Universidad de Valladolid

Gobernanza Descentralizada para Organizaciones No Gubernamentales

Implementación en Besu

Martin Villafruela, Jorge
Rocha Gallo, Nataly
Ureña, Loly

20 de enero de 2026

Índice

6. Anexos

1. Descripción general del sistema

Este proyecto implementa un sistema de gobernanza descentralizada para Organizaciones No Gubernamentales (ONG) en el ámbito educativo, utilizando la tecnología blockchain mediante Hyperledger Besu. El sistema aborda dos problemas fundamentales identificados en la fase conceptual: la falta de transparencia financiera y la gobernanza centralizada.

1.1. Objetivo

Como demostración académica del uso de Blockchain y Solidity, el sistema implementado busca garantizar la trazabilidad completa de las donaciones y democratizar las decisiones sobre el uso de fondos mediante un mecanismo de votación basado en tokens de gobernanza. Cada donante recibe tokens proporcionales a su aportación, permitiéndoles votar por los proyectos que consideran prioritarios.

1.2. Actores principales

El sistema implementado contempla tres actores principales:

- **Donantes:** Personas físicas o jurídicas que realizan contribuciones a la ONG y participan en la votación de proyectos mediante tokens de gobernanza.
- **Voluntarios:** Miembros de la ONG encargados de la gestión de proyectos específicos.
- **Administrador (Owner):** Cuenta que despliega el contrato y tiene permisos para crear proyectos.
- **Proveedores:** Entidades que suministran los materiales necesarios para realizar los proyectos.

1.3. Flujo operativo

El funcionamiento del sistema se estructura en torno a cuatro operaciones principales:

1. **Registro de donantes:** Los usuarios se registran en el sistema especificando su tipo (individual o empresa).
2. **Realización de donaciones:** Los donantes envían fondos en ETH a proyectos específicos, recibiendo automáticamente tokens de gobernanza.
3. **Votación de proyectos:** Los donantes utilizan sus tokens para votar por los proyectos que desean apoyar, o desprestigiar a los que no considere decentes (únicamente si han donado a los mismos).
4. **Compra de materiales:** Los voluntarios responsables de los proyectos adquieren materiales a los proveedores, utilizando los fondos para ello (validando que el dinero de las donaciones se usa de manera legítima).
5. **Validación de fondos:** El voluntario responsable valida el uso correcto de los fondos recaudados a manera de oráculo y automáticamente los fondos donados se liberan para el proveedor.

2. Simplificación en base al diseño inicial

Durante la fase conceptual se diseñó un sistema ambicioso que incluía múltiples actores, mecanismos complejos de distribución de fondos, gestión de proveedores y un sistema de votaciones con ciclos anuales. Sin embargo, al enfrentar la implementación de la práctica cuyo objetivo es experimentar con sistemas distribuidos Blockchain y contratos solidity se tomaron decisiones de simplificación para garantizar un sistema funcional y comprensible.

2.1. Sistema implementado

El contrato final `ContratoONG.sol` se centra en las funcionalidades esenciales:

- **Estructura de datos simplificada:** Se mantienen tres estructuras principales: `Donante`, `Proyecto` y `Donacion`, eliminando la complejidad de proveedores y administración.
- **Flujo directo de donaciones:** Los fondos van directamente al proyecto especificado sin distribución porcentual automática.
- **Tokens de gobernanza básicos:** Se asigna 1 token por cada ETH donado, sin distinción por tipo de donante.
- **Votación simplificada:** Los donantes pueden votar en cualquier momento, gastando sus tokens para incrementar o disminuir el contador de votos del proyecto.
- **Control de acceso básico:** Solo el owner puede crear proyectos, validar fondos y cambiar estados, implementado mediante el modificador `soloOwner`.

2.1.1. Diagrama de estructuras de datos

La Figura 1 muestra las relaciones entre las principales entidades del sistema.

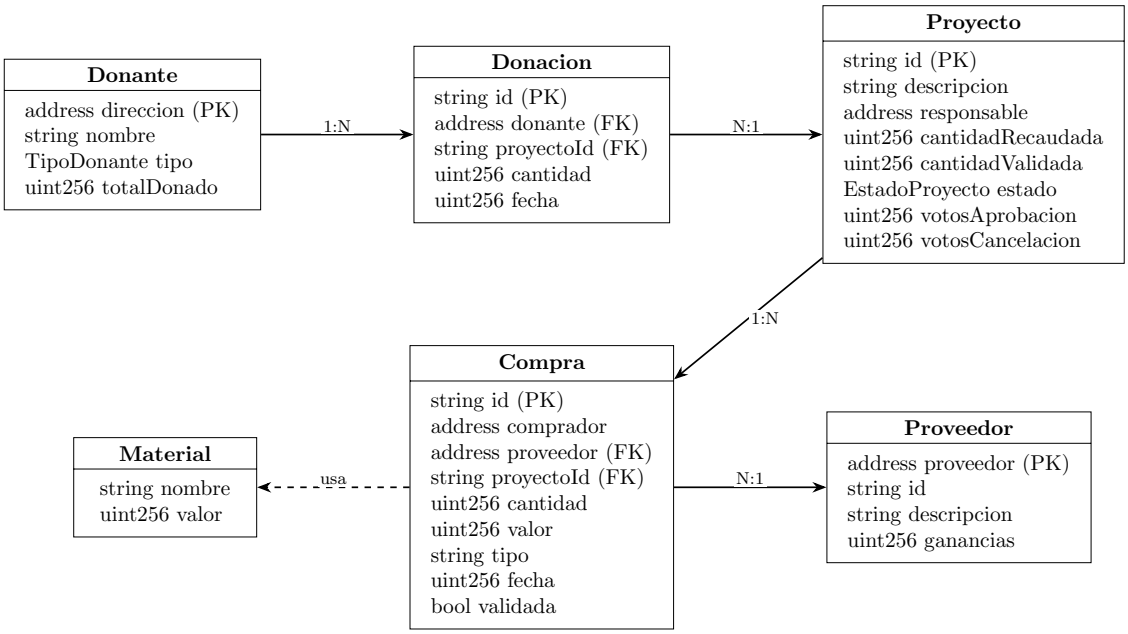


Figura 1: Diagrama entidad-relación de las estructuras de datos del contrato

2.2. Elementos eliminados del diseño original

El modelo conceptual inicial contemplaba diversos componentes que fueron simplificados en la implementación demostrativa:

- **Gestión de proveedores:** El diseño inicial incluía un registro de proveedores autorizados con historial de compras. En la versión a actual, todos los proveedores venden todo tipo de material, y el precio es estático, fijado en el contrato.
- **Administración:** Se planeaba una cuenta administrativa que recibiría automáticamente el 20 % de cada donación para gastos logísticos. Esta funcionalidad no se implementó.
- **Ciclos de votación y quema de tokens:** Se diseñó un sistema de votaciones anuales donde los tokens no utilizados se quemaban al final del ciclo. La implementación actual permite votaciones continuas sin restricciones temporales.
- **Objetivo de los tokens de votación:** En la misma línea, los tokens de gobernanza distribuidas ahora tienen una utilidad diferente. En vez de utilizarse para asignar los porcentajes de la donación que se lleva cada proyecto, ahora se utilizan para votar a favor/en contra de proyectos, que necesitan tener un balance de votos positivos determinado para estar activos (cuando un proyecto deja de estar activo, la cantidad donada sin validar se devuelve).
- **Distinción entre tipos de donante para tokens:** Aunque el contrato mantiene el enum `TipoDonante`, no se implementó la regla diferencial de asignación de tokens (raíz cuadrada para empresas vs. proporcional para individuos).

3. Implementación en Besu

El contrato inteligente fue desarrollado en Solidity[6] y desplegado en una red Hyperledger Besu[4] utilizando Remix IDE[3]. Esta sección describe la arquitectura del contrato, sus funciones principales y las consideraciones técnicas encontradas durante el despliegue (Apendice 6.3).

3.1. Arquitectura del contrato

El contrato `ContratoONG.sol` implementa un diseño modular, donde cada parte del contrato (donación, compra del material, votaciones, ...) viene definida en un "subcontrato" diferente para una mejor organización. De esta manera, en el contrato `ContratoONG.sol` únicamente se compila el resto de contratos, vinculados a éste por herencias con el siguiente orden:

OngBase → OngDonantes → OngProyectos → OngDonaciones → OngCompras → ContratoONG

OngBase: Define las estructuras de datos y mapeos básicos para donantes, proyectos, donaciones, compras y proveedores:

- **Estructuras (Structs):**
  - `Material`: almacena nombre y valor del material.
  - `Donante`: almacena dirección, nombre, tipo de donante y total donado.

- `Proyecto`: contiene ID, descripción, responsable, cantidades recaudada y validada, estado, votos de aprobación y votos de cancelación.
- `Donacion`: registra ID, donante, proyecto, cantidad y timestamp.
- `Compra`: registra ID, comprador, proveedor, proyecto, cantidad, valor, tipo de material, fecha y estado de validación.
- `Proveedor`: almacena ID, descripción, dirección del proveedor y ganancias acumuladas.
- `TrazabilidadDonacion`: estructura auxiliar que agrupa una donación con su proyecto y compras asociadas.

- **Enums:**
  - `TipoDonante`: Individual, Empresa.
  - `EstadoProyecto`: Propuesto, Activo, Cancelado.
- **Mapeos (Mappings):**
  - `mapping(address =>Donante) donantes`
  - `mapping(string =>Proyecto) proyectos`
  - `mapping(string =>Donacion) donaciones`
  - `mapping(string =>Compra) compras`
  - `mapping(address =>Proveedor) proveedores`
  - `mapping(string =>mapping(address =>bool)) donantesDeProyecto`
- **Arrays para iteración:**
  - `address[] listaDonantes`
  - `string[] listaProyectos`
  - `string[] listaDonaciones`
  - `string[] listaCompras`

Esta combinación de mappings para acceso directo y arrays para iteración permite tanto búsquedas eficientes como la capacidad de listar todos los registros desde el frontend.

3.2. Funciones principales

El contrato dividilo en los módulos mecionados (ver organización) implementa las siguientes operaciones:

3.2.1. Funciones de escritura

- `registrarDonante(string nombre, TipoDonante tipo)`: Permite a un usuario registrarse como donante. Verifica que no esté previamente registrado.
- `crearProyecto(string id, string descripcion, address responsable, EstadoProyecto estado)`: Función restringida al owner para crear nuevos proyectos. Valida que el ID no exista previamente y que el estado inicial no sea Cancelado.

- `donar(string proyectoId) payable`: Función principal que procesa donaciones. Recibe ETH, registra la donación, actualiza totales y asigna tokens de gobernanza (1 token por cada 0.001 ETH). Si el donante no está registrado, lo crea automáticamente como individuo. Solo permite donar a proyectos en estado Activo.
- `votarAprobacion(string proyectoId, uint256 cantidadVotos)`: Permite a los donantes registrados votar para aprobar un proyecto en estado Propuesto. Los tokens de gobernanza se queman al votar. Si se alcanza el umbral de votos mínimos (VOTOS_MINIMOS = 2), el proyecto pasa a estado Activo. Este umbral es debido a propósitos demostrativos y debe ajustarse en implementaciones reales.
- `votarCancelacion(string proyectoId, uint256 cantidadVotos)`: Permite votar para cancelar un proyecto. Para proyectos Activos, solo pueden votar quienes hayan donado a ese proyecto. Los tokens se queman al votar. Si se alcanza el umbral, el proyecto se cancela.
- `validarFondosProyecto(string proyectoId, uint256 cantidad)`: Función administrativa (solo owner) para marcar fondos como validados.
- `registrarProveedor(address proveedor, string nombre, string descripcion)`: Función restringida al owner para registrar proveedores autorizados.
- `realizarCompra(string compraId, string proyectoId, address proveedor, string tipoMaterial, uint128 cantidad)`: Permite al responsable del proyecto realizar compras. Verifica fondos suficientes y reserva el monto del balance del proyecto.
- `validarCompra(string compraId)`: El responsable del proyecto valida una compra después de verificar la recepción física del material. Al validar, se transfiere el ETH al proveedor (actúa como "oráculo humano").

