Tecnologías de Registro Distribuido y Blockchain
Máster Inter-Universitario en Ciberseguridad (MUNICS)
Universidade da Coruña (UDC) y Universidade de Vigo (UVigo)
Curso 2024-2025

Memoria Práctica 1

NFT Market

Luis Fernández Couñago
Cristian Vega Muñiz

1 Análisis y definición del escenario

El proyecto se centra en la creación de un mercado de NFTs en la blockchain pública de Ethereum, que permitirá a los usuarios comprar, vender, transferir y subastar NFTs.

1.1 Funcionalidades principales

1. **Compra-venta de NFTs**: Los usuarios podrán comprar y vender NFTs en una plataforma descentralizada utilizando criptomonedas.
2. **Transferencia de NFTs**: Los usuarios podrán transferir NFTs entre billeteras de forma directa, similar a una transacción normal en Ethereum.
3. **Subasta de NFTs**:
   (a) Subasta pública: Abierta a todos los usuarios de la plataforma.
   (b) Subasta privada: Limitada a usuarios autorizados por el subastador.

El sistema se apoya en contratos inteligentes que gestionarán todas las transacciones y subastas.

2 Diseño

2.1 Caso de uso

El sistema tendrá varios casos de uso para los diferentes roles de usuario. Se presenta a continuación un desglose de los casos más importantes:

1. **Compra de NFT**
   - Descripción: Un usuario puede navegar por los NFTs disponibles y comprar uno que esté listado para la venta.
   - Precondiciones: El NFT debe estar listado para la venta, y el usuario debe tener una billetera conectada y fondos suficientes en Ethereum para completar la compra.

- Postcondiciones: El NFT se transfiere automáticamente a la billetera del comprador, y el pago se envía al vendedor.

2. **Venta de NFT**
   - Descripción: Un usuario puede listar un NFT propio para vender.
   - Precondiciones: El usuario debe ser propietario del NFT y tener la billetera conectada.
   - Postcondiciones: El NFT queda listado en el mercado y se puede comprar por otros usuarios.
3. **Transferencia de NFT**
   - Descripción: Un usuario puede transferir un NFT de su propiedad a otra billetera.
   - Precondiciones: El usuario debe ser el propietario del NFT y contar con una dirección de la billetera de destino.
   - Postcondiciones: El NFT se transfiere a la billetera de destino.
4. **Subasta pública**
   - Descripción: Un usuario puede participar en una subasta pública, realizando pujas para adquirir un NFT.
   - Precondiciones: El usuario debe estar conectado y tener fondos suficientes para realizar pujas.
   - Postcondiciones: Si el usuario gana la subasta, el NFT se transfiere a su billetera y los fondos se envían al vendedor.
5. **Subasta privada**
   - Descripción: Un usuario autorizado puede participar en una subasta privada, realizando pujas dentro del grupo permitido.
   - Precondiciones: El usuario debe estar dentro de la lista de billeteras autorizadas para participar en la subasta.
   - Postcondiciones: Si el usuario gana la subasta, el NFT se transfiere a su billetera y los fondos se envían al vendedor.

Se puede ver en el siguiente diagrama, los usuarios que pueden participar en cada caso de uso. En el se definen los siguientes usuarios:

- Usuario: Usuario del sistema que puede puede participar en una subasta pública o en una transferencia de NFT.
- Vendedor: Heradado de Usuario, persona que posea un NFT y quiera venderlo o subastarlo.
- Comprador: Heredado de Usuario, persona que realiza la compra de un NFT.
- Comprador autorizado: Heradado de Comprador, persona que está autorizada para participar en una subasta privada.

