DaPP University: Elections Application Tutorial:

Sean al-Baroudi

December 10, 2021

1 Video 1:

1.1 Setting up Our Environment:

- The following was installed:
- **MetaMask:** Used to interface with Ethereum Blockchain in a browser.
- **NVM:** Node Version Management System: allows us to create different project workspaces managed by NPM. NVM was installed by downloading nvm.sh script. and running it in Bash.
- NPM and regular Node Package Management: After NVM is installed, run:

  ```
  nvm install --lts
  ```

- **Ganache-UI:** Get the .AppImage for the UI applicaiton, off of the main website. This is apart of the truffle suite.

  Note: See the folloiwng StackExchange Page to see common pitfalls, and missing linux libraries[1]
- **Atom:** Our programming environment for the project.

1.2 Getting the First Contract Running:

- Make a development root folder, and call:

  ```
  truffle unbox Pet-Shop
  ```

- Initialize Git and connect to a repository.
- Code the Election.sol basic contract.
- Run the Ganache .AppImage from commandline, to start our simulated blockchain and client addresses for testing.
- Once done, we need to make a simple migrate function to push our first contract to our Ganache Blockchain. We just copied a template one for a simple push.
- **Run:** *truffle migrate* in Bash. Once successful, run *truffle console* to do interactive testing.
- **Notes on Inspecting State:**
- For this appliction our state (Candidates, Votes, IDs) are stored on the blockchain network we simulate.
- As network availablilty is dynamic, we cannot just write variable assignments like we usually would - they need to be wrapped in a *JavaScript Promise*. An example of the code is below:

  ```
  Election.deployed().then( function(instance) {app = instance})
  ```

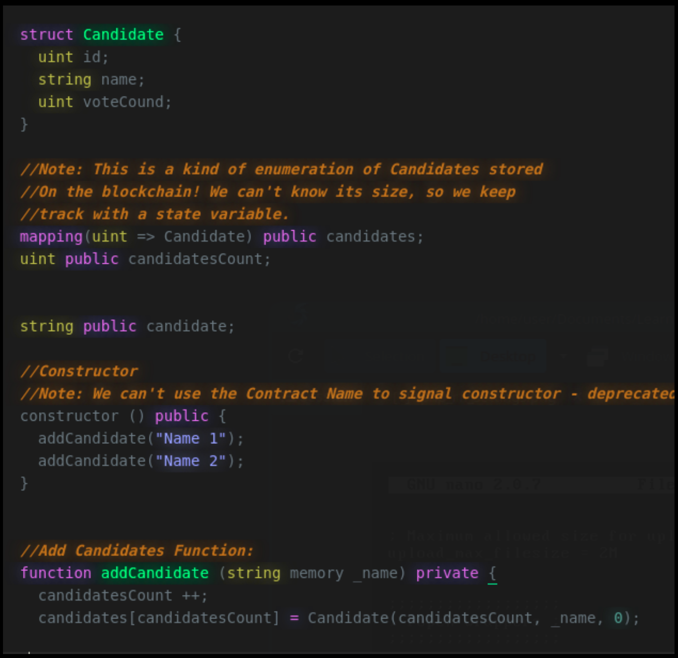
- **What does this piece of code do?**
- We use the .then() function to assign a promise. The body of the promise is written in our inline function.
- We set the app variable to the object representation of our Election application, in the blockchain.
- Once this is done, we may inspect our single candidate as follows:

  ```
  app.candidate()
  ```

-

1.3 Adding an Enumeration and Basic Character Structs:

- Lets look at our new code:



```
struct Candidate {
  uint id;
  string name;
  uint voteCound;
}

//Note: This is a kind of enumeration of Candidates stored
//On the blockchain! We can't know its size, so we keep
//track with a state variable.
mapping(uint => Candidate) public candidates;
uint public candidatesCount;


string public candidate;

//Constructor
//Note: We can't use the Contract Name to signal constructor - deprecated
constructor () public {
  addCandidate("Name 1");
  addCandidate("Name 2");
}


//Add Candidates Function:
function addCandidate (string memory _name) private {
  candidatesCount ++;
  candidates[candidatesCount] = Candidate(candidatesCount, _name, 0);
```

- First we make an *struct* data type, to store a particular Candidates data.
- We use a javascript **mapping**, which for our purposes is a kind of enumeration, linking IDs to Candidate structure objects.
- Note that this structure can return undefined values, and we cannot inspect its size. So we have a *candidateCount* object to keep this data around.
- We have a name for Candidate, outside of the Candidate Information structure.
- Our startup constructor() makes two basic candidates for us.
- *addCandidate()* updates our globally scoped counter, and calls the structure argument.
- **Updating our Pushed Contract:**
- As blockchains are immutable, we cannot rewrite our contract at a specific address on the network. Instead, we must launch an updated contract to the new address.
- This is done with a modified migration flag:

  ```
  truffle migrate --reset
  ```

- Accessing our enumeration of candidates is again done with a Promise, it is awkward:

  ```
  app.candidates(1).then(function(c) {candidate = c})
  ```

  This will give us a JSON object with our properties to inspect.

