

# BLOCKCHAIN TERM PROJECT

Swapnil Kishore  
Person Number: 50321753  
Email id: [kishore2@buffalo.edu](mailto:kishore2@buffalo.edu)

## PHASE 1: ABSTRACT

**Title:** Dapp for vote for new President of Operations.

**DAPP NAME:** VoteCorp-dapp

### REFERENCES:

<https://www.bcointalk.com/business/Blockchain-Voting-the-US-counties-rely-on-dApp-h2067.html>  
<https://voatz.com/>

**Clients and Benefactors:** Organizations and corporations looking for a trustworthy system to use for holding a vote to elect a member to a role/position. Members of the organization can take part in the voting process.

**Decentralized Issue:** The need for a reliable and simple system using which organizations can hold elections requiring members to be verified before being able to vote and the candidates to meet certain criteria to be eligible. The rules implemented by the dapp will ensure a fair voting process with any voter only being able to vote for a candidate once also ensuring that all votes are counted equally.

### Rules:

1) To be a candidate for President a person must:

- a) Be a member of the board of operations member for at least 5 years
- b) Be a part of the company for at least 10 years.
- c) Must submit to a background check regarding potential conflict of interest if appointed president.
- d) Must have valid links to social media profiles.
- e) Must be willing to sign a bond to stay as President for a minimum of 2 years if elected. (Bond can be broken in case of exceptional circumstance but only after approval from the Board of operations)

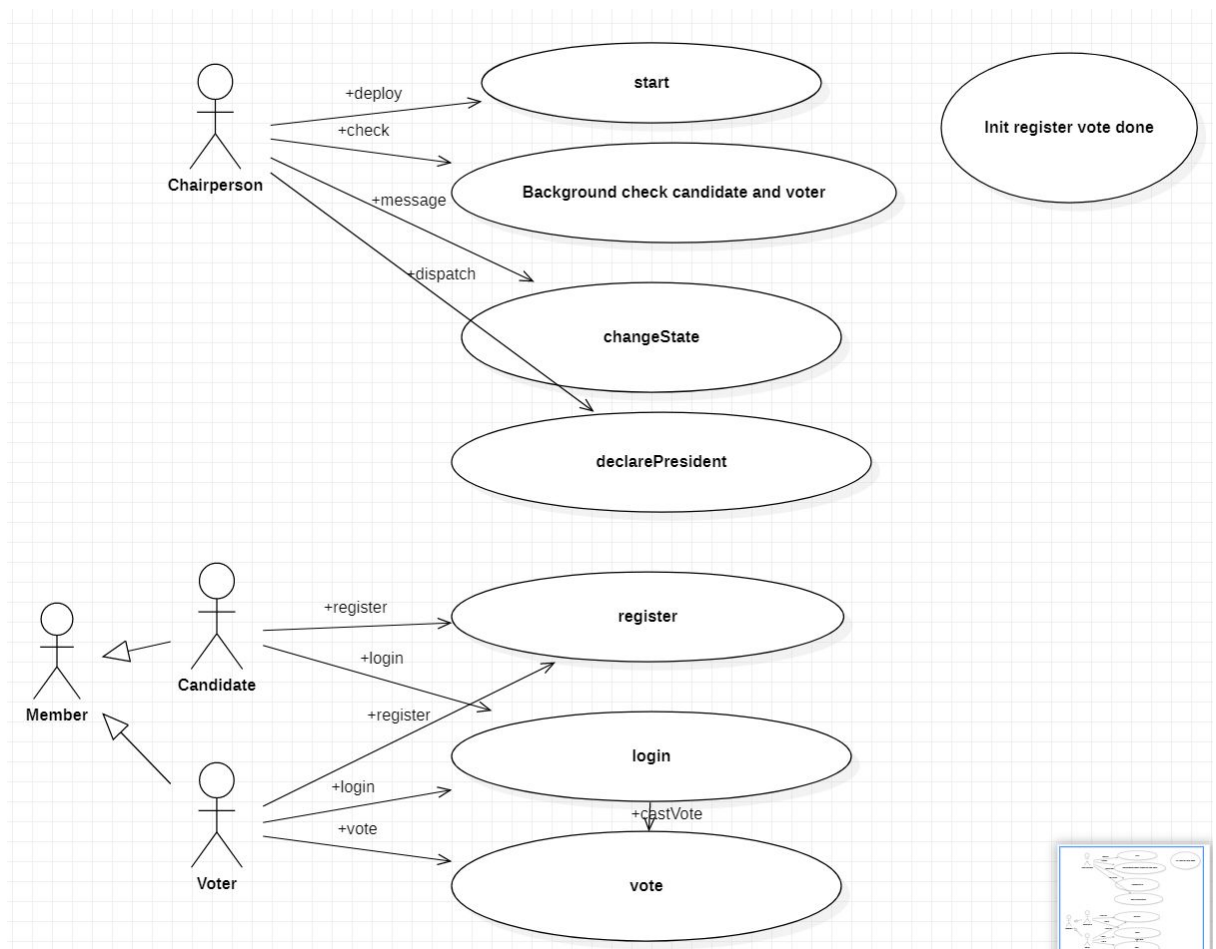
2) General voting rules for voters would include:

- a) Voter must be a member of XYZ company.
- b) Must submit to background check to make sure they have no personal relations to any candidate.

