E-Voting Distributed application (DAPP) Using Blockchain

Blockchain Assignment

Team Members:
Surendar S : 2020MT93162
Sunny Khanuja : 2020MT93110

Objectives of the Application

1

Make the Election process easy and secure.

2

Avoid the waiting for my turn to vote (avoid queue-based waiting system)

3

Make the vote counting process easy.

4

Avoid DOUBLE VOTING (casting invalid votes)

Scope of this document

- Steps to get started: UI application (React application) and running smart contracts.
- UI walkthrough and explanation
  Note: Explanation Video link shared in this doc
- Smart contracts explanation
  Note: Explanation Video link shared in this doc
- Links to access the source-code and Explanation video
- Now, Lets get started…..

Getting started: Deploy smart contracts

- **To deploy the Smart Contract**
  - Install Ganache and create a workspace.
  - Install Truffle npm package globally by running `***npm install -g truffle***`
  - Run `***truffle migrate --reset***` from the command line to deploy the smart contract to the blockchain.
  - Download Metamask Chrome extension for the browser to help interaction between the application and the blockchain.

NOTE:

- make sure to run ganache an appropriate port to connect it with metamask
- Connect an account with local RPC network by using the private key of an account from ganache (this will grant 100ETH by default)

Getting started: Run react development server

- **To run the UI application locally**
  - Run the command `***npm install –no-optional***` (To download the node modules to your local working directory)
  - In order to start the UI application run the following command, `***npm start***`
  - This will start the application in the localhost server.
  - NOTE: if the port is busy, then react will ask for confirmation to run in different port. The type in ‘y’ (Yes) to run the application.
  - This is start the application.

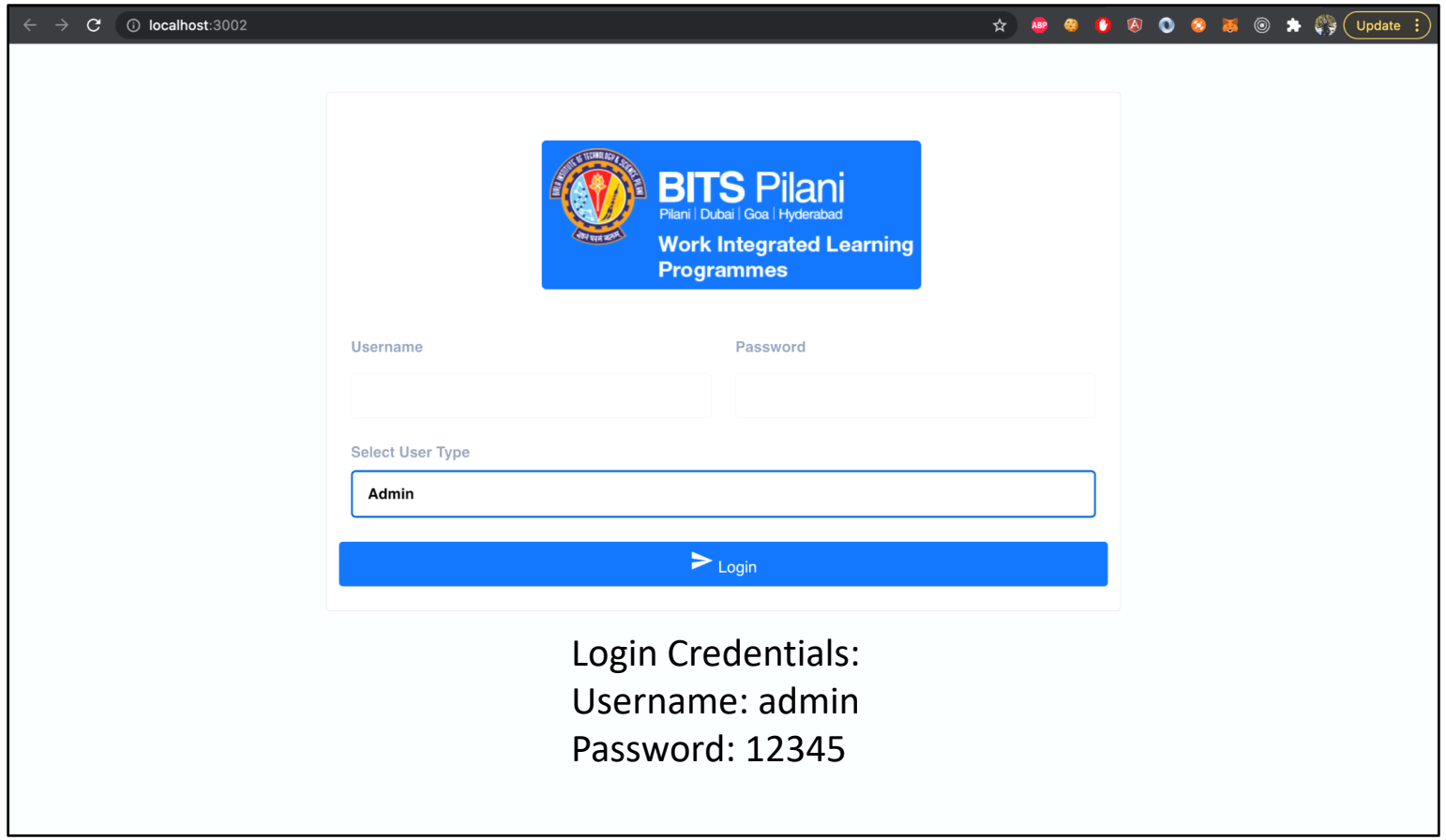
UI walkthrough – Explanation for different workflows

1. Login as Admin
   - managing election, voters and candidates
2. Login as Voter
   - for casting vote for a desired candidate

Admin workflow

Login as Admin

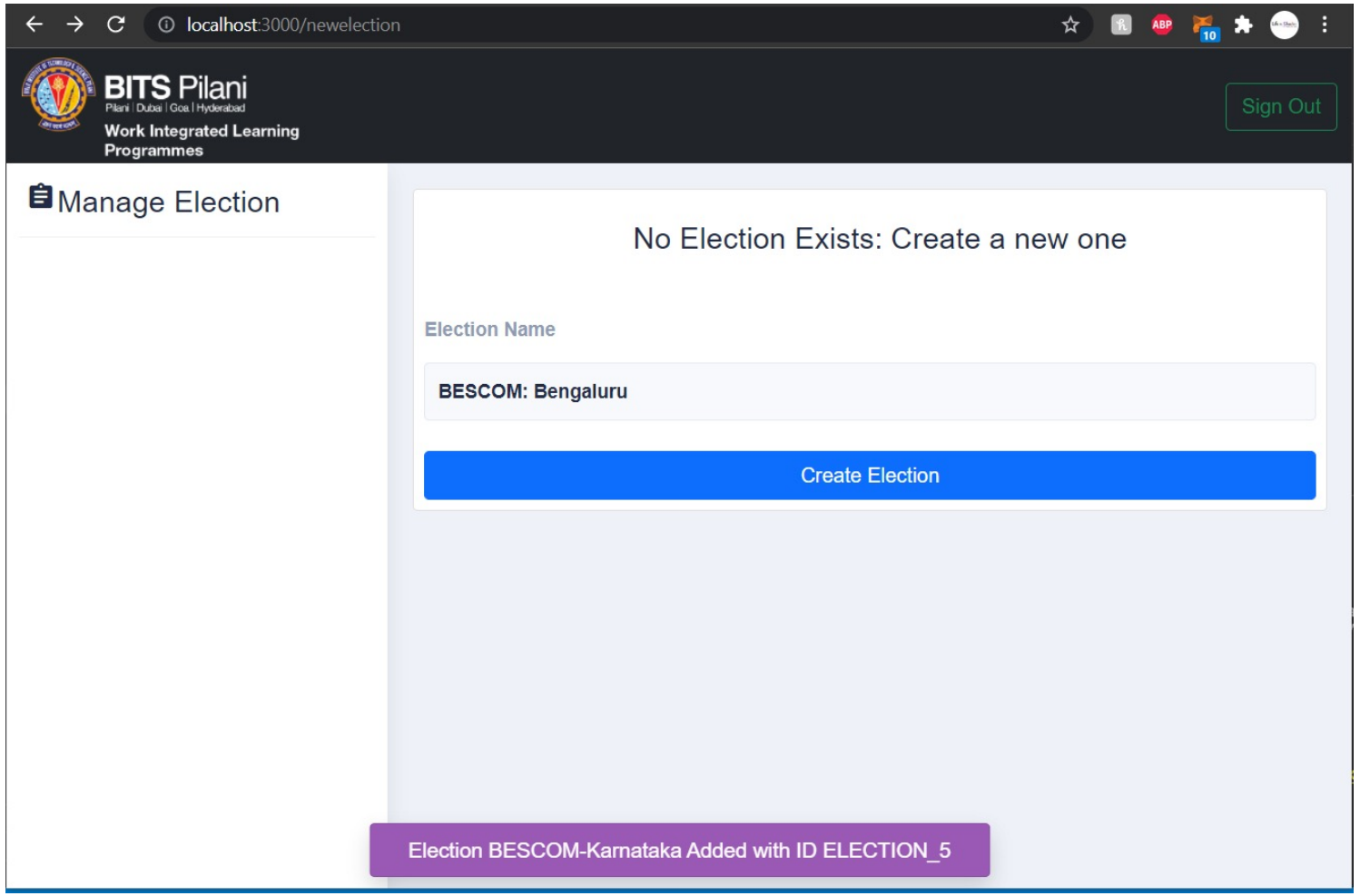
- This is how the landing page has been designed.
- There are two types of User, Admin and Voter.
- Admin has following privileges:
  - to create an Election (if not created before)
  - add candidates to election
  - Add voters
- **But they can’t cast a vote**



