



Universidad de
Oviedo



ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

GRADO EN INGENIERÍA INFORMÁTICA EN TECNOLOGÍAS DE
LA INFORMACIÓN

ÁREA DE LENGUAJES Y SISTEMAS INFORMÁTICOS

FOOLMEONCE: BLOCKCHAIN APLICADA A VERIFICACIÓN DE
PROMESAS

D. RODRÍGUEZ GONZÁLEZ, Miguel

TUTOR: D. ^a BLANCO AGUIRRE, Raquel

COTUTOR: D. APARICIO BERMEJO, Jorge

FECHA: Julio 2023

Índice

1	INTRODUCCIÓN	12
2	OBJETIVOS Y ALCANCE	14
2.1	OBJETO DEL TFG	14
2.2	ALCANCE DEL TFG	14
3	ASPECTOS TEÓRICOS	16
3.1	BLOCKCHAIN	16
3.1.1	CARACTERÍSTICAS DE UNA BLOCKCHAIN	16
3.1.2	RED DE PARES	17
3.1.3	TIPOS DE REDES QUE APLICAN LA TECNOLOGÍA BLOCKCHAIN	17
3.1.4	TRANSACCIONES Y BLOQUES	18
3.1.5	ALGORITMOS DE CONSENSO	18
3.1.6	CRIPTOMONEDAS Y ALTCOINS	19
3.1.7	CUENTAS Y CARTERAS (WALLETS)	20
3.1.8	CONTRATOS INTELIGENTES (SMART CONTRACTS)	20
3.1.9	TOKENS	21
3.2	APLICACIÓN DESCENTRALIZADA	21
3.2.1	CONEXIÓN <i>FRONTEND</i> CON LA BLOCKCHAIN	21
3.3	ETHEREUM Y <i>DAPPS</i>	21
3.3.1	LA MÁQUINA VIRTUAL DE ETHEREUM	22
3.3.2	GAS	22
3.3.3	LOS CONTRATOS INTELIGENTES DE ETHEREUM	23
3.3.4	BLOQUES EN ETHEREUM	24
3.3.5	EL ESTADO DE ETHEREUM	25
3.3.6	TRANSACCIONES Y CUENTAS EN ETHEREUM	26
3.3.7	ESTÁNDARES DE ETHEREUM	26
3.4	ORGANIZACIONES AUTÓNOMAS DESCENTRALIZADAS (DAOs)	27
3.4.1	FUNDAMENTOS DE LAS ORGANIZACIONES AUTÓNOMAS DESCENTRALIZADAS.	28

3.4.2	FLUJO BASE DE UNA ORGANIZACIÓN AUTÓNOMA DESCENTRALIZADA	28
3.4.3	TIPOS DE VOTACIÓN EN LA ORGANIZACIÓN AUTÓNOMA DESCENTRALIZADA	29
3.5	CONTROL DE VERSIONES	29
3.6	METODOLOGÍA DE TRABAJO	30
3.6.1	SCRUM	30
4	ESTUDIOS PREVIOS Y ANÁLISIS DE ALTERNATIVAS	32
4.1	ANÁLISIS DE ALTERNATIVAS	32
4.1.1	ELECCIÓN DEL PROTOCOLO QUE APLICA LA TECNOLOGÍA BLOCKCHAIN	32
4.1.1.1	Bitcoin	32
4.1.1.2	Ethereum	32
4.1.1.3	Solana	32
4.1.1.4	Cardano	33
4.1.1.5	Alternativa elegida	33
4.1.2	ELECCIÓN DE LENGUAJE DE PROGRAMACIÓN PARA ETHEREUM	33
4.1.2.1	Solidity	33
4.1.2.2	Vyper	34
4.1.2.3	Alternativa elegida	34
4.1.3	ELECCIÓN DE LA TECNOLOGÍA <i>FRONTEND</i>	34
4.1.3.1	Angular	34
4.1.3.2	React	35
4.1.3.3	Alternativa Elegida	35
4.1.4	ELECCIÓN DE UN ENTORNO DE DESPLIEGUE EFICIENTE DE CONTRATOS	35
4.1.5	ELECCIÓN DEL ESTÁNDAR PARA LA CREACIÓN DE TOKENS EN EL CONTRATO DESARROLLADO.	35
4.1.5.1	ERC-20	35
4.1.5.2	ERC-721	36
4.1.5.3	ERC-1155	37
4.1.5.4	EIP-5114	38
4.1.5.5	ERC-777	38
4.1.5.6	Alternativa elegida	39
4.1.6	ELECCIÓN DEL SISTEMA DE ARCHIVOS DESCENTRALIZADO	39
4.1.6.1	IPFS	40
4.1.6.2	Swarm	40

4.1.6.3	Alternativa Elegida	40
4.1.7	ELECCIÓN DE LAS HERRAMIENTAS PARA EL DESARROLLO DE CONTRATOS INTELIGENTES	40
4.1.7.1	Análisis de herramientas para el desarrollo	40
4.1.7.1.1	Truffle	41
4.1.7.1.2	Hardhat	41
4.1.7.2	Análisis de librerías de conexión entre los usuarios y los contratos	41
4.1.7.2.1	Web3.js	41
4.1.7.2.2	Ethers.js	41
4.1.7.3	Alternativa Elegida	41
4.1.8	ANÁLISIS DEL PROTOCOLO DE PARTICIPACIÓN DE VOTANTES PARA LAS DAO	42
4.1.8.1	Emisión de tokens de gobernanza	42
4.1.8.2	Skin in the Game	42
4.1.8.3	Prueba de Participación	42
4.1.8.4	Alternativa Elegida	43
4.1.9	ELECCIÓN DE LA BLOCKCHAIN DEDICADA AL DESPLIEGUE DE APLICACIONES DESCENTRALIZADAS	43
4.1.9.1	Polygon	46
4.1.9.2	Avalanche	47
4.1.9.3	Binance	47
4.1.9.4	Alternativa elegida	47
5	PLANIFICACIÓN	48
5.1	DIAGRAMA DE GANTT	48
5.2	DETALLE DE LAS TAREAS	48
6	PRESUPUESTO	50
6.1	CAPÍTULO 1: HARDWARE	50
6.2	CAPÍTULO 2: SOFTWARE	51
6.3	AMORTIZACIÓN DEL INMOVILIZADO MATERIAL	51
6.4	CAPÍTULO 3: MANO DE OBRA	52
6.5	PRESUPUESTO TOTAL	53
7	ANÁLISIS	54

7.1 REQUISITOS DE USUARIO	54
7.1.1 REQUISITOS FUNCIONALES	54
7.1.2 REQUISITOS NO FUNCIONALES	55
7.2 REQUISITOS DEL SISTEMA	55
7.2.1 MAPA DE HISTORIA	55
7.2.2 HISTORIAS DE USUARIO	59
7.2.2.1 Investigación de los diferentes protocolos existentes aplicados a la tecnología Blockchain	59
7.2.2.2 Investigación de la Red Blockchain Ethereum	59
7.2.2.3 Investigación librerías para el desarrollo, despliegue y comunicación con los contratos inteligentes	59
7.2.2.4 Investigación de los lenguajes de programación para contratos inteligentes en Ethereum	60
7.2.2.5 Investigación de las tecnologías de frontend	60
7.2.2.6 Investigación de las diferencias entre Solidity y Vyper	60
7.2.2.7 Investigación del lenguaje Solidity y su seguridad	60
7.2.2.8 Configuración de Visual Studio Code para despliegue de Backend y Frontend	61
7.2.2.9 Investigación de los estándares de Ethereum (ERCs)	61
7.2.2.10 Investigación de los sistemas de archivos descentralizados	61
7.2.2.11 Definición de la base para primer prototipo <i>backend</i>	61
7.2.2.12 Desarrollo de una función que permita el registro de usuarios	62
7.2.2.13 Desarrollo de una función que permita la creación de promesas electorales	62
7.2.2.14 Desarrollo de una función que permita la obtención de todas las promesas electorales	62
7.2.2.15 Definición del primer prototipo <i>frontend</i>	63
7.2.2.16 Definición de una cabecera de navegación	63
7.2.2.17 Definición de una pantalla que permita al usuario ver el listado de promesas	63
7.2.2.18 Definición de una pantalla donde se ilustren todos los detalles de la promesa electoral	63
7.2.2.19 Definición de una pantalla que permita registrarse	64
7.2.2.20 Definición de una pantalla que permita la creación de una promesa electoral	64
7.2.2.21 Auditar conjunto de contratos del primer prototipo	64
7.2.2.22 Investigación teórica Organizaciones Autónomas Descentralizadas	65

7.2.2.23	Investigación Práctica Organizaciones Autónomas Descentralizadas	65
7.2.2.24	Definición de una función que permita aprobar las promesas electorales	65
7.2.2.25	Resultados investigación Organizaciones Autónomas Descentralizadas	65
7.2.2.26	Investigación de otras cadenas de bloques dedicadas al despliegue de aplicaciones descentralizadas	66
7.2.2.27	Unión de la plataforma de creación y listado de promesas con la Organización Autónoma Descentralizada	66
7.3	PROTOTIPO DE INTERFAZ DE USUARIO	66
7.3.1	LOGO DE LA PÁGINA WEB	66
7.3.2	PÁGINA PRINCIPAL	67
7.3.3	PÁGINA DE REGISTRO DE USUARIO	68
7.3.4	PÁGINA DE CREACIÓN DE UNA PROMESA	69
7.3.5	PÁGINA DE LISTADO DE PROMESAS	70
7.3.6	PÁGINA DE VISTA DE LA PROMESA	71
7.4	MAPA DE NAVEGACIÓN	72
8	DISEÑO	73
8.1	ARQUITECTURA	73
8.2	DISEÑO FÍSICO DE LOS DATOS	74
8.2.1	ESTRUCTURA DE LOS DATOS	74
8.3	DIAGRAMA DE PAQUETES	75
8.3.1	PAQUETE PRIMER PROTOTIPO	76
8.3.2	PAQUETE INVESTIGACIÓN DAOS	77
8.3.3	PAQUETE DE PRUEBAS	78
8.4	DIAGRAMA DE INTERACCIÓN	79
8.4.1	DIAGRAMA DE CONEXIÓN CON LA PLATAFORMA VOTUM	79
8.4.2	DIAGRAMA PARA LA CONSULTA DE PROMESAS ELECTORALES	80
8.4.3	DIAGRAMA PARA EL REGISTRO DE PROMESAS ELECTORALES	81
9	PRUEBAS	82
9.1	INTRODUCCIÓN	82
9.2	ESPECIFICACIÓN DE LA APLICACIÓN	82

9.3	DISEÑO DE SITUACIONES DE PRUEBA	82
9.3.1	CLASES DE EQUIVALENCIA	82
9.3.2	TÉCNICA BASADA EN CAMINOS	83
9.3.2.1	Técnica de caminos simples	84
9.3.2.2	Técnica de pares de caminos	85
9.4	DATOS PARA LAS PRUEBAS (BASE DE DATOS)	85
9.5	CASOS DE PRUEBA	86
9.6	RESUMEN PRUEBAS	87
10	IMPLEMENTACIÓN DE UNA DAO	89
10.1	IMPLEMENTACIÓN DE LA DAO	89
10.1.1	PARTES DE UNA DAO CON OPENZEPELIN	89
10.1.1.1	Contrato de Gobernanza	90
10.1.1.2	Contrato que brinde el poder de voto	90
10.1.1.3	Contrato para el control del tiempo y manejo de privilegios	90
10.1.1.4	Contrato sobre el que se ejecutará la gobernanza	91
10.1.2	VERIFICACIÓN DE UNA PROMESA ELECTORAL	91
10.1.2.1	Función a Gobernar: ApprovePromise	91
10.1.2.2	Despliegue DAO	92
10.1.2.3	Preparación	93
10.1.2.4	Propuesta	93
10.1.2.5	Votación	94
10.1.2.6	Cola y ejecución	95
10.2	RESULTADOS	97
10.2.1	VENTAJAS	97
10.2.2	DESVENTAJAS	97
11	MANUAL DE USUARIO	99
11.1	INTRODUCCIÓN	99
11.2	USO DE LA APLICACIÓN	99
11.2.1	CONEXIÓN CON LA PLATAFORMA	100
11.2.2	LISTADO Y VISUALIZACIÓN DE PROMESAS	100

11.2.3	REGISTRO DE POLÍTICOS	102
11.2.4	CREACIÓN DE NUEVAS PROMESAS ELECTORALES	103
12	AMPLIACIONES	104
12.1	ESTABLECER DIFERENCIA ENTRE LOS USUARIOS	104
12.2	INVESTIGACIÓN PRÁCTICA DE UNA DAO OFF-CHAIN	105
12.3	INVESTIGACIÓN DE HELIA, LA NUEVA HERRAMIENTA DE IPFS	105
13	PROBLEMAS ENCONTRADOS	106
13.1	PROBLEMA CON LAS PRUEBAS Y ETHERS.JS	106
13.2	PROBLEMA CON NODE Y NPM	106
13.3	PROBLEMA CON EL CLIENTE DE IPFS	106
14	CONCLUSIONES	108
15	REFERENCIAS	109

Índice de Figuras

<i>Ilustración 1: Generación de Bloques</i>	18
<i>Ilustración 2: Ejemplo de un contrato para almacenar un valor en Blockchain</i>	24
<i>Ilustración 3: Coste en gas del contrato de ejemplo</i>	24
<i>Ilustración 4: Diagrama de votación</i>	29
<i>Ilustración 5: Tablero de tareas</i>	30
<i>Ilustración 6: Interfaz ERC-20</i>	36
<i>Ilustración 7: Interfaz ERC-721</i>	36
<i>Ilustración 8: Clase ERC721URIStorage</i>	37
<i>Ilustración 9: Interfaz ERC-1155</i>	38
<i>Ilustración 10: Interfaz ERC-5114</i>	38
<i>Ilustración 11: Interfaz ERC-777</i>	39
<i>Ilustración 12: Asociación de una persona, un voto</i>	43
<i>Ilustración 13: Compilación y despliegue de ERC-20</i>	44
<i>Ilustración 14: Despliegue plataforma FoolMeOnce</i>	45
<i>Ilustración 15: Arquitectura Polygon [80]</i>	46
<i>Ilustración 16: Árbol de Merkle [81]</i>	46
<i>Ilustración 17: Diagrama de Gantt</i>	48
<i>Ilustración 18: Diagrama de Gantt con tareas</i>	49
<i>Ilustración 19: Mapa de historias 1</i>	56
<i>Ilustración 20: Mapa de historias 2</i>	57
<i>Ilustración 21: Mapa de historias 3</i>	58
<i>Ilustración 22: Logo de la página web</i>	67
<i>Ilustración 23: Prototipo de la página principal</i>	67
<i>Ilustración 24: Prototipo de registro de usuarios</i>	68
<i>Ilustración 25: Prototipo de la pantalla de creación</i>	69
<i>Ilustración 26: Prototipo de listado de promesas electorales</i>	70
<i>Ilustración 27: Prototipo vista de promesa electoral</i>	71
<i>Ilustración 28: Mapa de navegación</i>	72
<i>Ilustración 29: Arquitectura</i>	73
<i>Ilustración 30: Estructuras de datos del proyecto</i>	75
<i>Ilustración 31: Diagrama de paquetes global del TFG</i>	76
<i>Ilustración 32: Diagrama de paquetes del primer prototipo</i>	77
<i>Ilustración 33: Diagrama paquetes Investigación DAO</i>	78
<i>Ilustración 34: Diagrama Paquetes Pruebas</i>	79
<i>Ilustración 35: Diagrama de interacción para la conexión con plataforma</i>	80

<i>Ilustración 36: Diagrama de interacción para la consulta de datos</i>	80
<i>Ilustración 37: Diagrama de interacción para el registro de promesas electorales</i>	81
<i>Ilustración 38: Diagrama de caminos 1</i>	84
<i>Ilustración 39: Diagrama de caminos 2</i>	84
<i>Ilustración 40: Cobertura de pruebas automatizadas (obtenida con la herramienta Hardhat)</i>	88
<i>Ilustración 41: Diagrama base de sucesos en la DAO a implementar</i>	89
<i>Ilustración 42: Función approvePromise - gobierno</i>	92
<i>Ilustración 43: Preparación de datos para la verificación de una promesa</i>	93
<i>Ilustración 44: Nueva propuesta para votación</i>	94
<i>Ilustración 45: Votación de la propuesta</i>	94
<i>Ilustración 46: Comprobar el estado de la propuesta</i>	95
<i>Ilustración 47: Proceso de poner en cola una propuesta aprobada</i>	95
<i>Ilustración 48: Proceso de ejecución de una propuesta aprobada</i>	96
<i>Ilustración 49: Promesa con id cero sin aprobar</i>	96
<i>Ilustración 50: Promesa con id cero aprobada</i>	97
<i>Ilustración 51: Página Principal de la plataforma</i>	99
<i>Ilustración 52: Conexión con la plataforma</i>	100
<i>Ilustración 53: Listado de promesas electorales</i>	101
<i>Ilustración 54: Visualización de una promesa</i>	101
<i>Ilustración 55: Visualización de una promesa desde un dispositivo móvil</i>	102
<i>Ilustración 56: Registro de políticos</i>	103
<i>Ilustración 57: Pantalla para la creación de nuevas promesas</i>	103
<i>Ilustración 58: Promesa electoral de un partido</i>	104
<i>Ilustración 59: Promesa electoral de un político</i>	104

Índice de Tablas

<i>Tabla 1: Unidades del GAS [5]</i>	23
<i>Tabla 2: Presupuesto Hardware</i>	50
<i>Tabla 3: Presupuesto Software</i>	51
<i>Tabla 4: Amortización del inmovilizado material</i>	52
<i>Tabla 5: Presupuesto mano de obra</i>	52
<i>Tabla 6: Presupuesto Total</i>	53
<i>Tabla 7: Caminos simples</i>	85
<i>Tabla 8: Pares de caminos</i>	85
<i>Tabla 9: Datos para las pruebas</i>	86
<i>Tabla 10: Casos de prueba</i>	87

1 Introducción

En la actualidad, cada cuatro años se celebran en nuestro país unas elecciones libres y democráticas y, llegado el momento, todo el conjunto de políticos y sus respectivos partidos se reúnen para establecer sus promesas electorales. ¿Qué es lo que quieren corregir? ¿Qué es lo que van a hacer nuevo? ¿Qué nuevas políticas quieren emprender? Las respuestas a estas preguntas son las ya nombradas promesas electorales.

Estas promesas son accesibles mediante los medios tradicionales (televisión, radio o correo electoral), y mediante sitios web como las páginas oficiales de los partidos y hemerotecas (periódicos o la Hemeroteca Digital Nacional [1]).

Los sitios web son un medio más permanente de consulta que los medios tradicionales. Por tanto, los ciudadanos cuando desean consultar los detalles de las promesas acuden a estos sitios web, lo que les permite verificar lo prometido. En la era actual que vivimos, la era de la información, a veces es complicado comprobar la veracidad de la información dispuesta, debido, por ejemplo, a las noticias falsas o que las formaciones políticas modifican los datos de sus páginas web (a conveniencia).

Para contrarrestar el problema de las noticias falsas (bulos) han surgido plataformas de verificación de hechos (*fact-checking*) como Newtral [2] o Maldita [3]. A pesar del trabajo realizado por estas plataformas, la verificación y recordatorio de las promesas está amenazado. Esta amenaza reside en las fuentes utilizadas para la comprobación, ya que pueden alterarse de forma voluntaria por parte de los propietarios o pueden verse comprometidas por ataques informáticos.

Dada la implicación que tienen las promesas electorales en nuestra sociedad actual, y en vista de lo anterior, surge la necesidad de plantear un sistema que permita salvaguardar las promesas electorales de manera inalterable y que permita al ciudadano verificarlas.

Este Trabajo Fin de Grado (TFG) denominado **FoolMeOnce**, elaborado dentro del marco de colaboración entre la Universidad de Oviedo y la empresa HP SDCS (Observatorio Tecnológico HP), va a intentar resolver este problema utilizando la tecnología **Blockchain** (cadena de bloques).

FoolMeOnce será una aplicación descentralizada (*dApp*), que al hacer uso de la tecnología Blockchain y los *contratos inteligentes* o *smart contracts* (programas que se ejecutan en la Blockchain) permitirá grabar todos los datos de las promesas electorales de manera inalterable, accesible y trazable.

Finalmente, esta aplicación dará credibilidad a los partidos políticos y ayudará también al ciudadano a recordar lo prometido, ya que la información estará guardada de manera segura en una red Blockchain.

2 Objetivos y alcance

2.1 OBJETO DEL TFG

Este TFG plantea dos retos muy importantes a realizar. En primer lugar, la creación de una aplicación descentralizada que permita la creación y listado de las promesas electorales por parte de las formaciones políticas, haciendo uso de los contratos inteligentes. Ello implica el estudio de los protocolos que hacen uso de la tecnología Blockchain, las soluciones que aportan, sus arquitecturas y las herramientas que han surgido gracias esta. En segundo lugar, investigar las Organizaciones Autónomas Descentralizadas (DAO, *Decentralized Autonomous Organization*) y realizar una implementación funcional que a modo de prueba de concepto permita verificar promesas electorales.

2.2 ALCANCE DEL TFG

Los puntos esenciales de este TFG son:

- Investigación de los protocolos que emplean la tecnología Blockchain.
- Investigación de las aplicaciones descentralizadas (dApp).
- Investigación de los servicios de almacenamiento descentralizado.
- Creación de una aplicación descentralizada que permita la creación y listado de promesas electorales.
 - Creación de un sistema para la identificación de los usuarios.
 - A aquellos usuarios registrados en la plataforma se les permitirá realizar un registro en la Blockchain de sus promesas electorales.
 - Se permitirá establecer imágenes que representen la promesa.
 - Se permitirá establecer un título, descripción, relaciones con otras promesas y temas que aborde la promesa.
 - Se permitirá establecer un nivel de obligatoriedad de las promesas.
 - Se generará un código QR único por cada promesa electoral creada en la aplicación.
 - Aquellos usuarios no registrados podrán acceder de manera libre y gratuita a todas las promesas.
 - Diseño de interfaz adaptable tanto a ordenadores como a dispositivos móviles.
 - Diseño de una interfaz sobria con objeto de no representar una orientación política.
- Investigación de las Organizaciones Autónomas Descentralizadas (DAOs).

- Implementación de una DAO para la verificación de las promesas electorales creadas en la plataforma.
 - Se realizará una implementación de una DAO que se ejecute al completo sobre la red Blockchain.
 - Permitirá realizar los procesos de proposición, votación y ejecución de las propuestas, logrando aprobar las promesas electorales creadas.

3 Aspectos teóricos

En este capítulo, se describirán los aspectos tecnológicos en los que se basa este TFG, los cuales resultan fundamentales para entender las discusiones que se llevarán a cabo más adelante.

3.1 BLOCKCHAIN

La *Blockchain* o *cadena de bloques* es un tipo de tecnología de *libro mayor distribuido* o *DLT* (*Distributed Ledger Technology*) [4]. Un DLT consiste en una lista de nodos simplemente enlazada, donde cada nodo se enlaza con otro mediante funciones *hash* o *resumen* [5].

Una función hash es una operación que transforma datos de tamaño arbitrario a una cadena de tamaño fijo. Una función hash cumple con ciertas propiedades [6]:

- Dado un m es sencillo calcular su $h(m)$ pero, al contrario, m sea difícil de calcular dada un $y=h(m)$
- También se pide que sea difícil encontrar una pareja (m, y) con $m \neq y$ tal que $h(m) = h(y)$.
- Entre otras propiedades importantes se encuentran: bajo coste computacional, que sea determinista (un mensaje 'm' siempre tenga el mismo $h(m)$) y con efecto avalancha (un pequeño cambio en la entrada cambia completamente la salida).

Los nodos de la lista, llamados *bloques*, guardan las transacciones realizadas en la red, además de ciertos metadatos que verifican la validez de dicho nodo. Gracias a estos metadatos y a las propiedades de las funciones hash, se consigue que el bloque sea inalterable, verificable y trazable.

3.1.1 Características de una Blockchain

La base sobre lo que se fundamenta una red basada en esta tecnología es [5]:

- Una cadena de bloques segura.
- Red de pares (P2P, *peer-to-peer*).
- Transacciones o mensajes.
- Conjunto de reglas del algoritmo de consenso.
- Una máquina de estados que procese las transacciones de acuerdo con las reglas de consenso.
- Un incentivo por la generación de nuevos bloques que asegure la economía de las transacciones. Dicho incentivo son las denominadas criptomonedas.

- Contratos inteligentes

Cada uno de estos elementos se explicará en los siguientes apartados.

3.1.2 Red de Pares

Una *red de pares* [7] es un tipo de red descentralizada donde el conjunto de ordenadores y servidores (*nodos de la red*) no poseen una jerarquía fija, sino que se comportan como iguales. Todo el conjunto de nodos que conforman la red intercambia información bajo un mismo protocolo y consenso, normalmente sin necesidad de disponer de un tercero.

En este tipo de redes, cuanto mayor es el número de nodos conectados, mayor es la potencia computacional, lo que otorga protección a la propia red.

3.1.3 Tipos de redes que aplican la tecnología Blockchain

Existen tres tipos de redes que aplican la tecnología Blockchain, atendiendo a quién la administra [8]:

- **Redes Públicas:** cualquiera puede conectarse a dicha red e integrarse con ella. Ejemplos de redes públicas son Bitcoin [9], Ethereum [10] y Solana [11].
- **Redes Privadas** (o permissionadas): sus usuarios son identificados previamente antes de poder interactuar en la red. Se caracterizan por ser estos mismos usuarios quienes verifican las transacciones. Otra característica importante es que, al tratarse de una red privada, no existe el incentivo económico por la generación de bloques.
- **Redes Híbridas** (o federadas): este tipo de redes parten de la base de una red privada, pero haciendo uso de una red pública para guardar los identificadores de los bloques. En este tipo de redes suelen estar involucradas varias partes y hacen uso de la Blockchain como herramienta de transparencia y confianza.

A las redes que aplican la tecnología Blockchain se les denomina *protocolos*, un ejemplo: Ethereum es un protocolo que aplica la tecnología Blockchain y hace uso de una moneda virtual (criptomoneda), el Ether, para regular su red.

Otra posible clasificación de las redes, que atiende al ambiente de desarrollo Blockchain, es la siguiente:

- **Red Local:** red Blockchain que se ejecuta de manera local sin conexión al exterior.

- **Red de pruebas** (o *testnet*): red expuesta de manera pública que contiene varios nodos para la generación de bloques. Esta red permite obtener de manera gratuita la moneda que rige el protocolo para permitir realizar pruebas en un ambiente parecido a producción.
- **Red de Producción** (o *mainnet*): red principal de la Blockchain, es el entorno final.

3.1.4 Transacciones y Bloques

Toda acción producida en una red Blockchain se denomina *transacción*. Estas acciones son consecuencia de un cambio del estado de la red.

Estas transacciones son almacenadas en los bloques que conforman la Blockchain. Existen dos tipos de bloques:

- **Bloque génesis**: se trata del primer bloque de la Blockchain, no tiene padre y suele identificarse con el número cero.
- **Resto de bloques**: su padre es el anterior bloque producido en la Blockchain.

Cada bloque almacena las transacciones de la red, un identificador único, el hash del anterior bloque y un conjunto de metadatos. Es el conjunto de datos que almacena el bloque y la unión entre ellos lo que brinda la integridad y la trazabilidad de la red Blockchain. En la Ilustración 1 se puede ver la integración de los bloques en la cadena.

