DECENTRALIZED VOTING APPLICATION USING ETHEREUM BLOCKCHAIN

Submitted in partial fulfillment of the requirements

for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE

SUPERVISOR

Prof C S Rai

Dean,USICT

SUBMITTED BY

Siddharth Lalwani

02816403213

University School of Information and Communication Technology

GGS Indraprastha University,Delhi-78

(2013-2017)

ACKNOWLEDGEMENT

I am sincerely thankful to all who have provided me with invaluable assistance during this project and have helped me to develop this project report. I take this opportunity to express my gratitude to my mentor Prof C.S Rai for the guidance and providing me with a high conceptual understanding of the project, which helped make my project extremely interesting to me and gave me the impetus to work hard. With his vast experience and knowledge, he has been able to guide me ably and successfully towards the successful completion of the project.

I also express my sincere thanks to University School of Information and Communication Technology (USICT), Guru Gobind Singh Indraprastha University for providing me this opportunity and allowing me to work on this project.

Last but not the least, I am grateful to my parents whose blessings and encouragement has provided me the strength to complete the project of this complexity.

Siddharth Lalwani

DECLARATION

I hereby declare that the project work entitled “Decentralized Voting Application using Ethereum Blockchain” submitted to USICT GGS IP University,Dwarka, is a record of an original work done by me under the guidance of Prof C S Rai, Dean USICT and this project work is submitted in the partial fulfillment of the requirements for the award of the degree of Master of Technology in Computer Science& Engineering. The results embodied in this thesis have not been submitted to any other University or Institute for the award of any degree or diploma.

Siddharth Lalwani

ABSTRACT

In 2008 a revolutionary brand new currency system was proposed in a paper called Bitcoin: Peer-to-Peer Electronic cash system. The system allows participants to transfer value within a decentralized network, instead of trusting central authority. Thus, it became obvious that more than a money, the concept that was Bitcoin built on top of, is a decentralized trust system, rather than ordinary currency, which was in its case just an application.

In the Bitcoin system, the transaction records and ownership authentications are proved and managed by chain of digital signatures within a distributed database called Block chain.

In paper-based voting, tallying is a critical process where the winner of an election is determined. When a voter inserts the completed ballot into the box, they lose sight of the ballot and have to trust election authorities to faithfully record and tally ballots.

Using Blockchain for voting can make the whole process End to End verifiable and transparent. The casted votes will be transactions, from which we can create a blockchain that will keep track of the tallies of the votes. using this approach, all the voters can agree on the result because they can count the votes themselves, and because of the blockchain audit trail, they can verify that no illegitimate votes were added, and no votes were changed or removed.

For implementing blockchain and coding the voting application,I will be using ethereum. Ethereum is smart contract platform inspired by block chain technology. The intention of Ethereum is to merge together enhanced scripting possibilities, meta protocol and time stamped database to allow development of an arbitrary application.

CONTENTS

1. Introduction …….. 1

2. Technologies Used ……. 5

   a. Ethereum
   b. Smart Contracts
   c. Accounts
   d. Merkle Patricia Trees
   e. Scripting Languages

3. Prerequisites …… 7

4. Data Variables ……. 8

5. Methods ……. 9

6. Smart Contract …….. 11

7. FrontEnd of Dapp (Meteor) …… 13

8. Using Truffle Framework …….. 17

9. Connecting Project to Ropsten Testnet …… 21

10. Interacting with FrontEnd (Function Calls) ……. 24

11. Summary…….. 26

12. Appendix

i. A: Solidity Code ……..…….. 27

ii. B : Main.js(Meteor Project) …… 29

iii. C : Truffle Files ……..…….…..... 31

13. Bibliography ……..... 35

LIST OF FIGURES

1. Figure 1: Mist Working Environment(Deploy Contract Window) 10
2. Figure 2: Solidity browser interface for the contract 12
3. Figure 3: Creating meteor project (command prompt) 13
4. Figure 4: Live meteor demo app with metamask 14
5. Figure 5: Confirmation of Metamask injected web3 15
6. Figure 6: Solidity Browser connecting to Metamask web3 15
7. Figure 7: Using Ethereum Javascript Api in console window 16
8. Figure 8: Creating Truffle demo Project (Command Prompt) 18
9. Figure 9: voteForcandidate function call using web3 Ethereum Javascript API 20
10. Figure 10: Mist Wallet Testnet Interface ` 22
11. Figure 11: Web Interface of Voting Contract 25

1. Introduction

Democratic voting is an extremely important and serious prospect for every country. In most cases, people vote through a paper based system. Digital voting is use of electronic devices, like voting machines or internet browsers, to cast votes.

The security of digital voting is the biggest concern when implementing a digital voting system. With such monumental decisions at stake, we can have no doubt about the voting system's ability to secure data and to defend against potential attacks.

Paper-based voting

In paper-based voting, tallying is a critical process where the winner of an election is determined. When a voter inserts the completed ballot into the box, they lose sight of the ballot and have to trust election authorities to faithfully record and tally ballots. But corrupted authorities may modify, miscount or exclude the voter's ballot without the voter's knowledge. The lack of assurance on the tallying integrity is one major cause for disputes in the aftermath of an election.

Modern e-voting products

In the modern digital era, e-voting products are being adopted by many countries to allow voters to cast ballots on a touch-screen direct-recording electronic machine or over the Internet. Similar as before, voters have to trust election authorities to faithfully record and tally their electronic ballots. However, as compared with tampering with physical ballots, it is much easier for a single corrupted authority to tamper with the electronic records and tally.

Using Blockchain for Voting

A blockchain is an audit trail for a database which is managed by a network of computers where no single computer is responsible for storing or maintaining the database, and any computer may enter or leave this network at any time without jeopardizing the integrity or availability of the database. Any computer can rebuild the database from scratch by downloading the blockchain and processing the audit trail.

Using Blockchain for voting can make the whole process **End to End verifiable** and transparent. The casted votes will be transactions, from which we can create a blockchain that will keep track of the tallies of the votes. using this approach, all the voters can agree on the result because they can count the votes themselves, and because of the blockchain audit trail, they can verify that no illegitimate votes were added, and no votes were changed or removed.

How Blockchain Works

Blockchain creates a network of computers called nodes and all of them store a copy of the database, and a set of rules that is the consensus protocol, to define the order in which these computers may take turns adding changes to the database. Using this method, all of the nodes can agree to the state of the database at any point of time, and no one node can falsify the data or censor changes. The blockchain further requires an audit trail of all changes to the database, which allows anyone to review that the database is correct at any point of time. This audit trail is composed of the individual changes to the database, which are called transactions. A group of transactions that were all added by a single node on its turn is called a block. Each block contains a reference to the particular block which preceded it, and this is used to establish an ordering of the blocks. This is the origin of the term “blockchain”: it is a chain of blocks, each one containing a link to the previous block and a list of new transactions since that previous block. When a new node joins the network, it starts with an empty database, and downloads all of the blocks, applying the transactions within them to the database, to fast-forward this database to the same state as all the other nodes have. In essence, a blockchain establishes the order in which transactions were applied to the database so that anyone can verify that the database is

accurate by rebuilding it from scratch and verifying that at no point was any improper change made.

This project will be dedicated to analysis of Block chain technology as a trust-free system for transparent and irreversible data exchange and storage. The principles of current and past distributed transaction ledgers will be provided to accent the huge improvement within the last years, mainly due to invention of Bitcoin, the decentralized currency system. Since that time, block chain has been adapted and modified into decentralized voting, token exchange or domain name registration systems.

