Free TON Governance (Part II)

Repo — https://github.com/RSquad/BFTG
(contest branch: contest)
Tg — @inyellowbus

About

This document presents RSquad's submission for the Free TON Governance (Part II) Contest facilitated by DGO Subgovernance.

The code of the developed system of smart contracts can be found at https://github.com/RSquad/BFTG

The document contains:

- an overview of the statuses of competition requirements in relation to the solution developed by RSquad;
- description of the main smart contracts of the BFTG smart contract system;
- description of the smart contracts of the SMV smart contract system;
- description of third-party smart contracts used by the BFTG smart contract system;
- description of smart contracts used for testing;
- DeBots description;
- description of basic user scenarios;
- description of the test environment and infrastructure;
- description of test cases.

The BFTG smart contract system has been developed following the requirements provided in the contest description and in accordance with the architectural specifications developed as a part of the Practical BFT Governance.

Limitations:

- Node SE was used for development and testing;
- the developed solution deployed by RSquad in the DevNet network, but at the time of March 24 DevNet does not work consistently. RSquad will retest the deployed solution and fix all troubles after DevNet becomes stable again.

Glossary

- smc — smart-contract
- System — the BFTG smart contract system
- SMV — the DGO SMV smart-contract system (https://github.com/RSquad/smv)

Overview

The purpose of the System is to automate the decentralized governance for Free TON communities through voting.

The following figure shows the top-level diagram that describes the System:

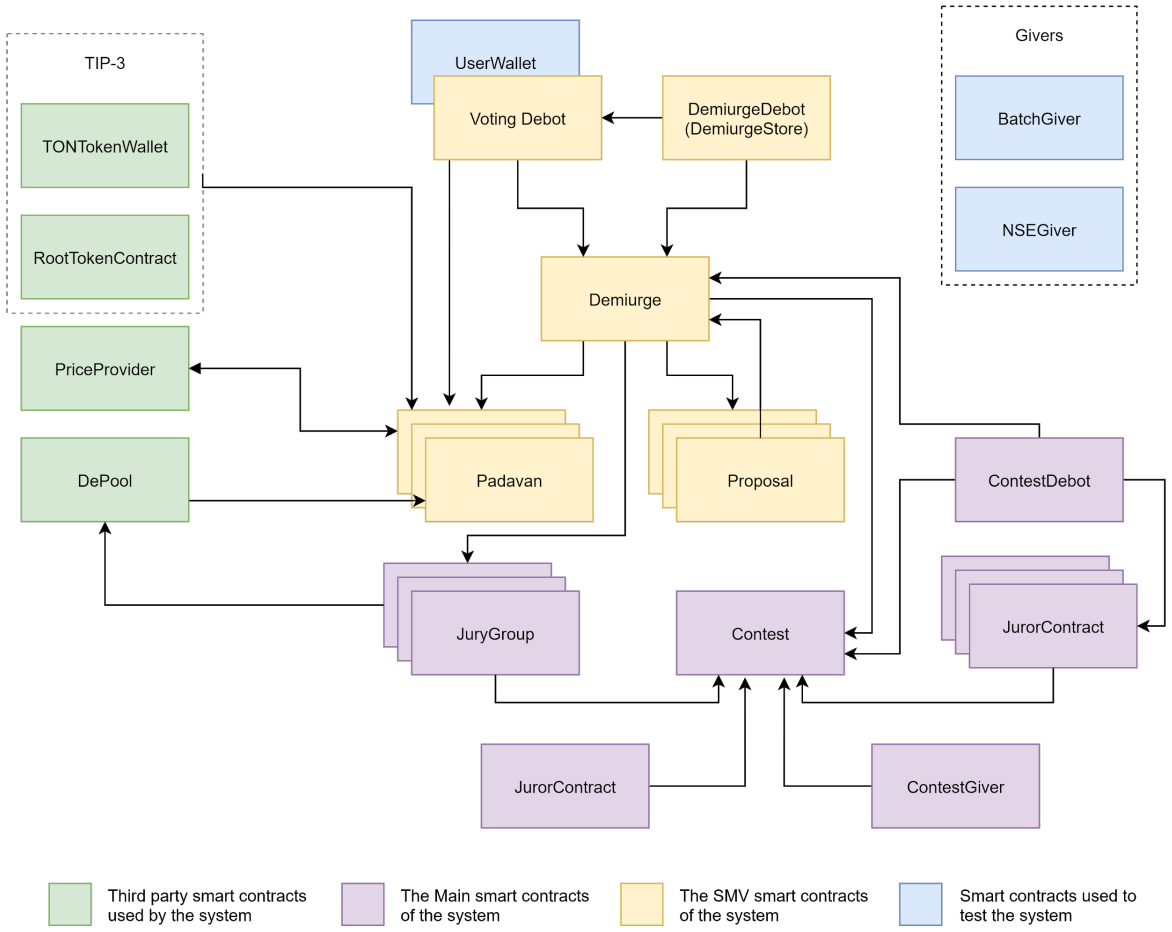


Figure 1 — High-level System Architecture

The implementation of the System supports all features described in previous contest (https://dgo.gov.freeton.org/proposal?proposalAddress=0:9d42e2c600d4d72ca45cdefb46763605bdaa70a549eed4f8619358613ce05eaa). Requirements from the current specifications of the contest is described in the table below:

Feature	**Status**	**Comment**
The BFTG system is built on the basis of the existing modified SMV solution	Fully supported	The SMV system is implemented and refined for the needs of the contest specification