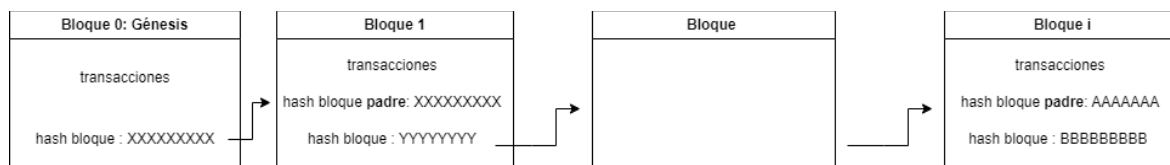


Ilustración 1: Generación de Bloques

El hash de un bloque dado se obtiene con el conjunto de datos de todas las transacciones dispuestas a validar/ejecutar, juntando el hash del bloque previo y ciertos metadatos que conforman el bloque. Esta operación resulta en una cadena criptográficamente unida, ya que si se cambia un valor dentro de un bloque cambiaría el resto. Solo existe una excepción para la generación, que es para el bloque cero (el génesis), que al no tener hash del nodo padre no requiere dicha generación.

3.1.5 Algoritmos de Consenso

Los *algoritmos de consenso* son el medio para que un sistema distribuido y sus usuarios se coordinen. En Blockchain, se utilizan para decidir qué bloque generado es el correcto o quién es el encargado

de crearlo. Es decir, se trata de un mecanismo para validar que la información del conjunto de todas las nuevas transacciones es correcta y posteriormente puedan quedar registradas en el libro mayor distribuido. Actualmente, existen varios tipos de algoritmos de consenso, entre los que se encuentran los siguientes:

- **Prueba de Trabajo** (*Proof of Work*, PoW) [12]: donde cada nodo de la red se pelea por crear y añadir el mismo bloque. El bloque que se elige es aquel que ha sido generado con anterioridad y es válido, y en caso de que se encuentre más de uno prevalecerá aquella cadena más larga. Este algoritmo es utilizado por la red Bitcoin y por Ethereum en sus inicios.
Un bloque es válido (para Bitcoin) en el momento en el que un nodo de la red encuentra un valor resumen de los datos de dicho bloque con una serie de ceros a la izquierda. Este número de ceros puede cambiar aumentando o decrementando la dificultad.
- **Prueba de Participación** (*Proof of Stake*, PoS) [13], [14]: es más eficiente, en cuanto a energía consumida se refiere respecto a la Prueba de Trabajo, ya que cada nuevo bloque es creado por un nodo de la red elegido al azar, considerando la cantidad de criptomonedas que poseen sobre el protocolo. Este algoritmo es utilizado actualmente por el protocolo Ethereum, donde los nodos dejan su Ether en manos de la red. Esta retención de fondos da permiso a los nodos a generar nuevos bloques y, con ello, obtienen la recompensa.
- **Prueba de Autoridad** (*Proof of Authority*, PoA) [14]: es un algoritmo de consenso donde los nodos de confianza, también llamadas validadores o autoridades, son entidades reales con una reputación, y de esta manera garantizan la confianza para validar los bloques generados. Este algoritmo es utilizado por la red Binance.

3.1.6 Criptomonedas y Altcoins

Las *criptomonedas* (*cryptocurrencies*) [15] son activos monetarios completamente digitales que, al igual que cualquier otra moneda, puede obtenerse, gastarse o invertirse, pero no dependen de los bancos o sistemas financieros.

Las criptomonedas nacen gracias a Bitcoin y a Satoshi Nakamoto [9] que, mediante dicha moneda, establecía una incentivo o recompensa para los nodos de la red, a los que se les denominó *mineros*. Los mineros son los encargados de validar o crear nuevos bloques para disponerlos en la red, reuniendo el conjunto de transacciones realizadas en un periodo anterior e introduciéndolo dentro de un bloque.

Como alternativa a Bitcoin han surgido numerosas criptomonedas, a las que se les denomina de manera genérica Altcoins[16]. Entre las Altcoins más populares se encuentran Ethers de Ethereum, Sol de Solana o Ada de Cardano.

3.1.7 Cuentas y Carteras (Wallets)

Para poder realizar transacciones en una red Blockchain es necesario tener una *cuenta*, gracias a la cual se le asigna al usuario una dirección única en la red. Este proceso se realiza mediante las *wallet* o *carteras* [17].

Una cartera es una herramienta software o hardware, que permite gestionar las criptomonedas de multitud de redes Blockchain. Estas carteras permiten no solo el envío de las monedas virtuales sino interactuar con las aplicaciones descentralizadas.

Las cuentas de los usuarios son creadas mediante criptografía de clave asimétrica, gracias a algoritmos como ECDSA [18] o EdDSA [19]. Algunas carteras vinculan la clave mediante un conjunto de 12 palabras (mnemónico), de esta manera se ayuda al usuario a recordar dicha clave mediante palabras que mediante un conjunto largo de dígitos.

Un ejemplo de cartera es MetaMask [20], que existe como una extensión de los navegadores web, la cual permite el uso de una cuenta para varias redes Blockchain y hace uso del mnemónico para la clave privada.

3.1.8 Contratos Inteligentes (Smart Contracts)

Los *contratos inteligentes* o *smart contracts* son programas inmutables que se ejecutan sobre un protocolo que aplica la tecnología Blockchain [5].

Cuando un contrato inteligente es desplegado en la red Blockchain, el código fuente subido a dicha red es inalterable y se le asocia a dicho contrato una dirección en la red para ser accesible. Si se despliega en una red Blockchain pública, se consigue que sea accesible por cualquier persona.

Como su invención se debe a la aparición de Ethereum, se explicará los contratos inteligentes con más detalle en el apartado 3.3.3.

3.1.9 Tokens

Un *token* [21] es una representación digital de un activo integrado dentro de la red Blockchain. Gracias a los contratos inteligentes, estos tokens pueden ser programados para representar bienes, monedas, recursos o derechos de acceso. Existen dos tipos de tokens:

- **Tokens Fungibles** (FT, *Fungible Token*): los tokens fungibles representan a aquellos que poseen las siguientes características: no son únicos, se pueden dividir y son consumibles.
- **Tokens No Fungibles** (NFTs, *Non-fungible Tokens*): en contra partida de los fungibles, este tipo de tokens son únicos y no son consumibles.

Ambos tipos de tokens pueden ser adquiridos por usuarios de la red.

3.2 APLICACIÓN DESCENTRALIZADA

Una *aplicación descentralizada* (*dApp*) [22] define a todas aquellas aplicaciones que se ejecutan gracias a una red descentralizada. Este sistema tiene dos partes fundamentales: un *backend*, formado por uno o más contratos inteligentes que implementan la lógica de negocio, y un *frontend*, que implementa el diseño de la interfaz de usuario.

Este conjunto de contratos inteligentes que forman la base de la aplicación se ejecutan sobre una red de pares, la cual es una red que aplica la tecnología Blockchain.

3.2.1 Conexión *frontend* con la Blockchain

Los contratos inteligentes, una vez compilados, disponen de un archivo ABI (*Application binary Interface*) [23], donde en un formato JSON (*JavaScript Object Notation*) se muestra el conjunto de funciones que posee el contrato.

Para la comunicación con los contratos en las aplicaciones descentralizadas, los usuarios deben conectarse a la red Blockchain. Para ello, las comunicaciones son establecidas por el protocolo JSONRPC [24]. Existen librerías como Web3.js [25] o Ethers.js [26] que simplifican estas llamadas y ayudan a los desarrolladores en la comunicación entre usuarios y la Blockchain.

3.3 ETHEREUM Y DAPPS

Ethereum surge como una red de propósito general donde no solo se registra el intercambio de su criptomoneda, sino que registra los cambios de cualquier tipo de dato dispuesto en la red. Además,

ofrece un modelo de plataforma para la programación de aplicaciones descentralizadas, haciendo uso de los contratos inteligentes.

Otro aspecto clave en Ethereum es el algoritmo de consenso, que en sus inicios se trataba del algoritmo de prueba de trabajo, pero a finales del 2022 se realizó el cambio al algoritmo de Prueba de Participación [27].

En los siguientes apartados, se mencionarán las características principales del protocolo Ethereum y se hará hincapié en aquellas que afecten al desarrollo de este TFG.

3.3.1 La Máquina Virtual de Ethereum

Ethereum es una gran red de nodos que ejecutan la misma máquina de estados, convirtiendo la red en una gran máquina distribuida de estado único (*singleton*).

La ejecución de programas almacenados, el permitir la lectura y grabación de datos sobre dicha máquina, convierte a Ethereum en una máquina “casi” Turing completa. Esta máquina puede ejecutar cualquier algoritmo que pueda ser calculado por cualquier máquina de Turing [5], salvo por una pequeña excepción, tiene un límite de ejecución.

Este límite de ejecución, denominado *gas*, se establece para solucionar el problema de los bucles infinitos, conocido como el problema de la parada (*the halting problem*).

La máquina encargada de procesar todos los datos es la denominada *Máquina Virtual de Ethereum* (*Ethereum Virtual Machine*, EVM) y cada nodo de la red ejecuta esta EVM.

3.3.2 Gas

El *gas* es una métrica que surge para resolver el problema de la parada. Bloquear la gran máquina de estados de Ethereum, ya sea con alguno de los dos algoritmos de consenso, es un problema que Ethereum solucionó con el gas, ya que bloquear un nodo que valida un conjunto de transacciones bloquearía la red, es decir, se tendría un ataque de Denegación de Servicio (DoS)[5].

Para evitar que un contrato se quede bloqueado en un nodo, Ethereum implementa esta métrica con la que mide cada instrucción de un contrato, de forma que cada instrucción tiene una medida numérica de coste. En el momento en que un usuario lanza una transacción, este usuario establece el valor límite dispuesto a gastar en gas (la comisión), el cual será adjudicado al minero. Cada vez que se ejecuta una instrucción, su coste en gas se va descontando del total adjudicado y se comprueba si se dispone aún de gas suficiente para continuar con la ejecución. En el momento en que se completa

la transacción sin acabar el gas, el sobrante es devuelto al emisor. El gas consumido es la propina del minero por validar la transacción. En el caso de que el gas se acabe en mitad de la transacción, se revierte al estado de la ejecución previo y el minero se queda con el gas utilizado.

El coste en gas se mide en Wei, una partición de la criptomoneda Ether. En la Tabla 1 se pueden ver las denominaciones del valor en Wei utilizados para indicar el coste del gas.

VALOR (EN WEI)	EXPONENTE	NOMBRE COMÚN	NOMBRE INTERNACIONAL	SISTEMA
1	1	Wei	Wei	
1000	10^3	Babbage	Kilowei o femtoether	
1000000	10^6	Lovelace	Megawei o picoether	
1000000000	10^9	Shannon	Gigawei o nanoether	
1000000000000	10^{12}	Szabo	Microether o micro	
1000000000000000	10^{15}	Finney	Milliether o milli	
1000000000000000000	10^{18}	Ether	Ether	
1000000000000000000000	10^{21}	Grand	Kiloether	
1000000000000000000000000	10^{24}		Megaether	

Tabla 1: Unidades del GAS [5]

3.3.3 Los Contratos Inteligentes de Ethereum

Para Ethereum, los contratos inteligentes son programas desarrollados para ejecutarse sobre la red de Ethereum. Estos programas están normalmente escritos en Solidity, un lenguaje de programación de alto nivel basado en C y JavaScript. Un ejemplo de programa escrito en este lenguaje se puede ver en la Ilustración 2.

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.18; // version del contrato

/// @title Prueba
/// @author Miguel Rodriguez González
/// @notice Guarda un valor
/// @dev almacena en la variable valor un nuevo dato
contract Prueba { // nombre del contrato

    uint256 valor; // estado del contrato
                  // en este caso una variable entera sin signo

    // funcion publica que modifica el estado
    function guardarSaludo(uint256 _nuevoValor) public {
        valor = _nuevoValor;
    }
}
```

Ilustración 2: Ejemplo de un contrato para almacenar un valor en Blockchain

Como se ha mencionado anteriormente, estos contratos se ejecutan sobre la EVM y cada ejecución de una instrucción resta gas. Pero este código de alto nivel no es lo que entiende la EVM, por lo que primero se debe traducir.

Solidity posee un compilador denominado Solc que permite compilar los contratos en los bytes necesarios que entiende la EVM, a los que se les denomina *ByteCode*. [5] Este compilador permite ver una estimación del consumo de gas del contrato, tanto para su despliegue como la ejecución de las diferentes funciones, tal y como se aprecia en la Ilustración 3:

```
miguel@ANDUIN:~/solidity_compiler_pruebas$ solc --gas
prueba.sol
===== prueba.sol:Prueba =====
Gas estimation:
construction:
  99 + 45400 = 45499
external:
  guardarSaludo(uint256):      22498
```

Ilustración 3: Coste en gas del contrato de ejemplo

3.3.4 Bloques en Ethereum

La Blockchain de Ethereum comenzó con su bloque génesis, ya minado en 2015 [28], definiendo el comienzo de este protocolo y de la criptomoneda que la regula.

Un bloque en Ethereum almacena los siguientes metadatos [5]:

- **timestamp**: hora en la que se minó el bloque.
- **blockNumber**: la longitud de la Blockchain en número bloques.
- **baseFeePerGas**: la comisión mínima por gas necesaria para que una transacción se incluya en el bloque.
- **difficulty**: el esfuerzo empleado para minar el bloque.
- **totalDifficulty**: esfuerzo acumulado con el resto de los bloques.
- **mixHash**: un identificador único del bloque.
- **parentHash**: el identificador único del bloque anterior (esta es la manera en la que los bloques son enlazados en la cadena).
- **transactions**: listado de las transacciones incluidas en el bloque.
- **stateRoot**: el estado entero del sistema: los saldos de las cuentas, el almacenamiento de contratos, el código de contratos y los nonces de las cuentas.
- **nonce**: un hash que, cuando se combina con mixHash, comprueba que el bloque ha pasado por una prueba de trabajo.

Los mineros son los encargados de la generación de estos bloques, como ya se ha mencionado con anterioridad.

Debido a que el algoritmo de consenso ha cambiado a un algoritmo de Prueba de Participación, los metadatos *mixHash* y *nonce* correspondientes a la prueba de trabajo ya no son necesarios. Por ello, si se verifica en la plataforma Etherscan¹ [29] uno de los últimos bloques generados a fecha 03/02/2023 (por ejemplo, el número de bloque 16548688 [30]) se observa que estos valores son cero.

3.3.5 El estado de Ethereum

El *estado* o *state* [31] en la red de Ethereum, hace referencia al conjunto de todos los datos de la red. Este estado se encarga de guardar desde los saldos de las cuentas hasta el código de los contratos desplegados en la red.

No se adentrará en cómo se estructura este estado, ya que se saldría del alcance del TFG, pero en una visión global se trataría de un gran árbol [32].

¹ Etherscan es una plataforma gratuita para la consulta de cuentas, bloques y transacciones.

3.3.6 Transacciones y Cuentas en Ethereum

En el protocolo Ethereum existen dos tipos de cuentas, las de los usuarios y la de los contratos inteligentes. Las cuentas de los usuarios son las denominadas *cuentas de propiedad externas* o *EOA* (*Externally Owned Accounts*), por las que se les asigna una dirección en la red y les permite tener un balance de Ethers. Las cuentas de los contratos, que dependen de la cuenta EOA de la persona que los despliega, también poseen una dirección de Ethereum y son capaces de tener asociado un balance de Ethers.

Las transacciones en Ethereum son acciones que se realizan sobre la red Blockchain, iniciadas siempre desde una cuenta EOA. Un ejemplo de transacción es el envío de Ether de una dirección a otra.

Una transacción se compone de los siguientes datos:

- **Nonce**: un número de secuencia generado por la cuenta que lo emite, previniendo transacciones duplicadas.
- **Precio gas** (en *wei*): el precio en gas que el emisor está dispuesto a pagar.
- **Destinatario**: dirección del destinatario de la acción de Ethereum.
- **Valor**: cantidad de Ether para la transacción (si es necesario).
- **Datos**: conjunto de datos de la transacción, de longitud variable.
- **v, r, s**: son las tres variables que componen la firma digital generada por el ECDSA (Algoritmo de firma digital de curva elíptica) correspondiente a la dirección de origen (la EOA).

Las cuentas EOA son accedidas mediante las carteras, definidas previamente en el apartado 3.1.7. En Ethereum, estas carteras se encargan de generar y almacenar de manera segura las direcciones de la red mediante el algoritmo ECDSA.

3.3.7 Estándares de Ethereum

En Ethereum existen propuestas para la mejora continua del protocolo, permitiendo adaptarse a nuevas tecnologías o a nuevos usos sobre este protocolo, y todo ello comienza con una *Propuesta de Mejora* o *EIP* (*Ethereum Improvement Proposal*) [33]. Una vez discutidas por la comunidad, pueden llegar a aprobarse y establecer un nuevo cambio en la red o crear una base para un nuevo producto sobre el protocolo. Las propuestas que se aprueban se convierten en *Solicitud de Comentarios para Ethereum* o *ERC* (*Ethereum Requests for Comments*) [5].

Uno de los objetivos de este TFG es lograr almacenar datos en la Blockchain y, debido a ello, es necesario investigar los estándares. Tanto la investigación como una posible implementación de un ERC daría al TFG una base conocida y segura, permitiendo a otros desarrolladores entender su funcionamiento.

Los ERC permiten la definición de tokens. Los tokens, como ya se mencionó, pueden representar recursos, arte digital, votaciones o certificados. Los estándares para la generación de tokens destacables para este TFG son:

- **EIP-20** [34]: propuesta para la estandarización de generación, transferencia y aprobación de tokens.
- **EIP-721** [35]: propuesta de estándar para la transferencia y rastreabilidad de Tokens No Fungibles o NFTs.
- **EIP-777** [36]: propuesta para la mejora del EIP-20, ofreciendo mayor control sobre la transferencia de tokens.
- **EIP-1155** [37]: propuesta de estándar para el manejo de los dos tipos de tokens anteriores, es decir, tanto fungibles como no fungibles, además de una serie de funciones nuevas
- **EIP-5114** [38]: (*Soulbound Badget*) propuesta de estándar de token asociado a un token no fungible sin permiso de transferencia.

Ligadas a estas propuestas se tienen dos muy importantes, que son base para los estándares: EIP-165 [39], para la identificación de interfaces en contratos inteligentes, y EIP-214 [40], que añade una nueva instrucción sobre la EVM para permitir hacer llamadas a otros contratos sin cambiar el estado de las variables del contrato que se llama. Gracias a estos EIPs es posible que un contrato se identifique con una o más interfaces [41].

3.4 ORGANIZACIONES AUTÓNOMAS DESCENTRALIZADAS (DAOS)

Una *Organización Autónoma Descentralizada* o *DAO* (*Decentralized Autonomous Organization*) [42] es una organización que, apoyándose en la tecnología Blockchain, utiliza algoritmos para controlarse y autogestionarse. Estos algoritmos son un conjunto de contratos inteligentes donde se definen las reglas que regularán la organización, logrando organizaciones autónomas, autogestionadas, más transparentes y eficientes. Gracias a las DAO, se puede lograr que el conjunto de personas involucradas tengan la misma igualdad y oportunidad de voto sobre cualquier propuesta.

El objetivo principal de las DAO es la toma de decisiones. Un ejemplo sería la decisión sobre la inversión en criptomonedas o en NFTs.

3.4.1 Fundamentos de las Organizaciones Autónomas Descentralizadas.

Existen cuatro funciones básicas (reglas, o mecanismos) sobre las que se rigen las DAO [43]:

- **Programar acciones ejecutadas atendiendo a sus parámetros de entrada:** la organización debe constar de un conjunto de reglas que gobiernen la DAO, lo que se consigue por medio de los contratos inteligentes que definen funciones como si de acciones o comportamientos se tratasen.
- **Protocolo de consenso:** toda decisión tomada, ya sea de conformidad o de rechazo, debe ser consensuada entre las partes involucradas. Además, se puede indicar qué conjunto de votantes es necesario para que dicha votación salga adelante.
- **Protocolo para la participación de votantes:** se debe definir cómo se quiere que el conjunto de miembros de la organización emita dicho voto para poder luego ser contabilizados.
- **Grabación de cualquier suceso que ocurra en la DAO:** se logra debido el carácter de trazabilidad e inmutabilidad de la tecnología Blockchain y los contratos inteligentes sobre los que se ejecuta la DAO.

3.4.2 Flujo Base de una Organización Autónoma Descentralizada

Una DAO creada en la Blockchain sigue unos eventos predecibles, como se aprecia en la Ilustración 4. Las distintas fases son:

- **Fase de propuesta y discusión:** un usuario plantea una propuesta con el fin de que ocurra cierta acción sobre la DAO. El conjunto de usuarios discutirá durante un tiempo marcado dicha propuesta antes de pasar a la fase de votación.
Un ejemplo de propuesta sería la aprobación de una promesa electoral.
- **Fase de votación:** en este periodo de tiempo se permite la votación sobre la propuesta generada. Al finalizar dicho periodo, se contabilizan los votos y la promesa cambia a un estado de aceptada o rechazada.
- **Fase de aprobación:** si la propuesta ha sido aprobada se ejecutará la acción correspondiente, la cual estará programada en el conjunto de contratos inteligentes sobre los que gobierna la DAO.

- **Fase de rechazo:** si la propuesta no ha sido aprobada, se descarta.

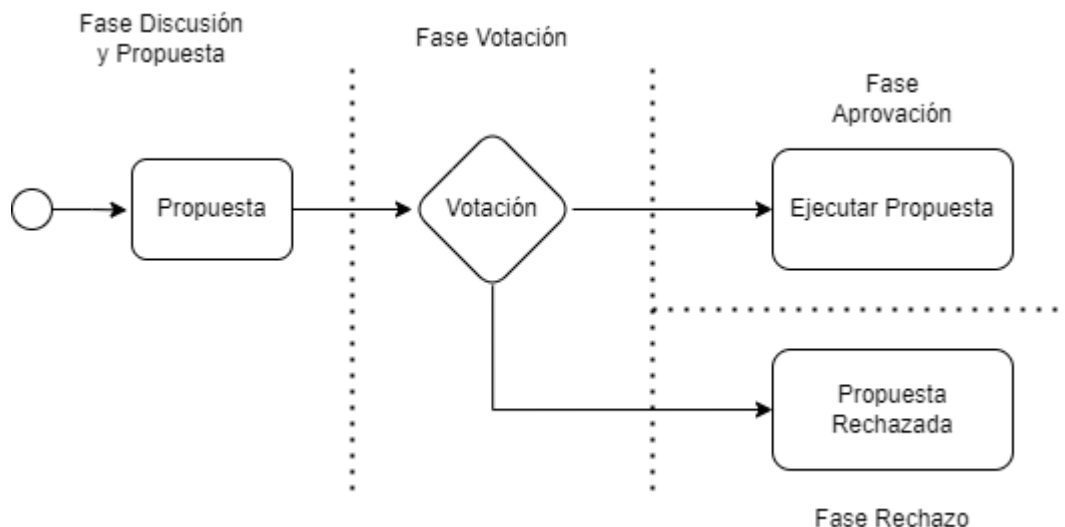


Ilustración 4: Diagrama de votación

3.4.3 Tipos de Votación en la Organización Autónoma Descentralizada

Para la fase de votación en este tipo de organizaciones existen dos tipos de votación: votación *on-chain* y votación *off-chain*, es decir, votar dentro o fuera de la Blockchain [42].

- **On-chain:** Una votación sobre la cadena implica que cada votación realizada es enviada como transacción sobre la red Blockchain donde se encuentre la DAO, teniendo en cuenta el coste por transacción.
- **Off-chain:** Una votación fuera de la cadena implica la utilización de un sistema descentralizado que permita de manera gratuita realizar transacciones seguras sobre ella, logrando que los usuarios voten sin coste. Finalmente, una vez finalizado el periodo de votación, un tercer sistema (también descentralizado) se encargaría de recoger las votaciones emitidas para grabarlas en la Blockchain y poder ser contabilizadas.

3.5 CONTROL DE VERSIONES

El control de versiones para el software será Git [44]. Esto se debe al marco de colaboración entre la Universidad de Oviedo y la empresa HP SDGS, la cual pide utilizar GitLab [45] y por ende, Git.

GitLab es un servicio para alojar repositorios Git y permitir a los desarrolladores supervisar, probar y desplegar su código.

Este servicio también ofrece un tablero de análisis donde se mostrarán las tareas (historias de usuario) y su estado, como se muestra en la Ilustración 5, esto será de ayuda en el desarrollo de este TFG.

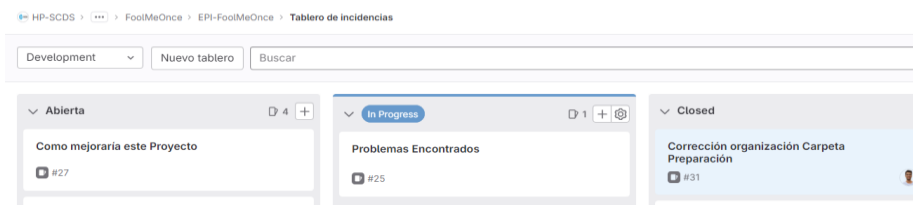


Ilustración 5: Tablero de tareas

3.6 METODOLOGÍA DE TRABAJO

La metodología de trabajo asignada para este TFG es la metodología ágil *Scrum* [46]. Esto se debe al marco de colaboración entre la Universidad de Oviedo y la empresa HP SCDS, la cual pide seguir dicha metodología.

3.6.1 Scrum

Scrum es un marco de gestión de proyectos que se aplica en la fase de desarrollo del ciclo de vida del software. Aplica un enfoque para la toma de decisiones basado en el desarrollo iterativo e incremental.

Esta metodología divide el proyecto en el tiempo, estableciendo iteraciones pequeñas que se resolverán mediante el enfoque incremental, de manera que se aumente el valor del producto en cada incremento.

Cada incremento se le denomina *sprint*, los cuales tienen una duración de entre dos y cuatro semanas. En cada sprint se desarrollará un conjunto de requisitos (*sprint backlog*), que serán elegidos por prioridad del total de estos (lo que se conoce como *product backlog*). Al finalizar el *sprint* se logra tener un producto software funcional que percibirá el cliente.

Scrum define tres roles de participantes:

- **Product Owner:** representa a las partes interesadas (*stakeholders*), puede ser el cliente o los propios usuarios. Se encargan de establecer los requisitos y sus prioridades en el *product backlog*.

Para este TFG este rol será ejercido por el tutor de la empresa HP SDSCS.

- **Scrum Master:** se trata del líder del equipo, vela por seguir la metodología y por cumplir los objetivos establecidos.

Para este TFG este rol será ejercido por la tutora de la Universidad de Oviedo.

- **Development team:** grupo de profesionales responsables de desarrollar las historias de usuario en cada *sprint*. Cada requisito vendrá

Para este TFG el equipo de desarrollo será ejercido por el alumno que realiza este TFG.

La primera fase de Scrum comienza con el arranque del proyecto donde se tienen los siguientes procesos:

- Identificación de los objetivos del proyecto.
- Identificación del Scrum Master y los *stakeholders*.
- Formación del equipo.
- Desarrollo de las épicas (*milestones*).

Siguientes fases de Scrum, que se repiten:

- **Planificación y estimación del *sprint*:** en esta fase se creará el *sprint backlog* con las nuevas tareas identificadas, junto a su prioridad y estimación.
- **Implementación:** en esta fase se realiza el desarrollo con el fin de obtener la entrega del *sprint*. Los involucrados realizarán una reunión diaria de pocos minutos donde el equipo actualizará el estado de sus tareas.
- **Revisión y retrospectiva del *sprint*:** una vez todo este implementado, se deberá realizar una revisión de los procesos desarrollados, normalmente mediante una demostración. También se realizará un estudio interno para indicar posibles mejoras de cara al siguiente *sprint*.

En este TFG debido a que el equipo de desarrollo está formado por una única persona, las reuniones diarias no son necesarias. También hay que indicar que el periodo de *sprint* será de dos semanas.

Una vez finalizado el *sprint*, se realizará la reunión final donde participaran el desarrollador, la tutora de la Universidad de Oviedo y el tutor de la empresa HP SCDS. En dicha reunión, se indicará lo logrado en el *sprint*, indicando si se han cumplido o no los criterios de aceptación, y finalmente, se establecerán los nuevos hitos para el próximo *sprint*.

4 Estudios previos y análisis de alternativas

En la actualidad, existen multitud de redes Blockchain y diferentes tecnologías que se podrían utilizar para desarrollar este TFG. En este capítulo, se llevará a cabo un análisis de distintas alternativas y se describirán las decisiones tomadas.

4.1 ANÁLISIS DE ALTERNATIVAS

4.1.1 Elección del protocolo que aplica la tecnología Blockchain

El primer punto esencial para este TFG es la elección del protocolo a utilizar, ya que se trata de la tecnología base que servirá como cimientos de la plataforma a crear. Para este análisis se valorarán los siguientes aspectos clave:

- La eficiencia del algoritmo de consenso.
- Permita la ejecución de contratos inteligentes.

4.1.1.1 Bitcoin

Bitcoin utiliza el algoritmo de consenso de Prueba de Trabajo, lo que lo hace costoso en términos de energía. Esta Blockchain permite el uso de scripts [47], pero su lenguaje no fue pensado para el desarrollo de aplicaciones, ya que no permite el uso de bucles.

4.1.1.2 Ethereum

Este protocolo ha sido migrado del algoritmo de consenso Prueba de Trabajo a Prueba de Participación [27], lo que en términos de energía consumida es más eficiente. Además, este protocolo permite ejecutar programas gracias a lenguajes como Solidity y Vyper [5].

4.1.1.3 Solana

Solana también hace uso de la Prueba de Participación. Los lenguajes de programación utilizados para el desarrollo de contratos inteligentes son C y Rust [11].

4.1.1.4 Cardano

Cardano hace uso de una variante del algoritmo de consenso Prueba de Participación denominada Ouroboros [48]. Esta variante destaca por su velocidad en la validación de bloques.

El lenguaje de programación que se utiliza para los contratos es Plutus [49], el cual está basado en una librería de Haskell, lo que lo hace muy robusto para la alta seguridad.

4.1.1.5 Alternativa elegida

Ethereum es la red escogida para la base de este TFG, debido a:

- Su gran comunidad, lo que permite la consulta de dudas en Internet.
- Su bajo consumo energético.
- Disponer de dos lenguajes de programación de alto nivel.

4.1.2 Elección de lenguaje de programación para Ethereum

Para implementar los contratos inteligentes a desplegar en Ethereum se analizarán los lenguajes Solidity y Vyper. Para este análisis se valorarán los siguientes aspectos clave:

- La documentación de los lenguajes.
- La comunidad detrás de ella.
- El número de actualizaciones que recibe.

4.1.2.1 Solidity

Solidity [50] es un lenguaje de programación de alto nivel tipado con influencia de los lenguajes C++ y JavaScript. Otras características importantes son: la alta frecuencia con la que se actualiza el lenguaje, su amplia documentación, permite la herencia y la utilización de librerías externas.

Para este lenguaje existe un servicio de código gratuito y libre denominado OpenZeppelin [51]. Este servicio ofrece las interfaces de los ERCs y ejemplos de implementaciones, además de otro tipo de contratos seguros y probados por la comunidad.

Asimismo, también existe un entorno de desarrollo (IDE) llamado Remix IDE [52]. Este IDE permite el despliegue y depuración de contratos inteligentes desarrollados en Solidity.

4.1.2.2 Vyper

Vyper [53] se trata de un lenguaje basado en Python, con un fuerte tipado, que contiene menos características que Solidity. No permite la herencia ni los modificadores de funciones, debido a que su objetivo es la creación de contratos seguros y sencillos de auditar. Por otro lado, no se actualiza tan frecuentemente como Solidity.

Otra característica de Vyper es su bajo coste de despliegue, en comparación con los contratos desarrollados en Solidity. Este bajo coste se debe a que Vyper no permite el uso de memoria dinámica por lo que, en el despliegue, no se necesita reservar memoria dinámica lo que reduce el gasto en gas del despliegue [54].

4.1.2.3 Alternativa elegida

Solidity es la alternativa elegida, debido a la gran documentación que existe, al servicio que ofrece OpenZeppelin y su alto número de actualizaciones.

4.1.3 Elección de la tecnología *frontend*

Entre la variedad de tecnologías aplicadas al *frontend*, se barajan dos tecnologías principales: Angular [55] y React [56]. Para este análisis se tendrán en cuenta los siguientes aspectos clave:

- La curva de aprendizaje debe ser sencilla.
- El conjunto de archivos finales deben ser ligeros.

4.1.3.1 Angular

Angular es un *framework* desarrollador por Google [57], basado en el lenguaje TypeScript. Su aprendizaje es más complejo debido a la cantidad de conceptos que introduce, como la inyección de dependencias o la vinculación del modelo de datos y la vista. Además, tampoco destaca por la ligereza de los archivos que descarga en el navegador web [58].

Angular posee herramientas integradas para el desarrollo y pruebas, además de contar con *angular cli* [59], el cual permite la generación de la estructura básica de un nuevo componente de manera rápida.

4.1.3.2 React

React es una librería desarrollada por la compañía Meta [60] para el diseño de interfaces, sencillo de aprender, ya que utiliza el lenguaje JavaScript para el desarrollo de componentes y destaca por ser una librería ligera [58].

También se apoya en un DOM (*Document Object Model*) virtual que permite una actualización y renderizado más rápidos que otros *frameworks*.

4.1.3.3 Alternativa Elegida

React es la alternativa elegida, debido a que la plataforma *frontend* será ligera y se necesita un tiempo de aprendizaje que sea lo menor posible.

4.1.4 Elección de un entorno de despliegue eficiente de contratos

Se quiere disponer de un entorno que permita la interacción rápida con contratos inteligentes, con una curva de aprendizaje corta y que permita depurar los contratos tras su despliegue.

El único existente que permite realizar este tipo de acciones es Remix IDE.

4.1.5 Elección del estándar para la creación de tokens en el contrato desarrollado.

Para la elección del estándar que permita la creación de promesas electorales, se analizarán los EIPs mencionados en el apartado 3.3.7. Para este análisis se valorarán los siguientes aspectos clave:

- Creación de tokens exclusivos y no gastables.
- No permita el traspaso de los tokens entre usuarios.
- Sea un estándar aprobado por la comunidad.

4.1.5.1 ERC-20

El estándar ERC-20 [34] brinda una interfaz que permite crear tokens fungibles que funcionan como una moneda digital, por lo tanto, no son exclusivos. Su interfaz permite la transferencia y delegación de tokens, como se muestra en Ilustración 6.

