NFT Marketplace Smart Contract Development

Zhan Shu UNI: zs2584 | Yuan Dou UNI: yd2676 | Zhuofei Lin UNI: zl3193 | Qiao Zhang UNI: qz2486 | Yufan Zhang UNI: yz4309
Columbia University
git repository link:https://github.com/bridges989/NFT_data_warehouse

Abstract
In this project, we develop a NFT marketplace smart contract which allows users to mint, buy, sell, and trade unique digital assets represented as NFTs. The project involves creating an ERC-1155 compliant smart contract, designing a user interface for the marketplace, and implementing additional features to ensure the marketplace's security and usability.

1. Summary

1.1 Procedure

Firstly, we design the NFT Marketplace Smart Contract by using ERC-1155 standards. Compared to ERC-721, ERC-1155 is more suitable for the NFT marketplace due to the following differences. ERC-1155 supports both the creation of NFTs and semi-fungible tokens, facilitating the conversion of digital or real-world assets into NFTs, while ERC-721 only supports the creation of NFTs. ERC-1155 provides more security by offering a unique feature known as the ‘safe transfer’ function, which enables hassle-free transactions and allows reclaiming the assets by the issuer if sent to the wrong address. ERC-1155 supports batch transfers, helping reduce the load on the network, thereby leading to less transaction cost and time. ERC-1155 has more flexibility in maintaining a large number of Uniform Resource Identifier (URI) codes, compared to ERC-721’s use of static metadata incorporated within a smart contract.[1] We define the structure of the smart contract and implement the minting, ownership, transferring, purchasing, sale list, and metadata functions.

Secondly, we deploy our smart contract to the target network. Because our design is developed in the Truffle development suite, we use the truffle command migrate to deploy our network. In the migration file, we firstly obtain our NFT Market contract artifact from its JSON file and export it to the truffle deployer. Then Truffle deployers deploy our smart contract to the according network based on the configuration file. We export our contract to the port 9545 for MetaMask to auto detect the balance amount of our account.

Lastly, we create and run the test script to verify the functionality of NFT Marketplace smart contract under different use cases. The smart contract should correctly interact with the accounts and the NFT in the network if the test has successfully passed.

1.2 Methodology

1.2.1 Smart Contract
During the design of the NFT Marketplace Smart Contract, we create a Truffle project and program the contract *contracts/NFTContract.sol* in Solidity. We import some contracts from an external package “openzeppelin”. We define a Struct “NFT” with 4 attributes: “name”, “description”, “price”, and “forSale”, representing the name, description, price,

and if it is for sale. We also define two public mappings: nfts, which associates the tokenId with the corresponding NFT; nftCreators, which associates tokenId with the creator of the NFT. Then we define 4 events NFTCreated(), NFTListed(), NFTUnlisted(), and NFTPurchased() which can be emitted by the contract when specific actions occur.

The function createNFT(name, description, tokenURI) is the process of creating a NFT on the marketplace. Each time a new NFT is created, a new tokenId is assigned to the NFT with an increment of 1. We use _mint(msg.sender, tokenId, 1, "") to create 1 token of tokenId, and assign it to account msg.sender, and use _setURI(tokenId, tokenURI) to associate tokenURI with the new tokenId.

Then we store the new NFT structure in nfts mapping using tokenId as the key, and store the address of the creator of the NFT in nftCreators mapping, using tokenId as the key. Finally emit the event NFTCreated(tokenId, name, description, 0) to create a NFT on the marketplace.

The function transferNFT(to, tokenId) is the process of transferring NFT from accounts on the marketplace. We check if the account requesting the transfer is the creator of the NFT. If yes, use safeTransferFrom() to safely transfer the NFT from the owner's account to the recipient account. And update the owner's address in nftCreators to the receipt's account on the marketplace.

The function listNFTForSale(tokenId, price) is the process of listing the NFT for sale on the marketplace. We check if the account requesting listing for sale is the owner of the NFT. We also check if the NFT is currently not listed for sale. If yes, set the price nfts[tokenId].price to price and the sale status nfts[tokenId].forSale to true on the marketplace. Then emit the event NFTListed(tokenId, price).

The function removeNFTFromSale (tokenId) is the process of removing the NFT from the sale list. We check if the account requesting removal from sale is the owner of the NFT. We also check if the NFT is currently listed for sale. If yes, set the sale status nfts[tokenId].forSale to false on the marketplace. Then emit the event NFTUnlisted(tokenId).

The function purchaseNFT(tokenId) is the process of purchasing a NFT on the marketplace. We check if the NFT is for sale, if the buyer has sufficient funds to purchase this NFT, if this NFT has already been sold, and if the buyer already owns this NFT. We use setApprovalForAll(seller, true) to ensure the contract is approved to handle tokens on behalf of the seller. Use safeTransferFrom(seller, msg.sender, tokenId, 1, "") to transfer the NFT token from the seller to the buyer. Update the token ownership using nftCreators[tokenId] = msg.sender and set the sale status of token nfts[tokenId].forSale to false. Transfer the funds from the buyer to the seller by using sellerPayable.sendValue(msg.value). At the end, emit the event NFTPurchased(tokenId, msg.sender, nfts[tokenId].price).

What's more, some functions are created to read the information of the NFTs according to the tokenId:
printNFTsName(tokenId),
printNFTsDescription(tokenId),
printNFTsOwner(tokenId),
printNFTsPrice(tokenId),
printNFTsforSale(tokenId).
And function printUserBalance(user) to get the user's balance and printAllNFTsAvailableforSale() to show all the NFTs for sale.

1.2.2 Test

In order to test the functionality of our NFTMarketplace contract, we write up a test script based on different scenarios. The test script is written by JavaScript under the Mocha testing framework and deployed under the local network created by the truffle development environment. By default, truffle development generates 10 user accounts with private keys in the local network. We can simply assign an account's address to a constant by doing const user = accounts[N], where N is the number of which account you wish to assign. Moreover, we can direct the testing network to the designated port in the Truffle config file and add the network RPC URL to the Metamask for observing the balance of the account we test. The structure of test script is as follows:

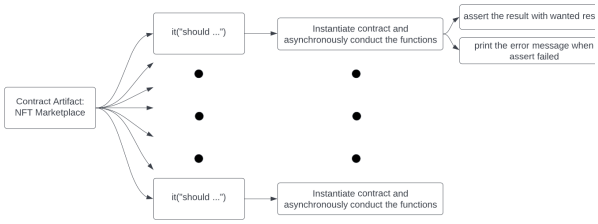

Figure 1. The structure of the test script.

Next, we break down each test scenario and see the performance of our contract. First, we start the truffle development environment and the accounts and the network will be automatically generated. We will primarily use two accounts:

1. 0xfe9f2e2e0d513f02cf37f7bc19215d43820846c4
2. 0x78baee156c3a1e42ee9dd38644bb1e64aadb2f6f

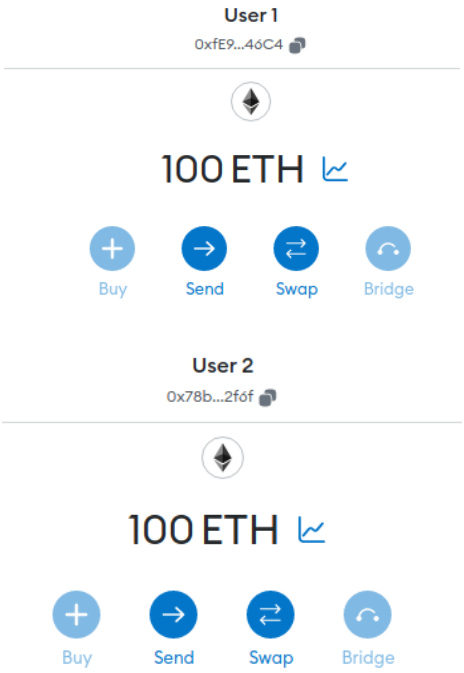

Figure 2. The interface of two users.

For our first test case, we test if our smart contract will create a new NFT. We use createNFT to create a NFT with name: 'banana' and description: 'It is banana'. Then we assert the reading value of the NFT result to the string of what we input. The assert result passed which means we successfully created a NFT.

For the second test case, we test if we can transfer the ownership of our NFT by asserting that the NFT ownership is owned by the account we transfer to.

For the third test, we create a new NFT and list the NFT for sale using the listNFTForSale function with sale price. Then assert that the NFT is now listed and the sale price matches the input value.

For the fourth test, we create a new NFT and list the NFT for sale as we did in the previous test. Then we use the function removeNFTFromSale to remove the NFT from the sale list. For obtaining the result,

the sale status of the NFT can be tested by asserting that the NFT is no longer listed for sale.

For the fifth test, we test if the contract can execute a successful NFT. First, we create two accounts: user 1 and user 2. The user 1 plays the role of seller and user 2 plays the sole of buyer. Then we use user 1 to create a new NFT and list it into the sale list with a price. Once the NFT has been created, we let the buyer account to purchase the NFT. In this purchasing process, we have compared several metrics to verify if the buyer has successfully owned the NFT by costing him the correct amount of wei. First we record the buyer and seller balance before the buyer calls the purchase function. After purchase is finished, we record the buyer and seller balance again and the ownership of the NFT. For testing if the ownership has been transferred, we assert that the NFT belongs to the buyer after purchase. For testing if the buyer spent the correct amount of wei to purchase the NFT, we assert that the buyer's balance is equal to the expected value of the balance, which is equal to (buyer balance before purchase - NFT's price - gas fee).

