Peer to Peer Systems and Blockchains

Fausto F. Frasca 559482

2020/21

Contents

1 Introduction

The smart contract of the final term has been extended to have more functions. The first one is the possibility to have more than only one candidate in the same elections and the voters can choose one of them. The second one is the possibility to have coalitions of candidates. In fact has been implemented the extra *The Pammerellum*, proposed by Pamela. Thanks to that, a candidate can be in at most one coalition. About the DApp, the main choice has been the use of javascript framework *React* for develop the front-end. Together with React, has been used the framework *Ant Design* for the components like card, header and icons. Others tools used are Ganache for the network, Web3.js, truffle-contract and Metamask to interact with the instance of the smart contract both back-end and front-end side. As server for the DApp web page it has been used lite-server. More details about the main decisions both the front-end and back-end are explained in the following chapters.

2 Smart Contract

2.1 Mayor.sol

The base of the contract is the same of the contract used in the Final Term, but some functions and attributes have been changed or added to allow to have both the coalitions and multiple candidates.

The variables state added are:

- *candidates_addrs*: an array containing all the addresses of the individual candidates.
- *candidates*: a mapping from address to Candidate. For each candidate, it contains the soul and the number of voters. It is present also a flag called *init* that it is used to check if a candidate has been registered to the elections or not. Thank to it, the checks are more faster than checking the presence of a candidate in *candidates_addrs*.
- *coalitions_addrs*: an array containing all the addresses of the coalitions.
- *coalitions*: a mapping from address to Coalition. For each coalition, they are archived the soul and all the members of each coalition. As in candidate, it is present the *init* flag.
- *voters_addrs*: an array containing all the addresses of the voters.
- *voters*: a mapping from address to Voter. For each voter are stored the soul, the address of the candidate voted (can be the address of the single candidate or the coalition) and if he/she has already opened him/her envelope.
- *winner*: the address of the winner of the elections. If there isn't a winner, its value is 0.

- *coalition_winner*: used during the election of the winning coalition. It has type *Coalition_winner* and contains the address of the winner and the soul that the winner has received. If there isn't a coalition that won the elections, the value of attribute *addr* is 0.
- *candidate_winner*: it is similar to coalition_winner, but for the election of the individual candidate. Its type is *Candidate_winner* and it contains the address of the winner and the soul and the number of votes that he/she has received. If there isn't a winner, the value of the attribute *addr* is 0.

The functions changed respect to the initial smart contract are:

- *constructor*
- *open_envelope*
- *mayor_or_sayonara*

While the added functions are marked as *external view* and they are added for a better integration with the DApp. The functions are:

- *get_candidate_addrs*
- *get_coalitions*
- *get_voting_condition*

2.1.1 Constructor

The input values to create an instance of the smart contract are:

- *_candidates*: this is an array of type *Register_candidate*. Each element is composed by two attributes:
  - *candidate_address*: the address of the candidate.
  - *coalition_address*: the address of the coalition to which the candidate belongs. If the candidate does not belong to any coalition, the value of this attribute is 0.
- *_escrow*: this is the address to which will be send the tokens if there wont be a winner.
- *_quorum*: the number of voters required to declare valid the elections.

For each element in _candidates, is required that the address of the candidate does not belong to *coalitions_addrs*, the candidate has not yet been registered to the elections, this also ensure that a candidate is at most in one coalition, and the address of the coalition is not present in *candidates_addrs*. To perform these checks, it is used the flag *init* discussed before.

2.1.2 Open_envelope

This function takes as input the sigil and the address of a candidate. First of all, the requires are performed to ensure that the voter has casted him/her envelope and he/she didn't yet opened it. Immediately after is checked if the envelope sent is the same to the one casted. If all the checks are successful, the algorithm to find the winner is performed. If the candidate voted is a coalition and it has received more soul than the actual winning coalition, the attributes in *coalition_winner* are updated. Instead if the soul are equal, the address of the actual winning coalition is set to 0, but not the soul. In fact the addr equal to 0 and the soul different from 0 indicate that there is a tie case. The attributes of the coalition_winner can be updated also in case of tie if there is a coalition with more soul. If the candidate voted is not a coalition, the procedure is similar but for the individual candidate. The *candidate_winner* attributes are update if the soul of the actual candidate are greater than the actual winning candidate or the soul are equal but the number of voters of the actual candidate are more than the winning candidate. In case of a tie, the addr attribute in candidate_winner is set to 0.

This approach allows to have an algorithm that is constant respect to the number of candidates, and this is very important in an environment like the blockchain and the smart contracts, where you pay each single step. This means that the algorithm scale up very well.