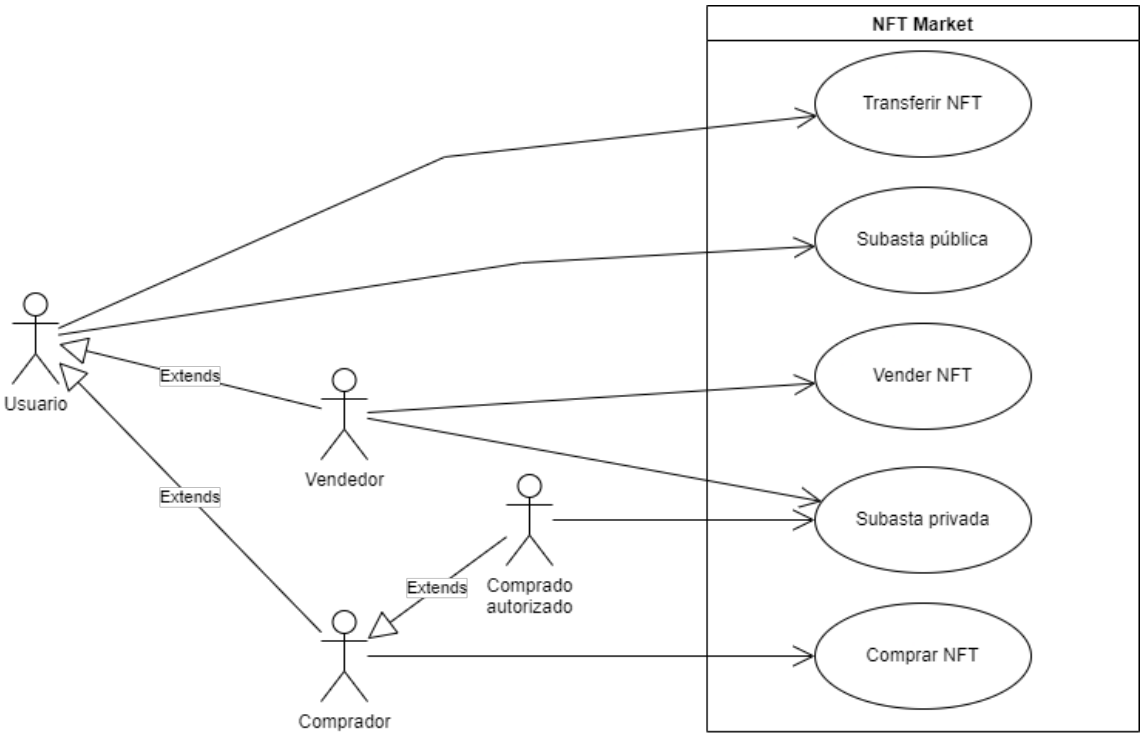


Figure 1: Diagrama de Casos de Uso

2.2 Contrato inteligente

Los contratos inteligentes serán los elementos principales para gestionar todas las interacciones dentro de la plataforma. Se desarrollarán varios contratos específicos para cada funcionalidad, utilizando Solidity, el lenguaje de programación de Ethereum.

1. **Contrato de mercado de NFTs**:
    - Funciones: Este contrato gestionará las operaciones de compra-venta de NFTs, almacenando el listado de NFTs disponibles.
    - Interacciones: Cuando un usuario lista un NFT para venta, se crea un nuevo listado de venta, en el que se guardará toda la información necesaria para su venta.
2. **Contrato de listado de venta:**
    - Funciones: Este contrato simboliza un NFT listado para venta. Guardará el id del NFT que se ofrece en la venta, así como su precio y la dirección del vendedor. El contrato proporcionará la función de compra.
    - Interacciones: Cuando un usuario compre el NFT, se comprobará que el balance ofrecido se corresponda con el precio. Tras esto, se le transferirá la propiedad del NFT al comprador, el vendedor recibirá los fondos, y se registrará la dirección del comprador en el contrato para indicar la compra.
3. **Contrato de subasta:**
    - Subasta pública: El contrato manejará las reglas de la subasta, permitiendo que cualquier usuario haga ofertas, y seleccionará al mejor postor al finalizar el tiempo de la subasta.

- Subasta privada: Este contrato tendrá un array de billeteras autorizadas, de modo que solo las billeteras incluidas puedan hacer ofertas.
- Lógica de pujas: Se gestionarán automáticamente las ofertas recibidas, y se actualizará la puja más alta hasta que termine la subasta. Al finalizar, el NFT se transferirá al ganador, y los fondos se enviarán al vendedor.

