Blockchain

Abhishek Prashant Chandurkar(BT22CSE104) Manas Sandip Jungade(BT22CSE127)

March 2025

1 Smart Contract Development & Testnet Deployment

1.1 Research and Setup

- **Smart Contract Development Environment**: Remix IDE (Online Browser-Based)
- **Ethereum Testnet**: Sepolia
- **Wallet**: MetaMask (Chrome Extension installed)
- **Faucet**: Provided by Google

1.2 Write and Deploy

Solidity Contract:

Listing 1: Simple Solidity Smart Contract

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.15;

contract SimpleContract {
    uint private age;
    string private name;

    // Set functions
    function setName(string memory _name) public {
        name = _name;
    }

    function setAge(uint _age) public {
        age = _age;
    }

    // Get functions
    function getName() public view returns (string memory) {
        return name;
    }

    function getAge() public view returns (uint) {
        return age;
    }
}
```

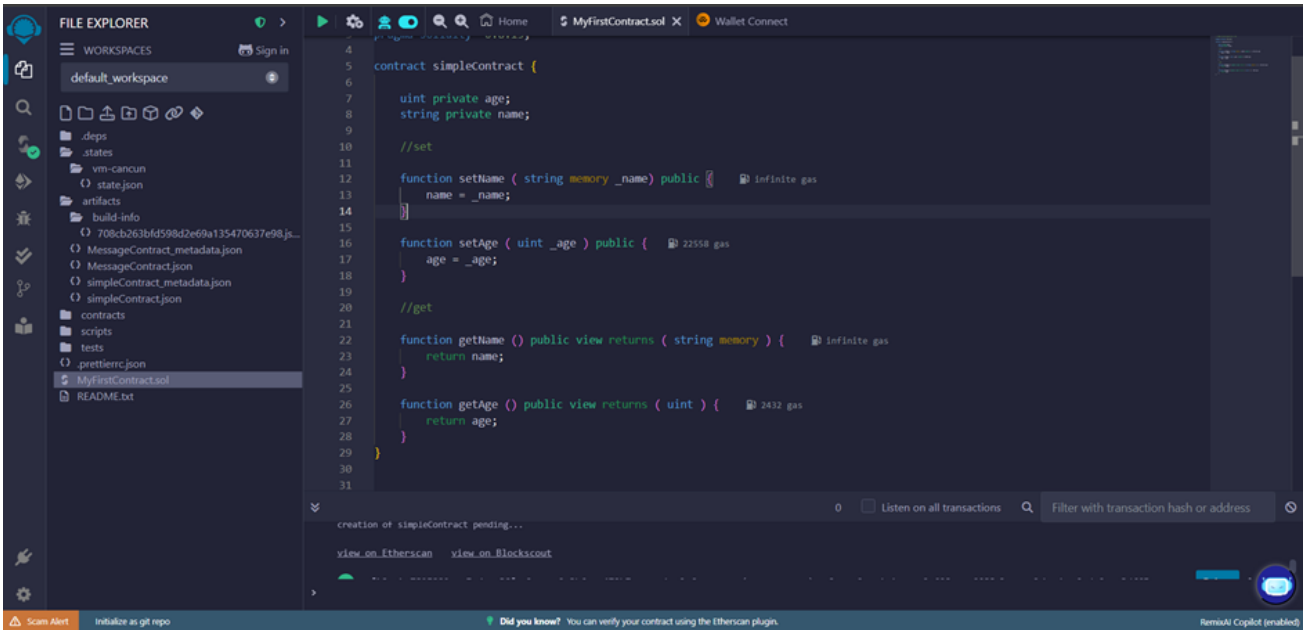

Figure 1: Remix IDE - MyFirstContract.sol file

Compilation is done by pressing **Ctrl+S** (Save). It can also be done manually.

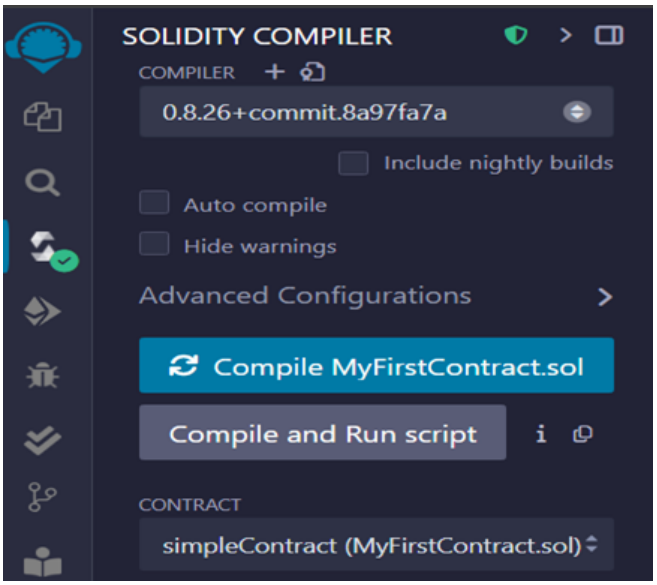

Figure 2: Compilation

Deploy on Remix VM Cancun to test the contract (Yet to get ETH).
Get SepoliaETH from here: Faucet for Sepolia Testnet

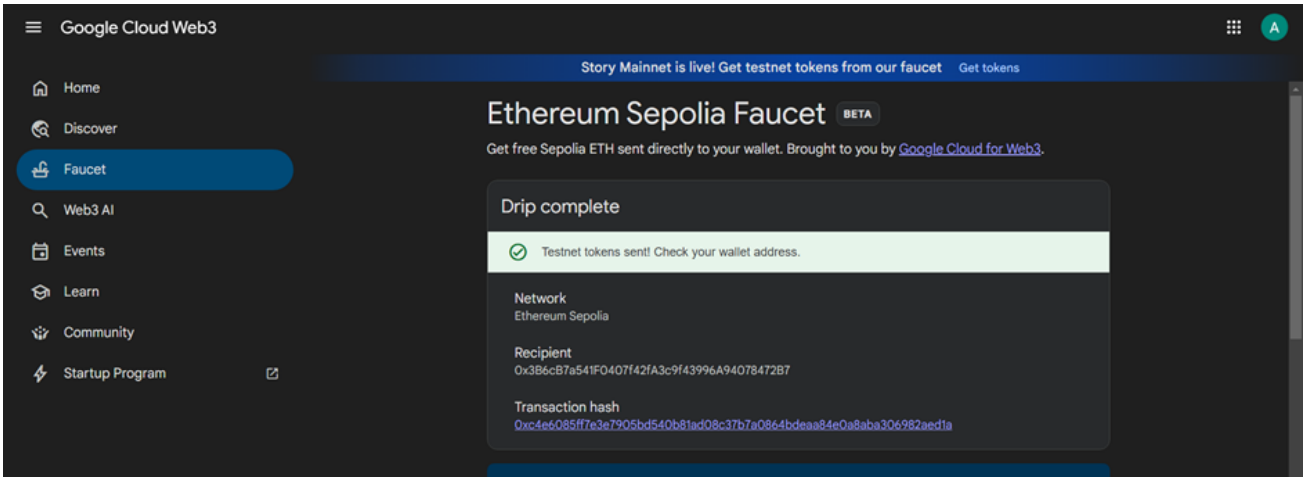

Figure 3: Getting ETH for free

Visit: `https://chainid.network/` and search "Sepolia". Click on "Add Chain" and accept the prompt from MetaMask.

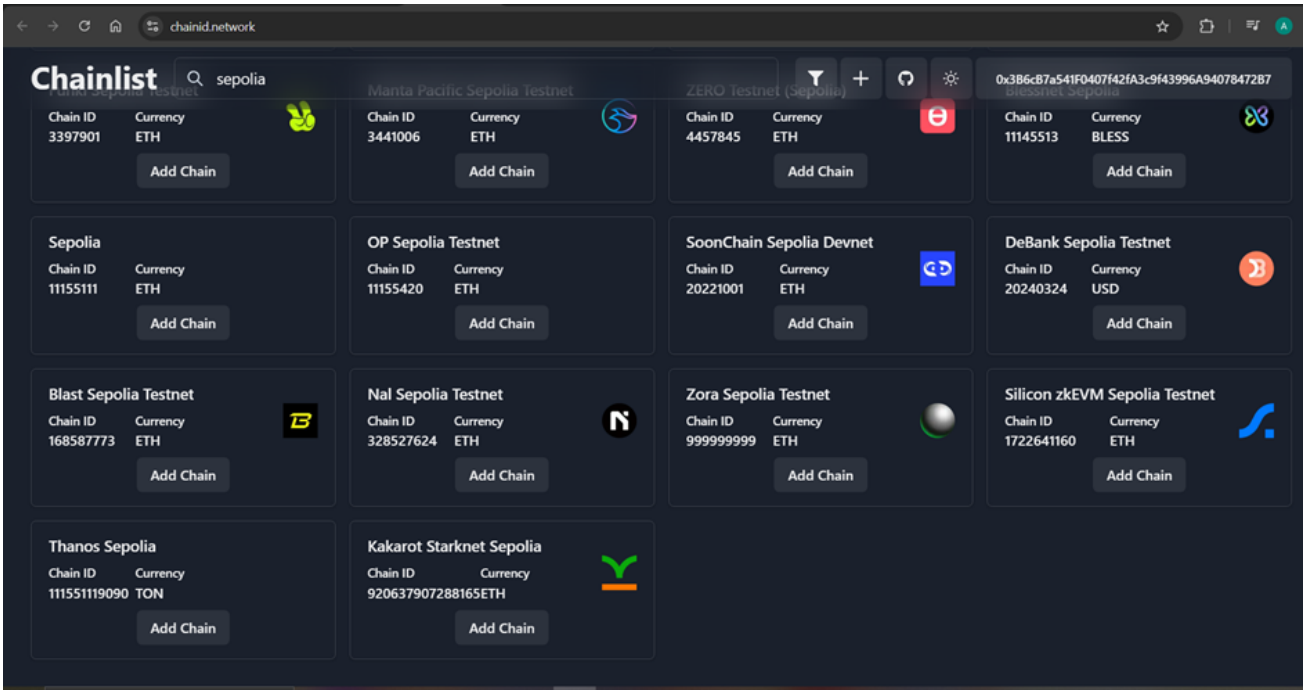

Figure 4: ChainID - Connect

Deployed on Sepolia Testnet using the **Injected Provider - MetaMask** option in Remix IDE.

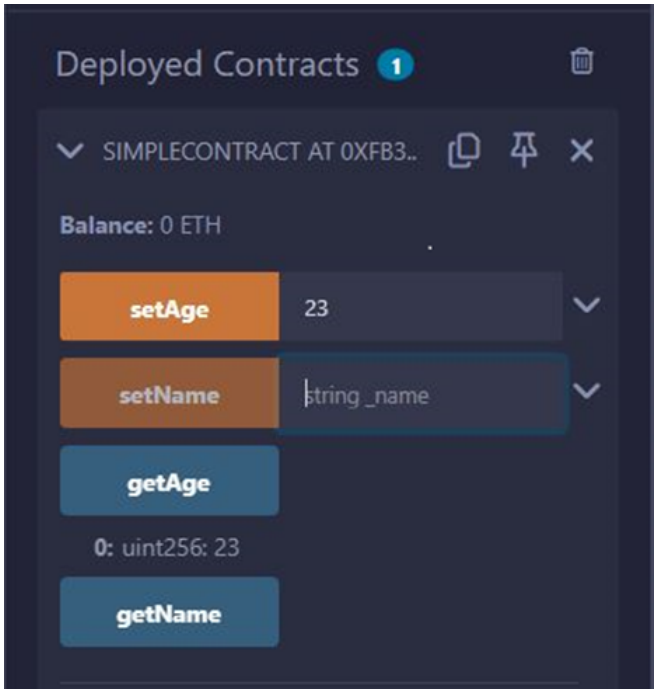

Figure 5: Performing setAge operation

Some ETH (0.0005) was charged for deployment, and for the `setAge` operation, it was around 0.0001. Deployment took longer than using Remix VM.

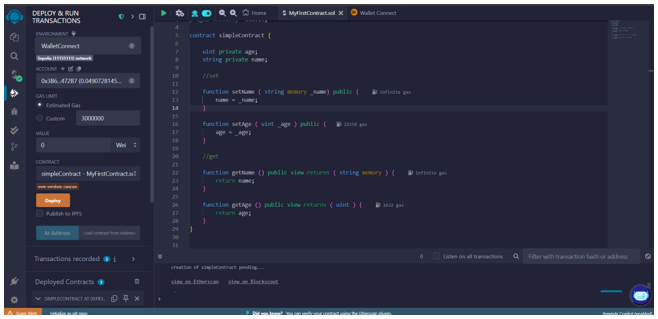

Figure 6: Look at the terminal - see the link to Etherscan

You can see the transaction on Etherscan - Blockchain Browser.

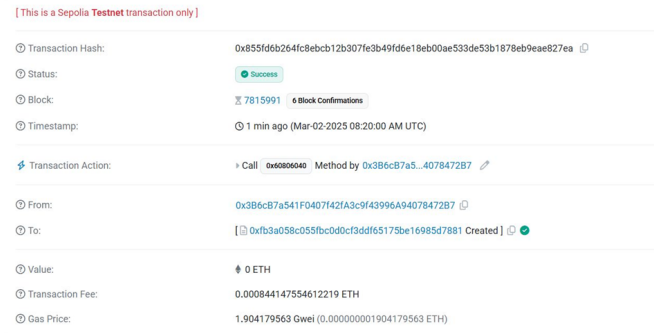

Figure 7: Etherscan - Transaction Details

Initialize a GitHub repository for version control: Blockchain

1.3 Challenges Faced

- Unable to find "Injected Provide Metamask" option at first. So used "Wallet Connect" in first attempt. Later the former option became available.
- Tried to create an Alchemy account and get tokens but failed due to a minimum cap on ETH tokens (mainnet) required to prevent spam.

1.4 References

- Understand Simple Solidity Contract and Deploy it on Remix VM
- Connect with Sepolia Testnet

Figure 8: Minimum cap on ETH balance in mainnet to be 0.01

2 Creating & Deploying Your Own ERC20 Token

Solidity Contract:

Listing 2: ERC20 Token - Fungible Tokens

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/access/Ownable.sol";

contract MyToken is Ownable {
    string public name = "Coin";
    string public symbol = "C#";
    uint public totalSupply;
    uint public maxSupply;

    mapping(address => uint) public balances;

    event Mint(address indexed to, uint amount);
    event Burn(address indexed from, uint amount);
    event Transfer(address indexed from, address indexed to, uint amount);

    constructor(uint _maxSupply) Ownable(msg.sender) {
        maxSupply = _maxSupply;
    }

    function mint(address _to, uint _value) public onlyOwner {
        require(totalSupply + _value <= maxSupply, "Exceeds␣max␣supply");
        totalSupply += _value;
        balances[_to] += _value;
        emit Mint(_to, _value);
    }

    function burn(uint _value) public {
        require(balances[msg.sender] >= _value, "Insufficient␣balance");
        totalSupply -= _value;
        balances[msg.sender] -= _value;
        emit Burn(msg.sender, _value);
    }

    function transfer(address _to, uint _value) public {
        require(balances[msg.sender] >= _value, "Insufficient␣balance");
        require(_to != address(0), "Invalid␣recipient␣address");
        balances[msg.sender] -= _value;
        balances[_to] += _value;
        emit Transfer(msg.sender, _to, _value);
    }

    function getMaxSupply() public view returns (uint) {
        return maxSupply;
    }

    function TotalSupply() public view returns (uint) {
        return totalSupply;
    }
}
```