Creation of ContestProposals, deployment of Contests based on the results of ContestProposals	Fully supported	For any contest of the system, a special Proposal must be created, according to the results of which Contest is accepted.
New full-cycle contests contracts	Fully supported	Contests have been completely redesigned and provide enhanced functionality relative to existing contests according to the competition specification
Hidden voting to the stage of the reveal	Fully supported	Two-phase voting mechanism has been developed in which jury members vote in encrypted votes and disclose them after the vote is completed
Receiving prizes and awards of the jury	Fully supported	All payments come from the system
Full featured tags and jury groups	Fully supported	The System emits a ProposalFinalized event when the timed / premature proposal ends.
Formation of jury groups from the winners by contest tags	Fully supported	New mechanism for selecting the jury, in which only the winner of the contest can become a jury member for the corresponding tag
Using tags to select a jury in a contest	Fully supported	The contestants are selected only from the groups of the corresponding topics
Lock of a winning prize in a special jury group	Fully supported	Part of the prize can be locked up with a special group
Sending the jury group stake to validation	Partially supported	A mechanism has been developed to implement this function, but since it is ideologically unclear who will be the holder of the node, the mechanism is not ready for production operation
Mechanics for calculating the duration of the contest stages	Partially supported	Contest stages are calculated according to a special formula depending on the number of submissions, the period for submitting applications, the maximum score, etc. At the moment, the formula is simplified relative to the one given in the document, since there is no possibility of implementing a floating point number
Mechanics of allocating funds for contests from giver	Fully supported	Contests receive a prize fund from special givers whose interface is developed
Point value calculation	Fully	The prize is calculated according to the

mechanics	supported	formula from the specification and is based on the number of earned points
DeBots coverage all of the main scenarios of the system	Fully supported	The functionality of the system is supported by the latest version of the DeBots on interfaces
Described slashing mechanics and jury blame	Currently is not supported	
Temporary jury voting slots and corresponding punishment mechanics for skipping a slot	Currently is not supported	

System's Smart contracts

The main smart contracts of the System

Contest

The Contest system smart-contract encompasses the full cycle of the FreeTON contest facilitation. Major segments of the functionality are:

- collection of contest entries;
- recording jury evaluations of the entries;
- assessment of the jury evaluations according to the contest regulations;
- calculation of the rewards for contestants and jurors following generally accepted principles of governance;
- distribution of the rewards to the winners per the specified process.

Collection of contest entries

submit(address participant, string forumLink, string fileLink, uint hash, address contact)
Records contest entries submitted by contenders.

Recording jury evaluations of the entries

vote(Evaluation evaluation)
Processes a single vote.

voteAll(Evaluation[] evaluations)
Processes mass votes.

Assessment of the jury evaluations according to the contest regulations

finalizeResults()

Processes the results and form the final set of raw data.

Calculation of the rewards for contestants and jurors following generally accepted principles of governance

rank()
Assesses the entries quality and the jurors' performance according to the specified metrics and criteria.

Distribution of the rewards to the winners per the specified process

claimContestReward()
Allows winners of the contest to claim their prizes.

claimJurorReward()
Allows qualified members of the jury of the contest to claim due rewards.

claimContestRewardAndBecomeJuror()
Allows winners of the contest to invest parts of the hard earned rewards in the selected areas of expertise governed by this contest, effectively joining the respective jury groups.

Transparency of the contest at all stages

The following get-methods provide access to the key data of the Contest contract:
getEntryStats(uint8 entryId) public view returns (Stats entryStats)
Stats for an entry.

getJurorStats(uint8 jurorId) public view returns (Stats jurorStats)
Stats for a juror.

contestStatistics() public view returns (uint16 pointsAwarded, uint16 totalVotes, uint16 avgScore, uint8 entries, uint8 jurorsVoted)
Overall contest statistics:

- total points awarded by all jurors combined;
- total number of votes;
- average score (multiplied by 100);
- number of entries submitted;
- unique jurors voted.

getCurrentData() external override view returns (ContenderInfo[] info, address[] juryAddresses, Stats[] allStats, mapping (uint16 => Mark) marks, mapping (uint16 => Comment) comments, mapping (uint16 => HiddenEvaluation) hiddens)
Snapshot of the contest data.

getFinalData() public view returns (Stats[] contestResults, Stats[] juryStatistics, mapping (address => Payout) contestPayouts, mapping (address => Payout) juryPayouts, ContestTimeline timeline)
Resulting contest data.

getJurorId(address addr) external view override returns (uint8 jurorId)
Returns juror ID for the specified address.

getContestInfo() external override returns (ContestInfo contestInfo)
Returns general information about the contest.

getContestTimeline() external override returns (ContestTimeline timeline)
Returns timeline of the contest.

getContestSetup() external override returns (ContestSetup setup)
Returns all components of the contest setup.

getContest() external view override returns (ContestInfo contestInfo, ContestTimeline timeline, ContestSetup setup, Stage stage)
Returns the exhaustive set of data points for the contest.

JurorContract

Contract for voting for submissions. Each member of the jury transmits the public key when he becomes a member and Demiurg deploys the JurorContract for him.

Through it, you can vote with hidden votes, reveal the votes and manage the balance in JurYGroup.

Also, the Contract stores encrypted lines of hidden voices so that each juror can download them through the DeBot (or any other client), disassemble them using the specified pin-code and reveal their vote without memorizing and repeating the full vote.

JuryGroup

Contest defining the groups of the jury by tags. Allows Demiurge to register new jury members based on Contest results, collecting stacks of winners.

The contract can be resolved by tag and provides information on the jury members to the Contest upon request.