2.1.3 Mayor_or_sayonara

In this function the reentracy is handled with the boolean flag *elections_over*. When this function is called, this means that all the envelopes have been opened and the attributes of *coalition_winner* and *candidate_winner* are filled. So, if the addr attribute in coalition_winner is not 0 and the soul received are greater than 1/3 of the total soul, the new mayor is the coalition. Instead if one of the two conditions is not satisfied and the addr attribute in candidate_winner is different from 0, the new mayor is the single candidate. If also the addr in candidate_winner is 0, then there will not be a new mayor.

About the soul, if there is a winner, all the voters that they have voted for a losing candidate, both coalition or single candidate, they receive their soul back and the winner receives the soul due. If there is not a winner all the soul are sent to the escrow account.

2.1.4 External views functions

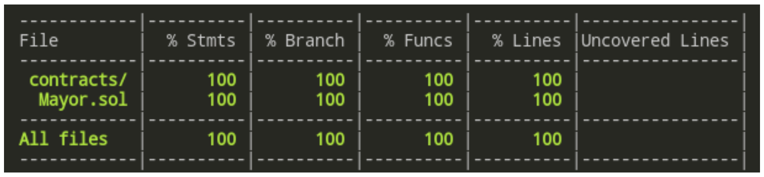
The functions described in this section are used by the DApp for a better integration between front and back end and a better user experience as described in the DApp chapter.

- *get_candidate_addrs*: it returns the array with all the addresses of the individual candidates.

- *get_coalitions*: it returns an array of type *Return_coalition*. Each element contains the address of the coalition and a list with the addresses of all the members of that coalition.
- *get_voting_condition*: it return a triple containing the quorum, the number of envelopes casted and the number of envelopes opened.

2.2 Code coverage

In the test folder there are all the tests for the smart contract that ensure a coverage of the 100% for the functions, statements, lines of code and branch, as shown in the Figure 1. To calculate the percentage of the coverage it has been used *solidity-coverage*, a plugin for truffle. For run the coverage's tests, you can type the the following command in the root directory of the project: `npm run coverage`.



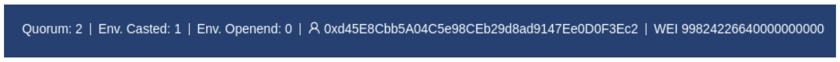
```
------------|----------|----------|----------|----------|----------------|
File        |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
------------|----------|----------|----------|----------|----------------|
 contracts/ |      100 |      100 |      100 |      100 |                |
  Mayor.sol |      100 |      100 |      100 |      100 |                |
------------|----------|----------|----------|----------|----------------|
All files   |      100 |      100 |      100 |      100 |                |
------------|----------|----------|----------|----------|----------------|
```

Figure 1: Full coverage for the code of the smart contract.

3 DApp

To develope the front-end, as mentioned before, they have been used React for the structure and Ant Design for the components. The UI has been designed to help the user to know in real time the state of the smart contract on the blockchain and to guide him/her through the vote process.

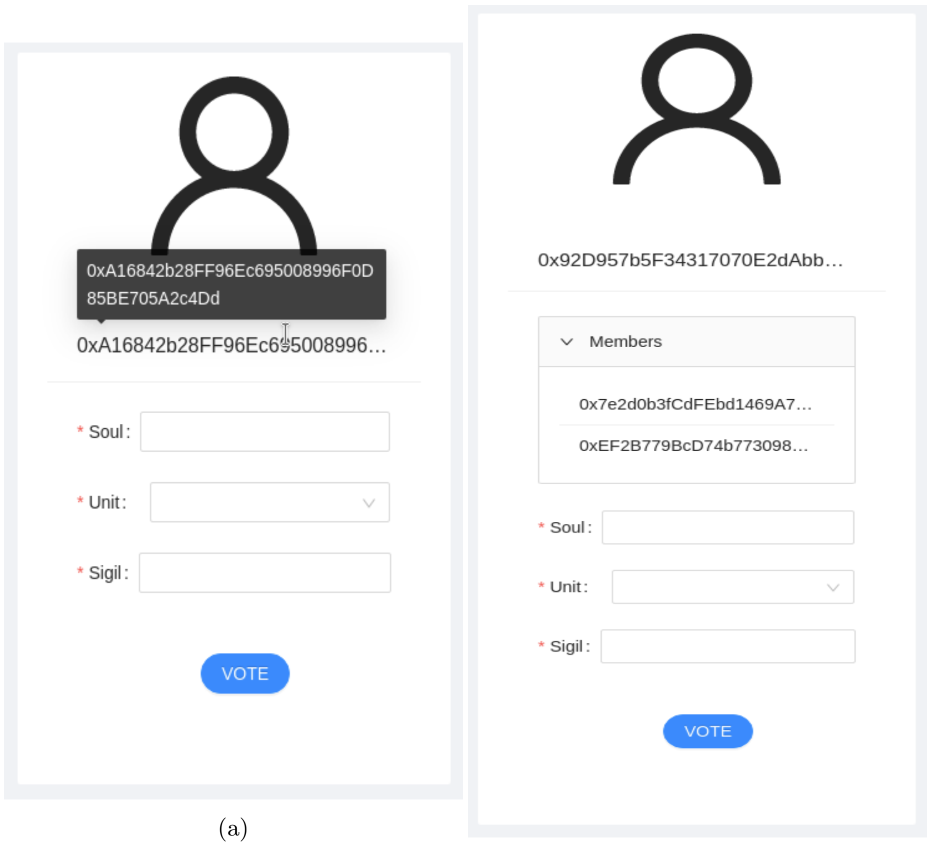
The web page is divided in header and body. Into the header are reported different information like the quorum, the envelopes casted and opened, the address of the current account and its balance in WEI, as we can see in Figure 2. All these information are update in real-time.



