Practical 6 -

Online voting mechanism using Blockchain

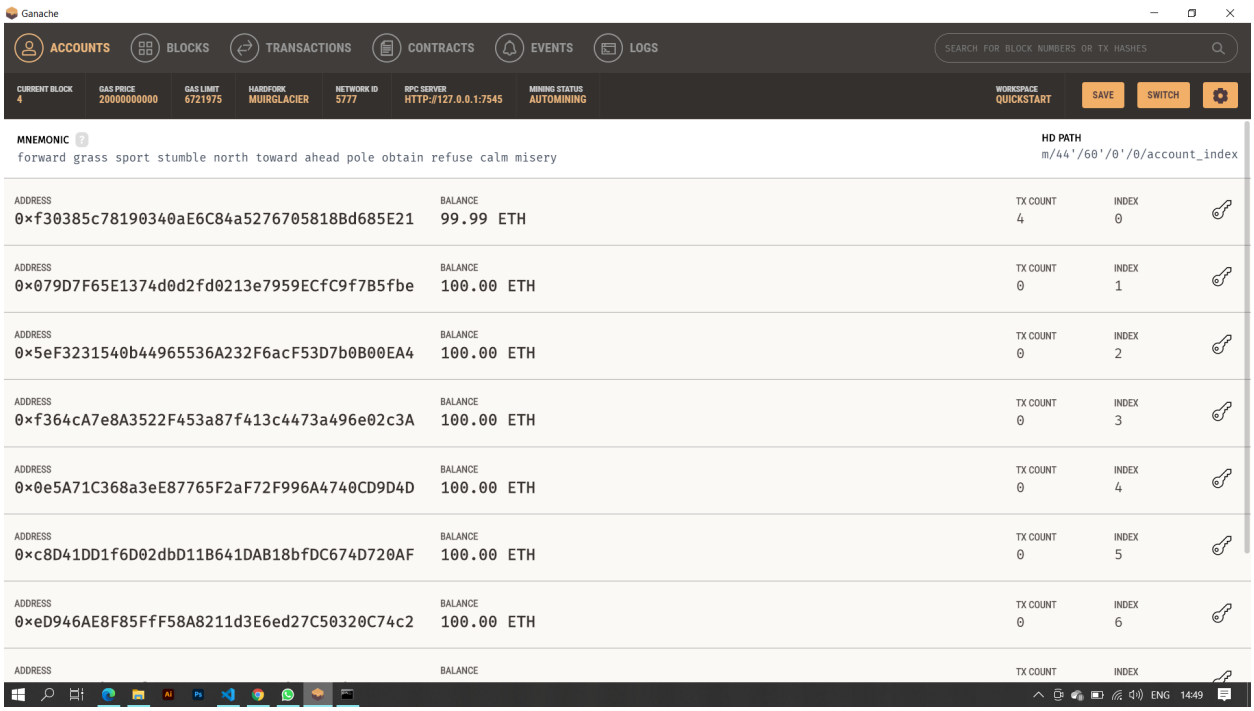
2CSDE93- Blockchain

18bce106

Divya Mahur

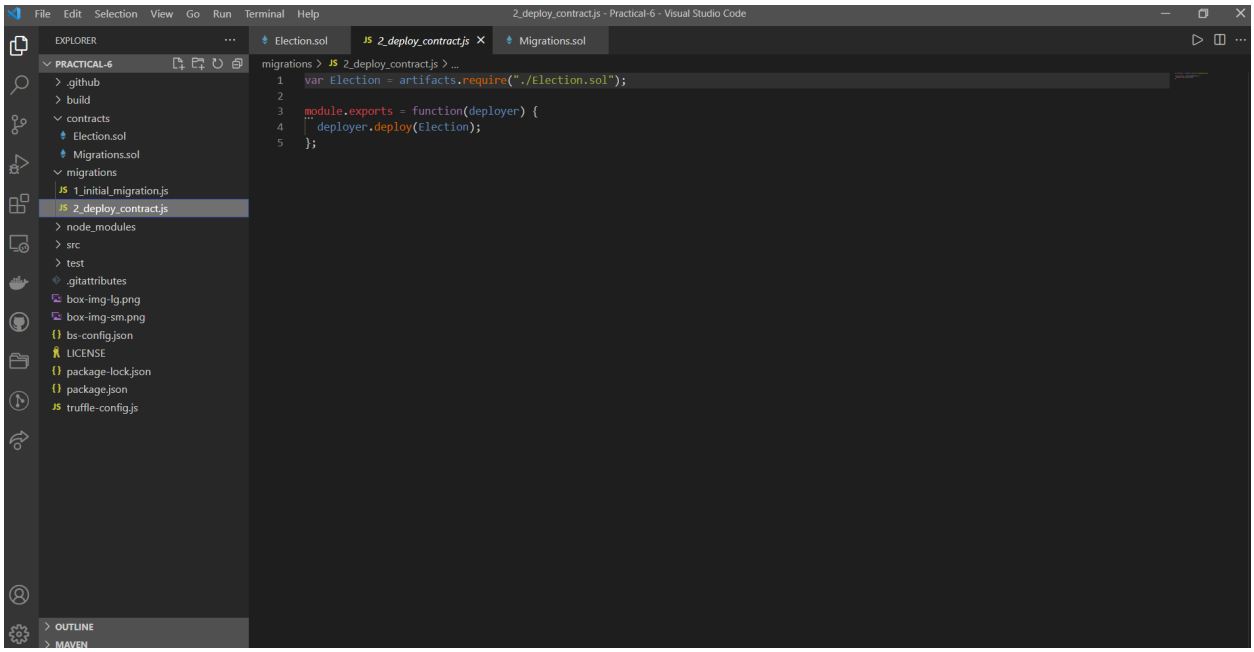
Steps:

Step-1: Start the project by installing all the dependencies: Node Package Manager (npm), Truffle suite, Ganache CLI, and Metamask extension. The Ganache workspace will look like this.

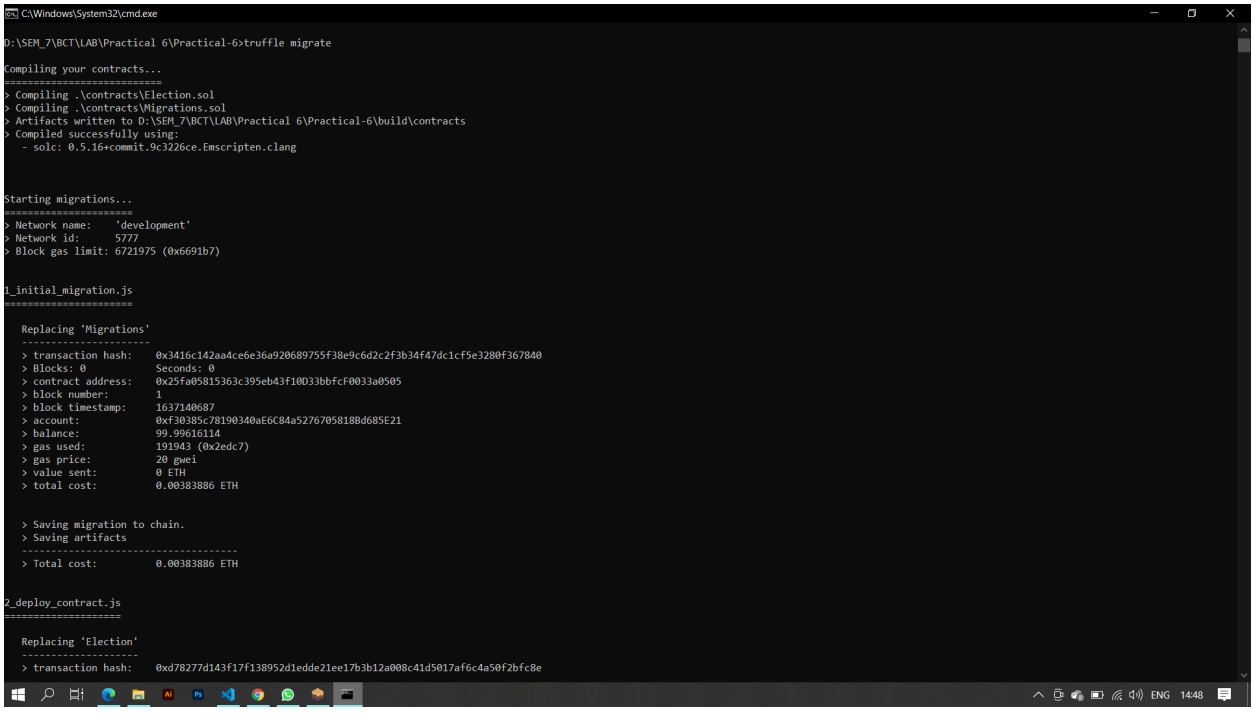


Step-2: Create a directory for the project. Now inside the directory, open the terminal and insert the command ‘truffle unbox pet-shop’. This will install all the required node modules along with a base project for decentralized blockchain application (DApp). Create a file under contracts named Election.sol. Add the below code in the file.