ContestGiver

ContestGiver is a smart contract which is developed to give prize pools to contests. It is deployed by Demiurge and is called when creating a Contest for transferring tokens.

DemiurgeStore

The contract is the central repository of the system. It accepts and stores codes of almost all contracts of the System, from JuryGroup to Contest. The contract is deployed first during System initialization and waits for the required codes to be loaded. After that, other

contracts of the system can apply for the code of the required contract and receive it in the callback.

This contract is necessary to update the system elements and also so that there is no need to "drag" heavy code between contractions.

ContestDebot

Entry point to the Contest voting system. Allows you to view information about the Contest, submit applications and vote for them using JurorContract. Moreover, this DeBot is deployed once for all available System Contests and allows any user to interact with any System Contest.

SMV smart contracts of the System

Demiurge

It is a central smart contract in a voting system. It is a ledger that creates and stores proposals and user padawan addresses. After deployment demiurge requests Demiurge Store smart contract to gain proposal and padawan images (tvc), list of depool addresses and address of vote price provider.

Demiurge starts in preworking mode in which it has several checks that must be passed before it will accept requests to deploy proposals and padawans. Checks contains the following:

1. Check that the demiurge contains a Proposal image.
2. Check that the demiurge contains a Padawan image.
3. Check that demiurge contains a list of depools.
4. Check that demiurge contains the address of a price provider.

When the check mask becomes equal to 0 Demiurge is ready to work.

Padawan

Padawan smart contract is a user ballot that allows users to vote for proposals. Padawan accepts deposits of different types (tons, tip3 tokens, depool stakes), converts them to votes and sends votes to proposals. Votes cannot be converted into deposits and received back until all the proposals that the Padawan voted for are completed. The User can vote for different proposals with a different number of votes, but a number of locked votes in padawan is always the maximum number of votes spent for one proposal.

At any time a user can ask to reclaim some deposits equivalent to a number of votes. When it happens Padawan starts to query the status of all voted proposals. If any of them is already completed Padawan removes it from the active proposals list and updates the value of locked votes. If the required number of votes becomes less or equal to unlocked votes

then Padawan converts the requested number of votes into a deposit (tons, tokens or stake) and sends it back to the user.

Padawan is controlled by a user contract that requested **deployPadawan** from Demiurge.

Proposal

Smart contract that accumulates votes from Padawans. Deployed by Demiurge by user request (**deployProposal**) Notifies about its state to Demiurge.
Can be optionally instantiated with a white list of Padawan addresses. In that case Proposal accepts votes only from addresses from this list.

Demiurge Debot

An entry point to an onchain voting system. Allows to deploy new Demiurge to blockchain or to attach to existing Demiurge. Also deploys Voting Debot for users.
Debot implements the interface of Demiurge Store and stores all images (tvc) and ABIs of voting system contracts.

Voting Debot

Debot that works on behalf of the user. Deploys Padawan and allows to create new proposals, deposit tons, convert them to votes and vote for existing proposals.

External smart contracts used by voting system

NSEGiver

Builtin giver of NodeSE. Used to deploy contracts in local node tests.

RootTokenContract & TONTokenWallet

TIP3 smart contracts. Can be found here:
https://github.com/tonlabs/ton-labs-contracts/tree/master/cpp/tokens-fungible

DePool

DePool smart contract. Used to transfer ownership of user stake to Padawan. Can be found here:
https://github.com/tonlabs/ton-labs-contracts/tree/master/cpp/solidity/depool

PriceProvider

Simple smart contract that implements an interface of converting tons and tokens to votes.

Smart contracts used by test system

UserWallet

Test user wallet used to send requests to Demiurge and control Padawan.

BatchGiver

Giver smart contract that allows to make several transfers in one transaction. Used to increase speed of contracts deployment.

TagsResolve

Smart contract for testing resolve of tags and jury groups.

TestDeployer

Smart contract for unit testing of various contracts of the system, implying an on-chain deployment, for example, JuryGroup.

Voting

Smart contract for unit testing of hidden voting scenario.

How to use DeBots

1) Compile all contracts.
2) Run nodeSE locally.
3) Download tonos-cli (version >= 0.8.2).
4) Deploy and initialize Demiurge Debot. It is an entry point to the whole SMV system. Go to the root of the project repository. Change the current directory to **sbin** and run bash script **deploy_pp.sh** and then **deploy_dd.sh.**
5) After Debot starts you will see the main menu.

```
Connecting to http://127.0.0.1
Demiurge DeBot, version 0.6.0

Hello, user, i'm a Demiurge Debot.
Current Demiurge: 0:0000000000000000000000000000000000000000000000000000000000000000
What do you want to do?
1) Deploy new demiurge
2) Attach existed demiurge
3) Deploy user voting debot

debash$
```

6) Choose menu item 1 to deploy new SMV voting system “demiurge”. Generate a new seed phrase and generate a public key from the phrase. Enter the public key.

```
debash$ 1
Demiurge balance: 0 nanotokens
Please, generate seed phrase for new demiurge.
Enter public key derived from this phrase:
801fe857364d946a0b6b7a88510aa49feb5341e1080c10986367876475b1bf10
New demiurge address:
0:630394fd1965e72cfc894a2de41f14c35d400d9a96165568ba6ae0a2f1732ef0
Ready to deploy? (y/n)
y
Sending message 24b73a893828753bb7bdb53b1a6c27a1296d427a26e65a91b39c94a1a7860500
Deploy succeeded.
```

