University of Groningen
Information Systems
Assignment 2
Group 18

Satyanarayan Nayak, Swastik
s.nayak.1@student.rug.nl
S4151968

Mitra, Siddharth
s.mitra.2@student.rug.nl
S4138430

December 2, 2019

Contents

1 Problem Statement

Use **remix.ethereum.org** to create a smart contract for the following use case
Use case: A company's sole managing director wants to allow for the shareholders to make (binary) decisions, which he will propose to the shareholders. (The director is aware of the fact that votes made by shareholders are public since they are stored on the Blockchain.)

Create an Ethereum smart contract that allows for the following functionality:

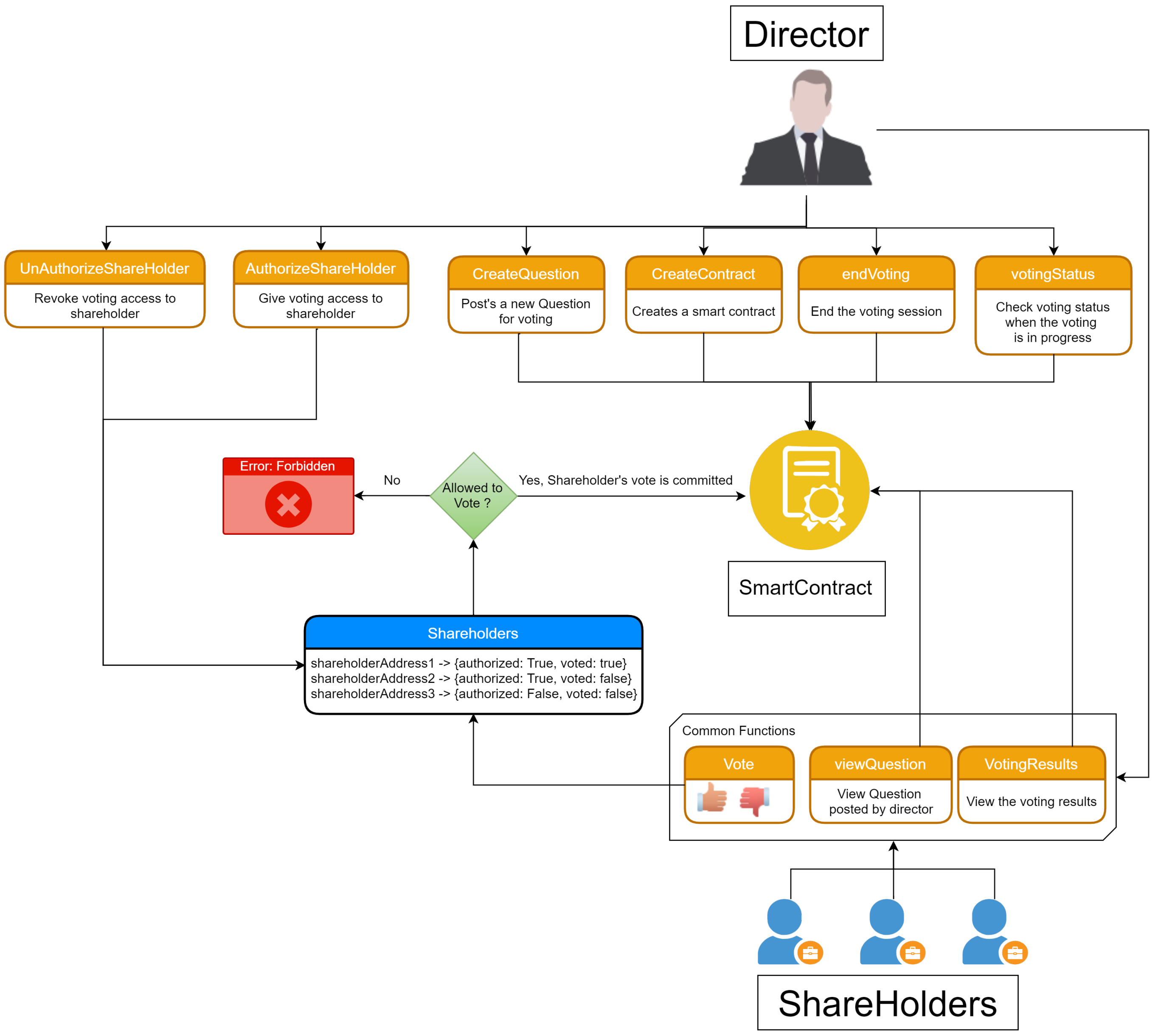
1. The director will be the one to upload the contract. He should thereafter be recognised as the director because he was the one to upload the contract.
2. The director would like the ability to upload any number of questions (which require a true or false response to). The director will upload each question, one at a time.
3. The director would like the ability to add and remove shareholders from being able to vote and being able to see results for approved decisions at any point.
4. Each shareholder may only vote for each decision once.
5. The director should be able to close the voting process for a specific question. The majority result should then be computed and able to be seen by all shareholders.

