1. Anish Nimbalkar (an4338)
2. Sumedh Sandeep Parvatikar (sp7479)

Blockchain Project

Voting Contract Application **12th May 2024**

https://github.com/Anish565/VotingContractApplication

Smart Contract

Smart contracts are self-executing scripts which capture the terms and conditions of a contract. They execute on the blockchain based on certain predefined rules and conditions. From a functional point of view, smart contracts act as third party intermediaries and simplify the business and trade between both anonymous and identified parties, sometimes both without the need of middlemen. In the blockchain world, smart contracts enable transparent, secure, and efficient transactions in various domains like finance, supply chain, healthcare and governance.

Introduction

In our modern globalized world, we have no modern mechanism to check for spurious voting and at the same time withholding the pillars like transparency, and security. To overcome this challenge, we decided to leverage the advantages of smart contracts and blockchain technology, and explore ways to pave the way for new Voting applications. In our project we use a **Sepholia test network** to store our smart contract and execute voting and other utility functions through a testing script created to document the behavior of the Voting application.

Working

We create a smart contract called VotingContract which contains the details about the candidates and standard logics for performing certain actions like Increasing the vote count, checking for duplicity, checking if the voter is a valid candidate, and more. We define a basic framework which when provided with the candidate names and voting parameters executes the contract and stores them in the **Sepholia test network** blockchain (via Alchemy). When we are deploying the contract, we pass the candidate names which will be used for voting thus not allowing any other updates to the candidate names throughout the voting process.
For the testing of our contract, we created a separate script to interact with our smart contract functions and store the response. In our test script we can also count the number of votes each candidate has received till that particular point in time by invoking the specific function.

Smart Contract Details
In our smart contract we check for duplicity by using the name + drivers license + address of the candidate in combination to create a unique token to associate with each voter. No two voters can have the same token in our system thus distinguishing duplicate voters. For our specific use cases, we relied on using the drivers license, but depending on one's requirements and objectives we can change this particular aspect to include other attributes like phone numbers, SSNs, employee ids, etc (which can be used to uniquely identify). Apart from duplicity we also validate if the candidate being voted for is a valid candidate by quickly skimming through the list of the candidates in our smart contract.
Our smart contract stores the votes, voter tokens and the candidate names which would internally be stored in the blockchain once deployed.