- c) Voters who have been a member of the board for more than 5 years will have their vote counter with twice as much weight as others.
- d) Must cast their vote before the deadline of two weeks from the starting date.
- e) If two or more candidates are tied for votes, the decision between those candidates will be given to the board of operations who will then decide who to appoint amongst themselves.

## PHASE 2: DESIGN DIAGRAMS

### 1) USE CASE DIAGRAM:



This is a use case diagram for VoteCorp-dapp. A use case diagram represents the actors and the use cases of the applications/system. It identifies the actors, the roles they will play and the functions that the application performs for the actors. The users for the VoteCorp-dapp are the Candidates and the Voters. Hence they are the actors in this case. The use cases for a candidate are:

- a) Register candidacy for verification: This includes registering their details to be a candidate so it can be verified by the admin/chairperson.
- b) Login: Candidates can login to access their registration or modify while the system is still in the Register phase.

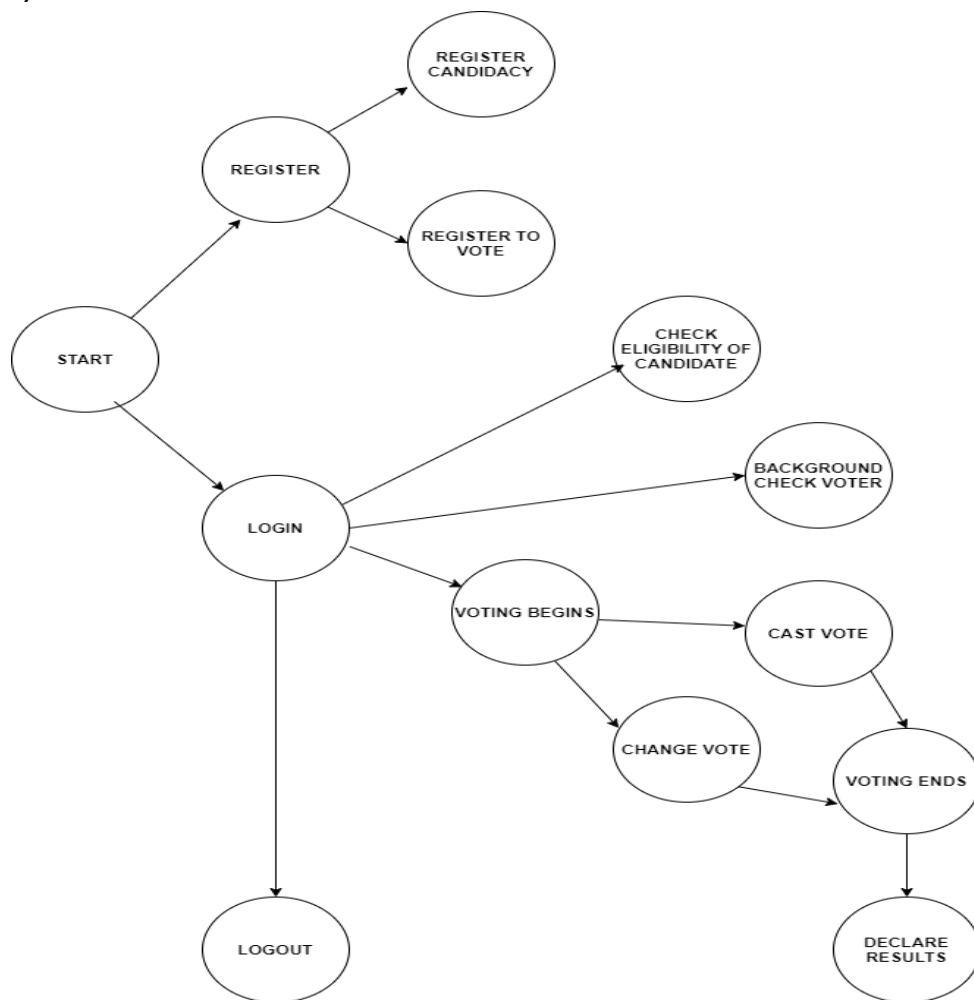
The use cases for a Voter are:

- a) Register to vote: The voters have to register with their required information to make sure it can be verified by the admin/chairperson before they can be eligible to cast their vote.
- b) Login: Voters can login to modify their details during the Register phase or cast/change their vote during the Vote phase.
- c) Vote: Voters cast their vote for their preferred candidate or change their vote. This can only be done as long as the system is in the Vote phase.