- In this contract, we have defined our own token with functionalities following to ERC20 token standards.
- We did the work in iterations , making improvements each time. The screenshots below are from different iterations. Final attempt has been provided at the end , whose details can be seen on Etherscan easily.
- Deploy on Sepolia Testnet OR Remix VM(to save ETH)

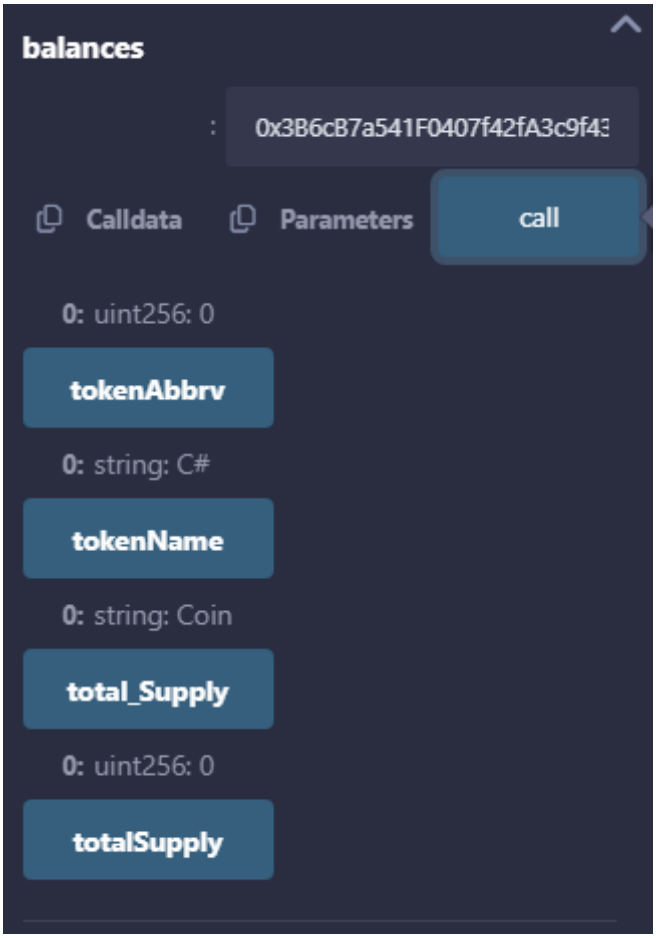

Figure 9: Initial State

We can see the name **(Coin)**,abbreviation**(C#)**,total supply**(Total tokens = 0)**.