7) You have deployed a new voting system. Now create a personal voting client - voting debot. Choose menu item 3 and follow the instructions of debot.

```
debash$ 3
Demiurge balance: 20522453000 nanotokens
I'll guide you step by step to deploy your personal debot for voting.
This debot will help you create proposals, vote for proposals and also deposit and reclaim funds for voting.
Generate seed phrase for your personal debot. Keep it in secret because you will use it to control debot.
Enter public key derived from seed phrase:
f87e2f09797d8bfb3aa8f777b096d0dc57b65127905586d4a374458d2140da11
Voting debot address:
0:8164418cbb73c7a8ee77c6649e4ab46744bf734559f7f7b6491b93c2399bca73
Ready to deploy? (y/n)
y
Sending message 7a30fd8b8ec14581a73e4ddfcc70a62bca03a80f5fce98b34898571ac01f01ee
Deploy succeeded.
```

8) Your Voting Debot is deployed. Lets run it.
tonos-cli debot fetch <address_of_voting_debot>

```
Connecting to http://127.0.0.1
Personal Voting DeBot, version 1.2.0

Hello, i'm your personal Voting Debot!
You don't have a padawan contract yet.
Ready to deploy? (y/n)
```

9)

10) You don’t have a Padawan yet. Deploy it. Sign it with a seed phrase for voting debot.

```
Ready to deploy? (y/n)
y
My balance: 9.742701000 tons
Deploy fee is 3.100000000 tons
enter seed phrase or path to keypair file
gloom person worth size opera build prepare like shoot relief axis minute
Sending message 86f9f89177884c5eeab034205d7e8166d51944c1bf9046fdc4de12c37fe16fc6
Transaction succeeded. Please, wait for a few seconds and restart debot.
Debot Browser shutdown
```

11) Padawan deployed. Restart the debot and you will see the extended menu of Voting Debot.

```
Connecting to http://127.0.0.1
Personal Voting DeBot, version 1.2.0

Hello, i'm your personal Voting Debot!
My balance: 6.611922685 tons
You voted for 0 proposals.
Your total votes: 0
Locked votes: 0
Unused votes: 0
Padawan address: 0:0d391669dd469965d0ab2c3c5c6a146e7655e5f36de500c0d3f58595b2fb171f
What do you want to do?
1) Acquire votes
2) Create new contest proposal
3) View all proposals
4) Vote for proposal
5) View contests
6) Update info

debash$
```

12) Choose menu item 2 to create a new Contest Proposal. Follow the instructions to set proposal parameters. Sign request with Voting Debot.

```
debash$ 2
Enter total votes:
(>= 3 and <= 1000000)
10
Enter unixtime when voting for proposal should start:
(>= 1616623127 and <= 4294967295)
1616623127
Enter duration of voting period for contest proposal (in seconds):
(>= 1 and <= 31536000)
3600
Enter title:
Test Contest Proposal 1
Enter description:
(Ctrl+D to exit)
This is a description of a contest
Choose voting model:
1) Super majority
2) Soft majority

debash$ 2
Enter contest prize pool (in tons):
(>= 1.000000000 and <= 1000000.000000000)
100
Enter contest duration (in seconds):
(>= 1 and <= 31536000)
300
Enter contest tag:
initial
Add one more tag? (y/n)
n
Creation fee: 3.100000000 tons.
Sign and create proposal? (y/n)
y
enter seed phrase or path to keypair file
```

13) After returning to the main menu choose menu item 3 to view a list of all proposals.

```
debash$ 3
List of proposals:

ID 0. "Test Contest Proposal 1"
Status: New
Start: 1616623127, End: 1616626727
Total votes: 10, options: "soft majority"
Address: 0:84597e292c20d7f244d7678d0bbba5bb16f71230449381a3e05bd99d0f1e034d
creator: 0:8164418cbb73c7a8ee77c6649e4ab46744bf734559f7f7b6491b93c2399bca73

Back to start
```

14) Choose item 1 to deposit tons into Padawan and receive votes.

```
debash$ 1
How do you want to get votes?
1) Deposit tons

debash$ 1
Enter a number of tons:
(>= 1 and <= 101)
20
Sign and deposit? (y/n)
y
enter seed phrase or path to keypair file
gloom person worth size opera build prepare like shoot relief axis minute
Sending message 1d3fe2798091a465bff25d9803152604f05ab334a94e9f6f474606bc23a30514
```

15) After returning to the main menu you will see updated voting stats. You have 20 available votes.

```
Hello, i'm your personal Voting Debot!
My balance: 81.157982863 tons
You voted for 0 proposals.
Your total votes: 20
Locked votes: 0
Unused votes: 20
```

16) Choose item 5 to vote for the proposal. Enter the number of votes to send. To complete a proposal send more than 50% of total votes. Sign request with seed phrase of Voting Debot .

```
debash$ 5
Enter proposal id:
(>= 0 and <= 4294967295)
0
Enter votes count:
(>= 0 and <= 20)
8
How to vote?
1) Vote "Yes"
2) Vote "No"

debash$ 1
Sign and send votes? (y/n)
y
enter seed phrase or path to keypair file
gloom person worth size opera build prepare like shoot relief axis minute
Sending message 17c7c6c6ca6e8347a5b04ecc8f4b909ede3ca7bf928c95c05e5996b19a6dcad4
```

17) Now you have 1 active proposal and 8 locked votes.

```
Hello, i'm your personal Voting Debot!
My balance: 81.125770013 tons
You voted for 1 proposals.
Your total votes: 20
Locked votes: 8
Unused votes: 12
```

