito.
RF13	Liberación de fondos	Permite al proponente obtener los fondos asignados de un hito validado.
RF14	Subsanación de hitos	Si la validación de un hito acaba siendo denegada por la comunidad, el proponente tiene 10 días para abordar los problemas propuestos, y solicitar nuevamente la validación del hito.
RF15	Cancelación de proyecto	Un proyecto se cancela automáticamente si muestra signos de estancamiento en un hito, o no el proponente deja de comunicarse.
RF16	Procesado de archivos	El sistema tendrá una funcionalidad en el frontend capaz de subir archivo a IPFS para su publicación en dicha blockchain.

Cuadro 2: Análisis y Diseño - Requerimientos funcionales del sistema.



4.2.2. Requerimientos no Funcionales

ID	Requerimiento	Descripción
RNF1	Seguridad	El sistema debe garantizar la seguridad de los fondos y los datos, evitando ataques y accesos no autorizados.
RNF2	Escalabilidad	El sistema debe ser capaz de manejar un creciente número de usuarios y proyectos sin degradar el rendimiento.
RNF3	Interoperabilidad	El sistema debe ser compatible con diversas plataformas y tecnologías blockchain, permitiendo la interacción con otros sistemas y servicios.
RNF4	Eficiencia	El sistema debe optimizar el uso de recursos y reducir costos operativos, incluyendo tarifas de transacción en la blockchain.
RNF5	Usabilidad	La interfaz de usuario debe ser intuitiva y fácil de usar para permitir a usuarios de diversos niveles de experiencia interactuar con el sistema.
RNF6	Transparencia	El sistema debe proporcionar a los usuarios información clara y accesible sobre el proceso de votación, financiación y validación de hitos.
RNF7	Confiabilidad	El sistema debe ser robusto y resistente a fallos, asegurando la disponibilidad continua del servicio.
RNF8	Mantenibilidad	El diseño del sistema debe facilitar la actualización, mejora y corrección de errores en el futuro.
RNF9	Legalidad	El sistema debe cumplir con las regulaciones y leyes aplicables en las jurisdicciones en las que opera.
RNF10	Privacidad	El sistema debe proteger la privacidad de los usuarios, almacenando y gestionando sus datos de manera segura y de acuerdo con las leyes de protección de datos.
RNF11	Trazabilidad y control	El sistema garantiza la trazabilidad y el control de los fondos y activos de la plataforma.
RNF12	Inmutabilidad	Los datos del proyecto, una vez aprobados, no pueden ser eliminados ni modificados.
RNF13	Sistema basado en Roles	Los agentes se distribuirán usando el estandard de roles access-control de OpenZeppelin.

Cuadro 3: Análisis y Diseño - Requerimientos no funcionales del sistema.



4.2.3. Requisitos excluidos

En el capítulo número 7 de este documento se presenta la evaluación de la prueba de concepto correspondiente a la propuesta desarrollada. Dado que se trata únicamente de una prueba del concepto el sistema implementado que se llevara a cabo se considera como una muestra de la propuesta de solución, ya que el tiempo correspondiente al trabajo (12 ECTS - 300 Horas) no es el suficiente para realizar la implementación completa de la solución propuesta.

También teniendo en cuenta que esta solución es una prueba de concepto, se ha considerado no indagar en profundidad en los aspectos legales del tratamiento de datos con respecto a los agentes.

En consecuencia, en este apartado se enumeran los requerimientos que no se han llevado a cabo debido a la limitación de tiempo y a que sobrepasan de los límites demandados para este trabajo:

Requerimientos.
RF11
RF12
RF14
RF16
RNF9
RNF13

Cuadro 4: Análisis y Diseño - Requerimientos no implementados en la prueba de concepto.

4.3. Gobernanza

La gobernanza del sistema propuesto se basa en un enfoque descentralizado, aprovechando la tecnología blockchain para gestionar de manera eficiente y transparente la financiación de proyectos.

La plataforma opera mediante un mecanismo de gestión autónomo, lo que elimina la necesidad de un administrador central y reduce el riesgo de malversación de fondos.

Esto se debe gracias a que el protocolo Ethereum permite la ejecución de smart contracts, código personalizado que una vez desplegado es inalterable.

Esto nos permite definir sistemas que pueden ser autogobernables con una gobernanza regida por el código (code-as-law), sin necesidad de un actor administrador.

4.4. Diseños

A continuación se muestran los diseños del sistema, tales como los diagramas de flujo del sistema y los mockups de la interfaz.

4.4.1. Diagrama de Flujo

El diagrama a continuación representa de forma generalizada el flujo del sistema.

4.4.1.1 Proceso de contribución en un proyecto

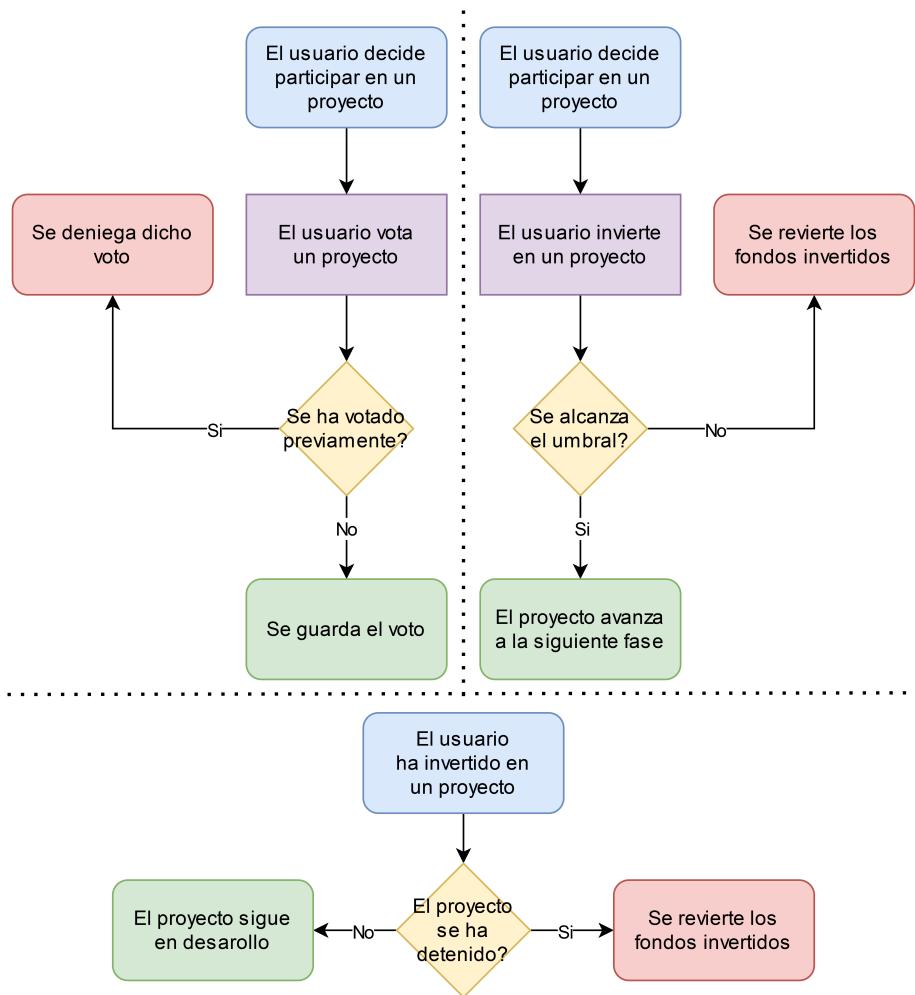


Figura 21: Diagrama de Flujo - Proceso de contribución con un proyecto.

4.4.1.2 Proceso de propuesta de un nuevo proyecto

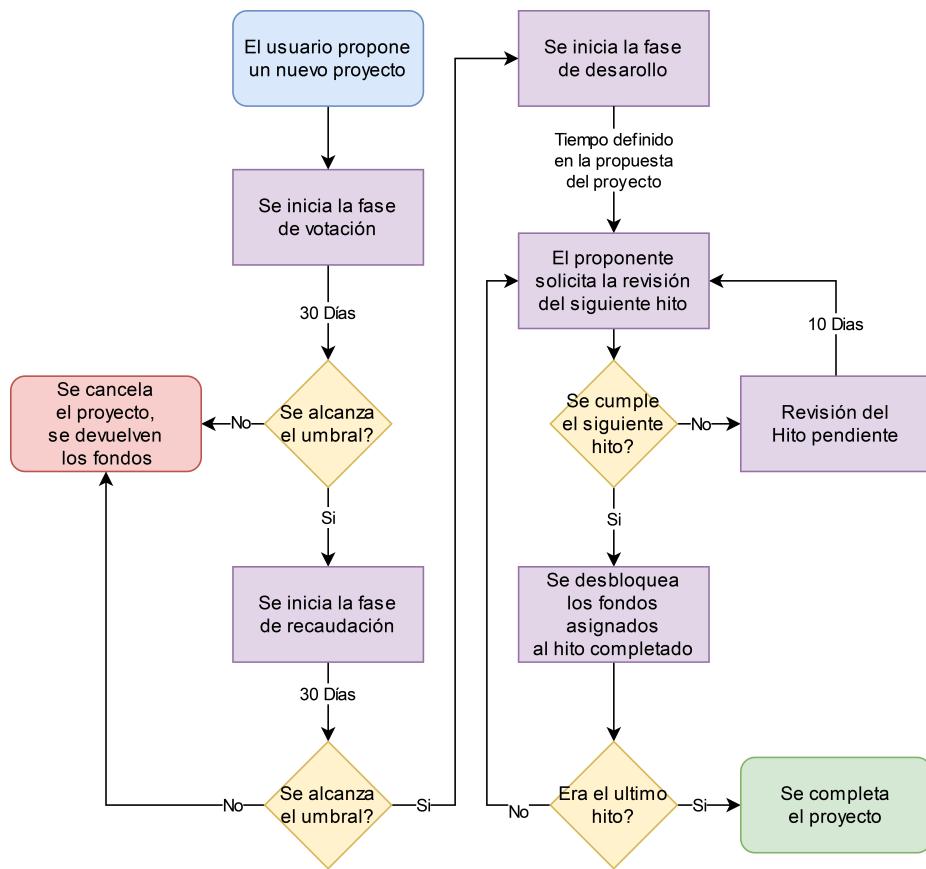


Figura 22: Diagrama de Flujo - Proceso de propuesta de proyecto.



4.4.2. Mockups

A continuación se mostrarán los mockups del diseño del de la interfaz (front-end) a desarrollar para la prueba de concepto.

Para abarcar los requerimientos funcionales, se ha desarrollado una interfaz donde cada pantalla provee la interacción necesaria para cumplir con los requerimientos a implementar en la prueba de concepto.