The work will be introducing advantages of decentralized applications used to overcome imperfections of centralized systems such as, single point of failure and lack of transparency. More specifically, this project will describe the engine and structure behind block chain. Moreover, how society in general could benefit from its usage. The usage of Block chain technology will be demonstrated by a smart-contract application built on top of Ethereum platform.

I will develop a decentralized application for voting. The purpose of this application is to introduce, test and evaluate available functions of Ethereum platform and also Solidity language that will be used for writing the smart contract itself.

This project shall serve purpose of decentralized voting system where any or specific addresses controlled by the network peers can participate in voting on a *Poll* controlled by smart contract. The participation is only limited to sending publicly defined commands defined within the smart contract and paying the transaction cost. The commands sent in the transaction invoke particular parts of the contract's source code stored on the block chain and modifies the current state (storage) of the contract.

As the transaction is being processed by the miners, modified state is permanently stored on the block chain and is practically impossible to be

reversed, because total computing power put into the verification would have to be computed again by every single node in the network.

2. Technologies Used

Ethereum

Ethereum is smart contract platform that is inspired by block chain technology. Its elemental unit is called *ether*. Ether, similarly to bitcoin is divisible up to 10-18, its smallest subunit is called *wei*. Due to the *fee-by-computation18* policy, Ether (abbr. ETH) is sometimes referred as the fuel of Ethereum.

The intention of Ethereum is to merge together enhanced scripting possibilities, meta protocol and time stamped database to allow development of an arbitrary application. The key difference from other block chain protocols is built-in programming language, various types of accounts and unlimited variation of application that can be built on top of it.

Smart contracts

Similarly to real-life contracts in business and finance, Ethereum enables to digitalize rules on the block chain that are transparently and globally enforced by participating nodes.

Ethereum allows much broader scale of contracting system known as Distributed Autonomous Organization. This is a scheme that encodes whole organizational structure (e.g. reward system tied to specific actions) into block chain.

As an example one can take open source development, where developers are paid from funds collected by supporters. The funds are publicly and without central authority distributed to the developers by the amount of effort measured in lines of code produced and validated as meaningful by rest of the nodes.

Accounts

In Ethereum, the state is defined by the objects named “accounts”, those are 20-bytes addresses and the state transitions are the transfers between such accounts.

Example of Ethereum address: 0xc2b1918bc7a2c398ec6f20b754992d7c10d3e2cb

Account contains four elemental fields:

1) The nonce – counter ensuring each transaction is processed only once

2) Ethereum balance

3) Contract code – optional depending if account is used as contract or as a standard transactions. Contracts specify hash of the bytecode, for standard transaction is used empty string.

4) Storage – space for contract bytecode

Merkle Patricia trees

Unlikely from Bitcoin, Ethereum uses more complex hashing structure called “Merkle Patricia trees”.

Merkle Patricia are a combination of Merkle’s scheme of hashing and Patricia (Radix) tree structure, providing a tree that that has cryptographically authenticated data structure that can store key-value bind data. (Xie, 2015) It is very efficient for insert, deletes and especially lookups which are very important for integrity checks made by nodes.

Scripting languages

Ethereum platform supports various programming languages to extend developer possibilities and also not to support only one language and thus enhance advantage of choosing the right programming language for different development purpose.

Solidity is a high-level programming language with syntax similar to JavaScript.

3. Prerequisites

The application is written in Solidity language. Solidity Browser was used for writing the contract online. It has interface that is already prepared for publishing the contract after its competition. Nonetheless, the application will be published using Ethereum Mist Wallet afterwards. Solidity Browser also checks for internal integrity of source code, which is really useful as currently there are not many easily available testing environments for smart contracts.

Furthermore it was necessary to have ETH cryptocurrency as a form of payment to miners for deploying the smart contract on the network and for payment of fees for its execution. Initially, I will be running the Ethereum Mist Wallet in Test Mode because Test ETH is mined relatively quickly as compared to real E

For initial startup, Kraken was used as the exchange for buying 1 ETH.

On below address 1 ETH withdrawal request was made on Kraken.com website.

0xaEA803e3849C06EA743eD7e480B4631C4b6DA441

For testing purposes, I will be working on Blockchain Testnet and mining Test Ethers.

Mist was used for managing funds and contract related operation was needed. It also manages private keys and offers an extended interface for publishing of smart contracts and invoking its functions.

4. Data Variables

1**) votesReceived** – It maps all the votes received by the different candidate options.

```
mapping (bytes32 => uint8) public votesReceived;
```

2) **Voter** – eligible addresses allowed to vote, if not specified any address can participate in voting. Voters can have different voting weight, but for contract simplicity static value 1 is used.

```
struct Voter{

        bool voted;
        bytes32 vote;
}
```

3) **voters** – Voter array of the addresses of all those who have voted

```
mapping(address => Voter) public voters;
```

4) **candidateList** – List of all the options /candidates .

```
bytes32[] public candidateList;
```

5. Methods

1) **totalVotesFor** – If the argument passed is a valid candidate, it returns the no. of votes received for that address.

```
function totalVotesFor(bytes32 candidate) returns (uint8) {
  if (validCandidate(candidate) == false) throw;
  return votesReceived[candidate];
 }
```

2) **voteForCandidate** – Checks if eligible voters are defined, if so the address is validated for voting. Otherwise anyone can cast a vote. Furthermore, the function checks if Poll deadline is not passed. Finally, casts a vote from senders address.

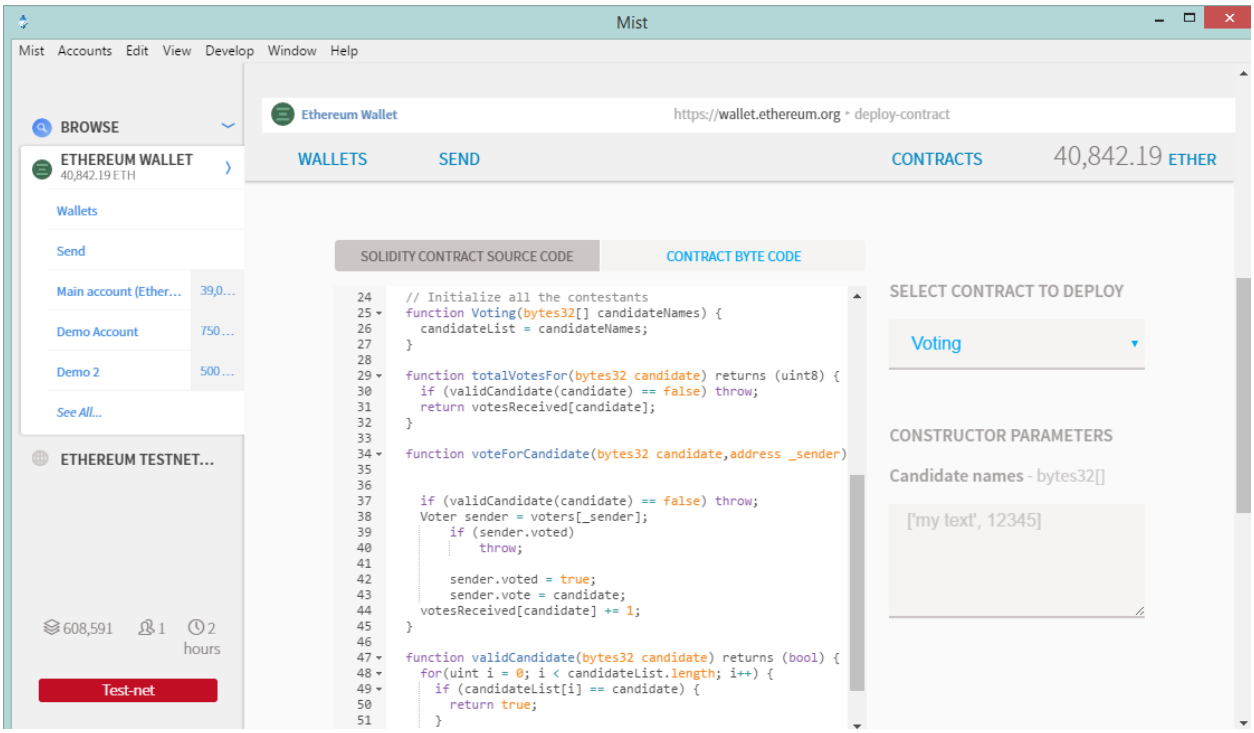
```
function voteForCandidate(bytes32 candidate) {

  if (validCandidate(candidate) == false) throw;
  Voter sender = voters[msg.sender];
    if (sender.voted)
       throw;

    sender.voted = true;
    sender.vote = candidate;
  votesReceived[candidate] += 1;
 }
```