4. **Contrato de gestión de NFTs**:

   - Este contrato gestionará la propiedad y la transferencia de NFTs utilizando el estándar ERC-721 o ERC-1155, asegurando que las propiedades del NFT sean correctamente transferidas entre billeteras.

5. **Acceso a subastas privadas**:

   - El contrato de subasta privada tendrá una función para actualizar o modificar el array de billeteras autorizadas, permitiendo que solo las billeteras listadas puedan realizar pujas. Se garantizará la inmutabilidad del resto del contrato.

2.3 Usuarios del sistema

El sistema tendrá varios tipos de usuarios, cada uno con diferentes privilegios y capacidades dentro del mercado de NFTs. A continuación se describen los principales tipos de usuarios:

1. **Usuario comprador**: Este usuario busca adquirir NFTs mediante compra directa o subasta. Puede conectarse al sistema a través de su billetera de Ethereum. Puede realizar compras inmediatas de NFTs o participar en subastas (públicas y privadas si está autorizado).

2. **Usuario vendedor**: Un usuario que tiene NFTs en su posesión y desea venderlos o subastarlos. Puede listar sus NFTs para la venta a un precio fijo o para una subasta. Administra la transferencia de sus NFTs al comprador o al mejor postor, todo a través de contratos inteligentes.

3. **Usuario participante en subasta privada**: Un comprador autorizado para participar en subastas privadas. Puede hacer ofertas solo si su billetera está incluida en el array de billeteras autorizadas en el contrato de la subasta privada. Similar al usuario común, puede realizar pujas y adquirir NFTs si gana la subasta.

3 Implementación

3.1 nftMarket.sol

El contrato NFTMarket permite al propietario del contrato crear subastas públicas y privadas para NFTs y transferir NFTs a otras direcciones. Utiliza los contratos *PublicAuction* y *PrivateAuction* definidos en *auction.sol* para gestionar las subastas. Además, emite eventos cuando se crean nuevas ventas o subastas para facilitar el seguimiento de las mismas.

Declaraciones y Variables:

```
pragma solidity ^0.8.10;

import "./purchaseListing.sol";
import "./auction.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/token/ERC721/IERC721.sol";

contract NFTMarket is Ownable {
    mapping(uint256 => PurchaseListing) public listings;
    mapping(uint256 => address) public auctions;
    event AuctionCreated(address auctionContract, uint256 nftId);
    constructor(address initialOwner) Ownable(initialOwner) {}
```

- *import "./purchaseListing.sol"*: Importa el contrato PurchaseListing desde el archivo purchaseListing.sol.
- *import "./auction.sol"*: Importa el contrato Auction desde el archivo auction.sol.
- *import "@openzeppelin/contracts/access/Ownable.sol"*: Importa la interfaz Ownable de OpenZeppelin para categorizar las propiedades.
- *import "@openzeppelin/contracts/token/ERC721/IERC721.sol"*: Importa la interfaz IERC721 de OpenZeppelin para interactuar con contratos de tokens ERC721.
- *contract NFTMarket is Ownable*: Define el contrato NFTMarket que hereda de Ownable, lo que permite que ciertas funciones solo puedan ser llamadas por el propietario del contrato.
- *mapping(uint256 => PurchaseListing) public listings*: Mapea el ID de un NFT a la dirección del contrato de venta correspondiente.
- *mapping(uint256 => address) public auctions*: Mapea el ID de un NFT a la dirección del contrato de subasta correspondiente.
- *event AuctionCreated(address auctionContract, uint256 nftId)*: Declara un evento que se emite cuando se crea una nueva subasta.
- *constructor(address initialOwner) Ownable(initialOwner)* : Contructor que toma una dirección *initialOwner* y la pasa al constructor de *Ownable*.

Función *createPublicAuction*

```
function createPublicAuction(
    address _nftContract,
    uint256 _nftId,
    uint _biddingTime,
    address _beneficiary
) external onlyOwner {
    PublicAuction auction = new PublicAuction(_nftContract, _nftId,
        _biddingTime, _beneficiary, msg.sender);
    auctions[_nftId] = address(auction);
    emit AuctionCreated(address(auction), _nftId);
}
```

Esta función crea una subasta pública con los contratos especificados. Almacena la dirección del contrato de subasta en el mapeo *auctions*. y emite el evento *AuctionCreated* con la dirección del contrato de subasta y el ID del NFT. Los parámetros son: *nftContract*: Dirección del contrato del NFT. *nftId*: ID del NFT que se subastará. *biddingTime*: Duración de la subasta. *beneficiary*: Dirección del beneficiario que recibirá los fondos de la subasta.

