CMPE 483 PROJECT I - DOCUMENTATION

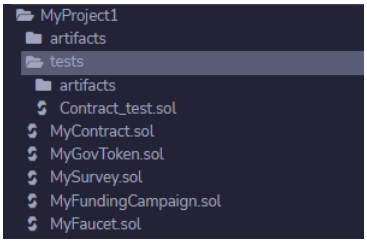
Tags

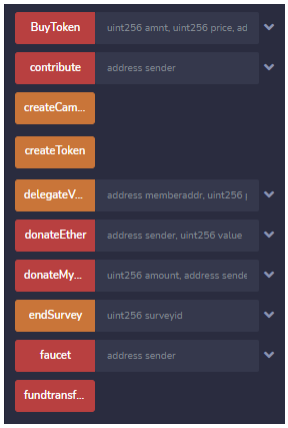
First I want to explain my situation, the reason why I had to do this project alone as a student don't have the full knowledge of doing this project. In the time that people were looking for team members, I submitted my request to withdraw this course but unfortunately it's rejected due to a problem about the courses that I've taken during Erasmus, apparently I'm not a senior student. So, sorry if I'm submitting a project with a bad documentation.

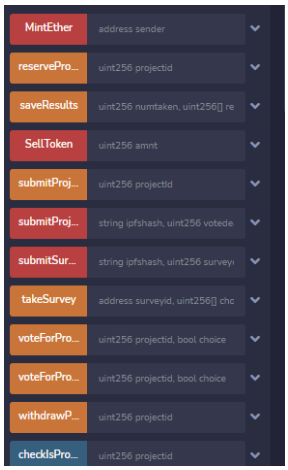
My project consists of 4 contracts. The main one is *MyContract.sol* where we control our interface functions for the owner of the project and the members. *MyGovToken.sol* is the contract where we do the transactions, take the records and all standard ERC20 functions are provided by this. *MySurvey.sol* is the contract for survey service. *MyFundingCampaign.sol* is a contract where we can manage the funding for the chosen project by the members, withdrawal, reversing money and etc. are provided by this.

Contract_test.sol is the unit testing contract. *MyFaucet.sol* is the contract where you can donate ether and get ether for the accounts and it was used for testing.

I wrote and tested my project on Remix:







Functions to be implemented:

1.constructor

When we deploy our main contract, we also deploy the Token and Funding contracts.

```
//9. MyGov token supply is 10 million (fixed).
    constructor() public payable{
        createToken();
        createCampaign();
    }
```

2.delegateVoteTo

For the deployed survey there would be different project proposals to be voted and all of the members have the right to vote (system gives them automatically). Here people can give their vote right to other people.

```
//12.Members can delegate vote.
    // Members who voted or delegated vote cannot reduce their MyGov balance to zero until the voting deadlines.
    function delegateVoteTo(address memberaddr,uint projectid) public onlyMembers(msg.sender){
    MySurvey.delegateVoteTo(memberaddr,projectid,msg.sender);
    }
```

MyMainContract.sol

We encounter different cases here for example what if the person who is delagated already voted for a proposal, here is the code with a comments for such cases.

```
function delegateVoteTo(address to, uint projectId, address senderAddr) public returns (bool success){//CODE FOR EACH PROPOSAL
//
// REST OF THE CODE IS OMITED
//
        Voter storage delegate_ = votersAdress[to][projectid];
        if (delegate_.voted) {
            // If the delegate already voted "yes,
            // directly add to the number of votes
            if(delegate_.vote){
            proposals[projectid].voteCount += sender.weight;}
        } else {
            // If the delegate did not vote yet,
            // add to her weight.
            delegate_.weight += sender.weight;
        }

        return true;
    }
```