3) **validCandidate** – Poll votes are computed and option with most casted votes is declared as a constant

```
function validCandidate(bytes32 candidate) returns (bool) {
   for(uint i = 0; i < candidateList.length; i++) {
    if (candidateList[i] == candidate) {
      return true;
    }
   }
   return false;
}
```



Mist Working Environment (Deploy Contract Window)

6. Smart Contract

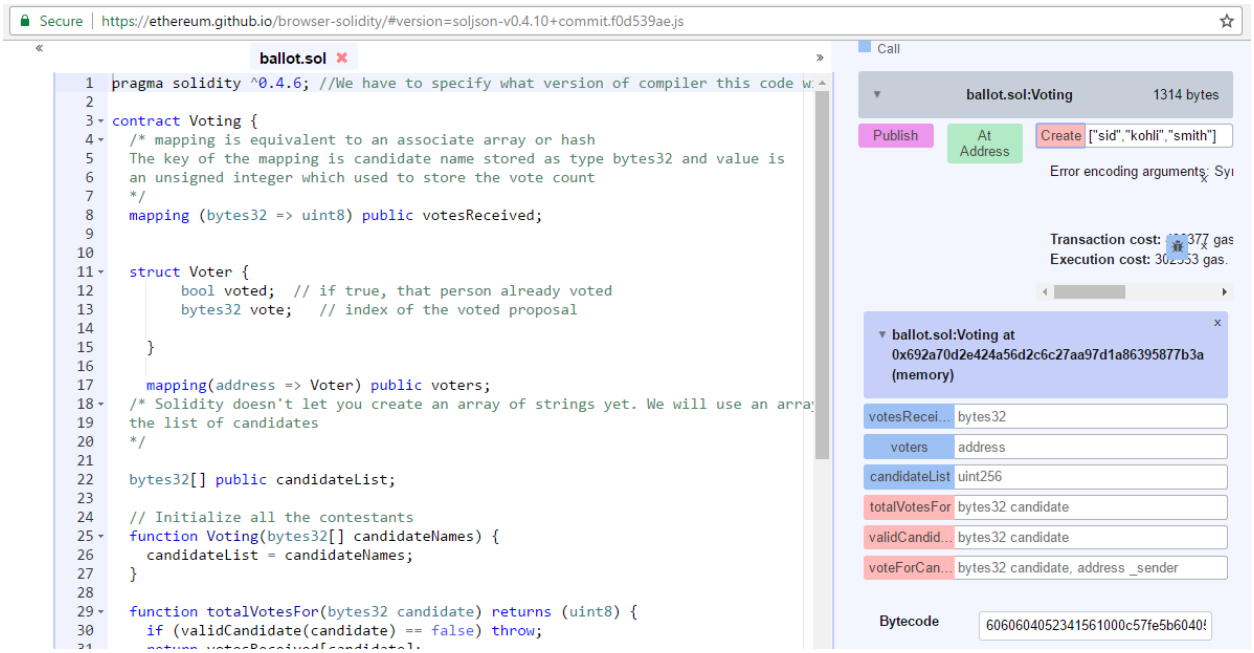
The contract was created by putting solidity source code to *New Contract* function in Mist Browser.

The application binary interface of the smart contract is

[{"constant":false,"inputs":[{"name":"candidate","type":"bytes32"}],"name":"totalVotesFor","outputs":[{"name":"","type":"uint8"}],"payable":false,"type":"function"},{"constant":false,"inputs":[{"name":"candidate","type":"bytes32"}],"name":"validCandidate","outputs":[{"name":"","type":"bool"}],"payable":false,"type":"function"},{"constant":true,"inputs":[{"name":"","type":"bytes32"}],"name":"votesReceived","outputs":[{"name":"","type":"uint8"}],"payable":false,"type":"function"},{"constant":true,"inputs":[{"name":"","type":"address"}],"name":"voters","outputs":[{"name":"voted","type":"bool"},{"name":"vote","type":"bytes32"}],"payable":false,"type":"function"},{"constant":true,"inputs":[{"name":"","type":"uint256"}],"name":"candidateList","outputs":[{"name":"","type":"bytes32"}],"payable":false,"type":"function"},{"constant":false,"inputs":[{"name":"candidate","type":"bytes32"},{"name":"_sender","type":"address"}],"name":"voteForCandidate","outputs":[],"payable":false,"type":"function"},{"inputs":[{"name":"candidateNames","type":"bytes32[]"}],"payable":false,"type":"constructor"}]

This Interface is needed for watching the contract from a different node from which it was deployed.

The code for the smart contract has been included with report in **Appendix A**.