3.2.2. Funciones de lectura

El contrato incluye funciones de consulta que no modifican el estado:

- `obtenerDonante(address)`, `obtenerProyecto(string)`, `obtenerDonacion(string)`, `obtenerCompra(string)`
- `obtenerTotalDonantes()`, `obtenerTotalProyectos()`, `obtenerTotalDonaciones()`
- `obtenerTokensGobernanza(address)`: retorna el balance de tokens de gobernanza de un donante
- `obtenerComprasPorProyecto(string proyectoId)`: retorna todas las compras asociadas a un proyecto
- `obtenerTrazabilidadDonante(address donante)`: retorna la trazabilidad completa de un donante, incluyendo sus donaciones, proyectos asociados y compras realizadas
- `getMaterialByName(string nombre)`: obtiene la información de un material por su nombre
- `obtenerBalance()`: retorna el balance de ETH del contrato

3.3. Control de acceso

Se implementa un sistema de permisos básico mediante el modificador `soloOwner`, que restringe funciones críticas al propietario del contrato:

```
modifier soloOwner() {
    require(msg.sender == owner,
            "Solo el owner puede ejecutar esto");
    _;
}
```

El owner se establece en el constructor como la dirección que despliega el contrato. También cada función tiene sus validaciones y restricciones específicas según la acción que realiza. Por ejemplo, solo los donantes registrados pueden votar, y solo el responsable del proyecto puede realizar compras.

3.4. Token de Gobernanza

El sistema incluye un contrato separado `TokenGobernanza.sol` que implementa un token ERC20 para la gobernanza de la ONG. Este token permite a los donantes participar en las votaciones de proyectos (Ver apendice 6.4).

- **Estándar**: ERC20 con nombre "Token Gobernanza ONG" y símbolo "TKN4GOOD"
- **Emisión**: Los tokens se mintean automáticamente al realizar donaciones (1 token por cada 0.001 ETH)
- **Consumo**: Los tokens se queman al votar, implementando un sistema de votación con peso económico
- **Control de acceso**: Solo el contrato de donaciones (ContratoONG) mintear y quemar tokens

El token utiliza las librerías de OpenZeppelin para garantizar la seguridad y el cumplimiento del estándar ERC20.

3.5. Eventos

Para facilitar la integración con el frontend y permitir el seguimiento de operaciones, se definieron los siguientes eventos:

- `DonacionRealizada(address indexed donante, string proyectoId, uint256 cantidad)`
- `ProyectoCreado(string id, string descripcion)`
- `DonanteRegistrado(address indexed direccion, string nombre)`
- `ProyectoAprobado(string id, uint256 votosTotal)`
- `ProyectoCancelado(string id, uint256 votosTotal, uint256 fondosAlFondoComun)`
- `VotoAprobacion(address indexed votante, string proyectoId, uint256 cantidadVotos)`
- `VotoCancelacion(address indexed votante, string proyectoId, uint256 cantidadVotos)`
- `CompraRealizada(address indexed comprador, string compraId, uint256 valor)`

- `CompraValidada(string compraId, address indexed validador, address indexed proveedor, uint256 valor)`
- `TokenGobernanzaActualizado(address indexed tokenAnterior, address indexed tokenNuevo)`

3.6. Proceso de despliegue

El contrato fue desplegado utilizando Remix IDE conectado a la red BLOCK LAB de Besu. El proceso incluyó (Ver apéndice 6.3):

1. Compilación del contrato con Solidity 0.8.0.
2. Conexión de MetaMask[1] a la red BLOCK LAB.
3. Configuración de permisos de dominio en MetaMask.
4. Despliegue mediante Remix utilizando el injected provider y london.
5. Obtención de la dirección del contrato desplegado.

3.7. Consideraciones técnicas importantes

Durante la implementación y despliegue se identificaron varios aspectos críticos:

- **Permisos de red en MetaMask:** Es fundamental configurar correctamente los permisos para el dominio específico en MetaMask, asegurando que la extensión utilice la red BLOCK LAB. Sin esta configuración, suelen aparecer problemas de conexión.
- **Inmutabilidad de contratos:** Los contratos en blockchain no son editables una vez desplegados. Cualquier modificación requiere desplegar un nuevo contrato con una nueva dirección, lo que implica actualizar la configuración en el frontend.
- **Coordinación frontend-contrato:** La dirección del contrato debe actualizarse manualmente en el archivo de configuración del frontend cada vez que se despliega una nueva versión, usando para ello el archivo ABI generado al desplegar el contrato.
- **Gas y costos:** Aunque Besu no requiere ETH real, se debe tener en cuenta el concepto de gas para futuras implementaciones en redes públicas.

4. Implementación del front end

Se desarrolló una aplicación web utilizando React con TypeScript y Vite como herramienta de construcción. El frontend se conecta al contrato inteligente mediante la biblioteca ethers.js y la extensión MetaMask, permitiendo a los usuarios interactuar con la blockchain de forma intuitiva.

La biblioteca **ethers.js** proporciona una interfaz de alto nivel para comunicarse con nodos de Ethereum, abstrayendo las complejidades del protocolo JSON-RPC subyacente. Por su parte, MetaMask actúa como proveedor Web3 (*provider*), inyectando el objeto `window.ethereum` en el navegador y gestionando de forma segura las claves privadas del usuario, de modo que estas nunca son expuestas a la aplicación.

