1. Election 2022

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.7;

contract Election2022

{

    struct Candidate

    {

        bool approved;

        address id;

        uint voteCount;

        string candidateName;

    }

    mapping (address=>bool) public voters;

    mapping (uint=>Candidate) public candidates;

    address ecadmin;

    address ec;

    uint public candidatesCount;

    uint public startTime;

    uint 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;

    }

    function addCandidate(string memory \_name) public payable

    {

        require(msg.value==1 ether,"Appropraite ether not sent.");

        candidatesCount++;

        candidates[candidatesCount]=Candidate(false,msg.sender,0,\_name);

        (bool success,)=payable(ec).call{value: msg.value}("");

        require(success, "Failed to send Ether");

    }

     function approveCandidiate(uint256 num) external ecOnly {

        require(candidates[num].id != address(0x00), "Not registered" );

        require(candidates[num].approved == false, "Already approved" );

        candidates[num].approved = true;

    }

    function setStart(uint256 num) external ecOnly {

        require(num > block.timestamp,"Start at later time" );

        startTime = num;

    }

    function setStop(uint256 num) external 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(candidates[\_candidateId].id != 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 returns (Candidate memory candidate) {

        require(block.timestamp >= stopTime,"Election yet to finish" );

        uint256 c;

        uint256 max=0;

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

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

                max = candidates[i].voteCount;

                c = i;

            }

        }

        return candidates[c];

    }

    function kill() external ecOnly

    {

        selfdestruct(payable(ec));

    }

    function test\_require() public view

    {

       require(ec==ecadmin,"they are not same");

    }

    function test\_revert() public view

    {

        if(ec!=ecadmin)

        {

            revert("they are not same");

        }

    }

    function test\_assert() public view

    {

        assert(ec==ecadmin);

    }

    event balance(address account,string message,uint val);

    function getBalance() public payable

    {

        emit balance(msg.sender,"has sent value",msg.value);

    }

}

1. ElectionCommison

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.7;

interface Election {

    function approveCandidiate(uint256 num) external;

    function setStart(uint256 start) external;

    function setStop(uint256 stop) external;

    function getResults() external;

    function kill() external;

}

contract ElectionCommision {

    address public admin;

    constructor() {

        admin = msg.sender;

    }

    modifier adminOnly() {

        require(msg.sender == admin,"No admin");

        \_;

    }

    function approveCandidate(address election, uint256 id) public adminOnly {

        Election e = Election(election);

        e.approveCandidiate(id);

    }

    function setStart(address election, uint256 time) public adminOnly {

        Election e = Election(election);

        e.setStart(time);

    }

    function setStop(address election, uint256 time) public adminOnly {

        Election e = Election(election);

        e.setStop(time);

    }

    function withdraw() public {

        uint amount = address(this).balance;

        (bool success, ) = admin.call{value: amount}("");

        require(success, "Failed to send Ether");

    }

    function kill(address election) public adminOnly {

        Election e = Election(election);

        e.kill();

    }

}

1. NewElection2022

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.7;

contract Election2022\_new

{

    uint public month;

    string public state;

    address public id;

    constructor(uint \_month,string memory \_state,address) payable

    {

        month=\_month;

        state=\_state;

        id=msg.sender;

    }

}

contract ElectionFactory

{

    Election2022\_new[] public election2022;

    function add(uint \_month,string memory \_state,address) public payable

    {

        Election2022\_new election=new Election2022\_new(\_month,\_state,msg.sender);

        election2022.push(election);

    }

}