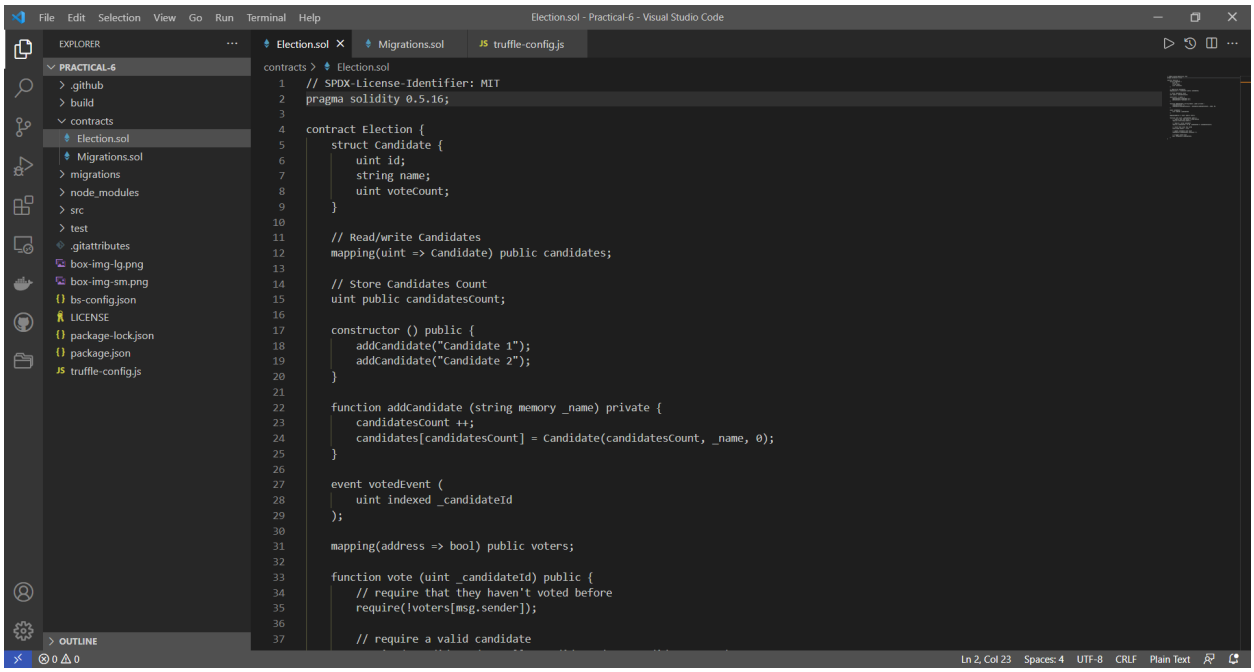
Step-3: Create another file 2_deploy_contract.js under migrations directory. Add the following code.

