

# **Peer to Peer Systems and Blockchains**

**Academic Year 2020/2021**

Final Project

## **The attempt of a democratic election system**

*24 June 2021*

*Renato Eschini matricola: 203021  
email: 20302111@studenti.unipi.it*

# Indice

1	Introduction.....	3
2	Part 1: The structure of the project.....	3
2.1	The environment and start development.....	3
2.2	The project structure.....	6
2.2.1	The smart contract.....	6
2.2.2	The frontend.....	7
3	Part 2: Main decisions and considerations.....	9
4	Part 3: The demo.....	11
5	Conclusion.....	13
6	Attachments.....	13
7	References.....	13

# 1 Introduction

This document contains the report about the final project "*The attempt of a democratic election system*" of the course "*Peer to Peer Systems and Blockchains*".

It's divided into three parts:

- in the first part "*The structure of the project*", there is a the description of the structure of the project, and a little user manual with the instructions to set and test it;
- in the second part "*Main decisions and considerations*", there are some consideration about the design and implementation of the smart contracts;
- in the third part "*The demo*", there are the instructions to execute a demo of the project.

I choose to implement the first extra proposal "*Join my side, I give you cookies*" from Phyllis.

## 2 Part 1: The structure of the project

### 2.1 The environment and start development

As a development environment I used Visual Studio Code (with Solidty plugin), on Ubuntu 21.04, with Truffle, Ganache-GUI and Metamask (as Chrome Plugin).

To create the project I started from the box "Pet -shop" from Truffle (<https://www.trufflesuite.com/boxes/pet-shop>), so:

- `$ truffle unbox pet-shop`
- `$ mv pet-shop democratic-election-system`

in this way we can create a project structure saving a lot of startup time

Now, with NodeJS up and running on my PC, I installed the environment to develop Smart Contract in the following way:

- `$ npm install -g truffle` # Truffle :)
- `$ npm install truffle-assertions` # assertions library for Truffle
- `$ npm install --save-dev eth-gas-reporter` # an extension of Mocha framework for beautiful report about gas consumption, see reference for link

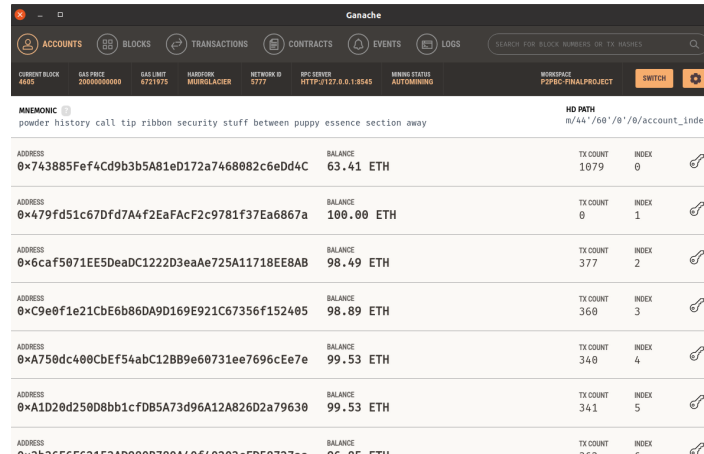
Follow the Dapp&Smartcontract lessons from the course, I installed the development webserver "*lite-server*" and other dependencies that we can found in "*package.json*", with the command:

- `$ npm install`

we can install in a one operation all of the missing libraries and dependencies.

Next, I imported, as a starting point, the old smartcontract from the final term "SayonaraMyMayor" and its related unit tests. I tested that everything worked (with Ganache up&running) and then I started with the developments, first with smartcontract (and related tests of course!) than with the frontend.

For the development&test I used Ganache-GUI for simulate my personal blockchain, so I have download from its site ganache-2.5.4-linux-x86\_64.AplImage and put it in my `${HOME}/bin` for auto-include in `${PATH}` environment in bash linux console. I create a workspace “*P2PBC-FINALPROJECT*” in Ganache, configure with my truffle project and with a lot of account for test.

