

CM3203 – One Semester Project (40 Credits)
May 2017

The Use of Blockchain Technology in Smart Contracts



Student: Deborah Khoo (C1461872)
Supervisor: Professor David W Walker
Moderator: Professor Paul L Rosin

Final Report

Table of Contents

Acknowledgement.....	4
Abstract.....	5
1. Introduction.....	6
1.1 Blockchain Technology.....	6
1.2 Research Business Impact of Blockchain Technology in Trade Finance.....	7
2. Background.....	8
2.1 Trade Credit and Factoring.....	8
2.2 Credit and Risk Factors.....	8
2.3 Existing System.....	9
2.4 Use of Blockchain Technology and Smart Contracts Prototype.....	9
2.5 Aim.....	9
3. Design & Specification.....	11
3.1 Hyperledger Fabric.....	11
3.2 System Architecture.....	11
3.3 Trade Finance System.....	13
3.4 Trade Finance System Design and Specification.....	13
3.4.1 Home Page.....	13
3.4.2 Home Page for Business Users.....	13
3.4.2.1 Add Customer.....	14
3.4.2.2 Add Invoice.....	15
3.4.2.3 Sales Invoices.....	15
3.4.3 Home Page for Customer Users.....	16
3.4.3.1 Verify Invoice.....	17
3.4.3.2 Pay Invoice.....	17
3.4.3.3 Invoices History.....	18
3.4.4 Home Page for Bank User.....	19
3.4.4.1 Finance Business.....	19
3.4.4.2 View All Invoices.....	20
3.4.5 Invoice Status.....	21
4. Implementation.....	22
4.1 Setting Up Network for Development.....	22
4.2 Writing Chaincode.....	22
4.2.1 Dependencies.....	23
4.2.2 Chaincode Interface.....	23
4.2.2.1 Init Function.....	23
4.2.2.2 Invoke Function.....	24
4.2.2.3 Query Function.....	24
4.2.2.4 Main Function.....	24
4.2.2.5 v0.6 to v1.0 Hyperledger Fabric.....	24
4.3 HFC SDK.....	25
4.4 NodeJS Application.....	26
5. Results and Evaluation.....	28
6. Future Work.....	32
7. Conclusion.....	32
8. Reflection on Learning.....	34

9. Appendix.....	36
9.1 Code	36
9.1.1 <i>Docker-Compose.yaml</i>	36
9.1.2 <i>Chaincode Query()</i>	38
9.1.3 <i>Chaincode Init()</i>	39
9.1.4 <i>Chaincode Invoke()</i>	40
9.1.5 <i>Main()</i>	40
9.1.6 <i>SDK Init</i>	40
9.1.7 <i>SDK Query</i>	42
9.1.8 <i>SDK Invoke</i>	43
9.2 Feedback from Equiniti	46
Glossary.....	47
Table of Abbreviations.....	47
Works Cited	48

Figure 1.1 Blockchain. A blockchain is a linked list that is built from hash pointers	6
Figure 1.2 Tamper-evident log. If an adversary modifies data anywhere in the blockchain, it will result in the hash pointer in the following block being incorrect.....	7
Figure 3.1 Web Application Interacting with Hyperledger Fabric [14].....	11
Figure 3.2 Hyperledger Network System Architecture	12
Figure 5.1 State of the blockchain when first initialised. Each doc represents a block in the blockchain	28
Figure 5.2 Blocks in the blockchain represented by CouchDB	28
Figure 5.3 Version update of transactions in blockchain	29
Figure 5.4 Message returned by chaincode when the same customer is added twice	29
Figure 5.5 Message returned by chaincode when adding the same invoice twice	30

Acknowledgement

This project may not have been possible without the kind support and help of many individuals and Equiniti. I would like to extend my sincere gratitude to them.

Firstly, I would like to express my deepest gratitude to my personal tutor and supervisor – Professor David Walker, for providing guidance and valuable advice throughout the year and for this project.

Secondly, I would like to express my gratitude to Equiniti, for their guidance and constant supervision as well as for providing necessary information to complete the project.

Last and not least, I would like to express my special gratitude towards my friends and family, for providing the best kind of support, especially my dad and sister Tabitha.

Abstract

Since the introduction of blockchain technology in Bitcoin, the application of distributed ledger has been a topic of discussion for commercialisation. Organisations are interested in demonstrating how blockchain technology could potentially improve existing systems without the need of an intermediary or centralised party.

This paper will discuss the basis of blockchains and distributed ledger. The objective is to provide a solution to the problem for the existing system in trade finance, and more specifically, in credit factoring. The original use case for the proof of concept in trade finance was by Equiniti, Risk Factor Solutions.

The use of smart contracts in blockchain technology can automatically enforce protocols on transactions to the distributed ledger without the need of an intermediary. The implementation of the trade finance system uses Hyperledger Fabric, an open source private blockchain by IBM.

The results of the prototype show how blockchain technology in smart contracts can improve the existing system in trade finance to prevent fraud and error. The trade finance system demonstrates the immutable and traceability of transactions in a distributed ledger. Additionally, this research examines the use of how blockchain technology could potentially change a business model or process with a distributed commerce.

1. Introduction

1.1 Blockchain Technology

Blockchain technology has gained considerable interest since the introduction of the decentralised cryptocurrency, Bitcoin in 2009 [1]. Aside from being a payment mechanism "native to the internet", the underlying blockchain technology solves the problem of the transfer of value between users, without relying on a third party. The basis of a blockchain is a shared ledger, secured and maintained between a set of peer-to-peer network operators (or miners) running a consensus protocol [2]. Through a consensus network, the ledger may guarantee more consistency and maintain a continuous growing list of transactions. Blockchain technology is essentially a distributed database, touted as a way to store and transact everything from property records to certificates for arts and jewellery [3].

Transactions recorded in a blockchain exist in a state of both anonymity and traceability. Real identities are not required to use the system and transactions on the blockchain are on an immutable open ledger [3]. Transactions in blockchains are a link-list of blocks built from hash-pointers to provide a tamper-evident log as shown in Figure 1.1. These blocks create an open ledger of transactions that are shared and transparent to all participants in the blockchain network. A hash-pointer is a pointer to where data is stored together with a cryptographic hash of the value [2]. This design makes it resistant to alteration of transactions retroactively.

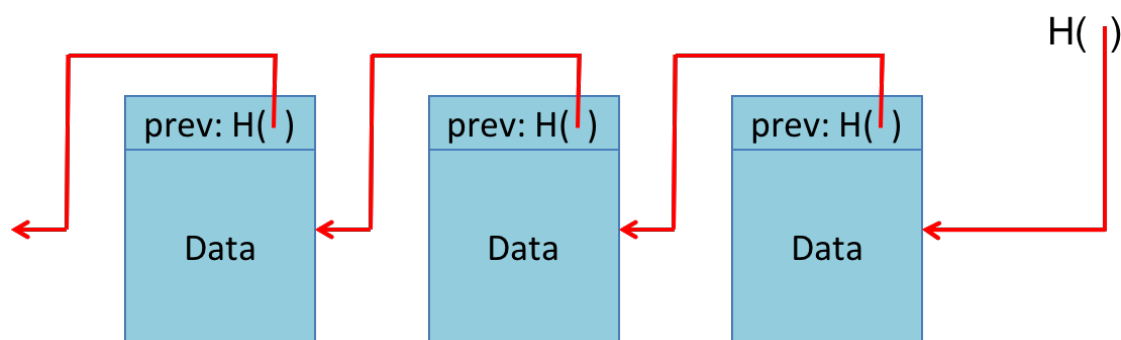


Figure 1.1 Blockchain. A blockchain is a linked list that is built from hash pointers

By comparing the hash-pointer of the next block, changes made to the data of a block can be detected. If an adversary modifies a data anywhere in the blockchain, it will result in the hash-pointer in the following block being incorrect (refer to Figure 1.2). It would be necessary to change all the hash-pointers in the subsequent blocks to cover up tampering. Storing the head of the list to make sure it is tamper-free ensures the list is tamper-evident [2].

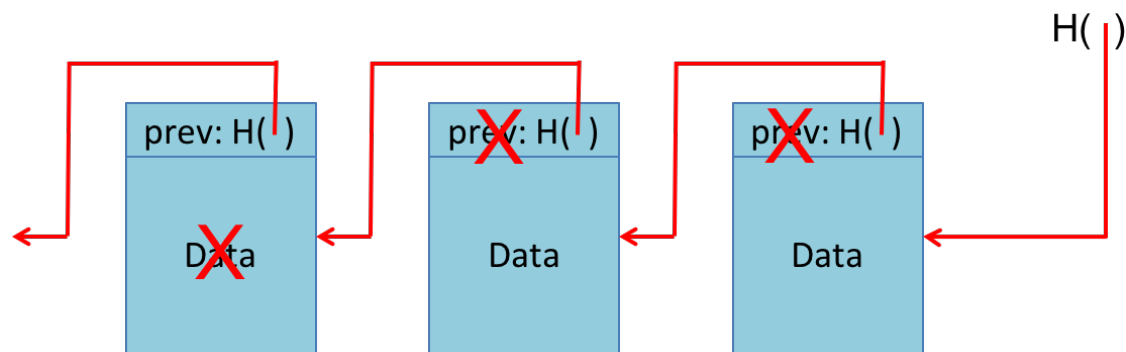


Figure 1.2 Tamper-evident log. If an adversary modifies data anywhere in the blockchain, it will result in the hash pointer in the following block being incorrect

A prominent use of blockchains is to enable smart contracts. A smart contract is a program that runs on a blockchain and is automatically executed by the consensus protocol. A contract can encode any set of rules, such as executing payments when payment is due [4].