1.4 Testing our Smart Contacts:

- For our test launch, we can run a unit testing suite that interacts with the Ganache simulated blockchain.
- We run a suite of tests, to ensure no functionality has broken.
- Simply run: *truffle test* to check for common breakages.
- Test are stored in the */test* folder.

1.5 Building our Front End:

- For this section of the tutorial, *index.html, app.js and bootstrap.js* are the parts of the code we wish to modify.
- Our **app.js** will load our front end, and connect it to our smart contract on the network. This is the main piece of code for our dynamic network.

- Note that app.js accesses the *$/build/contract/Election.json$* file. This contains meta-data and EVM code from our compiled smart contrat.
- *package.json and truffle-config.json* at the root directory give us global settings for our project, including our development network port/IP, and our front-end server to display our content.
- **Setting up our front end for testing:**
- First we relaunch our project on our Ganache BlockChain: **truffle migrate –reset**
- Next open another terminal, and use npm to launch your lite-server for the front end:

  ```
  npm run dev
  ```

- The server will start up in console. Note that it is dynamic: as you change code on pages and save, it will reload the pages for you (!)
- When you load the local front end (at localhost:7545), you will see nothing. You need to add the Ganache local network to your list of Metamask Accounts. Add a new network with the following information:

  ```
  address:http://localhost:7545
  ChainID:1337
  ```

1.6 Programming Vote Cast Functionality:

- For this upgrade, we are going to add the ability for a given user on the blockchain to vote.
- They will access our website, and connect to the chain using their Metamask account. When they vote, we will read their account address and keep a record of their voting. We will only let them vote once.
- **Elections.sol Changes:**
- **Added a vote() function:** This is called by our voter (information is accessed with msg.sender() method). The voter will make a transaction on the network, and request that the vote() function on the Ethereum network be called.
- Added a Voter Mapping: We map voter Address to a Vote (Yes/NO), recoreded with a boolean variable. This will be our database of voters.
- **Revisions to Tutorial:** As this tutorial was out of date, a few console commands needed to be changed:
- To run the changes, run *truffle migrate –reset* and then start the *truffle console*. Make sure your lite-server is running, as in the previous section.
- As usual, connect to the instance of the application to start:

  ```
  Election.deployed().then(function(i){app = i;})
  ```

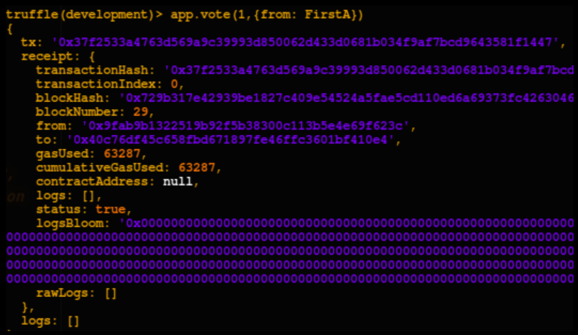
☐ Getting a Particular Account address. Done with [2]:

```
web3.eth.getAccounts().then( function(s){FirstA=s[0]})
```

☐ Calling our vote, with account metadata:

```
app.vote(1, {from: FirstA});
```

And will return the following:



```
truffle(development)> app.vote(1,{from: FirstA})
{
  tx: '0x37f2533a4763d569a9c39993d850062d433d0681b034f9af7bcd9643581f1447',
  receipt: {
    transactionHash: '0x37f2533a4763d569a9c39993d850062d433d0681b034f9af7bcd
    transactionIndex: 0,
    blockHash: '0x729b317e42939be1827c409e54524a5fae5cd110ed6a69373fc4263046
    blockNumber: 29,
    from: '0x9fab9b1322519b92f5b38300c113b5e4e69f623c',
    to: '0x40c76df45c658fbd671897fe46ffc3601bf410e4',
    gasUsed: 63287,
    cumulativeGasUsed: 63287,
    contractAddress: null,
    logs: [],
    status: true,
    logsBloom: '0x0000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
    rawLogs: []
  },
  logs: []
```

- ⊗ Unfortunately, for the test suite we have the double voting test failing. I cannot find a good reason for this. Tried another post suggestion about changing what property to look for for a failed transaction (tx code), but it did not work). [3]
- ⊗ When I moved on to the next pat of the tutorial - further issues with the UI were encountered. There are now too many mistakes to move forward - and my knowledge of javascript is not sufficent to debug this properly.
- **Tutorial Ended.**

References

[1] https://ethereum.stackexchange.com/questions/109847/how-to-install-ganache-ui-on-ubuntu-20-04-lts

[2] https://ethereum.stackexchange.com/questions/31967/web3-how-do-i-get-just-the-first-account-on-testrpc-using-web3-et 31973

[3] https://github.com/pauluhn/election/blob/paul/test/election.js#L54