Solidity browser interface for the contract

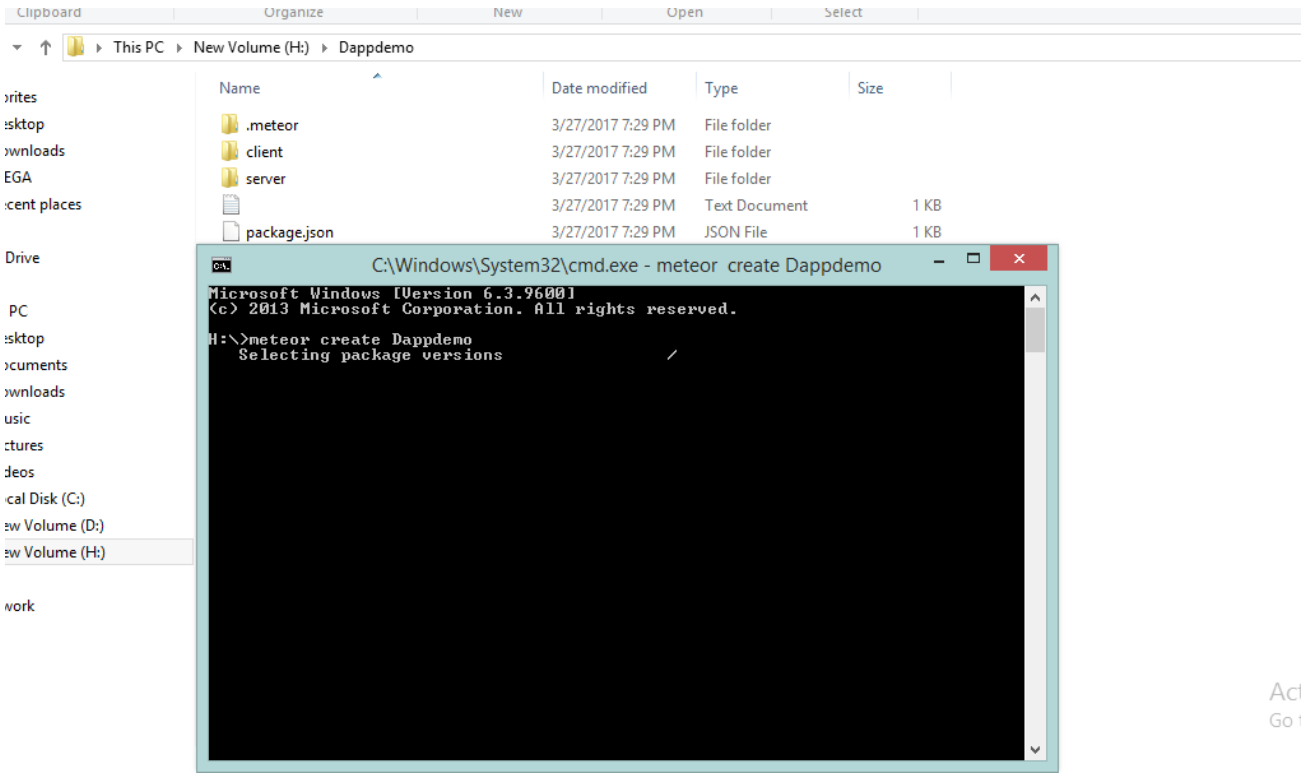
7. Frontend of Dapp

I will be using meteor framework to connect to ethereum contract using its javascript API.

Meteor is a full-stack JavaScript platform for developing modern web and mobile applications. Meteor includes a key set of technologies for building connected-client reactive applications, a build tool, and a curated set of packages from the Node.js and general JavaScript community.

To connect ethereum contract to meteor, we will create a new meteor app using

meteor create App_name

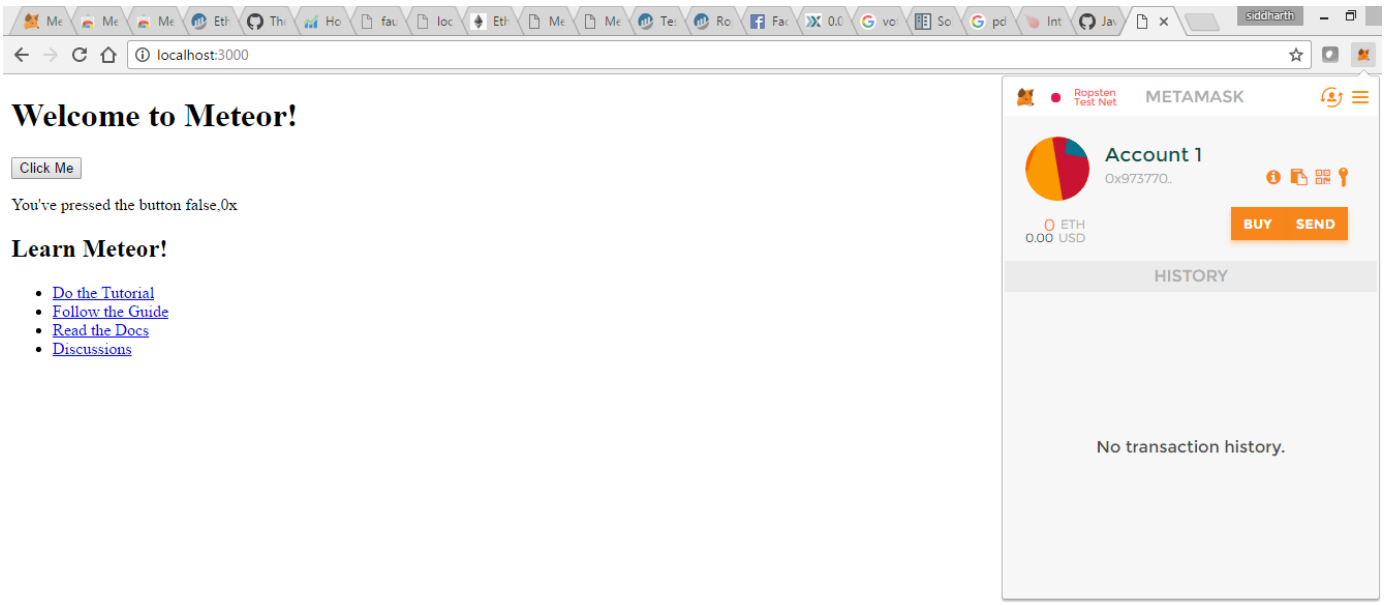


Creating meteor project (command prompt)

Then I created a folder inside the Client directory and added a lib.js file to create a web3 instance, setting a provider.

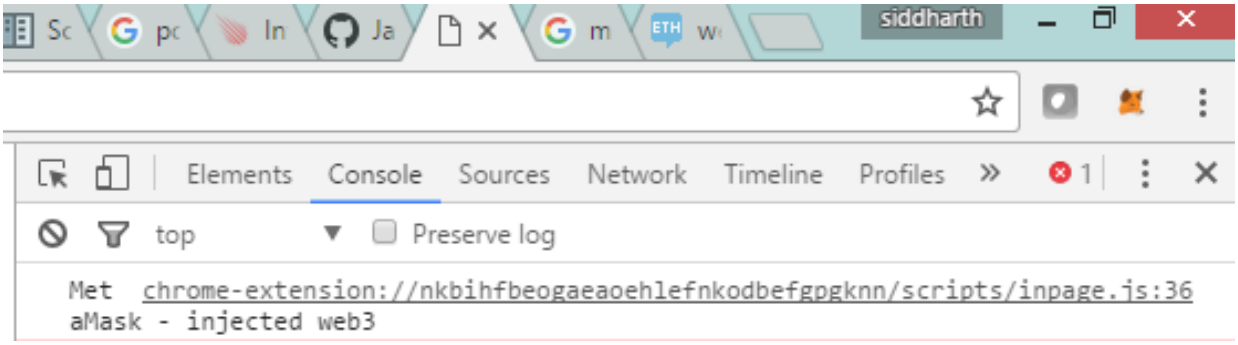
```
if (typeof web3 !== 'undefined') {
web3 = new Web3(web3.currentProvider);
}
else {
// set the provider you want from Web3.providers
web3 = new Web3(new
Web3.providers.HttpProvider("http://localhost:8545"));
}
```

Now ,to use ethereum accounts directly (without having to download the whole blockchain), we will install the Metamask chrome plugin and we will deploy our contracts on the Remix IDE by ethereum (remix.ethereum.org)