The aim of this project is to implement a blockchain prototype in smart contracts and understand its business impact. A wide range of applications can implement smart contracts. Keeping files synced with a traditional database can be a highly complex problem, particularly when many users and systems are simultaneously performing updates. By using blockchain technology, the means of managing files could be simplified by providing a consensus across a peer-to-peer network and keeping files in sync in real time. Physical asset management is one of the most prevalent uses of blockchain to date, with several industries (namely the diamond industry) already using it [5].

If properly utilised, aside from its immutable records of transactions and consensus protocol, smart contract technology could enable decentralised commerce. The blockchain network could potentially create various sorts of markets without intermediaries controlling them [6].

1.2 Research Business Impact of Blockchain Technology in Trade Finance

Equiniti is a financial service company that has an interest in commercially exploiting blockchain technology. Trade finance was a use case chosen from a list of proposals given by Equiniti.

With 80% to 90% of world trade relying on trade finance, mostly of a short-term nature [7], a system that involves paper trails for transactions leave businesses vulnerable to error and fraud.

If smart contracts are successfully carried out, it could provide transparency between parties involved in trade finance as well as real-time verification of transactions. The use of blockchain technology will also potentially reduce cost as records of transactions will not need to be maintained manually. To fully understand the process workflow and problems of the existing system, meetings with Equiniti were arranged, throughout the project. Doing so allowed flexibility and space for modification during the process of implementing the smart contract prototype. In completing the project, the design specification of the prototype was sent and validated by Equiniti.

2. Background

When putting blockchain technology into practice, it is important to understand the difference between public and private blockchains. Consensus in a blockchain network is carried out by nodes in the network confirming the record of previously verified transactions, and by which verifies new transactions. In a public blockchain, such as bitcoin, anyone is able to read or write transactions, and no user is implicitly trusted to verify transactions. Participants verify transactions by committing software and hardware resources to solving problems. The user who reaches the solution first is rewarded [8].

In private blockchains, operators have control over who can read the state of the ledger, who can add transactions, and who can verify transactions. Consensus in private blockchains are achieved by communication between nodes. Each node maintains a copy of the ledger and informs the other nodes of the new information. The applications for private blockchains allow multiple parties who wish to participate simultaneously but do not fully trust one another [8].

2.1 Trade Credit and Factoring

In trade finance, a company may issue invoices to customers on credit terms. For many businesses, trade credit is an essential tool for financial growth as it attracts more customers through its credit offer of 'buy now, pay later' [9]. A business may trade its invoice with a bank or financial institution for quick cash to facilitate other transactions; this is known as credit factoring. The factor, a bank or financial institution, is the funding source that agrees to pay the business the value of the invoice less a discount for commission and fees. Factoring allows a business to receive immediate capital based on the future income attributed to a particular amount due on an account receivable or business invoice [10]. The factor advances most of the invoiced amount to the company immediately as well as the balance upon receipt of funds from the invoiced party [10].

2.2 Credit and Risk Factors

There are a few credit and risk factors that need to be taken into consideration when a factoring company lends money to a business through credit factoring. In factoring, instead of lending against a tangible asset, an order book is given. The risk associated with credit factoring scales as the business scales. The interest rate is dependent on the risks associated with the trading as well as the trustworthiness and reliability of the customer involved in the business transaction. On condition that there are no unnecessary complications in the order and that the quality of the goods is satisfactory, the business would be able to obtain funding.

A risk that factoring companies may face is receiving fraudulent data from businesses that are trying to obtain funds. Businesses could be producing invoices without distributing goods. As such, unless a careful follow-up is carried out, the factoring company may not receive accurate information regarding the transactions. An analysis of the data is, therefore, necessary to look for patterns indicating that a business is behaving fraudulently. Based on such analysis, the factoring company can limit any risk involved by advancing only a certain amount of the invoiced amount to the business. This, however, does not eliminate all risks since the customer may never make payment [11].

2.3 Existing System

The existing system involves paper trails and emails moving between different parties in trade finance, that is the business, customer and factoring company. It begins when a customer raises an order. The business would then send a generated invoice and dispatch the goods to the customer. Besides that, the business will have to manually notify the factoring company, the invoices for credit factoring, for the factor to release the fund. For example, a business may issue 1000 invoices to its customer and send a copy of the invoices as well as the sum of all the invoices to the factoring company. The factoring company will then release the funds upon receiving the invoices and often within 24-48 hours. As notifications usually occur daily and are done in bulk figure, many of the invoices would be left unverified. Releasing the funds as soon as possible is nevertheless necessary. Otherwise, the business might go out of business if it does not receive the necessary money for production and sales at the right time.

2.4 Use of Blockchain Technology and Smart Contracts Prototype

A blockchain could allow for real-time settlement between the business, customer, and factoring company. When a customer raises a purchase order, the business despatches the goods, and an invoice will be issued, just like in the existing system. The improvement is that the invoice transaction is added to the blockchain and can be tracked by all parties immediately. This real-time settlement can therefore help to reduce the risk of fraud data. As and when the business has created the invoice and sent it to the customer, the customer can both receive and verify the invoice in the blockchain. In doing so, the factoring company that is funding the working capital of the business and acting as a third party will receive visibility of the transaction and verification.

In the existing system, when a business produces an invoice, the factoring company may have a difficult time determining if the invoice is part of a real transaction or if it is double financed. Through the system, as and when the customer validates that the product or service is received, the factoring company may obtain immediate recognition that the invoice is genuine. Furthermore, if all factoring companies use the blockchain, the factoring company will be able to check that an invoice is not double financed by another lender. Once the customer validates an invoice, the factoring company may approve and release the funds to the business automatically.

When a payment is due, the customer will be required to make the payment to the factoring company. When payment has been made, the customer may update the transaction to notify the business and factoring company. While the factoring company does not put in any data into the blockchain, the factoring company will be able to monitor the transactions through the blockchain asset and obtain affirmation.

2.5 Aim

The aim of the project is to develop a system prototype that uses blockchain technology in smart contracts to improve the existing system of credit factoring problem in a real business.

The prototype will demonstrate the concept of how blockchain technology and smart contracts can improve the existing system in the business and reduce error/fraud. The main functionalities of the prototype include adding and verifying transactions to the blockchain;

viewing and tracking real-time verification on transactions; and providing transparency between businesses.

3. Design & Specification

The goal for this project's prototype is to provide a solution to the problem of fraud and error that exists in the current system for credit factoring. The prototype system will allow banks and finance institution to have real-time tracking of businesses, customers and invoices using blockchain technology while maintaining and automating protocols on transactions with smart contracts.

A blockchain network consists of different nodes, specifically client and peer nodes. The client node is a system that allows different functionalities to be available to end users. End users will be able to interact with the peer nodes using the system and add transactions. A state, which is a sequence record of all transactions made in the open ledger, is kept by each peer node. When updates are made to the state using the client node, the smart contract will enforce predefined rules on the transactions.

3.1 Hyperledger Fabric

Existing blockchain technologies that are popular include Bitcoin, Ethereum, and Hyperledger Fabric. Open source blockchains, such as Bitcoin and Ethereum, focus on public chains. With public chains, implementing cryptocurrency is necessary to fund mining and participation in consensus [12]. Hyperledger creates a private network that allows users to define membership role and access rights to users within the business network [13].

In Hyperledger, smart contracts are called chaincode and this is used to enforce protocols on transactions. A web application for the trade finance system written in NodeJS is then able to interact with the blockchain network using the Hyperledger Fabric Client (HFC) SDK.

3.2 System Architecture

Blockchain technology is a distributed system consisting of many nodes communicating with each other. There can be multiple types of nodes that run on the same physical server depending on how the nodes are grouped into trust domains and associated to the logical entities that control them.

