Práctica 2: Ejercicio 2

Borja Salgado Sotelo

Noviembre 2025

Documentación: Crowdfunding descentralizado con IPFS y Ethereum

Objetivo

El objetivo de este caso de uso es demostrar la integración práctica entre una red de almacenamiento descentralizado (IPFS) y una blockchain pública (Ethereum), mediante el desarrollo de una aplicación descentralizada (*dApp*) que permite registrar proyectos de financiación colectiva (*crowdfunding*) sin depender de servidores centralizados.

Cada proyecto puede asociar un archivo descriptivo (por ejemplo, un documento, imagen o vídeo) que se almacena en la red IPFS, obteniendo un identificador único (CID). Posteriormente, este hash se registra en un contrato inteligente desplegado en la blockchain de Ethereum, garantizando así la trazabilidad, inmutabilidad y transparencia de la información.

—

Descripción del caso de uso

El sistema diseñado permite que un usuario:

1. Seleccione un archivo desde su ordenador.
2. Suba dicho archivo a la red IPFS a través de un nodo local ejecutado mediante Docker.
3. Obtenga el hash único (CID) del archivo almacenado en IPFS.
4. Registre ese hash en la blockchain Ethereum mediante la interacción con un contrato inteligente a través de MetaMask.

El caso de uso simula el registro descentralizado de campañas de financiación colectiva, donde cada archivo subido representa un proyecto o propuesta. De esta manera, se demuestra cómo una blockchain puede utilizarse para garantizar la autenticidad de los datos almacenados fuera de la cadena, en una red distribuida como IPFS.

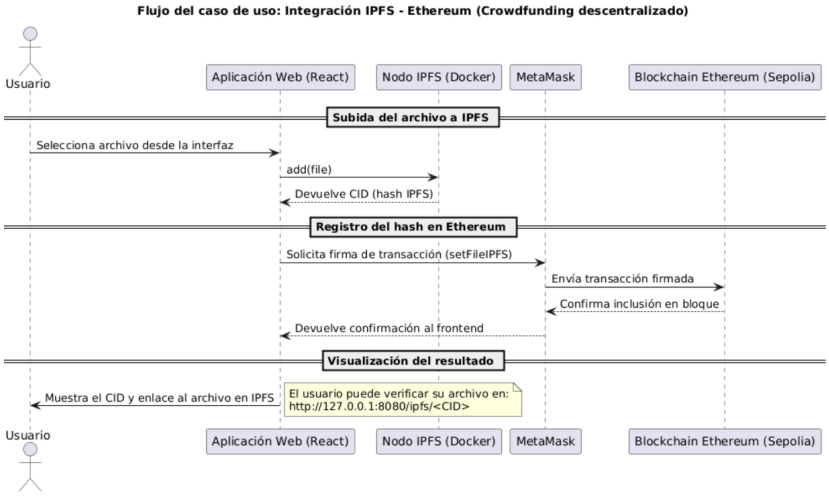

Figure 1: Flujo del caso de uso: interacción entre el usuario, IPFS y el contrato en Ethereum.

La Figura 1 representa el flujo de interacción del caso de uso, desde la acción del usuario hasta el registro final del hash en la blockchain Ethereum. El proceso comienza con la selección del archivo y concluye con la confirmación de la transacción a través de MetaMask, que garantiza la integridad de los datos registrados.

—

Arquitectura del sistema

La arquitectura del sistema se compone de los siguientes elementos:

- **Ethereum (Sepolia Testnet):** red pública utilizada para desplegar el contrato inteligente `IpfsStorage.sol`, que guarda los hashes IPFS asociados a cada usuario.
- **IPFS (Kubo):** sistema de almacenamiento descentralizado ejecutado en un contenedor Docker local, configurado para permitir conexiones CORS desde la aplicación web.
- **Smart Contract:** desarrollado en Solidity y desplegado desde Remix IDE. El contrato almacena los hashes IPFS de los archivos subidos por los usuarios:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract IpfsStorage {
```

```
    mapping(address => string) public userFiles;

