—

Universidad de Valladolid

Gobernanza Descentralizada para Organizaciones No Gubernamentales

Implementación en Besu

Martin Villafruela, Jorge
Rocha Gallo, Nataly
Ureña, Loly

4 de enero de 2026

Índice

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

El sistema 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.
- **Responsables de proyectos:** Miembros de la ONG encargados de la gestión de proyectos específicos.
- **Administrador (Owner):** Cuenta que despliega el contrato y tiene permisos para crear proyectos y validar fondos.

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.
4. **Validación de fondos:** El administrador valida el uso correcto de los fondos recaudados.

2. Implementación en Besu

El contrato inteligente fue desarrollado en Solidity y desplegado en una red Hyperledger Besu utilizando Remix IDE. Esta sección describe la arquitectura del contrato, sus funciones principales y las consideraciones técnicas encontradas durante el despliegue.

2.1. Arquitectura del contrato

El contrato `ONGDonaciones.sol` implementa un diseño modular con las siguientes estructuras de datos:

- **Estructuras (Structs):**
  - `Donante`: almacena dirección, nombre, tipo, total donado y tokens de gobernanza
  - `Proyecto`: contiene ID, descripción, responsable, cantidades recaudada y validada, estado y votos
  - `Donacion`: registra ID, donante, proyecto, cantidad y timestamp
- **Mapeos (Mappings):**
  - `mapping(address =>Donante) donantes`
  - `mapping(string =>Proyecto) proyectos`
  - `mapping(string =>Donacion) donaciones`
- **Arrays para iteración:**
  - `address[] listaDonantes`
  - `string[] listaProyectos`
  - `string[] listaDonaciones`

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.

2.2. Funciones principales

El contrato implementa las siguientes operaciones:

2.2.1. 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)`: Función restringida al owner para crear nuevos proyectos. Valida que el ID no exista previamente.
- `donar(string proyectoId) payable`: Función principal que procesa donaciones. Recibe ETH, registra la donación, actualiza totales y asigna tokens de gobernanza (1 token por ETH). Si el donante no está registrado, lo crea automáticamente como individuo.
- `votarProyecto(string proyectoId, uint256 cantidadVotos)`: Permite a los donantes gastar sus tokens para votar por proyectos. Verifica que el donante tenga tokens suficientes y que el proyecto esté activo.
- `validarFondosProyecto(string proyectoId, uint256 cantidad)`: Función administrativa para marcar fondos como validados/gastados.
- `cambiarEstadoProyecto(string proyectoId, EstadoProyecto nuevoEstado)`: Permite al owner cambiar el estado de un proyecto (Activo/Cancelado).

2.2.2. Funciones de lectura

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

- `obtenerDonante(address)`, `obtenerProyecto(string)`, `obtenerDonacion(string)`
- `obtenerTotalDonantes()`, `obtenerTotalProyectos()`, `obtenerTotalDonaciones()`
- `obtenerBalance()`: retorna el balance de ETH del contrato

2.3. 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.

2.4. 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)`
- `VotacionRealizada(address indexed donante, string proyectoId, uint256 cantidad_votos)`

2.5. Proceso de despliegue

El contrato fue desplegado utilizando Remix IDE conectado a la red BLOCK LAB de Besu. El proceso incluyó:

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

2.6. 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.
- **Gas y costos:** Aunque Besu no requiere ETH real, se debe tener en cuenta el concepto de gas para futuras implementaciones en redes públicas.

3. 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 MetaMask, permitiendo a los usuarios interactuar con la blockchain de forma intuitiva.

3.1. Tecnologías utilizadas

- **React 19:** 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:** Librería para interactuar con Ethereum/Besu
- **MetaMask:** Extensión de navegador para gestión de billeteras

3.2. Arquitectura del frontend

La aplicación se estructura en capas separando responsabilidades:

3.2.1. 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

3.2.2. 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)

3.2.3. 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

3.3. Funcionalidades implementadas

La aplicación permite a los usuarios:

1. **Conectar billetera:** Mediante MetaMask, conectándose automáticamente a la red BLOCK LAB
2. **Visualizar proyectos:** Lista de proyectos activos con su información detallada
3. **Realizar donaciones:** Seleccionar un proyecto y enviar ETH
4. **Consultar historial:** Ver todas las donaciones realizadas
5. **Votar proyectos:** Utilizar tokens de gobernanza para votar
6. **Registrarse como donante:** Crear un perfil en el sistema

