**3**

pragma solidity >=0.7.0;

contract BankAccount {

address public owner; // The owner of the bank account

uint256 public balance; // The balance of the account

constructor() {

owner = msg.sender;

}

modifier onlyOwner() {

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

\_;

}

function deposit(uint256 amount) public onlyOwner {

require(amount > 0, "Amount to deposit must be greater than 0");

balance += amount;

}

function withdraw(uint256 amount) public onlyOwner {

require(amount > 0, "Amount to withdraw must be greater than 0");

require(balance >= amount, "Insufficient balance");

balance -= amount;

}

function getBalance() public view returns (uint256) {

return balance;

}

}

**4**

// SPDX-License-Identifier: UNLICENSED

// the above approach

// Build the Contract

pragma solidity >=0.7.0;

contract MarksManagmtSys

{

// Create a structure for

// student details

struct Student

{

int ID;

string fName;

string lName;

int marks;

}

address owner;

int public stdCount = 0;

mapping(int => Student) public stdRecords;

modifier onlyOwner

{

require(owner == msg.sender);

\_;

}

constructor()

{

owner=msg.sender;

}

// Create a function to add

// the new records

function addNewRecords(int \_ID,

string memory \_fName,

string memory \_lName,

int \_marks) public onlyOwner

{

// Increase the count by 1

stdCount = stdCount + 1;

// Fetch the student details

// with the help of stdCount

stdRecords[stdCount] = Student(\_ID, \_fName,

\_lName, \_marks);

}

// Create a function to add bonus marks

function bonusMarks(int \_bonus) public onlyOwner

{

stdRecords[stdCount].marks =

stdRecords[stdCount].marks + \_bonus;

}

}

Mini Proj

// SPDX-License-Identifier: MIT

pragma solidity ^0.7.0;

contract Voting {

// Structure to represent a candidate

struct Candidate {

string name;

uint voteCount;

}

// Structure to represent a voter

struct Voter {

bool isRegistered;

bool hasVoted;

uint votedCandidateId;

}

address public admin;

bool public votingOpen;

uint public totalVotes;

Candidate[] public candidates;

mapping(address => Voter) public voters;

event NewCandidate(uint candidateId, string name);

event Voted(address voter, uint candidateId);

modifier onlyAdmin() {

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

\_;

}

modifier canVote() {

require(votingOpen, "Voting is closed");

require(voters[msg.sender].isRegistered, "You are not registered to vote");

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

\_;

}

constructor() {

admin = msg.sender;

votingOpen = false;

}

function openVoting() public onlyAdmin {

votingOpen = true;

}

function closeVoting() public onlyAdmin {

votingOpen = false;

}

function registerVoter(address voterAddress) public onlyAdmin {

require(!voters[voterAddress].isRegistered, "Voter is already registered");

voters[voterAddress].isRegistered = true;

}

function addCandidate(string memory name) public onlyAdmin {

uint candidateId = candidates.length;

candidates.push(Candidate(name, 0));

emit NewCandidate(candidateId, name);

}

function vote(uint candidateId) public canVote {

require(candidateId < candidates.length, "Invalid candidate index");

voters[msg.sender].hasVoted = true;

voters[msg.sender].votedCandidateId = candidateId;

candidates[candidateId].voteCount++;

totalVotes++;

emit Voted(msg.sender, candidateId);

}

function getNumCandidates() public view returns (uint) {

return candidates.length;

}

function getCandidate(uint index) public view returns (string memory, uint) {

require(index < candidates.length, "Invalid candidate index");

return (candidates[index].name, candidates[index].voteCount);

}

function declareWinner() public view returns (string memory, uint) {

require(!votingOpen, "Voting is still open");

require(candidates.length > 0, "No candidates available");

uint winningVoteCount = 0;

uint winningCandidateId;

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

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

winningVoteCount = candidates[i].voteCount;

winningCandidateId = i;

}

}

return (candidates[winningCandidateId].name, winningVoteCount);

}

}