

Peer to Peer Systems and Blockchains

Academic Year 2020/2021

Final Project

Democratic Election System

Project Structure:

The DAPP for election system consists of the backend smart contract programmed on solidity. It also have tests files with 6 main test cases as:

- **should be able to add the candidates**
- **should be able to cast envelope**
- **should not open envelope before the quorum is reached**
- **should be able to open envelope after reaching quorum**
- **should be able to open envelop - Draw scenario**
- **should be able to open envelop - Winner scenario**

Moreover, the application contains the client folder that has the frontend code based on ReactJS. The frontend (client) folder contains the contracts folder that contains the compiled form of contracts.

Project Setup:

Project required NodeJS v10. Ganache desktop application or Ganache-cli should also be installed and running while using the application. Once the NodeJS is installed copy the project code files to a specific folder.

Install dependencies for backend and frontend:

Backend Dependencies: at the root of the project folder run the command ***npm install***

Fontend Dependencies: navigate to client folder and run the command ***npm install***

Other commands: this includes truffle commands like: ***truffle compile, truffle migrate, truffle deploy***

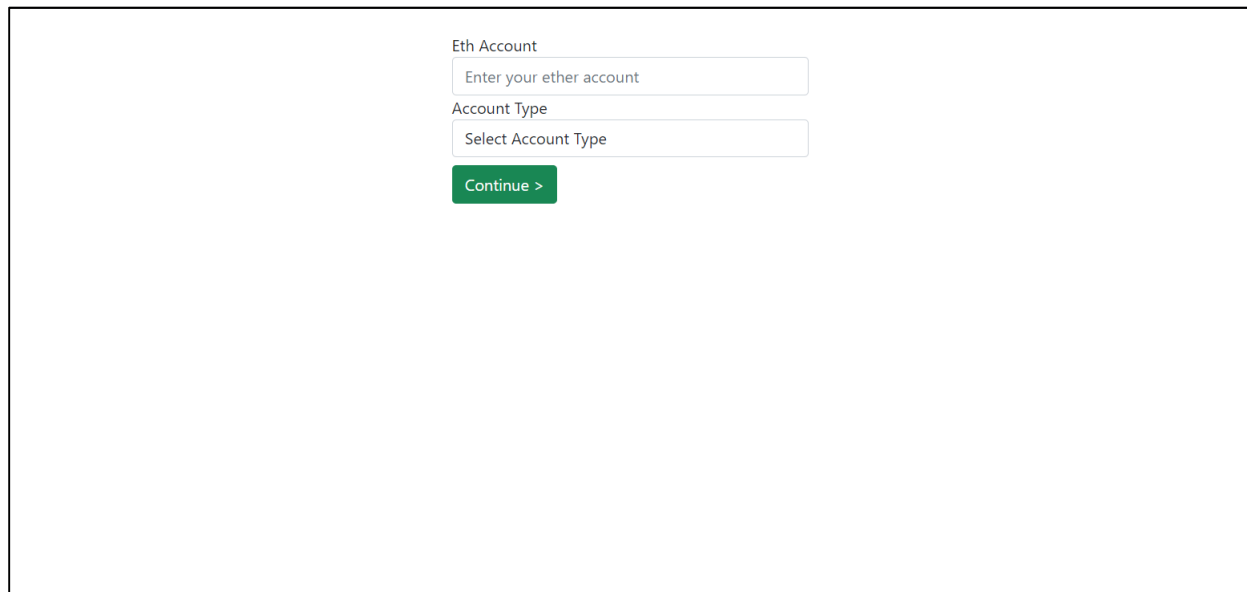
Project Execution:

Once all the dependencies are installed and Ganache running. We will start with migrating the contracts to Ganache blockchain network using command ***truffle migrate***

We can also test application and its dependencies setup successfully by running the test cases that are attached with the project. We can do using command ***truffle test*** at the root of the application folder.