Para que ethers.js pueda invocar las funciones del contrato desplegado en la red BLOCK LAB, es necesario disponer del ABI (*Application Binary Interface*), un archivo JSON que describe la firma de las funciones, sus parámetros y tipos de retorno. Este ABI se genera automáticamente durante la compilación del contrato en Solidity.

4.1. Tecnologías utilizadas

- **React 19[2]:** Framework principal para la interfaz de usuario
- **TypeScript:** Para tipado estático y mayor seguridad en el desarrollo
- **Vite:** Herramienta de construcción moderna y rápida
- **Material-UI (MUI):** Biblioteca de componentes para una interfaz profesional
- **TanStack Router:** Gestión de rutas y navegación
- **TanStack Query:** Manejo de estado y cache de datos blockchain
- **ethers.js[5]:** Librería para interactuar con Ethereum/Besu
- **MetaMask:** Extensión de navegador para gestión de billeteras

4.1.1. Hooks personalizados

Se crearon hooks de React para encapsular la lógica de interacción con el contrato:

- `useContract`: Hook base que inicializa la conexión con el contrato utilizando el ABI y la dirección configurada.
- `useProyectos`: Obtiene la lista de proyectos activos.
- `useDonaciones`: Consulta el historial de donaciones.
- `useDonante`: Recupera información del donante conectado.
- `useRealizarDonacion`: Gestiona el proceso de realizar una donación.
- `useRegistrarDonante`: Maneja el registro de nuevos donantes.
- `useVotarProyecto`: Permite votar por un proyecto específico.
- `useProveedores`
- `useRegistrarProveedor`
- `useCompras`

4.1.2. Capa de datos

Se implementó una arquitectura basada en queries y mutations siguiendo el patrón de TanStack Query:

- **Queries** (`data/query`): Funciones de solo lectura que consultan el estado del contrato
- **Mutations** (`data/mutations`): Operaciones que modifican el estado (donaciones, registros, votaciones)

4.1.3. Configuración del contrato

El archivo `contractConfig.ts` centraliza la configuración:

- Dirección del contrato desplegado.
- ABI del contrato (importado desde el JSON generado por Remix).
- Parámetros de conexión a la red.

4.2. Funcionalidades implementadas

La aplicación permite a los usuarios (Ver apéndice 6.2):

1. **Conectar billetera:** Mediante MetaMask, conectándose automáticamente a la red BLOCK LAB.
2. **Registrarse como donante:** Crear un perfil en el sistema.
3. **Visualizar proyectos:** Lista de proyectos activos con su información detallada.
4. **Consultar historial:** Ver todas las donaciones realizadas.
5. **Votar proyectos:** Utilizar tokens de gobernanza para votar.
6. **Ocupar fondos y validar su uso:** Los voluntarios pueden comprar materiales y validar el uso correcto de los fondos.

4.3. Manejo de transacciones

Un aspecto fundamental del diseño es el manejo correcto de las interacciones con la blockchain:

- **Transacciones de solo lectura:** Las consultas (queries) se ejecutan sin necesidad de conexión de billetera, permitiendo ver información pública antes de conectarse.
- **Transacciones de escritura:** Operaciones como donaciones y votaciones requieren que el usuario conecte su billetera y firme la transacción.
- **Feedback al usuario:** Se implementaron estados de carga y mensajes de error para informar sobre el progreso de las transacciones.
- **Actualización de estado:** Tras completar una transacción, se invalidan y refrescan automáticamente las queries afectadas.

4.4. Despliegue

El frontend fue desplegado en Vercel [7][8], lo que permite:

- Acceso público a la aplicación.
- Despliegue continuo desde el repositorio con CI/CD.
- Integración con MetaMask

Para acceder a la aplicación, visite: `https://uva-ong-block.vercel.app/`

4.5. Lecciones aprendidas

Durante el desarrollo del frontend se identificaron varios aspectos críticos:

1. **Blockchain como backend:** Al utilizar blockchain como capa de persistencia, todas las operaciones de lectura deben ejecutarse antes de requerir la conexión de billetera. Esto permite mostrar información al usuario y verificar que está en la red correcta antes de solicitar permisos.

2. **Actualización de direcciones:** Cada vez que se despliega un nuevo contrato, es necesario actualizar manualmente la dirección en `contractConfig.ts`. Este proceso es inevitable debido a la inmutabilidad de los contratos pero en un futuro se podría automatizar mediante scripts de despliegue al momento se lo realiza con variables de ambiente.

3. **Gestión de estado asíncrono:** Las interacciones con blockchain son inherentemente asíncronas y pueden fallar. TanStack Query facilita el manejo de estos estados (loading, error, success).

4. **Testing con cuentas de desarrollo:** Es fundamental tener múltiples cuentas de MetaMask configuradas para probar diferentes roles (owner, donantes, etc.). Por lo que por el momento una persona puede cumplir varios roles.

5. Conclusiones

Este proyecto demuestra la viabilidad de implementar un sistema de gobernanza descentralizada para ONGs utilizando tecnología blockchain con este MVP. A través del desarrollo del contrato `ContratoONG` y su integración con un frontend moderno, se han alcanzado los objetivos principales de transparencia y participación democrática.

5.1. Logros alcanzados

5.1.1. Transparencia financiera

El sistema implementado proporciona trazabilidad de las donaciones. Cada transacción queda registrada de forma inmutable en la blockchain, permitiendo a cualquier persona verificar:

- El origen de cada donación (dirección del donante).
- El destino específico de los fondos (proyecto beneficiado).
- El momento exacto de la transacción (timestamp).
- El total recaudado y validado para cada proyecto.

Esta transparencia proporciona verificación continua y en tiempo real.

5.1.2. Gobernanza democrática

El mecanismo de tokens de gobernanza permite la participación directa de los donantes en las decisiones sobre priorización de proyectos. Aunque simplificado respecto al diseño original, el sistema implementado cumple con el objetivo de democratizar la toma de decisiones, otorgando poder de voto proporcional a las contribuciones realizadas.

