NFT Marketplace Smart Contract

Han Liu (hl3608), Yuxuan Chen (yc4144), Danny Hou (dh3034), Jiafu Chen,(jc5874), Yifan Wang(yw3914)
GitHub Link: https://github.com/liuhanalice/NFT_data_warehouse.git

Abstract

This project explores the development of a decentralized NFT marketplace utilizing Ethereum smart contracts. This project was developed with Truffle and deployed on the Sepolia test network. The marketplace enables users to easily create, list, hold and eventually sell unique digital assets in a secure and transparent environment. Our implementation employs Truffle for smart contract development, testing, and deployment, while the front end allows users to interact with the marketplace using their MetaMask wallets for a secure and user-friendly experience. Overall, this project provides the potential of blockchain technology to revolutionize the digital asset landscape and foster a thriving ecosystem for NFTs.

1. Introduction:

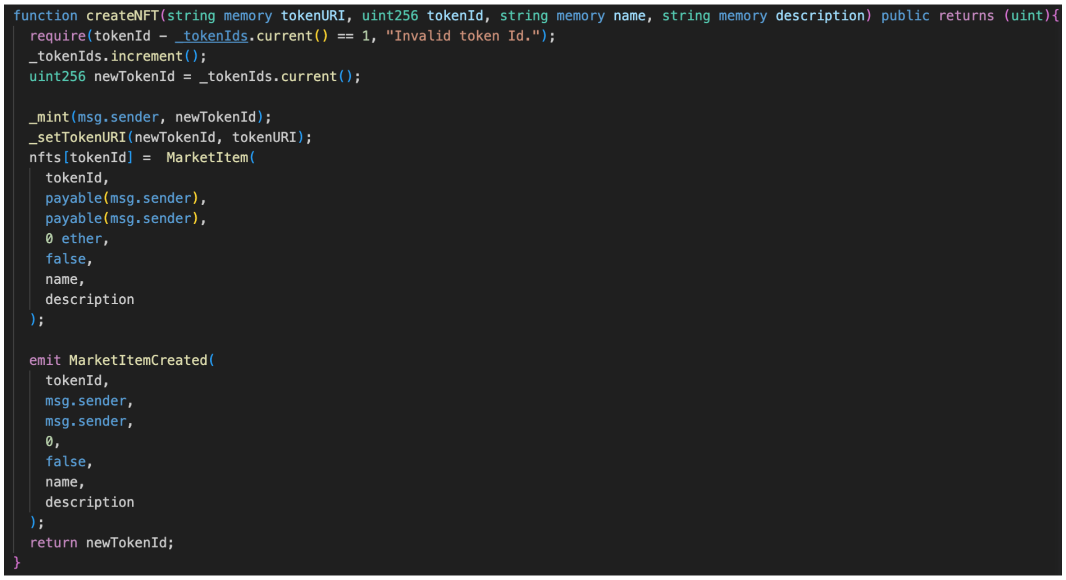
In recent years, non-fungible tokens (NFTs) have become a popular and innovative way to represent unique digital assets. However, the majority of existing marketplaces are still predominantly centralized, causing some potential issues related to transparency, trust, and accessibility. To address these challenges, our project focuses on developing an NFT marketplace, *LION NFT Marketplace*, using Ethereum smart contracts adhering to ERC-721 standards and Truffle with a user-friendly frontend that interacts with the MetaMask wallet. In addition, we used Ganache as a local test network to simulate blockchain interactions and validate the functionality of our smart contracts prior to deployment.
Throughout this report, we will demonstrate the implementation and functionality of our decentralized NFT marketplace along with its main features. Additionally, we will explore more potential improvements, making the platform more robust and secure.

2. Project Design:

The contract allows users to create and sell NFTs, list them for sale, remove them from sale, and transfer ownership. And the contract is protected by Reentrancy Guard to prevent reentrancy attacks. The marketplace supports a fixed listing price set by the owner of the contract, and the owner receives this small commission fee for each sale. The main components of the contract include but not limited to:

- ‘MarketItem’ struct: This defines the properties of an NFT, including the token ID, seller, owner, price, and whether it is listed for sale.
- ‘_tokenIds’ and ‘_itemsOnSale’ Counters: These are used to keep track of the total number of NFTs created and the number of NFTs currently listed for sale, respectively.
- updateListingPrice and getListingPrice functions: These two functions can update the commission or return the commission of the contract.

