



SMART CONTRACTS WITH BLOCKCHAIN

ABSTRACT

Blockchain is the technology that underpins Bitcoin, but there are many more applications for Blockchain, with potentially large business and societal impact. One of the recently proposed applications is smart contracts

Name: Vidhi Patel

Supervisor: George Theodorakopoulos

Moderator: Philipp Reinecke

CM3203 – One Semester Project (40 Credits)

Acknowledgements

I am very thankful for all the support I have received from the University members and the Equiniti. I am very thankful for their support and helpful guidance.

Firstly, I would like to thank my Supervisor Professor George Theodorakopoulos for giving me guidance and advice throughout this project.

Furthermore, I would like to extend my gratitude to Equinity, especially the blockchain developer for providing me with support and guidance and all the necessary information to finish the project.

Lastly, I would like to thank my Mother and family who have encouraged to pursue this project and supported me to achieve my goal.

Abstract

Since Blockchain technology has been introduced as the technology sustaining Cryptocurrencies, new applications of the Blockchain technology have emerged into the market so much so that the vast majority of organisations are analysing diverse ways in which they can implement it. Companies are interested in its capabilities especially how it can be developed to improve their current systems by removing the need of any intermediary entity.

The dissertation is based on Blockchain. The objective of my research was to analyse the feasibility of Blockchain as an e-voting system. In particular, to produce a system that offers the user a proof of vote, as well as prevents fraudulent activities that have occurred in the current e-voting systems. The use case was provided by Equiniti.

Smart Contracts exploit the Blockchain Platform, conducting automated payments or transfer of assets when the contractual conditions are met in the contract. They are used to automatically enforce certain conditions in the transaction without the need for a third-party involvement. The implementation of the Blockchain I have used to develop the proof of concept system e-voting system is Hyperledger, which is an open source private blockchain provided by IBM.

The prototype I have developed will determine whether blockchain can be used to develop an online voting platform to address the issues in the current e-voting systems. The e-voting systems will give people traceability of vote because it is an immutable ledger which restricts data modification. Furthermore, this research studies the use of Blockchain of technology and its implementation in government voting systems.

The conclusion demonstrated that Blockchain has potential to be a credible voting system however there are some limitations that are yet of concerns and requires attention.

Contents

Acknowledgements.....	1
Abstract.....	2
Table of Figures.....	5
Introduction	6
Aims and Goals.....	6
Project importance	8
Audience and Potential Users.....	8
Background	9
Issues with the Current E-voting and I-voting systems.....	9
Blockchain voting systems	10
Requirements.....	12
Aim	12
Justification for the Essential requirements	12
Assumptions.....	14
Methods & Tools for Solution.....	15
Choosing the Blockchain.	15
Language choice for Chaincode.	15
Data storage.....	16
Frameworks & JavaScript Client Network development	16
API development.....	17
Web Interface	17
Approach.....	18
System Design & Architecture Overview	20
E-voting system.....	22
Design.....	23
Web Interface	23
Admin Interface	23
Implementation	28
Network Development.....	28
Implementation of the chaincode	29
Dependencies for the chaincode	29
Chaincode Interface.....	29
Node SDK	30
Express.js (NodeJS web application).....	32
GUI Implementation	33

Testing.....	35
My approach to Testing	35
Fabric Network Usability testing.....	35
Smart Contract Testing	36
NodeJS Application Functionality Testing.....	36
REST API Testing.....	38
Interface Testing	38
Evaluation	40
Functionality Fulfilment	40
Project Management & Technique	41
Technology Used in Development	42
GUI	42
Chaincode	42
NodeJS SDK	44
API	44
Testing.....	44
Future Work	45
Conclusion.....	47
Reflection on learning.....	48
Bibliography	50
Appendix	52

Table of Figures

Figure 1 Blockchain. A linked list with hash pointers.....	6
Figure 2 Tamper Evident log Result of trying to alter the data	6
Figure 3 Feature driven development practise (W.Ambler, 2018).....	18
Figure 4 Hyperledger Fabric Infrastructure (Strukhoff, 2016)	20
Figure 5 My System Architecture for Blockchain development.	21
Figure 6 Admin Index page - design of the index page.....	23
Figure 7 Design of the candidate registration page.....	24
Figure 8 Mock-up of the viewing all the candidate's information.....	25
Figure 9 Design of user voting page to submit their votes.	26
Figure 10 Slide out panel for the user page to check their voting history.....	27
Figure 11 Slide out panel allowing the user to check their latest voting submission.....	27
Figure 12 Error to demonstrated that invalid function was invoked.	30
Figure 13 Commands to deploy the network using Couch DB and CA.	30
Figure 14 Import form the Node modules to use the Hyperledger Fabric Client.....	31
Figure 15 gRCP to add the peer and order to the network to initialise the channel.....	31
Figure 16 Resolving the promises to ensure this data is returned	31
Figure 17 Implementation of the Request object to invoke and query the ledger.	32
Figure 18 Generation of the transaction ID used for invoke.	32
Figure 19 This checks the proposal transaction to make sure it is valid.....	32
Figure 20 Example GET request in the REST API.....	33
Figure 21 Example POST request in the REST API.....	33
Figure 22 Query in the node SDK which will query the ledger.	33
Figure 23 Example AJAX Call to establish a connection between the application and the REST API...	34
Figure 24 Generates the certificates and genesis block for the blockchain using the cryptogen.	35
Figure 25 Docker command used to access a chaincode container.	36
Figure 26 Displaying the submission of valid transaction.....	37
Figure 27 Example error generated for adding duplicate data.	37
Figure 28 Viewing the information recorded in the ledger.	37
Figure 29 Postman Testing screen of a GET request which generates has no results.....	38
Figure 30 Testing of the web application on the localhost:3000	39
Figure 31 Images needed by docker to generate the Fabric network.....	41

Introduction

Aims and Goals

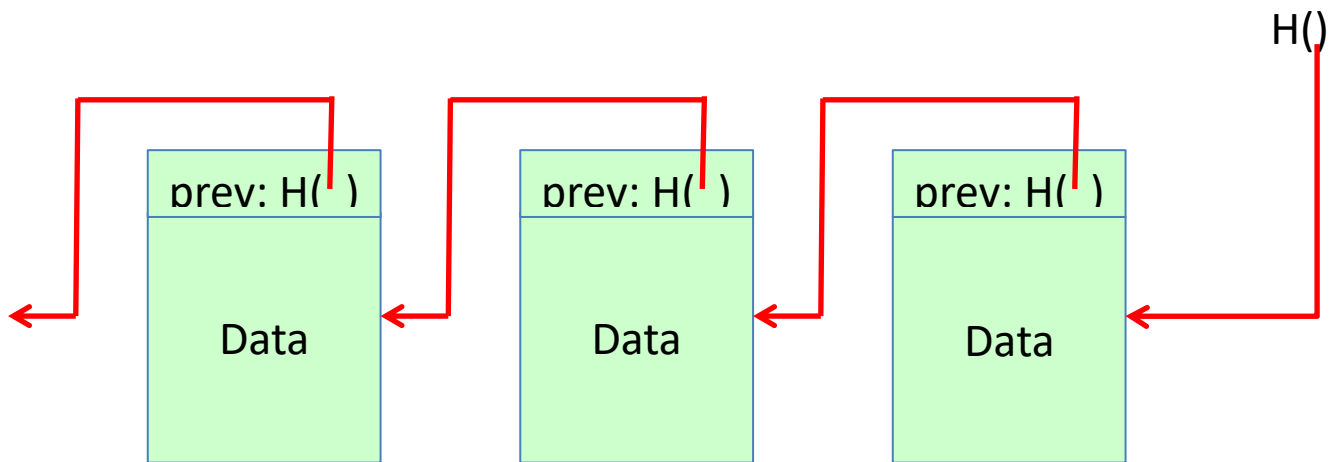


Figure 1 Blockchain. A linked list with hash pointers.

A Blockchain is a linked list which utilises hash pointers as shown in Figure 1 which will give the users and the government a platform where tampering of evidence is not possible as demonstrated. A hash pointer is a pointer to where some data is stored in memory with a cryptographic hash of the value (Walker, 2017-2018). Blockchain is a peer-to-peer system where the transactions are transparent to each individual involved in the network. Transactions written on the ledger are both traceable and anonymous. This happens because identities are not required to operate a system or execute the transaction.

This gives the users and the government traceability of transactions to ensure a secure system. This transaction will be recorded on the ledger therefore the peers in the network can all verify that the transactions are valid. The hash-pointer is developed to hold the history of the earlier blocks. This design aspect is what makes the modification of data held within the block challenging.

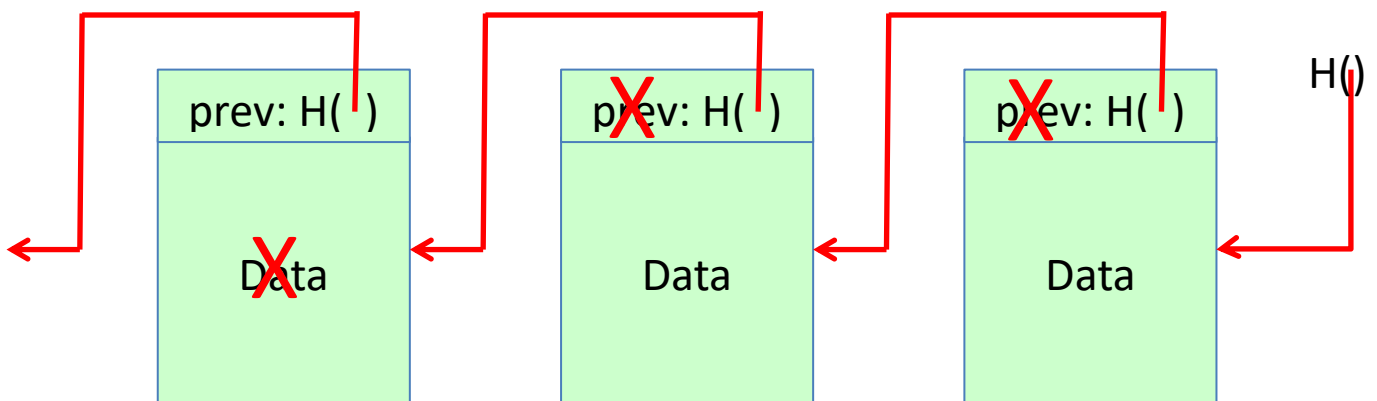


Figure 2 Tamper Evident log Result of trying to alter the data

Figure 2 shows a comparison of hash-pointers of the blocks can help to determine if any modifications have taken place. If data has been modified the hash-pointer value of the block containing the set of transactions will be incorrect, it would be necessary to modify all of subsequent blocks hashes to cover up the manipulation of a transaction. However, as long as the genesis block (first block) of the blockchain has not been manipulated the rest of the blockchain should remain secure (Walker, 2017-2018).

The most important implementation of Blockchain is smart contract. Smart contract is the business logic which decides how blockchain functions, it is automatically executed by the consensus protocol. Transactions are sent to all of the peers, where they are collected into a block. Randomly, a peer will be able to broadcast a block for inclusion in the block chain. The other peers will only accept the block if all the transactions recorded inside are valid (Walker, 2017-2018). This is the technology where conditions of how the e-voting application operates will be defined to ensure security and functionality of the system. Then based on the conditions and how they are fulfilled it will execute the payment, in this case allowing a user to cast a vote.

There are only a few e-voting platforms that have been developed or being used in the world. A country that has achieved this is Estonia. As blockchain is an emerging technology there is a possibility for the government to research and implement an e-voting system.

The purpose of this project was to discover the feasibility of Blockchain Technology, to provide a meaningful and convenient way for individuals to vote that provides the user with a proof of vote. This can be advantageous for the government as they are not involved in the intermediary entities by having them calibrating the e-voting systems.

To accomplish the task for this project, I will be developing a Blockchain application that can be used by individuals to vote for a candidate of their choice. This information will be recorded by a database which will behave as a state database. The Ledger will be the underlying platform for the software, alongside that I will create a network endpoint which will allow me to execute transactions. To provide the users with all of the necessary logic I will have REST API so that the underlying infrastructure can be used in an application.

Moreover, with the prototype, my goal is to implement all the necessary functionalities that allows the user to cast their vote and their votes to be correctly counted. This can enable the government to have an immutable record of the votes with a smart contract which allows for a decentralised voting system.

Project importance

Several papers and articles have been written on the concept of E-voting and with one country making the concept of E-voting possible. Many countries are looking and researching into making an E-voting system which has all of functional requirements that have been fulfilled by the current ballot voting.

The crucial point in the project is to provide a proof of concept for a system which is fraud-proof by keeping the identity of the individuals anonymous but still allowing the user to obtain proof of their vote. This will ensure that the fully transparent system can hold sensitive data and can be protected from malicious intent but giving the users the flexibility of using a safe secure platform.

My research has suggested that Blockchain could be an ideal implementation for the E-voting system however there is no proof of concept that it will work to meet all the requirements from the ballot voting systems or even the e-voting system implemented in Estonia. I aim to give a proof of concept of the blockchain e-voting application which can be fraud-proof.

Audience and Potential Users

Audience for my project I have assumed are the voters, the prospective is to allow and encourage younger generations to be able to vote from their homes. This is to primarily increase the number of people currently voting by giving them a convenient way of voting. An assumption is made that users do not need to have any technical abilities and has not already used an online voting platform prior to this.

I will also be offering a web interface which users can use to cast a vote. The government is to register the candidates. If the smart contract (ChainCode) is setup correctly so that it carries out real time verification by the nodes, where all the users can see the main functionality. To understand what Equinity wanted me to achieve I constantly kept in touch with the Equinity blockchain developer to see what sort of functionality he wanted to see in the application and I sent the definitive version of the ChainCode I had to be validated by them. To run the application, you will need to have the blockchain network deployed and the REST API running to facilitate the transfer of data to and from the blockchain network.

There is no target age for this application. I think it will be a useful system if they are of age when they are eligible to vote they can interact with the application. The application can be beneficial to both the government as it becomes easier to deploy and setup than the current e-voting systems in place and for users it just offers a more convenient way to Vote.

Background

Before I even began to consider the risks that the E-voting presents I wanted to discuss the two distinct types of blockchains available. Bitcoin which underpins this technology is a public blockchain allowing anyone to access the distributed ledger to read and write transactions. Any node in the network will have the opportunity to publish the transactions as long as the node has the ability to solve cryptographic puzzles. Consensus is a critical part of the blockchain which is carried out by the peers in the network. The peers in the network need to confirm the transactions carried by the users to ensure that they are valid before allowing the users to commit them. The transactions are not verified by the peers but are verified through an algorithm (Berke, 2017).

In comparison to public blockchain, in private blockchain the admin has control over who reads the ledger and who writes to the ledger and who can be trusted to verify the transaction. A consensus in these types can be accomplished by the peers communicating with each other, in this case each node can have a copy for the blockchain which then updates the other peers of the addition of new transactions. (Berke, 2017)

Issues with the Current E-voting and I-voting systems.

Over the years there have been many attempts at digitising the Voting system but none of the system implemented so far have been secure enough for when it comes to hosting or facilitating the elections. By researching online, I have come to find that the current electronic voting systems are failing at three requirements which is compromising the security of their systems.

