MAXWELL TETTEH

AMSCU3CDS20052

S5 BCA-DATA SCIENCE

20CSA349-BLOCKCHAIN TECHOLOGIES

ASSIGNMENT:

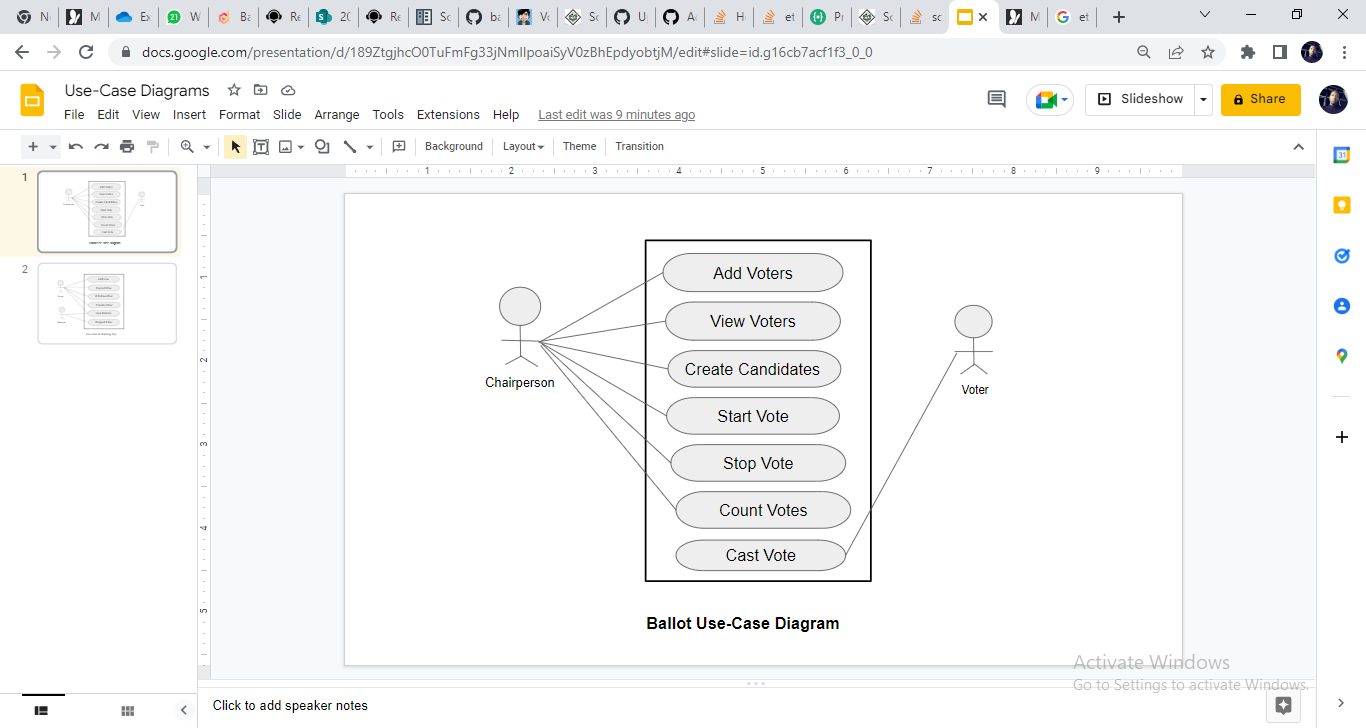
QUESTION 1:

**Ballot.sol**

1. [Ballot.sol](https://amritauniv-my.sharepoint.com/:u:/g/personal/jinesh_am_amrita_edu/ETLj-3iFoqpIqCmdg5fXA18BWsL34DutNG5oIbnUGGBxbg?e=khjKnF) is a smart contract to conduct voting with a specific set of candidates. Modify this contract with the following functionalities

* Chairperson can specify the voters (hint: take a set of valid EOA as voters list)
* Chairperson can stop the voting