2 Assumptions

1. The director is the user who deploys the contract.
2. At a particular time, voting for only one question will be dealt with. In a real world scenario, the director and shareholders discuss one question at any given time to avoid ambiguity in their discussions and the voting or confidence vote is performed at the end of a discussion on any given subject. It is unlikely that a director would discuss two or more questions at a given point of time.
3. There is no limit to the number of shareholders the company can have.
4. Only the director should have complete control over authorizing and unauthorizing a shareholder for any given voting session.
5. Only the director has the ability to end the voting process.
6. Only the director has the ability to view the voting status in the middle of a voting session.
7. The voting process begins as soon as the director creates a question and ends when the director invokes the function to end the voting process.
8. The shareholders are given access to only view the question posted by the director.
9. The shareholders should place their vote as binary values, either 0 or 1 (where, 0-disagree or false, 1-agree or true).
10. The shareholders have access to view the voting results once the director has closed the voting, but they are deferred from viewing the results in middle of the voting session.
11. The director should be able to post another question once the current voting sessions has ended.
12. In an event where the director's access to the contract should be revoked, the contract deployed should contain a function to revoke the director's access over the contract.

3 Design

3.1 Architecture



Director
UnAuthorizeShareHolder
Revoke voting access to shareholder
AuthorizeShareHolder
Give voting access to shareholder
CreateQuestion
Post's a new Question for voting
CreateContract
Creates a smart contract
endVoting
End the voting session
votingStatus
Check voting status when the voting is in progress
Error: Forbidden
No
Allowed to Vote ?
Yes, Shareholder's vote is committed
SmartContract
Shareholders
shareholderAddress1 -> {authorized: True, voted: true}
shareholderAddress2 -> {authorized: True, voted: false}
shareholderAddress3 -> {authorized: False, voted: false}
Common Functions
Vote
viewQuestion
View Question posted by director
VotingResults
View the voting results
ShareHolders

Figure 1: Cumulative Voting smart contract architecture

From the figure 1 we can see that the functionalities of the smart contract is mainly split into two distinct classifications.

- Functionalities invoked only by the director
  - Authorize a shareholder to vote (AuthorizeShareHolder).
  - Unauthorize a shareholder from voting (UnAuthorizeShareHolder).
  - Deploy the contract, the logic will lie in the constructor of the contract object, createContract is a proxy for representation purpose only.
  - Create and post new questions for the shareholders to vote upon (CreateQuestion).
  - End the voting to determine the result of a voting session (endVoting).
  - Check the voting status in the middle of a voting session (votingStatus).
- Common functions that can be invoked by both the director and the shareholders.

  - Cast a boolean vote over a question (vote).
  - View the question posted by the director (viewQuestion).
  - View the result of the voting after the voting session has ended or completed (votingResults).

Shareholders: A key value pair mapping (key → Address of the shareholder, value → A struct type ShareHolder) is maintained which will track the shareholder's activities and access rights.