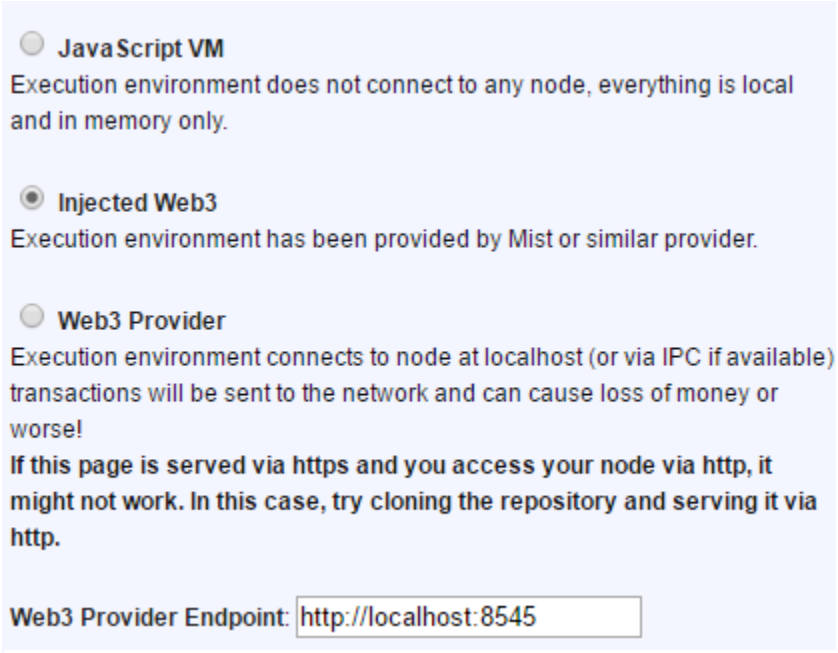
Live meteor demo app with metamask

Then, in inspect element's Console panel, Metamask injected web3 will appear.



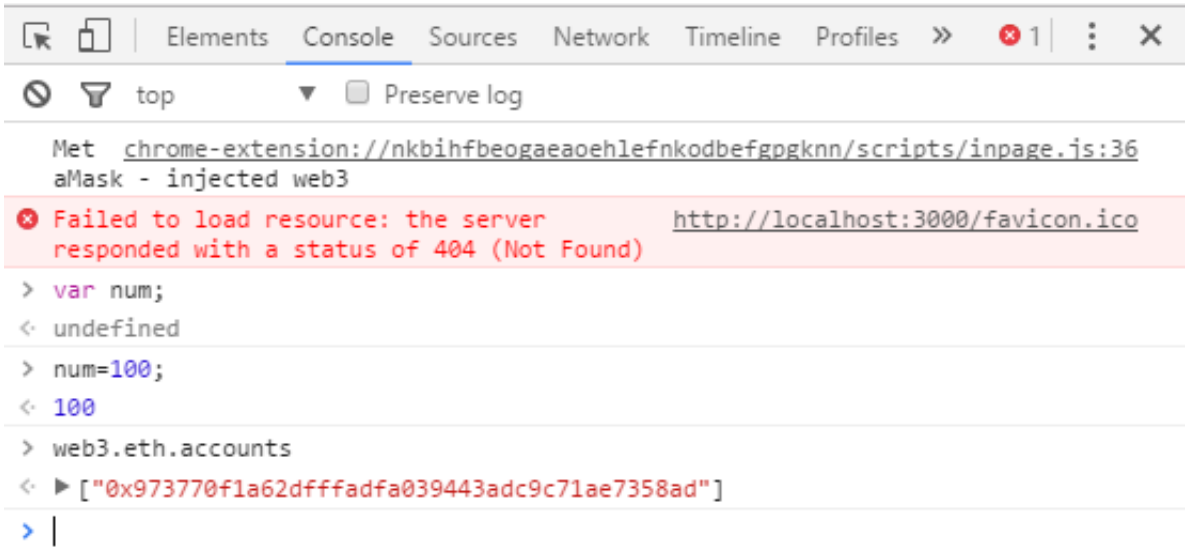
Confirmation of Metamask injected web3

If it is then Remix IDE will recognize that I am using MetaMask and it will connect to their servers.



Solidity Browser connecting to Metamask web3

Now we can directly access the data and member functions of the ethereum contract deployed on the Remix IDE using Metamask injected web3



Using Ethereum Javascript Api in console window

Using metamask and meteor, I connected to the ropsten blockchain and called functions using web3 javascript Dapp API. But Meteor's core *value proposition* is an improvement on a centralized problem: **how to sync a backend with a frontend**. It does so by providing a surrogate database on the client side that syncs with the backend as often as it can. Hence it is not suitable for developing Dapps. So, I will port my Dapp to Truffle framework.

8. Using Truffle Framework

Truffle is a development environment, testing framework and asset pipeline for Ethereum with built-in smart contract compilation, linking, deployment and binary management.

Firstly I installed geth which acts as an Ethereum node on my local machine and downloads the blockchain. There are 2 main blockchains in Ethereum- Ropsten (Testnet) and the Homestead (Mainnet). I will be again working on the Testnet.

The Ropsten Testnet has already been synced using the Mist Wallet. I will use specificy the bootnodes to connect to peers quickly using –bootnodes option

I will run geth in –rpc mode to answer the Remote Procedure calls. I will set the rpc port to 8545 and rpc address to localhost(127.0.0.1).My Geth console command is

geth --testnet --rpc --rpcapi db,eth,net,web3,personal --cache=1024 --rpcport 8545 --rpcaddr 127.0.0.1 --rpccorsdomain "*" --bootnodes "enode://20c9ad97c081d63397d7b685a412227a40e23c8bdc6688c6f37e97cfbc22d2b4d1db1510d8f61e6a8866ad7f0e17c02b14182d37ea7c3c8b9c2683aeb6b733a1@52.169.14.227:30303,enode://6ce05930c72abc632c58e2e4324f7c7ea478cec0ed4fa2528982cf34483094e9cbc9216e7aa349691242576d552a2a56aaeae426c5303ded677ce455ba1acd9d@13.84.180.240:30303" console

