Decentralized Voting System

By

GROUCHE

Takai Eddine Mohammed

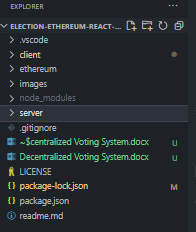
A voting decentralized application developed using React, Express, Web3, Solidity and Ganache.

**Introduction**

This voting project mimics the Indian Election Process

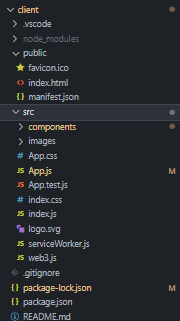
React, the standard framework for building web application frontend was used to bootstrap the web UI for this project. Also, Express, the standard Nodejs/JavaScript backend framework was used to route user requests buy connecting to the Ganache local Ethereum network backend.

User requests are made through React components and controls and sent to the Node-Ethereum combo backend using Axios and Web3. After bootstrapping the dapp, I simply have the following directory arrangements:



* Obviously, VS Code was used as my code Editor, thus .vscode contains necessary configuration script.
* The client directory on the other hand contains my react front end implementations.
* The Ethereum directory contains my voting smart contract written in Solidity
* The server directory contains my Nodejs implementation.
* Other files contains configuration scripts needed for the successful running of my dapp.

**The Client Directory**



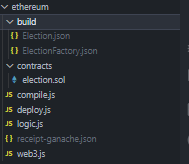
As seen in the image, the client contains the public folder for holding static files such as html and css files.

The src folder consists majorly of React/JavaScript source files. Note that these source files are arranged based on what they do, e.g. components subdirectory to hold react component source files, etc.

I also have the App.js which serves as the first or root component and containing the routing information of my React dapp, whereas index.js is the entry point, through which App.js or the root component is mounted.

Finally, I have my package files for configurations.

**The Ethereum Directory**



There are 2 contracts. First is ElectionFactory and Second is Election. Both contained in the contracts/election.sol file. All the Election related information will be created using ElectionFactory contract, whereas the Election contract contains functionalities to perform the actual polling.

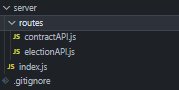
This directory also contains the compile.js and deploy.js files for compiling and deploying smart contracts to the blockchain.

Upon compilation (see compilation method below), a build directory is created with the compiled codes in json format for processing and interfacing with JavaScript through the Web3 framework.

Server side Web3js is used for this Dapp. All the transaction will be signed by server.

Client side Web3js will be added soon.

**The Server Directory**



The server directory contains the Nodejs implementation such as the routes to handle user requests and connectivity to the blockchain. It also contains the index.js file which serves as the entry point for the server.

**System requirement**

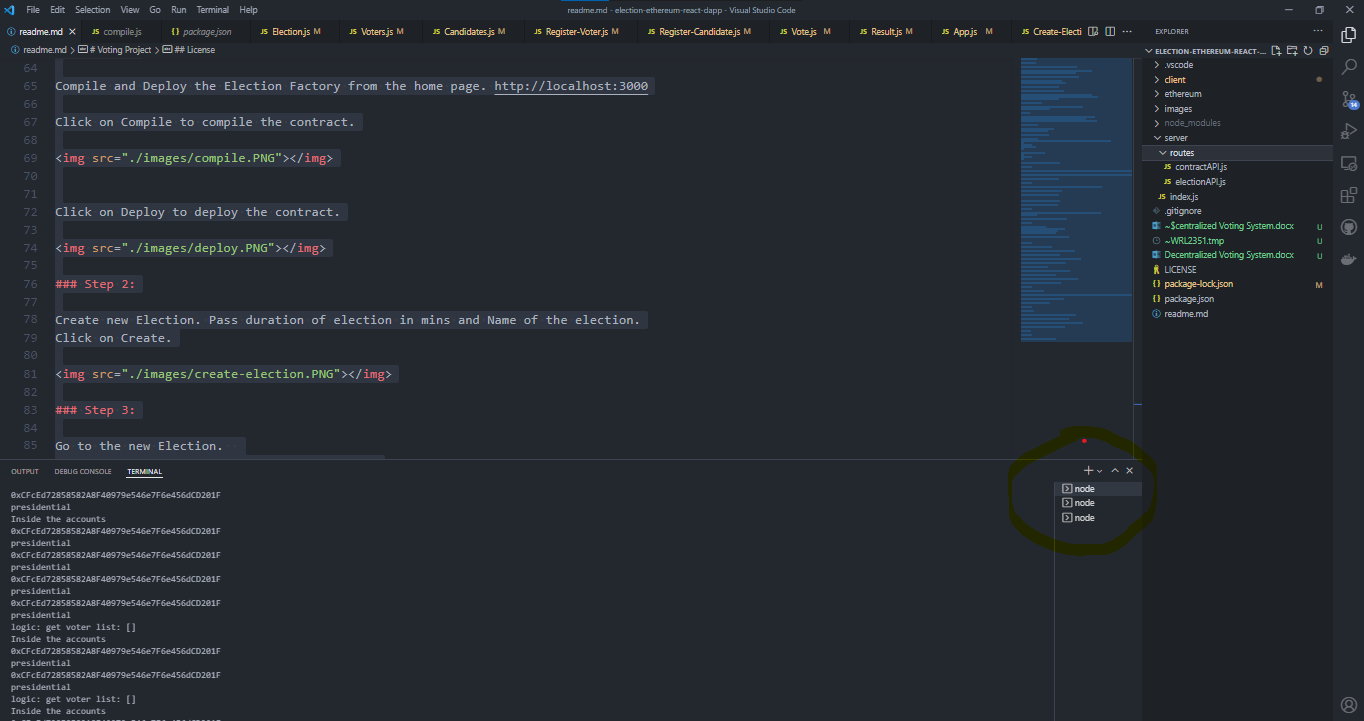
1. nodejs 8.x or greater (This dapp is developed on node v8.x)
2. npm install -g ganache-cli

**Set up**

1. npm install (from the root folder to install all the server and ethereum dependencies)
2. npm install (From client directory to install all the react dependencies)

**Start the DAPP**

Open 3 command window or terminal From Project Root Directory. You can do this so easily using VS Code.