For the last test, we test if the contract can correctly handle an unsuccessful NFT purchase. We have the same test setup as the previous one but the buyer will initiate a purchase with the incorrect amount of payment (in our case, payment = sale price - 1). Then, we assert that the NFT still belongs to the seller and the error message should contain the expected revert reason. Moreover, we compare the balance of the buyer before purchase and after the failure purchase.

1.3 Problem & Solution

During the purchase process from user 1 to user 2, we found an error: "VM Exception while processing transaction: revert ERC1155: caller is not token owner or approved" and the purchase was canceled.

It happens because we want to transfer the tokens on behalf of the seller's wallet address. but the seller hasn't approved our wallet address to be an operator.The solution is to let the seller of the address call : "setApprovalForAll". This function lets us as the contract owner be an operator to transfer the tokens on behalf of the sellers.

1.4 Results

The contract passes through all our test cases and the remaining amount of balance also meets our expectations. The detailed result will be in the showcase section.

2. Showcase

When we start the Truffle development environment, the account details are as follows:

```
Accounts:
(0) 0xfe9f2e2e0d513f02cf37f7bc19215d43820846c4
(1) 0x78baee156c3a1e42ee9dd38644bb1e64aadb2f6f
(2) 0x2da7bcacd121387b729c94354a9ba1d951fa1577
(3) 0x58593b6433890759e3c0be8b310cfb3b4a9726f6
(4) 0xbe8c9c8bc09d01b98d82fbd1e4e52c9e75167e4c
(5) 0xa630f726ee6ba544c3d9ced6914b2106607ba0ad
(6) 0xbd205ea4767548060aedbbdb0e8ac9fb86baadb7
(7) 0x1212da7825ef044ebda82800781341b7bd56bea8
(8) 0xfe54b1d8787499728fd94e0039eb5f7304feb4cf
(9) 0xcc239c27664d3c7eed6e0e2ac85a4793ce23d724

Private Keys:
(0) c30553639257660c431df00ad8f10abb1aa1c59b09beb797addb293d7340dd8f
(1) 68b701ca0cc390c27d5381c56ef895232e19be7374414c343ce71ecd7ae56269
(2) 78e46aa66436f1ea8f31ba14a4e893629fef2a8d6811d0911ef0f2c3923ef09e
(3) adee428848fb36d0e3d7f4ba2a6c4331b1d96976a90f9589fc4dd512d98a09f9
(4) 6c5484ee62653c268f251ee96f91aca850928ee4c0f6aac453f083ef0f213ca0
(5) 7ff9b5c314298e81b9dd8418a6b7d2d770d3f071bb2c8bc64d37023ce32a34f9
(6) 1b1447d8b11b888d9fbb3ab8ee668de67d7ef236f9f95c6bd9e2ef490b0c14e7
(7) 5057b39881707401db03a5a5ab5a8e78219db02646701efd21c347260d0e75fd
(8) dc74bcf590196cb4d39f9107f2538254462b014f71b9589ada448ac45eec8e90
(9) bf797a7c638cd7f11c81d72c24bfd567651fcddd21cd59ca9b643a9f975d8a4d
```

Figure 3. Truffle development environment.

Then we execute the test command in the truffle development environment, and the result is as follows:

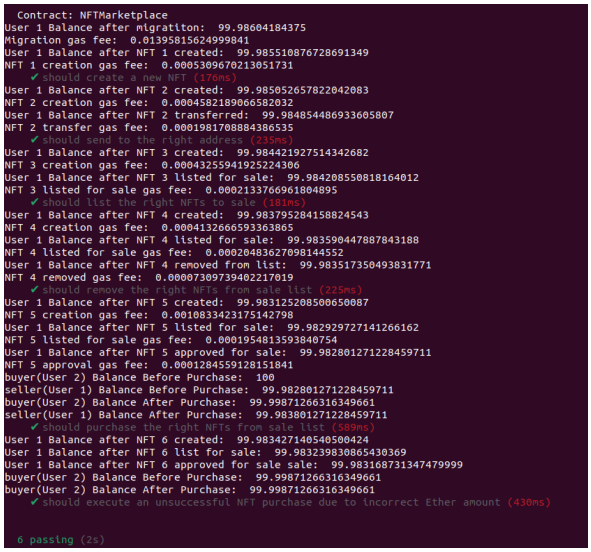

Figure 4. The Truffle test result.