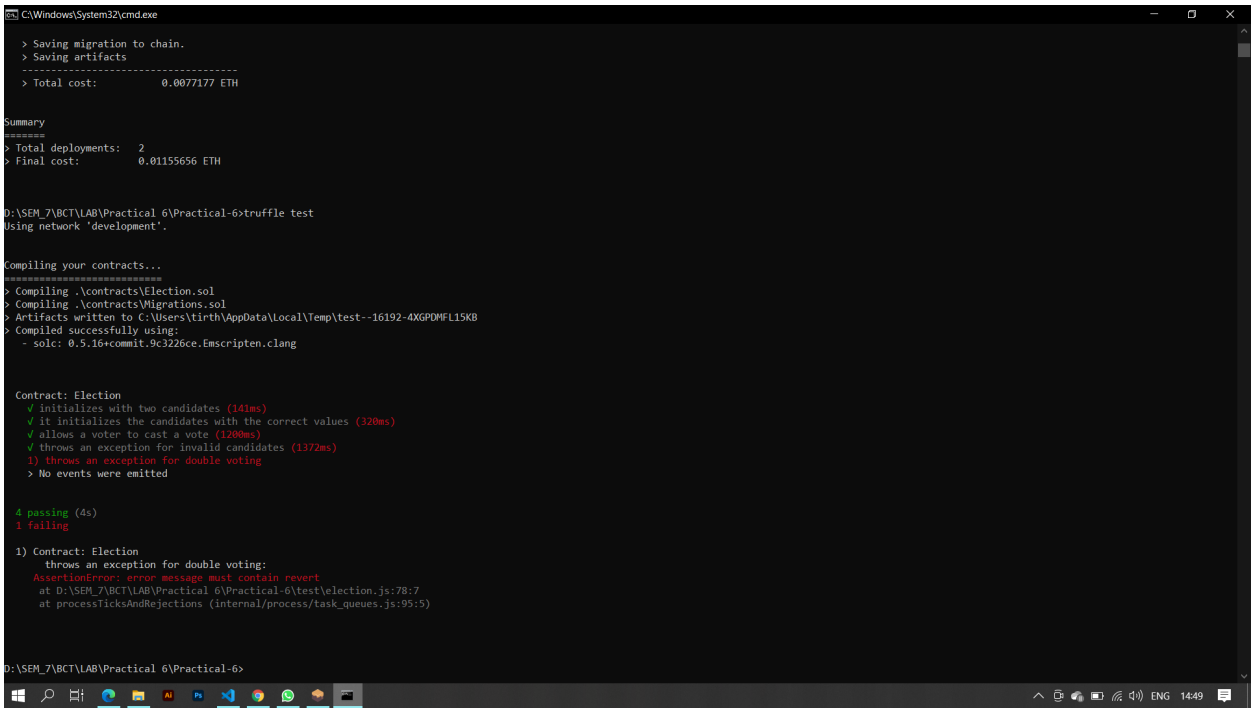


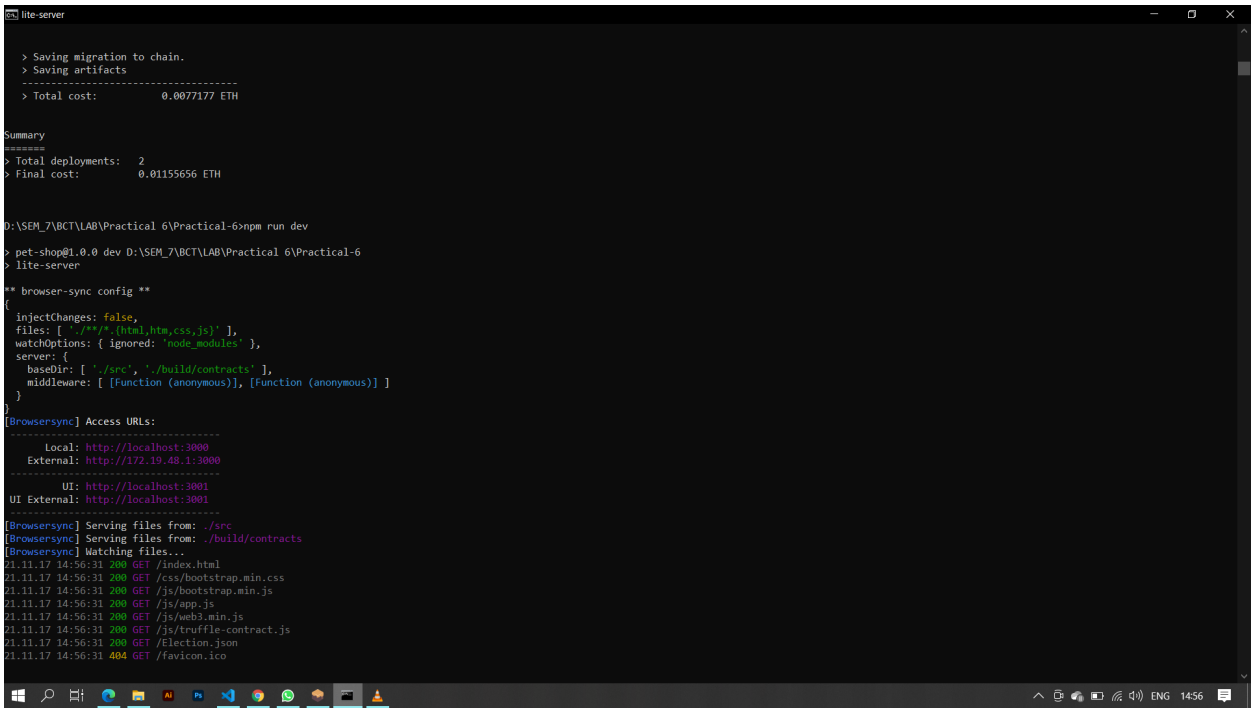
Step-4: Run the command ‘truffle migrate’ in the terminal to apply migrations. The command line will look like this after successful migration. By doing this, 0.01 ether will be deducted from the first account of Ganache. Each time migrations are applied, some amount of ether will be deducted.



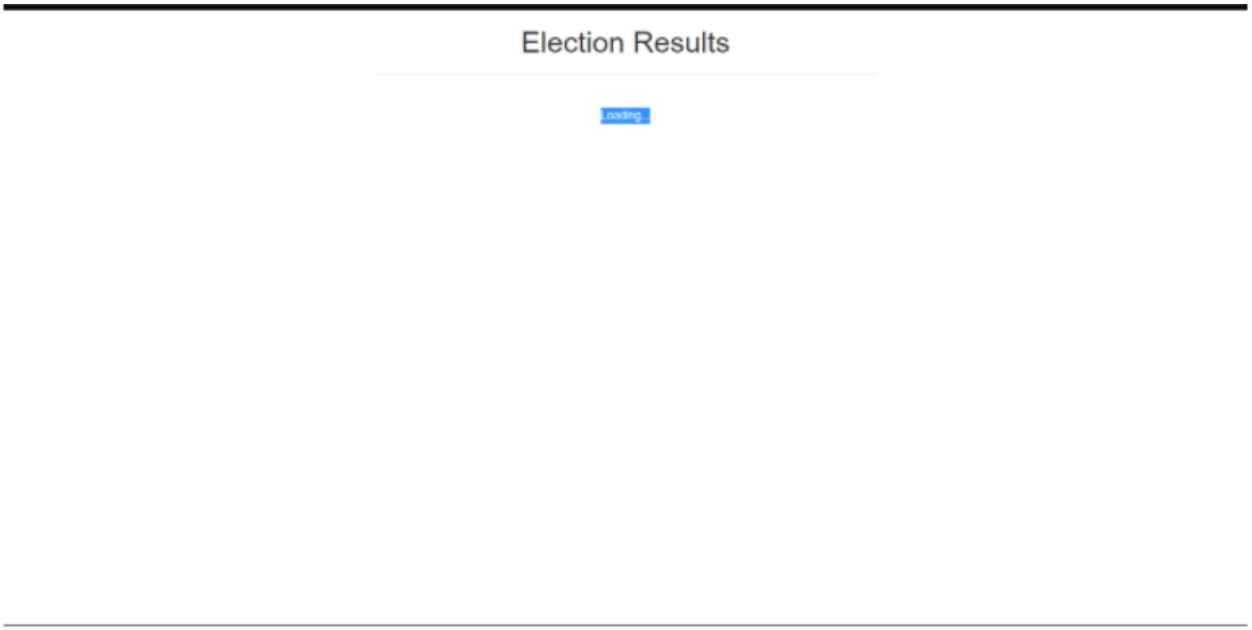
Step-5: Now add the below code in Election.sol. Create a new file election.js in test directory. Insert the below code. In the terminal, type the commands: ‘truffle migrate --reset’ and ‘truffle test’. After the successful test, the following output will be displayed.