Quorum: 2 | Env. Casted: 1 | Env. Openend: 0 | 0xd45E8Cbb5A04C5e98CEb29d8ad9147Ee0D0F3Ec2 | WEI 99824226640000000000

Figure 2: The header shows that the quorum is 2, it has been casted 1 envelope, no envelope has been opened and the balance of the account 0xd45E8Cbb5A04C5e98CEb29d8ad9147Ee0D0F3Ec2 is of 99824226640000000000 WEI

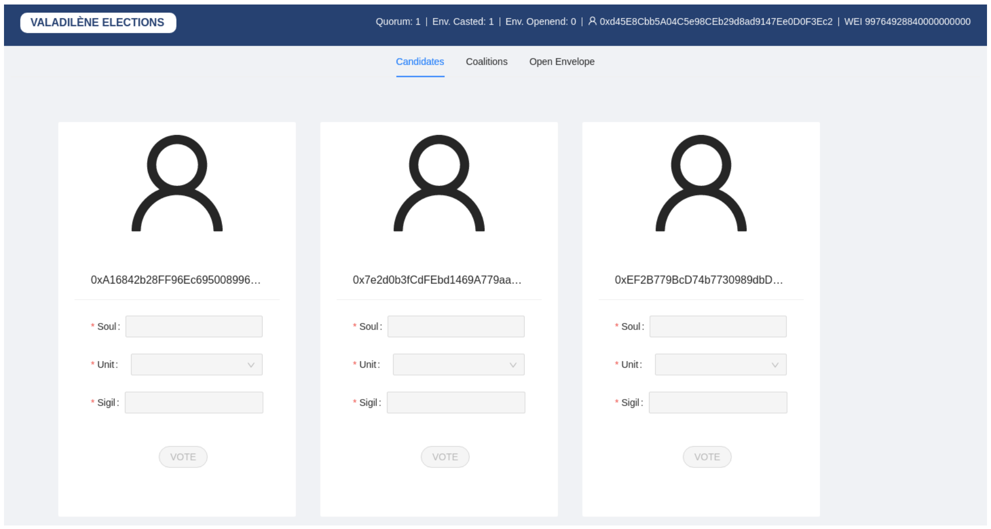
In the body there are initially two tab: *Candidates* and *Coalitions*. In each tab you can find a cards' list, where each card is a candidate (single candidate or a coalition). Into the coalitions' cards are reported also the members of each coalition. An example is reported in Figure 3b. If you want to know the whole address of a candidate, coalition o coalition's member, you can put your mouse on top of the partial address and it will appear a little popup that will shows you the full address.


0xA16842b28FF96Ec695008996F0D85BE705A2c4Dd
0xA16842b28FF96Ec695008996...
Soul:
Unit:
Sigil:
VOTE
0x92D957b5F34317070E2dAbb...
Members
0x7e2d0b3fCdFEbd1469A7...
0xEF2B779BcD74b773098...
Soul:
Unit:
Sigil:
VOTE

(a) (b)

Figure 3: Candidate's card for Figure 3a, coalition's card for Figure 3b.

