**Zorawar khan**

**20PWBCS0737 / Sec ‘A’**

**Blockchain Assignment**

**Documentation**

Here is the Contract for voting system that fulfills the requirements of assignment. Such as

*Requirements:*

*1. Add Candidates: Only the contract owner should be able to add candidates.*

*2. Voting: Each registered voter can cast one vote.*

*3. Results: Anyone can view the current standings of the election.*

*4. Ownership: Ensure that only the contract owner can manage the election (e.g., adding*

*candidates).*

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract Voting {

    // Contract owner

    address public owner;

    // Mapping of registered voters

    mapping(address => bool) public voters;

    // Mapping to track if a voter has voted

    mapping(address => bool) public hasVoted;

    // Mapping of votes for each candidate

    mapping(string => uint256) public votes;

    // Array to store the list of candidates

    string[] public candidates;

    // Struct to store candidate details

    struct CandidateResult {

        string name;

        uint256 voteCount;

    }

    // Modifier to restrict access to only the contract owner

    modifier onlyOwner() {

        require(msg.sender == owner, "Only the contract owner can perform this action");

        \_;

    }

    // Modifier to restrict access to only registered voters

    modifier onlyRegisteredVoter() {

        require(voters[msg.sender], "Only registered voters can vote");

        \_;

    }

    // Modifier to ensure the voter has not already voted

    modifier hasNotVoted() {

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

        \_;

    }

    // Constructor to set the contract owner

    constructor() {

        owner = msg.sender;

    }

    // Function to add a candidate (only callable by the owner)

    function addCandidate(string memory \_candidate) public onlyOwner {

        candidates.push(\_candidate);

    }

    // Function to register a voter (only callable by the owner)

    function registerVoter(address \_voter) public onlyOwner {

        voters[\_voter] = true;

    }

    // Function for a registered voter to cast a vote for a candidate

    function vote(string memory \_candidate) public onlyRegisteredVoter hasNotVoted {

        votes[\_candidate]++;

        hasVoted[msg.sender] = true;

    }

    // Function to get result containing all candidates and their vote counts

    function result() public view returns (CandidateResult[] memory) {

        CandidateResult[] memory resultArray = new CandidateResult[](candidates.length);

        for (uint256 i = 0; i < candidates.length; i++) {

            resultArray[i].name = candidates[i];

            resultArray[i].voteCount = votes[candidates[i]];

        }

        return resultArray;

    }

    // Function to get the details of a specific candidate

    function getCandidate(string memory \_candidate) public view returns (CandidateResult memory) {

        uint256 voteCount = votes[\_candidate];

        return CandidateResult({ name: \_candidate, voteCount: voteCount });

    }

}