Universidade Vigo

Proyecto: Contrato Inteligente de Crowdfunding en Ethereum

Práctica 1 sobre desarrollo y pruebas de Smart Contracts

Borja Salgado Sotelo

Índice

1. Análisis y Definición del Escenario

1.1. Objetivo

El objetivo de este proyecto es desarrollar un **contrato inteligente** que permita gestionar una campaña de financiación colectiva (*crowdfunding*) de forma automatizada en una **blockchain pública**, en este caso **Ethereum**.

Este contrato permite que varias personas contribuyan con fondos a un proyecto común, asegurando que las condiciones se cumplan de manera transparente y sin la intervención de intermediarios. Si la meta de recaudación se alcanza antes de la fecha límite, el creador del proyecto puede retirar los fondos; de lo contrario, los participantes pueden recuperar sus aportes.

1.2. Justificación del uso de la blockchain

La elección de Ethereum como tecnología de base se debe a sus características fundamentales:

- **Transparencia:** todas las transacciones y operaciones quedan registradas de forma pública e inmutable.
- **Seguridad:** los contratos inteligentes se ejecutan automáticamente, reduciendo el riesgo de fraude o manipulación.
- **Descentralización:** elimina la necesidad de intermediarios, ya que las reglas de negocio se programan directamente en el contrato.
- **Automatización:** las condiciones del contrato se cumplen de manera automática según las reglas establecidas en el código.

1.3. Escenario planteado

En este proyecto se plantea una situación donde:

- Un **creador de proyecto** despliega el contrato, estableciendo una meta económica (`targetAmount`) y una fecha límite (`deadline`).
- Los **contribuidores** pueden enviar fondos en *Ether (ETH)* mediante la función `contribute()` mientras la campaña esté activa.
- Si la meta se alcanza antes de la fecha límite, el creador puede retirar los fondos con `withdrawFunds()`.
- Si no se alcanza, los contribuyentes pueden recuperar su dinero utilizando `claimRefund()`.

De esta forma, el contrato garantiza la **seguridad de los aportes**, la **transparencia del proceso** y la **ejecución automática de las reglas** sin necesidad de confianza entre las partes.

1.4. Relevancia del proyecto

Este tipo de contrato representa un caso práctico de aplicación de la tecnología blockchain en el ámbito financiero, demostrando cómo los **smart contracts** pueden sustituir procesos tradicionales de recaudación o gestión de fondos mediante lógica automática y trazabilidad garantizada.

2. Diseño

2.1. Lógica de negocio

El contrato inteligente **Crowdfunding** se diseña con el objetivo de gestionar de forma automatizada una campaña de financiación colectiva.

La lógica de negocio se basa en tres procesos principales:

1. **Contribución:** los usuarios pueden aportar fondos en Ether durante el periodo activo de la campaña.
2. **Retiro de fondos:** el creador del proyecto puede retirar los fondos únicamente si se alcanza la meta establecida antes de la fecha límite.
3. **Reembolso:** si la meta no se cumple al finalizar el plazo, los contribuyentes pueden solicitar la devolución de su aporte.

Cada uno de estos procesos se encuentra controlado por funciones específicas dentro del contrato, que establecen condiciones claras para evitar comportamientos indebidos.

—

2.2. Estructura de datos

El contrato contiene una serie de variables globales que almacenan la información principal de la campaña, así como un `mapping` que registra los aportes individuales de los usuarios.

Variable	**Tipo**	**Descripción**
`creator`	`address`	Dirección del creador del proyecto.
`targetAmount`	`uint256`	Monto objetivo de recaudación (en wei).
`deadline`	`uint256`	Fecha límite (timestamp UNIX).
`totalRaised`	`uint256`	Total recaudado hasta el momento.
`fundsWithdrawn`	`bool`	Indica si el creador ya retiró los fondos.
`contributions`	`mapping(address =>uint256)`	Registro de las aportaciones individuales.

Cuadro 1: Variables principales del contrato *Crowdfunding*.

—

2.3. Funciones principales

Las funciones del contrato permiten ejecutar las operaciones básicas del sistema de forma segura y descentralizada. En la Tabla 2 se muestran las principales funciones y sus restricciones.