When a voter want vote for a candidate, he/she specify the soul that he/she wants give to the candidate, the unit between WEI, GWEI or ETH and him/her sigil. All the three inputs must be filled. As soon as the number of envelopes casted is equal to the quorum, the voting phase is closed, the voter cannot vote anymore because the inputs are disabled and appear the *Open Envelope* section, as shown in Figure 4. In this tab all the voters can open their envelopes. The procedure is similar to the one of the vote phase, where the voter selects him/her candidate and specify the soul, unit and sigil. The difference is that here all the candidates are shown together, both individual candidates and coalitions. But it is easily differentiate a coalition's card from the candidate one because of the presence of the members section.


VALADILÈNE ELECTIONS

Quorum: 1 | Env. Casted: 1 | Env. Openend: 0 | 0xd45E8Cbb5A04C5e98CEb29d8ad9147Ee0D0F3Ec2 | WEI 99764928840000000000

Candidates Coalitions Open Envelope

0xA16842b28FF96Ec695008996…

* Soul:

* Unit:

* Sigil:

VOTE

0x7e2d0b3fCdFEbd1469A779aa…

* Soul:

* Unit:

* Sigil:

VOTE

0xEF2B779BcD74b7730989dbD…

* Soul:

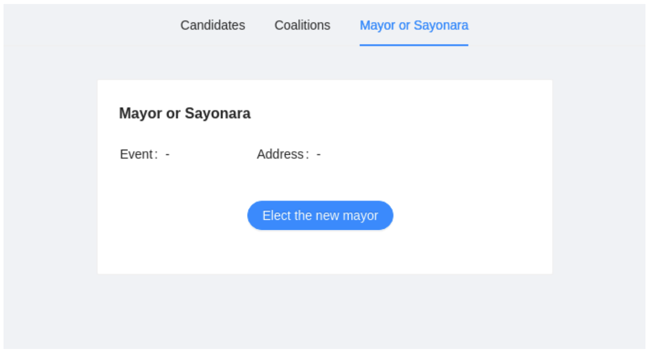
* Unit:

* Sigil:

VOTE

Figure 4: All the inputs are disable because the quorum is equal to the number of envelopes casted and the ”Open envelope” section is appeared.

When everyone has been opened him/her envelope, this section disappear and it is shows the section called *Mayor or Sayonara*, as you can see in Figure 5. Here there is a card with a button that, when it’s clicked, it triggers the execution of the function mayor_or_sayonara and elects the new mayor. The button is clickable only once, to avoid multiple call to the function. In the card there are also two labels that indicate the event, new mayor or sayonara, and the address of the new mayor or 0 in the case of sayonara. You can find an example in Figure 6. In case of new mayor, a little trophy is showed in the card of the winner in Candidates/Coalitions section, to more easily identify the new winner among all candidates.


Candidates Coalitions Mayor or Sayonara

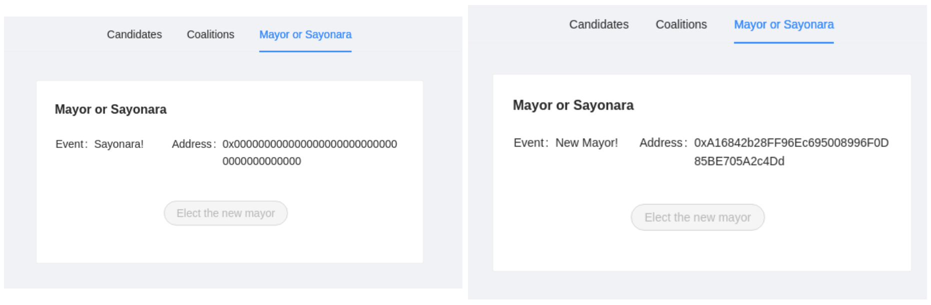
Mayor or Sayonara

Event: -

Address: -

Elect the new mayor

Figure 5: Mayor or sayonara section before to elect the new mayor.



Event: Sayonara! Address: 0x000000000000000000000000000
0000000000000

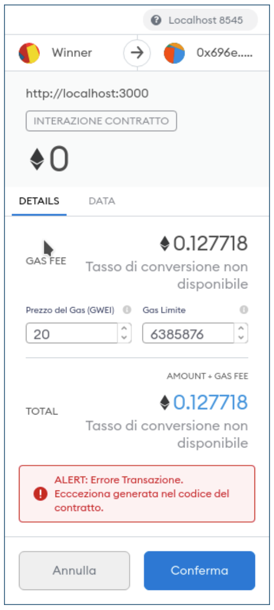
Event: New Mayor! Address: 0xA16842b28FF96Ec695008996F0D
85BE705A2c4Dd

(a)

(b)

Figure 6: Sayonara case (6a) and new mayor case (6b).