```
interface IERC20 {  
  
    event Transfer(address indexed _from, address indexed _to, uint256 _value)  
    event Approval(address indexed _owner, address indexed _spender, uint256 _value)  
  
    function name() public view returns (string)  
    function symbol() public view returns (string)  
    function decimals() public view returns (uint8)  
    function totalSupply() public view returns (uint256)  
    function balanceOf(address _owner) public view returns (uint256 balance)  
    function transfer(address _to, uint256 _value) public returns (bool success)  
    function transferFrom(address _from, address _to, uint256 _value) public returns (bool success)  
    function approve(address _spender, uint256 _value) public returns (bool success)  
    function allowance(address _owner, address _spender) public view returns (uint256 remaining)  
}
```

Ilustración 6: Interfaz ERC-20

4.1.5.2 ERC-721

El estándar ERC-721 permite crear tokens no fungibles (NFTs) y brinda una interfaz que permite la creación de tokens exclusivos. Sus operaciones son parecidas al estándar ERC-20 y se muestran en: Ilustración 7.

```
interface ERC721 {  
  
    event Transfer(address indexed _from, address indexed _to, uint256 indexed _tokenId);  
    event Approval(address indexed _owner, address indexed _approved, uint256 indexed  
    event ApprovalForAll(address indexed _owner, address indexed _operator, bool _approved);  
  
    function balanceOf(address _owner) external view returns (uint256);  
    function ownerOf(uint256 _tokenId) external view returns (address);  
    function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data)  
    external payable;  
    function safeTransferFrom(address _from, address _to, uint256 _tokenId) external payable;  
    function transferFrom(address _from, address _to, uint256 _tokenId) external payable;  
    function approve(address _approved, uint256 _tokenId) external payable;  
    function setApprovalForAll(address _operator, bool _approved) external;  
    function getApproved(uint256 _tokenId) external view returns (address);  
    function isApprovedForAll(address _owner, address _operator) external view returns (bool);  
}
```

Ilustración 7: Interfaz ERC-721

Otra característica importante de este estándar es un contrato desarrollado por la comunidad de OpenZeppelin llamado ERC721UriStorage [61]. Este contrato permite añadirle un identificador de

recursos uniforme (URI) como propiedad a un NFT, dando la posibilidad de vincular el NFT a un archivo guardado fuera de la Blockchain. El código de ERC721UriStorage se muestra en la Ilustración 8.

```
abstract contract ERC721UriStorage is ERC721 {
    using Strings for uint256;

    // Optional mapping for token URIs
    mapping(uint256 => string) private _tokenURIs;

    function tokenURI(uint256 tokenId) public view virtual override returns (string memory)
    {
        _requireMinted(tokenId);

        string memory _tokenURI = _tokenURIs[tokenId];
        string memory base = _baseURI();

        // If there is no base URI, return the token URI.
        if (bytes(base).length == 0) {
            return _tokenURI;
        }
        // If both are set, concatenate the baseURI and tokenURI (via abi.encodePacked).
        if (bytes(_tokenURI).length > 0) {
            return string(abi.encodePacked(base, _tokenURI));
        }

        return super.tokenURI(tokenId);
    }

    function _setTokenURI(uint256 tokenId, string memory _tokenURI) internal virtual {
        require(_exists(tokenId), "ERC721UriStorage: URI set of nonexistent token");
        _tokenURIs[tokenId] = _tokenURI;
    }

    function _burn(uint256 tokenId) internal virtual override {
        super._burn(tokenId);

        if (bytes(_tokenURIs[tokenId]).length != 0) {
            delete _tokenURIs[tokenId];
        }
    }
}
```

Ilustración 8: Clase ERC721UriStorage

4.1.5.3 ERC-1155

Este estándar une los estándares ERC-20 y ERC-721 en una misma interfaz, permitiendo generar tokens ERC-20, ERC-721 o una combinación de ambos [62].

Entre sus características destaca el tratamiento por lotes, lo que permite a los usuarios realizar una misma operación sobre un conjunto de tokens desde su creación hasta su transferencia, como se muestra en la Ilustración 9.

```
interface ERC1155 {

    event TransferSingle(address indexed _operator, address indexed _from, address indexed
    _to, uint256 _id, uint256 _value);
    event TransferBatch(address indexed _operator, address indexed _from, address indexed _to,
    uint256[] _ids, uint256[] _values);
    event ApprovalForAll(address indexed _owner, address indexed _operator, bool _approved);
    event URI(string _value, uint256 indexed _id);

    function safeTransferFrom(address _from, address _to, uint256 _id, uint256 _value, bytes
    calldata _data) external;
    function safeBatchTransferFrom(address _from, address _to, uint256[] calldata _ids,
    uint256[] calldata _values, bytes calldata _data) external;
    function balanceOfBatch(address[] calldata _owners, uint256[] calldata _ids) external view
    returns (uint256[] memory);
    function setApprovalForAll(address _operator, bool _approved) external;
    function isApprovedForAll(address _owner, address _operator) external view returns (bool);
}
```

Ilustración 9: Interfaz ERC-1155

4.1.5.4 EIP-5114

EIP-5114 aún sigue siendo un EIP, ya que no ha alcanzado el nivel requerido de discusión para convertirse en estándar de la red. Su objetivo es permitir la creación de tokens exclusivos y que no puedan ser transferidos. Este EIP propone las funciones que se muestran en la Ilustración 10.

```
interface IERC5114 {

    event Mint(uint256 indexed tokenId, address indexed nftAddress, uint256 indexed nftTokenId);

    function ownerOf(uint256 index) external view returns (address nftAddress, uint256 nftTokenId);
    function collectionUri() external pure returns (string collectionUri);
    function tokenUri(uint256 tokenId) external view returns (string tokenUri);
    function metadataFormat() external pure returns (string format);
}
```

Ilustración 10: Interfaz ERC-5114

4.1.5.5 ERC-777

ERC-777 mejora el estándar ERC-20, proporcionando la notificación al usuario receptor tras una transferencia, entre otras mejoras. Por lo tanto, se trata de un estándar que no permite la generación de tokens exclusivos. Sus funciones son las mostradas en la Ilustración 11.

```
interface IERC777 {

    event Minted(address indexed operator, address indexed to, uint256 amount,
        bytes data, bytes operatorData);
    event Burned(address indexed operator, address indexed from, uint256 amount,
        bytes data, bytes operatorData);
    event AuthorizedOperator(address indexed operator, address indexed tokenHolder);
    event RevokedOperator(address indexed operator, address indexed tokenHolder);

    event Sent(address indexed operator, address indexed from, address indexed to,
        uint256 amount, bytes data, bytes operatorData
    );

    function name() external view returns (string memory);
    function symbol() external view returns (string memory);

    function granularity() external view returns (uint256);
    function totalSupply() external view returns (uint256);
    function balanceOf(address owner) external view returns (uint256);
    function send(address recipient, uint256 amount, bytes calldata data) external;
    function burn(uint256 amount, bytes calldata data) external;
    function isOperatorFor(address operator, address tokenHolder) external view returns (bool);
    function authorizeOperator(address operator) external;
    function revokeOperator(address operator) external;
    function defaultOperators() external view returns (address[] memory);
    function operatorSend(address sender, address recipient, uint256 amount,
        bytes calldata data, bytes calldata operatorData ) external;
    function operatorBurn(address account, uint256 amount, bytes calldata data,
        bytes calldata operatorData) external
}
```

Ilustración 11: Interfaz ERC-777

4.1.5.6 Alternativa elegida

ERC-721 es la alternativa elegida. Aunque no cumple con el aspecto de no permitir la transferencia, soporta la generación exclusiva de tokens no consumibles y es un estándar aprobado por la comunidad.

Por lo tanto, para cumplir con todos los objetivos se eliminará de la implementación la posibilidad de transferir tokens de este estándar, logrando así crear promesas electorales exclusivas y unidas al propietario.

4.1.6 Elección del sistema de archivos descentralizado

Para la elección del sistema de archivos descentralizado que permita almacenar datos de manera externa a la Blockchain se analizarán IPFS [63] y Swarm [64]. Para este análisis se valorarán los siguientes aspectos clave:

- Posea una gran documentación.
- Posea una librería para conectarse al sistema.
- Su servicio sea gratuito.

4.1.6.1 IPFS

IPFS (*Inter-Planetary File System*) es un protocolo de archivos descentralizado, donde cada contenido tiene asociado una dirección única, gracias a las funciones hash. Estos archivos son distribuidos entre los nodos de la red de pares que conforman IPFS.

Este protocolo posee bastante documentación y una gran comunidad, gracias a estar disponible desde 2014. Además, posee una librería llamada *ipfs-http-client* [65] que permite conectar una aplicación con un nodo del protocolo para almacenar y consultar archivos de manera gratuita.

4.1.6.2 Swarm

Se trata de un servicio muy parecido a IPFS, creado por la Fundación Ethereum. Al igual que IPFS, permite almacenar archivos direccionables y posee una librería que permite conectarse a un nodo para almacenar y consultar archivos, llamada *bee.js* [66].

El almacenamiento en este servicio requiere de sellos postales (*Postage Stamps*) [67]. Estos sellos se deben de comprar [68]. Por lo tanto, no se trata de un servicio gratuito.

4.1.6.3 Alternativa Elegida

IPFS es la alternativa elegida, debido a su servicio gratuito y la gran documentación asociada.

4.1.7 Elección de las herramientas para el desarrollo de contratos inteligentes

Para el desarrollo de este TFG, se deberá analizar los entornos de desarrollo Hardhat [69] y Truffle [70], y las librerías de conexión entre el usuario y los contratos inteligentes Web3.js [25] y Ethers.js [26].

4.1.7.1 Análisis de herramientas para el desarrollo

Las herramientas Truffle y Hardhat ofrecen un entorno de desarrollo, pruebas y despliegue de contratos. Para este análisis se valorarán los siguientes aspectos clave:

- Posea una Blockchain interna para realizar test.
- La capacidad de realizar pruebas.
- Incluya librerías que ayuden al desarrollo.

4.1.7.1.1 Truffle

Truffle no posee una red Blockchain interna, pero sus creadores ponen una herramienta en sustitución denominada Ganache [71], que actúa como entorno local de desarrollo simulando la red de Ethereum.

Una característica importante de esta herramienta es su base de datos [72], denominada *Truffle db*. Esta base de datos almacena el histórico de cambios de los contratos inteligentes.

4.1.7.1.2 Hardhat

Hardhat si posee una red Blockchain interna que permita realizar pruebas y despliegues. Respecto a las pruebas, proporciona la medición de su cobertura sobre el código de los contratos inteligentes. Además, la biblioteca Ethers viene incluida con la instalación de Hardhat [73].

4.1.7.2 Análisis de librerías de conexión entre los usuarios y los contratos

Las herramientas Web3.js y Ethers.js ofrecen una manera de interactuar con los contratos inteligentes. Para este análisis se valorarán los siguientes aspectos clave:

- La libertad de licencia del Código.
- Permita el desarrollo en TypeScript
- Permita la conexión con otras redes Blockchain que tengan como base la EVM (Máquina Virtual de Ethereum).

4.1.7.2.1 Web3.js

Web3.js fue la primera de las librerías en nacer a manos de la propia fundación de Ethereum. Esta librería posee una licencia GPL [25]. Permite el uso de TypeScript como lenguaje de desarrollo y permite conectarse a otras redes Blockchain similares a la EVM.

4.1.7.2.2 Ethers.js

Ethers posee una licencia MIT [74], permite el uso de TypeScript y una conexión con otras redes Blockchain similares a la EVM. Además, se trata de una librería más ligera que Web3.js [26].

Otro punto importante es la integración que tiene Ethers con otros entornos para el desarrollo, como ocurre con Hardhat.

4.1.7.3 Alternativa Elegida

Hardhat y Ethers.js son las alternativas elegidas, debido a la integración entre estas dos herramientas, la Blockchain interna que ofrece Hardhat y su cobertura de pruebas.

4.1.8 Análisis del protocolo de participación de votantes para las DAO

En este apartado, se analizarán los tipos de protocolos que existen para la votación sobre una DAO. Para este análisis se valorarán los siguientes aspectos clave:

- La sencillez de realizar una implementación.
- No sea necesario terceros sistemas para implementar la DAO.

4.1.8.1 Emisión de tokens de gobernanza

La emisión de tokens de gobernanza, que supone un tipo de votación *on-chain*, se puede lograr mediante estándares como el ERC-20 o ERC-721, de manera que se asocia un voto a un token.

Este método implica que el usuario estaría comprando su voto, dejando el poder de voto en aquellos con mayor poder adquisitivo. En contraposición, se tendría un sistema sencillo de implementar. Además, existe una base de implementación de gobierno en OpenZeppelin, lo que ayudaría en el desarrollo.

4.1.8.2 Skin in the Game

El fundador de Ethereum, Vitalik Buterin, menciona en su blog esta solución, partiendo de la crítica hacia las DAO que incentivan o funcionan mediante los tokens de gobernanza [75].

La emisión de tokens de gobernanza implica otorgar el poder de voto a aquellos con mayor poder económico y permite a los inversores en tokens especular con su valor.

Por todo ello, Buterin propone que el propio voto tenga consecuencias, es decir, una vez generado un voto, este será grabado en la red y en el caso de que dicho voto haya supuesto una mala decisión para la comunidad, el poder de voto podría ser quitado al usuario.

Si se traslada este mecanismo a una DAO como la que se quiere plantear en este TFG, sería muy subjetivo valorar que un voto fue "malo" para una promesa electoral.

4.1.8.3 Prueba de Participación

Otro problema de la emisión de tokens reside en que la DAO no sabe cuántas direcciones están asociadas a una única persona. Lo ideal sería tener una sola dirección y un solo voto por persona, al igual que sucede en unas votaciones electorales como se muestra en Ilustración 12.



Ilustración 12: Asociación de una persona, un voto

El problema reside en cómo se verifica que dicha persona no tiene más de una cuenta para votar. Se trata del conocido problema como el ataque Sybil [76].

Una manera de verificar esta prueba de participación es utilizando datos que justifiquen que una persona solo posee una dirección de red. Una solución es la verificación que se realiza con los *exchange*², donde se pide documentación oficial y una imagen de la persona, permitiendo así verificarla. De esta manera, estos datos que están fuera de la Blockchain se podrían transmitir vía *Oráculos Descentralizados*³ o cualquier otro tipo de tecnología descentralizada para verificar la participación.

4.1.8.4 Alternativa Elegida

Emisión de tokens de gobernanza es la alternativa elegida, debido a que se dispone ya de una base gracias al servicio de OpenZeppelin y tampoco se necesita un tercer sistema para su funcionamiento.

4.1.9 Elección de la Blockchain dedicada al despliegue de aplicaciones descentralizadas

La plataforma va a ser creada bajo el protocolo Ethereum, lo que implica que el despliegue de contratos o aplicaciones complejas sea costoso.

Para ilustrar este coste, se puede hacer uso de la herramienta Truffle, que permite desplegar contratos inteligentes en un entorno simulado como Ganache, ya mencionado.

La Ilustración 13 muestra el despliegue de un contrato basado en ERC-20, donde se puede ver el coste final en Ethers a día 29 de abril del 2023. En esa fecha, el coste de un Ether era de 1710,00€, esto supondría un coste de despliegue de 49,77€.

² Un exchange es una plataforma que brinda un servicio de compraventa de criptomonedas. [97]

³ Un oráculo, en este ámbito, se refiere a un sistema que permite el traslado de información entre la Blockchain y la red Internet actual [98].

```
Compiling your contracts...
=====
> Compiling .\contracts\ERC20.sol
> Compiling .\contracts\ERC20.sol
> Compiling .\contracts\customERC20.sol
> Compiling .\contracts\customERC20.sol
> Artifacts written to C:\...\build\contracts
> Compiled successfully using:
  - solc: 0.8.4+commit.c7e474f2.Emscripten.clang

> block timestamp: 1673256127
> account: 0xCD052B05A4Ee665b3A7977eF3B3DB8b2660018E9
> balance: 497.99330252
> gas used: 1455373 (0x16350d)
> gas price: 20 gwei
> value sent: 0 ETH
> total cost: 0.02910746 ETH

> Saving artifacts
-----
> Total cost: 0.02910746 ETH

Summary
=====
> Total deployments: 1
> Final cost: 0.02910746 ETH
```

Ilustración 13: Compilación y despliegue de ERC-20

El proceso anterior también se puede realizar con la plataforma FoolMeOnce completa, como se muestra en la Ilustración 14. El coste total del despliegue son 0.17993088 ETH, lo que supondría aproximadamente 307,68 €.

