MÁSTER INTERUNIVERSITARIO EN CIBERSEGURIDAD

Universidade Vigo

Tecnoloxías de rexistro distribuído e Blockchain

Práctica 2: Almacenamiento y distribución con IPFS

Por: Pablo Pérez Paramos y Alexandre Moinelo Rodríguez

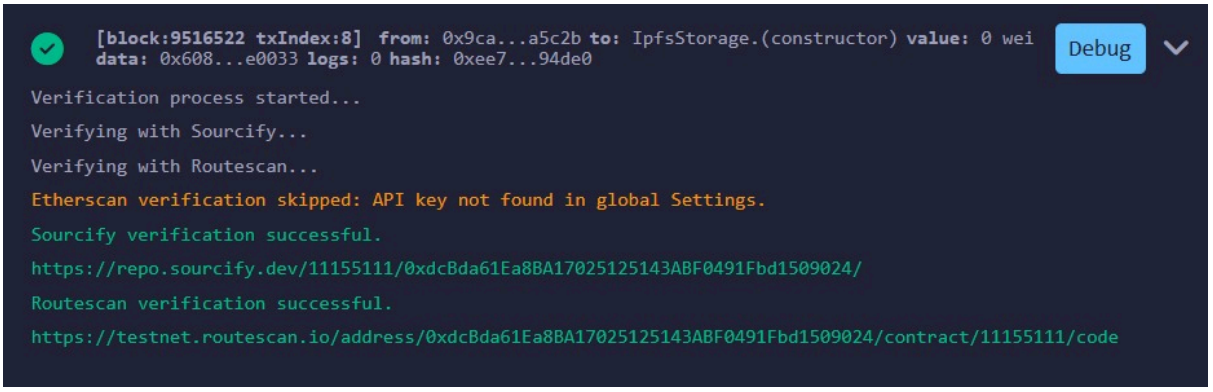
Índice de contenidos

Ejercicio 1: Realiza los pasos indicados en la sección 3: sube un fichero a IPFS y comprueba que las transacciones se han realizado correctamente.

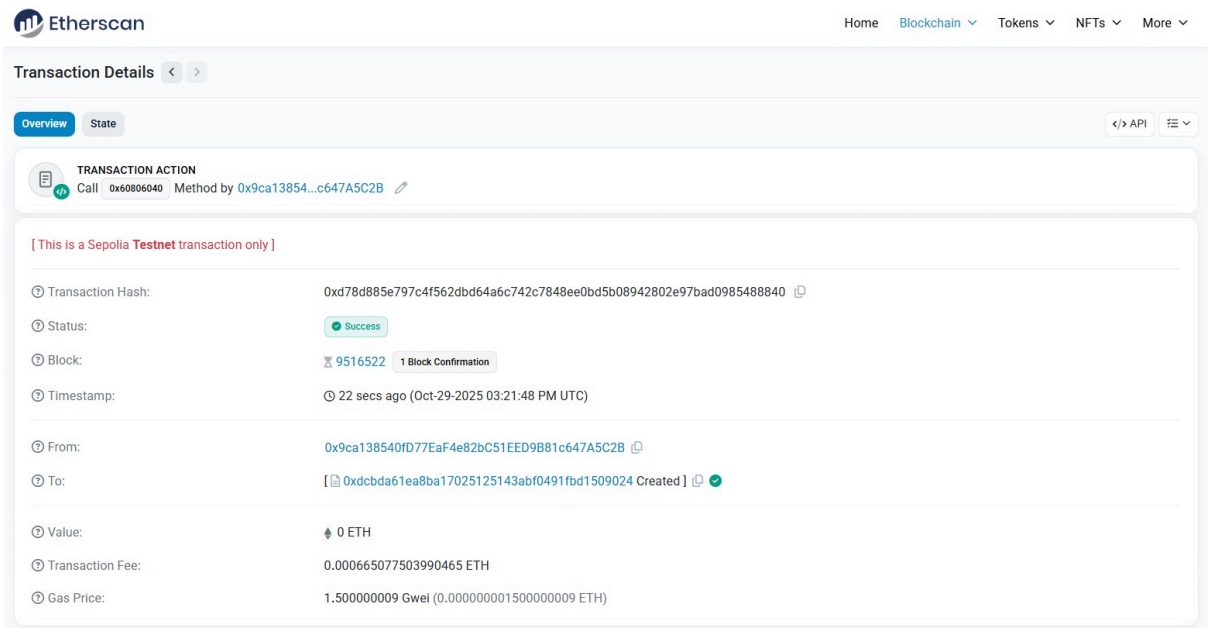
Vamos a describir un poco paso a paso lo realizado en este ejercicio para que quede constancia de que fue realizado con unas cuantas capturas realizadas durante el proceso, pero sin mucho detalle porque consideramos que ya viene muy bien explicado en la práctica.