The use cases for a Admin/Chairperson are:

- a) Background check for Candidate: Check if the candidate is eligible using their registration details.
- b) Background check for Voter: Check if the voter is eligible to vote using their registration details.
- c) Publish results and declare winner: Only the admin can declare the winner of the election after the voting process has concluded.
- d) Change state: Only the chairperson has the authority to change the states of the system and voting can only begin once the system is in the voting phase.

## 2) FSM DIAGRAM:

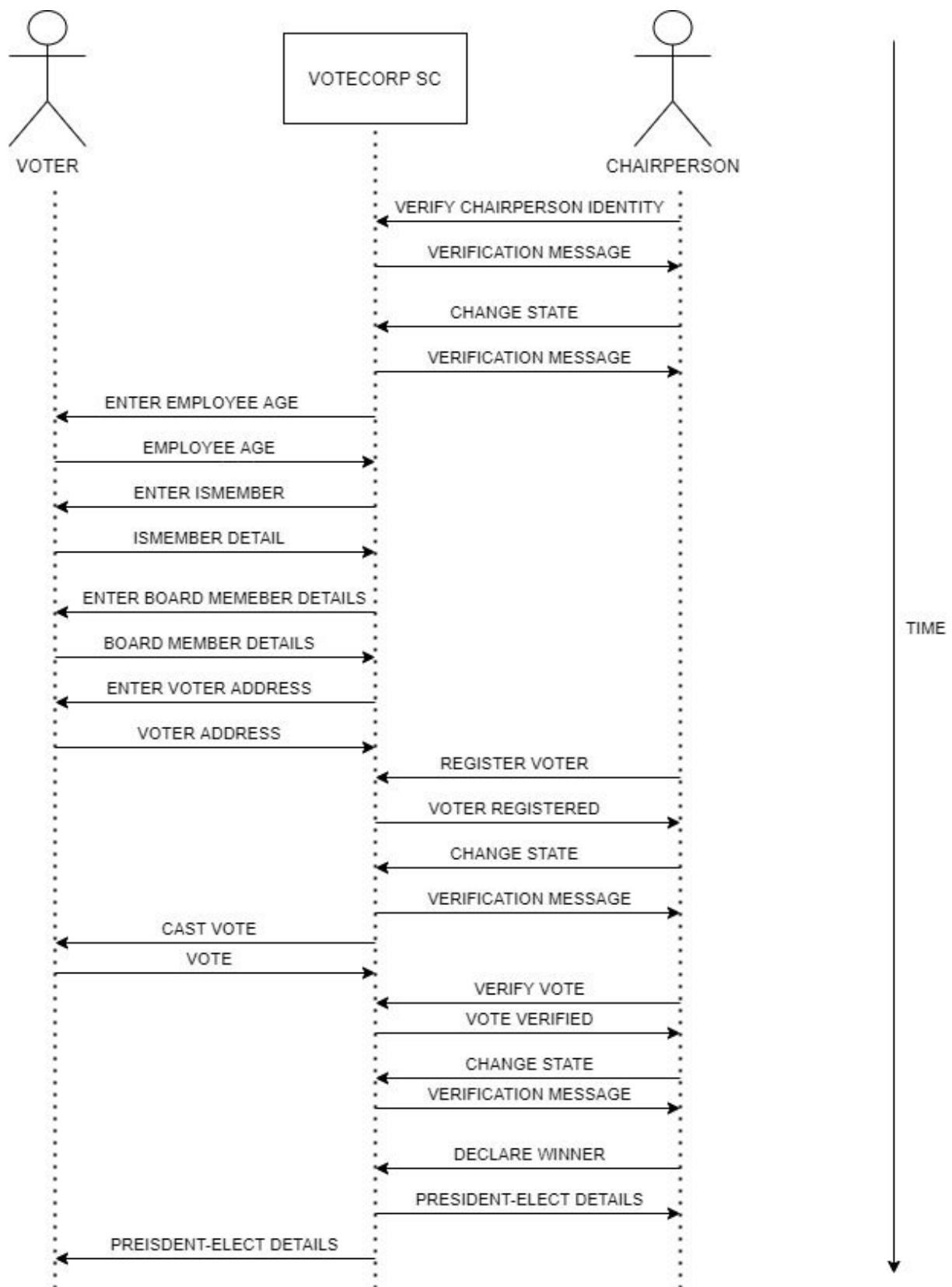


The FSM diagram represents the phases that the system goes through. The 4 main stages of the VoteCorp-dapp are Init, Register, Vote and Done. There are several states that comprise these main states. During the register phase, the users can register as voters or candidates and the admin must verify the registration details and eligibility criteria of the members. Only during the voting phase can the voters cast their vote or change it. Once the voting phase is over, no changes can be made to the votes and the winner can be declared by the chairperson.

