Ether Painting: an NFT painting displayer based on blockchain

Xuechen Zhou (xz3153) Danling Wei (dw3033) Yongjie Fu (yf2578) Yu Wu (yw3748) Tong Wu (tw2906)

Abstract

In this paper, we will introduce an NFT painting displayer called Ether Painting. It provides basic functions of creating, listing, selling, and buying Non-Fungible Tokens (NFTs) representing paintings. The contract is built using the Solidity programming language and based on the ERC721 standard. The OpenZeppelin library provides secure and tested building blocks for Ethereum applications.

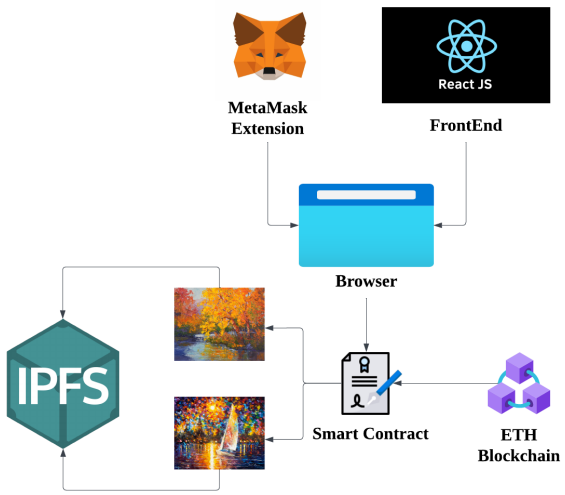

Figure 1. The framework of the Ether Painting.

1. Introduction

1.1. Motivation

For a typical painting sharing application, users first get access to the app on a mobile device or a laptop, and the mobile device communicates with a web server where all the image data is stored. Whenever people select an image or an image gallery to view, a request is made to retrieve the corresponding image files from the server to the user's mobile device, and this is how most image streaming apps are built.

For traditional painting download app, there are some drawbacks. Firstly, if you discover some artists or photographers that you believe have the potential in the future, there is not a convenient way to bet on her/his future success. To support artists, fans must buy physical prints, merchandise, or go to exhibitions. However, artists only earn money from the platform based on views, not fan support, which can distance them from their most devoted fans.

A non-fungible token (NFT) is a unique digital identifier that is recorded on a blockchain, and is used to certify ownership and authenticity. It cannot be copied, substituted, or subdivided. Because of that, we make use of NFT to create an NFT painting displayer based on the blockchain, which leverages fandom in a decentralized way with different revenue model. As a result, fans can give their artists direct support. Artists can earn the corresponding royalty fee for selling their paintings.

1.2. Architecture

The architecture of our project consists of two parts, the front-end part and the back-end part, which is illustrated in Fig 1. In the front-end, we develop a blockchain-based NFT painting display application. We connect it to Metamask wallet extension for the functions like buying and reselling tokens. And then, the operation information and data are delivered to the back-end part. In the back-end segment, we utilize the functions incorporated within the smart contract deployed on the Ethereum blockchain to manage front-end requests and data processing. Subsequently, relevant data, such as token details and image files, are stored in a decentralized file storage system known as IPFS.

The core the project is one smart contract with two main uses: showing the collection of image NFTs and a place where users can buy and sell NFTs.

2. Technical Detail

2.1. Frontend

2.1.1. Introduction

The frontend is constructed using modern React hooks and functional components, employing useState and useEffect hooks to manage state and side effects within components. The ethers.js library is utilized to interact with the Ethereum blockchain and the smart scontract.

2.1.2. Components

In the frontend section, we utilized the React framework to develop a blockchain-based NFT gallery application. This application comprises the following components:

- App Component - The primary component responsible for connecting to the Metamask wallet, fetching the smart contract instance, and rendering the main application layout and navigation.
- MyImages Component - Enables users to view their owned NFTs, fetch the list using the getMyNotListedNFT function from the smart contract, and display them in a grid layout. Users can set a resale price and list their NFTs for sale with the resellNFT function
- MySellings Component - Showcases the user's listed NFTs for sale, fetches the list using the getMyListedNFT function from the smart contract, and displays them in a grid layout. Users can cancel their NFT sales with the cancelSellNFT function.
- Home Component - Displays all available NFTs for sale in the market, fetches the list of unsold NFTs using the getUnsoldNFT function from the smart contract, and presents them in a grid layout. Users can buy an NFT by clicking the "Buy" button, which triggers the buyNFT function from the smart contract
- CreateNFT Component - Allows users to create and mint new NFTs. Users can upload an image file and input a title and description for the NFT. The image file is uploaded to IPFS, and the resulting IPFS hash is used as the token URI when minting the new NFT using the createNFT function from the smart contract.

2.1.3. Responsive Design

Designed to be responsive and mobile-friendly, the frontend uses React-Bootstrap components such as Card, Row, and Col to create a responsive grid layout for displaying NFTs, ensuring adaptability to different screen sizes and orientations.

