Electio - Technical Guide

Patrick H Morris

Advisor: Dr. Geoff Hamilton

Student ID: 14759021

May 19th, 2018

Contents

1 Abstract

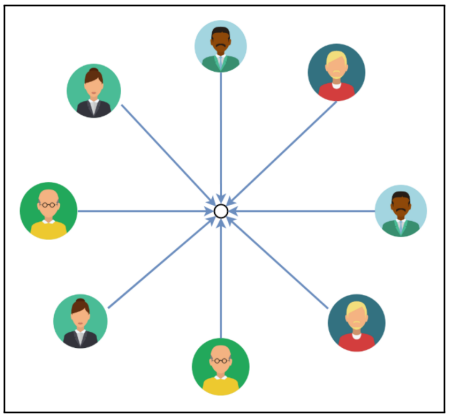
Electio is an e-voting decentralised-application (Dapp) which runs on top of the Ethereum blockchain platform.
Electio is intended to provide an alternative solution to state-level e-voting systems which advocate for a centralised tallying model. Tallying of elections in traditional e-voting systems utilise voting machines for voters to interface with an election and submit their votes. Usually, the data on the machine is either transferred physically or transmitted over the web to a central tallying server. The tallying server/machine is able to generate a result of this election.
The advantages to solutions like this is that they are inexpensive in comparison to paper-voting methods which requires human oversight to administer and tally. The negative to voting in a centralised model is that the central tallying system is a major security concern. An election is the core component of any democratic system and is intended to give people under these democracies the right to express their views and opinions. An election ought to provide every voter a degree of satisfaction that their vote was submitted and counted with the level of security that only the voter themselves knows for sure what their vote was.
The centralised e-voting model, in my opinion, does not and can not achieve this level of security. Ultimately, this model is asking every voter to trust that the tallying system will get it right and that it is a "good-actor" in the election process. The problem is that the tallying system is a black-box for voters, they do not have any way of achieving assurance for their vote or how the votes were counted. Should the central tallying system become compromised, it would be a far greater threat as it would enable governments or other organisations to undermine their own or other countries elections.

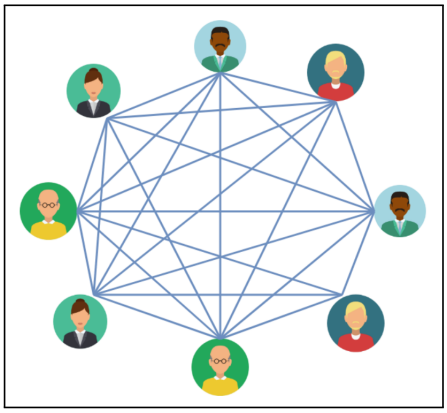
> It's not the people who vote that count, it's the people who count the votes.
>
> *J.Stalin*

2 General Description

2.1 Motivation

My ambitions for this project was to develop an application that was based in the blockchain space. In the last two years there has been an astronomic increase in interest into cryptocurrencies and the blockchain technology that underpins it. Ethereum is loosely defined as a cryptocurrency in that it has a fungible token that expresses a tradeable value but the underpinning design of its blockchain is what interested me the most.
Ethereum's blockchain is described as turing-complete virtual computing environment or more simply a global computer. Blockchain technology is a solution to achieving consensus across a group of participants in which a majority are deemed *good actors*. This consensus in the initial concept, Bitcoin, was agreeing on a state of a ledger which keeps track of all transactions on the network. Ethereum differs in that the ledger is represented by this virtual computer and the consensus is agreeing on the state of execution that computer has done. This immediately became an area I wanted to understand more and led me to designing and developing Electio using the Ethereum platform.





Both figures above simulate systems of centralisation and decentralisation. The traditional election model on the left is dependent on a single point. Should this single point fail then the system as a whole fails.
Alternatively, on the right, the decentralized model distributes system dependency across all actors involved withing the system. Everyone is responsible for an equal share of that system and securing it.

2.2 Project Context

The Ethereum platform enables developers to build decentralised applications (dapps) in the form of code called smart-contracts. Alluding to the abstract, this presented a viable solution to developing a solution to the centralised tallying model in e-voting systems. Electio enables anyone who has an Ethereum account address to dynamically generate and administer elections as well as participating in the full election pipeline. Electio emphasises the importance of achieving consensus as to the result of elections by enforcing all users, those who have participated and those who have not, to count the votes themselves using the electio application.

2.3 Research

- **A Smart Contract for Boardroom Voting with Maximum Voter Privacy**
  - https://fc17.ifca.ai/preproceedings/paper_80.pdf
  The above is a research project called the Open Vote Network which I found early into the project. It employed a self-tallying protocol which used ZK-snarks which are cryptographic methods to control the provacy of the vote. This gave me assurance as to the achievability of what I had intended for the project.

- **Public-key cryptosystems based on composite degree residuosity classes**
  - https://dl.acm.org/citation.cfm?id=1756146
  The paillier cryptosystem is a hommomorphic cryptosystem which has additive properties for encrypted numbers. Essentially, this enable me to generate a solution for plurality systems where the tally can be found without decryption of any individual vote.

- **CryptoKitties Ethereum Dapp**
  - https://www.cryptokitties.co/

Cryptokitties was released early on in the development of this application and provided me an understand in how a fully-fledged decentralised application would work in reality. Cryptokitties is a virtual collectibles marketplace where people can buy and breed virtual cats. It has a very refreshing user interface using React and Redux which is a frontend framework that enables components of the frontend to be more easily managed.

- **Ethereum Yellow Paper**
  - http://gavwood.com/paper.pdf
  The Ethereum yellow paper is a formal definition of the Ethereum protocol which defines specifically the inner workings of the Ethereum state machine. It was a more defined version of the initial description of Ethereum by Vitalik Buterin in his **White Paper**
  https://github.com/ethereum/wiki/wiki/White-Paper

- **Ad-hoc research** A large amount of informal reasearch was carried out throughout this project. As the area is in continual rapid development, the degree of documentation to consume would be a project unto itself. I spend most days reading the major ethereum-centric articles and an extension into react-redux on reddit, /r/ethereum and /r/ethdev as well on the gitter channels surrounding the open-source development teams.

2.4 User Roles

The Electio system defines all users of the application into four categories.

2.4.1 Administrators

Administrators are the owners of elections. It is on their own authority to generate the election and for them to regulate who the participants are. In motion voting, they are also responsible for generating the motion and the choices that are to be voted on. In all elections they are to also provide keys which allows users to encrypt their votes and also for them to hide their votes.

2.4.2 Voters

Voters are the essential users in an election. Electio enables Voters to register for an election as well as to vote in accordance to the electoral system they are using.

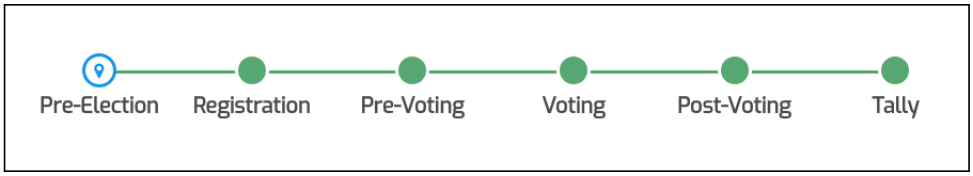
2.4.3 Candidates

Candidates in Electio can also easily register for an election. They provide their election details and are then identified as applied. In order for an applied candidate to become a valid candidate, they must canvas for nominations from valid voters. Voters who wish to nominate a candidate can do so once. When the candidate has filled the nomination limit they are validated. Voters can vote for them in the subsequent voting stage

2.4.4 Observers

Observers are people who are not directly involved in an election. They can be anyone who accesses Electio. Observers can apply to elections but even if its the case that they miss out on an election, they can still view the tally. The tally is globally observable for all people, this is important as it increases the validity of an election when everyone can view the result

2.5 Election Stages



The election stages are spread across 6 scenario's which all elections traverse. These stages are programmatically identified by the client-side react-redux application. This interprets the state of the election and infers what stage the election is at. On the basis of this stage, it then identifies what view is suitable for that user.