4.4.2.1 Pantalla de Registro

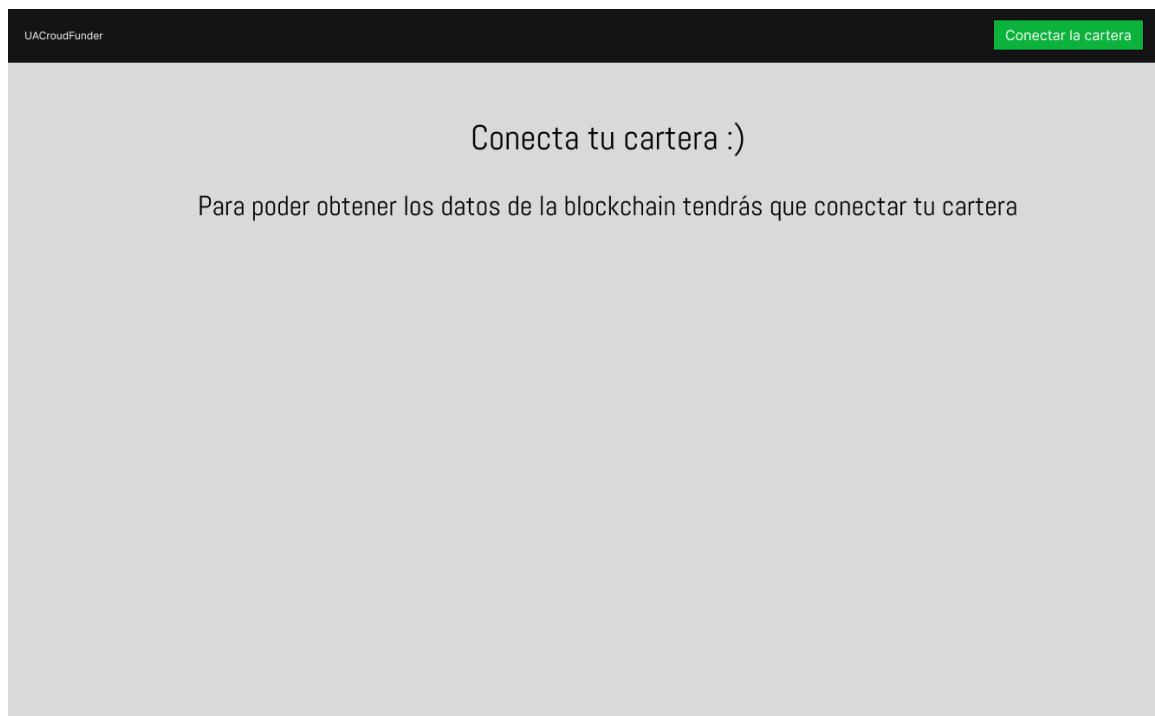


Figura 23: Mockup - Pantalla de Registro.

En esta pantalla se mostrará un mensaje de bienvenida, y un botón con la funcionalidad para conectar la cartera al front-end.

Requerimientos:

- RF1



4.4.2.2 Pantalla de notificación de red equivocada



Figura 24: Mockup - Notificación de red equivocada.

En caso de que el usuario esté conectado a una red diferente a donde está desplegado el contrato, se mostrará un mensaje de advertencia y la red a la que se debe de conectar.

Requerimientos:

- RF2



4.4.2.3 Pantalla de notificación de cartera inexistente



Figura 25: Mockup - Notificación de que la cartera no se ha detectado.

En caso de que el usuario se intente conectar pero no tenga ninguna cartera instalada en el navegador, se le notificará de la situación mediante un mensaje.

Requerimientos:

- RF3



4.4.2.4 Pantalla de la lista de Proyectos

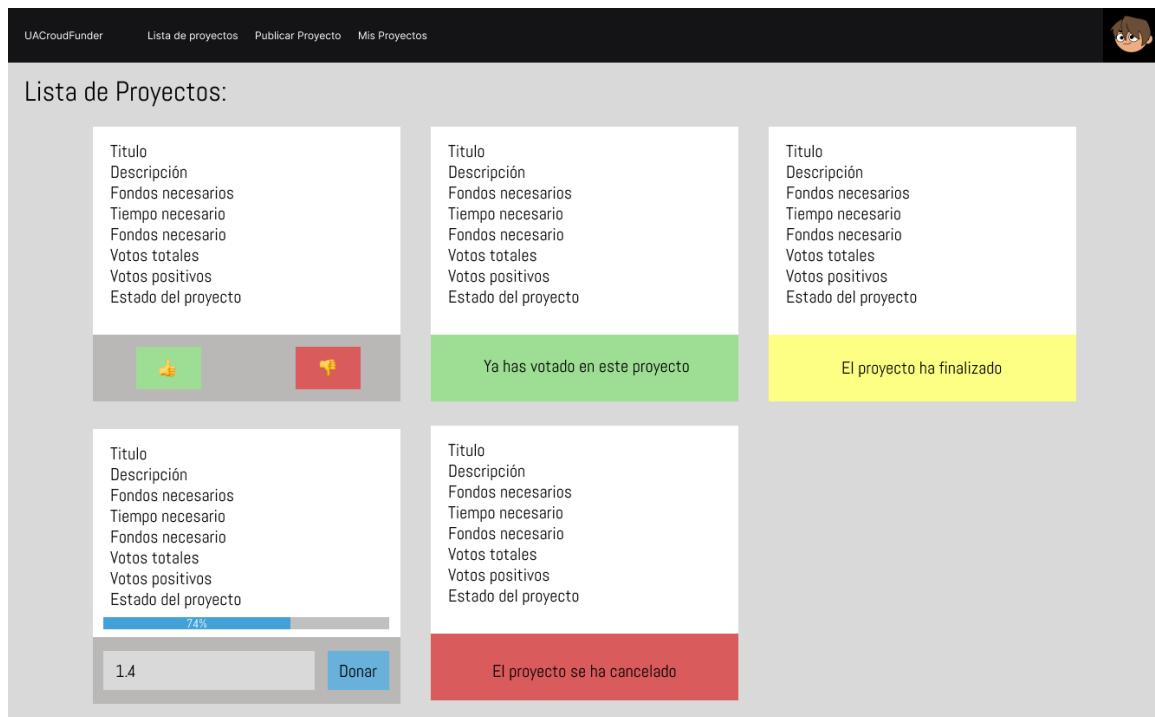


Figura 26: Mockup - Lista de Proyectos.

En la siguiente pantalla se mostrará una lista de los proyectos en forma de cards, donde se describirá sus detalles, además de poder votar y donar directamente.

Esta misma pantalla también servirá para filtrar por los proyectos propuestos por el propio usuario

Requerimientos:

- RF5
- RF8
- RF10



4.4.2.5 Pantalla del formulario para proponer un proyecto

The mockup shows a user interface for proposing a project. At the top, there is a navigation bar with links: 'UACloudFunder', 'Lista de proyectos', 'Publicar Proyecto', 'Mis Proyectos', and a user icon. Below the navigation bar, the main area has a light gray background. It contains four input fields with labels: 'Título' (Title), 'Descripción' (Description), 'Cantidad de fondos necesarios' (Amount of funds needed), and 'Tiempo necesario para el desarrollo' (Time required for development). Each input field is represented by a white rectangular box. Below these fields is a blue button labeled 'Proponer Poryecto' (Propose Project).

Figura 27: Mockup - Formulario para proponer un proyecto.

En esta pantalla se le permitirá al usuario proponer un nuevo proyecto, rellenando los datos necesarios.

Requerimientos:

- RF4



4.4.2.6 Pantalla de detalles de un proyecto

The mockup shows a user interface for a project detail page. At the top, there is a navigation bar with links: 'UACloudFunder', 'Lista de proyectos', 'Publicar Proyecto', 'Mis Proyectos', and a user icon. Below the navigation bar, the main content area is titled 'Detalles del Proyecto:'.

The 'Detalles del Proyecto:' section contains several input fields:

- Título
- Descripción
- Cantidad de fondos necesarios
- Cantidad de fondos recaudados
- Tiempo necesario para el desarrollo
- Fecha Límite
- Estado

Below these fields is a button labeled 'Fondos a donar:' followed by a progress bar and a 'Donar' button.

The 'Comentarios:' section contains three comment entries, each with a 'Dirección del comentante', 'Comentario', and 'Fecha'. Below this is a placeholder for a new comment entry: 'Añadir comentario' followed by a text area containing placeholder text 'Lorem Ipsum'.

At the bottom left of the main content area is another 'Donar' button.

Figura 28: Mockup - Detalles de un proyecto.

Y en esta pantalla se mostrará en detalle los datos de un proyecto, además de las funcionalidades de votar, donar, leer y comentar sobre sobre el proyecto.

Requerimientos:

- RF6
- RF7
- RF8
- RF10

4.4.3. Modelo del smart contract

Para el desarrollo de este sistema, necesitaremos varias estructuras para gestionar los datos en el smart contract.

A continuación se muestra un esquema con las estructuras de datos de la solución propuesta:



Figura 29: Modelo - Estructura de los datos del smart contract.



4.4.3.1 Project

Esta estructura representa un proyecto y sus atributos, el smart contract poseerá un array de este tipo de datos, donde se almacenarán todos los datos necesarios.

La estructura alberga la dirección del proponente, el título, descripción del proyecto, además de los fondos requeridos en MATIC, los fondos obtenidos hasta el momento, el tiempo total necesario para su desarrollo, fecha límite para la siguiente fase, votos totales, votos totales positivos, mapping de quien ha votado.

Además, esta estructura poseerá un array de Media para almacenar fotos y videos, Comment para almacenar un historial de comentarios, Milestones para almacenar la lista de hitos y Investments para registrar un historial de quien ha invertido en el proyecto y cuánto.

4.4.3.2 Milestone

La estructura de Milestone representa un Hito, este poseerá un título, descripción, porcentaje del tiempo necesario del total del proyecto para su realización, verificador si el hito está pendiente de validación, y fecha en formato UNIX cuando se ha validado.

4.4.3.3 Comment

Esta estructura representa un comentario, y contiene la dirección del autor, el mensaje y la fecha en formato UNIX.

4.4.3.4 Investment

Esta estructura contiene los datos necesarios para representar una inversión, siendo sus parámetros la dirección de la cuenta del inversor, la cantidad invertida y la fecha en formato UNIX.

4.4.3.5 Media

Mediante esta estructura se representa un archivo multimedia subido a IPFS, la cual contiene el URL-HASH de IPFS del archivo subido y el tipo (foto / video).

4.4.3.6 Document

Esta estructura representa un documento subido a IPFS, se usa principalmente para gestionar los certificados de los peritos, y contiene la URL-HASH de IPFS del modelo de datos de credenciales verificables de W3C[31].



4.4.4. Arquitectura del smart contract

En este apartado se presenta una interfaz de los métodos y variables que se implementan en el smart contract de la prueba de concepto.

Variables y Structs

4.4.4.1 ProjectState

Descripción: Enumeración para representar el estado de un proyecto.

Valores del enumerador:

- PENDING: Estado pendiente de aprobación.
- FUNDING: Estado en proceso de financiamiento.
- FUNDED: Estado financiado completamente.
- CANCELLED: Estado cancelado.

4.4.4.2 Comment

Descripción: Estructura para representar un comentario en un proyecto.