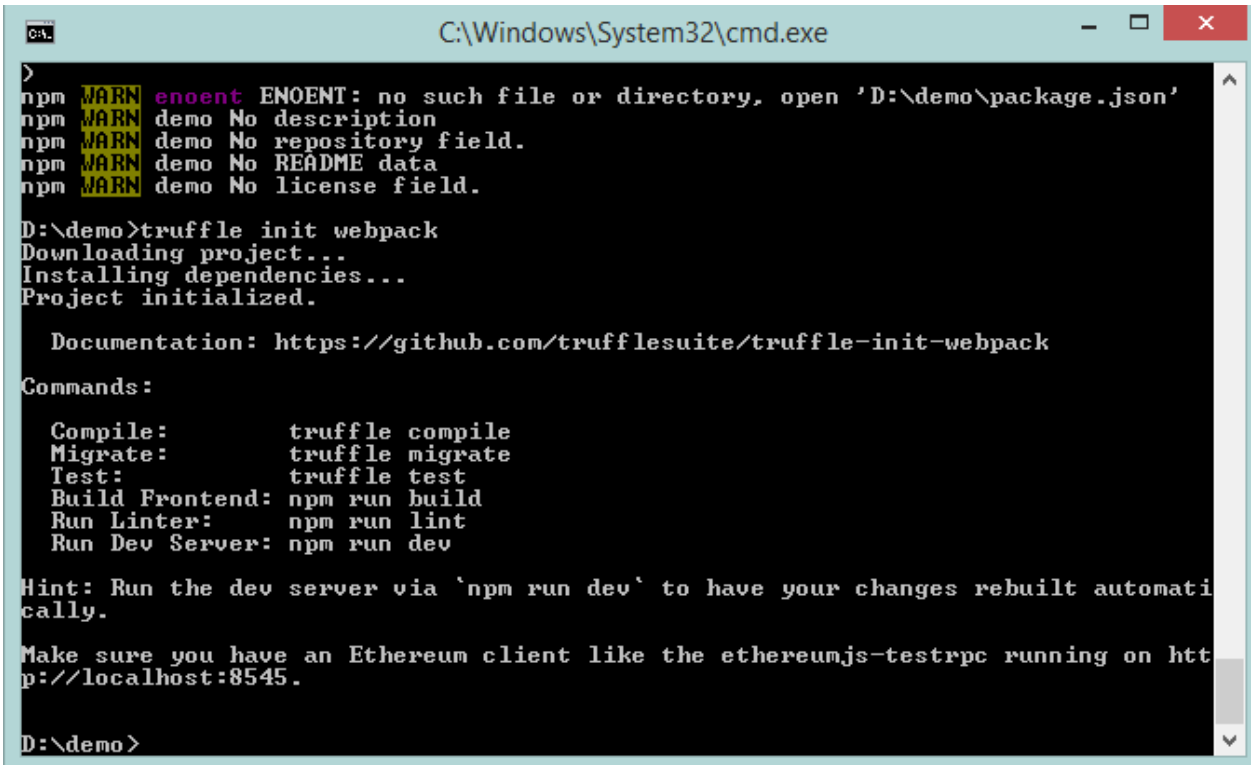
After installing truffle and starting the geth node in rpc mode, I set up the Truffle Project using the following commands->

mkdir voting

cd voting

npm install –g webpack

truffle init webpack



Creating Truffle demo Project (Command Prompt)

Truffle creates the necessary files and directories required to run a full stack dapp. Truffle also creates a sample application to get you started.

The structure of a Truffle Project is ->

```
dir


README.md                   contracts                   node_modules
test                        webpack.config.js           truffle.js
app                         migrations                  package.json


dir app/

index.html  javascripts  stylesheets
```

```
dir contracts/

ConvertLib.sol  MetaCoin.sol  Migrations.sol

dir migrations/

1_initial_migration.js  2_deploy_contracts.js
```

Migration files deploy the contracts to the blockchain. The first migration file,that is, '1_initial_migration.js' is used to deploy a contract named Migrations to the blockchain and is used to store the latest contract you have deployed. Every time we want to run the migration, truffle will automaticaly query the blockchain to find the contract that has been deployed last and if there are any contracts which haven't been deployed yet,it deploys them . After that,it updates the last_completed_migration field in the Migrations contract to indicate the latest contract deployed. You can simply think of it as a database table called Migration with a column named last_completed_migration which is kept up to date always.

Now I will add the solidity code file Voting.sol to Contracts folder.After that, I will replace the contents of 2_deploy_contracts.js in the migrations directory with->

```
var Voting = artifacts.require("./Voting.sol");
module.exports = function(deployer) {
deployer.deploy(Voting, ['Sid', 'Kohli', 'Smith'], {gas: 390000});
```

Here I will first create an artifact with the address of Voting.sol and then call the deployer function to deploy the Voting Contract. I will pass the name of 3 candidates to vote for and along with it,I will specify the gas (energy) to be used to deploy this contract.

After that I will replace the contents of app/javascipt/app.js and app/index.html with the frontend files. The code for voteForcandidate function call using web3 Ethereum Javascript API is

```
window.voteForCandidate = function(candidate,_sender) {
  let candidateName = $("#candidate").val();
  let senderAddress = $("#_sender").val();
  try {
    $("#msg").html("Vote has been submitted. The vote count will increment as soon as the vote is recorded on the blockchain. Please wait.")
    $("#candidate").val("");
    $("#_sender").val();


    Voting.deployed().then(function(contractInstance) {
      contractInstance.voteForCandidate(candidateName,senderAddress, {gas: 340000, from: web3.eth.accounts[0]}).then(function() {
        let div_id = candidates[candidateName];
        return contractInstance.totalVotesFor.call(candidateName).then(function(v) {
          $("#" + div_id).html(v.toString());
          $("#msg").html("");
        });
      });
    });
  } catch (err) {
    console.log(err);
  }
}
```

voteForcandidate function call using web3 Ethereum Javascript API

9. Connecting Project to Ropsten TestNet

Before I can deploy the contract to Blockchain, I will need an account with test ether in them to provide the energy for transactions. While setting up with the testrpc, I got 10 test accounts which came preloaded with over 100 ethers in them. But for test ether we need to create an account and add ether to it ourselves. We can do it in 2 ways. One is to ask the ropsten faucet for ether. Another way is to mine test ether using our geth node.

In our console with our truffle project as directory,I will use the following commands

truffle console

truffle(default)> **web3.personal.newAccount('**passphrase**')**
'0xeea9e979d86d9c32d1d2ab5ace2dcc8d1b446fa1'

This creates an account on Ropsten Testnet with address 0xeea9e979d86d9c32d1d2ab5ace2dcc8d1b446fa1 .

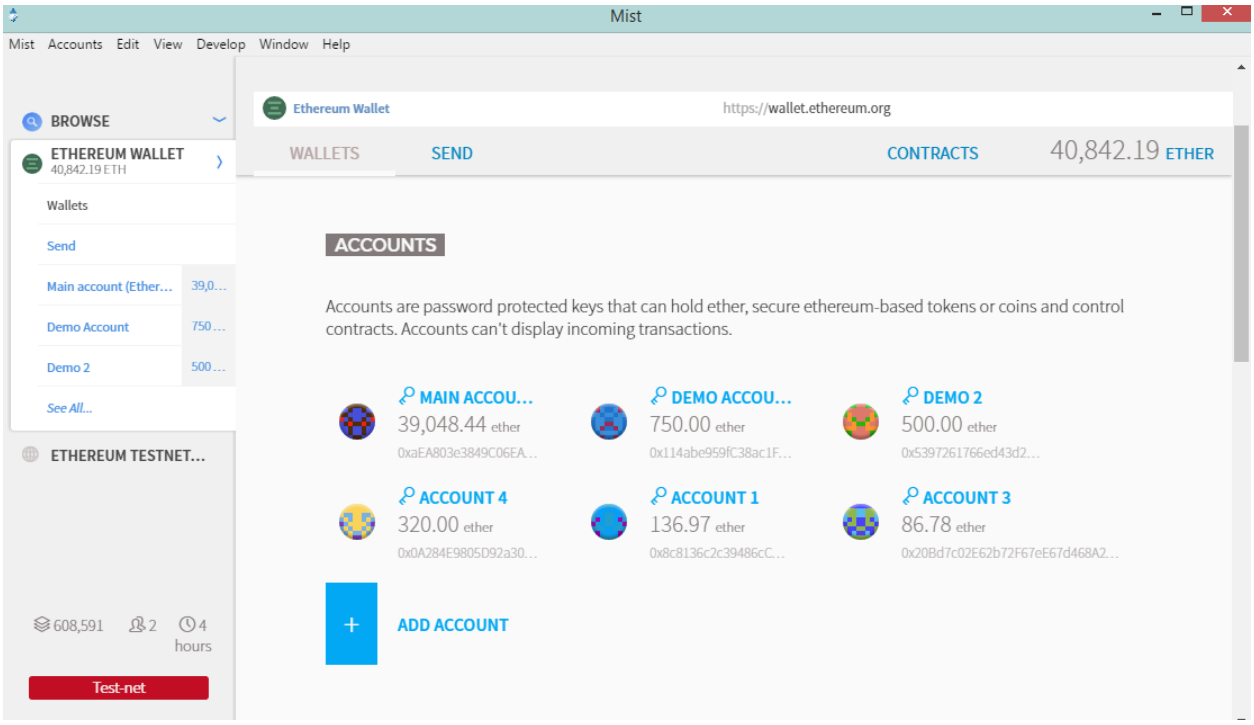
Then, I will unlock this account to perform transactions using my Truffle Project.