To ensure that the online system is secure they have to focus on these aspects which is to ensure that the voter can only ever cast a maximum of one vote at any one time, as well as that when it comes to counting the votes the system needs to ensure that the correct votes are counted, and all the votes are counted. Finally, when a user casts a vote a third party should never be able to identify who the user has voted for (Lambert, 2017).

The difficulties when it comes to onsite voting systems is that there is nothing to ensure that the electronic voting machine does not have malware embedded in it which when the user casts a vote it will take the vote and changes the candidate choice for that vote to a different candidate. Furthermore, this can also be achieved through virus and trojan horses which cause interference for the users when the user is trying to vote (Lambert, 2017).

Even if that is not the case and the votes are cast correctly they will be recorded into a centralised repository such as a server which will store all the votes. This can easily become the target of hackers to access the server which store the votes because it is a centralised repository of votes this is a big vulnerability in any electronic voting systems (Lambert, 2017).

In the past US presidential election, The White House released evidence about Russia's hacking into their electoral systems. One of the targeted states was Illinois, where sensitive information about the users was compromised. This included 15 million social security numbers of which 90000 were active voters. However, this did not affect the election they

were able to obtain access to the information about voter registration in the electoral system. This allows the hackers an opportunity to access the data they require to manipulate voting records (Robertson, 2017).

Online voting presents a different set of risk to onsite voting system where it becomes very difficult to authenticate the user as the they user would have be authenticated through the use of a server to ensure that they are real this is an issue because of public Wi-Fi and there is no way to make sure that the users device is not comprised of key loggers where the user's credentials could be recorded to gain access to the system. If this were to occur, it would instantly compromise the user's identity, so someone can remotely identify who casted the vote or where the vote was casted form (Lambert, 2017).

As of the Swiss e-voting trials systems it was concluded that the security of the systems need to focus on preventing vote manipulation. They concluded this because there was no use of a digital signature which would ensure the users authenticity, non-repudiation, and data integrity for when the votes are being casted to regard the possibility that the user is not whom they claimed to be (Kobie, 2015).

These are just some of the issues highlighted by the current implementation of the electronic voting system. Now I will be looking at the how and if the issues address here can be solved by blockchain.

Blockchain voting systems

A Blockchain could offer a potential solution where the users would be able to cast their votes. In a real world if a general election is announced the users would have to register to cast their votes online. This will be just like the existing process for the existing general election process.

To address the issues about user anonymity the blockchain could be used in a way that once the application network is deployed and the user will be able to authenticate themselves and use the application, they will then have to enter their authentication details alongside their candidate of choice and then the vote will be then submitted. The improvement for this system will be that the users will be able cast their vote multiple times and but only their final vote will ever be counted.

Since the user information never stored on the system there will be no way to determine the user's identity thus making it very difficult to identify who they have voted for. This will effectively reduce the chances of vote manipulations. Each time the user re-casts a vote they will be extended the ability to check their vote as well see their voting history to make sure that the votes they have casted are their own choices.

To address the issues about vote manipulation and fraud. Even what the admin has access to see and modify can be restricted. On the governments side, they have to setup the candidates by entering these details such as the First-name, Surname, and details of which constituency they are entering on behalf of. One of those details, candidates are setup on the blockchain and the application can be then deployed so that the other users will be able to access the application. The improvement is that is that the data is public and once the

candidates are written on the blockchain the tally of votes cannot be manipulated without it being recorded on to the blockchain and this is traceable. Once the election is closed the tallying of the votes can begin, to reduce vote tampering government officials have no access to change the vote variable which will reduce the risk of vote manipulation.

Requirements

Aim

The aim of my project is to develop a proof of concept by using the blockchain technology and smart contract logic to develop a secure voting system yet making the platform modular and extensible.

I aim to use blockchain and smart contracts to improve the current electronic systems in place to reduce election fraud. The main functionalities I want to focus on is to make sure on developing a secure system which is why I have written these as the essential requirements.

Here I aim to list out few of the essential requirements and desirable features I want to implement in the system.

Essential Requirements

- The individual should only be able to vote for a single entity.
- The potential for the individual to change their vote and confirm the change of vote.
 - This also requires the user to be authenticated and already logged on to the system where they should be able to vote.
 - The user should also need to remember their username and password because this will be a security measure to vote.
- The person who participates in e-voting needs to be able to make sure that their vote is correctly accounted for.
 - This works under the assumptions that the user has already registered to vote and has access to the network.
 - This mean that the user has already voted and once the Election day has ended, and when the vote is being counted a user should be able to log back into the system to see their vote has been counted.
 - This also means that If the user has voted more than once their most recent vote that they have casted will be taken as the final choice and the system will allocate votes according to candidates accordingly.
- No-one but the individual should be able to identify themselves and see who they have voted for.
 - This is based on the assumption that user remain anonyms, so that no data of their should be accessible or read by any other individual beside themselves.
- The application should be as modular as possible so that it can easily be extended at any point.

Justification for the Essential requirements

Requirement 1: This is a vital requirement as it in any normal election a voter is only allowed to vote for any one candidate. I have implemented this as this is how the elections work.

Requirement 2: This was an important requirement which was suggested by Equinity, I wanted to implement it to see if the user could vote multiple times and their most recent vote would be counted, this would be important to prevent and stop users from casting fake

votes and giving the users flexibility to change their minds. Not only this but by supporting this you can quickly find vote manipulation as the history of document will change and the users can be aware of the issue.

Requirement 3 - With this the user can check that their vote is counted, and they can confirm that it is counted correctly so they can be sure to see that the results of this election is reliable and accurate.

Requirement 4 - As blockchain records information on blocks which will be visible to each individual on the network especially the government officials who are deploying the blockchain network will have access to sensitive information which is why I have designed the application in a manor where no individual can identify and access the other individual's information and breach the anonymity of the user.

Requirement 5 – This would make the project more robust as you are able to quickly address the errors and address it for the system to become more fault tolerant and this would also help the extensibility of the system to that more functionality could be easier added and debugging becomes an easier process

Additional Functionality

- To encrypt all the data on the Blockchain
- To create a Rest API as intermediary service between the interface and the Blockchain
- To create a Website which will be the interface for the Blockchain for demonstration purposes.

Justification for Additional Functionality

Requirement 1, I have decided to not implement this functionality as I started to develop the project I decided it would be better to avoid storing sensitive user's information on the blockchain to preserve the user's identity and prevent voting fraud. I did this because if the users information was being stored in the blockchain it would be easier for the people contributing to the network to identify the blockchain containing the voter's information. By doing this I can ensure that the system is secure, and that election fraud cannot easily happen while keeping the transparency in the ledger.

Requirement 2 This requirement I wanted to implement as It would ensure that the application is returning the correct information from the ledger and the correct logic is being used to execute a transaction. It seals the applications off so that only the data is needed for a particular function is used and other irrelevant information is ignored.

Requirement 3 This was a vital requirement which I wanted to implement as it would make the user only have access to the functionality that they need to have access to sealing the rest of the system from interference so that the logic cannot be changed. So, there will only be one access point for the application.

Assumptions

While developing this project I made few assumptions which are:

As previously mentioned the end-product will be a fully functional prototype, but it will be more focused on meeting the functional requirements, not on having the most appealing application.

You have to ensure the network is online before bringing up the application server which will launch the web interface for this application.

To make sure the system remains secure and preserve the user's identity I will assume that users have already registered and have access to this system which will extend them with the functionality to cast their votes. This would ensure that even the admin will not have access to the most sensitive information about an individual.

Once the network is deployed, the admin has to ensure that voting has been open so that the other users will have access to correct functionality.

The duration of the election will be controlled by the admin and they will decide when the election will start and end.

Methods & Tools for Solution

To develop an online e-voting system I will be using various languages and frameworks, to design and build the system. The justification of why I have used the languages and the frameworks to develop the project as follows.

Choosing the Blockchain.

Hyperledger is an opensource project supporting different blockchains for development, and there are few named websites which can be found in the documentation to interact by asking question in the Hyperledger community. When it came down to choosing two blockchains in Hyperledger I was indecisive between Hyperledger Fabric and Burrow.

Fabric ChainCode is the equivalent of a smart contract and is executed in a secured docker container which is separated from the endorsing peers and the ChainCode initialises itself and keeps up the ledger states using the transactions that you submit through the application giving you a bit more control. On the other hand, Burrow executes a smart contract in an Ethereum Virtual Machine (EVM). Consensus is always agreed on which code is to be executed and all the peers agree of their behaviour and they have access to the same data that is available on the blockchain allowing all the peers to conduct the same calculations. (Softjourn, 2018)

One of the reasons I chose Fabric over Burrow was because once the chaincode is installed and instantiated I was able to change and upgrade the smart contract any time afterwards. Meanwhile in Burrow contract the contract could never initiate a transaction because they are reactive, and a new contract would have to be created by other users using the bytecode operation. To have control of what was being logged when the function has started running and when it has finished executing this functionality was offered by fabric whereas with Burrow I could not do that, those receipts returned by the transactions are not stored by the blockchain. (Softjourn, 2018)

Since e-voting systems are required to be highly secure because of their nature. I decided it would be better for me to pick Fabric over Burrow because it gives me as a programmer much more control over Hyperledger Fabric and control over its execution and how it behaves with the logic.

Language choice for Chaincode.

After choosing the Hyperledger fabric, I decided I would choose the Go programming language to program my ChainCode in rather than java. This was because Go made it easier for me to parse the data into JSON so that it could easily be added in to the database and retrieved from the ledger.

In addition to that the support for Java was still being built and more functionality and resources were needed by java which are still incomplete. Meanwhile, for Go the resources were already defined and completed. (Raja, 2017)

The Shim interface would allow me to read and write data into and out of ledger where I can get it to interact with the network by using the functionality available in the Shim interface. Another thing which convinced me to use Go was that Hyperledger Fabric was

written in it and it would be easier to ensure that smart contract is in the same language, so I can easily understand and depict any errors that's arise. (Hyperledger, 2018)

Data storage.

Data Storage for the Blockchain, I primarily had two options of either using Level DB or Couch DB. I chose Couch DB over Level DB because it gave me more features and a way to debug the system. Using Couch DB, I can always check that the correct data is recorded and written in the right format using the fauxton API meanwhile Level DB did not offer any functionality like this.

While keeping in mind the functionality of the application and the complex data model I wanted to implement, as well the speed and type of storage structure I wanted to have I decided that it would be best to use CouchDB. I chose CouchDB over Level DB because I wanted to model the data using JSON and use CouchDB syntax to perform some complex queries which were not supported Level DB. Nonetheless one of the other feature to note was that the speeds using CouchDB was very quick to read data out of and into the database (Snellinckx, 2017).

Furthermore, as I was using the Node SDK and JavaScript to interact with the smart contract which would read data out of the ledger CouchDB which very easy to work with, so that I could easily combine and use the framework that were already supported.

Frameworks & JavaScript Client Network development

NodeJS is an open-source platform which provides runtime environment for building applications. It is written in JavaScript so there are very minor if any changes between languages for defining functions. This will enable to learn and rapidly develop my application. Ideally NodeJS consists of a package manager which allows you access to diverse range of packages for development (sigoa, 2018).

This was vital as one of the packages it has is for building blockchain application. It allows me the access to Hyperledger Fabric Client framework which I utilised to get access to the tools and modules offered by the framework which allows API to interact with the Hyperledger Fabric Blockchain network (Node Js, 2018).

I used this module as it gave me a Fabric Client module which I could easily implement into my application to use it. What this module essential does is that it allows you to create a client object and the chain in the Blockchain, it goes to add all the clients through using port number of peer's orders and the CA.

The CA is otherwise known as the certificate authority, is another module in the framework. which is used to implement and use membership management. The CA use the http protocol. This is primarily used to ensure that when a user is enrolled into the system they are given certificate providing their existence and as a way to identify the user their role in the blockchain (Hyperledger, 2018).

This framework was vital to for me to be able to build the application which can invoke and query the ledger using the smart Contract I had implement.

API development.

When I started to search for popular technologies that were compatible with node.js and a NoSQL database. Express.js was the most popular choice, with so much support and resources available it was highly promising that I could implement functionality that I needed to facilitate the transactions from the web interface to the blockchain network (sigoa, 2018).

I have chosen Express.js, this framework is developed in JavaScript and it has NodeJS as runtime. It is one of the most popular web frameworks. It allows me to create a compatible application which will work with any underlying platform. Most importantly it provides with HTTP methods used in browser to send POST and retrieve data GET. To further justify I will also have the ability to route the application to a particular function using URL.

Web Interface

One of the other benefits offered by the Express.js is that it will work well with any templating engine supported by NodeJS. Aside from the REST API the router can also be used to dynamically render pages for the web application.

A templating engine enables you to have static file, that when they are running the templating engine it will use the template to generate the actual HTML file. This enables rapid prototyping for the GUI because tags don't have to be repeated. Most popular and compatible with Express.js is Pug (Express.js , 2018). It can be downloaded with ease into the existing node modules ready to use with Express.js as long the path for the folder containing the pages is defined in the Express.js application file.

JavaScript and AJAX (Asynchronous JavaScript And XML) will be used to establish a connection with the Express.js Sever file so that data can be exchanged in the background between the AJAX request and the API (w3schools.com, 2018). This will allow me to show the results without having to load the page making it efficient.

I will be using JavaScript to ensure that the data can be displayed correctly as well as AJAX to ensure that all of the latest query results are shown on the page. This will be helpful due to the fact that you don't have to keeping loading the pages to see the latest results AJAX deals with this automatically. This will ensure that the GUI can be implemented with ease and that the functionality can be displayed correctly and accurately.

Approach

My approach to this project has been based on a Feature-driven development. This an iterative Agile method which works well with my project as I have predefined what I want to achieve.

This is seen as one of best practises in the industry and more importantly for my project it allows me to develop by focusing my time on one stage at a time. As I work through all the stages it will enable me to develop features that are of excellent quality, effective and accurate. Due to its iterative nature I can easily make modifications with the designs and be aware of those changes.

Feature -driven development has 5 explicit stages which are as follows: Develop an overall model, build feature list, plan by feature, design by features and build by features.