### 3) CONTRACT DIAGRAM:

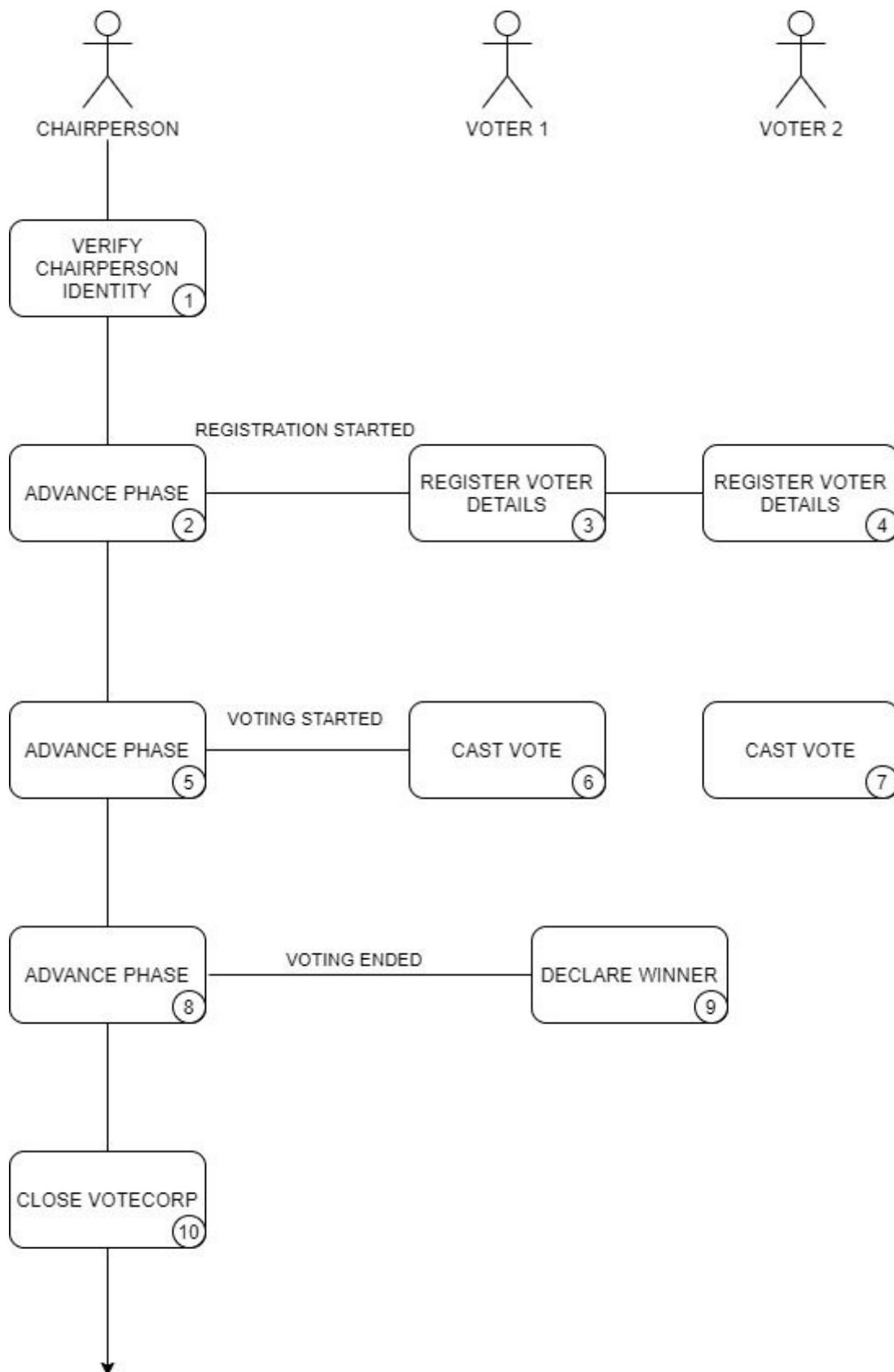
VoteCorp	
DATA	struct voter{
	address chairperson;
	Proposal[] proposals;
	address voterAddress;
	mapping(address => Voter) voters;
	enum Phase {Init, Regs, Vote, Done}
	Phase public state = Phase.Done;
MODIFIERS	modifier validPhase(Phase reqPhase) { require(state == reqPhase); }
	modifier onlyChair() {require(msg.sender == chairperson); }
	modifier onlyMemberofCorp() {require(voters[voterAddress].isMember== true); }
	modifier onlyMemberofBOP() {require(voters[voterAddress].isBOP== true); }
	modifier onlyLongTerm() {require(voters[voterAddress].termAge>= 10); }
	modifier checkRegister() {require(voters[voterAddress].isRegistered== true); }
FUNCTIONS	constructor (uint8 numProposals) public { }
	enterAddress(address voter) public { }
	enterTerm(uint8 Term) public { }
	enterMemberofCorp(bool isMember) public { }
	enterMemberofBOP(bool isBOP) public { }
	changeState(Phase X) public { }
	register(address voter)public validPhase(Phase.Regs) onlyMemberofCorp() onlyChair { }
	registerCandidate(address voter)public validPhase(Phase.Regs) onlyMemberofCorp() onlyMemberofBOP() onlyLongTerm() onlyChair { }
	vote(uint8 _toProposals) public validPhase(Phase.Vote) checkRegister() { }
	declarePresident() public { }