Open terminal in the frontend (client) folder of the application and run the frontend application using ***npm start***

You will see the following screen:

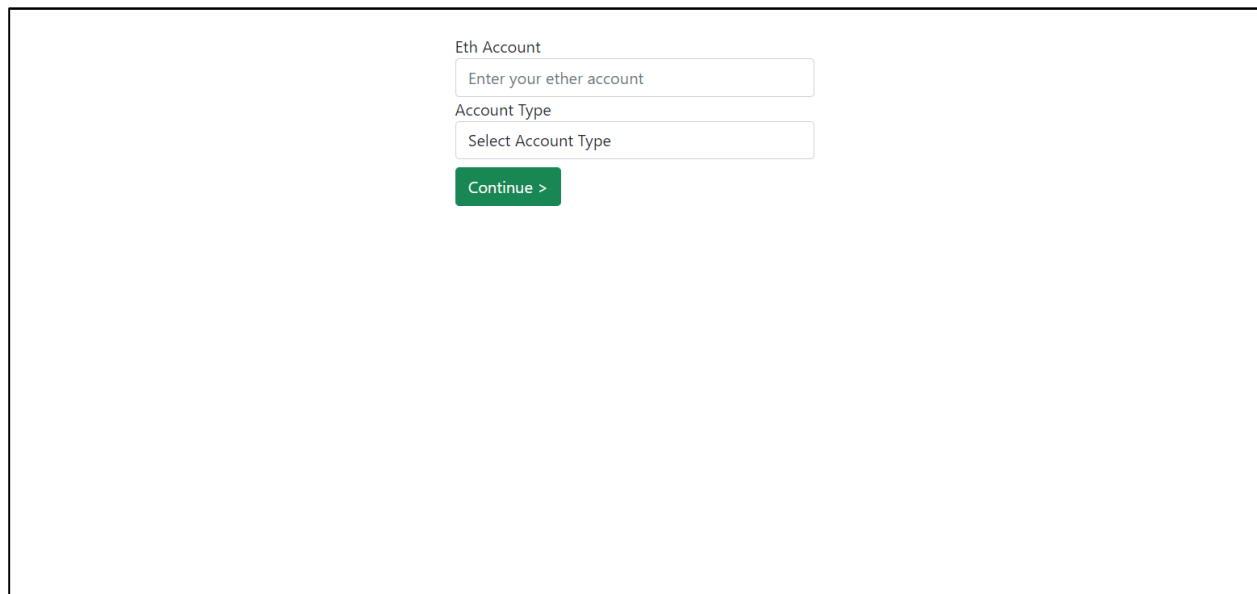


The screenshot shows a web form with two input fields and a button. The first field is labeled 'Eth Account' and contains the placeholder text 'Enter your ether account'. The second field is labeled 'Account Type' and contains the placeholder text 'Select Account Type'. Below these fields is a green button with the text 'Continue >'.

The application is now up and running with frontend.

User Manual:

Once the project is up and running you will see the following screen.



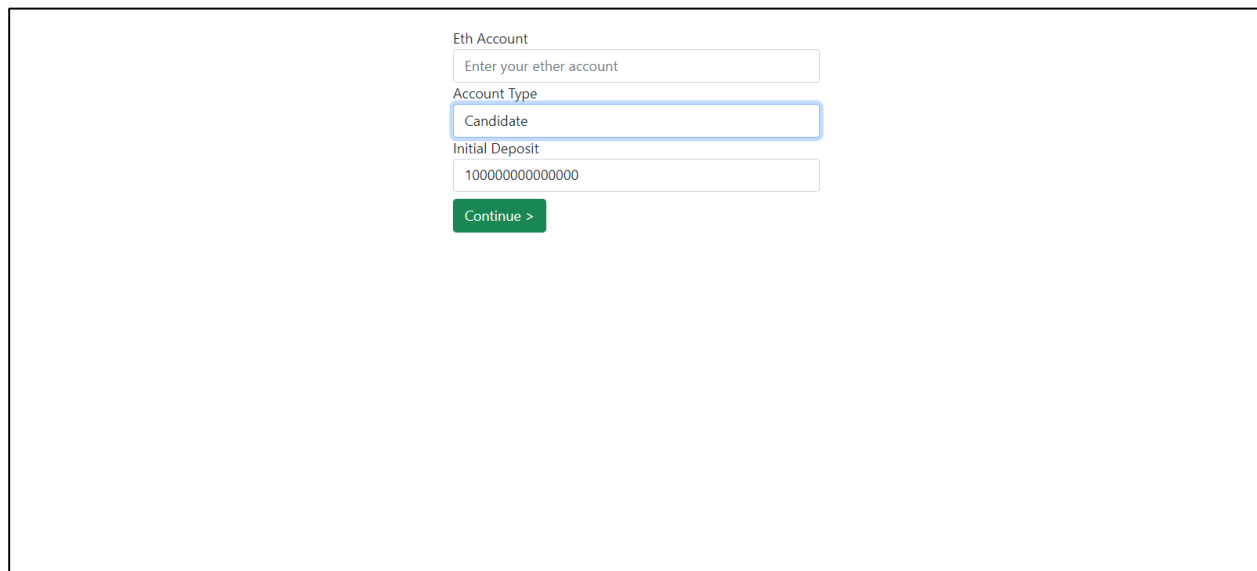
The screenshot shows a web form with two input fields and a button. The first field is labeled 'Eth Account' and contains the placeholder text 'Enter your ether account'. The second field is labeled 'Account Type' and contains the placeholder text 'Select Account Type'. Below these fields is a green button with the text 'Continue >'.