About all those situations that trigger a require of the smart contract, Metamask take care of handle the errors. Suppose that a voter votes for candidate A but when he/she opens the envelope, she/he selects candidate B. Then, what will appear in Metamask before to confirm the transaction is something similar to the one shown in Figure 7. If the voter click on confirm button, it will be thrown an exception.



http://localhost:3000

INTERAZIONE CONTRATTO

0

GAS FEE 0.127718
Tasso di conversione non disponibile

Prezzo del Gas (GWEI): 20
Gas Limite: 6385876

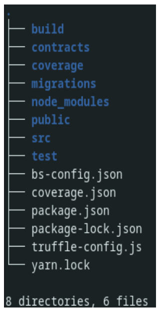
AMOUNT + GAS FEE
TOTAL 0.127718
Tasso di conversione non disponibile

ALERT: Errore Transazione.
Ecccezionа generata nel codice del contratto.

Figure 7: Mayor or sayonara section before to elect the new mayor.

4 Directory Structure and Setup the Project

As you can see in the Figure 8, into the root of the project's folder you find:



```
├── build
├── contracts
├── coverage
├── migrations
├── node_modules
├── public
├── src
├── test
├── bs-config.json
├── coverage.json
├── package.json
├── package-lock.json
├── truffle-config.js
└── yarn.lock

8 directories, 6 files
```

Figure 8: The tree structure of the project's directory. The name in blue are the folders.

- *build*: directory where there are the compiled contracts and the compiled code of the front-end.
- *contracts*: directory where is stored the source code of the smart contract.
- *coverage*: directory generated by the truffle's plugin *solidity-coverage*. This plugin is used to estimate the coverage of the code.
- *migrations*: directory where you can find the code for migrate the smart contract.
- *node_modules*: directory with all the dependencies needed to the project.
- *public*: directory with all public resources required by the front-end.
- *src*: directory with the source code of the front-end.
- *test*: directory with all the tests for the smart contract.

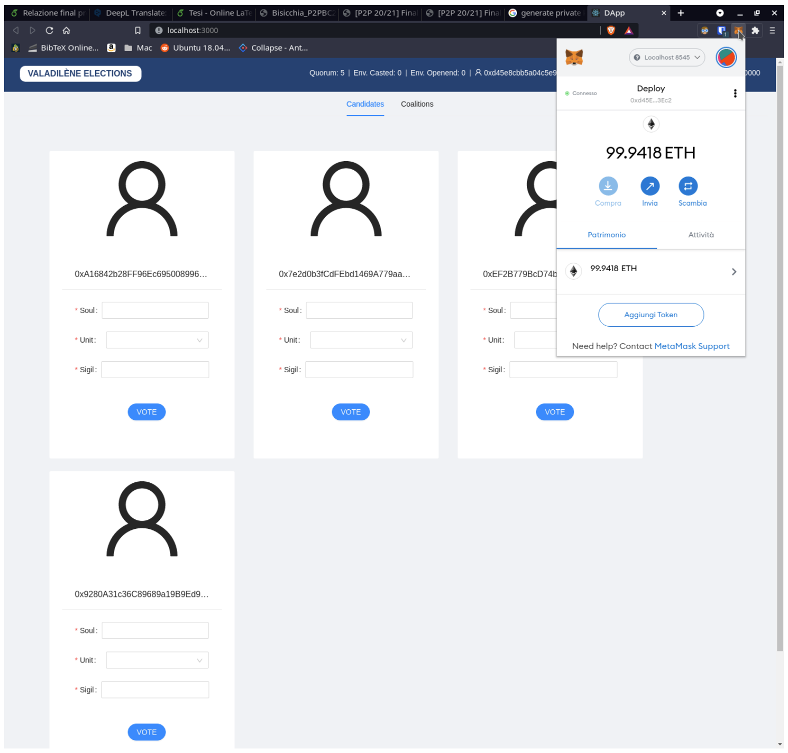
The others files are files of configuration for truffle, lite-server and npm.

To setup the environment for execute the smart contract you need to follow these steps. First of all, through the terminal, you go into the root of the project and you type the command `npm install`. This command will create the directory node_modules with all the dependencies required. After that, you can launch ganache. If you want use ganache-cli, you can write the command `npm run ganache`. This will launch an instance of ganache-cli with 100 accounts and the mnemonic phrase `dutch ivory morning blade episode boy quote cat utility spare chase skate`. Thanks to the passphrase we will have always the same accounts addresses on ganache. In another terminal, you go always into the root of the project, and you need to migrate the smart contract on the network. So you type the command `npm run migrate` and an instance of the contract will be deployed on ganache. After all of this, you can launch the DApp with the command `npm run dev`. An instance of lite-server will start and on your browser will appear the home screen of the DApp.