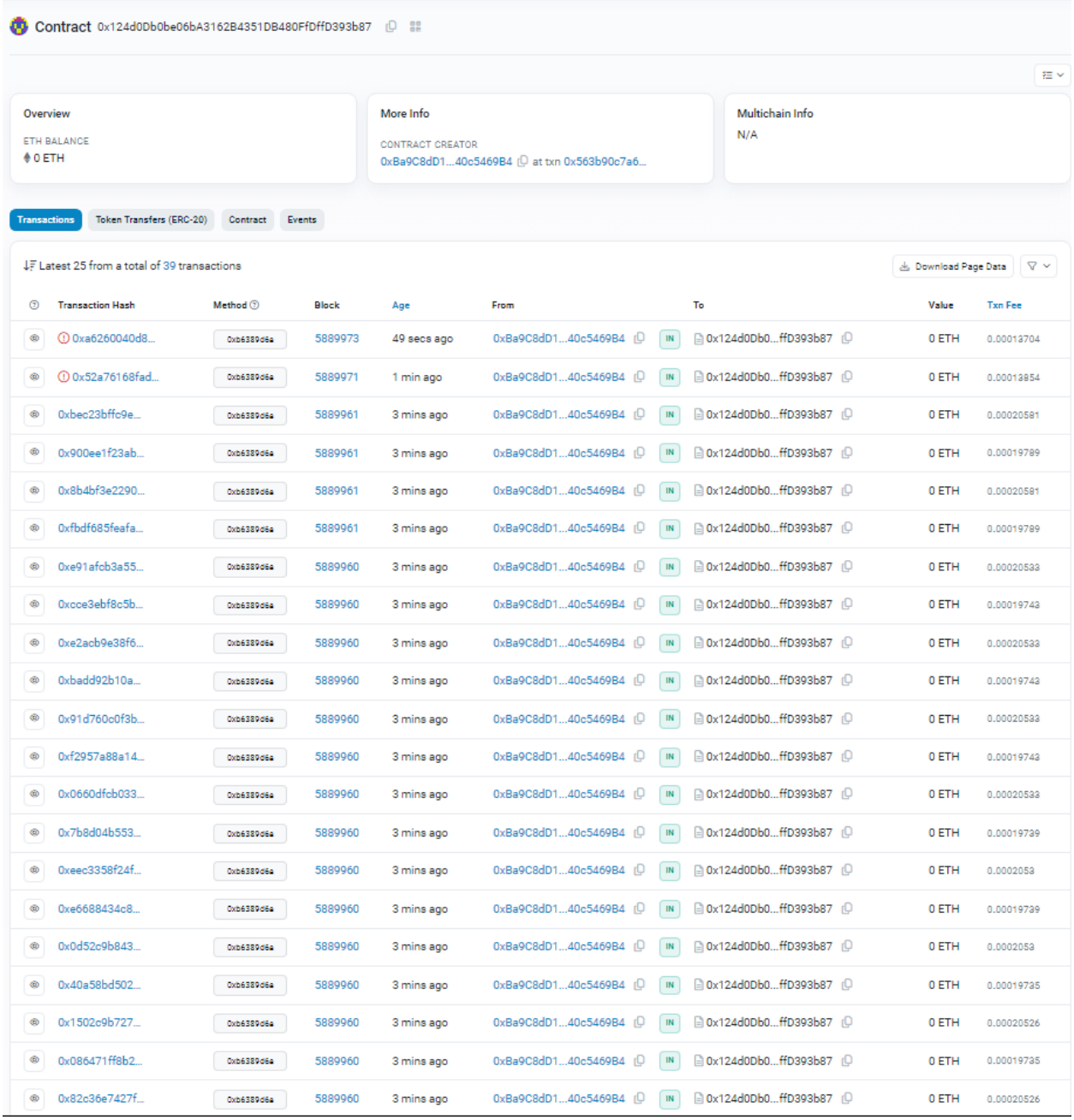
Results

So, we deploy the contract using the following command:

```
npx hardhat run scripts/deploy.js --network sepolia
```

Once we deployed this, we got the location at where the contract was deployed. And we were able to verify that using the link to the Etherscan website. Our contract address is **0x124d0Db0be06bA3162B4351DB480FfDffD393b87**.

We get the following transactions all logged on etherscan



Contract 0x124d0Db0be06bA3162B4351DB480FfDffD393b87

Overview
ETH BALANCE
0 ETH

More Info
CONTRACT CREATOR
0xBa9C8dD1...40c5469B4 at txn 0x563b90c7a6...