3.4. Integración con MetaMask

La conexión con MetaMask fue un aspecto crítico de la implementación. Se identificaron y resolvieron varios desafíos:

3.4.1. Consideraciones importantes

- **Permisos de dominio:** MetaMask debe tener permisos correctamente configurados para el dominio (tanto en desarrollo local como en producción). Sin los permisos adecuados para usar la red BLOCK LAB, aparecen errores de conexión.
- **Conexión automática:** MetaMask se conecta automáticamente tanto en desarrollo local (localhost) como en el servidor desplegado en Vercel. Lo esencial es configurar los permisos de red correctamente.
- **Verificación de red:** El frontend verifica que el usuario esté conectado a la red BLOCK LAB antes de permitir transacciones.
- **Reconocimiento de billetera:** Una vez configurados los permisos, MetaMask reconoce inmediatamente la billetera conectada.

3.5. 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.

3.6. Diseño de interfaz

La aplicación utiliza Material-UI para proporcionar una experiencia de usuario moderna:

- Dashboard con estadísticas de donaciones
- Cards visuales para cada proyecto con imágenes
- Formularios intuitivos para donaciones y registro
- Tablas con el historial completo de transacciones
- Modo claro/oscuro adaptativo

3.7. Despliegue

El frontend fue desplegado en Vercel, lo que permite:

- Acceso público a la aplicación
- Despliegue continuo desde el repositorio
- HTTPS automático
- Integración con MetaMask en entorno de producción

3.8. 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.

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.).

4. Conclusiones

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

4.1. Logros alcanzados

4.1.1. Transparencia financiera

El sistema implementado proporciona trazabilidad completa 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 elimina la dependencia de auditorías periódicas tradicionales, proporcionando verificación continua y en tiempo real.

4.1.2. 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.

4.1.3. 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

4.2. Desafíos identificados

4.2.1. Inmutabilidad como obstáculo y virtud

La característica inmutable de los contratos blockchain presenta un dilema: 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.

4.2.2. Complejidad de configuración

La configuración correcta de MetaMask, especialmente los permisos de red por dominio, resultó ser un punto crítico. Esta barrera técnica podría dificultar la adopción por usuarios no técnicos, requiriendo interfaces más amigables o documentación exhaustiva.

4.2.3. 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 existen soluciones como sistemas de nombres (ENS) o proxies actualizables, estas añaden complejidad adicional.

4.3. Simplificaciones y su impacto

Las decisiones de simplificación tomadas durante la implementación tuvieron efectos mixtos:
Aspectos positivos:

- Mayor facilidad de comprensión del código
- Menos superficie de ataque para vulnerabilidades
- Desarrollo más rápido y debugging simplificado
- Menor costo de gas en cada operación

Funcionalidades perdidas:

- No hay control automático de distribución de fondos
- Falta el mecanismo de proveedores verificados

- No se implementó la cuenta administrativa separada
- Sin ciclos de votación con renovación de tokens

4.4. 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. **Mejoras de seguridad:**
   - Implementar pausado de emergencia (circuit breaker)
   - Añadir límites máximos por transacción
   - Incluir mecanismos de recuperación ante errores
4. **Optimización de gas:** Revisar estructuras de datos y lógica para reducir costos de transacción, especialmente importante si se migra a redes públicas.
5. **Interfaz mejorada:**
   - Simplificar el proceso de configuración de MetaMask
   - Añadir visualizaciones gráficas de distribución de fondos
   - Implementar notificaciones en tiempo real de eventos blockchain
6. **Testing exhaustivo:** Desarrollar suite completa de tests unitarios y de integración utilizando Hardhat o Foundry.
7. **Auditoría de seguridad:** Antes de manejar fondos reales, realizar auditoría profesional del contrato.

4.5. Reflexión final

Este proyecto ilustra tanto el potencial como los desafíos de aplicar blockchain a problemas del mundo real. La tecnología proporciona garantías únicas de transparencia e inmutabilidad que son valiosas para organizaciones que manejan fondos públicos. Sin embargo, la complejidad técnica y las limitaciones de la inmutabilidad requieren un balance cuidadoso entre ambición y practicidad.

El sistema implementado, aunque simplificado, demuestra que es posible construir aplicaciones blockchain funcionales que aporten valor real. La experiencia adquirida en este proyecto proporciona una base sólida para comprender los trade-offs inherentes a esta tecnología y tomar decisiones de diseño informadas en futuros desarrollos.

Referencias