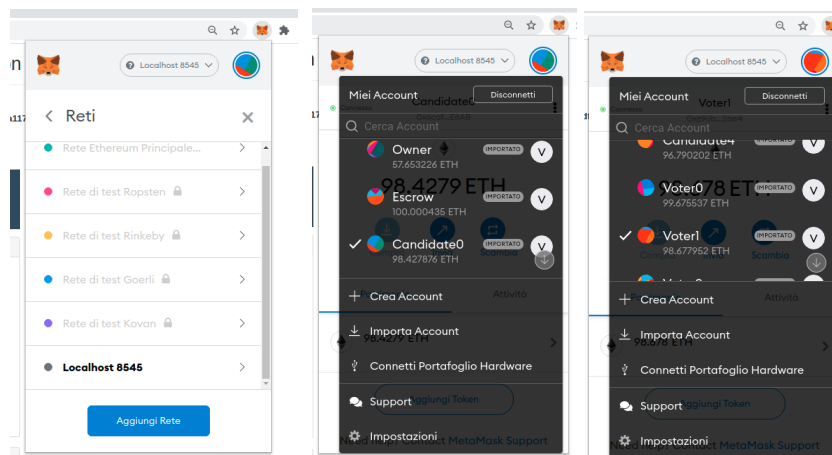


ADDRESS	BALANCE	TX COUNT	INDEX
0x743885Fef4Cd9b3b5A81eD172a7468882c6eDd4C	63.41 ETH	1079	0
0x479fd51c67Dfd7A4f2EaFACf2c9781f37Ea6867a	100.00 ETH	0	1
0x6caf5071EE5DeaDC1222D3eaAe725A11718EE8AB	98.49 ETH	377	2
0xC9e0f1e21CbE6b86DA9D169E921C67356f152405	98.89 ETH	360	3
0xA750dc408CbEf54abC128B9e0731ee7696cEe7e	99.53 ETH	340	4
0xA1D20d250D8bb1cfD85A73d96A12A826D2a79630	99.53 ETH	341	5
0x4565555555555555555555555555555555555555	99.53 ETH	341	6

Last, but not least, I setup my Metamask wallet. I had already installed the Metamask Chrome extension and a related account, I imported through its keys, 12 Ganache accounts in Metamask.

I imported 12 because, to avoid confusion with the tests and have a greater cleanliness (and save time of course!) I have imported, renamed and kept separate an account for different user roles, i.e.:

- the first account for the Owner of contract, the account that deploy the contract on blockchain (Ganache);
- the second is the Escrow;
- next five (from 3 to 7) are the candidate (I have simulate 5 candidate);
- next five (from 8 to 12) are the citizens (I have simulate 5 voters);



For recreate the environment, you just need to unzip the xxx.zip in attachment or clone from my public Github repository at [git@github.com:reny77/democratic-election-system.git](https://github.com/reny77/democratic-election-system) so:

- \$ unzip democratic-election-system.zip

or

- \$ git clone [git@github.com:reny77/democratic-election-system.git](https://github.com/reny77/democratic-election-system) # (if you have ssh key in Github, or use https, or download...)

**Remember:** Ganache up&running and Metamask set up and running in browser.

After this operations, you can:

- `$ cd democratic-election-system`
- `$ npm install`

now, you have environment installed.