2.5.1 Pre-Election

The pre-election stage is the initial state of an election post creation. The stage is intended to enable the Administrator of the election to create a time-window for the registration period. The smart-contract restricts this time declaration by enforcing the period to be picked in the future.

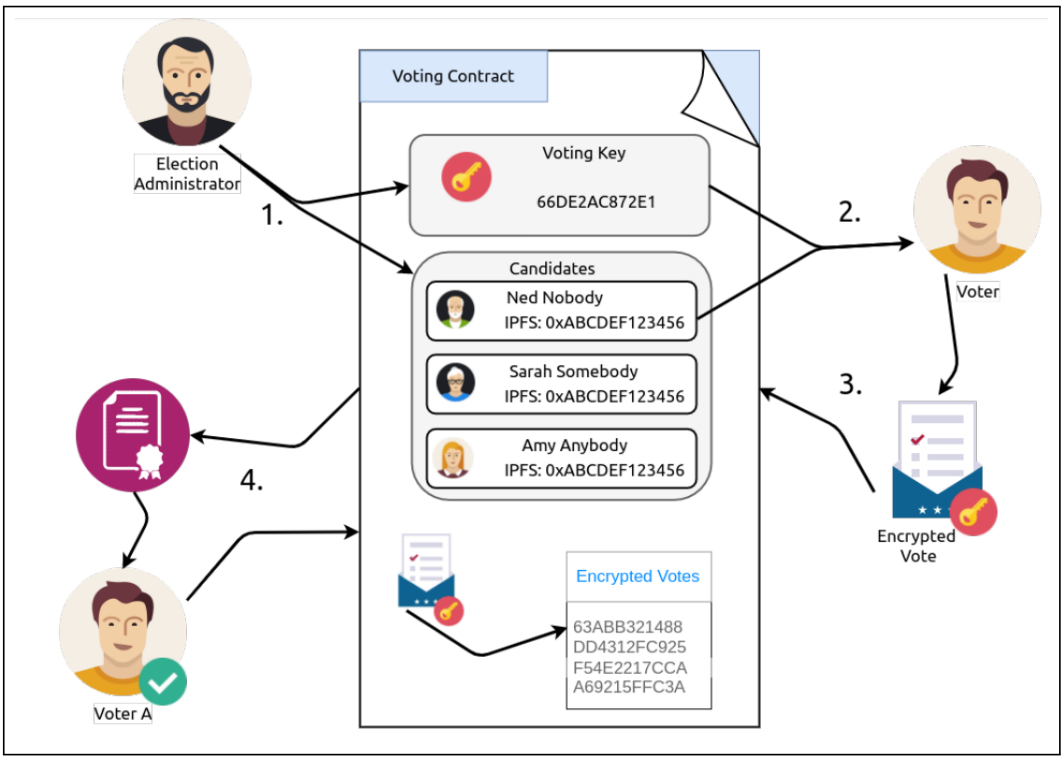
2.5.2 Registration

Once the registration window is open, the Observers of the election can select to register as either Voters or Candidates. The Voters in this case are presented an Election ID which is used to identify yourself to the Administrator externally to this system. When applied, the Administrator is represented a table of Election ID's and addresses. The Administrator can choose to validate or invalidate the addresses mapping to their respective ID's if they were represented to them.This serves as a method to obfuscating the Voter's identity to their vote later on in the voting stage Observers who apply as Candidates are register with their name and are only valid subsequent to nomination to Voters who have been already validated. Where the electoral system is Motion, no candidates can apply as the voting options are manually ascribed to the election by the Administrator.

2.5.3 Pre-Voting

The pre-voting stage is similar in nature to the pre-election stage with the exception that the election keys are generated here. The election keys are a public-private keypair with which the public is published on the blockchain with the time and dates for the voting period. After this point, only the valid participants are allowed engage in the voting period.

2.5.4 Voting



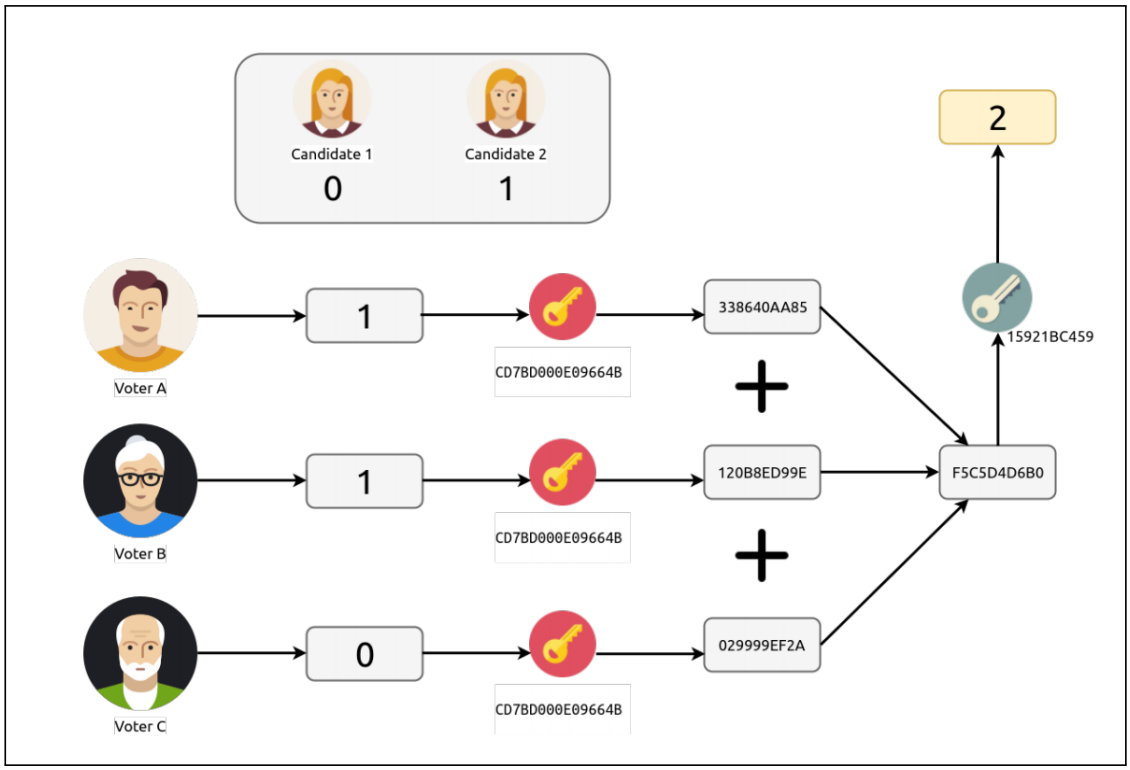
The voting stage represents valid Voters and Canididates the means in order to make their vote, specific to the electoral system that they are using. The Voter selects their vote and the application encrypts with the public-key taken from the blockchain. This encrypted vote can is submitted in a string format onto the blockchain.