5.1.3. Aprendizajes técnicos

La implementación proporcionó experiencia práctica en:

- Desarrollo de contratos inteligentes en Solidity.
- Despliegue en Hyperledger Besu utilizando Remix.
- Integración frontend-blockchain mediante ethers.js.
- Gestión de billeteras con MetaMask.
- Configuración de permisos y redes personalizadas.
- Manejo de transacciones síncronas y asíncronas.
- Utilización de los tokens ERC20 para implementar un sistema de decisión basado en tokens de gobernanza.

5.2. Desafíos identificados

5.2.1. Inmutabilidad

La característica inmutable de los contratos blockchain garantiza que las reglas no pueden modificarse arbitrariamente (virtud para la transparencia), pero dificulta corregir errores o implementar mejoras (obstáculo para el desarrollo iterativo). Cada modificación requiere desplegar un nuevo contrato, migrar datos si es necesario y actualizar todas las integraciones. Esto sería algo importante a explorar a futuro con el uso de proxies o patrones de contratos actualizables.

5.2.2. Complejidad de configuración

La configuración correcta de MetaMask, especialmente los permisos de red por dominio, tiene su curva de aprendizaje. Esta barrera técnica podría dificultar la adopción por usuarios no técnicos, requiriendo interfaces más amigables o documentación exhaustiva.

5.2.3. Coordinación frontend-contrato

La necesidad de actualizar manualmente la dirección del contrato en el frontend tras cada despliegue introduce un punto de fricción en el flujo de desarrollo. Aunque lo implementamos con variables de ambiente, en un entorno de producción sería ideal automatizar este proceso mediante scripts de despliegue.

5.2.4. Complejidad del sistema de gobernanza

El hecho de tener que controlar todos los posibles escenarios en los que los donantes pueden actuar de forma maliciosa o egoísta añade una capa significativa de complejidad al diseño del sistema de gobernanza. Nos encontramos con varios de estos escenarios durante la implementación, y aunque se implementaron algunas medidas básicas para mitigarlos, el sistema actual sigue siendo vulnerable a ciertos tipos de abuso.

5.3. Trabajo futuro

Para evolucionar este proyecto hacia un sistema de producción, se identifican las siguientes áreas de mejora:

1. **Implementar distribución automática:** Recuperar la lógica de porcentajes del diseño original para automatizar la asignación de fondos según votaciones.

2. **Sistema de proveedores:** Añadir registro y validación de proveedores autorizados para completar el ciclo de trazabilidad hasta el gasto final.

3. **Optimización de gas:** Revisar estructuras de datos y lógica para reducir costos de transacción, especialmente importante si se migra a redes públicas.

4. **Mejorar la lógica de uso de los tokens de gobernanza:** Implementar una lógica más compleja y segura para el uso de los tokens. El sitema actual tiene varios puntos débiles que confían en que los usuarios "sean buenos".

   Por ejemplo, un donante puede comprar tokens de gobernenza al resto de donantes, votando en contra de un proyecto al que ha donado, haciendo que se le devualva su dinero, incluso puede que más del otorgado inicialmente, o simplemente tener plena postestad para decidir que un proyecto se reanude, o se cancele.

5. **Testing exhaustivo:** Desarrollar suite completa de tests unitarios y de integración utilizando Hardhat o Foundry.

6. **Auditoría de seguridad:** Antes de manejar fondos reales, realizar auditoría profesional del contrato.

Referencias

[1] Metamask - a crypto wallet & gateway to blockchain apps. https://metamask.io/, 2026. Accedido: 4-ene-2026.

[2] React - a javascript library for building user interfaces. https://react.dev/, 2026. Accedido: 4-ene-2026.

[3] Remix - ethereum ide. https://remix.ethereum.org/, 2026. Accedido: 4-ene-2026.

[4] Hyperledger Foundation. Hyperledger besu. https://www.hyperledger.org/use/besu, 2026. Accedido: 4-ene-2026.

[5] Moore, R. ethers.js - a complete ethereum library. https://docs.ethers.org/, 2026. Accedido: 4-ene-2026.

[6] Solidity Team. Solidity documentation. https://docs.soliditylang.org/, 2026. Accedido: 4-ene-2026.

[7] Vercel Inc. Vercel: Develop, preview, and ship frontend applications. https://vercel.com, 2025. Accessed: 2026-01-17.

[8] Vercel Inc. Vercel platform documentation. https://vercel.com/docs, 2025. Accessed: 2026-01-17.