Authorization: The value present in the shareholders mappings (ShareHolder struct type) contains a Boolean, member variable called 'authorized', this will track the shareholder's rights to vote on a given question. The ability to provide or revoke the voting rights can only be provided by the director, which implies that this data-structure can only be accessed and modified by the director.

Voting status: Similar to authorization, If a shareholder has voted on a given question, the member variable 'voted' should be updated to true.

endVoting: A functionality used to stop the ongoing voting process. The access to this function is restricted to the director.

VotingStatus and VotingResults: VotingStatus can only be invoked by the director and is used to view the status of the voting process while it is still in progress. the VotingResults is a public function and is only available after the voting has finished i.e, after the director invokes the endVoting function.

Majority of the functions and variables declared in the code is private, only the functions and variables that will be accessed by the shareholders are made public. Modifiers are used to restrict the public functions that should only be invoked by the director. The only publically accessible resources are vote, viewQuestion and VotingResults functions, where viewQuestion and VotingResults are read-only view functions and the vote is strictly restricted by the underlying logic and can only be used by the authorized shareholders, which in-turn can only be authorized by the director. The primary focus of this smart contract design was to,

- Keep the design simple and make sound decisions on usage of data-structures and logic.
- Implement the required features without any compromises.
- Restrict public access to as many resources as possible, and secure the resources that are publicly exposed using logic and modifiers.
- Clean coding practices and concise code documentation.

The raw implementation with code comments can be found in the Smart Contract complete code section. The detailed breakdown of the solidity implementation based on the above mentioned design can be found below. A demonstration of this smart contract implementation can also found in the Demo section.

3.2 Workflow

In the section, the workflow of the voting process is provided.

1. The director deploys a new contract in the block-chain network.
2. The director creates a new question, upon which voting is to be undertaken.
3. The director selects particular stakeholders whom he gives the ability to vote. He does this using the 'authorize-ShareHolder' function to which he provides the address of the shareholder as an input argument.
4. All the shareholders have the ability to view the question created by the director.
5. The shareholder who has been given the access to vote, can now vote for the question created by the director. The shareholder does this by invoking the 'vote' function and passing an binary value, either 0(disagreement or false) or 1(agreement or true) as the input argument.
   The binary values 0 and 1 were selected to avoid data type conversions inside the code and to better facilitate the underlying logic. As the summation of all the votes 0's and 1's cast by the shareholders directly relates to the total number of votes cast in favour of the question. This design reduces the temporal and spacial requirement by avoiding the need for a collection to store the votes and iterating over the collection to compute the total votes cast in favour of the question.

6. The director has the option of viewing the current state of the voting process while the voting is going on. He or she can do this by invoking the 'votingStatus' function.

7. The director has the option of closing the voting process at any given time, after which the results are calculated. To end the voting process the director invokes the 'endVoting' function. The function will compute the percentage of votes in favour of the question, if the percentage is greater than 50% the result of the voting is deemed to be in favour of the question (Majority:Agree) else the voting is not in favour of the question (Majority:Disagree).

8. Once the voting process is over, all the shareholders can view the results by invoking the 'VotingResults' function.

9. After the end of the voting process, the director can now create a new question and begin the voting process for that question.

10. Special scenario: In an event where the directors access should be revoked from the contract, the terminate function should be called.

3.3 Structures

ShareHolder: A structure used to store the ShareHolders state. The various properties of the ShareHolders tracked is explained in the below code snippet.

```
/*
 * @authorized -> A boolean flag indicating if a ShareHolder is allowed to vote or Not
 * @voted -> A boolean flag indicating if the ShareHolder has already voted.
*/
struct ShareHolder {
    bool authorized;
    bool voted;
}
```

3.4 Member Variables

```
// Stores the directors address.
address private director;

// Stores the contracts title or description or contract.
string public contract_title;

// Creates a map object (addressOfShareHolder -> ShareHolder struct), to keep track of
    shareholders voting.
mapping(address => ShareHolder) private ShareHolders;
address[] private addressLookupTable;

// Stores the total number for votes cast in the ballot.
uint private totalVotes = 0;

// Stores the total number of votes in favour of the contract proposed.
uint private votesInFavour = 0;
```

director: Address type variables used to store the address of the individual who created the contract. The variable is declared as private and is not exposed as the address stored in this variable holds the absolute authority over the contract.

contract_title: String type variable used to store the Contract on which the voting has to be performed. The variable is made public as the Shareholders should be able to view the contract on which they are voting.