2.5.5 Post-Voting

Once the voting stage is over, the election awaits for the administrator to publish the election private keys. The importance of doing it after is that were it available, people would be able to see the generate the election result as the votes come in. This is unfair as the later into the election you cast your vote, the greater the influence that vote may have in that result. Once published, the tallying process can immediately start.

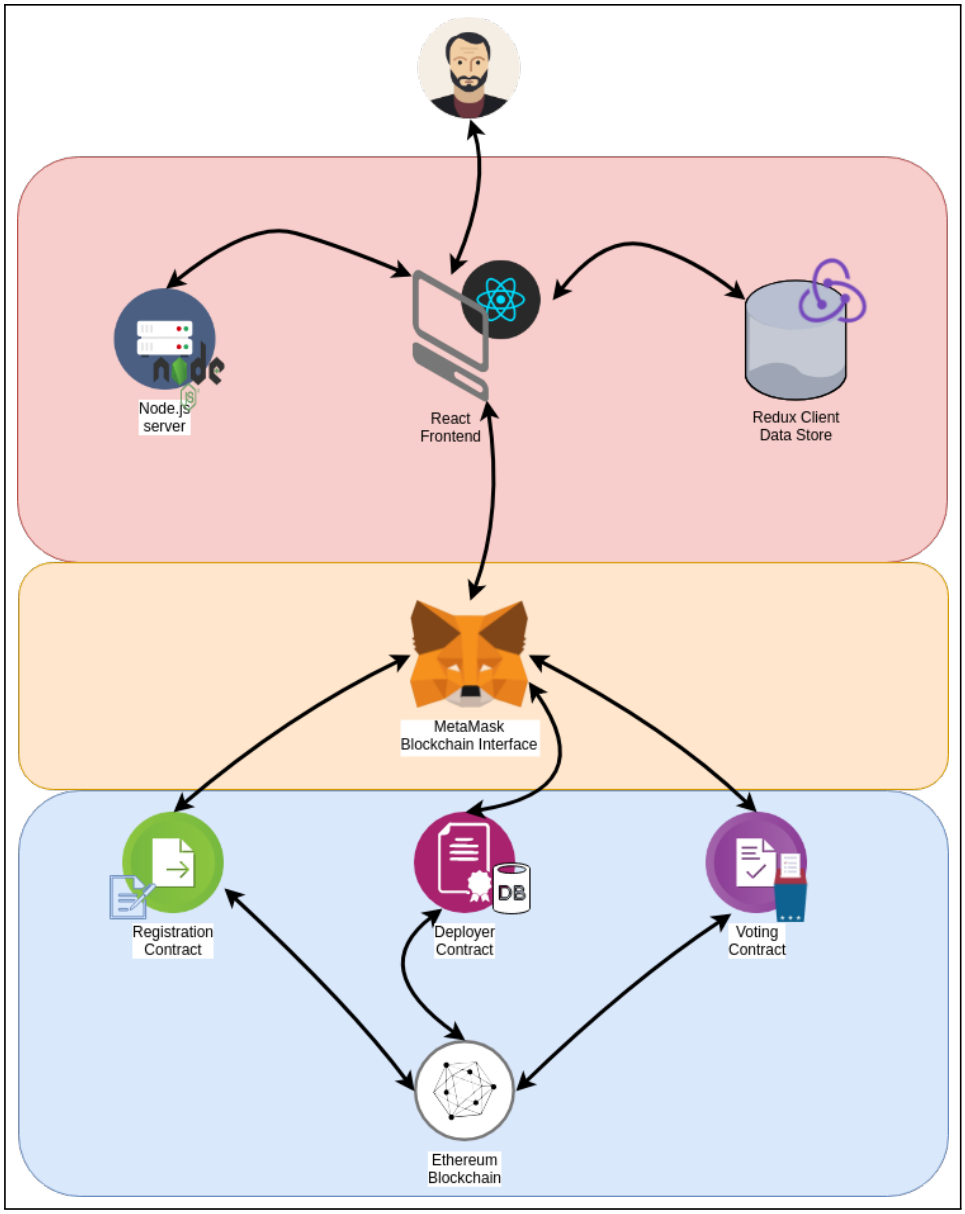
2.5.6 Tallying

The tallying process is done automatically by the application. All votes are taken from the blockchain and in the case of plurality and motion election systems, the votes are hommomorphically added together to produce a tally numeric. This tally numeric is specially defined to be able interpret the final sum of all Candidates in the election. This tally numeric is decrypted and analysed to produce the election result which is then displayed to the user.



3 Design

3.1 System Architecture



The architecture design in a decentralised application is removed from standard webapps which uses a central database and server to administer authentication and data retrieval and storage actions.

3.1.1 Smart Contracts

The smart-contracts in electio are written in Solidity which is a statically-typed programming language that compiles to ethereum virtual machine bytecode. When contracts are deployed onto the ethereum blockchain, it is the bytecode data that is deployed.
When deploying the smartcontracts, the deployer contract contains the logic to generate an election instance. This instance is defined by both the registration contract and the voting contract. Elections when created take the ethereum address of the voting contract as the election address which signifies where the election is located.
The Deployer contract acts as a store for all elections and holds an array of election objects which define an elections name, address and system.

3.1.2 MetaMask

MetaMask is a chrome plugin that enables access to an ethereum account through the web browser. The is a web3.js provider which enables interfacing with the ethereum blockchain through the web browser by calling smart-contract functions through javascript. Metamask serves as an authentication system with which users are identified only as an ethereum address. This allows for seamless user integration into the frontend UI by unlocking their ethereum account.

3.1.3 Node.js server

The node.js server has two main functions, the first is to provide the user with the frontend UI and any files which the system requires in the users local storage. The second is to provide an interface to the cryptographic system which allows the user to encryption, decryption, tallying, and election key-generation. This functions are integral.

