Blockchain Assignment

Behram Khan 20pwbcs0732

8A

UET Peshawar

CS&IT

## Assignment: Implementing a Voting Smart Contract using Solidity

**1. Objective:**

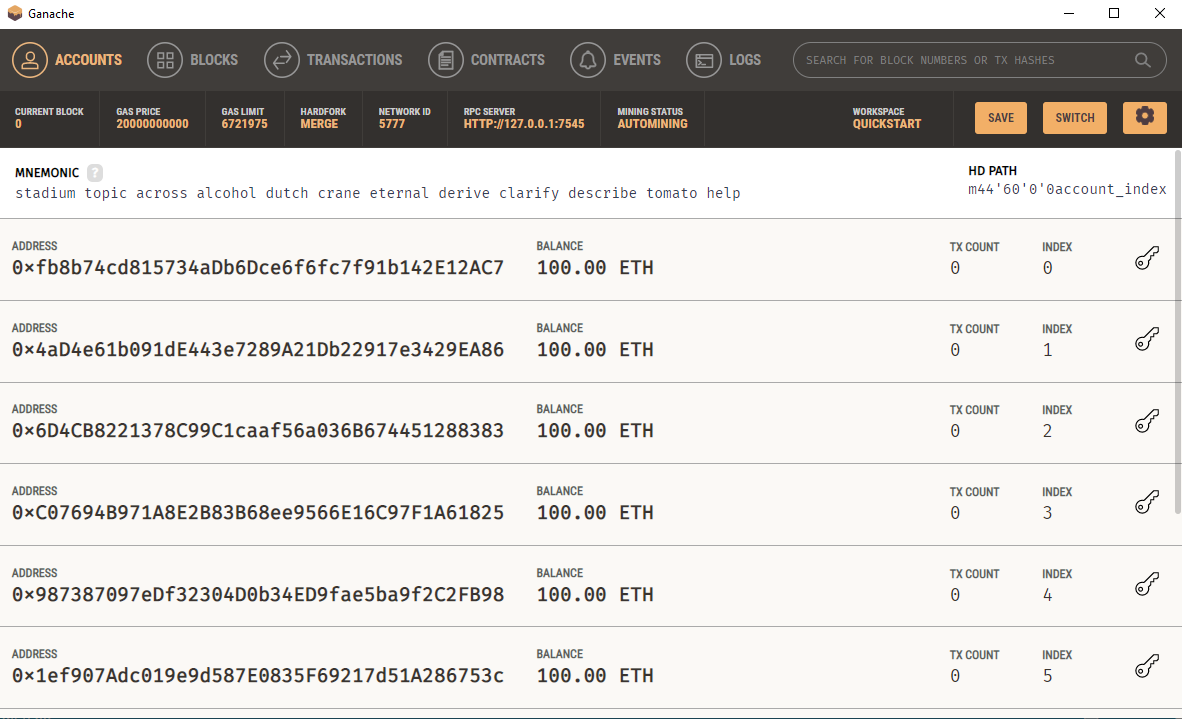
To create a basic voting application using Solidity. This assignment aims to help students understand the foundational concepts of writing smart contracts and deploying them on the Ethereum blockchain.

**Steps to Solve the Problem:**

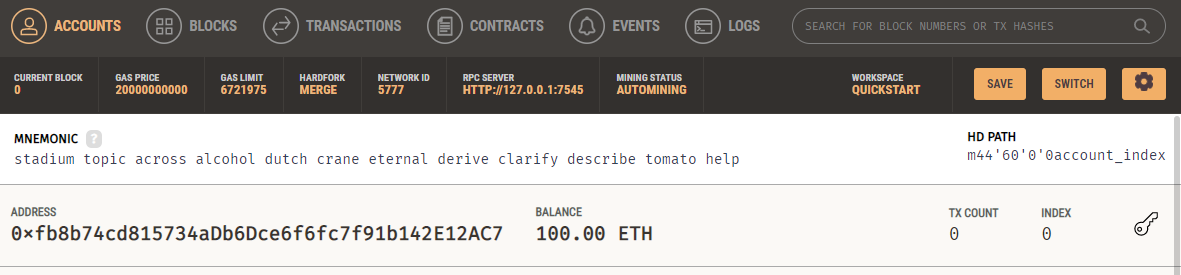
**1. Setup Your Development Environment:**