Atributos:

- **author:** Dirección del autor del comentario (address).
- **message:** Texto del comentario (string).
- **date:** Fecha del comentario (uint).

4.4.4.3 Investment

Descripción: Estructura para representar una inversión en un proyecto.

Atributos:

- **investor:** Dirección del inversor (address).
- **amount:** Cantidad invertida (uint).
- **date:** Fecha de la inversión (uint).

4.4.4.4 Project

Descripción: Estructura para representar un proyecto en la plataforma.

Atributos:

- **proposer:** Dirección del proponente del proyecto (address).
- **title:** Título del proyecto (string).
- **description:** Descripción del proyecto (string).
- **requiredFunds:** Fondos necesarios para el proyecto en Ether (uint).



- **collectedFunds:** Fondos recaudados en Ether (uint).
- **totalDevelopmentTime:** Tiempo total de desarrollo del proyecto en días (uint).
- **deadline:** Fecha límite para la siguiente fase (votación, financiación o validación de hitos) (uint).
- **totalVotes:** Total votos recibidos (uint).
- **positiveVotes:** Total votos positivos recibidos (uint).
- **state:** Estado del proyecto (ProjectState).
- **investments:** Array para controlar las inversiones de cada dirección en el proyecto (Investment[]).
- **comments:** Array para almacenar los comentarios del proyecto (Comment[]).
- **hasVoted:** Mapping para controlar quién ha votado el proyecto (mapping(address =>bool)).



Interfaz de las Funciones

4.4.4.5 getProjectsCount

Descripción: Devuelve el total de proyectos actuales en el smart contract.

Parámetros: No tiene parámetros.

Retorno: Devuelve un uint256 que representa el número total de proyectos.

4.4.4.6 proposeProject

Descripción: Propone un nuevo proyecto para ser aprobado por la comunidad.

Parámetros:

- `_title`: Título del proyecto (string memory).
- `_description`: Descripción del proyecto (string memory).
- `_fundingGoal`: Cantidad de fondos (en Ether) necesarios para su formalización (uint).
- `_totalDevelopmentTime`: Tiempo total (días) para su desarrollo (uint).

Retorno: Devuelve el ID del proyecto (uint256).

4.4.4.7 getProject

Descripción: Devuelve un proyecto dado un ID.

Parámetros:

- `_projectId`: ID del proyecto.

Retorno: Devuelve una objeto con los siguientes elementos del proyecto:

- `proposer`: Dirección del proponente (address memory).
- `title`: Título del proyecto (string memory).
- `description`: Descripción del proyecto (string memory).
- `requiredFunds`: Cantidad de fondos (en Ether) necesarios para su formalización (uint).
- `collectedFunds`: Cantidad de fondos (en Ether) recaudados hasta el momento (uint).
- `totalDevelopmentTime`: Tiempo total (días) para su desarrollo (uint).
- `deadline`: Días restantes hasta la siguiente fase, dependiendo del estado (uint).
- `status`: Estado actual del proyecto (enumerado: PENDING FUNDING, FUNDED, CANCELLED).
- `totalVotes`: Contador de la cantidad de votos totales (uint).
- `positiveVotes`: Contador de la cantidad de votos positivos del total (uint).
- `userVoted`: Booleano que indica si el usuario que solicita los datos ha votado en dicho proyecto (bool).



4.4.4.8 getCommentsCount

Descripción: Devuelve la cantidad de comentarios de un proyecto dado un ID.

Parámetros:

- `_projectId`: ID del proyecto (uint).

Retorno: Devuelve un entero que representa la cantidad de comentarios (uint).

4.4.4.9 addComment

Descripción: Añade un comentario a un proyecto.

Parámetros:

- `_projectId`: ID del proyecto (uint).
- `_comment`: comentario a añadir (string memory).

Retorno: No devuelve ningún valor.

4.4.4.10 getComment

Descripción: Devuelve un comentario dado un ID de proyecto y un ID de comentario.

Parámetros:

- `_projectId`: ID del proyecto (uint).
- `_commentId`: ID del comentario (uint).

Retorno:

- `author`: Address del autor del comentario (address).
- `message`: Contenido del comentario (string memory).
- `date`: Fecha en formato UNIX de cuando se añadió el comentario (uint).

4.4.4.11 getComments

Descripción: Devuelve todos los comentarios de un proyecto.

Parámetros:

- `_projectId`: ID del proyecto (uint).

Retorno:

- `comments`: Array de Comment (Comment[]).

4.4.4.12 voteProject

Descripción: Permite a los miembros de la comunidad votar un proyecto propuesto.

Parámetros:

- `_projectId`: ID del proyecto (uint256).
- `_positiveVote`: Voto a favor o en contra del proyecto (bool).

Retorno: No devuelve ningún valor.



4.4.4.13 hasVoted

Descripción: Devuelve si el usuario ha votado un proyecto.

Parámetros:

- `_projectId`: ID del proyecto (uint).
- `_user`: Address de un usuario (address).

Retorno:

- `bool`: Booleano que indica si el usuario ha votado o no en el proyecto (bool).

4.4.4.14 checkProjectApproval

Descripción: Comprueba si un proyecto ha sido aprobado por la comunidad, en caso positivo se cambia el estado del proyecto.

Parámetros:

- `_projectId`: ID del proyecto (uint).

Retorno: No devuelve ningún valor.

4.4.4.15 investInProject

Descripción: Permite a los miembros de la comunidad financiar un proyecto aprobado. La cantidad invertida se obtienen desde los metadatos de la transacción.

Parámetros:

- `_projectId`: ID del proyecto (uint256).

Retorno: No devuelve ningún valor.

4.4.4.16 checkProjectFunding

Descripción: Comprueba si un proyecto ha sido financiado, en caso de ser así, cambia el estado de dicho proyecto y transfiere lo recaudado al proponente.

Parámetros:

- `_projectId`: ID del proyecto (uint256).

Retorno: No devuelve ningún valor.

4.4.4.17 withdrawInvestment

Descripción: Permite a los inversores retirar sus fondos si el proyecto se ha cancelado.

Parámetros:

- `_projectId`: ID del proyecto (uint256).

Retorno: No devuelve ningún valor.



5. Implementación y Evaluación de la prueba de concepto

Para el desarrollo de esta prueba de concepto, se ha pensado crear un smart contract que abarque todas las funcionalidades mencionadas en los apartados anteriores, y una interfaz web basada en pantallas, donde cada pantalla proveerá de la interoperabilidad necesaria para que el usuario pueda interactuar con el smart contract en la blockchain, y se le ha dado el nombre de *UACrowdFunding* al sistema desarrollado.

5.1. Implementaciones

5.1.1. Front-End (Interfaz Web)

La aplicación de Front-End se ha desarrollado con Svelte como se ha mencionado en el apartado de Metodología.

Se ha decidido desarrollar un Front-End realizando un sistema de tipo SPA, por su simpleza y fácil implementación con Svelte.

Las SPAs (Single Page Application) son páginas web que en vez de forzar al navegador cargar un archivo cada vez que se cambia de página dependiendo de la URL, estas (mediante JavaScript) reemplazan el contenido que está mostrando el navegador.

Esto da lugar a que la página web no se basará en URLs para navegar entre pantallas, sino que trabajará con estados, de tal forma que la página al cambiar de un estado a otro, esta redirigirá un contenido HTML dinámico distinto. Los estados en los que se divide esta SPA son 6, una por cada pantalla de la interfaz:

DESCONECTADO, CARTERA-INEXISTENTE, RED-EQUIVOCADA, LISTA-PROYECTOS, CREAR-PROYECTO, DETALLE-PROYECTO.

También cabe mencionar que la aplicación de Svelte usa de la librería EthersJS para conectarse con la cartera MetaMask embebida en el navegador, y así poder comunicarse con la red blockchain a través de la cuenta del usuario.



A continuación se muestran las pantallas implementadas en el Front-End basadas en los Mockups.

5.1.1.1 Pantalla de Registro



Figura 30: Interfaz Web - Pantalla de Registro.



5.1.1.2 Pantalla de notificación de red equivocada



Figura 31: Interfaz Web - Notificación de red equivocada.



5.1.1.3 Pantalla de notificación de cartera inexistente



Figura 32: Interfaz Web - Notificación de cartera inexistente.



5.1.1.4 Pantalla de la lista de Proyectos

The screenshot displays a web application for managing crowdfunding projects. At the top, there is a navigation bar with links: 'UACrowdFunder', 'Lista de Proyectos', 'Publicar Proyecto', and 'Mis Proyectos'. On the right side of the header is a user icon.

The main content area is titled 'Lista de Proyectos:' and shows four project cards:

- Titulo: Salon de tuneo**
PropONENTE: 0x7030f4D0cC092449E4B68c8D0c9bc00a14C9f561
Descripción:
En mi garaje de tuneo de coches personalizamos y mejoramos motores, suspensiones, frenos y carrocerías de vehículos para hacerlos únicos y más potentes. También ofrecemos servicios de mantenimiento completo para mantener los autos en óptimas condiciones.
Fondos necesarios: 10.0 ETH
Tiempo de desarrollo: 60 Días
Estado: Pendiente de Votación
Fecha límite para votar: 14/5/2023
Votos totales: 0
Votos positivos: 0

Buttons: + -
- Titulo: Panaderia**
PropONENTE: 0x7030f4D0cC092449E4B68c8D0c9bc00a14C9f561
Descripción:
Mi panadería es un lugar donde puedes encontrar panes recién horneados, pasteles, galletas y empanadas para disfrutar en cualquier momento del día. Ofrecemos productos de alta calidad, preparados con ingredientes frescos y con la pasión de una panadería artesanal.
Fondos necesarios: 3.0 ETH
Fondos recaudados: 0.0 ETH
Tiempo de desarrollo: 60 Días
Estado: Pendiente de Financiación

Progress bar: Fondos a donar Donar
- Titulo: Evento contra el cancer**
PropONENTE: 0x7030f4D0cC092449E4B68c8D0c9bc00a14C9f561
Descripción:
Mi evento benéfico contra el cáncer reúne fondos para la investigación y tratamiento de esta enfermedad. Ofrecemos entretenimiento, subasta de obras de arte y productos locales.
Fondos necesarios: 10.0 ETH
Tiempo de desarrollo: 60 Días
Estado: Finalizado

Status bar: Proyecto Finalizado
- Titulo: Mi Lamborghini**
PropONENTE: 0x7030f4D0cC092449E4B68c8D0c9bc00a14C9f561
Descripción:
Mi Lamborghini es un auto deportivo de lujo. Ofrece un alto rendimiento, velocidad y estilo. Es un ícono de la ingeniería italiana y una experiencia de manejo emocionante y exclusiva.
Fondos necesarios: 100000.0 ETH

Made by Frenzoid (Evi Mihai Sabau Sabau) - Universidad de Alicante.

Figura 33: Interfaz Web - Lista de Proyectos.