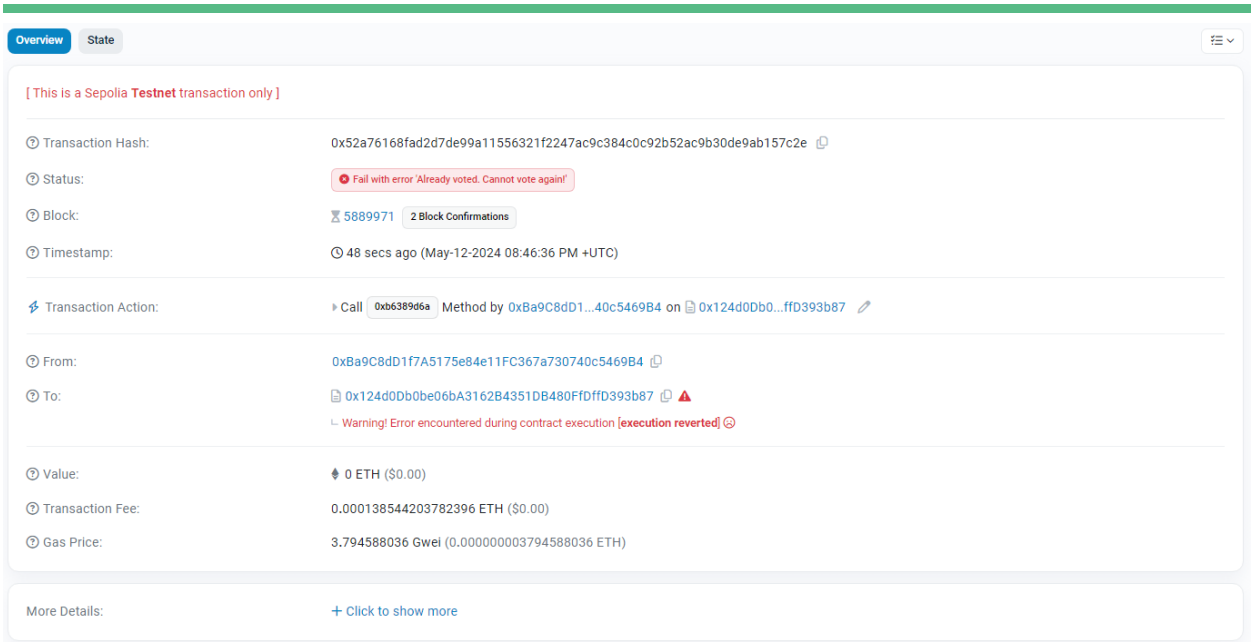
Multichain Info
N/A

Transactions | Token Transfers (ERC-20) | Contract | Events

Latest 25 from a total of 39 transactions — Download Page Data

Transaction Hash	Method	Block	Age	From	To	Value	Txn Fee
0xa6260040d8...	0xb6389d6a	5889973	49 secs ago	0xBa9C8dD1...40c5469B4	0x124d0Db0...ffD393b87	0 ETH	0.00013704
0x52a76168fad...	0xb6389d6a	5889971	1 min ago	0xBa9C8dD1...40c5469B4	0x124d0Db0...ffD393b87	0 ETH	0.00013854
0xbec23bffc9e...	0xb6389d6a	5889961	3 mins ago	0xBa9C8dD1...40c5469B4	0x124d0Db0...ffD393b87	0 ETH	0.00020581
0x900ee1f23ab...	0xb6389d6a	5889961	3 mins ago	0xBa9C8dD1...40c5469B4	0x124d0Db0...ffD393b87	0 ETH	0.00019789
0x8b4bf3e2290...	0xb6389d6a	5889961	3 mins ago	0xBa9C8dD1...40c5469B4	0x124d0Db0...ffD393b87	0 ETH	0.00020581
0xfbdf685feafa...	0xb6389d6a	5889961	3 mins ago	0xBa9C8dD1...40c5469B4	0x124d0Db0...ffD393b87	0 ETH	0.00019789
0xe91afcb3a55...	0xb6389d6a	5889960	3 mins ago	0xBa9C8dD1...40c5469B4	0x124d0Db0...ffD393b87	0 ETH	0.00020533
0xcce3ebf8c5b...	0xb6389d6a	5889960	3 mins ago	0xBa9C8dD1...40c5469B4	0x124d0Db0...ffD393b87	0 ETH	0.00019743
0xe2acb9e38f6...	0xb6389d6a	5889960	3 mins ago	0xBa9C8dD1...40c5469B4	0x124d0Db0...ffD393b87	0 ETH	0.00020533
0xbadd92b10a...	0xb6389d6a	5889960	3 mins ago	0xBa9C8dD1...40c5469B4	0x124d0Db0...ffD393b87	0 ETH	0.00019743
0x91d760c0f3b...	0xb6389d6a	5889960	3 mins ago	0xBa9C8dD1...40c5469B4	0x124d0Db0...ffD393b87	0 ETH	0.00020533
0xf2957a88a14...	0xb6389d6a	5889960	3 mins ago	0xBa9C8dD1...40c5469B4	0x124d0Db0...ffD393b87	0 ETH	0.00019743
0x0660dfcb033...	0xb6389d6a	5889960	3 mins ago	0xBa9C8dD1...40c5469B4	0x124d0Db0...ffD393b87	0 ETH	0.00020533
0x7b8d04b553...	0xb6389d6a	5889960	3 mins ago	0xBa9C8dD1...40c5469B4	0x124d0Db0...ffD393b87	0 ETH	0.00019739
0xeec3358f24f...	0xb6389d6a	5889960	3 mins ago	0xBa9C8dD1...40c5469B4	0x124d0Db0...ffD393b87	0 ETH	0.0002053
0xe6688434c8...	0xb6389d6a	5889960	3 mins ago	0xBa9C8dD1...40c5469B4	0x124d0Db0...ffD393b87	0 ETH	0.00019739
0x0d52c9b843...	0xb6389d6a	5889960	3 mins ago	0xBa9C8dD1...40c5469B4	0x124d0Db0...ffD393b87	0 ETH	0.0002053
0x40a58bd502...	0xb6389d6a	5889960	3 mins ago	0xBa9C8dD1...40c5469B4	0x124d0Db0...ffD393b87	0 ETH	0.00019735
0x1502c9b727...	0xb6389d6a	5889960	3 mins ago	0xBa9C8dD1...40c5469B4	0x124d0Db0...ffD393b87	0 ETH	0.00020526
0x086471ff8b2...	0xb6389d6a	5889960	3 mins ago	0xBa9C8dD1...40c5469B4	0x124d0Db0...ffD393b87	0 ETH	0.00019735
0x82c36e7427f...	0xb6389d6a	5889960	3 mins ago	0xBa9C8dD1...40c5469B4	0x124d0Db0...ffD393b87	0 ETH	0.00020526