Let us **"mint"** some tokens.
To "mint" tokens means to add the tokens in the supply. So the balance of address provided will be incremented but without deducting it
from someone else balance. So effectively we add new tokens in the supply.

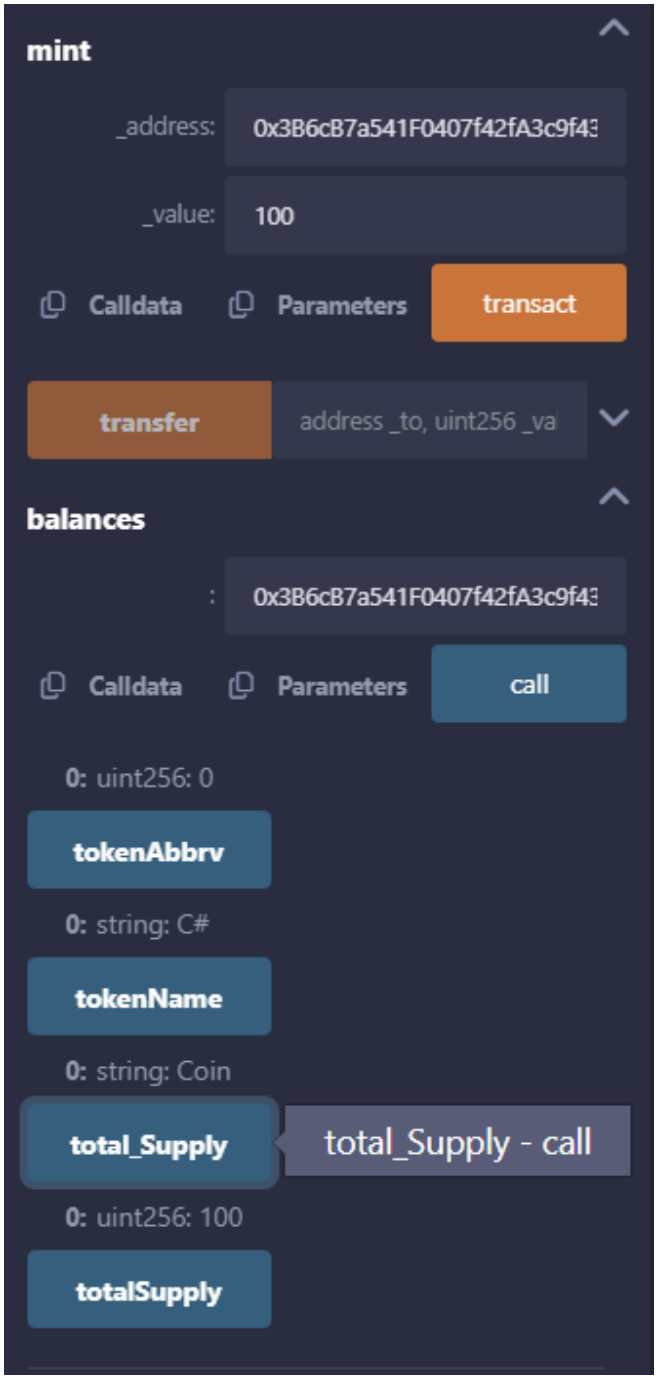

Figure 10: Mint 100 tokens. Total Supply Changes to 100.

Burning tokens is just the opposite. Total supply will decrease as well as the provided address balance.

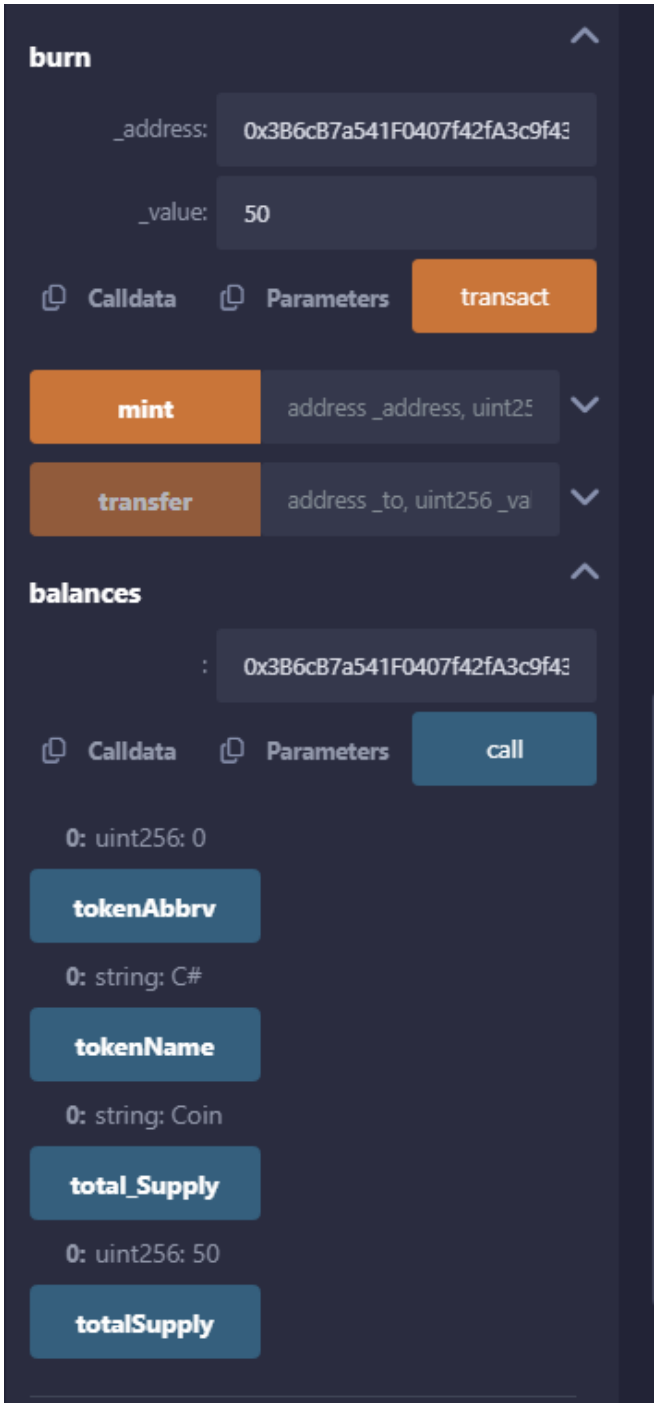

Figure 11: Burn 50 tokens. Total Supply Changes to 50.

We can see the transaction details on Etherscan - Blockchain Exploree. We can either click the link in the output terminal or can visit Etherscan for testnet and search for our contract address.

