Developing a Scalable Blockchain Voting Solution

Submitted by

Anthony, Biancia, Shunqi

Undergraduate and Graduate

Computer Science

Gildart Haase School of Computer Sciences and Engineering

A project submitted in partial fulfillment of the requirements

for the course - CSCI 4030 / 6830 Blockchain Technology.

Under the supervision

of

Dr. Avimanyou Vatsa

Assistant Professor, Department of Computer Science

(Summer 2020)

**The problem statement**

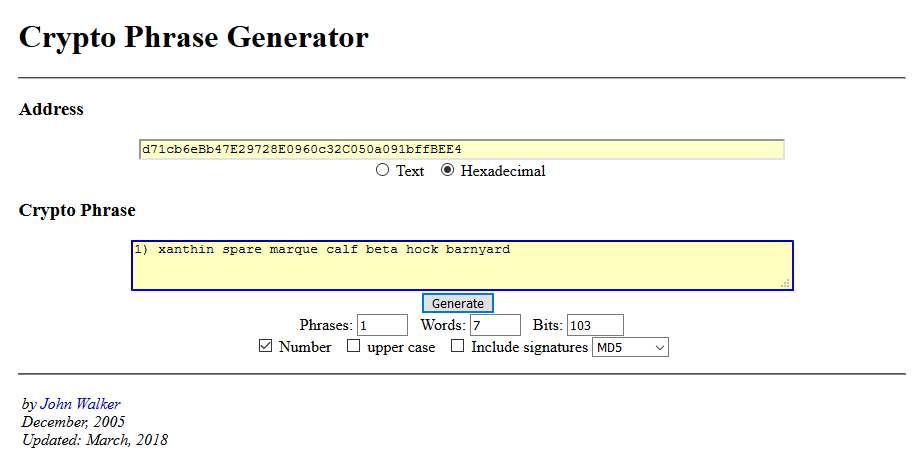
There is an issue of incorrectly counted ballots at voting polls, which in turn results in the misidentification of the candidate the public has selected to be their new representative. Following the closing of most poll booths, there is always a report of ballots needing to be recounted, mishandling of the collected information and the reduction of trust among citizens. Needless to say the lack of respect for the people that took time out to vote does not occur in only one state or from a preferred party, it occurs across many other states. The ability to cast a vote, as a citizen of the United States is a given freedom and therefore, citizens should be able to express themselves at their own will, without the after thought of the results being tampered with.

It is critical for a system with a seamless process to be developed to remove the multifaceted approach of giving too many third party access to voters information. This disparity is what this project aims to resolve, by developing a framework that will accurately count and record votes, while also limiting access to submitted voter data, to a select few. We understand that the voting in itself is already a stressful situation as a country opt to either stay with the current leader or move to elect a new one. As a technology, we’re beginning to realize that blockchain is an improvement to many old systems.

While blockchain voting systems have been developed in the past there is still a lot of progress that needs to be made until a widely deployable system can be viable for the public. Some issues that need to be resolved include improving ease of use/accessibility for voters as well as developing systems that can create an aspect of quality insurance for the results of these voting applications. Some Solutions that have been proposed include using crypto phrases, similar to what is used for account recovery with crypto wallets, as a unique key that can be tied to their public address and used in lieu of it to make voting easier. As well as developing a querying solution that can be used for users to pull up their voting history to ensure their votes are being counted for accurately as well as for individuals like voting authority to ensure elections are running as intended.

**Solution Statement**

To solve these issues three smart contracts were developed to fulfil different aspects such as voter registration, ballot creation/voting, and storage/access of voting history as well as to promote scalability with future use. The first smart contract is Register.sol (screenshots of code are found at the end of the paper) this smart contract registers voters using it’s register\_voter function that takes in the users address and a string value for the crypto phrase key then creates a voter object to save the users crypto key and set their voting permissions to true then maps the voting object to the address of that user. To generate the crypto phrase key a modified version of John Walker’s “Pass Phrase Generator” was used. While the backend remains very similar to the original a slightly modified front end was used to fit our purposes. (Shown Below)

From here you can see a user's hexadecimal public address entered in the top bar and the bottom bar generated a unique crypto phrase. The other functions are used by Ballot.sol to confirm voter’s credential, i.e. if their crypto phrase matches and if they are eligible to vote.

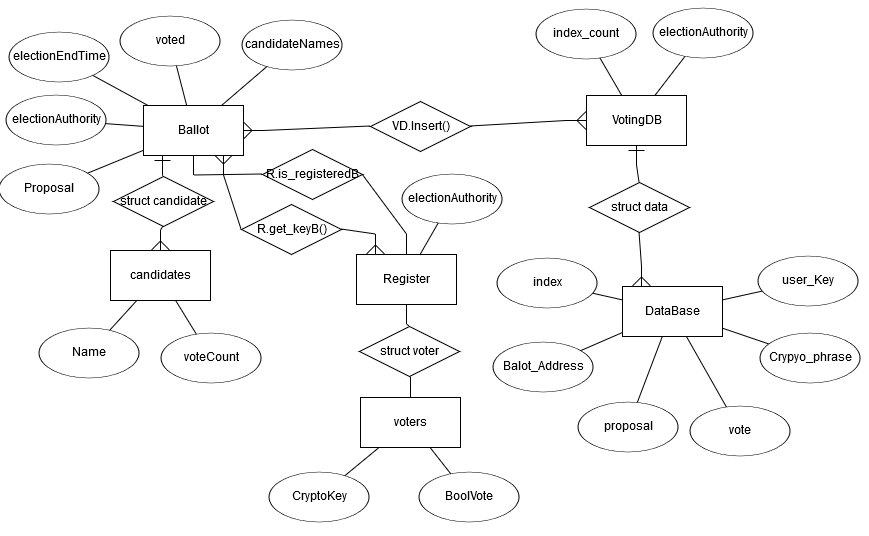
The second contract is Ballot.sol in this contract the ballot gets created and voters can vote, have their vote recorded and view the winner once the election ends. When the contract is initialized with the constructor function three parameters are required. The first is a string value of the proposal or what is being voted on. The second is a string array of the candidates or options to vote for. The third is an integer value for the length of the poll. The addresses for the register.sol and voteDB.sol smart contracts are imputed using the get\_add function as this smart contract is reliant on functions in those contracts to operate. At this time the addresses for the register contract and votingDB contract get saved to reference later and the endtime for the election gets set as well. The Vote function is where voters cast their votes for the election in this function the voter enters their vote and their crypto phrase key. The function then references register.sol to determine if the voter is using the valid crypto phrase for their address and checks if they are eligible to vote. If both these checks are passed then the vote is counted and a function from the third contract (VotingDB.sol) is used called Insert(), which passes the proposal, what the voter voted for, the voter crypto key, the voters public address, and the address of ballot.sol to VotingDB.sol where is is saved and stored. Once the election ends the winner can be found with this contract as well.

The last smart contract VotingDB.sol is where the data about user votes are stored. The data is entered from the Ballot.sol smart contract as described above, once the insert function is used it then takes the data entered, forms a data object with that information and adds an index value to keep track of the number of entries inserted. Once this object is created it gets pushed into Database which is an array of the data objects mentioned before. This smart contract has 3 query functions that work identically just accept different inputs to query by. The three query options include searching based on users crypto key, searching based on a users public address, or searching based on the address of the ballot contract. When a user queries based on one of those three parameters, the Database array will be queried through and look for matching instances in the array. When a match is found each of the values in the associated data object get added to their corresponding return array then the indexinging continues looking for matches until the database ends. Since solidity as a programing language is unable to return object arrays from it’s functions work around had to be developed, to get around this each quarry function has an array for each of the variables being returned, Ballot address, user address, user key, user’s vote, and proposal being voted on, then each of these array are returned as the result. The final function in the VotingDB smart contract is the Full\_db() function; this function will return the entire database in a similar way to the quarry function previously described.