En primer lugar, creamos el contrato en remix como se nos fue indicado, además de compilarlo y guardarlo con el nombre *IpfsStorage.sol* y también guardamos el *IpfsStorage.json* generado por el remix. Desplegamos el Smart Contract conectando nuestra cuenta de Metamask en la red Sepolia, lo cual es muy importante, porque nos costó un poco al principio, que no conseguiamos conectarnos a nuestra cuenta en la red, lo mostramos en la primera imagen.

Al confirmar la implementación del contrato verificamos que se implementa correctamente:



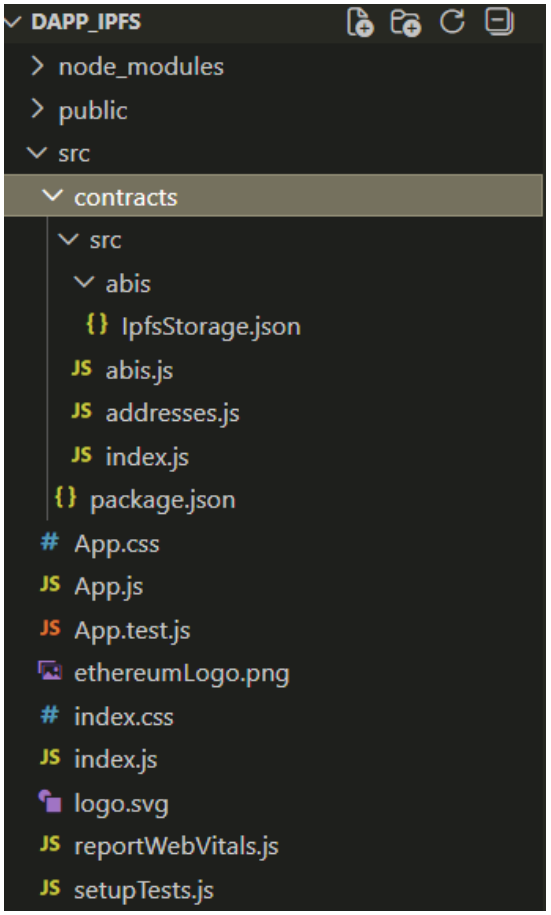
Y buscamos en código hash de la transacción para mostrar sus detalles en Etherscan:



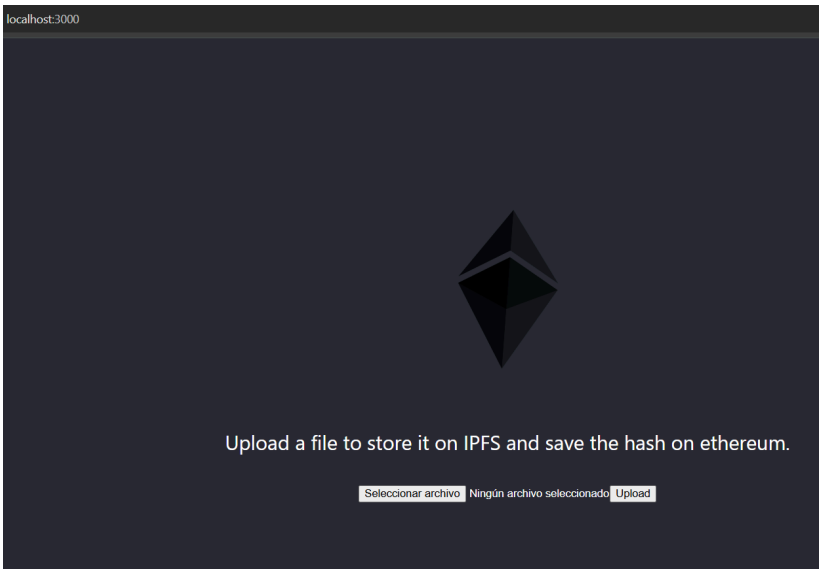
Copiamos y guardamos la dirección del smart contract como fue indicado en la práctica, *0xdcBda61Ea8BA17025125143ABF0491Fbd1509024*.