* In Terminal 1: enter the following command to start the ganache local blockchain on your machine:

>ganache-cli

* If you wish to compile and deploy the smart contracts yourself, you can optionally delete the build folder and then in Terminal 2, take the following steps to compile and deploy the smart contracts to Ganache:
  + Step 1: Compile the contract. It will compile and create the binary in the build folder.

>cd ethereum

>node compile.js

>node deploy.js

* Step 2: Run the server using

>npm run dev

However, since the deployment has already been done by me, opening the Terminal two is optional. More so, there are buttons in the web UI homepage to also compile and deploy the smart contracts should you prefer to run the compile and deploy functions by clicking buttons other than using commands in the command-line interface (CLI). It is easier to click than to type after all!.

* In Terminal 3, run the following command to start the web UI:

>cd client

>npm start

**Election**

Once the React web UI loads and opens (having followed the steps stated above), it is time to set up elections and start voting.

Every election is active for n minutes, where n is the duration of election. Duration of election in mins and name of the election must be passed during deployment, i.e. in the homepage of the web UI.

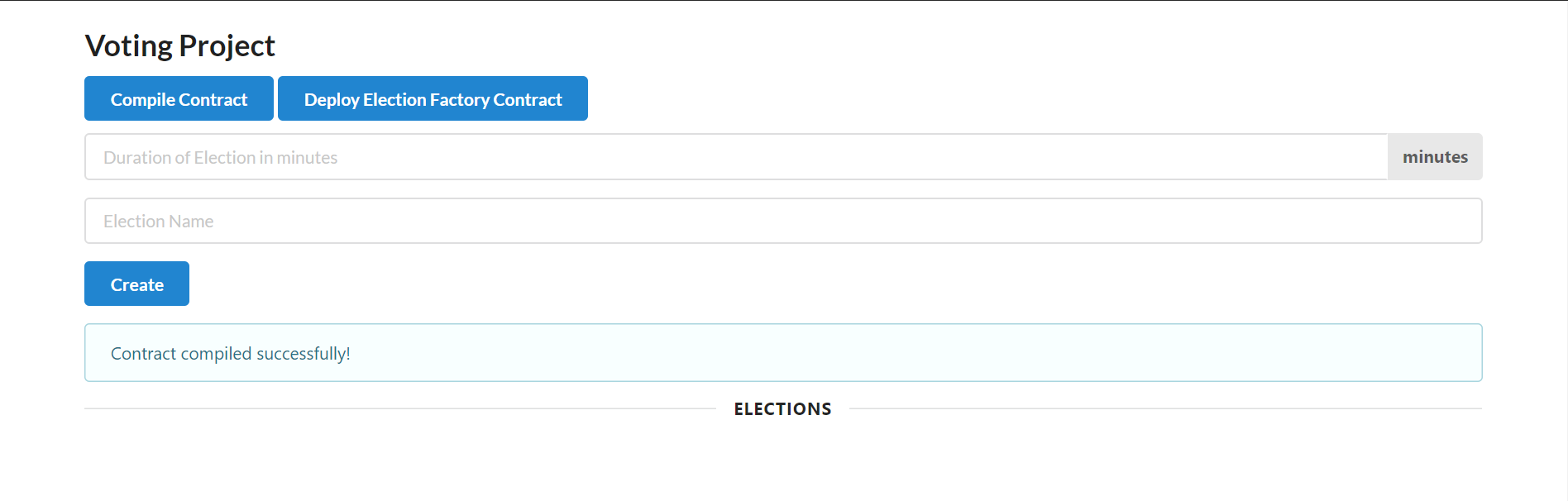
All the Create operation (Create Consituency, Voter, Candidate) and Close Election will be done by the Admin only.

Admin can't be a voter or a candidate.

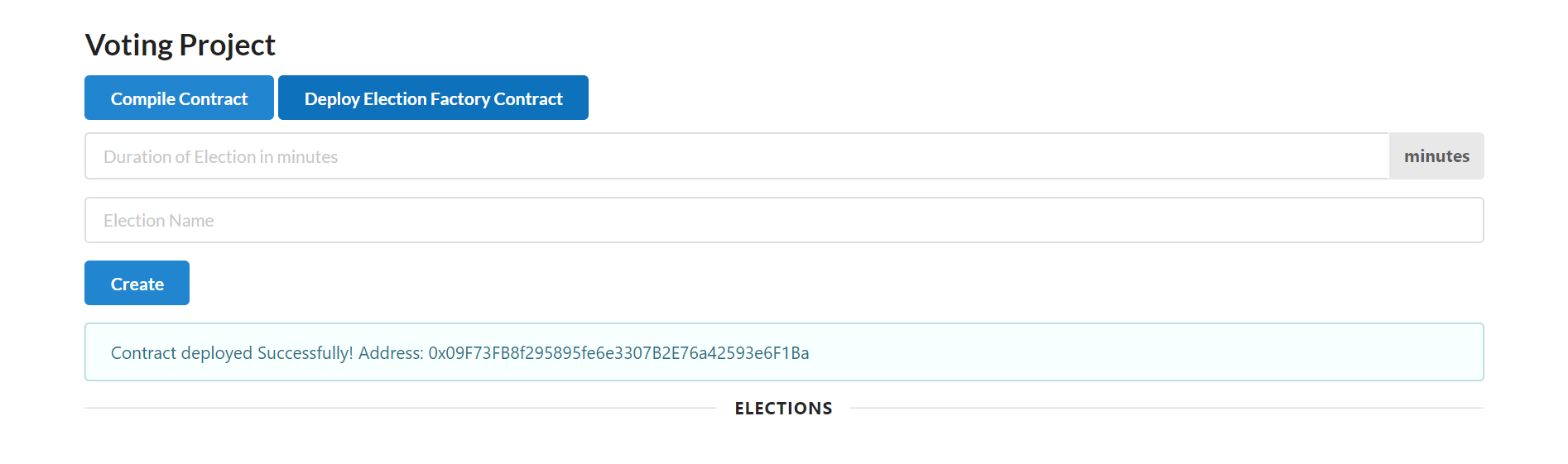
**Step 1**: **Compile and Deploy the Election Factory from the home page.**[**http://localhost:3000**](http://localhost:3000/)