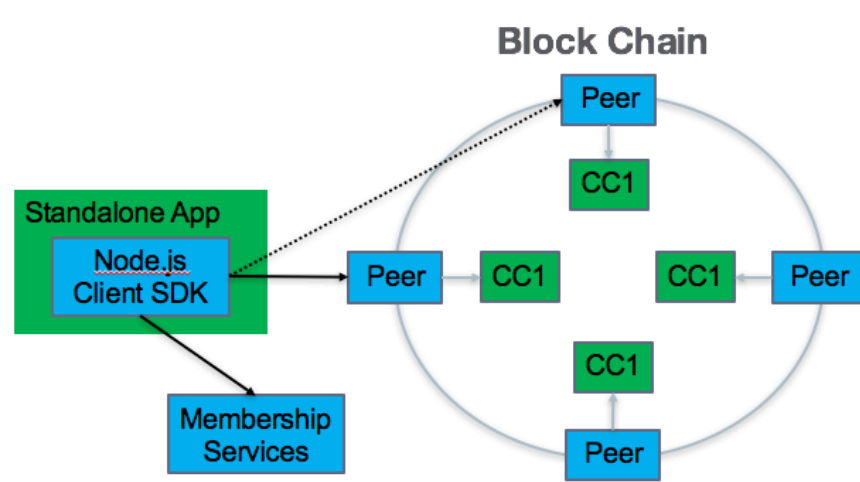


Figure 3.1 Web Application Interacting with Hyperledger Fabric [14]

The figure above represents the interaction in Hyperledger v0.6, before a much more stable version, Alpha v1.0, was released. The standalone app is the client node that represents the entity that acts on behalf of an end user. The client node is a NodeJS web application and which uses the HFC SDK to enrol and register users using membership services. The membership services will allow registered users to connect to the Hyperledger network and invoke transactions to peer nodes. The membership services can also issuance enrolment (ECerts) and transaction (TCerts) certificates. This is important as it provides both anonymity and non-repudiation when transacting on a Hyperledger Fabric blockchain.

In Hyperledger v1.0, there is the additional orderer node. The orderers form the ordering service, a service that provides consensus. The client node communicates with both the peers and ordering service. The peer node commits transactions from the ordering service and maintains the state and a copy of the ledger. The ordering service provides a shared communication channel to clients and peers and offers a broadcast service for messages containing transactions [15].

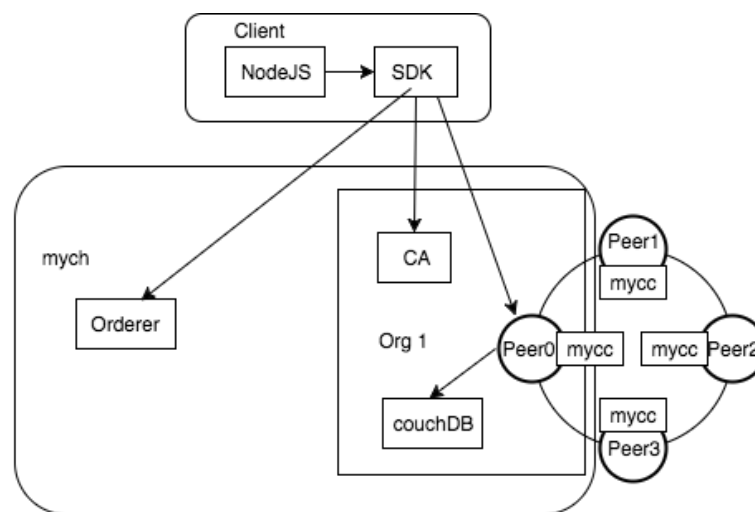


Figure 3.2 Hyperledger Network System Architecture

In the diagram above, each peer holds a copy of the state and ledger data after the chaincode “mycc” is instantiated. The chaincode is the central element as transactions are operations invoked on the chaincode. Transactions have to be “endorsed”, and only endorsed transactions may be committed and have an effect on the state. The ledger’s state data are represented as key values. In addition, Peer0 in Org1 uses CouchDB as the state database. CouchDB allows the same chaincode functions; however, there is the added ability to perform rich and complex queries against the state database [16].

The client is a NodeJS web application which uses the HFC SDK to connect to the Orderer, CA and Peer0. In v1.0 the membership service is known as CA, which stands for Certificate Authority. The CA registers identities or connects to the LDAP as user registers.

Org1 represents the peer and CA that are associated with it and is connected to the channel “mych” that is created by the orderer. Clients connected to the channel may broadcast messages on the channel. The broadcast messages are delivered to all peers connected, in this case, only Peer0.

3.3 Trade Finance System

The prototype of the trade finance system will be developed using NodeJS [17]. The system will support different user roles such as a bank, business and customer users. Not all functions, however, will be made available for every user.

A bank user will be able to register and enrol business users. The process of the prototype would consider that the bank and business users have negotiated the terms of the advance on payments and fees of the factoring facility on invoices, as would normally happen with the existing system.

Business users are able to add customer users as well as an invoice transaction to the blockchain for credit factoring in the system. When adding an invoice, the business will be able to select the customer receiving the invoice. The system will be able to perform checks using smart contracts to ensure that the invoice has not been added before so that there are no duplicate copies. At the same time, the customer will receive a digital copy and will be able to make immediate verification of the invoices. The customers will have a list of invoices that need verification and a history record of received invoices.

Since customers may receive invoices and make verification through the system, it eliminates the need for banks to approach and contact customers to ensure that invoices are valid and to confirm payments. The bank user may also approve invoices for the financing of the verified invoices. During the lifecycle of an invoice, as the invoice passes between users, the bank user will have a real-time tracking and history record of both paid and unpaid invoices by customers.

The status of an invoice will differ at different times during its lifecycle as it is passed between users and is updated. At the start, when an invoice is added to the system, its status will be set as "Pending". While waiting for the customer, the status will be updated to "Verify", then to be "Approved" by the bank, before being confirmed "Paid" by the customer.

The full design specification that was sent to Equiniti, to make sure the functionalities of the system fit the use case, is provided in section 3.4.

3.4 Trade Finance System Design and Specification

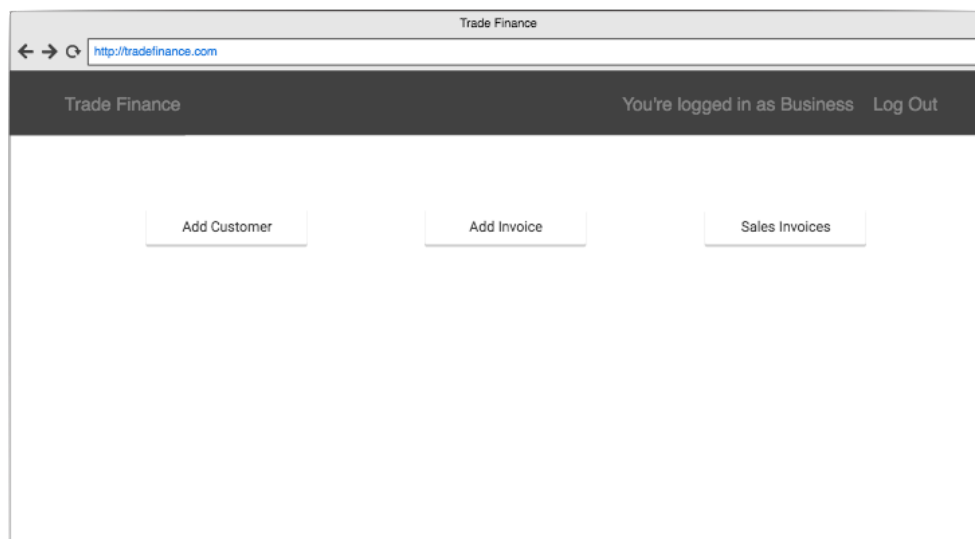
3.4.1 Home Page

When users log into the system, the "Home Page" will be the first page that will be displayed. The name of the currently logged in user will be shown up the top on all pages of the system. Depending on the user roles, different navigation buttons will be displayed.

3.4.2 Home Page for Business Users

Business users will be able to navigate to the "Add Customer", "Add Invoice" and "Customer Invoices" page from the "Home Page".

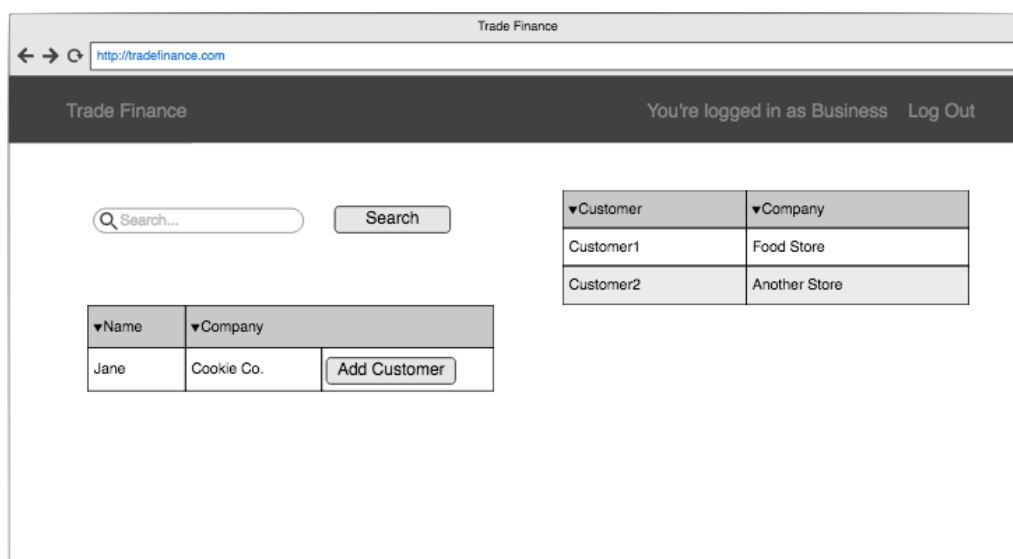
Wireframe



3.4.2.1 Add Customer

When business users navigate to the “Add Customer” page, business users will be able to view a list of customers that the business may send invoices to.