Función *createPrivateAuction*

```
1    function createPrivateAuction(
2        address _nftContract,
3        uint256 _nftId,
4        uint _biddingTime,
5        address _beneficiary,
6        address[] calldata authorizedUsers
7    ) external onlyOwner {
8        PrivateAuction auction = new PrivateAuction(_nftContract, _nftId,
             _biddingTime, _beneficiary, msg.sender, authorizedUsers);
9        auctions[_nftId] = address(auction);
10       emit AuctionCreated(address(auction), _nftId);
11   }
```

Al igual que la función *createPublicAuction* crea una subasta, en este caso en vez de ser pública es privada.

Función *transferNFT*

```
1    function transferNFT(
2        address _nftContract,
3        uint256 _nftId,
4        address _to
5    ) external onlyOwner {
6        IERC721(_nftContract).safeTransferFrom(msg.sender, _to, _nftId);
7    }
```

Transfiere un NFT desde el propietario del contrato a la dirección especificada. Utiliza la función *safeTransferFrom* de la interfaz IERC721 para realizar la transferencia segura del NFT.

Función *listNFTForSale*

```
1        function listNFTForSale(
2        address _nftContract,
3        uint256 _nftId,
4        uint256 _price,
5        address _sellerAddress
6    ) external {
7        PurchaseListing listing = new PurchaseListing(_nftContract, _nftId,
             _price, _sellerAddress, msg.sender);
8        listings[_nftId] = listing;
9
10       emit NFTListedForSale(msg.sender, _nftId, _price);
11   }
```

Crea un nuevo listado de venta de un NFT, y lo guarda en el mapping *listings*. Los parámetros son: nftContract: Dirección del contrato del NFT. nftId : ID del NFT que se

pone a la venta. price: Precio del NFT. sellerAddress: Dirección del vendedor del NFT que recibirá el pago tras la compra.

3.2 auction.sol

El sistema de subastas de NFTs definido en estos contratos permite la creación y gestión de subastas públicas y privadas. El contrato *Auction* proporciona la lógica común para las subastas, mientras que los contratos *PublicAuction* y *PrivateAuction* heredan de *Auction* y añaden funcionalidades específicas. El contrato *NFTMarket* centraliza la gestión de subastas y transferencias de NFTs, permitiendo al propietario del contrato crear nuevas subastas y transferir NFTs de manera segura.

3.2.1 Contrato *Auction*

Declaraciones y Variables:

```
1   pragma solidity ^0.8.10;
2
3   import "@openzeppelin/contracts/access/Ownable.sol";
4   import "@openzeppelin/contracts/token/ERC721/IERC721.sol";
5
6   contract Auction is Ownable {
7       IERC721 public nftContract;
8       uint256 public nftId;
9       address public beneficiaryAddress;
10      uint public auctionCloseTime;
11
12      address public topBidder;
13      uint public topBid;
14
15      mapping(address => uint) returnsPending;
16      bool auctionComplete;
17
18      event topBidIncreased(address bidder, uint bidAmount);
19      event auctionResult(address winner, uint bidAmount);
```

- *IERC721 public nftContract*: Dirección del contrato del NFT.
- *uint256 public nftId*: ID del NFT que se subastará.
- *address public beneficiaryAddress*: Dirección del beneficiario que recibirá los fondos de la subasta.
- *uint public auctionCloseTime;*: Tiempo de cierre de la subasta en formato de timestamp Unix.
- *address public topBidder*: Dirección del postor con la oferta más alta.
- uint public topBid: Cantidad de la oferta más alta.
- *mapping(address => uint) returnsPending*: Mapeo que almacena los fondos pendientes de devolución a los postores anteriores.
- *bool auctionComplete*: Booleano que indica si la subasta ha finalizado.