A user can enter his/her address

e.g. 0x983B296cEA9Ec70C19Be6a9A5977B2e04c53Fe42

A user can select if he is registering as a candidate or voter from the second field.

Selecting an option “CANDIDATE”, a candidate had to enter the initial deposit ether amount in the field at bottom.



The screenshot shows a registration form with the following fields and options:

- Eth Account**: A text input field with the placeholder text "Enter your ether account".
- Account Type**: A dropdown menu with "Candidate" selected and highlighted by a blue border.
- Initial Deposit**: A text input field containing the value "1000000000000000".
- Continue >**: A green button with white text.

A user can click continue button to see the candidate dashboard. The dashboard will look like this.

The screenshot shows a user interface for a candidate dashboard. At the top, a green banner displays the login status: "Logged in using account: 0xfa4A5565F5711d49EeD11a836Edfe909128B4889" with a "Logout" button on the right. Below this, the page is divided into two main sections. On the left, under the heading "# Candidate Accounts", there is a table with three rows of account IDs. On the right, a "Results" section indicates "Results not available yet..." and includes a "Check result" button. At the bottom left, a box titled "Account Type: CANDIDATE" shows the user's ID and a "Refresh Page" button.

#	Candidate Accounts
1	0xfa4A5565F5711d49EeD11a836Edfe909128B4889
2	0x586eA9e715Fed2a12eed837d88c9Dab2F1d16cCB
3	0x15b553F7B1267BEC441C60b498327137d0dDdC62

Results
Results not available yet...
[Check result](#)

Account Type: CANDIDATE
0xfa4A5565F5711d49EeD11a836Edfe909128B4889
Your id: 0xfa4A5565F5711d49EeD11a836Edfe909128B4889
[Refresh Page](#)

It will show all the registered candidates and logged in user details.

Now back to main login screen, if the user type is “VOTER” instead of “CANDIDATE”. If a user select VOTER then it the screen will look like.

The screenshot shows the main login screen for a voter. It contains several input fields: "Eth Account" with the value 0xfa4A5565F5711d49EeD11a836Edfe909128B4, "Account Type" with the value Voter, "Candidate Account" with the value 0xfa4A5565F5711d49EeD11a836Edfe909128B4, "Voting Amount" with the value 1000000000000000, and "SIGIL" with the value 1. A green "Continue >" button is located at the bottom.

Eth Account
0xfa4A5565F5711d49EeD11a836Edfe909128B4

Account Type
Voter

Candidate Account
0xfa4A5565F5711d49EeD11a836Edfe909128B4

Voting Amount
1000000000000000

SIGIL
1

[Continue >](#)

A user can enter the address of the candidate whom he/she want to vote, the voting amount and his/her SIGIL. Then a user can click continue button to move to VOTER's dashboard.

A VOTER dashboard will look like.

