**SCRYPTA**

**архитектура**

**децентрализованной автономной организации**

**для оказания юридических услуг**

/\*

ДАО Скрипта представляет из себя комплекс смарт-контрактов, регламентирующих и автоматизирующих управленческую деятельность юридической фирмы, состоящей из одного управляющего партнёра и нескольких исполняющих партнёров. ДАО Скрипта опосредует юридическую структуру общества с ограниченной ответственностью в российской юрисдикции (далее – ООО) таким образом, что все партнёры (а) участвуют в уставном капитале такого ООО и имеют право на его прибыль, а также (б) входят в состав высшего органа управления ООО, через голосование на собраниях которого определяют волю ООО, как участника гражданского оборота на рынке юридических услуг. Управляющий партнёр, помимо этого, является генеральным директором в указанном ООО и совершает (заключает) сделки от имени ООО в рамках тех полномочий, которые он получил от высшего органа управления.

Из-за особенностей своей структуры и привязки к юридической модели ООО указанное ДАО не является полностью автономным. Для создания более высокой степени автономности организации полномочия генерального директора существенно ограничены смарт-контрактами, оставляя ему возможность либо машинально следовать тем решениям, которые принял высший орган управления (перемещая конкретные смарт-контракты и ДАО в целом из состояния в состояние), либо своим бездействием накладывать на решения высшего органа управления «условное» вето, погружая ДАО в состояния частичного или полного останова.

Этапы работы

1. Согласование технического задания
2. Написание кода Solidity
3. Покрытие кода unit-тестами
4. Внутренний аудит
5. Деплой контрактов и эмуляция в testnet
6. Деплой контрактов в mainnet
7. Верификация байт-кода контрактов на etherscan.io
8. Передача управления контрактами партнёрам
9. Проверка связей и владельцев контрактов
10. Пробная инвестиция в mainnet
11. Оперативная поддержка и устранение багов

\*/

/\*

Деятельность ДАО регулируется несколькими взаимосвязанными смарт-контактами

\*/

**contract** Equity {

// регулирует порядок распределения долей ООО

}

**contract** Equity\_Ballot {

// регулирует порядок голосования участников ООО по уставу

}

**contract** Director {

// регулирует порядок действий директора ООО

}

**contract** GST\_Auction {

// формирует цену реализации человеко-часов ДАО

}

**contract** GST\_Sale {

// регулирует выпуск (продажу) человеко-часов ДАО

}

**contract** GST\_Ballot {

// регулирует порядок голосования партнёров по вопросам

// распределения человеко-часов между собой/исполнителями

}

**contract** GST\_Circle {

// регулирует оборот внутреннего расчетного токена ДАО

}

**contract** Pool {

// регулирует порядок формирования и распределения прибыли ДАО

}

/\*

Граф состояний ДАО:

To do

\*/

**contract** Equity {

// регулирует порядок распределения долей ООО

}

/\*

GPT (Gilda Prima Token) – внутренняя условная (расчетная) единица ДАО, которая представляет собой форму фиксации размера доли в ООО и предназначена для (а) ведения реестра участников ООО, (б) программирования оборота долей ООО на основе алгоритмов, предусмотренных в российском законодательстве, (в) опосредования процессов голосования на общем собрании участников ООО (см. контракт Equity\_Ballot) и (г) распределения прибыли, полученной по итогам отчетного периода (см. контракт Pool).

Multisig - требование заверения любого предусмотренного (а) события распределения (или перераспределения) долей ООО, (б) события по итогам голосования на общем собрании участников – кворумом электронных подписей участников; подписи и параметры кворума определены в уставе ООО и теоретически изменяемы.

\*/

**contract** Equity\_Ballot {

// регулирует порядок голосования участников ООО по уставу

}

**contract** Director {

// регулирует порядок действий директора ООО

}

**contract** GST\_Ballot {

// регулирует порядок голосования участников ООО по вопросам

// распределения человеко-часов между собой/исполнителями

}

pragma solidity ^0.4.22;

/// @title Voting with delegation.

contract Ballot {

// This declares a new complex type which will

// be used for variables later.

// It will represent a single voter.

struct Voter {

uint weight; // weight is accumulated by delegation

bool voted; // if true, that person already voted

address delegate; // person delegated to

uint vote; // index of the voted proposal

}

// This is a type for a single proposal.

struct Proposal {

bytes32 name; // short name (up to 32 bytes)

uint voteCount; // number of accumulated votes

}

address public chairperson;

// This declares a state variable that

// stores a `Voter` struct for each possible address.

mapping(address => Voter) public voters;

// A dynamically-sized array of `Proposal` structs.

Proposal[] public proposals;

/// Create a new ballot to choose one of `proposalNames`.

function Ballot(bytes32[] proposalNames) public {

chairperson = msg.sender;

voters[chairperson].weight = 1;

// For each of the provided proposal names,

// create a new proposal object and add it

// to the end of the array.

for (uint i = 0; i < proposalNames.length; i++) {

// `Proposal({...})` creates a temporary

// Proposal object and `proposals.push(...)`

// appends it to the end of `proposals`.

proposals.push(Proposal({

name: proposalNames[i],

voteCount: 0

}));

}

}

// Give `voter` the right to vote on this ballot.

