Information Systems assignment 2

Koen Bolhuis (s3167895)
Germán Calcedo (s5381541)

December 5, 2022

The complete contract can be found in Appendix A, as well as on GitHub: `https://github.com/gcalcedo/is-labs/tree/master/assignment-2`

Task 1

To recognize the director, we can use the constructor to store the uploader's address in a private variable:

```
    address private director;

    constructor() {
        director = msg.sender;
    }
```

Then, as a clean way to abstract access control, we can define a `modifier` that ensures certain functions can only be invoked by the director:

```
    modifier onlyDirector() {
        require(msg.sender == director, "Only the director can perform this action.");
        _;
    }
```

Task 2

We represent questions using a `struct`. Questions themselves are stored in a `mapping` from ID (`uint`) to the corresponding question. We use a mapping as opposed to an array because adding elements involves resizing the array, which is inefficient. The question count is also tracked, and is used to assign each question a new ID.

```
    struct Question {
        string text;
        uint votesFor;
        uint votesAgainst;
        bool closed;
    }
    uint private questionCount = 0;
    mapping(uint => Question) private questions;
```

Adding questions is done using the `addQuestion` function, which uses the `onlyDirector` modifier we defined before:

```
    function addQuestion(string calldata text) public onlyDirector returns (uint id) {
        questions[questionCount] = Question(text, 0, 0, false);
        questionCount++;
        return questionCount - 1;
    }
```

Task 3

Shareholders are tracked in a (private) mapping from their `address` to a `boolean` flag, representing their shareholder status. In this way, we emulate a set (a data type Solidity unfortunately does not support natively).

```
    mapping(address => bool) private shareholders;
```

As with the director, we can define a `modifier` to ensure only shareholders can perform certain actions:

```
    modifier onlyShareholder() {
        require(shareholders[msg.sender], "Only shareholders can perform this
            action.");
        _;
    }
```

The director (and only the director) can manage shareholders using the following `addShareholder` and `removeShareholder` functions. Instead of assigning `false` to remove shareholders, we could also `delete` the element, but a soft delete allows us to keep track of previous shareholders.

```
    function addShareholder(address shareholder) public onlyDirector {
        shareholders[shareholder] = true;
    }

    function removeShareholder(address shareholder) public onlyDirector {
        shareholders[shareholder] = false;
    }
```

Task 4

To prevent each shareholder from voting more than once, we create a `mapping` that stores the addresses of all the shareholders that have already voted for a given question, identified by its ID.

```
    mapping(uint => mapping(address => bool)) private voters;
```

When a shareholder tries to cast vote, we simply require that its address is not stored in the mapping.

```
    function voteForQuestion(uint id, Vote vote) public onlyShareholder {
        require(!questions[id].closed, "This question is already closed.");
        require(!voters[id][msg.sender]);

        voters[id][msg.sender] = true;

        if (vote == Vote.FOR) {
            questions[id].votesFor++;
        } else {
            questions[id].votesAgainst++;
        }
    }
```

To make the voting logic more readable, we create an **enum** to represent both types of votes.

```
    enum Vote {
        FOR,
        AGAINST
    }
```

Task 5

The director can close any question through the following function, using that question's ID.

```
    function closeQuestion(uint id) public onlyDirector {
        questions[id].closed = true;
    }
```

Since both the director and shareholders would like to view questions, we first define another modifier:

```
    modifier directorOrShareholder() {
        require(
            msg.sender == director || shareholders[msg.sender],
            "Only the director or shareholders can perform this action."
        );
        _;
    }
```

We then provide a function to display the information for each question. The vote count can be seen for every question, regardless of its state. However, only once a question has been closed the final decision is computed and shown.

```
    function getQuestion(uint id) public view directorOrShareholder returns (
        string memory) {
        require(id < questionCount, "ID is invalid.");

        return string.concat(
            Strings.toString(id),
            " - ",
            questions[id].text,
            questions[id].closed ? " (Closed)" : " (Open)",
            questions[id].closed
                ? (questions[id].votesFor > questions[id].votesAgainst ? ":
                    Approved" : ": Declined")
                : "",
            " - Votes For: ", Strings.toString(questions[id].votesFor),
            " | Against: ", Strings.toString(questions[id].votesAgainst)
        );
    }
```

As a side note, we use the `"@openzeppelin/contracts/utils/Strings.sol"` contract to convert vote counts to strings, as Solidity does not natively support integer-to-string conversions. This could be left out, but we argue it improves the output.

For completion purposes, we also implement a function to retrieve the information for every question, without needing to input its ID. For this, we use the `console.log` function from the `"hardhat/console.sol"` contract.

```
    function showQuestions() public view directorOrShareholder {
        for (uint i = 0; i < questionCount; i++) {
            console.log(getQuestion(i));
        }
    }
```

A Complete smart contract

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.8.12 <0.9.0;

import "hardhat/console.sol";
import "@openzeppelin/contracts/utils/Strings.sol";

contract ShareholderVoting {
    address private director;

    mapping(address => bool) private shareholders;
    mapping(uint => mapping(address => bool)) private voters;

    struct Question {
        string text;
        uint votesFor;
        uint votesAgainst;
        bool closed;
    }
    uint private questionCount = 0;
    mapping(uint => Question) private questions;

    enum Vote {
        FOR,
        AGAINST
    }

    modifier onlyDirector() {
        require(msg.sender == director, "Only the director can perform this action
            .");
        _;
    }

    modifier onlyShareholder() {
        require(shareholders[msg.sender], "Only shareholders can perform this
            action.");
        _;
    }

    modifier directorOrShareholder() {
        require(
            msg.sender == director || shareholders[msg.sender],
            "Only the director or shareholders can perform this action."
        );
        _;
    }

    constructor() {
        director = msg.sender;
    }

    function addShareholder(address shareholder) public onlyDirector {
        shareholders[shareholder] = true;
    }

    function removeShareholder(address shareholder) public onlyDirector {
        shareholders[shareholder] = false;
    }

```

```
58      function addQuestion(string calldata text) public onlyDirector returns (uint
            id) {
59          questions[questionCount] = Question(text, 0, 0, false);
60          questionCount++;
61          return questionCount - 1;
62      }
63
64      function closeQuestion(uint id) public onlyDirector {
65          questions[id].closed = true;
66      }
67
68      function voteForQuestion(uint id, Vote vote) public onlyShareholder {
69          require(!questions[id].closed, "This question is already closed.");
70          require(!voters[id][msg.sender]);
71
72          voters[id][msg.sender] = true;
73
74          if (vote == Vote.FOR) {
75              questions[id].votesFor++;
76          } else {
77              questions[id].votesAgainst++;
78          }
79      }
80
81      function getQuestion(uint id) public view directorOrShareholder returns (
            string memory) {
82          require(id < questionCount, "ID is invalid.");
83
84          return string.concat(
85              Strings.toString(id),
86              " - ",
87              questions[id].text,
88              questions[id].closed ? " (Closed)" : " (Open)",
89              questions[id].closed
90                  ? (questions[id].votesFor > questions[id].votesAgainst ? ":
                        Approved" : ": Declined")
91                  : "",
92              " - Votes For: ", Strings.toString(questions[id].votesFor),
93              " | Against: ", Strings.toString(questions[id].votesAgainst)
94          );
95      }
96
97      function showQuestions() public view directorOrShareholder {
98          for (uint i = 0; i < questionCount; i++) {
99              console.log(getQuestion(i));
100         }
101     }
102 }
```