5.1.1.5 Pantalla del formulario para proponer un proyecto

The screenshot shows a web page titled "Propone un proyecto:" (Propose a project). The form fields include:

- Título del proyecto**: A text input field labeled "Título del proyecto".
- Descripción**: A text area labeled "Descripción del proyecto".
- Fondos Necesarios (ETH)**: A text input field labeled "Fondos necesario en ETH".
- Tiempo total para el desarrollo (Días)**: A text input field labeled "Tiempo total necesario para el desarrollo (en días)".

At the bottom right of the form area is a blue button labeled "Propone Proyecto".

Made by Frenzoid (Elvi Mihai Sabau Sabau) - Universidad de Alicante.

Figura 34: Interfaz Web - Formulario para proponer un proyecto.



5.1.1.6 Pantalla de detalles de un proyecto

The screenshot shows a web application interface for a crowdfunding project. At the top, there's a navigation bar with links: 'UACrowdFunder', 'Lista de Proyectos', 'Publicar Proyecto', 'Mis Proyectos', and a user icon. Below the navigation is a section titled 'Detalles del Proyecto:'.

Panaderia
0x7030f4d0dc092449E4868c8DDc9bc00a14C9f561
Mi panadería es un lugar donde puedes encontrar panes recién horneados, pasteles, galletas y empanadas para disfrutar en cualquier momento del día. Ofrecemos productos de alta calidad, preparados con ingredientes frescos y con la pasión de una panadería artesanal.

Fondos requeridos: 3.0 ETH
Fondos recaudados: 0.0 ETH
Tiempo total de desarrollo: 60 días
Fecha límite: 14/5/2023
Votos totales: 0 (0 positivos)
Estado: 1

A continuación, there are two columns: 'Comentarios:' and 'Añadir nuevo comentario:'. The 'Comentarios:' column lists two comments from 'Comentario 1' and 'Comentario 3', both posted on '14/4/2023, 16:26:53'. The 'Añadir nuevo comentario:' column has a text input field and a blue 'Enviar' button.

At the bottom of the page, a footer bar reads: 'Made by Frenzoid (Elvi Mihai Sabau Sabau) - Universidad de Alicante.'

Figura 35: Interfaz Web - Detalles de un proyecto.

Se ha tomado la decisión de alterar el diseño de la implementación de esta pantalla con respecto al mockup desarrollado, para mostrar de forma más cómoda los comentarios y el formulario.

La modificación que se ha realizado separa la lista de comentarios y la funcionalidad de enviar un nuevo comentario en 2 columnas separadas.



5.1.2. Back-End (DAO)

El smart contract que se desarrolla deberá implementar las variables y funciones de la interfaz mencionada en el capítulo anterior (Capítulo 6.4.3).

Este implementará las funcionalidades necesarias para el desarrollo de la prueba de concepto, incluyendo los objetos necesarios para gestionar proyectos, comentarios, gestión de fondos, etc.

NOTA: Para agilizar la presentación de la prueba de concepto, se ha decidido agregar 4 propuestas de ejemplo desde el constructor del smart contract.

5.1.2.1 Implementación del Smart Contract

A continuación se muestra el código fuente del smart contract implementado.

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.17;
3
4 /**
5  * @title Sistema de gesti n descentralizado de la financiaci n de proyectos.
6  * @author Elvi Mihai Sabau Sabau, aka: Frenzoid => github.com/frenzoid.
7  * @notice Este contrato implementa una plataforma de financiamiento colectiva y
8  * aut noma,
9  * donde los usuarios pueden proponer nuevos proyectos, y la comunidad se encarga
10 * de gestionar
11 * la recaudaci n, integridad y viabilidad de dichos proyectos mediante votos.
12 */
13 contract ProjectPlatformDemo {
14     /// @dev Enumeraci n para representar el estado de un proyecto.
15     enum ProjectState {
16         PENDING,
17         FUNDING,
18         FUNDED,
19         CANCELLED
20     }
21
22     /// @dev Estructura para representar un comentario.
23     struct Comment {
24         address author; // Direcci n del autor del comentario.
25         string message; // Texto del comentario.
26         uint date; // Fecha del comentario.
27     }
28
29     /// @dev Estructura para representar una inversi n.
30     struct Investment {
31         address investor; // Direcci n del inversor.
32         uint amount; // Cantidad invertida.
33         uint date; // Fecha de la inversi n.
34     }
35
36     /// @dev Estructura para representar un proyecto.
37     struct Project {
38         address proposer; // Direcci n del proponente del proyecto.
39         string title; // T tulo del proyecto.
         string description; // Descripc i n del proyecto.
         uint requiredFunds; // Fondos necesarios para el proyecto.
```



```
40     uint collectedFunds; // Fondos recaudados.
41     uint totalDevelopmentTime; // Tiempo total de desarrollo del proyecto.
42     uint deadline; // Fecha límite para la siguiente fase (votación,
43     // financiación o validación de hitos).
44     uint totalVotes; // Total de votos recibidos.
45     uint positiveVotes; // Total de votos positivos recibidos.
46     ProjectState state; // Estado del proyecto (pendiente, aprobado, financiado
47     o cancelado).
48     Investment[] investments; // Array para controlar las inversiones de cada
49     // dirección en el proyecto.
50     Comment[] comments; // Array para almacenar los comentarios del proyecto.
51     mapping(address => bool) hasVoted; // Mapping para controlar quién ha
52     votado el proyecto.
53 }
54
55     /// @dev Tiempo límite para la votación y financiación en segundos (30 días)
56 )
57     uint private constant TIME_LIMIT = 30 days;
58
59     /// @dev Umbral de Votación. Porcentaje mínimo de votos positivos para
60     // aprobar un proyecto.
61     uint private constant APPROVAL_THRESHOLD = 85;
62
63     /// @dev Lista de proyectos
64     Project[] public projects;
65
66     /// @dev Eventos para registrar las acciones realizadas en el contrato.
67     event ProjectProposed(uint indexed projectId, address indexed proposer);
68     event ProjectApproved(uint indexed projectId);
69     event ProjectRejected(uint indexed projectId);
70     event ProjectInvested(
71         uint indexed projectId,
72         address indexed investor,
73         uint amount
74     );
75     event ProjectFunded(uint indexed projectId);
76     event ProjectCancelled(uint indexed projectId);
77     event ProjectInvestmentWithdrawn(
78         uint indexed projectId,
79         address indexed investor,
80         uint amount
81     );
82     event CommentAdded(
83         uint indexed projectId,
84         address indexed commenter,
85         string message
86     );
87
88     /// @dev Constructor del contrato, se ejecuta una sola vez al desplegar el
89     // contrato.
90     constructor() {
91         /**
92          * @dev Creamos 4 proyectos de ejemplo,
93          * uno con el estado PENDING, otro con el estado FUNDING,
94          * otro con el estado FUNDED, y el último con el estado CANCELLED.
95          *
96          * Usamos la función proposeProject para crear los proyectos,
97          */
98         proposeProject(
99             "Salón de túnico",
100            "Salón de túnico"
101        );
102    }
```



```
93         "En mi garaje de tuneo de coches personalizamos y mejoramos motores, suspensiones, frenos y carrocerias de vehiculos para hacerlos unicos y mas potentes. Tambien ofrecemos servicios de mantenimiento completo para mantener los autos en optimas condiciones.",  
94         10,  
95         30  
96     );  
97     proposeProject(  
98         "Panaderia",  
99         "Mi panaderia es un lugar donde puedes encontrar panes recien horneados, pasteles, galletas y empanadas para disfrutar en cualquier momento del dia. Ofrecemos productos de alta calidad, preparados con ingredientes frescos y con la pasion de una panaderia artesanal.",  
100        3,  
101        30  
102    );  
103    proposeProject(  
104        "Evento contra el cancer",  
105        "Mi evento benefico contra el cancer reune fondos para la investigacion y tratamiento de esta enfermedad. Ofrecemos entretenimiento, subasta de obras de arte y productos locales. ",  
106        10,  
107        30  
108    );  
109    proposeProject(  
110        "Mi Lamborghini",  
111        "Mi Lamborghini es un auto deportivo de lujo. Ofrece un alto rendimiento, velocidad y estilo. Es un icono de la ingenieria italiana y una experiencia de manejo emocionante y exclusiva.",  
112        100000,  
113        30  
114    );  
115  
116    /**  
117     * @dev Cambiamos el estado del proyecto 1 a PENDING.  
118     */  
119     projects[0].state = ProjectState.PENDING;  
120  
121    /**  
122     * @dev Cambiamos el estado del proyecto 2 a FUNDING.  
123     */  
124     projects[1].state = ProjectState.FUNDING;  
125  
126    /**  
127     * @dev Cambiamos el estado del proyecto 3 a FUNDED.  
128     */  
129     projects[2].state = ProjectState.FUNDED;  
130  
131    /**  
132     * @dev Cambiamos el estado del proyecto 4 a CANCELLED.  
133     */  
134     projects[3].state = ProjectState.CANCELLED;  
135  
136    /**  
137     * @dev A adimos comentarios a los proyectos.  
138     */  
139     addComment(0, "Comentario 1");  
140     addComment(0, "Comentario 2");  
141     addComment(1, "Comentario 1");  
142     addComment(1, "Comentario 3");
```



```
143     addComment(2, "Comentario 1");
144     addComment(2, "Comentario 3");
145     addComment(3, "Comentario 1");
146     addComment(3, "Comentario 3");
147 }
148 /**
149 * RF5
150 * @notice Devuelve el total de proyectos.
151 * @return uint256
152 */
153
154 function getProjectsCount() public view returns (uint256) {
155     return projects.length;
156 }
157 /**
158 * RF4
159 * @notice Propone un nuevo proyecto para ser aprobado por la comunidad.
160 * @param _title Título del proyecto
161 * @param _description Descripción del proyecto
162 * @param _fundingGoal Cantidad de fondos (en Ether) necesarios para su
163 formalización
164 * @param _totalDevelopmentTime Tiempo total (días) para su desarrollo
165 * @return ID del proyecto
166 */
167 function proposeProject(
168     string memory _title,
169     string memory _description,
170     uint _fundingGoal,
171     uint _totalDevelopmentTime
172 ) public returns (uint256) {
173     /// @dev Validaciones de los parámetros de entrada
174     require(bytes(_title).length > 0, "El título no puede estar vacío");
175     require(
176         bytes(_description).length > 0,
177         "La descripción no puede estar vacía"
178     );
179     require(
180         _fundingGoal > 0,
181         "El objetivo de financiación debe ser mayor que 0"
182     );
183     require(
184         _totalDevelopmentTime > 0,
185         "El tiempo total de financiación debe ser mayor que 0"
186     );
187     /// @dev Creación del nuevo proyecto
188     Project storage newProject = projects.push();
189
190     newProject.proposer = msg.sender;
191     newProject.title = _title;
192     newProject.description = _description;
193     newProject.requiredFunds = _fundingGoal * 1 ether;
194     newProject.totalDevelopmentTime = (_totalDevelopmentTime * 1 days);
195     newProject.deadline = block.timestamp + TIME_LIMIT;
196     newProject.state = ProjectState.PENDING;
197
198     // Emitimos el evento ProjectProposed
199     emit ProjectProposed(projects.length - 1, msg.sender);
200
201 }
```