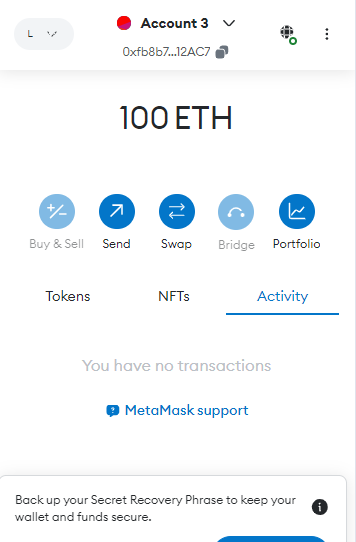
**o Install Ganache: Download and install Ganache from trufflesuite.com/ganache.**

**Ganache is a personal blockchain for Ethereum development.**

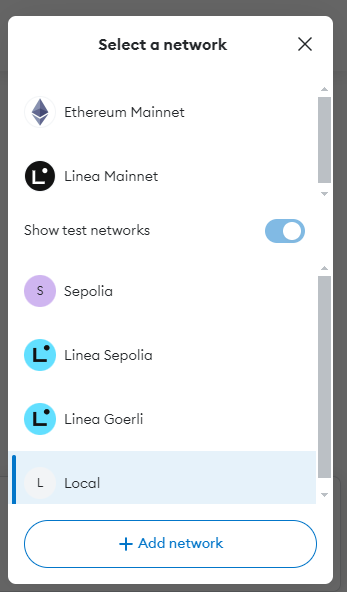
****

**o Install Metamask: Install the Metamask browser extension from metamask.io. Metamask is a cryptocurrency wallet used to interact with the Ethereum blockchain.**

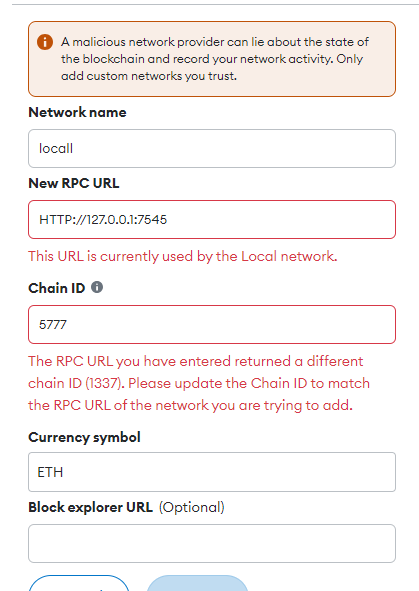
****

****

**Now in Metamask goto top right corner and click it and add network**

****

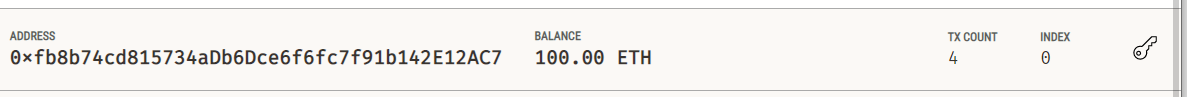
**Now we add a network for our local test blockchain to connect metamast with ganache.**