(Optional. Must be done if you have not compiled and deployed your app using the steps mentioned above)

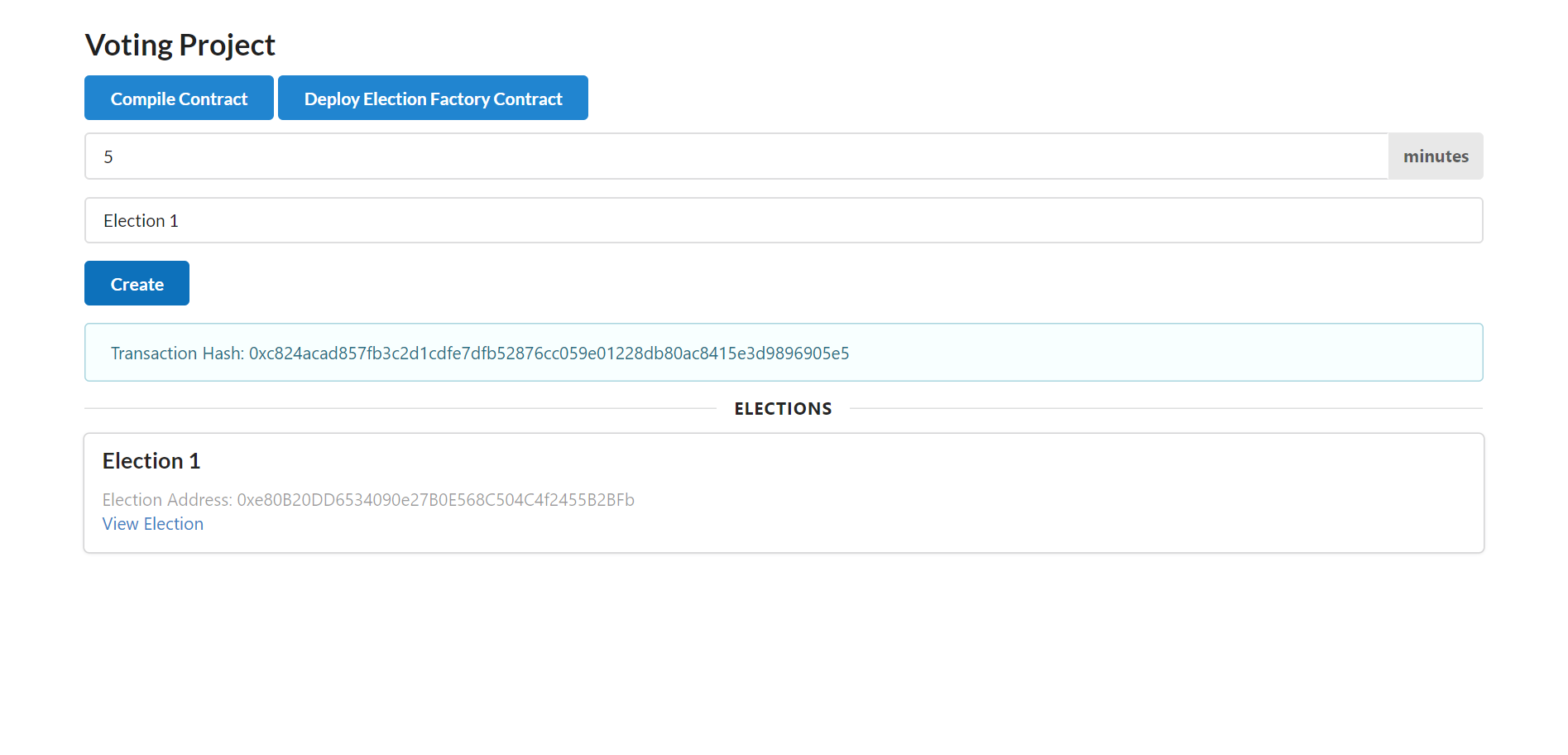
Click on Compile to compile the contract.

[](https://github.com/schadokar/election-ethereum-react-dapp/blob/master/images/compile.PNG)

Click on Deploy to deploy the contract.

