Blockchain
Labwork 2 : Almacenamiento y distribución con IPFS

Adrián Robles Camporro, David Olivares Muñoz

December, 2023

Contents

1 Análisis y explicación del uso de IPFS

IPFS, o InterPlanetary File System, es un protocolo y red diseñados para crear un sistema de archivos peer-to-peer distribuido. Se utiliza para descentralizar el almacenamiento y compartir archivos de manera eficiente. En el contexto de una subasta de arte, IPFS puede ser una opción interesante para almacenar datos relacionados con las obras de arte. Aquí hay una explicación del uso de IPFS en este caso:

- Descentralización del almacenamiento:

  En lugar de depender de un servidor centralizado para almacenar la información de las obras de arte, IPFS utiliza una red distribuida de nodos. Cada nodo en la red almacena una copia de los datos, lo que significa que no hay un único punto de falla.

- Inmutabilidad y verificación de integridad:

  IPFS utiliza hashes criptográficos para identificar los contenidos de los archivos. Cada archivo se representa por un hash único que se calcula a partir de su contenido. Esto proporciona integridad y garantiza que los datos no se alteren.

- Acceso descentralizado:

  Los archivos en IPFS se pueden acceder a través de su hash, lo que significa que no es necesario depender de una ubicación centralizada para obtener la información. Esto es especialmente útil en el contexto de una subasta de arte, donde se pueden tener participantes de diversas ubicaciones.

- Eficiencia en la transferencia de datos:

  IPFS utiliza la tecnología de red peer-to-peer para transferir datos de manera eficiente. Cuando alguien solicita un archivo, los nodos cercanos a esa persona pueden proporcionar el archivo, lo que reduce la carga en los servidores centrales y mejora la velocidad de acceso.

- Incentivos económicos:

  IPFS está diseñado para proporcionar incentivos económicos a los nodos que participan en la red. Esto puede alentar a más personas a contribuir con almacenamiento y ancho de banda, creando así una red más robusta y resistente.

- Reducir costos de almacenamiento:

  Al aprovechar la red distribuida de IPFS, se pueden reducir los costos asociados con el almacenamiento centralizado. No es necesario mantener servidores costosos, ya que la carga se distribuye entre los nodos de la red.

- Transparencia y confianza:

  Al utilizar IPFS para almacenar información sobre obras de arte en una subasta, se puede aumentar la transparencia y la confianza. Los participantes pueden verificar la autenticidad y la integridad de la información accediendo a los datos a través de la red IPFS.

En resumen, IPFS ofrece una solución descentralizada, eficiente y segura para almacenar datos relacionados con obras de arte en una subasta, brindando beneficios como la resistencia a la censura, la inmutabilidad de los datos y la mejora de la confianza en la información presentada.

2 Implementación

La implementación se basa en el código proporcionado por la profesora incluyendo una serie de modificaciones propias. La idea es crear un formulario en el que los usuarios introduzcan el nombre, época, fecha de subasta y autor de una obra de arte. Esta información se almacena como archivo txt en IPFS con un nombre de archivo que es un hash y este se almacena en un mapping en el smart contract de la subasta creado en la práctica 1.