File Edit Selection View Go Run Terminal Help
Election.sol - Practical-6 - Visual Studio Code
EXPLORER
PRACTICAL-6
.github
build
contracts
Election.sol
Migrations.sol
migrations
node_modules
src
test
.gitattributes
box-img-lg.png
box-img-sm.png
bs-config.json
LICENSE
package-lock.json
package.json
truffle-config.js
OUTLINE
Election.sol
Migrations.sol
truffle-config.js
contracts > Election.sol
// SPDX-License-Identifier: MIT
pragma solidity 0.5.16;
contract Election {
struct Candidate {
uint id;
string name;
uint voteCount;
}
// Read/write Candidates
mapping(uint => Candidate) public candidates;
// Store Candidates Count
uint public candidatesCount;
constructor () public {
addCandidate("Candidate 1");
addCandidate("Candidate 2");
}
function addCandidate (string memory _name) private {
candidatesCount ++;
candidates[candidatesCount] = Candidate(candidatesCount, _name, 0);
}
event votedEvent (
uint indexed _candidateId
);
mapping(address => bool) public voters;
function vote (uint _candidateId) public {
// require that they haven't voted before
require(!voters[msg.sender]);
// require a valid candidate
Ln 2, Col 23 Spaces: 4 UTF-8 CRLF Plain Text


C:\Windows\System32\cmd.exe
> Saving migration to chain.
> Saving artifacts
> Total cost: 0.0077177 ETH
Summary
> Total deployments: 2
> Final cost: 0.01155656 ETH
D:\SEM_7\BCT\LAB\Practical 6\Practical-6>truffle test
Using network 'development'.
Compiling your contracts...
> Compiling .\contracts\Election.sol
> Compiling .\contracts\Migrations.sol
> Artifacts written to C:\Users\tirth\AppData\Local\Temp\test--16192-4XGPDMFL15KB
> Compiled successfully using:
- solc: 0.5.16+commit.9c3226ce.Emscripten.clang
Contract: Election
√ initializes with two candidates (141ms)
√ it initializes the candidates with the correct values (320ms)
√ allows a voter to cast a vote (1200ms)
√ throws an exception for invalid candidates (1372ms)
1) throws an exception for double voting
> No events were emitted
4 passing (4s)
1 failing
1) Contract: Election
throws an exception for double voting:
AssertionError: error message must contain revert
at D:\SEM_7\BCT\LAB\Practical 6\Practical-6\test\election.js:78:7
at processTicksAndRejections (internal/process/task_queues.js:95:5)
D:\SEM_7\BCT\LAB\Practical 6\Practical-6>
ENG 14:49