truffle(default)>
web3.eth.getBalance('0xeea9e979d86d9c32d1d2ab5ace2dcc8d1b446fa1')

{ [String: '0'] s: 1, e: 0, c: [0] }

truffle(default)>
web3.personal.unlockAccount('0xeea9e979d86d9c32d1d2ab5ace2dcc8d1b446fa1', 'passphrase**', 15000)**

To get test ether,I mined using my Mist Wallet. Currently I have over 40,000 ether across my accounts.



Mist Wallet Testnet Interface

Now that I have test ether,I can compile and deploy my contract to the blockchain

To deploy all the solidity contracts of my truffle project, I will use the command truffle migrate

After running the command successfully,I will get

truffle migrate

Compiling Migrations.sol...

Compiling Voting.sol...

Writing artifacts to ./build/contracts

Running migration: 1_initial_migration.js

Deploying Migrations...

Migrations: 0x3cee101c94aaa06d549334372181bc5a7b3a8bee

Saving successful migration to network...

Saving artifacts...

Running migration: 2_deploy_contracts.js

Deploying Voting...

Voting: 0xd24a32f0eeaaa5e9d233a2ebab5a53d4d4986203

Saving successful migration to network...

Saving artifacts...

10. Interacting with FrontEnd (Function Calls)

After deploying the contract successfully, I am now able to both fetch the vote count and also vote using the truffle console.

truffle console

truffle(default)> **Voting.deployed().then(function(contractInstance) {contractInstance.voteForCandidate('Kohli',' 0xaEA803e3849C06EA743eD7e480B4631C4b6DA441').then(function(v) {console.log(v)})})**

After a few seconds, I got a transaction receipt like this:

receipt:

{ blockHash:
'0x7229f668db0ac33478d0c4c86e0394a35dd081a1095b8fafb52ebd7671433156',

blockNumber: 469448,

contractAddress: null,
....
....

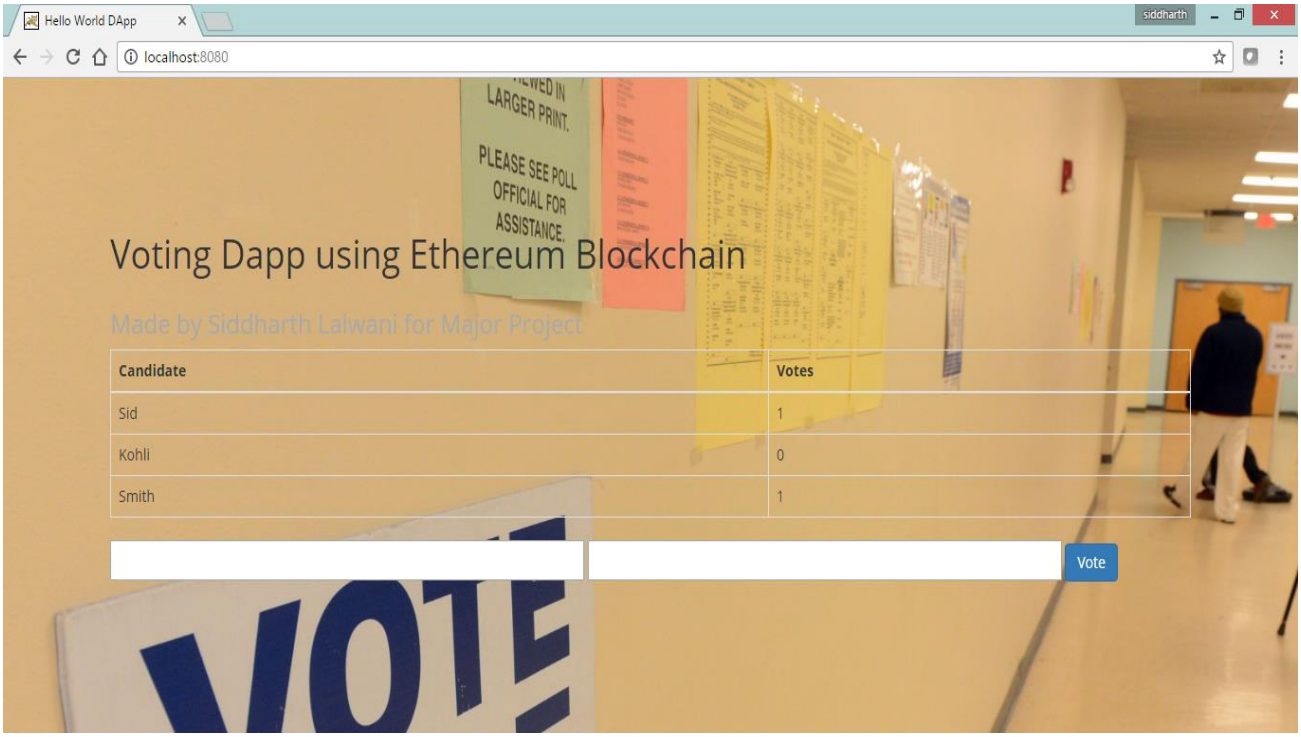
truffle(default)> **Voting.deployed().then(function(contractInstance) {contractInstance.totalVotesFor.call('Kohli').then(function(v) {console.log(v)})})**

{ [String: '1'] s: 1, e: 0, c: [1] }

The contract is now live and functional. Then, I started the server using command

npm run dev

The voting page appears on localhost:8080 and I am able to vote and see the vote counts of all candidates. Since we are dealing with the real blockchain, every transaction will take a few seconds to process.



Web Interface of Voting Contract

11. Summary

Currently, there is a huge obstacle for using Ethereum platform, which lays in difficulty of obtaining ETH units. The units can be obtained either by mining or purchased for fiat or cryptocurrency like Bitcoin. Once ETH units are available, the publishing and execution of smart contract, thanks to tools like Mist is really smooth.

At this stage, the Solidity code has been deployed and verified. The deployed contract is accessed with the help of Metamask injected web3.Following the port from Meteor to Truffle Framework,I have developed a web application user interface for the Voting Application. For increased security,we can add login features using some social platform and further secure it against vulnerabilities listed under “OWASP Top 10 vulnerabilities of Web Application 2013” which include Sql Injection, Cross Site Scripting etc.

This project lays great importance because various countries have already started making use of blockchain in important prospects like Electoral voting. The usage of block chain technology is currently being implemented throughout European state parliaments. Ukraine parliament decided to use block chain based voting system as a transparent mechanism for passing government proposals. Similar solution is being tested in Estonian Stock Market.