Para hacer esto, en primer lugar se debe modificar el archivo App.js, más concretamente la función handleSubmit y el formulario HTML, quedando de la siguiente manera:

```
  const handleSubmit = async (e) => {
  e.preventDefault();
  try {
    // Obtener datos del formulario
    const formData = new FormData(e.target);
    const name = formData.get("name");
    const epoca = formData.get("epoca");
    const date = formData.get("date");
    const author = formData.get("author");

    // Crear contenido del archivo txt
    const fileContent = `Nombre: ${name}\nEpoca: ${epoca}\nFecha:
    ${date}\nAutor: ${author}`;

    // Crear un objeto Blob con el contenido del archivo
    const blob = new Blob([fileContent], { type: "text/plain" });

    // Conectar a la instancia en local de IPFS
    const ipfsClient = await create('/ip4/0.0.0.0/tcp/5001');

    // Añadir el archivo a IPFS
    const result = await ipfsClient.add(blob);

    // Añadir al sistema de archivos del nodo IPFS en local para
    // visualizarlo en el dashboard
    await ipfsClient.files.cp(`/ipfs/${result.cid}`, `/${result.cid}`);
    console.log(result.cid);

    // Añadir el CID de IPFS a Ethereum a través del contrato inteligente
    await setFileIPFS(result.cid.toString());
  } catch (error) {
    console.log(error.message);
  }
};

return (
  <div className="App">
    <header className="App-header">
      <img src={logo} className="App-logo" alt="logo" />
      <p>
        Rellene el formulario con los datos de la obra a subastar.
      </p>
      <form className="form" onSubmit={handleSubmit}>
        {/* Campos del formulario */}
        <label>
          Nombre:   <input type="text" name="name" required />
        </label>
        <br />
        <label>
          Época:   <input type="text" name="epoca" required />
        </label>
        <br />
        <label>
          Fecha subasta:   <input type="date" name="date" required />
        </label>
        <br />
```

```
          <label>
            Autor:   <input type="text" name="author" required />
          </label>
          <br />
          {/* Botón para subir el archivo */}
          <button type="submit" className="btn">Upload</button>
        </form>
      </header>
    </div>
);
```

El formulario se ha modificado para añadir los campos necesarios y en la función handleSubmit se recogen estos datos y se pasan a string con un formato determinado. Tras esto, se genera un objeto de tipo Blob que crea el archivo y este es el que sube a IPFS.

El smart contract también se debe modificar pero en menor medida, ya que únicamente debemos añadir un nuevo mapping en el que se almacenan los hashes de los archivos asociados a un address de un usuario y una función para hacer este guardado (Es la que se llama externamente al subir un archivo con el formulario web).

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

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

3 Lecciones aprendidas

La dificultad para el desarrollo de la práctica, ya que se han encontrado diversos problemas en el desarrollo del ejercicio de prueba que se han finalmente solucionado.

Además, también ha habido problemas para modificar el formulario de la aplicación para subir archivos, ya que no se conseguía que las líneas escritas por el usuario se guardasen en un archivo. Finalmente, se ha podido solucionar mediante el uso de un objeto Blob, que permite la creación de archivos.

4 Prueba

A continuación se muestran una prueba que se han realizado para comprobar el funcionamiento del contrato modificado y de IPFS.

En primer lugar, el usuario con la address 0x63Cf53258D1027f288B5940012cBf3C71112987e (el cual tiene una cuenta con ethers en MetaMask) lanza el contrato con una duración de subastas de 3600 segundos como se ve en la imagen siguiente:

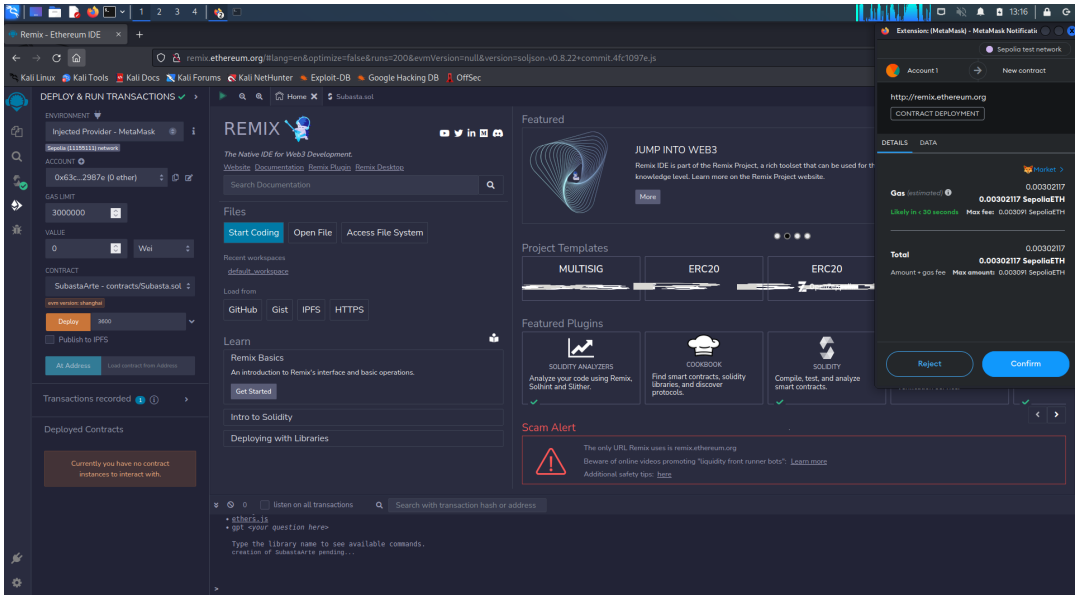

Figure 1: Deploying del contrato

Tras esto, se modifica el archivo 'addresses.js' de la aplicación 'dappipfs' de forma que se incluye la address del contrato una vez desplegado, la cual es 0x7F52D57447E45ccD4Dec24d0a8d78AaECF29c421.

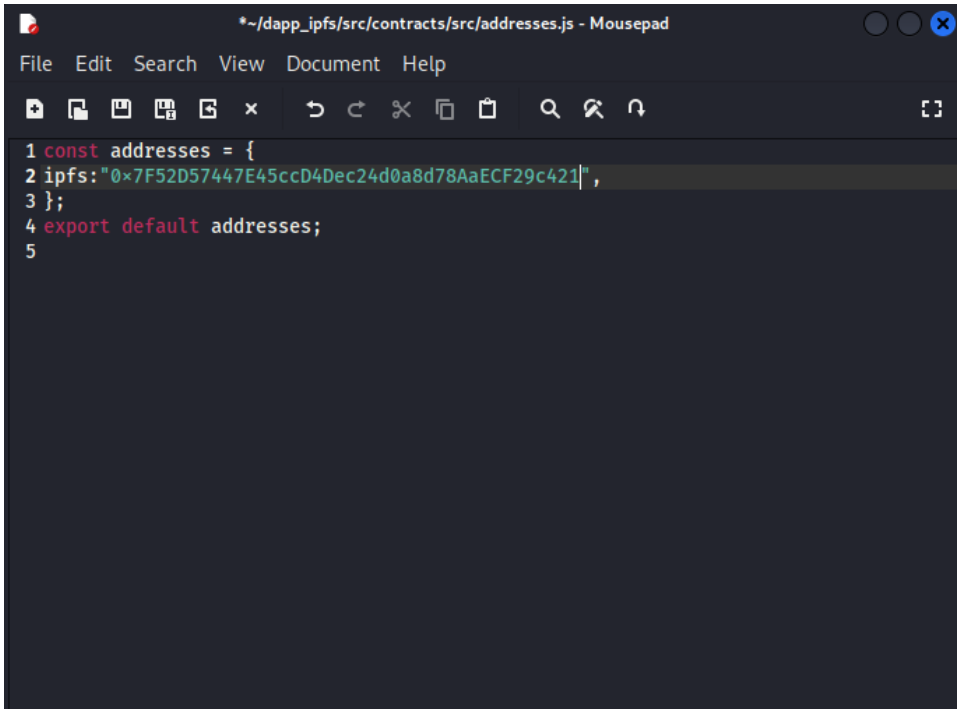

Figure 2: Modificación de 'addresses.js'

Tras esto, se despliega la aplicación dappipfs (con el comando npm start) en la dirección localhost:3000 y podemos ver el formulario en el que los usuarios subirán la información de las obras de arte que desean subastar.

