

Roll No: 20BCE204

Course: 2CSDE93 - Blockchain Technology

Practical No: 6

Aim: To build, implement and test voting mechanisms using Ethereum Blockchain. First, list the contestants on the screen and the vote they got. Whenever the user tries to vote for a particular contestant, the count of the votes for the particular contestant should increase by 1. Also, the user who has already voted should be marked. Marked means “the user has already voted once and will not be allowed to vote again”.

Code:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.9.0;

contract Ballot {

    modifier onlyOwner() {
        require(msg.sender == owner, "Only the owner can
call this function.");
        _;
    }

    /* data structures */
    address owner;
    struct Voter {
        bool voted;
        string votedTo;
        uint256 vote;
    }

    struct Proposal {
```

```

    string name;
    uint256 voteCount;
}

mapping(address => Voter) voters; // for keeping
track of who has already voted

Proposal[] public proposals; // array of the
candidates

/// Create a new ballot to choose one of
`proposalNames`.
constructor(string[] memory proposalNames) {
    owner = msg.sender;
    for (uint256 i = 0; i < proposalNames.length;
i++) {
        proposals.push(Proposal({name:
proposalNames[i], voteCount: 0}));
    }
}

function vote(uint256 proposal) external {
    Voter storage sender = voters[msg.sender];
    require(!sender.voted, "You have already voted.
Can not vote again!!");
    sender.voted = true;
    sender.vote = proposal;
    sender.votedTo = proposals[proposal].name;
    proposals[proposal].voteCount += 1;
}

function winningProposal() public onlyOwner view
returns (uint256 elected_winner) {

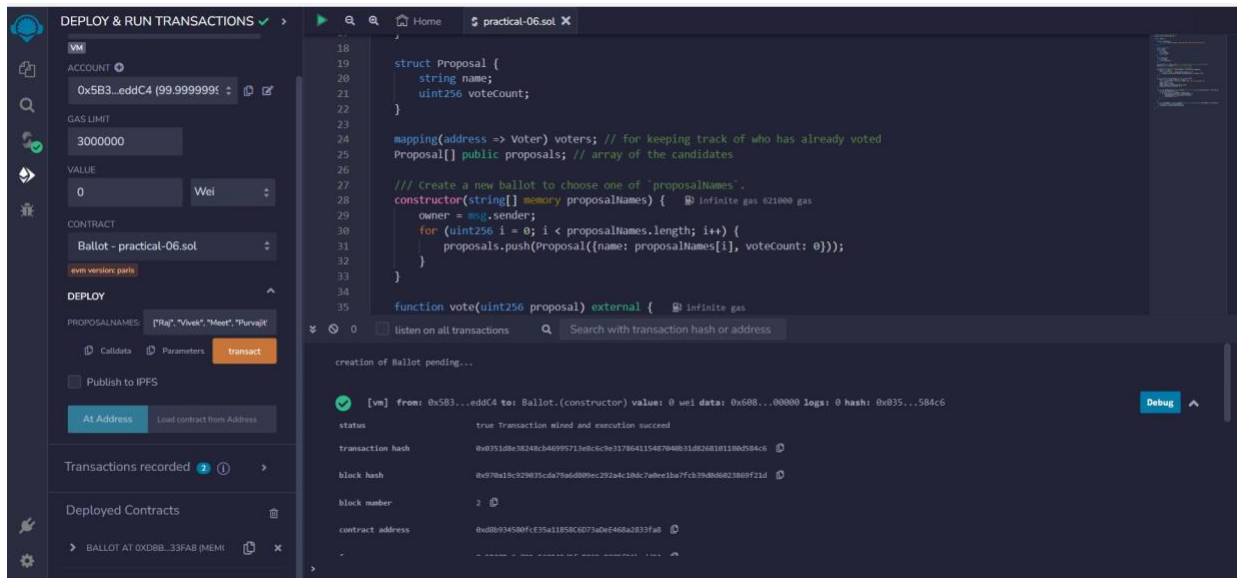
```

```
uint256 winningVoteCount = 0;
for (uint256 p = 0; p < proposals.length; p++) {
    if (proposals[p].voteCount >
winningVoteCount) {
        winningVoteCount =
proposals[p].voteCount;
        elected_winner = p;
    }
}

