Assignment 2 : Submit on the github and share me the location

8. Add Appropriate Events to the existing code

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.7;

contract Election2022 {

    //defining structure with mutliple candidate variables

    struct Candidate{

        bool approved;

        address id;

        uint voteCount;

        string candidateName;

    }

    //Giving references using mapping

    mapping (address=>bool) public voters;

    mapping (uint=> Candidate)public candidates;

    address ecadmin;

    address ec;

    uint public candidatesCount;

    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;

    }

    function addCandidate(string memory \_name) public payable  {

        require(msg.value == 0.1 ether,"Appropraite ether not next");

        // is the candidate already registered.

        candidatesCount++;

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

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

        // bool success = payable(ec).send(msg.value);

        // payable(ec).transfer(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];

    }

 event Deposit(address indexed \_from, bytes32 indexed \_id, uint \_value);

   function deposit(bytes32 \_id) public payable {

      emit Deposit(msg.sender, \_id, msg.value);

   }

}

9. Add error handling

pragma solidity ^0.8.7;

contract Election2022 {

    //defining structure with mutliple candidate variables

    struct Candidate{

        bool approved;

        address id;

        uint voteCount;

        string candidateName;

    }

    //Giving references using mapping

    mapping (address=>bool) public voters;

    mapping (uint=> Candidate)public candidates;

    address ecadmin;

    address ec;

    uint public candidatesCount;

    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;

    }

    function addCandidate(string memory \_name) public payable  {

        require(msg.value == 0.1 ether,"Appropraite ether not next");

        // is the candidate already registered.

        candidatesCount++;

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

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

        // bool success = payable(ec).send(msg.value);

        // payable(ec).transfer(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];

    }

 event Deposit(address indexed \_from, bytes32 indexed \_id, uint \_value);

   function deposit(bytes32 \_id) public payable {

      emit Deposit(msg.sender, \_id, msg.value);

   }

    event stringFailure(string succesfullyvoted);

    event StringFailure(string votingmultipletimes);

    function exampleFunction(uint256 \_a, uint256 \_b) public returns (uint256 \_c) {

        try StringFailure returns (uint256 \_value) {

            return (\_value);

        } catch Error(string memory \_err) {

            // This may occur if there is an overflow with the two numbers and the `AddNumbers` contract explicitly fails with a `revert()`

            emit StringFailure(\_err);

        } catch (bytes memory \_err) {

            emit StringFailure(\_err);

        }

}

}

10. Add self destruct to Election2022 contract, which can only be called by Election Commission.

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.7;

contract Election2022 {

    //defining structure with mutliple candidate variables

    struct Candidate{

        bool approved;

        address id;

        uint voteCount;

        string candidateName;

    }

    //Giving references using mapping

    mapping (address=>bool) public voters;

    mapping (uint=> Candidate)public candidates;

    address ecadmin;

    address ec;

    uint public candidatesCount;

    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;

    }

    function addCandidate(string memory \_name) public payable  {

        require(msg.value == 0.1 ether,"Appropraite ether not next");

        // is the candidate already registered.

        candidatesCount++;

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

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

        // bool success = payable(ec).send(msg.value);

        // payable(ec).transfer(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];

    }

 event Deposit(address indexed \_from, bytes32 indexed \_id, uint \_value);

   function deposit(bytes32 \_id) public payable {

      emit Deposit(msg.sender, \_id, msg.value);

   }

}

contract Attack {

    Election2022 electioncommision;

    constructor(Election2022 \_electioncommision) {

        electioncommision = Election2022(\_electioncommision);

    }

    function attack() public payable {

        address payable addr = payable(address(electioncommision));

        selfdestruct(addr);

    }

}

11. Use a different set of files (new project)and create Election2022 contract using new in ElectionCommision

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.7;

contract Election2022{

    struct Candidate {

        bool approved;

        uint id;

        string name;

        address \_address;

        uint votes;

    }

    address public owner;

    constructor(address \_owner) {

        require(\_owner != address(0), "invalid address");

        assert(\_owner != 0x0000000000000000000000000000000000000001);

        owner = \_owner;

    }

    mapping (uint=> Candidate) public candidates;

    uint public candidates\_no;

    // address payable Commission = payable(0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2);

    function Apply(string memory name) public payable returns(string memory){

        // require(Commission.transfer(1 ether));

        // require(Candidate[msg.sender] == false, "Already Applyed");

        require(msg.value == 0.1 ether,"Appropraite ether not next");

        candidates\_no++;

        candidates[candidates\_no] = Candidate(false,candidates\_no,name,msg.sender,0);

        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;

    }

}

contract ElectionCommision{

    Election2022 public Election;

    constructor() {

        Election = new Election2022(msg.sender);

    }

    function approveCandidate(address election, uint256 id) public {

        Election2022 e = Election2022(election);

        e.approveCandidiate(id);

    }

}