—

Función	Descripción	Restricciones
`constructor`	Inicializa la meta y la fecha límite.	La fecha debe ser futura.
`contribute()`	Permite aportar fondos en ETH.	Solo antes del deadline.
`withdrawFunds()`	Retira los fondos si la meta fue alcanzada.	Solo el creador.
`claimRefund()`	Devuelve los fondos a los contribuyentes.	Si no se alcanzó la meta.
`goalReached()`	Indica si la meta fue alcanzada.	Sin restricciones.
`timeLeft()`	Muestra el tiempo restante de campaña.	Sin restricciones.

Cuadro 2: Funciones principales del contrato.

2.4. Reglas de operación

El comportamiento del sistema se rige por las siguientes reglas lógicas y de negocio:

- No se pueden realizar aportes una vez superada la fecha límite.
- Solo el creador del proyecto puede retirar los fondos recaudados.
- El retiro solo es posible si la meta ha sido alcanzada.
- Los reembolsos solo se habilitan si no se cumple el objetivo.
- Cada usuario puede reclamar su reembolso una única vez.

—

2.5. Usuarios del sistema

El contrato define dos tipos de usuarios principales:

- **Creador del Proyecto:** despliega el contrato, establece la meta y puede retirar los fondos una vez alcanzada.
- **Contribuidor:** realiza aportes en Ether y puede solicitar un reembolso si la campaña no cumple su objetivo.

—

2.6. Diagrama de casos de uso

El diseño del sistema puede representarse mediante el siguiente diagrama de casos de uso, donde se muestran las interacciones entre los actores (creador y contribuidor) y las funciones disponibles del contrato.

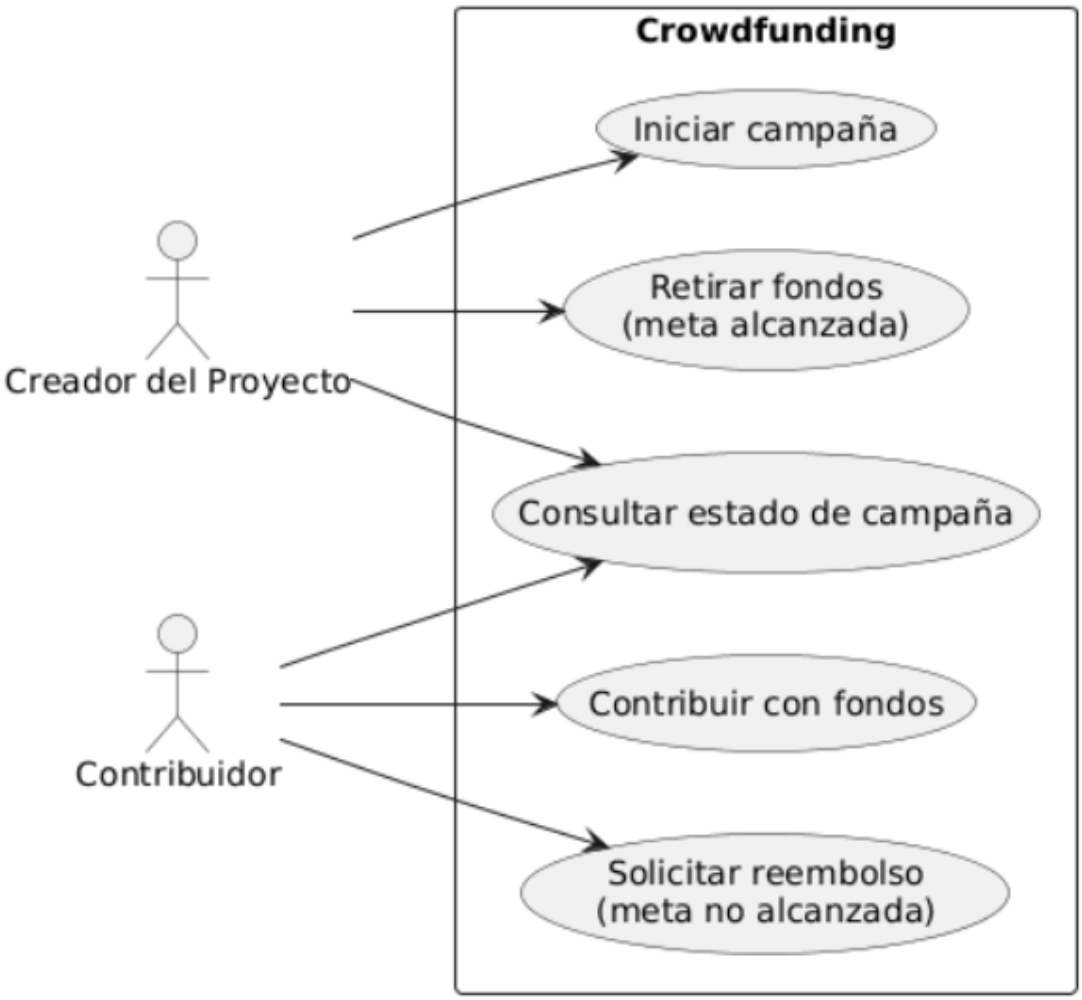

Figura 1: Diagrama de casos de uso del contrato inteligente *Crowdfunding*.