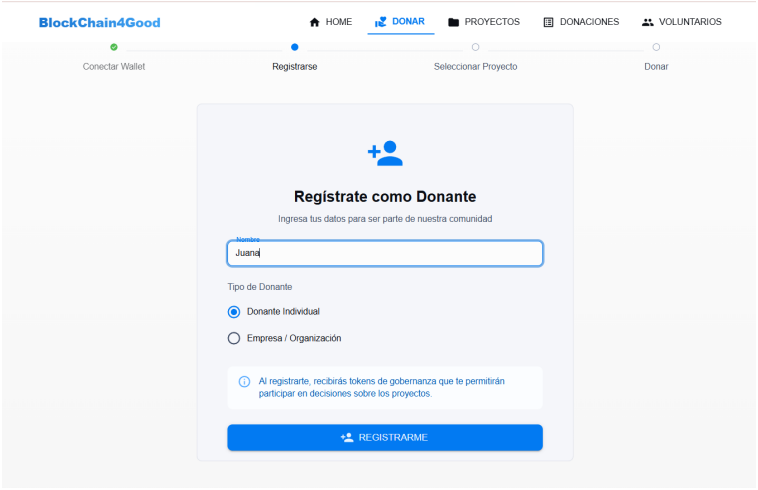
6. Anexos

6.1. Repositorio y página web

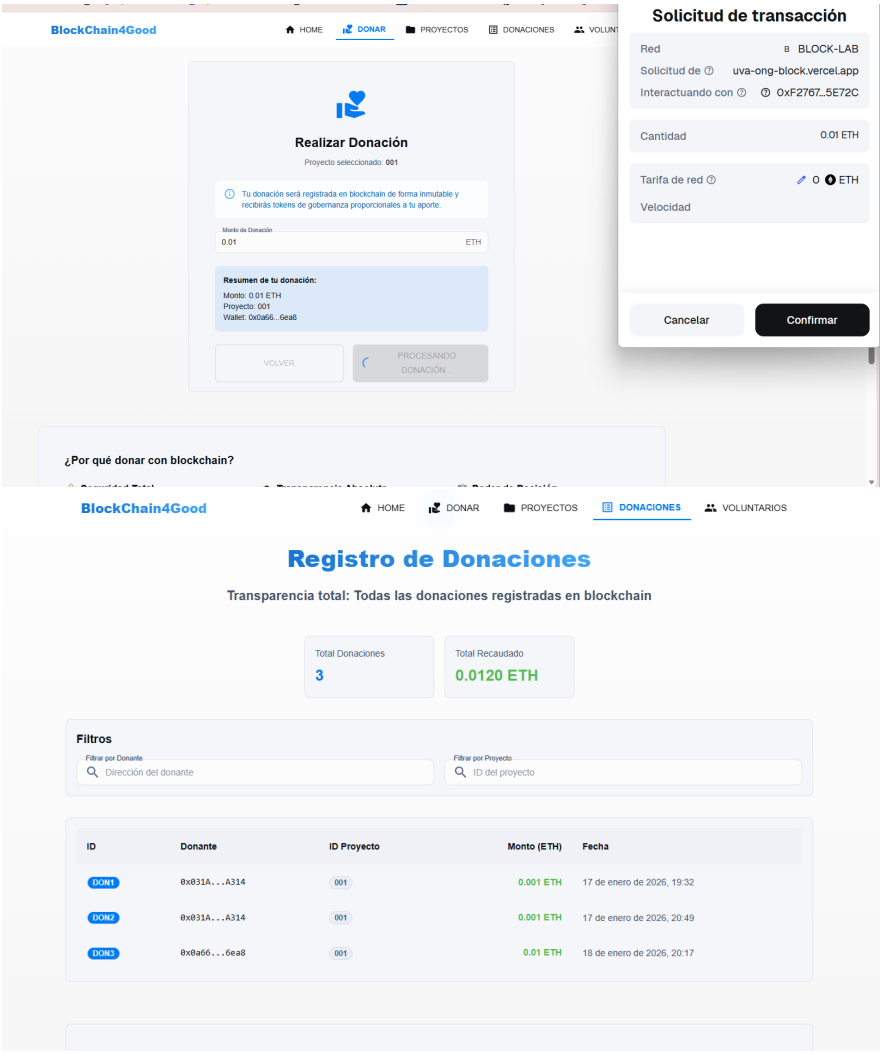
El código fuente de la aplicación está disponible en el siguiente repositorio de GitHub: https:github.comnatar10uva-ong-block
Además, se puede visitar e interactuar con la aplicación a través del siguiente enlace: https:uva-ong-block.vercel.app

6.2. *Walkthrough* de la aplicación

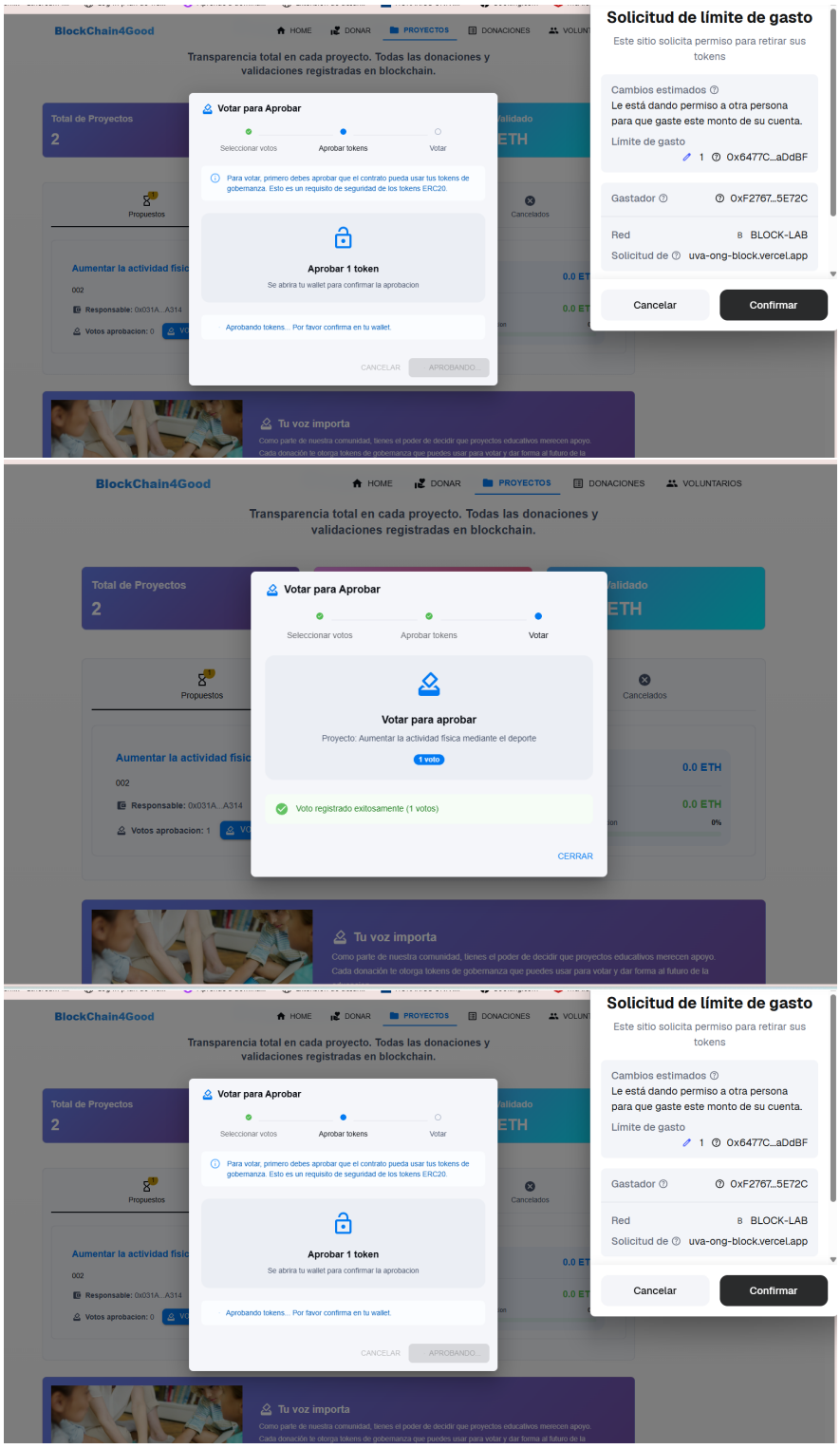
Primero, el usuario *Donante* se registra en la plataforma, proporcionando un nombre. Esto hará saltar la extensión de Metamask, donde el usuario deberá seleccionar la cuenta con la que se registra a la red blockchain.