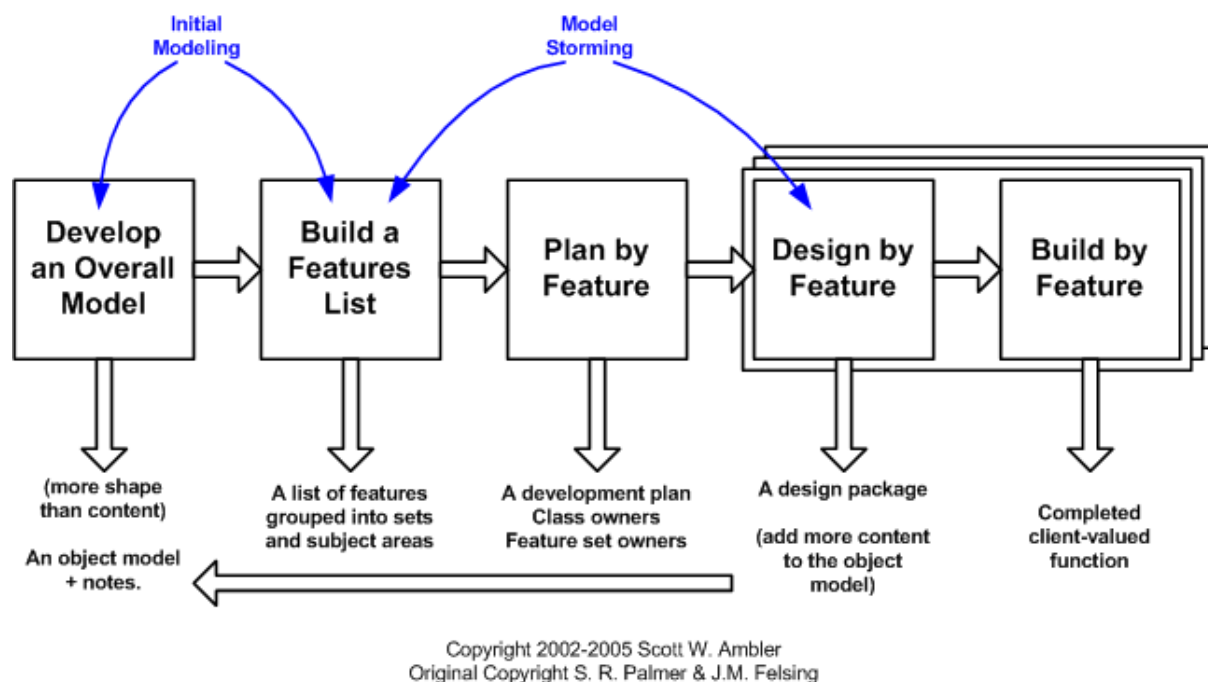


Figure 3 Feature driven development practise (W.Ambler, 2018)

The way I used feature driven development was to break the Hyperledger Fabric into a series of stages, and at each stage of the application development I used feature driven development. This was done to make sure that the system was effective, and it was working and functioning properly to create a fully working application. This was ideal as each stage of the applications needed a distinct set of features to be implemented for it to work in conjunction with the Hyperledger fabric network.

I wanted to ensure that the platform I developed will be extensible and modular. By choosing this practise of development it enabled me to continuously modify and iterate over the design to achieve modularity and extensibility of the platform. I wanted to focus on this as it ensures the quality of code and while ensuring clarity on how the code functions making it easy to identify the issues in the application.

I however did not follow all of the practises of this process because I did not have a clear idea of how I should implement the functionalities thus I did not design an overall model of what I want to achieve I based the overall model on the functional requirements to develop the system. I chose the best practises to focus on and to not focus on in feature driven development. While starting this project I had very vague ideas of the development of the project by speaking to Equinity and researching as I began to implement the functionalities, ideas became clearer about what the results of the application should be and how to achieve them.

The requirements were what I used to define the functions in chaincode and this was the logic that would be implemented in the application to ensure a secure functional system. Meanwhile, some requirements were derived from speaking to Equinity blockchain developer the others I have implemented were obtained because of research. These are the requirements I used to develop a feature list.

When implementation began it was clear that this project should be developed in a hierarchical way to ensure that all the functionality is effective. While keeping this in mind with the complexity of setting up Hyperledger fabric. I applied this to develop an overall general time plan for my project. I started to implement the features that were necessary and were essential within the system. I did this this way because without the logic of the system (chaincode) there was no point in me building a NodeJS application which would interact with the blockchain network. This enable me to ensure that each stage of the application has the correct set of features allowing me to link the application together with ease.

Overall because this project was constantly evolving, I reviewed this later stage when I was further ahead in my project and I realised that some of my requirements have changed and some are just obsolete then I had to decide to pick the features that complemented the project and focused on making them better. The features I have decided were necessary to implement, I assessed the complexity of the features in addition to how they could be incorporated within the system. The next aspect I analysed was the timeline of how long it would take to implement the feature to ensure that the feature will be fully functional within the given timeframe.

By doing it this way and testing the implemented functions against the network to ensure its outcome. This complemented the feature driven method perfectly as some features may need to be modified to work accurately and effectively. The most important and challenging was the implementation of the smart contract which was the feature that needed to be developed so by distributing enough time for the development of the smart contract it was fully operational and working. This enabled me to create the timelines where I could schedule the supervisor meeting for a review and to track how much of my project I have completed and to constantly keep my supervisor informed about the milestones in my project and to make sure that the ideas are worth implementing and to ask for advice about when I have been stuck. For every feature I did develop I made sure to check that they coincide with the initial requirements I wanted to implement.

System Design & Architecture Overview

Blockchain, is a distributed ledger which has peers who communication with each other. Each of those peers have a copy of the smart contract, these peers are often related to their domains which they trust and entities that will control them. So, all the peer run as if they are part of the same physical server.

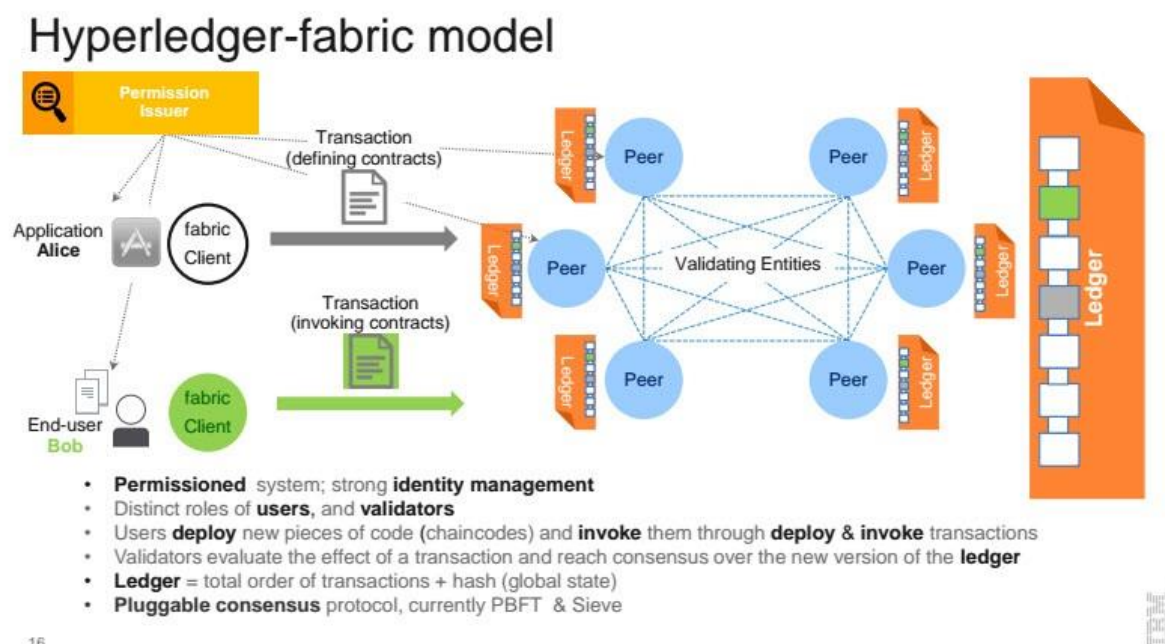


Figure 4 Hyperledger Fabric Infrastructure (Strukhoff, 2016)

I have been using the Stable version Alpha v1.0 which was released when I started with my project in February 2018. In this system architecture we have the client which will run the NodeJS web application which consists of and utilises Hyperledger Fabric Client SDK. The web application that I have developed will use the Hyperledger Fabric Client. The Client will be used to enrol and register the user by accessing the membership services. Membership services provided by the client CA are critical as it assigns the user enrolment certificates (ECerts) upon when they are registered, and it issues transaction certificates (TCerts) this is what gives the admin both anonymity and non-repudiation when conducting transactions on the blockchain. (Hyperledger, 2018) The registered individuals will be allowed to connect to the Hyperledger network to invoke transactions on the peers.

Figure 5 will demonstrate how the client application will function. The client application will consist of a website, REST API and the Node SDK. SDK for my application will connect to three different entities one of them being the Orderer which is an addition to the version in comparison to version 0.6. The Orderer from the ordering service provides the communication channel which is shared between the client and the peers, so essentially the client can access the channel to broadcast messages which are then sent to all of the peers, so they can build the block for the blockchain.

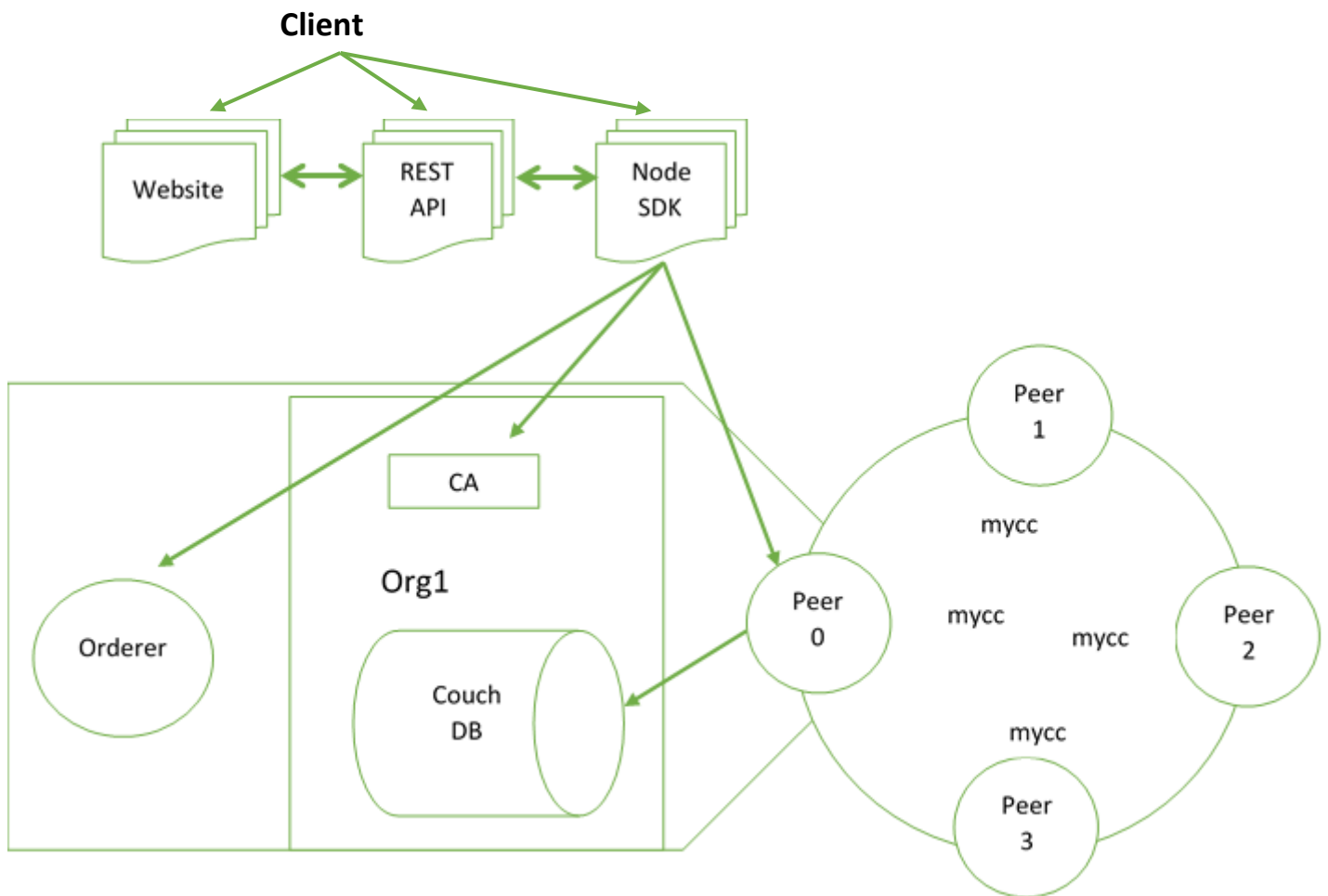


Figure 5 My System Architecture for Blockchain development.

Each peer node in this case will have an instantiated version of the ChainCode this happens when the network is brought up by default the instantiated version is called 'mycc' which stands for my chaincode. The incoming transactions from the client node will invoke the functions defined in the smart contract. The instantiated copy of the smart contract will allow all the peers to verify the transaction so, they can either include the transaction in the block. If all of the transactions recorded by the peers are valid then the block gets added to the blockchain (Hyperledger, 2018).

Any transaction sent from an application requires endorsement. The transaction proposal is sent to peers for validation to an endorsement peer. This is often the peer that you will connect to using node SDK Fabric client. The endorsing peer will endorse the transaction before it is committed to the ledger following an endorsement policy to ensure that the transaction that you trying to commit is valid, this is different for each chaincode thus the endorsement policy for each chaincode is different. Once it has declared the transaction to be valid it will execute against the ledger affecting the state of it (Bezgachev, 2017).

The Client node, which is our application for it to communicate with the network we connect the client to the CA which is the Certificate Authority. The CA then issues the registered user with the enrolment certificates upon bringing up the network application.

The CA is always contained inside the Organisation otherwise shown as Org1 when you are deploying or looking at the Hyperledger application this is what creates and connecting the web application to all the peer and the blockchain network via a channel which by default is set to mychannel. The messages broadcasted are normally delivered to all the peers unless there is an endorsing peer which is set in the Node SDK which in this case means that the messages will be sent to Peers unless you have selected an endorsing peer. In this case it will be sending the transactions through the channel to the Peer0.

E-voting system

The proof of concept system will be created using the NodeJS. The system will give access to different user's roles such as government official who should be responsible for deploying the blockchain network and making sure the application is up on election day. Not all of the functionality accessible to the admin will be available to the voter.

A government official should be able to register and enrol other candidates on to the blockchain. To register candidates some details will be needed such as the constituency and the candidate name. This should be done before the voting system is online where the users are able to access the application to submit their vote. Once the network is deployed and Blockchain voting is open only certain functionality will be available to ensure that the system runs smoothly. These functionalities will include the user's ability to cast their vote and modify it. They will also have the ability to check their vote and to get their voting history over the duration of the election period and even after it has ended so the user can see that their vote has been accounted for.

The user population of the system who are eligible to participate in the election process can enter the necessary details to be able to submit the vote. Modifications of votes can be made on behalf of the users if the users provide the correct credentials. This reason this is possible with the ledger is because the ledger can maintain a record of the users votes using the credentials they have submitted by generating cryptographical puzzles.

The participants will have the ability to comprehend their voting activities, as it will be made available to them. The submissions of their accountability of the votes will be represented by two different status depending whether it has been accounted or not.

On the other hand, once the votes that have been accounted for, the user will be notified of a change in their status. The reason that the votes are not being counted immediately towards the final tally is because I plan to use a Boolean value to determine whether the vote has been counted or not, this would allow me to make sure that only the latest vote input is counted, and it is only counted once. The user has the option to keep modifying their choice of participants until the election is finished. This will reduce voting fraud as nobody has control over the process of how the votes have been counted.

Once the general election has concluded, the government official will be able to evaluate the outcome of the election and declare a winner.

The full specification of the system agreed by Equinity, to make sure all the functionality that they wanted to develop have been met will be in the next section.

Design

I have tried to create a proof of concept prototype of how the e-voting system should work. As I have already mentioned I am more focused on creating an application that meets all of the essential requirements and not developing an a visually appealing application.

I have still tried to implement and follow all of the Nielsen's heuristic principles. To try and develop to develop an efficient and effective application.