Wireframe



Behaviour

The following behaviour will be observed:

- A table of existing list of customers will be displayed at the side.
- Business users may search for a customer by typing in the customer’s name.
- The results of the customer’s name and details of the company will be displayed below in a table.
- An “Add Customer” button will be displayed beside the results.
- When “Add Customer” is selected, the customer will be added to the business’ customers list.

3.4.2.2 Add Invoice

When business users navigate to the “Add Invoice” page, business users will be able to add a new invoice transaction to the blockchain for credit factoring.

Wireframe

The wireframe shows a web browser window titled "Trade Finance" with the URL "http://tradefinance.com". The page header includes "Trade Finance" and "You're logged in as Business Log Out". The main form area contains the following fields and buttons:

- Send Invoice To:** A dropdown menu with "Customer Name" selected.
- Invoice No.:** A text input field containing "IN12345".
- Invoice Date:** A text input field containing "3/3/2017".
- Amount:** A text input field containing "5000".
- Invoice Due:** A text input field containing "5/5/2017".
- Upload Invoice:** A text input field containing "IN12345.pdf" and an "Upload" button.
- Buttons:** "Back" and "Send Invoice" buttons at the bottom.

Behaviour

The following behaviour will be observed:

- Business users will be able to send an invoice to a customer by selecting from the dropdown list.
- The display will allow users to enter the details to create an invoice transaction.
- All fields are mandatory to make a valid transaction.
 - A copy of the pdf version of the invoice generated from the user's accounting software should be included.
 - Details entered should match the amount on the pdf copy.
- Clicking on the “Back” button will navigate user back to the Home page.
- Clicking the “Send Invoice” button will send a copy of the invoice transaction to the intended recipient.
- When an invoice has been added successfully, the timestamp of the invoice is created and it will navigate into the “Sales Invoices” page (See section 3.4.2.3).

3.4.2.3 Sales Invoices

When business users navigate to the “Sales Invoice” page, a history record that the business user had previously added will be displayed.

Wireframe

The wireframe shows a web browser window with the URL <http://tradefinance.com>. The page title is "Trade Finance". The user is logged in as "Business" and can click "Log Out". The main content area is titled "Sales Invoices" and contains a table with the following data:

▼Customer	▼Invoice No	▼Invoice Date	▼Amount	▼Invoice Due Date	▼Date Added	▼Status	▼
Customer1	IN12345	3/3/2017	5000	5/5/2017	5/3/2017	Pending	View Invoice
Customer1	IN12344	3/2/2017	5000	5/4/2017	5/2/2017	Awaiting	View Invoice
Customer1	IN12343	3/1/2017	5000	5/3/2017	5/1/2017	Approved	View Invoice
Customer1	IN12342	3/12/2016	5000	5/2/2017	5/12/2016	Paid	View Invoice

Behaviour

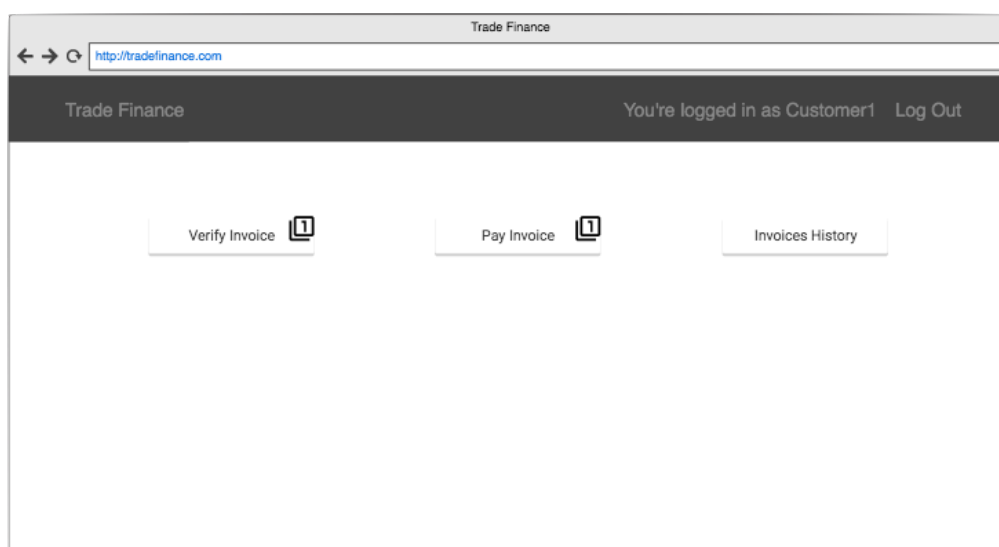
The following behaviour will be observed:

- A table displaying a record of all invoices in the order that invoices were added.
- A record that shows the name of the customers that received the invoice, details of the invoice and the status of the invoice (See section 3.4.5).

3.4.3 Home Page for Customer Users

Customer users may navigate to the “Verify Invoice”, “Pay Invoice” and “View All Invoices” page from the “Home Page”.

Wireframe



Behaviour

The following behaviour will be observed:

- The number of pending invoices waiting for verification will be displayed next to the “Verify Invoice” navigation button.
- The number of invoices with due payment will be displayed next to the “Pay Invoice” navigation button.
- For “Invoice Status” see section 3.4.5.

3.4.3.1 Verify Invoice

When customer users navigate to the “Verify Invoice” page, invoices that are waiting for the user to verify will be displayed.

Wireframe

▼Sender	▼Invoice No	▼Invoice Date	▼Amount	▼Invoice Due Date	▼Date Added	▼Status	▼
Business	IN12345	3/3/2017	5000	5/5/2017	5/3/2017	Verify	View Invoice

Behaviour

The following behaviour will be observed:

- A table record of invoices received and waiting for verification.
- A record with the name of business that sent the invoice, details of the invoice and a “Verify” button to verify the invoice.
- When the “Verify” button is selected, the invoice status will be updated from “Pending” to “Awaiting”.
- For “Invoice Status” see section 3.4.5.

3.4.3.2 Pay Invoice

When customer users navigate to the “Pay Invoice” page”, the invoices that are waiting for the user to confirm payment will be displayed.

Wireframe

The wireframe shows a web browser window with the URL <http://tradefinance.com>. The page title is "Trade Finance". The user is logged in as "Customer1" and can click "Log Out". The main heading is "Pay Invoice". Below it is a table with the following data:

▼Sender	▼Invoice No	▼Invoice Date	▼Amount	▼Invoice Due Date	▼Date Added	▼Status	▼
Business	IN12343	3/1/2017	5000	5/3/2017	5/1/2017	Pay	View Invoice

Behaviour

The following behaviour will be observed:

- A table record of invoices received and waiting for payment.
- A record with the name of the business that sent the invoice, details of the invoice and a "Pay" button to confirm that the payment for the invoice has been made.
- When the "Pay" button is selected, the invoice status will be updated from "Approved" to "Paid".
- For "Invoice Status" see section 3.4.5.

3.4.3.3 Invoices History

When customer users navigate to the "Invoices History" page, a history of invoices that the user has received will be displayed.

Wireframe

The wireframe shows a web browser window with the URL <http://tradefinance.com>. The page title is "Trade Finance". The user is logged in as "Customer1" and can click "Log Out". The main heading is "Invoices History". Below it is a table with the following data:

▼Sender	▼Invoice No	▼Invoice Date	▼Amount	▼Invoice Due Date	▼Date Added	▼Status	▼
Business	IN12342	3/12/2016	5000	5/2/2017	5/12/2016	Paid	View Invoice

Behaviour

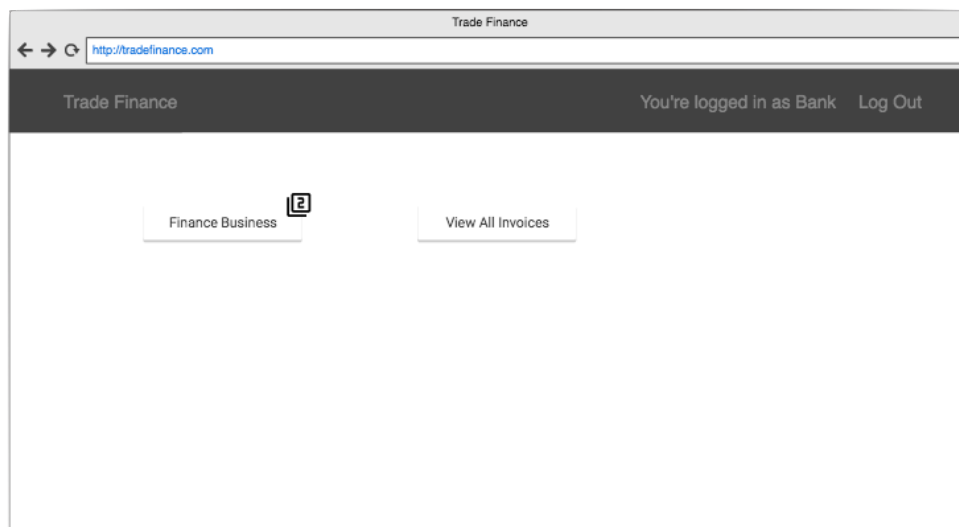
The following behaviour will be observed:

- A table record of invoices that have been received and paid.
- A record that displays the name of business that sent the invoice and details of the invoice.