Logged in using account: 0x47Cea8931Aa5946F9c1daBCfb0c27f50495f41f5

Logout

#Candidate Accounts

10xfa4A5565F5711d49EeD11a836Edfe909128B4889

20x586eA9e715Fed2a12eed837dB8c9Dab2F1d16cCB

30x15b553F7B1267BEC441C60b498327137d0dDdC62

Account Type: VOTER

0x47Cea8931Aa5946F9c1daBCfb0c27f50495f41f5

You voted for candidate: 0x15b553F7B1267BEC441C60b498327137d0dDdC62

Refresh Page

Results

Results not available yet...

Check result

Open Envelope

0x47Cea8931Aa5946F9c1daBCfb0c27f50495f41f5

SIGIL

0

Candidate ID

0x15b553F7B1267BEC441C60b498327137d0dDdC62

Open Envelope

There is a limit of QUORUM and for opening an envelope the QUORUM limit must be reached. Once a QUORUM is reached a user can click open envelope button which will open an envelope for that user. A screen will turn to this after envelope is opened.

Logged in using account: 0x47Cea8931Aa5946F9c1daBCfb0c27f50495f41f5

Logout

#Candidate Accounts

10xfa4A5565F5711d49EeD11a836Edfe909128B4889

20x586eA9e715Fed2a12eed837dB8c9Dab2F1d16cCB

30x15b553F7B1267BEC441C60b498327137d0dDdC62

Account Type: VOTER

0x47Cea8931Aa5946F9c1daBCfb0c27f50495f41f5

You voted for candidate: 0xfa4A5565F5711d49EeD11a836Edfe909128B4889

Refresh Page

Results

Results not available yet...

Check result

Open Envelope

0x47Cea8931Aa5946F9c1daBCfb0c27f50495f41f5

SIGIL

0

Candidate ID

0xfa4A5565F5711d49EeD11a836Edfe909128B4889

Envelope already opened

The voting result can be calculated once all the registered voters had opened their envelope.

Once all the voters has opened their envelope, one of them can click check result button to calculate result. Result can be calculated by one of them only. Based on WIN or TIE the results box will look like.

Logged in using account: 0x47Cea8931Aa5946F9c1daBCfb0c27f50495f41f5Logout

#Candidate Accounts

1	0xfa4A5565F5711d49EeD11a836Edfe909128B4889
2	0x586eA9e715Fed2a12eed837dB8c9Dab2F1d16cCB
3	0x15b553F7B1267BEC441C60b498327137d0dDdC62

Account Type: VOTER

0x47Cea8931Aa5946F9c1daBCfb0c27f50495f41f5

You voted for candidate: 0xfa4A5565F5711d49EeD11a836Edfe909128B4889

Refresh Page

Results

Result:

0xfa4A5565F5711d49EeD11a836Edfe909128B4889

Check result

Open Envelope

0x47Cea8931Aa5946F9c1daBCfb0c27f50495f41f5

SIGIL

0

Candidate ID

0xfa4A5565F5711d49EeD11a836Edfe909128B4889

Envelope already opened

Logged in using account: 0x47Cea8931Aa5946F9c1daBCfb0c27f50495f41f5Logout

#Candidate Accounts

1	0xfa4A5565F5711d49EeD11a836Edfe909128B4889
2	0x586eA9e715Fed2a12eed837dB8c9Dab2F1d16cCB
3	0x15b553F7B1267BEC441C60b498327137d0dDdC62

Account Type: VOTER

0x47Cea8931Aa5946F9c1daBCfb0c27f50495f41f5

You voted for candidate: 0x15b553F7B1267BEC441C60b498327137d0dDdC62

Refresh Page

Results

Draw

Check result

Open Envelope

0x47Cea8931Aa5946F9c1daBCfb0c27f50495f41f5

SIGIL

0

Candidate ID

0x15b553F7B1267BEC441C60b498327137d0dDdC62

Envelope already opened

Decision/Logics in Smart Contracts and Frontend:

In this system we implemented few extra things (a bit of change in data structures) apart from the previous version. The implementation includes make candidates variable, an array. And a STRUCT implementation or storing candidate id, vote counts and voter's list as we need to track which voter belongs to which candidate. We made the following changes:

```
struct Candidate {  
    address payable id;  
    uint vote_count;  
    address payable[] voters;  
}  
  
address payable[] public candidate_list;  
mapping(address => Candidate) candidates;
```

