Pune Institute of Computer Technology,
Dhankawadi, Pune

Mini Project Report on

dApp for e-Voting using Blockchain

Submitted by

Sushilkumar Dhamane

Roll No: 41123

BE-1

Vinayak Jamadar

Roll No: 41137

BE-1

Sanket Jhavar

Roll No: 41138

BE-1

Under the guidance of

Prof. S. W. Jadhav

Department of Computer Engineering

Academic Year 2023-24

Contents

1 Title: dApp for e-Voting using Blockchain

2 Problem Statement

Develop a Blockchain based application dApp for e-voting system.

3 Abstract

For long, different e-voting systems have been provided with the goal of increasing security and minimizing cost Blockchain is a major breakthrough in the technological industry that provide immense secured platform. With the launch of Ethereum, a decentralized platform which nuns decentralized applications (DApps) on it, a secured voting system now seems passable. Many organizations have now shifted their focus on voting through blockchain platforms.There's a very high chance that a normal voting method won't lead to a clear majority. There can be many ways to deal with this issue which includes another voting process to take place which can be quite expensive in terms of time and resources. Here, we introduce the vote-trading concept where the votes can be redistributed to other candidates in case if there is no clear majority and also this 'majority' factor can be set by the organization according to their requirement. We discuss the design for the blockchain based preferential e-voting system using the Salidity programming language where instead of one vote per candidate, we provide the concept of giving preference to the candidates.

4 Learning Objectives

- **Understand Blockchain Technology**:
  - Learn the fundamental concepts of blockchain technology, including decentralization, immutability, and transparency.
- **Explore Blockchain-Based Voting Systems**:
  - Explore the potential of blockchain in revolutionizing voting systems and understand how it can address issues related to security and transparency.
- **Smart Contract Development**:
  - Gain knowledge and practical skills in developing smart contracts for a blockchain-based voting dApp using frameworks like Truffle.
- **Identity Verification and Security**:
  - Learn methods of identity verification in blockchain-based voting systems and understand the security measures involved in ensuring the integrity of votes.
- **User Interface Integration with MetaMask**:
  - Understand the role of MetaMask in providing a user-friendly interface for blockchain-based voting and learn how to integrate it into a dApp.
- **Testing with Ganache**:
  - Learn to use Ganache for local testing of smart contracts, ensuring that the voting dApp functions as expected before deployment.
- **Communication with Ethereum using Web3.js**:
  - Develop the ability to use Web3.js to connect the frontend of a voting dApp to the Ethereum blockchain, enabling secure and efficient communication.
- **Challenges and Scalability**:
  - Explore the challenges in implementing blockchain-based voting systems, including issues related to accessibility and scalability.

5 Learning Outcomes

- **Demonstrate a Solid Understanding of Blockchain Technology**:
  - Understand the core principles of blockchain technology, including its decentralized nature, immutability, and transparency.
- **Evaluate the Potential of Blockchain in Voting Systems**:
  - Assess the advantages and disadvantages of using blockchain for voting and make informed decisions regarding its implementation.
- **Develop Secure and Functional Smart Contracts**:
  - Create, test, and deploy smart contracts for a blockchain-based voting system with confidence in their security and functionality.
- **Implement Robust Identity Verification**:
  - Implement identity verification methods within a voting dApp to ensure that only eligible voters can participate.
- **Integrate User-Friendly Interfaces with MetaMask**:
  - Integrate MetaMask effectively to provide users with a seamless and secure experience in accessing and using the voting dApp.
- **Conduct Rigorous Testing with Ganache**:
  - Test smart contracts locally using Ganache to ensure the proper functioning and security of the voting dApp.
- **Establish Secure Communication with Ethereum via Web3.js**:
  - Successfully connect the frontend of a voting dApp to the Ethereum blockchain using Web3.js, enabling secure and efficient interactions.
- **Address Challenges in Implementation**:
  - Identify and propose solutions to the challenges faced when implementing blockchain-based voting systems, particularly in terms of scalability and accessibility.

6 S/W and H/ Requirements

Software Requirements

- **Operating System**:
  - The development environment should support Windows, macOS, or Linux. Common choices include Windows 10, macOS High Sierra or later, and popular Linux distributions like Ubuntu.
- **Integrated Development Environment (IDE)**:
  - Use a code editor or IDE suitable for blockchain development. Recommended choices include Visual Studio Code with relevant extensions, Remix, or Truffle's development environment.
- **Node.js**:
  - Install Node.js, a JavaScript runtime, for running JavaScript-based tools and libraries. Versions 10.x or later are generally recommended.
- **npm or Yarn**:
  - Node Package Manager (npm) or Yarn for managing and installing JavaScript libraries and packages.

