CMPE 483
Blockchain Programming

Homework-1 Report

Team Members:

Batuhan Çelik 201840051

Onur Kömürcü 2018400147

Ezgi Aysel Batı 2018400207

Task Achievement Table	Yes	Partially	No
I have prepared documentation with at least 6 pages.	X		
I have provided average gas usages for the interface functions.		X	
I have provided comments in my code.	X		
I have developed test scripts, performed tests and submitted test scripts as well ented test results.	X		
I have developed smart contract Solidity code and submitted it.	X		
Function delegateVoteTo is implemented and works.	X		
Function donateEther is implemented and works.	X		
Function donateMyGovToken is implemented and works.	X		
Function voteForProjectProposal is implemented and works.	X		
Function voteForProjectPayment is implemented and works.	X		
Function submitProjectProposal is implemented and works.	X		
Function submitSurvey is implemented and works.	X		
Function takeSurvey is implemented and works.	X		
Function reserveProjectGrant is implemented and works.	X		
Function withdrawProjectPayment is implemented and works.	X		
Function getSurveyResults is implemented and works.	X		
Function getSurveyInfo is implemented and works.	X		
Function getSurveyOwner is implemented and works.	X		
Function getIsProjectFunded is implemented and works.	X		
Function getProjectNextPayment is implemented and works.	X		
Function getProjectOwner is implemented and works.	X		
Function getProjectInfo is implemented and works.	X		
Function getNoOfProjectProposals is implemented and works.	X		

Task Achievement Table	Yes	Partially	No
Function getNoOfFundedProjects is implemented and works.	X		
Function getEtherReceivedByProject is implemented and works.	X		
Function getNoOfSurveys is implemented and works.	X		
I have tested my smart contract with 100 addresses and documented the results e tests.		X	
I have tested my smart contract with 200 addresses and documented the results e tests.		X	
I have tested my smart contract with 300 addresses and documented the results e tests.		X	
I have tested my smart contract with more than 300 addresses and documented ults of these tests.		X	

IMPLEMENTATION

We have implemented this smart contract in two parts. The first contract mygovtoken, is an implementation of an ERC20 token which is the basis of our governance token. This includes everything regarding the governance token itself, such as the total balances and delegation.

The address owner represents the address of the contract itself, which is useful for operations like donating, creating surveys/proposals and receiving from the faucet. The owner acts like a user account, and receives balance when an operation with a cost is done, and transfers only when distributing tokens from the faucet.

The second contract, myGov, implements the governance operations that are performed with the mygov token. This includes all operations for creating proposals and surveys, as well as participating in votings and surveys and getting results or information on the current status of projects/surveys.

FUNCTIONS

function submitProjectProposal:

This function is implemented in myGov. It allows members to create a new proposal along with its payment schedule and amounts. This function is implemented with the use of defined projectProposal struct and proposals(`mapping(uint256 => projectProposal)`) map which maps proposal id's to proposal struct objects. When this function is called, it assigns the relevant fields of struct in mapping and also makes the necessary preparations for unreserved balance of contract. It also increments projectProposal id assigning counter and makes the necessary transfers to take the fee from creating member to the "owner" address of the contract. The correctness of this function was tested with a unit test(see relevant section).

function submitSurvey:

This function is also implemented in myGov and works similar to submitProjectProposal. It has its own relevant struct survey and mapping surveys (`mapping(uint256 => survey)`) and separate counter, but the implementation is parallel and the correctness was tested with a unit test.

function voteForProjectProposal:

This function takes project id and choice as an argument. It requires that voter should be member which means that voter should have at least 1 token. If that user didn't vote for proposal before, it will be added to voters and his/her choice will be saved. Otherwise, his/her choice will be changed according to last choice.

function voteForProjectPayment:

This function takes project id and choice as an argument. It requires that voter should be member which means that voter should have at least 1 token. If that user didn't vote for project payment before, it will be added to voters and his/her choice will be saved. Otherwise, his/her choice will be changed according to last choice.

function clearProjectVotes:

This function resets the vote contents for a project in between different schedule deadlines. It is necessary since after each received payment the requirement for 1% approval still stands and funds can be rejected at any schedule deadline.

function reserveProjectGrant:

It requires project id as an argument. It checks there is such project, fund didn't reserved before, deadline isn't passed, proposer is calling this function and myGov currently have enough financial power to support this project.

By these requirements, function ensures that proposer can reserves the project grant for the specific milestone of the project. nonReservedBalance is decreased by the amount of the reserved grant. Number of funded projects is increased and this project is now funded.

function withdrawProjectPayment:

It requires project id as an argument. It checks there is such project, project payment is reserved before the withdraw request, currentPhase of the project payment schedule deadline isn't passed and proposer is calling this function.

After these requirements, in the function users vote for withdraw project payment. If the total "yes" power among total faucets exceeds 1/100, proposer can withdraw the payment. And the current phase will increased by one. Otherwise, payment is not approved and transaction, withdraw is not occurred. However, current phase is also increased by one.

function getIsProjectPassed:

It requires project id as an argument. It checks there is such project. This function returns boolean. If project is already funded, it means that this project is passed. Otherwise, it looks the total true votes among total faucets. If the weight of the true votes exceed 1/10, this function returns true. If not, returns false.

function delegateVoteTo:

Vote delegation is implemented like the ERC20 token where each user holds a voting power balance and delegation is equivalent to transferring that token to some other user. Unlike ERC20, we are keeping track of these vote movements so that a voter can cancel his/her delegation or change his/her delegatee. In addition, our implementation can track delegation chains. For illustration, if I delegated my vote to user X and user X delegated her vote to user Y, the system automatically serves my vote to user Y and X can cancel this operation and claim my vote, whereas I can revoke X's delegation and get back my vote.