5 Demo

In this demo there is a quorum of 3 voters. The candidates are composed by 4 individual candidates and 2 coalitions. We will simulate a tie case for the coalitions with the victory of the last single candidate. You can replicate this demo only after you have done the setup the project with the commands discussed in the previous chapter. For convention, the first 8 accounts are reserved in this way: account 0 is the account in charge to deploy the smart contract on the network and to call the mayor_or_sayonara function, the accounts 1,3,5 and 6 are reserved for the candidates, the accounts 2 and 4 are reserved for the coalitions and the account 7 is the escrow account. So, following this pattern, the accounts for the voters will be accounts from 8 to 10.

After you have imported all the needed accounts into Metamask, you can see a screen similar to the Figure 9.



Quorum: 5 | Env. Casted: 0 | Env. Openend: 0 | 0xd45e8cbb5a04c5e9

0xd45E...3Ec2

99.9418 ETH

0xA16842b28FF96Ec695008996...

0x7e2d0b3fCdFEbd1469A779aa...

0xEF2B779BcD74b

0x9280A31c36C89689a19B9Ed9...

Figure 9: The home screen of the DApp with the balance of the deploy account after deployed the smart contract on the network.

Voter 1 will vote for the first coalition, with 10 ETH for the soul while the sigil will be him index, 1. The voter 2 will vote for the second coalition, with 10 ETH and the same rule for the sigil used by the previous voter. The voter 3 instead will vote for the last single candidate with 2 ETH of soul and sigil equal to 3. At the end of the vote phase the balance of the voters is of 99998939980000000000 WEI, while the balance of the deploy account is of 99941762220000000000 WEI. The page should be equal to Figure 10.

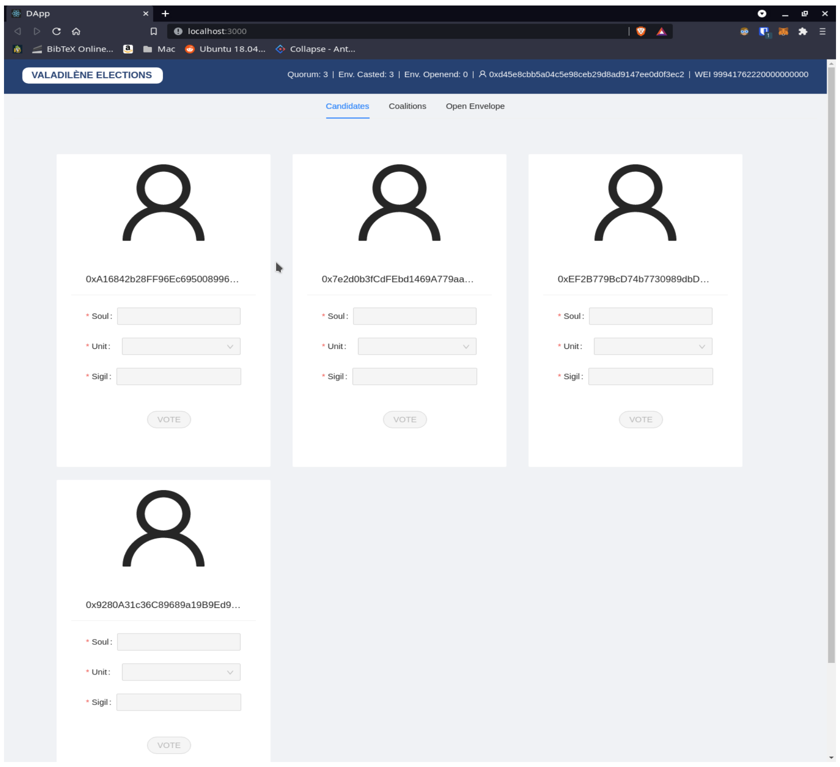
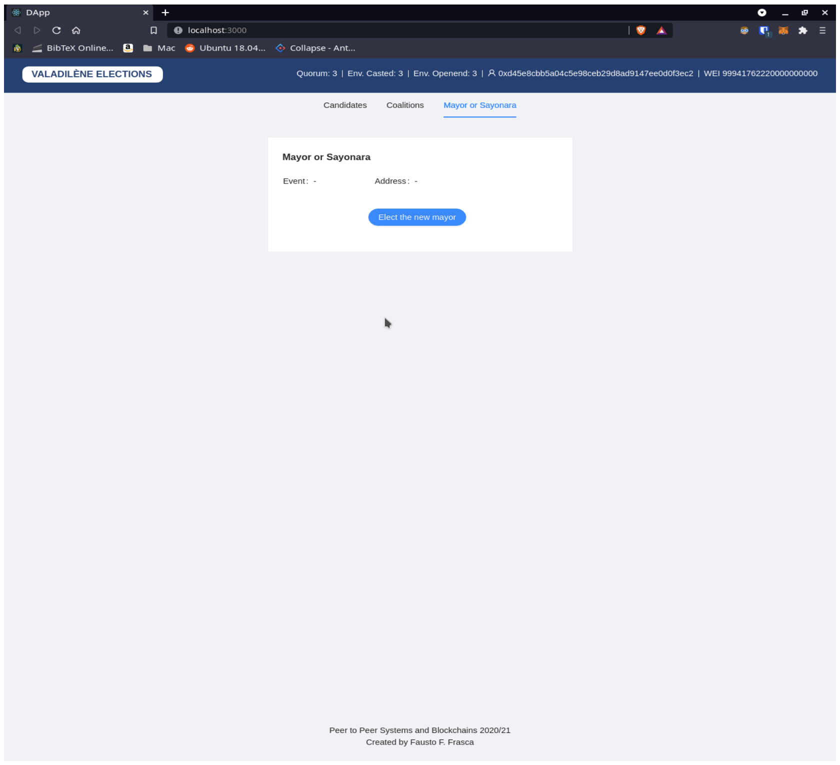