#### 4) SEQUENCE DIAGRAM:



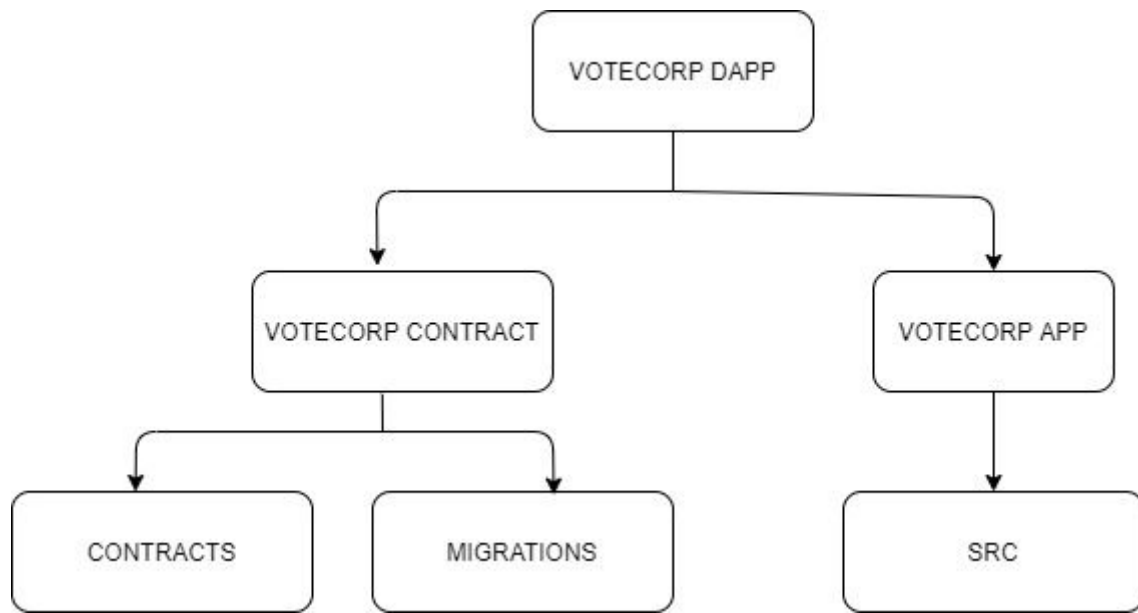
The sequence diagram details the sequence of interactions that take place within the system between the voters, chairperson and the smart contract.

## 5) INTERACTION DIAGRAM:



This is how the client/users interact with the dapp. The steps are numbered in a linear manner from 1-10.

## 6) DIRECTORY DIAGRAM:



This is the directory structure of the Dapp visualized as a diagram. There are two main directories, one for the contracts and one for the web application. These two main directories have several sub-directories and files under them.

## PHASE 3

### Remix Screenshots:

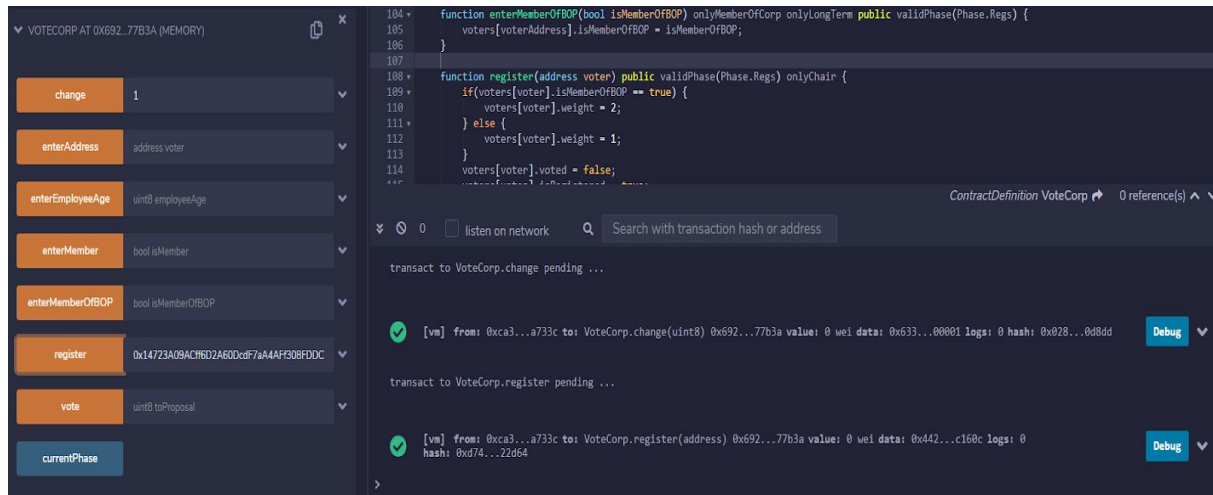
The screenshot displays the Remix IDE interface. On the left, a sidebar shows the 'VOTECORP AT 0X692...77B3A (MEMORY)' contract with a list of functions: change, enterAddress, enterEmployeeAge, enterMember, enterMemberOfBOP, register, vote, currentPhase, declarePresident, and voterCount. The main editor shows the Solidity source code for the Votecorp contract, which includes a Voter struct, a Proposal struct, and various functions like register, validPhase, and onlyChair.