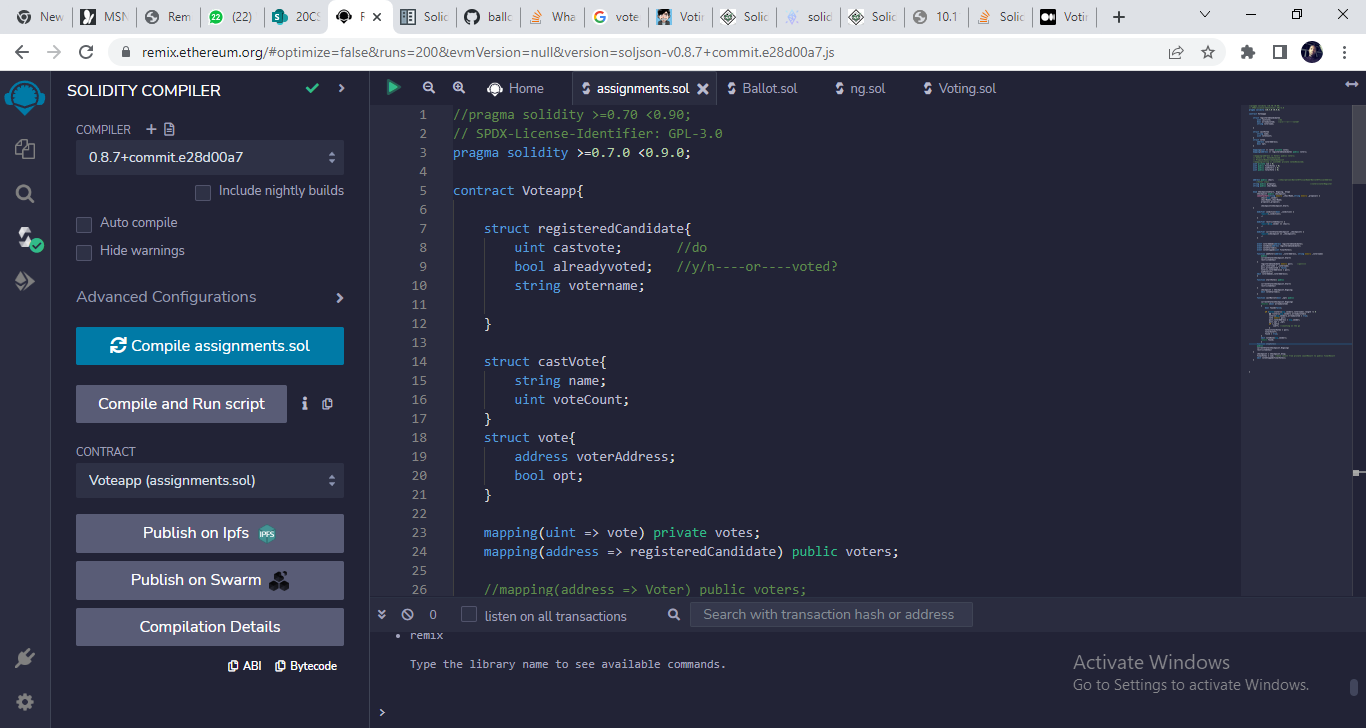
Use-Case Diagram:

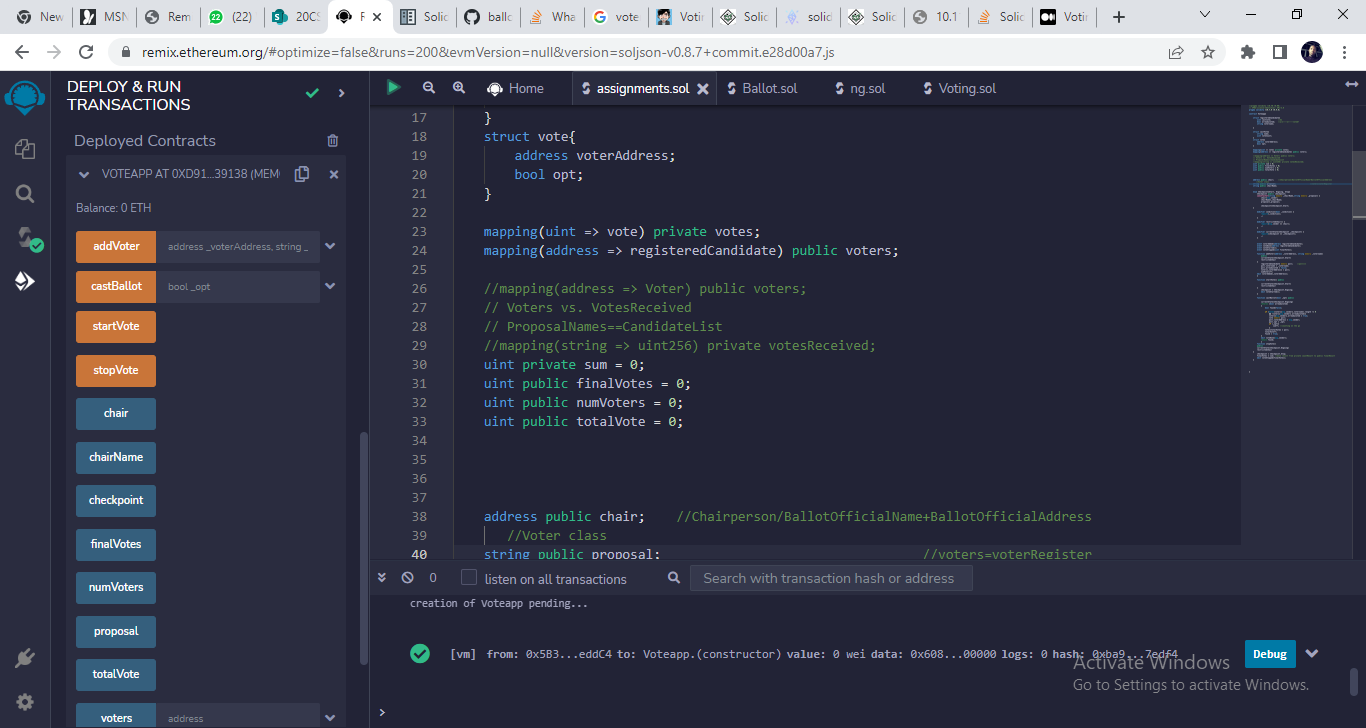


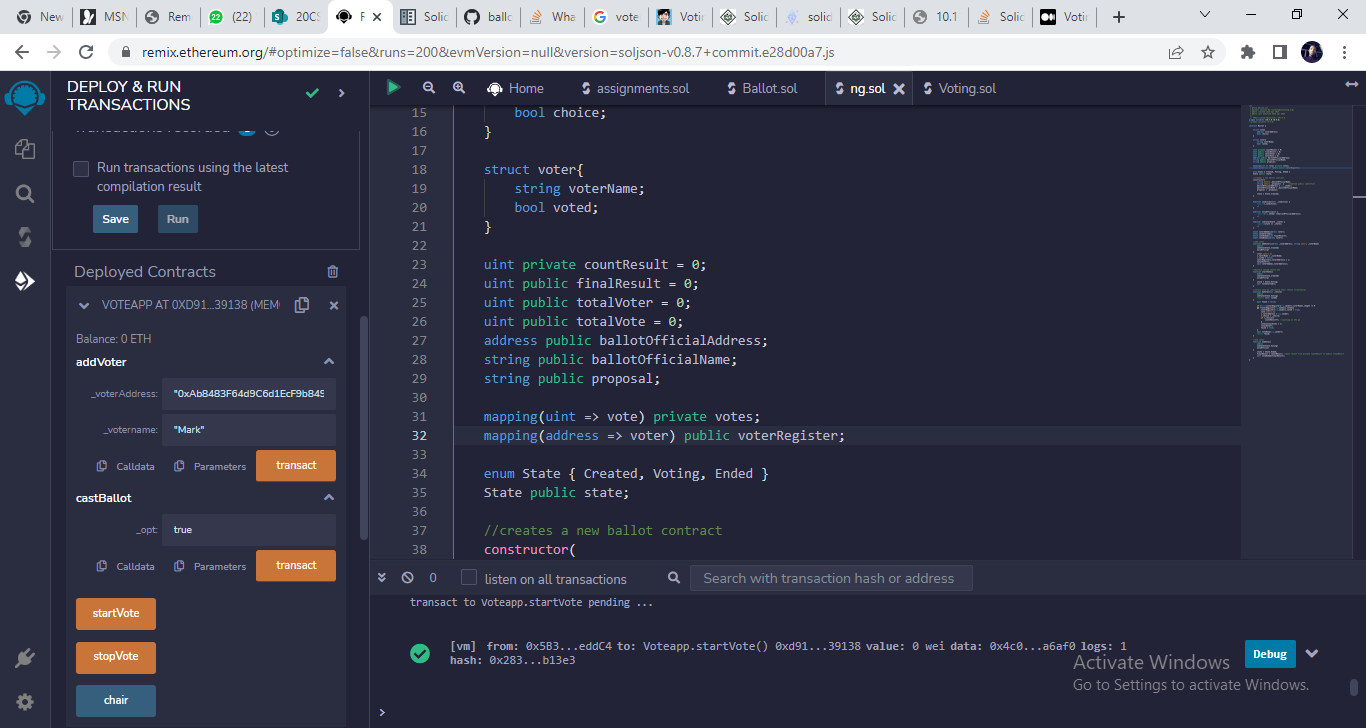
**Code:**

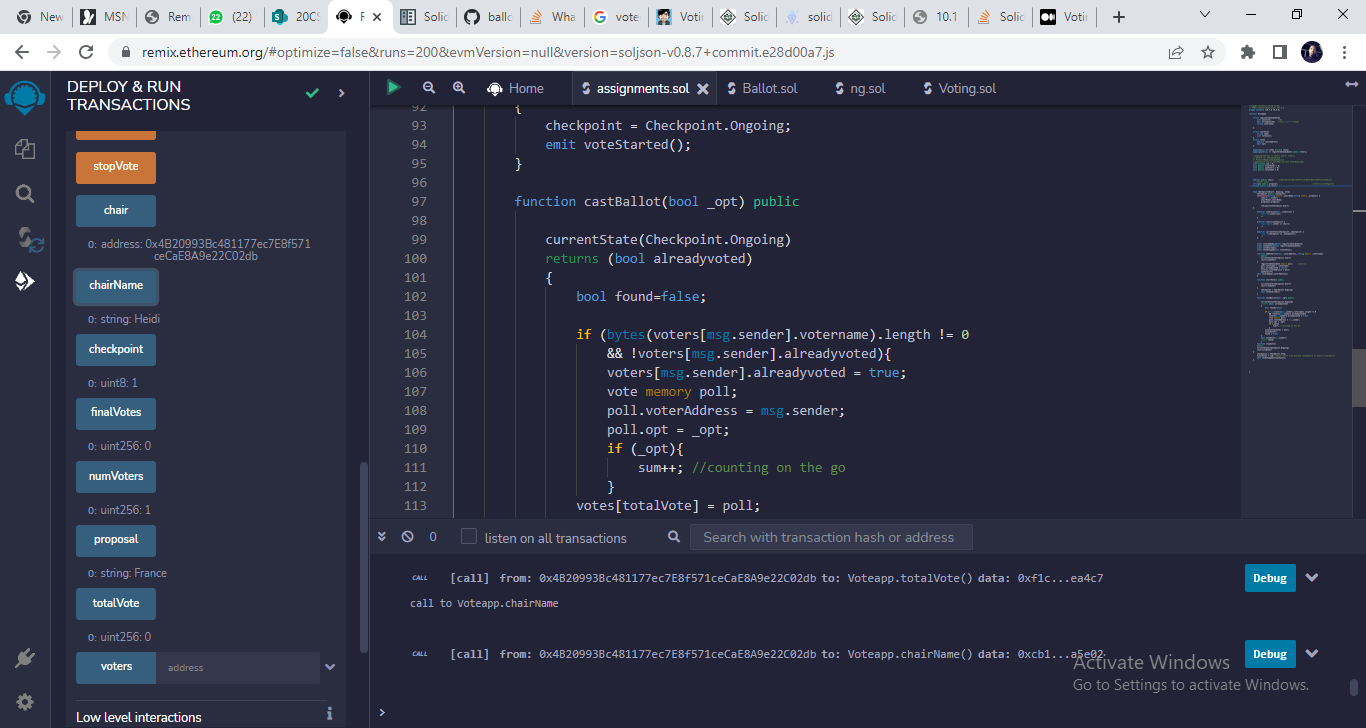
|  |
| --- |
| //pragma solidity >=0.70 <0.90;  // SPDX-License-Identifier: GPL-3.0  pragma solidity >=0.7.0 <0.9.0;  contract Voteapp{      struct registeredCandidate{          uint castvote;       //do          bool alreadyvoted;   //y/n----or----voted?          string votername;      }      struct castVote{          string name;          uint voteCount;      }      struct vote{          address voterAddress;          bool opt;      }      mapping(uint => vote) private votes;      mapping(address => registeredCandidate) public voters;      //(registeredCandidate => voters);        //mapping(string => uint256) private votesReceived;      uint private sum = 0;      uint public finalVotes = 0;      uint public numVoters = 0;      uint public totalVote = 0;        address public chair;    //Chairperson   //Voter class      string public proposal;      string public chairName;      enum Checkpoint{Start, Ongoing, Stop}          Checkpoint public checkpoint;          constructor(string memory \_chairName,string memory \_proposal) {              chair=msg.sender;              chairName=\_chairName;              proposal=\_proposal;              checkpoint=Checkpoint.Start;      }          modifier condition(bool \_condition) {              require(\_condition);              \_;          }          modifier restrictedChair() {              require(msg.sender == chair);              \_;          }          modifier currentState(Checkpoint \_checkpoint) {              require(checkpoint == \_checkpoint);              \_;          }            event voterAdded(address registeredCandidate);          event voteDone(address