Appendix A

Solidity File for Voting contract

```
pragma solidity ^0.4.6;

contract Voting {
  mapping (bytes32 => uint8) public votesReceived;



  struct Voter {
       bool voted;
       bytes32 vote;
   }

    mapping(address => Voter) public voters;



  bytes32[] public candidateList;

    function Voting(bytes32[] candidateNames) {
    candidateList = candidateNames;
  }

  function totalVotesFor(bytes32 candidate) returns (uint8) {
    if (validCandidate(candidate) == false) throw;
    return votesReceived[candidate];
  }
```

```
  function voteForCandidate(bytes32 candidate,address _sender) {


    if (validCandidate(candidate) == false) throw;
    Voter sender = voters[_sender];
        if (sender.voted)
            throw;

        sender.voted = true;
        sender.vote = candidate;
    votesReceived[candidate] += 1;
  }

  function validCand  ( bytes32 cand ) returns ( bool ) {
    for ( uint i = 0 ; i < candidateList.length; i++) {
      if ( candidateList[i] == cand )
 {
        return true;
  }
  }
    return false;
  }
}
```

Appendix B

Main.js (Meteor Project)

```
import { Template } from 'meteor/templating';
import { ReactiveVar } from 'meteor/reactive-var';

import './main.html';

contractAddress = "0x7950fe2ecce2c82572aad6c4b38bf409e253254d"

ABIArray =
[{"constant":false,"inputs":[{"name":"candidate","type":"bytes32"}],"name":"totalVotesFor","outputs":[{"name":"","type":"uint8"}],"payable":false,"type":"function"},{"constant":false,"inputs":[{"name":"candidate","type":"bytes32"}],"name":"validCandidate","outputs":[{"name":"","type":"bool"}],"payable":false,"type":"function"},{"constant":true,"inputs":[{"name":"","type":"bytes32"}],"name":"votesReceived","outputs":[{"name":"","type":"uint8"}],"payable":false,"type":"function"},{"constant":true,"inputs":[{"name":"","type":"address"}],"name":"voters","outputs":[{"name":"voted","type":"bool"},{"name":"vote","type":"bytes32"}],"payable":false,"type":"function"},{"constant":true,"inputs":[{"name":"","type":"uint256"}],"name":"candidateList","outputs":[{"name":"","type":"bytes32"}],"payable":false,"type":"function"},{"constant":false,"inputs":[{"name":"candidate","type":"bytes32"}],"name":"voteForCandidate","outputs":[],"payable":false,"type":"function"},{"inputs":[{"name":"candidateNames","type":"bytes32[]"}],"payable":false,"type":"constructor"}]

Template.hello.onCreated(function helloOnCreated() {
  // counter starts at 0
  this.counter = new ReactiveVar(0);
});

Template.hello.helpers({
  counter() {
    var template = Template.instance();
```

```
    myContract = web3.eth.contract(ABIArray).at(contractAddress);



    myContract.totalVotesFor.call("ram",function(err,res)
    {
       TemplateVar.set( template, "counter", res)
    })


  },
});


Template.hello.events({
  'click button'(event, instance) {
    // increment the counter when button is clicked
    instance.counter.set(instance.counter.get() + 1);
  },
});
$( document ).ready(function() {
  if (typeof web3 !== 'undefined') {
    console.warn("Using web3 detected from external source like Metamask")
    // Use Mist/MetaMask's provider
    window.web3 = new Web3(web3.currentProvider);
  } else {
    console.warn("No web3 detected. Falling back to http://localhost:8545. You should remove this fallback when you deploy live, as it's inherently insecure. Consider switching to Metamask for development. More info here: http://truffleframework.com/tutorials/truffle-and-metamask");
    // fallback - use your fallback strategy (local node / hosted node + in-dapp id mgmt / fail)
    window.web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));
  }
```

Appendix C

Truffle Files

app.js

```
import "../stylesheets/app.css";
import { default as Web3} from 'web3';
import { default as contract } from 'truffle-contract'
import voting_artifacts from '../../build/contracts/Voting.json'

var Voting = contract(voting_artifacts);

let candidates = {"Sid": "candidate-1", "Kohli": "candidate-2", "Smith": "candidate-3"}

window.voteForCandidate = function(candidate,_sender) {
  let candidateName = $("#candidate").val();
  let senderAddress = $("#_sender").val();
  try {
    $("#msg").html("Vote has been submitted. The vote count will increment as soon as the vote is recorded on the blockchain. Please wait.")
    $("#candidate").val("");
    $("#_sender").val();


    Voting.deployed().then(function(contractInstance) {
```

```
    contractInstance.voteForCandidate(candidateName,senderAddress, {gas: 340000, from:
web3.eth.accounts[0]}).then(function() {
      let div_id = candidates[candidateName];
      return contractInstance.totalVotesFor.call(candidateName).then(function(v) {
        $("#" + div_id).html(v.toString());
        $("#msg").html("");
      });
    });
  });
 } catch (err) {
  console.log(err);
 }
}
```

index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Hello World DApp</title>
  <link href='https://fonts.googleapis.com/css?family=Open+Sans:400,700' rel='stylesheet' type='text/css'>
  <link href='https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css' rel='stylesheet' type='text/css'>
</head>
<body class="container" >
  <h1>Voting Dapp using Ethereum Blockchain</h1>
  <h3>Made by Siddharth Lalwani for Major Project</h3>
  <div id="address"></div>
  <div class="table-responsive">
   <table class="table table-bordered">
     <thead>
       <tr>
         <th>Candidate</th>
         <th>Votes</th>
       </tr>
     </thead>
     <tbody>
       <tr>
         <td>Sid</td>
         <td id="candidate-1"></td>
```

```
      </tr>
      <tr>
        <td>Kohli</td>
        <td id="candidate-2"></td>
      </tr>
      <tr>
        <td>Smith</td>
        <td id="candidate-3"></td>
      </tr>
    </tbody>
  </table>
  <div id="msg"></div>
 </div>
 <input type="text" id="candidate" />
 <input type="text" id="_sender" />
 <a href="#" onclick="voteForCandidate()" class="btn btn-primary">Vote</a>
</body>
<script src="https://cdn.rawgit.com/ethereum/web3.js/develop/dist/web3.js"></script>
<script src="https://code.jquery.com/jquery-3.1.1.slim.min.js"></script>
<script src="app.js"></script>
</html>
```

12. Bibliography

[1] *Ethereum Development - Github*. (2015).[Online] Available : https://github.com/ethereum/wiki/wiki/Ethereum-Development-Tutorial

[2] Ethereum GmbH. (2015). *Ethereum Frontier Guide.* [Online] Gitbooks.io: Available : https://www.gitbook.com/book/ethereum/frontier-guide/details

[3] ETHEREUM SWITZERLAND GMBH. (2015, 8 28). *Developer documentation.* Retrieved from ETHEREUM.org: [Online] Available : https://github.com/ethereum/wiki/wiki

[4] ETHEREUM SWITZERLAND GMBH. (2016). *Go-ethereum developer documentation*. [Online] Available: https://github.com/ethereum/go-ethereum/wiki

[5] Schiener, D. (2015, 10 28). PublicVotes: Ethereum-based Voting Application. [Online] Medium.com Available: https://medium.com/@DomSchiener/publicvotes-ethereum-based-voting-application-3b691488b926#.sqrsm4uvu

[6] Mahesh Murthy, (Feb 14,2016). Full Stack Hello World Dapp Tutorial [Online] Available: https://medium.com/@mvmurthy/full-stack-hello-world-voting-ethereum-dapp-tutorial-part-3-331c2712c9df