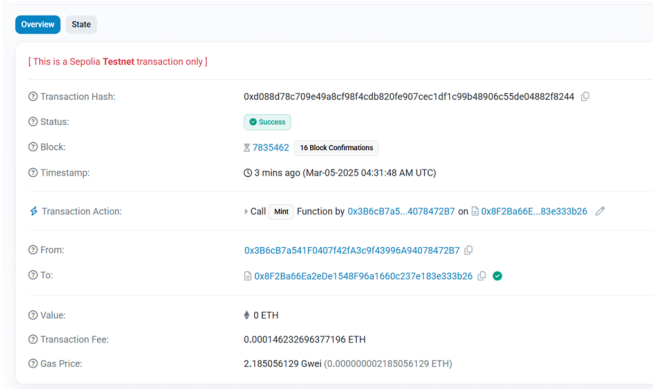

Figure 12: Etherscan - Mint transaction

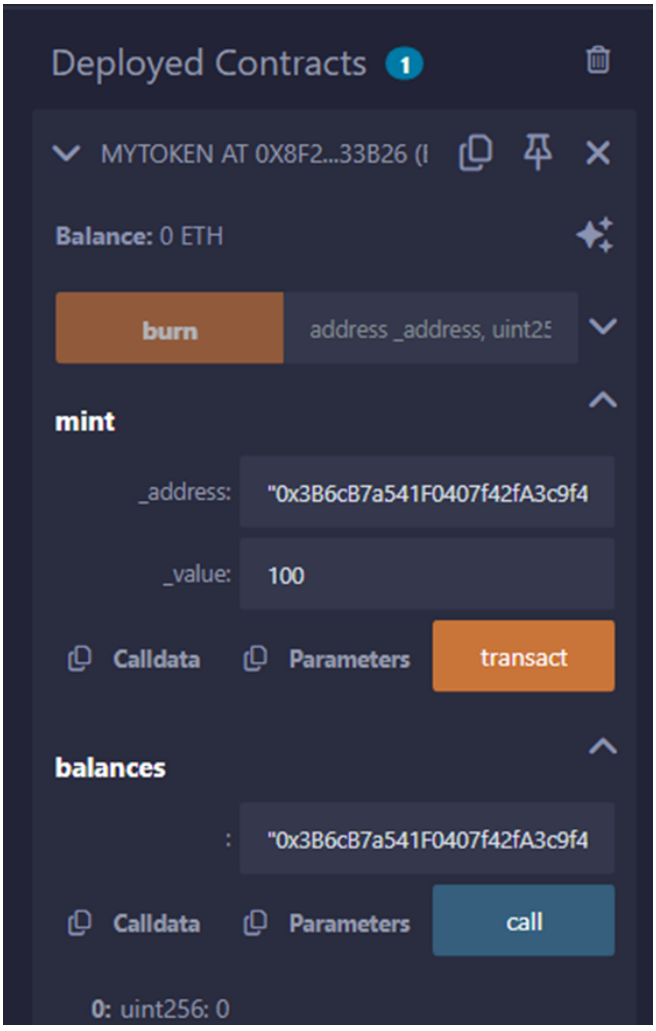

Figure 13: Etherscan - Burn Transaction

Note: This process was performed both on Remix VM (for saving ETH) and then Sepolia Testnet. The Etherscan screenshots correspond to the latter, while the other screenshots are from the former.
The following screenshots are from our first attempt on Sepolia Testnet. We initially forgot to define the total supply function, so we repeated the process on Remix VM (earlier screenshots).
Till now we did it on Remix VM Cancun. Now let us try it on Sepolia testnet.