Como segundo paso, instalamos docker, ya que no lo teniamos. Para lanzarlo tuvimos algún probremilla pero nada, con la ayuda de ChatGPT la arreglamos sin mayor fallo.

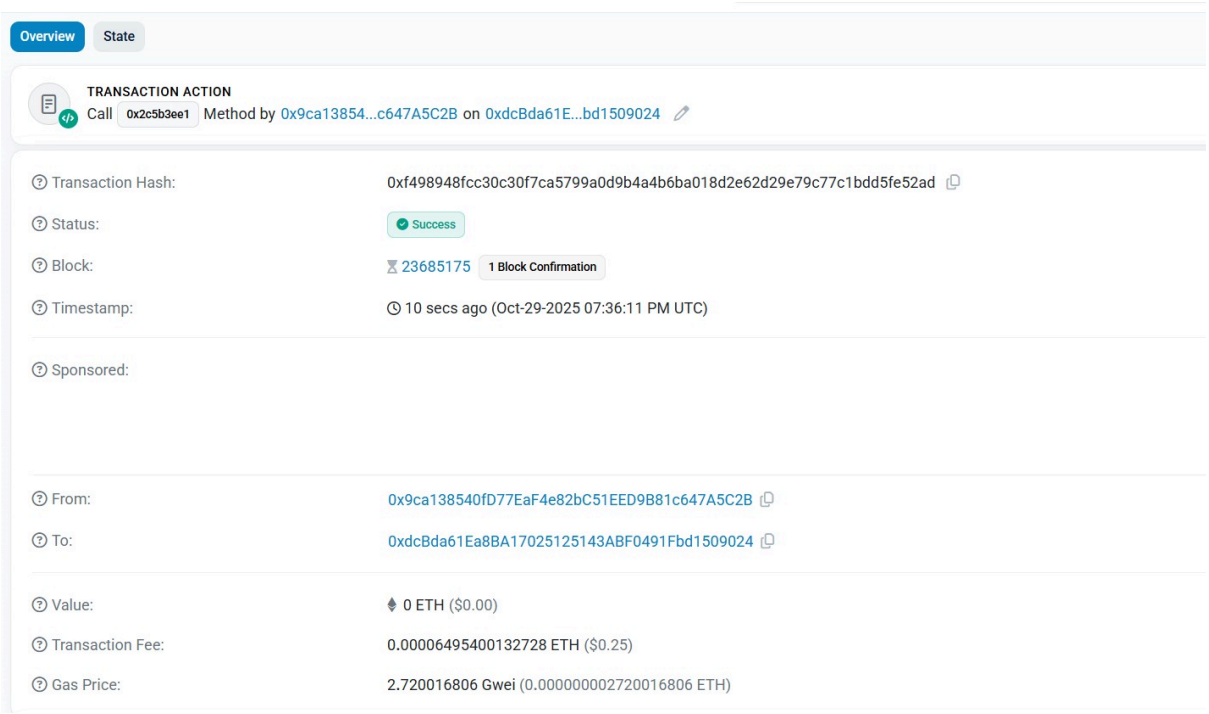
Por último, instalamos el Node.js, la versión recomendada. Creamos la estructura, dentro del directorio *dapp_ipfs* instalamos las dependencias y todas las carpetas que marca la práctica, de modo que el organigrama del proyecto quedaría tal que así:



Para terminar lanzamos el proyecto con *npm start*, y en localhost:3000 subimos el archivo.