DApp
localhost:3000
BibTeX Online...
Mac
Ubuntu 18.04...
Collapse - Ant...
VALADILÈNE ELECTIONS
Quorum: 3 | Env. Casted: 3 | Env. Openend: 0 | 0xd45e8cbb5a04c5e98ceb29d8ad9147ee0d0f3ec2 | WEI 99941762220000000000
Candidates
Coalitions
Open Envelope
0xA16842b28FF96Ec695008996...
0x7e2d0b3fCdFEbd1469A779aa...
0xEF2B779BcD74b7730989dbD...
0x9280A31c36C89689a19B9Ed9...
Soul:
Unit:
Sigil:
VOTE

Figure 10: The home screen of the DApp after the vote phase.

The "Open Envelope" tab is appeared on the top of the screen, so now we open the envelope of the voters with the inputs used in the previous phase. The sequence of the envelopes opened follow the index of the voters, so first the envelope of the first voter, after of the second voter and lastly the envelope of the third voter. At the end of this phase the balance of the voters' account is:

- *deploy account*: 99941762220000000000 WEI
- *voter 1*: 89996370520000000000 WEI
- *voter 2*: 89996370520000000000 WEI
- *voter 3*: 97994527360000000000 WEI

The "Open Envelope" section disappear automatically when it is opened the last envelope and start the last phase, the election of the new mayor. The screen should be equal to the one shown in Figure 11.



Quorum: 3 | Env. Casted: 3 | Env. Openend: 3 | 0xd45e8cbb5a04c5e98ceb29d8ad9147ee0d0f3ec2 | WEI 99941762220000000000

Mayor or Sayonara

Event: -
Address: -

Peer to Peer Systems and Blockchains 2020/21
Created by Fausto F. Frasca

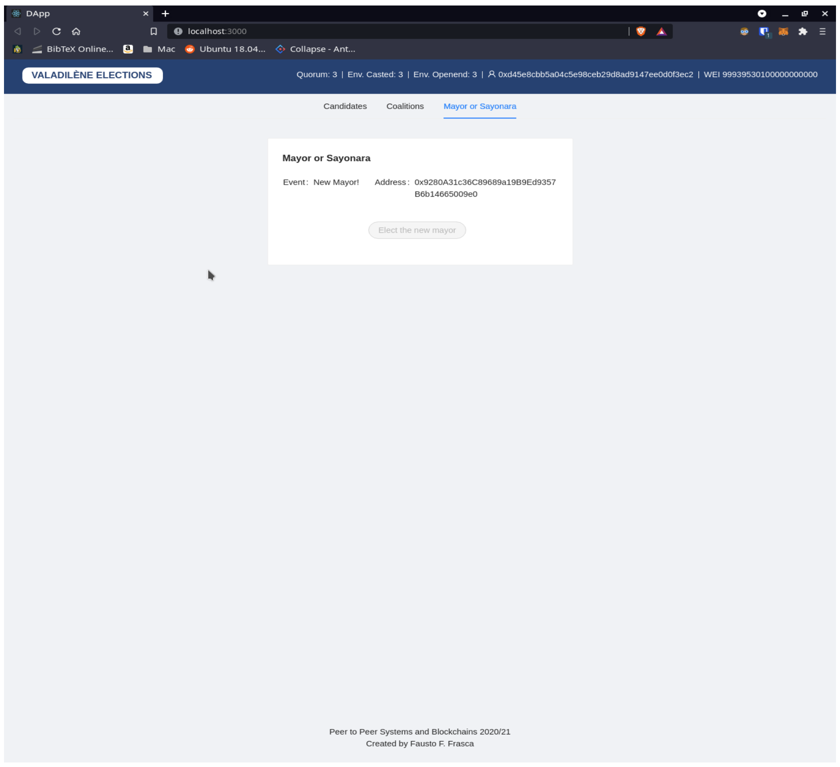
Figure 11: The home screen of the DApp after the open envelope phase.

