Blockchain Project Report
RealEstate Blockchain

Name	ID's
Abdelrahman Hazem	320210054
Ahmed Fahmy	320210044
Abdallah Radwan	320210053

Summary

This blockchain project involves the development of a decentralized application (DApp) for managing real estate transactions through smart contracts on the Ethereum blockchain.
The project comprises two main smart contracts: RealEstate.sol and Escrow.sol, and a testing script Escrow.js which ensures the functionality and correctness of the smart contracts.
The project aims to streamline property transactions by utilizing blockchain technology to ensure transparency, security, and efficiency.

Smart Contracts Overview

RealEstate.sol contract

Purpose:

The **RealEstate.sol** contract is designed to Governs the property.
This contract leverages the ERC721 standard for NFTs and extends its functionality with additional features for storing metadata through the ERC721URIStorage extension. Below, we break down the contract's components and functionality in detail.

Key Components and Functions

- **Libraries and Inheritance**:
  - **import "@openzeppelin/contracts/utils/Counters.sol";**: Imports the Counters library from OpenZeppelin, which provides a counter that can be incremented or decremented.
  - **import "@openzeppelin/contracts/token/ERC721/ERC721.sol";**: Imports the base ERC721 contract from OpenZeppelin.
  - **import"@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";**: Imports the ERC721URIStorage extension, which adds support for token URIs.

- **Contract Definition**:
  - **contract RealEstate is ERC721URIStorage**: Defines the **RealEstate** contract that inherits from the **ERC721URIStorage** contract.
- **State Variables:**
  - **Counters.Counter private _tokenIds;**
    - A counter that keeps track of the next token ID to be minted. It uses the Counters library for safe incrementing and decrementing operations.

Constructor:

- Initializes the RealEstate contract.
- The ERC721 constructor is called with the name "Real Estate" and the symbol "REAL"

Functions

1. **mint**:

```
11
12
13      function mint(string memory tokenURI) public returns (uint256) { // mint with specific tokenURI
14          _tokenIds.increment;
15          uint256 newItemId = _tokenIds.current();
16          _mint(msg.sender, newItemId); // mint it with the intenal minting function from ERC721URIStorage
17          _setTokenURI(newItemId, tokenURI);
18          return newItemId;
19      }
20
```

- **Purpose**: Mints a new real estate token with a specific URI.
- **Parameters**: **tokenURI** - A string representing the metadata URI of the property.
- **Returns**: **uint256** - The ID of the newly minted token.
- **Functionality**:
  - **_tokenIds.increment()**: Increments the token ID counter.
  - **uint256 newItemId = _tokenIds.current();**: Retrieves the current value of the token ID counter.
  - **_mint(msg.sender, newItemId);**: Mints a new token with the ID **newItemId** and assigns it to the address that called the function (**msg.sender**).
  - **_setTokenURI(newItemId, tokenURI);**: Sets the token URI for the newly minted token to the provided **tokenURI**.

2. **TotalSupply:**

```
function totalSupply() public view returns (uint256) { // to see how many NFT have been currently minted
return _tokenIds.current();

}
```

- **Purpose**: Returns the total number of tokens minted so far.
- **Returns**: **uint256** - The current value of the token ID counter, representing the total supply of tokens.
- **Functionality**: Simply returns the current count of the **_tokenIds** counter.

Escrow.sol Contract:

Overview:

The 'Escrow.sol' contract is designed to facilitate secure and transparent real estate transactions on the blockchain by leveraging smart contracts.
It acts as an intermediary that holds funds and the property (represented as NFTs) until all conditions of the sale are met. This ensures that both the buyer and seller adhere to the agreed-upon terms before the transaction is finalized.