- **Truffle Framework**:
  - Install Truffle, a development framework for Ethereum, typically through npm. This framework facilitates smart contract development, testing, and deployment.
- **Ganache**:
  - Download and install Ganache, a personal blockchain for Ethereum development, which allows for local testing of smart contracts.
- **MetaMask**:
  - Install the MetaMask browser extension for interacting with the Ethereum blockchain in a user-friendly manner.
- **Web3.js**:
  - Include Web3.js, a JavaScript library for Ethereum, in your project to facilitate communication with the Ethereum network.

Hardware Requirements

- **Processor**:
  - A multi-core processor (e.g., Intel Core i5 or AMD Ryzen) for efficient development and testing.
- **Memory (RAM)**:
  - A minimum of 8GB of RAM is recommended for running development tools and local blockchain nodes.
- **Storage**:
  - Ensure sufficient free disk space for your development environment, project files, and the blockchain's local data. A minimum of 20GB is recommended.
- **Solid State Drive (SSD)**:
  - An SSD is preferable over a traditional hard drive for faster compilation and better overall performance.
- **Internet Connection**:
  - A stable and reasonably fast internet connection is essential for accessing blockchain data and deploying smart contracts.
- **Web Browser**:
  - Ensure you have a modern web browser like Google Chrome or Mozilla Firefox for testing dApps and MetaMask integration.

7 Theory

- **Blockchain Technology:**
  - The foundation of this concept is blockchain, a distributed ledger technology known for its immutability and transparency.
  - Each vote is recorded as a transaction on the blockchain, forming a chain of blocks that are linked through cryptographic hashes.
  - This ledger is decentralized and maintained by a network of nodes, making it nearly impossible for any single entity to manipulate the data.
- **Immutable Records:**

  - Once a vote is recorded on the blockchain, it becomes immutable. This means that it cannot be altered, deleted, or tampered with.
  - This immutability ensures the integrity of the voting process, as past votes remain verifiable.

- **Transparency:**
  - The entire voting process, from registration to counting, is transparent and open to anyone with access to the blockchain.
  - Voters can verify that their votes have been recorded correctly, promoting trust in the system.

- **Security:**
  - The blockchain's cryptographic protocols and consensus mechanisms make it highly secure.
  - Votes are encrypted, and voters have a unique private key to access their vote and verify its authenticity.

- **Identity Verification:**
  - Blockchain-based voting systems can incorporate robust identity verification methods, such as biometrics, to ensure that only eligible voters can cast their ballots.

- **Smart Contracts:**
  - Smart contracts are self-executing contracts with the terms of the agreement directly written into code.
  - In a voting dApp, smart contracts can automate the entire process, from verifying the eligibility of voters to tallying the results.

- **Token-Based Voting:**
  - Some blockchain voting systems use tokens to represent votes. Each voter receives tokens corresponding to the number of votes they are entitled to.
  - These tokens are transferred via smart contracts, and the blockchain records the transactions.

- **Decentralization and Security:**
  - The decentralization of the network prevents a single point of failure, making it extremely secure against hacking and manipulation.

- **Auditability:**
  - All transactions are publicly accessible and auditable. This allows for independent verification of the voting results.