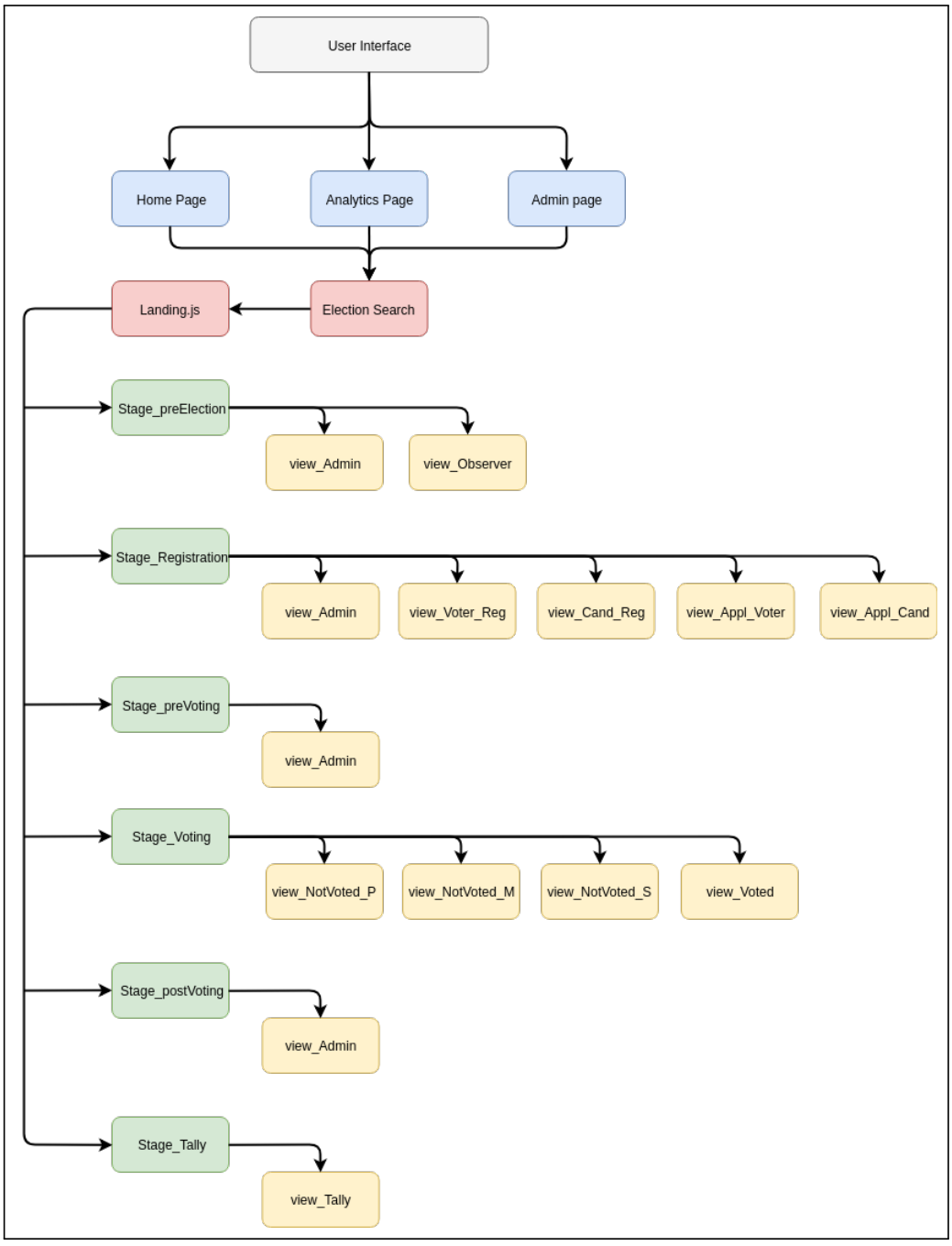
3.1.4 Redux Store

Redux is often integrated into react applications as it provides a method to seamlessly generate user storage on the client side of the application. The redux store is used by the react frontend to store the state of the election data taken from the smart-contracts. Using redux allows the data to be examined efficiently.

3.1.5 React

React is a javascript web framework which builds user interfaces using react components. These components are utilised to create single-page applications. A single-page application performs context switching which instead of rerouting the user to another document, the data presented to them on the user-interface is manipulated or exchanged. The redux data store enables for efficient retrieval and presentation of the data to the user.
The majority of the electio functionality exists here and is to be found in the /src/src directory. The application root is to be found in App.js in this directory

3.2 High-Level Design



The above displays the main components of the react user-interface used to derive the stages of the elections and the specific views which are to be displayed on context.

3.2.1 Main Routes

The main routes in the application, highlighted in blue are the Home page, Analytics page and Admin page. The home page is used to welcome the user and explain how electio works. The Admin page is used by Administrators to create elections. The Analytics page lists all elections that have been created by electio.

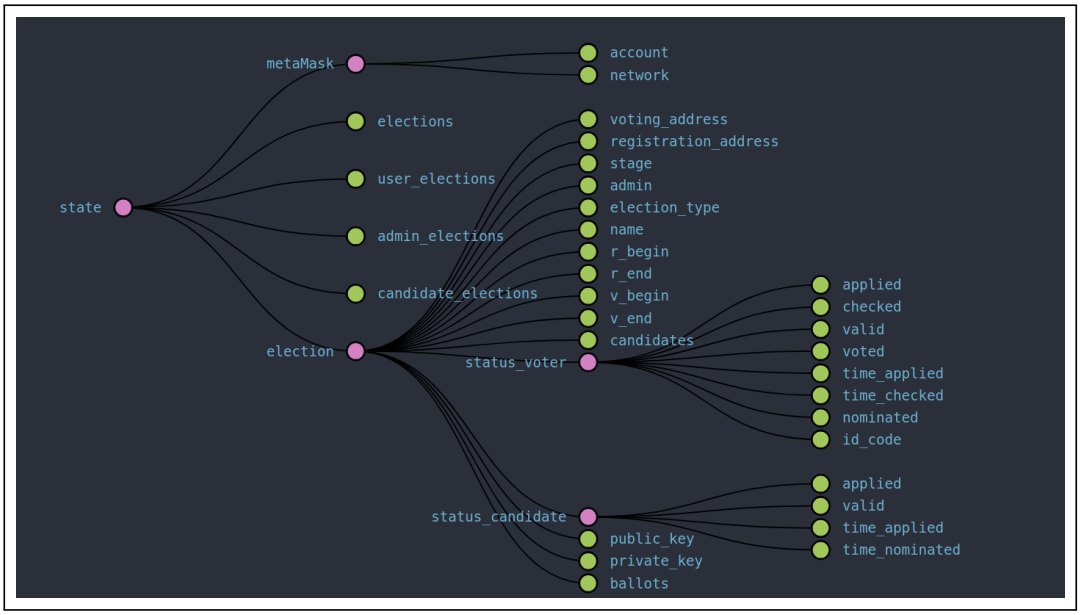
3.2.2 Election Search

At any point, a search bar is available to search for an election. This involves a user, inputting an election address which the react application interprets to load and view that specific election. The landing file determines what stage the election is at and renders the specific stage view.

3.2.3 Stage Views

The main portion of stage views are listed above. At this point, the election data has been read off of the blockchain and the stage is deteremined. This stage identifies the users relationship with the election at that stage. The react application then renders a view specific to that relationship.

3.3 Database Design



The above is the data object design for the redux client database. When the application is loaded, the frontend application will take a read of the blockchain and change the votes. This data store is representative of the current state of the election and enables the redux portion of the application, represent a correct view of the election.

4 Implementation

4.1 Election Creation

```
contract Deployer {

    mapping (address => address[]) public userElections;
    mapping (address => Election) public elections;
    address[] public electionList;

    struct Election {
        address _owner;
        string name;
        address contractRegistration;
        address contractVoting;
        uint createdAt;
        uint system;
    }
```

Above is the contracts/Deployer.sol file and the definition of an election object. Each election is defined with an owner, name, registration contract, voting contract, timestamp and electoral system.
The two mappings give the utility to the system to finding elections which are specific to a user and elections with a certain address trivial. The electionList array holds all election addresses for the elections.

```
function newElection(
    string name,
    uint _nomLimit,
    uint electionType,
    uint _numSeats
)
    public
{
    Registration r = new Registration(name, msg.sender, _nomLimit);
    Voting v = new Voting(name, r, msg.sender, electionType, _numSeats);

    Election memory election = Election(
        msg.sender,
        name,
        r,
        v,
        block.timestamp,
        electionType
        );

    elections[v] = election;
    electionList.push(v);
    userElections[msg.sender].push(v);
}
```

To create a new election a transaction is sent to the blockchain to generate a new election instance. The two contracts are declared and the data is recorded in the storage objects.

