



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 Experiment :

### \* **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).
9. Confirm ownership transfer and funds received.

### \* **Softwares used**

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

## \* 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->

```

1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.20;
3 import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";
4 import "@openzeppelin/contracts/utils/Counters.sol";
5 import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
6
7 contract SimpleNFTMarket is ERC721URIStorage, ReentrancyGuard {
8     using Counters for Counters.Counter;
9     Counters.Counter private _tokenIds;
10    struct Listing {
11        address seller;
12        uint256 price;
13        bool active;
14    }
15    mapping(uint256 => Listing) public listings;
16    mapping(address => uint256) public proceeds;
17    uint256[] private listedTokenIds;
18    event Minted(address indexed minter, uint256 indexed tokenId, string tokenURI);
19    event Listed(address indexed seller, uint256 indexed tokenId, uint256 price);
20    event PriceUpdated(address indexed seller, uint256 indexed tokenId, uint256 newPrice);
21    event Sale(address indexed buyer, address indexed seller, uint256 indexed tokenId, uint256 price);
22    event Cancelled(address indexed seller, uint256 indexed tokenId);
23    event ProceedsWithdrawn(address indexed seller, uint256 amount);
24    constructor() ERC721("Simple Market NFT", "SMN") {}
25    function mint(string calldata uri) external returns (uint256 tokenId) {
26        _tokenIds.increment();
27        tokenId = _tokenIds.current();
28        _safeMint(msg.sender, tokenId);
29        _setTokenURI(tokenId, uri);
30        emit Minted(msg.sender, tokenId, uri);
31    }
32    function listNFT(uint256 tokenId, uint256 price) external nonReentrant {
33        require(ownerOf(tokenId) == msg.sender, "Not owner");
34        require(price > 0, "Price must be > 0");
35        require(!listings[tokenId].active, "Already listed");
36        safeTransferFrom(msg.sender, address(this), tokenId);
37        listings[tokenId] = Listing({
38            seller: msg.sender,
39            price: price,
40            active: true
41        });
42        listedTokenIds.push(tokenId);
43        emit Listed(msg.sender, tokenId, price);
44    }
45    function updatePrice(uint256 tokenId, uint256 newPrice) external {
46        Listing storage lst = listings[tokenId];
47        require(lst.active, "Not listed");
48        require(lst.seller == msg.sender, "Not seller");
49        require(newPrice > 0, "Price must be > 0");
50        lst.price = newPrice;
51        emit PriceUpdated(msg.sender, tokenId, newPrice);
52    }

```

## \* Implementation Phase: Final Output (no error)

```

1   function cancelListing(uint256 tokenId) external nonReentrant {
2       Listing storage lst = listings[tokenId];
3       require(lst.active, "Not listed");
4       require(lst.seller == msg.sender, "Not seller");
5       lst.active = false;
6       _safeTransfer(address(this), msg.sender, tokenId, "");
7       emit Cancelled(msg.sender, tokenId);
8   }
9   function buyNFT(uint256 tokenId) external payable nonReentrant {
10      Listing storage lst = listings[tokenId];
11      require(lst.active, "Not listed");
12      require(msg.value == lst.price, "Incorrect value");
13      address seller = lst.seller;
14      lst.active = false;
15      proceeds[seller] += msg.value;
16      _safeTransfer(address(this), msg.sender, tokenId, "");
17      emit Sale(msg.sender, seller, tokenId, msg.value);
18  }
19  function withdrawProceeds() external nonReentrant {
20      uint256 amount = proceeds[msg.sender];
21      require(amount > 0, "No proceeds");
22      proceeds[msg.sender] = 0;
23      (bool ok, ) = payable(msg.sender).call{value: amount}("");
24      require(ok, "Transfer failed");
25      emit ProceedsWithdrawn(msg.sender, amount);
26  }
27  function getListing(uint256 tokenId) external view returns (address seller, uint256 price, bool active) {
28      Listing memory lst = listings[tokenId];
29      return (lst.seller, lst.price, lst.active);
30  }
31  function totalMinted() external view returns (uint256) {
32      return _tokenIds.current();
33  }
34  function getActiveListings()
35      external
36      view
37      returns (uint256[] memory ids, address[] memory sellers, uint256[] memory prices)
38  {
39      uint256 n = listedTokenIds.length;
40      uint256 count;
41      for (uint256 i = 0; i < n; i++) {
42          if (listings[listedTokenIds[i]].active) {
43              count++;
44          }
45      }
46      ids = new uint256[](count);
47      sellers = new address[](count);
48      prices = new uint256[](count);
49      uint256 idx;
50      for (uint256 i = 0; i < n; i++) {
51          uint256 tid = listedTokenIds[i];
52          Listing memory lst = listings[tid];
53          if (lst.active) {
54              ids[idx] = tid;
55              sellers[idx] = lst.seller;
56              prices[idx] = lst.price;
57              idx++;
58          }
59      }
60  }
61  receive() external payable {}
62 }
63
64

