Date : ***Mai 2020*** ***TP_Blockchain*** **Responsable du cours : Yousfi Souheib**

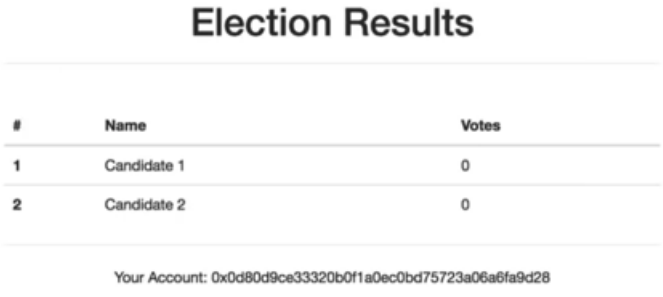
Introduction

Ce TP a pour objectif de développer une application de vote en ligne en utilisant la Blockchain [1]. Avant de commencer notre manipulation, il faut préparer l'environnement de travail. Pour ce faire, nous allons travailler sur une distribution Linux (Ubuntu). Ensuite, Installer ces dépendances sur votre machine :

1. NPM (Node Package Manager) : https ://nodejs.org (vérifiez la version avec node -v)
2. Truffle Framework : https ://github.com/trufflesuite/truffle ou bien npm install -g truffle
3. Ganache (Notre Blockchain de développement) : http ://truffleframework.com/ganache/
4. Installer le plugin Metamask Ethereum Wallet : https ://metamask.io/ avec le navigateur Chrome.
5. installer l'éditeur sublime-text, puis faites le lien symbolique pour la rapidité du lancement de votre éditeur :
   (sudo ln -s /Applications/Sublime Text.app/Contents/SharedSupport/bin/subl /usr/-local/bin/sublime)
6. Pour avoir de la couleur à votre sublime text installer Ethereum from Package Control via ce lien (https ://packagecontrol.io/installation#Manual)
7. Vérifier les versions de vos dépendances en tapant : truffle version

E-voting utilisant Ethereum

Le votant pour pouvoir voter, il utilisera une interface pareille, qui contiendra la liste des candidats ainsi que l'adresse de son compte.



Créer un répertoire Élection pour notre projet e-voting en utilisant la Blockchain Ethereum. Puis, pour faciliter la tâche de développement, installer via votre terminal sous le répertoire Élection le projet pet-shop comme suit : truffle unbox pet-shop.

1. https ://www.dappuniversity.com/articles/the-ultimate-ethereum-dapp-tutorial

Lancer le projet et familiarisez vous avec les différents répertoires installés en lançant la commande "subl . " On trouve par exemple le répertoire contracts où tous les smarts contacts seront installés. Le répertoire migration que nous utiliserons pour changer l'état de notre base de données après le déploiement de nos smarts contracts...

Créer votre smart contract qui vous permettra de lire les données à partir de la Blockchain et d'en écrire. Dans notre cas de figure, il permettra de lister les candidats qui participeront à l'élection, et gérer les votants et leur vote. Par exemple, il assurera l'exigence qu'un votant ne peut voter qu'une seule fois.

Créer votre Election.sol sous le répertoire contracts : touch contracts/Election.sol

Puis, tapez ce code.

```
pragma solidity ^0.5.12;
//we declare the smart contract with the "contract" keyword
contract Election {
    // Model a Candidate
    struct Candidate {
        uint id;
        string name;
        uint voteCount;
    }

    // Store accounts that have voted
    mapping(address => bool) public voters;
    // Read/write candidates
    mapping(uint => Candidate) public candidates;
    // Store Candidates Count
    uint public candidatesCount;

     event votedEvent (
        uint indexed _candidateId
    );
//create a constructor that will get called whenever we
   deploy the smart contract to the blockchain.
    constructor () public {
        addCandidate("Candidate 1");
        addCandidate("Candidate 2");
    }

    function addCandidate (string memory _name) private {
        candidatesCount ++;
        candidates[candidatesCount] =
           Candidate(candidatesCount, _name, 0);
    }

    function vote (uint _candidateId) public {
        // require that they haven't voted before
        require(!voters[msg.sender]);

        // require a valid candidate
```

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

        // record that voter has voted
        voters[msg.sender] = true;

        // update candidate vote Count
        candidates[_candidateId].voteCount ++;

        emit votedEvent(_candidateId);
    }
}
```

Après la création de notre smart contract, vérifions son déploiement sur la blockchain. Pour ce faire, il faut créer le fichier 2_*deploy_contracts.js* sous le répertoire migration. On précède le nom du fichier par 2 pour que truffle sache l'ordre d'exécution des fichiers : touch *migrations*/2_*deploy_contracts.js*