—

2.7. Resumen del diseño

El contrato *Crowdfunding* combina una estructura de datos sencilla con funciones claramente delimitadas y condiciones de ejecución seguras. El uso de variables públicas y funciones de lectura facilita la transparencia, mientras que las funciones de transferencia de fondos garantizan que solo se realicen bajo las condiciones previstas.

3. Implementación

3.1. Entorno de desarrollo

La implementación del contrato inteligente se realizó utilizando las siguientes herramientas y configuraciones:

- **Lenguaje:** Solidity versión 0.8.20.
- **IDE:** Remix IDE, entorno de desarrollo web especializado en contratos inteligentes.
- **Red de pruebas:** Remix VM (Prague), que emula una blockchain local sin necesidad de conexión a Internet.
- **Compilador:** solc-js (Solidity Compiler for JavaScript).
- **Gas limit:** 8,000,000.

El uso de Remix facilita la escritura, despliegue y prueba del contrato de forma visual e inmediata, permitiendo la simulación de diferentes cuentas y escenarios de interacción con el contrato.

—

3.2. Descripción general del contrato

El contrato `Crowdfunding` se compone de:

- Variables que almacenan la información clave de la campaña (meta, fecha límite, recaudación total, estado de los fondos, etc.).
- Funciones que controlan las contribuciones, los retiros y los reembolsos.
- Reglas de verificación que garantizan la seguridad de los fondos y evitan operaciones indebidas.

El contrato está diseñado para ser sencillo y funcional, mostrando la potencia de la automatización en Ethereum mediante reglas inmutables.

—

3.3. Código fuente en Solidity

A continuación, se presenta el código completo del contrato inteligente, implementado en el lenguaje Solidity:

```
pragma solidity ^0.8.20;

contract Crowdfunding {
    address public creator;
    uint256 public targetAmount;
    uint256 public deadline;
    uint256 public totalRaised;
    bool public fundsWithdrawn;
    mapping(address => uint256) public contributions;

    constructor(uint256 _targetAmount, uint256 _deadlineTimestamp) {
        require(_targetAmount > 0, "Target must be > 0");
        require(_deadlineTimestamp > block.timestamp, "Deadline must be in the future");
        creator = msg.sender;
        targetAmount = _targetAmount;
        deadline = _deadlineTimestamp;
    }

    function contribute() external payable {
        require(block.timestamp < deadline, "Campaign finished");
        require(msg.value > 0, "Must send ETH");
        contributions[msg.sender] += msg.value;
        totalRaised += msg.value;
    }

    function withdrawFunds() external {
        require(msg.sender == creator, "Only creator");
        require(block.timestamp >= deadline, "Too early");
        require(totalRaised >= targetAmount, "Goal not reached");
        require(!fundsWithdrawn, "Already withdrawn");

        fundsWithdrawn = true;
        payable(creator).transfer(address(this).balance);
    }

    function claimRefund() external {
        require(block.timestamp >= deadline, "Too early");
        require(totalRaised < targetAmount, "Goal reached");
        uint256 amount = contributions[msg.sender];
        require(amount > 0, "Nothing to refund");
        contributions[msg.sender] = 0;
        payable(msg.sender).transfer(amount);
    }

    function goalReached() external view returns (bool) {
        return totalRaised >= targetAmount;
    }

    function timeLeft() external view returns (uint256) {
        if (block.timestamp >= deadline) return 0;
        return deadline - block.timestamp;
    }
}
```

Listing 1: Contrato inteligente Crowdfunding en Solidity.

—

3.4. Consideraciones de seguridad

El contrato fue diseñado siguiendo las buenas prácticas recomendadas por la comunidad de desarrolladores de Ethereum:

- Uso de **require()** para verificar las condiciones previas antes de ejecutar operaciones críticas.