```

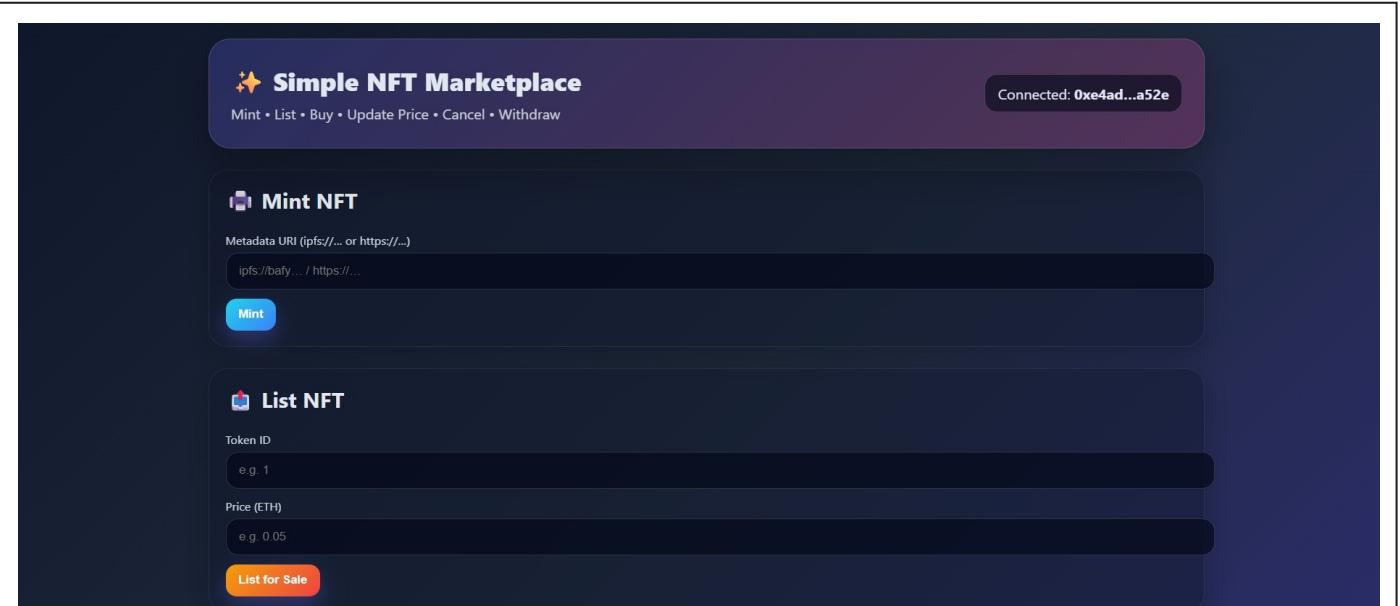
## \* Implementation Phase: Final Output (no error)

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

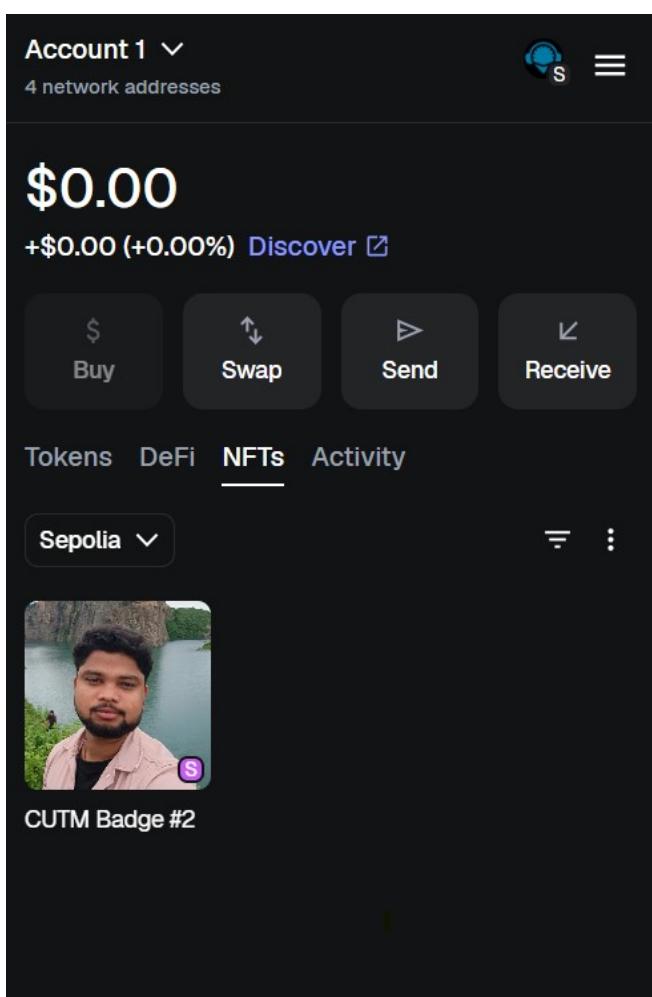
Our contract is successfully deployed

Transaction	
Nonce	76
Amount	-0 SepoliaETH
Gas Limit (Units)	1215908
Gas Used (Units)	1205176
Base fee (GWEI)	2.796537176
Priority fee (GWEI)	1.5
Total gas fee	0.005178 SepoliaETH
Max fee per gas	0.000000006 SepoliaETH
Total	0.00517808 SepoliaETH

## \* 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

## \* Implementation Phase: Final Output (no error)

Applied and Action Learning

The screenshot displays a dark-themed user interface for managing NFT listings. At the top, there's a field for setting a price in ETH (e.g., 0.05) and a button to 'List for Sale'. Below this is a 'Manage Listing' section where users can update the price (e.g., 1), cancel the listing, or withdraw proceeds (0.0 ETH). A green 'Withdraw' button is visible. At the bottom, there's a section titled 'Active Listings' which currently shows 'No active listings'.

## \* 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		
<b>Total</b>	<b>50</b>		

**Signature of the Student:**

Name :

Regn. No. :

Page No.....

**Signature of the Faculty:**

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