- createNFT function



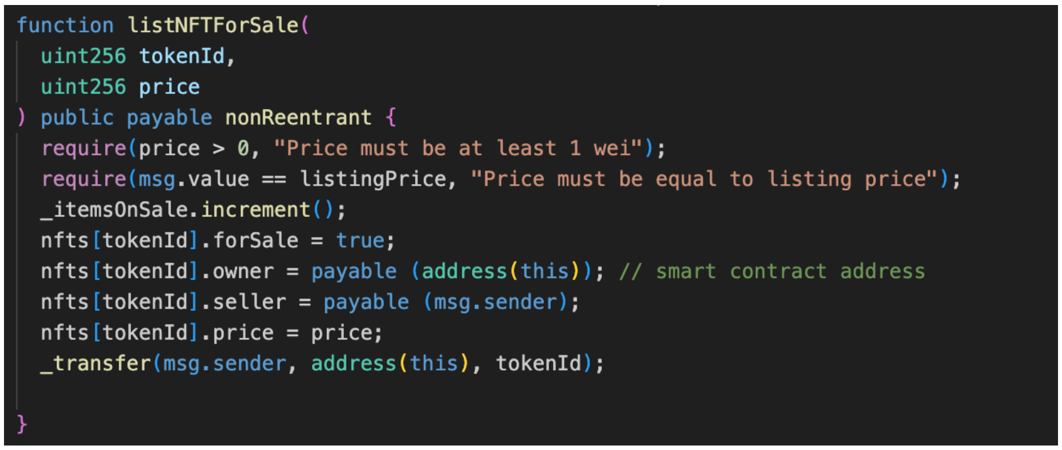
```
function createNFT(string memory tokenURI, uint256 tokenId, string memory name, string memory description) public returns (uint){
  require(tokenId - _tokenIds.current() == 1, "Invalid token Id.");
  _tokenIds.increment();
  uint256 newTokenId = _tokenIds.current();

  _mint(msg.sender, newTokenId);
  _setTokenURI(newTokenId, tokenURI);
  nfts[tokenId] =  MarketItem(
    tokenId,
    payable(msg.sender),
    payable(msg.sender),
    0 ether,
    false,
    name,
    description
  );

  emit MarketItemCreated(
    tokenId,
    msg.sender,
    msg.sender,
    0,
    false,
    name,
    description
  );
  return newTokenId;
}
```

This is one of the main functions within our contract. It first checks if the tokenId is valid and matches the next sequential tokenId by checking the difference between the two. This ensures that the tokenIDs are being issued in sequential order. If valid, the function then increments the _tokenIds counter to generate a new, unique token ID. The built-in '_mint' function creates a new Token which newTokenID is the parameter to represent the unique identifier and the msg.sender is where the token is coming from. Along with minting NFT, the function also stores the associated data in the MarketItem structure which a single value essentially holds other values as a map.

- listNFTForSale function

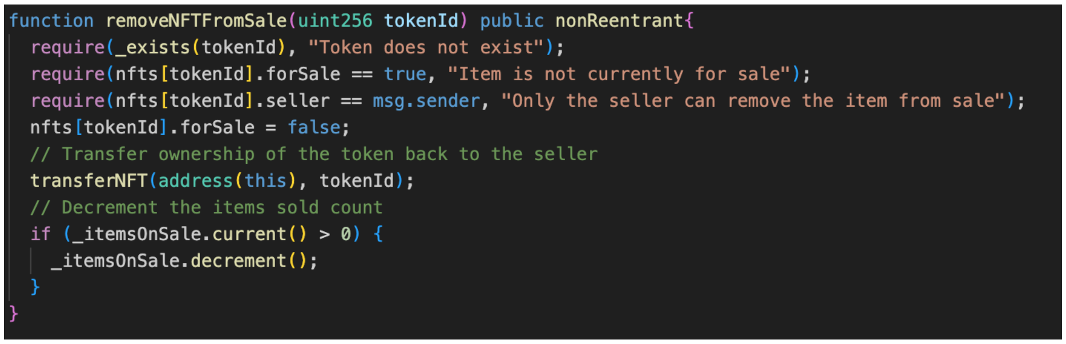


```
function listNFTForSale(
  uint256 tokenId,
  uint256 price
) public payable nonReentrant {
  require(price > 0, "Price must be at least 1 wei");
  require(msg.value == listingPrice, "Price must be equal to listing price");
  _itemsOnSale.increment();
  nfts[tokenId].forSale = true;
  nfts[tokenId].owner = payable (address(this)); // smart contract address
  nfts[tokenId].seller = payable (msg.sender);
  nfts[tokenId].price = price;
  _transfer(msg.sender, address(this), tokenId);

}
```

The function first checks the requirement that if the price being passed is greater than zero and if the amount of ether being sent in the transaction is equal to the listing price. If both of the conditions are met, the function increments the '_itemsOnSale' counter to indicate that a new NFT is now available for sale. Lastly, it transfers the NFT's ownership.

