UNIVERSITÀ DI PISA

PEER TO PEER SYSTEMS AND BLOCKCHAINS
A.Y. 20/21

# FINAL PROJECT REPORT
*The Attempt Of a Democratic Election System*

*Monica Amico* *516801*

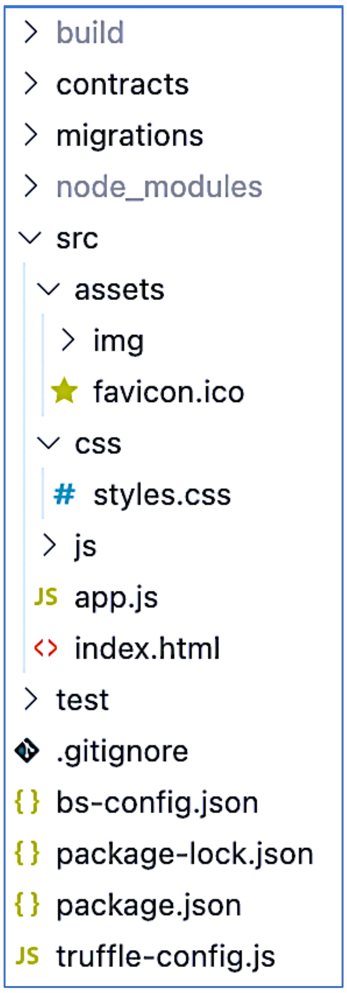
## 1. Structure of The Project

A ***Decentralized Application*** (DApp) is an application built on a decentralized network, with some smart contracts and a user interface to interact with them.

**Used tools:**

- *Truffle* is a development framework for Ethereum that manages the contract artefacts and network artefacts.
- *Ganache:* is a private blockchain for Ethereum development that can be used to deploy contracts, develop applications, and run tests. I used the Desktop version but is also available the CLI version.
- *Metamask*: is a Browser extension that let connect with an Ethereum network (i.e., Ganache, Truffle, Ropsten...), it is also useful to interact with the Dapp in the browser.
- *Bootstrap:* is a frontend toolkit.

The Project is structured as follow:



```
> build
> contracts
> migrations
> node_modules
v src
  v assets
    > img
    favicon.ico
  v css
    # styles.css
  > js
  JS app.js
  <> index.html
> test
.gitignore
{} bs-config.json
{} package-lock.json
{} package.json
JS truffle-config.js
```

The folders `contracts`, `migrations`, `test` and the file `truffle-config.js` are generated by the command: `truffle init`.

The `build` folder contains JSON files generated during the deployment of the smart contract `Mayor.sol`, which is inside the `contracts` folder.

Inside the folder `migrations`, there is a JavaScript file used to deploy the contract on the Ganache network.

Into the folder `src` there are all the files useful for the frontend and to interact with the smart contract.

The file `truffle-config.js` is a JavaScript file used to configure the project, for example, we can specify the compiler version and the development network: the host and the port. In our case, Ganache runs on localhost using its default port 7545.

The `package.json` file is created by the command `npm init` and it was automatically updated after the command `npm install -save lite-server`.

The `bs-config.json` is the configuration file that tells *lite-server* where are the folders useful to serve the application.

## 2. Implementation

### `Mayor.sol`

I have chosen to implement the Election plan named: "*Join my side, I give you cookies*."
I have started by extending the smart contract of the Final Term, in particular, I have added some functions and data structures, in order to implement the additional task and to consider a list of candidates instead of only one candidate.

Mainly, these structures have been added:

- `address payable[] public candidates;`
- `mapping(address => uint) public deposit;`
- `mapping(address => Vote) public votes;`
- `struct Vote { uint souls; uint number; }`

I have changed the constructor; it also contains the list of candidates as a parameter:

-
  ```
  constructor(address payable[] memory _candidates,
              address payable _escrow,
              uint32 _quorum)
  ```

I have also added the following functions:

- `function getEscrow() public view returns(address payable):`

  used by the script to get the Escrow address

- `function getCandidates() public view returns(address payable[] memory):`

  used by the script to get the Candidates address list

- `function deposit_soul() public payable:`

  implements the additional functionality, the function is used to deposit eth and can be called only by the candidates. To start the election all candidates must have deposited some souls

- `function hasDeposited(address account) public view returns(bool):`

  used by the script to know if the account has already deposited some souls

- `function getWinner() canCheckOutcome private returns(address payable):`

  called by the `mayor_or_sayonara` function, is used to calculate the winner and return the address, in case of a tie it returns the escrow's address, otherwise one of one the candidates

- `function seeWinner() public view returns(address payable):`

  used by the script to know who won the election, can be called only if the `mayor_or_sayonara` function has already been called.