18) Choose menu item 4 to see a list of all proposals. Mention that the proposal has status “Passed”.

```
ID 0. "Test Contest Proposal 1"
Status: Passed
Start: 1616623127, End: 1616626727
Total votes: 10, options: "soft majority"
Address: 0:84597e292c20d7f244d7678d0bbba5bb16f71230449381a3e05bd99d0f1e034d
creator: 0:8164418cbb73c7a8ee77c6649e4ab46744bf734559f7f7b6491b93c2399bca73
```

19) If a proposal is passed it means that the contest smart contract is automatically deployed by demiurge. Deploy and run Contest Debot to see contests. Open **deploy_cd.sh** script and update vars **demiurge** (set to address of demiurge) and **store** (set to address of DemiurgeDebot). Run the script.

```
Connecting to http://127.0.0.1
Contest DeBot, version 0.2.0

What do you want to do?
1) View contests
2) Deploy juror wallet
```

20) Choose menu item 1 to see a list of all running contests.

```
debash$ 1
Sending message b9c2f495f741808e44d78a7ed82046407d3c30a3c62855ddb491d861f44330ac
Sending message 1f434e7615f300f805b8bfe0cfaff9541219096c732e7515b688b1825e095c90
List of contests:
1) Test Contest Proposal 1
2) Test proposal 2
3) To main
```

21) Choose one of the contests.

```
debash$ 2
ID1 "Test proposal 2"
Status: Accepting submissions
Address: 0:5a1a1336281d6490bce93f78af80bdbfa8fcea2a59c7f62901d609397e9ff2d9
Created at: 1616624434, ends at: 1616624614
Prize pool: 70.000000000
Tags: #initial

Submissions:


Jury:

ID0 Address: 0:54d27f3a1af5b1f43bf7bdb409e80711ed9457d86cef28ee2e2ad520d30cdb88
ID1 Address: 0:575368375493ec5e411f748823e4af4a7efd05b42ab98a19c8c0a4dbb6d88f90
ID2 Address: 0:a4fd1988e71b91d44d5d91b56c80708dac09d3904bfae6efbca7c3877288ff39
Options:
1) Add submission
2) Return to contest list
```

You will see a list of submissions and jurors. Choose menu item 1 to add new submission.

```
debash$ 1
Enter your Free TON wallet address:
0:841288ed3b55d9cdafa806807f02a0ae0c169aa5edfe88a789a6482429756a94
Enter link to submission pdf:
https://pdf
Enter forum link for discussion:
https://forum.freeton.org/
Sign and submit? (y/n)
y
enter seed phrase or path to keypair file
gloom person worth size opera build prepare like shoot relief axis minute
Sending message 82066256ac20a9fb9c82ea799889389948db2ad8e3b1cc795b9a4166568d8df8
Submission added.
```

Return to the main menu. Choose the contest again and see the updated list of submissions.

```
debash$ 2
ID1 "Test proposal 2"
Status: Accepting submissions
Address: 0:5a1a1336281d6490bce93f78af80bdbfa8fcea2a59c7f62901d609397e9ff2d9
Created at: 1616624434, ends at: 1616624614
Prize pool: 70.000000000
Tags: #initial

Submissions:

ID0
Wallet address: 0:841288ed3b55d9cdafa806807f02a0ae0c169aa5edfe88a789a6482429756a94
Forum link: https://pdf
File link: https://forum.freeton.org/
Applied at: 1616624533


Jury:

ID0 Address: 0:54d27f3a1af5b1f43bf7bdb409e80711ed9457d86cef28ee2e2ad520d30cdb88
ID1 Address: 0:575368375493ec5e411f748823e4af4a7efd05b42ab98a19c8c0a4dbb6d88f90
ID2 Address: 0:a4fd1988e71b91d44d5d91b56c80708dac09d3904bfae6efbca7c3877288ff39
Options:
1) Add submission
2) View submission marks
3) Return to contest list
```

After the contest is finished go to the contest view. `Add submission` menu item changed to `Vote for submission`. If you are a juror choose it and follow the instructions.

```
debash$ 1
Enter juror pubkey:
82c9fc3bf4a0afedc84e1a087807d31fbc5ebd4f3589172b983852af3622ca63
Enter submission id:
(>= 0 and <= 0)
0
Enter score:
(>= 0 and <= 10)
8
Enter comment:
(Ctrl+D to exit)
good submission
Enter encryption key:
11223344
Juror wallet address: 0:575368375493ec5e411f748823e4af4a7efd05b42ab98a19c8c0a4dbb6d88f90
Sign and record vote? (y/n)
y
enter seed phrase or path to keypair file
return fortune vague flower fold appear family label alpha enforce machine fun
Sending message f2f327042bf0a7fe2ca6e4f7c6135f95a6d4bcfa6d5386f3567e96955db311b7
Vote recorded
What do you want to do?
1) View contests
2) Deploy juror wallet

debash$
```

When the voting period for submissions is over run the debot again and go to the contest view. Now you can reveal your vote as a juror. Choose `reveal vote for submission`.

```
Options:
1) Reveal vote for submission
2) View submission marks
3) Return to contest list
```

```
debash$ 1
Enter juror pubkey:
82c9fc3bf4a0afedc84e1a087807d31fbc5ebd4f3589172b983852af3622ca63
Enter submission id:
(>= 0 and <= 0)
0
Enter encryption key:
11223344
Juror wallet address: 0:575368375493ec5e411f748823e4af4a7efd05b42ab98a19c8c0a4dbb6d88f90
```

