School: ..... Campus: .....

Academic Year: ..... Subject Name: ..... Subject Code: .....

Semester: ..... Program: ..... Branch: ..... Specialization: .....

Date: .....

Applied and Action Learning

(Learning by Doing and Discovery)

Name of the Experiement :

* Coding Phase: Pseudo Code / Flow Chart / Algorithm

Algorithm:

1. Set up dev environment and wallet.
2. Deploy the NFT (ERC-721) contract.
3. Mint the NFT (create token + metadata).
4. Write the simple marketplace contract.
5. Deploy the marketplace contract.
6. Approve the marketplace to handle your NFT.
7. List the NFT for sale on the marketplace.
8. Buy the listed NFT (pay price).
7. Confirm ownership transfer and funds received.

* Softwares used

1. Brave browser
2. MetaMask Wallet
3. Remix IDE
4. Sepolia Testnet

*As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.

* Implementation Phase: Final Output (no error)

First created your NFT and MINT it after successfully deployed the smart contract then write the smart contract for your simple NFT-Marketplace . Here is the code below->

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;
import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";
import "@openzeppelin/contracts/utils/Counters.sol";
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";

contract SimpleNFTMarket is ERC721URIStorage, ReentrancyGuard {
    using Counters for Counters.Counter;
    Counters.Counter private _tokenIds;
    struct Listing {
        address seller;
        uint256 price;
        bool active;
    }
    mapping(uint256 => Listing) public listings;
    mapping(address => uint256) public proceeds;
    uint256[] private listedTokenIds;
    event Minted(address indexed minter, uint256 indexed tokenId, string tokenURI);
    event Listed(address indexed seller, uint256 indexed tokenId, uint256 price);
    event PriceUpdated(address indexed seller, uint256 indexed tokenId, uint256 newPrice);
    event Sale(address indexed buyer, address indexed seller, uint256 indexed tokenId, uint256 price);
    event Cancelled(address indexed seller, uint256 indexed tokenId);
    event ProceedsWithdrawn(address indexed seller, uint256 amount);
    constructor() ERC721("Simple Market NFT", "SMN") {}
    function mint(string calldata uri) external returns (uint256 tokenId) {
        _tokenIds.increment();
        tokenId = _tokenIds.current();
        _safeMint(msg.sender, tokenId);
        _setTokenURI(tokenId, uri);
        emit Minted(msg.sender, tokenId, uri);
    }
    function listNFT(uint256 tokenId, uint256 price) external nonReentrant {
        require(ownerOf(tokenId) == msg.sender, "Not owner");
        require(price > 0, "Price must be > 0");
        require(!listings[tokenId].active, "Already listed");
        safeTransferFrom(msg.sender, address(this), tokenId);
        listings[tokenId] = Listing({
            seller: msg.sender,
            price: price,
            active: true
        });
        listedTokenIds.push(tokenId);
        emit Listed(msg.sender, tokenId, price);
    }
    function updatePrice(uint256 tokenId, uint256 newPrice) external {
        Listing storage lst = listings[tokenId];
        require(lst.active, "Not listed");
        require(lst.seller == msg.sender, "Not seller");
        require(newPrice > 0, "Price must be > 0");
        lst.price = newPrice;
        emit PriceUpdated(msg.sender, tokenId, newPrice);
    }
```


*As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.

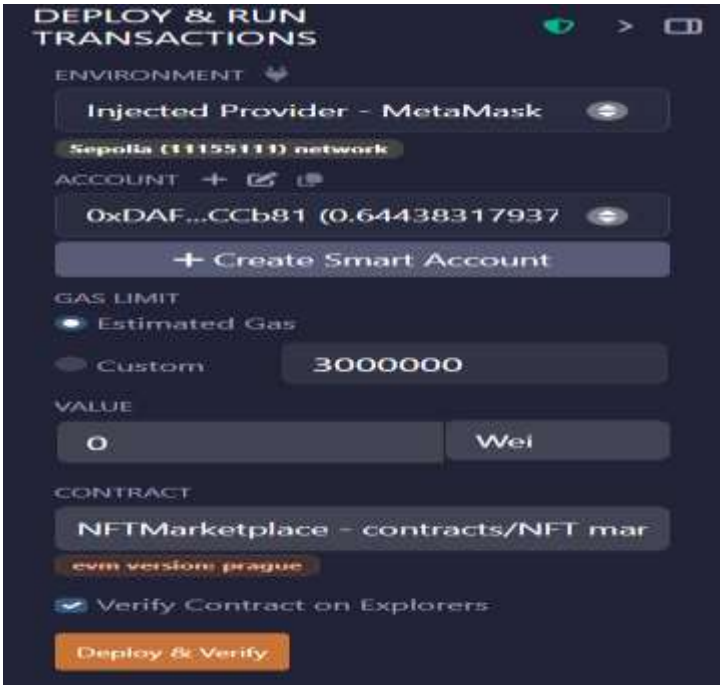
* Implementation Phase: Final Output (no error)