### Deployed contract on Remix

The screenshot shows the Remix IDE with the Votecorp contract deployed. The 'register' function is being called, but it fails with a 'VM error: revert. revert The transaction has been reverted to the initial state. Reason provided by the contract: "Invalid Phase".' The error message is displayed in the console, and the transaction is shown as 'pending'.

### Negative test case on remix: Failed registration of voter while on Init phase.





**Positive test case on remix: Successful registration of voter after changing stage to Regs.**

## SMART CONTRACT:

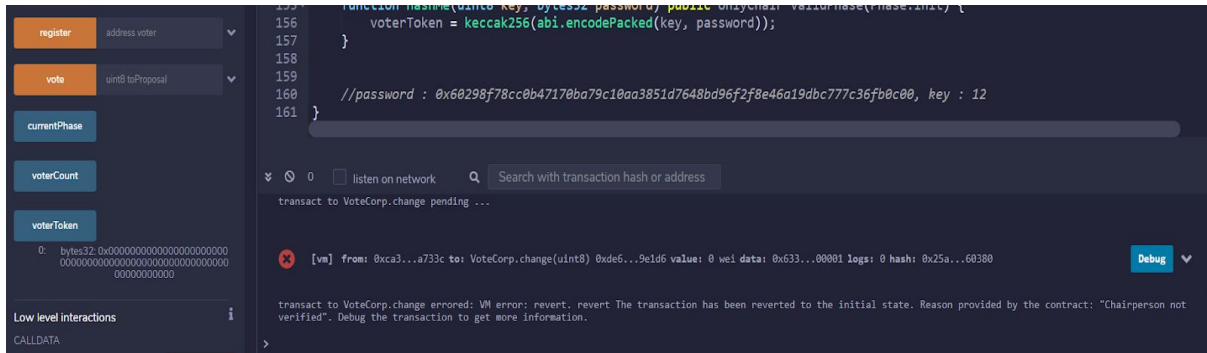
### Execution Instructions:

- 1) Install Ganache test chain
- 2) Navigate to the VoteCorp-contract directory and use the command **truffle compile** to compile the contracts
- 3) The configuration details for deployment are present in the truffle-config.js file and can be modified as per usage. Currently it is set to listen on port 7545.
- 4) The last step before deployment is to configure the file 2\_deploy\_contracts.js. This is used to initialize the contract by making use of the constructor defined in the contract. Specify the number of candidates (proposals) as a parameter and then the contract is ready to be deployed.
- 5) To deploy, navigate to VoteCorp-contract directory and use the command **truffle migrate --reset** to deploy. The reset option will redeploy all contracts. Only deploy without the reset option if you are ready for deployment for the production phase. Use the reset option when in development to test and debug.

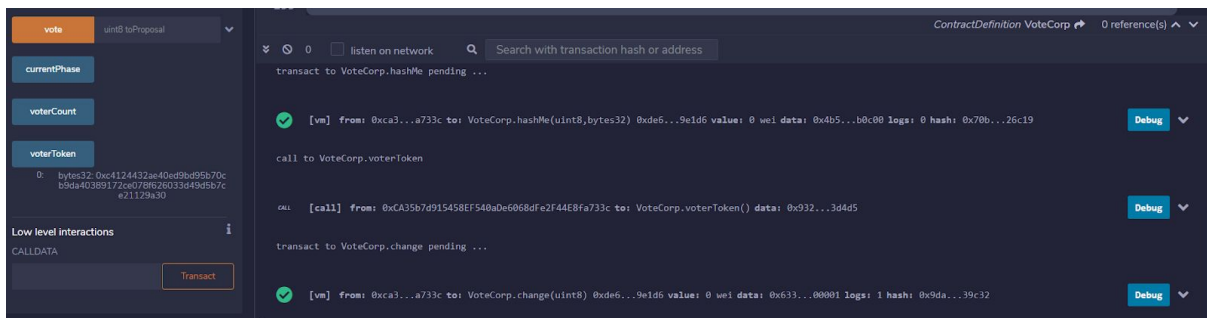
## PHASE 4

### HASHING AND SECURITY:

The keccak hash function available in solidity is used to verify the identity of the chairperson to provide secure access to VoteCorp. The chairperson can only perform their functions such as **register** and **changeState** without first verifying their identity by passing the key and password required to the hash function.