Key Components and Functions

- **Interface and Inheritance**:
  - The contract includes an interface, **IERC721**, which defines the **transferFrom** function necessary for transferring NFTs.
  - The **Escrow** contract itself manages the escrow process, interacting with the **IERC721** interface to handle NFT transactions.
- **State Variables**:
  - **nftAddress**: The address of the NFT contract.
  - **seller**: The address of the seller, who is the owner of the property.
  - **inspector**: The address of the inspector, responsible for verifying the property condition.
  - **lender**: The address of the lender, who may provide financing for the purchase.
  - Mappings to track various details:
    - **isListed**: Tracks whether a property (by NFT ID) is listed for sale.
    - **purchasePrice**: The purchase price for each listed property.
    - **escrowAmount**: The escrow amount required for each property.
    - **buyer**: The address of the buyer for each listed property.
    - **inspectionPassed**: The inspection status of each property.
    - **approval**: Tracks approval statuses from the buyer, seller, and lender.
    -

- **Modifiers**:

```
    modifier onlyBuyer(uint256 _nftID) {
        require(msg.sender == buyer[_nftID], "Only buyer can call this method");
        _;
    }

    modifier onlySeller() {
        require(msg.sender == seller, "Only seller can call this method");
        _;
    }

    modifier onlyInspector() {
        require(msg.sender == inspector, "Only inspector can call this method");
        _;
    }
```

  - **onlyBuyer**: Ensures that only the designated buyer can call certain functions.
  - **onlySeller**: Ensures that only the seller can call certain functions.
  - **onlyInspector**: Ensures that only the inspector can call certain functions.

Constructor

The constructor initializes the contract by setting the addresses for the NFT contract, seller, inspector, and lender. These addresses are crucial for the contract's operations as they define the key participants in the escrow process.

Functions

1. **list**:

```
function list(uint256 _nftID,address _buyer,uint256 _purchasePrice,uint256 _escrowAmount) public payable onlySeller {

    // Transfer NFT from seller to this contract

    IERC721(nftAddress).transferFrom(msg.sender, address(this), _nftID);
    isListed[_nftID] = true;
    purchasePrice[_nftID] = _purchasePrice;
    escrowAmount[_nftID] = _escrowAmount;
    buyer[_nftID] = _buyer;
}
```

- **Purpose**: Lists a property for sale.
- **Parameters**: Takes the NFT ID, buyer's address, purchase price, and escrow amount.
- **Functionality**:
  - Transfers the NFT from the seller to the escrow contract.
  - Updates mappings to reflect the listing status, purchase price, escrow amount, and buyer information.

2. **depositEarnest**:
   - **Purpose**: Allows the buyer to deposit the earnest money.
   - **Parameters**: Takes the NFT ID.
   - **Functionality**: Ensures the deposit meets the required escrow amount.
3. **updateInspectionStatus**:

```
68      // Update Inspection Status (only inspector)
69      function updateInspectionStatus(uint256 _nftID, bool _passed)public onlyInspector
70      {
71          inspectionPassed[_nftID] = _passed;
72      }
```

   - **Purpose**: Updates the inspection status of the property.
   - **Parameters**: Takes the NFT ID and a boolean indicating whether the property passed inspection.
   - **Functionality**: Sets the inspection status for the specified property.
4. **approveSale**:

```
74      // Approve Sale
75      function approveSale(uint256 _nftID) public {
76          approval[_nftID][msg.sender] = true;
77      }
```

   - **Purpose**: Allows participants to approve the sale.
   - **Parameters**: Takes the NFT ID.
   - **Functionality**: Updates the approval status for the caller (buyer, seller, or lender).
5. **finalizeSale**:

```
85      function finalizeSale(uint256 _nftID) public {
86          require(inspectionPassed[_nftID]);
87          require(approval[_nftID][buyer[_nftID]]);
88          require(approval[_nftID][seller]);
89          require(approval[_nftID][lender]);
90          require(address(this).balance >= purchasePrice[_nftID]);
91
92          isListed[_nftID] = false;
93
94          (bool success, ) = payable(seller).call{value: address(this).balance}( // send ethers to the seller
95              ""
96          );
97          require(success);
98
99          IERC721(nftAddress).transferFrom(address(this), buyer[_nftID], _nftID);
100     }
```

   - **Purpose**: Finalizes the sale of the property.
   - **Parameters**: Takes the NFT ID.
   - **Functionality**:
     - Ensures that all conditions (inspection passed, all approvals, correct funds) are met.
     - Transfers the NFT from the escrow contract to the buyer.
     - Transfers the purchase funds to the seller.
     - Updates the listing status to reflect that the sale is complete.