Web Interface

The web interface is separated into two sections one for the admin who should be a government official assigned to deploy the blockchain application. The functionality given to the admin will be different to the users who will be using the application to vote.

Admin Interface

First, they will be able to see the navigation bar which will show them all of their functionality that is available to them. The admin should have these functionalities in the menu, Register Candidate and View Candidates. Aside from this the index page will have two buttons for opening and closing the voting system. As well as a search bar where they will be able to search for things.

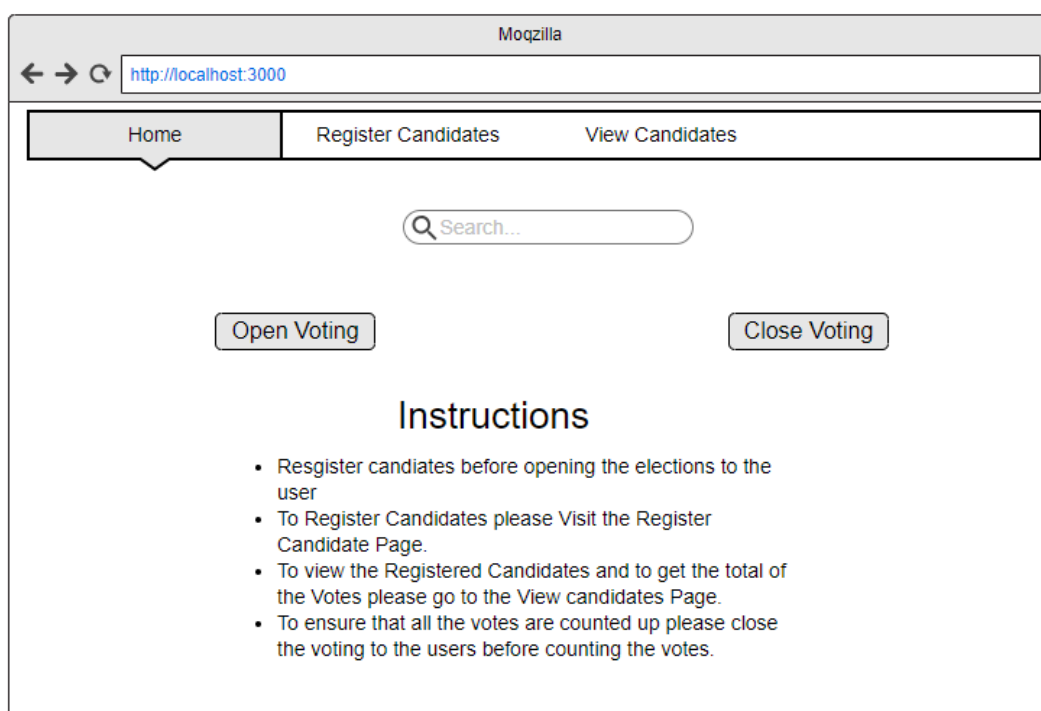


Figure 6 Admin Index page - design of the index page

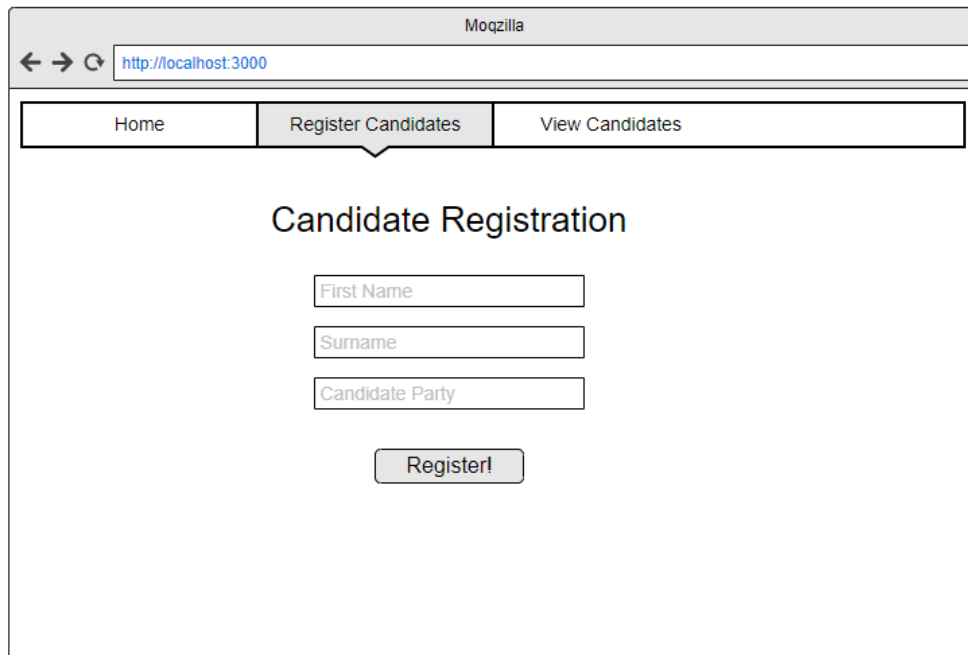
Behaviour

- if the Open voting button is pressed the Voting for the users will be open.
- if the Closing voting button is pressed the voting for the users will be closed.
- If you use the search bar, then if the results exist then they will be displayed underneath the search bar.

- If the navigation bar is used then it will take you the Page you have selected to view.

Register Candidates

Once you have navigated to 'Register Candidate' this page the functionality will allow you to add Candidates who are participating in the Elections. This will then register the Candidate on to the Blockchain.



The screenshot shows a web browser window with the title 'Mozilla'. The address bar displays 'http://localhost:3000'. The navigation bar contains three links: 'Home', 'Register Candidates' (which is highlighted with a dropdown arrow), and 'View Candidates'. The main content area is titled 'Candidate Registration' and contains three input fields labeled 'First Name', 'Surname', and 'Candidate Party'. Below these fields is a 'Register!' button.

Figure 7 Design of the candidate registration page

Behaviour

- To register the Candidates to the blockchain you will need the following details their first name surname and the party they are running candidacy on behalf of.
- Then they should be able to submit the application. By pressing the Register button.
- The candidates will only be registered on blockchain if the voting is closed for the users because without any candidates they should not be able to register any votes on the platform.

View Candidates

Once you have navigated to 'View Candidates' it will show you all the candidates that are registered on the blockchain and all information associated to them.

View Candidate Information

▼ Candidate Firstname	▼ Candidates Surname	▼ Candidate Party	▼ Votes
One	Test	Labour	0
Two	Test	Conservatives	0
Three	Test	Green	0
Four	Test	Liberal Democrats	0

Search Candidates Party

Count Votes

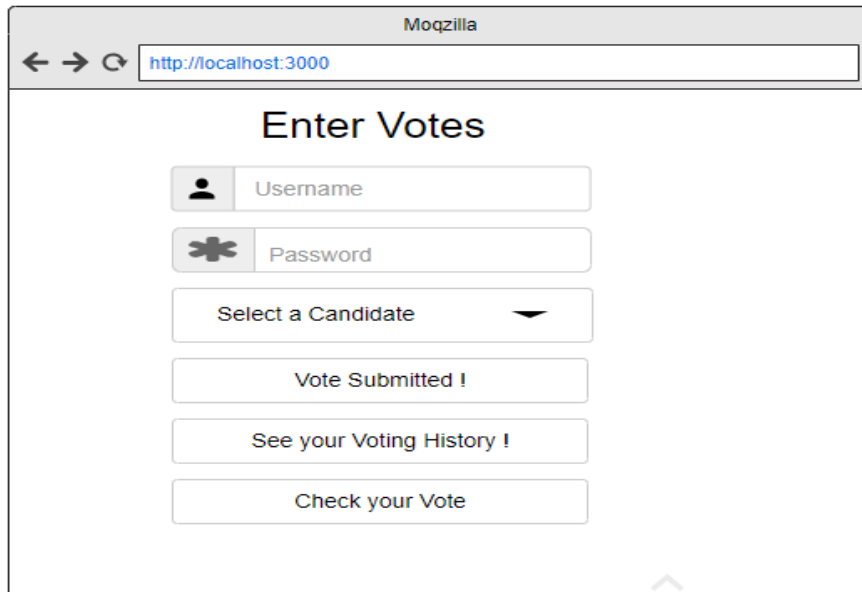
Figure 8 Mock-up of the viewing all the candidate's information.

Behaviour

- A table will be displayed to the users showing all the candidates on the ledger.
- At the bottom of the page a button will be displayed labelled 'Count Votes'. Once the election is closed only then the admin will have the ability to count the votes and have the total votes displayed in the table.
- There will also be a search bar placed underneath the table displaying the record. The search will take the name of the candidate party and show you all the votes casted by the users for this particular candidate.

Users Interface

Once the users are logged in they will be directed to a Cast Vote page which will define all the functionality that the user will have access to on the system. This is the functionality where the user should be able to write data to blockchain.



The image shows a web browser window titled 'Mozilla' with the address bar displaying 'http://localhost:3000'. The main content area is titled 'Enter Votes' and contains a form with the following elements:

- A 'Username' input field with a user icon on the left.
- A 'Password' input field with a key icon on the left.
- A dropdown menu labeled 'Select a Candidate' with a downward arrow.
- A button labeled 'Vote Submitted !'.
- A button labeled 'See your Voting History !'.
- A button labeled 'Check your Vote'.

At the bottom right of the page, there is a small upward-pointing arrow.

Figure 9 Design of user voting page to submit their votes.

Behaviour

- Once the user is on this page the user should be able to see a form with a set of fields that need to be filled before submitting the form.
- To fill out the form they will need to have their username and password and then there will be a dropdown menu element which will allow the user to select the candidate that they want to vote for.
- Once the user has filled out the form they can select to submit their votes. These will then be written on the ledger.

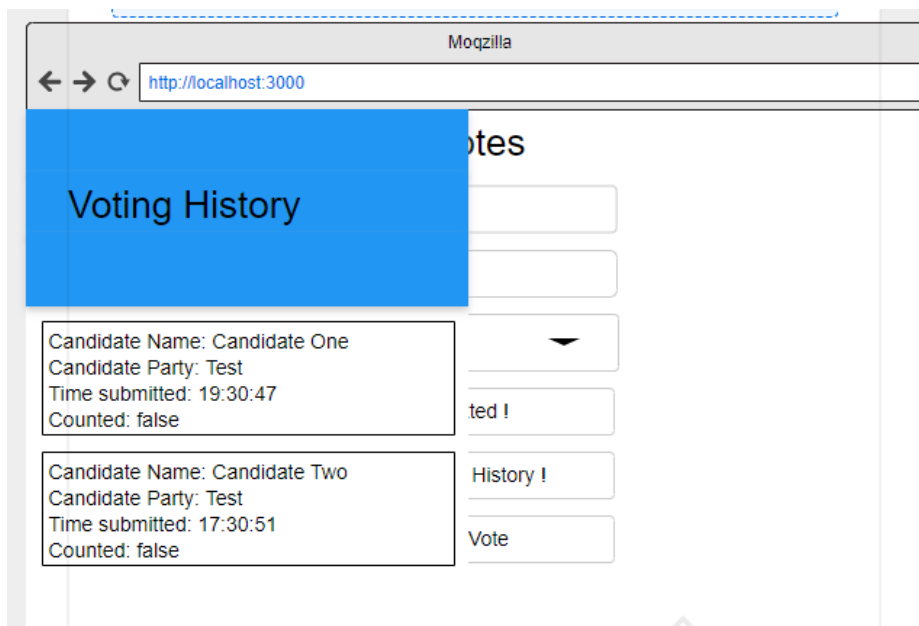


Figure 10 Slide out panel for the user page to check their voting history.

- They can see their Voting history to see all of their submission of votes. All the information related about their votes and modifications to their choices will be retrieved and displayed. To use this functionality the user will need to have their username and password.



Figure 11 Slide out panel allowing the user to check their latest voting submission.

Lastly the user will also be able to check their latest vote this will show them when their vote has been counted. Keeping the user informed about the votes. To check their votes, they would also need their username and password.

Implementation

Hyperledger Fabric is a network which has been developed and supported by the Hyperledger community. Fabric is a GitHub repository which you download, which consist of all the files that you would require to build your ChainCode and bring up and deploy it to the network (Hyperledger, 2018).

Network Development

Setting up the Hyperledger network for smart contract development, this can easily be achieved using Docker. Docker is a containerisation software. (Docker, 2018) This tool will allow you to setup the network in your local machine by providing you with containers where the code will be executed. These containers which are provided by docker use libraries and settings to enable docker to work with the network. When your setting up the network the Hyperledger community tells you about the images published by them which are needed by docker for it to operate correctly with the blockchain network. Docker is not the only option to setup the network there are multitude of option such as IBM Bluemix, but it requires a subscription to access it, however, there is a trial phase for 30 days.

Moreover, reading the documentation is essential when you are trying to generate the network to ensure that all the prerequisite tools for the project are installed correctly if this has been achieved it becomes easier to generate the network. Docker needs to be running while you are trying to bring up the network. What docker does in this scenario is that it creates the containers in which the CA, Orderer, CouchDB and endorsing Peer which in our case will be Peer0 will reside.

Due to the design of Hyperledger Fabric there are two diverse ways to interact with the network either using the Node SDK or the CLI (command line Interface). By incorporating the CA in my network, I had to move my CLI container from the docker-compose file. The container had to be moved to the docker-compose-e2e.yaml and the template file where the docker-compose-e2e.yaml file is generated from so the that the CA and CLI container can both run at the same time. The CLI runs the script.sh file. The script.sh calls the orderer which creates the channel, and this is the aspect that handles the consensus of the network, thus this is where all the transactions are executed in the network. It sets up the orderer and creates a channel called 'mychannel' and has Peer0 and org1 join the channel. The scriph.sh also installs and instantiated the ChainCode so now all the peers will have access to the newest version of ChainCode, so verification of transaction can be conducted. Once this is done then the node Js application can be used to carry out Invoke and Query transaction. (Hyperledger, 2018)

Issues with setting up the network were uncovered when I was testing the network for its usability and its functionality, so I have addressed all the issues I experienced while setting up the network there.

Implementation of the chaincode

Chaincode is the business logic otherwise known as the smart contract. This is where you outline the instructions for the transactions. These rules that are outlined in the chaincode will become the rules that are enforced by the contract before the data is recorded. The transaction is sent in the form of transaction proposal to ensure its validity before the transaction is executed against the states of the ledger, the results are stored in the ledger as key-value pairs. Once this transaction is sent to the network it is sent to all the peers and they all apply the same logic and verify the transactions.

I have written my chaincode in Golang to make sure it can be understood and executed by Hyperledger fabric network. I have previously discussed the benefits of this in my methods and tools for solution section.

Dependencies for the chaincode

I have implemented a few dependencies which were imported to make sure I had the correct tools to make sure the application was done to high code quality.

- **Fmt** – used to log and print result returned by the network
- **Errors** – the error library is used to make sure that relevant, clear, and concise errors shown to the user so that they can understand what the issue is in order to address the issue.
- **Shim interface** – this is the interface that allows me to write the go code into chaincode and this is important to interact with the ledger.
- **Peers** - this was part of the Shim, though it is to make sure that it interacts with the peer so that the transactions can be verified by the peers.
- **Sha512** – this dependency was used to develop a hash function which will generate a hash based on parameters. This particular library was chosen because it has become the industry practice.

