// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract Election {

    address public admin;

    bool public isElectionOngoing;

    struct Candidate {

        uint256 id;

        string name;

        string proposal;

        uint256 voteCount;

    }

    struct Voter {

        string name;

        uint256 votedForCandidateId;

        bool hasVoted;

        bool hasDelegated;

        address delegate;

    }

    mapping(uint256 => Candidate) public candidates;

    mapping(address => Voter) public voters;

    uint256 public candidateCount;

    uint256 public voterCount;

    modifier onlyAdmin() {

        require(msg.sender == admin, "Only the admin can call this function");

        \_;

    }

    modifier onlyBeforeElection() {

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

        \_;

    }

    modifier onlyDuringElection() {

        require(isElectionOngoing, "Election has not started yet");

        \_;

    }

    constructor() {

        admin = msg.sender;

        isElectionOngoing = false;

        candidateCount = 0;

        voterCount = 0;

    }

    function addCandidate(string memory \_name, string memory \_proposal) public onlyAdmin onlyBeforeElection {

        candidateCount++;

        candidates[candidateCount] = Candidate(candidateCount, \_name, \_proposal, 0);

    }

    function addVoter(address \_voter) public onlyAdmin onlyBeforeElection {

        require(voters[\_voter].votedForCandidateId == 0, "Voter already added");

        voterCount++;

        voters[\_voter] = Voter("", 0, false, false, address(0));

    }

    function startElection() public onlyAdmin onlyBeforeElection {

        isElectionOngoing = true;

    }

    function displayCandidateDetails(uint256 \_id) public view returns (uint256, string memory, string memory) {

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

        return (candidates[\_id].id, candidates[\_id].name, candidates[\_id].proposal);

    }

    function showWinner() public view returns (string memory, uint256, uint256) {

        uint256 winnerId = 0;

        uint256 maxVotes = 0;

        for (uint256 i = 1; i <= candidateCount; i++) {

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

                winnerId = i;

                maxVotes = candidates[i].voteCount;

            }

        }

        return (candidates[winnerId].name, candidates[winnerId].id, maxVotes);

    }

    function delegateVotingRight(address \_delegateTo) public onlyDuringElection {

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

        require(\_delegateTo != msg.sender, "Cannot delegate to yourself");

        voters[msg.sender].hasDelegated = true;

        voters[msg.sender].delegate = \_delegateTo;

    }

    function castVote(uint256 \_candidateId) public onlyDuringElection {

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

        require(voters[msg.sender].hasDelegated == false, "Vote has been delegated");

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

        voters[msg.sender].votedForCandidateId = \_candidateId;

        voters[msg.sender].hasVoted = true;

        candidates[\_candidateId].voteCount++;

    }

    function endElection() public onlyAdmin onlyDuringElection {

        isElectionOngoing = false;

    }

    function showCandidateVotes(uint256 \_candidateId) public view returns (uint256, string memory, uint256) {

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

        return (candidates[\_candidateId].id, candidates[\_candidateId].name, candidates[\_candidateId].voteCount);

    }

    function viewVoterProfile(address \_voter) public view returns (string memory, uint256, bool) {

        require(voters[\_voter].votedForCandidateId > 0, "Voter does not exist");

        return (voters[\_voter].name, voters[\_voter].votedForCandidateId, voters[\_voter].hasDelegated);

    }

}