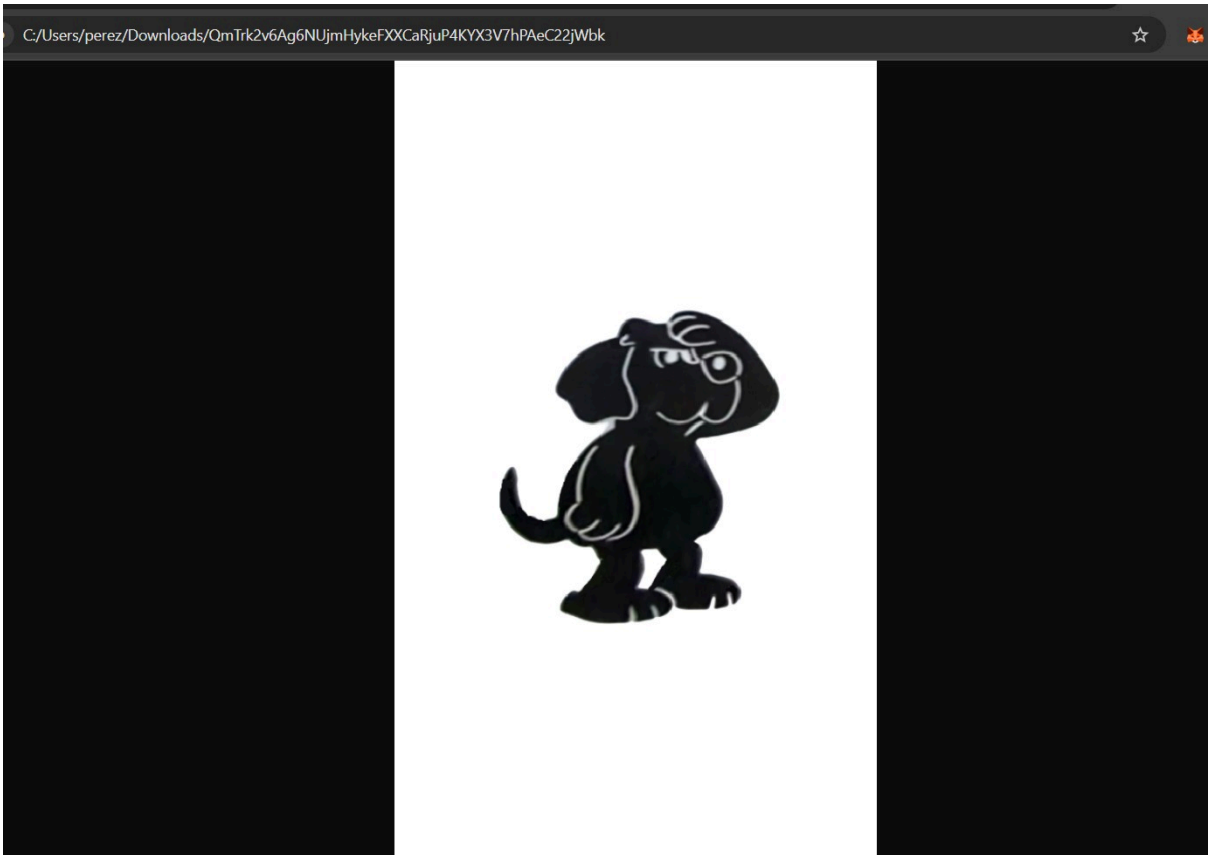


Subimos el archivo, lo que nos genera un hash, el cual buscamos en Etherscan para ver los detalles de la transacción. Podemos verificar que sale desde nuestra dirección de Sepolia y va hacia la dirección del smart contract y que la transacción tuvo un gasto de 0.25$ por el gas gastado en el proceso.