2.1.4. Connection to metamask

The frontend connects to the user's MetaMask wallet using the ethers.js library. Upon clicking the "Connect Wallet" button in the navigation bar, the application requests access to the user's Ethereum account, retrieves the user's public address, and establishes a connection to the smart contract upon approval

2.1.5. Error Handling

To ensure a seamless user experience, the frontend incorporates error handling and user feedback for various actions. For instance, when a user tries to create an NFT without providing an image url, an alert is displayed, prompting the user to complete the required fields. Similarly, if an error occurs during the listing or delisting of an NFT, the frontend displays an alert with an error message to inform the user of the issue.

2.2. Backend

2.2.1. Smart Contract

To implement the data processing functions we wrote a solidity smart contract. In this step, we used the solidity of version 0.8.4 and imported library OpenZeppelin, which provides secure and tested building blocks for Ethereum applications. It is based on the ERC721 standard, which defines the interface for NFTs on the Ethereum blockchain. The smart contract we wrote allows the creation, listing, selling and buying of Non-Fungible Tokens (NFTs) that represent paintings.

The *PaintingNFTContract.sol* is designed as a contract owned by a specific address we call the operator. The contract has a tax value that is defined during contract creation and can be changed by the operator. The tax value is applied to all purchases made on the contract, and the tax is transferred to the operator's address.

In the contract, we used a struct to define the Painting object. Each Painting has a unique token ID, an owner address, a price, and a URI (Uniform Resource Identifier) that points to the painting's metadata. The contract uses events to emit BoughtEvent, ListedEvent, and CancelListedEvent whenever a Painting is bought, listed or delisted, respectively.

The contract includes several functions, which are createNFT, listNFT, cancelListNFT, buyNFT, getUnsoldNFT, and getMyNFT.

- createNFT
  The createNFT function allows the owner to create a new NFT by minting a new ERC721 token and adding a new Painting object to the paintings array.
- listNFT

The listNFT function allows the owner of a Painting to list it for sale by setting its price and transferring ownership of the Painting to the contract.

- cancelListNFT
  The cancelListNFT function allows the owner of a listed Painting to remove the painting from the sale list which is by transferring ownership back to the owner.
- buyNFT
  The buyNFT function allows a buyer to send the required amount of ETH to the contract to purchase a listed Painting. The price of the Painting is transferred to the Painting's original owner and the tax value is transferred to the contract operator and the buyer gets ownership.
- getUnsoldNFT
  Anyone can obtain a list of all unsold Paintings currently held under the contract using the getUnsoldNFT function. An owner of a Painting may obtain a list of all of their own Paintings by using the getMyNFT function.
- getMyNotListedNFT
  This function can get all the NFTs that are created but not listed on the market. It is like a repository of NFTs for users to check all the NFTs they have and determine which one to be listed on market.
- getMyListedNFT
  This function will get all the NFTs that are listed by current user. It will help users to manage their sales and determine which one to be canceled in market and return to repository.

The contract is built to be safe and guard against unauthorized access and data tampering for security reasons. To limit access to important functions and data, the contract makes use of the Ownable and Access Control aspects of the OpenZeppelin library.

The contract also includes several requirements and checks to ensure that transactions are valid and that inputs are correct. For example, the contract checks that the price of a listed Painting is greater than zero, that the buyer has sent the correct amount of ETH to purchase the Painting, and that only the owner of a listed Painting can cancel the listing.

2.2.2. Contract Test

In order to avoid some potential errors produced while deploying the contract or after deployment, we also wrote the test script for security.

The tests were conducted using *Node.js* version is 14.17.5, and the following libraries and frameworks *Mocha* version 9.1.3, *Chai* version 4.3.4, *ethers.js* version 5.5.5

We deployed a new instance of the contract and a number of Signers to represent various user addresses before each test. An operator address and a tax value needed in several tests were initialized in the contract.

The Contract Deployment Test section checked that the contract was deployed correctly and that the name, symbol, tax, and operator values were set correctly. This was done by comparing the expected values with the values retrieved from the contract using the Chai *expect()* function.

The NFT Creation Test section tested the creation of a new NFT and the ownership of that NFT. This was completed by calling the *createNFT()* function with one of the Signers and checking the owner of the new NFT was set correctly.

The NFT Transaction Test section tested various functions related to buying and selling NFTs. We tested *listNFT()*, *cancelListNFT()*, and *buyNFT()* functions to ensure that they were correct and that the ownership of the NFT was transferred as expected. In addition, we also tested *getUnsoldNFT()* and *getMyNFT()* functions to ensure that they returned the correct list of NFTs.

Overall, the contract can be deployed successfully. And the final test result is shown in figure 2.

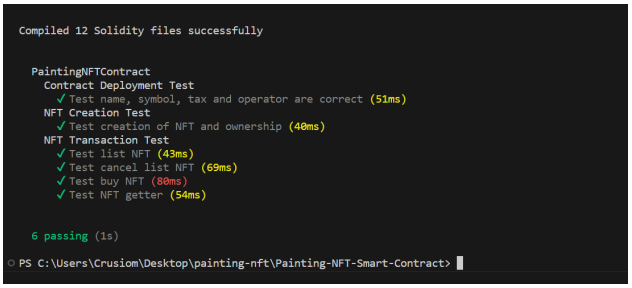

Figure 2. The test result of the contract

3. Showcase

The frontend provides various functionalities for users to engage with the smart contract. In the Home component, users can view available NFTs for sale and purchase them by clicking the "Buy NFT" button. The buyNFT function sends the required ETH amount to the smart contract and transfers NFT ownership to the buyer.