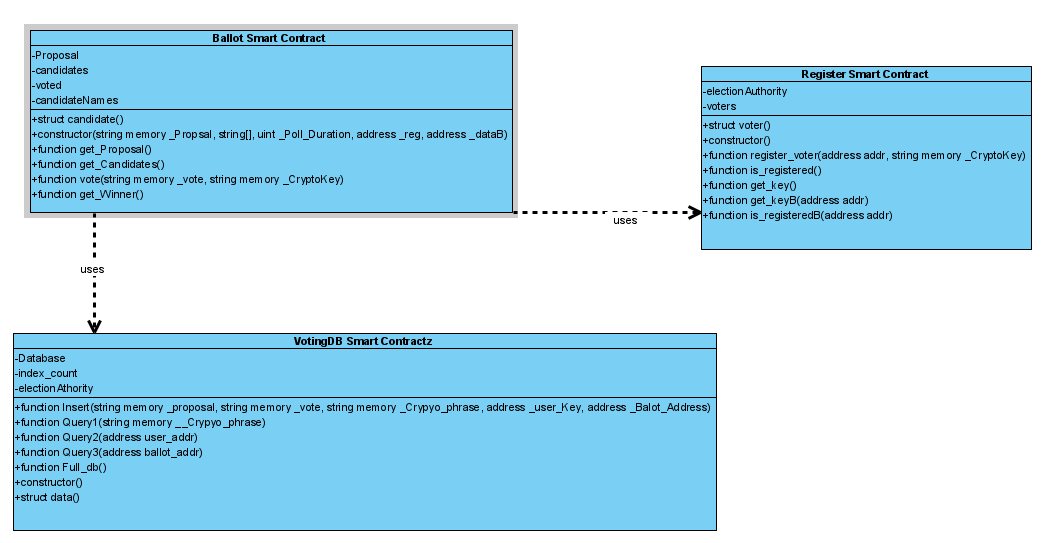
All of the smart contracts utilize modifiers, these modifiers are functions that check for specific requirements either about the user of the contract, parameters about the contract, or to pull then compare aspects of other contracts. These modifiers essentially used as gates to deny or allow users to utilize a function based on specific requirements.

**GUI Design**

For smart contracts GUI, there are many pages: one homepage, one set up subpage(for individual and authority), input and output of the user shown results page, and one voting database page. Used html, css, bootstrap and jquery in UI, to show the function values of get\_key( ), is\_registered( ), get\_proposal( ), etc. Once inputting related values in html, they would pass the values from the javascript interface, to the test network, and then get the values from that test network via Web3.js (which is . By using Web3.js, the ABI can be used to pass the values from these setup functions.



**Software Diagram Modeling:**

The image to the left shows the ER diagram of how data is stored and connected within the smart contracts. All data storage takes place on the chain in smart contracts and because of this all data relationships represented can be assumed to be on the blockchain. It is ultimately the “DataBase” data structure where the necessary data from the Register contract and Ballot contract will be passed to for users to access. The register smart contract holds the data about voter information through the voter data structure that holds a voter’s cryptokey and a boolean value that states if they are able to vote, then this data structure is mapped to the users public address when they register to vote. The Ballot smart contract holds all relevant information to the election taking place such as the Proposal, electionEndTime, candidateNames, voted(a boolean array that maps what addresses have voted) and a data structure candidates that keeps track of each candidate and how many votes they have. 

The UML diagram shows an overview of how the contracts interact with each other. The Ballot contract both relies on the use of functions in the Register and the VotingDB smart contracts. To register the Ballot contract needs to access voter information in the register smart contract and thus Ballot relies on Register to perform some of its functions. With VotingDB, Ballot needs to use it’s insert() function to send the voting data into its database.

**Outcome Description**

For outcome, we saw the ballot, voting and register contracts resulted in the proper collection and tallying of votes during the election period. Instead of there being conflict between voters and the authorities counting the votes, you now have a more seamless process whereby only the voters have access and control of the votes cast, and overall, the voting process has become a more secure individual process.

The main issue was that votes were not locked and voting authorities had the opportunity to make edits to reallocate votes to the desired candidate. As a result of the new contract system, votes are locked in by the voter without possibilities of there being any future alterations.

In addition our contracts correctly mapped voters addressed with their respective crypto phrase keys both when voting and when used to look up a voter's history using the voting DB contract to query, querying was also successful based on searching by ballot address as well.

Future aspects of this project, we hope to develop a platform that is scalable, reliable and transformative to be able to accommodate the increased number we expect to see from voters as the voting system evolves into a platform that is trusted. Our hope for the future is to see more and more young people get involved by making their voices heard. In addition to this, we hope to remove the ever growing layer of authoritarian control expressed by the authorities during the voting process.

While this platform is expected to reduce the level of intricacies experienced during elections, we understand its introduction may also introduce a new set of problems. Some other possible problems we hope to address in the future would be to allow for more visibility in who a voter is and to be able to correctly verify that the individual is unable to cast a vote for more than one candidate.

**Future implementations**

While this project has laid a platform for scalability in voting systems there are many more improvements that could be made to this BOS in future implementations. One glaring issue is with security the way many of the functions are set up many of the commands can be done right from the console. Adding modifiers and user permission functions would need to be added to each of the contracts to maintain security integardy.

Originally proposed as a feature that would allow users search on-chain data about voters and ballots with SQL-like functionality using the EQL framework, however this became difficult to accomplish. There was a considerable amount of time spent researching EQL and it’s most recent iterations, however once found the source code was written in a language called Pharo. Then to utilize Pharo with the ethereum network required additional projects and plugins. While these projects are publicly available there are little resources for individuals on how to implement them into their own work. While this was an original requirement there were other requirements that we aimed to reach for this project and it did not seem feasible in the amount of time left till the due date to learn Pharo/figure out how to implement EQL and develop the other parts needed for the BOS software. As a compromise the voting DB contract was made so that querying based on crypto key could still be deployed. In future implementations this would like to be explored more and hopefully be implemented.

There were complications when trying to connect the smart contract with the GUI. Because of this in the current version the GUI isn’t functional with the BOS. Having the GUI integrated with the smart contracts is planned to be implemented at a later date.

**Details about the hardware (your machine) and software (include name & version, Ethereum environment & solidity) used in this project**.

The hardware that was used in the process of these developments are a windows 10 computer. The software used was the Ethereum Remix IDE for prototyping smart contracts and the Atom IDE for preparing contracts for deployment. The BOS was deployed using the Truffle development environment on the Ganache personal blockchain through the use of the powershell terminal. The latest Truffle version 5.1.37 was used and Solidity 7.0 installed with VotingDB & Ballot smart contracts running “pragma experimental ABIEncoderV2” and Register smart contract running “pragma solidity >=0.4.0 <0.6.0” For the UI, vscode editor was used, language html, css, javascript, bootstrap4, and jquery (web3 included) were used.