```

Compiling your contracts...
=====
...
> Compiling .\contracts\ElectoralManager\DataInfo.sol
> Compiling .\contracts\ElectoralManager\ElectoralManager.sol
> Compiling .\contracts\ElectoralManager\ElectoralPromise.sol
> Compiling .\contracts\ElectoralManager\IElectoralPromise.sol
> Compiling .\contracts\Gobierno\ElectoralToken.sol
> Compiling .\contracts\Gobierno\Estandar\GovernorContract.sol
> Compiling .\contracts\Gobierno\Estandar\TimeLock.sol

> Compiled successfully using:
- solc: 0.8.17+commit.8df45f5f.Emscripten.clang

Starting migrations...
=====
> Network name: 'Ganache'
> Network id: 5777
> Block gas limit: 6721975 (0x6691b7)

1_initial_migration.js
=====

Deploying 'ElectoralToken'
-----
> total cost: 0.03652668 ETH

Deploying 'ElectoralManager'
-----
> total cost: 0.03684388 ETH

Deploying 'GovernorContract'
-----
> total cost: 0.069549 ETH

Deploying 'TimeLock'
-----
> total cost: 0.03701132 ETH

-----
> Total cost: 0.17993088 ETH

Summary
=====
> Total deployments: 4
> Final cost: 0.17993088 ETH

```

Ilustración 14: Despliegue plataforma FoolMeOnce

Debido al coste que supone desplegar contratos en Ethereum, se realizará un análisis de otras redes Blockchain cuyo coste de despliegue sea inferior. En este caso, se analizarán Polygon [77], Avalanche [78] y Binance [79], teniendo en cuenta los siguientes aspectos clave:

- Permita el despliegue de contratos inteligentes desarrollados en Solidity.
- El algoritmo de consenso sea eficiente.
- El bajo coste de la criptomoneda que la regule.

4.1.9.1 Polygon

La solución que ofrece Polygon es posicionarse por encima de Ethereum como una segunda capa, como se muestra en la Ilustración 15. De esta manera, se tendría una red Blockchain con dos capas: la capa uno es Ethereum y la capa dos es Polygon.

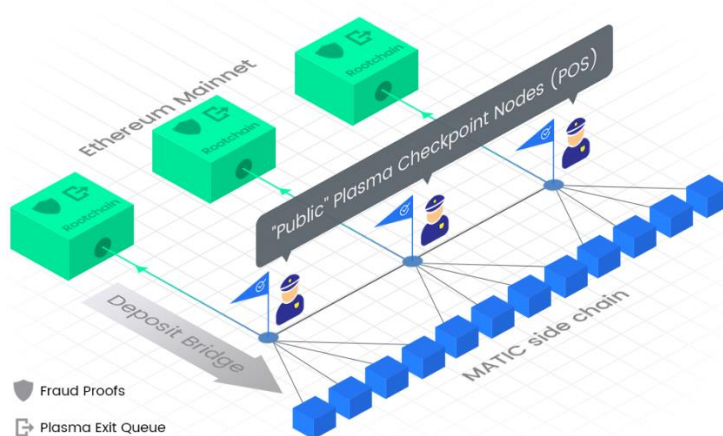


Ilustración 15: Arquitectura Polygon [80]

Los usuarios realizan el conjunto de transacciones y despliegues mediante la red de Polygon, la cual cuenta con un conjunto de validadores que utilizan el protocolo de consenso de Prueba de Participación. Además, cada cierto tiempo, el conjunto de transacciones generadas en Polygon serán llevadas a la red de Ethereum. Este traspaso se lleva a cabo reuniendo el conjunto de hashes de cada transacción y generando un único hash mediante el árbol de Merkel [81] (ver Ilustración 16).

El precio medio de su moneda durante el mes de mayo de 2023 es aproximadamente 1 dólar [82].

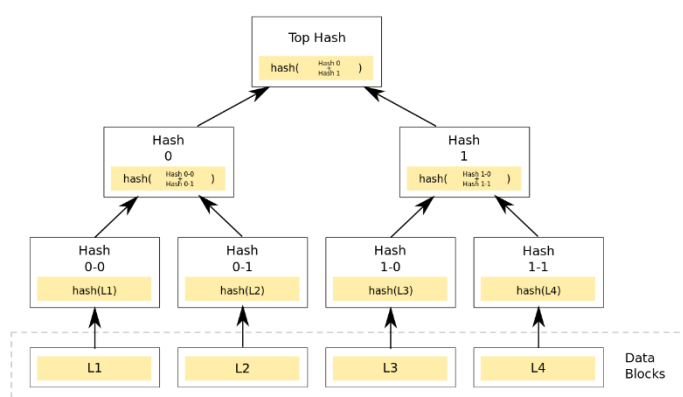


Ilustración 16: Árbol de Merkle [81]

4.1.9.2 Avalanche

Avalanche [78] es una red que se compone de tres redes Blockchain distintas [83], las cuales interoperan mediante subredes. Las tres Blockchain son:

- **X-Chain** (*Exchange Chain*): específica para activos digitales.
- **P-Chain** (*Platform Chain*): organiza a los validadores y se encarga de crear otras redes más pequeñas.
- **C-Chain** (*Contract Chain*): sus nodos siguen una implementación de la máquina virtual de Ethereum y permite la creación de contratos inteligentes.

Su protocolo de consenso es *Snowman* [83], el cual está basado en el protocolo de Prueba de Participación, pero está mejorado para ofrecer la escalabilidad y velocidad que Ethereum no ofrece.

Esta plataforma se rige por su token AVAX, cuyo precio medio durante mayo de 2023 ronda los 15 dólares [84].

4.1.9.3 Binance

Binance [79] es una empresa dedicada al intercambio de criptomonedas o *exchange*, que también posee una red Blockchain denominada BNB Chain. La criptomoneda que rige es BNB.

BNB Chain es una red más barata y rápida que Ethereum, pero no es tan descentralizada como Ethereum debido a su protocolo de consenso, un híbrido entre Prueba de Participación y Prueba de Autoridad.

Binance es criticado por su descentralización, debido a que posee alrededor de 21 validadores [85] mientras que Ethereum ya sobrepasa los 400 000.

El precio medio de su criptomoneda en mayo de 2023 ronda los 300 dólares [86].

4.1.9.4 Alternativa elegida

Polygon es la alternativa elegida, debido a que el coste de su moneda es el menor. Respecto a los otros aspectos considerados en el análisis, las tres redes Blockchain utilizan algoritmos de consenso basados en Prueba de Participación y permiten el despliegue de contratos inteligentes desarrollados en Solidity.

5 Planificación

5.1 DIAGRAMA DE GANTT

En este apartado, se muestra la planificación del TFG en la Ilustración 17. Se han realizado 14 *sprints* siguiendo la metodología Scrum. Dos de estos *sprints*, al representar hitos más cortos, han tenido una duración de una semana en vez de dos.

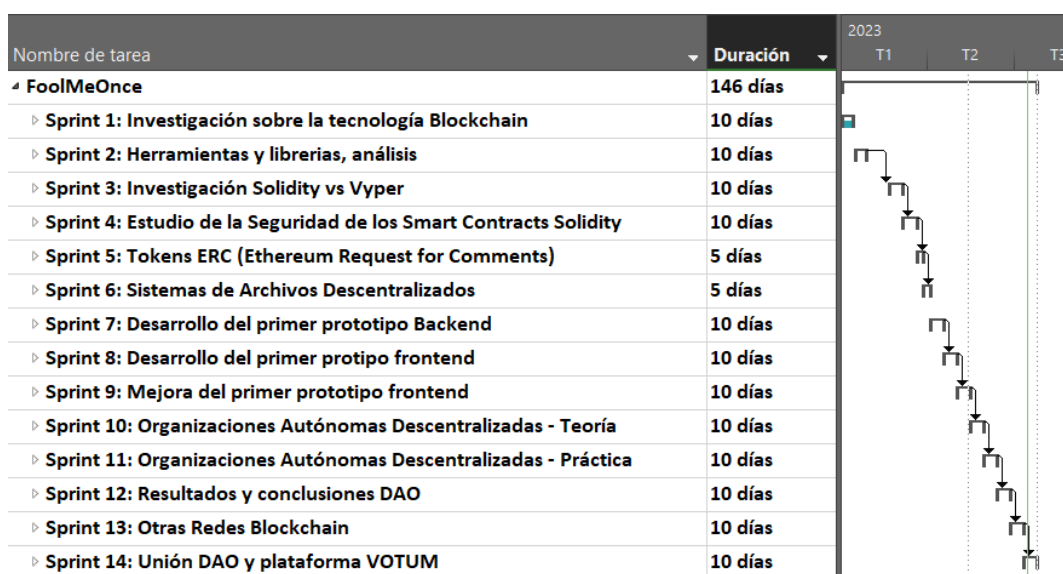


Ilustración 17: Diagrama de Gantt

5.2 DETALLE DE LAS TAREAS

En la Ilustración 18 se muestra el detalle de las tareas.



Ilustración 18: Diagrama de Gantt con tareas

6 Presupuesto

En este capítulo, se muestra la estimación del presupuesto derivado de la realización del TFG FoolMeOnce. Se ha dividido en tres partes: Hardware, Software y mano de obra. Los precios no incluyen el IVA.

6.1 CAPÍTULO 1: HARDWARE

En este apartado se describe el equipamiento hardware necesario para la realización del TFG (ver Tabla 2).

ID	DESCRIPCIÓN	UNIDAD	CANTIDAD	PRECIO UNITARIO	PRECIO
1	HP Victus Laptop 16- e0xxx	u	1	999,00€	999,00€
1.1	Procesador AMD Ryzen 5 5600H 3.30 GHz	u	1	-	
1.2	RAM 16 GB a 3200 MHz	u	1	-	
1.3.	Pantalla 16.1“FHD (1920x1080)	u	1	-	
1.4	Batería de 70 Wh con adaptador de corriente de 200W	u	1	-	
1.5	Teclado retroiluminado	u	1	-	
1.6	Tarjeta Gráfica NVIDIA RTX3050 de 4GB	u	1	-	
1.7	SSD 512 GB	u	1	-	
2	Ratón MX Master 2	u	1	50,00€	50,00€
TOTAL					1049,00€

Tabla 2: Presupuesto Hardware

El coste total asciende a MIL CUARENTA Y NUEVE EUROS.

6.2 CAPÍTULO 2: SOFTWARE

En este apartado se describe el software de pago necesario para la realización del TFG (ver Tabla 3).

ID	DESCRIPCIÓN	UNIDAD	CANTIDAD	PRECIO UNITARIO	PRECIO
1	Windows HOME 64 bit	11 u	1	110,00 €	110,00 €
2	Microsoft office 365	u/año	1	70,00 €	70,00 €
3	Microsoft Project Professional	u/año	1	199,99€	199,99€
TOTAL					379,99 €

Tabla 3: Presupuesto Software

El coste total asciende a TRESCIENTOS SETENTA Y NUEVE EUROS CON NOVENTA Y NUEVE CÉNTIMOS.

6.3 AMORTIZACIÓN DEL INMOVILIZADO MATERIAL

En este apartado se muestra la amortización del inmovilizado material (ver Tabla 4). Se establece un tiempo de amortización para los recursos de 6 años y un tiempo de uso de 7 mes. El tiempo de uso coincide con el tiempo de duración de este TFG.

MATERIAL	CUOTA ADQUISICION	TIEMPO AMORTIZACION (AÑOS)	TIEMPO DE USO (MES)	AMORTIZACIÓN
HARDWARE	1049,00 €	6	7	101,99 €
SOFTWARE	379,99 €	6	7	36,94 €
AMORTIZACIÓN TOTAL				138,93€

Tabla 4: Amortización del inmovilizado material

El total de la amortización del inmovilizado material es de CIENTO TREINTA Y OCHO EUROS CON NOVENTA Y TRES CÉNTIMOS.

6.4 CAPÍTULO 3: MANO DE OBRA

En este apartado, se muestra el presupuesto de las personas involucradas en el desarrollo y gestión de este TFG (ver Tabla 5).

ID	DESCRIPCIÓN	UNIDAD	CANTIDAD	PRECIO UNITARIO	PRECIO
1	Analista programador web3	horas	520	23,00 €	11.960,00 €
2	Desarrollador Aplicaciones descentralizadas base Solidity	horas	240	20,00 €	4.800,00 €
3	Desarrollador frontend React	horas	100	20,00 €	2.000,00 €
4	Scrum Master	horas	80	25,00€	2.000,00 €
5	Product Owner	horas	80	25,00€	2.000,00 €
TOTAL					22.760,00 €

Tabla 5: Presupuesto mano de obra

El total de la mano de obra es de VEINTIDOS MIL SETECIENTOS SESENTA EUROS.

6.5 PRESUPUESTO TOTAL

En este apartado se presenta el resumen del presupuesto total (ver Tabla 6). Se destina un porcentaje del presupuesto en referencia a gastos generales, un 13 %. También se presupuesta un porcentaje en concepto de beneficio industrial, un 6% del presupuesto de ejecución material de margen para el contratista por el proyecto realizado.

APARTADOS	PROCENTAJE	TOTAL
TOTAL INMOVILIZADO		138,93 €
MANO DE OBRA		20.760,00 €
PRESUPUESTO DE EJECUCIÓN MATERIAL		22.898,93 €
GASTOS GENERALES	13%	2.976,86 €
BENEFICIO INDUSTRIAL	6%	178,61 €
PRESUPUESTO DE EJECUCIÓN POR CONTRATA		26.054,40 €
IVA	21%	5.471,42 €
PRESUPUESTO TOTAL		31.525,83 €

Tabla 6: Presupuesto Total

El presupuesto total para el TFG es de TREINTA Y UN MIL QUINIENTOS VEINTICINCO EUROS CON OCHENTA Y TRES CENTIMOS.

7 Análisis

En este capítulo, se describirán los requisitos de usuario, tanto funcionales como no funcionales. A continuación, se describirán los requisitos, se presentará el mapa de historias y las historias de usuarios y se finalizará con el modelo de dominio y los prototipos.

7.1 REQUISITOS DE USUARIO

7.1.1 Requisitos funcionales

1. Acceso sin necesidad de registro a la plataforma
 - 1.1. Las funcionalidades de listado y visualización de promesas electorales quedarán visibles para cualquier persona solo siendo necesario una cuenta en la Blockchain.
2. Registro en la plataforma
 - 2.1. Cualquier usuario poseedor de una dirección en la red de despliegue podrá registrarse.
 - 2.2. Deberá de registrar los siguientes datos:
 - 2.2.1. Nombre completo.
 - 2.2.2. Nombre del partido político.
 - 2.2.3. Indicar si se trata de un partido político.
3. Registro de una nueva promesa electoral:
 - 3.1. Debe permitir establecer un nivel de obligatoriedad.
 - 3.2. Debe permitir establecer una relación con otras promesas ya creadas.
 - 3.3. Debe permitir establecer un conjunto de temas relacionados.
 - 3.4. Debe permitir establecer una descripción de la promesa.
4. Menú de navegación:
 - 4.1. La plataforma dispondría de un menú permita la navegación rápida entre las distintas partes de la aplicación:
 - 4.1.1. Página principal.
 - 4.1.2. Página de listado de promesas.
 - 4.1.3. Botón para la conexión de la cuenta del usuario.
 - 4.1.4. En caso de no estar registrado:
 - 4.1.4.1. Página para el registro del político.
 - 4.1.5. En caso de estar registrado:
 - 4.1.5.1. Página para la creación de promesa electorales.
5. En la página principal se indicará como debe usar la aplicación para cada tipo de usuario.

7.1.2 Requisitos no funcionales

1. Interfaz de usuario:

1.1. La interfaz debe presentar un diseño sencillo y con colores neutrales que no representen los colores de ninguna formación política:

1.1.1. El logotipo tendrá relación con el voto.

1.1.1.1. Deberá representar la acción de votar.

1.1.1.2. Será con colores neutros.

1.2. La interfaz debe adaptarse a los dispositivos móviles.

2. Seguridad

2.1. Toda operación ocurrirá gracias al cifrado con clave asimétrica que ofrece la Blockchain de Ethereum.

7.2 REQUISITOS DEL SISTEMA

7.2.1 Mapa de historia

En la Ilustración 19, Ilustración 20 e Ilustración 21 se muestra el mapa de historias.

	Investigación de tecnologías y lenguajes	Prototipo Backend	Prototipo Frontend	Investigación Organizaciones Autónomas Descentralizadas
Sprint 1	<div>Protocolos aplicados a la Blockchain</div> <div>Red Blockchain Ethereum</div>			
Sprint 2	<div>Tecnologías frontend</div> <div>Librerías desarrollo y comunicación con contratos</div> <div>Configuración de Visual Studio para despliegue de Backend y Frontend</div>			
Sprint 3	<div>Lenguajes de programación para Smart Contracts</div> <div>Diferencias entre Solidity y Vyper</div>			
Sprint 4	<div>Solidity y su seguridad</div>			
Sprint 5	<div>Estándares de Ethereum ERCs</div>			
Sprint 6	<div>Sistemas de archivos Descentralizadas</div>			

Ilustración 19: Mapa de historias 1

	Investigación de tecnologías y lenguajes	Prototipo Backend	Prototipo Frontend	Investigación Organizaciones Autónomas Descentralizadas
Sprint 7		Desarrollo del primer prototipo Backend Desarrollo función registro de usuarios Desarrollo función para obtener las promesas Desarrollo función registro de promesas		
Sprint 8		Auditar contratos del primer prototipo	Definición del primer prototipo Frontend Definición pantalla para detalles de la promesa electoral Definición pantalla listado de promesas Definición cabecera navegación	
Sprint 9			Definición de una pantalla que permita registrarse Definición pantalla para la creación de una promesa electoral	

Ilustración 20: Mapa de historias 2

	Investigación de tecnologías y lenguajes	Prototipo Backend	Prototipo Frontend	Investigación Organizaciones Autónomas Descentralizadas
Sprint 10				Investigación Teórica de las DAO
Sprint 11		Desarrollo función para aprobar promesas		Investigación Práctica de las DAO
Sprint 12				Resultados de la Investigación de la DAO
Sprint 13	Blockchains para el despliegue de dApps			
Sprint 14				Unión de ElectoralManager con la DAO

Ilustración 21: Mapa de historias 3

7.2.2 Historias de usuario

En este apartado, se detalla el conjunto de las historias de usuario desarrolladas en cada *sprint*, con sus correspondientes criterios de aceptación.

7.2.2.1 Investigación de los diferentes protocolos existentes aplicados a la tecnología Blockchain

Como desarrollador, quiero investigar los protocolos y la tecnología Blockchain existentes, como base del estudio de alternativas.

Criterios de aceptación:

- La Blockchain debe permitir el despliegue de contratos inteligentes.
- Dichos contratos inteligentes deben ser Turing-completos para permitir cualquier desarrollo.

7.2.2.2 Investigación de la Red Blockchain Ethereum

Como desarrollador, quiero investigar el protocolo Ethereum, conocer los lenguajes de programación preparados para el despliegue y su seguridad, como base del estudio de alternativas.

Criterios de aceptación:

- Generación de nuevos bloques mediante el algoritmo de consenso Prueba de Participación
- El lenguaje de programación debe ser actualizado de manera continuada, debe disponer de herramientas para pruebas, compilación y despliegue.

7.2.2.3 Investigación librerías para el desarrollo, despliegue y comunicación con los contratos inteligentes

Como desarrollador, quiero investigar las herramientas existentes que permitan la compilación de contratos, despliegue en local y depuración de estos, como base del estudio de alternativas.

Criterios de aceptación:

- Permita establecer en su configuración la versión global de Solidity.
- Permita establecer scripts para despliegue en local.
- Permita las pruebas y mostrar cobertura de lo probado.
- Permita el lenguaje TypeScript.
- Venga incluida una biblioteca para conectar con la Blockchain requerida.

7.2.2.4 Investigación de los lenguajes de programación para contratos inteligentes en Ethereum

Como desarrollador, quiero investigar los lenguajes existen para el desarrollo de Contratos inteligentes.

Criterios de aceptación:

- Debe poseer buena documentación y ejemplo de código,
- Debe poseer gran comunidad detrás de su desarrollo.
- Debe ser parecido a lenguajes como C o java.

7.2.2.5 Investigación de las tecnologías de frontend

Como desarrollador, quiero investigar los entornos de desarrollo para las interfaces de usuario, como base del estudio de alternativas.

Criterios de aceptación:

- El entorno debe ser sencillo de mantener, de desplegar, de depurar y tiene que permitir la integración rápida de herramientas web3.
- Debe permitir la generación de nuevas interfaces de manera rápida y sencilla.
- Debe permitir integrar librerías para la mejora de estilos.
- Debe permitir establecer un diseño de interfaz adaptable a los dispositivos móviles y al ordenador.

7.2.2.6 Investigación de las diferencias entre Solidity y Vyper

Como desarrollador, quiero investigar qué ventajas tienen cada uno de los lenguajes de programación de Contratos inteligentes.

Criterios de aceptación:

- Debe tener actualizaciones de manera continuada.
- Debe ser seguro.

7.2.2.7 Investigación del lenguaje Solidity y su seguridad

Como desarrollador, quiero investigar las características del lenguaje de programación Solidity y conocer las posibles vulnerabilidades.

Criterios de aceptación:

- Aprender sobre las vulnerabilidades que afectan al conjunto de contratos que se van a desplegar para lograr minimizar los fallos en su ejecución.

7.2.2.8 Configuración de Visual Studio Code para despliegue de Backend y Frontend

Como desarrollador, quiero configurar el IDE para el desarrollo del *backend* y *frontend*.

Criterios de adaptación:

- Deben permitir la integración y configuración con Git.
- Debe permitir ejecutar por la línea de comandos. Debe permitir instalar extensiones que mejoren la experiencia de desarrollo.

7.2.2.9 Investigación de los estándares de Ethereum (ERCs)

Como desarrollador, quiero investigar los estándares creados para Ethereum que permitan la generación de nuevos elementos en la red, como base del estudio de alternativas.

Criterios de aceptación:

- El estándar deberá permitir generar de manera única nuevos elementos, permitir transformarlos y modificarlos.
- El estándar no permitirá transferir los elementos creados, evitando que se especule con su posible valor monetario en la red.

7.2.2.10 Investigación de los sistemas de archivos descentralizados

Como desarrollador, quiero investigar sobre las diferentes tecnologías que existen para el almacenamiento descentralizado, como base del estudio de alternativas.

Criterios de aceptación:

- Debe disponer de un entorno que permita conectarse de manera local.
- Debe permitir hacer uso de este de manera gratuita.

7.2.2.11 Definición de la base para primer prototipo *backend*

Como desarrollador, quiero definir el conjunto de contratos que conformarán el *backend*

Criterios de aceptación:

- El proyecto partirá del estándar ERC721.

- Debe realizarse un conjunto de pruebas sobre el código desarrollado con una amplia cobertura de dicho código.

7.2.2.12 Desarrollo de una función que permita el registro de usuarios

Como desarrollador, quiero crear una función que permita el registro de políticos.

Criterios de aceptación:

- Debe guardar la información en un mapa clave-valor, con clave, la dirección de red y valor el conjunto de datos siguientes.
- Debe guardar dirección de red de la Blockchain.
- No debe guardar direcciones de red repetidas
- Debe guardar el nombre del político.
- Debe guardar el nombre del partido político.
- Debe guardar el identificador único del político.

7.2.2.13 Desarrollo de una función que permita la creación de promesas electorales

Como desarrollador, quiero crear una función que permita el registro de promesas electorales.

Criterios de aceptación:

- Debe crearse una nueva estructura para almacenar los datos de las promesas electorales.
- La función debe permitir crear nuevas promesas electorales solo si la dirección que envía la transacción ya está registrada como político.
- La función debe obtener el identificador, el nombre y el partido político mediante la estructura creada en el requisito anterior.
- La función debe poder indicársele la obligatoriedad de la promesa.
- La función debe permitir guardar una URI como ruta a un sistema de archivos descentralizados.

7.2.2.14 Desarrollo de una función que permita la obtención de todas las promesas electorales

Como usuario, quiero obtener toda la información guardada en la Blockchain.

Criterios de aceptación:

- Debe de permitir obtener toda la información de las promesas.

- Debe de permitir una consulta de los datos gratuita.

7.2.2.15 Definición del primer prototipo *frontend*

Como desarrollador, quiero definir el apartado *frontend* que permita mediante una interfaz web interactuar con la plataforma *backend*.

Criterios de aceptación:

- Permita conectarse de manera sencilla.
- Permita adaptarse a cualquier dispositivo, ordenador o móvil.

7.2.2.16 Definición de una cabecera de navegación

Como usuario, quiero que la plataforma me permita conectarme a la red de manera sencilla y navegar por las distintas partes de manera rápida.

Criterios de aceptación:

- La barra de navegación seguirá un diseño *responsive*.
- La barra de navegación mostrará enlaces a los distintos apartados de la plataforma dependiendo del usuario registrado.

7.2.2.17 Definición de una pantalla que permita al usuario ver el listado de promesas

Como usuario ciudadano, quiero que la plataforma permita visualizar todas las promesas que han creado los político o partidos.

Criterios de aceptación:

- La transacción no debe tener coste.
- Presentación en formato de tarjeta donde se muestre autor de la promesa, imagen y breve descripción de esta.

7.2.2.18 Definición de una pantalla donde se ilustren todos los detalles de la promesa electoral

Como usuario ciudadano, quiero una pantalla que permita ilustrar todos los datos de la promesa.

Criterios de aceptación:

- La visualización de la pantalla no supondrá un coste para el usuario.

- Permitirá un diseño adaptado a la pantalla del usuario.
- Permitirá mostrar los identificadores de las promesas electorales que están relacionadas con la visualizada. Estos identificadores serán enlaces directos a la visualización de la promesa.
- Permitirá mostrar todos los datos de la promesa electoral.
- Permitirá mostrar un código QR que permita compartir dicha promesa electoral.
- Permitirá ver si la promesa ha sido verificada o no.
- Permitirá indicar si la promesa es obligatoria o no.

7.2.2.19 Definición de una pantalla que permita registrarse