4.2 Loading an Election

Loading an election in the application for the user is as trivial as pasting an election address into the searchbar. What goes on in the behind the scenes is somewhat more complicated.
This logic is primarily controlled by the setElection() function in App.js:84, which takes an entire read of the specific election attributing to the election input. When the data is filled, the setCurrentElectionStage() is called. One of the attributes which was not determined is the election stage.

```
setCurrentElectionStage() {

  const election = this.props.election
  var stage;
  var now = helper.now()

  if(parseInt(election.r_begin, 10) === 0 || parseInt(election.r_begin, 10) > now) {
    stage = 1
  } else if(parseInt(election.r_begin, 10) <= now && parseInt(election.r_end, 10) > now) {
    stage = 2
  } else if(parseInt(election.r_end, 10) <= now && (parseInt(election.v_begin, 10) === 0 || parseInt(election.v_begin, 10) > now)) {
    stage = 3
  } else if(parseInt(election.v_begin, 10) <= now && parseInt(election.v_end, 10) > now) {
    stage = 4
  } else if(parseInt(election.v_end, 10) <= now && election.private_key.lambda === "") {
    stage = 5
  } else if(parseInt(election.v_end, 10) <= now && election.private_key.lambda !== "" && election.private_key.denom !== "") {
    stage = 6
  } else {
    stage = null
  }

  this.props.handleElectionStage(stage)
}
```

The steps of execution through this function examine the registration and voting time windows. Using the current time as reference, an election stage number is set.

```
class Landing extends Component {

  render() {
    switch(this.props.election.stage) {
      case 0:
        return (<HomePage/>)
      case 1:
        return (<div className="column is-12"><StagePreElection {...this.props}/></div>)
      case 2:
        return (<div className="column is-12"><StageRegistration {...this.props}/></div>)
      case 3:
        return (<div className="column is-12"><StagePreVoting {...this.props}/></div>)
      case 4:
        return (<div className="column is-12"><StageVoting {...this.props}/></div>)
      case 5:
        return (<div className="column is-12"><StagePostVoting {...this.props}/></div>)
      case 6:
        return (<div className="column is-12"><StageTally {...this.props}/></div>)
      default:
        return (<HomePage/>)
    }
  }
}
```

The landing component then interprets the election stage number and renders the correct view.

4.3 Starting an Election

```
async startRegistration() {

    var func = async (receipt, num) => {
        this.props.renderOnTransaction(1, receipt, num)
    }

    this.props.registration.methods.start(
        this.state.unix_start,
        this.state.unix_end,
        this.props.election.voting_address
    ).send({from: this.props.metaMask.account}
    ).on('transactionHash', function(hash){
        console.log("Hash")
        console.log(hash)
    })
    .on('receipt', function(receipt){
        console.log("Receipt")
        console.log(receipt)
    })
    .on('confirmation', async function(confirmationNumber, receipt){
        func(receipt, confirmationNumber)
    })

    this.props.history.push('/')
}
```

```
function start(uint regStart, uint regEnd, address vc) public {

    if(end != 0) {
        revert();
    }

    begin = regStart;
    end = regEnd;
    votingContract = vc;
}
```

Above is the javscript web3.js function call to the blockchain that is used by the administrator to start the election. By executing this, Metamask will provide a popup asking to confirm a transaction to the blockchain. When it is confirmed, the solidity start function will be executed.
revert() in solidity smart-contracts are used to prevent waste of execution. Users are heavily penalised if they perform a function of the contract incorrectly so everyone is economically incentivised to be a /emphgood actor

4.4 Registering for an Election

```
mapping (address => Voter) public voters;
mapping (address => Candidate) public candidates;

address[] public applicantVoters;
address[] public applicantCandidates;
address[] public validVoters;
address[] public inValidVoters;
address[] public validCandidates;
string[] public motions;

struct Voter {
    address _voter;
    bool applied;
    bool checked;
    bool isValid;
    bool hasVoted;
    uint id_code;
    uint timeApplied;
    uint timeChecked;
    address nominated;
    uint ptr;
}

struct Candidate {
    address _candidate;
    string firstName;
    string lastName;
    address[] nominators;
    uint[] nominatedTimes;
    bool applied;
    uint timeApplied;
    bool isValid;
    uint ptr;
}
```

The registration contract defines a system to identify both voters and actions. Users who register are first defined as applicantVoters or applicantCandidates. When validated, they are put in their respective categories of valid or not

```
function registerAsVoter(uint code)
  public
  inTime
{
    if(!isAppliedVoter(msg.sender)) {

        Voter memory vtr = Voter(
            msg.sender,
            true,
            false,
            false,
            false,
            code,
            block.timestamp,
            0,
            address(0),
            0);

        voters[msg.sender] = vtr;
        applicantVoters.push(msg.sender);

    } else {
        revert();
    }
}


function addMotion(string _motion) public inTime onlyOwner {
    motions.push(_motion);
}
```

```
function registerAsCandidate(
    uint code,
    string fName,
    string lName
)
    public
    inTime
{
    if(!isAppliedCandidate(msg.sender)) {

        if(!isAppliedVoter(msg.sender)){
            registerAsVoter(code);
        }

        address[] memory arr1;
        uint[] memory arr2;

        Candidate memory cdt = Candidate(
            msg.sender,
            fName,
            lName,
            arr1,
            arr2,
            true,
            block.timestamp,
            false,
            0);

        candidates[msg.sender] = cdt;
        applicantCandidates.push(msg.sender);

    } else {
        revert();
    }
}
```

The process of registration is as above. To register as a voter the user submits a random code number which is used to identify themselves on the blockchain. Solidity comes with function modifiers with the modifier inTime being used in this instance. This simply reverts the execution if the function is called outside of the registration window.

Where the voter has already applied, the process is reverted(). If a user is an honest applicant their details will be added to the blockchain.

Registration of the candidate is much similar with the exception of declaring a first and last name. The candidate instantiates a new candidate object and adds it to the applicantCandidates list.

4.5 Voting in an Election

```
function submitVote(string vote)
    public
    onlyValidVoter
    hasNotVoted
{
    Vote memory vt = Vote(vote, block.timestamp);
    ballotBox.push(vt);
    registration.changeVoterStatus(msg.sender);
}
```

The smart-contract logic to submit a vote is relatively starightforward. The function accepts a string value and provided that the sender address is valid and has not voted prior, then the string is published.
The logic to generate the correct string is more complicated and involves calculating the correct voting numeric to generate in order that I can apply the hommomorphic additive property to count them.

```
function calculateVoteNumeric(candidateIndex, numVoters) {

  if(candidateIndex == 0) {
    return 1
  }

  var x = 1;
  while(x <= numVoters) {
    x = x << 1;
  }

  return x ** candidateIndex
}
```

This function is used to determine the correct voting numeric for a given selection. A voter would select a candidate and generate this numeric on the basis of the candidates index in the validCandidates array object in the Registrtion contract. The numVoters is the limit to the highest amount of vote any candidate can receive in the election.
By generating a binary power greater than the number of votes possible in the

election, we can take a substring of the summmed votes to represent a candidate. If there were 3 candidates and 10 voters in an election, their numerics would be:

- Candidate 0 - 1
- Candidate 1 - 10000
- Candidate 2 - 100000000

When these are summed together they would produce a binary number:

- - 100010001

which would be three votes for each candidate.
Both the plurality and motion voting systems work by this method

4.6 Tallying an Election

```
async getElectionMotionResult() {
  var votes = []

  for(var i = 0; i < this.props.election.ballots.length; i++) {
    votes.push(this.props.election.ballots[i].vote)
  }

  var encryptedTally = await helper.tally(votes, this.props.election.public_key)
  var decryptedTally = await helper.decrypt(encryptedTally, this.props.election.public_key, this.props.election.private_key)
  var numMotions = (await this.props.registration.methods.getCounts().call())[5]
  var winnerIndexes = await helper.calculateWinner(decryptedTally.toString(2), this.props.election.ballots.length, numMotions)
```

Generating the tally can be done by executing the following steps:

- Read all encrypted votes off the blockchain
- Tally all encrypted votes to a single tally numeric
- Decrypt this tally numeric
- From this decrypted numeric, generate the number of votes each candidate received

The exception to this comes in the manner of STV voting. The calculation of the election result in this election meant that using the paillier method to add encryptions together would not work. STV votes require multiple examinations through the tallying process so the votes are all decrypted individually to determine the result.