**** ****

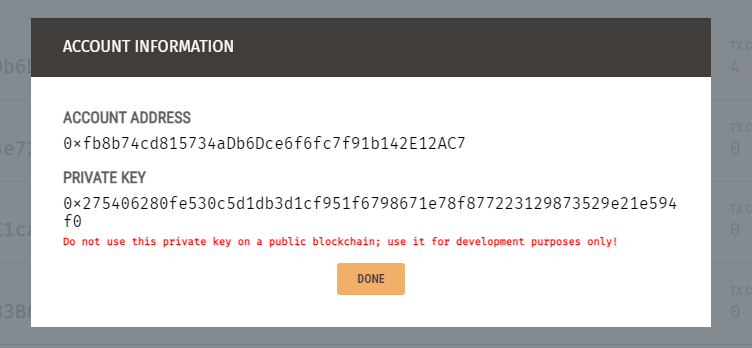
get these from ganache



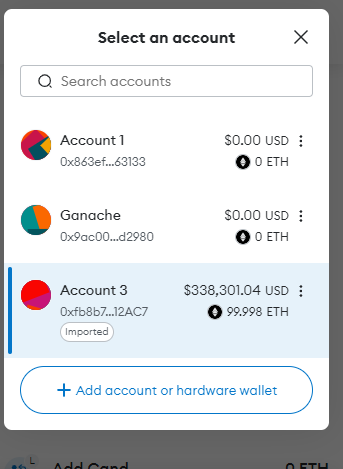
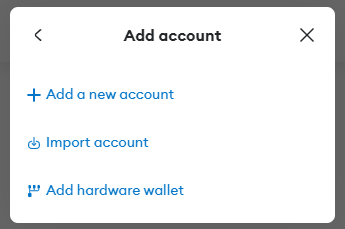
**Now add account in metamask and get private key for account from ganache**

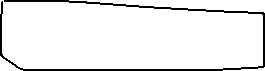
****

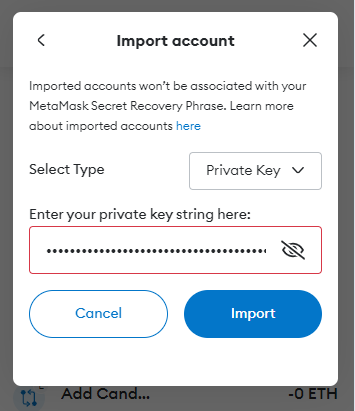


****

**Copy private key and paste it in metamask**

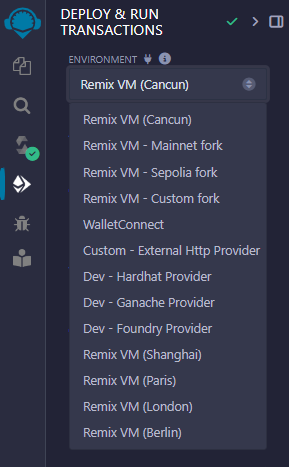
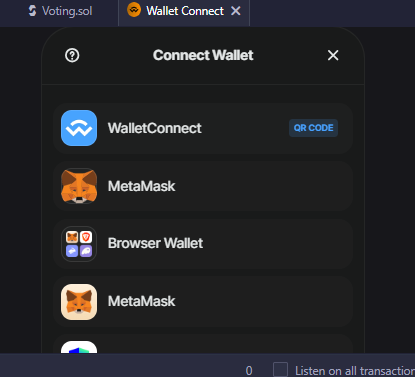
**** ****





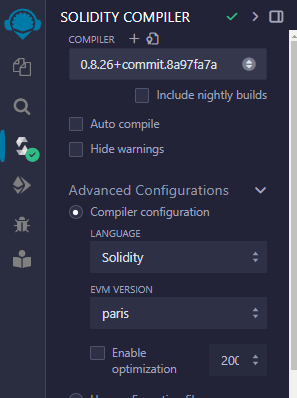
**Now in remix IDE**

**Goto deploy and press Environment and then Wallet connect and select Metamask from it**

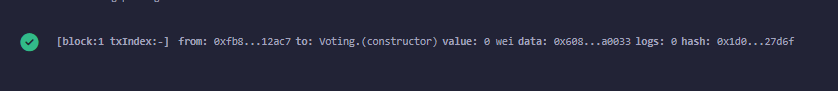
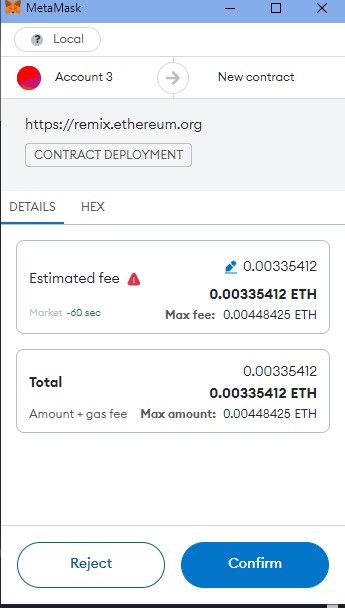
**** ****

Then metamask will ask you if you want to allow this to connect to remix or not, allow it.