The top of this list represents the transaction where we used the details of the same voter to cast a vote again. When we do this, we get the following response:



In this transaction, the function gets reverted with the message "`Already voted. Cannot vote again!`". This ensures the same person doesn't cast his vote again to the same or a different candidate.

Also, when we validate the candidates, the candidates already in the chain return true while those not in it return false. This can be seen as follows:

So, we initially gave the candidate names as `"Anish", "Sumedh"`, then when testing we validated for the names: `"Anish", "Sumedh", "John"`.
Then the output is as follows in this order:

```
Retrieved the contract..
Validating candidates..
true
true
false
```

And when we check for the final votes in the chain, we get the following information:

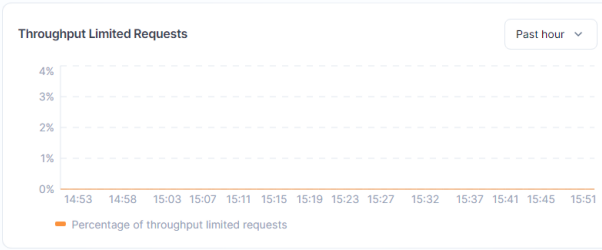
```
Total Votes..
Total votes for Anish : 16
Total votes for Sumedh : 16
```

The activity of the transactions can be seen on the Alchemy dashboard and request logs:

BlockChain Final Project

API key

99.7%	Success rate (last 1 hr)
99.7%	Success rate (last 24 hr)
--	Median response (last 5 min)
33.8	CU/s (avg over last 5 min)
1	Invalid requests (last 24 hr)
397	Total requests (last 24 hr)
0.1	Concurrent requests (last 5 min)
0%	Throughput limited rate (last 24 hr)



#	METHOD	APP	ERROR CODE	HTTP	RESPONSE TIME	SENT	NETWORK
1	eth_call	BlockChain Final …		200	18ms	5m ago	Ethereum Sepolia

PARAMS RAW REQUEST COPY

Data: 0x2f265cf753756d6564680000000000000000000000000000000000000…
From: 0xba9c8dd1f7a5175e84e11fc367a730740c5469b4
To: 0x124d0db0be06ba3162b4351db480ffdffd393b87
1: latest