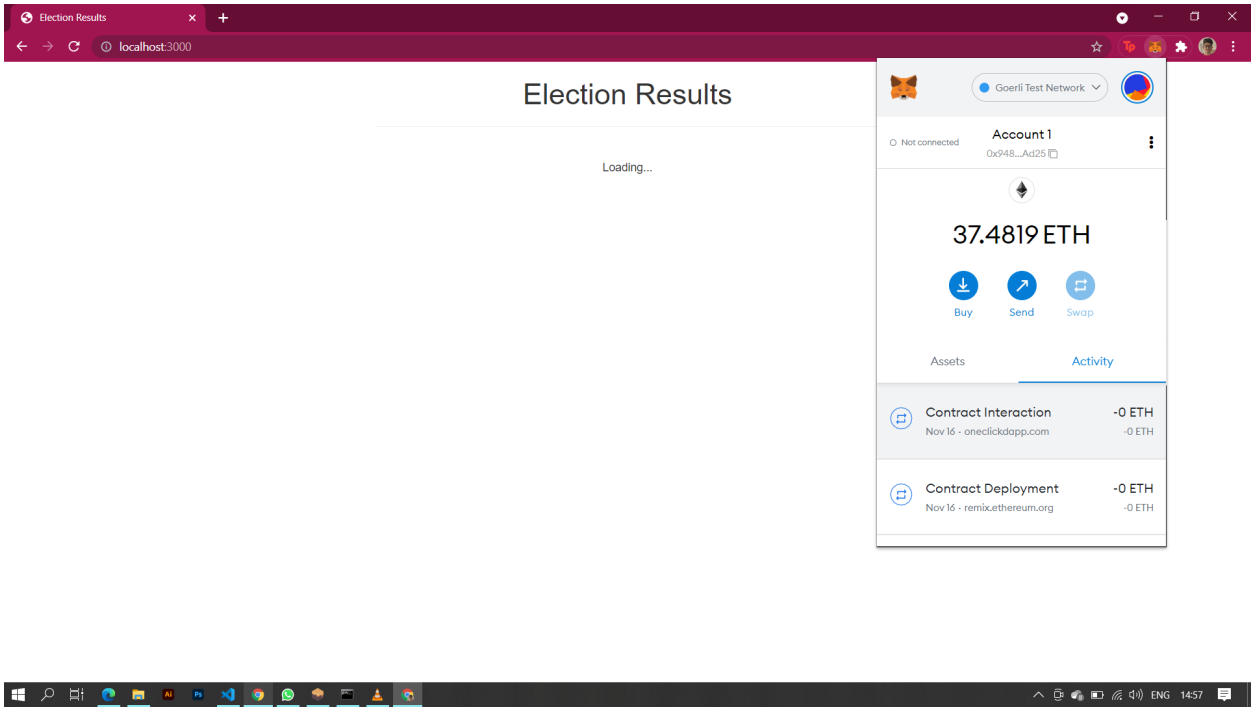


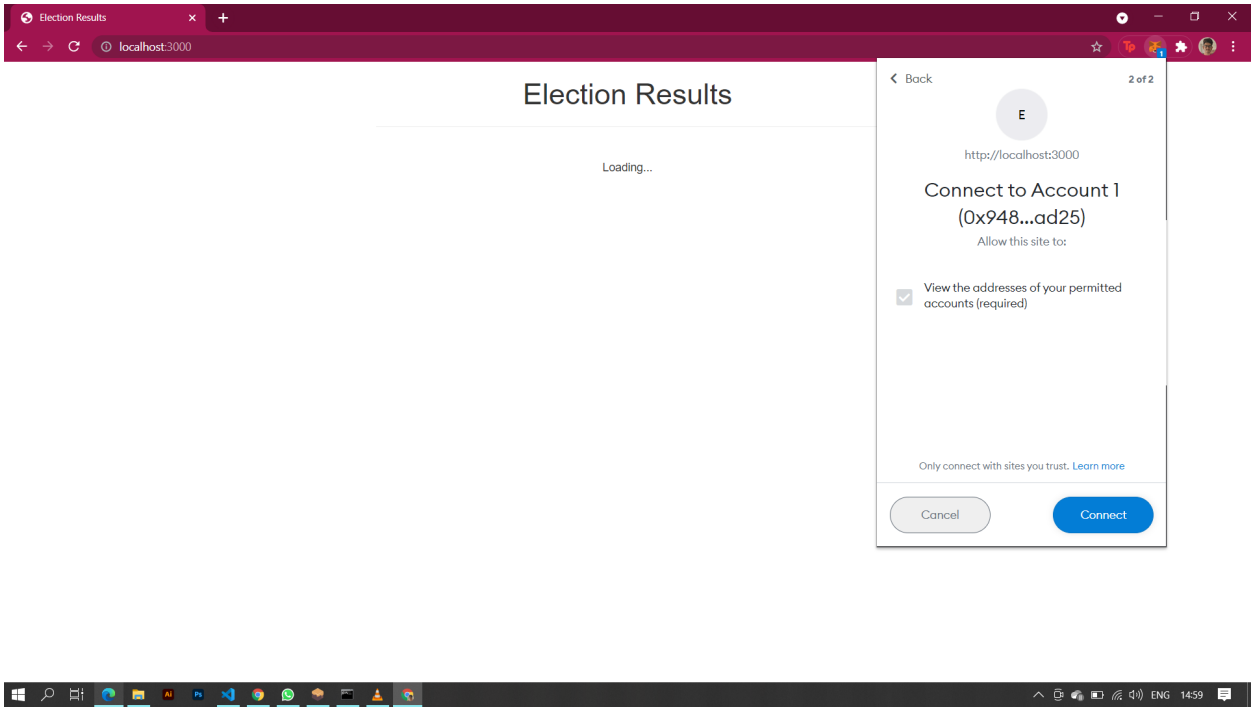
Step-6: Replace the code in the index.html file insider the src directory and app.js file under the src/js directory. The code can be found in the zip file attached. Then run the commands: ‘truffle migrate --reset’ and ‘npm run dev’ to migrate the changes and run the application on the browser at localhost:3000. The browser window looks like this.