MySurvey.sol

3.faucet

11.MyGov tokens are distributed via a faucet function. Faucet gives 1 token to an address. If the address obtained a token before, it cannot get token from faucet any longer. If someone has a token, automatically a member and has right to vote.

```
    function faucet(address sender) public payable returns(bool success){
        MyGov.giveToken(sender);
        //3. Anyone who owns at least 1 MyGov token is MyGov member.
        isMember[sender]=true;
        members.push(sender);
        membersCount++;
        if(isSurveyDeployed){
            MySurvey.giveRightToVote(sender);//gives right to vote for existing survey
        }
        return true;
        }
```

MyMainContract.sol

Anyone can get a token once. This is provided in the *MyGov.sol* contract.

```
//allow users to call the requestTokens function to mint tokens
    function giveToken (address requestor) public returns (bool success) {
        //perform a few check to make sure function can execute
        require(!minted[requestor], "lock time has not expired. Please try again later");
        minted[requestor]=true;
        //mint tokens
        balances[mainSupplyAddress] = safeSub(balances[mainSupplyAddress], 1);
        balances[requestor] = safeAdd(balances[requestor], 1);
        emit Transfer(mainSupplyAddress, requestor, 1);
        return true;
        //no locktime! but only once
        emit Mint(requestor);
        return true;
    }
```

MyGov.sol

4.donate

Donations are kept in the main contract to be used later for the funding campaigns. Note that anyone can buy or sell tokens in exchange of ethers, it’s also implemented.

```
    function donateEther(address sender, uint value) public payable{
        MyGov.donateEther(value, sender);
        contribute(sender);
    }

    //10. Donations can be accepted in ethers and MyGov tokens only. Ethers can be granted to winning Project proposals.
    function donateMyGovToken(uint amount, address sender) public payable{
        MyGov.donateMyGovToken(amount,sender);
    }
```

MyMainContract.sol

5.vote

First, people vote for the deployed survey and for the winning proposals there will be another voting period for its funding.

```
//3. Anyone who owns at least 1 MyGov token is MyGov member.
//inputs are which project you are voting and what answer you give (yes/no)
    function voteForProjectProposal(uint projectid,bool choice) public onlyMembers(msg.sender){
        MySurvey.voteTo(projectid,choice,msg.sender);
    }

    //for the winning projects? People vote the projects to get it paid, among the funded projects.
    function voteForProjectPayment(uint projectid,bool choice) public onlyMembers(msg.sender){
        require(projects[projectid].funded,"Project is not funded!");
        MyFundingCampaign.approvalWithdrawal(projectid, choice); //number of approvals for the project needs to reach a number to be approved for the payment
   }
```

MyMainContract.sol

Each member has a right to vote for each proposals (yes/no). These rights are recorded as an array that is mapped from an address. Moreover, each proposal has a id in the MySurvey contract corresponding to its unique project Id.

```
    mapping(address => Voter[]) public votersAdress; //Voter struct corresponding to each temporary id to keep record the vote right for each projects

    mapping(uint => Proposal) public proposals; //submitted project to this survey

    mapping(uint => uint) public convertId; //unique project Id-> temporary countId for the deployed survey

    uint public countId;//temporary number of submitted proposals
```

MySurvey.sol

6.submitProjectProposal & submitSurvey

Here, we submit a project with all of its details to a survey and record as a struct in the main contract to be used later.

```
//7. Submitting Project Proposal costs 5 MyGov tokens and 0.1 Ether.Payable
    function submitProjectProposal(string ipfshash, uint votedeadline,uint [] paymentamounts, uint [] payschedule, address sender)
public payable onlyMembers(sender) returns(uint projectid) {
        require(getBalanceOf(msg.sender)>=5, "Proposal costs 5 MyGov Tokens");
        require(msg.value>=0.1 ether,"Proposal costs 0.1 Ether");//0.1 ether = 100000000000000000 wei
        donateEther();
        donateMyGovToken(5);

        projects[Id]=(Project({
        ipfsHash: ipfshash,
        voteDeadline: votedeadline,//deadline for the survey
        paymentAmounts: paymentamounts,//how much ether is needed
        paySchedule: payschedule,//payments are scheduled
        recipient: sender,//project proposer
        voteCount: 0,
        approvalCount: 0, //payment approvals count
        approved:false,
        funded:false
       // approvals:  //addresses who approve the
            }));

        MySurvey.submitProposal(ipfshash, votedeadline/*, paymentamounts, payschedule*/,Id);
        Id++;
        //Id=MySurvey.getCountId();
        return Id;
    }
```