RESULT RAW RESPONSE COPY

Value: 0x0000000000000000000000000000000000000000000000000000000…

#	METHOD	APP	ERROR CODE	HTTP	RESPONSE TIME	SENT	NETWORK
2	eth_chainId	BlockChain Final …		200	8ms	5m ago	Ethereum Sepolia
3	eth_call	BlockChain Final …		200	15ms	5m ago	Ethereum Sepolia
4	eth_chainId	BlockChain Final …		200	6ms	5m ago	Ethereum Sepolia
5	eth_sendRawTransaction	BlockChain Final …		200	18ms	5m ago	Ethereum Sepolia
6	eth_chainId	BlockChain Final …		200	7ms	5m ago	Ethereum Sepolia
7	eth_blockNumber	BlockChain Final …		200	5ms	5m ago	Ethereum Sepolia
8	eth_chainId	BlockChain Final …		200	8ms	5m ago	Ethereum Sepolia
9	eth_getTransactionCount	BlockChain Final …		200	17ms	5m ago	Ethereum Sepolia
10	eth_chainId	BlockChain Final …		200	7ms	5m ago	Ethereum Sepolia
11	eth_getBlockByNumber	BlockChain Final …		200	10ms	5m ago	Ethereum Sepolia
12	eth_gasPrice	BlockChain Final …		200	15ms	5m ago	Ethereum Sepolia
13	eth_chainId	BlockChain Final …		200	9ms	5m ago	Ethereum Sepolia
14	eth_chainId	BlockChain Final …		200	7ms	5m ago	Ethereum Sepolia
15	eth_call	BlockChain Final …		200	15ms	6m ago	Ethereum Sepolia
16	eth_chainId	BlockChain Final …		200	19ms	6m ago	Ethereum Sepolia
17	eth_call	BlockChain Final …		200	20ms	6m ago	Ethereum Sepolia
18	eth_chainId	BlockChain Final …		200	6ms	6m ago	Ethereum Sepolia
19	eth_sendRawTransaction	BlockChain Final …		200	20ms	6m ago	Ethereum Sepolia
20	eth_chainId	BlockChain Final …		200	7ms	6m ago	Ethereum Sepolia

Future Scope

The future scope of this project is vast and (hopefully) promising. By implementing a voting contract to automate the voting process in a decentralized fashion, we are attempting to create a secure, transparent, and decentralized system. Using this project as a stepping stone, there are a lot of potential future developments possible, such as:

1. Wider adoption of this technology by major institutions like the government, organizations, and communities for various types of voting, like political elections, corporate governance, etc.
2. Increased security during the voting process to ensure its integrity and security, reducing the risk of fraud or tampering.
3. The blockchain ledger provides a transparent and tamperproof record of all votes, allowing for real-time monitoring and verification of the voting results.
4. Potential integration with multiple blockchain networks can help increase decentralization and security.
5. And finally, development of a user-friendly interface to make the voting process accessible to a broader audience.

One of the major challenges will still be user adoption and integration with existing systems (like voting mechanisms across the world). However, once we overcome this issue, this technology has the potential to transform the way we conduct voting and decision-making processes.

Code

VotingContract.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract VotingContract {
    uint256 public chainId = 11155111; // Sepolia chain ID
    mapping(bytes32 => uint8) public votes;
    bytes32[] public candidatesNames;
    mapping(bytes32 => bool) public voters;

     function Voting(bytes32[] memory candidates) public {
        candidatesNames = candidates;
    }
```