registeredCandidate);          event voteStarted();          event voteStopped(uint finalVotes);          function addVoter(address \_voterAddress, string memory \_votername)              public              currentState(Checkpoint.Start)              restrictedChair          {              registeredCandidate memory poll;    //poll              poll.votername = \_votername;              poll.alreadyvoted = false;              voters[\_voterAddress] = poll;              numVoters++;          emit voterAdded(\_voterAddress);          }          function startVote() public                currentState(Checkpoint.Start)              restrictedChair          {              checkpoint = Checkpoint.Ongoing;              emit voteStarted();          }          function castBallot(bool \_opt) public                currentState(Checkpoint.Ongoing)              returns (bool alreadyvoted)              {                  bool found=false;                  if (bytes(voters[msg.sender].votername).length != 0                      && !voters[msg.sender].alreadyvoted){                      voters[msg.sender].alreadyvoted = true;                      vote memory poll;                      poll.voterAddress = msg.sender;                      poll.opt = \_opt;                      if (\_opt){                          sum++;                      }                  votes[totalVote] = poll;                  totalVote++;                  found = true;              }              emit voteDone(msg.sender);              return found;              }          function stopVote()          public          currentState(Checkpoint.Ongoing)          restrictedChair      {          checkpoint = Checkpoint.Stop;          finalVotes = sum;          emit voteStopped(finalVotes);      }    } |

