CMPE 483 – Blockchain Programming
Homework 1 Report

Hüseyin Karataş - 2020400021
Furkan Şenkal - 2020400249
Üveys Aydemir - 2020400069

1 Introduction

This report presents the implementation and documentation of a decentralized autonomous organization (DAO) governance system, **MyGov**, using Solidity.

We built MyGov on top of the OpenZeppelin ERC20 standard and extended it with additional interface functions to enable community governance, surveys, project proposals, funding mechanisms, and a token faucet system. We developed the smart contract using Hardhat and ethers.js, and wrote our test cases using Chai assertions.

2. System Architecture

2.1 Smart Contracts

The system is composed of three smart contracts that collaboratively enable governance, token transfer, and project funding functionalities:

- **MyGov.sol:** This is the core smart contract of the system. It:
  - Implements a governance token (`MyGov`) compliant with the ERC20 standard.
  - Manages all DAO-related operations including:
    - Survey creation and participation.
    - Project proposal submission and voting.
    - Funding disbursement and tracking.
    - Membership registration and verification.
  - Enforces access control by allowing only verified members to perform governance actions.
  - Includes a custom logic (we overrided the default transfer function inherited from ERC20) to restrict token transfers if they invalidate a member's participation in active proposals.

- **TLToken.sol:** A simplified ERC20-compatible token contract that represents a stable currency pegged to the Turkish Lira (TL). It is used for:
  - Submitting surveys and proposals.
  - Making donations.
  - Paying out project funding upon approval.

2.2 Membership

- A user becomes a **member** by obtaining at least **1 MGOV token** either through:
  - The `faucet()` function (available only once per address), or
  - Direct token transfer from another user.
- To robustly track and enforce membership, the `transfer()` function from ERC20 is **overridden**. This:
  - Detects when a user receives tokens for the first time (thus becoming a member).
  - Checks if the user's token balance drops below 1 MGOV due to a transfer or donation, and if so, revokes their member status.
- **Member privileges include:**
  - Submitting and voting on project proposals.
  - Participating in surveys.

2 Implemented Features

The following interface functions were implemented in addition to standard ERC20:

Function	**Purpose**
delegateVoteTo(address, uint)	Delegate voting power to another member
donateTLToken(uint)	Donate TLToken to the DAO
donateMyGovToken(uint)	Donate MyGov tokens to DAO
voteForProjectProposal(uint, bool)	Vote for a submitted project
voteForProjectPayment(uint, bool)	Vote for TL payment phase
submitProjectProposal(string, uint, uint[], uint[])	Submit new project proposal with TL/MyGov payment
submitSurvey(string, uint, uint, uint)	Create a new survey
takeSurvey(uint, uint[])	Submit answers to a survey
reserveProjectGrant(uint)	Reserve grant for approved proposal

withdrawProjectTLPayment(uint)	Withdraw TL payments for funded projects
getSurveyResults(uint)	Return results of completed survey
getSurveyInfo(uint)	Fetch survey details
getSurveyOwner(uint)	Return creator of the survey
getIsProjectFunded(uint)	Check if project is funded
getProjectNextTLPayment(uint)	Show next scheduled payment amount
getProjectOwner(uint)	Return project owner
getProjectInfo(uint)	Return full project metadata
getNoOfProjectProposals()	Total submitted project proposals
getNoOfFundedProjects()	Return number of funded proposals
getTLReceivedByProject(uint)	Show total TL received
getNoOfSurveys()	Return total surveys created

4. Implementation Details

4.1 Token Distribution and Faucet

- The `MyGov` contract has a fixed total supply of **10,000,000 MGOV tokens**.
- A faucet mechanism is provided to onboard new users:
  - Each unique address can call `faucet()` exactly once.
  - Upon successful call, the user receives **1 MGOV token**.
  - This also registers the user as a **contract member**.

4.2 Proposals and Voting

- To submit a project proposal, the user must:
  - Hold at least **5 MGOV tokens**.
  - Approve and transfer **4000 TLToken** to the MyGov contract.
- Voting on proposals is restricted to users holding at least **1 MGOV token** and who are recognized as members.
- A proposal is considered **approved** if:
  - At least **1/10 of the total members** have voted "yes".
- Vote delegation is supported, allowing users to transfer their vote right to another member.

4.3 Survey Mechanism

- Creating a new survey requires:
  - **2 MGOV tokens** from the sender's balance.
  - Approval and transfer of **1000 TLToken**.
- Each survey includes:
  - A fixed deadline.
  - A number of answer choices (`numchoices`).
  - A limit on how many choices a user can select (`atmostchoices`).
- Survey results, including vote counts per choice and total participants, can be accessed via public view functions.

4.4 Payment Scheduling

- Approved project proposals define:
  - A sequence of **payment amounts**.
  - A corresponding **payment schedule** (time intervals).
- Once approved and reserved, the proposer can withdraw funds incrementally:
  - Withdrawals are only allowed **after the scheduled time** has passed.
  - Each scheduled payment must be re-approved via voting, where at least **1/100 of members vote "yes"**.
  - Failure to meet quorum prevents the current and future payments.