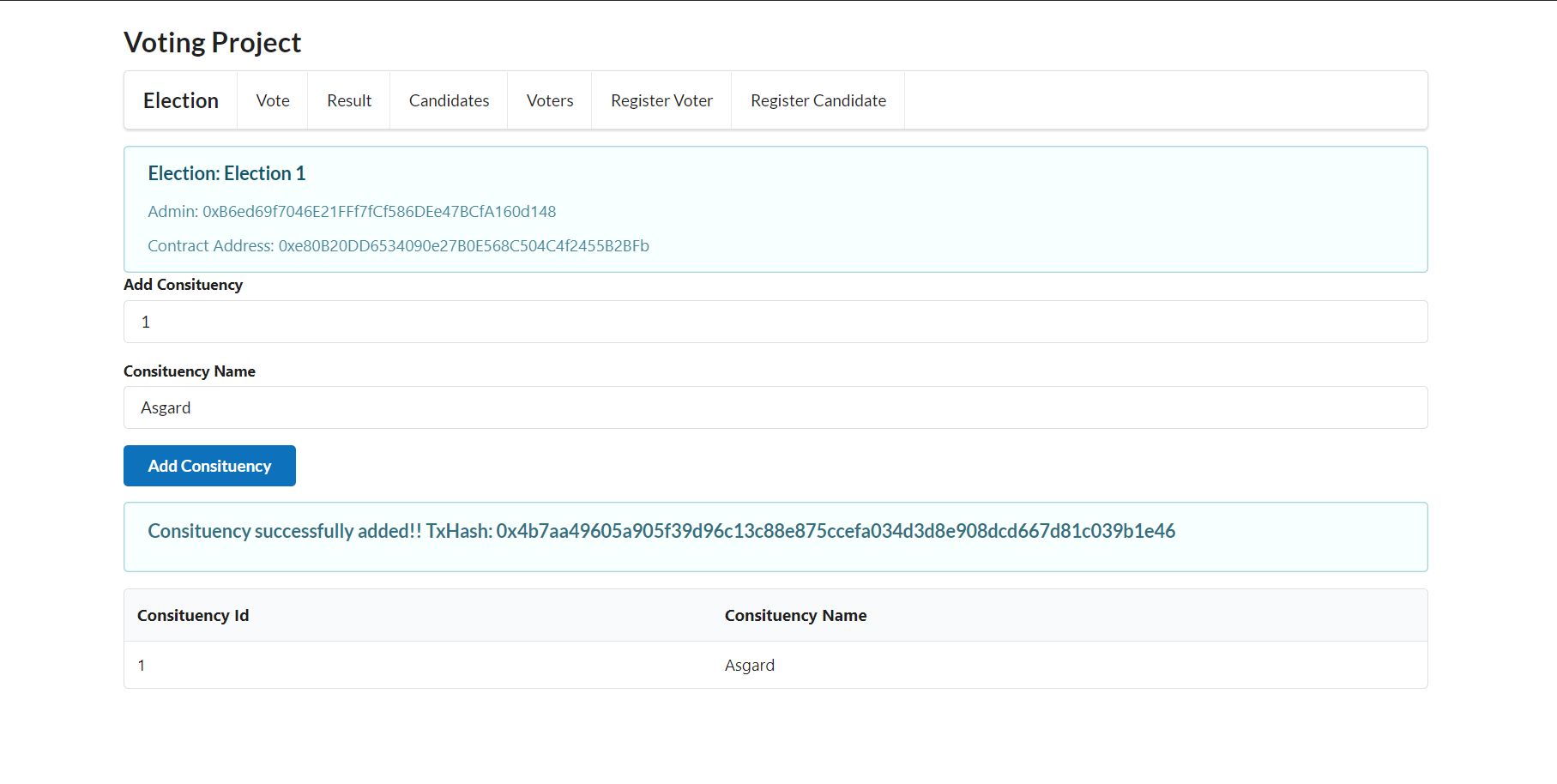
[](https://github.com/schadokar/election-ethereum-react-dapp/blob/master/images/deploy.PNG)

**Step 2: Create new Election. Pass duration of election in mins and Name of the election. Click on Create.**

[](https://github.com/schadokar/election-ethereum-react-dapp/blob/master/images/create-election.PNG)

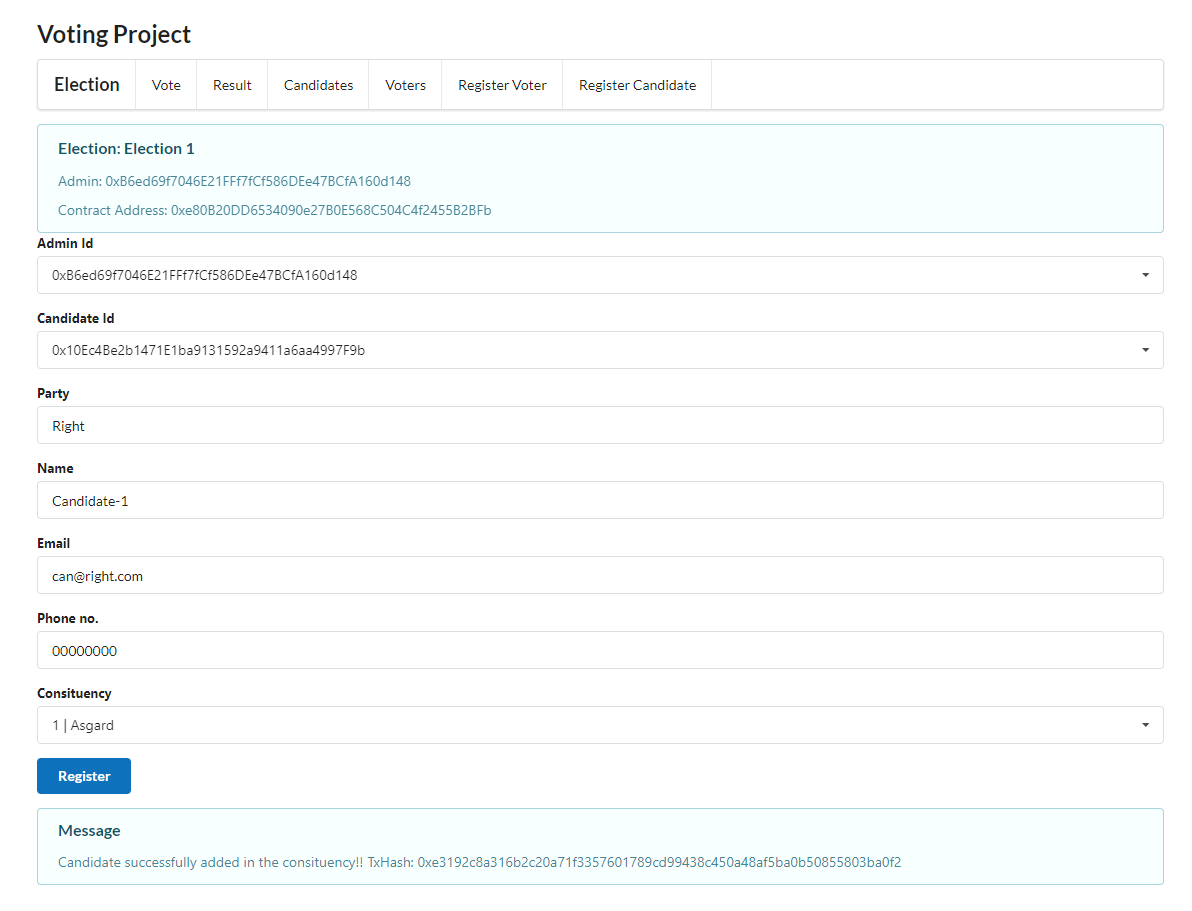
**Step 3:**

**Go to the new Election.  
Create the Consituencies for the election.  
Pass the consituency Id -- Integer  
Pass the consituency name. String**