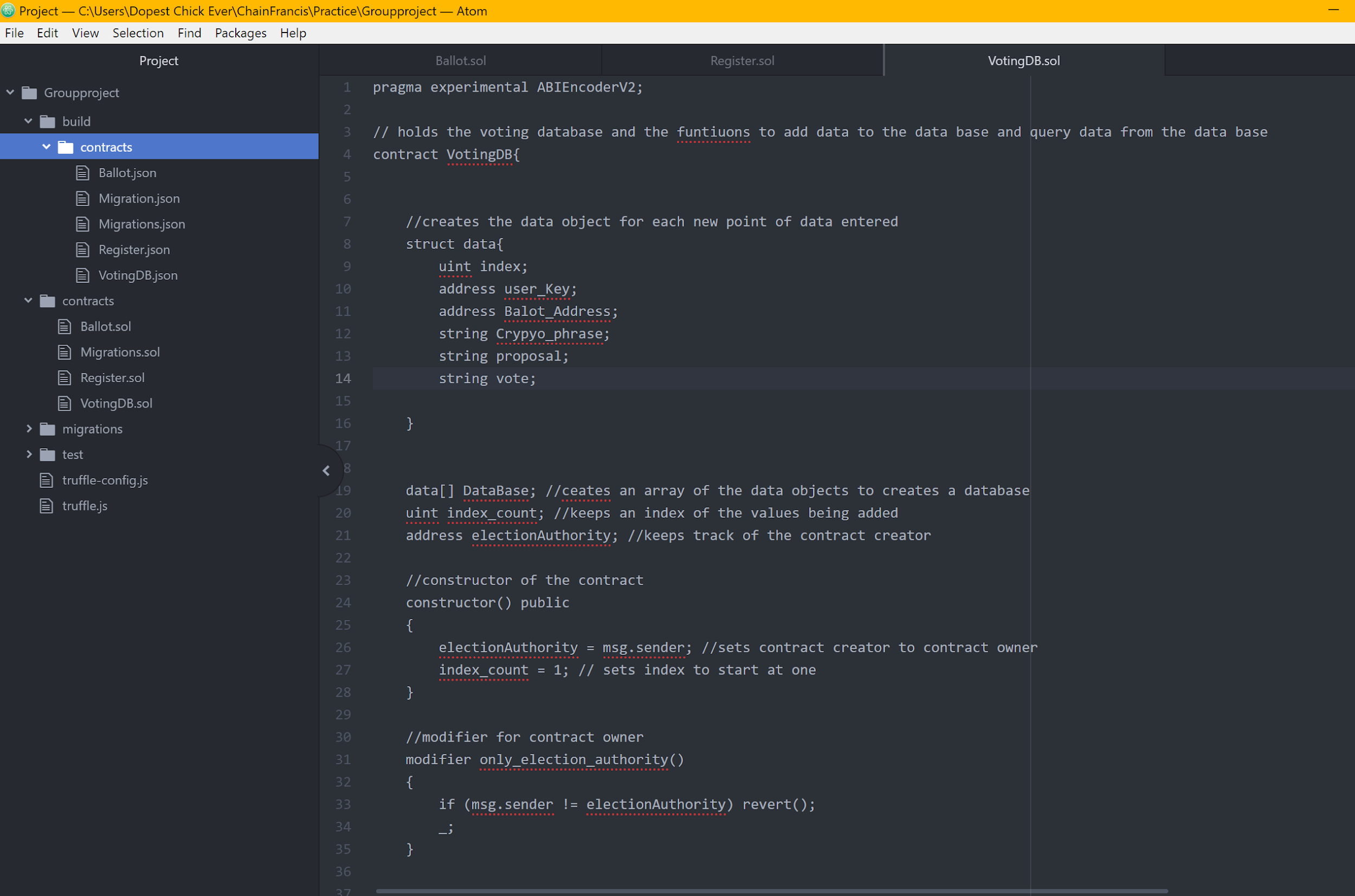
**Contributions**

Anthony: Smart contract writing/deployment, Crypto Phrase GUI editing, model creation, paper writing

Biancia: Paper and presentation organization, writing, and editing, Crypto phrase research

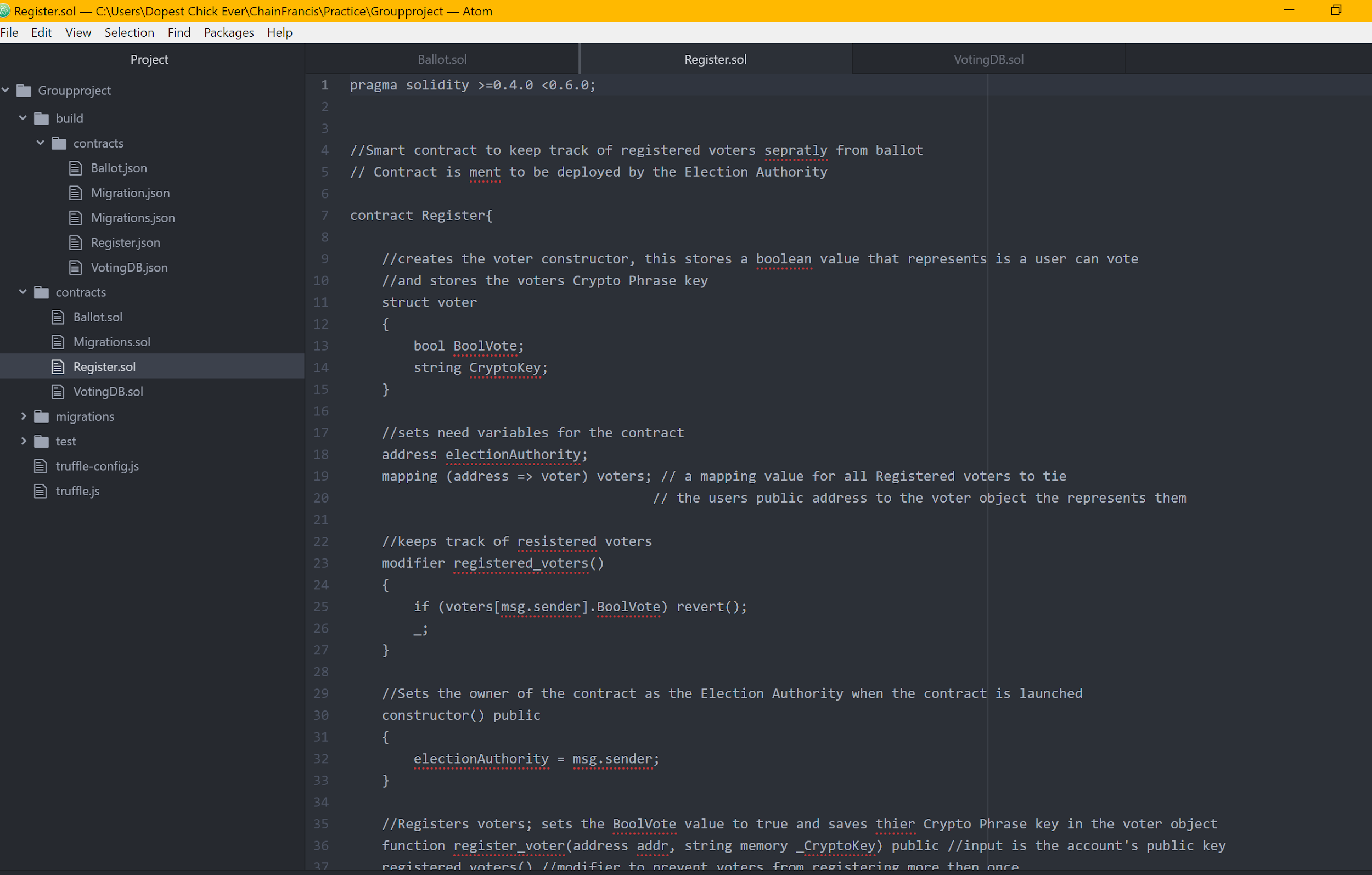
Shunqi: B.O.S. GUI and smart contract connection