Login Credentials:
Username: admin
Password: 12345

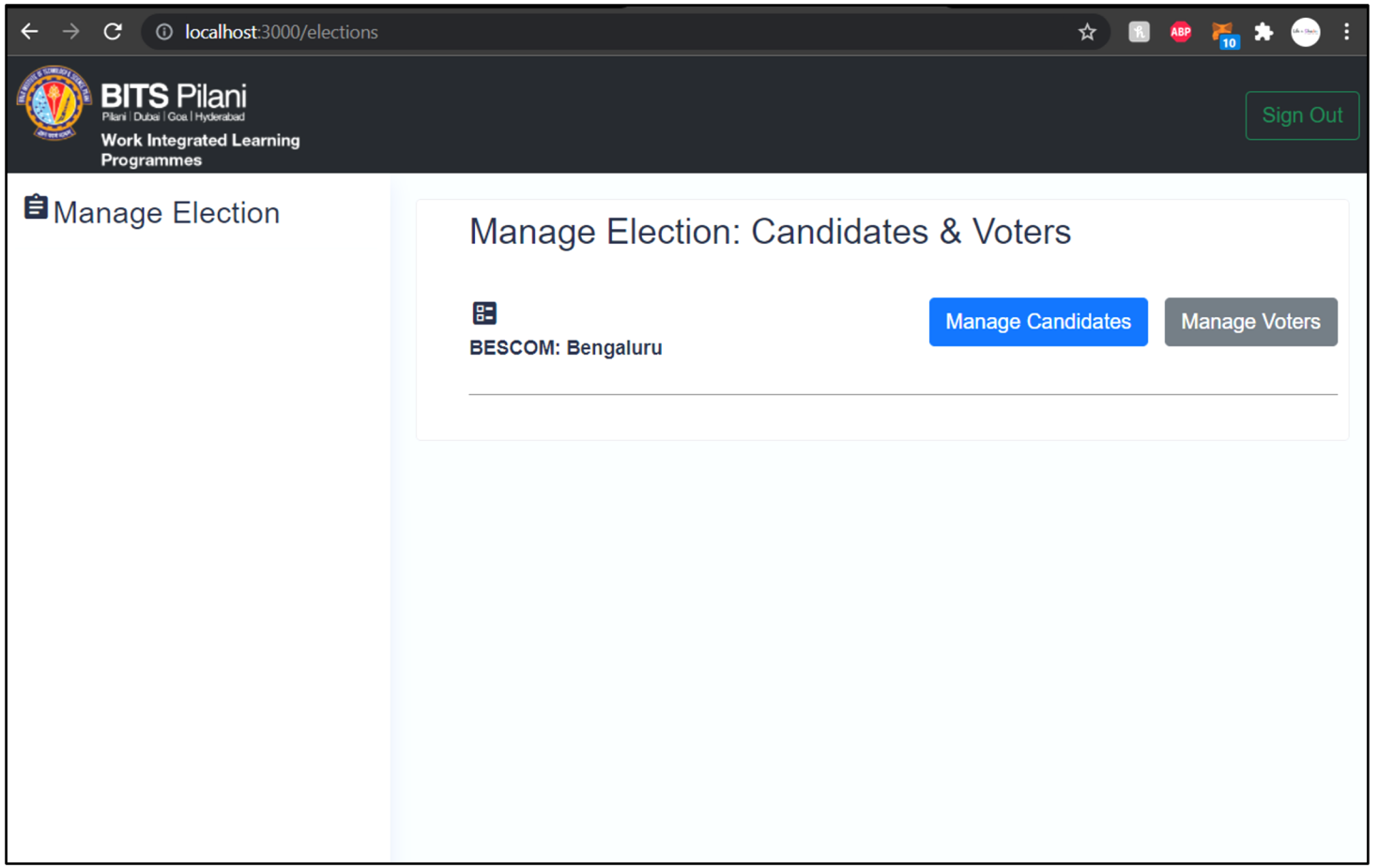
Create Election (For first time)

- After a successful admin login, this is how the admin dashboard looks like.
- The admin need to create the election for the first time.
- With the help of "**web3.js**" we connect to the local Ethereum network (ganache) to store the election details.



Manage Election

- Once the Election is created, then the admin can start add candidates and voters.
- As of now, there is a one-to-one mapping for the Election organizer (Admin) and the election smart contract.
- This can be extended further to provide access to an organizer to handle multiple elections.

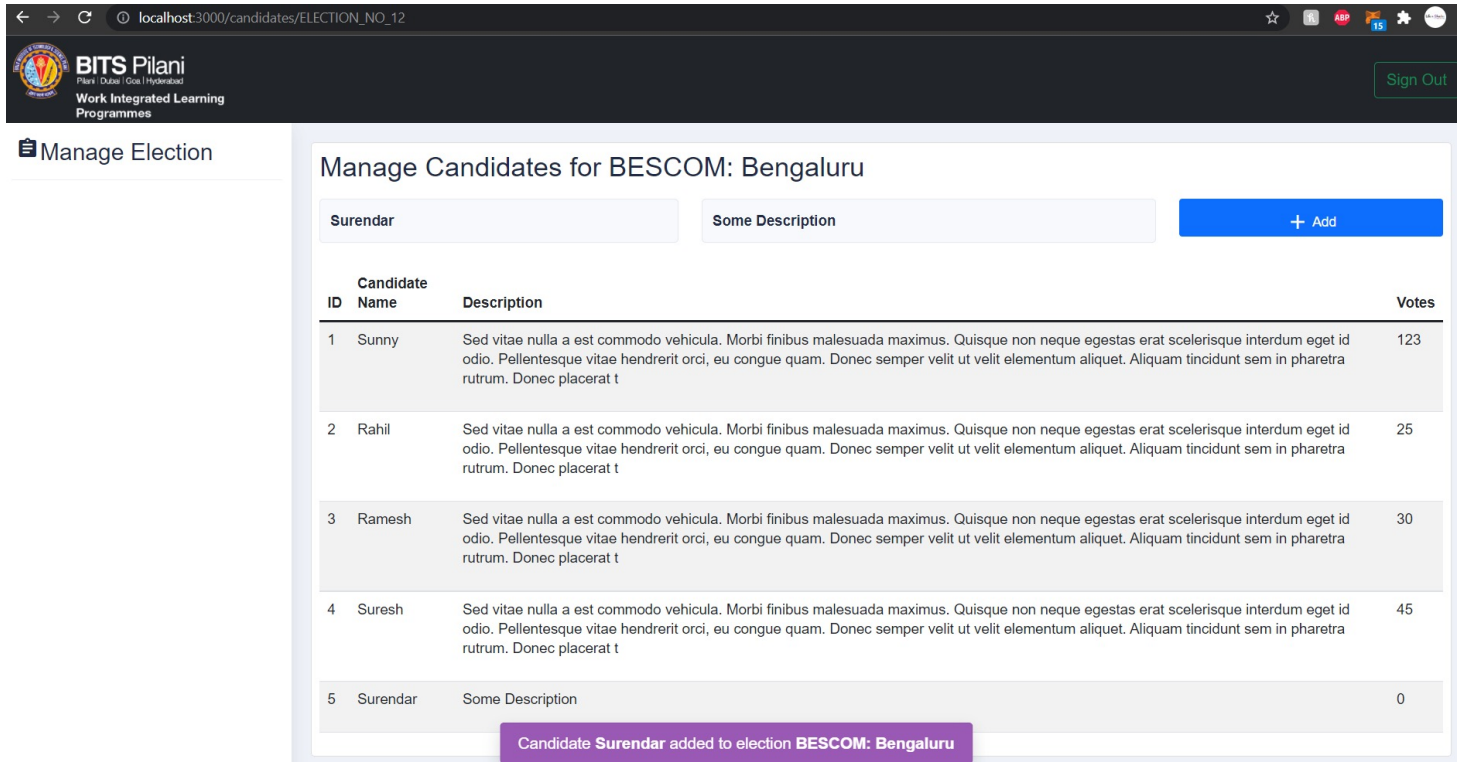


Managing Candidates

- Once the Manage Candidates button is clicked, one can add the candidates under an election contract.
- As of now, we store the
  - Candidate ID
  - Candidate name
  - Description
  - Vote count