So, you can try:

- `$ truffle migrate --reset --network development`
- `$ truffle test`

```
machin@tari:~/repository/u/p2pbc/finalproject/democratic-election-system$ truffle test
Using network 'development'.

Compiling your contracts...
=====
> Everything is up to date, there is nothing to compile.

Contract: Testing DemocraticMayor
  ✓ Test constructor ok (17ms, 3171743 gas)
  ✓ Test cast envelop (165ms, 3224767 gas)
  ✓ Test open_envelope without reaching quorum (427ms, 3224767 gas)
  ✓ Try open_envelope with quorum achievement (392ms, 4226701 gas)
  ✓ Test candidate deposit soul (255ms, 3395628 gas)
  ✓ Test non-candidate deposit soul (140ms, 3194288 gas)
  ✓ Test call mayor_or_sayonara without opening the envelopes (123ms, 3195978 gas)
  ✓ Testing mayor_or_sayonara: NewMayor is candidate by souls (771ms, 4914862 gas)
  ✓ Testing mayor_or_sayonara: NewMayor is candidate by votes, soul tied with candidate 2 (767ms, 4951593 gas)
  ✓ Testing mayor_or_sayonara: no NewMayor but DrawMayor, there is a tie (soul and votes) with candidates 1 and 2 (661ms, 4839814 gas)
  ✓ Test call mayor_or_sayonara two time (489ms, 4844478 gas)
  ✓ Test view functions (515ms, 4852610 gas)

  Methods
  |-----|
  | Contract | Method | Min | Max | Avg | # calls | eur (avg) |
  |-----|-----|-----|-----|-----|-----|-----|
  | DemocraticMayor | add_deposit | - | - | 44777 | 26 | - |
  | DemocraticMayor | cast_envelope | 53012 | 53024 | 53022 | 30 | - |
  | DemocraticMayor | mayor_or_sayonara | 193279 | 305983 | 298644 | 7 | - |
  | DemocraticMayor | open_envelope | 145978 | 201970 | 174308 | 27 | - |
  |-----|-----|-----|-----|-----|-----|-----|
  | Deployments | | | | | | % of limit |
  |-----|-----|-----|-----|-----|-----|-----|
  | DemocraticMayor | - | - | - | 3171743 | - | 47.2 % |
  |-----|-----|-----|-----|-----|-----|-----|

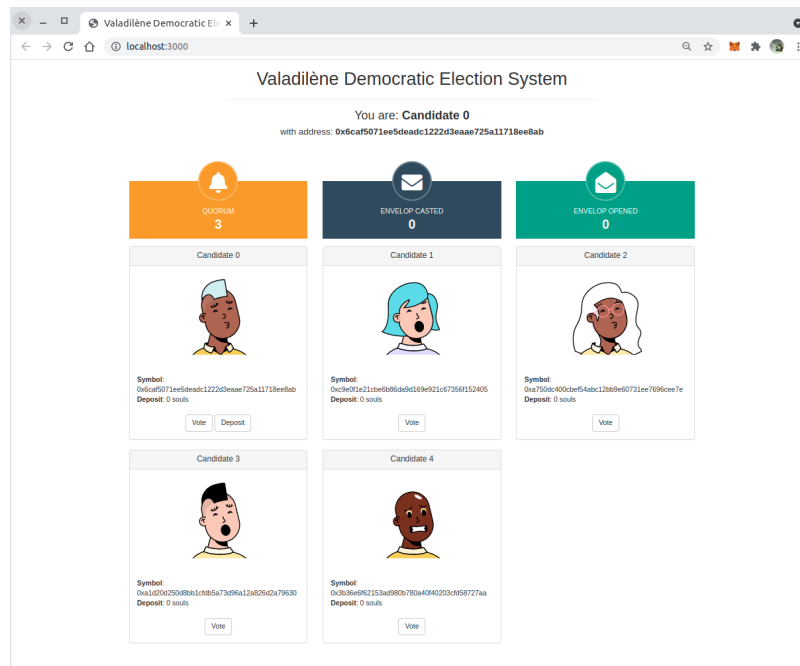
12 passing (14s)
machin@tari:~/repository/u/p2pbc/finalproject/democratic-election-system$
```

and for Dapp webapplication:

- `$ npm run dev`

the application should now accessible via browser at:

- **localhost: 3000**



The last shrewdness used, always with a view to cleaning, I created a small spreadsheet with "Libreoffice Calc" (test.ods), to track the "voting intentions" and then manually simulate the frontend interface tests, see in the §4.