We created candidate_list so to loop through the candidate mapping and struct. We further added a function called add_candidate to push candidate to candidate_list and its details to candidates mapping.

```
function add_candidate(address payable _candidate) public payable {  
    require(msg.value > 0, "Deposit souls must be greater than 0...");  
    candidates[_candidate] = Candidate({ id: payable(_candidate), vote_count:  
        0, voters: new address payable[](0)});  
    voting_condition.candidate_count++;  
    candidate_list.push(_candidate);  
    souls[_candidate] = Refund(msg.value, _candidate);  
    emit CandidateCounts(voting_condition.candidate_count);  
}
```

Further the main change was made in the function called mayor_or_sayonara(). This function has multiple loops for iterating the candidates for vote's counts and a sub loop for iterating the votes to calculate the sum of souls. This data structures help in calculating the win or tie situation based on sum of souls followed by vote counts (if required). The mayor_or_sayonara() function calculates the win and tie

situation and it further calls the function called WinDraw that further proceeds with souls managements based on win tie calculated by mayor_or_sayonara() function.

For the frontend we initialized the instance and accounts objects in a single file which can be used to retrieve data for rest of the project files. A single source files for making connection to backend and initializing objects.

```
import Web3 from "web3";
import truffleContract from "@truffle/contract";
import MayorMultipleCandidates from "../contracts/MayorMultipleCandidates.json";

const ganache_address = "http://127.0.0.1:8545";
const web3 = new Web3(new Web3.providers.HttpProvider(ganache_address));

const accounts = async () => {
  return await web3.eth.getAccounts();
};

const contract = async () => {
  const contract = truffleContract(MayorMultipleCandidates);
  contract.setProvider(web3.currentProvider);
  const instance = await contract.deployed();

  return instance;
};

export default { contract, accounts, web3 };
```

Rest, we made a login page and the home page. Login for registering voters and candidates and home page for showing stats and results. Both uses above file for performing operation using contract instance.

Use Cases:

The project has two side, the backend consists of Smart Contract written in Solidity and deployed on local blockchain and the frontend part.

The project has two types of users: CANDIDATES and the VOTERS

Voters can vote for their favorite candidates and can invest the souls (ether). The more the soul amount, the more changes the candidate will win.

Each vote consist of an envelope that contains the SIGIL, the soul amount, the symbol (address of the candidate they are voting).

Candidate will invest some amount ether > 0 . This ether will be divided equally between its voters if the candidate has won. If the candidate lose, all the loosing candidate's ether (souls) will be transferred to the winning candidates. In case of a tie, all the ethers will be transferred to escrow.

The result is computed based on soul amount the voters had invested in their candidate and if in case of tie here, the number of votes are counted and the winner is selected. If still the same based on soul amount and vote count the result is a tie and the souls are deposited in the escrow.

Smart Contract Implementation:

Smart contract has the following functions implemented:

- **constructor** takes the escrow address (the first account from the Ganache account list automatically) and the quorum. These two values can be set in the migration file under migration folder at the root of the application.

- **add_candidate** function is used for registering candidate in the application.
- **get_candidate_count** function returns the candidate count.
- **get_candidate_vote_count** function expects the address of the candidate and returns the vote count.
- **cast_envelope** function takes the envelope and register it into the application.
- **open_envelope** function takes the sigil and the candidate address and open the envelope.
- **get_candidate_using_idx** function takes the array index number and returns the candidate address.
- **check_result** function checks the voting result, returns the bool values in-case of win or tie (true or false) and the winner address.
- **mayor_or_sayonara** function calculates the voting result and process the souls based on the scenarios mentioned above.
- **WinDraw** function is the sub-function of the mayor_or_sayonara function and do internal processing or souls.
- **compute_envelope** function takes, SIGIL, candidate address (symbol) and the soul and returns the hash of the envelope.

Front-end Implementation:

Front-end is implemented based on ReactJS standard library that uses react supporting libraries like, react-router, react-router-dom for navigation, reactstrap for bootstrap responsive design. It also uses web3 and truffle-contract packages to support operation on ethereum smart contract network.