- *event topBidIncreased(address bidder, uint bidAmount)*: Evento que se emite cuando hay una nueva oferta más alta.
- *event auctionResult(address winner, uint bidAmount)*: Evento que se emite cuando la subasta finaliza.

Constructor:

```
constructor(
        address _nftContract,
        uint256 _nftId,
        uint _biddingTime,
        address _beneficiary,
        address initialOwner
    ) Ownable(initialOwner) {
        nftContract = IERC721(_nftContract);
        nftId = _nftId;
        beneficiaryAddress = _beneficiary;
        auctionCloseTime = block.timestamp + _biddingTime;
    }
```

- *address nftContract*: Dirección del contrato del NFT.
- *uint256 nftId*: ID del NFT que se subastará.
- *uint biddingTime*: Duración de la subasta.
- *address beneficiary*: Dirección del beneficiario que recibirá los fondos de la subasta.
- *address initialOwner*: Dirección del propietario del NFT.

Función *bid*:

```
function bid(uint amount) external payable virtual {
        require(block.timestamp <= auctionCloseTime, "Auction already
            closed");
        require(amount > topBid, "There already is a higher bid");

        if (topBidder != address(0)) {
            returnsPending[topBidder] += topBid;
        }
        topBidder = msg.sender;
        topBid = amount;
        emit topBidIncreased(msg.sender, amount);
    }
```

Permite a los usuarios hacer una oferta en la subasta. Verifica que la subasta no haya terminado y que la oferta sea mayor que la oferta actual más alta. Si hay una oferta anterior, se guarda en *returnsPending* para que el postor anterior pueda retirarla. Por último, actualiza el postor y la oferta más alta y emite el evento *topBidIncreased*.

Función *withdraw*:

```
function withdraw() external returns (bool) {
    uint bidAmount = returnsPending[msg.sender];
    if (bidAmount > 0) {
        returnsPending[msg.sender] = 0;
        if (!payable(msg.sender).send(bidAmount)) {
            returnsPending[msg.sender] = bidAmount;
            return false;
        }
    }
    return true;
}
```

Permite a los postores retirar sus ofertas anteriores que fueron superadas. Verifica si hay fondos pendientes para el remitente y los envía de vuelta. Si la transferencia falla, restablece el monto pendiente.

Función *closeAuction*:

```
function closeAuction() external onlyOwner {
    require(block.timestamp >= auctionCloseTime, "Auction not yet ended
        ");
    require(!auctionComplete, "closeAuction has already been called");

    auctionComplete = true;
    emit auctionResult(topBidder, topBid);

    nftContract.safeTransferFrom(beneficiaryAddress, topBidder, nftId);
    payable(beneficiaryAddress).transfer(topBid);
}
```

Verifica que la subasta haya terminado y que no se haya cerrado previamente. Marca la subasta como completa, emite el evento *auctionResult* y transfiere el NFT al ganador.

3.2.2 Contrato *PublicAuction*

```
contract PublicAuction is Auction {
    constructor(
        address _nftContract,
        uint256 _nftId,
        uint _biddingTime,
        address _beneficiary,
        address initialOwner
    ) Auction(_nftContract, _nftId, _biddingTime, _beneficiary,
        initialOwner) {}
}
```

El contrato *PublicAuction* hereda del contrato *Auction* y no necesita ninguna funcionalidad adicional específica.

3.2.3 Contrato *PrivateAuction*

```
contract PrivateAuction is Auction {
    mapping(address => bool) authorizedUsers;

    constructor(
        address _nftContract,
        uint256 _nftId,
        uint _biddingTime,
        address _beneficiary,
        address initialOwner,
        address[] memory _authorizedUsers
    ) Auction(_nftContract, _nftId, _biddingTime, _beneficiary,
        initialOwner) {
        for (uint i = 0; i < _authorizedUsers.length; i++) {
            authorizedUsers[_authorizedUsers[i]] = true;
        }
    }

    function authorizeUser(address user) external onlyOwner {
        authorizedUsers[user] = true;
    }

    function bid(uint amount) external payable override {
        require(authorizedUsers[msg.sender], "You are not authorized to
            participate in this auction");
        require(block.timestamp <= auctionCloseTime, "Auction already
            closed");
        require(amount > topBid, "There already is a higher bid");

        if (topBidder != address(0)) {
            returnsPending[topBidder] += topBid;
        }
        topBidder = msg.sender;
        topBid = amount;
        emit topBidIncreased(msg.sender, amount);
    }
}
```

El contrato *PrivateAuction* hereda del contrato *Auction* y añade la funcionalidad de autorización de usuarios.