## 2.2 The project structure

In this chapter I present the structure of the project in its two parts:

- the smart contract
- the frontend

In the root of project, the main folders and files are:

- bs-config.json: *the configuration of lite server;*
- contracts: *contains the smart contract;*
- migrations: *contains the javascript for migrate the smartcontract in blockchain;*
- node\_modules: *contains the necessary javascript modules;*
- package.json: *the configuration of node project and dependencies;*
- src: *the sources of frontend;*
- test: *unit tests in Javascript;*
- truffle-config.js: *truffle config;*

The code is fully commented on in the various application parts, the descriptions of the most important points are given below in the relative sections.

### 2.2.1 The smart contract

In the root of project, related to smartcontract, we can find:

- /contracts/DemocraticMayor.sol: *the source of smart contract;*
- /test/0\_DemocraticMayorTest.js: *the unit tests of smart contract;*
- /build/contracts/DemocraticMayor.json: *the compiled smart contract;*
- /migrations/1\_initial\_democratic\_mayor.js: *the initial migrate of smart contract in blockchain;*
- /truffle-config.js: *the config of truffle environment;*

I describe the most important file for this part of smartcontract.

#### *DemocraticMayor.sol*

These are the key points, in brackets the line number:

- [#7] The owner (election master) address, setted in constructor;
- [#26] The candidates are rappresented by Candidate structure that contains symbol (the address), personal soul deposited, the soul from voters, the number of voters and the list of voters (address);
- from [#40] to [#72] there are modifiers, in particular: in canCheckOutcome is also check the owner (election master), the checkCandidate is used for check that add\_deposit is called from a candidate;
- from [#74] to [#93] there are the state attribute, in particular: the candidate list (address list), the mapping (by address key) of Candidate structure, two list for winners by vote and by soul, are list because we can have a draw;
- at [#100] there is the the constructor with candidates list, the escrow and quorum as constructor parameters. There is also a requirement, that candidates and quorum must be at least 3;

- at [#134] `add_deposit` is used only (checked by modifier) by candidates for send crypto as souls deposit;
- at [#142] `cast_envelope` is the same of `finalterm`, and at [#158] `open_envelope` has been modified for register votes in candidate mapping;
- at [#186] there is `mayor_or_sayonara`, I kept the same name even the use is different, instead of declaring the confirmation or kicking the mayor, in this case it is used to evaluate the winner or the draw.
  - There is a flag, that mark result checked, for avoid "reentrancy";
  - [#192]: check winner by souls, fill a smartcontract attribute list `winner_list_by_souls` with candidates with more souls;
  - [#210]: if the `winner_list_by_souls` is  $> 1$ , there is a draw by souls, so count the votes and put the candidates with more votes in `winner_list_by_votes`;
  - [#229]: the list are evaluated for check if there is a winner by souls, by votes or if there is a draw, an event `NewMayor` od `DrawMayor` is emitted, with args the address of winner or the list of candidates who draw; are called function for pay electors of for handle the draw;
- [#243]: `pay_electors_and_winner` implements the flow for pay winner and for distribute souls of winner candidate to his electors (the first proposal);
- [#268]: `no_winners`: send all souls to escrow;
- [#285]: `compute_envelope` is the same of `finalterm`;
- from [#290] to the end there are some simple but useful, for GUI, view functions;

### [1\\_initial\\_democratic\\_mayor.js](#)

Used for deploy contract in blockchain. Instantiet the smartcontract by call the constructor. There is the configuration of quorum, escrow and there is the creation of the list of candidates.

In particular:

- the first account is used for owner;
- the second for escrow;
- the following , configured, are used as candidates; are pushed to a list and passed to constructor;

### [0\\_DemocraticMayorTest.js](#)

Contains unit test to verify the correctness of the functions.