MyMainContract.sol

```
    //8. Submittting Survey costs 2 MyGov tokens and 0.04 Ether.Payable
        function submitSurvey(string ipfshash,uint surveydeadline,uint numchoices, uint atmostchoice, address sender)
    public payable onlyMembers(sender) returns (address surveyid){
            require(getBalanceOf(sender)>=2, "Proposal costs 2 MyGov Tokens");
            require(msg.value>=0.04 ether,"Proposal costs 0.04 Ether");//0.04 ether = 40000000000000000 wei
            donateEther(sender,msg.value);
            donateMyGovToken(5, sender);
            isSurveyDeployed=true;
            MySurvey= new Survey(ipfshash,surveydeadline, numchoices, atmostchoice,sender);
            emit SurveyAdresses(MySurvey);
            for(uint n=0; n<membersCount;n++){
            MySurvey.giveRightToVote(members[n]);}

            surveysCount++;
            return address(MySurvey);
        }
```

MyMainContract.sol

View functions to get the details of the deployed survey by its id.

```
    function getSurveyResults(uint surveyid) public view returns(uint numtaken, uint [] results)
    function getSurveyInfo(uint surveyid) public view returns(string ipfshash, uint surveydeadline,uint numchoices, uint atmostchoice)
    function getSurveyOwner(uint surveyid) public view returns(address surveyowner)
```

7.reserveProjectGrant & withdrawProjectPayment

```
    //13.Project proposer must call reserveProjectGrant function in order to reserve the funding by the proposal deadline.
        //If the project proposer does not reserve by the deadline, funding is lost.
        //Also, if there is not sufficient ether in MyGov contract when trying to reserve, funding is lost.
        function reserveProjectGrant(uint projectid) public{
            MyFundingCampaign.finalizeWithdrawal(projectid, membersCount);
        }

        //cancels the subscription for the funding a project by its id
        function withdrawProjectPayment(uint projectid) public{
            MyFundingCampaign.cancel(projectid);
        }
```

MyMainContract.sol

Project proposer must call reserveProjectGrant function in order to reserve the funding by the proposal deadline. If the project proposer does not reserve by the deadline, funding is lost after the deadline.

```
    function finalizeWithdrawal(uint index, uint membersCount) public {
          Withdrawal storage withdrawal =withdrawals[index];

          require(withdrawal.approvalCount >= (membersCount/100));//for its funding, it's need to be approved by 1/100 members
          require(!withdrawal.complete);//checks if it's ended

          Subscription storage subscription = subscriptions[index];
          require(subscription.valid != false, 'this subscription does not exist'); //checks the subscription, if it's still kept funded
          require(block.timestamp > subscription.nextPayment,'not due yet'); //checks the deadline, funding is given when the time has come

         //token.transferFrom(subscriber, plan.merchant, plan.amount);
          withdrawal.recipient.transfer(withdrawal.paymentamounts[subscription.currentState]);
          //withdrawal.complete = true;

          subscription.nextPayment = subscription.nextPayment + withdrawal.paymentamounts[subscription.currentState];
          subscription.currentState++;
        }
```

MyFundingCampaign.sol

```
    function cancel(uint Id) public {
        Subscription storage subscription = subscriptions[Id];
        require(
          subscription.valid != false,
          'this subscription does not exist'
        );
        delete subscriptions[Id];
        emit SubscriptionCancelled(msg.sender, Id, block.timestamp);
      }
```

MyFundingCampaign.sol

8.endSurvey

I tried to write a function that executes the function endSurvey when the deadline has come but instead owner needs to do it manually.