Deploy and initialize the System

There are two ways to initialize the system - manual and through DeBot

Manual system start

To start the system manually, you need to prepare:

- Demiurge contract
- DemiurgeStore contract
- Padawan contract
- PriceProvider contract
- Proposal contract
- Contest contract
- JuryGroup contract
- Juror contract

System deployment script:

1. Deploy PriceProvider
2. Deploy DemiurgeStore
3. Point Proposal's tvc to DemiurgeStore
4. Point Padawan's tvc to DemiurgeStore
5. Point Contest's tvc to DemiurgeStore
6. Point JuryGroup's tvc to DemiurgeStore
7. Point JurorContract's tvc to DemiurgeStore
8. Point address of PriceProvider to DemiurgeStore
9. Point addresses of DePools to DemiurgeStore
10. Point DemiurgeStore address to Demiurge and deploy
11. The System is ready to use

System start using DeBots

The voting system can be configured and used with debots. There are 3 debots:

1) Demiurge Debot - central debot. One for the whole voting system. Deploys Demiurge and Voting Debot for each user.
2) Voting Debot. One debot per user. Deploy Padawan and allow users to vote.
3) Contest Debot - to work with contests: view, vote, add submissions, reveal votes.

See section 'How to use Debots' that describes how these debots are working

User Scenarios

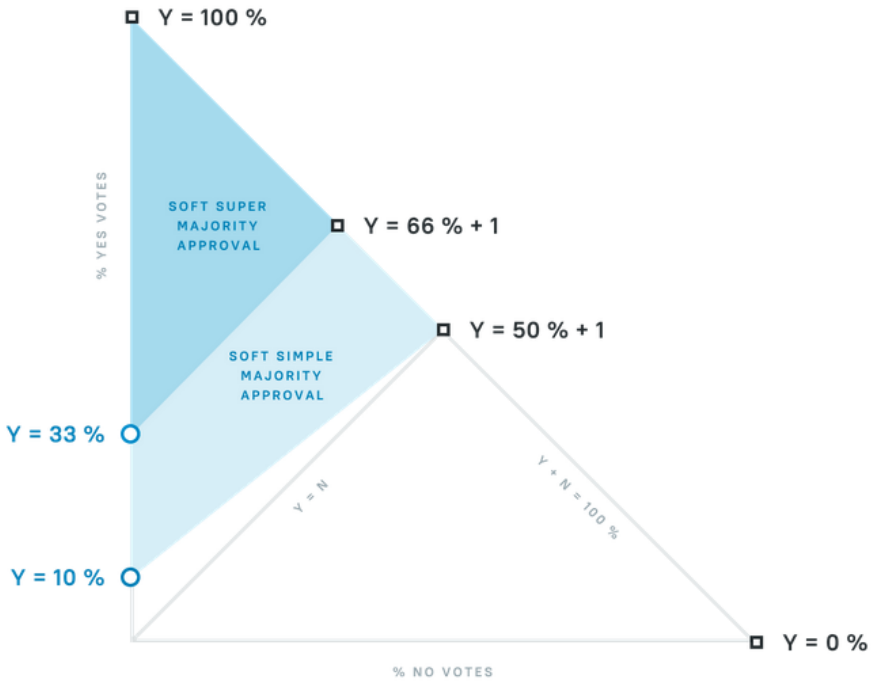
Proposal creation scenarios

Proposal can be created by any user after deploying and initializing the system. Importantly, Proposal accepts only internal messages, therefore, to deploy and work with Proposal, you should use Multisig, UserWallet from the example or analogs.

The main parameters and functions of the Proposal are described in System's Smart Contracts paragraph.

To create a Proposal, you need to specify:

- Voting period — the start and end time of voting after which the results are summed up
- The voting model is Majority, Soft Majority or Super Majority, below are the formulas for calculating the model, where y - votes for, n - votes against, t - total votes according to the picture:



    - Majority (y > n)
    - Soft Majority (y * t * 10 >= t * t + n * (8 * t + 20))
    - Super Majority (y * t * 3 >= t * t + n * (t + 6))

  More details can be found here —
  https://forum.freeton.org/t/developers-contest-soft-majority-voting-system-finished/65
  An example of using all models can be found in the majorities test
- Description
- Accompanying text
- White sheet of voters:
    - Not specified, in which case all Padawan owners can vote

    - Specified, in this case, only those Padawans whose identifier is indicated in the sheet vote
    - A link to the group is specified - in this case, only Padawans members of the specified group vote
- Appointment proclaimed
    - No final result handler
    - To create a contest
    - To add to the group
    - To remove from the group
    - To create a group

Please, see proposal creation examples here ./tests/parts/deploy-proposal.ts

Group scenarios

1. Adding a new member to the group
    a. invoke *applyFor(string name)* function of the *Group* contract, as specified in the *IGroup* interface. Address of the sender is considered to be applying for the group membership. NB: *Padawan* contract can be efficiently used for the application process. The corresponding function is *applyToGroup(address group, string name)* from the *IPadawan* interface.
    b. Provided the input data is valid, a proposal to include a new member to the group is automatically created and put to voting.
    c. Upon voting completion, the results are evaluated.
    d. If the proposal passes, the applicant is added to the list of group members, and becomes eligible (and responsible) to vote for the proposals in scope of this group from now on.
2. Removing a member from the group
    a. invoke *unseat(uint32 id, address addr)* function of the *Group* contract, as specified in the *IGroup* interface. The respective helper in the *IPadawan* interface is *removeFromGroup(address group, uint32 id, address addr)*.
    b. Provided the input data is valid, a proposal to remove the specified member from the group is automatically created and put to voting.
    c. Upon voting completion, the results are evaluated.
    d. If the proposal passes, the specified member is removed from the group, thus revoking voting rights for the proposals deployed subsequently.
3. Voting using groups (whitelist)
    a. proposals deployed by a group are put to voting in a very special fashion, enabling only a selected list of individual contracts to vote for them. This voting model is sometimes referred to as “whitelist”. Proposals with this feature disregard any votes cast from the addresses not on the list.