Chaincode Interface

This interface offers three functions that can be implemented which are Init, Invoke and Query. This is done through stub which allows the functions like these to read and write to the ledger (Hyperledger, 2018).

To insert the values as a key value document into the ledger this is done through using `stub.PutState()` before the document is inserted it needs to be marshalled into JSON. On the other hand to retrieve values from the ledger you need to use `stub.GetState()` and when it is read out it has to be unmarshalled before the value can be read.

Init is always the first function to be executed, when the chaincode is deployed to the network Init functions instantiates the state of the ledger and set the initial Key-pair values on to the ledger before any other function is executed.

In the system only, when the Init function in the chaincode is executed it will initialise the current state of the vote system. The initial state of the voting system will be set to closed hence only allowing the certain set of functionalities to be invoked. This will remain until the

state of the blockchain has changes for the system, it means that until the election is opened to the voters.

Invoke is the function that is called when you want to read or write transaction to the ledger. What I have done is defined all the method invocations inside an Invoke function so that when a method is invoked the request and the data is grouped in to a transaction which in turn is written on to the ledger. Invoke also confirms that the function is available so that it can be executed otherwise returning an error saying that this function does not exist.

```
Error: Error endorsing invoke: rpc error: code = Unknown desc = chaincode error (status: 500, message: Received unknown function invocation) - <nil>
Usage:
peer chaincode invoke [flags]
```

Figure 12 Error to demonstrated that invalid function was invoked.

Query will often be used to read data out of the ledger and it will extract the data in key value pairs. For queries to be syntactically correct they must use Mango query syntax which is supported by Couch DB.

Finally, you have the main function which is where you execute this statement `shim.start()` with an empty struct in go, this is used to set-up the communication between all the peer that will hold a copy of the chaincode. Once the struct initialises the chaincode, if any changes are made by executing new transactions they can be verified by the other peer before being committed to the ledger. This function will remain same for any chaincode you implement, and it required to start running your chaincode (Hyperledger, 2018).

```
vidhi@vidhi-Vostro-5468: ~/go/src/gitlab.cs.cf.ac.uk/c1524659/My-Blockchain-E-voting/first-network$ ./byfn.sh -m up -s couchdb -f docker-compose-e2e.yaml
Creating couchdb0 ... done
Creating couchdb3 ... done
Continue (y/n)? y
proceeding ...
```

Figure 13 Commands to deploy the network using Couch DB and CA.

Using these commands, I was able to change the underlying database from LevelDB to CouchDB, which I decided I would use in my tools. Moreover, I was able to use the second come to use the CA in my application. This proved to be critical, without this the NodeJS application would not have been able to enrol the admin for the application to function while interacting with the chaincode. This has made my system architecture in figure 5 possible to implement in reality.

I have attempted to implement the application as I have discussed in the system architecture of the E-voting system. To ensure all the functionality has been met. I will also add the functions Init Invoke and query in the appendix.

Node SDK

This is the phase in which I start to use the Hyperledger Fabric Client SDK which is a module in node where all the relevant modules required for this part of the application are downloaded in a node modules folder. This will provide a way where it allows us to create a network endpoint where the normal users can interact with the application that will extend the functionality of Invoking and querying ledger.

The easiest way to implement or to learn about this is to use the examples provided by Hyperledger community. These examples are available on GitHub. These dependencies need to be recorded into the package.json file. To engage with the functionality extended by the installed modules you must use require to import them in the files to use the modules.

```
var Client = require('fabric-client');
```

Figure 14 Import form the Node modules to use the Hyperledger Fabric Client.

This will load the client and connect the peers to it to get the CA to generate the certificates needed for the users for him to have the ability to invoke and query. The CA uses the http protocol which I have mentioned in the tools I am using to build this network.

gRPC allows a client application to invoke methods in a server. In a gRPC systems you can define specific methods that can be called remotely on the server. On the Server side it will run a gRPC server to execute the request you have just sent, once it has the results from the execution it will send it back to the client. (Google, 2018).

```
}).then((user) => {
  var peer = client.newPeer('grpc://localhost:7051');
  var order = client.newOrderer('grpc://localhost:7050');
  channel.addPeer(peer);
  channel.addOrderer(order);
  return channel.initialize();
});
```

Figure 15 gRPC to add the peer and order to the network to initialise the channel.

The most difficult part for me was to establish the communication and to enrol the user. By default, the blockchain network TLS was enabled. TLS allows secure communication between the server and privacy for web the application is being built but since the application will be using localhost it should not compromise the security of the system.

This caused issues when I was attempting to enrol the user because they did not have the correct credentials to overcome this I turned the default TLS setting off in the blockchain to be able to enrol the admin.

Moreover, for communication to work initialisation of the channel is important, to make sure modularity and extensibility is achieved in all of this I did all of these functions in a separate file. Network.js is the file which is responsible for setting up the network and this is where the client and channel were initialised I had issue where there promises were being overloaded and that lead to me to not have access to the client or the channel, so I couldn't execute any transactions. To deal with this issue I had to identify what was overloading the promises and then resolve the last promise before returning the channel and the client. The access to channel and the client was required to carry out Invoke and Query on the ledger.

```
}).then(() => {
  resolve([client, channel])
});
```

Figure 16 Resolving the promises to ensure this data is returned

The hardest part was for me to learn about the concept of promises and how to deal with them. Another thing I had a challenging time with was finding the correct ports for the Peer Order and CA so that I could connect them together. I had to look at the docker-compose.yaml files to see the external port so I could connect the client to the port.

After establishing the network in the node js, to test if it operating I made a request object by using the initialised channel I was able to run a query. Similarly, I was able to do the same with invoke but I had to utilise the client to generate the transaction ID of the transaction.

I created separate helper functions to ensure the extensibility of the program. After creating a function for a request object.

```
module.exports.createRequest = (contractID, fcnName, fcnArgs, databaseID, transID) => {
  var request = {
    chaincodeId: contractID,
    fcn: fcnName,
    args: fcnArgs,
    chainId: databaseID,
    txId: transID
  };

  return request
};
```

Figure 17 Implementation of the Request object to invoke and query the ledger.

I made this design choice because I could call the object whenever it was needed and the parameters that are required by query for example chaincodeID, channelID the arguments the function name and for the invoke request object it requires an additional value which is transaction ID because it writes things to the ledger, so It needs a transaction ID.

```
module.exports.setTransactionID = (client) => {
  tx_id = client.newTransactionID();
  console.log("Assigning transaction_id: ", tx_id.transaction_id);
  return tx_id
}
```

Figure 18 Generation of the transaction ID used for invoke.

In the Invoke function the proposal is first sent to check to see if it is good proposal only then is the transaction sent and then accepted and the transaction is committed to the ledger once it has checked proposal is valid for the submission.

```
return channel.sendTransactionProposal(request)
```

Figure 19 This checks the proposal transaction to make sure it is valid.

Express.js (NodeJS web application)

I used Express.js as a framework that would allow me to develop the REST API. Express.js uses HTTP methods such as GET and POST. The functionality of the Node SDK and the chaincode is accessible in the REST API so the application can include it. Any function within the application where data is written onto the Blockchain it will be a POST request and for

querying the ledger it will be a GET request. All of this is achieved by using a router. All of the data will be sent to the server on the server-side, the router will collect all the data and pass to the corresponding function in the REST API which will then generate a request for the correct function. Invoke or Query will be called to carry out the transaction.

An example GET Request in my API would look like this.

```
app.get('/getAllCandidates', function(req, res){
  query(channel, misc.createRequest('mycc', 'getAllCandidates', [], 'null', 'null')).then((result) =>{
    res.send(result)
  });
});
```

Figure 20 Example GET request in the REST API.

POST:

```
app.post('/initCandidates', function (req, res) {
  invoke(channel, client, misc.createRequest('mycc', 'initCandidates', [req.body.forename, req.body.surname, req.body.candidateParty, 'isVoting']
  ));
});
```

Figure 211 Example POST request in the REST API.

In the GUI it will be done through AJAX so there is no need for constant reloading to see the updates, However the POST request can fail from the client side when its being sent to the server

Invoke is done by creating request and generating the transaction ID and sending the request object and channel and client that it requires for the function to work from the API.

The results from a query will always be returned as a buffer or bytes so I used a function that will Parse the data coming in from the query into JSON.parse() which makes it easier for me to use the data.

```
channel.queryByChaincode(request).then((query_responses) => {
  return resolve(JSON.parse(query_responses));
});
```

Figure 22 Query in the node SDK which will query the ledger.

While I was developing the queries in the application, I found that if you ever query a document or data in a document which does not exist it will return a null value. I think it will be ideal for me to implement checks for null key or values that do not exist in CouchDB to prevent error and then this can be conveyed to the users.

GUI Implementation

To implement my interface, I used pug which is a templating engine which is highly compatible with Express. To make sure that the application was functional and interactive I used the jQuery AJAX library. I have used the router plus the server that is in the REST API to render the pages. Furthermore, to use the functionality in the API I then used the AJAX calls to call the REST API endpoint to join the application to the blockchain network.

Example AJAX calls connection to the REST API will look like this

```
function openVoting () {  
    $.ajax({  
        url: "http://localhost:3000/openVoting", //connecting to the API  
        type: "POST", //request type  
        success: function(result){  
            alert("Voting has now been Opened");  
        }  
    });  
}
```

Figure 23 Example AJAX Call to establish a connection between the application and the REST API.

Testing

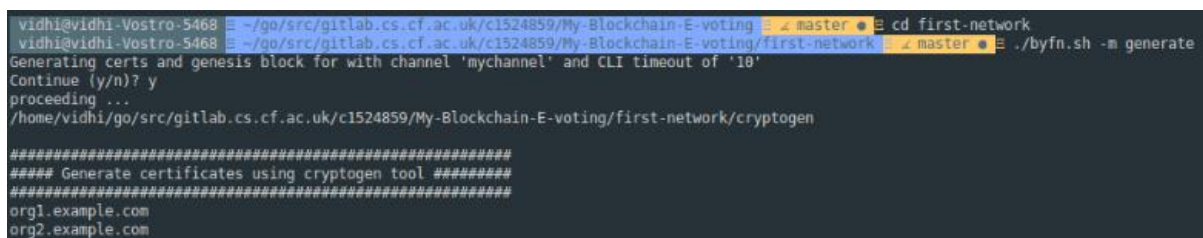
I wanted to discuss the result and my methods of testing my application to decide the results and the success of achieving the requirements. Due to defining my functionality at an early stage I had no idea of testing the functionality of smart contract and thus I had no clear specification for testing.

My approach to Testing

As I was not aware of many ways to test the application I ended up resorting to manual testing to see if the functionality was working as intended. At each stage I tested the application for several reasons to test the systems Usability, functionality, and interface testing. The test I carried out on the Fabric Network was to ensure the structure and the usability of the project and to determine its interaction application. I did this to achieve results that were correct and reliable. By testing each phase of the application, it reduced the risk of the application failing and to improve the quality of my code therefore making it easier to find the issues in the network and functions that were not performing as intended so that they could be fixed.

Fabric Network Usability testing

when installing docker and all of the prerequisite tools alongside Hyperledger Fabric samples. I want to see if I could generate the network to see if it was functional. I had followed these commands from the From Hyperledger fabric docs `./bfn.sh -m` generated followed by `./bfn.sh -m up` to bring the network up. The first time I was testing the functionality of the network I had some issues where the tools such as cryptogen was not found in the correct folder thus the network could not generate the genesis block which is the first block on the blockchain. Then I had to change the structure and re-install the tools to generate and bring up the Hyperledger Network.



```
vidhi@vidhi-Vostro-5468 ~/go/src/gitlab.cs.cf.ac.uk/c1524859/My-Blockchain-E-voting : master ● $ cd first-network
vidhi@vidhi-Vostro-5468 ~/go/src/gitlab.cs.cf.ac.uk/c1524859/My-Blockchain-E-voting/first-network : master ● $ ./bfn.sh -m generate
Generating certs and genesis block for with channel 'mychannel' and CLI timeout of '10'
Continue (y/n)? y
proceeding ...
/home/vidhi/go/src/gitlab.cs.cf.ac.uk/c1524859/My-Blockchain-E-voting/first-network/cryptogen

#####
#### Generate certificates using cryptogen tool ####
#####
org1.example.com
org2.example.com
```

Figure 24 Generates the certificates and genesis block for the blockchain using the cryptogen.

Subsequently to see execution of the smart contract I attempted to execute the example chaincode provided in the Fabric samples chaincode directory. I executed the `chaincode_example2` on the network to view the full functionality of the network. This was ideally where I started to encounter more errors in my structure because when you run the chaincode you create a GO directory where the smart contract needs to be placed to be executed for the network to interact with it. So, with my Installation of GO language the folder was created in Root. Once this issue was identified I was able to create the Go folder into the documents and then go in the src directory and then mount the Fabric samples. After moving the project and bringing up the network again I was able to get the Peers to

install and instantiate the chaincode. I tested the example chaincode by executing invoke and query transactions on it.

Once I could run the functions such as Init, Invoke and Query then I knew I could start to define my own smart contract as it was ready for my application to be implemented. This was the vital step for me to take at the beginning of the project otherwise I would not have been able to test my application quickly and efficiently. This was done to set up the development environment for the rest of the project so that the network was functional henceforth it would only require minimal modifications to get my implementation of smart contract running.

Smart Contract Testing

To test my smart contract, I used the dependencies such as the `fmt.Println()` to see the values that were being return by the methods. Also, I was very reliant on docker as one of my testing tools because I can used docker log to check the chaincode to see if the function was invoked to visualise the outcome of it. Furthermore, if an error has occurred with the chaincode I could use it to retain more information about the error by checking the peer and Organisations container to pin point the reason that the error has occurred. Ideally this was helpful when I was attempting to execute the query in the network to validate if the syntax is correct. The write transactions were tested with the Fauxton API.

A terminal window showing the command `docker logs dev-peer0.org1.example.com-mycc-1.0` and its output. The output shows the chaincode container's logs, including the message "invoke is running test" and "invoke is running initCandidates", followed by a list of candidates: "- start init candidates" and "- end init candidate".

```
vidhi@vidhi-Vostro-5468:~/go/src/gitlab.cs.cf.ac.uk/cl524859/My-Blockchain-E-voting/first-network$ docker logs dev-peer0.org1.example.com-mycc-1.0
invoke is running test
invoke is running initCandidates
- start init candidates
- end init candidate
```

Figure 25 Docker command used to access a chaincode container.

This is the docker container where the code was being executed, the main the results are seen by using the `Fmt` dependency.

NodeJS Application Functionality Testing

This is ideally where I was able to initialise my own network which interacted with the blockchain network and to initiate the channels so that the transactions can be made. This allowed me to test the rest of the smart contract that I had not been able to test. With this the testing became easy as I was able to send transactions to `peer0`. In my network, I used the `console.log()` to make sure that the correct results were retrieved. This also highlighted that the results for some functions are being returned as bytes and then I was able to write the code and address the issues that would not fulfil the requirements. Here I was able to test all my requirements and test real life scenarios of how the system can be used and how it should be used.

This was functionality testing through a separate file which I had created and stored commands in, so I can see if the following sets of functionality were executed.