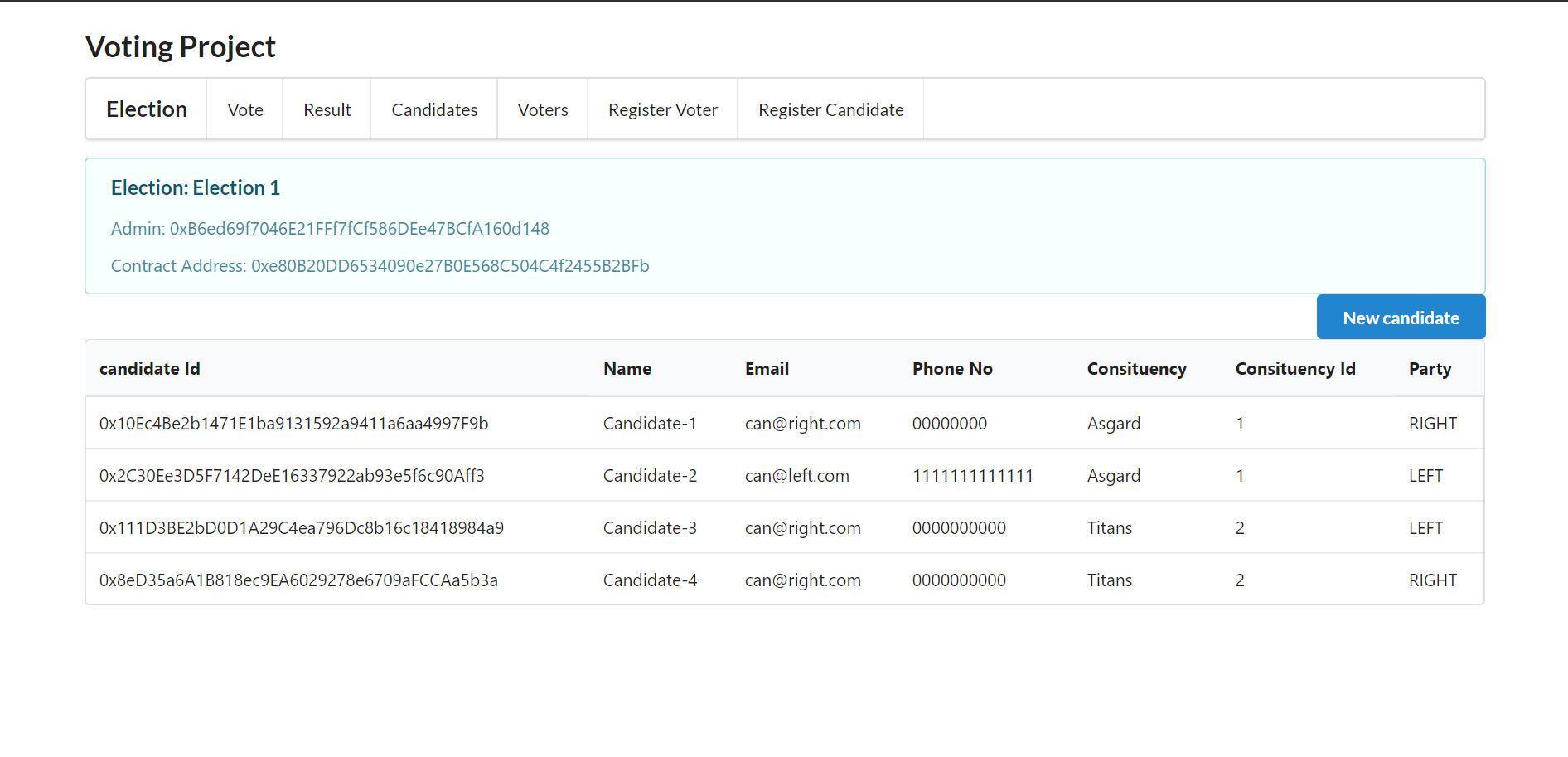
[](https://github.com/schadokar/election-ethereum-react-dapp/blob/master/images/add-consituency.PNG)

