pragma solidity 0.7.5;

pragma abicoder v2;

contract Wallet {

address[] public owners;

uint limit;

struct Transfer{

uint amount;

address payable receiver;

uint approvals;

bool hasBeenSent;

uint id;

}

event TransferRequestCreated(uint \_id, uint \_amount, address \_initiator, address \_receiver);

event ApprovalReceived(uint \_id, uint \_approvals, address \_approver);

event TransferApproved(uint \_id);

Transfer[] transferRequests;

mapping(address => mapping(uint => bool)) approvals;

//Should only allow people in the owners list to continue the execution.

modifier onlyOwners(){

bool owner = false;

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

if(owners[i] == msg.sender){

owner = true;

}

}

require(owner == true);

\_;

}

//Should initialize the owners list and the limit

constructor(address[] memory \_owners, uint \_limit) {

owners = \_owners;

limit = \_limit;

}

//Empty function

function deposit() public payable {}

//Create an instance of the Transfer struct and add it to the transferRequests array

function createTransfer(uint \_amount, address payable \_receiver) public onlyOwners {

emit TransferRequestCreated(transferRequests.length, \_amount, msg.sender, \_receiver);

transferRequests.push(

Transfer(\_amount, \_receiver, 0, false, transferRequests.length)

);

}

//Set your approval for one of the transfer requests.

//Need to update the Transfer object.

//Need to update the mapping to record the approval for the msg.sender.

//When the amount of approvals for a transfer has reached the limit, this function should send the transfer to the recipient.

//An owner should not be able to vote twice.

//An owner should not be able to vote on a tranfer request that has already been sent.

function approve(uint \_id) public onlyOwners {

require(approvals[msg.sender][\_id] == false);

require(transferRequests[\_id].hasBeenSent == false);

approvals[msg.sender][\_id] = true;

transferRequests[\_id].approvals++;

emit ApprovalReceived(\_id, transferRequests[\_id].approvals, msg.sender);

if(transferRequests[\_id].approvals >= limit){

transferRequests[\_id].hasBeenSent = true;

transferRequests[\_id].receiver.transfer(transferRequests[\_id].amount);

emit TransferApproved(\_id);

}

}

//Should return all transfer requests

function getTransferRequests() public view returns (Transfer[] memory){

return transferRequests;

}

}