Como usuario político, quiero una pantalla donde se me concedan permisos para la creación de promesas electorales.

Criterios de aceptación:

- Permitirá registrar nombre y partido político al que se pertenece.
- Una vez registrado, se detectará que se trata de un político y permitirá la creación de promesas electorales bajo su nombre.

7.2.2.20 Definición de una pantalla que permita la creación de una promesa electoral

Como usuario político, quiero una pantalla que permita la creación de una promesa electoral:

Criterios de aceptación:

- Permita establecer un nivel de obligatoriedad.
- Permita establecer título, temas, imagen y descripción.
- Permita establecer nombre y partido político del usuario que la crea.
- Permita establecer una relación con otras promesas ya creadas.

7.2.2.21 Auditar conjunto de contratos del primer prototipo

Como desarrollador, quiero auditar el conjunto de contratos y exponer las posibles vulnerabilidades.

Criterios de aceptación:

- Exposición de las posibles vulnerabilidades de los contratos desarrollados.

7.2.2.22 Investigación teórica Organizaciones Autónomas Descentralizadas

Como desarrollador, quiero investigar las aplicaciones descentralizadas autónomas, conceptos funcionamiento teórico, ventajas y desventajas, como base del estudio de alternativas.

Criterios de aceptación:

- Permita una implementación práctica *on-chain* unida al conjunto de contratos ya implementados.
- Conocer características de funcionamiento, estados de las votaciones y permisos sobre la DAO.

7.2.2.23 Investigación Práctica Organizaciones Autónomas Descentralizadas

Como desarrollador, quiero investigar una implementación práctica sobre el conjunto de contratos denominados Electoral Manager.

Criterios de aceptación:

- Permita verificar las promesas electorales creadas.
- Permita establecer mediante un ejemplo la transición de estados de una propuesta hasta su verificación.

7.2.2.24 Definición de una función que permita aprobar las promesas electorales

Como desarrollador, quiero implementar una función que cambie el valor de la fecha de aprobación de las promesas.

Criterios de aceptación:

- Debe permitir pasar como parámetro el identificador de la promesa a aprobar.
- No debe permitir aprobar una promesa que ya haya sido aprobada.
- No debe permitir aprobar una promesa donde la diferencia entre la fecha actual y su fecha de creación sea mayor de cuatro años.
- Debe aprobar la promesa guardando la fecha actual en el atributo de fecha de aprobación.
- Solo el propietario del contrato debe poder ejecutar la función de aprobar.

7.2.2.25 Resultados investigación Organizaciones Autónomas Descentralizadas

Como desarrollador e investigador, quiero exponer los resultados obtenidos de la investigación teórica y práctica:

Criterios de aceptación:

- Establecer los beneficios de este tipo de aplicaciones.
- Indicar las desventajas encontradas de estas plataformas y sus alternativas.

7.2.2.26 Investigación de otras cadenas de bloques dedicadas al despliegue de aplicaciones descentralizadas

Como desarrollador, quiero investigar qué otras Blockchain permiten el despliegue de *dApps* escritas en el lenguaje de programación Solidity, como base del estudio de alternativas.

Criterios de aceptación:

- Sean de precio más barato que Ethereum.
- Posean una red de pruebas para probar los contratos de manera gratuita.

7.2.2.27 Unión de la plataforma de creación y listado de promesas con la Organización Autónoma Descentralizada

Como desarrollador, quiero desplegar en una red común los contratos de la DAO y que la plataforma *frontend* tome de ahí las promesas.

Criterios de aceptación:

- Creación de dos scripts que permitan el despliegue de la DAO.
- Creación de información de ejemplo para la visualización.
- Verificación de una de las promesas electorales por parte de la DAO.

7.3 PROTOTIPO DE INTERFAZ DE USUARIO

7.3.1 Logo de la Página Web

Se realizó un prototipo que identifique el objetivo de este TFG, es decir, ayudar al ciudadano y al político.



Ilustración 22: Logo de la página web

7.3.2 Página Principal

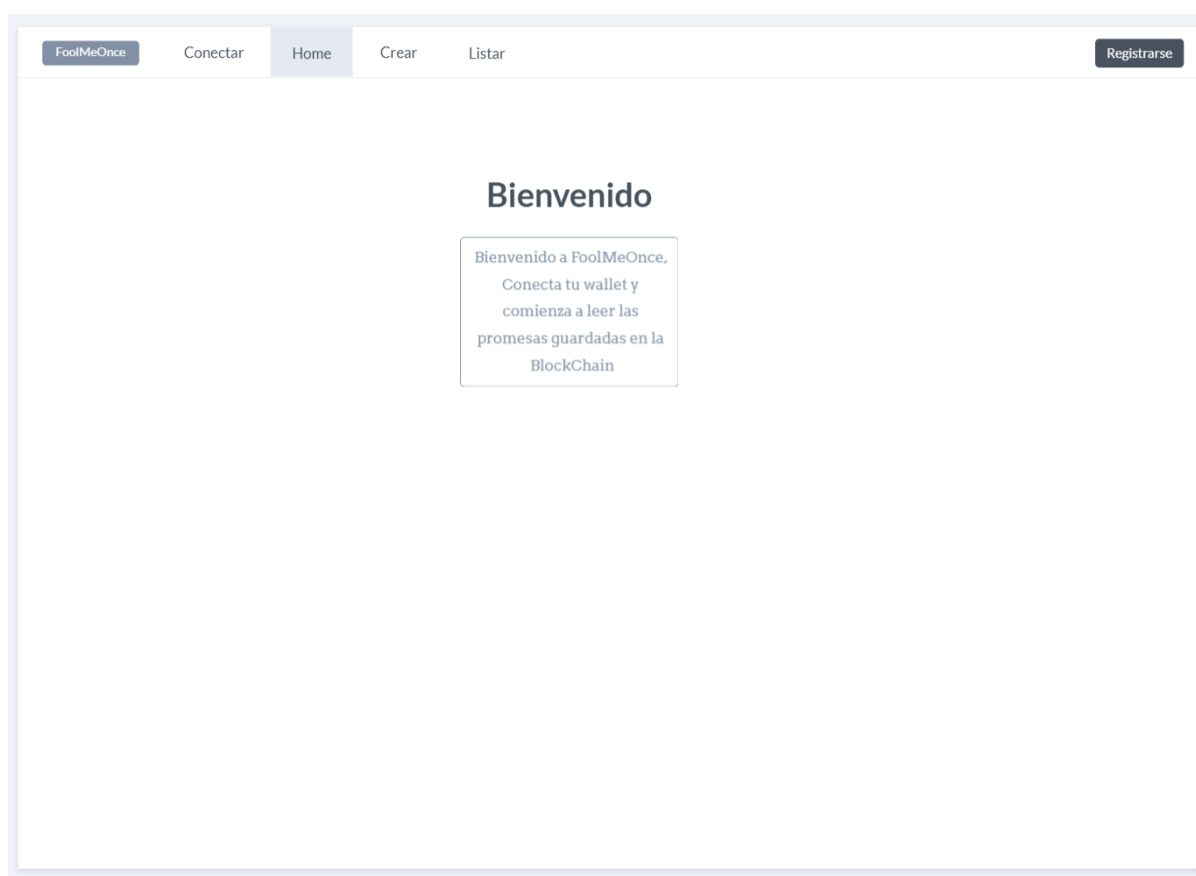
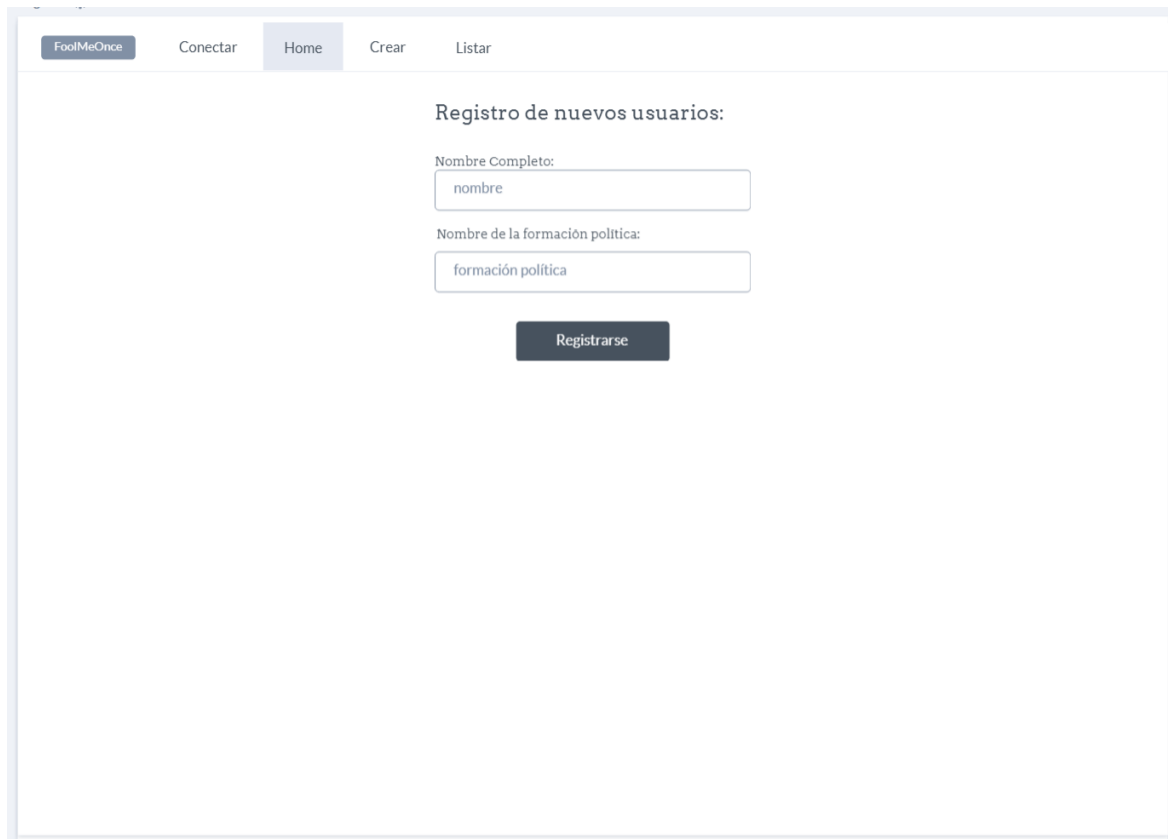


Ilustración 23: Prototipo de la página principal

7.3.3 Página de Registro de usuario



The image shows a web interface for user registration. At the top, there is a navigation bar with five items: 'FoolMeOnce' (highlighted in dark grey), 'Conectar', 'Home' (highlighted in light blue), 'Crear', and 'Listar'. Below the navigation bar, the main content area is titled 'Registro de nuevos usuarios:'. It contains two text input fields. The first field is labeled 'Nombre Completo:' and has the placeholder text 'nombre'. The second field is labeled 'Nombre de la formación política:' and has the placeholder text 'formación política'. Below these fields is a dark grey button with the text 'Registrarse' in white.

Ilustración 24: Prototipo de registro de usuarios

7.3.4 Página de Creación de una promesa

FoolMeOnce

Conectar

Home

Crear

Listar

Registrado

Título de la promesa:

título promesa

✓ Será de debido cumplimiento

Temas de la promesa:

temas promesa

¿Deseas añadir una imagen?

Selecciona Archivo: Imagen

Descripción Promesa:

Type here your message

Crear Promesa

Promesa relacionadas:

#	ID	Título	Autor	Descripción
<input type="radio"/>	1	Nueva Promesa	Miguel Rguez	# Nueva biblioteca
<input type="radio"/>	2	Mejora Parques	2012	£800
<input checked="" type="radio"/>	3	Ejemplo de	2010	£600

Ilustración 25: Prototipo de la pantalla de creación

7.3.5 Página de Listado de promesas

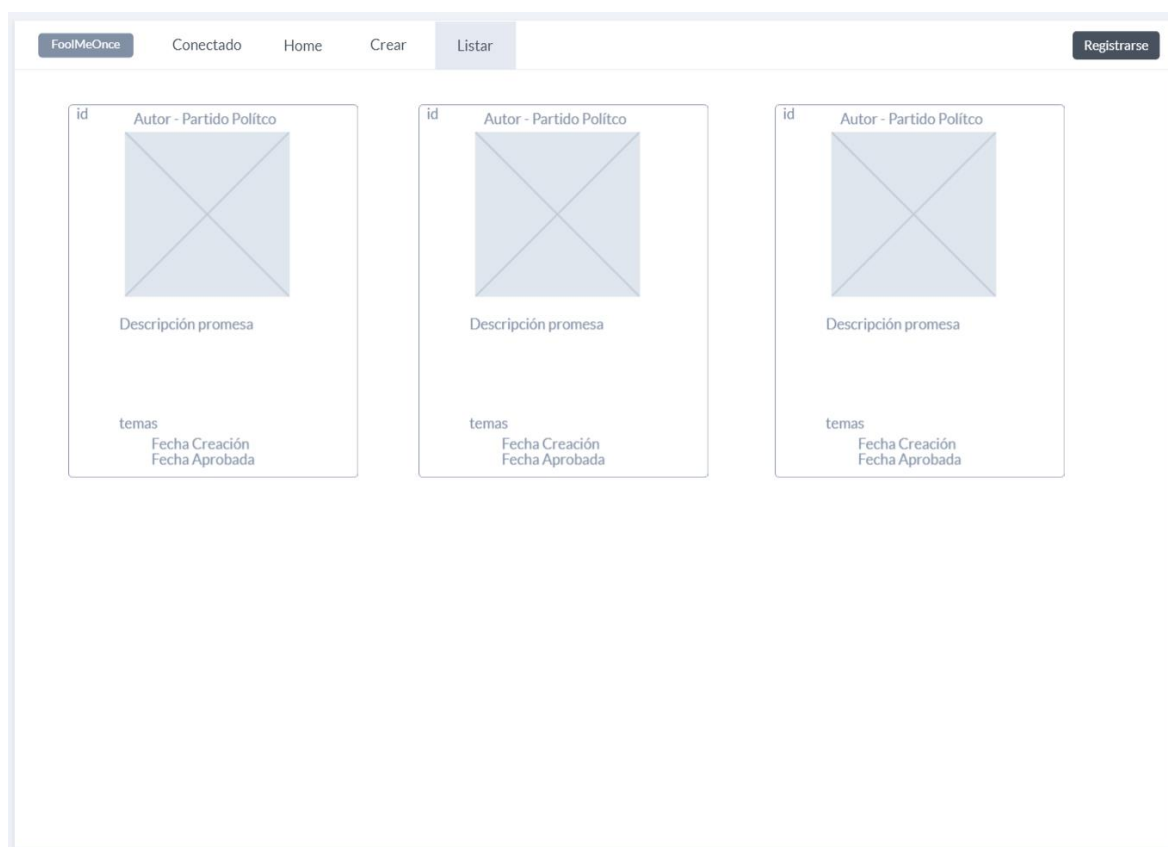
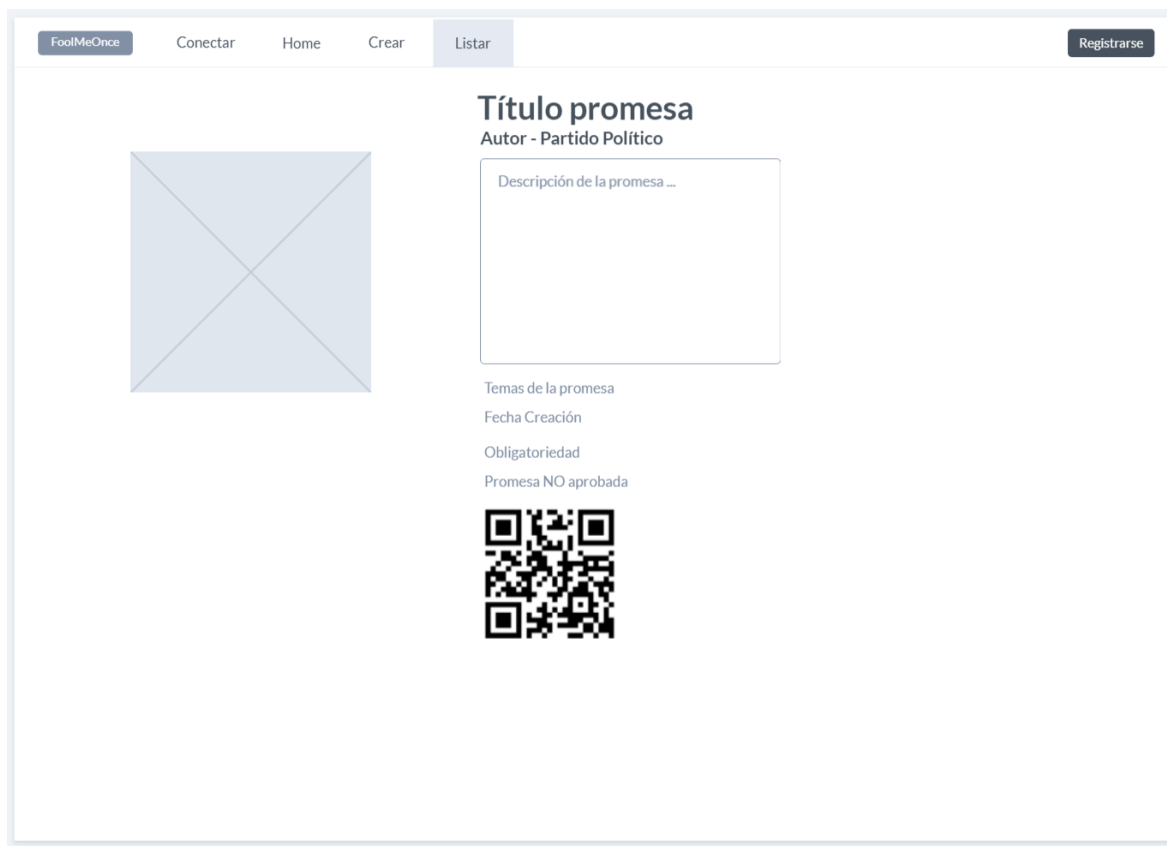


Ilustración 26: Prototipo de listado de promesas electorales

7.3.6 Página de Vista de la Promesa



Prototipo de la página de vista de una promesa electoral. La interfaz incluye una barra de navegación superior con los botones: 'FoolMeOnce', 'Conectar', 'Home', 'Crear', 'Listar' (seleccionado) y 'Registrarse'. El contenido principal se divide en dos columnas. La columna izquierda contiene un espacio reservado para una imagen, representado por un cuadrado gris con una 'X' diagonal. La columna derecha contiene el título 'Título promesa', el autor 'Autor - Partido Político', un campo de texto para la 'Descripción de la promesa ...', y una lista de metadatos: 'Temas de la promesa', 'Fecha Creación', 'Obligatoriedad' y 'Promesa NO aprobada'. En la parte inferior de esta columna se encuentra un código QR.

Ilustración 27: Prototipo vista de promesa electoral

7.4 MAPA DE NAVEGACIÓN

En este apartado, se ilustra el mapa de navegación de la aplicación web en la Ilustración 28. Las flechas indican los enlaces directos a pantallas.

Se debe tener en cuenta:

- Se debe diferenciar entre dos tipos de usuario: ciudadano y político. Esta diferencia habilitará las distintas pestañas permitidas al usuario en el menú.
- No se ha incluido la navegación a la página principal ya que es accesible desde el menú siempre.
- Desde la página de Vista de promesa se puede ir a otra vista, ya que puede haber promesas relacionadas.

El estado inicial de la aplicación será la pantalla de inicio donde se da la bienvenida a la plataforma. Para comenzar, el usuario debe conectar su cuenta de la red Blockchain a la plataforma.

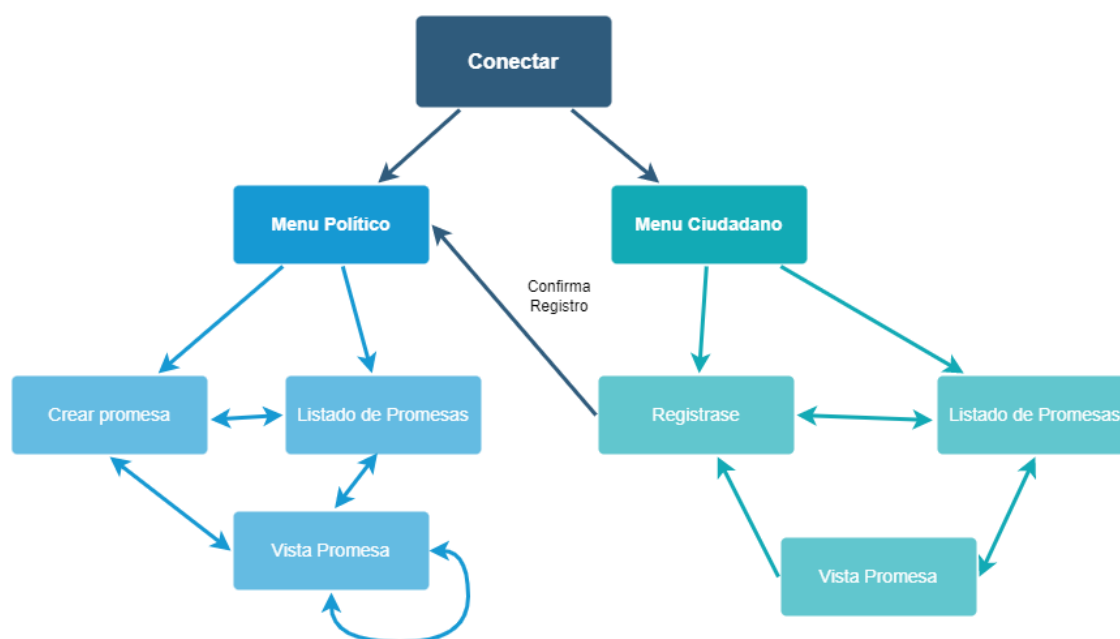


Ilustración 28: Mapa de navegación

8 Diseño

En este capítulo, se describirá el conjunto de tecnologías que conforman el diseño de la plataforma a implementar.

8.1 ARQUITECTURA

En este apartado, se describe la arquitectura planteada, que consta de diversas partes, como se muestra en Ilustración 29.

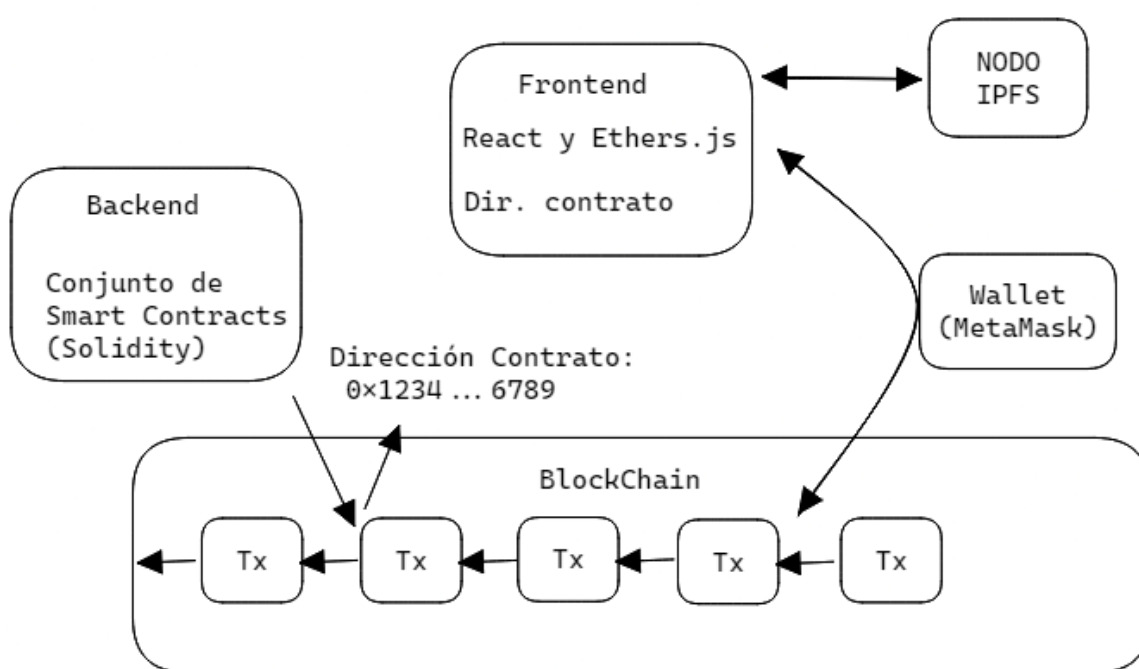


Ilustración 29: Arquitectura

El *backend* de esta aplicación es el conjunto de contratos inteligentes desplegados en la red Blockchain, obteniendo en dicha acción la dirección de su despliegue.

En una primera parte, se tendrá el contrato creado para la gestión y creación de promesas electorales que se denominará **ElectoralManager**. Este programa, basado en el estándar ERC-721 (NFTs), contará con las funciones requeridas de registro de políticos, creación de promesas, obtención de promesas y aprobación de promesas.

ElectoralManager será un conjunto de cuatro contratos:

- **IElectoralPromise:** será la interfaz base de los contratos y tendrá las funciones del ERC-721 sin las funciones de transferencia.
- **ElectoralPromise:** será el contrato que implemente todas las funciones de IElectoralPromise.
- **DataInfo:** será un contrato librería que almacenará las estructuras de datos para el registro de usuarios y promesas electorales.
- **ElectoralManager:** será el contrato que herede las funciones de ElectoralPromise y utilice la librería DataInfo.

En una segunda parte, se creará la plataforma de gobierno, que se explicará en el apartado 10.

Para la parte *frontend*, se maquetará una interfaz de usuario mediante la librería React con las funciones requeridas para la gestión de promesas electorales. Este *frontend*, denominado **VOTUM**, hará uso de tres elementos:

- **IPFS:** conexión con un nodo local para el almacenamiento de los datos de las promesas.
- **Ethers:** para la conexión del usuario con las funciones del contrato inteligente. Las funciones de consulta de datos no necesitan aprobación por parte del usuario.
- **Cartera** (en Metamask): para que el usuario apruebe las transacciones de pago, como el registro del político y la creación de promesas electorales.
- **Contratos inteligentes:** el *frontend* almacenará la dirección del contrato desplegado y el ABI, de esta manera el *frontend*, haciendo uso de la librería Ethers.js, podrá conocer las funciones a las que tiene que llamar.

8.2 DISEÑO FÍSICO DE LOS DATOS

En este apartado, se muestra cómo se almacenan los datos en la Blockchain mediante el uso de estructuras definidas en los contratos inteligentes.

8.2.1 Estructura de los datos

En Solidity, al igual que en el lenguaje de programación C, se permite crear “*structs*”, los cuales permiten crear nuevos tipos de datos para el sistema. Serán necesarias dos estructuras relacionadas denominadas Usuarios y Promesas, que se muestran en la Ilustración 30, cuyo significado es el siguiente:

- **Usuarios:** es un mapa clave-valor para guardar datos de los usuarios. La clave es la dirección del usuario en la red y el valor, el identificador numérico, de esta manera se evita la repetición de cuentas.
- **Promesas:** es una lista de datos para las promesas, donde se añadirán cada una de las promesas electorales con sus respectivos datos.

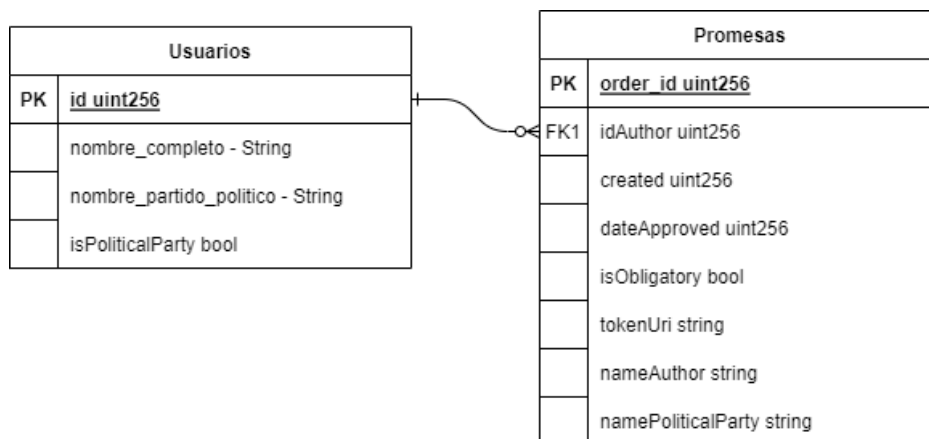


Ilustración 30: Estructuras de datos del proyecto

8.3 DIAGRAMA DE PAQUETES

La Ilustración 31 muestra los paquetes en los que se divide este TFG. En los siguientes apartados se entrará en el detalle de cada uno de ellos.

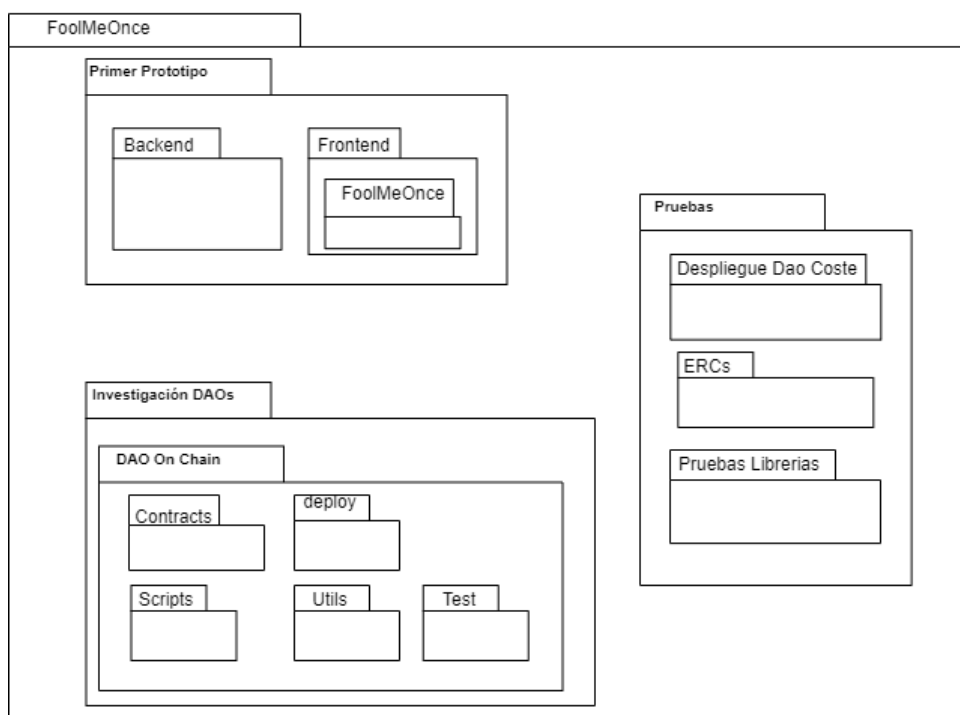


Ilustración 31: Diagrama de paquetes global del TFG

8.3.1 Paquete Primer Prototipo

Este paquete contiene la plataforma sin la verificación de promesas electorales (ver Ilustración 32).

La plataforma cuenta con dos partes:

- **Backend:** almacena el conjunto de contratos inteligentes, programas de prueba, una carpeta para recursos y un conjunto de programas que permitan el despliegue de los contratos inteligentes.
 - **Contracts:** almacena el conjunto de contratos inteligentes.
 - **Test:** almacena un conjunto de programas que contienen las pruebas automatizadas.
 - **Assets:** almacena ficheros de recursos que son necesarios para el *backend*.
 - **Scripts:** almacena un conjunto de programas que sirven para el despliegue de los contratos.
- **Frontend:** tiene una estructura de ficheros fija:
 - **Public:** almacena los ficheros de carácter público, como son el icono de la página web y el archivo principal de html.
 - **Src:** almacena los archivos fuentes divididos en cuatro partes:
 - **Assets:** almacena ficheros de recursos que son necesarios para el *frontend*.

- **Components:** almacena ficheros que representan componentes, los cuales representan una pequeña funcionalidad dentro de la interfaz de usuario.
- **ContractsData:** almacena el ABI y la dirección del contrato inteligente ElectoralManager.
- **Pages:** almacena ficheros que representan la estructura de una pantalla. Estas pantallas se forman uniendo componentes.

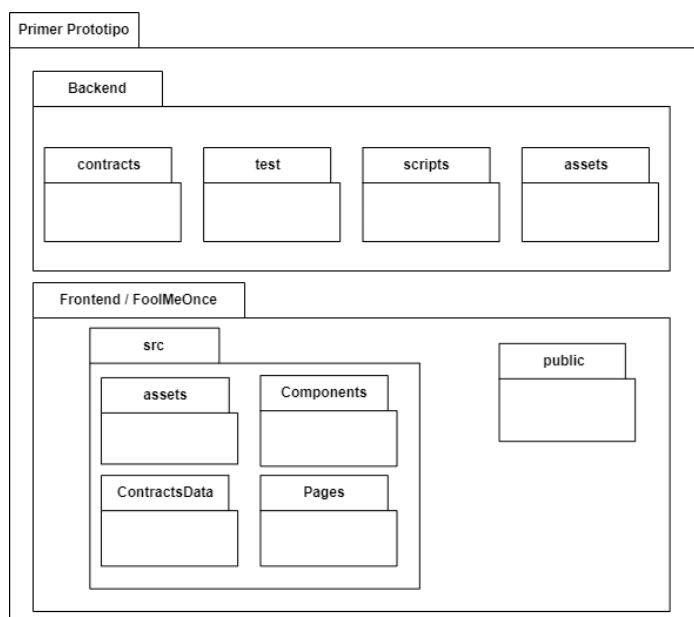


Ilustración 32: Diagrama de paquetes del primer prototipo

8.3.2 Paquete Investigación DAOs

Para la investigación de las DAO se crea el paquete mostrado en la Ilustración 33. El paquete de investigación DAOs se divide en:

- **Contracts:** almacena el conjunto de contratos inteligentes. Se dividen en dos partes:
 - **ElectoralManager:** almacena el conjunto de contratos para la creación y listado de promesas.
 - **Gobierno:** almacena el conjunto de contratos para la gobernanza en cadena.
- **Deploy:** almacena el conjunto de programas para desplegar los contratos inteligentes.
 - Ganache: contiene los programas para el despliegue en la red simulada Ganache.
- **Test:** almacena el conjunto de pruebas automatizadas
- **Scripts:** almacena el conjunto de programas necesarios para aprobar una promesa en la DAO,

- **Utils:** almacena el conjunto de programas con funciones y valores constantes necesarios para la carpeta Scripts y Deploy.

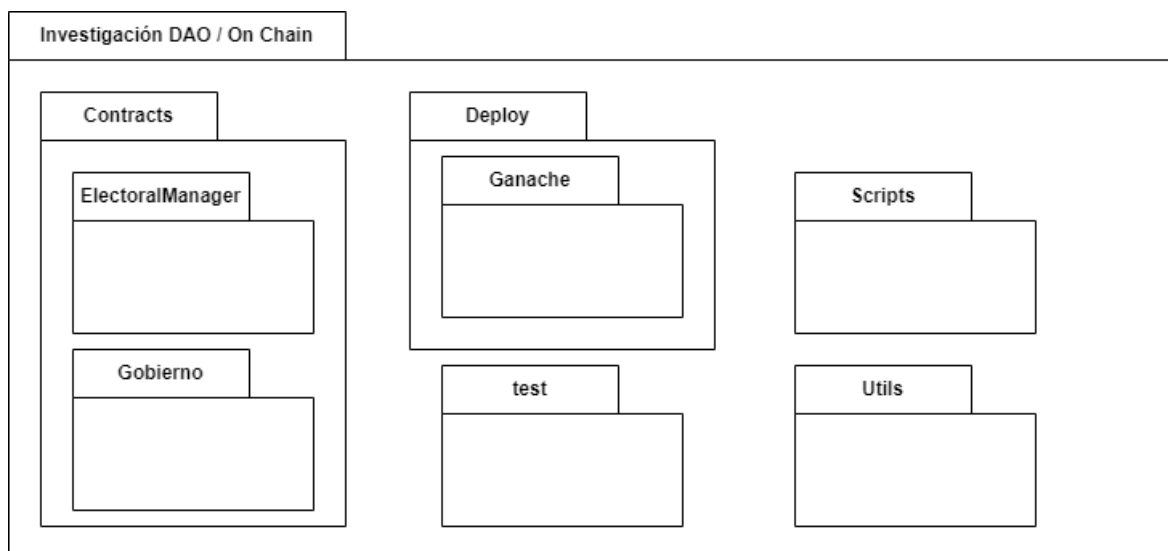


Ilustración 33: Diagrama paquetes Investigación DAO

8.3.3 Paquete de Pruebas

En este paquete, mostrado en la Ilustración 34, se guardan todas las investigaciones que no forman parte del desarrollo global, sino que sirven como estudio o como ejemplo de prueba. Las diferentes partes son:

- **ERCs:** pequeña implementación de los estándares: ERC-165 y ERC-721.
 - **ERC-165:** interfaz, implementación del interfaz y un contrato inteligente que ejemplifique lo dispuesto en la interfaz.
 - **ERC-721:** interfaz, implementación del interfaz y un contrato inteligente que ejemplifique lo dispuesto en la interfaz.
- **Pruebas Librerías:** en Solidity existen un tipo de contratos inteligentes que, marcados con la palabra clave *library*, permiten reutilizar dicho código en otros contratos sin necesidad de replicar código. Para probarlo se realiza un duplicado del *backend* del Primer Prototipo.
- **Despliegue DAO Coste:** se utiliza para ilustrar el problema del coste de un despliegue en la red Ethereum. Se divide en las siguientes partes:
 - **Contracts:** todo el conjunto de contratos inteligentes, tanto los que gestionan las promesas electorales como los de gobierno.
 - **Migrations:** alberga el *script* de despliegue de todos los contratos.

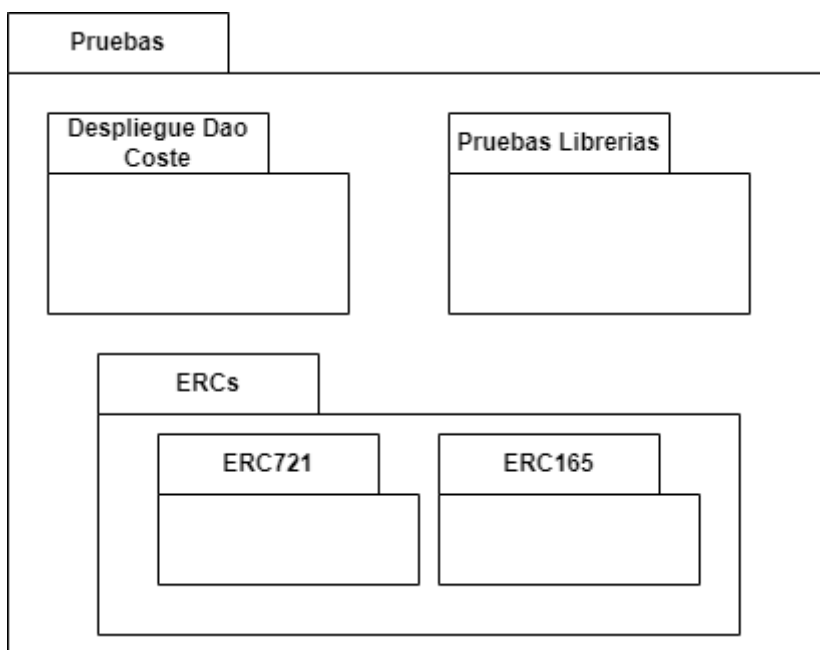


Ilustración 34: Diagrama Paquetes Pruebas

8.4 DIAGRAMA DE INTERACCIÓN

En este apartado, se mostrarán los procesos de conexión a la plataforma, obtención de datos y registro de datos.

8.4.1 Diagrama de conexión con la plataforma VOTUM

En la Ilustración 35 se muestra como el usuario se conecta a la plataforma de creación de promesas electorales o VOTUM.

Para la conexión el usuario dispondrá de una cuenta en la herramienta MetaMask. Esta acción es necesaria para interactuar con la Blockchain. Lo único que necesitará conocer el usuario será su contraseña para su cartera y, finalmente, indicar que desea conectarse a la plataforma.

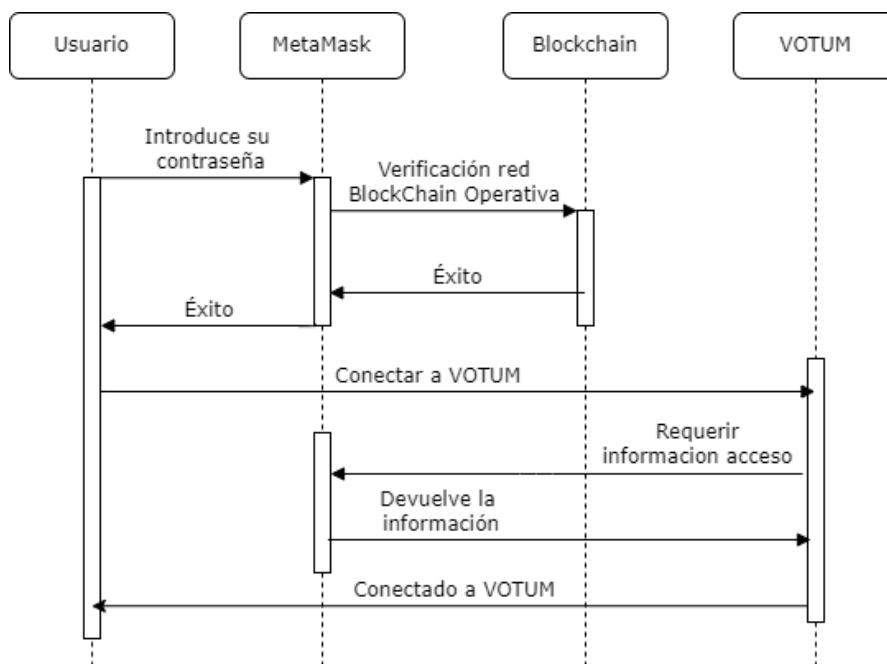


Ilustración 35: Diagrama de interacción para la conexión con plataforma

8.4.2 Diagrama para la consulta de promesas electorales

En la Ilustración 36 se muestra como el ciudadano, una vez conectado a la plataforma, consulta el listado de promesas electorales o su visualización.

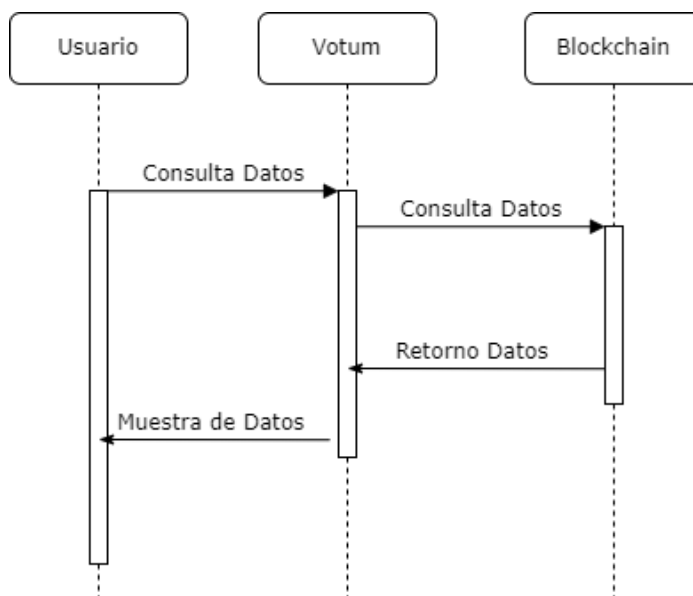


Ilustración 36: Diagrama de interacción para la consulta de datos

8.4.3 Diagrama para el registro de promesas electorales

En la Ilustración 37, se muestra como el político, una vez conectado a la plataforma, registra una nueva promesa electoral.

Esta acción debe ser aprobada por el político, debido a que se trata de un cambio en el estado de la Blockchain, y por tanto, tiene un coste.

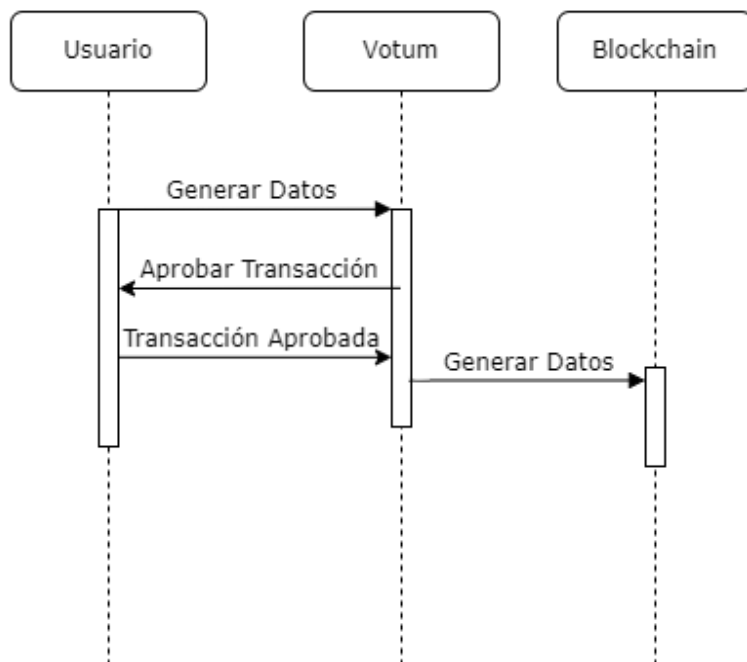


Ilustración 37: Diagrama de interacción para el registro de promesas electorales

9 Pruebas

9.1 INTRODUCCIÓN

Este capítulo, tiene como objetivo plantear las situaciones de pruebas para el contrato inteligente Electoral Manager: creación y verificación de promesas electorales, derivando de dichas situaciones, sus respectivos casos de prueba.

9.2 ESPECIFICACIÓN DE LA APLICACIÓN

Esta aplicación se divide en tres partes:

- **Registro de usuarios:** cada usuario que desee crear una promesa deberá registrar su nombre y partido político, de esta manera, con cada creación de promesa, su nombre y partido aparecerán asociados a la promesa.
- **Creación de promesas:** una vez registrado el usuario, podrá crear cualquier promesa electoral. Se necesitará la URI con los datos de la promesa, si es obligado cumplimiento y quién es la persona que registra dicha promesa.
- **Aprobación de una promesa:** esta acción, debido a que se produce con una llamada desde una DAO, debe ser pública. Se puede aprobar cualquier promesa que exista dentro del periodo de gobierno, el actual son cuatro años.

9.3 DISEÑO DE SITUACIONES DE PRUEBA

9.3.1 Clases de equivalencia

1. Políticos

1. (1.1) Dirección Político.
 - a. Texto
 - b. Vacía (inválida)
2. (1.2) Nombre Completo.
 - a. Texto
 - b. Vacía (inválida)
3. (1.3) Nombre Partido político.
 - a. Texto

- b. Vacía (inválida)
- 4. (1.4) Es un Partido político (booleano).
 - a. Texto
 - b. Vacía (inválida)
- 5. (1.5) Nuevo Político (numérico)
 - a. Número mayor que cero
 - b. Otro (inválido)
- 6. (1.6) Político existente
 - a. Número mayor que cero
 - b. Otro (inválido)
- 2. Promesa
 - 1. (2.1) URI.
 - a. Texto
 - 2. (2.2) Obligatoriedad.
 - a. Booleano
 - b. Vacío (inválido)
 - 3. (2.3) Dirección Usuario.
 - a. Valida
 - b. Vacío (inválido)
- 3. Verificación Promesa
 - 1. (3.1) identificador Promesa.
 - a. Valor numérico
 - b. Otro (inválido)
- 4. Aprobar una promesa
 - 1. (4.1) Identificador promesa (numérico)
 - a. Identificador existe, entra dentro del tiempo y no ha sido aprobado.
 - b. Identificador existe, no entra dentro del tiempo y no ha sido aprobado.
 - c. Otro (inválido)

9.3.2 Técnica Basada en caminos

Se aplica esta técnica debido a la transición de estados existente en el proceso de creación de la promesa, es decir, existe una transición dentro de la aplicación desde el registro de usuario hasta el estado de aprobada. En la Ilustración 38 se muestran las transiciones existentes.

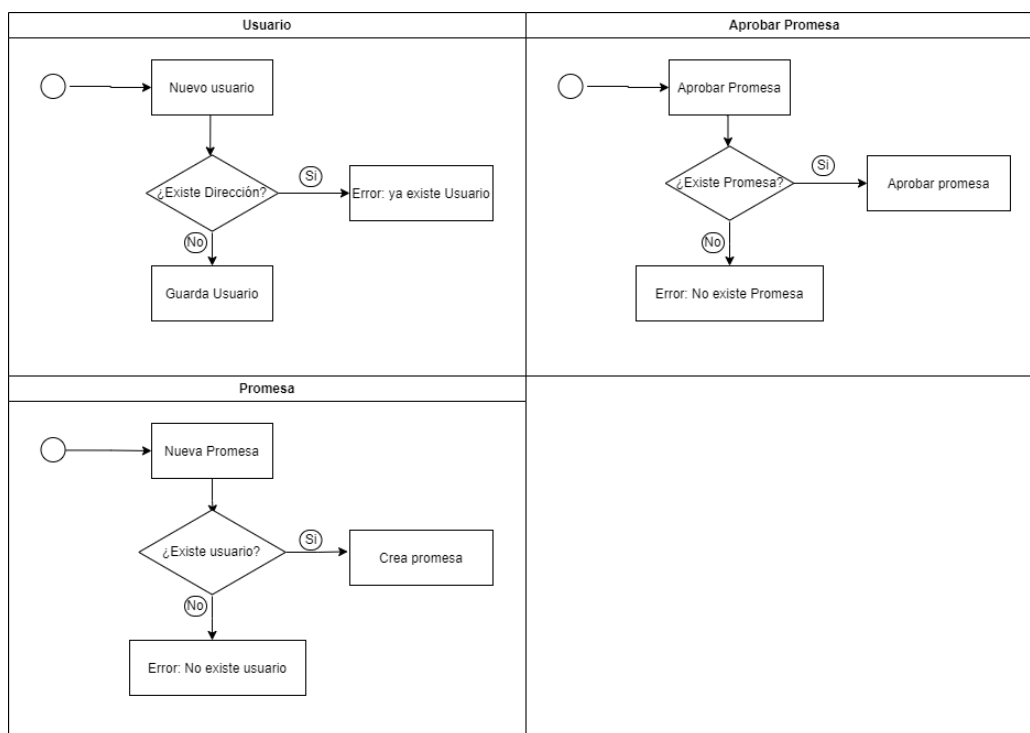


Ilustración 38: Diagrama de caminos 1

Los caminos que se obtienen se muestran en la Ilustración 39.

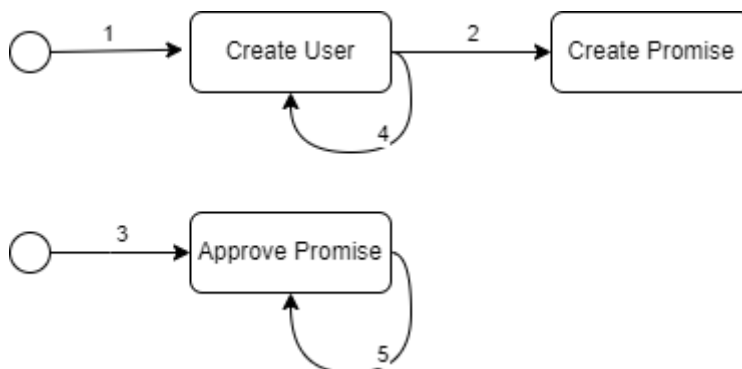


Ilustración 39: Diagrama de caminos 2

9.3.2.1 Técnica de caminos simples

Aplicando la técnica de caminos simples sobre los caminos de la Ilustración 39, se obtienen las situaciones de prueba de la Tabla 7.

ID	SITUACIÓN DE PRUEBA
1	1-2-3
2	1-4 (C.I)
3	1-2-3-4-5(C.I)

Tabla 7: Caminos simples

9.3.2.2 Técnica de pares de caminos

Aplicando la técnica de pares de caminos, se obtienen las situaciones de prueba de la Tabla 8.

ID	SITUACIÓN DE PRUEBA
1-2	1-2-3
2-3	1-4-2 (C.I)
4-2	1-2-3-5 (C.I)

Tabla 8: Pares de caminos

9.4 DATOS PARA LAS PRUEBAS (BASE DE DATOS)

A continuación, se presentan en la Tabla 9 el conjunto de datos que se manejarán en las pruebas automatizadas.

ID	PARÁMETRO	VALOR
D1	Dirección 1	Addr1
D2	Dirección 2	Addr2
D3	Segundos Legislatura	16144000
D4	Nombre Completo	Juan Alberto Rodríguez
D5	Nombre Partido Político	Partido por la Blockchain
D6	Se trata de un partido político	true
D7	Promesa Obligatoria	true
D8	Promesa No Obligatoria	False
D9	URI	URI
D10	Dirección del Propietario / Deployer	Deployer

Tabla 9: Datos para las pruebas

9.5 CASOS DE PRUEBA

En la Tabla 10 se muestran los casos de prueba. Para cada uno de ellos se indica el objetivo de la prueba, las situaciones de prueba que se cubren, los datos a utilizar en la prueba, además de las salidas esperada.

CP	OBJETIVO	SITUACIONES DE PRUEBA	DATOS	SALIDA ESPERADA
1	Crear un nuevo usuario, nueva promesa del usuario y aprobarla	PC1, 1.1.a, 1.2.a, 1.3.a, 1.4.a, 1.5.a, 1.6.a, 2.1.a, 2.2.a, 2.3.a	D1, D4, D5, D6, D9, D10	Nuevo usuario, promesa creada y aprobada
2	Crear un nuevo usuario y Reintento de crearlo	PC2	D1, D4, D5, D6, D9	Usuario 1 creado y Fallo Ya existe
3	Reintento de aprobar una promesa	PC3	D1, D4, D5, D6, D9	No se puede aprobar una promesa ya aprobada
4	Aprobar una promesa fuera del límite de tiempo	4.1.b	D1, D4, D5, D6, D9	No se puede aprobar una promesa, el tiempo ha pasado
5	Aprobar una promesa que no exista	4.1.c	D1, D3, D4, D5, D6, D9	No se puede aprobar una promesa que no existe

Tabla 10: Casos de prueba

9.6 RESUMEN PRUEBAS

Tras solucionar los fallos encontrados, todas las pruebas han pasado, obteniendo las mismas salidas que las esperadas. Los casos de prueba ejecutados han permitido obtener la cobertura de sentencias, ramas, funciones y líneas que muestra en la Ilustración 40.