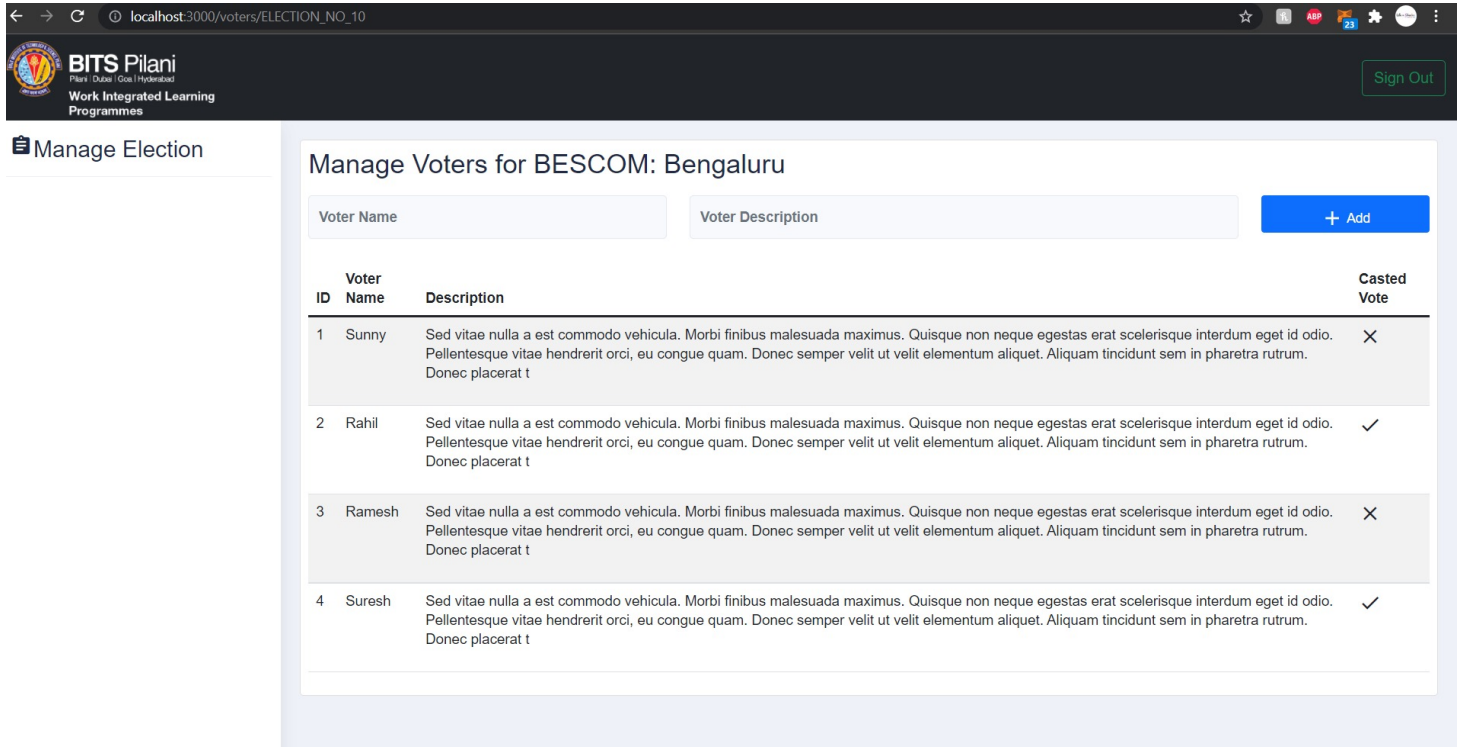
Adding Candidate to Election

- When the add button is clicked, then the candidate will be added to the blockchain data.
- The newly added candidate can be seen in the below list of candidate.



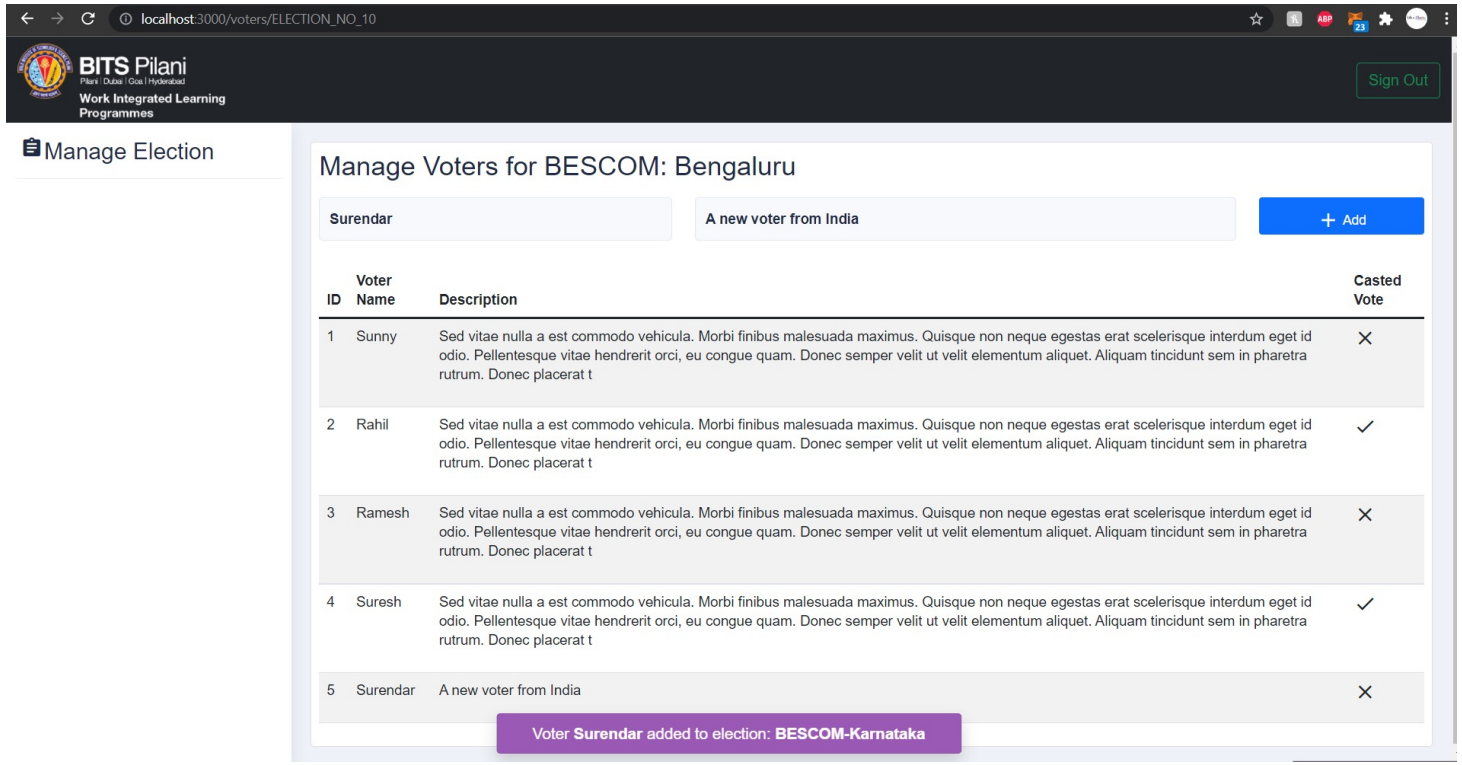
Manage Voters

- When manage voter button is clicked, one can manage the list of voters by adding them.
- As of now, we store the
  - Voter ID
  - Voter name
  - Description
  - If voter Casted a vote or not



Manage Voters: Add new Voter

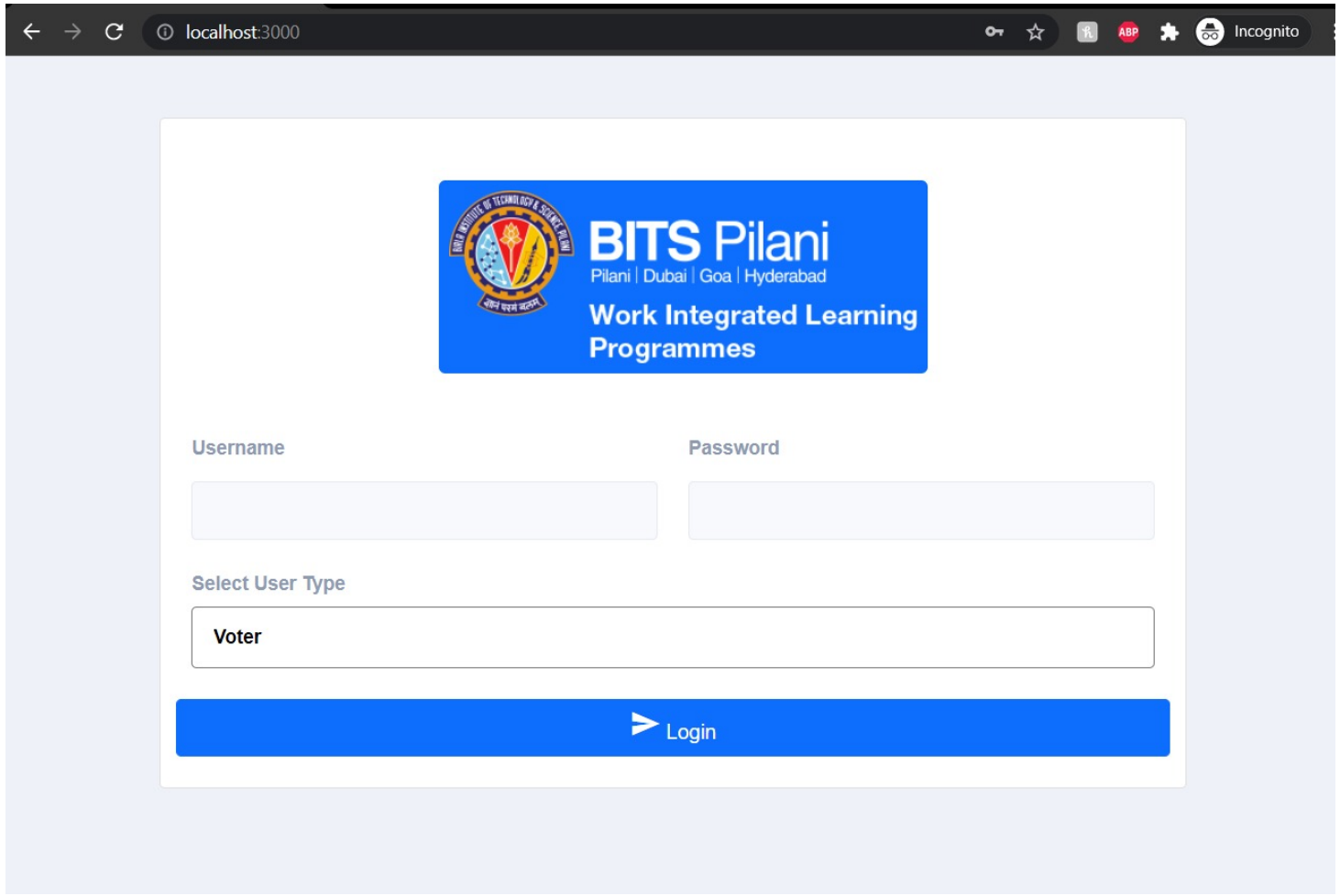
- When the add button is clicked, then the voter information will be added to the blockchain data under the current Election.
- The newly added candidate can be seen in the below list of candidate.