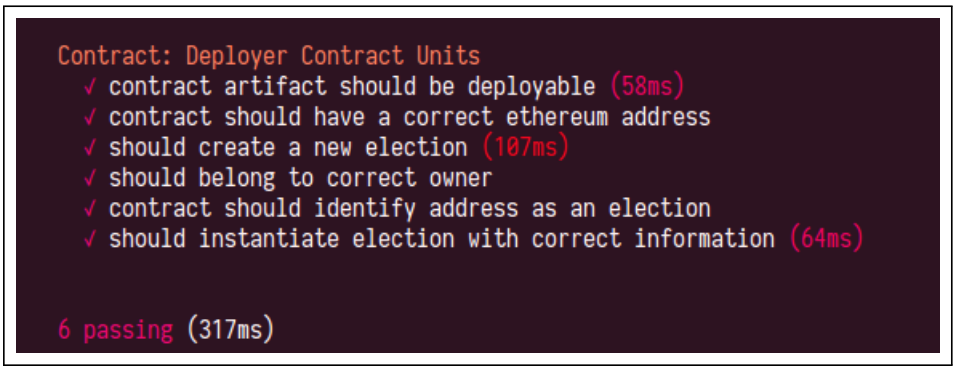
5 Testing

Testing electio proved to be initially quite difficult as there are a number of obstacles in the to create the correct environment. I used a derived version of Mocha.js and Chai.js to automate my tests which proved successful when environment issues were sorted out.
A large number of scenario's that occur in the election timeline are time-dependent, which meant I had to generate other scripting functionality in order to account for timing issues.
Another was automation of multiple users actions. In order to provided some satisfactory results I generated scripts which would automate the election processes programmatically. This saved a great amount of time and stress.

5.1 Unit Testing

Unit testing covered the smart-contract functionality and the client side helper functions which are used to interface with the server-side functionality and the generation of election data such as the voting numeric and the tally result. Some tests require the waiting until a time-window begins and ends which is normal functionality for the system.

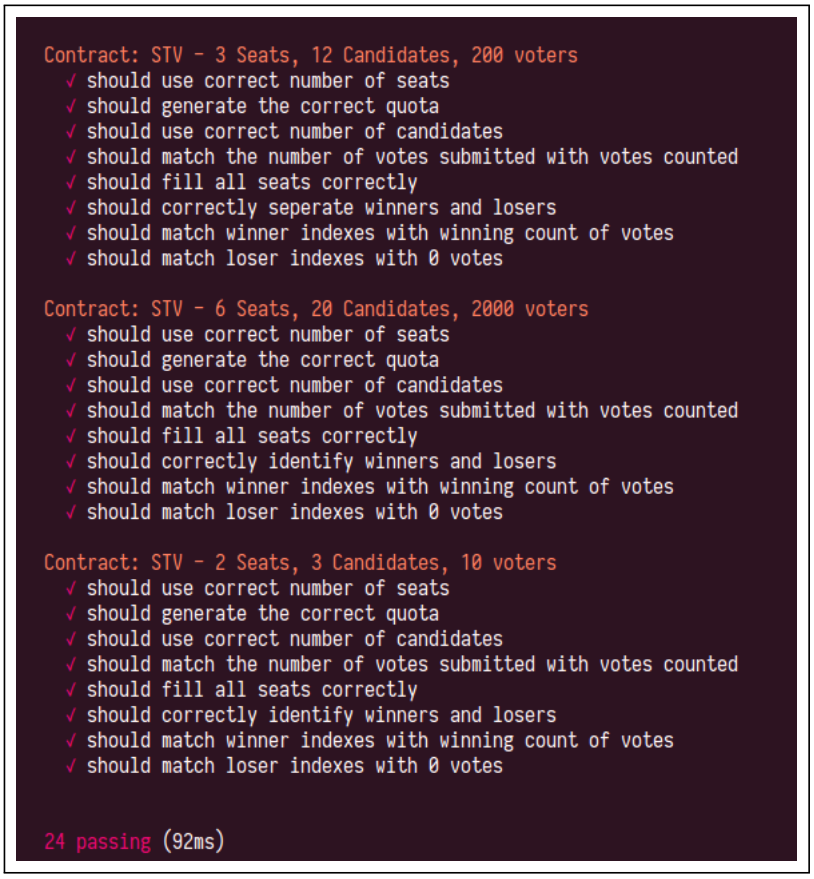
5.1.1 Unit-Deployer



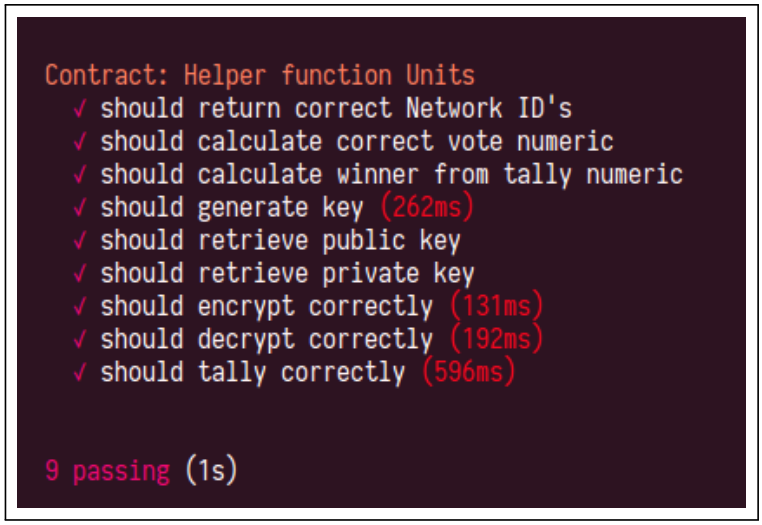
5.1.2 Unit-Registration

5.1.3 Unit-Voting

5.1.4 Unit-STV



5.1.5 Unit-helper



5.2 End-to-End Testing

End-to-End testing tests each electoral system in full with 40 participants in each case. These are generated to simulate what real elections would be like.

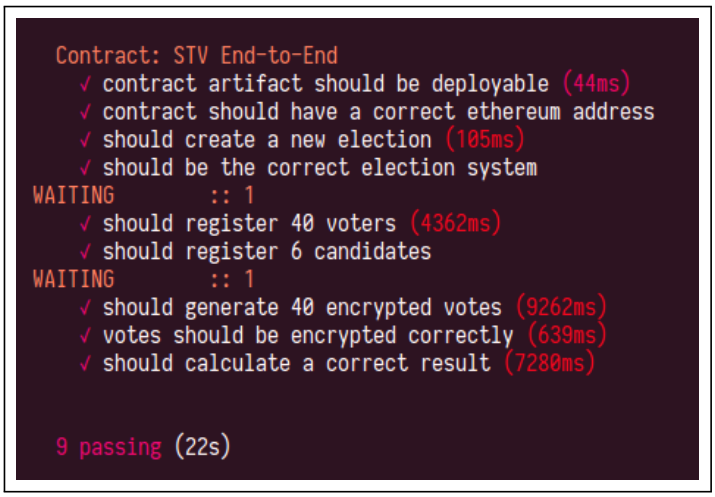
5.2.1 Plurality

```
  Contract: Plurality End-to-End
    ✓ contract artifact should be deployable (49ms)
    ✓ contract should have a correct ethereum address
    ✓ should create a new election (69ms)
    ✓ should be the correct election system
WAITING        :: 1
    ✓ should register 40 voters (4301ms)
    ✓ should register 2 candidates
WAITING        :: 1
    ✓ should generate 40 encrypted votes (9247ms)
    ✓ votes should be encrypted correctly (655ms)
WAITING        :: 0.20000004768371582
    ✓ should calculate the tally as encrypted string (886ms)
    ✓ should decrypt string to correct value (181ms)
    ✓ should calculate a result from decrypted tally


  11 passing (15s)
```