```
PS C:\Users\miguel\OneDrive\Escritorio\epi-foolmeonce\Investigacion_DAOs\DAO_On_Chain> yarn hardhat coverage

Version
=====
> solidity-coverage: v0.8.2

Instrumenting for coverage...
=====

> ElectoralManager\DataInfo.sol
> ElectoralManager\ElectoralManager.sol
> ElectoralManager\ElectoralPromise.sol
> ElectoralManager\IElectoralPromise.sol
> Gobierno\ElectoralToken.sol
> Gobierno\Estandar\GovernorContract.sol
> Gobierno\Estandar\TimeLock.sol
  ✓ should track the incorrect attempt to approve an electoral promise not created
  ✓ should track the incorrect attempt to approve an electoral promise before legislature ends
  ✓ Should emit an error when the caller is not the owner for function approve
Register and try again to register the same address
  ✓ should track the register of a person and return an error when register a person with the same address (52ms)

7 passing (2s)

-----|-----|-----|-----|-----|-----|
File      | % Stmts | % Branch | % Funcs | % Lines | Uncovered Lines |
-----|-----|-----|-----|-----|-----|
ElectoralManager\
DataInfo.sol      |    95.45 |    67.86 |    100 |    96.43 |                  |
ElectoralManager.sol |    100 |    91.67 |    100 |    100 |                  |
ElectoralPromise.sol |    90.48 |     50 |    100 |    92.59 |      106,113 |
IElectoralPromise.sol |    100 |    100 |    100 |    100 |                  |
Gobierno\
ElectoralToken.sol |     0 |    100 |     0 |     0 |      10,18,22,29 |
Gobierno\Estandar\
GovernorContract.sol |     0 |    100 |     0 |     0 | ... 109,118,124 |
TimeLock.sol      |    100 |    100 |     0 |    100 |                  |
-----|-----|-----|-----|-----|-----|
All files          |    72.41 |    67.86 |    51.52 |    77.14 |                  |
-----|-----|-----|-----|-----|-----|

> Istanbul reports written to ./coverage/ and ./coverage.json
PS C:\Users\miguel\OneDrive\Escritorio\epi-foolmeonce\Investigacion_DAOs\DAO_On_Chain> 
```

Ilustración 40: Cobertura de pruebas automatizadas (obtenida con la herramienta Hardhat)

10 Implementación de una DAO

En este capítulo, se describe la implementación funcional realizada de una DAO, a modo de prueba de concepto. Para ello, se ha seguido el módulo de gobernanza ofrecido por OpenZeppelin [87].

10.1 IMPLEMENTACIÓN DE LA DAO

Para construir una DAO *on-chain* que se gestione mediante un token de gobernanza se necesita:

- **Contrato de gobernanza:** el cual posea las funciones de propuesta, votación y ejecución.
- **Contrato que brinde el poder de voto.**
- **Contrato para el control del tiempo y manejo de privilegios:** se necesita establecer periodos de tiempos para las fases de propuesta y votación.
- **Contrato sobre el que se ejecutará la gobernanza:** en este caso será ElectoralManager, desarrollado en la primera parte del TFG.

10.1.1 Partes de una DAO con OpenZeppelin

Una DAO, de manera genérica posee cuatro eventos en su flujo, como se mostró en la Ilustración 4 del apartado 3.4.2, pero en el conjunto de contratos de OpenZeppelin se realiza una fase más denominada fase *cola*, como se muestra en la Ilustración 41.

La fase cola se encarga de establecer un punto intermedio entre la votación y ejecución de la propuesta.

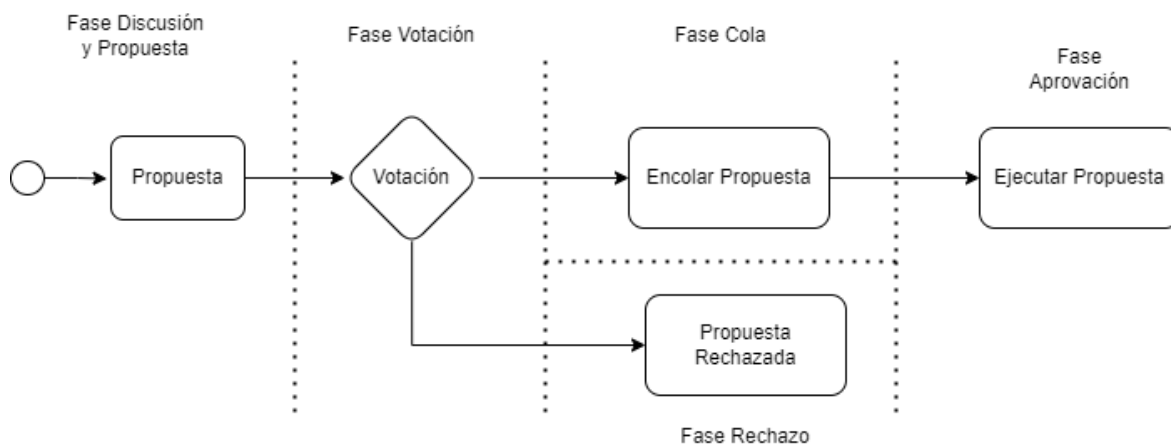


Ilustración 41: Diagrama base de sucesos en la DAO a implementar

10.1.1.1 Contrato de Gobernanza

Para crear el contrato de Gobernanza, **GovernorContract**, se utiliza el contrato *Governor* [88]. Este contrato contiene las operaciones básicas de una DAO:

- Propuesta, Votación, Cola y Ejecución
- Configuración de tiempos para la votación.

Para operar entre las distintas fases, se marcan tiempos para dar margen a la comunidad a que proceda con su voto, como ya se ha mencionado. Este manejo de tiempos se cuantifica por medio del número de bloques de media que genera la red Blockchain.

10.1.1.2 Contrato que brinde el poder de voto

Este poder de voto se puede representar mediante los estándares ERC-20 o ERC-721. Para este caso, se ha implementado **ElectoralToken**, el cual utiliza una adaptación del estándar ERC-20, en concreto el ERC20Votes [89].

Esta adaptación brinda la posibilidad de establecer puntos de control para contabilizar el poder de voto de cada cuenta. Gracias a este contrato, la DAO tiene su Token De Gobernanza.

10.1.1.3 Contrato para el control del tiempo y manejo de privilegios

Para esta parte, se ha implementado **TimeLock**, basado en el contrato *TimeLockController* [90].

Además, se realizarán las siguientes acciones:

- Se indicará al contrato que cualquiera puede realizar las tareas de propuesta y ejecución de propuestas.
- Se posicionará **TimeLock** como dueño del contrato sobre el que se va a ejecutar la gobernanza.
- Se marcarán aquellas funciones del contrato gobernado con el modificador *onlyOwner*⁴.

Al establecer dicho modificador, se establece a la DAO como el único con permisos de ejecución, logrando que solo mediante una propuesta y votación satisfactorias se ejecute finalmente la función.

⁴ En Solidity los modificadores permiten cambiar el comportamiento de las funciones. En este caso, *onlyOwner* solo hace accesible la ejecución de la función al propietario del contrato.

10.1.1.4 Contrato sobre el que se ejecutará la gobernanza

Se necesita el contrato sobre el que la gobernanza tendrá su peso, en este caso será el desarrollado en este TFG para la creación de promesas (ElectoralManager). Este contrato, una vez desplegado, tendrá como administrador el contrato **TimeLock**, como se mencionó anteriormente.

Aunque el proceso de verificación de promesas se marque con el modificador *onlyOwner*, esto no afectará al resto de procesos del contrato.

10.1.2 Verificación de una promesa electoral

En este apartado, se presentará la función encargada de aprobar una promesa y el despliegue del conjunto de contratos involucrados. También se realizará el proceso completo de las fases de una DAO, se partirá de una preparación de datos, para posteriormente generar una propuesta, una votación y finalmente la ejecutar la aprobación de una promesa.

10.1.2.1 Función a Gobernar: ApprovePromise

La función que aprueba las promesas consta de tres verificaciones:

- La promesa existe.
- La promesa no ha sido aprobada con anterioridad.
- Los cuatro años de la legislatura desde que se realizó la promesa no han pasado.

De esta manera, se valida que la promesa existe, que no ha sido aprobada anteriormente y que no han pasado los 4 años que dura una legislatura. La función desarrollada se muestra en Ilustración 42.

Es importante destacar los siguientes parámetros dentro de la función:

- Parámetro *onlyOwner*: para que solo el propietario del contrato pueda ejecutar la función.
- Parámetro *Storage* de *DataInfo.DataPromise* de nombre *promiseToApprove*: gracias a este parámetro se logra que cualquier cambio sobre la variable produzca un cambio permanente en la Blockchain.

