Requirements:
Build an application that

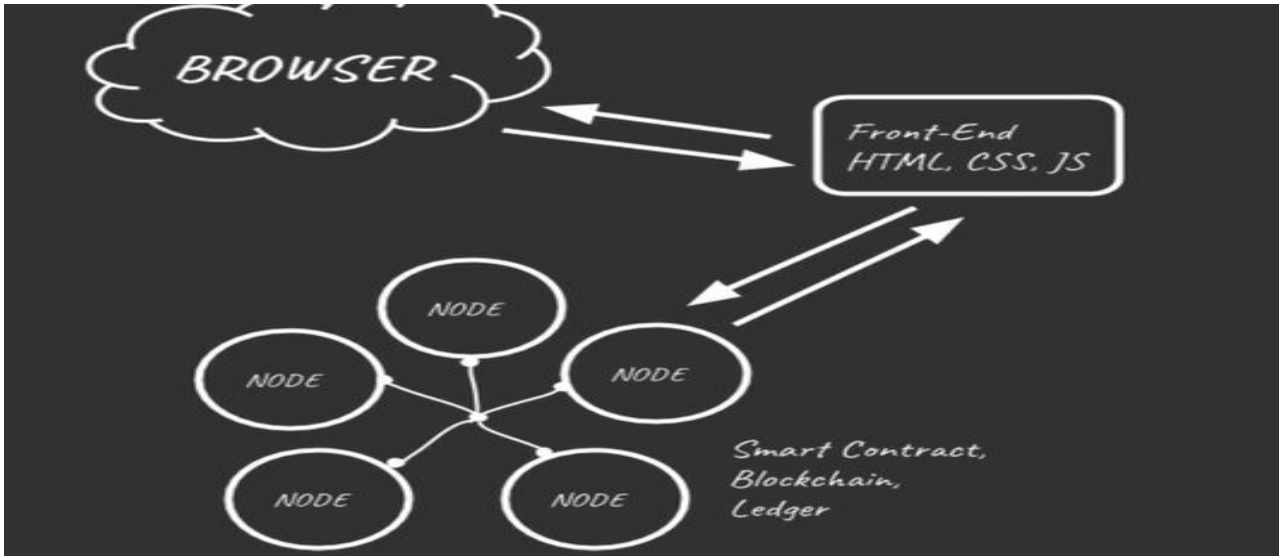
- Count votes once
- Doesn't change votes
- Most votes win
- No one should be able to change the rules by modifying the code

Cannot make it a web app whose database lies in a server, which makes it dependent that someone could modify it.
Thus, Blockchain!

What's a smart Contract and a DApp?

When we deploy our code that governs the rules of election on the ethereum blockchain, the nodes on the network will execute that code. The code gets executed on the ethereum virtual machine. This code will be secured and unchangeable.
This code where all the business logic, reading, writing data will lie, which communicates with other services is written using smart contracts. The way public ledger is like the database, Smart contract is like a microservice that sits on the web. Sort of like an agreement. A smart contract is written in solidity.

DApp is a decentralized application with a peer to peer network, shared data and code.
Structure of a DApp:



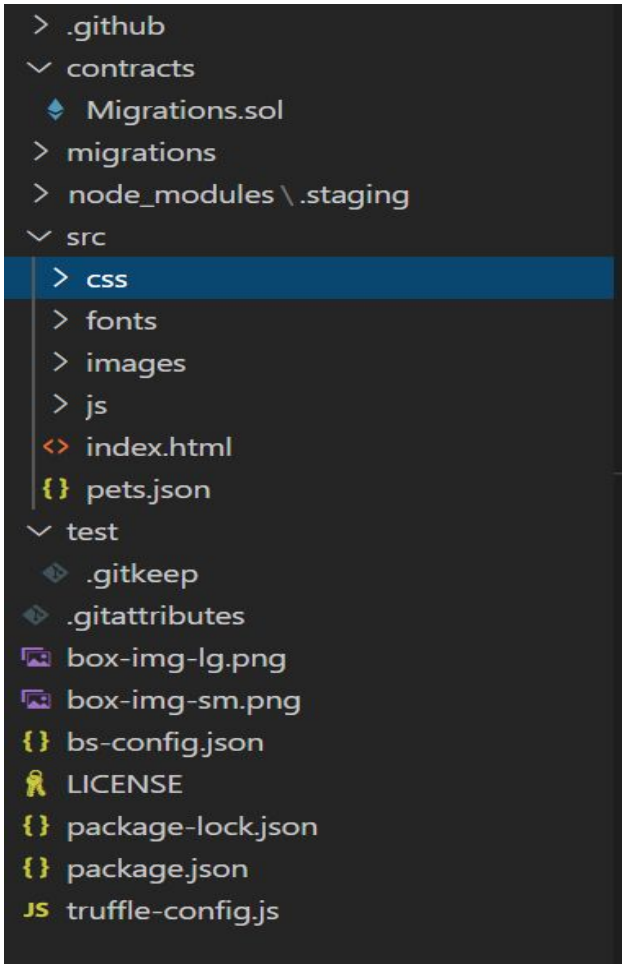
Dependencies:

- **NPM**. Thus install node as well
  node v10.15.0
- **Truffle:**
  Truffle is a framework used to create DApps on the ethereum network. We use it's tools to write a smart contract with solidity, to test our smart contracts, deploy our smart contracts to the blockchain, can develop client side applications inside truffle. It gives us boxes/packages of boiler code to start which we can unbox to get started.