**Step 4:**

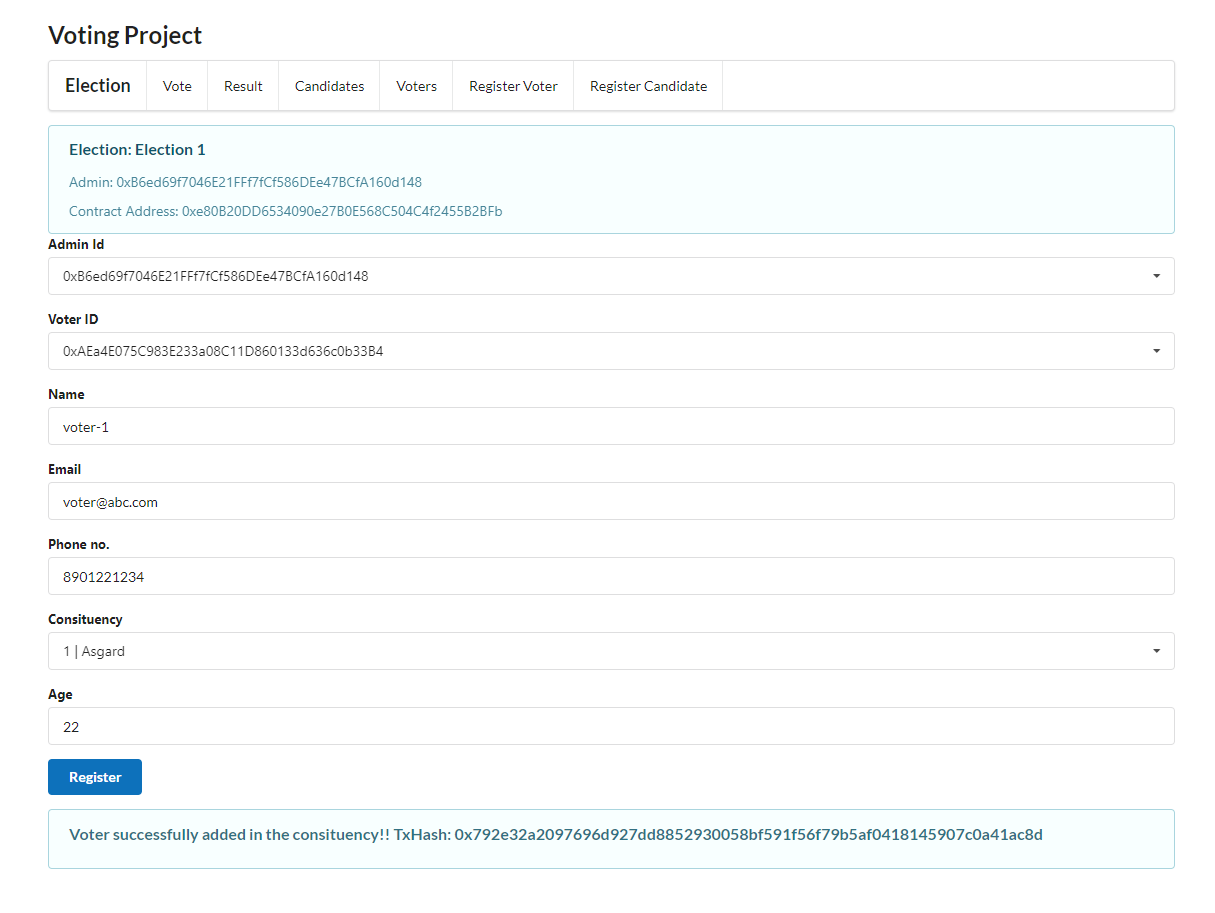
**Click on Register candidates**

[](https://github.com/schadokar/election-ethereum-react-dapp/blob/master/images/register-candidate.PNG)

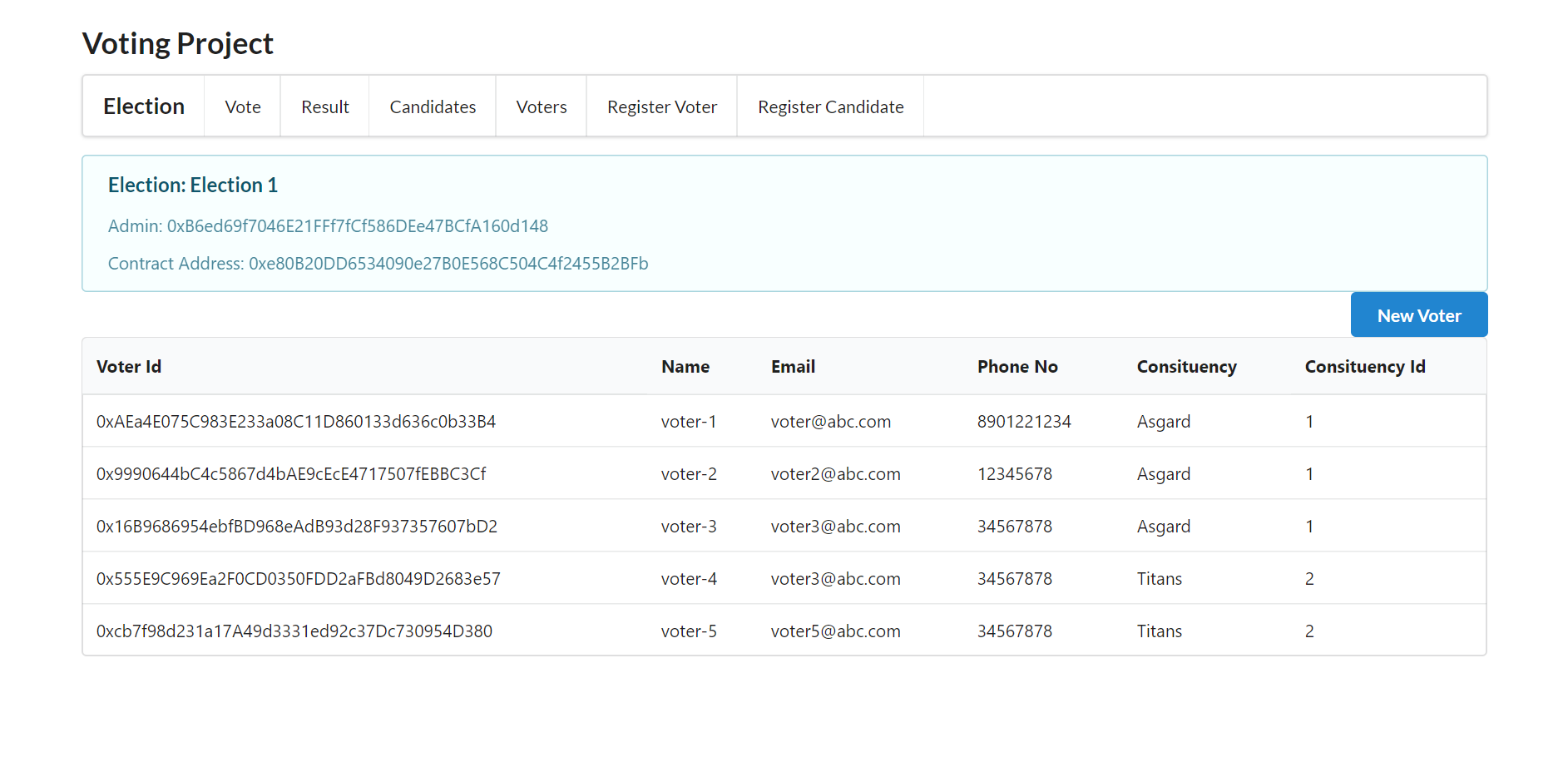
I have created the extra candidates. Click on Candidates to see all the registered candidates.