```
/**
 * @dev approve an electoral promise updating the parameter dateApproved
 * @param promiseId uint that identifies the electoral promise
 */
function approvePromise(uint256 promiseId) external onlyOwner {
    require(
        promiseId < counterElectoralPromises,
        "Error: electoral promise do not exists"
    );
    DataInfo.DataPromise storage promiseToApprove = listElectoralPromises[
        promiseId
    ];
    require(
        promiseToApprove.dateApproved == 0,
        "Error: electoral promise already approved"
    );
    //check if 4 years of legislature have passed
    uint256 currentSeconds = block.timestamp;
    uint256 difference = currentSeconds - promiseToApprove.created;
    require(
        difference < SECONDS_LEGISLATURE,
        "Error: a legislature already gone"
    );

    promiseToApprove.dateApproved = currentSeconds;
    emit ApprovedPromise(promiseId);
}
```

Ilustración 42: Función approvePromise - gobierno

10.1.2.2 Despliegue DAO

Para desplegar la DAO, es necesario una serie de pasos para la puesta a punto:

1. Se desplegará el contrato ElectoralToken.
2. Se desplegará el contrato Timelock.
3. Se desplegará el contrato de gobernanza (GovernorContract).
 - En el constructor del contrato se le pasarán los tiempos y las direcciones de los contratos TimeLock y ElectoralToken.
4. (Configuración) Se establecerá que dirección tendrá permisos de ejecución, propuesta y administración sobre el contrato TimeLock.
 - Se da permiso a GovernorContract como generador de propuestas.
 - Se revoca el permiso de administrador a la cuenta que despliega el contrato TimeLock.
5. Se despliega ElectorManager y se transfiere su propietario al contrato TimeLock, de esta manera la DAO gobernará sobre las funciones que tengan el modificador *OnlyOwner*.

Gracias a este proceso, un usuario realizará la operación de propuesta, voto y ejecución mediante el contrato de gobierno (GovernorContract). Por todo ello, solo será necesario el manejo de dos contratos para el funcionamiento total de la plataforma, es decir, ElectoralManager y GovernorContract.

10.1.2.3 Preparación

Para realizar un ejemplo de verificación se necesita crear un usuario y dos promesas electorales. Para esta parte es necesario conocer la dirección de despliegue del contrato ElectoralManager.

Este proceso puede realizarse mediante scripts como se muestra en Ilustración 43, donde se crea un usuario y una promesa electoral.



```
/* New User*/
const registerIdTx = await electoralManagerContract
    .connect(addr1)
    .registerUser(_completeName, _namePoliticalParty, _isPoliticalParty);

const registerEndTx = registerIdTx.wait(1);
console.log(registerEndTx);

// Data Info Electoral Promise
const _tokenUri = "random uri";
const _isObligatory = false;

/* Create Electoral Promise*/
const createElectoralPromise = await electoralManagerContract
    .connect(addr1)
    .createElectoralPromise(_tokenUri, _isObligatory);
```

Ilustración 43: Preparación de datos para la verificación de una promesa

10.1.2.4 Propuesta

Para esta fase se utilizará el contrato de gobernanza. El usuario realizará una llamada de propuesta con los siguientes datos:

- **Dirección del contrato** sobre el que se aplicará la gobernanza, en este caso es la dirección de despliegue del contrato ElectoralManager.
- **Función del contrato:** función *approvePromise* junto al identificador de la promesa electoral.
- **Descripción de la Propuesta:** una descripción sobre la propuesta que se va a plantear.

Finalmente, se puede conocer el identificador único de la propuesta. Un ejemplo de este proceso sería la mostrada en la Ilustración 44.

```
// CODIFICACION DE FUNCION QUE GOBIERNA LA DAO
const encodeFunctionParameter =
  electoralManagerContract.interface.encodeFunctionData(
    FUNC_APPROVE_PROMISE,
    [PROPOSAL_ID_FOR_APPROVE]
  );

// ENVIO DE UNA PROPUESTA:
const proposeTxResponse = await governorContract.propose(
  [electoralManagerAddress], // Direccion contrato
  [0],                       // Otros Parametros
  [encodeFunctionParameter], // Funcion y Parametros
  DESCRIPCION_PROPUESTA     // Descrion sobre la propuesta
);

const proposeReceipt = await proposeTxResponse.wait(1);

await moveBlocks(VOTING_DELAY + 1);

const proposalId = proposeReceipt.events[0].args.proposalId;
```

Ilustración 44: Nueva propuesta para votación

10.1.2.5 Votación

Para realizar la votación se necesitará conocer el identificador de la propuesta, anteriormente obtenido e indicar el tipo de votación mediante un valor numérico, estos valores son [91]:

- **Cero:** indica estar en contra de la propuesta.
- **Uno:** indica estar a favor de la propuesta.
- **Dos:** indica abstención.

También se permite establecer una razón de voto, como se muestra en la Ilustración 45 con la función *castVoteWithReason* y su tercer parámetro.

```
const voteTxResponse = await governorContract.castVoteWithReason(
  proposalId,
  FOR, // A FAVOR (1)
  "reasonVote"
);
```

Ilustración 45: Votación de la propuesta

Una vez finalizado el periodo de votación, se puede preguntar en qué estado se encuentra la propuesta como se muestra en la Ilustración 46.

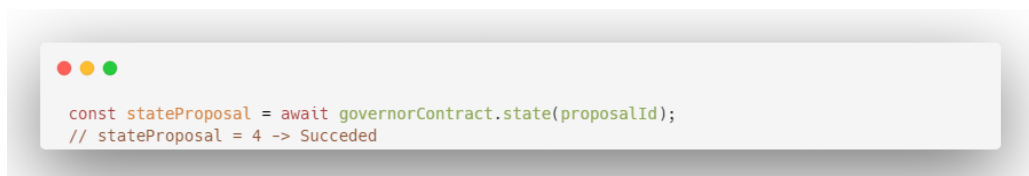


Ilustración 46: Comprobar el estado de la propuesta

El cuatro indicado en la Ilustración 46, es el valor asociado a *succeeded*.

10.1.2.6 Cola y ejecución

En esta última parte, si la votación ha resultado favorable, se procede a poner en cola para permitir finalmente ejecutar la propuesta.

Para pasar a cola se necesita indicar a **GovernorContract**, llamando a su función *queue* con los mismos parámetros que en la función de propuesta, como se muestra en la Ilustración 47. Esto permite pasar a ejecución la propuesta.



Ilustración 47: Proceso de poner en cola una propuesta aprobada

Y finalmente, la ejecución. Esta función también requiere los mismos parámetros que la función de propuesta, como se muestra en la Ilustración 48.