3.4.4 Home Page for Bank User

The bank user will be able to navigate to the “Finance Business” and “View All Invoices” page from the “Home Page”.

Wireframe



Behaviour

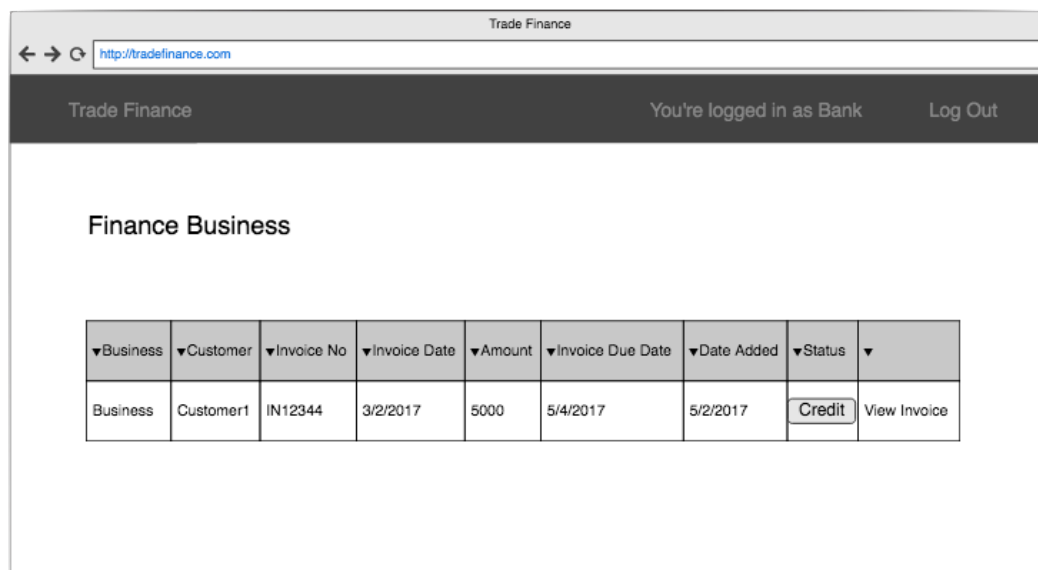
The following behaviour will be observed:

- The number of verified invoices waiting for credit factoring approval will be displayed next to the “Finance Business” navigation button.
- For “Invoice Status” see section 3.4.5.

3.4.4.1 Finance Business

The bank user will have a “Finance Business” page. The “Finance Business” will display a list of invoices by businesses waiting for credit factoring approval.

Wireframe



Behaviour

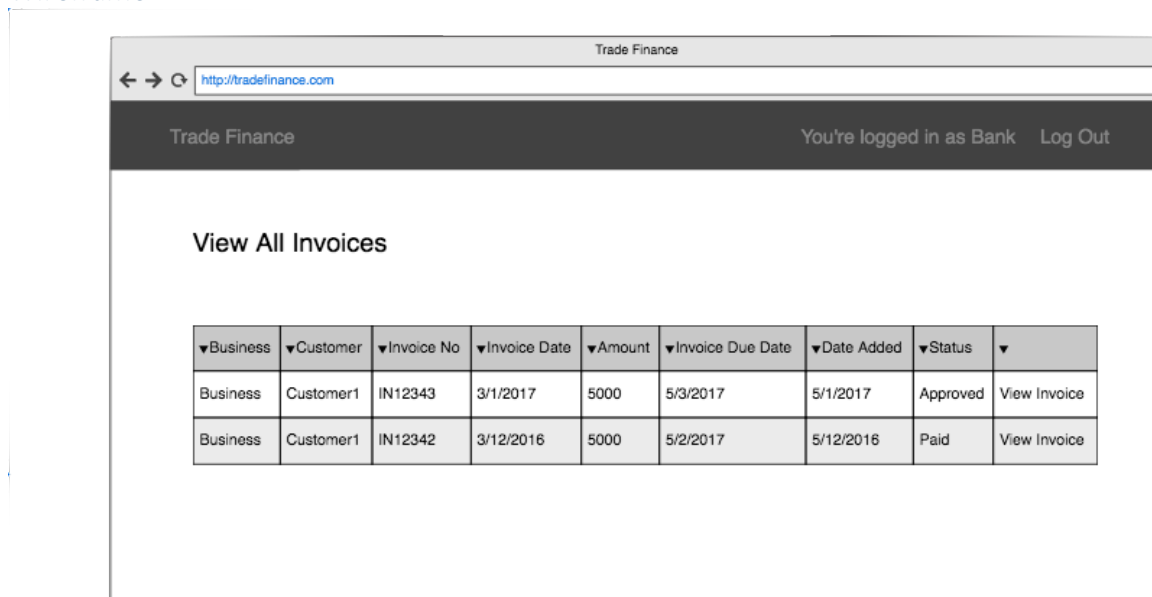
The following behaviour will be observed:

- A table record of invoices that have been verified and waiting for approval by the bank for credit factoring.
- A record with the name of the business that sent the invoice, name of customer that the invoice was sent to, details of the invoice and a “Credit” button.
- When the “Credit” button is selected, the invoice status will be updated from “Verified” to “Approved”.

3.4.4.2 View All Invoices

The bank user will have a “View All Invoices” page that will display a list of invoices that the bank has approved.

Wireframe



Behaviour

The following behaviour will be observed:

- A table with a record of all approved and paid invoices.
- Users can search for a business based on the business name.
 - A possible functionality is to include having an “ageing”/ages of invoice, that is, the days that an invoice has been overdue.

3.4.5 Invoice Status

An invoice transaction will undergo several changes of statuses through its lifecycle:

- Pending – An invoice has been added and sent but is not yet verified by the intended recipient.
- Verified/Awaiting – The customer has received their goods and verified that the invoice is genuine. Verified invoices will be displayed to the Bank user as “Awaiting” approval.
- Approved – Invoice that the bank has approved for credit factoring.
- Paid – The customer has made payment for the invoice.

4. Implementation

Hyperledger Fabric is an open-source, modular, multi-channel transaction network that is built and maintained by the Hyperledger community [13]. The source for Hyperledger Fabric can be found on the Hyperledger Fabric's GitHub repository. Before the Alpha version was released, version 0.6 branch was a more stable version compared to the master branch. The version 0.6 branch was thus used at the inception of this project's implementation before switching to the Alpha version. As Hyperledger is an open source, support can be found using the forums and chat groups in the community for discussions [13].

Even though version 0.6 was the most stable version at the time, there would be errors such as "Error: sql: no rows in result set". Such errors occurred when trying to obtain transaction certificates from the membership service. Since all of the validating peers are registered and enrolled with the membership service, the only way to fix it was to restart the entire network by restarting or resetting Docker.

In the middle of February 2017, the Alpha version (V1.0) was released. V1.0 consists of a few additional features such as having orderers, creating channels and using CouchDB as its state database. Even though upgrading meant that a few changes would be required in the chaincode, the more important thing is that V1.0 is the most stable version to date. V1.0 is also highly recommended as it has better support for debugging by providing well-written and effective error messages. The documentations in V1.0 have also vastly improved compared to the documentations available in v0.6.

4.1 Setting Up Network for Development

For development purposes, the entire Hyperledger network for chaincode development can be set up on a local machine by using Docker. Docker provides a software container platform, which does not bundle a full operating system like virtual machines. Instead, the containers use libraries and settings required for the software to work [18]. The Hyperledger Fabric project publishes Docker images that can be used by pulling the images from Dockerhub [19]. The Hyperledger blockchain network can also be set up on Bluemix, a web service provided by IBM. The only version available by Bluemix is v0.6 and with only a 30-day trial.

After pulling the right images and version, the entire network environment as represented in Figure 3.2, is configured in the docker-compose.yaml file. Running the command **docker-compose up** in the folder containing the docker-compose.yaml file will bring up the Hyperledger network. Docker will create the containers for the CA, Orderer, CouchDB and Peer0.

There are two ways to interact with the Hyperledger network, using the SDK or command line interface (CLI). In the docker-compose file, there is the additional container for CLI. The CLI will run the init.sh file in the container for the orderer bootstrap and channel creation, and is known as end-to-end verification [20]. The CA CLI consumes the init.sh file which will set the environment of the orderer and create the channel "mych". It then sets the environment for the Peer0 as Org1 and joins the channel. It is also able to install and instantiate the chaincode "mycc". All peers in the network will receive and hold a copy of the state. Once the chaincode has been initiated, the web application will handle the query and invoke functions using the SDK.