## 2.2.2 The frontend

In the root of `src`, related to the frontend, we can find:

- `/css/`: **contains `bootstrap.min.css`, `bootstrap.min.css.map`, the stylesheet of dapp;**
- `/index.html`: **the `html-onepage dapp`;**
- `/fonts`: **contains some fonts (from `Pet-shop`);**
- `/js/web3.min.js`: **the `Web3` library for interact with smart contract;**
- `/js/truffle-contract.js`: **ethereum contract abstraction, for better work with smart contract;**
- `/js/bootstrap.min.js`: **bootstrap javascript;**
- `/js/app.js`: **the `Dapp` javascript**

I describe the most important file for this part of smartcontract.

### [index.html](#)

This is the html one page app inherited from the "Pet-shop" application. Graphically, in fact, it is very similar.

It uses bootstrap and jquery. There is a simple html template, used to make the candidate box, is taken and repeated via javascript for each candidate.

There are various buttons that are enabled and disabled via javascript based on the status of the smartcontract.

User interaction is handled by bootstrap modals declared, hidden, at the bottom of the html file and activate by buttons.

### [app.js](#)

This file contains the Dapp javascript code with which it interacts with the smartcontract.

These are the key points, in brackets the line number:

- at [#1] there is the declaration of App "object";
- from [#3] to [#9] there are the state attributes (current account, candidates list and latestblockNumber used for avoid the continue reading of last event (see below), the configuration of URL for web3;
- from [#9] to [#56] there are the init functions
  - at [#44] there is the trick for save the last block number, used for avoid the continue reading the last event after e reload (<https://ethereum.stackexchange.com/questions/57803/solidity-event-logs>)
- at [#59] there is code for subscribe to event: when add a deposit, vote or open envelope, the GUI freeze with a block modal and waiting for event; when event arrives hide the modal and call App.render for reload the new state;
- from [#115] to [#214] render function interact with smartcontract for read state and display the data on the screen;
  - display the role of user and the status of elections by read condition in the smartcontract;
  - read the list of candidates, iterate and create the boxes; enable/disable buttons for deposit, vote and open envelope;
  - display avatar by this service: <https://avatars.dicebear.com>
- from [#215] to [#265] there function for the interaction between the GUI and the smart contract:
  - depositClick
  - voteClick
  - openEnvelopeClick
  - mayorOrSayonaraClick
  - seeResults
- at [#269] there is the listener waiting for change account on Metamask, when account change, this call render for redesign the GUI;
- from [#275] to [#298] there are functions for handle the modal;
- at the end there is the first call, to initialize the App.



### 3 Part 2: Main decisions and considerations

In this chapter, I describe the main decisions and arguments I made on some aspects of the project, both on the smartcontract and the frontend.

#### *The boilerplate*

The first decision was about: how do I start?

I tried first through Dapp Tutorial (from lesson), I set up the environment and tested everything (from Ganache to Metamask), then I tried Truffle's "pet-shop" box. At the end I decided to use the Pet-shop startup because it allowed you to start right from scratch with a DAPP structure (avoiding to copy a project by hand) and then take some ideas from the tutorial of the lesson.

#### *Update finalterm smartcontract "SayonaraMyMayor"*

As suggested in the text of the project, I modified the "SayonaraMyMayor" smartcontract of the final term by generalizing it to support, through dedicated internal structures, the list of candidates, the functions to calculate winners and draws, and the view functions necessary for the DAPP.

Rebuilding the entire smart contract from scratch was not the most productive solution.

#### *Use of Ganache*

As a local blockchain simulator for the developments I used Ganache. I created a workspace, I hooked up the project (truffle-config.js), I decided the number of accounts that could be useful for development, tests and demos.

#### *Use of Metamask*

I used Metamask, connecting it to the local blockchain on Ganache and importing the accounts into Metamask (by Ganache keys), I also gave it a name on Metamask to better recognize who I was working with during development and demo.