**Then in compilation tab select EVM as paris and press compile to compile the script.**

****

**And then deploy it from the deploy tab.**



**O Define the Contract Structure: Outline the structure of your contract. This should include state variables, modifiers, constructor, and functions.**

State Variables: Define variables to store the owner address, candidate details (ID, name, vote count), and voter status.

struct Candidate

    {

        uint CandidateID;

        string Name;

        uint VoteCount;

    }

 mapping(uint => Candidate) Candidates;

    mapping(address => bool) Voter;

    uint CandidateNo;

    address public Owner;

Modifiers: Create a modifier to restrict certain functions to the contract owner.

 modifier OnlyManager()

    {

        require(msg.sender == Owner, "You are not the Owner");

        \_;

    }

Constructor: Initialize the contract by setting the deployer as the owner.

 constructor()

    {

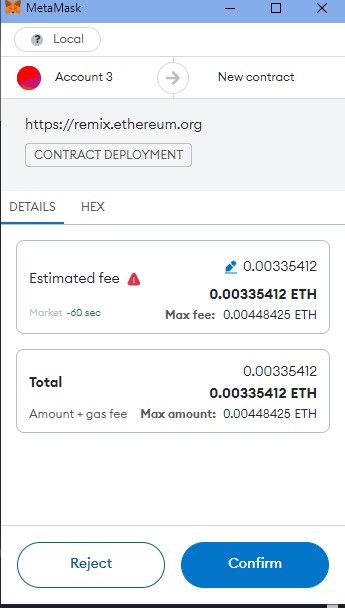
        Owner = msg.sender;

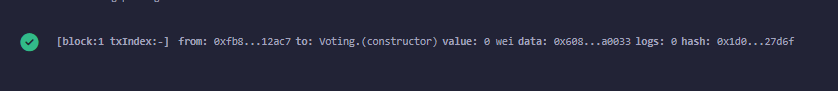
        //gives address of person runnig the smart contract

    }

**5. Compile the Smart Contract:**

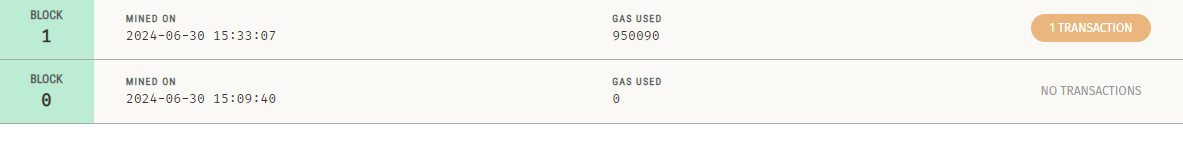
o Run the Compiler: In the terminal, compile the contract by running:

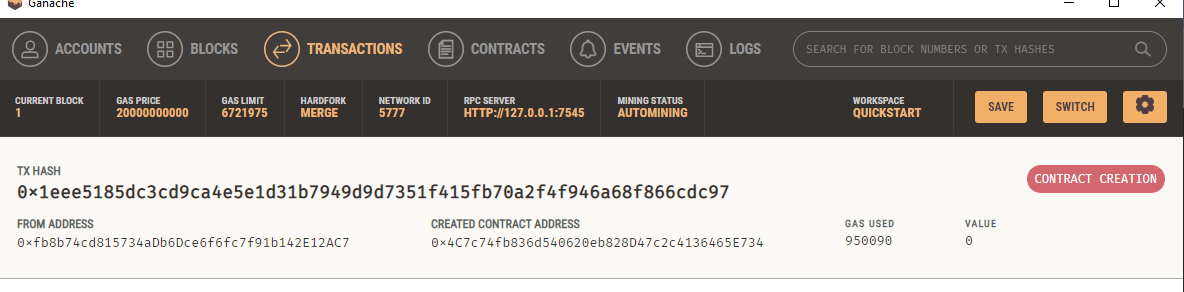




**6. Deploy the Smart Contract:**

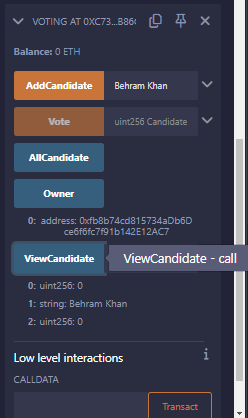
o Deploy to Local Blockchain: Start Ganache to run a local blockchain, then deploy the contract by running:





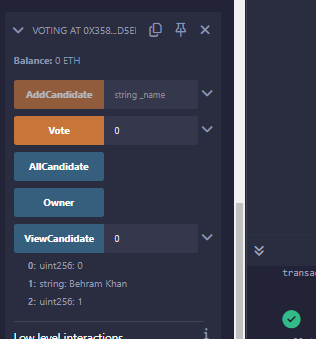
**7. Interact with the Smart Contract:**

o Add Candidates: Use the addCandidate function to add candidates to the election.

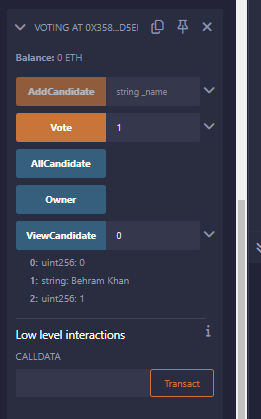


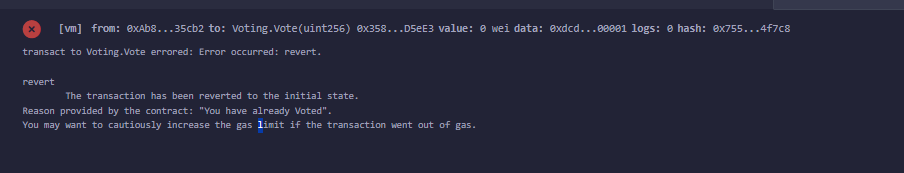
* Added by Owner 0 address: 0xfb8b74cd815734aDb6Dce6f6fc7f91b142E12AC7

o Vote for Candidates: Use the vote function to cast votes for candidates.



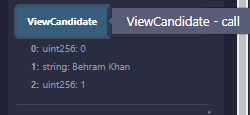
Vote casted by address 0xA... to candidate 0 Successfully



Vote casted by address 0xA... to candidate 1 so we get error

**Cannot cast 2 votes**

o View Results: Use the getCandidate function to view the current vote count foreach candidate.

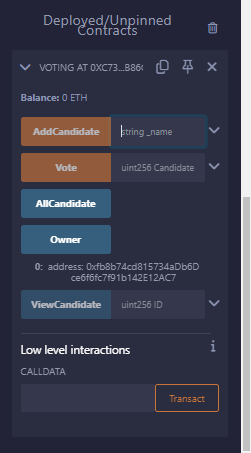


**6. Details of the Features Required in the Smart Contract:**

To create a fully functional voting smart contract, the following features should be included:

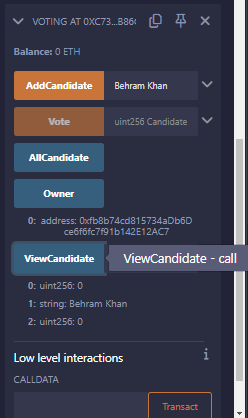
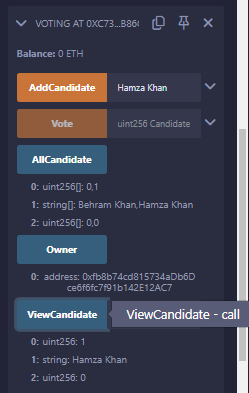
**1. Owner Management:**

o State Variable for Owner: Store the address of the contract owner.