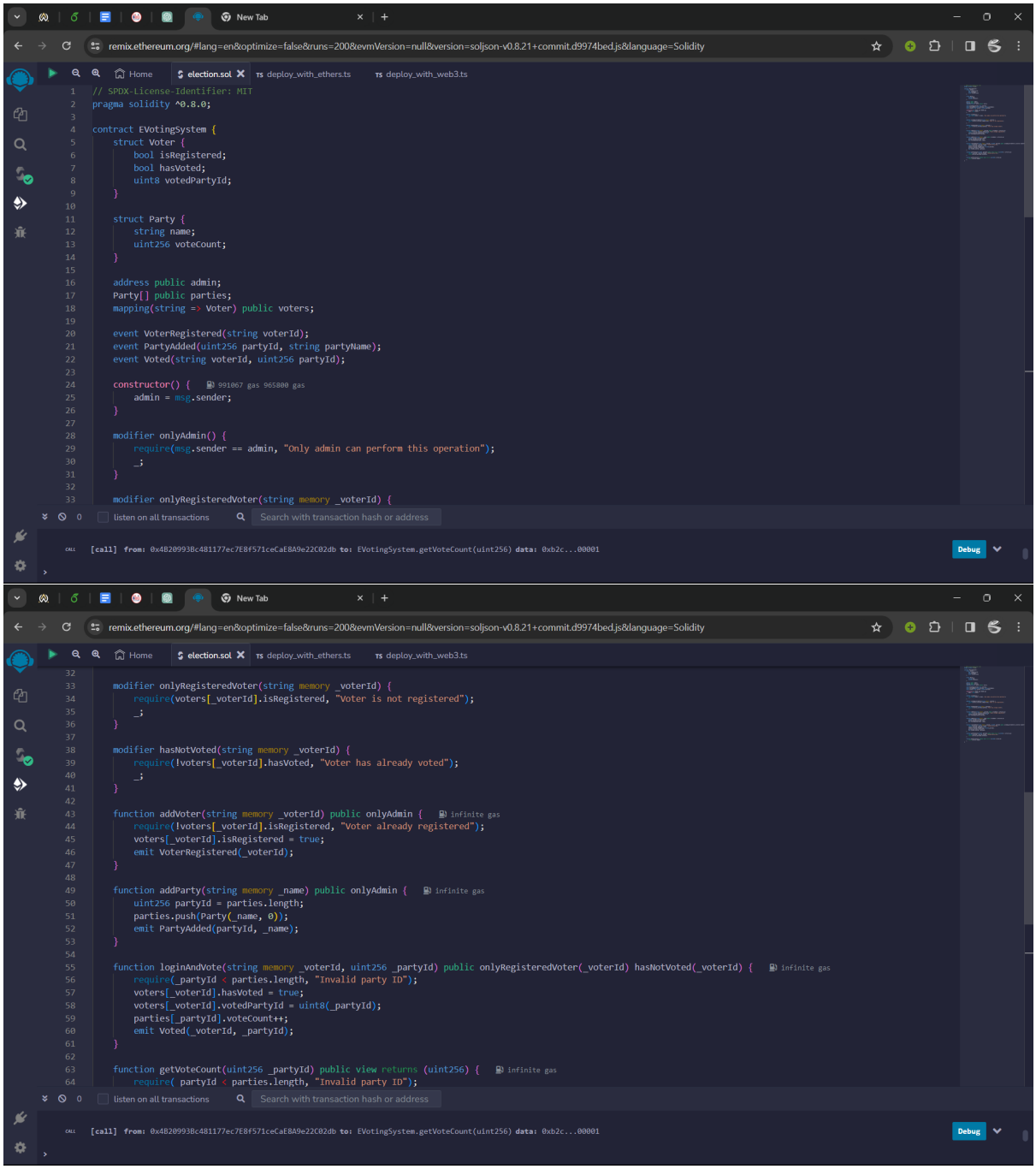
- **Challenges:**
  - Implementation of blockchain-based voting systems faces challenges, including accessibility, scalability, and ensuring voter privacy.
  - Scalability is a major concern when dealing with a large number of voters, as it requires significant computational power and bandwidth.