Images/Screenshot of work

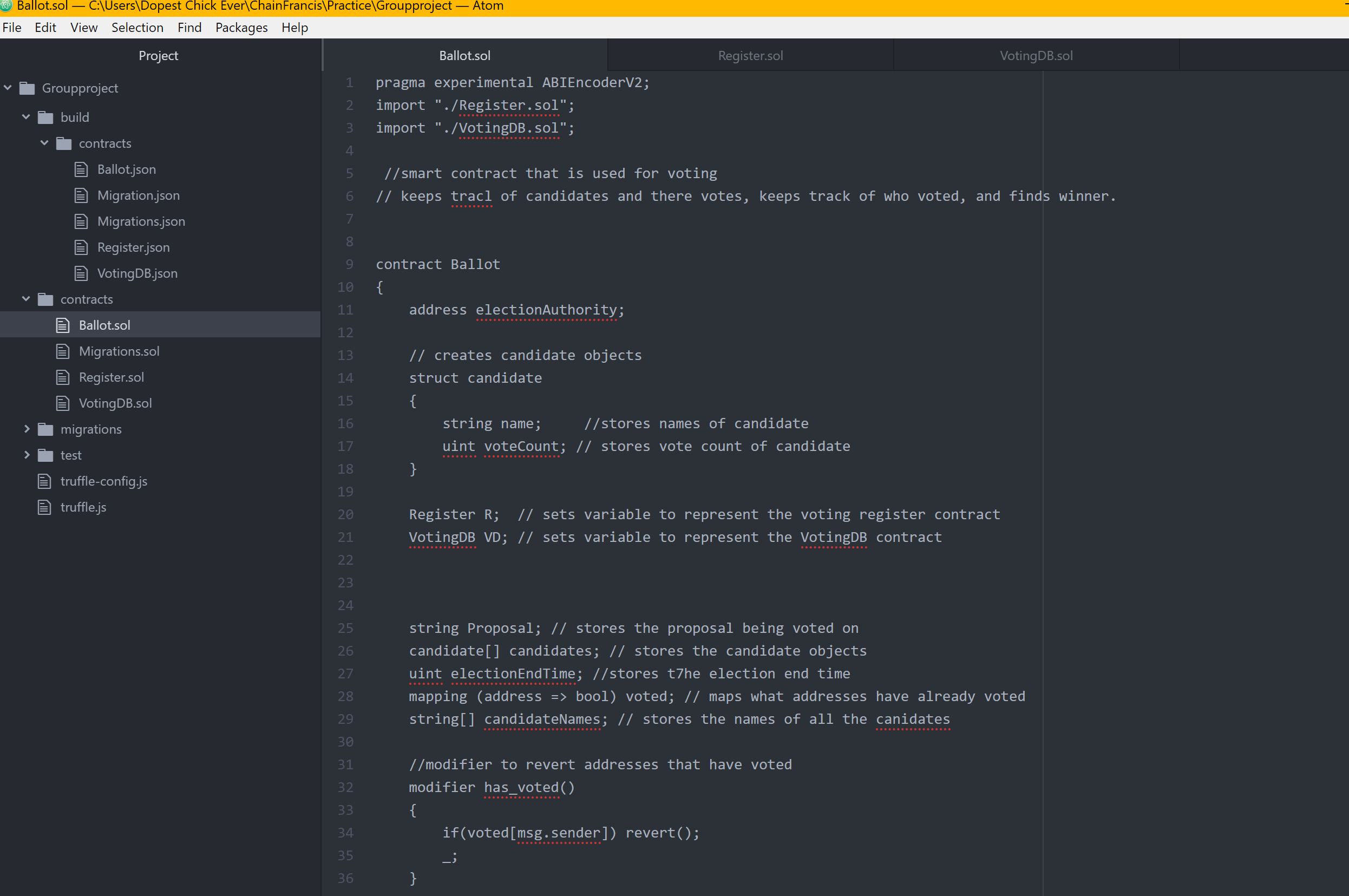
**Voting:**

The above depicts images of the smart contracts in Atom

**Register:**

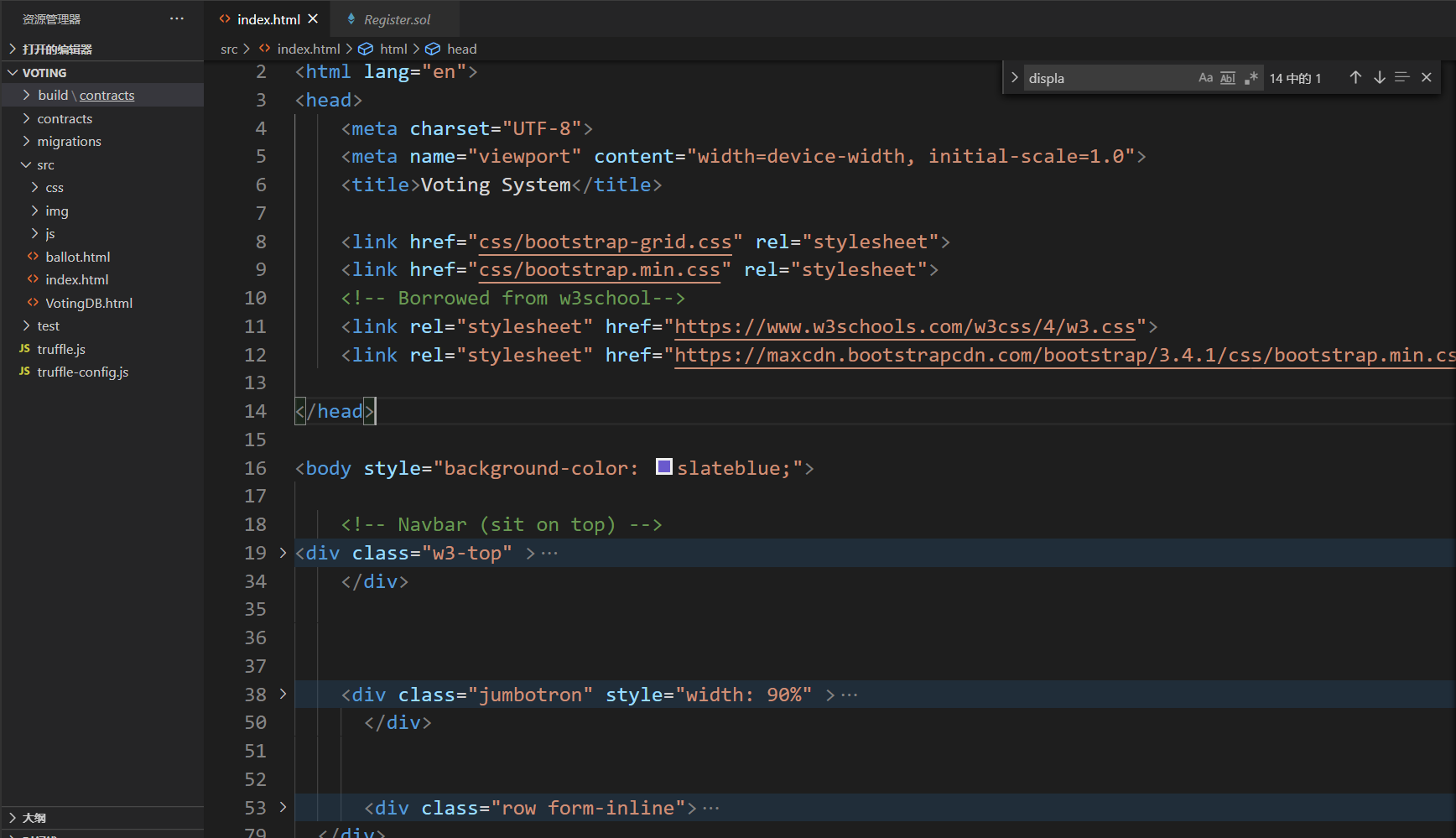
****

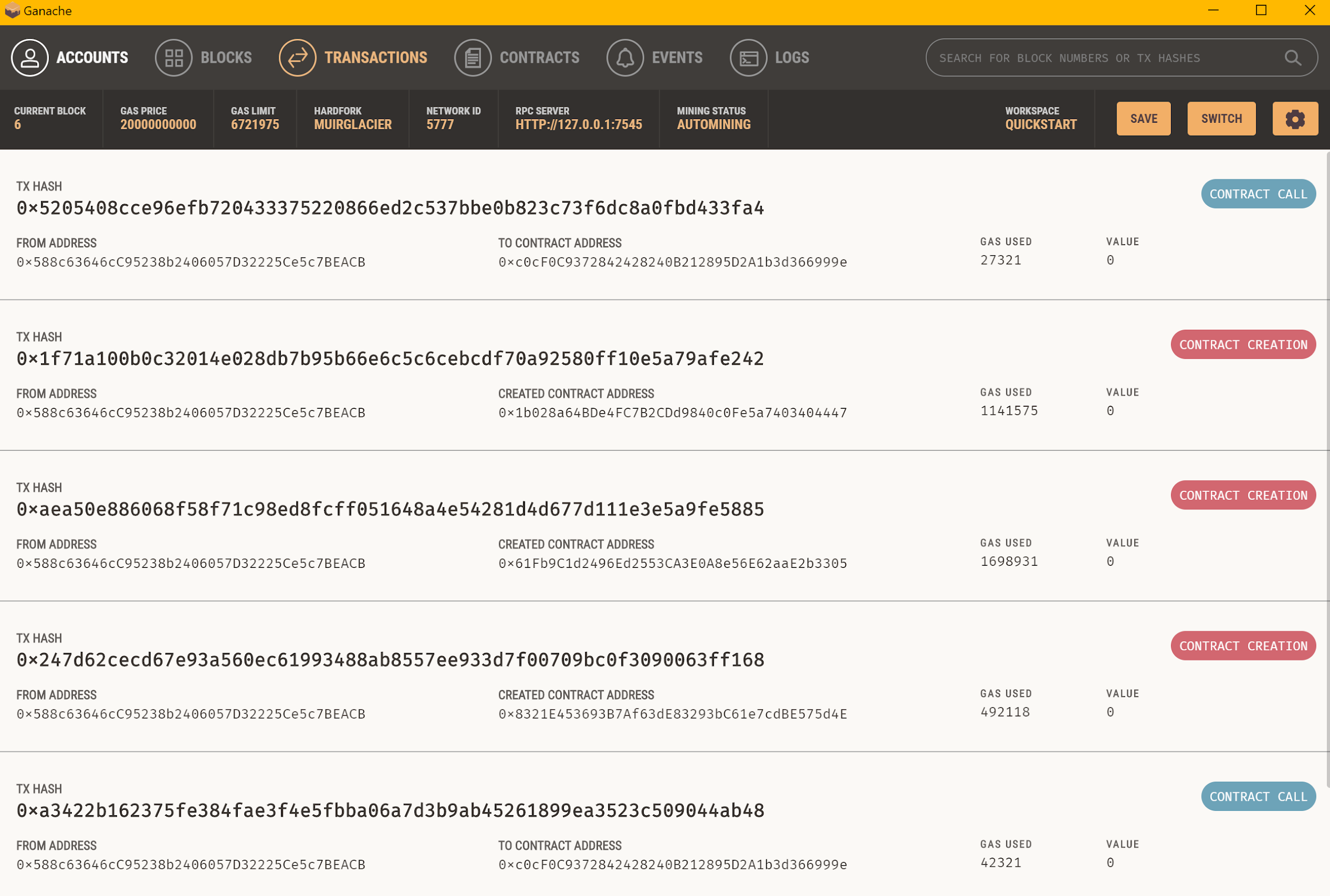
The above depicts images of the smart contracts in Atoml