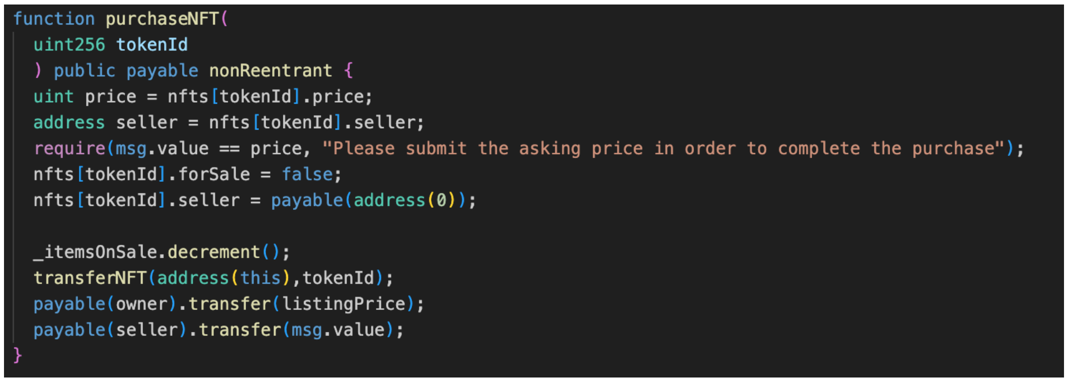
- removeNFTFromSale function



```
function removeNFTFromSale(uint256 tokenId) public nonReentrant{
  require(_exists(tokenId), "Token does not exist");
  require(nfts[tokenId].forSale == true, "Item is not currently for sale");
  require(nfts[tokenId].seller == msg.sender, "Only the seller can remove the item from sale");
  nfts[tokenId].forSale = false;
  // Transfer ownership of the token back to the seller
  transferNFT(address(this), tokenId);
  // Decrement the items sold count
  if (_itemsOnSale.current() > 0) {
    _itemsOnSale.decrement();
  }
}
```

This function takes in a token id and removes an NFT from sale. It first checks that if the token id exists, then checks that the relevant NFT is currently for sale, and then checks that if the message sender is the seller. If all conditions are met, we set the ‘forSale’ field to be False, transfer the ownership of the token back to the seller and decrement _itemsOnSale.

- purchaseNFT function

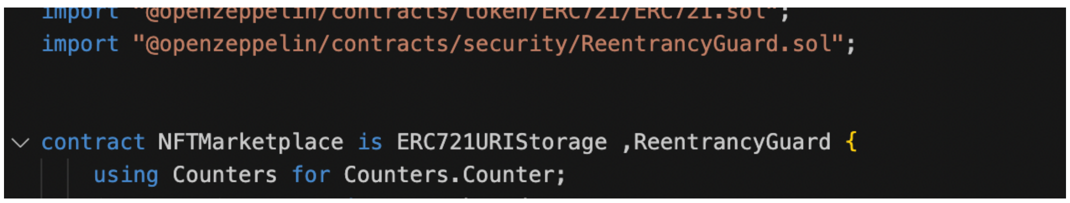


```
function purchaseNFT(
  uint256 tokenId
  ) public payable nonReentrant {
  uint price = nfts[tokenId].price;
  address seller = nfts[tokenId].seller;
  require(msg.value == price, "Please submit the asking price in order to complete the purchase");
  nfts[tokenId].forSale = false;
  nfts[tokenId].seller = payable(address(0));

  _itemsOnSale.decrement();
  transferNFT(address(this),tokenId);
  payable(owner).transfer(listingPrice);
  payable(seller).transfer(msg.value);
}
```

This function allows a user to purchase an NFT that is listed for sale. The function will first check the buyer submitted asking price of the NFT. Then, the function transfers the ownership of the NFT to the buyer and lets the seller receive the sale price; the contract owner receives a commission fee.