function getVotes:

This function returns the number of votes controlled by the given address.

function takeSurvey:

This function allows the caller to add their choices to a given survey. It has many checks to ensure correct functionality. It requires that survey deadline has not yet been reached, the caller is a member, submitted choices are at most as much as maximum allowed choices, the caller has not taken the survey before, and each of the submitted choices are valid options. If all these requirements are ensured, the function increments the place representing the choice for each choice submitted. The correctness of this function was tested with a unit test(see relevant section).

function getSurveyResults:

This is a simple getter which returns the results and total participants of a survey. The correctness of this function was tested with a unit test(see relevant section)

function getSurveyInfo:

This is a simple getter which returns the ipfshash, survey deadline, number of choices and maximum allowed choice count for a given survey. The correctness of this function was tested with a unit test(see relevant section).

function getSurveyOwner:

This is a simple getter which returns the address of the user who created a given survey. The correctness of this function was tested with a unit test(see relevant section)

function getIsProjectFunded:

This is a simple getter which returns whether a given project is funded or not. Since in order to be funded a projects' payment needs to be reserved, the checks for vote requirements are performed under that function and this only returns true for projects with payments reserved.

function getProjectNextPayment:

This function returns the amount requested for the next upcoming payment for a given project. It requires that the project exists and that is has been funded. It makes use of the mappings created in the struct which are assigned as the voting/reserving operations take place.

function getProjectOwner:

This is a simple getter which returns the address of the user who created a given project. The correctness of this function was tested with a unit test(see relevant section)

function getProjectInfo:

This is a simple getter which returns the ipfshash, project deadline, payment amounts and payment schedule for a given project. The correctness of this function was tested with a unit test(see relevant section).

function getNoOfProjectProposals:

This function returns the total number of project proposals created. This is implemented simply by returning the projectCounter used to assign id's during proposal creation. The correctness of this function was tested with a unit test(see relevant section).

function getNoOfFundedProjects:

This function returns the total number of project proposals which have been funded . This is implemented simply by returning the fundedProject counter used to determine which projects are funded during fund reserve/retrieve operations.

function getEtherReceivedByProject:

This function returns the total amount of payment received by a project. It simply sums all retrieved payments, or returns 0 for non funded projects.

function getNoOfSurveys:

This function returns the total number of surveys created. This is implemented simply by returning the surveyCounter used to assign id's during survey creation. The correctness of this function was tested with a unit test(see relevant section).

function getVoteContent:

This function returns whether a member approved a given project's creation or its current status in payment schedule. If no vote was given, it returns the default assumption which is reject. It is implemented simply by retrieving the relevant field in the mapping in projectProposal struct.

GAS USAGE

We did not perform calculations for average gas usages but we tried functions including any loops with their worst case scenarios. There are 2 duty heavy loops in our implementation: first one is the getIsProjectPlanPassed, this function iterates over all the voters voting choices, gets their voting power, this step was essential to prevent duplicate votings, and sums the voting powers of those who said yes. We tried this function with the 10.000.000 users and only 600.000 gas is used which is 20% of the regular gas limit.

The second duty heavy loop is pointing delegation which in theory may need to iterate over all users if there is a delegation chain containing all users. This is not likely but as we tried out and observed in the last test, iterating over all users requires 600.000 gas which is not ideal nor infeasible.

Thus, we are sure our implementation works in the gas limits set by the ethereum blockchain.

UNIT TESTS

We have tested varios functions we implemented. To do so, we wrote 16 unit tests using chai framework. The tests can be found in our code submission in the test file under the title *mygovtest.js .*

We run these tests on remix online IDE by creating 30 user accounts. Due to IDE's memory limitations we chose to write more tests but use less accounts to run these tests with.

The tests include console logs to track progress in test and have titles explaining the functionality they test. The results of these tests can be seen in the screenshots provided below:

```
35 users created!
   ✓ Distributes faucets (4043 ms)
mygov deployed at:0xd9145CCE52D386f254917e481eB44e9943F39138
faucets distributed
   ✓ Transfers tokens (10790 ms)
creating survey
   ✓ Creates and gets surveys (551 ms)
creating proposal
   ✓ Creates and gets project proposals (1064 ms)
Test description: User 0 donates .1 ether
   ✓ Donates ether (915 ms)
Test description: User 25 and 26 transfer 1 token to User 27.
   User 0 creates Survey 1.
 User 28-30 takes survey and prompt the results
creating survey...
   ✓ Creates survey, takes survey and gets survey results (1060 ms)
found initial balance...
donating...
   ✓ donates MyGov tokens (283 ms)
vote for...
vote against...
   ✓ votes for project payment (227 ms)
   ✓ retrieves survey info (54 ms)
is errror
   ✓ retrieves project owner (62 ms)
creating proposals...
   ✓ retrieves total proposal count (2691 ms)
retrieves initial
   ✓ retrieves Ether Received By Project (36 ms)
   ✓ checks if project is funded (42 ms)
Test description: Getting owner of the survey 1
trying to get owner of a non-existed survey...
getting owner of the survey 0...
   ✓ Gets survey owner (1006 ms)
```

```
Test description: Getting information of the project 0
trying to get information of a non-existed project...
Expected error is occurred!
   ✓ Gets project info (340 ms)
Test description: Getting number of the funded projects
   ✓ Gets number of funded projects (35 ms)
Passed: 16
Failed: 0
Time Taken: 23845 ms
```