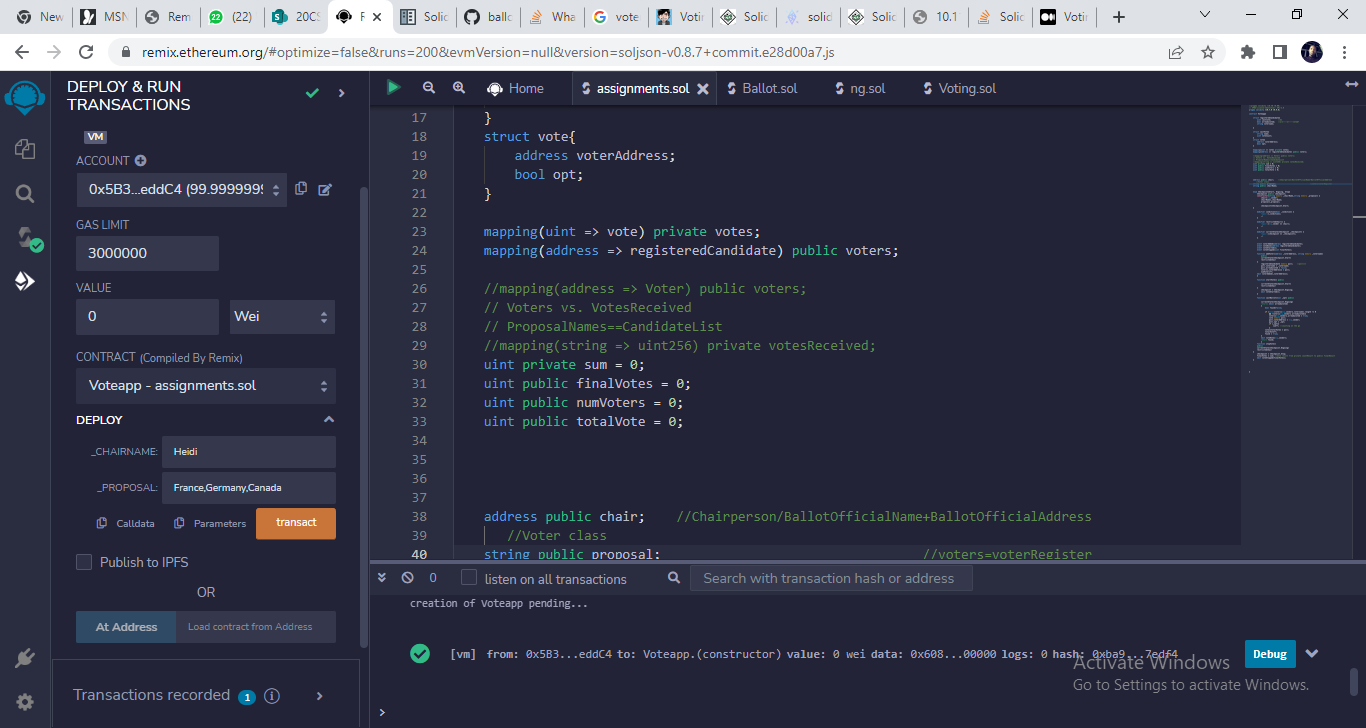
**Execution Screenshots:**

****

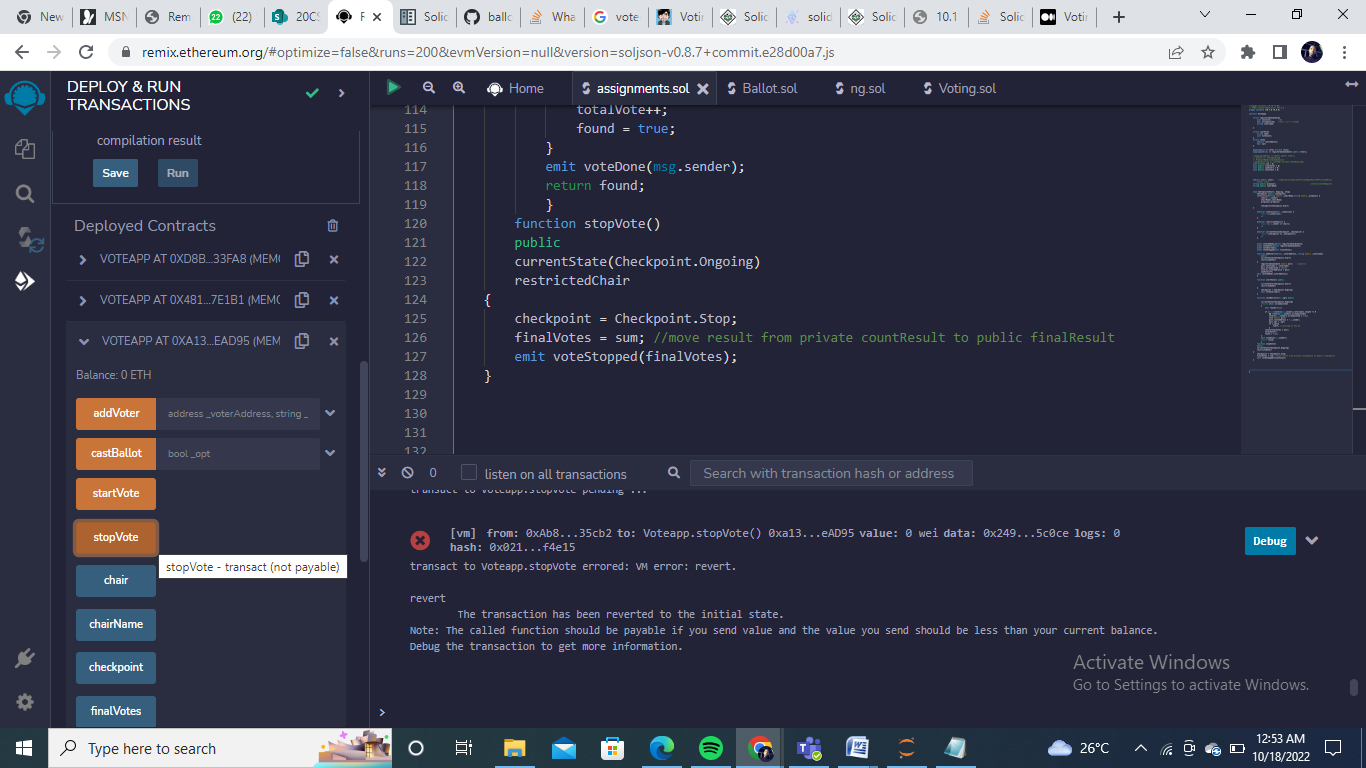
****

****

****



Changing the Address to a different one other than the Chairperson/Admin reverts the transaction when we try and access functions that are restricted for only the Chairperson to call/view.