#### *Init params of the constructors*

I asked myself: how do I add candidates? How do they deposit the soul for the implementation of the "first proposal"?

Since this information are relating to the creation of an election instance, I thought of passing the candidates as a list of blockchain accounts / addresses directly to the constructor together with the quorum and the escrow address in the "migration" phase on the blockchain. The alternative was to create a dedicated "addCandidate" function but I didn't like this idea.

The deposit of "soul" instead comes through a dedicated function by sending crypto from the candidate's account, so in addition to the smartcontract, there will be a dedicated function in the frontend GUI.

#### *User roles*

I have thought a lot about defining user roles in the smart contract and DAPP.

At the end, since it is an educational project, I decided to experiment, introduce and manage 3 user roles:

1. the **owner** (election master): deploys the contract and calculates the result;
2. **candidates**: the avatar is visible on the GUI and can deposit the soul (proposal 1)
3. the others citizens, the **voters**; obviously also the candidates and the owner can vote, but always to experiment, I put the voter accounts in Metamask also to organize the demo.

Lastly, not considered a user role, is the **escrow** account.

Why did I have doubts? Because since the election is based on blockchain, the concept of "master" (the owner who can take the check of result) sounds bad to me in a decentralized context where the important thing is the absence of a central control government (I am for the Ethereum Classic fork :)).

The alternative was to give everyone the opportunity to check the result by first verifying that all the envelopes have been opened, otherwise returning an error.

### *Gas usage and cost*

I have tried to optimize as much as possible (considering the short time I have available because I work full time), the functions that perform calculations and data storage to avoid excessive gas consumption. Something can certainly be improved (for example the function that calculates winners and draws), then there are the view functions that return state or do small checks that should not consume anything.

### *Consideration about the flow of frontend and function of smartcontract*

I have introduced several controls, through modifiers, in the smartcontract to prevent operations or following flows that are not allowed (some changed from the final term "SayonaraMyMayor").

I also introduced some view functions used by the Dapp to make the GUI consistent with the progress of the election.

As a basic scheme I used:

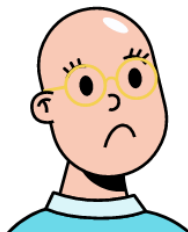
1. init functions for instance of smartcontract in the javascript environment running on the browser;
2. the creation of listeners waiting for the events issued by the smartcontract; the GUI waits (blocked) for the arrival of events, for example against a deposit, and when the event arrives, the rendering is carried out in order to update the status of the GUI (see the new deposits, the number of casted envelope, the number of open envelopes, the result of the election ...);
3. the use of simple buttons and modals for user interaction;
4. a main render function for redesign the GUI and display new state;
5. I used a listener to recognize the change of account on Metamask, make the new render of the interface so as to see the linked account;
6. I do not store the status of anything concerning the election on the browser or in javascript, but all the status is taken from the smartcontract so all the state is committed in transactions on the blockchain and taken through view functions;
7. working on the events I realized that web3 reads, after a page reload, the last published event. I don't know the implementation but I imagined that the event is published at the top of some topics queue on Ganache / Blockchain. This obviously created problems for me to keep the GUI updated with the status of the election on the smartcontract. Searching on Google I found the solution (it seems a known problem): the number of the last mined block is stored in a variable and, from time to time when a new events arrives, the number of the related block is checked that is greater than the last one. If it is, it handles the event and updates the number of the last block, otherwise skip  
(source: <https://ethereum.stackexchange.com/questions/57803/solidity-event-logs>).

8. The main script creates an “App structure” (borrowed from Pet-shop) in which there are all the variables and functions to interact with the GUI.

#### *Consideration about the frontend technologies*

For simplicity, I preferred not to use complex frameworks such as ReactJs or similar, but to use minimal javascript, jquery and bootstrap functions.