Una vez registrado, el usuario puede empezar a visualizar y donar a los diferentes proyectos de la ONG:

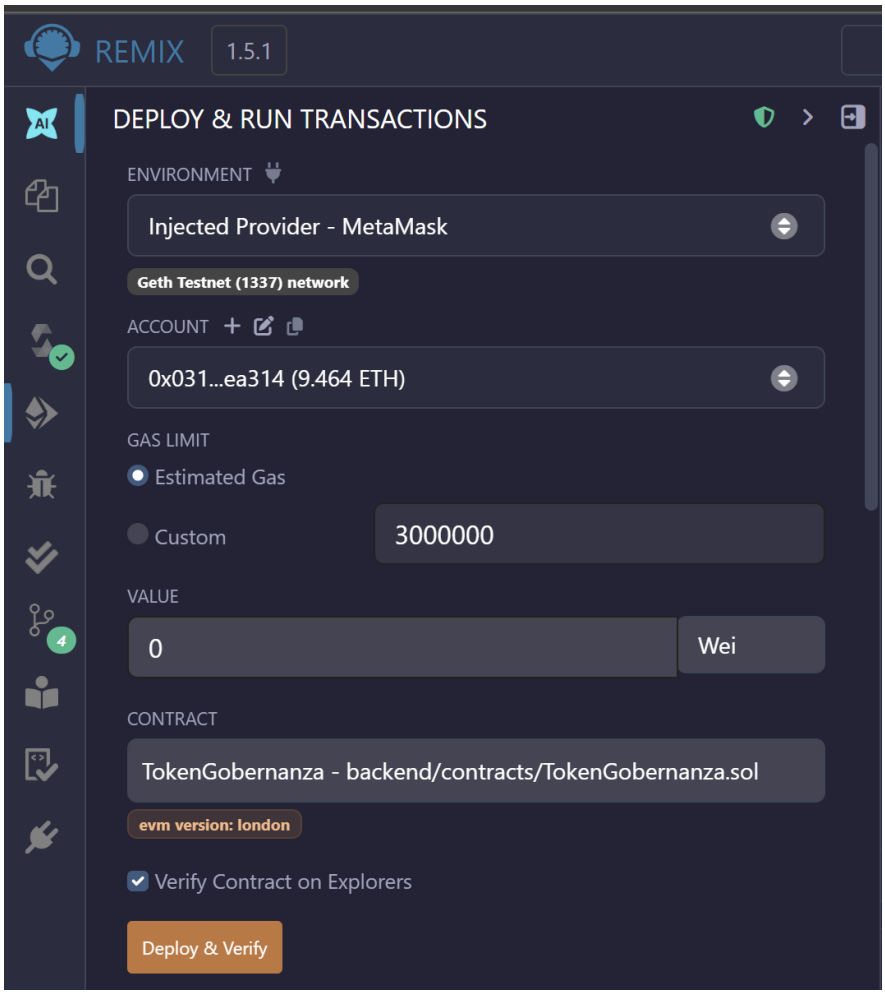


Gracias a las donaciones, el donante recibirá votos, que podrá usar para votar para que se aprueben nuevos proyectos, cuando se supere el margen establecido.