Voter workflow

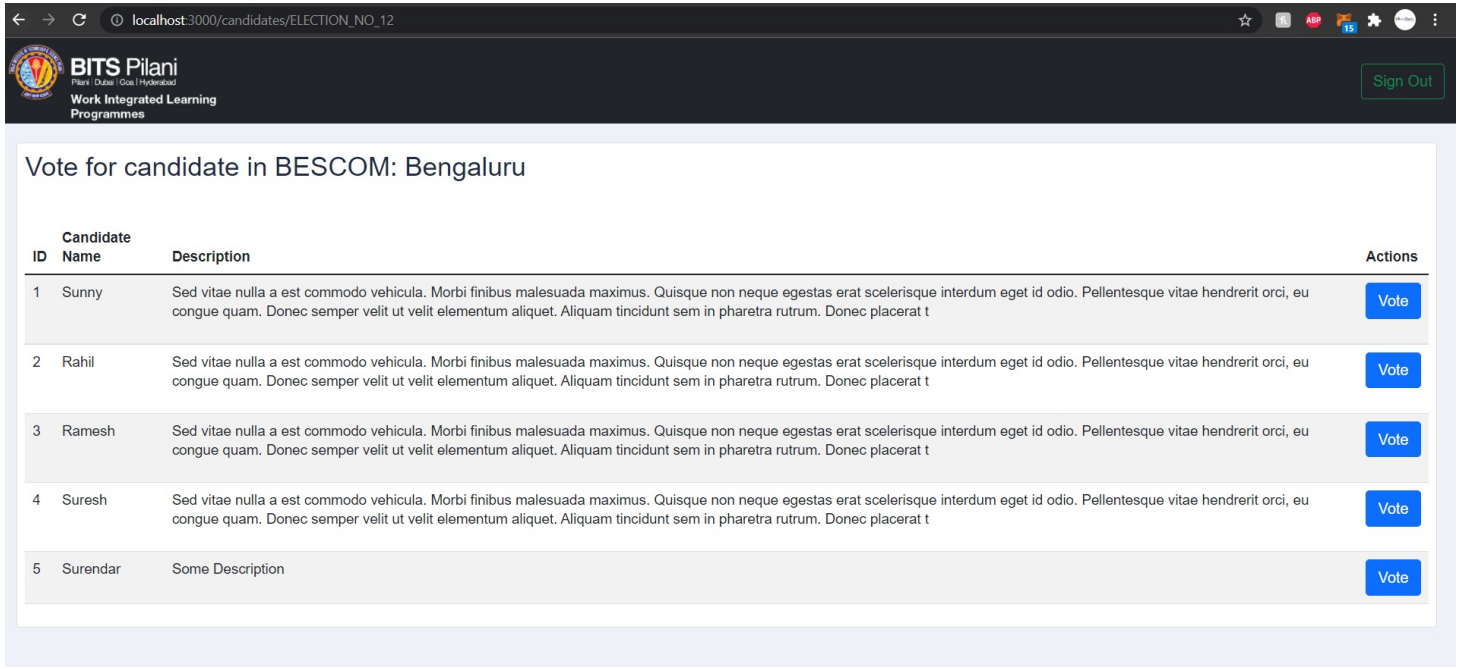
Login as Voter

- A voter can login to the application by selection the user type as **Voter**
- The valid Username will be the same name the provided while creating Voter (by admin)
- As of now the default password is 12345.



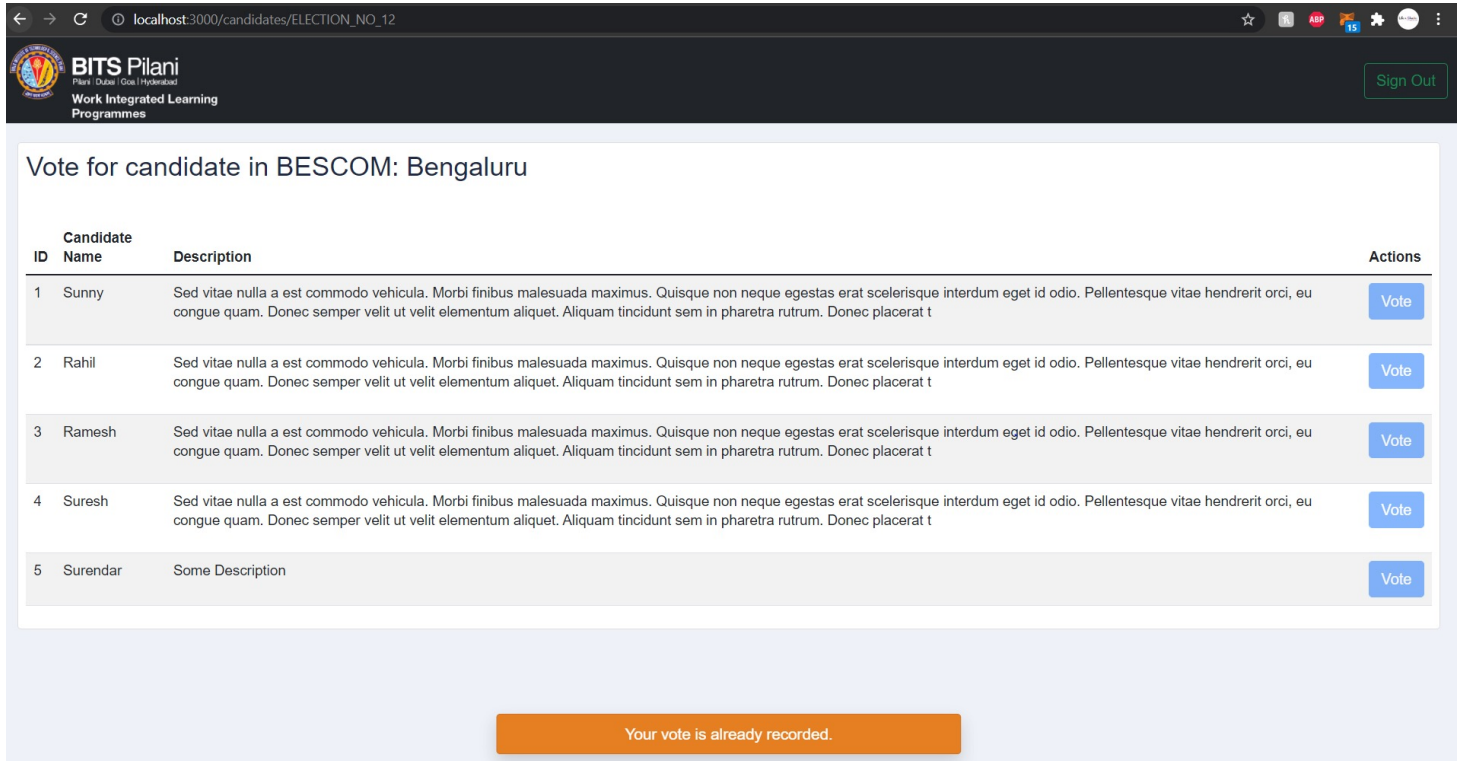
Voter Login: Vote for Candidate

- When the Voter logs in, then the list of candidates will be displayed.
- One can cast the vote for any set of the listed candidates.
- **‘ONLY ONCE’**.
- The **Double voting** is restricted.



Candidate Login: Voted for Candidate

- When a User tries to login again, then they will not be able to cast the vote.
- It is almost not possible to change the vote, because of the advantage of blockchain, **Data is Immutable.**
- **The vote button will be disable.**
- The total number of votes casted can be seen in admin page.



Explanation for Smart contracts

Smart contracts

- In this DAPP we have used two smart contracts ***(./src/contracts)***
  1. Election.sol
  2. Migration.sol
- **Election.sol**
  - This contract holds the model to store election details like
    - election name
    - List of candidates and their details *(id, name, desc, voteCount, candidatesCount)*
    - List of voters and their details *(id, name, desc, hasVoted, votersCount)*
  - Also, the methods to cast vote, add voter, add candidate
- **Migration.sol**
  - This is generated (automatically) when we do truffle migrate for the first time.