- Control de acceso mediante `msg.sender`, restringiendo funciones sensibles al creador del contrato.
- Prevención de reentradas al establecer el valor de `contributions[msg.sender]` en cero antes de realizar transferencias.
- Transferencias seguras utilizando la función `transfer()` hacia direcciones externas.

Estas medidas aseguran que el contrato sea resistente a errores comunes y que los fondos se manejen de forma segura durante todo el ciclo de vida de la campaña.

—

3.5. Despliegue del contrato

Durante el proceso de despliegue, se establecieron los siguientes parámetros iniciales:

- `_targetAmount`: `1000000000000000000` wei (equivalente a 1 Ether).
- `_deadlineTimestamp`: `1767139200` (correspondiente al 31 de diciembre de 2025).

El contrato fue desplegado correctamente sin enviar valor inicial (campo *Value* = 0), dado que el constructor no es *payable*. Una vez desplegado, las cuentas secundarias realizaron contribuciones y el creador pudo retirar los fondos al alcanzar la meta, confirmando el correcto funcionamiento del contrato.

4. Pruebas

4.1. Objetivo de las pruebas

El propósito de esta fase es verificar el correcto funcionamiento del contrato inteligente `Crowdfunding` bajo diferentes condiciones, asegurando que:

- Las reglas de negocio se cumplan tal como fueron definidas.
- Las operaciones financieras se ejecuten correctamente (aportaciones, retiros y reembolsos).
- No existan errores de ejecución o comportamientos indeseados.

Todas las pruebas se realizaron en el entorno **Remix IDE** utilizando la red virtual **Remix VM (Prague)**.

—

4.2. Configuración del entorno de prueba

- **Compilador:** Solidity 0.8.20.
- **Gas limit:** 8,000,000 unidades.
- **Cuentas utilizadas:** 3 direcciones simuladas en Remix.
- **Parámetros iniciales:**
  - _targetAmount = `1000000000000000000` wei (equivalente a 1 ETH).
  - _deadlineTimestamp = `1767139200` (31 de diciembre de 2025).

—

4.3. Escenario 1: Meta alcanzada exitosamente

- Dos contribuyentes aportan 0.5 ETH cada uno mediante la función `contribute()`.
- Se verifica que `totalRaised` = 1 ETH.
- Al llegar la fecha límite, el creador ejecuta `withdrawFunds()`.

Resultado esperado:

- La transacción se ejecuta correctamente.
- El balance del contrato pasa a 0.
- El balance del creador aumenta en 1 ETH.

Estado final:

- `fundsWithdrawn = true`
- `totalRaised = 1 ETH`
- Recaudación completada con éxito.

—

4.4. Escenario 2: Meta no alcanzada

- La meta establecida es de 2 ETH.
- Solo un contribuyente aporta 0.5 ETH.
- Expira el plazo de la campaña.

Resultado esperado:

- El creador no puede retirar fondos (falla la función `withdrawFunds()`).
- Los contribuyentes pueden ejecutar `claimRefund()` y recuperar sus aportes.

Estado final:

- `fundsWithdrawn = false`
- `totalRaised = 0.5 ETH`
- Todos los participantes reciben su reembolso completo.

—

4.5. Escenario 3: Intento de retiro anticipado

- El creador intenta ejecutar `withdrawFunds()` antes de alcanzar la fecha límite.

Resultado esperado:

- La transacción se revierte.
- Remix muestra el error: `revert: Too early`.
- No hay movimiento de fondos.

—

4.6. Escenario 4: Prevención de reembolsos indebidos

- Tras alcanzar la meta, un contribuyente intenta ejecutar `claimRefund()`.

Resultado esperado:

- La llamada se revierte.
- El contrato muestra el mensaje: `Goal reached`.
- No se realiza ningún reembolso.

—

4.7. Resultados generales

En todos los casos, las funciones respondieron conforme a las restricciones establecidas:

- Se bloquearon correctamente las operaciones fuera del plazo.
- Los montos fueron transferidos o reembolsados según las condiciones.
- Las verificaciones `require()` impidieron la ejecución de funciones indebidas.

—

4.8. Conclusión de las pruebas

Las pruebas realizadas confirman que el contrato *Crowdfunding* cumple correctamente su lógica de negocio y maneja de forma segura los fondos aportados. El comportamiento del contrato fue consistente en los diferentes escenarios, validando la robustez de sus condiciones de ejecución y la correcta gestión de los recursos económicos.