ShareHolders: A key value pair Map object, where key is the address of the Shareholder and the value is the Shareholder struct type that is used to track the state of the Shareholder. The collection allows for a quick push and pop mechanism with a time complexity of approximately O(1).

addressLookupTable: Used to keep track of the shareholders currently registered. This collection is used to handle the environment reset after the voting is finished.

totalVotes: A private unsigned integer type variable used to track the total number of votes being cast in the voting session.

votesInFavour: A private unsigned integer type variable used to track the total number of votes in favour of the contract, that is the number of share-holders who are in agreeance with the contract.
votesInFavour and total Votes together are used to compute the result of the voting session. This designed is used to avoid the use of array or any other collections.

3.5 Constructor

```
constructor () public {
    /* msg.sender -> Contains the address of the address that deployed the contract.
    * Save the address of the deployer as director
    */
    director = msg.sender;
}
```

The constructor supports the critical functionality of initializing the director, who will have the complete authority of the preceding voting. The "msg.sender" is a variable provided by solidity which holds the address of the account that invoked the Contract or resources within the contract.

3.6 Modifiers

```
modifier directorOnly() {
    // Assert or require before executing the body _;
    require(msg.sender == director);
    _;}
```

directorOnly(): A reusable component used to restrict access to the resources. The member functions and member variables that are tagged by the modifier directorOnly can only be invoked by the sender or caller having the director's address. This eliminates the need to write multiple conditional checks and makes the code verbose.
example:

```
/* This function can only be invoked by the director */
function createContract(string memory _name) directorOnly public {
   /* Set the contract's Title/Description */
    contract_title = _name;
}
```

isVotingInProgress(): Used as a modifier to restrict some functions or resources to be invoked only during the progress of the voting.

```
modifier isVotingInProgress() {
    require(votingInProgress == true);
    _;
}
```

3.7 Functions

3.7.1 Create Question

```
    /*
    * Create a new Question.
    * @_name -> Title/Description of the Question.
    */
    function createQuestion(string memory _name) directorOnly public {
       /* Set the Question's Title/Description */
        Question = _name;
        votingInProgress = true;
        votesInFavour = 0;
        totalVotes = 0;
    }
```

The function can only be invoked by the director. The value passed during invoking is captured in the _name variable and is used to initialize the Question variable creating the question that will be voted upon.

3.7.2 Authorize Share-Holder

The function can only be invoked by the director, it is used to authorize a share holder to give them the right to cast their vote on the target question.

```
/*
* Give the right of voting to a ShareHolder.
* @_shareholder -> The address of the ShareHolder to whom voting right should be provided.
*/
function authorizeShareHolder(address _shareholder) directorOnly public {
    /* Set the authorized boolean flag for the shareholder as true */
    ShareHolders[_shareholder].authorized = true;
    addressLookupTable.push(_shareholder);
}
```

3.7.3 Unauthorize Share-Holder

```
/*
* Retract the right of voting from a ShareHolder.
* @_shareholder -> The address of the ShareHolder to whom voting right should be retracted.
*/
function unauthorizeShareHolder(address _shareholder) directorOnly public {
    /* Set the authorized boolean flag for the shareholder as false */
    ShareHolders[_shareholder].authorized = false;
}
```

The function can only be invoked by the director, it is used to revoke a shareholder's right to cast their vote on the question.