```
    // constructor(bytes32[] memory _candidates) {
    //     candidatesNames = _candidates;
    // }

    function totalVotesFor(bytes32 candidate) public view returns (uint8) {
        if (!validCandidate(candidate)) {
            revert("Not a valid candidate! Exiting..");
        }
        return votes[candidate];
    }

    function voteForCandidate(bytes32 candidate, string memory _name,
string memory _driverLicense, string memory _address) public {
        if (!validCandidate(candidate)) {
            revert("Not a valid candidate. Exiting..");
        }
        bytes32 token = generateToken(string(abi.encodePacked(
            msg.sender,
            bytes32(abi.encodePacked(_name)),
            bytes32(abi.encodePacked(_driverLicense)),
            bytes32(abi.encodePacked(_address))
        )));
        if (voters[token]) {
            revert("Already voted. Cannot vote again!");
        }
        votes[candidate]++;
        voters[token] = true;
    }

    function validCandidate(bytes32 candidate) public view returns (bool) {
        for (uint i = 0; i < candidatesNames.length; i++) {
            if (candidatesNames[i] == candidate) {
                return true;
            }
        }
        return false;
    }

    function generateToken(string memory info) public pure returns
(bytes32){
        return keccak256(abi.encodePacked(info));
    }
```

```
}
```

Deploy.js

```
const { ethers } = require("hardhat");
// const { utils, abi } = require("hardhat").ethers;
// const { eth } = require("ethers");

async function main() {
    const VotingContract = await ethers.getContractFactory("VotingContract");
    const votingContract = await VotingContract.deploy(); // Provide candidate names here ["Anish", "Sumedh"]
    // const votingAddress = await votingContract.getAddress();
    console.log("Contract deploy phase passed!");

    console.log("VotingContract deployed to:", votingContract.target);
    //const candidate1 = await votingContract.generateToken("Anish");
    //console.log(candidate1);
    // //const candidate2 = await votingContract.generateToken("Sumedh");
    // //console.log(candidate2);
    const candidate1 = ethers.encodeBytes32String("Anish");
    const candidate2 = ethers.encodeBytes32String("Sumedh");
    console.log(candidate1, candidate2);
    // const candidate2 = utils.formatBytes32String("Sumedh");
    // console.log(candidate1, candidate2);
    await votingContract.Voting([candidate1, candidate2]);
//    await votingContract.deploy();

    console.log("VotingContract deployed to:", votingContract.target);
//
https://sepolia.etherscan.io/address/0x124d0Db0be06bA3162B4351DB480FfDffD393b87
}

main()
    .then(() => process.exit(0))
    .catch(error => {
        console.error(error);
        process.exit(1);
    });
```

Testing,js

```
const { ethers } = require("ethers");
const { artifacts } = require("hardhat/internal/lib/hardhat-lib");
const contractInfo =
require("../artifacts/contracts/VotingContract.sol/VotingContract.json");
const {API_KEY, API_URL, PRIVATE_KEY} = process.env
async function main() {
//    const intermediary = new ethers.providers;
//    console.log(intermediary)
    //            const            provider            =            new
ethers.providers.JsonRpcProvider("http://127.0.0.1:8545");              //
"http://localhost:8545" Connect to local Hardhat node
//                            const      contractAddress      =      await
provider.getContractAddress('YourContractName');
// Get Alchemy App URL
//const API_KEY = process.env.API_KEY;
// Alchemy provider
const provider = new ethers.providers.JsonRpcProvider(API_URL);
//  const  provider  =  new  ethers.providers.AlchemyProvider('eth-sepolia',
API_KEY);
// console.log(contractAddress);

    // Replace the contract address and ABI with your deployed contract's
address and ABI
    const contractAddress = "0x124d0Db0be06bA3162B4351DB480FfDffD393b87";
    // Another way below
    //const                          contract                          =
require("../artifacts/contracts/MyNFT.sol/MyNFT.json");
    // contract.abi
    const abi = contractInfo.abi;

    const signer =  new ethers.Wallet(PRIVATE_KEY, provider);
    // const signer = provider.getSigner(); // Get a signer (account) to
send transactions
    const  contract  =  await  new  ethers.Contract(contractAddress,  abi,
signer);
    console.log(contract);
    // Example: Triggering the voteForCandidate function
    const candidate = "Anish";
    const candidate2 = "Sumedh";
```

