**Source Code of** **Voting with Blockchain**

**A Usecase of Decentralized Voting System**

pragma solidity ^0.8.0;

contract candidatePortal {

    address public admin;

    bool public electionEnded;

    bool public electionStarted;

    struct Voter {

        uint id;

        string name;

        bool hasVoted;

    }

    struct Candidate {

        uint id;

        string title;

        uint voteCount;

        bool isRegistered;

    }

    mapping(address => Voter) public voters;

    mapping(uint => Candidate) public candidates;

    uint public voterCount;

    uint public candidateCount;

    uint public Id\_winningCandidate;

    string public Name\_winningCandidate;

    uint public VoteCount\_winning;

    event WinnerAnnounced(string winnerName, uint voteCount);

    event ElectionResult(uint candidateId, string candidateName, uint voteCount);

    event ElectionStarted();

    event ElectionEnded();

    constructor() {

        admin = msg.sender;

    }

    modifier onlyAdmin() {

        require(msg.sender == admin, "Only admin can perform this action");

        \_;

    }

    modifier onlyDuringElection() {

        require(electionStarted && !electionEnded, "Election is not active");

        \_;

    }

    function startElection() public onlyAdmin {

        require(!electionStarted, "Election already started");

        electionStarted = true;

        emit ElectionStarted();

    }

    function endElection() public onlyAdmin {

        require(electionStarted && !electionEnded, "Election has already ended");

        electionEnded = true;

        uint maxVotes = 0;

        for (uint i = 1; i <= candidateCount; i++) {

            if (candidates[i].voteCount > maxVotes) {

                maxVotes = candidates[i].voteCount;

                Id\_winningCandidate = candidates[i].id;

                Name\_winningCandidate = candidates[i].title;

                VoteCount\_winning = maxVotes;

            }

        }

        emit WinnerAnnounced(Name\_winningCandidate, VoteCount\_winning);

        emit ElectionEnded();

    }

    function addVoter(address \_voterAddress, string memory \_name) public onlyAdmin onlyDuringElection {

        require(voters[\_voterAddress].id == 0, "Voter already registered");

        voterCount++;

        voters[\_voterAddress] = Voter(voterCount, \_name, false);

    }

    function addCandidate(string memory \_title) public onlyAdmin onlyDuringElection {

        candidateCount++;

        candidates[candidateCount] = Candidate(candidateCount, \_title, 0, true);

    }

    function voteForCandidate(uint \_candidateId) public onlyDuringElection {

        require(\_candidateId > 0 && \_candidateId <= candidateCount, "Invalid candidate ID");

        require(voters[msg.sender].id != 0, "Only registered voters can vote");

        require(!voters[msg.sender].hasVoted, "You have already voted");

        Candidate storage candidate = candidates[\_candidateId];

        require(candidate.isRegistered, "Candidate not registered");

        candidate.voteCount++;

        voters[msg.sender].hasVoted = true;

        emit ElectionResult(candidate.id, candidate.title, candidate.voteCount);

    }

    function getElectionResults() public view returns (uint[] memory, string[] memory, uint[] memory) {

        uint[] memory candidateIds = new uint[](candidateCount);

        string[] memory candidateNames = new string[](candidateCount);

        uint[] memory voteCounts = new uint[](candidateCount);

        for (uint i = 1; i <= candidateCount; i++) {

            candidateIds[i - 1] = candidates[i].id;

            candidateNames[i - 1] = candidates[i].title;

            voteCounts[i - 1] = candidates[i].voteCount;

        }

        return (candidateIds, candidateNames, voteCounts);

    }

}