**QUESTION 2:**

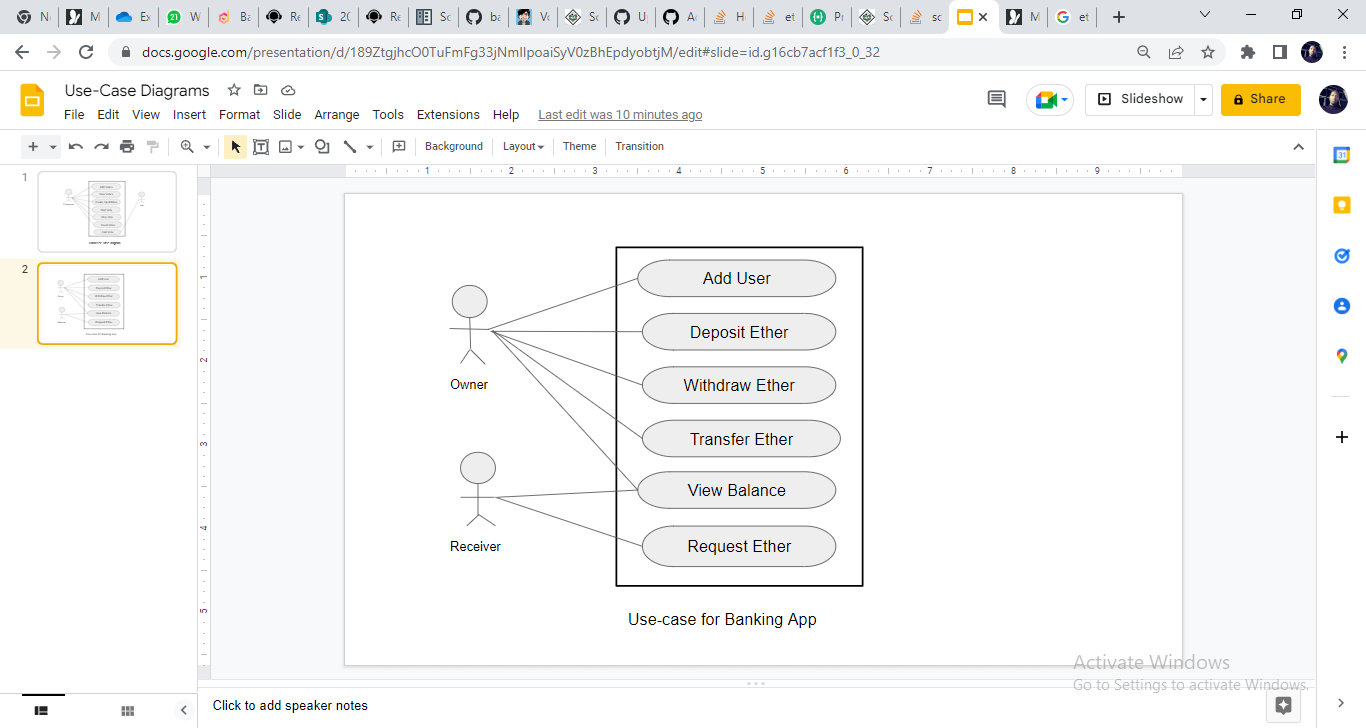
1. Implement a simple bank application using a smart contract with the following functionalities

a. Deposit the money

b. Allow the withdrawal by keeping a minimum balance (set minimum balance as 1 ETH)

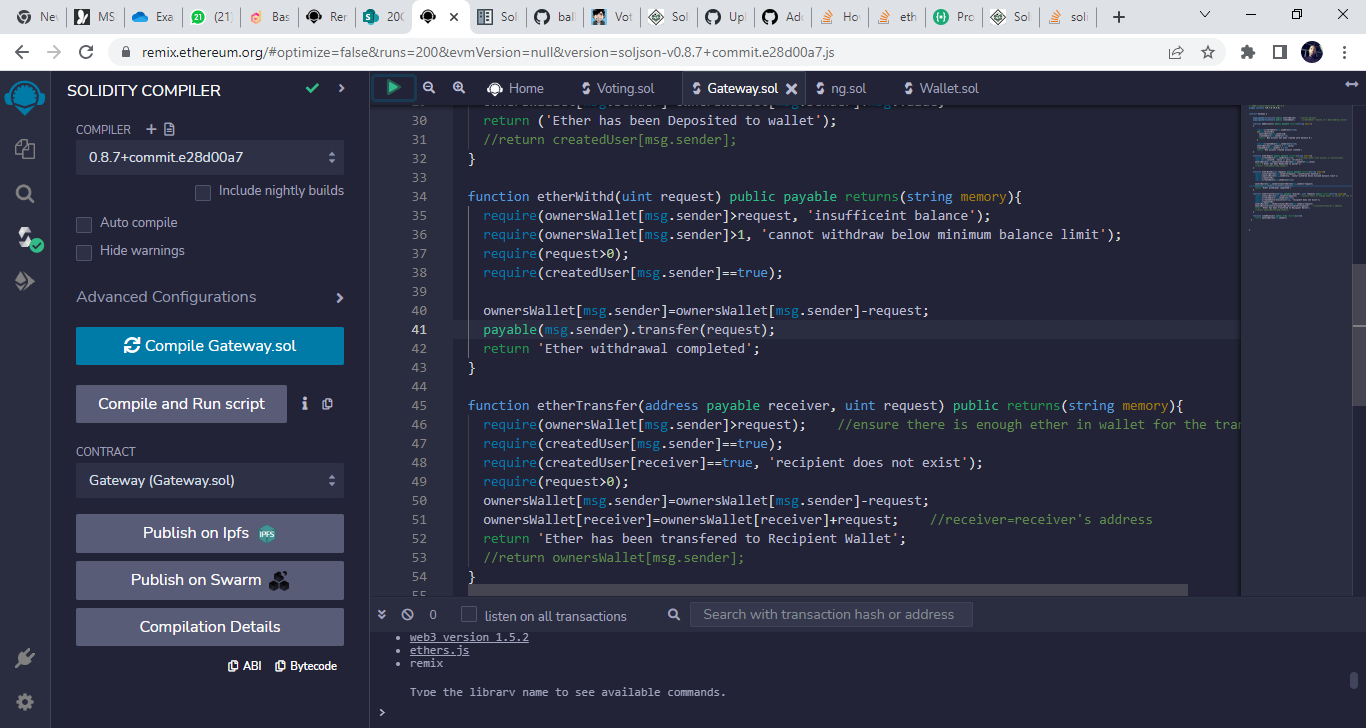
c. Transfer money between two valid accounts