```
202         return projects.length - 1;
203     }
204
205     /**
206      * RF5
207      * @notice Devuelve un proyecto dado un ID
208      * @param _projectId ID del proyecto
209      * @return proposer Dirección del creador del proyecto
210      * @return title Título del proyecto
211      * @return description Descripción del proyecto
212      * @return requiredFunds Fondos requeridos para el proyecto
213      * @return collectedFunds Fondos recaudados para el proyecto
214      * @return totalDevelopmentTime Tiempo total de desarrollo del proyecto
215      * @return deadline Fecha límite del proyecto
216      * @return totalVotes Votos totales del proyecto
217      * @return positiveVotes Votos positivos del proyecto
218      * @return userVoted Booleano que indica si el usuario ha votado el proyecto
219      * @return state Estado del proyecto
220      */
221     function getProject(
222         uint256 _projectId
223     )
224         public
225         view
226         returns (
227             address proposer,
228             string memory title,
229             string memory description,
230             uint requiredFunds,
231             uint collectedFunds,
232             uint totalDevelopmentTime,
233             uint deadline,
234             uint totalVotes,
235             uint positiveVotes,
236             bool userVoted,
237             ProjectState state
238         )
239     {
240         require(_projectId < projects.length, "El proyecto no existe");
241
242         Project storage project = projects[_projectId];
243
244         proposer = project.proposer;
245         title = project.title;
246         description = project.description;
247         requiredFunds = project.requiredFunds;
248         collectedFunds = project.collectedFunds;
249         totalDevelopmentTime = project.totalDevelopmentTime;
250         deadline = project.deadline;
251         totalVotes = project.totalVotes;
252         positiveVotes = project.positiveVotes;
253         userVoted = hasVoted(_projectId, msg.sender);
254         state = project.state;
255     }
256
257     /**
258      * RF7
259      * @notice Devuelve la cantidad de comentarios de un proyecto dado un ID.
260      * @param _projectId ID del proyecto
261      * @return uint256
```



```
262      */
263     function getCommentsCount(
264         uint256 _projectId
265     ) public view returns (uint256) {
266         require(_projectId < projects.length, "El proyecto no existe");
267         return projects[_projectId].comments.length;
268     }
269
270     /**
271      * RF6
272      * @notice Aade un comentario a un proyecto.
273      * @param _projectId ID del proyecto
274      * @param _comment Comentario aadir
275      */
276     function addComment(uint256 _projectId, string memory _comment) public {
277         /// @dev Validaciones de los parametros de entrada
278         require(_projectId < projects.length, "El proyecto no existe");
279         require(
280             bytes(_comment).length > 0,
281             "El comentario no puede estar vacio"
282         );
283
284         /// @dev Aadimos el comentario al proyecto
285         Comment storage c = projects[_projectId].comments.push();
286         c.author = msg.sender;
287         c.message = _comment;
288         c.date = block.timestamp;
289
290         // Emitimos el evento CommentAdded
291         emit CommentAdded(_projectId, msg.sender, _comment);
292     }
293
294     /**
295      * RF7
296      * @notice Devuelve un comentario dado un ID de proyecto y un ID de comentario
297      * @param _projectId ID del proyecto
298      * @param _commentId ID del comentario
299      * @return author Direccion del autor del comentario
300      * @return message Mensaje del comentario
301      * @return date Fecha del comentario
302      */
303     function getComment(
304         uint256 _projectId,
305         uint256 _commentId
306     ) public view returns (address author, string memory message, uint date) {
307         require(_projectId < projects.length, "El proyecto no existe");
308         require(
309             _commentId < projects[_projectId].comments.length,
310             "El comentario no existe"
311         );
312
313         Comment storage comment = projects[_projectId].comments[_commentId];
314
315         author = comment.author;
316         message = comment.message;
317         date = comment.date;
318     }
319
320     /**
321      * @notice Devuelve todos los comentarios de un proyecto
```



```
322     * @param _projectId ID del proyecto
323     * @return comments Array de comentarios
324     */
325     function getComments(
326         uint256 _projectId
327     ) public view returns (Comment[] memory comments) {
328         require(_projectId < projects.length, "El proyecto no existe");
329         return projects[_projectId].comments;
330     }
331
332     /**
333      * RF8
334      * @notice Permite a los usuarios votar por un proyecto.
335      * @param _projectId ID del proyecto a votar
336      * @param _positiveVote Si el voto es positivo (true) o negativo (false)
337      */
338     function voteProject(uint256 _projectId, bool _positiveVote) public {
339         /// @dev Validaciones de los parametros de entrada
340         require(_projectId < projects.length, "El proyecto no existe");
341         require(
342             projects[_projectId].state == ProjectState.PENDING,
343             "El proyecto no esta pendiente de aprobacion"
344         );
345         require(
346             block.timestamp < projects[_projectId].deadline,
347             "El plazo para votar ha expirado"
348         );
349         require(
350             !projects[_projectId].hasVoted[msg.sender],
351             "Ya has votado este proyecto"
352         );
353
354         /// @dev Actualizamos los votos del proyecto
355         projects[_projectId].totalVotes++;
356         if (_positiveVote) projects[_projectId].positiveVotes++;
357
358         /// @dev Marcamos al usuario como que ha votado el proyecto
359         projects[_projectId].hasVoted[msg.sender] = true;
360     }
361
362     /**
363      * RF8
364      * @notice Devuelve si el usuario ha votado un proyecto.
365      * @param _projectId ID del proyecto
366      * @param _user Dirección del usuario a comprobar
367      * @return boolean
368      */
369     function hasVoted(
370         uint256 _projectId,
371         address _user
372     ) public view returns (bool) {
373         require(_projectId < projects.length, "El proyecto no existe");
374         return projects[_projectId].hasVoted[_user];
375     }
376
377     /**
378      * RF9
379      * @notice Comprueba si un proyecto ha sido aprobado por la comunidad.
380      * @param _projectId ID del proyecto a comprobar
381      */
```



```
382     function checkProjectApproval(uint256 _projectId) public {
383         /// @dev Validaciones de los parámetros de entrada
384         require(_projectId < projects.length, "El proyecto no existe");
385         require(
386             projects[_projectId].proposer == msg.sender,
387             "No eres el autor"
388         );
389         require(
390             projects[_projectId].state == ProjectState.PENDING,
391             "El proyecto no está pendiente de aprobación"
392         );
393         require(
394             block.timestamp >= projects[_projectId].deadline,
395             "El plazo para votar no ha expirado"
396         );
397
398         /// @dev Calculamos el porcentaje de votos positivos
399         uint256 positiveVotesPercentage = (projects[_projectId].positiveVotes *
400             100) / projects[_projectId].totalVotes;
401
402         /// @dev Actualizamos el estado del proyecto
403         if (positiveVotesPercentage >= APPROVAL_THRESHOLD) {
404             projects[_projectId].state = ProjectState.FUNDING;
405
406             /// @dev Reiniciamos el plazo para la siguiente fase
407             projects[_projectId].deadline = block.timestamp + TIME_LIMIT;
408
409             emit ProjectApproved(_projectId);
410         } else {
411             projects[_projectId].state = ProjectState.CANCELLED;
412             emit ProjectRejected(_projectId);
413         }
414     }
415
416     /**
417      * RF10
418      * @notice Permite a los usuarios invertir en un proyecto.
419      * @param _projectId ID del proyecto a invertir
420      */
421     function investInProject(uint256 _projectId) public payable {
422         /// @dev Validaciones de los parámetros de entrada
423         require(_projectId < projects.length, "El proyecto no existe");
424         require(
425             projects[_projectId].state == ProjectState.FUNDING,
426             "El proyecto no está en fase de financiación"
427         );
428         require(
429             block.timestamp < projects[_projectId].deadline,
430             "El plazo para invertir ha expirado"
431         );
432         require(msg.value > 0, "La cantidad de inversión debe ser mayor que 0");
433
434         /// @dev Actualizamos los fondos recaudados del proyecto
435         projects[_projectId].collectedFunds += msg.value;
436
437         /// @dev Creamos un nuevo registro de inversión para el proyecto
438         Investment storage newInvestment = projects[_projectId]
439             .investments
440             .push();
441         newInvestment.investor = msg.sender;
```



```
442     newInvestment.amount = msg.value;
443     newInvestment.date = block.timestamp;
444
445     /// @dev Emitimos el evento ProjectInvested
446     emit ProjectInvested(_projectId, msg.sender, msg.value);
447
448     /// @dev Si se ha alcanzado el umbral, actualizamos el estado del proyecto
449     if (
450         projects[_projectId].collectedFunds >=
451         projects[_projectId].requiredFunds
452     ) {
453         projects[_projectId].state = ProjectState.FUNDED;
454
455         /// @dev Enviamos los fondos al autor del proyecto
456         address payable wallet = payable(projects[_projectId].proposer);
457         wallet.transfer(projects[_projectId].collectedFunds);
458
459         /// @dev Emitimos el evento para que quede registrado
460         emit ProjectFunded(_projectId);
461     }
462 }
463
464 /**
465 * RF10 && RF 13
466 * @notice Comprueba si un proyecto ha sido financiado.
467 * @param _projectId ID del proyecto a comprobar
468 */
469 function checkProjectFunding(uint256 _projectId) public {
470     /// @dev Validaciones de los par metros de entrada
471     require(_projectId < projects.length, "El proyecto no existe");
472     require(
473         projects[_projectId].proposer == msg.sender,
474         "No eres el autor"
475     );
476     require(
477         projects[_projectId].state == ProjectState.FUNDING,
478         "El proyecto no esta en fase de financiacion"
479     );
480     require(
481         block.timestamp >= projects[_projectId].deadline,
482         "El plazo para invertir no ha expirado"
483     );
484
485     /// @dev Actualizamos el estado del proyecto
486     if (
487         projects[_projectId].collectedFunds >=
488         projects[_projectId].requiredFunds
489     ) {
490         projects[_projectId].state = ProjectState.FUNDED;
491
492         /// @dev Enviamos los fondos al autor del proyecto
493         address payable wallet = payable(projects[_projectId].proposer);
494         wallet.transfer(projects[_projectId].collectedFunds);
495
496         /// @dev Emitimos el evento para que quede registrado
497         emit ProjectFunded(_projectId);
498     } else {
499         /// @dev Actualizamos el estado del proyecto, ya que no se ha logrado
500         projects[_projectId].state = ProjectState.CANCELLED;
```