Setting up the network can be tricky because everything could be running even though the settings might not be right. When using v0.6, there was a problem with enrolling members and it was automatically assumed that the problem was a missing step using the SDK. It took a long time before realising that the problem lied with the docker files. Inspecting the settings of containers can be done by running the command **docker inspect membersvc** in terminal. It turns out that the IP of the membersvc container was 7054/tcp because in the docker file it was – “7054”. The NodeJS application was unable to connect to the membersvc because there was no port forward. Port forward can be done by setting the ports to – “7054:7054”, then the IP for the membersvc will be 0.0.0.0:7054 -> 7054/tcp.

4.2 Writing Chaincode

Chaincode in Hyperledger is the smart contract that defines the business logic of an asset or assets and the transaction instructions for modifying asset(s). Chaincode enforces the rules for reading or altering key value pairs or other state database information. Chaincode functions execute against the ledger current state database and are initiated through a transaction proposal. Chaincode execution results in a set of key value writes (write set) that can be submitted to the network and applied to the ledger on all peers [21].

Chaincode is usually written in Golang or alternatively, in Java. To turn a piece of Go code into a chaincode, all that is needed is to implement the chaincode shim interface [22].

4.2.1 Dependencies

There are a few dependencies that need to be included in the import statement list in order for the go code to build successfully.

- **fmt** – contains `Println` for debugging/logging
- **errors** – standard Go error format
- **github.com/hyperledger/fabric/core/chaincode/shim** – to implement the shim interface. It contains the definition for the chaincode interface and the chaincode stub. This is needed to interact with the ledger.

4.2.2 Chaincode Interface

In v0.6, the three functions to be implemented are **Init**, **Invoke** and **Query**. All three functions take in a ‘stub’, which is used to read and write to the ledger, the function name and an array of strings. The main difference between the functions is when they are called.

4.2.2.1 Init Function

When the chaincode is deployed, the `Init` function is called to instantiate the state of the ledger. This function is used to do any initialization the chaincode needs, such as to configure the initial state of a key/value pair on the ledger.

For the prototype, there is only one bank user. The initial state for the bank user is configured to hold the secondary keys of businesses and trading. The bank user is, therefore, able to view the business users that have sent invoice transactions to the customer users. The values for businesses and trading act as a key for indexing related invoice transactions. The businesses key holds values of business users that have invoices for credit factoring. The key for trading

holds values of invoices that has been verified by customer users and added to the list for the bank user to index.

Setting the key/value pair can be done by using the stub function `stub.PutState()`. The `stub.PutState()` function takes two arguments: the first is the key and the second is the value in bytes.

4.2.2.2 Invoke Function

The invoke function is called to add transactions to the blockchain. Invocations will be captured as transactions, which get grouped into blocks on the chain. To update the ledger requires invoking the chaincode. The structure of the invoke function is simple. The invoke function receives a function name and an array of arguments. Based on the name that was passed in through the function parameter in the invoke request, invoke will either call a helper function or return an error.

4.2.2.3 Query Function

As the name implies, Query is used to querying the chaincode's state. Queries do not add blocks to the chain nor use functions like `PutState` inside of Query and helper functions. Query is used to read the value of the chaincode state's key/value pairs using the function `GetState`.

The code demonstrating the query function is as attached in Appendix 9.1.2.

4.2.2.4 Main Function

Finally, there is the Main function which needs to be included. The Main function is executed when each peer deploys an instance of the chaincode and runs the `shim.Start` function. This sets up the communication between the chaincode and the peer that deployed it. The Main function must be included in any chaincode, without the need for alteration.

The Main function can be referred to in Appendix 9.1.5.

4.2.2.5 v0.6 to v1.0 Hyperledger Fabric

A few alterations were required for the chaincode when upgrading from v0.6 to v1.0. In v1.0 the chaincode shim supports two functions, **Init** and **Invoke** [23]. Both updating or querying the ledger are done by invoking the chaincode.

When the **Init** or **Invoke** function of a chaincode is called, the fabric passes the `stub.ChaincodeStubInterface` parameter and the chaincode returns a `pb.Response`. This stub can be used to call APIs to access the ledger services, transaction context or to invoke other chaincodes. The chaincode response comes in the form of a protocol buffer and will also return message events as well as chaincode events.

The code demonstrating the **Init** and **Invoke** functions is as attached in Appendix 9.1.3 and Appendix 9.1.4.

After deploying the chaincode, the SDK and CLI can be used to test the functions and interact with the chaincode [23].

4.3 HFC SDK

NodeJS web applications are able to use the HFC SDK, which is an API, to interact with the Hyperledger Fabric blockchain network. The NodeJS application is the client that takes care of all the query and invokes transactions by the end user.

The best way to learn how to use the SDK is by looking at the examples on GitHub, provided by IBM and Hyperledger. In order for the application to interact with the blockchain network, the HFC dependencies need to be included in the package.json files. Dependencies can then be included in the app by using the require() function which will load libraries and modules.

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

The line above is included in the application to create the client object and chain object. The chain adds the client by connecting to the ports of the Peer, Orderer, and CA. Apart from the CA that uses a http protocol, the rest uses gRPC, a remote procedure call developed by google [24]. GRPC can use protocol buffers as both its interactive data language and its underlying interchange format [24]. Once connected, it will enrol the application as the admin user with the CA. One of the roles of the admin is to register a new identity.

The hardest part of using the SDK is successfully connecting and enrolling members. There is a lot of code involved when using the SDK to connect and it is easy to miss a step. It is also worth noting how port forward works and how to use the right ports. The app will need to connect to the Peer, Orderer and CA nodes. The ports are set in the docker file are in the format of ports: - <host:container>, where host is the external port and container is the internal port. Since the app itself is not on the docker network, it can only connect to its external port.

The code demonstrating how to use the SDK to connect to the Hyperledger network and to enrol the admin can be referred to in Appendix 9.1.6.

After enrolling the admin, the web application will be able to invoke and query transactions in the blockchain. To invoke or query a transaction, the function will take in two arguments: a string which is the name of the function to call and an array which defines the key/value.

```
exports.query = (func, args) => {
```

```
exports.invoke = (func, args) => {
```

To make a request, it is necessary to include the chaincode ID and channel ID. In this case, “mycc” and “mych”, a nonce and transaction ID. The transaction ID is created from the nonce and user.

```
var nonce = utils.getNonce();

tx_id = hfc.buildTransactionID(nonce, admin);

var request = {
  chaincodeId: config.app.chaincodeId, // mycc
```

```
chainId: config.app.channelId,      // mych
txId: tx_id,
nonce: nonce,
fcn: func,
args: args
};
```

Querying a transaction can be done simply by making a request using the `queryByChaincode()` function, and including the function name and arguments of the key/value in the request.

```
return chain.queryByChaincode(request)
```

The code demonstrating a query using the SDK is as attached in Appendix 9.1.7.

The steps to create a proposal for a transaction use a similar request as a query. In contrast to creating a proposal, sending a transaction is a bit more complex. A proposal for the transaction can be made using the request as shown above.

```
return chain.sendTransactionProposal(request)
```

The transaction proposal, as shown above, will return a response. Frequent checking for the response is required to ensure that the status of the proposal and header are both valid. Once all checks are valid, the request of the transaction will be accepted.

```
var sendPromise = chain.sendTransaction(request);
```

The code demonstrating how to add a transaction using the SDK can be referred to in Appendix 9.1.8.

4.4 NodeJS Application

NodeJS has the largest ecosystem of open source libraries available to develop web applications known as node modules [17]. Initially, Swagger, to speed up RESTful API development, was tried and tested to get forms to add an invoice. It was, however, soon recognised that Swagger was not needed as the HFC SDK had already provided the API that the system needs. Finally, the system settled with Express, a minimalist web framework for NodeJS.

Express does the HTTP GET and POST method using routers. The router renders a page from the server to the client with the GET method, and the POST method will send requests from the client to the server. When a user, for example, wants to add a user, it calls the POST method from the client's side and sends the data of the request. On the server's side, the POST router receives the data and sends the request to query or invoke, using the SDK. Since these can all be done in AJAX, the web application's page does not need refreshing at every request. There is the possibility of an AJAX POST request failing from the client's side while sending to the server. It is important to note that in order for the HTTP request to work, it will need the `access-control-allow-origin` included in the header of the application. `Access-control-allow-origin` is a CORS header. The response header allows the content of an application to be accessible to a certain origin.