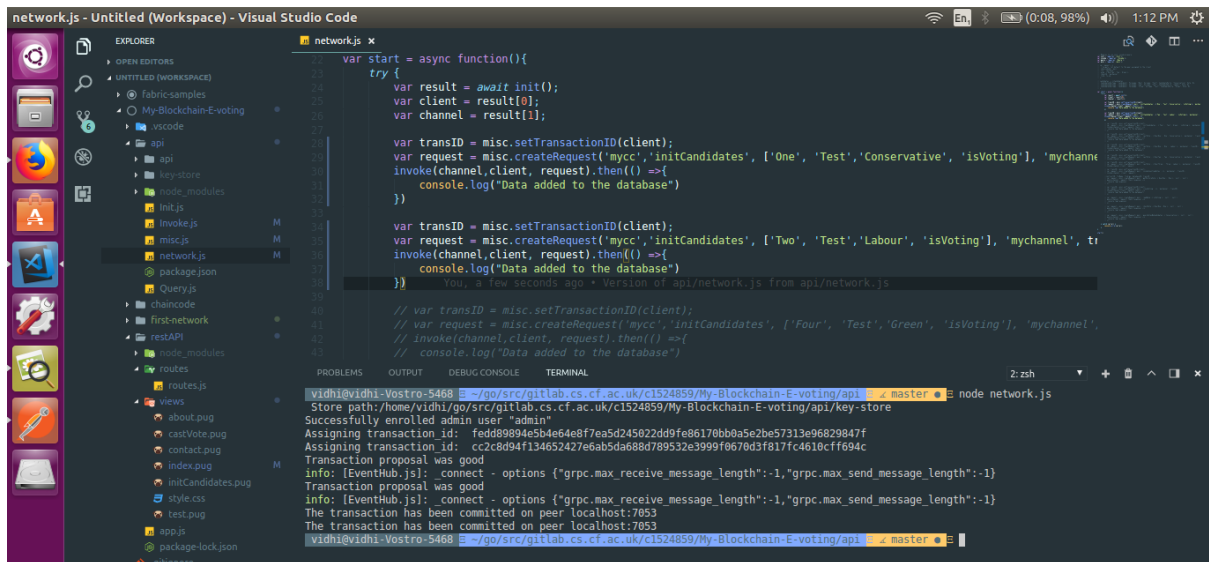


Figure 26 Displaying the submission of valid transaction.

To test the security and durability of the system I tested the network by adding duplicated sets of data of which were not written to the database and error was printed out to inform the user that this record already existed on the system

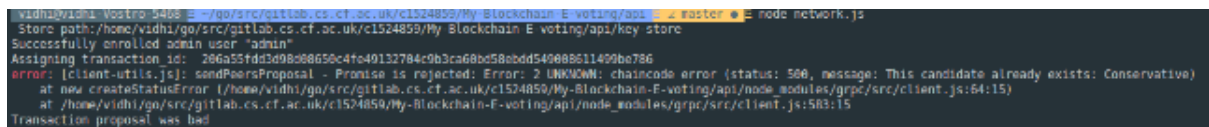


Figure 27 Example error generated for adding duplicate data.

Ideally, I used the NodeJS application to carry out functionality testing to check if the application was performing the tasks correctly and that it was carrying out the intended tasks. To further justify this, I would use docker to access the container where the execution of the code has been taken place as well the peer that has executed the transaction to ensure the intended result of the transactions. Finally, I would check the Fauxton API to ensure that the data is retrieved and written in the correct format that it should be in.

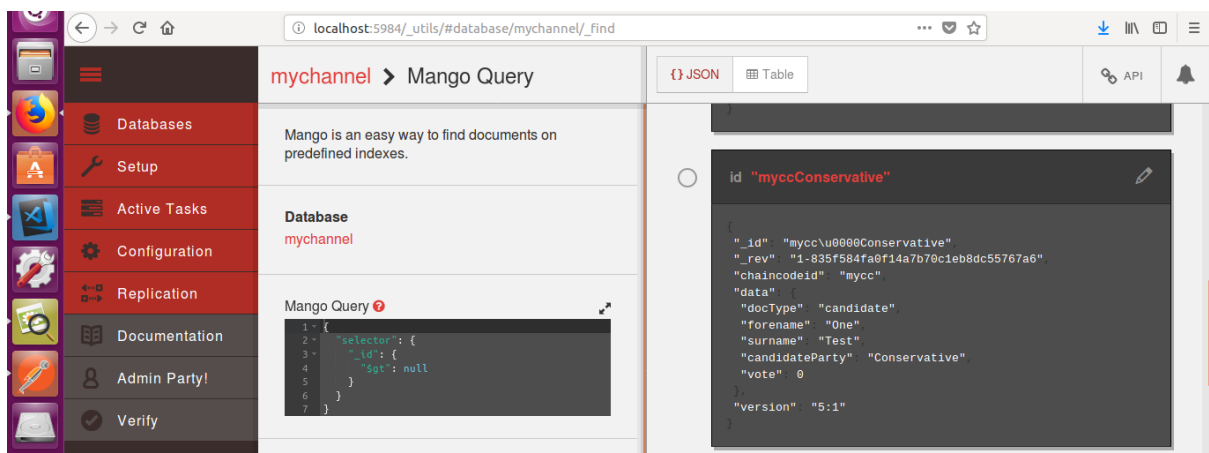


Figure 28 Viewing the information recorded in the ledger.

This is to make sure that is actually being stored in the CouchDB and the correct is being stored.

REST API Testing

The Rest API testing was a structured way to test the transfer of Information. I was successfully able to use the Http protocols I had defined in the REST API and connect to the API In Postman and supply the correct Information to see the data transfer then I would open the Fauxton API to see whether the correct information was placed into and gotten from the database to display into the webpage.

This was done to ensure that the data was routed to and from a correct path and results were correct and working as expected. This also ensures that the system managed errors better and it was easy to pinpoint the issues.

This is the Postman interface which I have used to test the REST API. This was an example of it trying to send a GET request where the result came back as null.

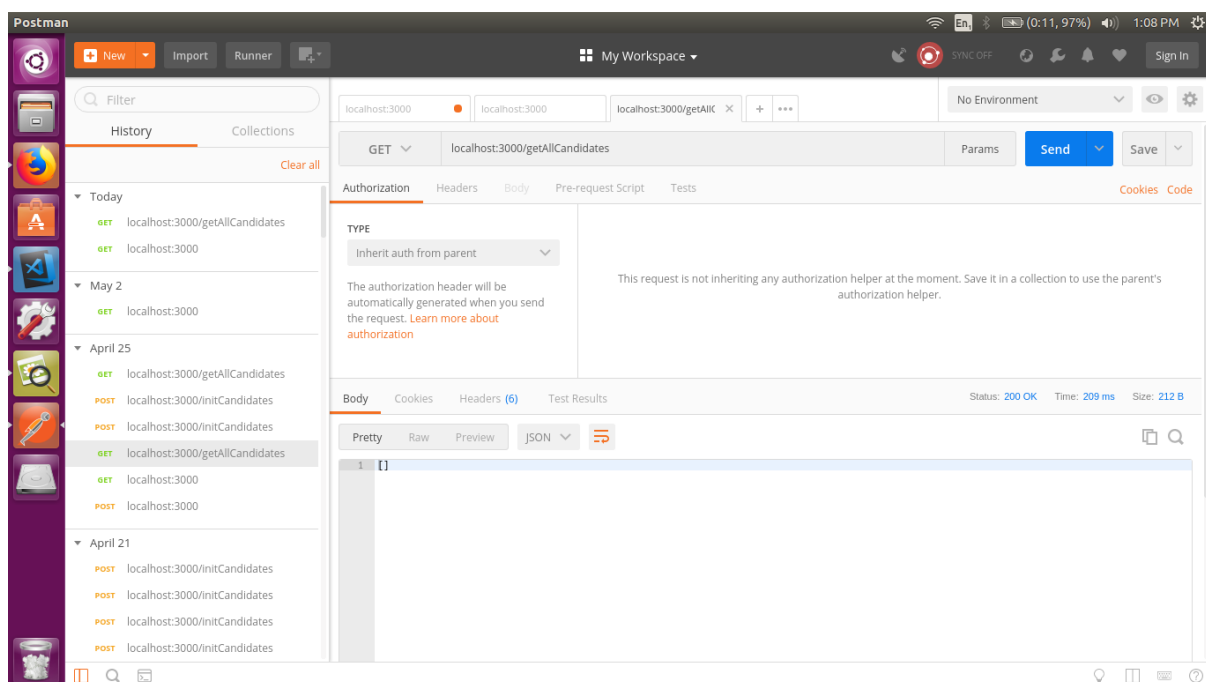


Figure 29 Postman Testing screen of a GET request which generates has no results..

Interface Testing

I tested the interface manually to see if the interface was interacting with the underlying blockchain platform and that data transfer was working as necessary and that there were no unnecessary delays in timing. This testing involved checking the usability of the system and especially the navigational functions as the data was sent to and from the application to the API. I carried out this type of testing using Postman which enable me to see the request that was sent and response that were coming back to ensure that the system was fully functional. This also enables me to check the robustness of it and see how the users will interact with the system.

All the functionality of the web application was tested using the localhost using the port number 3000.

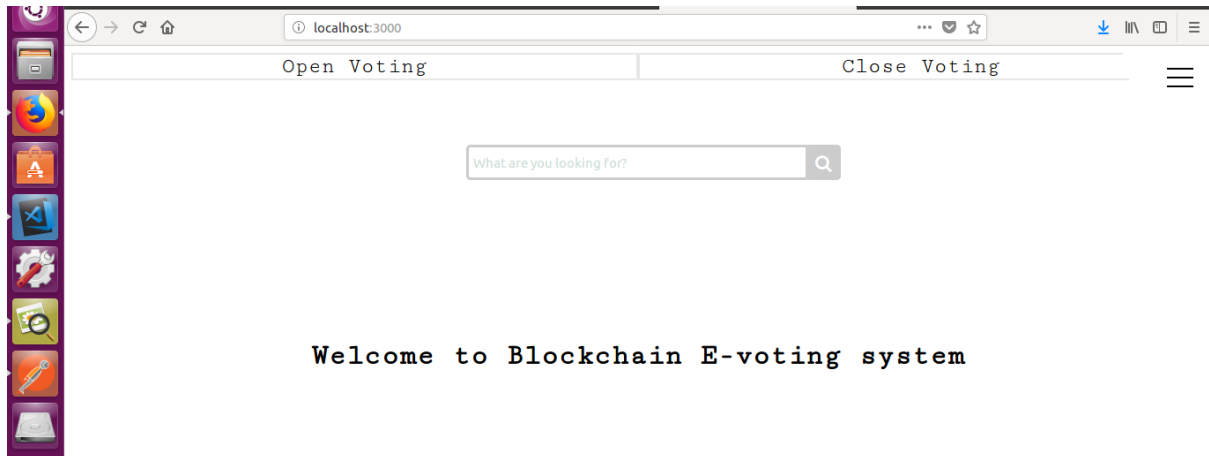


Figure 30 Testing of the web application on the localhost:3000

Evaluation

I wanted to evaluate my project to identify, if I have achieved my objectives that I established in my initial project, to judge the overall development, the outcome of the system and its functionalities. In my Initial set of requirements, I can see that I have achieved most of the required functionalities. I have attempted to incorporate them when implementing a smart contract accordingly that it offers an improved way to vote considering the frauds that have happened in the current e-voting systems. Overall, I think the systems has achieved the functionality it should have within the timeframe.

The system that I have created presents an extensible platform. There are many other features that can be added to extend the platform to be completely safe and secure. I have named a few in the future works section. By adding them I think it will enhance the security of the system.

Nonetheless due to the amount of time, some difficulties in implementing the project and development of the chaincode I have not been able to create all the system that is fully distributed that handles user registration. There are still some limitations and steps that can be taken further to make sure the system is fully secure and robust. In my evaluation I want to analyse and evaluate the limitations and achievements of the project.

Functionality Fulfilment

I have accomplished a system which performs the functional requirements effectively and efficiently, all functionalities were fulfilled and working as expected. The required functionalities that I wanted to implement have been very successful for the proof of concept of e-voting. What the application strives to show is that how users are able to register their vote and how their votes will be counted accurately and reliably.

As I have said, all the transactions are being sent through a channel created by the orderer, and the default name of that is mychannel, so when you have the network connected fauxton API should show up at the port number 5894. This will show a database called mychannel which will responsible for storing the documents. This enables me to do see the state of the database and try and conduct rich queries for the documents that are currently stored in there.

My primary goal was to assess and address issues that the current e-voting platform proposed, moreover, how they could be solved by using Blockchain. While developing the system I discovered that storing the users (voters) on ledger would make it challenging to preserve the individual anonymity. Additionally, I found that by storing the users on the ledger could result in identity fraud so to maintain the systems security and prevent fraud, I made the design choice of not including the users on the blockchain. By following this choice, I have designed the system such that it preserves the anonymity of the user on the distributed ledger so that no one can see who they voted for beside the individual themselves achieving one of the problematic functionalities.

Project Management & Technique

At the beginning as I wanted to follow the agile method development, I found development practises that complemented the agile method which was feature driven development. While Initially I had a very clear outline of what I wanted to achieve in the system and I had defined clear goals and timeline definition in my project. By adhering to the goals that I wanted to achieve it ensured that I was on track with my project and development and aided me in achieving my functionalities. Due to this I had a very clear idea of what the most complex functionalities will be before I started to implement them. I think most of the timelines I had were estimated from preliminary research into the project and Blockchain platform. This gave me estimates about the issues I should be expecting to arise making it better to solve the complexities.

I had a few issues at the beginning of the implementation of the project even though I had allocated enough time to be able to finish implementing that aspect of the project. This delayed my project by several days, so I had to spend additional time working on getting the issues resolved. Throughout the project I had run across many issues which I did not know how to solve. For example, while I began my implementation into a windows platform I had issues with Docker where the firewall of the Laptop would not let docker mount the images which I have shown below to generate the network.

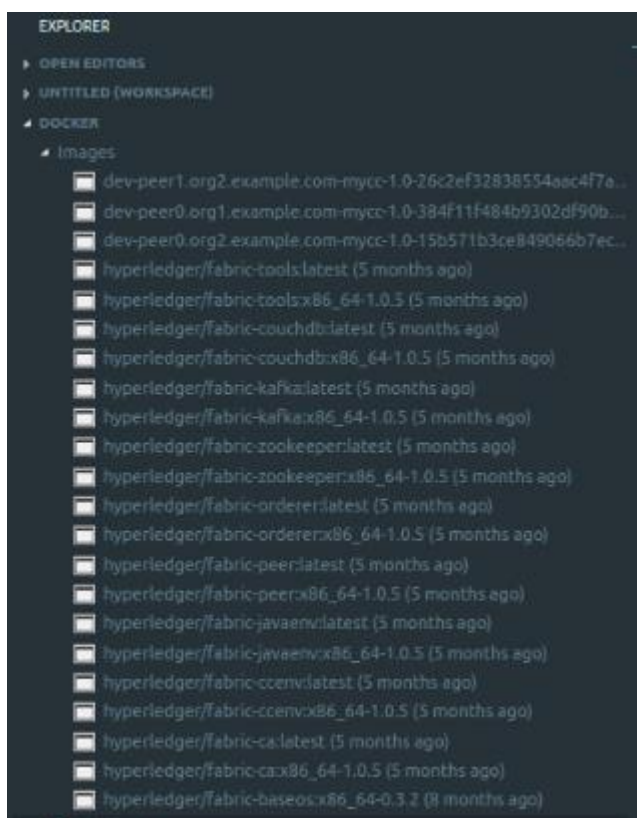


Figure 31 Images needed by docker to generate the Fabric network.