3.7.4 Vote

```
  /*
    * Function that allows the ShareHolders to cast their vote.
    * @_voteStatus -> A boolean value 0/1 that can be cast by the shareholders; 0 -> Disagree, 1
        -> Agree.
    */
    function vote(uint _voteStatus) public isVotingInProgress {

        /* Check if the ShareHolder casting the vote has already voted */
        require(!ShareHolders[msg.sender].voted);

        /* Check if the ShareHolder casting the vote is authorized to vote */
        require(ShareHolders[msg.sender].authorized);

        /* Check if the vote casted by the ShareHolder is a valid binary (0|1) */
        require(_voteStatus == 0 || _voteStatus == 1);

        /* Record the vote casted by the ShareHolder */
        /* Flip the voted flag of the ShareHolder to block them from casting a duplicate vote */
        ShareHolders[msg.sender].voted = true;

        /*
        * @totalVotes -> Total number of votes casted in the ballot.
        * @votesInFavour -> Total number of votes in favour of the Question.
        */
        totalVotes += 1;
        votesInFavour += _voteStatus;
    }
```

This function allows a shareholder to place a vote(either a 0 in case of disagreement or 1 in case of agreement) for the question created by the director. A shareholder who does not have the ability to vote or has already voted once is restricted from invoking this function successfully.

3.7.5 Voting Status

```
/*
 * View the status of voting activity on the Question.
 */
 function votingStatus() directorOnly public view returns (string memory) {
     /* Check if votes in favour of Question is greater than 50% of the total votes */
     if(votingInProgress) {
         if(votesInFavour > totalVotes/2)
             return "Voting is currently In Favour of the Question";
         else
             return "Voting is currently Not In Favour of the Question";
     }
     else
         return "No Active Questions being voted: Voting ended! use VotingResults";
 }
```

This function can only be invoked by the director. It allows the director to view the status of the voting process for the question at any time, during the voting process.

3.7.6 Voting Results

```
function VotingResults() public view returns(string memory) {
        /* Check if votes in favour of Question is greater than 50% of the total votes */
        if(!votingInProgress) {
            if(electionResult)
                return "Majority: Agree";
            else
                return "Majority: Disagree";
        } else return "Voting is still in progress: end the voting";
    }
```

This function allows all the shareholders to view the results of the voting once the voting process has been terminated.

3.7.7 End Voting

```
    /*
    * Function to stop the voting process. Any votes cast after the invocation of this function
        will be ignored.
    */
    function endVoting() directorOnly public {
        electionResult = votesInFavour > totalVotes/2;
        votingInProgress = false;
        Question = "--Create a new Question to begin Voting--";
        for (uint i=0; i< addressLookupTable.length ; i++) {
            ShareHolders[addressLookupTable[i]].voted = false;
        }
    }
```

This function can only be invoked by the director and allows the director to end the voting process.

3.7.8 Terminate

```
    /*
    * Function to deinitalize the director and selfdestruct the Contract application.
    */
    function terminate() directorOnly public {
        selfdestruct(msg.sender);
    }
```

This function can only be invoked by the director and is used to release the director's access from the deployed contract.

3.8 Demo

Kindly find the demonstration of the smart contract implementation on our github here → Demo