- `function splitElectors(address payable winner_par) canCheckOutcome private:`
  if there wasn't a tie case, is called by the `mayor_or_sayonara` function, to split the electors into two parts: the ones who voted for the winner and the others. Is useful to then transfer the winner's deposit to the first part of electors.

- `function canSeeWinner() public view returns(bool):`
  used by the script to check if the Dapp can show the winner

- `function canSetWinner() public view returns(bool):`
  used by the script to check if the Dapp is in the final phase and we can check who is the winner

- `function canOpenEnvelope(address account) public view returns(bool):`
  is used by the script to check if the Dapp is in the open phase (quorum reached) and the electors can start to open their envelope

- `function canCastEnvelope(address account) public view returns(bool):`
  is used by the script to check if the Dapp is in the casting phase and the electors can start to cast their envelope

## `App.js`

Is the Dapp script, takes information about the contract instance status, modifies them and implements the frontend. Every time the window loads, it does the `web3 init` and the `contract instance inits`, then it does the page render and let users interact with the smart contract.

## `Index.html`

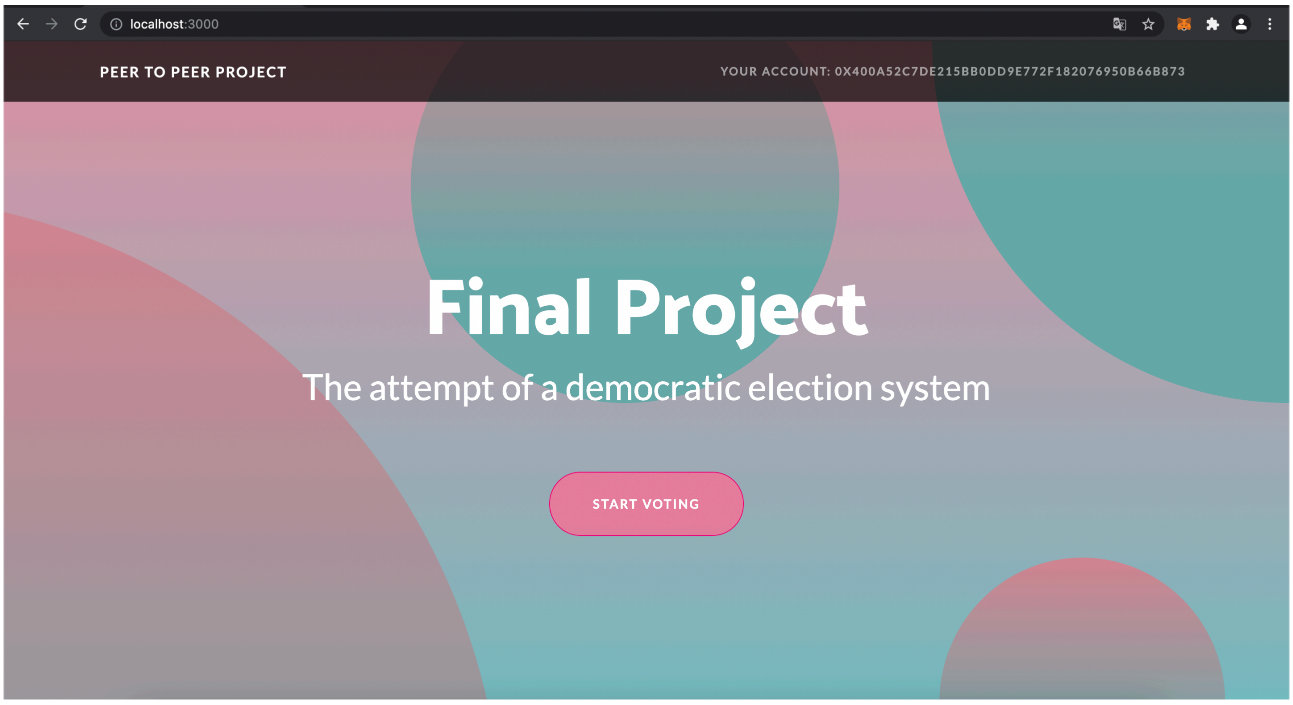
Together with app.js is the client-side application to interact with the contract. It has some sections; each corresponds to a phase of the smart contract instance and is visible when the contract let the user does a specific operation, e.g.: cast an envelope, open an envelope.