```
    function cancelListing(uint256 tokenId) external nonReentrant {
        Listing storage lst = listings[tokenId];
        require(lst.active, "Not listed");
        require(lst.seller == msg.sender, "Not seller");
        lst.active = false;
        _safeTransfer(address(this), msg.sender, tokenId, "");
        emit Cancelled(msg.sender, tokenId);
    }
    function buyNFT(uint256 tokenId) external payable nonReentrant {
        Listing storage lst = listings[tokenId];
        require(lst.active, "Not listed");
        require(msg.value == lst.price, "Incorrect value");
        address seller = lst.seller;
        lst.active = false;
        proceeds[seller] += msg.value;
        _safeTransfer(address(this), msg.sender, tokenId, "");
        emit Sale(msg.sender, seller, tokenId, msg.value);
    }
    function withdrawProceeds() external nonReentrant {
        uint256 amount = proceeds[msg.sender];
        require(amount > 0, "No proceeds");
        proceeds[msg.sender] = 0;
        (bool ok, ) = payable(msg.sender).call{value: amount}("");
        require(ok, "Transfer failed");
        emit ProceedsWithdrawn(msg.sender, amount);
    }
    function getListing(uint256 tokenId) external view returns (address seller, uint256 price, bool active) {
        Listing memory lst = listings[tokenId];
        return (lst.seller, lst.price, lst.active);
    }
    function totalMinted() external view returns (uint256) {
        return _tokenIds.current();
    }
    function getActiveListings()
        external
        view
        returns (uint256[] memory ids, address[] memory sellers, uint256[] memory prices)
    {
        uint256 n = listedTokenIds.length;
        uint256 count;
        for (uint256 i = 0; i < n; i++) {
            if (listings[listedTokenIds[i]].active) {
                count++;
            }
        }
        ids = new uint256[](count);
        sellers = new address[](count);
        prices = new uint256[](count);
        uint256 idx;
        for (uint256 i = 0; i < n; i++) {