For the avatars I used this nice web service <https://avatars.dicebear.com>, I pass a string (the candidate's address) and he generates an avatar different from the others based on the string passed (ie: <https://avatars.dicebear.com/api/micah/0x6caf5071EE5DeaDC1222D3eaAe725A11718EE8AB.svg>).



## 4 Part 3: The demo

The demo is organized like this:

### **step 1**

verify that everything is configured correctly as described in §2.1

### **step 2**

use this table for track the deposit for Candidate:

candidate	deposit
candidate0	5000000000000
candidate1	6000000000000
candidate2	7000000000000
candidate3	8000000000000
candidate4	9000000000000

Choose the first account in Metamask for first candidate, you can see the GUI re-rendered and the deposit button enable In first candidate. Click on deposit and inser the value reported in the table. After the vote is mined by blockchain, the GUI will'be re-render.

Repete (remember to change account in Metamask) for all candidate.

### step 3

use a table for “remember” the vote, for example you can use this:

	Sigil	Vote candidate	Soul
<b>Voter0</b>	100	Candidate 1	1000000000000
<b>Voter1</b>	200	Candidate 2	2000000000000
<b>Voter2</b>	300	Candidate 2	3000000000000
<b>Voter3</b>	400	Candidate 2	4000000000000
<b>Voter4</b>	500	Candidate 3	5000000000000

choose in Metamask the first Voter0, click on button vote of the Candidate 1, inser sigil and soul as reported in table. After the vote mined by blockchain, the GUI will’be re-render.

Repete for every voter with the params reported in table.

You can see the status on top updated about casted envelop.

### step 3

When the casted enveloped are equals to quorum, the voter can open the enevelopes.

Connect with Metamask to first voter, click on “open envelop” button corresponding of its vote, put the information of vote from test table above.

Repete for each voter (remember to change account in Metamask).

You can see the status on top updated about open envelop.

### step 4

When all envelop are opened, all buttons disapper. Now, if you connect by Metamask the Owner, you can see on top the button “Compute winnner”.

Click on it and you can see now a new button: “See result”.

If you click on it you can see the winner or the candidates who tied.

You can repeate and simulate winner by soul, winner by votes and draw. Remeber to construct a test table for remember the votes.

## 5 Conclusion

Working on the Dapp was very fun and challenging, especially in managing the interaction between the GUI and the smartcontract. Event handling was not trivial initially, but after understanding the behavior, the use of listeners was very clean and elegant in the interaction.

Despite the little time I could devote to development it was quite easy, in the end I decided to give a cut to the functions and behaviors otherwise the application would have grown enormously :).

In any case, I feel satisfied with the result achieved.

## 6 Attachments

1. Finalproject-p2pbc-eschini-2021.pdf: this report
2. democratic-election-system.zip: contains the Project

## 7 References

1. My Github repo <https://github.com/reny77/democratic-election-system>
2. Truffle <https://www.trufflesuite.com/truffle>
3. Truffle Pet-shop box <https://www.trufflesuite.com/boxes/pet-shop>
4. Ganache <https://www.trufflesuite.com/ganache>
5. Metamask <https://metamask.io>
6. Gas Reporter for Mocha in Truffle <https://github.com/cgewecke/eth-gas-reporter>
7. Solidity docs <https://docs.soliditylang.org/en/v0.8.1>
8. <https://ethereum.stackexchange.com>
9. Lesson of "P2P and Blockchain" Unipi course 20/21:
  - "Lesson 17: PROGRAMMING SMART CONTRACTS: SOLIDITY" 28-04-21-Solidity.pdf
  - "Ethereum Smart contracts development With Javascript (2021)" Lab-1-Development.pdf
  - "Ethereum Smart contracts development With Javascript (2021)" Lab - 2 - Truffle.pdf
  - "Ethereum DAPP development With Javascript (2021)" Lab - 3 - DAPP.pdf