**Ballot:**

The above depicts images of the smart contracts in Atom

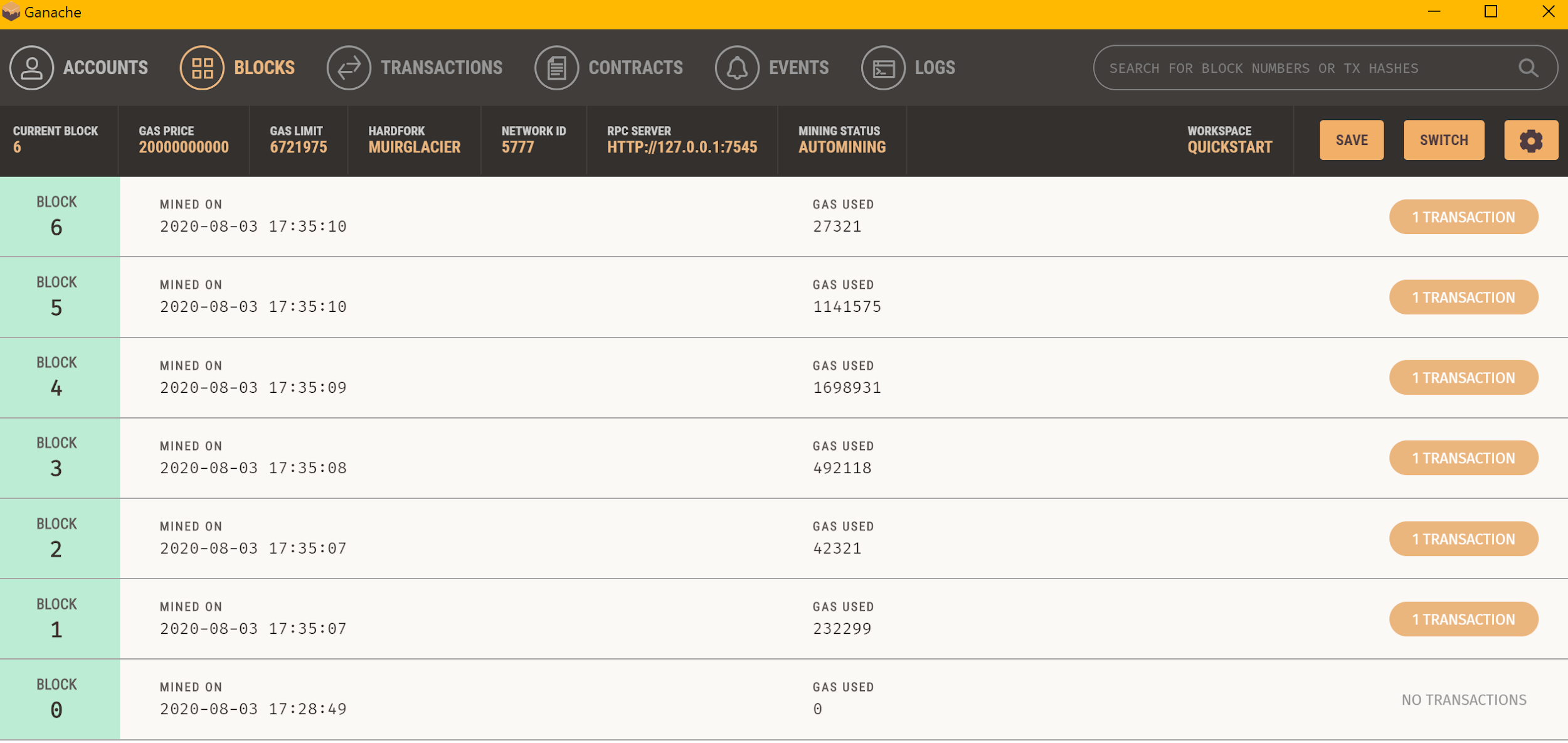
**Code for UI**



**Results in Ganache**

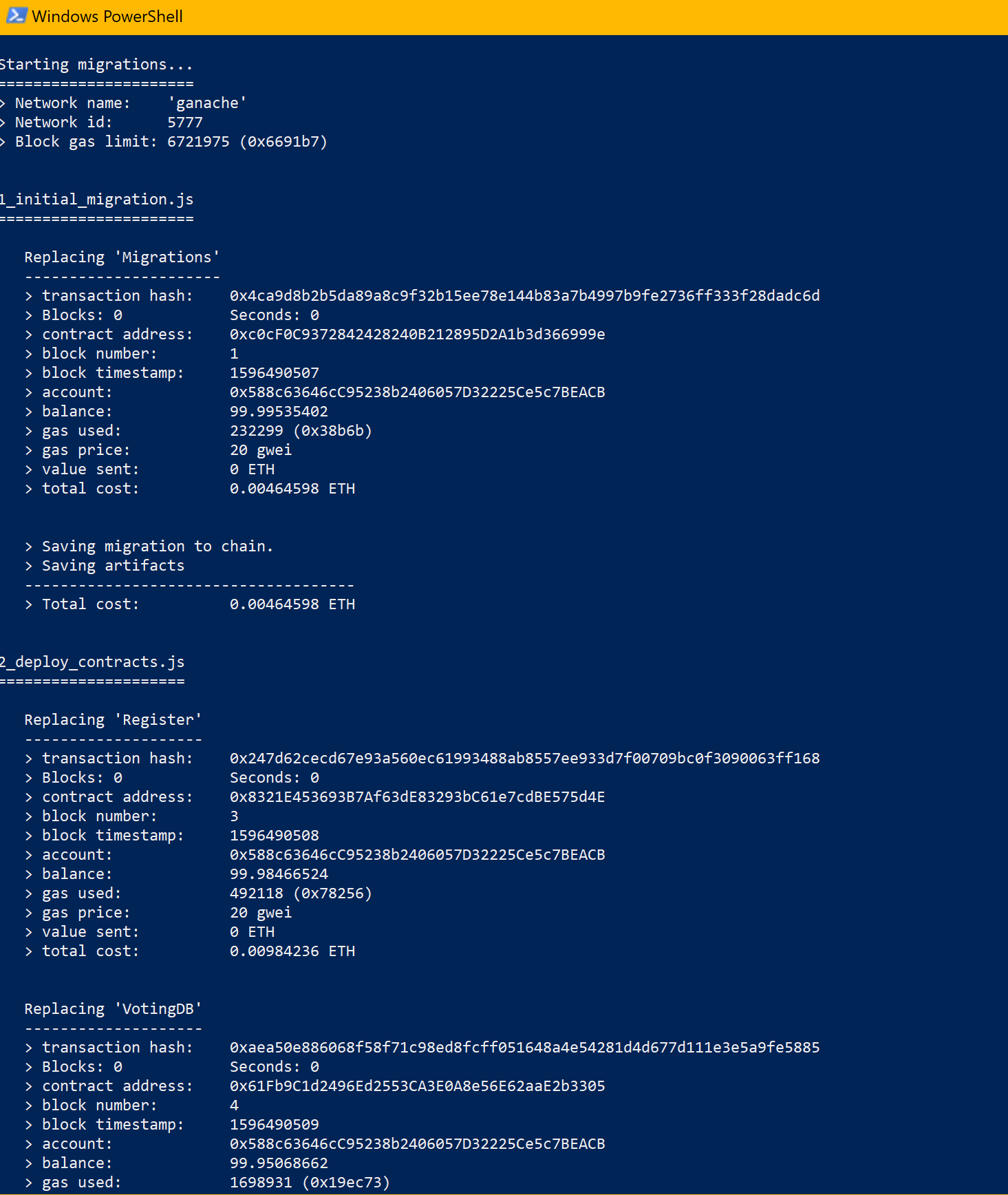
The above is a deployment of the smart contracts in ganache