## 3. The Demo

First of all, we need to install Ganache and Metamask, then add the private Ganache network on Metamask and add some accounts using the accounts' private keys that can be founded in Ganache.
When all is ready, we can compile and migrate the contract and start the lite-serve using the commands:

```
    sudo truffle migrate —reset
    npm run dev
```

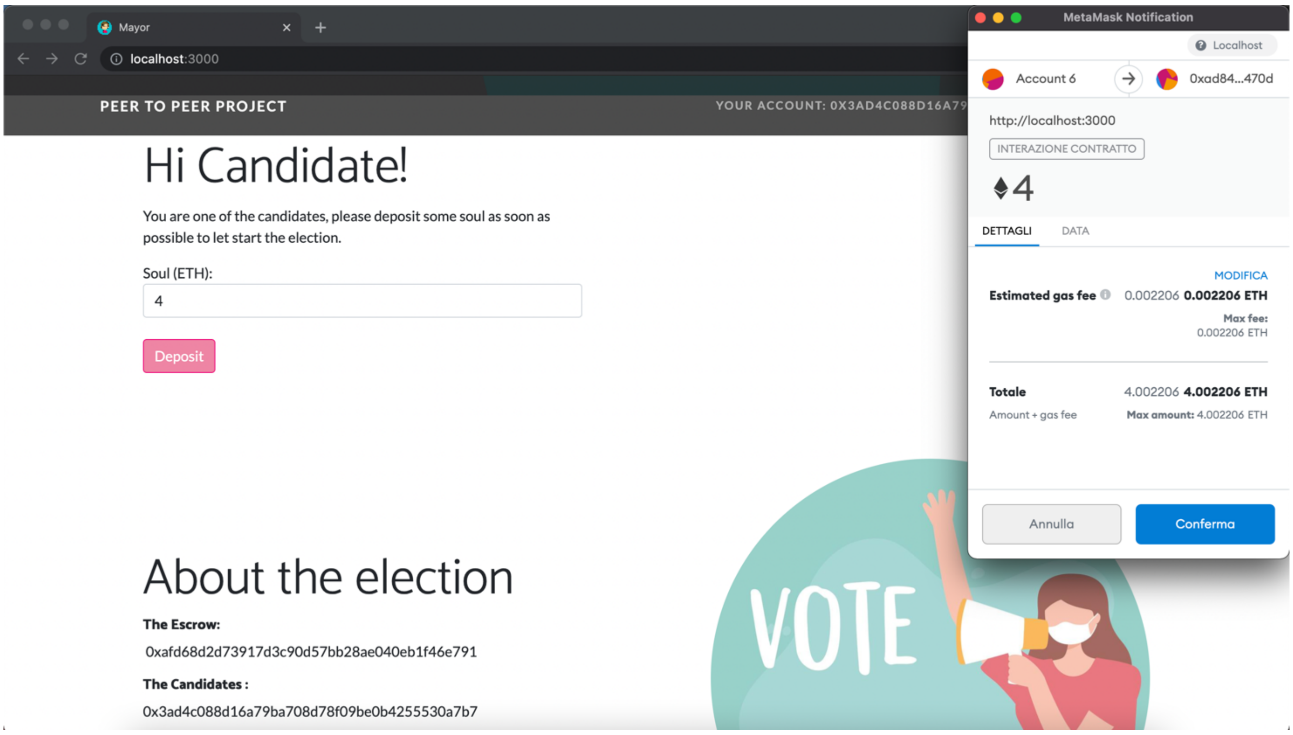
Open the browser and digit "localhost:3000" it will appear the Dapp client page.