Now we switch to deploy account, we click the "Elect new mayor" button and what we should see is something equal to the Figure 12.

We can ensure that the winner is the last candidate by going to "Candidate" section and checking if there is a little green trophy in the last candidate's card, as shown in Figure 13.

The final balance of each account is:

- *deploy account*: 99939530100000000000 WEI
- *voter 1*: 99994786800000000000 WEI
- *voter 2*: 99996370520000000000 WEI
- *voter 3*: 97994527360000000000 WEI
- *winner*: 102000000000000000000 WEI



Quorum: 3 | Env. Casted: 3 | Env. Openend: 3 | 0xd45e8cbb5a04c5e98ceb29d8ad9147ee0d0f3ec2 | WEI 9993953010000000000

Mayor or Sayonara

Event: New Mayor! Address: 0x9280A31c36C89689a19B9Ed9357B6b14665009e0

Peer to Peer Systems and Blockchains 2020/21
Created by Fausto F. Frasca

Figure 12: The home screen of the DApp after the elect phase.

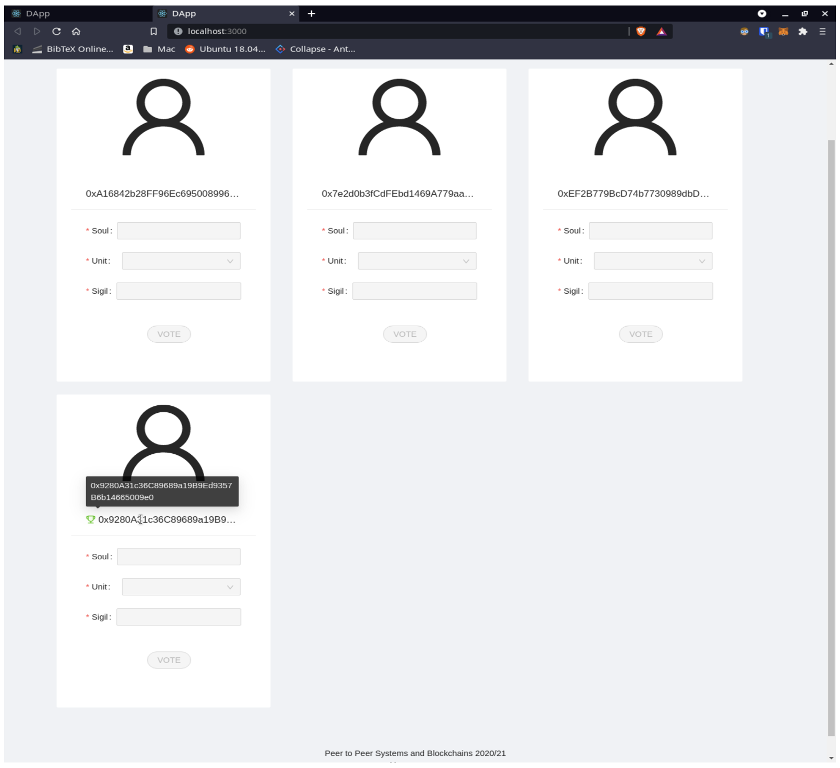

DApp
DApp
localhost:3000
BibTeX Online...
Mac
Ubuntu 18.04...
Collapse - Ant...
0xA16842b28FF96Ec695008996...
0x7e2d0b3fCdFEbd1469A779aa...
0xEF2B779BcD74b7730989dbD...
* Soul :
* Unit :
* Sigil :
VOTE
0x9280A31c36C89689a19B9Ed9357B6b14665009e0
0x9280A31c36C89689a19B9...
Peer to Peer Systems and Blockchains 2020/21

Figure 13: Little green trophy near the winner of the elections.