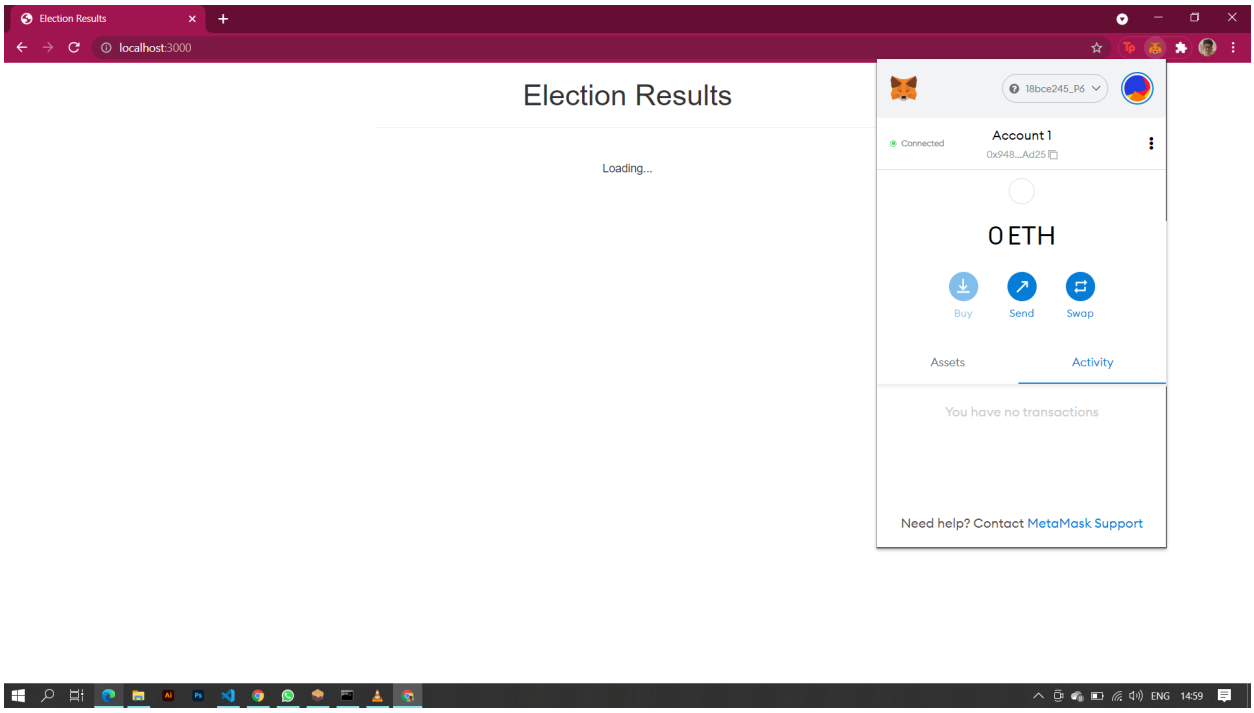


Step-7: Now, to set up Ganache with metamask, click on your metamask extension and then to network. Select ‘Custom RPC’ network. Add the name of the network and in URL add ‘http://localhost:7545’. In chain ID, add 1337 for Ganache. We can import accounts from Ganache using an import account and using its private key. Once it is

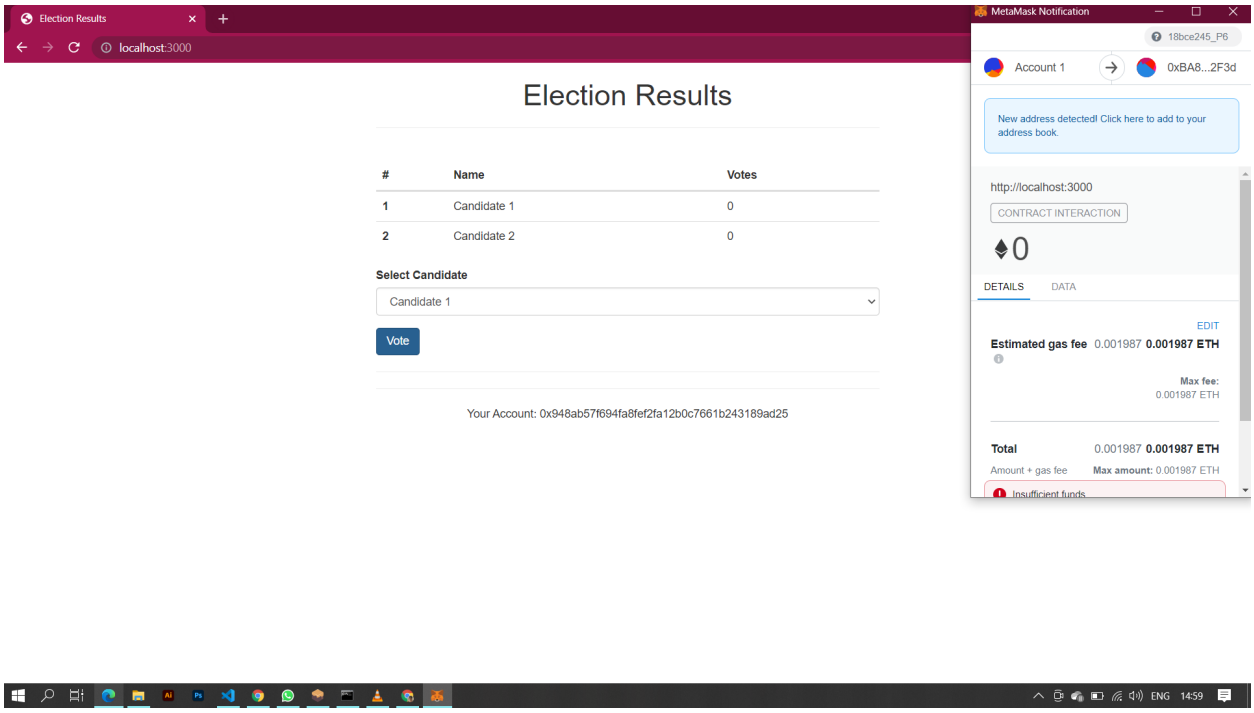
saved, we can see the candidate names but your account will return undefined. To correct that, we have to manually connect our account with http://localhost:3000. To do so, open the connected sites as shown below and connect and just click save. Now we can see our account address on the network.

