19BCE248

Blockchain Technology

Practical 6

AIM: To build, implement and test voting mechanism using Ethereum Blockchain.

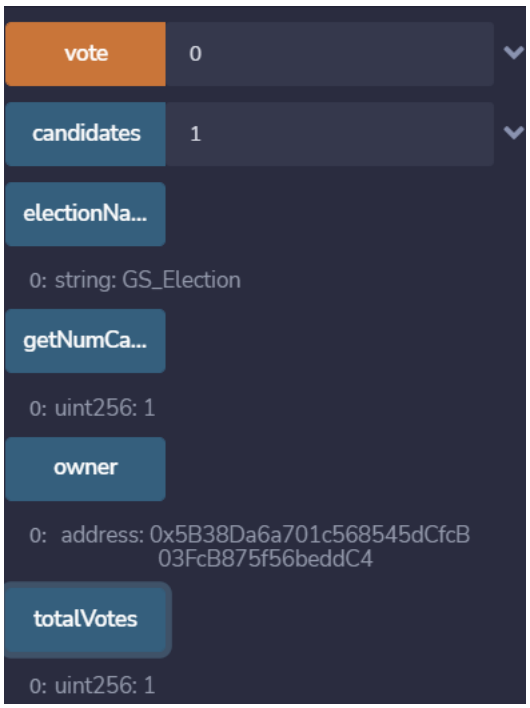
Code:

```
pragma solidity ^0.4.0;
//SPDX-License-Identifier: GPL-3.0-or-later
// Smart Contract for voting system
contract VotingSystem{
    /*Structure is similar to class where it has various attributes
associated.
    Two Such structures are created namely Candidate and Voter.
    1) Candidate
      --> name : Name of the canditate opted for election
      --> voteCount : No. of votes a candidate gets.

    2) Voter
      --> authorized : Checks whether he/she is well authorized to vote.
Indian Voting system has
      election card for the same.
      --> voted : Boolean variable to keep track of whether voted or not.
      --> vote : Number of votes given.
    */

    struct Candidate {
    string name;
    uint voteCount;
    }

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

    constructor() public {
    owner=msg.sender;
    }

    // Owner of the complete process
    address public  owner;
```

```
    //A unique name given to each election
    string public electionName;

    //Similar to HashMap in languages like python,java where address is
assigned to each voter
    mapping(address => Voter) public voters;

    //Declarations of varibale and array to be used later
    Candidate[] public candidates;
    uint public totalVotes;

    //Checks for the validity whether it is by the owner only or not
    modifier ownerOnly() {
    require(msg.sender == owner);_;
    }

    // Assigns before Election paramaters
    function Election(string  _name ) public {
    owner = msg.sender;
    electionName = _name;
    }

    //Addition of new Candidate as there are more than one candidate in
election system and we need
    //to keep track of each of them
    function addCandidate(string   _name ) ownerOnly public {
    candidates.push(Candidate(_name,0));
    }

    //No. of candidated participating
    function getNumCandidate()
    public view returns(uint) {
    return candidates.length;
    }

    //Checks whether voter is authenticated ot not
    function authorize(address _person) ownerOnly public {
    voters[_person].authorized = true;
    }


    function vote(uint _voteIndex) public {
    //shouldn't vote beforehand
    require(!voters[msg.sender].voted);

    //should be authorized
    require(voters[msg.sender].authorized);
```

```
    //assigning unique Id of candidate to a vote attribute
    voters[msg.sender].vote = _voteIndex;

    //makes voted to true
    voters[msg.sender].voted = true;

    //increase the count of votes in that particular candidate
    candidates[_voteIndex].voteCount += 1;

    totalVotes += 1;
    }

    //Termination of contract
    function end() ownerOnly public {
    selfdestruct(owner);
    }
}
```

Output:



Here Similarly we can create multiple candidates for the same election with multiple voters.

Each voter will be assigned with unique address which would be assigned to each vote of candidate from where it came. These makes our smart contract work at a very effective manner with major conditions being checked.