En IPFS podemos verificar que se subió perfectamente el archivo e incluso descargar el archivo para comprobar que es correctamente el que hemos subido anteriormente cuando lanzamos la aplicación.

Y por último, al descargar desde el IPFS, en la imagen anterior el archivo subido, verificamos que es el archivo subido.



Ejercicio 2: Define un caso de uso similar al explicado en la sección 2 que encaje con la temática de tu proyecto e implementarlo partiendo del código del ejercicio 1. Documenta el caso de uso de forma similar a la sección 2.

Para la realización de este ejercicio vamos a usar el contrato que realizamos para la práctica 1, en el ejercicio 5, pero con alguna modificación porque creemos que así se ve más visual para este ejercicio, hemos cambiado que en el anterior el creador del contrato define los points ganados por el comprador, en este caso el creador recibe una imagen.

En este caso de uso se diseña e implementa una plataforma de donaciones basada en blockchain que permite a los usuarios apoyar económicamente a un determinado proyecto (por ejemplo, una ONG o un creador de contenido digital) y, a cambio, asociar una imagen almacenada en IPFS a su dirección en la blockchain.

Cada usuario puede subir una imagen (por ejemplo, un certificado digital de donación, una ilustración exclusiva, un "badge" de apoyo, etc.) a un sistema de almacenamiento descentralizado como IPFS. El hash/CID de esa imagen se almacena de forma inmutable en la blockchain mediante el contrato inteligente DonationPlatform.

El contrato se encarga de:

- Registrar, para cada dirección de usuario, el hash IPFS de su imagen (mapping(address => string) public userFiles).
- Gestionar las donaciones en ETH, distribuyendo automáticamente el importe entre:
- Una cuenta beneficiaria (recipient), que recibe un porcentaje configurable de la donación.
    - El propietario de la plataforma (owner), que recibe la comisión restante.
    - Emitir eventos que permiten a la aplicación cliente conocer cuándo se ha realizado una donación y qué imagen se ha asociado.

De esta forma, se consigue:

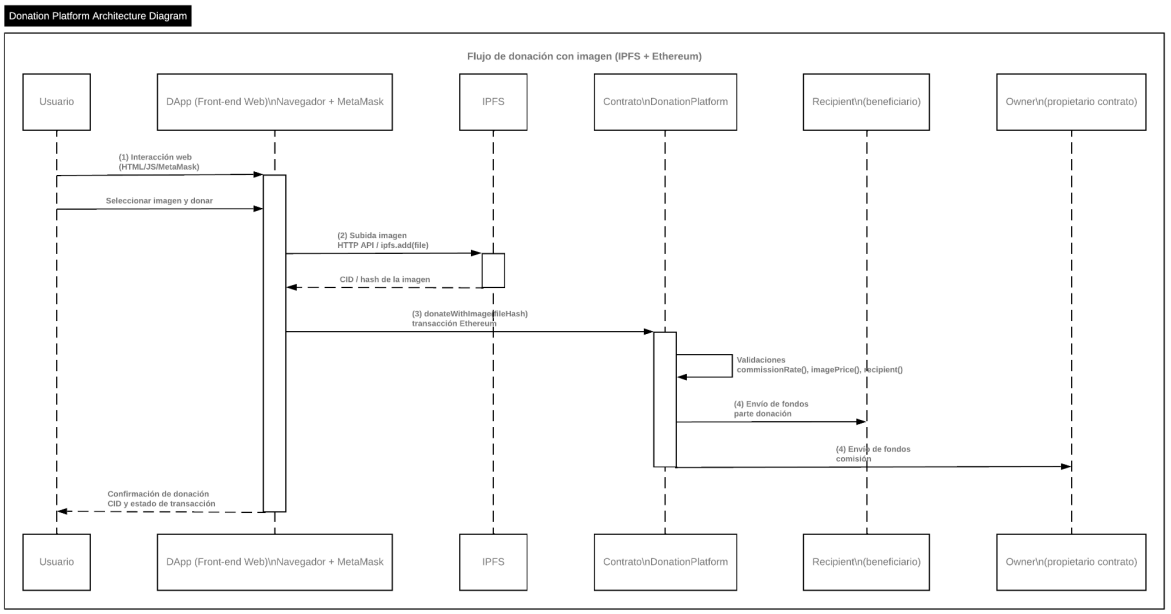
- **Transparencia** en el flujo de fondos (todas las donaciones se registran on-chain).
- **Trazabilidad** de los "recompensas" digitales asociadas a cada donante (imagen en IPFS vinculada a su dirección).
- **Descentralización** del almacenamiento de los ficheros, evitando depender de un único servidor centralizado.