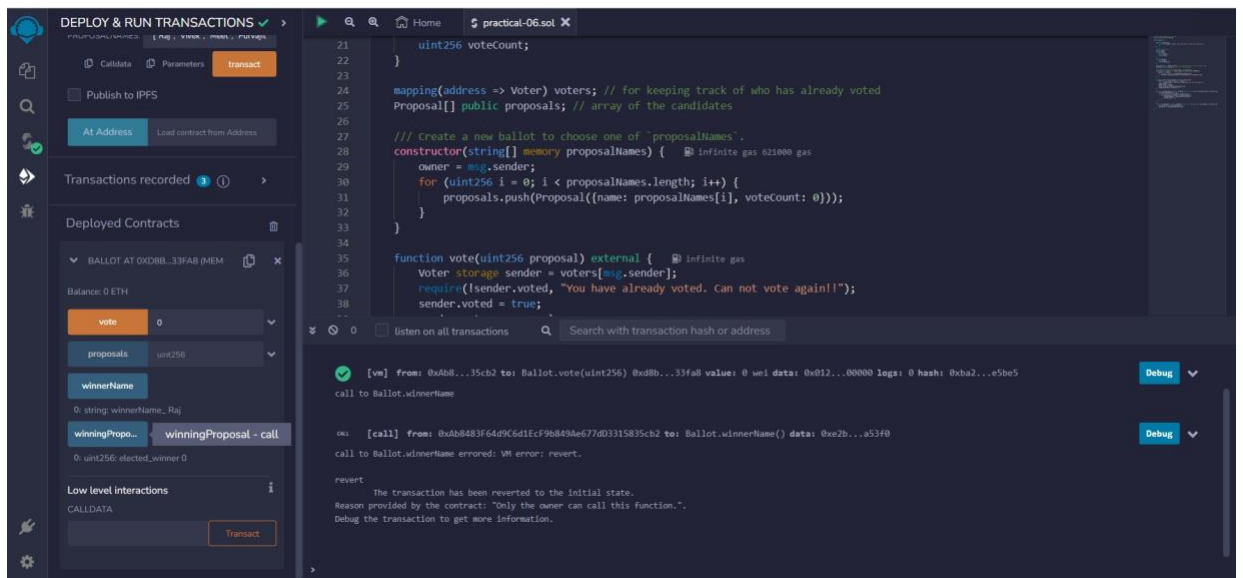
function winnerName() external onlyOwner view returns
(string memory winnerName_) {
    uint256 winnerId = winningProposal();
    winnerName_ = proposals[winnerId].name;
}
}
```

Output:

1. Contact deployment



2. Voting for a candidate



3. Can not vote again

The screenshot shows a web application interface on the left and a debug console on the right. The web application has a sidebar with a 'vote' button and a 'proposals' dropdown. The main area shows a 'winnerName' field with the value 'Raj' and a 'winningPropo...' field with the value '0'. Below these fields is a 'Low level interactions' section with a 'CALLDATA' field and a 'Transact' button. The debug console on the right shows a transaction that failed with the message 'The transaction has been reverted to the initial state. Reason provided by the contract: "You have already voted. Can not vote again!!". Debug the transaction to get more information.' The console also shows the transaction details: '[vm] from: 0x488...35cb2 to: Ballot.vote(uint256) @x8b...33fa8 value: 0 wei data: 0x012...00001 logs: 0 hash: 0x373...939b4'.

4. Only owner can announce the results

The screenshot shows the same web application interface as in the previous image, but with different values. The 'winnerName' field now shows 'Raj' and the 'winningPropo...' field shows '0'. The 'Low level interactions' section is still present. The debug console on the right shows two successful transactions. The first transaction is a call to 'Ballot.winnerName' with data '0xe2b...a53f0'. The second transaction is a call to 'Ballot.winningProposal' with data '0x689...ff1bd'. Both transactions are from the address '0x58380a701c568545dcfc803fc8875f56bed8c4'.