6. **cancelSale**:

```
104         function cancelSale(uint256 _nftID) public {
105             if (inspectionPassed[_nftID] == false) {
106                 payable(buyer[_nftID]).transfer(address(this).balance);
107             } else {
108                 payable(seller).transfer(address(this).balance);
109             }
110         }
```

   - **Purpose**: Cancels the sale and handles the return of the earnest deposit.
   - **Parameters**: Takes the NFT ID.
   - **Functionality**:
     - If the inspection did not pass, the earnest deposit is refunded to the buyer.
     - If the inspection passed, the funds are transferred to the seller.
7. **receive**:
   - **Purpose**: Allows the contract to receive Ether.
   - **Functionality**: Enables the contract to accept and store funds.
8. **getBalance**:
   - **Purpose**: Retrieves the balance of the contract.
   - **Functionality**: Returns the current balance of the contract.

Detailed Functionality

- **Listing and Escrow Management**:
  - The **list** function ensures that the property is transferred to the contract and sets up all necessary details for the sale.
  - **depositEarnest** secures the buyer's commitment by requiring an earnest deposit.
- **Inspection and Approval**:
  - The inspection process is managed by **updateInspectionStatus**, ensuring that only the inspector can update the status.
  - The **approveSale** function allows the buyer, seller, and lender to approve the sale, ensuring that all parties agree to proceed.
- **Finalizing and Cancelling Sales**:
  - **finalizeSale** checks multiple conditions to ensure a secure and fair transaction, transferring ownership and funds appropriately.
  - **cancelSale** provides a mechanism to handle the sale cancellation, ensuring fair handling of the earnest deposit based on inspection results.
- **Balance and Fund Management**:
  - The **receive** function ensures the contract can receive Ether.
  - **getBalance** provides transparency about the contract's funds.

Testing the Escrow Contract

The Escrow.js file contains a series of tests written using Hardhat and the Chai assertion library. These tests ensure the proper functionality of the Escrow contract. Each section of the test script is designed to verify specific features and behaviors of the contract.

Overview:

The testing process involves several key steps:

- Setting up the environment and deploying the contracts.
- Verifying the initial deployment and state variables.
- Testing the listing process for NFTs.
- Ensuring the deposit of earnest money functions correctly.
- Checking the inspection process and status updates.
- Confirming the approval process by all parties involved.
- Finalizing the sale and verifying the transfer of ownership and funds.

Detailed Explanation of Tests

1. Setup:

- **Accounts Setup:** The test script begins by setting up accounts using ethers.getSigners(), which provides different signers for the buyer, seller, inspector, and lender.

- **Deploying Contracts:**

```
16          // Deploy Real Estate
17          const RealEstate = await ethers.getContractFactory('RealEstate')
18          realEstate = await RealEstate.deploy()
19
24        // Deploy Escrow
25        const Escrow = await ethers.getContractFactory('Escrow')
26        escrow = await Escrow.deploy(
27            realEstate.address,
28            seller.address,
29            inspector.address,
30            lender.address
31        )
```

  - The RealEstate contract is deployed first.
  - A transaction is then created to mint an NFT (representing a property) by the seller.
  - The Escrow contract is deployed next, initialized with the addresses of the RealEstate contract, seller, inspector, and lender.
  - The seller approves the transfer of the NFT to the Escrow contract.
  - The property is listed on the Escrow contract with the specified buyer, purchase price, and escrow amount.

2. Deployment:

- **Returns NFT Address:** Ensures the NFT address stored in the Escrow contract matches the deployed RealEstate contract address.
- **Returns Seller:** Confirms the seller's address in the Escrow contract.
- **Returns Inspector**: Checks the inspector's address.
- **Returns Lender:** Verifies the lender's address.