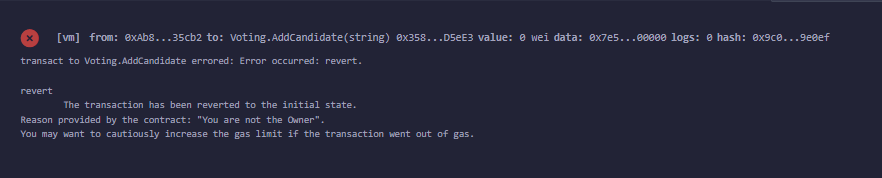


o Modifier for Owner-Only Functions: Restrict certain functions to be callable

only by the contract owner.

* Added by Owner 0 address: 0xfb8b74cd815734aDb6Dce6f6fc7f91b142E12AC7



**When not Owner we get error**

**2. Candidate Management:**

o State Variable for Candidates: Use a struct to define candidate properties (ID, name, vote count).

 struct Candidate // A structure defining election candidate attributes.

    //using a struct so we can have multiple candidates with attributes and can access each candidate using a uint mapping

    // 0 -> candidate1

    // 1 -> candidate2 and so on..

    {

        uint CandidateID;

        string Name;

        uint VoteCount;

    }

o Mapping for Candidates: Map candidate IDs to their respective Candidate struct.

    mapping(uint => Candidate) Candidates;

o Function to Add Candidates: Allow the owner to add new candidates to the election.

 function AddCandidate(string calldata \_name) public OnlyManager// we take candidate name as input

    //funcction to add elecrion candidates, and only manager can use it becuase of OnlyManager modifier.

    {

        //We are making a \_candidate struct object and setting it as first enter of the struct

        //Initially Candidates[candidateNo] = Candidates[0] so its first electrion candidate enter.

        //we set its attributes and increment the CandidateNo interger so that next candidate is always different and doesnt

        //override this one.

        Candidate storage \_candidate = Candidates[CandidateNo];

        \_candidate.Name = \_name;

        \_candidate.CandidateID = CandidateNo;

        \_candidate.VoteCount = 0; //initally votes are 0

        CandidateNo++;

    }

**3. Voting Mechanism:**

o State Variable for Voters: Use a mapping to track whether an address has voted.

 mapping(address => bool) Voter; // a key-value pair of voters address with a bool to check if the voter has cast vote or not

    // initially its false so every voter has not cast vote

o Function to Cast Votes: Allow users to vote for a candidate by specifying the candidate ID. Ensure each voter can only vote once and that the candidate ID is valid.

    function Vote(uint CandidateId) public // We take candidate id as input for voting

    {

        require(Voter[msg.sender] == false, "You have already Voted");// if Voter who is using this function has not voted

        //then it will be == false and we can vote else we will stop here as we have voted and can only cast 1 vote.

        require(CandidateId < CandidateNo);// the input candidateID for vote should be elss then total candidates, no illegal entry

        //also since its uint, it cant be negative

        Voter[msg.sender] = true; // we set vote to true so we cant cast vote again.

        Candidates[CandidateId].VoteCount += 1;// add vote to the candidates vote count.

    }

**4. Result Viewing:**

o Function to Get Candidate Details: Allow anyone to view the details of a specific candidate, including their ID, name, and vote count.

function ViewCandidate(uint ID) public view returns(uint, string memory, uint)

    //We can see a specific candidates result by inputing their id.

    {

        return (Candidates[ID].CandidateID, Candidates[ID].Name,Candidates[ID].VoteCount);

    }

o Function to Get All Candidates: Optionally, provide a function to retrieve details of all candidates.

    function AllCandidate() public view returns(uint[] memory, string[] memory, uint[] memory)

    //to see all candidates result, we store candidates and their attributes in arrays and then loop over them and output it

    {

        uint[] memory ids= new uint[](CandidateNo);

        string[] memory name= new string[](CandidateNo);

        uint[] memory voteCount= new uint[](CandidateNo);

        uint i; //0

        while(i < CandidateNo)

        {

            ids[i] = Candidates[i].CandidateID;

            name[i] = Candidates[i].Name;

            voteCount[i] = Candidates[i].VoteCount;

            i++;

        }

        return(ids,name,voteCount);

    }

}