```
501         /// @dev Emitimos el evento para que quede registrado
502         emit ProjectCancelled(_projectId);
503     }
504 }
505 /**
506 * @notice Permite a los inversores retirar sus fondos si el proyecto se ha
507 cancelado.
508 * @param _projectId ID del proyecto a comprobar.
509 */
510 function withdrawInvestment(uint256 _projectId) public {
511     /// @dev Validaciones de los par metros de entrada
512     require(_projectId < projects.length, "El proyecto no existe");
513
514     /// @dev Si el proyecto a n est en fase de financiaci n, lanzamos un
515 error.
516     require(
517         projects[_projectId].deadline < block.timestamp,
518         "El proyecto aun esta en fase de financiacion."
519     );
520
521     /// @dev Recorremos los registros de inversi n del proyecto
522     for (uint256 i = 0; i < projects[_projectId].investments.length; i++) {
523         /// @dev Si el inversor coincide con el que llama a la funci n
524         if (
525             projects[_projectId].investments[i].investor == msg.sender &&
526             projects[_projectId].investments[i].amount != 0
527         ) {
528             /// @dev Actualizamos el registro de inversi n
529             projects[_projectId].investments[i].amount = 0;
530
531             /// @dev Enviamos los fondos que ha invertido en el proyecto al
532             inversor
533             address payable wallet = payable(msg.sender);
534             wallet.transfer(projects[_projectId].investments[i].amount);
535
536             /// @dev Emitimos el evento para que quede registrado
537             emit ProjectInvestmentWithdrawn(
538                 _projectId,
539                 msg.sender,
540                 projects[_projectId].investments[i].amount
541             );
542             break;
543         }
544
545         /// @dev Si no se ha encontrado el inversor, lanzamos un error.
546         require(false, "No has participado en el proyecto.");
547     }
548
549 /**
550 * @notice funciones de fallback
551 * @dev Funciones que se ejecutan cuando se llaman a funciones inexistentes del
552 contrato,
553 * o se manda ether al contrato directamente.
554 */
555 fallback() external payable {
556     revert("No se aceptan fondos en la direccion del contrato");
557 }
```



```
557
558     receive() external payable {
559         revert("No se aceptan fondos en la direccion del contrato");
560     }
561 }
```

Bloque de Código 24: Smart Contract de la prueba de concepto.



5.2. Despliegue del sistema

5.2.1. Código fuente del proyecto

El código del proyecto se encuentra en la siguiente [dirección](#), para arrancarlo solo se debe seguir los pasos descritos en los archivos “README” de ambas carpetas (Front-End y Back-End).

Para el despliegue de este sistema usaremos diferentes herramientas que se han mencionado en el contexto tecnológico.

5.2.2. Despliegue de la interfaz web

Para desplegar la aplicación de Svelte usaremos Docker. Docker, como ya se ha explicado en apartados anteriores, es una plataforma que sirve para encapsular servicios en contenedores, y desplegarlos en cualquier máquina que ejecute Docker.

De esta forma, el servicio a desarrollar se vuelve portable, y se aísla tanto en recursos como en configuración del resto de servicios ejecutándose en el servidor.

5.2.2.1 Sobre la metodología del despliegue

Tal como se menciona en el capítulo de Metodología, se propone un despliegue con CI mediante Github Actions y utilizando DockerHub para automatizar dicho despliegue. Sin embargo, en esta sección se detalla un despliegue manual con el objetivo de explicar en profundidad este proceso, abordando las partes que conforman Docker y el despliegue de la aplicación en un contenedor.

Así que, en este caso la aplicación se despliega en un VPS²⁹ con Docker instalado, de esta forma tendremos la aplicación de Svelte ejecutándose en un contenedor en el VPS las 24hrs del día.

5.2.2.2 Encapsulando la aplicación web

Para generar un contenedor desde la app de Svelte, primero vamos a necesitar generar una “imagen” de la aplicación. Una imagen es el conjunto de archivos y comandos que contendrá y ejecutará un contenedor, muy similar a una ISO, pero en Docker.

²⁹VPS: Servidor Virtual Privado.



Para generar una imagen, lo único que se necesita es declarar en un archivo llamado “Dockerfile” los archivos a encapsular en la imagen, los comandos necesarios para preparar la aplicación, y por ultimo el comando que se ejecutará dentro del contenedor para desplegar el servidor web.

A continuación se muestra el Dockerfile.

```
frontend > 📁 Dockerfile > ...
1  # Imagen base de nodejs
2  FROM node:latest
3
4  # Directorio de trabajo
5  WORKDIR /app
6
7  # Copia de archivos
8  COPY package.json ./
9  COPY package-lock.json ./
10 COPY . ../
11
12 # Instalación de dependencias
13 RUN npm install
14 RUN npm run build
15
16 # Puerto en el cual se expone el servidor web
17 EXPOSE 5000
18
19 # Comando por defecto
20 CMD ["npm", "run", "serve"]
```

Figura 36: Dockerfile - Dockerfile del Front-End.

Para generar la imagen se ejecuta el comando: “docker build . -t elvi/tfg:latest”. El nombre de la imagen se asigna con el parámetro -t, y la nomenclatura del nombre es “autor”/“servicio”:“versión”.

Mediante docker se publica la imagen a DockerHub, un repositorio de imágenes de Docker, y una vez publicado es posible arrancar el Front-End desarrollado en cualquier ordenador que tenga docker instalado.

Para arrancar un contenedor con la imagen se ejecuta el siguiente comando: “docker run –name ‘FRONTEND_TFG’ -p 5000:5000 elvi/tfg:latest”.

El parametro -p indica el mapeo de puertos del “anfitrion”:“contenedor”.



The screenshot shows a 'Container list' view in Portainer. It lists several Docker containers:

Name	State	Image	Created	IP Address	Published Ports	Ownership
PORTAINER_INTERFACE	running	portainer/portainer-ce:latest	2023-04-03 23:26:50	172.17.0.2	none	administrators
FRONTEND_TFG	healthy	elvi/tfg:latest	2023-04-14 20:39:14	172.17.0.5	5000:5000	administrators
NGINX_PROXY_MANAGER	running	jc21/nginx-proxy-manager:latest	2023-04-07 02:19:56	172.17.0.4	443:443 80:80 81:81	administrators
REDIS_LANGCHAIN_DB	running	redis/redis-stack:latest	2023-04-07 02:20:26	172.17.0.3	6379:6379	administrators
JUPYTER_HUB_SERVER	running	jupyterhub/jupyterhub	2023-04-09 00:16:18	172.23.0.5	none	administrators
CLICKHOUSE_SERVER	running	chroma/clickhouse-server:22.9-alpine	2023-04-09 01:36:14	172.23.0.6	8123:8123 9000:9000	administrators

Figura 37: Portainer - Interfaz de Portainer, gestor de contenedores de Docker.

En el VPS, mediante Portainer, se tiene la posibilidad de gestionar los contenedores desplegados. Al acceder al dominio o dirección IP del VPS seguido de :5000³⁰, se puede visualizar la aplicación de Svelte que ha sido desplegada en el servidor.

The screenshot shows the front-end application's main page, titled 'Lista de Proyectos' (List of Projects). It displays three projects:

- Salón de tuneo**: Propone: 0x7039f4d0dc092449e4868c80dc9bc00a14c9f561. Descripción: En mi garaje de tuning de coches personalizamos y mejoramos motores, suspensiones, frenos y carrocerías de vehículos para hacerlos únicos y más potentes. También ofrecemos servicios de mantenimiento completo para mantener los autos en óptimas condiciones. Fondos necesarios: 10.0 ETH. Tiempo de desarrollo: 60 Días. Estado: Pendiente de Votación. Fecha límite para votar: 14/5/2023. Votos totales: 0. Votos positivos: 0.
- Panadería**: Propone: 0x7039f4d0dc092449e4868c80dc9bc00a14c9f561. Descripción: Mi panadería es un lugar donde puedes encontrar panes recién horneados, pasteles, galletas y empanadas para disfrutar en cualquier momento del día. Ofrecemos productos de alta calidad, preparados con ingredientes frescos y con la pasión de una panadería artesanal. Fondos necesarios: 3.0 ETH. Fondos recaudados: 1.0 ETH. Tiempo de desarrollo: 60 Días. Estado: Pendiente de Financiación. Progreso: 33.33%. Fondos a donar: [button] Donar [button].
- Evento contra el cancer**: Propone: 0x7039f4d0dc092449e4868c80dc9bc00a14c9f561. Descripción: Mi evento benéfico contra el cáncer reúne fondos para la investigación y tratamiento de esta enfermedad. Ofrecemos entretenimiento, subasta de obras de arte y productos locales. Fondos necesarios: 10.0 ETH. Tiempo de desarrollo: 60 Días. Estado: Finalizado. Proyecto finalizado.

Figura 38: Front-End - Muestra de la aplicación desplegada.

El Front-End ahora está desplegado en la siguiente dirección: <https://tfg.storage.frenzoid.dev/>

³⁰5000 es el puerto en el que la aplicación web permanece a la espera de conexiones.



5.2.3. Despliegue del Smart Contract

El despliegue del smart contract se realizará mediante la herramienta HardHat, que posibilita la compilación, el despliegue y la validación de un contrato en cualquier red blockchain basada en el protocolo Ethereum.

A través de una plantilla³¹ que desarrollé, el despliegue de los contratos se automatiza, siendo sólo necesario especificar en el comando de despliegue la red donde se desea desplegar el contrato. Dicho comando se ejecuta en la carpeta “Back-End”, y presenta la siguiente sintaxis: “npm run deploy-RED”.

En el escenario de este proyecto, el contrato se despliega en la red de pruebas "Mumbai" de Polygon, por lo que la cuenta utilizada para desplegar el contrato necesitará disponer de MATIC, la criptomonedra de la red de Polygon.

La obtención de MATIC en la red de prueba puede llevarse a cabo utilizando las **faucets públicas de Mumbai**, con el objetivo de conseguir suficiente MATIC para cubrir los costes de desplegar el contrato en la red de prueba. Una vez adquirido un cierto volumen de MATIC, es posible proceder con el despliegue del contrato usando HardHat. Para ello, simplemente se necesita ejecutar el comando “npm run deploy-mumbai” en la carpeta “Back-End” del proyecto.

Al ejecutar este comando, HardHat revisará el código notificando cualquier error encontrado durante la compilación, empaquetará el contrato y exportará 2 archivos: el binario del contrato que se cargará en la blockchain, y una interfaz del contrato que se usará por la librería Ethers del Front-End para poder comunicarse con el contrato.

Una vez que el contrato es cargado y minado en la red, HardHat procederá automáticamente a validar el contrato, subiendo el código fuente del contrato a la red, de esta manera, se hacen públicas las funcionalidades del contrato a través de los exploradores de bloques como Polyscan.

```
Deployed on network: mumbai
Deployed by account: 0x7030f4D0dC092449E4868c8DDc9bc00a14C9f561
Verifying contract, waiting 6 tx for propagation...
Nothing to compile
Successfully submitted source code for contract
contracts/ProjectPlatformDemo.sol:ProjectPlatformDemo at 0xDC03f4b2EFBe513BD2a627e2F3D09fFde105C6F0
for verification on the block explorer. Waiting for verification result...

Successfully verified contract ProjectPlatformDemo on Etherscan.
https://mumbai.polygonscan.com/address/0xDC03f4b2EFBe513BD2a627e2F3D09fFde105C6F0#code
PS G:\TFG-INGINF\backend> █
```

Figura 39: Verificación del contrato - Captura de la terminal, HardHat notificando de la correcta validación del contrato en la red Mumbai.

NOTA: La dirección del contrato que se muestra en la captura es diferente debido a que se desplegó múltiples veces durante la realización de este documento.

Validar un contrato inteligente sirve para verificar y asegurar la transparencia del código fuente de dicho contrato en la red. Polyscan permite a los usuarios examinar el contrato en busca de posibles vulnerabilidades, errores o actividades sospechosas antes de interactuar con él.

³¹https://github.com/Frenzoid/DAPP_DevKit_1



5.2.3.1 Verificación del contrato

Una vez desplegado el contrato en la red de Mumbai, se puede presenciar el contrato desde el explorador de bloques de Polygon denominado Polyscan, y validar que se ha verificado el código fuente correctamente.

The screenshot shows the Polyscan interface for the Mumbai network. The URL in the address bar is <https://mumbai.polygonscan.com/address/0x6029f14C4Acca68d3eA64D13daCCcc5a2Fb00983#code>. The page displays the 'Contract Overview' for the contract at address 0x6029f14C4Acca68d3eA64D13daCCcc5a2Fb00983. It shows a balance of 0 MATIC and indicates that the contract source code has been verified. The 'Code' tab is selected, showing the Solidity source code for the ProjectPlatformDemo contract. The code includes comments in Spanish describing the decentralized project management system and its features like crowdfunding and voting.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

/*
 * Etiquetado: Sistema de gestión descentralizado de la financiación de proyectos
 * Author: Elliot Wihni, Sobau Sobau
 * Notifica: Este contrato implementa una plataforma de financiamiento colectivo y autónoma,
 * donde los usuarios pueden proponer nuevos proyectos, y la comunidad se encarga de revisar,
 * y votar para aceptar la financiación de dicho proyecto.
 */
contract ProjectPlatformDemo {
    // Edad: Enumeración para representar el estado de un proyecto
    enum ProjectState {
        PENDING,
        PENDING,
        FUNDED,
        CANCELLED
    }
    // Edad: Estructura para representar un comentario
    struct Comment {
        address author; // Dirección del autor del comentario
        string message; // Texto del comentario
        uint date; // Fecha del comentario
    }
}
```

Figura 40: Polyscan - Captura del contrato desplegado y verificado en la red de Mumbai.

Se puede observar el contrato que se usará para la demostración en esta dirección:

<https://mumbai.polygonscan.com/address/0xE29548B54a978eeb4d071e6D7bd83871b2d56CAE>



5.3. Evaluación de la prueba de concepto

En el siguiente enlace se muestra un vídeo de la prueba de concepto desarrollada, donde se explica cada pantalla y se demuestra su funcionalidad. El objetivo de este vídeo es validar la viabilidad del sistema desarrollado de la solución propuesta.

<https://youtu.be/DrpZKpXmicU>



6. Aspectos Éticos, Económicos y Sociales

En esta apartado, se profundiza en ciertos aspectos así destacando los aspectos éticos, económicos y sociales del proyecto.

Estas aspectos son clave para destacar diferentes puntos de vista, sobre el proyecto y sobre la tecnología blockchain ya que difiere radicalmente de un sistema común, y así poder debatir con un contexto más sólido la naturaleza del trabajo.

6.1. Aspectos Éticos

- **Transparencia:** La tecnología blockchain garantiza la trazabilidad y transparencia en el uso de los fondos, permitiendo a los usuarios verificar en qué se gastan los activos y asegurando un control adecuado de los recursos.
- **Participación comunitaria:** La comunidad tiene la posibilidad de votar, comentar y validar los proyectos propuestos y sus hitos, fomentando la colaboración y el compromiso ético en la toma de decisiones.
- **Protección contra malversación de fondos:** El mecanismo de gestión autónomo y la blockchain garantizan que ningún administrador o entidad pueda malversar los fondos de la plataforma.

6.2. Aspectos Económicos

- **Financiación descentralizada:** El sistema de gestión propuesto permitiría la financiación descentralizada de proyectos, eliminando intermediarios y reduciendo los costos asociados a la gestión de fondos.
- **Inversión comunitaria:** Los usuarios pueden invertir en proyectos aprobados, diversificando sus inversiones y apoyando proyectos en los que creen.
- **Reembolsos a inversores:** En caso de que un proyecto no alcance sus metas, los fondos serán reembolsados a los inversores, protegiendo así sus intereses.

6.3. Aspectos Sociales

- **Empoderamiento de la comunidad:** El sistema permite a la comunidad decidir qué proyectos se aprobarán, promoviendo la toma de decisiones colectiva y el empoderamiento de sus miembros.
- **Interacción y colaboración:** La posibilidad de comentar y debatir sobre los proyectos propuestos fomenta la interacción entre los usuarios y fortalece el sentido de comunidad.
- **Estímulo a la innovación:** Al facilitar la financiación y el apoyo a proyectos que de otro modo podrían tener dificultades para acceder a recursos, se promueve la innovación y el desarrollo de nuevas ideas y soluciones.



7. Presupuesto

En este apartado se detalla el presupuesto estimado necesario para desplegar el sistema propuesto. La estimación final del presupuesto será el costo total a duración de un año.

7.1. Infraestructura Web

El coste de la infraestructura se fracciona en 2 partes, el dominio y el servidor a alquilar.

7.1.1. Dominio

El coste actual de los dominios varia dependiendo del proveedor y de la extensión que se deseé contratar. Para este trabajo, el dominio que se desea contratar es “uacrowdfunder.com”, con el proveedor Namecheap³².

El costo estimado del dominio con Namecheap es de 8,76eur / Año.

The screenshot shows a search result for "uacrowdfunder.com" on a website. The domain is marked as available with a green checkmark. The price is listed as \$8.76/yr and €12.79/yr. An "Add to cart" button is visible.

Figura 41: Presupuesto - Coste anual del dominio.

³²www.namecheap.com

7.1.2. Alojamiento Web

Para el alojamiento de la página web en cuestión, existen diversas alternativas. En este caso, la aplicación front-end se encuentra completamente encapsulada en una imagen docker, lo que permite considerar tres opciones: la utilización de un servidor privado virtual con docker instalado, el uso de un servicio de alojamiento de contenedores autogestionado, o la elección de un servicio de alojamiento de páginas web, evitando así la necesidad de utilizar docker.

Para el propósito de este trabajo, se opta por un VPS por conveniencia para el desarrollador. Al emplear un VPS, el único pago a gestionar es el correspondiente al servidor. Los precios por este tipo de servicios suelen variar dependiendo del proveedor, pero a diferencia de otros servicios, el costo es único, y no fluctúa en función del uso de los recursos del servidor.

En este caso, se ha contratado un VPS con el proveedor Contabo³³, que ofrece recursos más que suficientes para el despliegue de la aplicación web

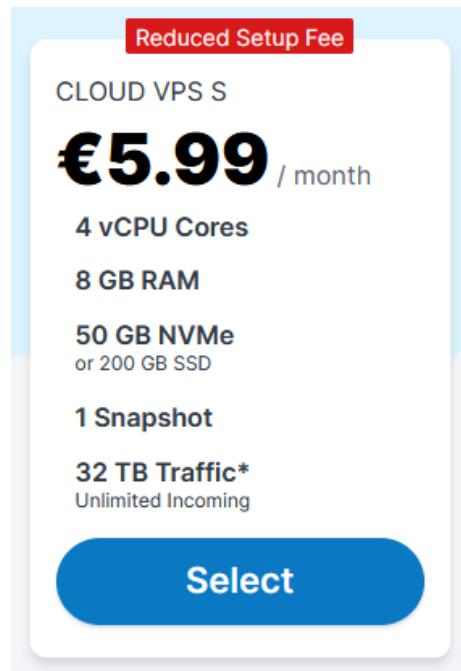


Figura 42: Presupuesto - Coste mensual del VPS.

Este plan tiene un coste único de instalación de 4eur, y un coste mensual de 6eur / mes.

³³www.contabo.com



7.2. Smart Contract

El coste del smart contract varía dependiendo del precio actual del gas y de la red que se usa. Para ello, los costes usados para esta estimación serán de la fecha actual (14:20 - 21/04/2023) en la red Polygon.

7.2.1. Despliegue

El coste asociado al despliegue se incurre una única vez durante la publicación del smart contract en la blockchain y está condicionado por la densidad del contrato.

Por otro lado, el coste de cada función se estima en gas, variando según el coste computacional de cada método del contrato. En este contexto, únicamente se toma en cuenta como parte del coste el despliegue del contrato, dado que el coste del gas utilizado por los métodos será asumido por los usuarios de la plataforma.

Considerando que el contrato desarrollado no cumple con todos los requisitos de la plataforma, para la estimación se duplicará el coste del despliegue de este contrato, proporcionando así una cota superior pesimista del coste real que podría implicar el despliegue de un contrato con todas las funcionalidades implementadas.

Para determinar el coste del despliegue en la Mainnet de Polygon, se recurre al estimador de MetaMask. Durante el despliegue del contrato, se puede observar el coste en Euro del despliegue del contrato en la red.

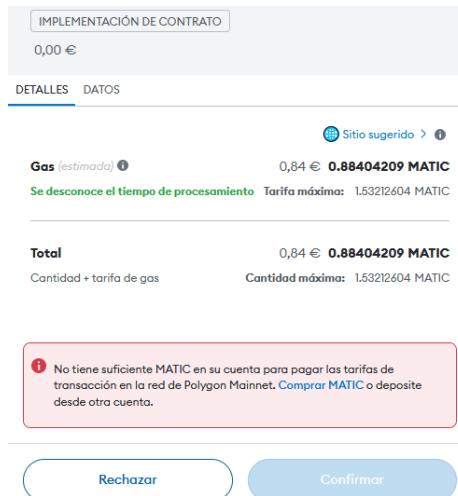


Figura 43: Presupuesto - Coste del despliegue del contrato en Euro.

La captura revela el gas requerido para realizar la transacción y el coste equivalente en Euro, representando un pago único de un total de 0,84 Eur. Dado que el despliegue se refiere al contrato de la prueba de concepto, al duplicar dicho coste se obtiene una estimación de 1.68 Eur.



7.3. Tabla de totales

A continuación se muestra una tabla con los costes totales del mantenimiento y despliegue del sistema a plazo de un año.

Concepto	Coste único (EUR)	Coste mensual (EUR)	Coste anual (EUR)
Dominio	0	0	8.76
Alojamiento VPS	4	6	6*12
Despliegue Smart Contract	1.68	0	0
Total	5.68	6	80.76 Eur

Cuadro 5: Presupuesto - Tabla de totales.



8. Conclusiones

8.1. Conclusión sobre el trabajo

Los objetivos establecidos al inicio de este trabajo se han logrado satisfactoriamente. Se ha investigado, creado e implementado un sistema de gestión descentralizado para la financiación de proyectos, el cual permite a los participantes proponer, votar, verificar y financiar proyectos de manera transparente. Este enfoque elimina la necesidad de intermediarios que puedan desviar fondos o manipular propuestas, logrando así un sistema autónomo y descentralizado en el que cualquier individuo puede presentar un proyecto. Además, tanto la financiación como el desarrollo del proyecto no se ven afectados por terceros, garantizando la ausencia de censura de cualquier tipo.