Revealing truffle-config.js

- The deployment is done in localhost, port 7545
- This is the same port where ganache RPC is running
- The built contracts are generated once contracts are migrated.

```
module.exports = {
    networks: {
      development: {
        host: "127.0.0.1",
        port: 7545,
        network_id: "*" // Match any network id
      },
    },
    contracts_directory: './src/contracts/',
    contracts_build_directory: './src/build/',
    compilers: {
      solc: {
        optimizer: {
          enabled: true,
          runs: 200
        }
      }
    }
}
```

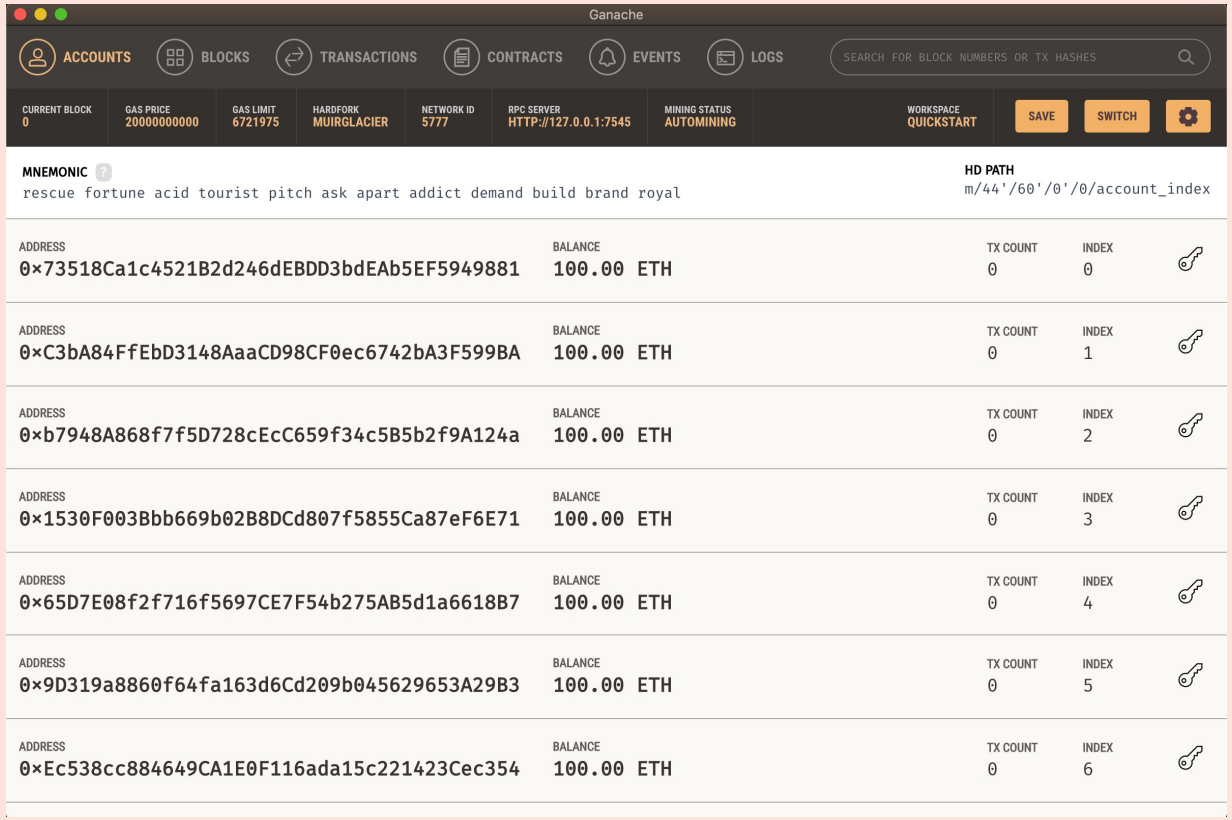
How to run truffle contracts in console

- Considering the pre-requisites (truffle), one can use terminal to deploy, and make changes to blockchain data using smart contracts.
- In order to do so, please follow these steps:
  1. Make sure you are in the project directory (../Blockchain-EVoting-UI) ***(Project Link available in the last slide)***
  2. To deploy the smart contract, use the following command `***truffle migrate --reset***`
  3. To start the truffle console, type in `***truffle console***`
  4. Now, we are good to start with command-line instructions to deal with smart conracts

Let us consider a real time scenario

- For an election to be conducted,
  - We need ‘n’ number of candidates enrolled for the election.
  - We need ‘n’ number of valid voters to cast their vote for the enrolled candidates.
  - **Important**: A voter can cast their vote only once, **NO DUPLICATE VOTING**
  - The one with the Maximum number of votes will win the election.
- Let us consider a similar scenario.
- We will be creating an Election-block, to which we will add three voters and three candidates.
- Let’s see who wins the election.

Ganache UI: Create workspace



- Make sure that ganache UI is up and running.
- We can also make use of ganache-cli. Use following command to install in local *`npm install ganache-cli`*
- Once ganache is up we can proceed with truffle commands