**Failed test case: Chairperson cannot verify identity and hence cannot change state.**



**Successful test case: Chairperson generates voterToken with correct key and password and is able to change state.**

### EVENTS:

Events have been added that alert the voter when the state of the system is changed and also when the voter's vote has been cast. An event alert is also generated when the chairperson decides to declare the winner of the election so that all participants are informed who the winner is.





## PHASE 5

### WEB APP:

#### Execution Instructions:

- 1) After deploying the smart contract, navigate to the VoteCorp-app directory
- 2) From the command prompt, run the command `npm install` to install the node modules required to run the web application.
- 3) After having installed the node modules run the command `npm start`
- 4) The web application is configured to run on port 3000. Hence navigate to the address <http://localhost:3000> to access the web app.

*Vote for your candidate of choice.*

Jason Schriener	William Christensen	Alexandria Smith	Derrick Henry
			
<input type="button" value="Vote"/>	<input type="button" value="Vote"/>	<input type="button" value="Vote"/>	<input type="button" value="Vote"/>

Address :	<input type="text"/>	<input type="button" value="Submit"/>	ChairPerson Address	<input type="text"/>
<input type="button" value="Change State"/>	Enter state: (0: Init, 1: Regs, 2: Vote, 3: Dor)		<input type="button" value="Submit"/>	
Employee Age:	<input type="text" value="Enter Your employee age"/>	<input type="button" value="Submit"/>		

## PHASE 6

### PUBLIC DEPLOYMENT ON INFURA:

The smart contract is deployed on Infura after creating an account and using Infura to obtain ropsten endpoints for public access. This allows participants from any location to access the smart contract if they have the web application code. These are the steps required to interact with the smart contract deployed on infura:

- 1) Configure the app.js of the web application with the ropsten endpoint url and the smart contract address as shown below:

```
1  App = {
2    web3Provider: null,
3    contracts: {},
4    names: new Array(),
5    url: '',
6    chairPerson:null,
7    currentAccount:null,
8    address:'',
9    init: function() {
10     $.getJSON('../proposals.json', function(data) {
11       var proposalsRow = $('#proposalsRow');
12       var proposalTemplate = $('#proposalTemplate');
13
14       for (i = 0; i < data.length; i++) {
15         proposalTemplate.find('.panel-title').text(data[i].name);
16         proposalTemplate.find('img').attr('src', data[i].picture);
17         proposalTemplate.find('.btn-vote').attr('data-id', data[i].id);
18
19         proposalsRow.append(proposalTemplate.html());
20         App.names.push(data[i].name);
21       }
22     });
23     return App.initWeb3();
24   },
25
26   initWeb3: function() {
27     // Is there is an injected web3 instance?
28     if (typeof web3 !== 'undefined') {
29       App.web3Provider = web3.currentProvider;
30     } else {
31       // If no injected web3 instance is detected, fallback to the TestRPC
32       App.web3Provider = new Web3.providers.HttpProvider(App.url);
33     }
34     web3 = new Web3(App.web3Provider);
35
36     ethereum.enable();
37
38     App.populateAddress();
39     return App.initContract();
40   },
41 }
```

- 2) Once the app.js has been configured save it and open a terminal at the location of the web application.
- 3) Use the command **npm install** to install all the node modules required to run the web application.
- 4) Use the command **npm start** to start the web application
- 5) Open metamask using your ropsten account.
- 6) Navigate to <http://localhost:3000/> and give permission to metamask to connect to the web application. After this, the user is able to interact with the smart contract deployed on infura using the web app.