[](https://github.com/schadokar/election-ethereum-react-dapp/blob/master/images/candidates.PNG)

Click on Register voters

[](https://github.com/schadokar/election-ethereum-react-dapp/blob/master/images/register-voter.PNG)

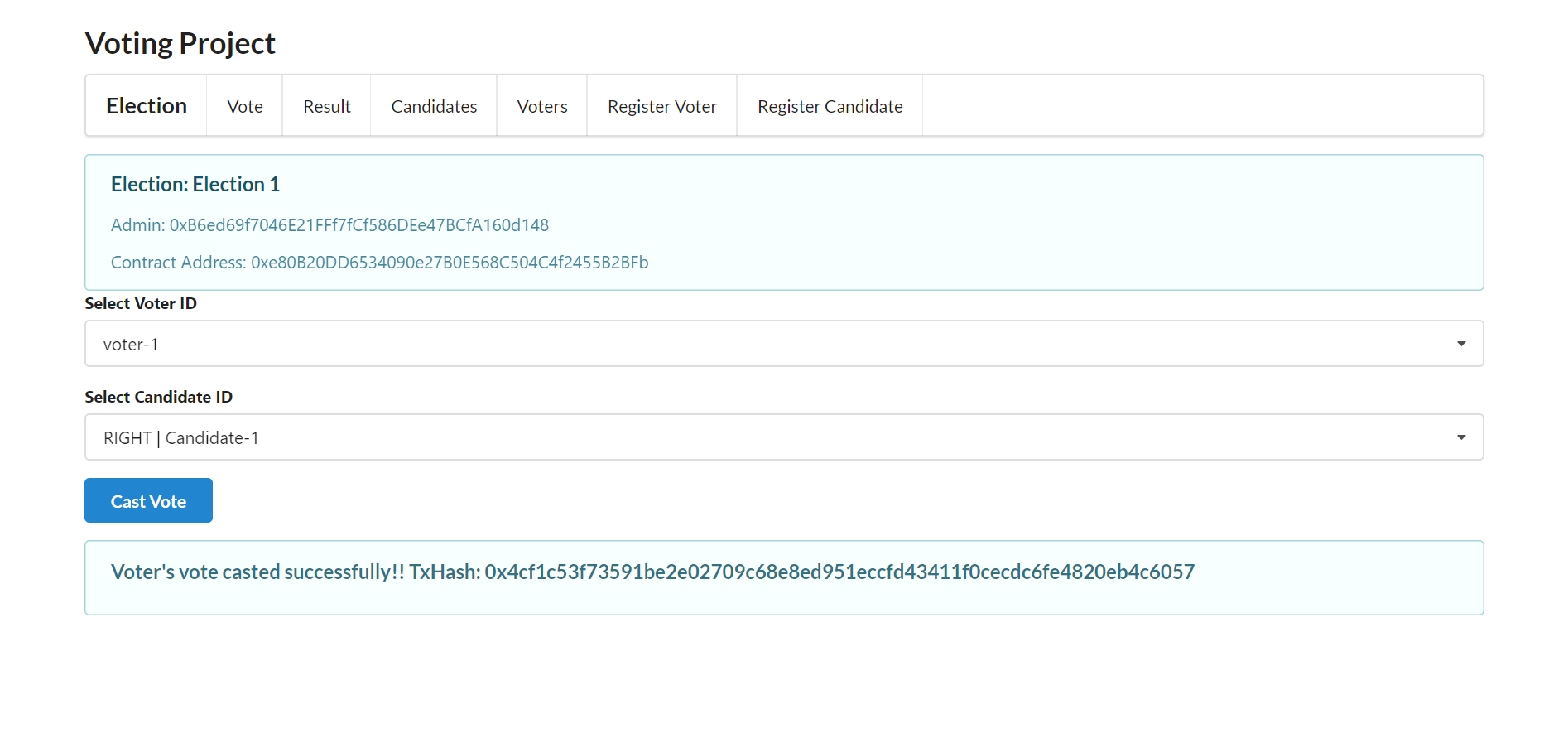
I have created the extra voters. Click on the Voters to see all the registered voters.

[](https://github.com/schadokar/election-ethereum-react-dapp/blob/master/images/voters.PNG)

**Step 5:**

**Vote Tab: Cast Vote**

Cast your vote by selecting the voter. Once the voter is selected its respective constituency candidates will gets loaded.

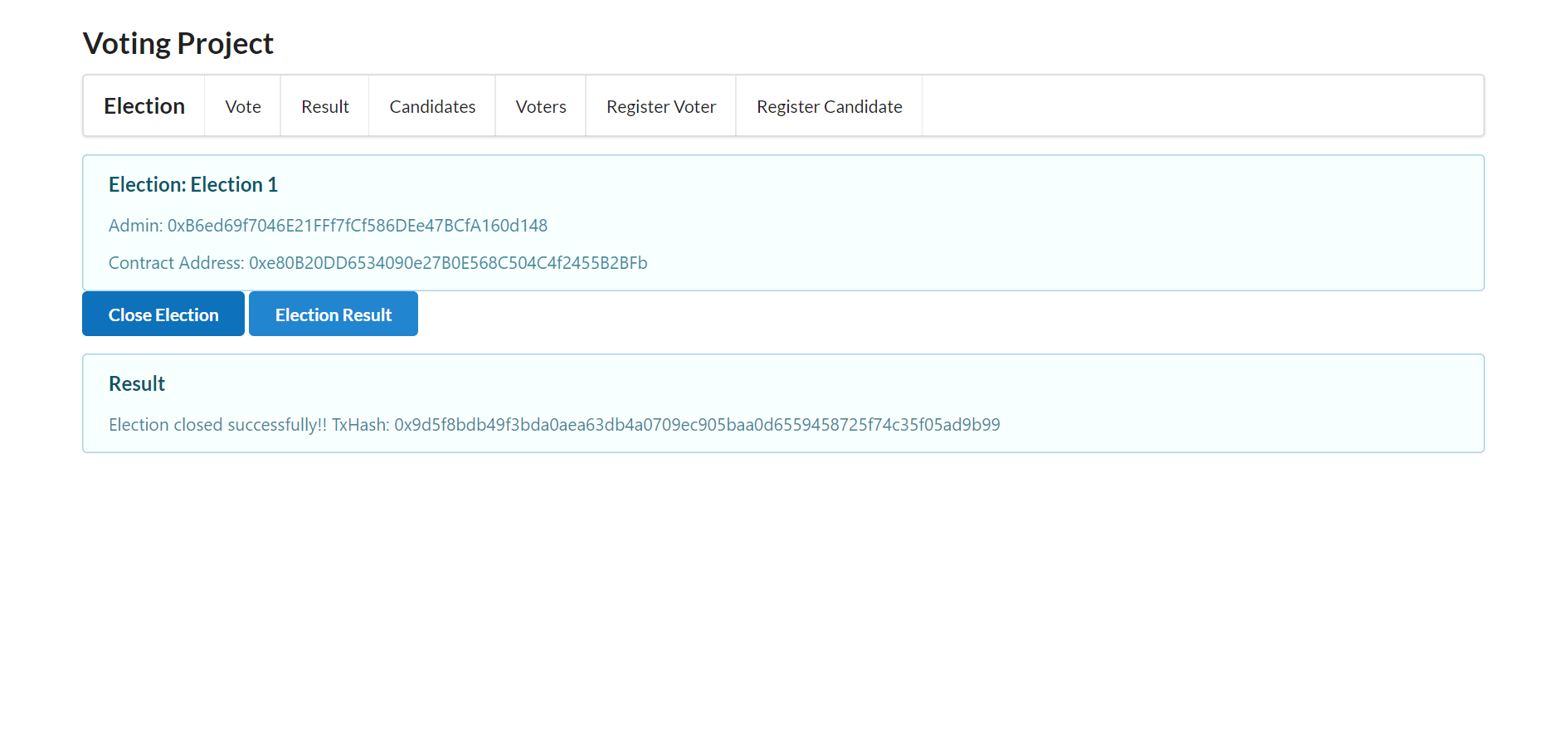
[](https://github.com/schadokar/election-ethereum-react-dapp/blob/master/images/cast-vote.PNG)