8.2. Beneficios personales

El desarrollo de este trabajo ha brindado una valiosa oportunidad para ampliar mis horizontes y aplicar los conocimientos adquiridos a lo largo de los años previos y durante mi carrera en un proyecto concreto. Asimismo, me ha permitido aprender y perfeccionar mis habilidades en la elaboración de documentos de esta índole, así como en la investigación y análisis de artículos científicos.

Gracias a este proyecto, he aprendido en detalle como funciona la tecnología blockchain, y como aplicarla en un proyecto real, además de aplicar varios patrones de desarrollo de software aprendidos durante la carrera.

8.3. Sobre la tecnología

Originalmente, la tecnología blockchain emergió como una propuesta alternativa al sistema económico convencional. Esta propuesta se centraba en el uso de la criptografía como medio para conseguir una plataforma económica inmune a la censura y la manipulación centralizada por parte de las empresas y los gobiernos. Con el transcurso del tiempo, esta idea inicial experimentó una evolución significativa, se empezaron a desarrollar más tecnologías que aprovechaban el potencial de la arquitectura descentralizada y la criptografía en la que se fundamenta la blockchain. En este contexto es donde surge Ethereum, un protocolo que permite la ejecución de código y el almacenamiento de datos en una blockchain descentralizada.

La implementación de esta tecnología posibilita el desarrollo de sistemas caracterizados por su inmutabilidad, donde además todos los procesos son trazables y los datos transparentes. Esto permite la creación de aplicaciones en las que los usuarios pueden validar los procesos en vez de confiar ciegamente en que el sistema funciona como se afirma. De este modo, se incrementa la confiabilidad y la seguridad, ya que los usuarios pueden verificar por sí mismos las operaciones y los datos.

El enfoque propuesto en este estudio podría haberse llevado a cabo utilizando una opción centralizada, incorporando una arquitectura tradicional cliente-servidor. Sin embargo, un sistema de tal naturaleza estaría expuesto a manipulaciones externas de diversas formas, incluso si se diseñara para operar de manera intrínsecamente autónoma.

Por consiguiente, la plataforma que se diseña en este trabajo de un sistema de gestión descentralizado, se beneficia directamente de las ventajas proporcionadas por la tecnología blockchain. Esta tecnología, por su propia naturaleza, asegura la transparencia y la trazabilidad tanto de los procesos como de los datos. En otras palabras, los procesos y los datos almacenados en una blockchain pueden ser rastreados y verificados por todos los participantes, lo que refuerza la integridad del sistema.



8.4. Trabajo a futuro

La solución propuesta y el proyecto presentan un considerable margen para futuras mejoras. Hay posibilidad de expansión mediante la adición de nuevas funcionalidades y mejoras en el diseño. Tal como se ha señalado en el capítulo *Modelo del Sistema*, el proyecto enfrenta un dilema legal en relación al tratamiento de los datos personales y legales de los proponentes y los peritos. Este aspecto, por ejemplo, representa una valiosa oportunidad para su desarrollo y mejora en un futuro.



Referencias

- [1] *2RB - Tourbillon Watch USA - Built In America (Suspended)*. [Online; accessed 3. Feb. 2023]. Feb. de 2023. URL: https://www.kickstarter.com/projects/590244552/2rb-tourbillon-watch-usa-built-in-america?ref=recently_launched.
- [2] *All About Testnets – Collective Shift*. [Online; accessed 7. Feb. 2023]. Feb. de 2022. URL: <https://collectiveshift.io/infrastructure/testnet-guide>.
- [3] Binance Academy. *Layer 2*. [Online; accessed 3. Feb. 2023]. Feb. de 2023. URL: <https://academy.binance.com/en/glossary/layer-2>.
- [4] *Blockchain Merkle Trees*. [Online; accessed 2. Feb. 2023]. Sep. de 2022. URL: <https://www.geeksforgeeks.org/blockchain-merkle-trees>.
- [5] *Blue Laser Lamp- with passive cooling technology*. [Online; accessed 3. Feb. 2023]. Feb. de 2023. URL: <https://www.kickstarter.com/projects/1251236349/blue-laser-lamp-with-passive-cooling-technology>.
- [6] *Cobra Wallet - A modern wallet that compliments your phone*. [Online; accessed 3. Feb. 2023]. Feb. de 2023. URL: <https://www.kickstarter.com/projects/1210691057/cobra-wallet-a-modern-wallet-that-compliments-your/comments>.
- [7] *Consensus Mechanisms in Blockchain - Naukri Learning*. [Online; accessed 3. Feb. 2023]. Feb. de 2023. URL: <https://www.naukri.com/learning/articles/consensus-mechanisms-in-blockchain>.
- [8] Contributors to Wikimedia projects. *Crowdfunding - Wikipedia*. [Online; accessed 7. Feb. 2023]. Ene. de 2023. URL: <https://en.wikipedia.org/w/index.php?title=Crowdfunding&oldid=1133881145>.
- [9] Contributors to Wikimedia projects. *Indiegogo - Wikipedia*. [Online; accessed 31. Jan. 2023]. Ago. de 2022. URL: <https://en.wikipedia.org/w/index.php?title=Indiegogo&oldid=1107486841>.
- [10] Contributors to Wikimedia projects. *Kickstarter - Wikipedia*. [Online; accessed 31. Jan. 2023]. Ene. de 2023. URL: <https://en.wikipedia.org/w/index.php?title=Kickstarter&oldid=1135393616>.
- [11] *Dash - The Smart phone Car Stereo*. [Online; accessed 3. Feb. 2023]. Feb. de 2023. URL: <https://www.kickstarter.com/projects/devium/dash-the-smart-phone-car-stereo>.
- [12] Massimo Di Pierro. «What is the blockchain?» En: *Computing in Science & Engineering* 19.5 (2017), págs. 92-95.
- [13] *ethereum*. [Online; accessed 7. Feb. 2023]. Feb. de 2023. URL: <https://github.com/ethereum>.
- [14] *Ethereum Virtual Machine (EVM) | ethereum.org*. [Online; accessed 7. Feb. 2023]. Feb. de 2023. URL: <https://ethereum.org/en/developers/docs/evm>.
- [15] *Eyez™ by ZionEyez HD Video Recording Glasses for Facebook*. [Online; accessed 3. Feb. 2023]. Feb. de 2023. URL: <https://www.kickstarter.com/projects/zioneyez/eyeztm-by-zioneyez-hd-video-recording-glasses-for/posts>.
- [16] *Gas and fees | ethereum.org*. [Online; accessed 7. Feb. 2023]. Feb. de 2023. URL: <https://ethereum.org/en/developers/docs/gas>.



- [17] Balastegui Garcia Guillermo et al. «Corporate Digital Identity Based on Blockchain». En: *Research and Innovation Forum 2022*. Ed. por Anna Visvizi, Orlando Troisi y Mara Grimaldi. Cham: Springer International Publishing, 2023, págs. 645-655. ISBN: 978-3-031-19560-0.
- [18] Tina H Ho. «Social purpose corporations: The next targets for greenwashing practices and crowdfunding scams». En: *Seattle J. Soc. Just.* 13 (2014), pág. 935.
- [19] *Home | ethereum.org*. [Online; accessed 7. Feb. 2023]. Feb. de 2023. URL: <https://ethereum.org/en>.
- [20] *How do I know the project I backed wasn't a scam?* [Online; accessed 31. Jan. 2023]. Ene. de 2023. URL: <https://help.kickstarter.com/hc/en-us/articles/1260805483330-How-do-I-know-the-project-I-backed-wasn-t-a-scam->.
- [21] *How do I report a project?* [Online; accessed 31. Jan. 2023]. Ene. de 2023. URL: <https://help.crowdfunder.co.uk/en/articles/1870122-how-do-i-report-a-project>.
- [22] etherscan. io. *Ethereum (ETH) Blockchain Explorer*. [Online; accessed 12. Feb. 2023]. Feb. de 2023. URL: <https://etherscan.io>.
- [23] *Opcodes for the EVM | ethereum.org*. [Online; accessed 7. Feb. 2023]. Feb. de 2023. URL: <https://ethereum.org/en/developers/docs/evm/opcodes>.
- [24] Aitana Pastor Osuna et al. «Addressing the Challenges of Biological Passport Through Blockchain Technology». En: *Research and Innovation Forum 2022*. Ed. por Anna Visvizi, Orlando Troisi y Mara Grimaldi. Cham: Springer International Publishing, 2023, págs. 133-143. ISBN: 978-3-031-19560-0.
- [25] *Patricia Merkle Trees | ethereum.org*. [Online; accessed 7. Feb. 2023]. Feb. de 2023. URL: <https://ethereum.org/en/developers/docs/data-structures-and-encoding/patricia-merkle-trie>.
- [26] Blockchain Presence. «Getting more MATIC on Polygon Mumbai - Blockchain Presence - Medium». En: *Medium* (mayo de 2022). ISSN: 5251-3480. URL: <https://medium.com/the-blockchain-presence-blog/getting-more-matic-on-polygon-mumbai-5db25d13480d>.
- [27] Abdul Qayum y Abdul Razzaq. «A Self-Evolving Design of Blockchain-based Open Source Community». En: *2020 3rd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*. IEEE. 2020, págs. 1-11.
- [28] Gurinder Singh, Vikas Garg y Pooja Tiwari. «Introduction to Blockchain Technology». En: *Transforming Cybersecurity Solutions using Blockchain*. Springer, 2021, págs. 1-18.
- [29] Solidity Team. *Solidity Programming Language*. [Online; accessed 11. Feb. 2023]. Feb. de 2023. URL: <https://soliditylang.org>.
- [30] upGrad. «Everything You Need to Know About Distributed Ledger Technology». En: *Medium* (dic. de 2021). URL: <https://medium.com/@upGrad/everything-you-need-to-know-about-distributed-ledger-technology-8e3d26703b78>.
- [31] *Verifiable Credentials Data Model v1.1*. [Online; accessed 21. Apr. 2023]. Feb. de 2022. URL: <https://www.w3.org/TR/vc-data-model>.
- [32] *Vyper — Vyper documentation*. [Online; accessed 11. Feb. 2023]. Sep. de 2022. URL: <https://docs.vyperlang.org/en/stable>.
- [33] *What Are Blockchain Nodes and How Do They Work?* [Online; accessed 3. Feb. 2023]. Feb. de 2023. URL: <https://builtin.com/blockchain/blockchain-node>.



- [34] *What happens to my money if a campaign fails?* [Online; accessed 31. Jan. 2023]. Ene. de 2023. URL: <https://support.indiegogo.com/hc/en-us/articles/205665518-What-happens-to-my-money-if-a-campaign-fails->.
- [35] *What is a REST API?* [Online; accessed 12. Feb. 2023]. Feb. de 2023. URL: <https://www.redhat.com/en/topics/api/what-is-a-rest-api>.
- [36] *What is a token?* [Online; accessed 7. Feb. 2023]. Feb. de 2023. URL: <https://www.coinbase.com/es/learn/crypto-basics/what-is-a-token>.