```
    const failCase = "John";

    console.log("Retrieved the contract..");
/*          const   byteCode   =   await   contract.generateToken(candidate,
{gasLimit:3000000});
    const    byteCode2    =    await    contract.generateToken(candidate2,
{gasLimit:3000000});
    const     byteCode3     =     await     contract.generateToken("Name",
{gasLimit:3000000});*/

    const byteCode = ethers.utils.formatBytes32String(candidate);
    const byteCode2 = ethers.utils.formatBytes32String(candidate2);
    const byteCode3 = ethers.utils.formatBytes32String(failCase);
    // const candidates = [byteCode, byteCode2];
    // console.log(byteCode);
    // await contract.Voting(candidates, {gasLimit:3000000});
    console.log("Validating candidates..");
    const testCandidate = await contract.validCandidate(byteCode);
    console.log(testCandidate)
    const testCandidate2 = await contract.validCandidate(byteCode2);
    console.log(testCandidate2)
    const testCandidate3 = await contract.validCandidate(byteCode3);
    console.log(testCandidate3)

    // console.log("Finished validating candidates..");

    // // byteCode Candidate Name, NameofVoter, Drivers License, Address
    //  const  testVariable  =  await  contract.voteForCandidate(byteCode,
"bnXlejdrhsSGEfr",     "BikdWlbsdfklWuB",     "hFdRLJcanTOsdflkbvzQLwRek",
{gasLimit:3000000});
    // console.log(testVariable);
    const   testVariable1   =   await   contract.voteForCandidate(byteCode,
"XyZabc123", "DefGhi456", "JklMno789", {gasLimit:3000000});
    console.log(testVariable1);
    const   testVariable2   =   await   contract.voteForCandidate(byteCode2,
"AbcDefGhi", "JklMnoPqr", "StuVwxYz1", {gasLimit:3000000});
    const   testVariable3   =   await   contract.voteForCandidate(byteCode,
"BcdEfgHij", "KlmNopQrs", "TuvWxyZ23", {gasLimit:3000000});
    const   testVariable4   =   await   contract.voteForCandidate(byteCode2,
"CdeFghIjk", "MnoPqrStu", "VwxYzA12", {gasLimit:3000000});
    const   testVariable5   =   await   contract.voteForCandidate(byteCode,
"DefGhiJkl", "NopQrsTuv", "WxyZab34", {gasLimit:3000000});
```