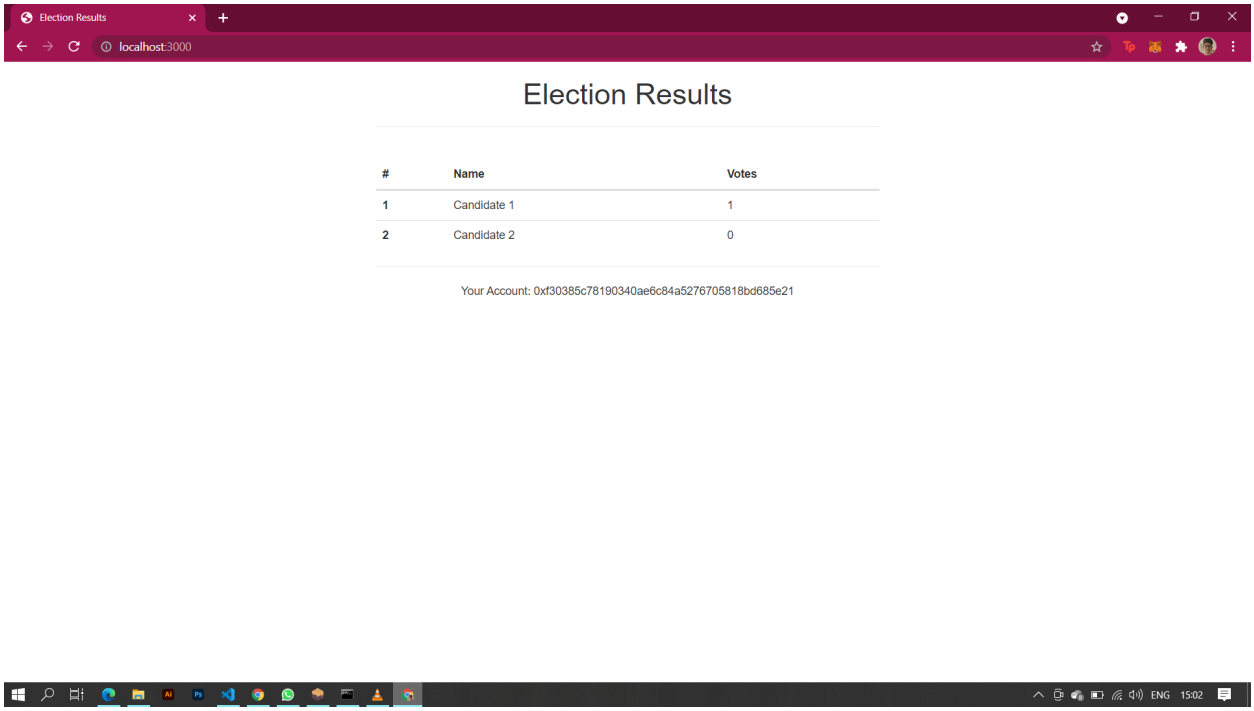






Step-8: This is the final step in the voting application. Add all the required code in different files as provided in the zip file. Run the server again. The output is shown below. We can see that we need to confirm the transaction before voting. And once voted, the same account cannot vote again. We can again import an account from Ganache, connect it to metamask using the connected sites option, and then vote to a different candidate. We have successfully created a voting application.




Election Results
localhost:3000
Election Results
Name Votes
1 Candidate 1 1
2 Candidate 2 0
Your Account: 0xf30385c78190340ae6c84a5276705818bd685e21
ENG 15:02