// May only be called by `chairperson`.

function giveRightToVote(address voter) public {

// If the first argument of `require` evaluates

// to `false`, execution terminates and all

// changes to the state and to Ether balances

// are reverted.

// This used to consume all gas in old EVM versions, but

// not anymore.

// It is often a good idea to use `require` to check if

// functions are called correctly.

// As a second argument, you can also provide an

// explanation about what went wrong.

require(

msg.sender == chairperson,

"Only chairperson can give right to vote."

);

require(

!voters[voter].voted,

"The voter already voted."

);

require(voters[voter].weight == 0);

voters[voter].weight = 1;

}

/// Delegate your vote to the voter `to`.

function delegate(address to) public {

// assigns reference

Voter storage sender = voters[msg.sender];

require(!sender.voted, "You already voted.");

require(to != msg.sender, "Self-delegation is disallowed.");

// Forward the delegation as long as

// `to` also delegated.

// In general, such loops are very dangerous,

// because if they run too long, they might

// need more gas than is available in a block.

// In this case, the delegation will not be executed,

// but in other situations, such loops might

// cause a contract to get "stuck" completely.

while (voters[to].delegate != address(0)) {

to = voters[to].delegate;

// We found a loop in the delegation, not allowed.

require(to != msg.sender, "Found loop in delegation.");

}

// Since `sender` is a reference, this

// modifies `voters[msg.sender].voted`

sender.voted = true;

sender.delegate = to;

Voter storage delegate\_ = voters[to];

if (delegate\_.voted) {

// If the delegate already voted,

// directly add to the number of votes

proposals[delegate\_.vote].voteCount += sender.weight;

} else {

// If the delegate did not vote yet,

// add to her weight.

delegate\_.weight += sender.weight;

}

}

/// Give your vote (including votes delegated to you)

/// to proposal `proposals[proposal].name`.

function vote(uint proposal) public {

Voter storage sender = voters[msg.sender];

require(!sender.voted, "Already voted.");

sender.voted = true;

sender.vote = proposal;

// If `proposal` is out of the range of the array,

// this will throw automatically and revert all

// changes.

proposals[proposal].voteCount += sender.weight;

}

/// @dev Computes the winning proposal taking all

/// previous votes into account.

function winningProposal() public view

returns (uint winningProposal\_)

{

uint winningVoteCount = 0;

for (uint p = 0; p < proposals.length; p++) {

if (proposals[p].voteCount > winningVoteCount) {

winningVoteCount = proposals[p].voteCount;

winningProposal\_ = p;

}

}

}

// Calls winningProposal() function to get the index

// of the winner contained in the proposals array and then

// returns the name of the winner

function winnerName() public view

returns (bytes32 winnerName\_)

{

winnerName\_ = proposals[winningProposal()].name;

}

}

**contract** GST\_Auction {

// формирует цену реализации человеко-часов ДАО

}

/\*

GST (Gilda Secunda Token) – внутренняя условная (расчетная) единица ДАО, которая опосредует процессы (а) предварительной купли-продажи человеко-часов юридической фирмы (см. процесс Sale контракта GST\_Sale), (б) заключения основного договора (см. процесс Utilize контракта GST\_Sale), (в) распределения заказа между партнёрами (см. контракт GST\_Ballot), (г) учёта отработанного времени и полученной прибыли (см. контракт GST\_Circle) и (г) распределения прибыли, полученной по итогам выполнения задания (см. контракт Pool).

Auction – метод формирования стоимости 1 GST для всего выпуска в процессе Sale.

\*/

**contract** GST\_Sale {

// регулирует выпуск (продажу) человеко-часов ДАО

}

/\*

Sale – процесс приобретения клиентами юридической фирмы заранее эмитированных GST, свидетельствующий о заключении предварительного договора на оказание консультационных (исследовательских) услуг (работ) юридической фирмы в рамках количества человеко-часов, соразмерных количеству приобретенных GST.

Sale\_Cap – порог сбыта GST, при достижении которого контракт входит в режим Pre-pause.

Pre\_Pause – режим ограниченной функциональности контракта, когда процесс Sale приостановлен для целей согласования технического задания от клиентов и допустим лишь процесс Utilize.

Utilize – процесс возврата клиентами юридической фирмы ранее приобретенных GST для целей их обмена на соразмерное количество человеко-часов юридической фирмы для исполнения заказа, описанного в основном договоре на оказание консультационных (исследовательских) услуг (работ) юридической фирмы.

Utilize\_Cap – порог возврата GST, при достижении которого контракт входит в режим Pause.

Pause – режим минимальной функциональности контракта, когда процесс Utilize приостановлен и допустимы лишь некоторые управляющие воздействия.

\*/

/\*

Параметры GST\_Sale устанавливаются генеральным директором (см. контракт Director) по итогам голосования ДАО (см. контракт GST\_Ballot):

\*/

uint Start\_Date; // variable for Sale start

uint Stop\_Date; // variable for Sale stop

uint Sale\_Cap; // variable for Pre\_Pause regime start

uint Utilize\_Cap; // variable for Pause regime start

/\*

Стоимость 1 GST на старте устанавливается через аукцион

Передача GST на контракт GST\_Circle передается через процесс Utilize

В случае не достижения Sale\_Cap средства клиентов возвращаются, а контракт GST\_Sale возвращается в первоначальное состояние.

Граф состояний контракта GST\_Sale

To do

\*/