Apollo

by:

Alex, Jorge, Jenny, & David

Summary: Apollo

NFT Marketplace that integrates blockchain technology to construct 'the amazon' marketplace for NFT creators, enthusiasts, and collectors.

Allow users to upload images, record the mint transaction on the ethereum block chain, and uploads image to IPFS.

Enable the creation of smart contracts that support royaltys, ensuring artists continue to financialy benefit from their work after the initial sale.

The Problem

NFT copyright infringement enforcement is largely left to the creators themselves. There are minimal laws to protect them, and without sufficient funds to pay for costly legal battles -- which many creators lack, there is little creators can do to stop/prevent plagiarism, which can be highly costly.

Approach

Method:

Distributed tasks and met frequently to ensure progress was made smoothly.

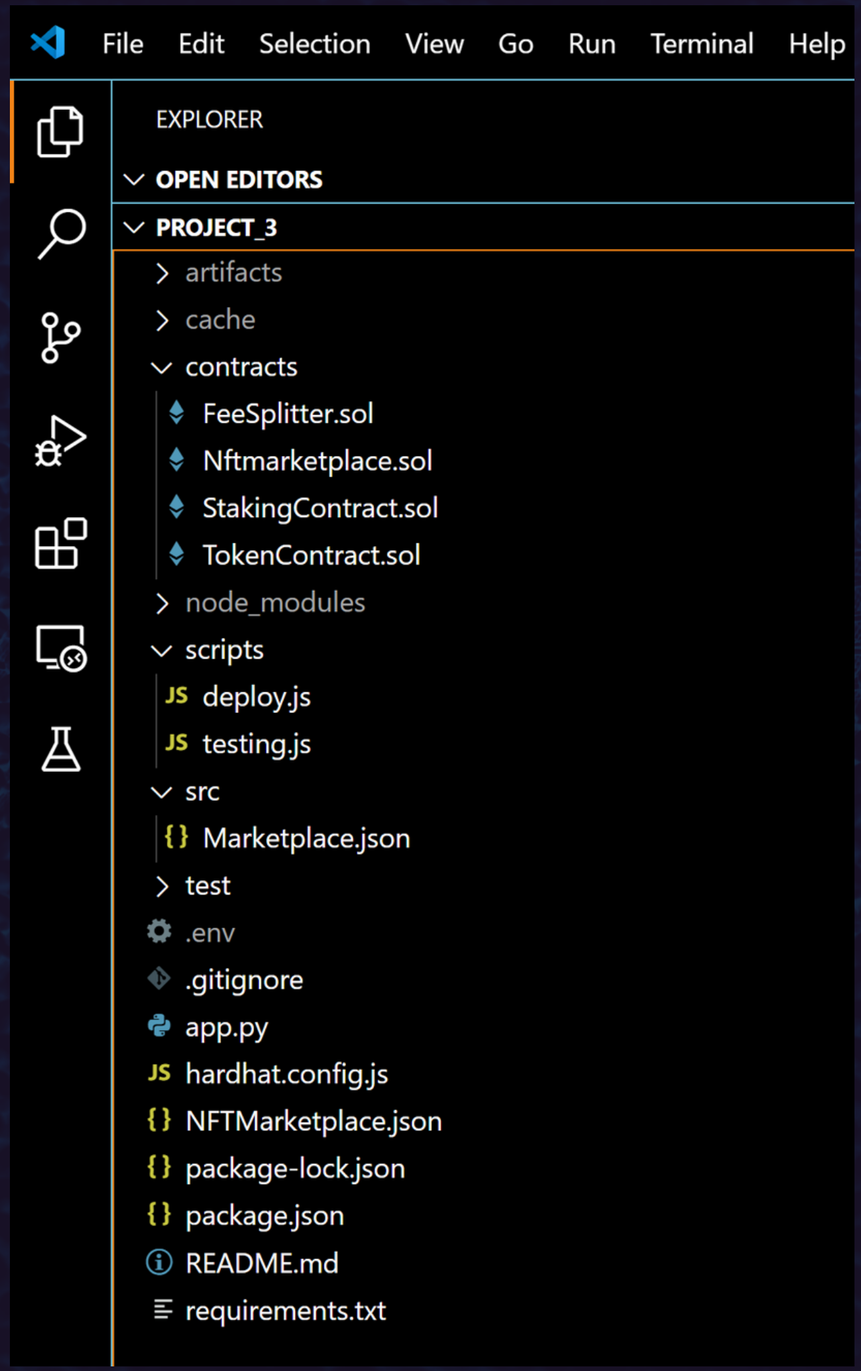
Unexpected (-):

- Front-end communication with contracts
- Package compatibility

Unexpected (+):