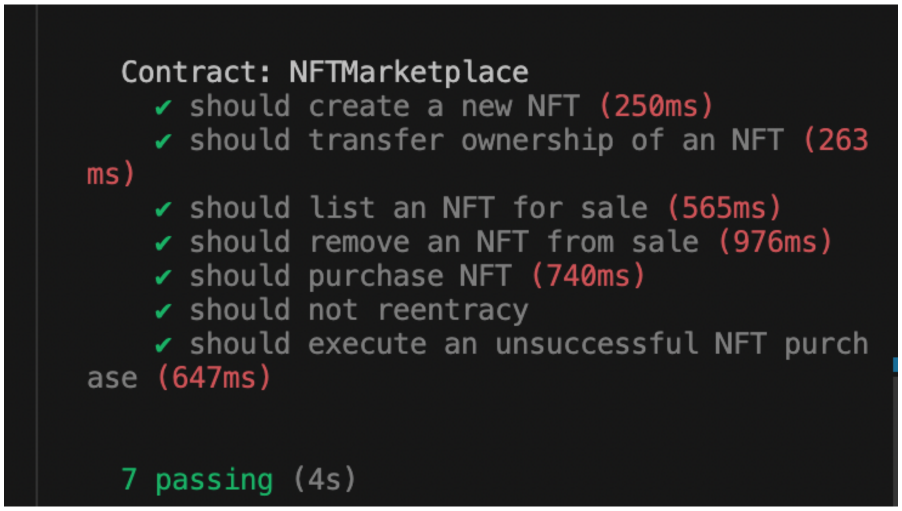
- **Security**



```
  import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
  import "@openzeppelin/contracts/security/ReentrancyGuard.sol";


contract NFTMarketplace is ERC721URIStorage ,ReentrancyGuard {
    using Counters for Counters.Counter;
```

- **Test cases:**
  1. Test that the smart contract can create a new NFT
  2. Test that the smart contract can transfer ownership of an NFT
  3. Test that the smart contract can list an NFT for sale
  4. Test that the smart contract can remove an NFT from sale
  5. Test that the smart contract can execute a successful NFT purchase
  6. Test that the smart contract can execute an unsuccessful NFT purchase



```
  Contract: NFTMarketplace
    ✔ should create a new NFT (250ms)
    ✔ should transfer ownership of an NFT (263
ms)
    ✔ should list an NFT for sale (565ms)
    ✔ should remove an NFT from sale (976ms)
    ✔ should purchase NFT (740ms)
    ✔ should not reentracy
    ✔ should execute an unsuccessful NFT purch
ase (647ms)


  7 passing (4s)
```

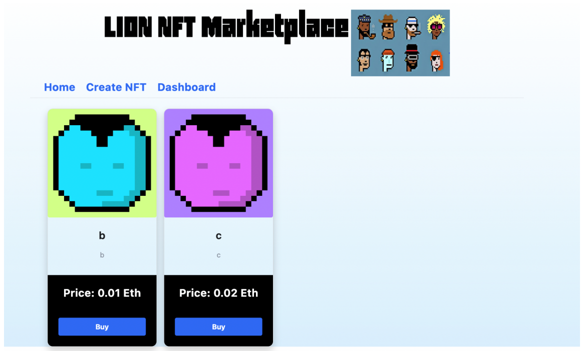
For detailed implementation, please refer to our GitHub repo.

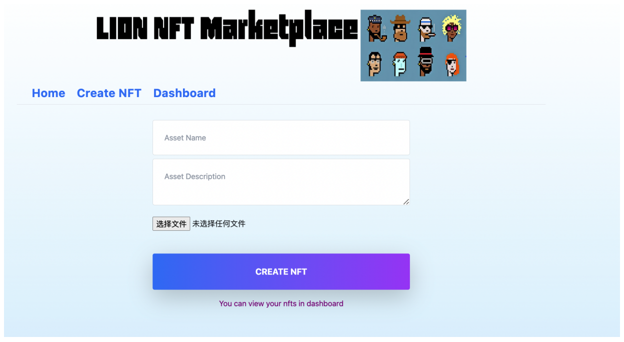
3. Showcase:

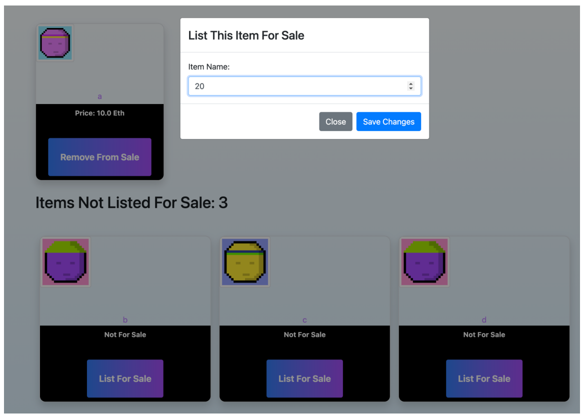
After successfully implementing the required dependencies, our project can then be opened through 'http://localhost:3000/.' There are mainly 3 pages inside our NFT marketplace.
The first page is the Home page which shows all the listing items that are currently for sale from all the platforms. Users can purchase items freely on this page as long as they have sufficient funds in their wallets. After buying successfully, the nft will add to the dashboard of this buyer and remove from the home page of all the users.