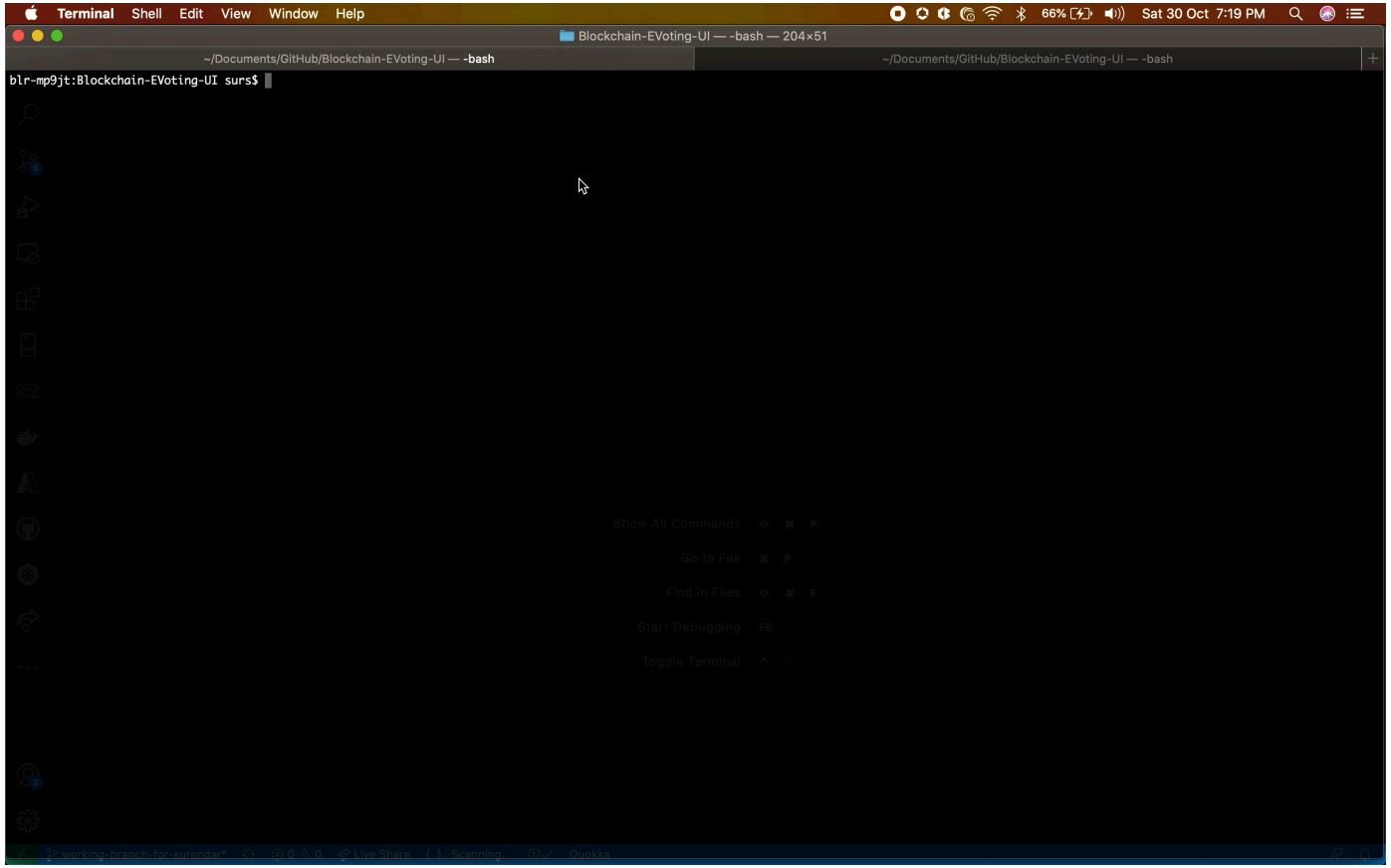
Truffle console commands

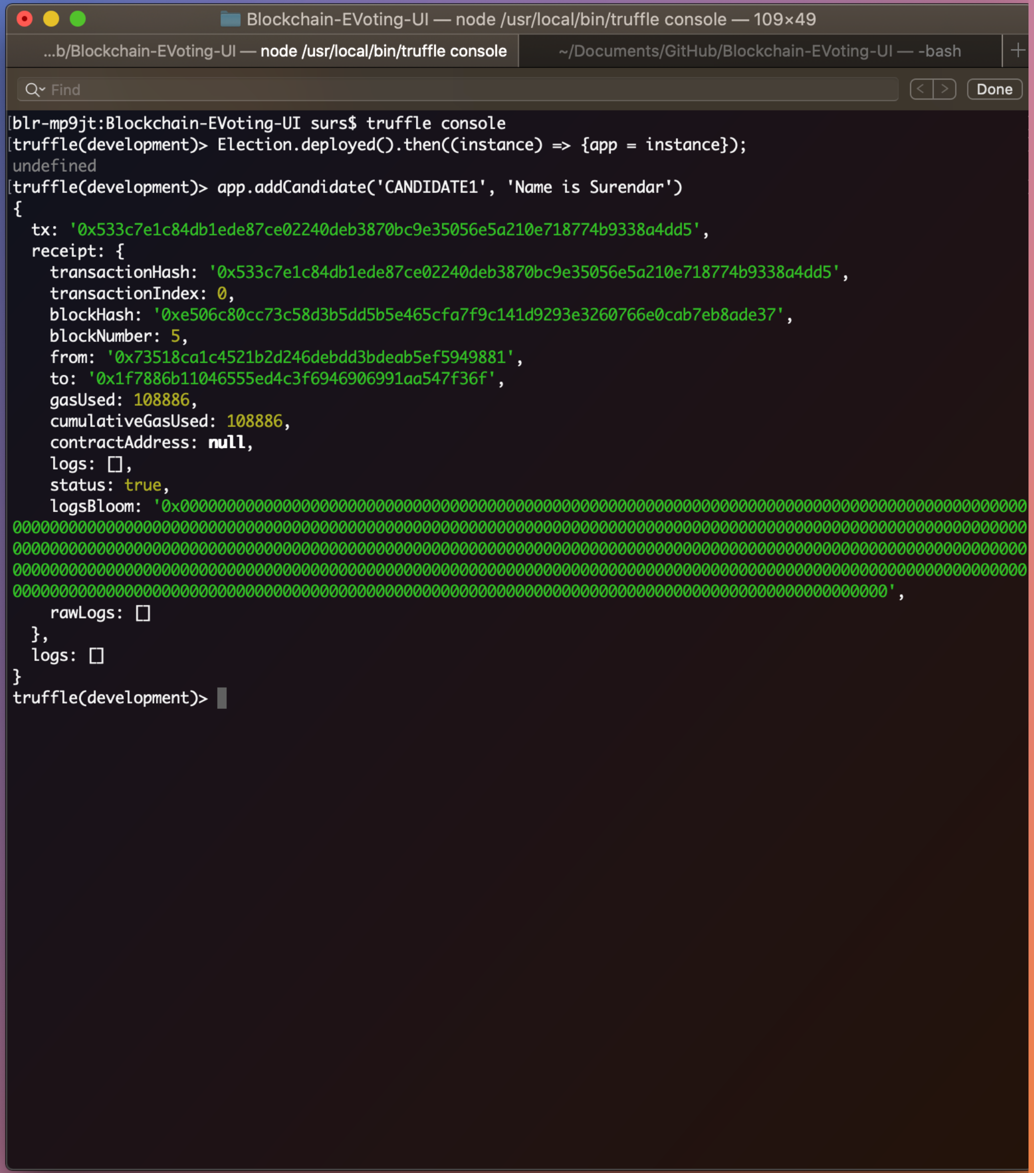
Step	Command	Description
1	truffle migrate --reset	Deploys the smart contract to Ethereum network
2	truffle console	To start the truffle console
3	Election.deployed().then((instance) => {app = instance})	To get the Election object (the deployed contract is a promise)
4	app.addCandidate('CANDIDATE_1', 'Name is Sunny')	To add a new candidate NOTE: Replace name and description appropriately
5	app.candidatesCount()	To get the count of candidates added
6	app.addVoter('VOTER_1', 'Name is Surendar')	To add a new candidate NOTE: Replace name and description appropriately
7	app.votersCount()	To get the count of voters added
8	app.vote(1,1)	To cast a vote for a candidate by a voter NOTE: the arguments are func vote(voter_id, candidate_id) as stored in blockchain

Migration of smart contracts

Summary

- Total deployments: 2
- Final cost: 0.02391962 ETH

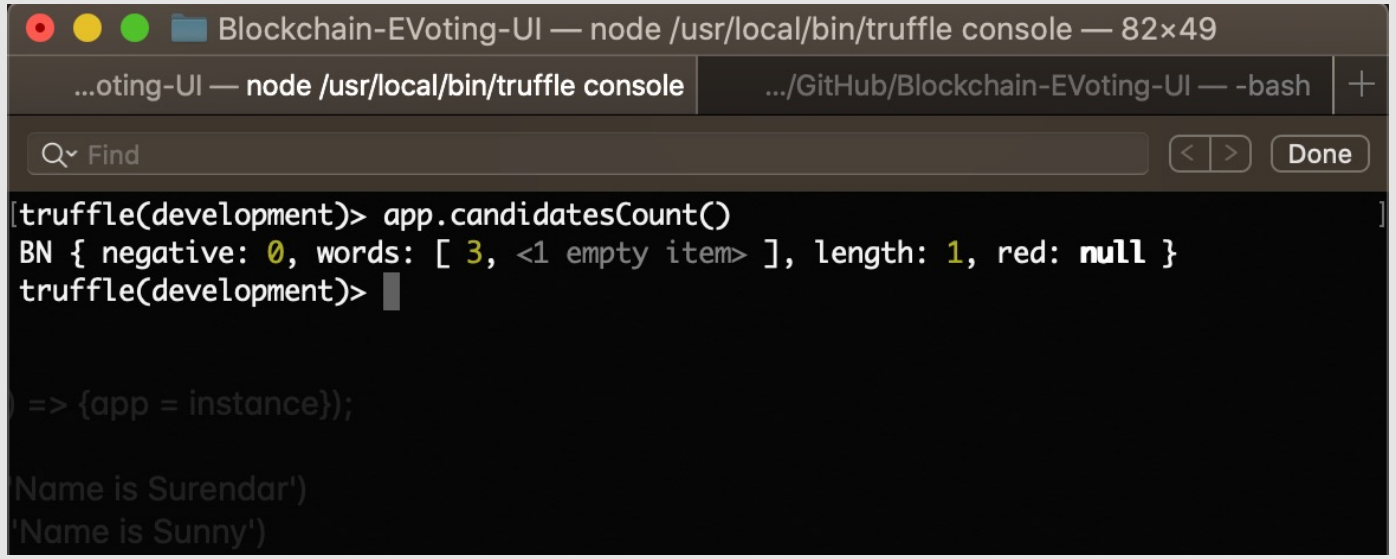




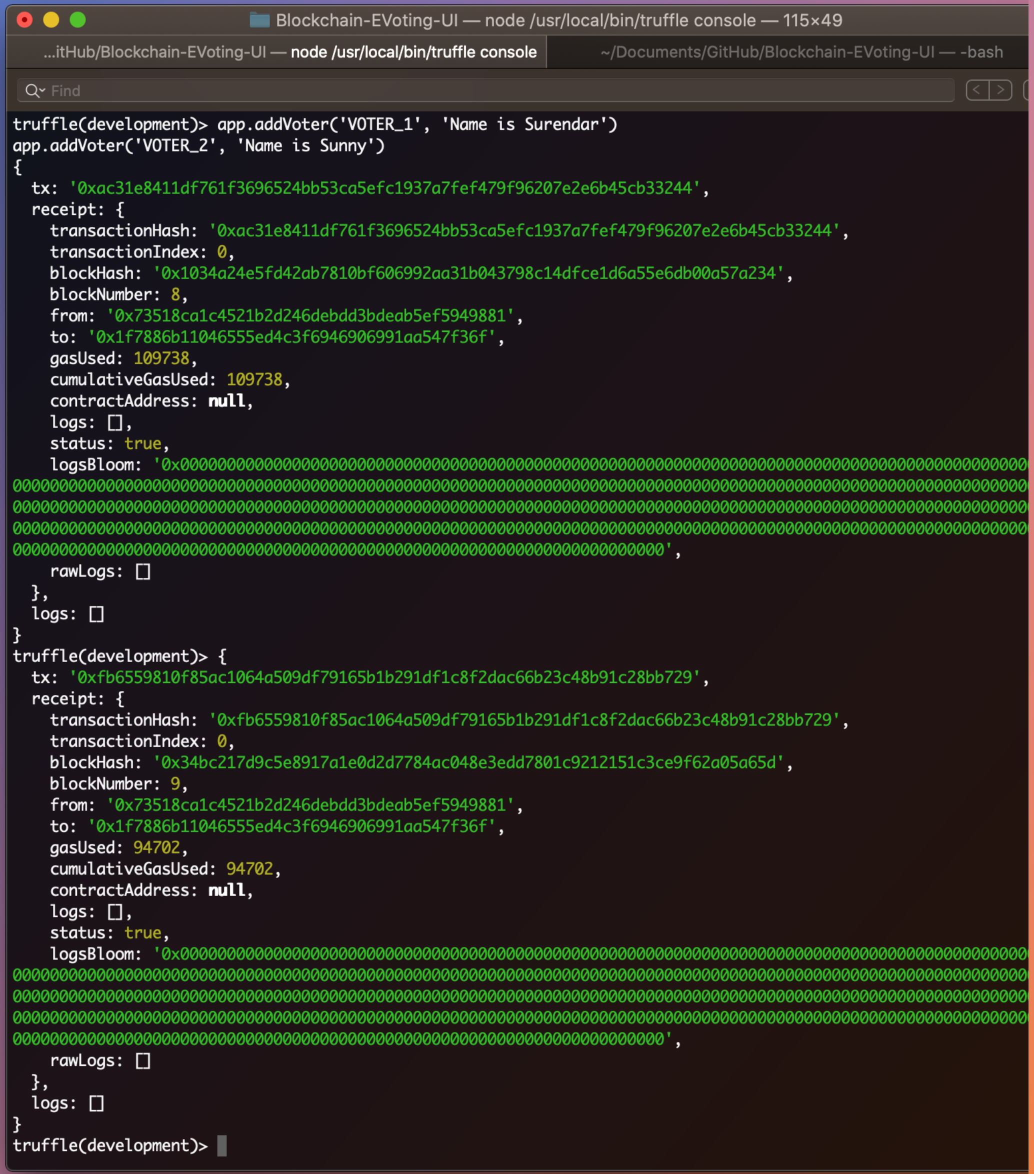
Add Candidate

- Start the Truffle console
- Get the Election object by resolving the promise
- Now add 3 candidates.
- For each addition of candidates, a transaction will be made to the Election block address.
- The data will be stored in the block.
- This method is integrated in the UI using web3.js to add candidates for an election by admin.
- The candidate voteCount is set to '0' initially.