Este caso de uso es aplicable, por ejemplo, a:

- Plataformas de micromecenazgo (“crowdfunding”) donde cada donante recibe un diploma digital.
- Coleccionables digitales simples (tipo “NFT light”) en los que solo se almacena el hash del recurso en IPFS.
- Sistemas de membresía donde la pertenencia se justifica mediante una imagen certificada almacenada en IPFS y vinculada a un pago.

2.1. Arquitectura de comunicaciones

La arquitectura propuesta combina tres elementos principales: usuarios donantes, blockchain Ethereum y almacenamiento descentralizado (IPFS), además de una aplicación cliente (DApp) que actúa como interfaz de usuario. La Figura 1 ilustra esta arquitectura.



Los componentes principales son:

- Usuarios / Donantes. Utilizan un navegador web o una aplicación cliente con un monedero compatible con Ethereum (por ejemplo, MetaMask) para:
  - Subir su imagen a IPFS (a través de la propia DApp o de un servicio externo).

- Enviar transacciones al contrato DonationPlatform para realizar donaciones y asociar la imagen (función donateWithImage).

- Aplicación cliente (DApp). Proporciona una interfaz web en la que el usuario puede:
  - Seleccionar el fichero a subir.
  - Obtener el hash/CID de IPFS.
  - Introducir la cantidad de ETH a donar.

- Se comunica con:
  - IPFS, para subir la imagen y obtener el hash.
  - La red Ethereum, utilizando una librería como Web3.js o Ethers.js, para invocar las funciones del contrato.

- Sistema de almacenamiento descentralizado (IPFS)
  - Almacena la imagen o recurso digital asociado a cada donante.
  - Devuelve un CID (Content Identifier), que se guarda en la blockchain.
  - Permite recuperar el contenido a través de distintas puertas de enlace (gateways) IPFS, manteniendo la disponibilidad y redundancia.

El flujo de comunicaciones puede resumirse así:

1. El usuario interactúa con la DApp desde su navegador.
2. La DApp sube la imagen a IPFS y obtiene el hash.
3. La DApp invoca al contrato DonationPlatform en Ethereum (por ejemplo, donateWithImage(fileHash)), firmando la transacción con el monedero del usuario.
4. La red Ethereum ejecuta el contrato, reparte los fondos y guarda el hash en userFiles.
5. Cualquier cliente puede posteriormente:
   - Leer userFiles[usuario] en la blockchain.
   - Usar ese hash para recuperar la imagen desde IPFS.

2.2. Implementación de la arquitectura propuesta

La plataforma que se ha desarrollado combina tres piezas: una página web (DApp), un contrato inteligente en Ethereum (DonationPlatform) y un sistema de ficheros distribuido (IPFS). El objetivo es que un usuario pueda apoyar económicamente una causa y, a cambio, obtener una imagen digital asociada a su dirección en la blockchain.

El uso típico empieza en el navegador. El usuario abre la DApp y se conecta con su monedero (MetaMask). En la interfaz encuentra una pequeña galería de imágenes: cada una tiene un título, un precio en ETH y una previsualización. Esas imágenes no están guardadas en un servidor clásico, sino en IPFS; la DApp simplemente muestra las URLs construidas a partir del CID que devuelve IPFS.