3.9 Smart Contract complete code

Git Hub link: CumulativeVoting.sol

```
pragma solidity >=0.4.22 <0.6.0;

contract CumulativeVoting {

    /*
    * A structure to hold ShareHolders vote status and his voting rights (authorized)
    * @authorized -> A boolean flag indicating if a ShareHolder is allowed to vote or Not
    * @voted -> A boolean flag indicating if the ShareHolder has already voted.
    * @vote -> An integer variable that indicated the vote placed by the ShareHolder this is
        either 0 or 1.
    */
    struct ShareHolder {
        bool authorized;
        bool voted;
    }

    // Stores the directors address.
    address private director;

    // Stores the Question's title or description.
    string public Question;

    bool private votingInProgress = false;
    bool private electionResult = false;

    // Creates a map object (addressOfShareHolder -> ShareHolder struct), to keep track of
        shareholders voting.
    mapping(address => ShareHolder) private ShareHolders;
    address[] private addressLookupTable;

    // Stores the total number for votes cast in the ballot.
    uint private totalVotes = 0;

    // Stores the total number of votes in favour of the Question proposed.
    uint private votesInFavour = 0;

    /*
    * Reusable assert functionality to restrict access to the resources to ShareHolders except
        the director.
    */
    modifier directorOnly() {
        // Assert/require before executing the body _;
        require(msg.sender == director);
        _;
    }

    modifier isVotingInProgress() {
        require(votingInProgress == true);
        _;
    }

    /* Override the constructor to initialize the director on startup */
    constructor () public {
        /* msg.sender -> Contains the address of the address that deployed the contract.
        * Save the address of the deployer as director
        */
        director = msg.sender;
    }

    /*
    * Create a new Question.
    * @_name -> Title/Description of the Question.
    */
    function createQuestion(string memory _name) directorOnly public {
       /* Set the Question's Title/Description */
        Question = _name;
        votingInProgress = true;
        votesInFavour = 0;
        totalVotes = 0;
    }

    /*
    * Give the right of voting to a ShareHolder.
    * @_shareholder -> The address of the ShareHolder to whom voting right should be provided.
    */
    function authorizeShareHolder(address _shareholder) directorOnly public {
        /* Set the authorized boolean flag for the shareholder as true */
        ShareHolders[_shareholder].authorized = true;
        addressLookupTable.push(_shareholder);
    }

    /*
    * Retract the right of voting from a ShareHolder.
    * @_shareholder -> The address of the ShareHolder to whom voting right should be retracted.
    */
    function unauthorizeShareHolder(address _shareholder) directorOnly public {
        /* Set the authorized boolean flag for the shareholder as false */
        ShareHolders[_shareholder].authorized = false;
    }

    /*
    * Close the voting activity on the Question.
    */
    function VotingResults() public view returns(string memory) {
        /* Check if votes in favour of Question is greater than 50% of the total votes */
        if(!votingInProgress) {
            if(electionResult)
                return "Majority: Agree";
            else
                return "Majority: Disagree";
        } else return "Voting is still in progress: end the voting";
    }

    /*
    * Function to stop the voting process. Any votes cast after the invocation of this function
        will be ignored.
    */
    function endVoting() directorOnly public {
        electionResult = votesInFavour > totalVotes/2;
        votingInProgress = false;
        Question = "--Create a new Question to begin Voting--";
        for (uint i=0; i< addressLookupTable.length ; i++) {
            ShareHolders[addressLookupTable[i]].voted = false;
        }
    }

    /*
    * View the status of voting activity on the Question.
    */
    function votingStatus() directorOnly public view returns (string memory) {
        /* Check if votes in favour of Question is greater than 50% of the total votes */
        if(votingInProgress) {
            if(votesInFavour > totalVotes/2)
                return "Voting is currently In Favour of the Question";
            else
                return "Voting is currently Not In Favour of the Question";
        }
        else
            return "No Active Questions being voted: Voting ended! use VotingResults";
    }

    /*
    * Function that allows the ShareHolders to cast their vote.
    * @_voteStatus -> A boolean value 0/1 that can be cast by the shareholders; 0 -> Disagree, 1
        -> Agree.
    */
    function vote(uint _voteStatus) public isVotingInProgress {

        /* Check if the ShareHolder casting the vote has already voted */
        require(!ShareHolders[msg.sender].voted);

        /* Check if the ShareHolder casting the vote is authorized to vote */
        require(ShareHolders[msg.sender].authorized);

        /* Check if the vote casted by the ShareHolder is a valid binary (0|1) */
        require(_voteStatus == 0 || _voteStatus == 1);

        /* Record the vote casted by the ShareHolder */

        /* Flip the voted flag of the ShareHolder to block them from casting a duplicate vote */
        ShareHolders[msg.sender].voted = true;

        /*
        * @totalVotes -> Total number of votes casted in the ballot.
        * @votesInFavour -> Total number of votes in favour of the Question.
        */
        totalVotes += 1;
        votesInFavour += _voteStatus;
    }

    /*
    * Function to deinitalize the director and selfdestruct the Contract application.
    */
    function terminate() directorOnly public {
        selfdestruct(msg.sender);
    }

}
```