### CREDENTIALS:

- 1) **SMART CONTRACT ADDRESS:**

**0x687Afa812D4F920A2DcD51456D8a705F42CA9caA**

## PHASE 7

### SMART CONTRACT CODE:

```
pragma solidity >=0.4.22 <=0.6.6;

contract VoteCorp {
    struct Voter {
        uint weight;
        uint8 employeeAge;
        uint8 vote;
        uint8 numVotes;
        bool isMemberOfBOP;
        bool isMember;
        bool isRegistered;
        bool voted;
    }

    struct Proposal {
        uint voteCount;
        //bool bondAvail;
    }

    address chairperson;
    address voterAddress;
    Proposal[] proposals;
    mapping(address => Voter) voters;
    //mapping(address => Proposal) proposals;
    enum Phase {Init, Regs, Vote, Done}
    Phase public currentPhase = Phase.Init;
    uint8 public voterCount = 0;
    bytes32 public voterToken;

    event DapplInit();
    event RegsStarted();
    event VotingStarted();
    event VotingClosed(uint8 winningProposal);
    event voted(string message);

    modifier validPhase(Phase reqPhase) {
        require(currentPhase == reqPhase, "Invalid Phase");
        _;
    }

    modifier onlyChair() {
        require(msg.sender == chairperson, "Only Chairperson");
```

```

    _;
}

modifier onlyMemberOfCorp() {
    require(voters[voterAddress].isMember == true, "Member needed");
    _;
}

modifier onlyMemberOfBOP() {
    require(voters[voterAddress].isMember == true &&
voters[voterAddress].isMemberOfBOP == true, "Board Member needed");
    _;
}

modifier checkMax()
{
    require(voters[voterAddress].numVotes <= 3, "Vote limit reached!");
    _;
}

modifier onlyLongTerm() {
    require(voters[voterAddress].employeeAge >= 10, "Long term employee
needed");
    _;
}

modifier checkRegister() {
    require(voters[voterAddress].isRegistered == true, "Not registered");
    _;
}

modifier checkVoted() {
    require(voters[voterAddress].voted == false, "Already voted");
    _;
}

// modifier bondCheck() {
//     require(proposals[voterAddress].bondAvail == true, "Must accept bond to
register as candidate!");
//     _;
// }

constructor (uint8 numProposals) public {
    chairperson = msg.sender;
    for (uint8 prop = 0; prop < numProposals; prop ++)
        proposals.push(Proposal(0));
    voters[chairperson].weight = 2;
    //state = 1;
}

```

```

}

function change(Phase phase) onlyChair public {

    require(voterToken ==
0xc4124432ae40ed9bd95b70cb9da40389172ce078f626033d49d5b7ce21129a30,
"Chairperson not verified");
    currentPhase = phase;
    if (currentPhase == Phase.Vote) emit VotingStarted();
    if (currentPhase == Phase.Regis) emit RegisStarted();
    if (currentPhase == Phase.Init) emit Daplnit();
}

function enterAddress(address voter) public validPhase(Phase.Regis) {
    voterAddress = voter;
}

function enterEmployeeAge(uint8 employeeAge) public validPhase(Phase.Regis) {
    voters[voterAddress].employeeAge = employeeAge;
}

function enterMember(bool isMember) public validPhase(Phase.Regis) {
    voters[voterAddress].isMember = isMember;
}

// function enterBondAvail(bool bondAvail) public validPhase(Phase.Regis) {
//     proposals[voterAddress].bondAvail = bondAvail;
// }

function enterMemberOfBOP(bool isMemberOfBOP) onlyMemberOfCorp
onlyLongTerm public validPhase(Phase.Regis) {
    voters[voterAddress].isMemberOfBOP = isMemberOfBOP;
}

function register(address voter) public validPhase(Phase.Regis) onlyChair {
    require(voterToken ==
0xc4124432ae40ed9bd95b70cb9da40389172ce078f626033d49d5b7ce21129a30,
"Chairperson not verified");
    require(voters[voter].isRegistered == false, "Already registered");
    if(voters[voter].isMemberOfBOP == true) {
        voters[voter].weight = 2;
    } else {
        voters[voter].weight = 1;
    }
    voters[voter].voted = false;
    voters[voter].isRegistered = true;
    voterCount++;
}

```

```

function vote(uint8 toProposal) public validPhase(Phase.Vote) checkRegister
checkMax {
    //Voter memory sender = voters[voterAddress];
    require(toProposal < proposals.length);
    voters[voterAddress].voted = true;
    voters[voterAddress].vote = toProposal;
    voters[voterAddress].numVotes++;
    proposals[toProposal].voteCount += 1 * voters[voterAddress].weight;
    emit voted("Vote cast!");
}

```

```

function declarePresident() public validPhase(Phase.Done) onlyChair returns
(uint8 winningProposal) {

    uint256 winningVoteCount = 0;
    for (uint8 prop = 0; prop < proposals.length; prop++)
        if (proposals[prop].voteCount > winningVoteCount) {
            winningVoteCount = proposals[prop].voteCount;
            winningProposal = prop;
        }
    assert(winningVoteCount>=1);
    emit VotingClosed(winningProposal);
}

```

```

function hashMe(uint8 key, bytes32 password) public onlyChair
validPhase(Phase.Init) {
    voterToken = keccak256(abi.encodePacked(key, password));
}

```

```

//password :
0x60298f78cc0b47170ba79c10aa3851d7648bd96f2f8e46a19dbc777c36fb0c00, key : 12
}

```