- discovery of Hardhat
- Ethereum sepolia

APOLLO


File
Edit
Selection
View
Go
Run
Terminal
Help
EXPLORER
OPEN EDITORS
PROJECT_3
artifacts
cache
contracts
FeeSplitter.sol
Nftmarketplace.sol
StakingContract.sol
TokenContract.sol
node_modules
scripts
deploy.js
testing.js
src
Marketplace.json
test
.env
.gitignore
app.py
hardhat.config.js
NFTMarketplace.json
package-lock.json
package.json
README.md
requirements.txt

```
//Allows users to create an NFT Token
function createToken(string memory tokenURI, uint256 price) public payable returns (uint) {    infinite gas
    _tokenIds += 1;
    uint256 newTokenId = _tokenIds;

    _safeMint(msg.sender, newTokenId);
    _setTokenURI(newTokenId, tokenURI);
    createListedToken(newTokenId, price);

    return newTokenId;
}

function createListedToken(uint256 tokenId, uint256 price) private {    infinite gas
    require(msg.value == listingPrice, "Price must be equal to listing price");
    require(price > 0, "Price must be greater than zero");
```

```
// ExexuteSale function
function executeSale(uint256 tokenId) public payable {    infinite gas
    uint price = idToListedToken[tokenId].price;
    address seller = idToListedToken[tokenId].seller;
    require(msg.value == price, "Please submit the asking price in order to complete the purchase");

    idToListedToken[tokenId].currentlyListed = false;
    idToListedToken[tokenId].seller = payable(msg.sender);
    _itemsSold++;

    _transfer(address(this), msg.sender, tokenId);
    payable(owner).transfer(listingPrice);
    payable(seller).transfer(msg.value);
}
```

APOLLO

```
// Auction functions
function createAuction(uint256 tokenId, uint256 minPrice, uint256 duration) public {    infinite gas
    require(msg.sender == ownerOf(tokenId), "Only the owner can create an auction");
    require(duration > 0, "Duration must be greater than zero");

    auctions[tokenId] = Auction(
        tokenId,
        payable(msg.sender),
        minPrice,
        block.timestamp + duration,
        address(0),
        0,
        false
    );

    emit AuctionCreated(tokenId, msg.sender, minPrice, block.timestamp + duration);
}

function bid(uint256 tokenId) public payable {    infinite gas
    Auction storage auction = auctions[tokenId];
    require(block.timestamp < auction.endTime, "Auction has ended");
    require(msg.value > auction.highestBid, "Bid must be higher than the current highest bid");

    if (auction.highestBidder != address(0)) {
        // Refund the previous highest bidder
        payable(auction.highestBidder).transfer(auction.highestBid);
    }
```

scripts > JS deploy.js > ...

```
1   const { ethers } = require("hardhat");
2   const hre = require("hardhat");
3   const fs = require("fs");
4
5   async function main() {
6     const [deployer] = await ethers.getSigners();
7     const balance = await deployer.getBalance();
8     const Marketplace = await hre.ethers.getContractFactory("NFTMarketplace");
9     const marketplace = await Marketplace.deploy();
10
11    await marketplace.deployed();
12
13    const data = {
14      address: marketplace.address,
15      abi: JSON.parse(marketplace.interface.format('json'))
16    }
17
18    //This writes the ABI and address to the mktplace.json
19    fs.writeFileSync('./src/Marketplace.json', JSON.stringify(data))
20  }
21
22  main()
23    .then(() => process.exit(0))
24    .catch((error) => {
25      console.error(error);
26      process.exit(1);
27    });
```

```
contract FeeSplitter is ReentrancyGuard, Ownable {
    // ApolloToken contract interface
    IERC20 public apolloToken;

    // Addresses for fee distribution
    address public stakingContract;
    address public treasury;
    address public otherBeneficiary; // Example: community fund or charity

    // Fee distribution percentages (must add up to 100%)
    uint256 public stakingShare = 50;
    uint256 public treasuryShare = 30;
    uint256 public otherShare = 20;

    // Event for logging fee distribution
    event FeesDistributed(uint256 totalAmount, uint256 stakingAmount, uint256 treasuryAmount, uint256 otherAmount);
```

```
//Bulk transfer function to allow sending tokens to multiple recipients
function bulkTransfer(address[] calldata recipients, uint256[] calldata amounts) public {
    require(recipients.length == amounts.length, "Recipients and amounts length mismatch");
    for (uint256 i = 0; i < recipients.length; i++) {
        _transfer(msg.sender, recipients[i], amounts[i]);
    }
}
```

```
// Function to distribute fees
function distributeFees() public nonReentrant {
    uint256 totalBalance = address(this).balance;
    require(totalBalance > 0, "No fees to distribute");

    uint256 stakingAmount = (totalBalance * stakingShare) / 100;
    uint256 treasuryAmount = (totalBalance * treasuryShare) / 100;
    uint256 otherAmount = (totalBalance * otherShare) / 100;

    // Ensure the total distribution does not exceed the total balance due to rounding
    uint256 totalDistributed = stakingAmount + treasuryAmount + otherAmount;
    if (totalDistributed > totalBalance) {
        // Adjust to prevent overflow
        otherAmount -= totalDistributed - totalBalance;
    }
    // Notify staking contract about new rewards
    IApolloStaking(stakingContract).notifyNewRewards(stakingAmount);
    // Transfer funds
    payable(treasury).transfer(treasuryAmount);
    payable(otherBeneficiary).transfer(otherAmount);

    emit FeesDistributed(totalBalance, stakingAmount, treasuryAmount, otherAmount);
}
```

APOLLO

Results

01. Front End Works

02. NFT Minting

03. Hardhat

04. Pinata

Next Steps

- Further develop FE UI with BE contracts
- Deploy on public Testnet
- Hosting application through AWS

APOLLO

APOLLO

THANK YOU.