```
//we require the contract we've created, and assign it to a
    variable called "Election"
var Election = artifacts.require("./Election.sol");
//we add it to the manifest of deployed contracts to ensure
    that it gets deployed when we run the migrations.
module.exports = function(deployer) {
  deployer.deploy(Election);
};
```

Lancer votre BLockchain personnel Ganache avec la commande $ganache - cli$ ou bien en lançant l'application graphique ganache. Tâchez à laissant en écoute votre Blockchain.

Lancer votre migration avec la commande $truffle migrate$. Vous remarquez le déploiement de votre smart contract et le coût alloué.

Vous pouvez interagir avec votre smart contract en récupérant quelques informations, en lançant $truffle console$. Une fois le prompt : truffle(development)> vous donne la main, créez votre instance $Election.deployed().then(function(instance)\ \{app = instance\})$ que le déploiement de votre variable Election sera assigné à la variable app. Tester par exemple $app.address$, $app.candidates(1)$ ...

Sous le répertoire Test, vous allez créer votre fichier Javascript (touch *test*/*election.js*) qui simulera l'interaction du client avec notre smart contract, comme vous venez de le faire via la console truffle.

Tapez le code de election.js

```
// we require the contract Election.sol and assign it to a
    variable
var Election = artifacts.require("./Election.sol");
// we call the "contract" function, and write all our
// tests within the callback function that provides an
    "accounts" variable
// that represents all the accounts on our blockchain,
    provided by Ganache.
contract("Election", function(accounts) {
  var electionInstance;
//checking the candidates count is equal to 2
```

```
  it("initializes with two candidates", function() {
    return Election.deployed().then(function(instance) {
      return instance.candidatesCount();
    }).then(function(count) {
      assert.equal(count, 2);
    });
  });
  //ensuring that each candidate has the correct id, name,
     and vote count.

  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");
    });
  });
});
```

Remarque : Si vous avez changé le contenu de votre smart contract, tâchez à lancer cette commande truffle migrate –reset.

Lancer la commande d'exécution de votre fichier Javascript, comme suit : truffle test.

Préparer la partie du votant, en changeant l'index.html comme suit :

```
<!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>
    <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/>
              <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>
              <p id="accountAddress" class="text-center"></p>
            </div>
          </div>
        </div>
      </div>
  <script
     src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/
  jquery.min.js"></script>
      <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>
```

Puis, remplacez le contenu de app.js sous le répertoire js par le code ci dessous :

```
App = {
  // TEMP
  gotPromise: false,
  // END TEMP
```

```
  web3Provider: null,
  contracts: {},
  account: '0x0',
  hasVoted: false,
// set up web3.js : is a javascript library that allows
// our client-side application to talk to the blockchain.
// We configure web3 inside the "initWeb3" function.
  init: function() {
    return App.initWeb3();
  },

  initWeb3: function() {
    // TODO: refactor conditional
    if (typeof web3 !== 'undefined') {
      // If a web3 instance is already provided by Meta
         Mask.

      ethereum.enable().then(msg => console.log(msg ,
         "test"))
      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();
  },
//Initialize contracts: We fetch the deployed instance of
//the smart contract inside this function and assign some
// values that will allow us to interact with it.
  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);

      App.listenForEvents();

      return App.render();
    });
  },

  // Listen for events emitted from the contract
  listenForEvents: function() {
    App.contracts.Election.deployed().then(function(instance)
```

```
      {
      // Restart Chrome if you are unable to receive this
         event
      // This is a known issue with Metamask
      //
         https://github.com/MetaMask/metamask-extension/issues/2393
      instance.votedEvent({}, {
        fromBlock: 0,
        toBlock: 'latest'
      }).watch(function(error, event) {
        console.log("event triggered", event)
        // Reload when a new vote is recorded
        App.render();
      });
    });
  },
// The render function lays out all the content on the page
   with data
// from the smart contract. For now, we list the candidates
   we created
// inside the smart contract. We do this by looping through
   each candidate
// in the mapping, and rendering it to the table.
// We also fetch the current account that is connected to
   the blockchain
//inside this function and display it on the page.
  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) {
      if(! App.gotPromise ) {
        App.gotPromise = true;
```

```
        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);
            //console.log("arrived to load template")
            // 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);
      });
    }
  };

  $(function() {
    $(window).load(function() {
      App.init();
    });
  });
```

Puis, lancez votre serveur de développement comme suit : npm run dev

Automatiquement, chrome s'ouvre en affichant la page loading vu que vous n'êtes pas encore connecté à la blockchain.

Á partir de ganache, importer un compte et l'ajouter à Metamask.

Tester votre application avec plusieurs compte. Tâchez à choisir le bon candidat ;)

♣ S.Y. ♣
Bon travail