I casted the remaining votes.

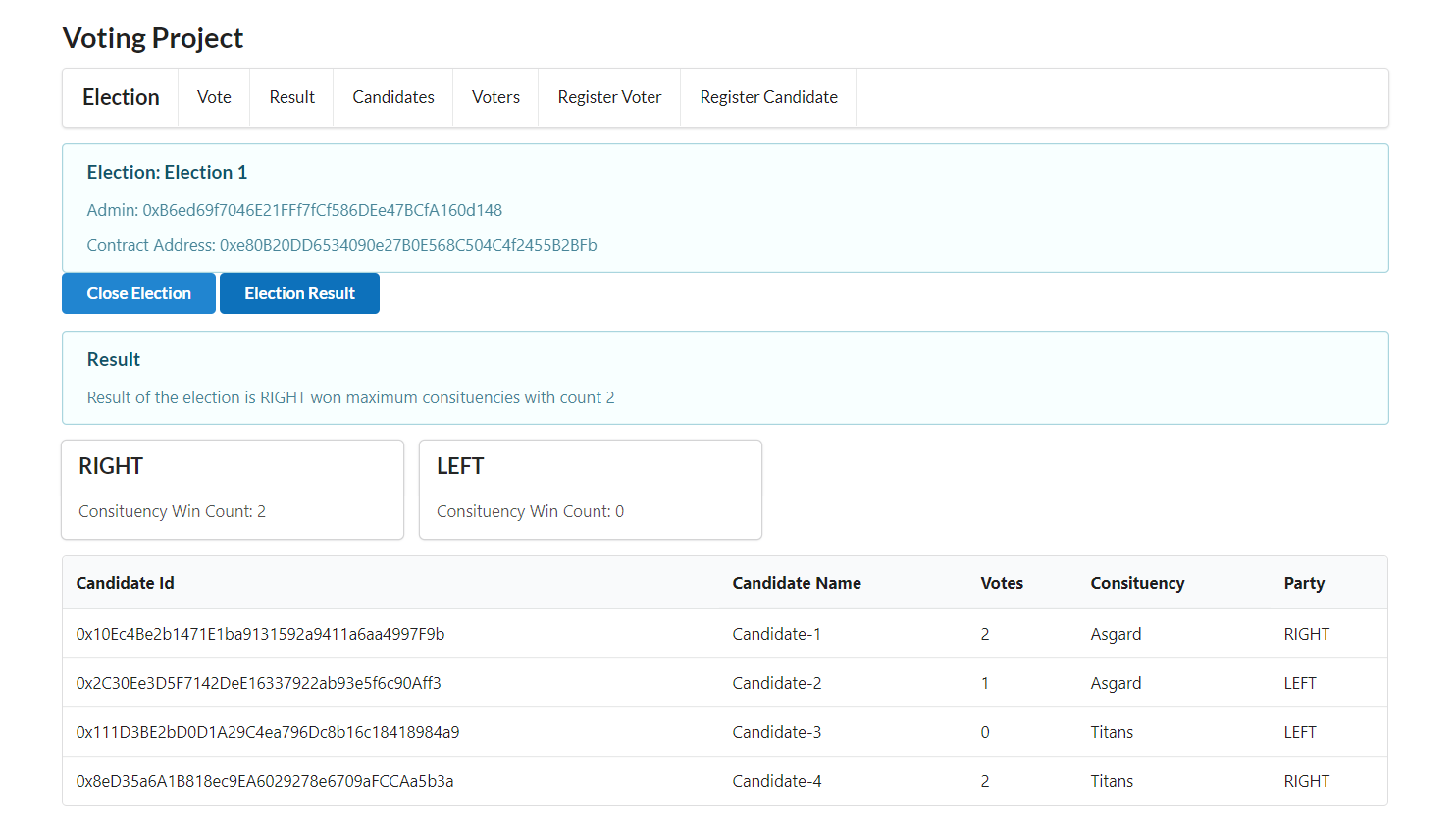
**Step 6:**

**Result Tab:**

1. Click on the Close Election to close the election

[](https://github.com/schadokar/election-ethereum-react-dapp/blob/master/images/close-election.PNG)

1. Click on the Election Result to get the election result.

[](https://github.com/schadokar/election-ethereum-react-dapp/blob/master/images/voting-result.PNG)

**Limitations**

1. Revert message from contract is not replicated on React application.
2. It is assumed that in every constituency, one party must get majority votes.
3. Form Validation is not implemented.

**Reference**

https://github.com/schadokar/election-ethereum-react-dapp