localhost:3000
PEER TO PEER PROJECT
YOUR ACCOUNT: 0X400A52C7DE215BB0DD9E772F182076950B66B873
Final Project
The attempt of a democratic election system
START VOTING

On the right top, we can see the current address account.
To start the election, all the candidates must deposit some souls (eth), which at the end in case of win will be distributed to all the electors that have voted for the candidate winner. So, we need to switch to the candidates' account (in Metamask) and start the first phase: *the deposit*.

In this phase, the eth, that each candidate deposits, are effectively transferred to the contract. To do that, there is a function on the `app.js` file that is called when the `Deposit` button is clicked, this function does necessary controls (that are also in the backend) and then calls the `send transaction` to send the eth.

So, the first page seen by candidates is the following page:


Mayor

localhost:3000

PEER TO PEER PROJECT

YOUR ACCOUNT: 0X3AD4C088D16A79

Hi Candidate!

You are one of the candidates, please deposit some soul as soon as possible to let start the election.

Soul (ETH): 4

Deposit

About the election

**The Escrow:**
0xafd68d2d73917d3c90d57bb28ae040eb1f46e791

**The Candidates :**
0x3ad4c088d16a79ba708d78f09be0b4255530a7b7

MetaMask Notification

Localhost

Account 6 → 0xad84...470d

http://localhost:3000

INTERAZIONE CONTRATTO

4

DETTAGLI DATA

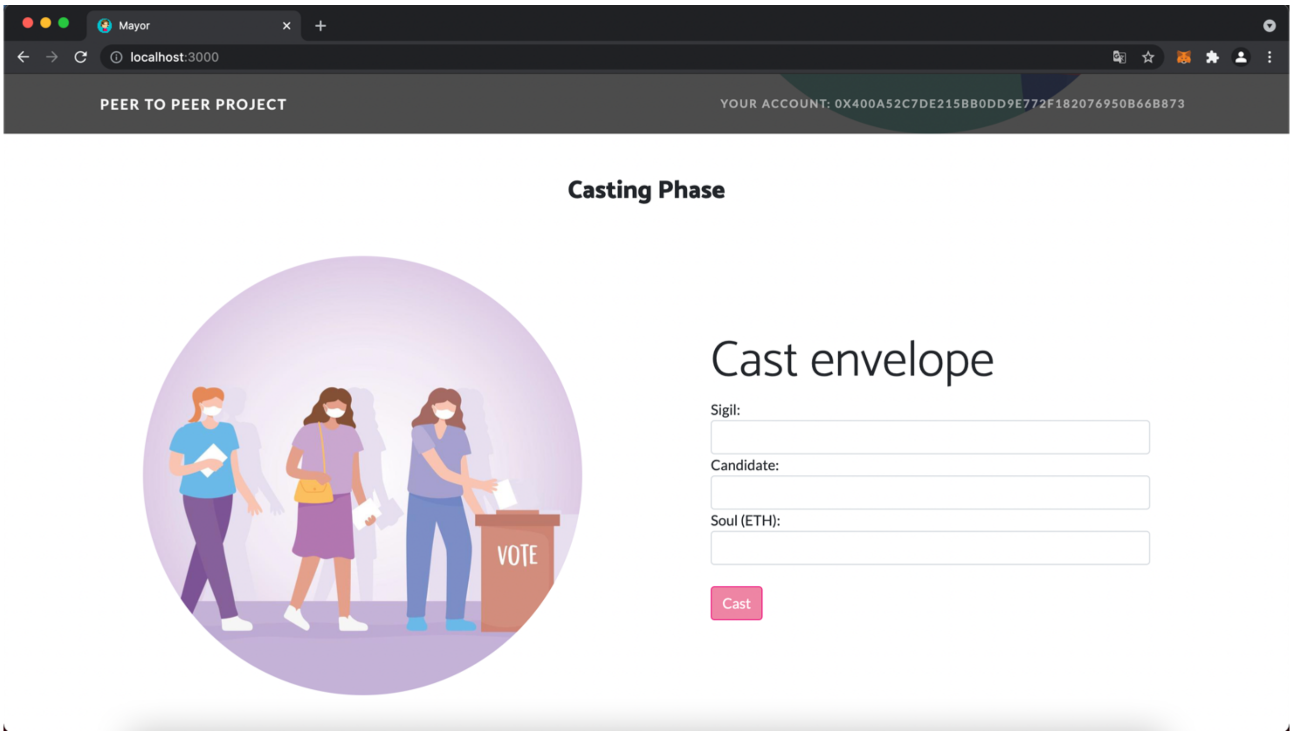
MODIFICA

| | |
|---|---|
| **Estimated gas fee** | 0.002206 **0.002206 ETH** |
| | **Max fee:** 0.002206 ETH |
| **Totale** | 4.002206 **4.002206 ETH** |
| Amount + gas fee | **Max amount:** 4.002206 ETH |

Annulla Conferma

In this image, we can see a candidate that wants to deposit 4 eths. When the Deposit button is clicked a Metamask pop-up appears, and the transaction has to be confirmed. After all, candidates have deposited, the section of the next phase is shown, and the electors can start to cast their envelope until the quorum will be reached.

Each elector has to choose a candidate, the sigil and how much eth wants to send during the next phase. These 3 parameters are sent to the smart contract, which will compute the hash.


Mayor

localhost:3000

PEER TO PEER PROJECT

YOUR ACCOUNT: 0X400A52C7DE215BB0DD9E772F182076950B66B873

**Casting Phase**

Cast envelope

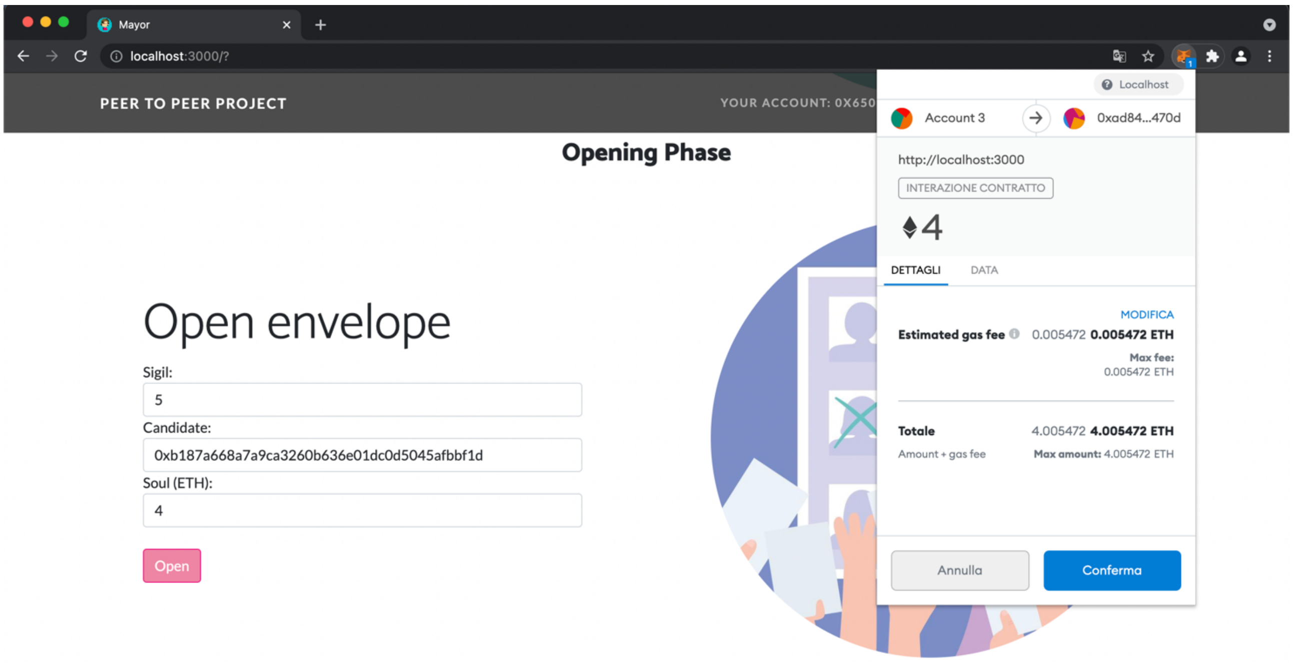
Sigil:

Candidate:

Soul (ETH):

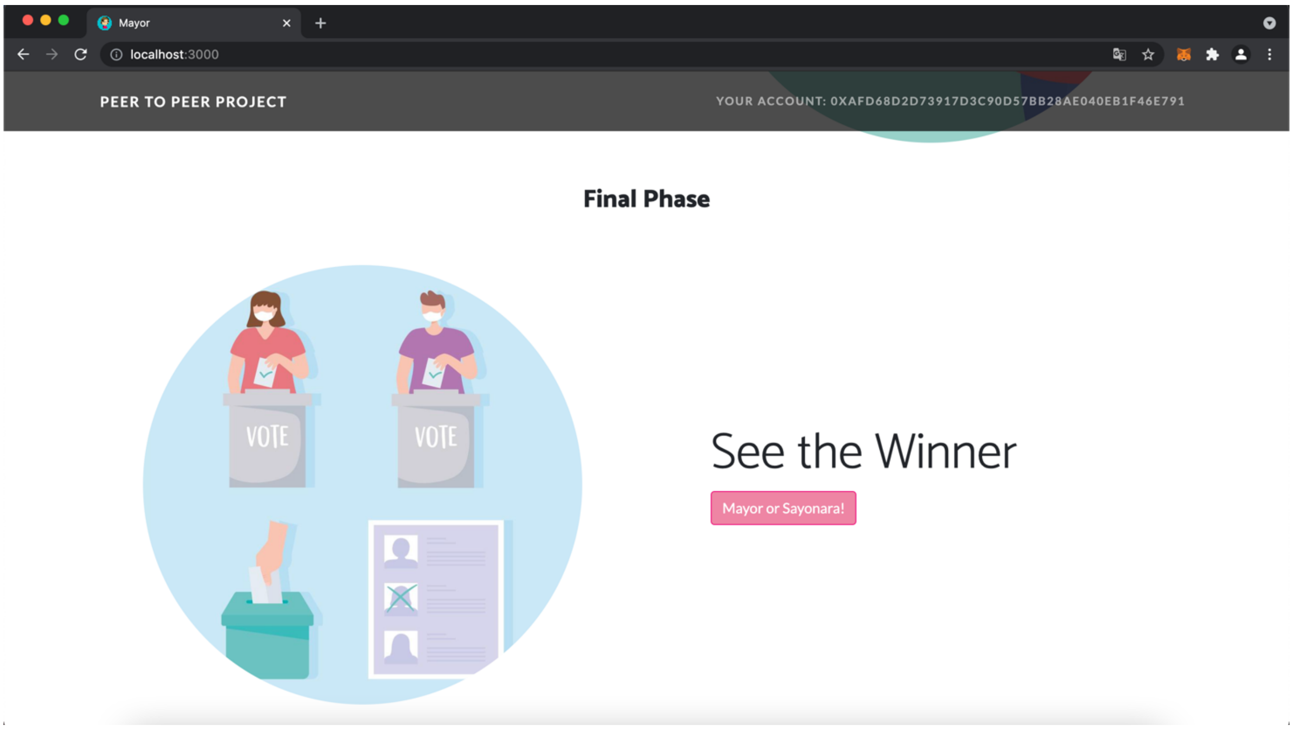
Cast

When the quorum has reached the section of the third phase is available to the electors that have cast their envelopes before and have to open them. They have to write again the same parameters used during the cast envelope, to confirm their vote. Now the eths in the Soul form are transferred to the smart contract. As always, the transaction has to be confirmed into the Metamask pop-up.