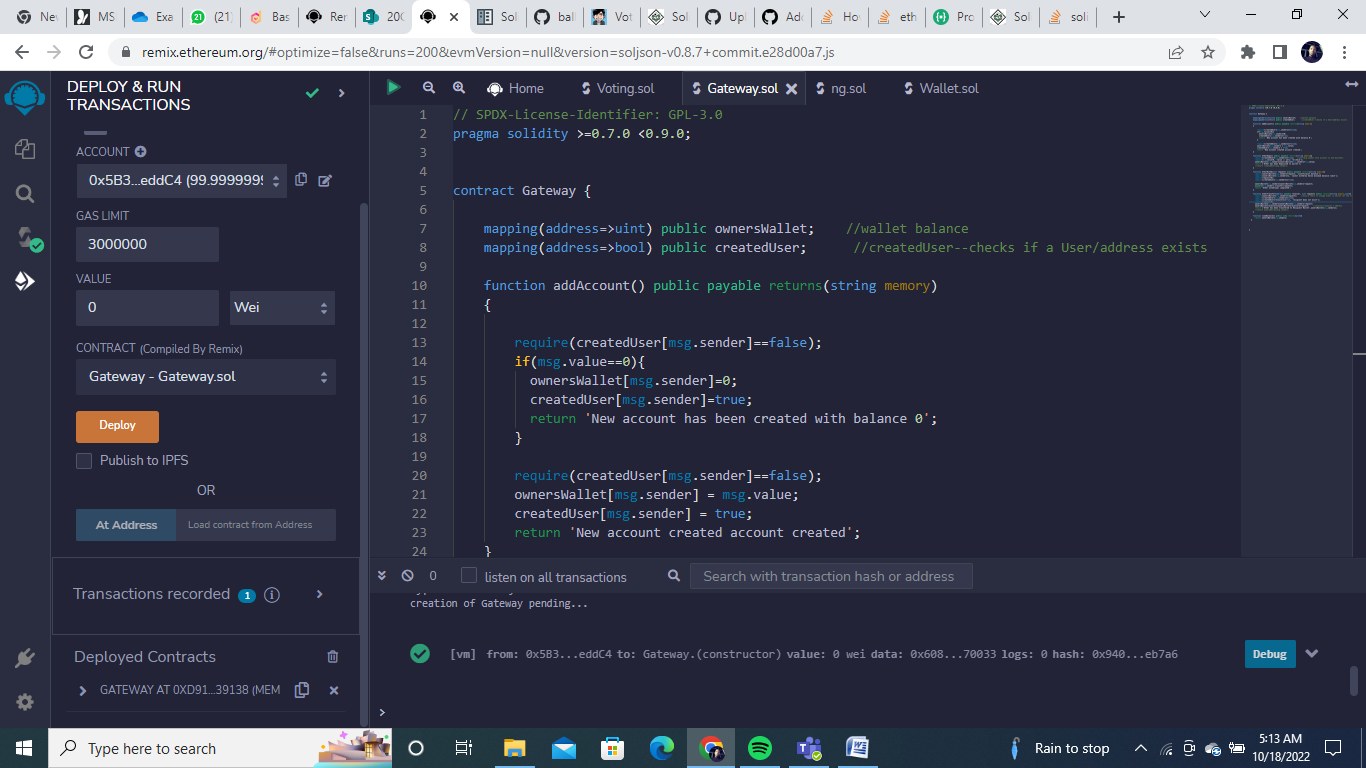
Use-Case Diagram:

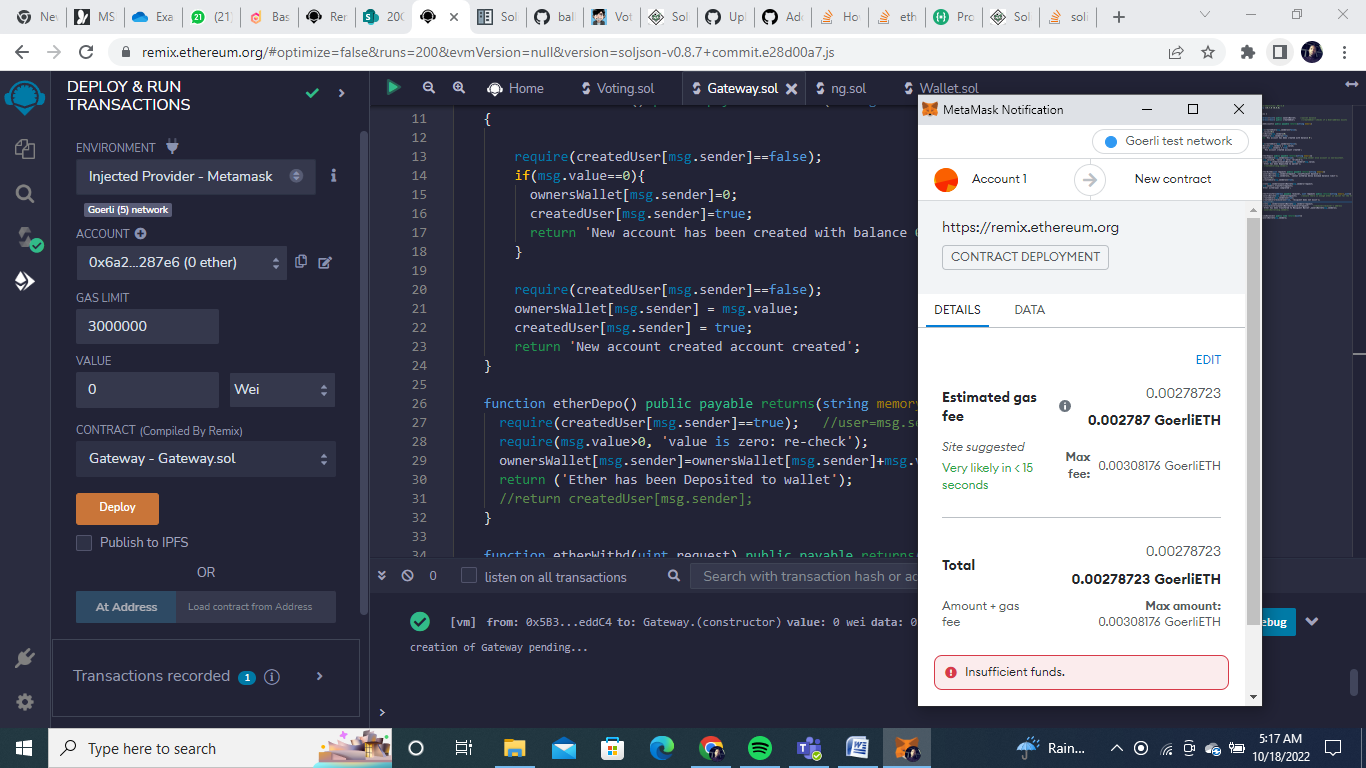


**Code:**

|  |
| --- |
| // SPDX-License-Identifier: GPL-3.0  pragma solidity >=0.7.0 <0.9.0;  contract Gateway {        mapping(address=>uint) public ownersWallet;    //wallet balance      mapping(address=>bool) public createdUser;      //createdUser--checks if a User/address exists        function addAccount() public payable returns(string memory)      {            require(createdUser[msg.sender]==false);          if(msg.value==0){            ownersWallet[msg.sender]=0;            createdUser[msg.sender]=true;            return 'New account has been created with balance 0';          }          require(createdUser[msg.sender]==false);          ownersWallet[msg.sender] = msg.value;          createdUser[msg.sender] = true;          return 'New account created account created';      }      function etherDepo() public payable returns(string memory){        require(createdUser[msg.sender]==true);   //user=msg.sender else account is non-existent.        require(msg.value>0, 'value is zero: re-check');        ownersWallet[msg.sender]=ownersWallet[msg.sender]+msg.value;        return ('Ether has been Deposited to wallet');        //return createdUser[msg.sender];      }      function etherWithd(uint request) public payable returns(string memory){        require(ownersWallet[msg.sender]>request, 'insufficeint balance');        require(ownersWallet[msg.sender]>1, 'cannot withdraw below minimum balance limit');        require(request>0);        require(createdUser[msg.sender]==true);          ownersWallet[msg.sender]=ownersWallet[msg.sender]-request;        payable(msg.sender).transfer(request);        return 'Ether withdrawal completed';      }      function etherTransfer(address payable receiver, uint request) public returns(string memory,uint){        require(ownersWallet[msg.sender]>request);    //ensure there is enough ether in wallet for the transfer        require(createdUser[msg.sender]==true);        require(createdUser[receiver]==true, 'recipient does not exist');        require(request>0);        ownersWallet[msg.sender]=ownersWallet[msg.sender]-request;        ownersWallet[receiver]=ownersWallet[receiver]+request;    //receiver=receiver's address        return ('Ether has been transfered to Recipient Wallet',ownersWallet[msg.sender]);        //return ownersWallet[msg.sender];      }      function viewBalance() public view returns(uint){        return ownersWallet[msg.sender];    }    } |







With GoerLiETH Wallet(MetaMask):