5.2.2 Motion

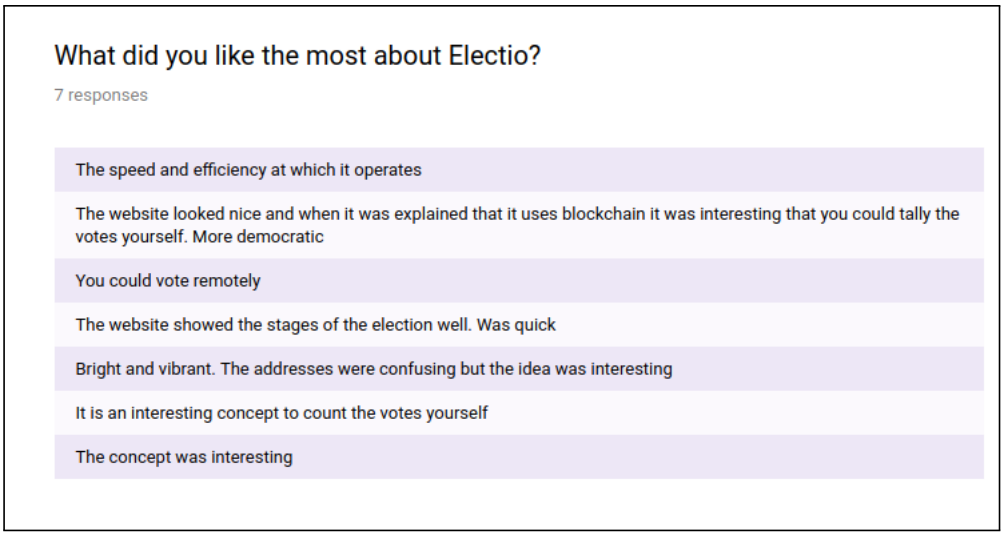
```
  Contract: Motion End-to-End
    ✓ contract artifact should be deployable (44ms)
    ✓ contract should have a correct ethereum address
    ✓ should create a new election (117ms)
    ✓ should be the correct election system
WAITING        :: 1
    ✓ should register 40 voters (3795ms)
    ✓ should have added 2 motions
WAITING        :: 1
    ✓ should generate 40 encrypted votes (9327ms)
    ✓ votes should be encrypted correctly (659ms)
WAITING        :: 0.20000004768371582
    ✓ should calculate the tally as encrypted string (925ms)
    ✓ should decrypt string to correct value (184ms)
    ✓ should calculate a result from decrypted tally


  11 passing (15s)
```

5.2.3 Single Transferable Vote

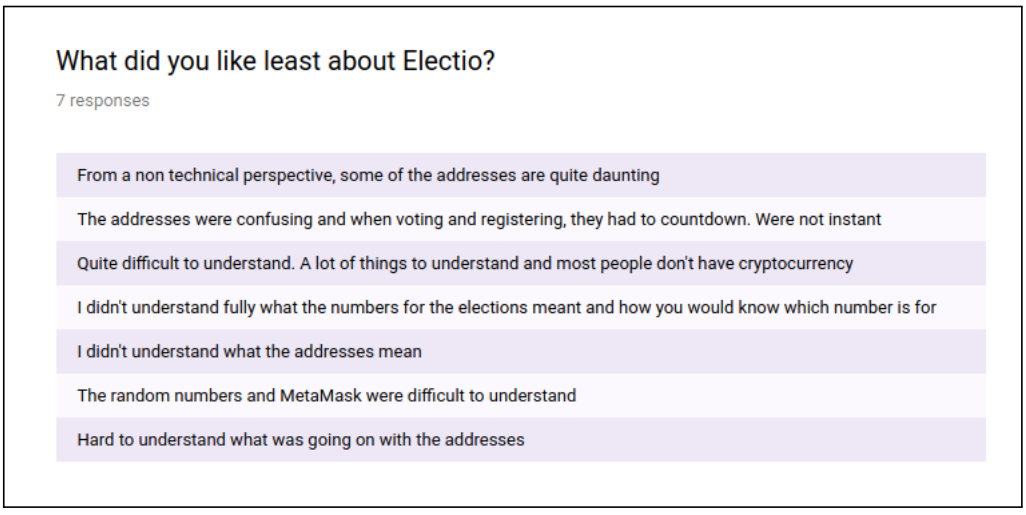


5.3 Usability Testing

When I had achieved a desirable level of functionality for the project I began usability testing. This involved me sitting down with potential users to demonstrate how the project work and discuss how they would use it. After demonstration a google form was sent to help them express their opinions. It resulted as follows.

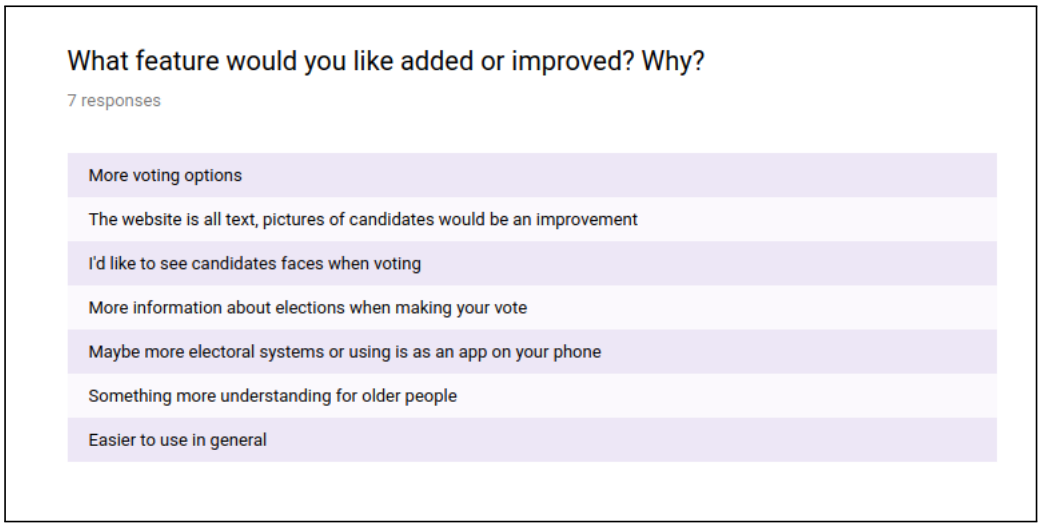


People in general found that the concept of tallying the votes yourself to be interesting and overall more democratic

