## Лабораторная работа №4 - Реализация распределенного приложения (dApp) для голосования

Источник - <https://www.dappuniversity.com/articles/the-ultimate-ethereum-dapp-tutorial> исправленный и адаптированный под реалии 2022 года.

## Предустановка

Установить node.js

На ОС семейства Windows- установить доп компоненты по инструкции <https://github.com/nodejs/node-gyp#on-windows> :

1. Python
2. Visual C++ Build Environment: Visual Studio Build Tools в отдельном пакете или в составе Visual Studio Community.

Ставим Truffle – npm install truffle -g

Создаем каталог для проекта:

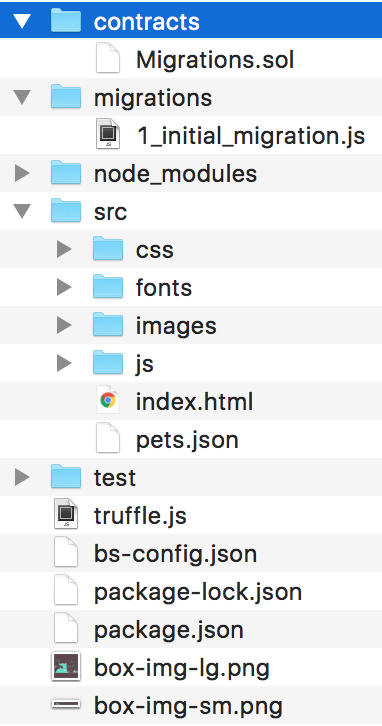
mkdir dapp

cd dapp

Инициализируем шаблон приложения, используя готовый пример из фреймворка Truffle

truffle unbox pet-shop

Это создаст базовую структуру приложения – папки для контрактов, тестов, фронтенда и т.д.



## Проверка работоспособности окружения

Создать contracts/Election.sol

pragma solidity >=0.4.2;

contract Election {

// Read/write candidate

string public candidate;

// Constructor

constructor() public {

candidate = "Candidate 1";

}

}

Создать скрипт миграции migrations/2\_deploy\_contracts.js

var Election = artifacts.require("./Election.sol");

module.exports = function(deployer) {

deployer.deploy(Election);

};

В терминале:

Накатываем миграции: truffle migrate

Запускаем JS-консоль: truffle console

Election.deployed().then(function(instance) { app = instance }) – инициализация app

app.candidate() – смотрим поле candidate. Если оно соответствует прописанному "Candidate 1" – значит окружение работает корректно, контракт установился и обращение к нему прошло.

## Базовый Контракт

Переписываем contracts/Election.sol

pragma solidity >=0.4.2;

contract Election {

// Модель данных кандидата

struct Candidate {

uint id;

string name;

uint voteCount;

}

// Хранилище конадидатов

// Получаем отсюда же, без геттеров

mapping(uint => Candidate) public candidates;

// Счетчик кандидатов

uint public candidatesCount;

constructor() public {

addCandidate("Candidate 1");

addCandidate("Candidate 2");

}

function addCandidate (string memory \_name ) private {

candidatesCount ++;

candidates[candidatesCount] = Candidate(candidatesCount, \_name, 0);

}

}

Пишем на него тест test/election.js

var Election = artifacts.require("./Election.sol");

contract("Election", function(accounts) {

var electionInstance;

it("initializes with two candidates", function() {

return Election.deployed().then(function(instance) {

return instance.candidatesCount();

}).then(function(count) {

assert.equal(count, 2);

});

});

it("it initializes the candidates with the correct values", function() {

return Election.deployed().then(function(instance) {

electionInstance = instance;

return electionInstance.candidates(1);

}).then(function(candidate) {

assert.equal(candidate[0], 1, "contains the correct id");

assert.equal(candidate[1], "Candidate 1", "contains the correct name");

assert.equal(candidate[2], 0, "contains the correct votes count");

return electionInstance.candidates(2);

}).then(function(candidate) {

assert.equal(candidate[0], 2, "contains the correct id");

assert.equal(candidate[1], "Candidate 2", "contains the correct name");

assert.equal(candidate[2], 0, "contains the correct votes count");

});

});

});

Запускаем тест: truffle test. Убеждаемся в успешном прохождении.

## Фронтенд

Правим src/index.html – базовую разметку интерфейса приложения

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>Election Results</title>

<!-- Bootstrap -->

<link href="css/bootstrap.min.css" rel="stylesheet">

</head>

<body>

<div class="container" style="width: 650px;">

<div class="row">

<div class="col-lg-12">

<h1 class="text-center">Election Results</h1>

<hr/>

<br/>

<div id="loader">

<p class="text-center">Loading...</p>

</div>

<div id="content" style="display: none;">

<table class="table">

<thead>

<tr>

<th scope="col">#</th>

<th scope="col">Name</th>

<th scope="col">Votes</th>

</tr>

</thead>

<tbody id="candidatesResults">

</tbody>

</table>

<hr/>

<p id="accountAddress" class="text-center"></p>

</div>

</div>

</div>

</div>

<!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>

<!-- Include all compiled plugins (below), or include individual files as needed -->

<script src="js/bootstrap.min.js"></script>

<script src="js/web3.min.js"></script>

<script src="js/truffle-contract.js"></script>

<script src="js/app.js"></script>

</body>

</html>

Правим src/js/app.js – файл с кодом фронтенд-приложения

App = {

web3Provider: null,

contracts: {},

account: '0x0',

init: function() {

return App.initWeb3();

},

initWeb3: function() {

if (typeof web3 !== 'undefined') {

// If a web3 instance is already provided by Meta Mask.