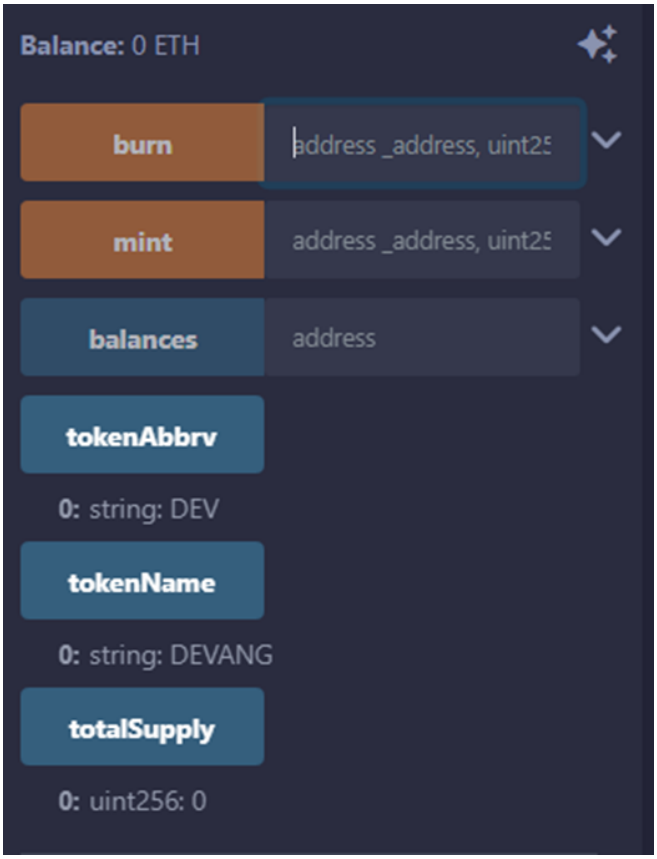

Figure 14: Initial State

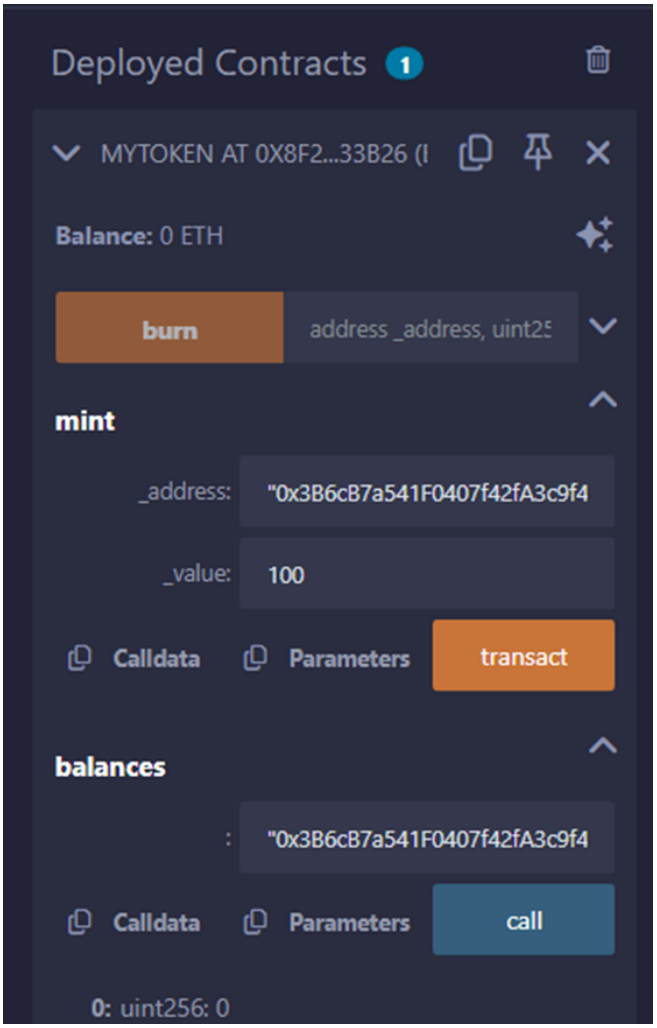

Figure 15: Mint 100 tokens

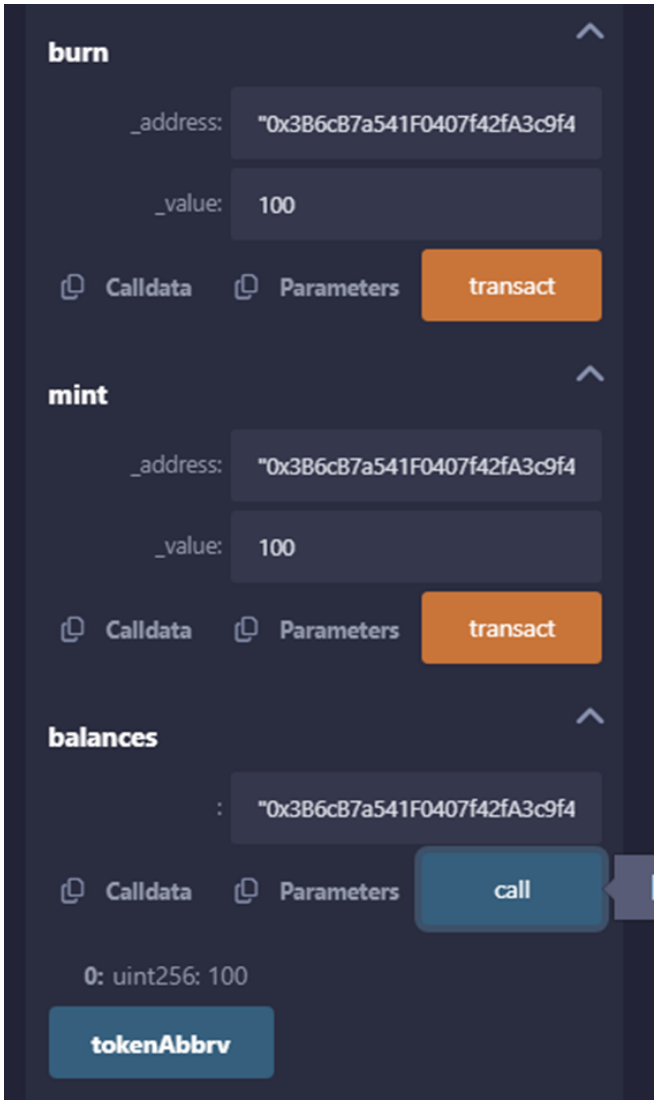

Figure 16: Burn 100 tokens

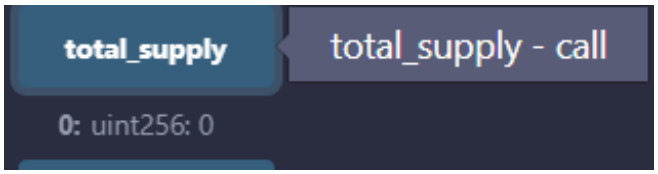

Figure 17: After Burning, total supply is 0

After having done some improvements in the solidity contract, we will now look at the final results: **Final Attempt:**
Contract Address: `0x8AeAf38d4080AfF05c381A60E34bA4d64a518402`
You can use the contract address to see the transaction details. We only provide the screenshot for Initial State to show the improvement :
Initial State:

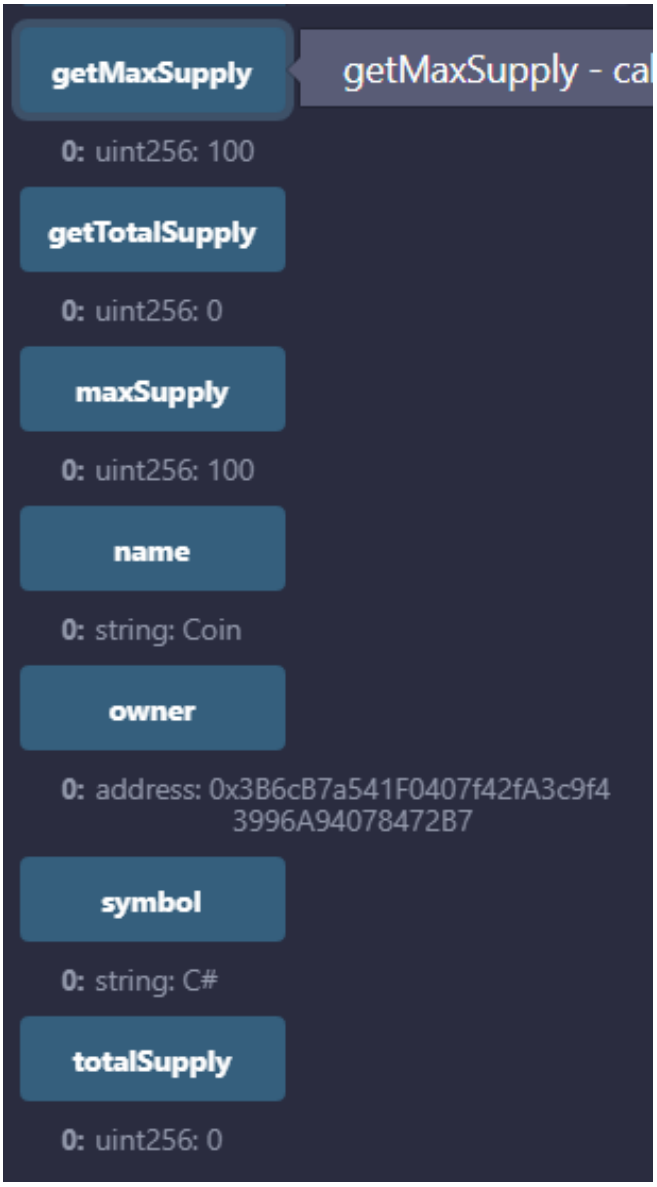

Figure 18: Initial State

3 Decentralized Voting System

Solidity Contract:

Listing 3: Voting System

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.8.18;
pragma abicoder v2;

/**
 * @title Ballot
 * @dev Implements voting process along with vote delegation
 */
contract Ballot {

    struct Voter {
        uint weight;
        bool voted;
        uint vote;
    }

    struct Candidate {
        string name;
        uint voteCount;
    }

    address public chairperson;
    mapping(address => Voter) public voters;
    Candidate[] public candidates;

    enum State { Created, Voting, Ended }
    State public state;

    // Events for transparency
    event VoterRegistered(address voter);
    event Voted(address voter, uint candidate);
    event ElectionStarted();
    event ElectionEnded();
    event CandidateAdded(string name);

    constructor(string[] memory candidateNames) {
        chairperson = msg.sender;
        voters[chairperson].weight = 1;
        state = State.Created;

        for (uint i = 0; i < candidateNames.length; i++) {
            candidates.push(Candidate({name: candidateNames[i], voteCount: 0}));
        }
    }

    // Modifiers
    modifier onlyChairperson() {
        require(msg.sender == chairperson, "Only␣chairperson␣can␣perform␣this␣action");
        _;
    }

    modifier inState(State _state) {
        require(state == _state, "Invalid␣state␣for␣this␣action");
        _;
    }

    // Allow users to self-register as voters
    function registerAsVoter() public {
        require(voters[msg.sender].weight == 0, "Already␣registered.");
        voters[msg.sender].weight = 1;
        emit VoterRegistered(msg.sender);
    }

    function addCandidate(string memory candidateName) public onlyChairperson inState(State.Created) {
        candidates.push(Candidate({name: candidateName, voteCount: 0}));
```

```
        emit CandidateAdded(candidateName);
    }

    function startVote() public onlyChairperson inState(State.Created) {
        state = State.Voting;
        emit ElectionStarted();
    }

    function endVote() public onlyChairperson inState(State.Voting) {
        state = State.Ended;
        emit ElectionEnded();
    }

    function vote(uint candidate) public inState(State.Voting) {
        Voter storage sender = voters[msg.sender];
        require(sender.weight != 0, "No right to vote");
        require(!sender.voted, "Already voted");
        sender.voted = true;
        sender.vote = candidate;
        candidates[candidate].voteCount += sender.weight;
        emit Voted(msg.sender, candidate);
    }

    function getCurrentVotes() public view returns (Candidate[] memory) {
        return candidates;
    }

    function winningCandidate() public view inState(State.Ended) returns (string memory winnerName_) {
        uint winningVoteCount = 0;
        for (uint i = 0; i < candidates.length; i++) {
            if (candidates[i].voteCount > winningVoteCount) {
                winningVoteCount = candidates[i].voteCount;
                winnerName_ = candidates[i].name;
            }
        }
    }
}
```

Here is a summary :

- We have three states : Created,Voting and Ended. When we deploy it, it is in the Created state. When we start voting it is Voting and after voting ends it is Ended.
- We use the states to impose restrictions. Like,WinningCandidate can be found only after voting ends that is Ended state. Voting can be done only in Voting state.
- We can see real time votes by calling getCandidateVotes.
- Voters can self register by resgisterAsVoter. Note however the chairperson is automatically registerd as voter and no voter can register twice.
- Only cjhairperson can Start and End Voting

To see it in action, we need atleat two people : one will be chairperson and other will be the normal voter.

4 NFT Marketplace Development

ERC721 is Non-Fungible tokens standard (NFT). Till now we looked at Fungible tokens(ERC20). The difference is very simple : Fungible tokens all have the same value and can be exchanged one to one. But NFTs cannot. For example, a 1 rupee coin can be exchanged with a 1 rupee coin since the hold the same value. But we cannot exchange a 1kg potato with a 1kg tomato (they may have different market rate - 1kg tomato is much expensive than 1kg potato). Generally we use art pieces as NFTs (not vegetables!). Let us take a look at the solidity contract now :

Solidity Contract:

Listing 4: ERC721 Token

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts@5.0.0/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts@5.0.0/token/ERC721/extensions/ERC721URIStorage.sol";
import "@openzeppelin/contracts@5.0.0/token/ERC721/extensions/ERC721Burnable.sol";
import "@openzeppelin/contracts@5.0.0/access/Ownable.sol";

contract MyToken is ERC721, ERC721URIStorage, ERC721Burnable, Ownable {
    constructor(address initialOwner)
        ERC721("MyToken", "MTK")
        Ownable(initialOwner)
    {}

    function safeMint(address to, uint256 tokenId, string memory uri)
        public
        onlyOwner
    {
        _safeMint(to, tokenId);
        _setTokenURI(tokenId, uri);
    }

    // The following functions are overrides required by Solidity.

    function tokenURI(uint256 tokenId)
        public
        view
        override(ERC721, ERC721URIStorage)
        returns (string memory)
    {
        return super.tokenURI(tokenId);
    }

    function supportsInterface(bytes4 interfaceId)
        public
        view
        override(ERC721, ERC721URIStorage)
        returns (bool)
    {
        return super.supportsInterface(interfaceId);
    }
}
```

This is a very simple contract using which we can mint ne NFTs and burn them also. There is also one more thing : tokenURI. When we perform "mint", we add a new NFT into the supply i.e also in the block representing the transaction. Now say I put the image of size 1MB as a NFT, will it not be memory expensive? To deal with this issue we store the image on IPFS instead : IPFS is InterPlanetart File System (Peer to Peer). We then take the URI(address) of the NFT metadata(name,description,image address json file). So the process is as follows:

- Upload the image on IPFS
- Upload the JSON file describing the image.