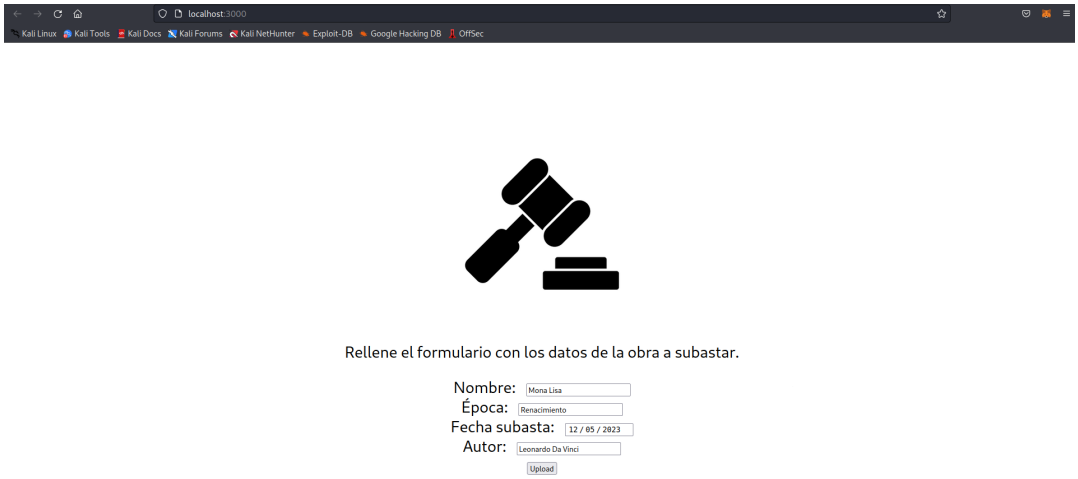

Figure 3: Dapp IPFS desplegado

A la hora de subir el archivo, el usuario debe validar la transacción, la cual tendrá un gasto mínimo de ethers (gas).

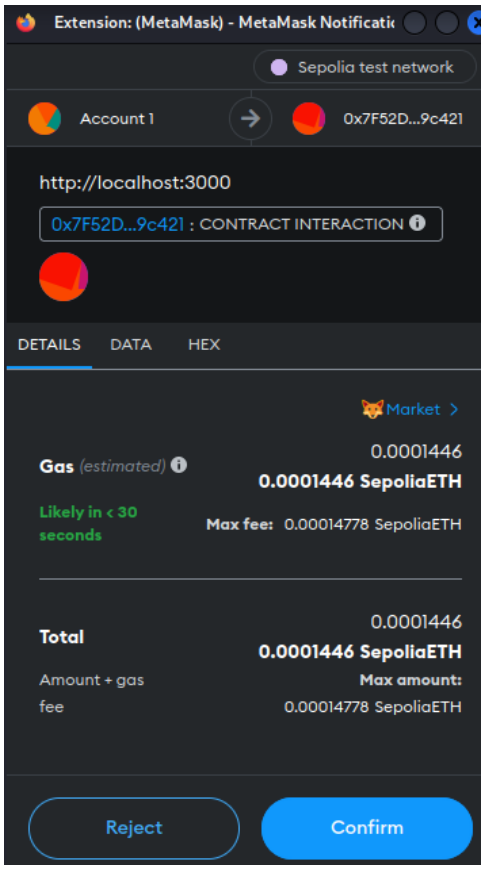

Figure 4: Validación de subida de archivo

Una vez subido el archivo, lo podremos ver en la sección archivos del nodo IPFS. Está guardado con un nombre de archivo igual a un hash.

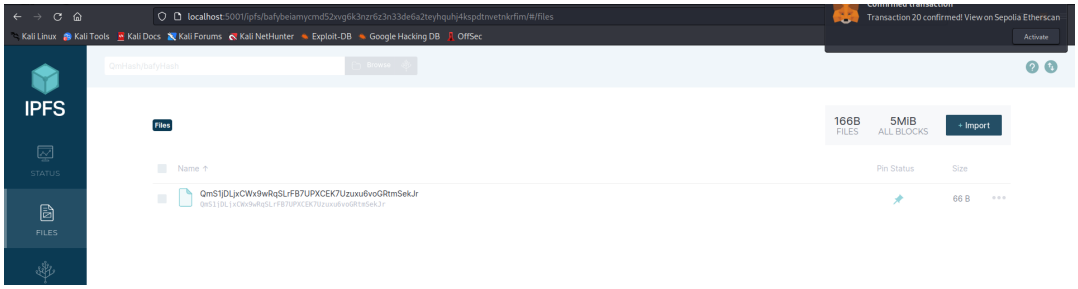

Figure 5: Archivo guardado en nodo IPFS

Por último, podemos comprobar en el contrato que el hash del nombre del archivo se ha guardado asociado al address del usuario en un mapping. Para verlo, sólo hace falta llamar al método userFiles pasándole el address del usuario.

Figure 6: Archivos asociados a un usuario