**Results in Ganache**

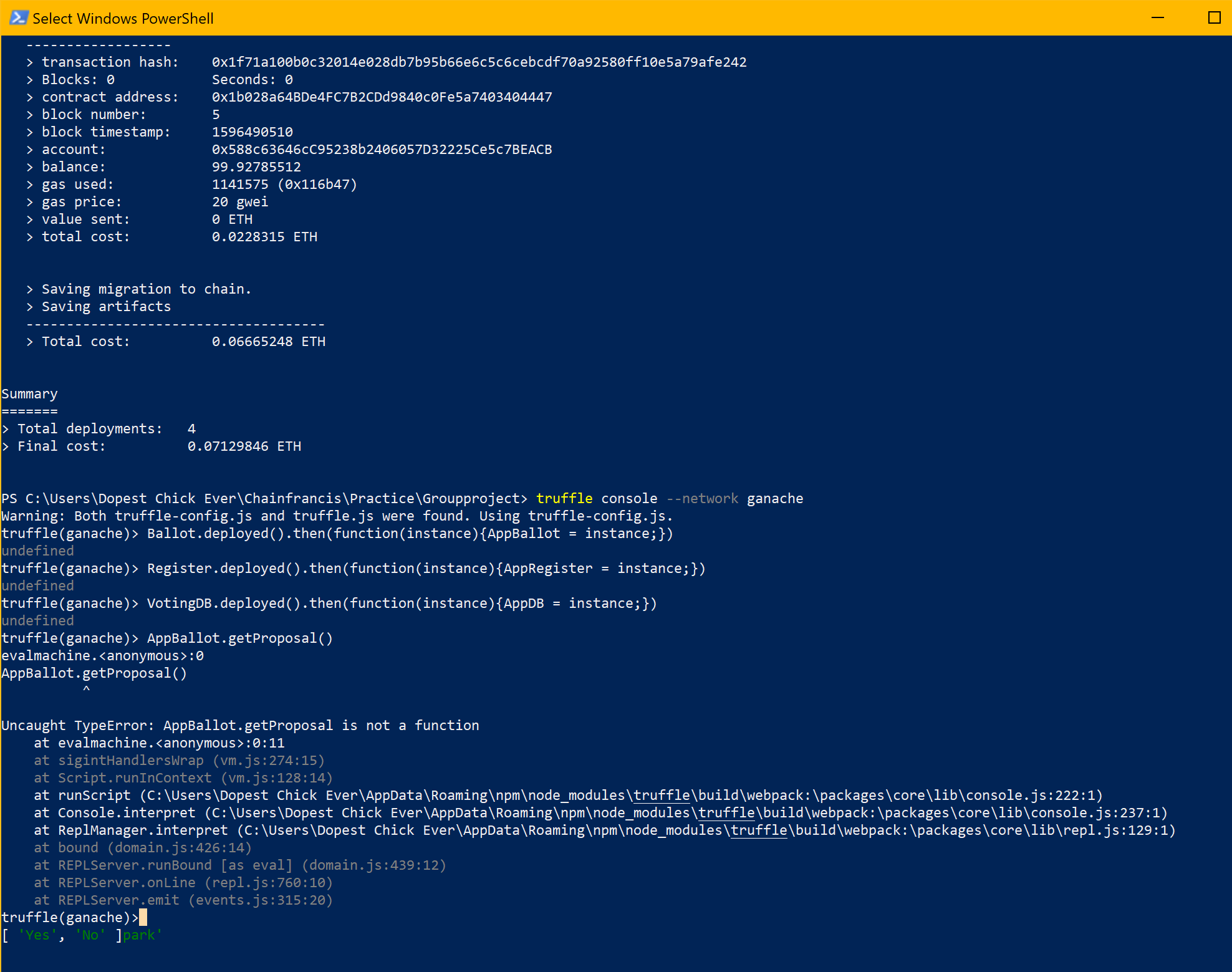
****

The above is a deployment of the smart contracts in ganache