App.web3Provider = web3.currentProvider;

web3 = new Web3(web3.currentProvider);

} else {

// Specify default instance if no web3 instance provided

App.web3Provider = new Web3.providers.HttpProvider('http://localhost:7545');

web3 = new Web3(App.web3Provider);

}

return App.initContract();

},

initContract: function() {

$.getJSON("Election.json", function(election) {

// Instantiate a new truffle contract from the artifact

App.contracts.Election = TruffleContract(election);

// Connect provider to interact with contract

App.contracts.Election.setProvider(App.web3Provider);

return App.render();

});

},

render: function() {

var electionInstance;

var loader = $("#loader");

var content = $("#content");

loader.show();

content.hide();

// Load account data

web3.eth.getCoinbase(function(err, account) {

if (err === null) {

App.account = account;

$("#accountAddress").html("Your Account: " + account);

}

});

// Load contract data

App.contracts.Election.deployed().then(function(instance) {

electionInstance = instance;

return electionInstance.candidatesCount();

}).then(function(candidatesCount) {

var candidatesResults = $("#candidatesResults");

candidatesResults.empty();

for (var i = 1; i <= candidatesCount; i++) {

electionInstance.candidates(i).then(function(candidate) {

var id = candidate[0];

var name = candidate[1];

var voteCount = candidate[2];

// Render candidate Result

var candidateTemplate = "<tr><th>" + id + "</th><td>" + name + "</td><td>" + voteCount + "</td></tr>"

candidatesResults.append(candidateTemplate);

});

}

loader.hide();

content.show();

}).catch(function(error) {

console.warn(error);

});

}

};

$(function() {

$(window).load(function() {

App.init();

});

});

Пересобираем truffle migrate --reset

и запускаем npm run dev – открывается браузер, приложение висит на Loading... т.к. мы не залогинены в Метамаске

Настраиваем Метамаск:

1. Создаем новый кошелек
2. В настройках разрешаем отображать тестовые сети
3. Добавляем тестовую сесть по данным из Ganache и выбираем ее
4. Убеждаемся, что кошелек видит наш баланс в Ganache.

Обновляем страницу приложения и видим список кандидатов.

## Добавляем возможность голосования

Дорабатываем контракт голосования contracts/Election.sol, добавляем

// Хранилище проголосовавших аккаунтов

mapping(address => bool) public voters;

…

function vote (uint \_candidateId) public {

// Проверка, что аккаунт не голосовал еще

require(!voters[msg.sender]);

// Проверяем валидность айди кандидата

require(\_candidateId > 0 && \_candidateId <= candidatesCount);

// записываем факт голосования аккаунтом

voters[msg.sender] = true;

// Обновляем счетчик голосов кандидата

candidates[\_candidateId].voteCount ++;

Покрываем новую функциональность тестами в test/election.js

it("allows a voter to cast a vote", function() {

return Election.deployed().then(function(instance) {

electionInstance = instance;

candidateId = 1;

return electionInstance.vote(candidateId, { from: accounts[0] });

}).then(function(receipt) {

return electionInstance.voters(accounts[0]);

}).then(function(voted) {

assert(voted, "the voter was marked as voted");

return electionInstance.candidates(candidateId);

}).then(function(candidate) {

var voteCount = candidate[2];

assert.equal(voteCount, 1, "increments the candidate's vote count");

})

});

it("throws an exception for invalid candiates", function() {

return Election.deployed().then(function(instance) {

electionInstance = instance;

return electionInstance.vote(99, { from: accounts[1] })

}).then(assert.fail).catch(function(error) {

assert(error.message.indexOf('revert') >= 0, "error message must contain revert");

return electionInstance.candidates(1);

}).then(function(candidate1) {

var voteCount = candidate1[2];

assert.equal(voteCount, 1, "candidate 1 did not receive any votes");

return electionInstance.candidates(2);

}).then(function(candidate2) {

var voteCount = candidate2[2];

assert.equal(voteCount, 0, "candidate 2 did not receive any votes");

});

});

it("throws an exception for double voting", function() {

return Election.deployed().then(function(instance) {

electionInstance = instance;

candidateId = 2;

electionInstance.vote(candidateId, { from: accounts[1] });

return electionInstance.candidates(candidateId);

}).then(function(candidate) {

var voteCount = candidate[2];

assert.equal(voteCount, 1, "accepts first vote");

// Try to vote again

return electionInstance.vote(candidateId, { from: accounts[1] });

}).then(assert.fail).catch(function(error) {

});

});

it("produces correct voting results", function() {

return Election.deployed().then(function(instance) {

electionInstance = instance;

candidateId = 2;

electionInstance.vote(candidateId, { from: accounts[1] });

return electionInstance.candidates(1);

}).then(function(candidate1) {

var voteCount = candidate1[2];

assert.equal(voteCount, 1, "candidate 1 received 1 vote");

return electionInstance.candidates(2);

}).then(function(candidate2) {

var voteCount = candidate2[2];

assert.equal(voteCount, 1, "candidate 2 received 1 vote");

});

});

Пересобираем: truffle migrate --reset. Прогоняем truffle test. Убеждаемся в успешном прохождении.