Base Voting Scenarios

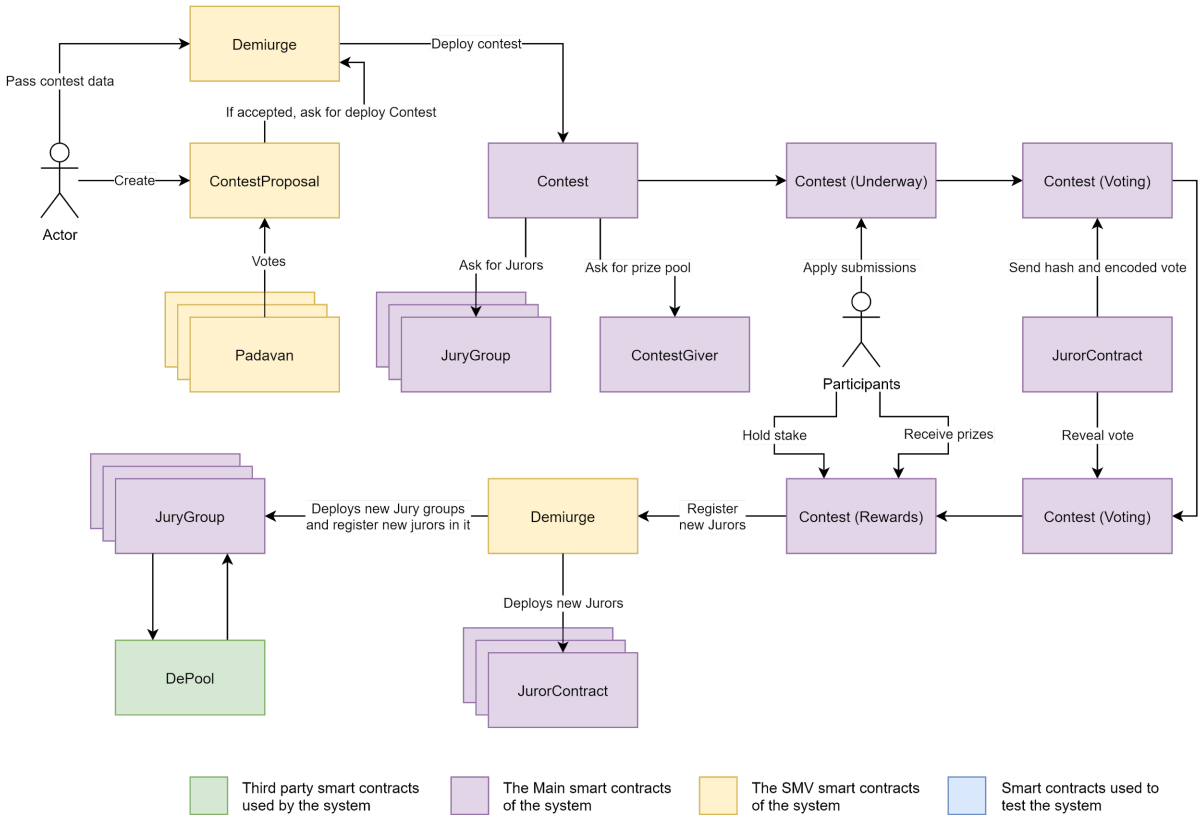
1. Voting with TON
    - User deploys Padawan, or requests previously deployed Padawan
    - User sends TONs to Padawan

    - Padawan "converts" TONs into voices
    - User sends votes from Padawan to Proposal
    - The volume of sent votes in tokens is frozen
    - Proposal ends on time, or prematurely, if the result is unambiguous
    - The volume of votes sent by all users is unfrozen
    - Proposal informs Demiurge of the voting result
    - The Demiurge performs an action if it was described and the result was accepted
    - User withdraws deposited TONs

Base and base-against test (to check voting for and against, respectively)

2. Voting with DePool:
    - User deploys Padawan, or requests previously deployed Padawan
    - The user transfers the stake from the DePool specified in the Demiurge to the Padawan
    - Padawan "converts" stake into votes
    - User sends votes from Padawan to Proposal
    - The volume of sent votes is frozen
    - Proposal ends on time, or prematurely, if the result is unambiguous
    - The volume of votes sent by all users is unfrozen
    - Proposal informs Demiurge of the voting result
    - The Demiurge performs an action if it was described and the result was accepted
    - User withdraws stake
3. Voting using TIP-3
    - User deploys Padawan, or requests previously deployed Padawan
    - User creates a token account for Padawan
    - User transfers tokens to Padawan
    - Padawan "converts" tokens into votes at the rate given by PriceProvider
    - User sends votes from Padawan to Proposal
    - The volume of sent votes in tokens is frozen
    - Proposal ends on time, or prematurely, if the result is unambiguous
    - The volume of votes sent by all users is unfrozen
    - Proposal informs Demiurge of the voting result
    - The Demiurge performs an action if it was described and the result was accepted
    - User withdraws sent tokens
4. Basic voting scenario with combined votes. Combines the first three scenarios and combines ways to get votes.