Additional Technologies and Tools

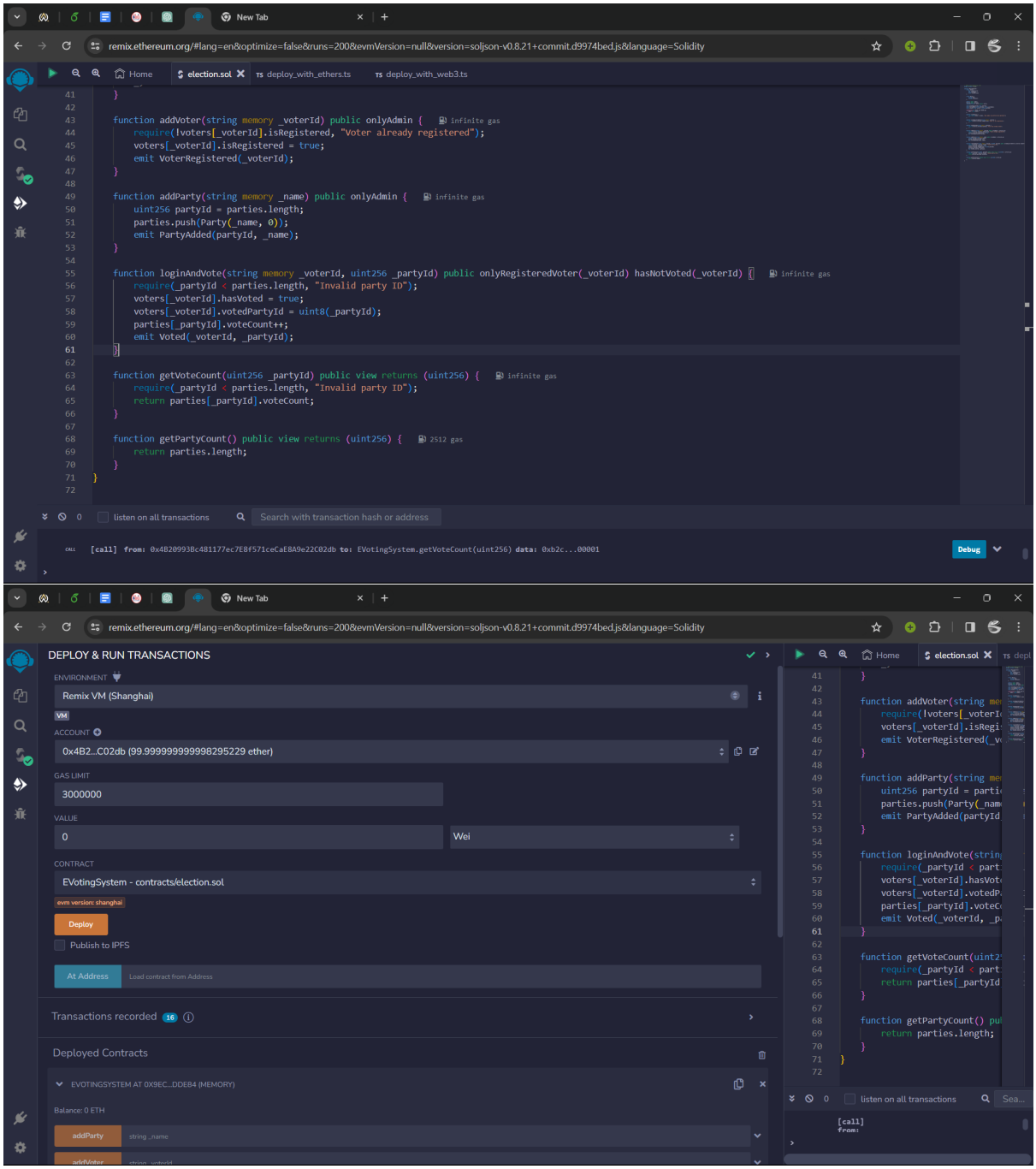
- **MetaMask:**
  - **Theory:** MetaMask is a browser extension and mobile application that serves as a bridge between users' web browsers and the Ethereum blockchain. It allows users to manage their Ethereum accounts, interact with dApps, and securely store their private keys.
  - **Role in Voting dApp:** MetaMask provides a user-friendly interface for voters to access the blockchain-based voting system. Users can sign transactions and interact with the dApp seamlessly through MetaMask.

- **Truffle:**
  - **Theory:** Truffle is a development framework for Ethereum that simplifies the process of building, testing, and deploying smart contracts. It provides a suite of tools for smart contract development and testing.
  - **Role in Voting dApp:** Truffle is used by developers to create, test, and deploy the smart contracts that power the voting system. It streamlines the development process and ensures the security and reliability of the smart contracts.
- **Ganache:**
  - **Theory:** Ganache is a personal blockchain for Ethereum development. It allows developers to create a local Ethereum network for testing smart contracts without the need to interact with the main Ethereum network.
  - **Role in Voting dApp:** Ganache is an essential tool for testing and debugging smart contracts in a controlled environment. It helps ensure that the voting dApp functions as expected before deploying it to the mainnet.
- **Web3.js:**
  - **Theory:** Web3.js is a JavaScript library that provides an interface for interacting with Ethereum nodes. It allows developers to connect their dApps to the Ethereum blockchain, send transactions, and retrieve data.
  - **Role in Voting dApp:** Web3.js is used by the frontend of the voting dApp to communicate with the Ethereum blockchain. It enables users to interact with the smart contracts on the blockchain securely.