Note : It need not be an image always.You can have anything.

The format of JSOn file is same as :

```
{
    "name": "[Name]",
    "description": "[Description]",
    "image": "https://ipfs.io/ipfs/[CID]"
}
```

Listing 1: JSON example

We used Pinata in our case. It is very simple to get started.
Now deploy the code on Sepolia Testnet.

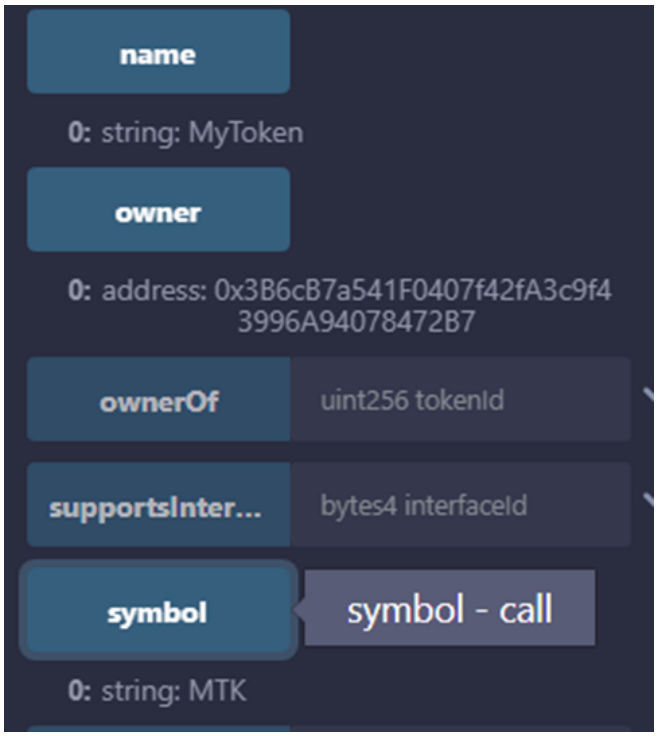

Figure 19: Token Name, Symbol, Owner

The "owner" is the address of the one owning the token. Each NFT "token" is owned by someone (unlike Fungible tokens where each address is mapped to a balance).

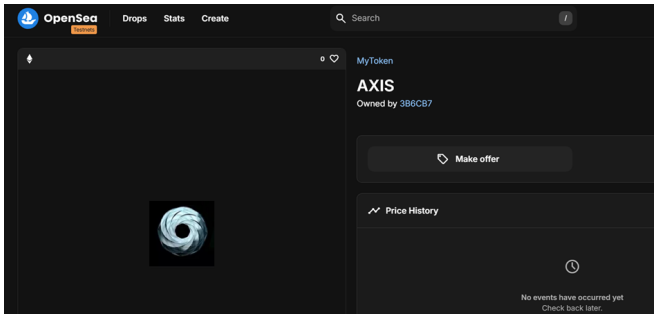

Figure 21: Opensea for Testnets

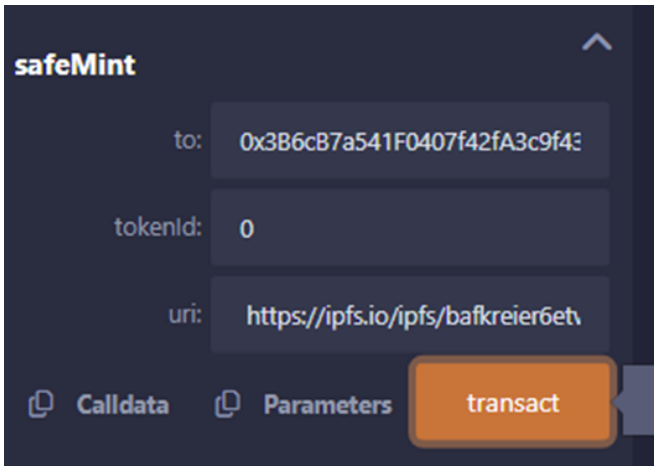

Figure 20: Mint new NFT - We mint the image of AXIS icon - It is a India's Largest Technical Fest!(or will become soon

Contract Address : 0xCfDAd3b53E9548DF28cC892f5d0f0495138F6193.
Go to Etherscan for more details.
We minted NFT. But where did they go? We want to see the majestic logo of AXIS that we minted! Go to Opensea for testnet. Here , you just have to search the contract address to see the collection of NFTs we minted. Now let us build a web interface where we can mint tokens easily as well as see our minted NFTs. We will also allow to burn them.

- Start by some pre-requesites like installing npm package manager.
- Create a new next.js application. We name it "nft".
- Go to src/pages folder.
- Edit the index.tsx (It is the start point or entry point which will go to NFTMinter.tsx, our main file) : **index.tsx:**

```
import NFTMinter from "@/components/NFTMinter";

export default function Home() {
  return (
    <main className="flex␣min-h-screen␣flex-col␣items-center␣justify-center␣p-6">
      <NFTMinter />
    </main>
  );
}
```

