# Blockchain Technology

# Mini Project

# Code

pragma solidity ^0.4.25; contract ElectionFact { struct ElectionDet {

address deployedAddress;

string el\_n; string el\_d;

}

mapping(string=>ElectionDet) companyEmail;

function createElection(string memory email,string memory election\_name, string memory election\_description) public{

address newElection = new Election(msg.sender , election\_name, election\_description);

companyEmail[email].deployedAddress = newElection; companyEmail[email].el\_n = election\_name; companyEmail[email].el\_d = election\_description;

}

function getDeployedElection(string memory email) public view returns (address,string,string) {

address val = companyEmail[email].deployedAddress; if(val == 0)

return (0,"","Create an election."); else

return (companyEmail[email].deployedAddress,companyEmail[email].el\_n,companyEmail[em ail].el\_d);

}

}

contract Election {

//election\_authority's address address election\_authority; string election\_name;

string election\_description; bool status;

//election\_authority's address taken when it deploys the contract constructor(address authority , string name, string description) public {

election\_authority = authority; election\_name = name; election\_description = description; status = true;

}

//Only election\_authority can call this function modifier owner() {

require(msg.sender == election\_authority, "Error: Access Denied.");

\_;

}

//candidate election\_description

struct Candidate {

string candidate\_name; string candidate\_description;

string imgHash; uint8 voteCount; string email;

}

//candidate mapping mapping(uint8=>Candidate) public candidates;

//voter election\_description

struct Voter {

uint8 candidate\_id\_voted; bool voted;

}

//voter mapping mapping(string=>Voter) voters;

//counter of number of candidates uint8 numCandidates;

//counter of number of voters uint8 numVoters;

//function to add candidate to mapping

function addCandidate(string memory candidate\_name, string memory candidate\_description, string memory imgHash,string memory email) public owner {

uint8 candidateID = numCandidates++; //assign id of the candidate candidates[candidateID] =

Candidate(candidate\_name,candidate\_description,imgHash,0,email); //add the values to the mapping

}

//function to vote and check for double voting function vote(uint8 candidateID,string e) public {

//if false the vote will be registered require(!voters[e].voted, "Error:You cannot double vote");

voters[e] = Voter (candidateID,true); //add the values to the mapping numVoters++;

candidates[candidateID].voteCount++; //increment vote counter of candidate

}

//function to get count of candidates

function getNumOfCandidates() public view returns(uint8) { return numCandidates;

}

//function to get count of voters

function getNumOfVoters() public view returns(uint8) { return numVoters;

}

//function to get candidate information

function getCandidate(uint8 candidateID) public view returns (string memory, string memory, string memory, uint8,string memory) {

return (candidates[candidateID].candidate\_name, candidates[candidateID].candidate\_description, candidates[candidateID].imgHash, candidates[candidateID].voteCount, candidates[candidateID].email);

}

//function to return winner candidate information

function winnerCandidate() public view owner returns (uint8) { uint8 largestVotes = candidates[0].voteCount;

uint8 candidateID;

for(uint8 i = 1;i<numCandidates;i++) { if(largestVotes < candidates[i].voteCount) {

largestVotes = candidates[i].voteCount; candidateID = i;

}

}

return (candidateID);

}

function getElectionDetails() public view returns(string, string) { return (election\_name,election\_description);

}

}

# Output:



