**Voting With Blockchain Source Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract Election {

    // Variables

    struct candidate {

        uint candidateid;

        string name;

        string proposal;

        address candidateaddress;

        uint voteCount;

    }

    struct voter {

        uint voterid;

        string votername;

        address voteraddress;

        bool canVote;

        bool hasVoted;

    }

    struct voterprofile {

        string voterName;

        string votedCandidateName;

        bool VoteDelegated;

    }

    mapping (address => candidate) CandidateDetails;

    mapping (address => voter) VoterDetails;

    mapping (address => voterprofile) voterprofiledetails;

    address VotingAdmin;

    bool public electionStarted;

    bool public electionEnded;

    address[] public candidateList;

    event DelegateVote (address \_voteraddress,address \_delegatedVoter);

    event ElectionStarted();

    event ElectionEnded();

    /\*Constructor to Initialize VotingAdmin to contract owner address

    and initialize boolean variables\*/

    constructor () {

        VotingAdmin = msg.sender;

        electionStarted = false;

        electionEnded = false;

    }

    /\*modifier to restrict access to only Voting Admin which checks

    the caller address is same as contract owner address\*/

    modifier OnlyVotingAdmin{

        require(msg.sender == VotingAdmin,"Only Voting Admin can perform this action");

        \_;

    }

    /\*modifier which restricts access to only eligible voters which checks wheter

    canVote field is true\*/

    modifier onlyVoter() {

        require(VoterDetails[msg.sender].canVote, "Only eligible voters can perform this action");

        \_;

    }

    //Functions

    //Function to add candidate details only by Voting Admin

    function addCandidate (uint \_candidateid,string memory \_name,string memory \_proposal,address \_candidateaddress) public OnlyVotingAdmin {

        require(!electionStarted,"Election has already started candidate cannot be added");

        require(bytes(\_name).length > 0, "Candidate name is required");

        require(bytes(\_proposal).length > 0, "Proposal is required");

        candidate memory CD = candidate (\_candidateid,\_name,\_proposal,\_candidateaddress,0);

        CandidateDetails[\_candidateaddress] = CD;

        candidateList.push(\_candidateaddress);

    }

    /\*Function to add voter details only by Voting Admin

      For eligible voter canVote is set to true and

      hasVoted is set to false before election is started\*/

    function addVoter (uint \_voterid,string memory \_name,address \_voteraddress,bool \_canvote,bool \_hasvote) public OnlyVotingAdmin {

        require(\_voterid > 0,"Voter id is required");

        require(bytes(\_name).length > 0, "Voter name is required");

        voter memory VD = voter (\_voterid,\_name,\_voteraddress,\_canvote,\_hasvote);

        VoterDetails[\_voteraddress] = VD;

    }

    /\*Function to start election only by Voting admin boolean

    variable is set to true and an event is emitted

    for start of election which can be seen in transaction logs\*/

    function startElection () public OnlyVotingAdmin {

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

        electionStarted = true;

        emit ElectionStarted();

    }

    //Function to display candidate details when provided with Candidate address

    function displayCandidateDetails (address \_candidateaddress) public view  returns (candidate memory) {

        return CandidateDetails[\_candidateaddress];

    }

    /\*Function to display winner of election by using for loop to iterate through

      CandidateDetails struct to find the maximum value in voteCount field

      and store it in maxVotes local variable and the corresponding address

      as winningCandidate and return winning candidate name ,id and votes secured  \*/

    function WinnerOfElection () public view returns (string memory,uint,uint) {

        require(electionEnded,"Election has not ended");

        uint maxVotes = 0;

        address winningCandidate;

        for (uint i=0; i < candidateList.length;i++){

            if (CandidateDetails[candidateList[i]].voteCount > maxVotes){

                maxVotes = CandidateDetails[candidateList[i]].voteCount;

                winningCandidate = candidateList[i];

            }

        }

        return (CandidateDetails[winningCandidate].name,CandidateDetails[winningCandidate].candidateid,maxVotes);

    }

    /\*Function to delegate vote only by eligible voters

      Once the voter eligbility conditions are satisfied hasVoted is set to true for original voter

      and delegated voter address's canVote is set to false and an event is emitted with the voter

      and delegated voter address and VoteDelegated is set to true\*/

    function delegateVote (address \_voteraddress,address \_delegatedVoteraddress) public onlyVoter {

        require(electionStarted,"Election is no longer going on");

        require(!VoterDetails[\_voteraddress].hasVoted,"The voter has already voted");

        require(VoterDetails[\_delegatedVoteraddress].canVote,"The delegated person is not eligible to vote");

        VoterDetails[\_voteraddress].hasVoted = true;

        VoterDetails[\_delegatedVoteraddress].canVote = false;

        voterprofiledetails[\_voteraddress].VoteDelegated = true;

        emit DelegateVote(\_voteraddress, \_delegatedVoteraddress);

    }

    /\*Function to cast vote only by eligible voters

      Store Voter name and the candidate name who was voted in voterprofiledetails mapping \*/

    function castVote (address \_candidateaddress,address \_voteraddress) public onlyVoter {

        require(electionStarted,"Election has not started");

        require(!electionEnded,"Election has ended");

        require(VoterDetails[\_voteraddress].canVote, "You are not eligible to vote");

        require(!VoterDetails[\_voteraddress].hasVoted, "You have already voted");

        CandidateDetails[\_candidateaddress].voteCount++;

        voterprofiledetails[\_voteraddress].votedCandidateName = CandidateDetails[\_candidateaddress].name;

        voterprofiledetails[\_voteraddress].voterName = VoterDetails[\_voteraddress].votername;

        VoterDetails[\_voteraddress].hasVoted = true;

    }

    /\*Function to end election boolean variable is set to true and an event is

      emitted for end of election\*/

    function endElection () public OnlyVotingAdmin {

        require(electionStarted, "Election has not started");

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

        electionEnded = true;

        emit ElectionEnded();

    }

    /\*Function which takes candidate address as input and

      displays Candidate id, name and votes secured by the candidate\*/

    function electionResults (address \_candidateaddress) public view returns (uint,string memory,uint){

        return (CandidateDetails[\_candidateaddress].candidateid,CandidateDetails[\_candidateaddress].name,CandidateDetails[\_candidateaddress].voteCount);

    }

    /\*Function takes voteraddress has input and displays Voter name, Candidate name to

    whom the voter has voted and whether the vote is delegated or not by using boolean variable \*/

    function voterProfile (address \_voteraddress) public view returns (voterprofile memory) {

        return voterprofiledetails[\_voteraddress];

    }

}