Main full-cycle scenario



1. Deploy the system
2. Create ContestProposal with needed information
3. Pass Contest parameters (contest duration, giver, tags, prize pool etc.) to Demiurge what would be in the Contest if the ContestProposal will be accepted
4. Follow “Proposal creation scenarios” and “Base Voting Scenarios” to accept ContestProposal
5. After ContestProposal acceptance, contest will be deployed immediately
6. Contest asks for jurors from JuryGroups by tags that was passed to the Contest
7. Contest asks Giver for prize pool
8. After the Contest receives the list of the jury and the prize pool, it will automatically start accepting submissions
9. Participants accept their submissions
10. After the completion of the stage of accepting submissions, the Contest calculates the voting period, which depends on the number of submissions received, and proceeds to the voting stage
11. At the voting stage, all the jury received from the groups by the corresponding tags, through their JurorContract, can send a vote, which consists of
    a. hash of vote (type, rating, comment, submission number)
    b. encoded vote strings (type, rating, comment) with a key according to the chacha20 algorithm
12. Contest will store all votes

13. After the completion of the voting stage, the reveal stage begins, which lasts one day. In it, the judges send their revealed votes to the Contest, or, using the DeBot, they reveal the encryption key. DeBot, in turn, downloads the encrypted strings and converts them to the desired format.
14. The Contest will take the hash from the revealed vote and compare it with the original hash. If they converge, the vote will be revealed and registered
15. All jury members who have not revealed their votes are not counted in the final tally
16. After reveal stage, the Contest will calculate point value and make reward tables, where the reward is the multiplication between the value of the point and the points earned
17. Participants can now claim their prizes or stake a piece to become the jury.
18. The participant transmits an array of tags and the amount of TONs to each of them, as well as his public key to create a jury contract (or not create a contract if one already exists)
19. Contest calls Demiurge, to make winner a juror
20. Demiurge deploys JurorContract for participant and register him in JuryGroup by tags (if JuryGroup doesn't exists Demiurge will deploy it)
21. Now participant can pick up the leftover prize

Testing

All tests of the System are located in the ./tests directory.

To run tests, it is needed to install Node and tondev package

1. Install node.js
2. Install docker
3. Install tondev. npm install -g tondev. If you encounter problems during installation, read the instructions in the official repository
4. Go to the project folder and install the dependencies with npm install
5. Create .env file at the root of the project and fill it in.
    - Available variables (this example is used to work with Node SE)
        i. NSE_GIVER_ADDRESS=0:841288ed3b55d9cdafa806807f02a0ae0c169aa5edfe88a789a6482429756a94
        ii. NETWORK=LOCAL
6. Create ./build folder
7. Create ./ton-packages folder
8. Run Node SE — tondev se start
9. Run — npm run test:compile — it will compile all smart-contract and run all available tests

Infrastructure

ton-contracts.ts

Class for working with TON contracts. It provides a convenient interface for deploying, calling, getting balance, and so on. Used in tests everywhere.

Interface:

```
export class TonContract {
 client: TonClient;
 name: string;
 tonPackage: TonPackage;
 keys?: KeyPair;
 address?: string;

 async init(params?: any): Promise<void> {}

 async callLocal({ functionName, input = {} }: { functionName: string; input?: {}
}): Promise<DecodedMessageBody> {}

 async call({ functionName, input }: { functionName: string; input?: any }):
Promise<ResultOfProcessMessage> {}

 async calcAddress({ initialData } = { initialData: {} }): Promise<string> {}

 async deploy({ initialData, input }: { initialData?: any; input?: any } = {}):
Promise<ResultOfSendMessage> {}

 async getBalance(): number {}
}
```

ton-packages.ts

Package which consists of ABI and tvc.

Interface:

```
type TonPackage = {
 image: string;
 abi: {};
```

```
};
```

Description of tests

Test “voting.unit.test”

It tests the contest voting scenario. The test checks:

- calculating evaluation hash,
- calculating evaluation encoded,
- voting with hashed vote,
- revealing of hidden votes

Test “tags-resolve.unit.test”

It tests the jury groups resolving by tags. The test checks:

- JuryGroup deploying,
- new member registering,
- JuryGroup addresses calculation by tags.

Test “register-jury.test”

It tests scenario when the winner holds his stake to become a juror. The test checks:

- registration of new jury member and new group through Demiurge,
- registration of new jury member to existed group through Demiurge,
- registration of new jury member to existed group two times (to check balance top up) through Demiurge

Test “jury-group.unit.test”

It tests the jury group's simplest case. The test checks:

- JuryGroup deploying,
- registration of new members.

Test “initial-jury.test”

It tests the case of adding initial groups to Demiurge. The test checks:

- DemiurgeStore deploying,
- Demiurge deploying,
- Initial members

Test “contest.unit.test”

It tests the case of initiating Contest. The test checks:

- Contest deploying,
- Contest tags revealing.

Conclusion

The system developed during the contest is undoubtedly extremely voluminous. But despite this, we did it. We applied all our knowledge accumulated during the whole time of working with FreeTON, from writing simple contracts to DePool, TIP-3 and DeBots. The total volume of the source code of contracts is close to 200Kb, more than 20 developed contracts and the same number of interfaces. This allows us to say that this is one of the largest systems developed within the FreeTON network.

In addition to the implementation of the contest requirements, we also updated the first part of the contest — the DeBots of the SMV were completely rewritten from events to interfaces.

As part of the support and subsequent DGO contests, we will definitely finalize a number of features, and we will also improve the developed system with the latest technologies and patterns.