- *mapping(address => bool) authorizedUsers*: Mapeo que almacena los usuarios autorizados.
- *function authorizeUser(address user) external onlyOwner*: Permite al propietario del contrato autorizar a los usuarios para participar en la subasta.
- *function bid() external payable override*: Permite a los usuarios hacer una oferta en la subasta. Verifica que la subasta no haya terminado y que la oferta sea mayor que la oferta actual más alta. Si hay una oferta anterior, se guarda en *returnsPending* para que el postor anterior pueda retirarla. Por último, actualiza el postor y la oferta más alta y emite el evento *topBidIncreased*.

3.3 purchaseListing.sol

El sistema de compra-venta de NFTs diseñado permite guardar una entrada por cada venta disponible en forma del siguiente contrato PurchaseListing, el cual define la estructura de datos que se guardarán para cada venta disponible.

Declaraciones y Variables:

```
pragma solidity ^0.8.10;

import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/token/ERC721/IERC721.sol";

contract PurchaseListing is Ownable {

    IERC721 public nftContract;
    uint256 public nftId;
    uint256 public price;
    address public sellerAddress;
    address public buyerAddress;

    event NFTPurchased(address buyer, uint256 nftId, uint256 price);
```

- *IERC721 public nftContract*: Dirección del contrato del NFT.
- *uint256 public nftId*: ID del NFT que está en venta.
- *uint256 public price*: Precio del NFT en venta.
- *address public sellerAddress*: Dirección del vendedor del NFT.
- *address public buyerAddress*: Dirección del comprador del NFT.
- *event NFTPurchased(address buyer, uint256 nftId, uint256 price)*: Evento que se emite cuando se realiza la compra.

Constructor:

```
    constructor(
        address _nftContract,
        uint256 _nftId,
        uint256 _price,
        address _sellerAddress,
        address initialOwner
    ) Ownable(initialOwner) {
        nftContract = IERC721(_nftContract);
        nftId = _nftId;
        price = _price;
        sellerAddress = _sellerAddress;

    }
```

- *address nftContract*: Dirección del contrato del NFT.
- *uint256 nftId*: ID del NFT que está en venta.

- *uint256 price*: Precio del NFT en venta.
- *address sellerAddress*: Dirección del vendedor del NFT.
- *address initialOwner*: Dirección del propietario del NFT.

Función *buyNFT*:

```
1    function buyNFT() external payable {
2        require(msg.value == price, "Funds are not the exact price to
             purchase the NFT");
3        nftContract.safeTransferFrom(sellerAddress, msg.sender, nftId);
4        payable(sellerAddress).transfer(msg.value);
5        buyerAddress=msg.sender;
6        emit NFTPurchased(msg.sender, nftId, price);
7    }
```