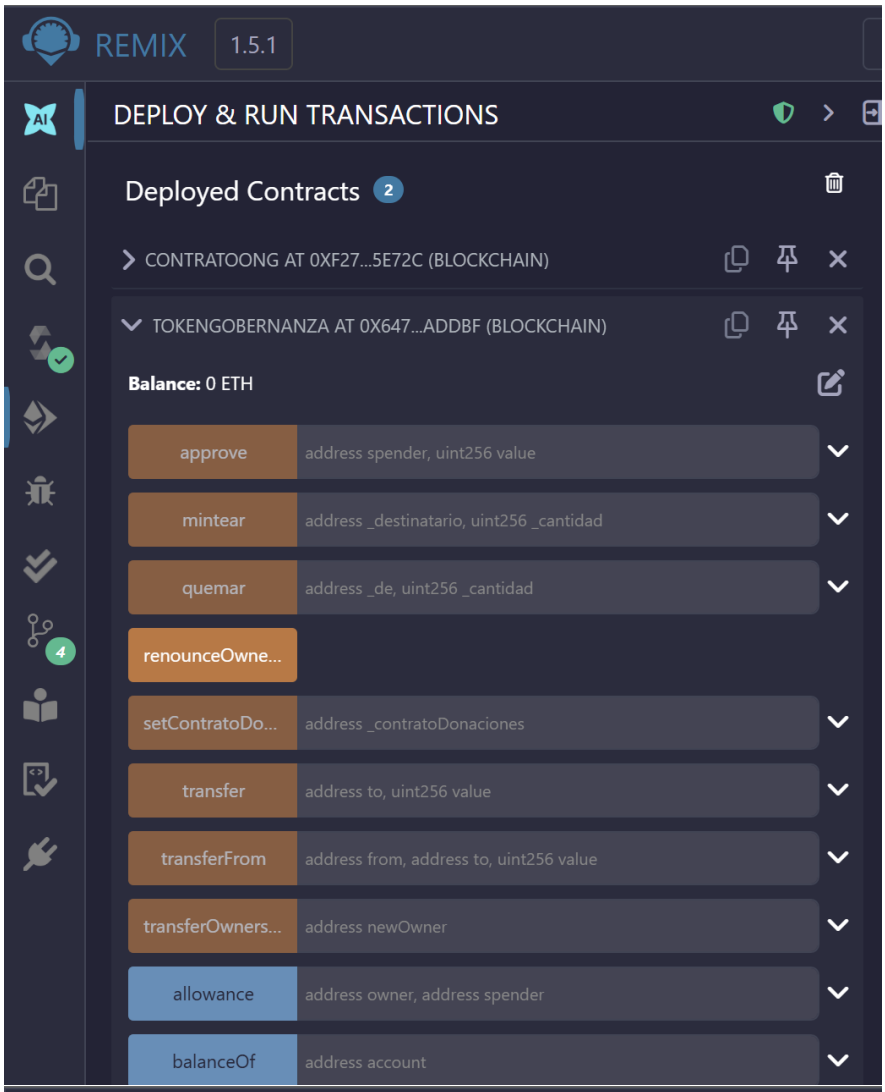


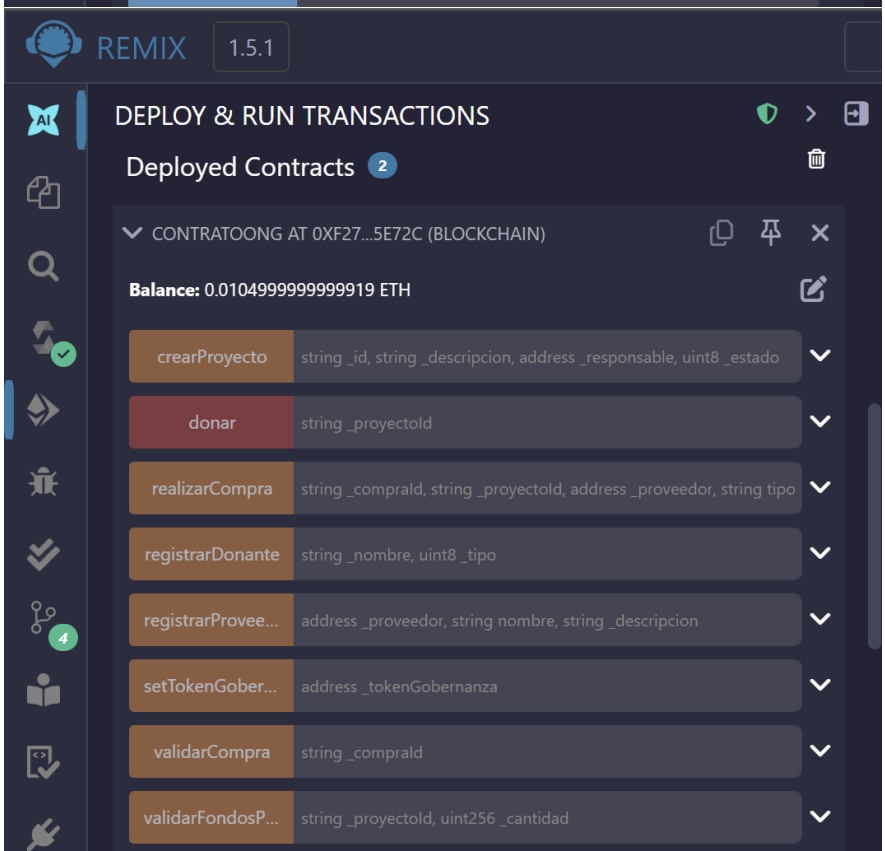
6.3. Walkthrough *Remix*

El deploy de la aplicación se realizó a través de Remix IDE, compilándolo con el compilador usando el evm versión *London*.



Para ello, hay que hacer el deploy del contrato en un orden concreto, ya que primero hay que inicializar el contrato del token ERC20 sobre el que se moverán los votos, para a continuación desplegar el contrato principal de la ONG pasándole la dirección del contrato por el que se gestionarán los tokens de Gobernanza.


REMIX
1.5.1
DEPLOY & RUN TRANSACTIONS
Deployed Contracts
2
CONTRATOONG AT 0XF27...5E72C (BLOCKCHAIN)
TOKENGOBERNANZA AT 0X647...ADDBF (BLOCKCHAIN)
Balance: 0 ETH
approve
address spender, uint256 value
mintear
address _destinatario, uint256 _cantidad
quemar
address _de, uint256 _cantidad
renounceOwne...
setContratoDo...
address _contratoDonaciones
transfer
address to, uint256 value
transferFrom
address from, address to, uint256 value
transferOwners...
address newOwner
allowance
address owner, address spender
balanceOf
address account


REMIX
1.5.1
DEPLOY & RUN TRANSACTIONS
Deployed Contracts
2
CONTRATOONG AT 0XF27...5E72C (BLOCKCHAIN)
Balance: 0.0104999999999919 ETH
crearProyecto
string _id, string _descripcion, address _responsable, uint8 _estado
donar
string _proyectoId
realizarCompra
string _compraId, string _proyectoId, address _proveedor, string tipo
registrarDonante
string _nombre, uint8 _tipo
registrarProvee...
address _proveedor, string nombre, string _descripcion
setTokenGober...
address _tokenGobernanza
validarCompra
string _compraId
validarFondosP...
string _proyectoId, uint256 _cantidad

6.4. Token de Gobernanza