As mentioned in section 4.2.2, a query or invoke request is done by calling the method and sending a string of the function name as well as an array of the arguments. For example, querying the list of businesses that are trading can be done with the function 'query', and an array of one argument '_businessIndex'.

```
queryResults('query', ["_businessIndex"]).then((results) => {
```

The results from the chaincode are returned in the form of a protocol buffer. The protocol buffer can be changed to a human readable format using the toString() function.

```
// buffer to string
var businessUsers = bufferToString(results);
```

```
function bufferToString(bytes) {
  return bytes.toString("utf8");
}
```

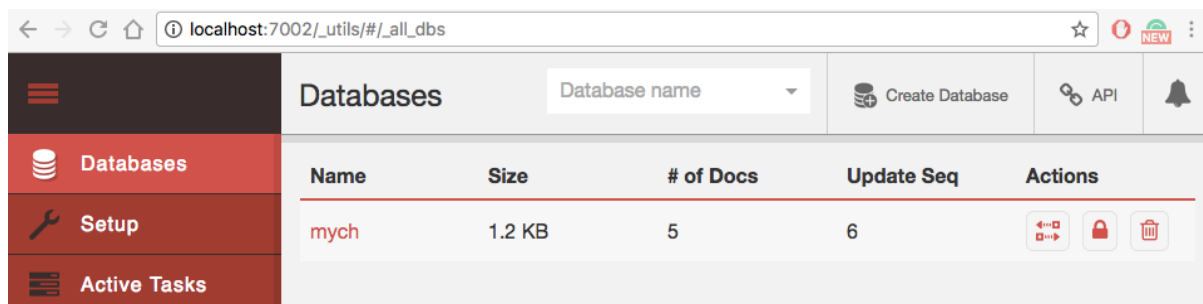
When needed, the results can be formatted into a JSON object using the JSON.parse() function. By doing so, the values of the results will be readable and easier to use.

```
var json = JSON.parse(businessUsers);
```

When developing the NodeJS system that interacts with the Hyperledger blockchain, querying non-existing transactions will return a null value. Consequently, regular checks for null values should be made throughout the development to prevent errors from occurring, when querying any key/values that do not exist in the ledger.

5. Results and Evaluation

The results of the main functionalities of the smart contract prototype have been successful. The prototype implementation demonstrates how a blockchain can be used to keep a digital trail of an invoice as it passes between users. In the existing system of trade finance, a business would often give the bank invoices in bulk for credit factoring, resulting many invoices being left unverified, and leaving space for error and fraud. The prototype system will allow bank, customer and business users to keep track of invoices using blockchain technology in real-time. By using the system, a bank user may keep careful track of business users that have invoices for credit factoring. The business users may add customer users and send invoices using the system. Through the blockchain, customer users may receive and verify the invoices instantaneously. When a customer has confirmed an invoice to be true, the bank will be able to approve the verified invoice for credit factoring. Using smart contracts, the status of an invoice may vary as it passes between users during its lifecycle. All invoices added to the blockchain are tracked in real time and include history records.

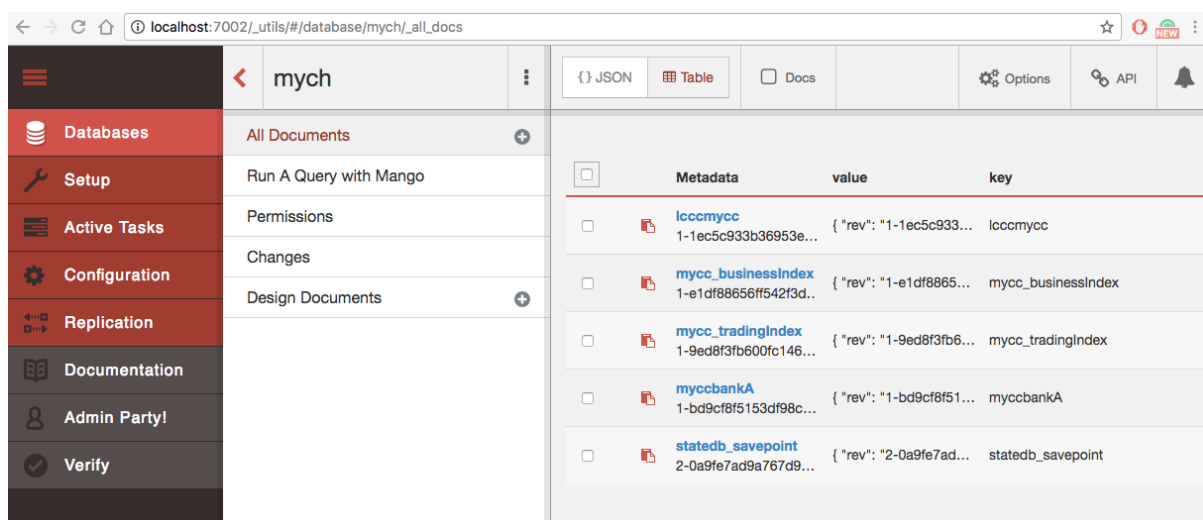


The screenshot shows the CouchDB Databases interface in a web browser. The address bar shows 'localhost:7002/_utils/#/_all_dbs'. The interface has a sidebar with 'Databases', 'Setup', and 'Active Tasks'. The main area shows a table of databases. The 'mych' database is listed with a size of 1.2 KB, 5 documents, and 6 update sequences. There are icons for 'Create Database', 'API', and a bell.

Name	Size	# of Docs	Update Seq	Actions
mych	1.2 KB	5	6	[Icons for document, lock, delete]

Figure 5.1 State of the blockchain when first initialised. Each doc represents a block in the blockchain

In Hyperledger, smart contracts are called chaincode. The businesses and trading key are initialised when the smart contract “mycc” is deployed. The state of the ledger and transactions can be access using CouchDB. CouchDB allows rich and complex queries against the state ledger content [16]. The instantiate state of the ledger for the channel “mych” is as shown in Figure 5.1 above. The number of docs represents the data in the blocks of a blockchain.



The screenshot shows the CouchDB interface for the 'mych' database. The address bar shows 'localhost:7002/_utils/#/database/mych/_all_docs'. The interface has a sidebar with 'Databases', 'Setup', 'Active Tasks', 'Configuration', 'Replication', 'Documentation', 'Admin Party!', and 'Verify'. The main area shows a table of documents. The documents are listed with their keys and values. The keys are 'lcccmmycc', 'mycc_businessIndex', 'mycc_tradingIndex', 'myccbankA', and 'statedb_savepoint'. The values are JSON objects containing revision numbers and keys.

Metadata	value	key
<input type="checkbox"/> lcccmmycc 1-1ec5c933b36953e...	{ "rev": "1-1ec5c933..." }	lcccmmycc
<input type="checkbox"/> mycc_businessIndex 1-e1df88656ff542f3d..	{ "rev": "1-e1df8865..." }	mycc_businessIndex
<input type="checkbox"/> mycc_tradingIndex 1-9ed8f3fb600fc146...	{ "rev": "1-9ed8f3fb6..." }	mycc_tradingIndex
<input type="checkbox"/> myccbankA 1-bd9cf8f5153df98c...	{ "rev": "1-bd9cf8f51..." }	myccbankA
<input type="checkbox"/> statedb_savepoint 2-0a9fe7ad9a767d9...	{ "rev": "2-0a9fe7ad..." }	statedb_savepoint

Figure 5.2 Blocks in the blockchain represented by CouchDB

The businessIndex and tradingIndex are keys to index business users that have invoices for trading and invoices that have been verified by customers. A bank user, Bank A, holds the keys businessIndex and tradingIndex for indexing. The initial state of the blockchain is represented in figure 5.2, as mentioned in section 4.2.2.1 above.

