|  |
| --- |
|  |
| // SPDX-License-Identifier: MIT |
|  |  |
|  | pragma solidity ^0.8.7; |
|  |  |
|  | contract Election2022{ |
|  |  |
|  | struct Candidate { |
|  | bool approved; |
|  | uint id; |
|  | string name; |
|  | address \_address; |
|  | uint voteCount; |
|  |  |
|  | } |
|  |  |
|  | address ecadmin; |
|  | address ec; |
|  | uint256 public startTime; |
|  | uint256 public stopTime; |
|  |  |
|  | constructor( address \_admin) { |
|  | ecadmin = \_admin; |
|  | } |
|  |  |
|  | modifier ecOnly() { |
|  | require(msg.sender == address(ec),"EC only operation"); |
|  | \_; |
|  | } |
|  |  |
|  | modifier ecAdminOnly() { |
|  | require(msg.sender == ecadmin,"EC admin only operation"); |
|  | \_; |
|  | } |
|  |  |
|  | function setEC(address \_ec) public ecAdminOnly { |
|  | ec = \_ec; |
|  | } |
|  |  |
|  | mapping (uint=> Candidate) public candidates; |
|  |  |
|  | mapping (address=>bool) public voters; |
|  |  |
|  | uint public candidates\_no; |
|  |  |
|  |  |
|  |  |
|  | // Event |
|  | event Candidate\_Data(address candidate, Candidate data); |
|  |  |
|  | function Apply(string memory name) public payable returns(string memory){ |
|  |  |
|  | require(msg.value == 0.1 ether,"Appropraite ether not next"); |
|  | candidates\_no++; |
|  | candidates[candidates\_no] = Candidate(false,candidates\_no,name,msg.sender,0); |
|  | emit Candidate\_Data(msg.sender, candidates[candidates\_no]); |
|  | return "Done"; |
|  | } |
|  |  |
|  |  |
|  |  |
|  | function approveCandidiate(uint256 num) external { |
|  | require(candidates[num].\_address != address(0x00), "Not registered" ); |
|  | require(candidates[num].approved == false, "Already approved" ); |
|  | candidates[num].approved = true; |
|  | } |
|  |  |
|  | function setStart(uint256 num) public ecOnly { |
|  | require(num > block.timestamp,"Start at later time" ); |
|  | startTime = num; |
|  | } |
|  |  |
|  | function setStop(uint256 num) public ecOnly { |
|  | require(num > block.timestamp && num > startTime,"Stop at later time" ); |
|  | stopTime = num; |
|  | } |
|  |  |
|  | event consolePrint( string, address); |
|  |  |
|  | function vote(uint \_candidateId) public{ |
|  | require(block.timestamp > startTime,"Election not started" ); |
|  | require(block.timestamp <= stopTime,"Election over" ); |
|  | // require(!voters[msg.sender] == false, "Already voted"); |
|  | require(!voters[msg.sender], "Already voted"); |
|  | require(candidates[\_candidateId].\_address != address(0x00), "Not registered condidate" ); |
|  | require(candidates[\_candidateId].approved == true, "Dont vote not approved" ); |
|  |  |
|  | voters[msg.sender] == true; |
|  | candidates[\_candidateId].voteCount++; |
|  |  |
|  | } |
|  |  |
|  | function getResults() public view ecOnly returns (Candidate memory candidate) { |
|  | require(block.timestamp >= stopTime,"Election yet to finish" ); |
|  | uint256 c; |
|  | uint256 max=0; |
|  | for( uint i =1; i<=candidates\_no; i++) { |
|  | if ( candidates[i].voteCount > max ) { |
|  | max = candidates[i].voteCount; |
|  | c = i; |
|  | } |
|  | } |
|  |  |
|  | return candidates[c]; |
|  | } |
|  |  |
|  |  |
|  |  |
|  | // Self destruct |
|  | function votingclose() public ecOnly { |
|  | selfdestruct(payable(msg.sender)); |
|  | } |
|  |  |
|  |  |
|  | } |
|  |  |
|  |  |
|  |  |
|  | contract ElectionCommision{ |
|  |  |
|  | Election2022 public Election; |
|  |  |
|  | // constructor() { |
|  |  |
|  | // Election = new Election2022(msg.sender); |
|  | // } |
|  |  |
|  | event done(string msge); |
|  | event notdone(string msge); |
|  |  |
|  | function approveCandidate(address candidate, uint256 id) public { |
|  | try new Election2022(candidate) returns (Election2022 Election) { |
|  | Election2022 e = Election2022(candidate); |
|  | e.approveCandidiate(id); |
|  | emit done("Approve"); |
|  | } |
|  | catch { |
|  | emit notdone("Not Approve"); |
|  | } |
|  | } |
|  | } |
|  |  |