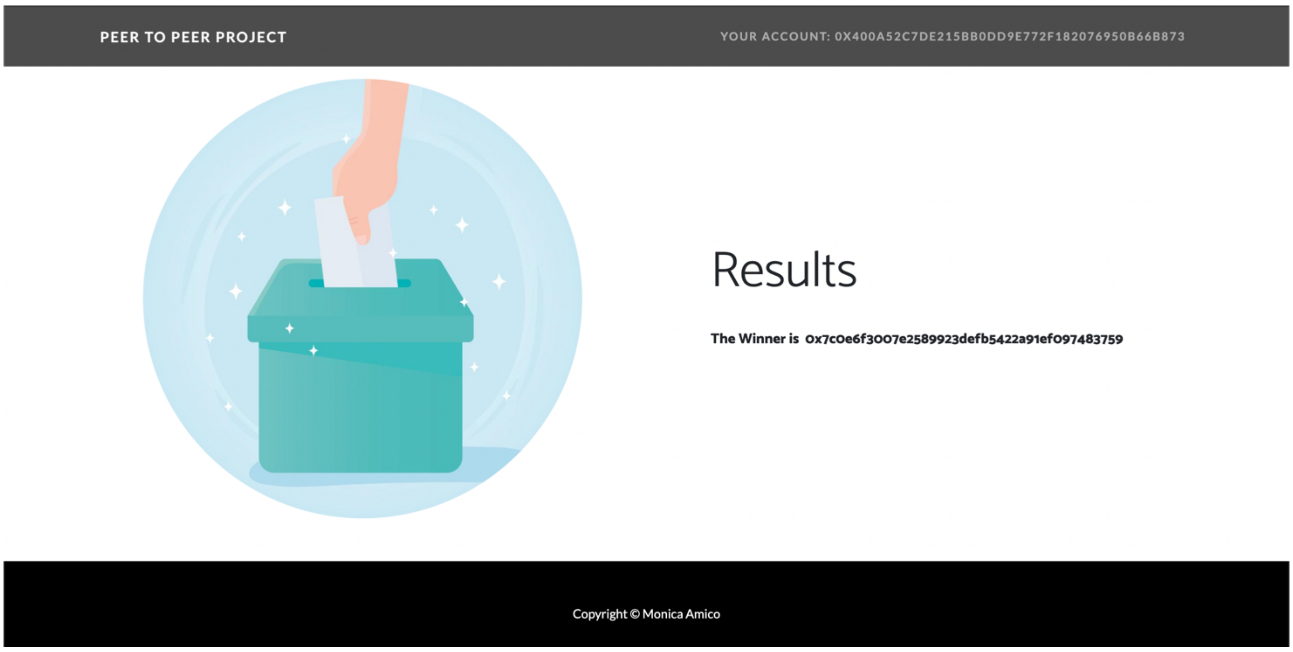

Mayor
localhost:3000/?
PEER TO PEER PROJECT
YOUR ACCOUNT: 0X650
Opening Phase
Open envelope
Sigil: 5
Candidate: 0xb187a668a7a9ca3260b636e01dc0d5045afbbf1d
Soul (ETH): 4
Open

Localhost
Account 3 → 0xad84...470d
http://localhost:3000
INTERAZIONE CONTRATTO
4
DETTAGLI DATA
MODIFICA
Estimated gas fee 0.005472 0.005472 ETH
Max fee: 0.005472 ETH
Totale 4.005472 4.005472 ETH
Amount + gas fee Max amount: 4.005472 ETH
Annulla Conferma

Only when all the electors that have to open their envelopes, had opened them, the final phase will be available, and the winner could be calculated by the `Mayor or Sayonara` function.


Mayor
localhost:3000
PEER TO PEER PROJECT
YOUR ACCOUNT: 0XAFD68D2D73917D3C90D57BB28AE040EB1F46E791
Final Phase
See the Winner
Mayor or Sayonara!

At the end, all the phases are concluded, and the winner is shown in one of the HTML sections.



PEER TO PEER PROJECT

YOUR ACCOUNT: 0X400A52C7DE215BB0DD9E772F182076950B66B873

Results

**The Winner is 0x7c0e6f3007e2589923defb5422a91ef097483759**

Copyright © Monica Amico

To restart the election the smart contract need to be compiled again, otherwise the lite-server will show again the same result.

The parameters of the constructor (candidates list, escrow, quorum) could be changed, changing the initial migration’s deploy function.