This issue was time consuming and I researched for this issue online but since no one had experienced it I was not able to find any viable solutions. I had to result into speaking to people to see if they had encountered this issue, because of the time restriction It was

suggested that it would be better to change the file system to Linux and to setup my project on there. This is just one example of an issue that delayed me in my project.

I had another issue while I was implementing the chaincode to test if the business logic was correct. Testing the applications logic could be easy, all I had to do was edit the script.sh file to access the correct smart contract so it could be instantiated by the peers. Meanwhile even though the scrip was edited it only allowed me to test the write functionality for the ledger. I had spent quite a while to see why this issue would occur only to understand that a block is created every second and read functions where reading the data before it was written to the ledger. To test the logic and finish off the implementation of the rest of the functionality I had to build a NodeJS app which allowed to test the read functionality instantaneously.

I think this practise became helpful because I could constantly re-iterate over the design process to make the necessary changes. I think this was the most helpful in developing this project. The ideas I wanted to implement became clear while I was progressively developing the smart contract it was ideal to be able to constantly edit the smart contract to make sure it had all the necessary requirements.

It was very beneficial to work and define the communication of the client and the network in NodeJS. It was helpful to find other modules which were compatible and complementary to this framework to build the application and GUI as it became much more simplistic to connect everything together.

Technology Used in Development

The choice of tools and business logic that I used for it were very useful and was the most effective and easiest way to implement the application. Now I will be analysing the Technologies that was used and the results I was able to generate from them to see whether they were effective or not.

GUI

The implementation of the web interface and its functionality were either done through JavaScript , AJAX or pug. Pug was definitely an excellent choice because it's a templating engine for Express JS which allows me to create my application rapidly and to connect it to the API. This was ideal because it aided me to produce the UI rapidly and efficiently. As I am not focusing not creating the most visually appealing application this was the most efficient solution.

By using this templating engine and linking it to JavaScript it allows me to update the information and get the application interacting to the API plus the network with ease. AJAX enables me to display the results without the needing to refreshing the page to display the latest information because it handles auto update.

Chaincode

The core requirements for the system were to be implemented in the chaincode. This shows how a blockchain can be used as an e-voting system to prevent the election ridging and other types of fraud for sensitive matter such as e-voting. In an existing system for e-voting

and I-voting there have been many cases where a malware can be implemented to alter an individual vote to influence the election either way.

This prototype of the system is designed to prevent just that. This system will allow a government official to deploy a chaincode and setup candidates so that on the voting day as users and voters log on to the system they can cast their votes and keep a track of each individual's votes and their choices and gives the government a functionality to count the votes to declare the results to the public. All of these modifications are made in real time, since blockchain holds a history of the records I have extended the functionality to a user's voting history to show an individual their voting history.

To preserve the anonymity of a user and fraud prevention by the user, I have used hash of the individual's username and password to generate a hash. This was problematic to implement properly because if a colon was missing or incorrectly placed it could have created a collision compromising the security of the system. On the other hand, for the candidate the key is the candidate's party because there is only ever one elected candidate for each party this is set as a key so that it can be accessible. By allowing the government official to register a candidate but to not set the vote variable would reduce fraud.

When you view this on fauxton API the transactions are showing you the most updated values. When the Vote is casted the counted value in vote document is set to false as it has not been counted yet. After the election has been closed and votes have been counted, only then it will be set to true this modification will show up on the fauxton. I have designed it so that this way so that if a user does change their vote and chose another candidate that they want to vote for their vote will not be counted twice. The functionality of counting votes is only available after the voting for the users has been closed.

I have set it up this way because once a user has access to the system they can cast as many votes as they want, they will have the ability to check their votes to see if they are happy with their choices. Once the voting has closed their latest vote will be counted. They can also see the history of their votes to see how they have changed their choices as the ledger will generate its history.

To stop anyone from countless adding or editing the candidate the smart contract performs checks to make sure that candidate is not already been setup from that party. To do this the chaincode iterates over all the records for the candidates and checks to see if another document id matches with this one and if they do, it sends an error back saying that a person already exists for it. If there is no match, then it takes your request to the invoke and writes it to the ledger.

This was all done to make sure the system could not be tampered with and to make sure that individuals have less interaction with the system. The only problem I think needs to be assessed is that as a government official has entered the details for setting up a candidate to reduce the possibility of human error.

NodeJS SDK

I discovered this issue that I was not able to enrol the admin while I was implementing. I decide this would be the best course of action to set TLS enable to false. However, if the application was in production for general elections TLS would have to be enabled to ensure that security of the system could not be compromised. Due to the time limitations and the lack of resources I was not able to implement it with the TLS. In addition to that I did confirm with the blockchain developer from Equinity and he reassured me that it should not affect the security of the proof of concept system.

API

To test the application and its success I had to test the REST API and routes established in the API, so I can ensure that correct data is sent and transferred to the API. To test the API, I used Postman to test the transfer of data from postman to NODE SDK, so I could see if the application would interact with the network, this was ideal as it cuts down the testing time and I could test more effectively by saving the test to see whether they passed or failed and the status of the transactions.

Testing

With choosing Feature driven development, I was able to test feature by feature and keep iterating over the design to make the necessary changes need to meet the functionality and to make sure they worked. I had to spend a lot of time manually trying to test it which was not very effective but there aren't way to test them using Unit tests. I think given more time I would have liked to test the system in various different ways to make sure the system is robust and will not break so easily.

Future Work

Overall, I think there are many ways this project can be expanded and improved further to produce a fully functional and commercially viable product that the government can implement. I think there are few features that I have outlined below that would help make the system much more desirable.

Validation

I think it would be very useful to include validations In the Node SDK application when end users are entering data a message could be shown to inform that the action they are waiting for is finished. It will also be helpful to inform them that they have entered incorrect information to help them correct that action. So that they can enter the correct information which then make the record and the votes more correct and help make the system more secure.

Access Level for Users

Currently the system has one user enrolled on it which actually has admin access which means that they have the permission to interact with the blockchain giving all the users in the system a prominent level of access to modify and see all of the transactions. In the future I would like to set permissions for all levels of access such that when a user is enrolled on the system they only have access to certain types of transactions and services in comparisons to the admin. This will enable me to develop a hierarchical systems which allows certain group of individuals certain privileges on the function to increase the systems security this will make it easier to outline the functions and the logic that the individual can access.

Tokens

To enhance the security of the system it would be better that once a user logs on they are given an authorised token to which confirms their identity they would require their authentication details as well to be able to cast a vote. This will ensure that if a user's password and username have been compromised the token would make sure that the user authenticity and this could use to prevent identity fraud and prevent election fraud.

Detailed statistics

To improve and extend the systems analytical ability the blockchain can be used to record the location data so you can identify where each of the votes are being casted from. By doing this you can determine which constituency received the most vote in the county by allowing the users the ability to visualise that in the map. This also lets the government and the public to visualise who and how many people voted In the election.

The statistics that are being generated can also be used to personalise election campaigns for the candidates to improve their current strategies. As each individual vote's they could be question and asked about why they are voting for this individual. Then machine learning algorithms can be applied to decide the factors the general public looks at before voting for

a particular candidate. This information can then be used to personalise the campaign for the candidate's campaign.

Commercial Application

I think the application and the logic of the smart contract can be extended to be used in the company elections where investors and the management have the ability to vote about the issues such as pay rise of the individual or about the management. By incorporating the logic by allowing them to vote not as a whole but in fractions and by adding this up to determine whether the vote was in favour of or in against of the decision of this agenda.

Legislation

The blockchain can be extend so that a candidate when they are registered has the ability to hold and view documents.

By doing this we can allow the candidates to upload their agenda and the legislation they want to implement during their candidacy. In doing this the voting system can be used to vote on the legislation the public wants the candidates to implement. This would enable the candidate view and implement the legislation favoured by the majority of the public and not determine what is in their or anyone else best interest. This will be allowing the government and public to participate in more important legislation increasing the number of referendums on general topics.

These are just some of the features I think can be used to extend the system and the functionality that would make the e-voting system much better.

Conclusion

The main Aim of the project has always been to develop a proof of concept prototype which will be a secure system which will reduce and prevent fraud prevention. I have done this to address the issues that are in the current I-voting and e-voting systems. I have mostly achieved the requirements that I defined to address the issue of election fraud.

To achieve this, I have developed a smart contract which would provide the correct business logic that can be used to develop the secure system. I have achieved this because unlike the I-voting system I have not used a centralised repository such as a server so that there is no obvious vulnerability to the system. By recording transactions and verifying them with the other nodes before the transactions is committed has numerous advantages one being that if at any point one node was to fail another, the data written in the ledger will also be available to view as the other peer nodes on the network retain their own copy of the ledger and the transactions making the system robust.

This also makes the electric voting systems more reliable and secure where the votes could only be accessed by their owners. I had to generate a hash this would ensure that if the voter wanted to check their votes they would have to use their username and password which no one else but them should have access to. Even if an attacker could have access to this information the user can and should be able to view their voting history to ensure that all the votes that are being written to the ledger are their own choices. This was ideal as it becomes easy to identify vote manipulation and to address it as it is an invoke transactions it can be traced and tracked in real time.

Blockchain is a traceable ledger, I used those functionalities to ensure the security of the system, the ledger is also transparent to its contributing member it was very difficult to ensure that an individual's identity is anonymous thus hash had to be used to conceal the user's identity. This was the most effective way which would ensure no one will be able to find each other.

Aside from the security to ensure that the correct data is obtained and written to the ledger I have developed a REST API to ensure that correct data is sent to and from the application this ensures that the correct business logic is called so it ensure that the system can function more effectively and efficiently.

I also wanted to focus a bit on the modularity of the code and the extensibility of the platform, I wanted to focus on this because this will make the process of error debugging and will make it easier to add more features later.

I think blockchain platform could potentially be used to create a voting system which can be integrated with existing systems and if more features where added to ensure the security and to improve the system. I think the application can also be tested a bit better in terms of stress testing and it scalability.

Reflection on learning

Overall this project was a very challenging for me, and while working on this project it has taught me many value skills. One of the major skills I gained was project management where I was working together with Equinity. So initially before I started to research into what could be achieved with this project, I met with the person who has proposed this idea to come up a set of specifications and ideas. Throughout the project I made sure to update them on my progress and what I see the end product becoming. Furthermore, I also kept in touch with my supervisor discussing my progress and any issues I was having in my project.

I have had a lot of support and exposure to resources from Equinity which made developing the project a lot easier for me. As blockchain technology is an emerging topic, the problems I have found were not easily comprehensible nor could they be found online. To solve them I had to do a lot of research to identify and solve problems at hand. I had many issues while establishing my development environment. One of the issues I was having is that on my computer, docker was unable to mount the correct images for me to run the network. So, I was able to go the blockchain developer who proposed an alternative solution which could aid me in solving my problem. To solve this issue, I had to change the operating system to Linux and setup my development again to overcome as Hyperledger was first developed in Linux it had a bit more guidance on how to set things up. I think this was vital for me as it helped me avoid the most common problems that are not available online.

I think everything I have programmed and achieved in this project was a big learning curve for me because I had never used or worked with Hyperledger, Node SDK used for my network endpoint and Express.js which I used to create the API and implement a hashing function to preserve the anonymity of user on the system as I have before mentioned in the background section. Even though this was a learning curve I found the demos more useful than the documentation as It was not explained very well. One average I spent two days' trying to understand to solve the issues that I experienced but while attempting to solve the issue I learnt a lot about why this issue was happening and how to overcome it. One of the things I found most difficult to program was Node SDK where I spent a lot of time trying to initialise the channel and to setup the blockchain. I encountered several issues when I was trying to distribute and modularise my code as each class I was working with in Node SDK was returning promises. Since it was not resolved, the last promise it was leading to the channel not being initialise properly which meant that I could not execute any transactions.

In duration of the project I have gained many major technical skills which could help in the future. As all of these technologies and programming languages such a Go using the Node and Express framework were something that I have never done before. Moreover, I learnt debugging techniques I could use for when developing programs to get a better understanding of the error. While I was developing the ChainCode it is very hard to be able to tell why your ChainCode is not working so for that I would use the Fmt library to print out results to see whether they are correct or not but to view the results I had to make sure to open up the docker log for one of the peer containers to be able to see the results. When I was comfortable with the programming languages and skills I started to modularise the code so that it can be done efficiently.

Learning about the Hyperledger was another skill that I gained to see the use the distributed system, to build the application on top of that. I also got a deeper insight into how consensus is achieved and most importantly how blockchain technology operates. As I was starting to develop the app in Node SDK and trying to configure the network to write data and query data in the blockchain. While configuring the network I came across issues like this because TLS was enable which meant that the admin was not being enrolled correctly. In trying to solve this issue I learnt a lot about the blockchain network to correct the default setting so that I could build my app.

If I must try and do this project again, I think I would spend more time on the design phase of the system which I have not done by doing this I could make the requirements for the system much clearer. So that less time needs to be spent on changing designs or modifying the plan that is originally in place. I would also focus on the testing aspect a bit to try and research ways in testing the ledger and each phase of my application. Preferably I would like to conduct automated testing which already allows me to define tests and build the application according to it reducing the time spent on testing and allowing me to effectively test many more scenarios.

Due to time constraints I have not been able to focus on creating the most visually appealing GUI and to ensure that next time I would allocate my time wisely so that I am able to manage and have time to implement the extra functionality within the system. However, while doing the project I have gained many skills that have improved the existing skills and have given me new technical skills.