3. Listing:

```
describe('Listing', () => {
    it('Updates as listed', async () => {
        const result = await escrow.isListed(1)
        expect(result).to.be.equal(true)
    })

    it('Returns buyer', async () => {
        const result = await escrow.buyer(1)
        expect(result).to.be.equal(buyer.address)
    })

    it('Returns purchase price', async () => {
        const result = await escrow.purchasePrice(1)
        expect(result).to.be.equal(tokens(10))
    })

    it('Returns escrow amount', async () => {
        const result = await escrow.escrowAmount(1)
        expect(result).to.be.equal(tokens(5))
    })

    it('Updates ownership', async () => {
        expect(await realEstate.ownerOf(1)).to.be.equal(escrow.address) // ownerOf(1) stands for the first NFT was created
    })
})
```

- **Updates as Listed:** Verifies that the property is marked as listed.
- **Returns Buyer:** Ensures the correct buyer's address is stored.
- **Returns Purchase Price:** Confirms the stored purchase price.
- **Returns Escrow Amount:** Verifies the escrow amount.
- **Updates Ownership:** Checks that the ownership of the NFT is transferred to the Escrow contract.

4. Deposits:

- **Depositing Earnest Money:**
    - The buyer deposits the earnest money.
    - Updates Contract Balance: Confirms that the contract balance is updated with the deposited amount.

5. Inspection:

```
    describe('Inspection', () => {
        beforeEach(async () => {
            const transaction = await escrow.connect(inspector).updateInspectionStatus(1, true)
            await transaction.wait()
        })

        it('Updates inspection status', async () => {
            const result = await escrow.inspectionPassed(1)
            expect(result).to.be.equal(true)
        })
    })
```

- **Updates Inspection Status:**
  - The inspector updates the inspection status.
  - Ensures Status is True: Verifies that the inspection status is correctly updated to true.

6. Approval:

```
describe('Approval', () => {
    beforeEach(async () => {
        let transaction = await escrow.connect(buyer).approveSale(1)
        await transaction.wait()

        transaction = await escrow.connect(seller).approveSale(1)
        await transaction.wait()

        transaction = await escrow.connect(lender).approveSale(1)
        await transaction.wait()
    })

    it('Updates approval status', async () => {
        expect(await escrow.approval(1, buyer.address)).to.be.equal(true)
        expect(await escrow.approval(1, seller.address)).to.be.equal(true)
        expect(await escrow.approval(1, lender.address)).to.be.equal(true)
    })
})
```

- **Approval Status:**
  - Each party (buyer, seller, lender) approves the sale.
  - Updates Approval Status: Confirms that the approval statuses for each party are updated correctly.

7. Sale:

```
133     describe('Sale', () => {
134         beforeEach(async () => {
135             let transaction = await escrow.connect(buyer).depositEarnest(1, { value: tokens(5) })
136             await transaction.wait()
137 
138             transaction = await escrow.connect(inspector).updateInspectionStatus(1, true)
139             await transaction.wait()
140 
141             transaction = await escrow.connect(buyer).approveSale(1)
142             await transaction.wait()
143 
144             transaction = await escrow.connect(seller).approveSale(1)
145             await transaction.wait()
146 
147             transaction = await escrow.connect(lender).approveSale(1)
148             await transaction.wait()
149 
150             await lender.sendTransaction({ to: escrow.address, value: tokens(5) })
151 
152             transaction = await escrow.connect(seller).finalizeSale(1)
153             await transaction.wait()
154         })
155 
156         it('Updates ownership', async () => {
157             expect(await realEstate.ownerOf(1)).to.be.equal(buyer.address)
158         })
159 
160         it('Updates balance', async () => {
161             expect(await escrow.getBalance()).to.be.equal(0)
162         })
163     })
164 })
```

- **Finalizing the Sale:**
  - All conditions (deposit, inspection, approvals) are met.
  - The sale is finalized, transferring the NFT to the buyer and the funds to the seller.
  - **Updates Ownership:** Ensures the NFT ownership is transferred to the buyer.
  - **Updates Balance:** Confirms that the contract balance is zero after the sale is finalized.