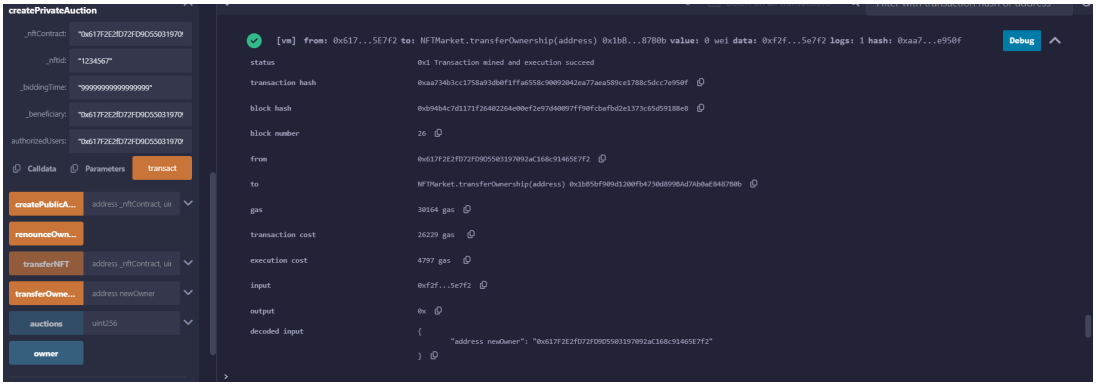
Permite a los usuarios comprar en el NFT listado. Verifica que el valor pagado se corresponde con el precio del NFT. Si es correcto, transfiere la propiedad del NFT a la dirección del comprador, y se le transfieren los fondos a la dirección del vendedor. Por último, se actualiza el campo *buyerAddress* con la dirección del comprador, y sedel actualiza el postor y se emite el evento *NFTPurchased*.

4 Pruebas

4.1 Pruebas de nftMarket.sol

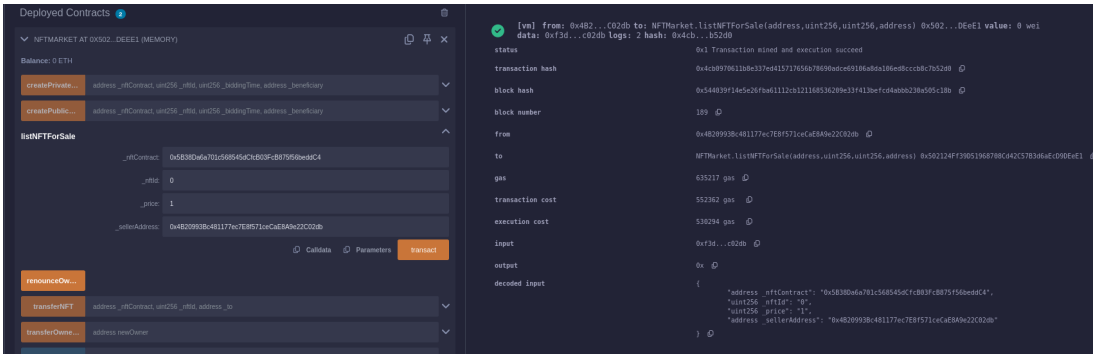
Crear una puja pública

Crear una puja privada



Transferir la propiedad del NFT a otro usuario

Listar NFT para venta

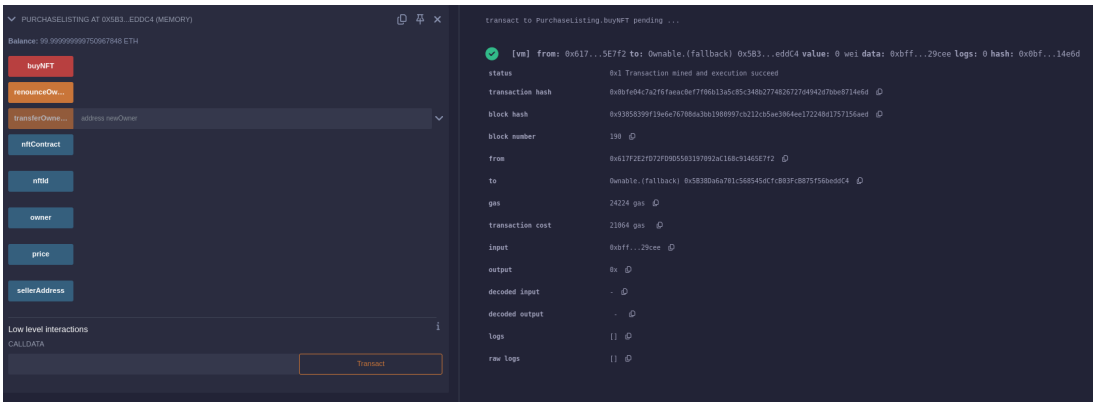


4.2 Pruebas de auction.sol

Hacer una puja

4.3 Pruebas de purchaseListing.sol

Comprar un NFT



Comprar un NFT sin enviar dinero