Bibliography

- Agarwal, A. (2017, October 07). *Fabric CA setup - client*. Retrieved from Medium: <https://medium.com/mlg-blockchain-consulting/fabric-ca-setup-client-852136f6a63c>
- Berke, A. (2017, March 03). *Technology*. Retrieved from Harvard Business Review : <https://hbr.org/2017/03/how-safe-are-blockchains-it-depends>
- Bezgachev, V. (2017, July 22). *Hyperledger Fabric in practise*. Retrieved from Medium: <https://medium.com/@vitaly.bezgachev/hyperledger-fabric-in-practice-part-1-main-components-and-running-them-locally-aa4b805465fa>
- Boucher, P. (2016, September). *EPRS | European Parliamentary Research Service*. Retrieved from http://www.europarl.europa.eu/RegData/etudes/ATAG/2016/581918/EPRS_ATA%282016%29581918_EN.pdf
- Docker. (2018, January). *What is Docker*. Retrieved from Docker: <https://www.docker.com/whatdocker>.
- Express.js . (2018, April 23). *Using template engines with Express*. Retrieved from Express: <https://expressjs.com/en/guide/using-template-engines.html>
- Google. (2018, April). *About gRPC*. Retrieved from GRPC: <http://www.grpc.io/about/>
- hyperldger Fabric* . (2018, January). Retrieved from Github: <https://github.com/hyperledger/fabric>
- Hyperledger. (2018, April). *Architecture Explained*. Retrieved from Hyperledger Fabric Documentation: <http://hyperledger-fabric.readthedocs.io/en/latest/arch-deep-dive.htm>
- Hyperledger. (2018, February). *Building Your First Network*. Retrieved from Hyperledger Fabric: http://hyperledger-fabric.readthedocs.io/en/latest/build_network.html
- Hyperledger. (2018, April). *CA - Users Guide*. Retrieved from Hyperledger : http://hyperledgerdocs.readthedocs.io/en/latest/ca_setup.html
- Hyperledger. (2018, February). *Chaincode for Developers* . Retrieved from Hyperledger Fabric Docs: <http://hyperledger-fabric.readthedocs.io/en/latest/chaincode4ade.html>
- Kobie, N. (2015, March 30). *Why electronic Voting isn't secure -but may be safe enough*. Retrieved from The Guardian: <https://www.theguardian.com/technology/2015/mar/30/why-electronic-voting-is-not-secure>
- Koven, J. B. (2016, August 30). *Block the Bote: Could Blockchain Technology cybersecure Election?* Retrieved from Forbes : <https://www.forbes.com/sites/realspin/2016/08/30/block-the-vote-could-blockchain-technology-cybersecure-elections/#1228ba572ab3>
- Lambert, L. (2017, May 26). *Cryptocurrencies Blockchain E-voting Voting: A Bastion of Democracy*. Retrieved from markey Mogul: <https://themarketmogul.com/blockchain-e-voting-democracy/>
- Leetaru, k. (2017, June 07). *How Estonia's E-voting system Could be the Future*. Retrieved from Forbes: <https://www.forbes.com/sites/kalevleetaru/2017/06/07/how-estonias-e-voting-system-could-be-the-future/#1b19bc73b950>
- Node Js. (2018, March). *Node Js - Download page*. Retrieved from Node Js: <https://nodejs.org/en/>

- Raja, R. (2017, September 29). *Chaincode on the Go*. Retrieved from Medium :
<https://medium.com/@ramSocializing/chaincode-on-the-go-smart-contracts-on-the-hyperledger-fabric-blockchain-82dd61b3c669>
- Robertson, M. R. (2017, June 13). *Russian Cyber Hacks on Us Electoral Systems Far Wider Than Previously Known*. Retrieved from Bloomberg:
<https://www.bloomberg.com/news/articles/2017-06-13/russian-breach-of-39-states-threatens-future-u-s-elections>
- sigoa, a. o. (2018, March 12). *Express/Node introduction*. Retrieved from Developer Mozilla:
https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction
- Snellinckx, J. (2017, September 22). *Hyperledger fabric & couchDB* . Retrieved from The Ledger:
<https://medium.com/wearetheledger/hyperledger-fabric-couchdb-fantastic-queries-and-where-to-find-them-f8a3aecef767>
- Softjourn. (2018, January 29). *Fabric Chaincode vs Burrow EVM*. Retrieved from Softjourn:
<https://softjourn.com/blog/fabric-chaincode-vs-burrow-evm/>
- Strukhoff, R. (2016, NOVEMBER 14). *Hyperledger Fabric* . Retrieved from Altoros:
<https://www.altoros.com/blog/how-hyperledger-fabric-delivers-security-to-enterprise-blockchain/>
- W.Ambler, S. (2018, May). *Feature Driven Development (FDD) and Agile Modeling*. Retrieved from Agile modeling: <http://agilemodeling.com/essays/fdd.htm>
- w3schools.com. (2018, May). *AJAX Introduction*. Retrieved from w3schools.com:
https://www.w3schools.com/xml/ajax_intro.asp
- Walker, D. W. (2017-2018, February). *Bitcoin*. Retrieved from Learning Central :
https://learningcentral.cf.ac.uk/bbcswebdav/pid-4534148-dt-content-rid-9473610_2/courses/1718-CM3202/Bitcoin.pdf

Appendix

Docker-compose-e2e.yaml

```
version: '2'

networks:
  byfn:
services:
  ca0:
    image: hyperledger/fabric-ca
    environment:
      - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
      - FABRIC_CA_SERVER_CA_NAME=ca-org1
      - FABRIC_CA_SERVER_TLS_ENABLED=false
      - FABRIC_CA_SERVER_TLS_CERTFILE=/etc/hyperledger/fabric-ca-server-
config/ca.org1.example.com-cert.pem
      - FABRIC_CA_SERVER_TLS_KEYFILE=/etc/hyperledger/fabric-ca-server-
config/CA1_PRIVATE_KEY
    ports:
      - "7054:7054"
    command: sh -c 'fabric-ca-server start --ca.certfile
/etc/hyperledger/fabric-ca-server-config/ca.org1.example.com-cert.pem --
ca.keyfile /etc/hyperledger/fabric-ca-server-config/CA1_PRIVATE_KEY -b
admin:adminpw -d'
    volumes:
      - ./crypto-
config/peerOrganizations/org1.example.com/ca:/etc/hyperledger/fabric-ca-
server-config
      container_name: ca_peerOrg1
    networks:
      - byfn

  ca1:
    image: hyperledger/fabric-ca
    environment:
      - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
      - FABRIC_CA_SERVER_CA_NAME=ca-org2
      - FABRIC_CA_SERVER_TLS_ENABLED=false
      - FABRIC_CA_SERVER_TLS_CERTFILE=/etc/hyperledger/fabric-ca-server-
config/ca.org2.example.com-cert.pem
      - FABRIC_CA_SERVER_TLS_KEYFILE=/etc/hyperledger/fabric-ca-server-
config/CA2_PRIVATE_KEY
    ports:
      - "8054:7054"
    command: sh -c 'fabric-ca-server start --ca.certfile
/etc/hyperledger/fabric-ca-server-config/ca.org2.example.com-cert.pem --
ca.keyfile /etc/hyperledger/fabric-ca-server-config/CA2_PRIVATE_KEY -b
admin:adminpw -d'
```

```

    volumes:
      - ./crypto-
config/peerOrganizations/org2.example.com/ca/:/etc/hyperledger/fabric-ca-
server-config
    container_name: ca_peerOrg2
    networks:
      - byfn

orderer.example.com:
  extends:
    file: base/docker-compose-base.yaml
    service: orderer.example.com
  container_name: orderer.example.com
  networks:
    - byfn

peer0.org1.example.com:
  container_name: peer0.org1.example.com
  extends:
    file: base/docker-compose-base.yaml
    service: peer0.org1.example.com
  networks:
    - byfn

peer1.org1.example.com:
  container_name: peer1.org1.example.com
  extends:
    file: base/docker-compose-base.yaml
    service: peer1.org1.example.com
  networks:
    - byfn

peer0.org2.example.com:
  container_name: peer0.org2.example.com
  extends:
    file: base/docker-compose-base.yaml
    service: peer0.org2.example.com
  networks:
    - byfn

peer1.org2.example.com:
  container_name: peer1.org2.example.com
  extends:
    file: base/docker-compose-base.yaml
    service: peer1.org2.example.com
  networks:
    - byfn

```

```

cli:
  container_name: cli
  image: hyperledger/fabric-tools
  tty: true
  environment:
    - GOPATH=/opt/gopath
    - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
    - CORE_LOGGING_LEVEL=DEBUG
    - CORE_PEER_ID=cli
    - CORE_PEER_ADDRESS=peer0.org1.example.com:7051
    - CORE_PEER_LOCALMSPID=Org1MSP
    - CORE_PEER_TLS_ENABLED=false
    -
    CORE_PEER_TLS_CERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.crt
    -
    CORE_PEER_TLS_KEY_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.key
    -
    CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
    -
    CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
    working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
    command: /bin/bash -c './scripts/script.sh ${CHANNEL_NAME} ${DELAY}; sleep 999999'
  volumes:
    - /var/run:/host/var/run/
    -
    ../../chaincode:/opt/gopath/src/github.com/hyperledger/fabric/examples/chaincode/go
    - ./crypto-
    config:/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/
    -
    ./scripts:/opt/gopath/src/github.com/hyperledger/fabric/peer/scripts/
    - ./channel-
    artifacts:/opt/gopath/src/github.com/hyperledger/fabric/peer/channel-artifacts
  depends_on:
    - orderer.example.com
    - peer0.org1.example.com
    - peer1.org1.example.com
    - peer0.org2.example.com
    - peer1.org2.example.com
  networks:

```

- byfn

Chaincode Main function

```
func main() {
    err := shim.Start(new(SimpleChaincode))
    if err != nil {
        fmt.Println("Error starting chaincode: " + err.Error())
    }
}

// Init initialises the states of the application.
func (t *SimpleChaincode) Init(stub shim.ChaincodeStubInterface) pb.Response {
    var err error

    err = stub.PutState("isVoting", []byte("false"))
    if err != nil {
        return shim.Error(err.Error())
    }

    return shim.Success(nil)
}

// Invoke calls other functions
func (t *SimpleChaincode) Invoke(stub shim.ChaincodeStubInterface) pb.Response {
    function, args := stub.GetFunctionAndParameters()
    fmt.Println("invoke is running " + function)

    // Handle different functions
    if function == "openVoting" { // open the voting process
        return t.openVoting(stub)
    } else if function == "closeVoting" { //close the voting process
        return t.closeVoting(stub)
    } else if function == "initCandidates" { //create all the candidates
        return t.initCandidates(stub, args)
    } else if function == "castVote" { //cast the vote for the
        return t.castVote(stub, args)
    } else if function == "readData" { //read a voters
        return t.readData(stub, args)
    } else if function == "queryVotesByCandidates" { //find votes for a
particular candidates
        return t.queryVotesByCandidates(stub, args)
    } else if function == "getAllCandidates" { //find votes for a particular
candidates
        return t.getAllCandidates(stub, args)
    }
}
```



```

    } else if function == "invokeCountingVotes" { //Count the votes all the
candidates
        return t.invokeCountingVotes(stub, args)
    } else if function == "getHistoryVote" { //get history of voter so see
vote modification
        return t.getHistoryVote(stub, args)
    } else if function == "queryByString" { //get the result of the query by
string
        return t.queryByString(stub, args)
    } else if function == "checkVote" { //get the result of the query by
string
        return t.checkVote(stub, args)
    } else if function == "" {
        return shim.Error("No function name given")
    } else if function == "test" {
        return shim.Success([]byte("Welcome!"))
    }
}

fmt.Println("invoke did not find func: " + function) //error
return shim.Error("Received unknown function invocation")
}

```

Chiancode query.

```

// =====
// queryByString - send a queryString to query any documents available
// =====
func (t *SimpleChaincode) queryByString(stub shim.ChaincodeStubInterface, args
[]string) pb.Response {

    // 0
    // "queryString"
    if len(args) < 1 {
        return shim.Error("Incorrect number of arguments. Expecting 1")
    }

    queryString := args[0]

    queryResults, err := getQueryResultForQueryString(stub, queryString)
    if err != nil {
        return shim.Error(err.Error())
    }
    return shim.Success(queryResults)
}

```

Init.js file which initialises the network and enrolls the users.

```

module.exports = () => {

```

```

var Client = require('fabric-client');
var CA_Client = require('fabric-ca-client/lib/FabricCAClientImpl.js');
var Channel = require('fabric-client/lib/Channel.js');
var User = require('fabric-client/lib/User.js');
var path = require('path');

var store_path = path.join(__dirname, 'key-store');
console.log(' Store path:'+store_path);

var client = new Client();
var channel = new Channel('mychannel', client);
var ca_client = null;
var user;

return new Promise( function(resolve, reject) {
    Client.newDefaultKeyValueStore({ path: store_path
    }).then((state_store) => {
        client.setStateStore(state_store);

        ca_client = new CA_Client('http://localhost:7054');
        user = new User('admin', client);
        user.setCryptoSuite(ca_client.cryptoSuite);

        return ca_client.enroll({
            enrollmentID: 'admin',
            enrollmentSecret: 'adminpw'

        }).then((enrollment) => {
            console.log('Successfully enrolled admin user "admin"');
            return user.setEnrollment(enrollment.key,
enrollment.certificate, 'Org1MSP');

        }).then(() => {
            return client.setUserContext(user);

        }).catch((err) => {

            console.error('Failed to enroll and persist admin. Error: ' +
err.stack ? err.stack : err);
            throw new Error('Failed to enroll admin');
        });

    }).then((user) => {
        var peer = client.newPeer('grpc://localhost:7051');
        var order = client.newOrderer('grpc://localhost:7050');
        channel.addPeer(peer);
    });

```

```

        channel.addOrderer(order);
        return channel.initialize();

    }).then(() => {
        resolve([client, channel])
    }).catch((err) => {

        console.log(err);
    });
});

}

Invoke.js to execute the transaction against the ledger.

module.exports = (channel, client, request) => {
    var util = require('util');
    var return_status = {};

    return new Promise((resolve, reject) => {

        return channel.sendTransactionProposal(request)

        .then((results) =>{
            var proposalResponses = results[0];
            var proposal = results[1];
            var proposalHeader = results[2]
            let isProposalGood = false;
            if (proposalResponses && proposalResponses[0].response &&
proposalResponses[0].response.status === 200) {
                isProposalGood = true;
                console.log('Transaction proposal was good');
            } else {
                console.error('Transaction proposal was bad');
            }

            if(isProposalGood == true){

                var request = {
                    proposalResponses: proposalResponses,
                    proposal: proposal,
                    header: proposalHeader
                };

                var transaction_id_string = tx_id.getTransactionID();
                var promises = [];
                var sendPromise = channel.sendTransaction(request);
                promises.push(sendPromise);

                let event_hub = client.newEventHub();

```

```

        event_hub.setPeerAddr('grpc://localhost:7053');

        let txPromise = new Promise((resolve, reject) => {
            let handle = setTimeout(() => {
                reject(new Error('Transaction did not complete
within 30 seconds'));
            }, 3000);
            event_hub.connect();

            event_hub.registerTxEvent(transaction_id_string, (tx,
code) => {

                clearTimeout(handle);
                event_hub.unregisterTxEvent(transaction_id_string);
                event_hub.disconnect();

                return_status = {event_status : code, tx_id :
transaction_id_string};
                if (code !== 'VALID') {
                    console.error('The transaction was invalid, code =
' + code);

                    reject(new Error('Problem with the tranaction,
event status ::'+code));
                } else {
                    console.log('The transaction has been committed on
peer ' + event_hub._ep._endpoint.addr);
                    resolve();
                }
            }, (err) => {
                reject(new Error('There was a problem with the
eventhub ::'+err));
            });
        });
        promises.push(txPromise);
        return Promise.all(promises);
    } else {
        //console.error('Failed to send Proposal or receive valid
response. Response null or status is not 200. exiting...');
        Promise.reject( new Error('Failed to send Proposal or receive
valid response. Response null or status is not 200. exiting...'));
    }
    }).catch((err) => {
        throw reject(return_status)
    })

    }).then(() => {
        return resolve(return_status);
    })
}

```

Query.js to read data from the ledger.

```
module.exports = (channel, request) => {  
  return new Promise((resolve, reject) => {  
    // send the query proposal to the peer  
    channel.queryByChaincode(request).then((query_responses) => {  
      return resolve(JSON.parse(query_responses));  
    }).catch((err) => {  
      throw reject("Error");  
    })  
  });  
}
```