Informe de Contrato Inteligente: Escrow por Hitos en Ethereum

Autor: Adrián Rey Monteagudo

Fecha: Octubre 2025

1. Análisis y definición del escenario

1.1. Contexto

En el entorno actual de economía digital, la contratación de servicios freelance se ha vuelto común. Sin embargo, existe desconfianza entre las partes: el cliente teme pagar por trabajos no entregados o de mala calidad, mientras que el freelancer teme trabajar sin recibir el pago acordado. Los contratos inteligentes ofrecen una solución al automatizar y garantizar el cumplimiento de acuerdos sin intermediarios.

1.2. Requerimiento de una blockchain pública

Se selecciona Ethereum como blockchain pública debido a su soporte nativo para contratos inteligentes, uso de Ether como moneda, inmutabilidad y descentralización. El sistema requiere una blockchain pública porque la custodia y liberación de fondos deben ser verificables y automáticas, sin depender de una entidad central.

1.3. Objetivo

Desarrollar un contrato inteligente de Escrow por hitos (milestones) que automatice los pagos entre un cliente y un freelancer, con la intervención opcional de un árbitro en caso de disputa.

2. Diseño

2.1. Caso de uso principal

El cliente crea un contrato que especifica los hitos del trabajo y sus importes. Deposita el total en el contrato inteligente. El freelancer entrega cada hito, el cliente lo revisa y lo acepta para liberar el pago. Si existe desacuerdo, se puede abrir una disputa que un árbitro resolverá.

2.2. Lógica de negocio

1. Creación del contrato de trabajo
2. Financiamiento del contrato
3. Entrega del hito
4. Aprobación o disputa
5. Resolución de disputa
6. Retiro de fondos

2.3. Datos y estructuras

Se emplean estructuras de datos Job y Milestone que almacenan información sobre el cliente, freelancer, árbitro, montos y estado.

2.4. Funciones principales

createJob(), fundJob(), submitMilestone(), acceptMilestone(), raiseDispute(), resolveDispute(), withdraw(), getJobBasic(), getMilestone()

2.5. Usuarios del sistema

Cliente: Crea y financia trabajos.
Freelancer: Entrega hitos.
Árbitro: Resuelve disputas.
Contrato inteligente: Administra la lógica y los fondos.

3. Implementación

El contrato se desarrolló en Solidity 0.8.20 utilizando Remix IDE. Se emplearon las librerías OpenZeppelin ReentrancyGuard y Ownable para protección y control de acceso. Se implementó el patrón Pull Payments para asegurar la retirada de fondos de forma segura.

4. Pruebas

Las pruebas se realizaron en Remix IDE con el entorno JavaScript VM. Se utilizaron tres cuentas: cliente, freelancer y árbitro. Se verificó el flujo completo de creación, financiamiento, entrega, aceptación y resolución de disputas. Todos los eventos fueron emitidos correctamente y las funciones cumplieron su propósito.

5. Conclusiones

El contrato inteligente cumple con los objetivos propuestos: automatizar pagos condicionales entre partes sin intermediarios. Se garantiza la seguridad mediante blockchain pública, trazabilidad y protección contra reentrancia. El proyecto puede ampliarse para soportar tokens ERC-20, penalizaciones o almacenamiento de entregables en IPFS.

6. Referencias

- Ethereum Foundation. Ethereum Smart Contract Best Practices.
- OpenZeppelin Contracts.
- Remix IDE Documentation.
- Solidity Documentation v0.8.20.