            uint256 tid = listedTokenIds[i];
            Listing memory lst = listings[tid];
            if (lst.active) {
                ids[idx] = tid;
                sellers[idx] = lst.seller;
                prices[idx] = lst.price;
                idx++;
            }
        }
    }
    receive() external payable {}
}
```

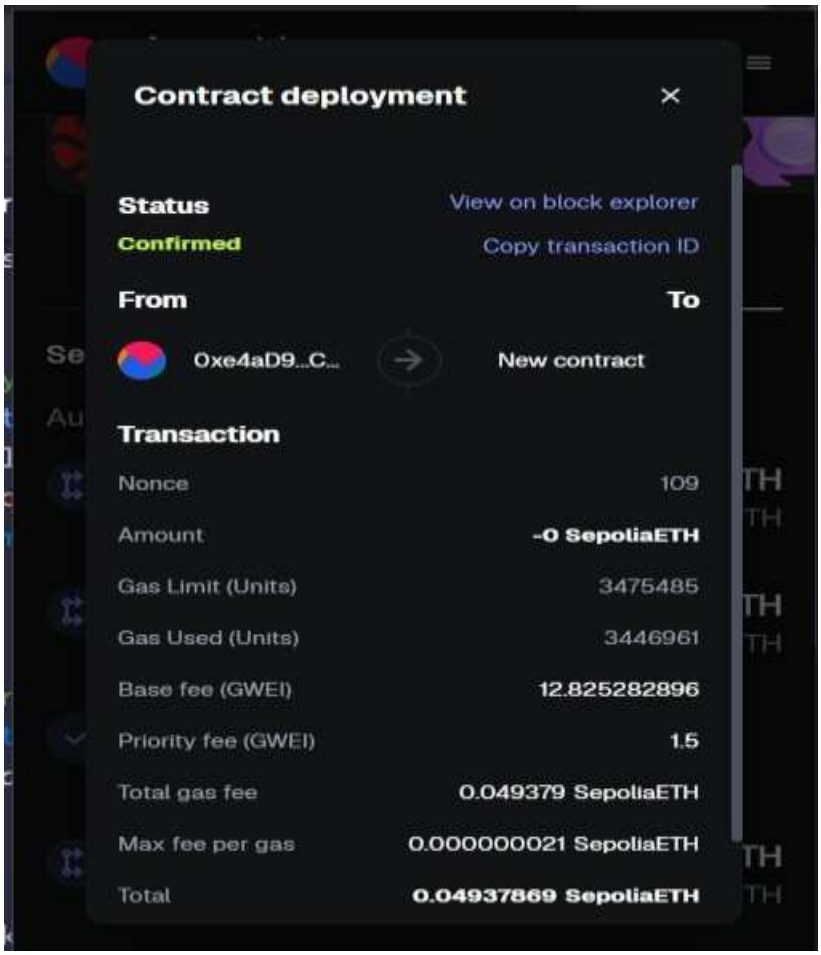

*As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.

* Implementation Phase: Final Output (no error)

After writing the smart contract now deploy the smart contract by choosing inject provider metamask

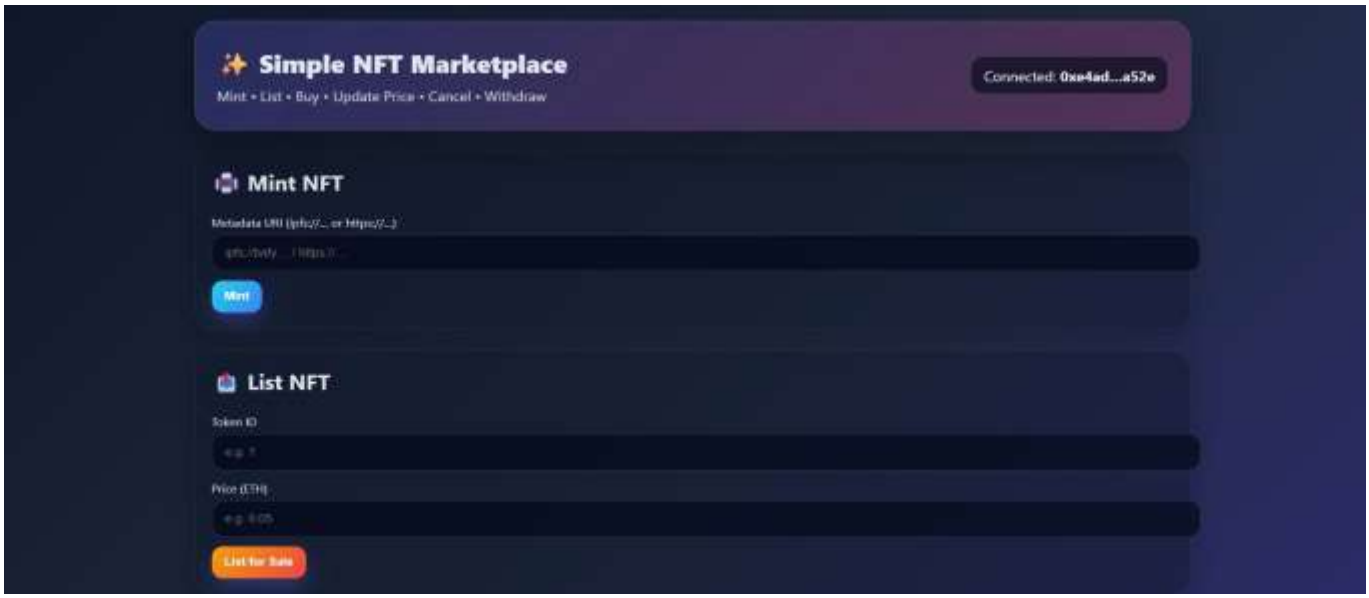


Our contract is successfully deployed

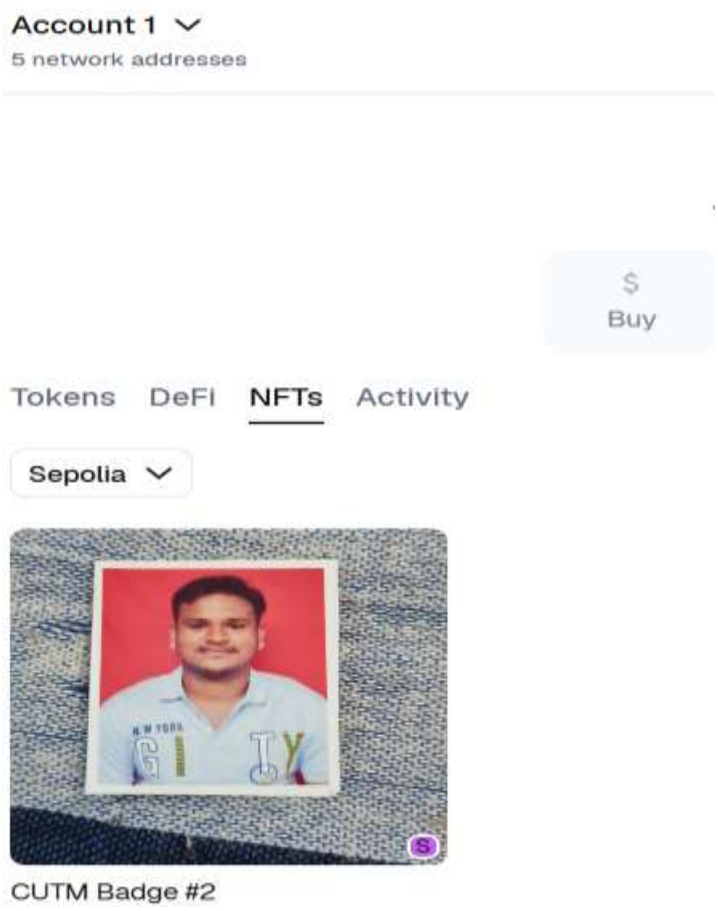


*As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.

* Implementation Phase: Final Output (no error)



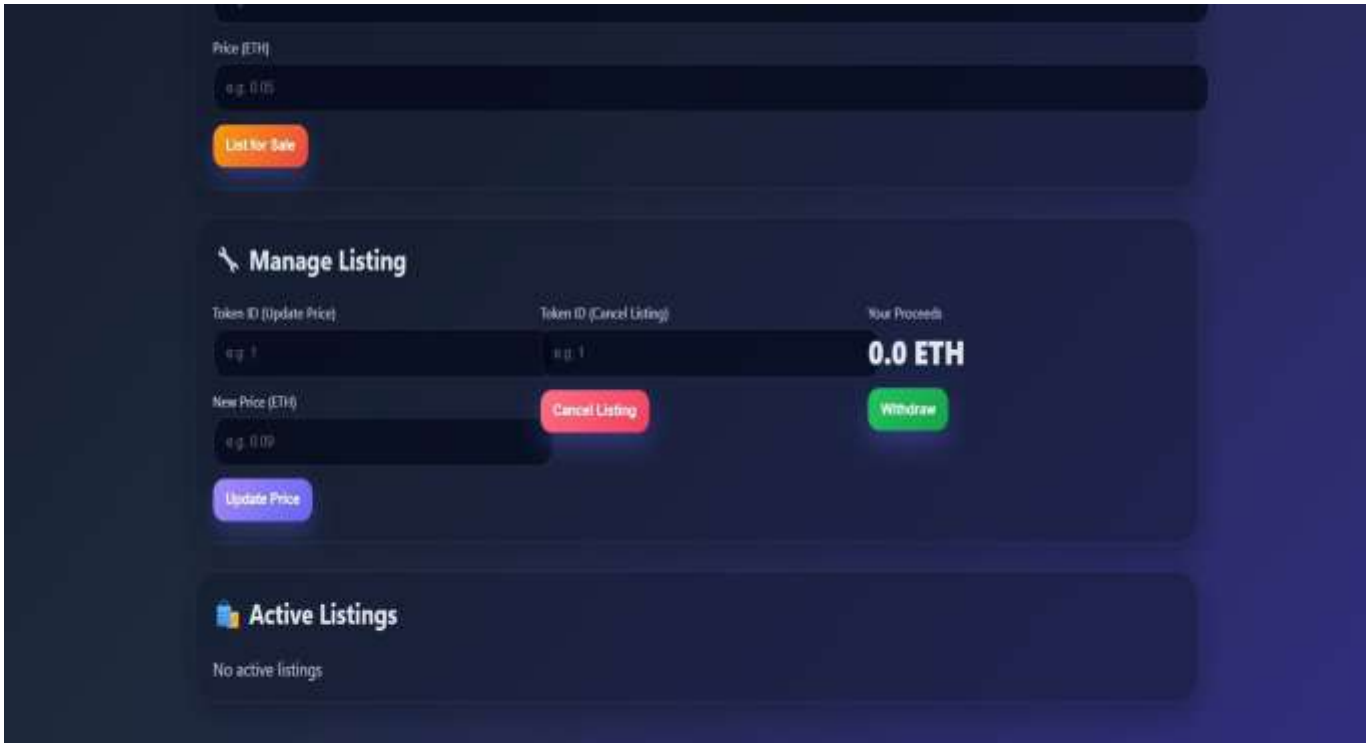
After this paste your metadata URI to mint your NFT



Now you can add the nft list and set the price of the NFT and also manage the listing

*As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.

* Implementation Phase: Final Output (no error)



* Observations

The project involved deploying an NFT smart contract and successfully minting NFTs with associated metadata. After that, a simple marketplace smart contract was developed and deployed to enable listing and purchasing of NFTs. The functionality was tested by listing an NFT for sale and completing a purchase transaction, which resulted in a successful transfer of ownership to the buyer. This process demonstrated the complete NFT lifecycle, from minting to listing, buying, and verifying ownership transfer.

ASSESSMENT

Rubrics	Full Mark	Marks Obtained	Remarks
Concept	10		
Planning and Execution/ Practical Simulation/ Programming	10		
Result and Interpretation	10		
Record of Applied and Action Learning	10		
Viva	10		
Total	**50**		

Signature of the Student:

Name :

Regn. No. :

Signature of the Faculty:

****As applicable according to the experiment. Two sheets per experiment (10-20) to be used.***