```
    const testVariable6 = await contract.voteForCandidate(byteCode2,
"EfgHijKlm", "PqrStuVwx", "XyZabc45", {gasLimit:3000000});
    const testVariable7 = await contract.voteForCandidate(byteCode,
"FghIjkLmn", "QrsTuvWxy", "YzAbcDe6", {gasLimit:3000000});
    const testVariable8 = await contract.voteForCandidate(byteCode2,
"GhiJklMno", "RstUvwXyZ", "AbcDefG7", {gasLimit:3000000});
    const testVariable9 = await contract.voteForCandidate(byteCode,
"HijKlmNop", "StuVwxYzA", "BcdEfgH8", {gasLimit:3000000});
    const testVariable10 = await contract.voteForCandidate(byteCode2,
"IjkLmnOpq", "TuvWxyZab", "CdeFghI9", {gasLimit:3000000});
    const testVariable11 = await contract.voteForCandidate(byteCode,
"JklMnoPqr", "UvwXyZabc", "DefGhiJ10", {gasLimit:3000000});
    const testVariable12 = await contract.voteForCandidate(byteCode2,
"KlmNopQrs", "VwxYzAbcD", "EfgHijK11", {gasLimit:3000000});
    const testVariable13 = await contract.voteForCandidate(byteCode,
"LmnOpqRst", "WxyZabcdE", "FghIjkL12", {gasLimit:3000000});
    const testVariable14 = await contract.voteForCandidate(byteCode2,
"MnoPqrStu", "XyZabcdeF", "GhiJklM13", {gasLimit:3000000});
    const testVariable15 = await contract.voteForCandidate(byteCode,
"NopQrsTuv", "YzAbcdefG", "HijKlmN14", {gasLimit:3000000});
    const testVariable16 = await contract.voteForCandidate(byteCode2,
"OpqRstUvw", "AbcdefGhi", "IjkLmnOp15", {gasLimit:3000000});
    const testVariable17 = await contract.voteForCandidate(byteCode,
"PqrStuVwx", "BcdefGhij", "JklMnoPq16", {gasLimit:3000000});
    const testVariable18 = await contract.voteForCandidate(byteCode2,
"QrsTuvWxy", "CdefGhijk", "KlmNopQr17", {gasLimit:3000000});
    const testVariable19 = await contract.voteForCandidate(byteCode,
"RstUvwXyZ", "DefGhijkl", "LmnOpqRs18", {gasLimit:3000000});
    const testVariable20 = await contract.voteForCandidate(byteCode2,
"StuVwxYzA", "EfgHijklm", "MnoPqrStu19", {gasLimit:3000000});
    const testVariable21 = await contract.voteForCandidate(byteCode,
"TuvWxyZab", "FghIjklmn", "NopQrsTuv20", {gasLimit:3000000});
    const testVariable22 = await contract.voteForCandidate(byteCode2,
"UvwXyZabc", "GhiJklmno", "OpqRstUvw21", {gasLimit:3000000});
    const testVariable23 = await contract.voteForCandidate(byteCode,
"VwxYzAbcD", "HijKlmnop", "PqrStuVwx22", {gasLimit:3000000});
    const testVariable24 = await contract.voteForCandidate(byteCode2,
"WxyZabcdE", "IjkLmnopq", "QrsTuvWxy23", {gasLimit:3000000});
    const testVariable25 = await contract.voteForCandidate(byteCode,
"XyZabcdeF", "JklMnopqr", "RstUvwXyZ24", {gasLimit:3000000});
    const testVariable26 = await contract.voteForCandidate(byteCode2,
"YzAbcdefG", "KlmNopqrs", "StuVwxYzA25", {gasLimit:3000000});
```

```
    const testVariable27 = await contract.voteForCandidate(byteCode, "ZabcdEfgH", "LmnOpqrst", "TuvWxyZab26", {gasLimit:3000000});
    const testVariable28 = await contract.voteForCandidate(byteCode2, "AbcdefGhi", "MnoPqrstu", "UvwXyZabc27", {gasLimit:3000000});
    const testVariable29 = await contract.voteForCandidate(byteCode, "BcdefGhij", "NopQrstuv", "VwxYzAbcD28", {gasLimit:3000000});
    const testVariable30 = await contract.voteForCandidate(byteCode2, "CdefGhijk", "OpqRstuvw", "WxyZabcdE29", {gasLimit:3000000});


    // // console.log("Failing here");
    //  const testVariable2 = await contract.voteForCandidate(byteCode2, "bnXlsSGEfr", "BikdWlbWuB", "hFdRLJcanTObvzQLwRek", {gasLimit:3000000});
    // console.log(testVariable2);
    // console.log("Failing here");
    setTimeout(function() {

    },240000);
    console.log("Total Votes..");
    // Example: Triggering the totalVotesFor function
    const totalVotes1 = await contract.totalVotesFor(byteCode);
    const totalVotes2 = await contract.totalVotesFor(byteCode2);
    // console.log(totalVotes);
    console.log("Total votes for", candidate, ":", totalVotes1);
    console.log("Total votes for", candidate2, ":", totalVotes2);
}

main()
    .then(() => process.exit(0))
    .catch(error => {
        console.error(error);
        process.exit(1);
    });
```

References

Alchemy notes - https://docs.alchemy.com/

Hardhat Notes - https://hardhat.org/docs

Npm packages - https://www.npmjs.com/