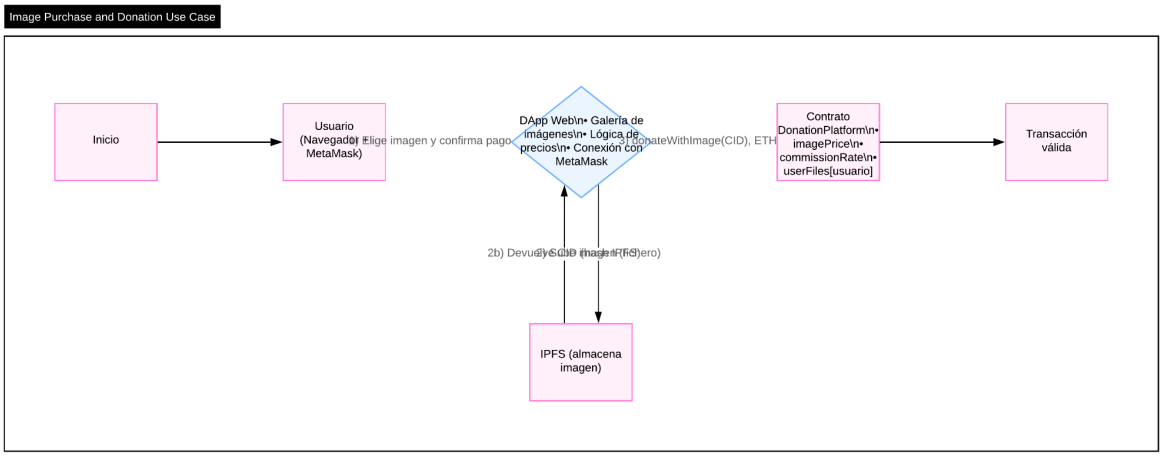
Cuando el usuario decide apoyar la causa, escoge una de las imágenes de la galería y pulsa “Comprar imagen”. La DApp le pide a MetaMask que firme una transacción. Esa transacción llama a la función donateWithImage del contrato DonationPlatform, pasando como parámetro el CID de la imagen y enviando como valor (msg.value) el precio que marca la DApp.

En el contrato, toda la lógica económica está encapsulada. Primero se comprueba que la cantidad enviada es, como mínimo, el precio mínimo on-chain definido por imagePrice. Si el pago no alcanza ese umbral, la operación se rechaza. Si es suficiente, el contrato registra el hash de la imagen en userFiles[msg.sender], de manera que a partir de ese momento esa dirección de Ethereum queda asociada a ese CID de IPFS de forma pública e inmutable.

A continuación, el contrato reparte el importe de la transacción. El porcentaje commissionRate determina qué parte se envía a la cuenta recipient (que puede representar a la ONG, artista o proyecto financiado) y qué parte se envía a la cuenta owner (propietario de la plataforma, que recibe una comisión). Ambas transferencias se realizan desde el propio contrato, y el reparto queda reflejado en el evento DonationReceived.

Mientras tanto, la DApp espera a que la transacción se confirme y muestra al usuario un resumen comprensible: cuánto ha pagado, qué parte se ha destinado a donación y qué parte corresponde a la comisión. Además, como el CID de la imagen ha quedado grabado en userFiles, la aplicación puede comprobar en cualquier momento qué imagen tiene actualmente asociada esa dirección de Ethereum, consultando directamente el contrato.

En conjunto, el caso de uso refleja un ciclo completo: selección de contenido, subida de la imagen a IPFS, compra mediante una transacción en Ethereum, registro del contenido adquirido en la blockchain y reparto automático de los fondos entre los participantes, garantizado por el contrato inteligente.