    function setFileIPFS(string memory file) external {
        userFiles[msg.sender] = file;
    }
}
```

- **Frontend:** construido en React, con conexión a Ethereum mediante Ethers.js y al nodo IPFS mediante la librería `kubo-rpc-client`.
- **MetaMask:** utilizada como pasarela para firmar las transacciones que registran los hashes en Ethereum.

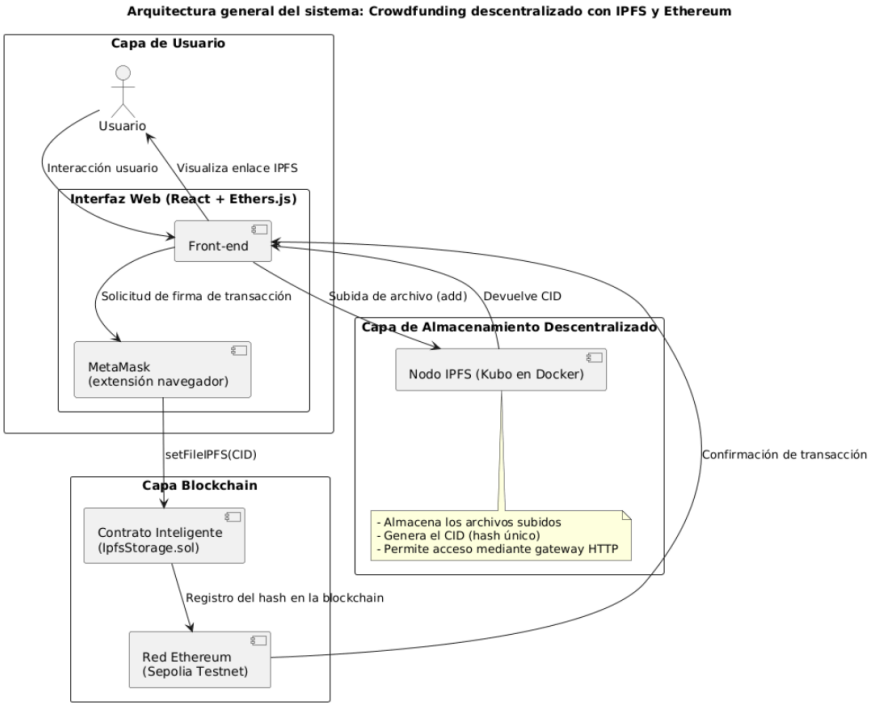

Figure 2: Arquitectura general del caso de uso: Crowdfunding descentralizado con IPFS y Ethereum.

La Figura 2 muestra la arquitectura general del sistema, compuesta por tres capas principales:

- **Capa de usuario (Frontend):** interfaz web desarrollada con React que permite seleccionar un archivo, subirlo a IPFS y registrar su hash en Ethereum a través de MetaMask.

- **Capa de almacenamiento descentralizado (IPFS):** nodo Kubo ejecutado en Docker que recibe los archivos, genera el identificador de contenido (CID) y los replica en la red.
- **Capa blockchain (Ethereum):** red Sepolia que ejecuta el contrato inteligente `IpfsStorage`, encargado de almacenar los hashes IPFS vinculados a cada dirección de usuario.

—

Funcionamiento

1. El usuario selecciona un archivo en la interfaz y presiona el botón `Upload`.
2. El archivo se envía al nodo IPFS local, ejecutado mediante Docker con el siguiente comando:

```
docker run -d --name ipfs_host \
-v $PWD:/export \
-v $PWD:/data/ipfs \
-p 4001:4001 \
-p 4001:4001/udp \
-p 127.0.0.1:8080:8080 \
-p 127.0.0.1:5001:5001 \
ipfs/kubo
```

3. Una vez subido, el sistema obtiene el **CID (Content Identifier)** del archivo.
4. El hash CID se muestra en la interfaz y se registra en la blockchain Ethereum mediante el contrato inteligente `IpfsStorage`.
5. Finalmente, el usuario puede verificar su archivo accediendo al enlace generado:

`http://127.0.0.1:8080/ipfs/<CID>`

—

Tecnologías utilizadas

- **Ethereum Sepolia Testnet:** red de pruebas usada para desplegar el contrato y realizar transacciones con Ether de prueba.
- **Remix IDE:** entorno para compilar, desplegar y verificar contratos inteligentes en la red Ethereum.

- **IPFS (Kubo):** protocolo de almacenamiento distribuido basado en el direccionamiento por contenido.
- **Ethers.js (v5.7.0):** biblioteca utilizada para interactuar con la blockchain Ethereum desde el navegador.
- **React y Node.js:** base del frontend que permite subir archivos y comunicarse con IPFS y MetaMask.
- **MetaMask:** extensión que gestiona las claves privadas y firma las transacciones del contrato inteligente.

—

Problemas e incidencias

Durante la realización del ejercicio se presentaron varios problemas técnicos:

- **Configuración CORS:** el nodo IPFS bloqueaba las peticiones externas hasta habilitar los orígenes con:

```
ipfs config --json API.HTTPHeaders.Access-Control-Allow-Origin \
'["http://127.0.0.1:8080", "http://localhost:3000", \
"http://127.0.0.1:5001", "https://webui.ipfs.io"]'
```

- **Conflicto de nombres en IPFS:** se produjo el error *"directory already has entry by that name"* al intentar copiar archivos ya existentes al sistema de ficheros IPFS. Se solucionó eliminando la instrucción redundante:

```
await client.files.cp(`/ipfs/${result.cid}`, `/${result.cid}`);
```

- **Error de red en Ethers.js:** al detectar cambios entre la red de MetaMask (Sepolia) y la red del proveedor. Se resolvió modificando la inicialización del proveedor a:

```
const provider = new ethers.providers.Web3Provider(window.ethereum, "any");
```

—

Lecciones aprendidas

A través de esta práctica se comprendió el proceso completo de interacción entre un sistema de almacenamiento distribuido y una blockchain pública:

- Cómo desplegar un contrato inteligente en Ethereum y vincularlo a una aplicación web.

- Cómo utilizar IPFS para almacenar archivos de forma descentralizada y obtener su hash CID.
- Cómo firmar y enviar transacciones desde MetaMask para registrar información en la cadena de bloques.
- La importancia de la correcta configuración de CORS y de la sincronización de redes entre Ethers.js y MetaMask.

El resultado final es una aplicación funcional que ejemplifica un caso de uso real de las Tecnologías de Registro Distribuido (DLT), aplicadas al contexto del *crowdfunding descentralizado*.

—

Conclusión

El caso de uso desarrollado demuestra el potencial de combinar blockchain e IPFS para crear aplicaciones descentralizadas en las que la información se mantiene accesible, segura e inmutable. El almacenamiento distribuido permite eliminar la dependencia de servidores centrales, mientras que la blockchain garantiza la integridad y trazabilidad de los datos. Con esta práctica se alcanzó plenamente el objetivo de integrar tecnologías DLT en un escenario funcional y comprensible.