NFT Market place with custom currency token support using Polygon

Solution details

- The NFT Marketplace will be built using ReactJS for the user interface and interaction.
- The smart contract functionality will be implemented using Solidity and Hardhat for development and Chai for testing.
- The Ethereum network interaction will be handled using ethers.js and web3.js libraries.
- MetaMask will be used as the only form of connecting to a wallet.
- The NFTs and custom tokens will be deployed and managed on the Polygon network using the Mumbai TestNet.

List of Technologies with version and functionality

1. ReactJS: Version ^18.2.0 - Used to create the user interface for the NFT Marketplace.
2. Tailwind CSS Version 3.0 - Used for the CSS of the user interface.
3. Typescript Version ^4.9.5 - Programming language for writing the user interface
4. Solidity: Version ^0.8.0 - Programming language for developing smart contracts.
5. Hardhat: Version ^2.16.1 - Development environment for compiling, testing, and deploying smart contracts.
6. ethers.js: Version ^6.6.3 - JavaScript library for interacting with Ethereum networks and contracts.
7. web3.js: Version ^4.0.3 - JavaScript library for interacting with Ethereum networks and contracts.
8. Chai: Version 4.3.7 - Testing framework for writing unit tests for Solidity smart contracts.
9. Polygon: Utilizing the Polygon network and Mumbai TestNet for deploying and testing the NFT Marketplace.
10. OpenZepplin: for utilizing useful contracts such as Counters for generating unique ID's, ERC721 for NFTs, Ownable contract which provides a basic access control mechanism

Development environment setup

- Install Node.js and NPM (Node Package Manager).
- Create a new project using create-react-app for the client
- Install the required dependencies, such as ethers.js, web3.js, and other necessary packages.
- Set up the Hardhat development environment for Solidity smart contract development and testing.
- Configure the connection to the Polygon Mumbai TestNet in the Hardhat network configuration.

Functionalities (Don’t include code snippets or function names)

1. User Login
    - The user will click on a “Connect to Wallet” button which will open up MetaMask, using window.ethereum to check for MetaMask existence and ethers library to connect to MetaMask.
    - If MetaMask does not exist in the browser, the user will receive an error message on the screen notifying the user that he should have MetaMask on the screen. using a try-and-catch statement to receive MetaMask error message.
    - The user will enter his MetaMask credentials, and MetaMask will deal with handling the verification and validation of the credentials.
    - If the credentials entered are correct, then the User’s MetaMask wallet will be connected to the marketplace. The user information like his address and other account details will be received from MetaMask by ethers.
    - If the credentials are incorrect, then the user will receive a message.
2. Custom tokens
    - Deploying the Solidity smart contract following the ERC721 standard, which supports the creation of Custom tokens.
    - The smart contract will define the properties and behaviors of the custom tokens, such as the name, symbol.
    - Implement the functionality to mint new tokens. This can be done by calling a specific function in the smart contract, which will increase the token supply and assign the newly minted tokens to specific addresses.
3. How will the NFT marketplace support custom token
    - The marketplace includes features to enable users to list and trade their custom tokens.
    - Users are able to specify the price of their tokens and cancel their listings.
    - Technically speaking, a user can interact with the UI and mint a token, using ethers library to interact with the contract, the frontend will call upon the contract and assign the minted token to the user address which will be his own.
    - Again from the frontend the user will be able to change the price of the token, which will interact with the contract and change the price of the token using ethers library)
    - The tokens the user has are his own and to use them in the UI we will use the ethers library to interact with the contract and receive these NFTs related to a specific user account. In the UI, the user has the ability to display them or cancel them. As they are stored in a “My NFTs” tab.Once they are displayed in the UI, they can be bought by other accounts on the site.

    - The marketplace displays the listings and allows other users to browse, search, and purchase the listed custom tokens. The Marketplace will interact with the contract and retrieve all minted NFTs and display them (only listed NFTs) in a section on the UI. User can interact with that section and look for the NFTs he liked, then they can click on the NFT where they will be able to purchase the NFT,
    - Upon a successful purchase, the ownership of the custom token will be transferred to the buyer. The marketplace will call on the contract buyToken function with specific arguments like to (buyer’s address) and price (buying price). To invoke a transaction in

metamask, the user will accept or reject the transaction on metamask, on accepting the NFT ownership will be transferred to the buyer.
    - How?
- The marketplace processes the transactions using MetaMask securely. Using ethers library. Because the verification and validation is done by MetaMask, and any vulnerability in the frontend itself does not affect the security as there's no functionality affecting the user wallet without MetaMask's own security.

4. What happens when a user adds a product to the cart
    - Allow users to view and manage the items in their cart in the UI.
        - Any backend call involved in this?
    - Users can purchase the item in the cart, invoking the buy token function from the contract and such transaction will be processed by MetaMask.
5. What happens when a user buys an item?
    - The ownership of that item (token) will be transferred to him, and he will be able to list it again and specify the price.
    - This ownership transfer functionality is included in the buy token function inside the contract.

Testing

1. How to do the unit testing
    - Use Chai and Hardhat to write and execute unit tests for the Solidity smart contracts.
2. How to do the automation testing
    - Set up automated tests using Truffle to test the smart contracts' behavior in different scenarios.
3. How to do the manual testing
    - Manually test the user interface and user flows to ensure a smooth user experience and for detecting unwanted behaviors and bugs.
4. Any set up instructions for testing
    - Installing hardhat will set up Chai as it's included inside of it and running 'npx hardhat test' will test the solidity code.
    - Automation testing setup using waffle