Get the count of Candidates



- Once the candidates are added we can fetch the total number of candidates added
- This method is helpful in case of UI integration.
- The candidates are stored in (candidate id => Candidate struct) `mapping  uint => Candidate`
- This method has been used in UI to fetch and populate the Total candidates in Voters' page and admin page (manage candidates)
- In this case the Candidate count is **3,** as per the considered scenario.



Add Voter

- Now add 3 Voters.
- For each addition of voters, a transaction will be made to the Election block address.
- The data will be stored in the block.
- This method is integrated in the UI using web3.js to add Voters for an election by admin.
- The voter's "hasVoted" field is set to false initially
- This will be toggled when a vote is caseted.

Get the count of Voters

- Once the voters are added we can fetch the total number of candidates added
- This method is helpful in case of UI integration. (Admin page)
- The candidates are stored in mapping
(voter id => Voter struct)
`mapping  uint => Voter`
- This method has been used in UI to fetch and populate the Total candidates in admin page (manage voters)
- Here the voters count is **3,** as per the considered scenario.

```
Blockchain-EVoting-UI — node /usr/local/bin/truffle console — 111×49
...ub/Blockchain-EVoting-UI — node /usr/local/bin/truffle console    ~/Documents/GitHub/Blockchain-EVoting-UI — -bash
Find    Done
truffle(development)> app.vote(1,1)
{
  tx: '0x77236ef512cfca3f476dc3b9489e102c05b0513609a5655c3d0bc61fc27c90bd',
  receipt: {
    transactionHash: '0x77236ef512cfca3f476dc3b9489e102c05b0513609a5655c3d0bc61fc27c90bd',
    transactionIndex: 0,
    blockHash: '0xaeade91749ddb02c96f722390a74af599de1d88309a3cc23e31f29f55d22092b',
    blockNumber: 11,
    from: '0x73518ca1c4521b2d246debdd3bdeab5ef5949881',
    to: '0x1f7886b11046555ed4c3f6946906991aa547f36f',
    gasUsed: 66548,
    cumulativeGasUsed: 66548,
    contractAddress: null,
    logs: [ [Object] ],
    status: true,
    logsBloom: '0x0000000000000000000000000000000000000000000000000000000800000000000000000000000000000000000000
080000000000000000000000000000400000080000040000000000000000000000000000000000000040000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000080000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000004100000000000000000000000000000000000',
    rawLogs: [ [Object] ]
  },
  logs: [
    {
      logIndex: 0,
      transactionIndex: 0,
      transactionHash: '0x77236ef512cfca3f476dc3b9489e102c05b0513609a5655c3d0bc61fc27c90bd',
      blockHash: '0xaeade91749ddb02c96f722390a74af599de1d88309a3cc23e31f29f55d22092b',
      blockNumber: 11,
      address: '0x1F7886b11046555Ed4c3F6946906991Aa547F36f',
      type: 'mined',
      id: 'log_3c70c197',
      event: 'votedEvent',
      args: [Result]
    }
  ]
}
truffle(development)>
```

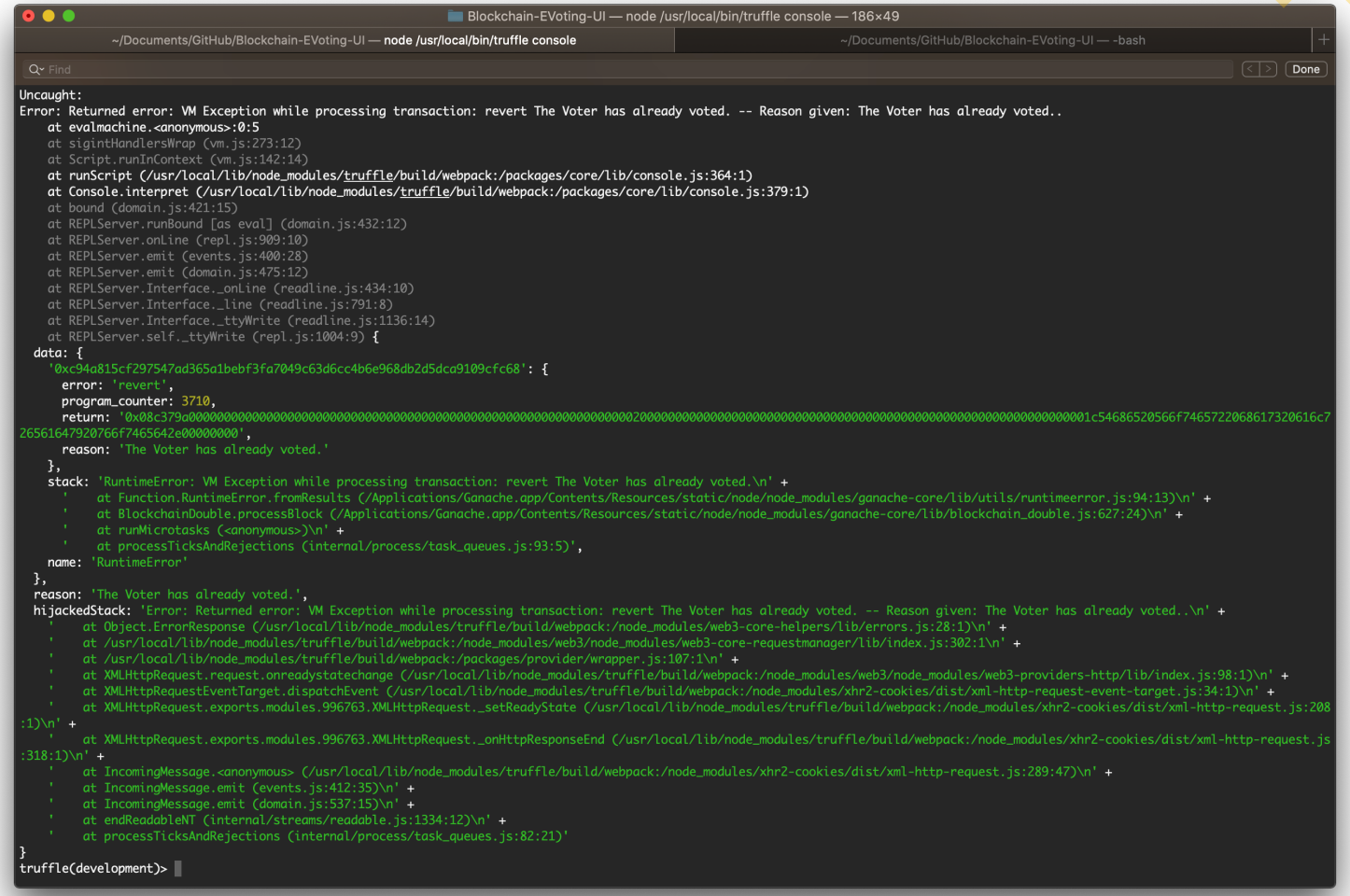
Now lets cast the vote…

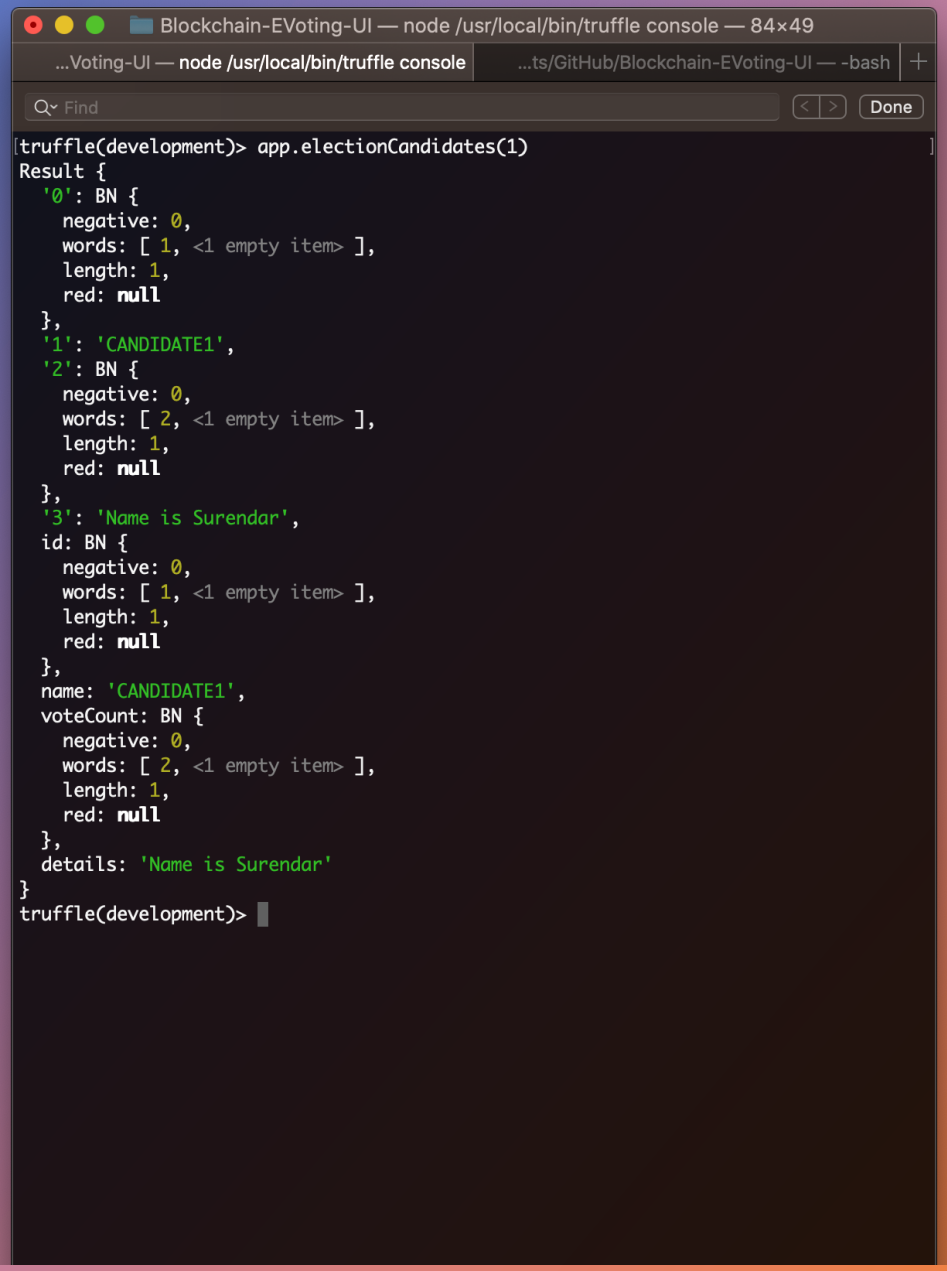
- We can make use of the function `*vote(voter_id, cand_id)*` to cast the vote.
- A voter can cast vote only once, as the hasVoted value will be set true after voting.
- In this scenario, the voter with id 1 (VOTER1) is casting the vote for the candidate with id 1 (Surendar).
- Similarly, voter 2 to candidate 1 and voter 3 to candidate 2, has casted their corresponding votes.