In the CreateNFT component, users can upload an image and provide a description to create a new NFT. Upon image upload, the frontend generates a unique URI based on user input and timestamp. The frontend then calls the createNFT function from the smart contract to mint the new NFT with the generated URI.

The MyImages component displays the NFTs owned by

the connected user. Users can list their NFTs for sale by specifying a price and clicking the ”Sell NFT” button, which calls the listNFT function from the smart contract. Users can also delist their NFTs by clicking the ”Cancel Sell” button, which calls the cancelListNFT function from the smart contract. Furthermore, users can view and manage their listed NFTs, including updating the sale price.

Fig 3 to Fig 7 show the interface of our application:

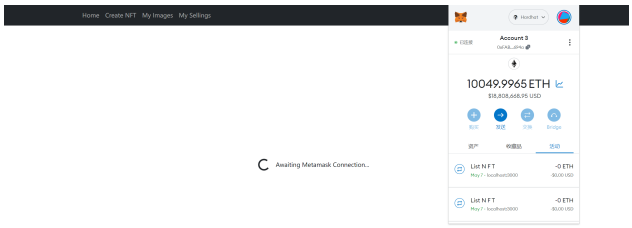

Figure 3. Entering the page for the first time, ready to log in

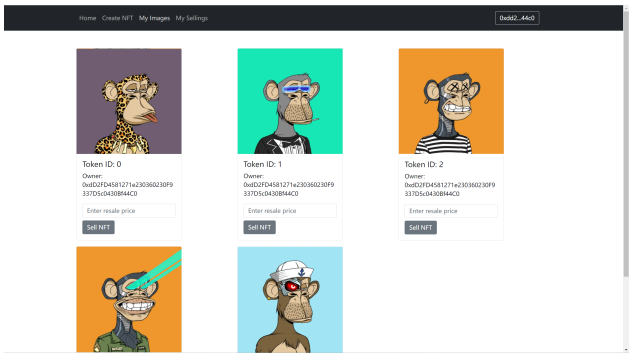

Figure 4. My own unsold NFTs

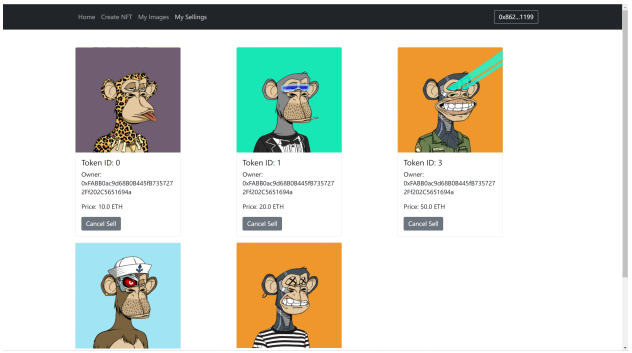

Figure 5. The NFT in selling

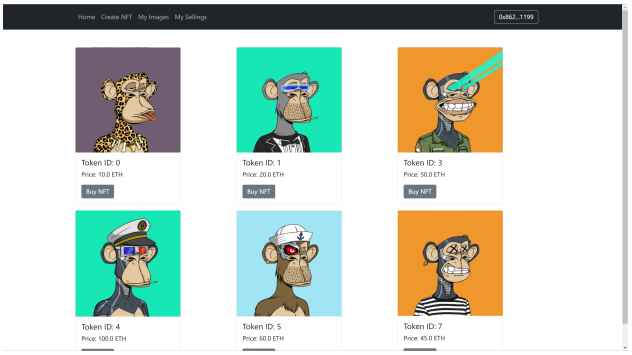

Figure 6. All NFTs that users are selling

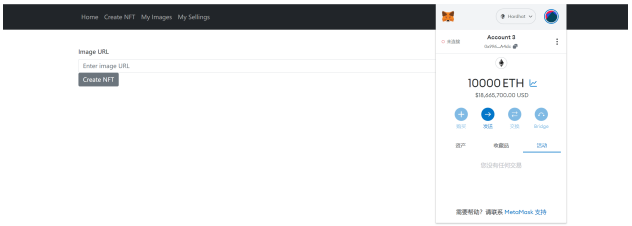

Figure 7. Create NFTs

—

4. Conclusion

In this project, we develop a full-stack painting displayer application, which consists of front-end and back-end parts.

We first implement the contract and use IPFS to store the images files. And we figure out the test file to test our functions and deploy the small contract on the blockchain. In the end, we use react to build the client side to interact with users.

This platform is robust and decentralized, which supports search, purchase and resell the paintings from the painters. To sum up, Ether painting is a successful paintings displayer and NFT trading platform based on blockchain

Reference

Link to our project's Github: https://github.com/Crusiom/Painting-NFT-Smart-Contract

Link to our project's showcase video: https://www.youtube.com/watch?v=InthOm-4mQ0