The second page is the Create NFT page. After importing the image from local, users can then add a name and description to upload the NFT to the website. After successful creation, the item can be viewed on the Dashboard page.

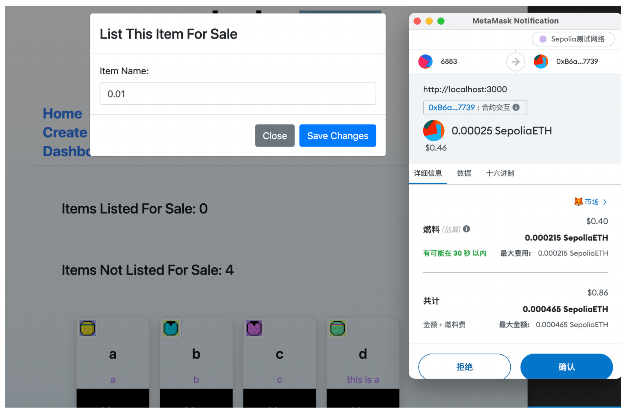
The third is the Dashboard page. On our dashboard page, all the NFTs that the user created will be displayed here. The top is the items that the users currently listed, whereas the bottom is the items in his/her inventory. The user can have the option to remove the listing items or list items for sale. The image has a zoom effect when the mouse hovers over them. Moreover, when users click the save changes button, it will connect to the MetaMask to purchase.






List This Item For Sale

Item Name: 20


List This Item For Sale

Item Name: 0.01

MetaMask Notification

Sepolia测试网络

6883 → 0xB6a...7739

http://localhost:3000

0xB6a...7739 : 合约交互

0.00025 SepoliaETH
$0.46

详细信息 数据 十六进制

市场 >

燃料 (估算) $0.40
0.000215 SepoliaETH
有可能在 30 秒 以内 最大费用: 0.000215 SepoliaETH

共计 $0.86
0.000465 SepoliaETH
金额 + 燃料费 最大金额: 0.000465 SepoliaETH

4. Conclusion:

Our project has successfully developed an NFT marketplace smart contract, complete with all its essential components. It not only allows users to mint, purchase, sell, and exchange unique digital assets and prevent attacks, but its reliability is also demonstrated by passing all required test cases. In addition to the smart contract, we have also designed an intuitive and user-friendly frontend interface to provide an enhanced browsing and transaction platform. The interface allows users to easily engage in various activities such as browsing, listing, purchasing, and selling digital assets. In the future, we seek to continuously improve our platform by incorporating additional features and expanding its capabilities. One key area of focus is enabling users to add a variety of digital assets beyond single images. Some other possible features may include advanced filtering and sorting, creating and managing collections, tracking transaction history, etc.

Contribution:
Han Liu: smart contract, frontend; Yuxuan Chen: smart contract; Danny Hou: smart contract, test cases; Jiafu Chen: frontend, final report; Yifan wang: frontend, test network

References:

[1] Dabit, Nader. "How to Build a Full Stack NFT Marketplace - V2 (2022)." *DEV Community*, DEV Community, 11 Mar. 2022, https://dev.to/edge-and-node/building-scalable-full-stack-apps-on-ethereum-with-polygon-2cfb.
[2] Dawsey, William. "ERC-721 vs ERC-1155: Which Is Better for Nfts?" *Chetu*, https://www.chetu.com/blogs/blockchain/erc-721-vs-erc-1155.php.
[3] Vester CORE , John. "Let's Build an End-to-End NFT Project Using Truffle Suite - DZone." *Dzone.com*, 23 Nov. 2022, https://dzone.com/articles/lets-build-an-end-to-end-nft-project-using-truffle.
[4] McCubbin, Gregory. "The Complete Blockchain Developer Toolkit for 2019 & Beyond." *Dapp University*, https://www.dappuniversity.com/articles/blockchain-developer-toolkit.
[5] *Truffle Suite*, https://trufflesuite.com/.