`npm install -g truffle`
Truffle version 5.1.8

- **Ganache:**
  Local, memory blockchain used for development purposes.
- **Metamask:**
  To connect to the ethereum blockchain network and interact with our smart contracts. (Browser extension)
- **Solidity extension in vscode:**
  For syntax highlighting

Setting up:
Ganache starts local blockchain running and gives us 10 accounts with addresses, each of those accounts have 100 fake ethers.
We'll be using truffle pet-shop box, we unbox that package by :
`truffle unbox pet-shop`
This is what is gives:



Examining the directory structure:

- **Contracts:**
  This is where our smart contracts live. We already have Migrations.sol, it is going to handle our migrations whenever we deploy our smart contracts.
- **Migrations:**
  We used to get a migration file whenever we changed something in the database in django. Similarly, when we deploy a smart contract, transactions are created, and thus change in blockchain and a migration file is thus created.
  This is were the migrations file are stored when we deploy the smart contract to blockchain.
- **node_modules:**
  For the node dependencies
- **src:**
  For the client side application
- **test:**
  For the test files.
- **truffle.js:**
  Main configuration file for the truffle project
- **package.json:**
  For all the node project dependencies

Followed this DApp tutorial : https://www.youtube.com/watch?v=3681ZYbDSSk
Starting with the code:
Create a file Election.sol in contracts directory.
Version of solidity.

```
pragma solidity >=0.4.21 <0.6.0;
```

Declaring contract

```
contract Election {

}
```

Smoke test : to ensure everything is set up properly, contract is correctly set up, it can be deployed, it will respond the way we want it to, etc.

To create a migration to deploy our smart contract to a local blockchain so that we can interact with it through the console.
We name the migrations by appending an index_ to their name so truffle knows which order to run them in.

Create a 2_deploy_contracts.js file for creating migration:

```
var Election = artifacts.require("./Election.sol");

module.exports = function(deployer) {
  deployer.deploy(Election);
};
```

Reading Election contract in the same directory and assigning it to Election variable
Artifacts: represents contract abstraction that is specific to truffle. This will give us an election artifact that represents our smart contract and truffle will expose this so that we can interact with, maybe through console or tests or front end application.

Before deploying, use:

```
truffle migrate
```

To run the migration:

```
truffle migrate
```

To get the console:

```
truffle console
```

In console, write:

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

Election - defined inside our migration file
With deployed function we can get a copy of the deployed abstraction

Smart contract deployments are asynchronous in nature, thus the use of promises.
This then function would thus finish once the execution of asynchronous function deployed finishes. The callback function would take the ***instance*** of our application, and we assign it to to ***app*** variable
That's how we'll have access to our app once it is deployed.

```
truffle(development)> app.address
'0xe7a08c4dC402858CAbAED1a3e7C46eE9cDdab95b'
truffle(development)> app.candidate()
'Candidate_1'
```

candidate() is the getter function which solidity itself created for us.
Reads on the blockchain are free but writes cause gas.
To again run the smart contract we need to reset all the previous code similar to dropping tables in the databases, and creating new ones.

```
truffle migrate --reset
```

Truffle comes with Mocha - testing framework, and chai, which is an assertion library.
We create an election.js file in test folder to write the tests
To run the tests.

```
truffle test
```

App.js

- Initialise the app whenever the window loads
- Initialise web3 :
  Connects our client side application to the local blockchain.
- Initialise contract
- Render out the content of the application on the page

Lite-server used to build our client side.

`npm run dev`

To run the lite server

Whenever we do a transaction (here, vote), we get a transaction receipt. Reads in a blockchain are free, but writes cost gas.

Ethereum price : gasUsed * gasPrice

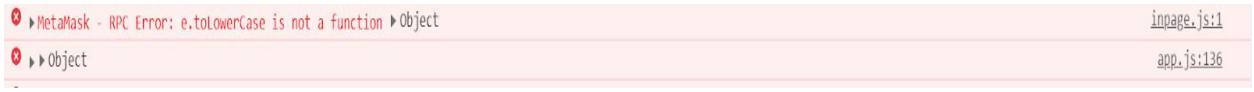
```
truffle(development)> app.vote(1,{from:FirstA})
{ tx:
   '0x55a2e4c333fa26d29b0cb3ef577ac62045d9e2bc46bebc6510208dc9acc75dd2',
  receipt:
   { transactionHash:
      '0x55a2e4c333fa26d29b0cb3ef577ac62045d9e2bc46bebc6510208dc9acc75dd2',
     transactionIndex: 0,
     blockHash:
      '0xa28b6639efbcfc43be755521d6ffc5a057d7c5a0ac0e0691fda1d9cddfdedb6e',
     blockNumber: 49,
     from: '0x983c4a963a28f8e281e83c21ceec5102490d1273',
     to: '0xea229988170c1653ba8b514b799066fbaa39e032',
     gasUsed: 62347,
     cumulativeGasUsed: 62347,
     contractAddress: null,
```

```
     logs: [],
     status: true,
     logsBloom:
      '0x0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000',
     v: '0x1b',
     r:
      '0xb41cce0a3cf8846fcf6862f02b59e76a4a5c5bb38249a4b200091d554e2c3ca7',
     s:
      '0x79930d628cc6bbc9e8ffe893a80eabb051767b200f1bef6f345c7c3600c4430b',
     rawLogs: [] },
  logs: [] }
truffle(development)>
```

The tests pass as following:

```
  Contract: Election
    √ initializes with two candidates (58ms)
    √ inititalizes the candidates with the correct values (111ms)
    √ allows a voter to cast a vote (311ms)
    √ throws an exception for invalid candidates (270ms)
    √ throws an exception for double voting (464ms)


  5 passing (1s)
```



Facing the above error!!
Can vote from the truffle console, but getting this error when voting from client side

Will update after solving the error!