What happens when a voter tries to cast a vote again…?

- When a voter who already casted a vote tries to vote again, they will not be able to vote.
- We have added a validation check

  `require(!voterList[_voterId].hasVoted, 'The Voter has already voted.');`
- Hence, `***The Voter has already voted.***` message will be thrown as an error and the transaction will be failed.





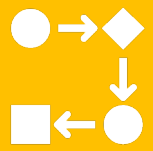
The candidate 1 has won the election…👏

- The candidate 1 got 2 votes, candidate 2 got 1 vote and candidate 3 got no vote.
- The member var, **voteCount** has been used in UI to show the vote count for a candidate.
- And this is how the smart contracts work.

Objectives that we achieved

Made the vote counting easier.

Since we make use of blockchain to store the data, it is immutable hence it adds security to the process workflow.



Avoidance of waiting for my turn to cast my vote, since the process has been automated now.

Scope and Extensions

Scope/ Limitations

- The scope of this Application is restricted to one election per admin.
- An admin can organize only a single election.

Extensions

This can be extended in such a way that, the smart contracts can be split into three different entities

- Election (indexed with address)
- Candidates (referenced with election address)
- Voters (referenced with election address)

Contribution split

Surendar S (2020MT93162)

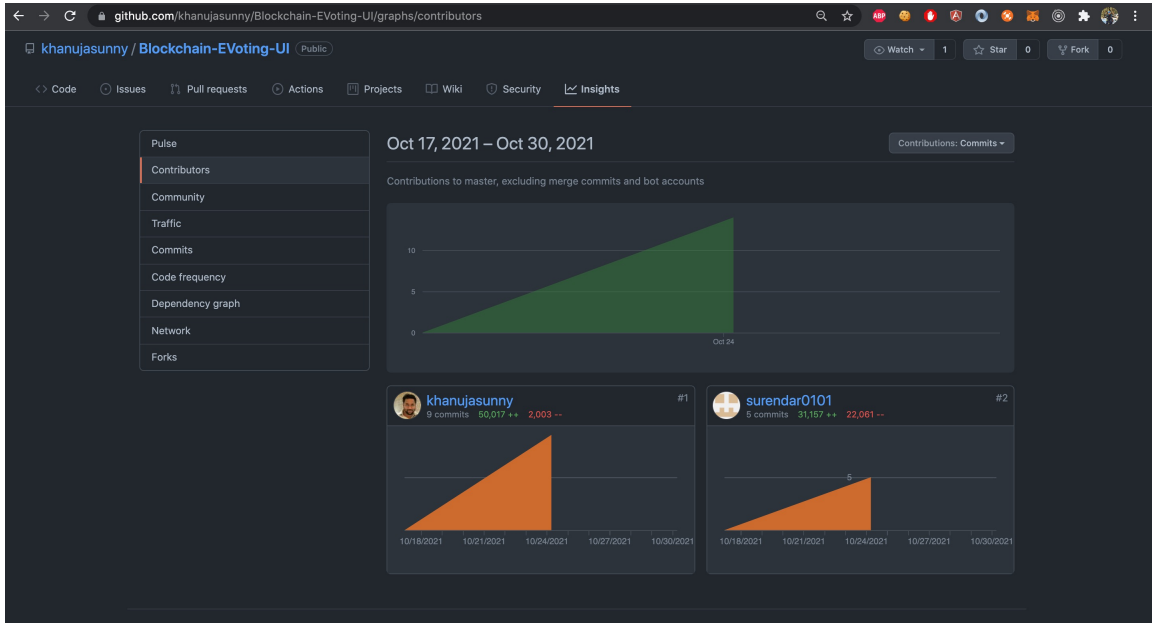
- API application For login
- Worked on smart contracts
- Worked on integrating UI application with web3.js
  (Few modules: Like adding a new Candidate)

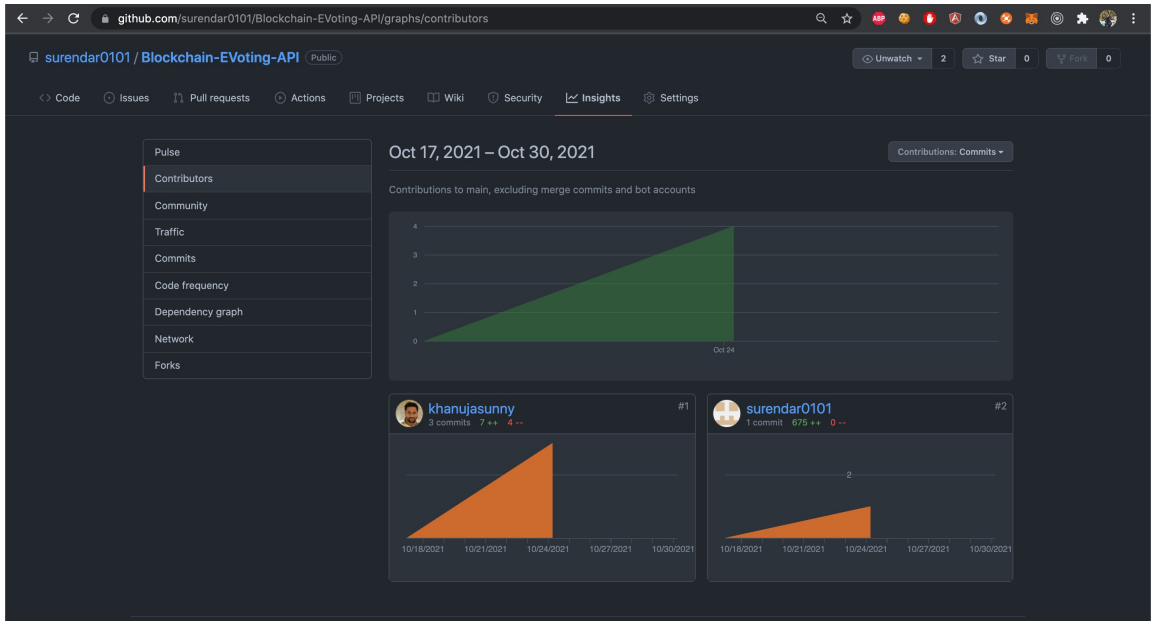
Sunny Khanuja (2020MT93110)

- Worked entirely setting up UI project
- Also, worked on Integrating the UI application

Overall, it was a collaborative work. We followed parallel coding.

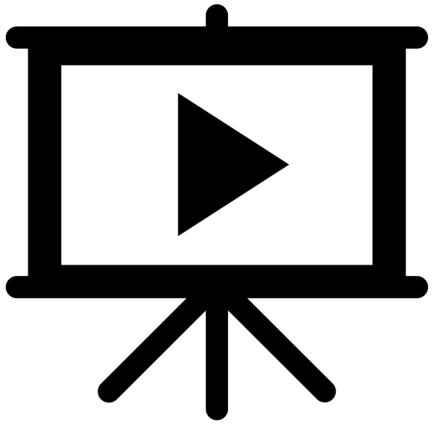
Work contribution (UI and API)

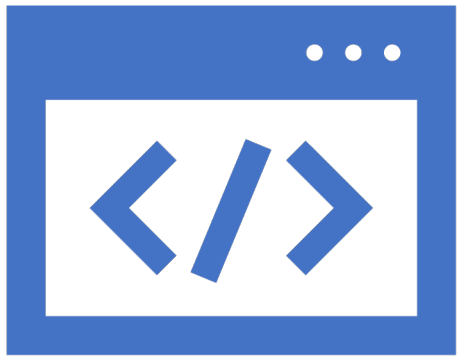




Learnings

- Working on smart contracts was something which gave us a lot of learning.
- The assignments gave us a lot of learning when compared to the theoretical content.
- Since we got to work practically, this gave us more understanding.
- The online lectures to work with metamask, ganache, Ethereum solidity added more learning.
- Overall, it was something new that we learnt.





Links

Github:

1. UI Application: https://github.com/khanujasunny/Blockchain-EVoting-UI
2. API Application: https://github.com/surendar0101/Blockchain-EVoting-API

Explanation Video:

(contains detailed workflow and code explanation)

https://drive.google.com/drive/u/1/folders/1OjIGRGbXGS1qT2LlQEPOWEb1IZ8n_geh

Thank you