2.3. Funcionamiento del sistema

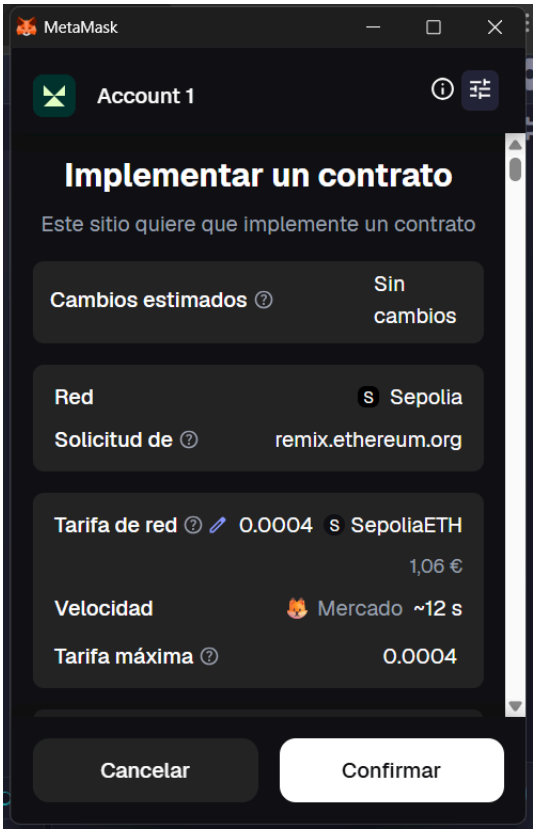
Así que, como smart contract usaremos *donate.sol* de la práctica anterior, pero un poco modificado. De tal forma que las variables son las siguientes:

```
contract DonationPlatform {

    address public owner;
    address public recipient;
    uint256 public commissionRate;   // porcentaje para recipient (0-100)

    // --- Imagen de cada usuario (hash/CID de IPFS) ---
    mapping(address => string) public userFiles;
```

Copiamos el .json generado en un archivo con el nombre *donate.json*. A continuación, pasamos a desplegar el contrato conectando nuestra cuenta de Metamask en la red de Sepolia que tiene la siguiente dirección: *0x9ca138540fD77EaF4e82bC51EED9B81c647A5C2B* (la cuenta de Pablo). Al desplegar el contrato indicamos que las donaciones tengan un 20% del precio de compra y que vayan a esta dirección (la cuenta de Alexandre): *0xd33d8fbd6cc5a8d3910882a9841d6a72127fa80c,* al desplegar de todo el contrato nos crea esta dirección, *0x67905c98cf7746fd09ec72706189052bAA0Ba898*, la cual guardaremos para usar más adelante. Añadimos una imagen de forma de demostración de que realmente lo hemos implementado el contrato.



Para ejecutarlo y enseñar cómo funciona el sistema hemos grabado un video el cual subiremos a Moovi.

Lecciones aprendidas

Durante el desarrollo de esta práctica he comprobado que trabajar con una DApp no es simplemente hacer una web normal: todo está dividido entre lo que ocurre en el front, lo que ocurre en IPFS y lo que se guarda en la blockchain. Una de las primeras cosas que entendí es que en Ethereum no conviene almacenar archivos completos, solo la información imprescindible. Por eso el contrato guarda únicamente el CID de la imagen, mientras que el archivo real va a IPFS.

Otra lección importante fue aprender a validar las cosas dos veces: una en la interfaz, para avisar al usuario, y otra en el contrato, para evitar que alguien intente saltarse las

normas. Por ejemplo, el precio mínimo (imagePrice) se comprueba tanto en el App.js como dentro de donateWithImage.

También me quedó claro por qué se usa el modificador nonReentrant. Aunque este proyecto es sencillo, mover ETH desde un contrato siempre requiere protegerse contra ataques de reentrancia y seguir el orden correcto de operaciones.

En la parte del front-end, trabajar con ethers.js me hizo prestar atención a la diferencia entre ETH y wei, y a cómo gestionar correctamente las transacciones: pedir la firma, esperar la confirmación, mostrar mensajes de estado, etc. Esto es importante para que la aplicación no dé la sensación de que se ha bloqueado mientras se mina la transacción.

Por último, usar IPFS me enseñó que los sistemas descentralizados no funcionan igual que un servidor tradicional: a veces tardan un poco en responder, y es necesario controlar esos tiempos y avisar al usuario. A cambio, la imagen queda accesible de forma descentralizada y el CID queda vinculado a la dirección del usuario de forma verificable.

En general, la práctica me ayudó a entender cómo se combinan estos tres elementos (IPFS, contrato y DApp) para construir una aplicación completa y funcional.