- Now move back to "src" folder and create a "components" folder.
- In the filder create a file "NFTMinter.tsx (This is the minter and burne we wrote - it depends on the contract address, ABI. You have to change them for your contract)" :

```
'use client';

import { useState, useEffect } from 'react';
import { ethers } from 'ethers';
import Button from '@/components/ui/Button';
import { Card, CardContent } from '@/components/ui/Card';
import Input from '@/components/ui/Input';


const contractAddress = '[CONTRACT ADDRESS]';
const contractABI = [
    //YOUR ABI - EXAMPLE ABI IS GIVEN
  {
    "inputs": [
      { "internalType": "address", "name": "to", "type": "address" },
      { "internalType": "uint256", "name": "tokenId", "type": "uint256" },
      { "internalType": "string", "name": "uri", "type": "string" }
    ],
    "name": "safeMint",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "inputs": [{ "internalType": "uint256", "name": "tokenId", "type": "uint256" }],
    "name": "tokenURI",
    "outputs": [{ "internalType": "string", "name": "", "type": "string" }],
    "stateMutability": "view",
    "type": "function"
  },
  {
    "inputs": [{ "internalType": "uint256", "name": "tokenId", "type": "uint256" }],
    "name": "burn",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  }
];

export default function NFTMinter() {
  const [walletAddress, setWalletAddress] = useState('');
  const [tokenId, setTokenId] = useState('');
  const [mintURI, setMintURI] = useState('');
  const [isConnected, setIsConnected] = useState(false);
  const [nftList, setNftList] = useState([]);

  async function connectWallet() {
    if (window.ethereum) {
      const provider = new ethers.BrowserProvider(window.ethereum);
      const signer = await provider.getSigner();
      setWalletAddress(await signer.getAddress());
      setIsConnected(true);
      fetchAllNFTs();
    } else {
      alert('Please install MetaMask');
    }
  }

  async function mintNFT() {
    if (!isConnected) return alert('Connect wallet first!');
    const provider = new ethers.BrowserProvider(window.ethereum);
    const signer = await provider.getSigner();
    const contract = new ethers.Contract(contractAddress, contractABI, signer);
    try {
      const tx = await contract.safeMint(walletAddress, tokenId, mintURI);
      await tx.wait();
```

```
      alert('NFT␣Minted!');
      fetchAllNFTs();
    } catch (error) {
      console.error(error);
      alert('Error␣minting␣NFT');
    }
  }

  async function burnNFT(tokenId) {
    if (!isConnected) return alert('Connect␣wallet␣first!');
    const provider = new ethers.BrowserProvider(window.ethereum);
    const signer = await provider.getSigner();
    const contract = new ethers.Contract(contractAddress, contractABI, signer);
    try {
      const tx = await contract.burn(tokenId);
      await tx.wait();
      alert('NFT␣Burned!');
      fetchAllNFTs();
    } catch (error) {
      console.error(error);
      alert('Error␣burning␣NFT');
    }
  }

  async function fetchAllNFTs() {
    if (!isConnected) return;
    const provider = new ethers.BrowserProvider(window.ethereum);
    const contract = new ethers.Contract(contractAddress, contractABI, provider);
    try {
      const nftData = [];
      for (let id = 0; id < 100; id++) { // Adjust max token ID range accordingly
        try {
          const uri = await contract.tokenURI(id);
          if (uri.endsWith('.png') || uri.endsWith('.jpg') || uri.endsWith('.jpeg')) {
              nftData.push({ tokenId: id, uri });
        } else {
        const response = await fetch(uri);
        const metadata = await response.json();
        nftData.push({ tokenId: id, uri: metadata.image });
        }

        } catch (error) {
          break;
        }
      }
      setNftList(nftData);
    } catch (error) {
      console.error(error);
    }
  }

  useEffect(() => {
    if (isConnected) fetchAllNFTs();
  }, [isConnected]);

  return (
    <div className="p-6">
      <Button onClick={connectWallet}>{isConnected ? 'Connected' : 'Connect␣Wallet'}</
          Button>
      <p className="mt-4">Wallet: {walletAddress || 'Not␣connected'}</p>

      <Card className="mt-4">
        <CardContent>
          <h2 className="text-lg␣font-bold">Mint NFT</h2>
          <Input placeholder="Token␣ID" onChange={(e) => setTokenId(e.target.value)}
              className="mt-2" />
          <Input placeholder="Token␣URI" onChange={(e) => setMintURI(e.target.value)}
              className="mt-2" />
          <Button onClick={mintNFT} className="mt-2">Mint</Button>
        </CardContent>
      </Card>
```

```
      <Card className="mt-4">
        <CardContent>
          <h2 className="text-lg font-bold">All Minted NFTs</h2>
          {nftList.length === 0 ? (
            <p>No NFTs minted yet.</p>
          ) : (
            nftList.map((nft, index) => (
              <div key={index} className="mt-2">
                <p>Token ID: {nft.tokenId}</p>
                <img src={nft.uri} alt={`NFT ${nft.tokenId}`} className="w-32 h-32 mt-2"
                    />
                <Button onClick={() => burnNFT(nft.tokenId)} className="mt-2">Burn</Button
                    >
              </div>
            ))
          )}
        </CardContent>
      </Card>
    </div>
  );
}
```

We can get the ABI from the compile window in Remix IDE at the bottom.

- Define the ui by creating a folder "ui" in the components folder and defining Button, Card and Input.
- Same goes for global.css in the src/styles folder.

We have successfully set up the project. Now go to cmd and then to the path to your project. Type **"npm run dev" or "npm start"** to run the application. Go to localhost:3000in the browser.

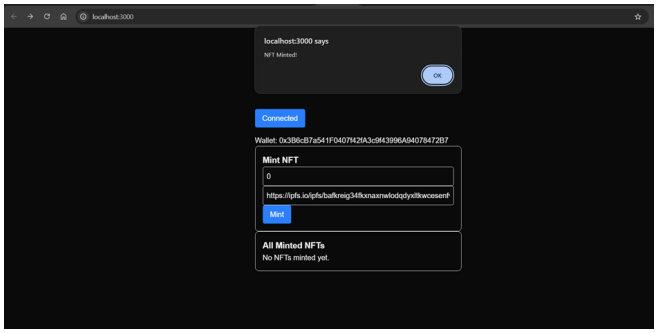

Figure 22: Mint NFT

Mint the NFT by providing token Id(no repetition allowed) and the URI.
You can see the minted NFT list also.

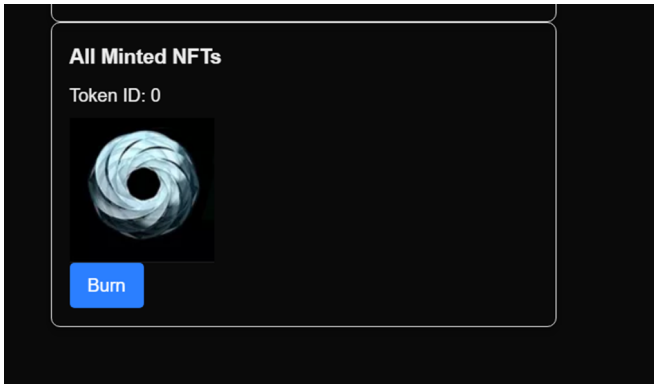

Figure 23: All Minted NFTs

5 Hyperledger Fabric – Setting Up a Private Blockchain Network

5.1 Prerequisites for Hyperledger Fabric

Before setting up Hyperledger Fabric, ensure that the following dependencies are installed on your Linux system.

- Git Install the latest version of Git if it is not already installed.

```
sudo apt-get install git
```

- cURL Install the latest version of cURL if it is not already installed.

```
sudo apt-get install curl
```

- Docker Install the latest version of Docker if it is not already installed.

```
sudo apt-get -y install docker-compose
```

  Once installed, confirm that the latest versions of both Docker and Docker Compose executables were installed.

```
docker --version
Docker version 19.03.12, build 48a66213fe

docker-compose --version
docker-compose version 1.27.2, build 18f557f9
```

  Make sure the Docker daemon is running.

```
sudo systemctl start docker
```

  Optional: If you want the Docker daemon to start when the system starts, use the following:

```
sudo systemctl enable docker
```

  Add your user to the Docker group.

```
sudo usermod -a -G docker <username>
```

- Go : https://go.dev/doc/install Follow : https://dev.to/karleeov/wsl-install-go-27ji for help.
- Jq : https://jqlang.org/download/

5.2 Install Fabric and Fabric Samples

Follow : https://hyperledger-fabric.readthedocs.io/en/release-2.5/install.html

5.3 Getting Started - Run Fabric

Follow : https://hyperledger-fabric.readthedocs.io/en/release-2.5/getting_started_run_fabric.html

5.4 Writing your own chaincode - mychaincode

Follow : https://hyperledger-fabric.readthedocs.io/en/release-2.5/chaincode4ade.html
We had cloned the repository "fabric-samples" in one of the previous steps into a folder named "a5" (say). Go to the fabric-samples folder and create a new folder named "mychaincode" and put the chaincode fil in it (written in GO). Here the names "a5" and "mychaincode" can be anything. Also the name of the chaincode should be same as folder name (so you can execute the script easily).Go to mychaincode folder in the terminal and type "go mod init mychaincode" to generate go.mod file. Now run "go mod tidy" and "go mod vendor". You chanincode folder is ready to use. The other steps are the same as for "asset-transfer-basic" given in docs.The "invoke" and "query" will obviously change as it depends on chaincode.

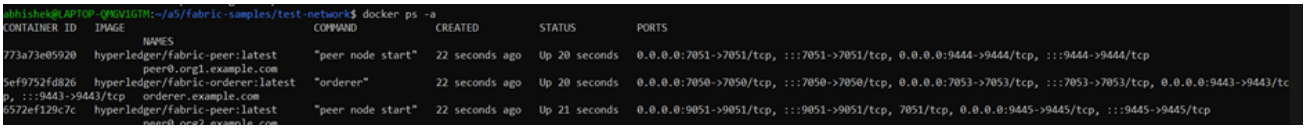

Figure 24: Running Network - Shows the containers

Figure 25: Get All Assets

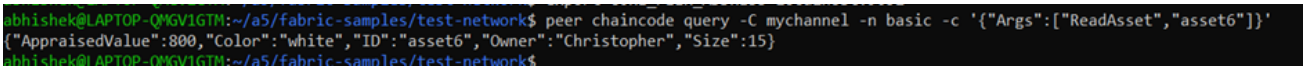

Figure 26: Read an Asset

Figure 27: Add an asset

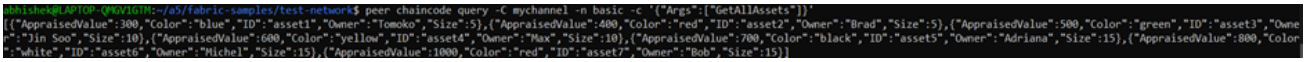

Figure 28: Read all Assets to confirm if the asset7 was added successfully

We did this for the chaincode of asset-transfer-basic as given in the docs. Now we do the same for the "mychaincode" we had written. Remember to do this : For custom chaincode do this in the folder where you stored the chaincode: go mod init [CHAINCODE NAME] go mod tidy , go mod vendor