```
const executeTxResponse = await governorContract.execute(
  [electoralManagerAddress],
  [0],
  [encodeFunctionParameter],
  descriptionHash
);
```

Ilustración 48: Proceso de ejecución de una propuesta aprobada

Finalmente, se muestra la promesa de identificador **#0** antes de ser aprobada, en la Ilustración 49, y después de ser aprobada, en la Ilustración 50.

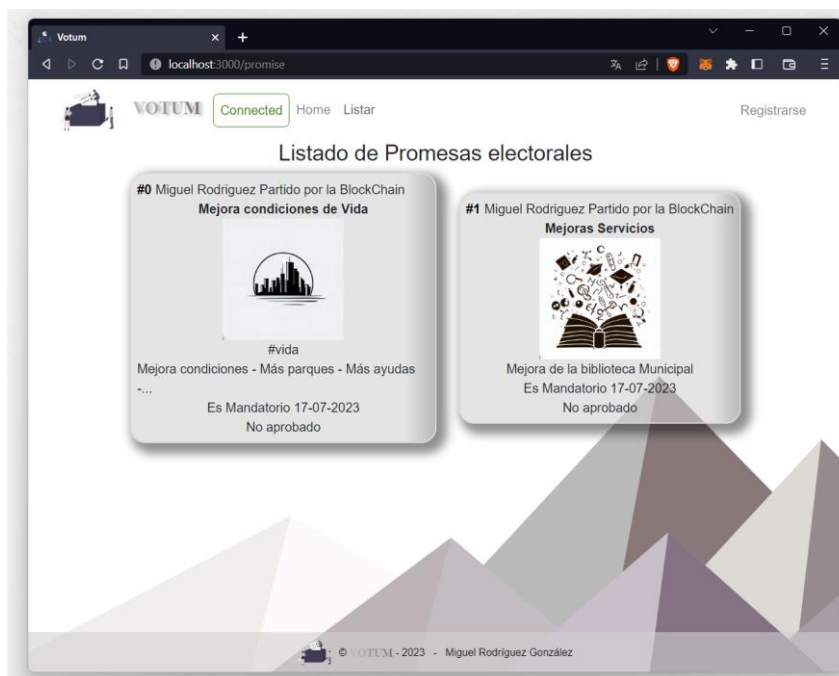


Ilustración 49: Promesa con id cero sin aprobar

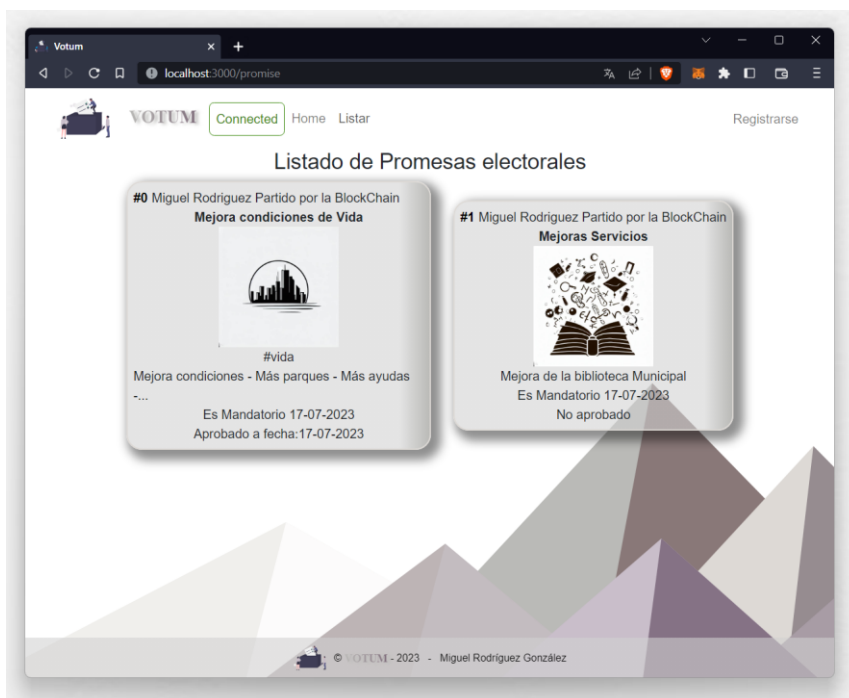


Ilustración 50: Promesa con id cero aprobada

10.2 RESULTADOS

Una vez implementada la prueba de concepto de una DAO *on-chain*, se indicarán las ventajas y desventajas ofrecen este tipo de soluciones.

10.2.1 Ventajas

Una DAO proporciona una serie de ventajas:

- Una organización totalmente transparente, segura y descentralizada.
- Sus reglas de goberna quedan dispuestas en la red para que cualquier persona pueda verlas y entenderlas.
- Cualquier persona puede acceder a la DAO, incluso sin un *frontend*, tan solo necesita saber la dirección del contrato de gobernanza.

10.2.2 Desventajas

Una gobernanza completa en cadena implica conocer varios puntos que podrían retractar a la gente de utilizarla y/o implementarla. Los problemas esenciales para afrontar son:

- El voto en la Blockchain no es gratuito, este pago por transacción en el caso de una DAO escala a cientos o tal vez miles, dependiendo del número de miembros que quieren participar.
- Un usuario que desee participar de manera activa realizando propuestas y votando, necesitaría poseer buenos fondos. Por lo tanto, si solo participan un conjunto de usuarios con grandes fondos se estaría cediendo el poder de encauzar las verificaciones a los grupos con mayor poder económico.
- El despliegue de una DAO tiene un coste que se debe tener en cuenta. Este impacto se analiza en detalle en el apartado 4.1.9.
- El problema de las cuentas relacionadas con una misma persona.

11 Manual de usuario

11.1 INTRODUCCIÓN

FoolMeOnce ofrece la plataforma VOTUM para la creación y visualización de promesas electorales. Permite a los ciudadanos la consulta de cualquier detalle de las promesas electorales, pudiendo conocer desde la fecha en que se creó hasta la formación política que la elaboró.

Esta plataforma también dispone de la funcionalidad necesaria para que los políticos registren todos los datos de las promesas de manera inalterable y única en una red Blockchain, ofreciendo más credibilidad para obtener el voto del ciudadano.

11.2 USO DE LA APLICACIÓN

En este apartado, se muestra como los usuarios, ciudadano y político, interactúan con la plataforma de creación y listado de promesas electorales.

Con el acceso a la plataforma, la página principal es la mostrada en la Ilustración 51

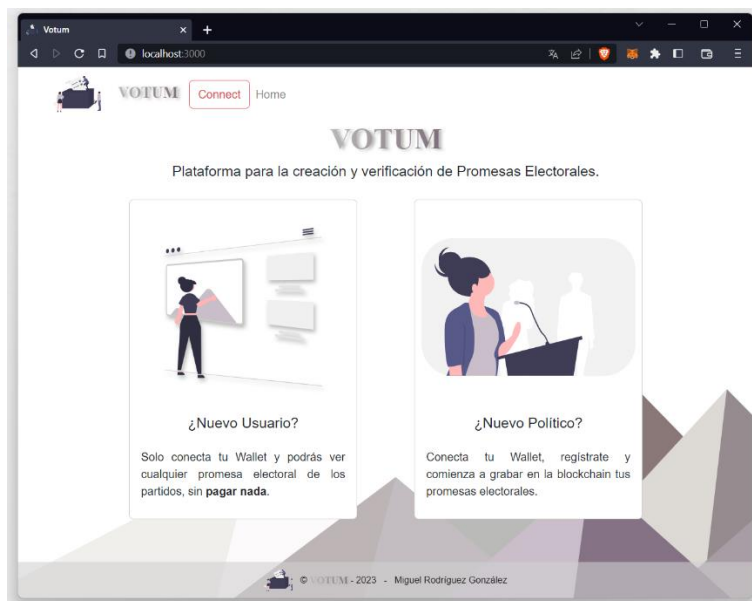


Ilustración 51: Página Principal de la plataforma

11.2.1 Conexión con la plataforma

El usuario para conectarse a la plataforma solo necesita darle al botón de conectar que se mostrará en rojo. Al conectarse satisfactoriamente a la plataforma, el botón de conexión se pondrá en verde. Realizada la conexión el usuario tendrá acceso al listado de promesas, mediante el menú de navegación.

Además, si el usuario desea grabar promesas electorales, puede proceder mediante un registro previo, también mediante el menú de navegación. Todo esto se muestra en la Ilustración 52.

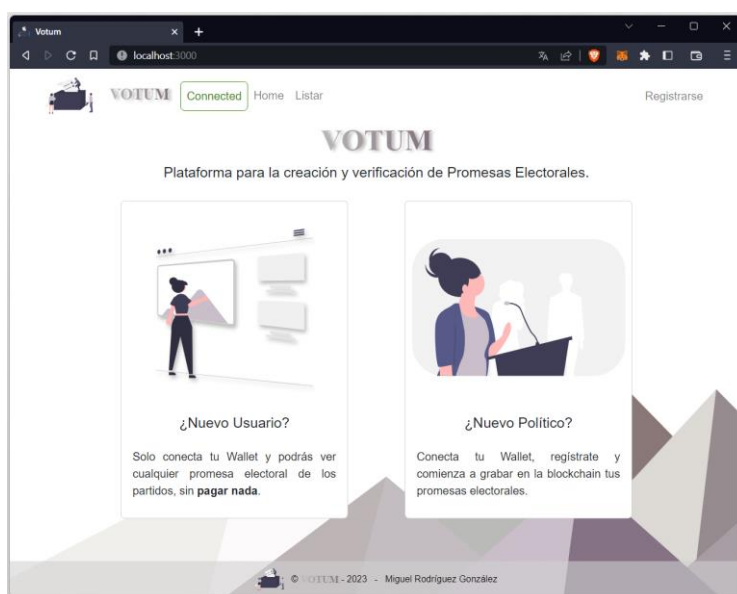


Ilustración 52: Conexión con la plataforma

11.2.2 Listado y visualización de promesas

El usuario al dirigirse al menú de listar podrá visualizar el conjunto de promesas electorales grabadas en la Blockchain, como se muestra en la Ilustración 53.

El ciudadano podrá acceder a la visualización completa de la promesa mediante su identificador único, dispuesto en la parte superior izquierda de la promesa. Al acceder a su visualización se mostrarán todos sus detalles, como se muestra en la Ilustración 54. En el caso de acceder desde un dispositivo móvil la visualización sería como lo mostrado en la Ilustración 55.

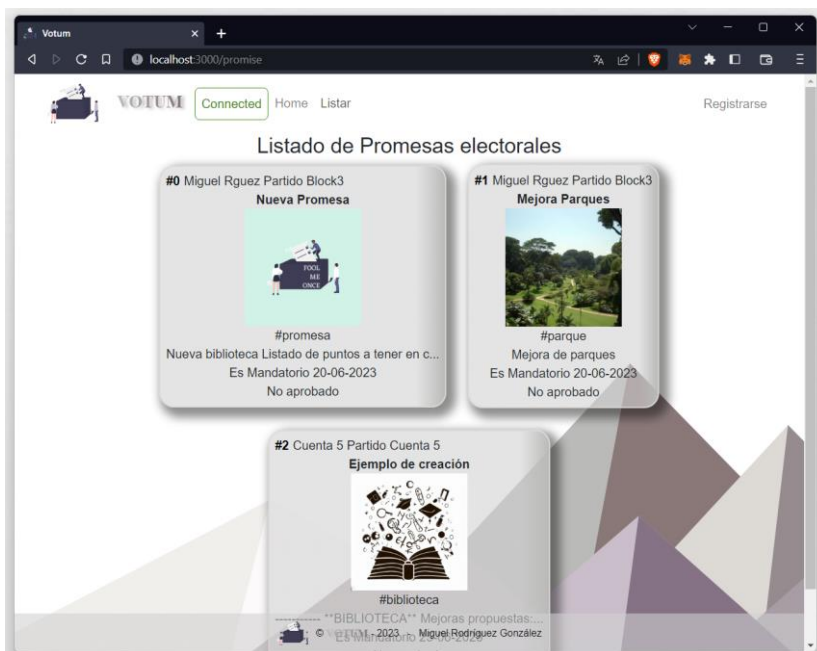


Ilustración 53: Listado de promesas electorales

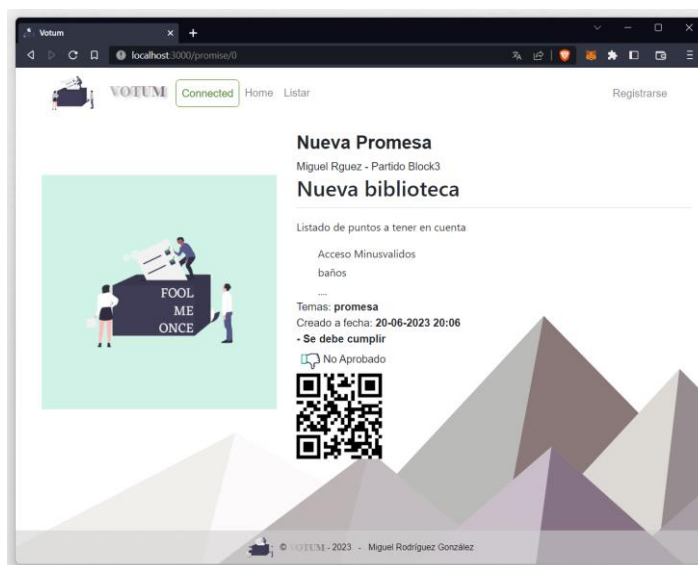


Ilustración 54: Visualización de una promesa

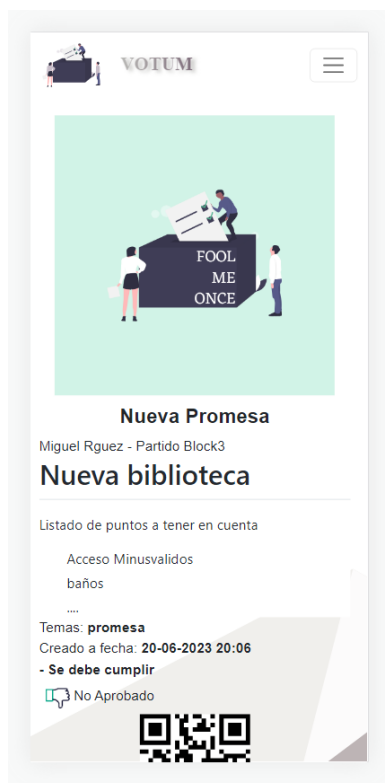


Ilustración 55: Visualización de una promesa desde un dispositivo móvil

11.2.3 Registro de políticos

Para permitir a los políticos grabar sus promesas y que sus datos, nombre y formación política, aparezcan siempre en sus promesas, se requiere un registro. El registro se realiza mediante un formulario, mostrado en la Ilustración 56. Este formulario solo será accesible si el político no fue registrado previamente y está conectado a la aplicación.

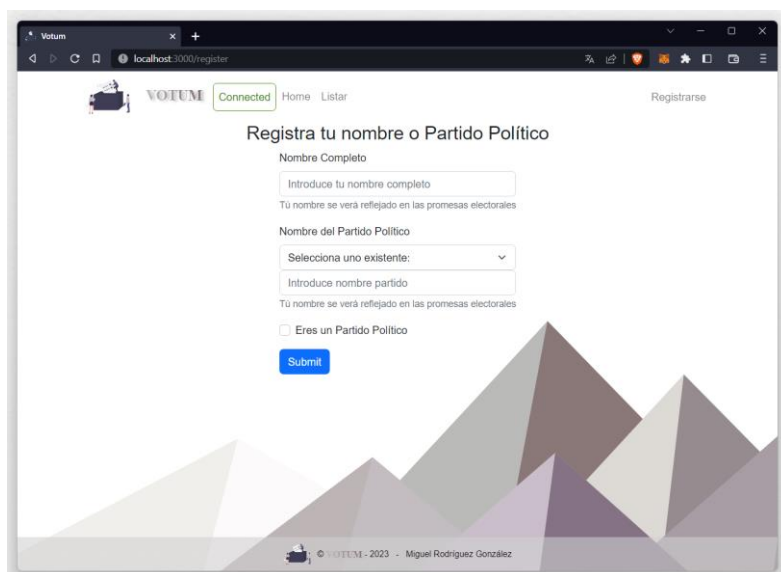
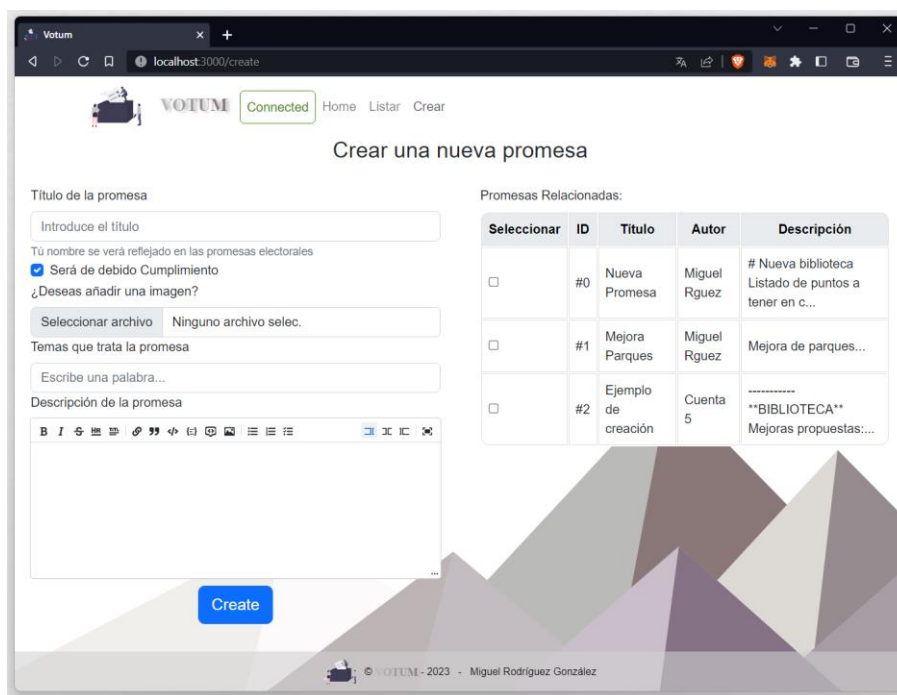


Ilustración 56: Registro de políticos

11.2.4 Creación de nuevas promesas electorales

Una vez que el político se haya registrado, ya podrá mediante el menú de navegación dirigirse la página de creación de promesas, como se muestra en la Ilustración 57.



Seleccionar	ID	Título	Autor	Descripción
<input type="checkbox"/>	#0	Nueva Promesa	Miguel Rguez	# Nueva biblioteca Listado de puntos a tener en c...
<input type="checkbox"/>	#1	Mejora Parques	Miguel Rguez	Mejora de parques...
<input type="checkbox"/>	#2	Ejemplo de creación	Cuenta 5	***** **BIBLIOTECA** Mejoras propuestas:...

Ilustración 57: Pantalla para la creación de nuevas promesas

12 Ampliaciones

En este capítulo, se describen algunas de las posibles ampliaciones al TFG realizado.

12.1 ESTABLECER DIFERENCIA ENTRE LOS USUARIOS

En el proceso de registro, se da la posibilidad de indicar si se es un partido político, de esta manera se da la posibilidad de como formación grabar las promesas que representa, como se muestra en la Ilustración 57.

Un nuevo evolutivo sería posibilitar que dicho usuario, es decir, el partido político, al presentar alguna promesa con su información solo indique el partido, ya que no es necesario presentar el nombre de la persona. Este evolutivo muestra el cambio en la Ilustración 58 e Ilustración 59.



Ilustración 58: Promesa electoral de un partido



Ilustración 59: Promesa electoral de un político

12.2 INVESTIGACIÓN PRÁCTICA DE UNA DAO OFF-CHAIN

Una ampliación sobre la investigación realizada sería realizar un ejemplo de transacciones firmadas mediante IPFS, permitiendo abaratar el coste de la votación y lograr convertir la DAO en una herramienta más barata para el ciudadano.

12.3 INVESTIGACIÓN DE HELIA, LA NUEVA HERRAMIENTA DE IPFS

Helia [92] es un cliente de IPFS para el lenguaje JavaScript, este surge en sustitución de *ipfs-http-client*, que al ser marcado como obsoleto hace poco tiempo no ha sido posible migrar por falta de tiempo. Este nuevo cliente da la posibilidad de ejecutar un nodo en la parte del cliente, es decir, en el navegador, logrando no necesitar un nodo de IPFS global conectado al *frontend*.

13 Problemas Encontrados

En este capítulo, comentaré el conjunto de problemas a los que me he tenido que enfrentar sobre este proyecto, quitando que se trata de una tecnología que no he estudiado anteriormente y partiendo de una base cero, me he visto metido en problemas curiosos

13.1 PROBLEMA CON LAS PRUEBAS Y ETHERS.JS

En el primer prototipo, se había decidido en una primera instancia que la relación entre las promesas fuera directamente grabado sobre la Blockchain y las pruebas unitarias así lo contemplaban. En una segunda iteración, al utilizarse IPFS como almacenamiento de los datos generales, se decidió trasladar la relación con otras promesas a IPFS, quedando más limpio el contrato y una transacción más barata.

El problema viene en que las pruebas no se realizó el cambio, es decir, sobre estos seguía existiendo la grabación de las relaciones entre promesas, lo curioso reside en que cada una de las pruebas funcionaba de manera correcta con la última versión de los contratos, cuando no debería ser así y debería de advertir que sobre uno de los argumentos.

Se podría pensar que no se realizó correctamente la compilación, pero no es así, se eliminaron los ABI de los contratos previamente a ejecutar las pruebas.

13.2 PROBLEMA CON NODE Y NPM

Durante la investigación de las DAO, se intentó realizar un ejemplo de implementación de una DAO utilizando el instalador de paquetes npm. El problema reside en que para el manejo de ciertas dependencias de Hardhat y Ethers se necesitaba sobrescribir algunos módulos, resultando en problemas, esto se solucionó con el instalador Yarn [93], muy parecido a npm.

13.3 PROBLEMA CON EL CLIENTE DE IPFS

Se necesita siempre un nodo corriendo de IPFS conectado, las alternativas que se podrían valorar son:

- Desplegar Front e IPFS en una máquina normal.

- Utilizar un servicio de terceros como *Infura* [94] o *nft.storage* [95] que mediante una clave permita subir archivos a IPFS, con los siguientes problemas:
 - Ambos tienen límite de datos.
 - Para utilizar Infura tienes que dar tu tarjeta de crédito.
- Profundizar más en el uso de Helia, mencionado en el apartado 12.3.

14 Conclusiones

Este TFG me ha emocionado desde el principio, la tecnología Blockchain está comenzando a incluirse como una solución a problemas actuales, un ejemplo es la nueva aplicación de la DGT: denominada DNI-Car [96], un servicio para la digitalización de la documentación sobre la Blockchain.

El hecho de realizar el TFG junto a HP SDCS y hacer uso de la metodología Scrum ha sido muy beneficioso, ya que me ha dado la oportunidad de investigar y desarrollar algo que nunca hasta ahora he conocido. Agradezco a mis tutores todo el apoyo en el desarrollo ya que han sido muy importantes para lograr los objetivos propuestos.

FoolMeOnce logra cumplir con su principal objetivo, consigue que las promesas electorales sean inalterables y se encuentren al alcance de toda la ciudadanía. También cabe destacar que estas promesas no estarán almacenadas en un servidor web centralizado, sino que se encontrarán en una red Blockchain, haciéndolo seguro, independiente y transparente para cualquier persona.

Además, no solo se aumenta la credibilidad de los políticos, sino que los ciudadanos tendrán el conocimiento de lo prometido. Este conocimiento permitirá a los ciudadanos juzgar en mejor medida a sus representantes, teniendo en cuenta qué se ha prometido y qué acciones se han llevado a cabo en anteriores legislaturas.

En cuanto a las Organizaciones Autónomas Descentralizadas se logra ofrecer una plataforma donde sus participantes se encuentran en las mismas condiciones, es decir, todos tienen la misma oportunidad de voto y todos pueden proponer una acción. Todo ello de manera sencilla, inalterable y totalmente accesible.

Es cierto que esta plataforma asiste al ciudadano en el recuerdo de lo prometido, mejorando la información de la que dispone, pero también es importante que los ciudadanos se involucren más en la política, ya que nos afecta todos. Y, al fin y al cabo, es una herramienta, no toma decisiones por nosotros.

Finalmente, solo debo indicar que me encuentro muy orgulloso de los resultados obtenidos, no solo por cumplir los objetivos marcados, sino por todo el camino realizado, la curiosidad de aprender y el trabajo constante. También quiero agradecer a mi familia y amigos todo el apoyo recibido.

15 Referencias

- [1] Plataforma Digital de la Biblioteca Nacional, «Plataforma Digital de la Biblioteca Nacional», Accedido: 27 de mayo de 2023. [En línea]. Disponible en: <https://www.bne.es/es>
- [2] A. Pastor García, «Newtral». <https://www.newtral.es/> (accedido 19 de mayo de 2023).
- [3] C. Jiménez Cruz y J. Montes, «Maldita». <https://maldita.es/> (accedido 19 de mayo de 2023).
- [4] Bit2me, «¿Qué es un ledger distribuido (libro mayor)?», *¿Qué es un ledger distribuido (libro mayor)?*, 11 de julio de 2019.
- [5] A. M. Antonopoulos, *Mastering Ethereum: Building Smart Contracts And Dapps*. 2018. Accedido: 29 de noviembre de 2022. [En línea]. Disponible en: <https://github.com/ethereumbook/ethereumbook>
- [6] Wikipedia, «Función hash», *Función hash*. https://es.wikipedia.org/wiki/Funci%C3%B3n_hash (accedido 11 de julio de 2023).
- [7] Bit2me, «¿Qué es una red P2P?», *¿Qué es una red P2P?*, 9 de julio de 2020.
- [8] Observatorio Blockchain, «Conoce los distintos tipos de blockchain», *Conoce los distintos tipos de blockchain*. <https://observatorioblockchain.com/hypernifty/redes-blockchain-tipos/> (accedido 24 de junio de 2023).
- [9] Satoshi Nakamoto, «Bitcoin WhitePaper», 2008. Accedido: 22 de mayo de 2023. [En línea]. Disponible en: <https://bitcoin.org/bitcoin.pdf>
- [10] V. Buterin, «Ethereum WhitePaper». https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum_Whitepaper_-_Buterin_2014.pdf (accedido 31 de mayo de 2023).
- [11] J. Maldonado, «¿Qué es Solana Network? Una blockchain de alta velocidad para DApps», *¿Qué es Solana Network? Una blockchain de alta velocidad para DApps*, 29 de abril de 2021. <https://es.cointelegraph.com/explained/solana-network-a-high-speed-blockchain-for-dapps#:~:text=Pues%20bien%2C%20Solana%20se%20sirve,Solana%20han%20identificado%20esta%20funci%C3%B3n>. (accedido 4 de julio de 2023).
- [12] Ethereum Foundation, «Prueba de Trabajo (PoW)». <https://ethereum.org/es/developers/docs/consensus-mechanisms/pow/> (accedido 28 de mayo de 2023).

- [13] Ethereum Foundation, «Prueba de Participación (PoS)». <https://ethereum.org/es/developers/docs/consensus-mechanisms/pos/> (accedido 28 de mayo de 2023).
- [14] Bit2me, «¿Qué es PoA (Proof of Authority – Prueba de Autoridad)?», *¿Qué es PoA (Proof of Authority – Prueba de Autoridad)?*, 1 de abril de 2019. <https://academy.bit2me.com/que-es-proof-of-authority-poa/> (accedido 13 de julio de 2023).
- [15] Bit2me, «¿Qué es una criptomoneda?», *¿Qué es una criptomoneda?*, 21 de marzo de 2018.
- [16] Bit2me, «Altcoins», *¿Qué son las altcoins y qué ofrecen al ecosistema de criptomonedas?*, 29 de febrero de 2016. <https://academy.bit2me.com/que-son-las-altcoins/> (accedido 17 de junio de 2023).
- [17] Bit2me, «¿Qué es una wallet o monedero de criptomonedas?», *¿Qué es una wallet o monedero de criptomonedas?*, 1 de agosto de 2018. <https://academy.bit2me.com/wallet-monederos-criptomonedas/> (accedido 2 de julio de 2023).
- [18] Bit2me, «¿Qué es el algoritmo de firma ECDSA?», *¿Qué es el algoritmo de firma ECDSA?* <https://academy.bit2me.com/que-es-ecdsa-curva-eliptica/> (accedido 2 de julio de 2023).
- [19] Bit2me, «¿Qué es EdDSA?», *¿Qué es EdDSA?*, 6 de junio de 2019. <https://academy.bit2me.com/que-es-eddsa/> (accedido 2 de julio de 2023).
- [20] MetaMask, «MetaMask», *What is a crypto Wallet?* <https://learn.metamask.io/lessons/what-is-a-crypto-wallet> (accedido 4 de junio de 2023).
- [21] M. García y Tutellus, «Qué es un token y cuáles son los principales», *Qué es un token y cuáles son los principales*, 7 de julio de 2022.
- [22] blog ITDO, «¿Qué es una dApp?» [https://www.itdo.com/blog/que-es-una-dapp-y-en-que-se-diferencia-de-una-aplicacion-normal/#:~:text=Una%20Dapp%20\(aplicaci%C3%B3n%20descentralizada\)%20es,descentralizada%20peer%2Dto%2Dpeer.](https://www.itdo.com/blog/que-es-una-dapp-y-en-que-se-diferencia-de-una-aplicacion-normal/#:~:text=Una%20Dapp%20(aplicaci%C3%B3n%20descentralizada)%20es,descentralizada%20peer%2Dto%2Dpeer.) (accedido 25 de mayo de 2023).
- [23] Ethereum Org, «Contract ABI Specification», *Solidity Docs*. <https://docs.soliditylang.org/en/v0.8.13/abi-spec.html> (accedido 4 de junio de 2023).
- [24] Ethereum Org, «JSON RPC - Ethereum». <https://ethereum.org/es/developers/docs/apis/json-rpc/> (accedido 4 de junio de 2023).

- [25] Web3 Org, «Web3JS», *Web3 Library*. <https://web3js.readthedocs.io/en/v1.10.0/> (accedido 4 de junio de 2023).
- [26] 101blockchains y J. Howell, «Web3.Js Vs Ethers.Js – Key Differences», *What Is Ethers.Js – A Detailed Guide*, 25 de enero de 2023. <https://101blockchains.com/web3-js-vs-ethers-js/> (accedido 4 de junio de 2023).
- [27] Ethereum Org, «What is The merge?», *Frequently Asked Questions*, 12 de mayo de 2023. <https://ethereum.org/es/developers/docs/consensus-mechanisms/pos/faqs/#what-is-the-merge> (accedido 19 de junio de 2023).
- [28] Etherscan, «Bloque Génesis Ethereum», 2015. <https://etherscan.io/block/0> (accedido 31 de mayo de 2023).
- [29] Etherscan, «Etherscan». <https://etherscan.io/> (accedido 19 de junio de 2023).
- [30] Etherscan, «Bloque Generado 16548688». <https://etherscan.io/block/16548688> (accedido 31 de mayo de 2023).
- [31] Eiki, «Estado Ethereum». <https://medium.com/@eiki1212/ethereum-state-trie-architecture-explained-a30237009d4e> (accedido 31 de mayo de 2023).
- [32] Lee, «Interpretación Árbol de estados», *Interpretación Árbol de estados*, 22 de junio de 2016. <https://ethereum.stackexchange.com/questions/268/ethereum-block-architecture/6413#6413> (accedido 18 de junio de 2023).
- [33] Ethereum, «Introduction to Ethereum Improvement Proposals (EIPs)». <https://ethereum.org/es/eips/> (accedido 3 de junio de 2023).
- [34] F. Vogelsteller y V. Buterin, «ERC-20: Token Standard», 2015. Accedido: 3 de junio de 2023. [En línea]. Disponible en: <https://eips.ethereum.org/EIPS/eip-20>
- [35] W. Entriken, D. Shirley, J. Evans, y N. Sachs, «ERC-721: Non-Fungible Token Standard», 24 de enero de 2018. <https://eips.ethereum.org/EIPS/eip-721> (accedido 3 de junio de 2023).
- [36] J. Dafflon, J. Baylina, y T. Shababi, «ERC-777: Token Standard», *ERC-777: Token Standard*, 20 de noviembre de 2017. <https://eips.ethereum.org/EIPS/eip-777> (accedido 3 de junio de 2023).
- [37] W. Radomski, A. Cooke, P. Castonguay, J. Therien, E. Binet, y R. Sandford, «ERC-1155: Multi Token Standard», *ERC-1155: Multi Token Standard*, 17 de junio de 2018. <https://eips.ethereum.org/EIPS/eip-1155> (accedido 3 de junio de 2023).

- [38] M. Zoltu, «ERC-5114: Soulbound Badge», *ERC-5114: Soulbound Badge. A token that is attached to a «soul» at mint time and cannot be transferred after that.*, 30 de mayo de 2022. <https://eips.ethereum.org/EIPS/eip-5114> (accedido 3 de junio de 2023).
- [39] C. Reitwießner, N. Johnson, G. Vogelsteller, J. Baylina, K. Feldmeier, y W. Entriken, «ERC-165: Standard Interface Detection», 23 de enero de 2018. <https://eips.ethereum.org/EIPS/eip-165> (accedido 3 de junio de 2023).
- [40] V. Buterin, «EIP-214: New opcode STATICCALL», *EIP-214: New opcode STATICCALL*, 13 de febrero de 2017. <https://eips.ethereum.org/EIPS/eip-214> (accedido 3 de junio de 2023).
- [41] C. Reitwießner, N. Johnson, F. Vogelsteller, J. Baylina, K. Feldmeier, y W. Entriken, «How to Detect if a Contract Implements ERC-165», *ERC-165: Standard Interface Detection*, 23 de enero de 2018. <https://eips.ethereum.org/EIPS/eip-165#How%20to%20Detect%20if%20a%20Contract%20Implements%20ERC-165> (accedido 3 de junio de 2023).
- [42] P. Collins, «What is a DAO? What is the Architecture of a DAO? (How to Build a DAO — High Level)», *What is a DAO? How to Build a DAO? (High Level)*, 21 de febrero de 2022. <https://betterprogramming.pub/what-is-a-dao-what-is-the-architecture-of-a-dao-how-to-build-a-dao-high-level-d096a97162cc> (accedido 5 de junio de 2023).
- [43] Bit2Me, «Curso Sobre Organizaciones Autónomas Descentralizadas», 2023. <https://learn.bit2me.com/cursos/curso-sobre-organizaciones-autonomas-descentralizadas-dao/> (accedido 5 de junio de 2023).
- [44] L. Torvalds y J. Hamano, «Git». 2007. Accedido: 15 de julio de 2023. [En línea]. Disponible en: <https://git-scm.com/>
- [45] GitLab, «GitLab», *GitLab*. <https://gitlab.com/> (accedido 15 de julio de 2023).
- [46] Atlassian, «Guía de la metodología scrum: qué es, cómo funciona y cómo empezar», *Qué es scrum y cómo empezar*. <https://www.atlassian.com/es/agile/scrum> (accedido 15 de julio de 2023).
- [47] Bit2me, «¿Qué es Bitcoin Script?», *¿Qué es Bitcoin Script?*, 28 de noviembre de 2019. <https://academy.bit2me.com/que-es-bitcoin-script/> (accedido 14 de julio de 2023).
- [48] Bit2me, «¿Qué es Cardano (ADA)?», *¿Qué es Cardano (ADA)?*, 19 de marzo de 2021. <https://academy.bit2me.com/que-es-cardano-ada/> (accedido 14 de julio de 2023).

- [49] A. Takyar, «Plutos - Cardano». <https://www.leewayhertz.com/smart-contracts-on-cardano/#What-programming-languages-does-Cardano-use-for-its-smart-contract-development?> (accedido 31 de mayo de 2023).
- [50] Solidity Team, «Solidity Lang ORG», Accedido: 3 de junio de 2023. [En línea]. Disponible en: <https://soliditylang.org/>
- [51] OpenZeppelin, «OpenZeppelin». <https://www.openzeppelin.com/contracts> (accedido 3 de junio de 2023).
- [52] Remix, «Remix IDE». <https://remix-project.org/> (accedido 3 de junio de 2023).
- [53] Vyperlang, «Vyper». <https://github.com/vyperlang/vyper> (accedido 3 de junio de 2023).
- [54] Patrick Collins, «Diferencia Gas - Lenguajes Smart Contracts», *SOLIDITY vs VYPER | Smart Contract Language SMACKDOWN*, 2020. <https://www.youtube.com/watch?v=sbc74oU94FM> (accedido 3 de junio de 2023).
- [55] Google, «Introduction to the Angular Docs», *Angular*. <https://angular.io/docs> (accedido 4 de junio de 2023).
- [56] Meta, «React», *Aprende React*. <https://es.react.dev/learn> (accedido 4 de junio de 2023).
- [57] Google, «Google for developers», *Google*. <https://developers.google.com/learn/topics/angular> (accedido 14 de julio de 2023).
- [58] L. Delgado, «Angular vs React:Cuál Elegir Para tu Aplicación», *Angular vs React:Cuál Elegir Para tu Aplicación*, 27 de diciembre de 2020. <https://www.freecodecamp.org/espanol/news/angular-vs-react-cual-elegir-para-su-aplicacion/> (accedido 4 de junio de 2023).
- [59] Google, «Angular CLI», *Angular CLI*. <https://angular.io/cli> (accedido 4 de junio de 2023).
- [60] Meta, «Meta», *Meta*. <https://about.meta.com/es/> (accedido 14 de julio de 2023).
- [61] OpenZeppelin, «ERC 721 - ERC721URIStorage», *ERC721URIStorage*. <https://docs.openzeppelin.com/contracts/4.x/api/token/erc721#ERC721URIStorage> (accedido 4 de junio de 2023).
- [62] Bit2Me, «¿Qué es un token ERC-1155 en Ethereum?», *¿Qué es un token ERC-1155 en Ethereum?*, 1 de marzo de 2021. <https://academy.bit2me.com/que-es-token-erc-1155/> (accedido 14 de julio de 2023).

- [63] Protocol Labs, «IPFS - What is IPFS», *IPFS - What is IPFS*. <https://docs.ipfs.tech/> (accedido 4 de junio de 2023).
- [64] Swarm Foundation, «Swarm». <https://www.ethswarm.org/why> (accedido 4 de junio de 2023).
- [65] Protocol Labs, «ipfs-http-client», *ipfs-http-client*. <https://www.npmjs.com/package/ipfs-http-client> (accedido 15 de abril de 2023).
- [66] Swarm, «Cliente Bee JS», *Bee JS*. <https://bee-js.ethswarm.org/docs/getting-started/> (accedido 15 de julio de 2023).
- [67] Swarm, «Postage Stamps - Swarm», *Upload or download on Swarm*. <https://bee-js.ethswarm.org/docs/upload-download/> (accedido 15 de julio de 2023).
- [68] Swarm Foundation, «How to upload data to the Swarm network», *How to upload data to the Swarm network*, 21 de junio de 2022. <https://medium.com/ethereum-swarm/how-to-upload-data-to-the-swarm-network-c0766c3ae381#:~:text=Buying%20postage%20stamps&text=Select%20the%20Files%20tab%20on,to%20specify%3A%20Depth%20and%20Amount>. (accedido 15 de julio de 2023).
- [69] Nomic Foundation, «Hardhat», *Hardhat*. <https://hardhat.org/> (accedido 4 de junio de 2023).
- [70] Consensys Software Inc, «Truffle». <https://trufflesuite.com/truffle/> (accedido 4 de junio de 2023).
- [71] ConsenSys Software Inc., «Ganache», *TruffleSuite Ganache*.
- [72] Consensys Software Inc, «Truffle Db». Accedido: 4 de junio de 2023. [En línea]. Disponible en: <https://trufflesuite.com/docs/truffle/db/>
- [73] Nomic Foundation, «Hardhat start», *Hardhat start*. <https://hardhat.org/hardhat-runner/docs/getting-started#quick-start> (accedido 15 de julio de 2023).
- [74] R. Moore, «Licencia Ethers - MIT», *Ethers*. <https://docs.ethers.org/v5/license/> (accedido 19 de julio de 2023).
- [75] V. Buterin, «Moving beyond coin voting governance», *Moving beyond coin voting governance*, 16 de agosto de 2021. <https://vitalik.ca/general/2021/08/16/voting3.html> (accedido 5 de junio de 2023).
- [76] Wikipedia, «Ataque Sybil - Descripción», *Ataque Sybil*. https://es.wikipedia.org/wiki/Ataque_Sybil (accedido 5 de junio de 2023).

- [77] Polygon Labs UI, «Polygon BlockChain». <https://polygon.technology/> (accedido 5 de junio de 2023).
- [78] Avalanche, «Avalanche». <https://www.avax.network/> (accedido 5 de junio de 2023).
- [79] Binance, «Binance Chain: La Blockchain para Intercambiar el Mundo». <https://www.binance.com/es/blog/all/binance-chain-la-blockchain-para-intercambiar-el-mundo-304219301536473088> (accedido 5 de junio de 2023).
- [80] Polygon Labs UI, «Arquitectura Polygon», *Introduction to Polygon PoS*. <https://wiki.polygon.technology/docs/pos/polygon-architecture/> (accedido 24 de junio de 2023).
- [81] Wikipedia, «Árbol de Merkle», *Árbol de Merkle*. https://es.wikipedia.org/wiki/%C3%81rbol_de_Merkle (accedido 5 de junio de 2023).
- [82] CoinMarketCap, «Valor Polygon», *Valor Polygon*. <https://coinmarketcap.com/es/currencies/polygon/> (accedido 15 de julio de 2023).
- [83] D. Philips y Lutz Sander, «¿Qué es Avalanche Network (AVAX)? El Competidor “Flexible” de Ethereum», *¿Qué es Avalanche Network (AVAX)? El Competidor «Flexible» de Ethereum*, 30 de mayo de 2022. <https://decrypt.co/es/resources/que-es-avalanche-network-avax-el-competidor-flexible-de-ethereum> (accedido 5 de junio de 2023).
- [84] CoinMarketCap, «Valor Avalanche», *Valor Avalanche*.
- [85] N. Antiporovich, «¿Qué es y cómo funciona la BNB Chain (ex Binance Smart Chain)?», 11 de marzo de 2022. <https://www.criptonoticias.com/cryptopedia/que-es-como-funciona-bnb-chain-ex-binance-smart-chain/> (accedido 5 de junio de 2023).
- [86] CoinMarketCap, «Valor Binance», *Valor Binance*. <https://coinmarketcap.com/currencies/bnb/> (accedido 15 de julio de 2023).
- [87] OpenZeppelin, «Governance», *Governance*. <https://docs.openzeppelin.com/contracts/4.x/api/governance> (accedido 19 de julio de 2023).
- [88] OpenZeppelin, «Governor», *Governance*. <https://docs.openzeppelin.com/contracts/4.x/api/governance#Governor> (accedido 16 de julio de 2023).

- [89] OpenZeppelin, «ERC20Votes», *ERC20Votes*.
<https://docs.openzeppelin.com/contracts/4.x/api/token/erc20#ERC20Votes> (accedido 7 de junio de 2023).
- [90] OpenZeppelin, «TimeLockController», *Goverance*.
<https://docs.openzeppelin.com/contracts/4.x/api/governance#TimelockController> (accedido 16 de julio de 2023).
- [91] OpenZeppelin, «Valores para la votación», *Governance*.
https://docs.openzeppelin.com/contracts/4.x/api/governance#IGovernor-COUNTING_MODE-- (accedido 16 de julio de 2023).
- [92] Protocol Labs, «Helia», *Helia*. <https://github.com/ipfs/helia> (accedido 10 de julio de 2023).
- [93] Yarn, «Yarn», *Yarn*. <https://yarnpkg.com/> (accedido 17 de julio de 2023).
- [94] Infura Inc, «Infura», *Infura*. <https://www.infura.io/> (accedido 16 de julio de 2023).
- [95] Protocol Labs, «nft storage», *nft storage*. <https://nft.storage/> (accedido 16 de julio de 2023).
- [96] DGT, «DGT y FENEVAL presentan el DNI-Car, un servicio de digitalización de la documentación de los vehículos de alquiler para evitar el robo de vehículos», *DGT y FENEVAL presentan el DNI-Car, un servicio de digitalización de la documentación de los vehículos de alquiler para evitar el robo de vehículos*.
<https://www.dgt.es/comunicacion/notas-de-prensa/dgt-y-feneval-presentan-el-dni-car-un-servicio-de-digitalizacion-de-la-documentacion-de-los-vehiculos-de-alquiler-para-evitar-el-robo-de-vehiculos/> (accedido 19 de julio de 2023).
- [97] Bit2me, «¿Qué es un exchange de criptomonedas?», *¿Qué es un exchange de criptomonedas?*, 4 de octubre de 2019. <https://academy.bit2me.com/que-es-exchange-criptomonedas/> (accedido 4 de julio de 2023).
- [98] Bit2me, «¿Qué son los oráculos blockchain?», *¿Qué son los oráculos blockchain?*, 3 de enero de 2020. <https://academy.bit2me.com/que-es-oraculos-blockchain/> (accedido 15 de julio de 2023).