Here, when the survey is ended, function gets the results and records them in the main contract.

```
function endSurvey(uint surveyid) public returns(bool){ //End the survey manually and get the results (before the deadline)
        Survey(surveyid).sortByVotes();//sorts the votes to get most voted proposals at first
        (uint numtaken,uint[] memory resultsId,uint[] memory results)=getSurveyResults(surveyid);
        saveResults(numtaken,resultsId,results);//records the results in this contract

        for(uint i=0; i<Survey(surveyid).getSurveyAtmostchoice(); i++){
        if(checkIsProjectFunded(resultsId[i])==true){//check if it's funded
        Project storage proj = projects[resultsId[i]];
        fundedProjectCount++;
        proj.funded=true;
        submitProjectCampaign(resultsId[i]); }
        }
        fundtransfer();//sends all of the donated money to the MyFundingCampaign contract to be used for the funded projects
    return true;
    }
```

MyMainContract.sol

9. getIsProjectFunded

//"At least 1/10 of the members must vote yes!" & "There should be sufficient ether amount in the MyGov contract!"

```
    //13.In order to get a Project proposal funded, at least 1/10 of the members must vote yes AND there should be sufficient ether amount in the MyGov contract.
  function checkIsProjectFunded(uint projectid) public view returns(bool funded){
        if(projects[projectid].voteCount >= membersCount/10 && projects[projectid].paymentAmounts[0] <= address(this).balance){
        return true;}
        else{
            return false;
        }
    }
```

MyMainContract.sol

10.Rest of the codes

Functions to get the details of the projects

```
function getProjectNextPayment(uint projectid) public view returns(int next)
function getProjectOwner(uint projectid) public view returns(address projectowner)
function getProjectInfo(uint projectid) public view returns(string ipfshash,
 uint votedeadline,uint [] paymentamounts, uint [] payschedule)
```

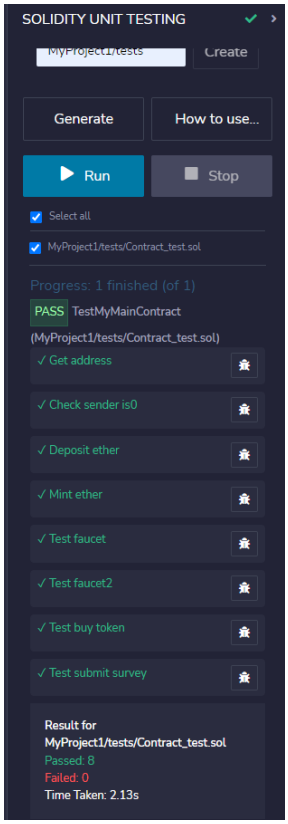
Also the other informations which are obtained throught the proccess.

```
function getNoOfProjectProposals() public view returns(uint numproposals)
function getNoOfFundedProjects () public view returns(uint numfunded)
//it should return total amount of money withdrawn.
function getEtherReceivedByProject (uint projectid) public view returns(uint tamount? amount)
function getNoOfSurveys() public view returns(uint numsurveys)
```

problems that I had in testing:

In solidity remix, how to use {from: member1} to change msg.sender? (can be applied in new versions)
How to determine msg.value and msg.sender from a another contract? (It's not possible, it's possible only in testing, truffle, python)
While testing, how to test if a function is supposed to be reverted? (solved)
solidity how to create accounts with ether (using test network)
Ethereum - Interacting with Deployed Contract
Having difficulties with testing on an already deployed contract.
I couldn't figure out how to use ipsfhash and store the survey result or project proposals

I couldn’t find time to write test scripts for all functions.


SOLIDITY UNIT TESTING
Create
Generate
How to use...
Run
Stop
Select all
MyProject1/tests/Contract_test.sol
Progress: 1 finished (of 1)
PASS TestMyMainContract
(MyProject1/tests/Contract_test.sol)
✓ Get address
✓ Check sender is0
✓ Deposit ether
✓ Mint ether
✓ Test faucet
✓ Test faucet2
✓ Test buy token
✓ Test submit survey
Result for
MyProject1/tests/Contract_test.sol
Passed: 8
Failed: 0
Time Taken: 2.13s