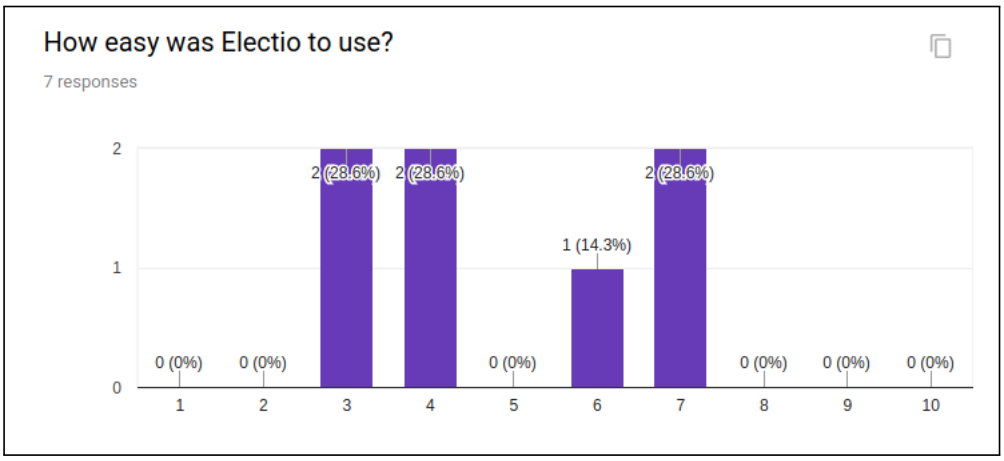


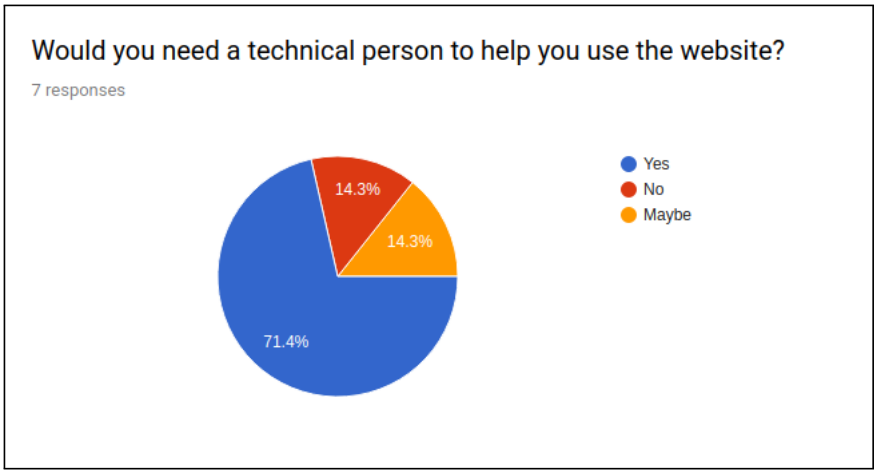
People mostly found it difficult in understanding how the election is performed on the blockchain. I would say in response to this account that there were too many

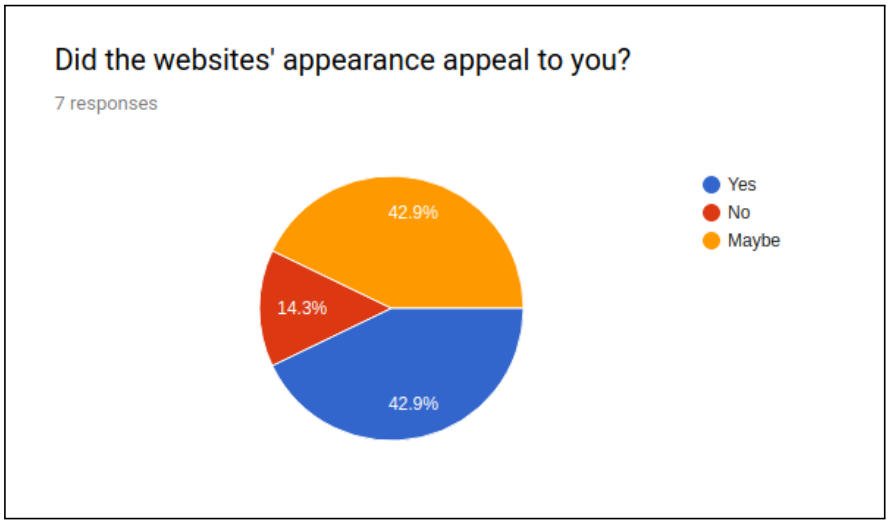
instances where I used the ethereum address for an election or an account in the frontend.



This question had a varied response. One feature lacking was more candidate information. I had intended to provide pictures through IPFS which is a decentralised storage system but as the project progressed it was too time-consuming to implement. Using the application on your phone would most likely increase user participation. The ease-of-use is for a project of this nature one of the most difficult hurdles against it.




Would you need a technical person to help you use the website?
7 responses
14.3%
14.3%
71.4%
Yes
No
Maybe


Did the websites' appearance appeal to you?
7 responses
42.9%
14.3%
42.9%
Yes
No
Maybe

6 Conclusion

6.1 Changing Requirements

Examining the project now in comparison to where I had written my functional specification, a lot has changed. The main issue was a considerable under-estimation as to the size of the codebase required to create this project. From what I had intended, a lot of the previous design choices had to be removed.

- InterPlanetary FileSystem
  IPFS is a decentralised file storage system that is akin to a BitTorrent file distribuition system. By this method, I would have enabled candidates to register and upload election information of their choosing. This would have made it possible for voters to be able to view candidates policies on a particular election.

- Peer-to-Peer User registration
  The registration process is somewhat lacklustre where the voter is required to submit an election ID to the administrator. What I had intended was to use Whisper which is a peer-to-peer messaging system which is built into the Ethereum platform. At the current moment in time, the Whisper implementation is only a prototyped version and would not be suitable had I enough time to implement it.
  Were it possible, I would have created a system where the applicant voter would publish their ID on the blockchain and send thier personal identification details to the administrators server. The server could then utilise a method of cross-referencing against a database of valid users for the election and validate the users address in an automated way.

- Removal of Spoiled Votes
  One functional requirement which was not developed was the removal of votes. In smart-contracts deletion of array values is extremely expensive and to implement would have caused the deployment costs of such contracts to

go over the network gas-limit. It also raises a security concern to trust the administrator of the election to not remove votes which are valid.

6.2 Future Work

There are many opportunities for the project to be further developed:

- Currently, the Ethereum environment is in rapid development with new tools and languages being developed. As the project stands, some of the packages required to develop, build and deploy the system are only possible using deprecated versions. Refactoring the project to a level which it is using the most up-to-date functionality will prevent breaking changes occurring.
- The cost of initial deployment of the election contracts is very expensive and as such needs to be refactored. One method to do this is to use a library contract which can abstract a lot of the costly functions into a standalone contract. The high cost of deployment of the contracts meant it difficult to deploy the system onto a test network. Deployment costs for the system stand at around 6.5 million with the Ropesten test network having a gas-limit of 4.7 million gas.
- Another method to mitigate against high costs is to use a different smart-contract language other than solidity. Solidity is a high-level language that was designed with amenability to web developers in mind. LLL (Low-level, Lisp-like, Language), is as it says a low level language that will compile to evm bytecode. Using a lower level language in an area where your mitigated against costly execution is obviously the smarter choice if implemented correctly. Higher levels of optimisation can be achieved using LLL and would could result in a more efficient solution to the deletion problem mentioned in the previous section.

6.3 Final Reflections

The intention of this project for me was to develop an application that involved the Ethereum blockchain platform. It is an area which I have a large interest in and I believe will have a strong future in the sphere of software development. Fundamentally, this project enabled me to achieve a large amount of experience in the area, I was also able to explore other area's such as UI frameworks and cryptographic protocols and hopefully I will use these again in the future.
On reflection, the project was a huge positive regardless of the levels of stress it caused and the missing of some portions of functionality. I would hope to work in an area using blockchain technology into my career.