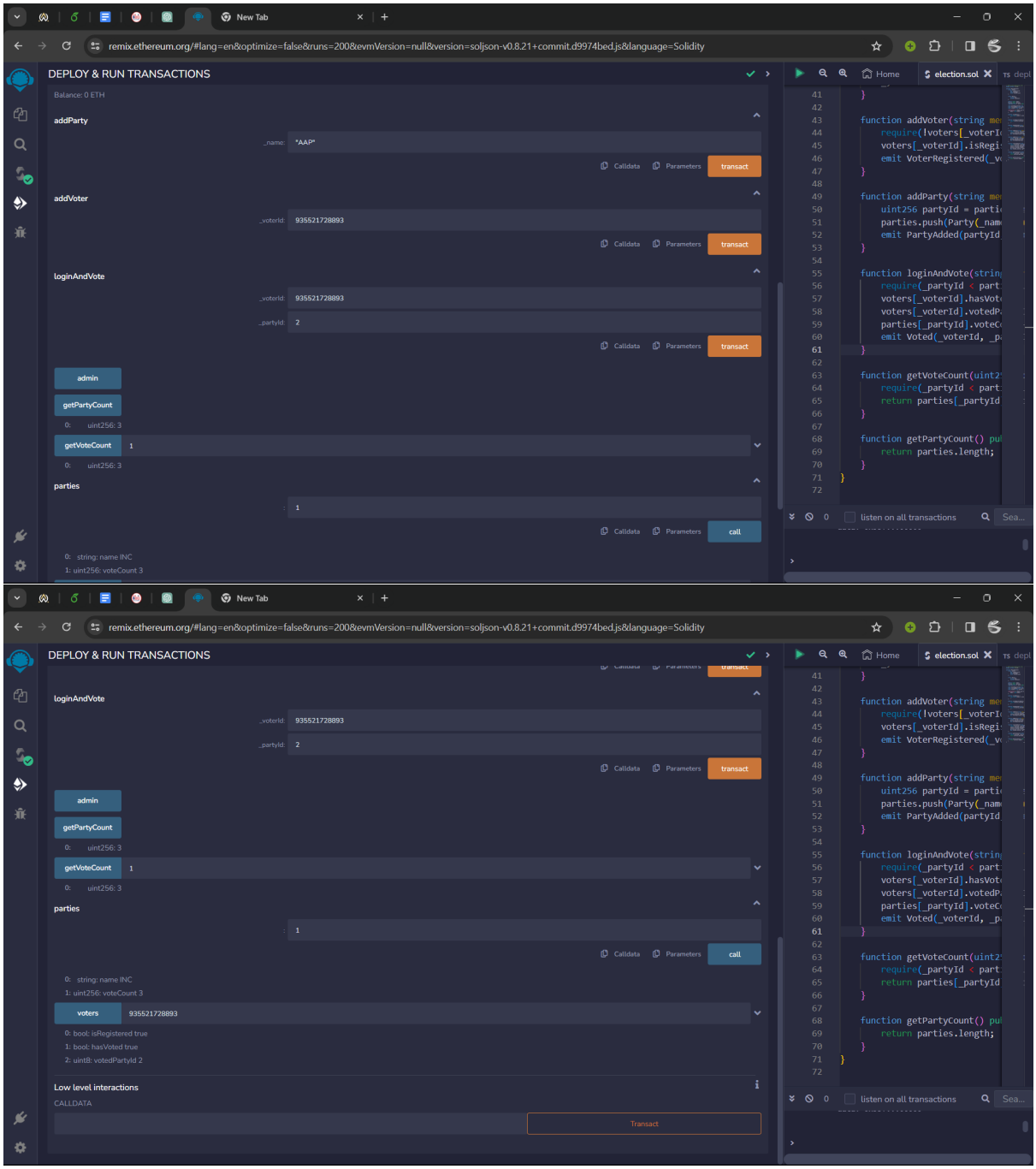
8 Output


New Tab
remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.21+commit.d9974bed.js&language=Solidity
Home
election.sol
deploy_with_ethers.ts
deploy_with_web3.ts
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract EVotingSystem {
struct Voter {
bool isRegistered;
bool hasVoted;
uint8 votedPartyId;
}
struct Party {
string name;
uint256 voteCount;
}
address public admin;
Party[] public parties;
mapping(string => Voter) public voters;
event VoterRegistered(string voterId);
event PartyAdded(uint256 partyId, string partyName);
event Voted(string voterId, uint256 partyId);
constructor() { 991067 gas 965800 gas
admin = msg.sender;
}
modifier onlyAdmin() {
require(msg.sender == admin, "Only admin can perform this operation");
_;
}
modifier onlyRegisteredVoter(string memory _voterId) {
listen on all transactions
Search with transaction hash or address
[call] from: 0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db to: EVotingSystem.getVoteCount(uint256) data: 0xb2c...00001
Debug