**Powershell:**

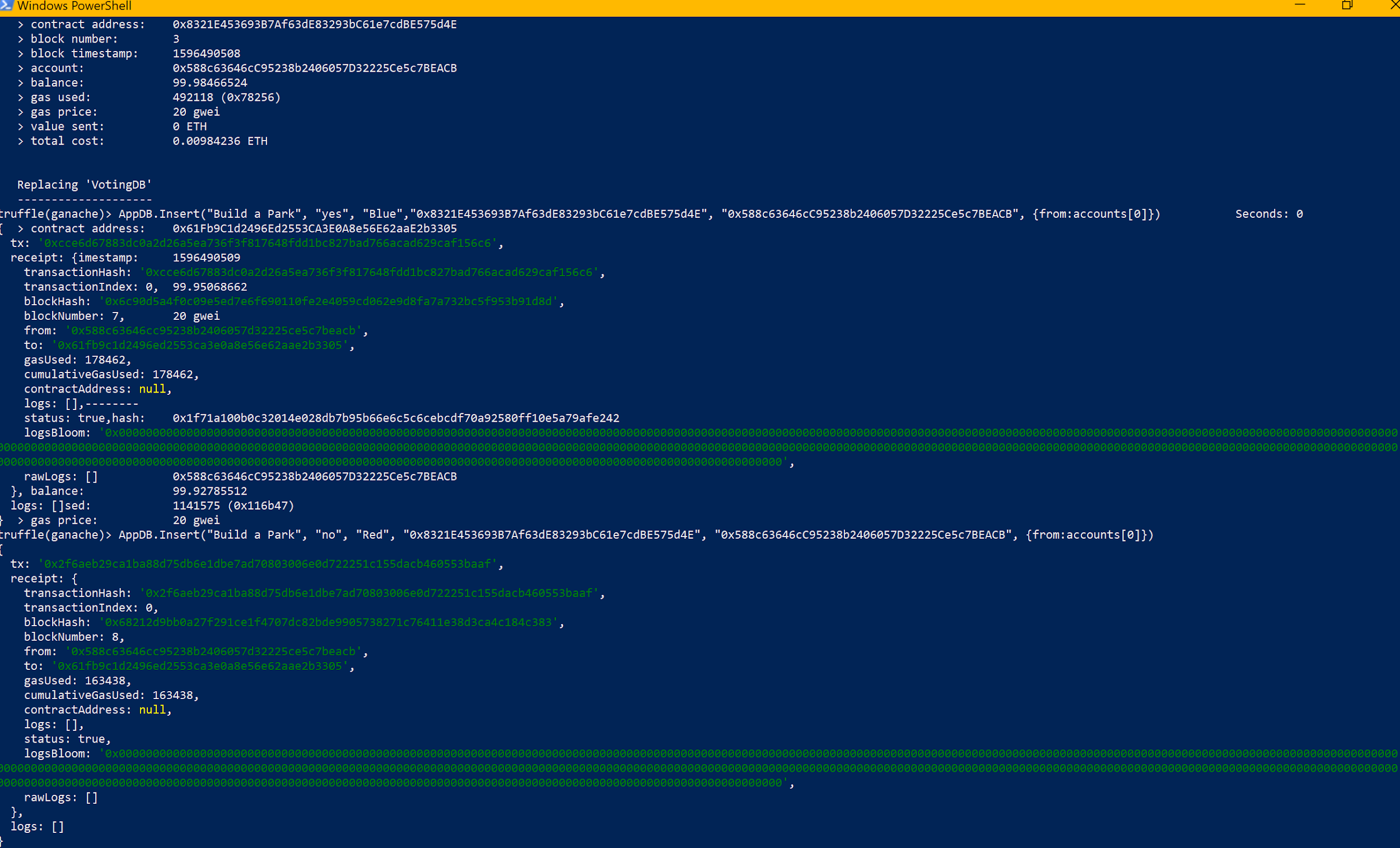


**Powershell:**

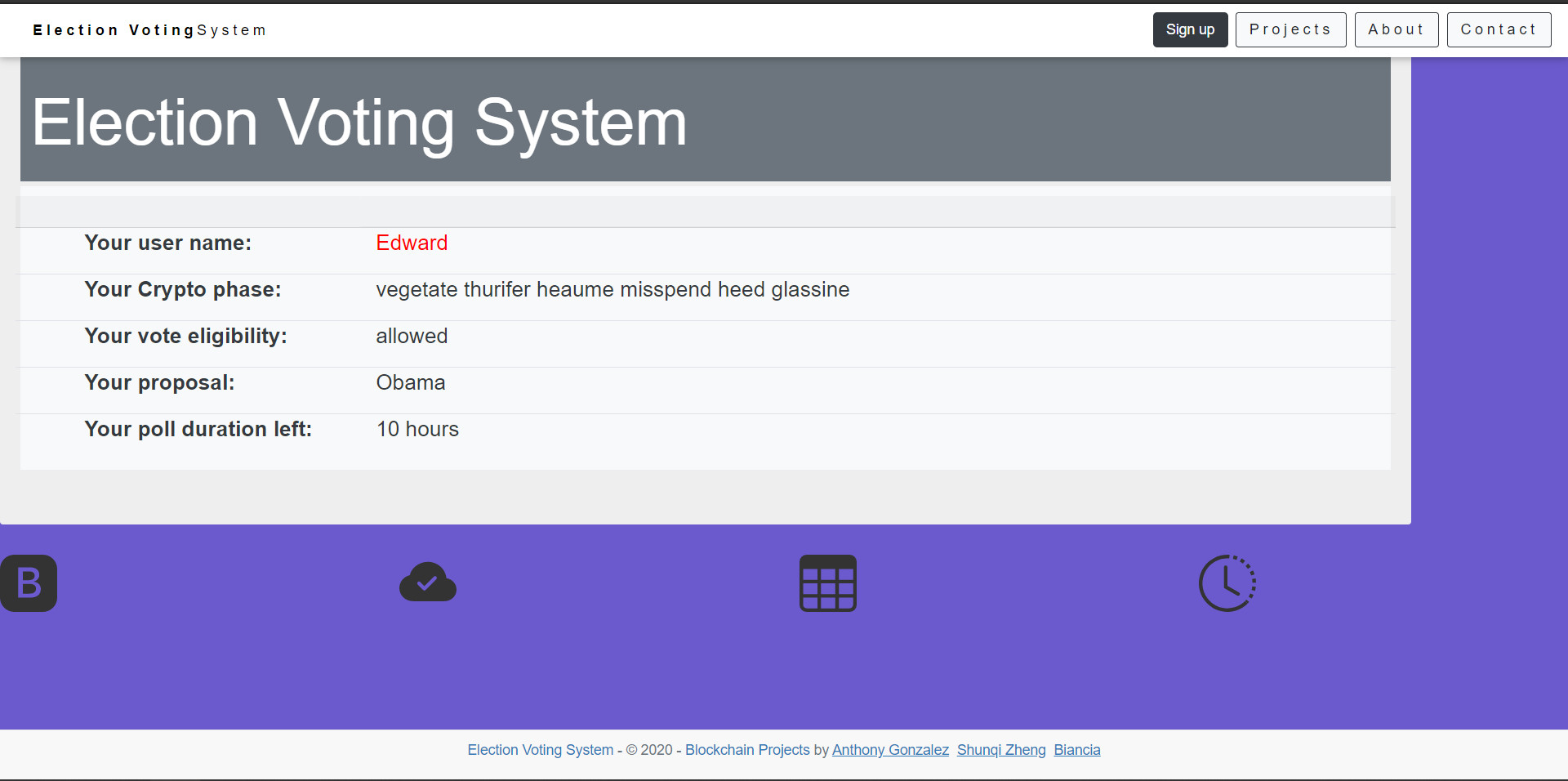
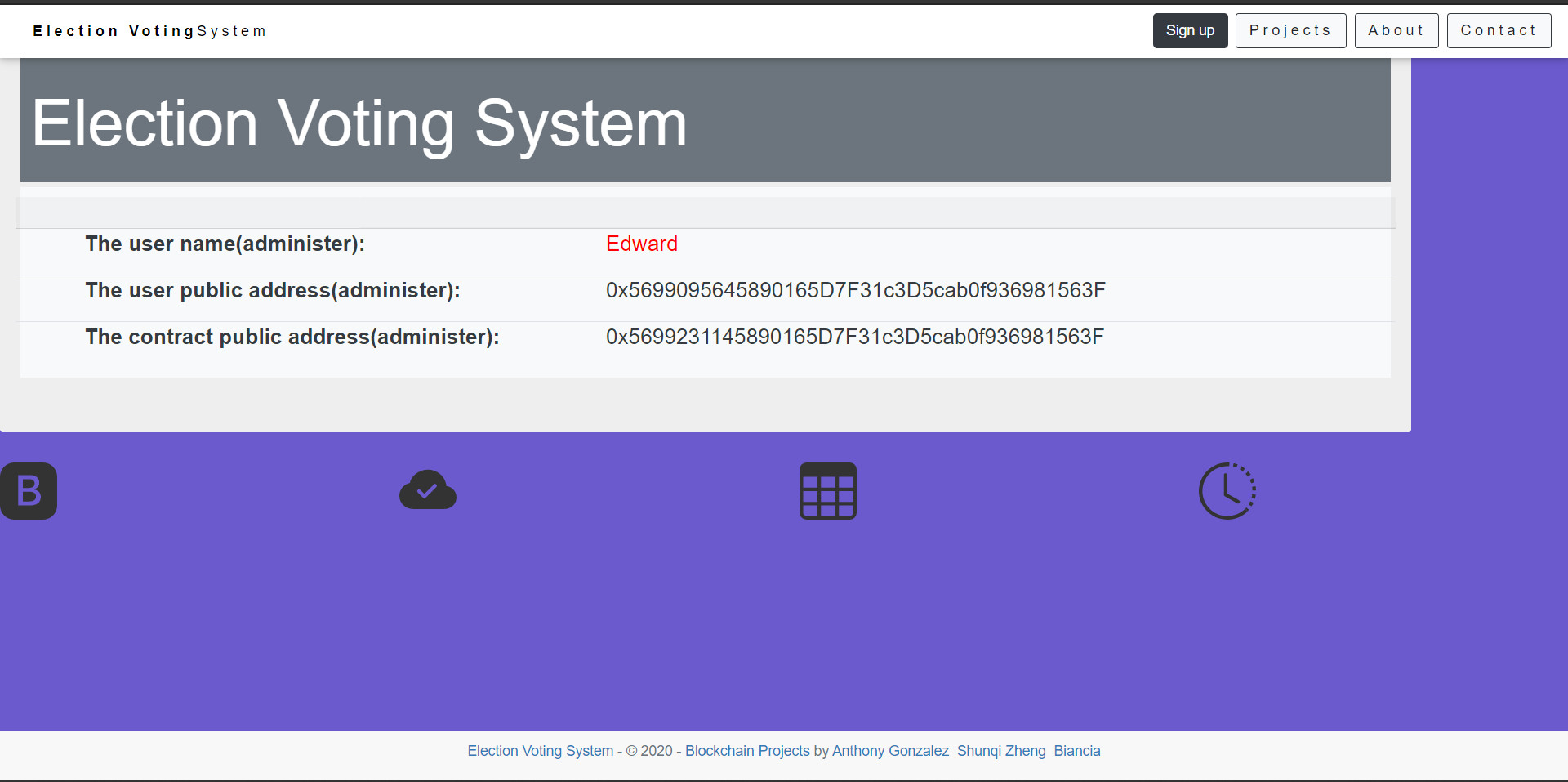