mychaincode.go

```go
package main

import (
        "encoding/json"
        "fmt"
        "github.com/hyperledger/fabric-contract-api-go/contractapi"
)

// SmartContract provides functions for managing assets
type SmartContract struct {
        contractapi.Contract
}

// Asset represents an object in the ledger
type Asset struct {
        ID    string `json:"ID"`
        Color string `json:"Color"`
        Size  int    `json:"Size"`
        Owner string `json:"Owner"`
        Value int    `json:"Value"`
}

// CreateAsset adds a new asset to the ledger
func (s *SmartContract) CreateAsset(ctx contractapi.TransactionContextInterface, id string,
    color string, size int, owner string, value int) error {
        asset := Asset{
                ID:    id,
                Color: color,
                Size:  size,
                Owner: owner,
                Value: value,
        }
        assetJSON, err := json.Marshal(asset)
        if err != nil {
                return err
        }
        return ctx.GetStub().PutState(id, assetJSON)
}

// ReadAsset retrieves an asset from the ledger
func (s *SmartContract) ReadAsset(ctx contractapi.TransactionContextInterface, id string) (*
    Asset, error) {
        assetJSON, err := ctx.GetStub().GetState(id)
        if err != nil {
                return nil, fmt.Errorf("failed to read asset: %v", err)
        }
        if assetJSON == nil {
                return nil, fmt.Errorf("asset %s does not exist", id)
        }
        var asset Asset
        err = json.Unmarshal(assetJSON, &asset)
        if err != nil {
                return nil, err
        }
        return &asset, nil
}

func main() {
        chaincode, err := contractapi.NewChaincode(new(SmartContract))
        if err != nil {
                fmt.Printf("Error creating chaincode: %v", err)
                return
        }
        if err := chaincode.Start(); err != nil {
                fmt.Printf("Error starting chaincode: %v", err)
        }
}
```

This is the chaincode we wrote for "mychaincode" in GO langauge.

Let us have look at the Store and Retreive queries

Figure 29: Store an Asset

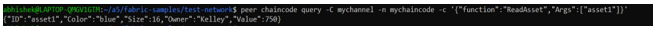

Figure 30: Retrive an Asset

Here, we store the "asset1" and then retrieve it successfully.

6 Hyperledger Fabric & IoT Integration

In this section, we utilize the same setup as in the previous example to demonstrate how Hyperledger Fabric can be integrated with IoT devices.

iot_chaincode

```
67  package main
68
69  import (
70      "encoding/json"
71      "fmt"
72      "github.com/hyperledger/fabric-contract-api-go/contractapi"
73  )
74
75  // IoTContract for managing sensor data
76  type IoTContract struct {
77      contractapi.Contract
78  }
79
80  // SensorData stores temperature and humidity
81  type SensorData struct {
82      Temperature string `json:"temperature"`
83      Humidity    string `json:"humidity"`
84  }
85
86  // StoreSensorData (Only Org1 can call this)
87  func (c *IoTContract) StoreSensorData(ctx contractapi.TransactionContextInterface, sensorID
        string, temperature string, humidity string) error {
88
89      // Get the caller's organization
90      clientMSPID, err := ctx.GetClientIdentity().GetMSPID()
91      if err != nil {
92          return fmt.Errorf("failed to get MSP ID: %v", err)
93      }
94
95      // Restrict write access to Org1 only
96      if clientMSPID != "Org1MSP" {
97          return fmt.Errorf("unauthorized: only Org1 can write data")
98      }
99
100     data := SensorData{Temperature: temperature, Humidity: humidity}
101     dataJSON, err := json.Marshal(data)  // Convert struct to JSON
102     if err != nil {
103         return fmt.Errorf("failed to marshal sensor data: %v", err)
104     }
105
106     // Store data on ledger
107     err = ctx.GetStub().PutState(sensorID, dataJSON)
108     if err != nil {
109         return fmt.Errorf("failed to store sensor data: %v", err)
110     }
111
112     return nil
```

```
}

// QuerySensorData (Only Org2 can call this)
func (c *IoTContract) QuerySensorData(ctx contractapi.TransactionContextInterface, sensorID
    string) (*SensorData, error) {

    // Get the caller's organization
    clientMSPID, err := ctx.GetClientIdentity().GetMSPID()
    if err != nil {
        return nil, fmt.Errorf("failed to get MSP ID: %v", err)
    }

    // Restrict read access to Org2 only
    if clientMSPID != "Org2MSP" {
        return nil, fmt.Errorf("unauthorized: only Org2 can read data")
    }

    dataJSON, err := ctx.GetStub().GetState(sensorID)
    if err != nil {
        return nil, fmt.Errorf("failed to get sensor data: %v", err)
    }
    if dataJSON == nil {
        return nil, fmt.Errorf("sensor data not found")
    }

    var data SensorData
    err = json.Unmarshal(dataJSON, &data)
    if err != nil {
        return nil, fmt.Errorf("failed to parse sensor data: %v", err)
    }

    return &data, nil
}

func main() {
    chaincode, err := contractapi.NewChaincode(new(IoTContract))
    if err != nil {
        fmt.Printf("Error creating IoT chaincode: %v", err)
        return
    }

    if err := chaincode.Start(); err != nil {
        fmt.Printf("Error starting IoT chaincode: %v", err)
    }
}
```

This chaincode defines two primary functions:

- **StoreSensorData**: Allows only clients from `Org1` (IoT devices) to write temperature and humidity data to the ledger.
- **QuerySensorData**: Grants read access exclusively to clients from `Org2` (authorized users).

This access control is enforced by verifying the client's Membership Service Provider Identifier (MSPID).

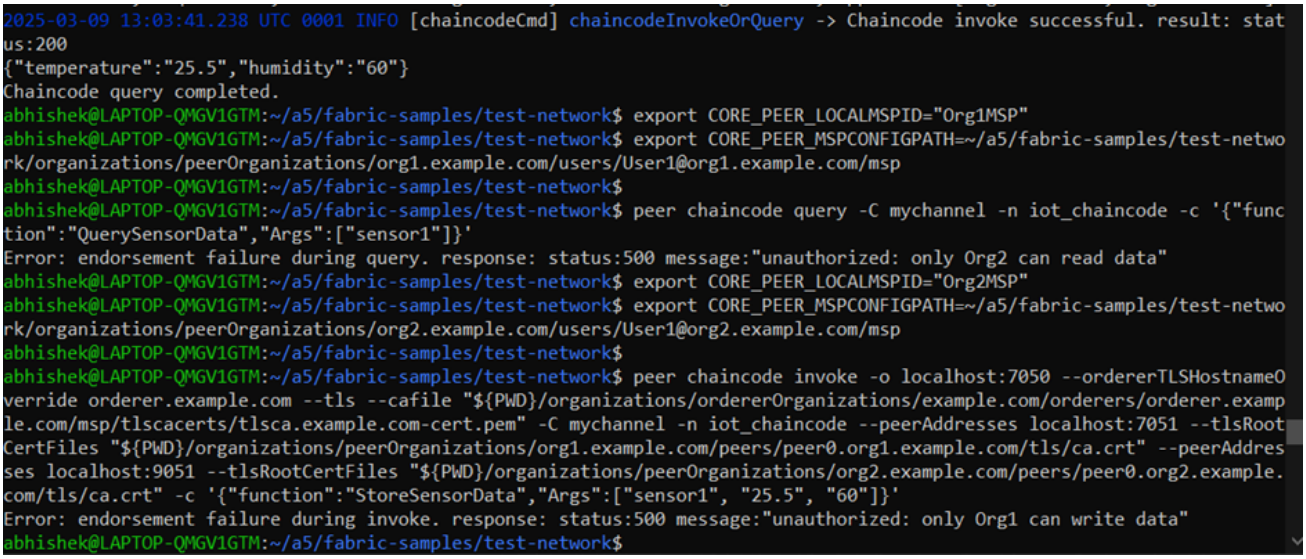

Figure 31: Illustration of read and write restrictions

To demonstrate this restriction, we first allow an `Org1` client (IoT device) to write data and an `Org2` client (authorized user) to read it. Subsequently, we swap their organization memberships—although simply attempting to perform unauthorized actions would suffice. This swap helps illustrate how membership changes affect access control. The system will return an error when an unauthorized action is attempted.