Модифицируем фронтенд - в src\index.html под таблицей добавляем форму для голосования

<form onSubmit="App.castVote(); return false;">

<div class="form-group">

<label for="candidatesSelect">Select Candidate</label>

<select class="form-control" id="candidatesSelect">

</select>

</div>

<button type="submit" class="btn btn-primary">Vote</button>

<hr />

</form>

В src\js\app.js меняем render и добавляем castVote

render: function() {

var electionInstance;

var loader = $("#loader");

var content = $("#content");

loader.show();

content.hide();

// Load account data

web3.eth.getCoinbase(function(err, account) {

if (err === null) {

App.account = account;

$("#accountAddress").html("Your Account: " + account);

}

});

// Load contract data

App.contracts.Election.deployed().then(function(instance) {

electionInstance = instance;

return electionInstance.candidatesCount();

}).then(function(candidatesCount) {

var candidatesResults = $("#candidatesResults");

candidatesResults.empty();

var candidatesSelect = $('#candidatesSelect');

candidatesSelect.empty();

for (var i = 1; i <= candidatesCount; i++) {

electionInstance.candidates(i).then(function(candidate) {

var id = candidate[0];

var name = candidate[1];

var voteCount = candidate[2];

// Render candidate Result

var candidateTemplate = "<tr><th>" + id + "</th><td>" + name + "</td><td>" + voteCount + "</td></tr>"

candidatesResults.append(candidateTemplate);

// Render candidate ballot option

var candidateOption = "<option value='" + id + "' >" + name + "</ option>"

candidatesSelect.append(candidateOption);

});

}

return electionInstance.voters(App.account);

}).then(function(hasVoted) {

// Do not allow a user to vote

if(hasVoted) {

$('form').hide();

}

loader.hide();

content.show();

}).catch(function(error) {

console.warn(error);

});

},

castVote: function() {

var candidateId = $('#candidatesSelect').val();

App.contracts.Election.deployed().then(function(instance) {

return instance.vote(candidateId, { from: App.account });

}).then(function(result) {

// Wait for votes to update

$("#content").hide();

$("#loader").show();

}).catch(function(err) {

console.error(err);

});

}

};

Перезапускаем отладочный сервер - npm run dev

В браузере заходим на страницу и голосуем, возможно, придется переподключить Метамаск. Обратите внимание, что для проведения голосования необходимо провести транзакцию – т.к. мы не просто читаем, а меняем данные в блокчейне.

Пробуем зарегистрировать в Метамаске разные кошельки и, переключаясь между ними, проголосовать за разных кандидатов.

## Обработка событий смарт-контракта

Добавим в контракт contracts/Election.sol событие о голосовании, его параметром будет кандидат, за которого проголосовали.

contract Election {

// ...

event votedEvent (

uint indexed \_candidateId

);

// ...

}

а также добавим запуск этого события при голосовании:

function vote (uint \_candidateId) public {

require(!voters[msg.sender]);

require(\_candidateId > 0 && \_candidateId <= candidatesCount);

voters[msg.sender] = true;

candidates[\_candidateId].voteCount ++;

// После завершения обработки транзакции – запустим событие.

emit votedEvent(\_candidateId);

}

Выполним миграцию через truffle migrate –reset

Дописываем тест в в test/election.js:

contract("Election", function(accounts) {

var electionInstance;

it("allows a voter to cast a vote", function() {

return Election.deployed().then(function(instance) {

electionInstance = instance;

candidateId = 1;

return electionInstance.vote(candidateId, { from: accounts[2] });

}).then(function(receipt) {

assert.equal(receipt.logs.length, 1, "an event was triggered");

assert.equal(receipt.logs[0].event, "votedEvent", "the event type is correct");

assert.equal(receipt.logs[0].args.\_candidateId.toNumber(), candidateId, "the candidate id is correct");

})

});

});

и прогоняем truffle test

Добавляем на фронтенде в src\js\app.js функцию подписки на событие:

listenForEvents: function() {

App.contracts.Election.deployed().then(function(instance) {

instance.votedEvent({}, {

fromBlock: 0,

toBlock: 'latest'

}).watch(function(error, event) {

console.log("event triggered", event)

// Перерисовываем UI по приходу сообщения

App.render();

});

});

},

А также добавляем вызов этой функции App.listenForEvents(); в initContract

Перезапускаем сервер с фронтендом - npm run dev, проводим транзакции и наблюдаем за перерисовкой интерфейса.

В итоге мы получили полноценное распределенное приложение для голосования с двунаправленным пользовательским интерфейсом.