Helper Functions

willViolateProposal The `willViolateProposal` function ensures voting integrity by checking whether a user has voted or delegated a vote for any ongoing project proposal (i.e., one where the vote deadline has not passed). If such participation exists, and the transfer amount would reduce the user's MGOV balance below the minimum required threshold (1 MGOV), the function returns `true` to block the transfer. This prevents vote manipulation through post-vote token transfers.

isContractMember The `isContractMember` function is a simple public view function that returns whether a given address is currently recognized as a member of the MyGov contract. A member is defined as an account holding at least 1 MGOV token and is marked as such in the `isMember` mapping. This function is used as a prerequisite check in multiple parts of the system, such as submitting surveys, proposals, or casting votes.

testFund The `testFund` function is a utility added for testing purposes. It allows the contract owner to transfer MGOV tokens from the contract's own balance to a specified address without triggering the usual membership checks and constraints. This function is used in the test environment to quickly assign tokens to users and validate various system behaviors under controlled conditions.

3 Test

Test Suite

The test suite for `MyGov.sol` comprehensively evaluates the smart contract's functionalities across membership, surveys, project proposals, voting, and payments. Tests are implemented using Hardhat with the Chai assertion library.

Token Distribution and Faucet

- 300 unique addresses successfully claimed 1 MGOV token each via the `faucet`.
- Second claim attempt by the same address correctly failed.

Donations

- Rejected MGOV and TL donations from accounts with insufficient balances.
- Allowed valid donations and confirmed contract balances accordingly.

Survey System

- Rejected survey creation due to insufficient MGOV tokens or non-member status.
- Allowed survey creation with valid MGOV and TL token allowances.
- Permitted valid survey participation; updated results verified.
- Verified survey detail retrieval (URL, deadline, choice count, owner).

Project Proposals and Voting

- Prevented proposal submission when MGOV or TL allowance was insufficient.
- Allowed proposals meeting token requirements and deadline criteria.
- Confirmed correct registration of project metadata and proposer identity.

Voting and Payments

- Verified voting on project proposals, including delegated votes.
- Prevented TL withdrawal before scheduled time or insufficient votes.
- Approved withdrawals only after meeting voting quorum and time lock.
- Simulated and verified vote delegation and its usage.

Metrics and Validation

- Successfully retrieved total survey count, proposal count, funded projects.
- Tested for out-of-bound and invalid project/survey IDs, ensuring expected reverts.
- Verified clean initial state behavior in a fresh deployment.

MyGov Full Test

```
MyGov Full Test
   should allow 300 unique addresses to claim MGOV from faucet (1234ms)
   should reject second faucet claim
   should reject MGOV and TL donation without enough tokens
   should reject TL donation without enough tokens
   should allow donation of TL and MGOV tokens for valid member
   should reject survey submission when insufficient funds
   Should reject survey submission if not member
   should allow survey submission and voting
   should allow another member to take a survey
   should get survey details and ownership
   should reject proposal if not enough MGOV
   should allow submitting project proposal
   should allow voting for project proposal and payment
   should allow reserve and withdraw project payment (203ms)
   should allow vote for project payment after reserve and time passed (273ms)
   should delegate votes and allow target member to use them
   should get project status, owner and info
   should return correct counts and totals
   should revert when accessing non-existent survey or project
   should return zero for initial counts and values
```

Result: ✓ 29 passing (4s)
All tests passed successfully across all user scales (100, 200, 300, 400 members), demonstrating that the `MyGov.sol` contract enforces its governance constraints, token requirements, membership rules, and time-based payment logic as intended.

4 Gas Usage Analysis (Based on Test Results)

The gas consumption for key `MyGov` and `Lock` contract methods was measured using Hardhat's gas reporting. The average usage for each major function is summarized below.

Function	Avg Gas Used
`delegateVoteTo`	72,046
`donateMyGovToken`	36,096
`donateTLToken`	59,073
`faucet`	95,670
`reserveProjectGrant`	100,141
`submitProjectProposal`	278,674
`submitSurvey`	219,013
`submitSurvey`	219,013
`takeSurvey`	71,609
`testFund`	54,265
`transfer`	55,232
`voteForProjectPayment`	64,271
`voteForProjectProposal`	60,376
`withdrawProjectTLPayment`	96,700

The most expensive operations are related to project submission and funding due to storage updates and token transfers. The faucet system remains efficient despite being accessed by 300 addresses.

5 Conclusion

We designed the MyGov governance token contract to support democratic participation through voting, project proposals, and surveys. It complies with the ERC20 standard and extends functionality to match DAO needs.

Task Achievement	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 as documented test results.	X		
I have developed smart contract Solidity code and submitted it.	X		
I have used Hardhat and ethers.js for smart contract development and testing.	X		
I have implemented Chai assertions for testing.	X		
Function `delegateVoteTo` is implemented and works.	X		
Function `donateTLToken` is implemented and works.	X		

Task Achievement (continued)	**Yes**	**Partially**	**No**
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 `withdrawProjectTLPayment` 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 `getProjectNextTLPayment` 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		
Function `getNoOfFundedProjects` is implemented and works.	X		
Function `getTLReceivedByProject` 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 of these tests.	X		
I have tested my smart contract with 200 addresses and documented the results of these tests.	X		
I have tested my smart contract with 300 addresses and documented the results of these tests.	X		
I have tested my smart contract with more than 300 addresses and documented the results of these tests.	X		