For easier to observe the remaining balance of account, we convert the balance unit wei to the ether. We also print out the operation's gas fee for our user to better observe how much they cost during the trading. From all the tests, we can see that during deployment of the smart contract and creation of NFT, some amount of the gas fee has been charged from the user. Also, every operation interacted with the network will also be charged. Nevertheless, each test has been successfully passed and checked. During the purchase function verification, we can observe that the user 2, who hasn't done any operation, remains the 100 eth initially given by the truffle development. After successful purchase, the seller gains the exact value of corresponding NFT sale price and the buyer is deducted by the price and the gas fee. During the false purchase verification, if the purchase is failed, the ownership of NFT has not changed and the buyer is also uncharged from sale price and the gas fee.

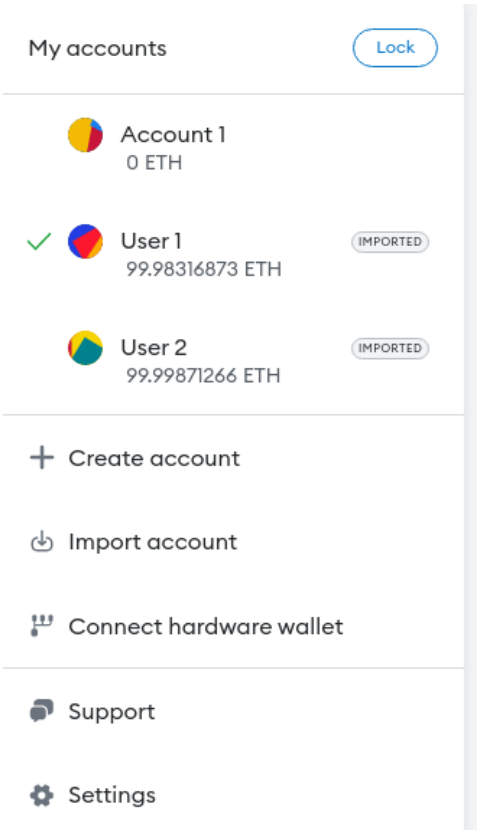

Figure 5. Account details in MetaMask.

The MetaMask also shows the correct remaining balance of two accounts.

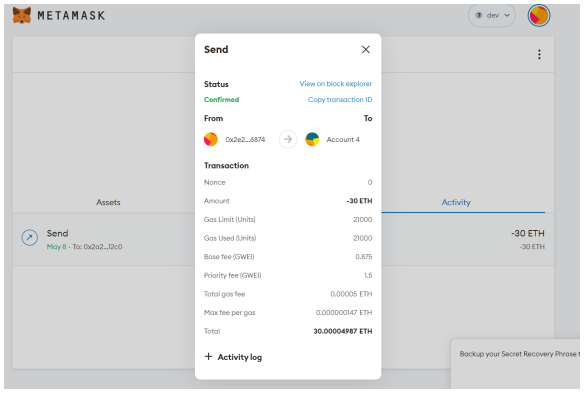

Figure 6. An example of a transaction.

Here is an example of a transaction between two local accounts. The users can complete all transactions and check the status through Metamask.

3. Conclusion

In this project, we developed an NFT marketplace smart contract.We will summarize the four aspects of the requirement:

In terms of Setting Up the Development Environment,we installed the necessary tools and created a new Truffle project.

In terms of Designing the NFT Marketplace Smart Contract,we define a structure "NFT", two public mappings: nfts and nftCreators, and some events, such as NFTCreated(), NFTListed(), NFTUnlisted() and NFTPurchased(), etc. To meet the requirements of users to mint, buy, sell and trade unique digital assets represented by NFT.

In terms of Deploying the NFT Marketplace,we used "Ganache" and "MetaMask" to complete the deployment.

In terms of Test Cases,we have tested "create a new NFT", "send to the right address", "list the right NFTs to sale", "remove the NFTs from sale list", "purchase the right NFTs from sale list" and "execute an unsuccessful NFT purchase" to meet user requirements in actual operation.

The final results show that the NFT market smart contract we developed can well allow users to mint, buy, sell, and trade unique digital assets, and has an intuitive user interface and good market security and usability.

Contributions:
Zhan Shu finishes the coding of the contract and the contract part of the report.
Yuan Dou finishes the contract and procedure part of the report writing.
Yufan Zhang finishes the problem and solution part and the summary and modification of the report.
Zhuofei Lin & Qiao Zhang responsible for coding of contract and test cases and composing the test and showcase part of report writing.

4. Reference

[1] William Dawsey, “ERC-721 VS ERC-1155: WHICH IS BETTER FOR NFT MARKETPLACES?”, https://www.chetu.com/blogs/blockchain/erc-721-vs-erc-1155.php