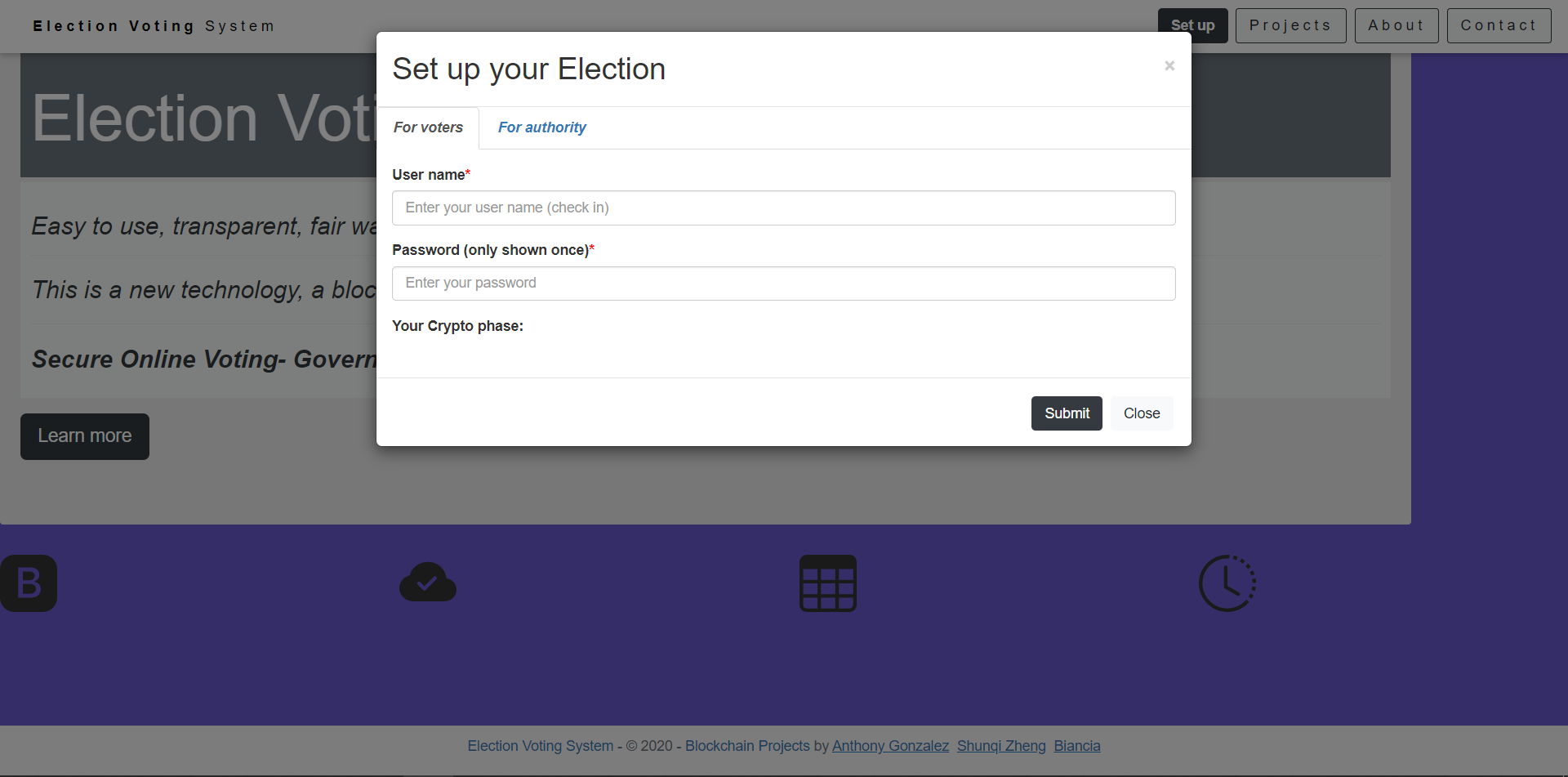
The above is a depiction of the contracts deploying

**Powershell**



**GUI Screenshots**





**Work Cited:**

Walker, J. (n.d.). Pass Phrase Generator. Retrieved August 05, 2020, from <https://www.fourmilab.ch/javascrypt/pass_phrase.html>

Gürsoy, G., Brannon, C.M. & Gerstein, M. Using Ethereum blockchain to store and query pharmacogenomics data via smart contracts. BMC Med Genomics 13, 74 (2020). <https://doi.org/10.1186/s12920-020-00732-x>

Bragagnolo, S., Marra, M., Polito, G., &amp; Boix, E. G. (2019). Towards Scalable Blockchain Analysis. 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). doi:10.1109/wetseb.2019.00007

Bragagnolo, S., Rocha, H., Denker, M., &amp; Ducasse, S. (2018). Ethereum query language. Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain - WETSEB '18. doi:10.1145/3194113.3194114

Bartolett, M., Lande, Pompianu, S. , L., and Bracciali, A. "A general framework for blockchain analytics", 1st Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers ser. SERIAL ‘17, pp. 7:1-7:6, 2017, [online] Available: <http://doi.acm.org/10.1145/3152824.3152831>.

Marra, M., Bèra, C. and Boix, E. G. "A debugging approach for big data applications in pharo", To Appear in Proceedings of the 13th Edition of the International Workshop on Smalltalk Technologies (IWST ‘18), 2018.

Dean, J. and Ghemawat, S. “Mapreduce: Simpliﬁed data processing on large clusters,” Commun. ACM, vol. 51, no. 1, pp. 107–113, Jan. 2008.

Rocha, H., & Ducasse, S. (2018). Preliminary steps towards modeling blockchain oriented software. *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain - WETSEB '18*. doi:10.1145/3194113.3194123

Khoury, D., Kfoury, E. F., Kassem, A., & Harb, H. (2018, November). Decentralized voting platform based on ethereum blockchain. In *2018 IEEE International Multidisciplinary Conference on Engineering Technology (IMCET)* (pp. 1-6). IEEE.

Mebane Jr, W. R. (2010). Fraud in the 2009 presidential election in Iran?. *Chance*, *23*(1), 6-15.