New Tab
remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.21+commit.d9974bed.js&language=Solidity
Home
election.sol
deploy_with_ethers.ts
deploy_with_web3.ts
modifier onlyRegisteredVoter(string memory _voterId) {
require(voters[_voterId].isRegistered, "Voter is not registered");
_;
}
modifier hasNotVoted(string memory _voterId) {
require(!voters[_voterId].hasVoted, "Voter has already voted");
_;
}
function addVoter(string memory _voterId) public onlyAdmin { infinite gas
require(!voters[_voterId].isRegistered, "Voter already registered");
voters[_voterId].isRegistered = true;
emit VoterRegistered(_voterId);
}
function addParty(string memory _name) public onlyAdmin { infinite gas
uint256 partyId = parties.length;
parties.push(Party(_name, 0));
emit PartyAdded(partyId, _name);
}
function loginAndVote(string memory _voterId, uint256 _partyId) public onlyRegisteredVoter(_voterId) hasNotVoted(_voterId) { infinite gas
require(_partyId < parties.length, "Invalid party ID");
voters[_voterId].hasVoted = true;
voters[_voterId].votedPartyId = uint8(_partyId);
parties[_partyId].voteCount++;
emit Voted(_voterId, _partyId);
}
function getVoteCount(uint256 _partyId) public view returns (uint256) { infinite gas
require(partyId < parties.length, "Invalid party ID");
listen on all transactions
Search with transaction hash or address
[call] from: 0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db to: EVotingSystem.getVoteCount(uint256) data: 0xb2c...00001
Debug


New Tab
remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.21+commit.d9974bed.js&language=Solidity
Home
election.sol
deploy_with_ethers.ts
deploy_with_web3.ts
function addVoter(string memory _voterId) public onlyAdmin {
require(!voters[_voterId].isRegistered, "Voter already registered");
voters[_voterId].isRegistered = true;
emit VoterRegistered(_voterId);
}
function addParty(string memory _name) public onlyAdmin {
uint256 partyId = parties.length;
parties.push(Party(_name, 0));
emit PartyAdded(partyId, _name);
}
function loginAndVote(string memory _voterId, uint256 _partyId) public onlyRegisteredVoter(_voterId) hasNotVoted(_voterId) {
require(_partyId < parties.length, "Invalid party ID");
voters[_voterId].hasVoted = true;
voters[_voterId].votedPartyId = uint8(_partyId);
parties[_partyId].voteCount++;
emit Voted(_voterId, _partyId);
}
function getVoteCount(uint256 _partyId) public view returns (uint256) {
require(_partyId < parties.length, "Invalid party ID");
return parties[_partyId].voteCount;
}
function getPartyCount() public view returns (uint256) {
return parties.length;
}
}
listen on all transactions
Search with transaction hash or address
[call] from: 0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db to: EVotingSystem.getVoteCount(uint256) data: 0xb2c...00001
Debug

New Tab
remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.21+commit.d9974bed.js&language=Solidity
DEPLOY & RUN TRANSACTIONS
ENVIRONMENT
Remix VM (Shanghai)
VM
ACCOUNT
0x4B2...C02db (99.999999999998295229 ether)
GAS LIMIT
3000000
VALUE
0
Wei
CONTRACT
EVotingSystem - contracts/election.sol
evm version: shanghai
Deploy
Publish to IPFS
At Address
Load contract from Address
Transactions recorded 16
Deployed Contracts
EVOTINGSYSTEM AT 0X9EC...DDEB4 (MEMORY)
Balance: 0 ETH
addParty
string _name
Home
election.sol
listen on all transactions
[call]
from:



9 Conclusion

Our voting system uses the concept of preference based voting and makes use of Ethereum platform The votes are counted, and if we have a clear majority as defined by the organization, then our system will declare the winner But if we don't have clear winner, the votes of the last candidate are distributed according to the preferences given by the voters for that candidate Thus, the last candidate is removed from the competition and its votes distributed to other candidates according to the preferences given in the votes of the eliminating candidates This process continues as long as we don't get a clear majority winner.