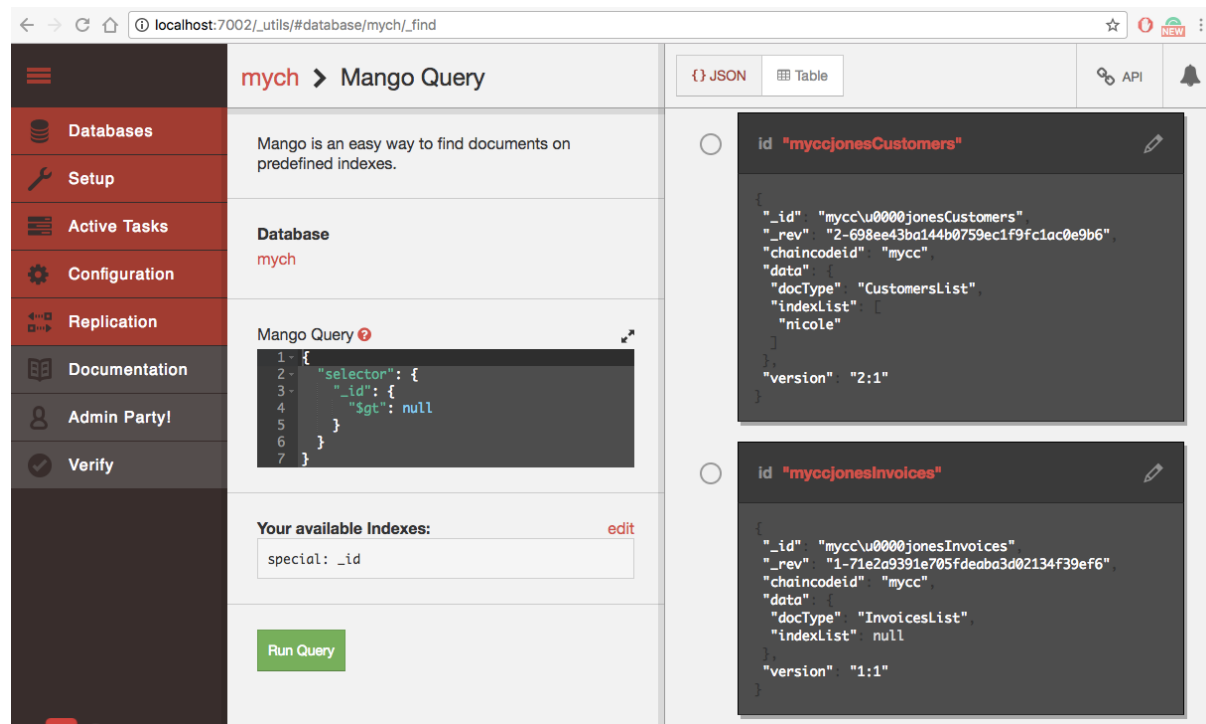


Figure 5.3 Version update of transactions in blockchain

At any time, the version of a transaction will be in its most updated value. When a business user is first initialised, the customers and invoices list that the business user holds are with a null value. When the business user adds a new customer, the customer's name will be added to the list of the business' customers. New invoices are added to the invoices list similarly. The version of the customers or invoices list will be updated as shown in figure 5.3 above, demonstrating the way transactions are immutable in a blockchain.

When a business adds a customer, the smart contract will perform checks to ensure that the customer has not already been added. The chaincode will loop through the list of the business' customers to check if there is a match with the customer that is being added. If there is a match, the chaincode will return an error message to the NodeJS client, stating that the customer has already been added. If there is no match, the chaincode will invoke the request and add the customer to the business' customers list.

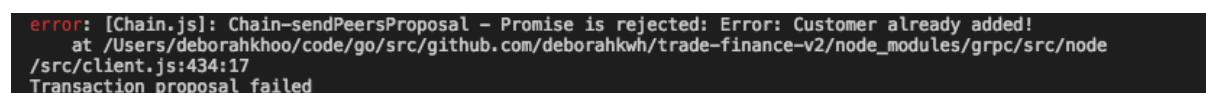


Figure 5.4 Message returned by chaincode when the same customer is added twice

The smart contract is also able to successfully perform checks when adding invoices so that the same invoice is not added twice in the ledger. If business A adds and sends invoice A to customer A, and later sends the same invoice to business B, the chaincode will reject adding the invoice transaction. The same will happen if business B adds and sends invoice A to customer B. Performing these checks ensures that users are not able to double finance an

invoice by creating multiple business accounts and sending to multiple customer accounts for verification.

```
5/5/2017
[ 'david', 'tom', 'IN1234', '3/3/2017', '5000', '5/5/2017' ]
error: [Chain.js]: Chain-sendPeersProposal - Promise is rejected: Error: Invoice already exist -in1234, davidtom
    at /Users/deborahkhoo/code/go/src/github.com/deborahkwh/trade-finance-v2/node_modules/grpc/src/node/src/client.js:434:17
Transaction proposal failed
[ 'david', 'susan', 'iones' ]
```

Figure 5.5 Message returned by chaincode when adding the same invoice twice

In theory, no changes in the details of an invoice should be permitted once the invoice transaction has already been added. All the same, an invoice is open to human errors as the invoice details are entered manually. Consequently, changes to an invoice are permitted before the invoice is verified. When the ledger receives an update for a transaction, the new values along with a record of the previous version of the transaction, are added to a new block for the changes. This means that the ledger will provide a verifiable history of all changes to the state.

As and when a user updates an invoice, other users associated with that invoice will be able to track the updates. When a business adds an invoice and sends it to a customer, the status of the invoice will be displayed as pending. The customer receiving the invoice will see that a pending invoice is sent by the business and is waiting for verification. Verified invoices will be made available to the bank user. Once verified, the bank user will be able to confirm and approve the invoices for credit factoring. When payments have been made for invoices that are due, customer users may confirm payment. Throughout the process, businesses will be able to keep track of invoices as it is being verified, approved and paid.

The trade finance system is also able to present an invoice in different tables according to its status and at different stages of its lifecycle. For example, a bank user may view all invoices that have been either approved or paid, but not the verified invoices, displayed in the same table. This example demonstrates the concept of how data can be extracted from transactions and used for analysis. Carrying out an analysis of data is useful to identify when a business user is behaving fraudulently, as mentioned in section 2.2.

Transactions in a blockchain are immutable, and records are available to all peers [6]. Factoring companies can choose to offer credit factoring based on the analysis of the reputation of the customers. Business could produce invoice transactions and create a separate account for its customer. If a customer has little to no records of payments made or is not deemed reputable, the factoring company may choose not to factor the invoice.

For the prototype, there is no login. The system will request and render the view of different details and functionalities depending on the user's name and user type of the selected user from a select option list. For the reason that NodeJS is a relatively new territory that has much to be explored, it felt easier to include all the functionalities in the same JavaScript file. The more functionalities there were, however, the slower the performance of the application. Different users have access to different pages that have different content to display in various tables. The same JavaScript function has to perform a lot of checks for each page, whether it is for the bank, business or customer user. By refactoring the code into separate JavaScript files, it improved the performance of the application. A page would only load the JavaScript

file associated with the user type. By doing so eliminates the need to perform so many checks on each page to display the right content.

There are a few improvements, mostly on the NodeJS client, that may improve the system. From debugging and testing, the system displays error messages on the NodeJS server's side when transactions fail, as seen in figure 5.4 and 5.5. These error messages are, however, not properly displayed to the end user. When the message from the shim interface is returned, NodeJS does not catch it as an error. Instead, according to NodeJS, the transaction is a "success". When a customer adds an invoice that already exists in the ledger, the system will be able to redirect to another page but is unable to display the error message. These are, nevertheless, considered minor issues as it does not affect transactions in the blockchain.

6. Future Work

For future works, it will be good to have validations for the NodeJS application. When end users are updating the status of an invoice, a validation message will appear for the user to confirm. Validation messages can prevent mishaps such as verifying and approving the wrong invoice. It will also be beneficial to display confirmation or error messages to the end user. Confirmation or error messages will allow the user to know what the system is doing, whether actions are successfully carried out.

Additionally, the blockchain network can be enhanced by allowing the system to enrol different users to the blockchain network. As of now, there is only one admin user for the application performing all transactions, while the bank, customer and business users are attributes in the ledger, just like invoices. By enrolling different users, transactions will have better security by having their own private keys.

Given more time, there are countless of useful functionalities that could be added to enhance the trade finance system in credit factoring. Instead of manually adding the details of an invoice into a transaction, if the system allows users to upload an invoice and obtain the data from an invoice pdf, it will eliminate human error. For instance, a business could enter £50,000 instead of £5000 inadvertently and the customer, having to verify a myriad of invoices, fails to notice the difference as well. When an invoice is due for payment, the error may cause a loss to either or both the business and the customer.

In credit factoring, the advance payment of an invoice is agreed on between the factoring company and the business. The current system only approves invoices for credit factoring. Through the blockchain system, it would benefit the factor to additionally set the advance payment of an invoice and the service charge when initializing the business user. The factoring facility can be negotiated on the system and can set a limit as to how much it is willing to lend to the business using smart contracts. When an invoice is added for credit factoring, the amount is deducted from the designated amount, and payments would mean a refund. If the amount for a business reaches its limit, the system would automatically reject further invoices for factoring from that business. Live data of the amount of the factoring facility used and the balance left for a user is visible in the system. This live data can be accessible to both the business user and the bank financing the invoice.

If a business has an invoice for £100k, but only requires £20k or if the factoring limit agreement has only a £40k balance left but the invoice value is worth £100k, the system could allow for partial factoring of an invoice. When a payment is made, the system will notify both the business and the bank and refund part of the payment to the business. This form of partial factoring might not be practised in the existing system as keeping track of it would be difficult. It may, however, be a practical approach when the value of an invoice is more than the amount needed.

The system should also have a history record of customers' payment data and a feature where banks can rate customers according to how timely payments are made. The data from the transactions can then be extracted from for analysis. If a customer is a bad paymaster, the finance company should consider disapproving the invoice for factoring.

7. Conclusion

A private key, which will allow a user to have control over one's transaction or unique digital asset, will be given when the user is enrolled. The deterrent of using blockchain technology lies in the security of the system or device that stores the private keys. In a public blockchain, security vulnerabilities have led a system operating the DAO (Decentralised Autonomous Organization) that runs on Ethereum, to financial losses. In private blockchains, operators must decide how to resolve the problem of lost identification credentials. Overcoming the issue of security will allow the enabling of decentralised commerce [6] which can be a fundamental change in business model and processes.

A distributed ledger may be considered more reliable compared to a traditional database as it is not controlled by a centralised party. Trust is established through peer-to-peer mass collaboration and sophisticated computer code rather than through a centralised powerful institution. Peer-to-peer mass collaboration makes transactions and interactions between people and businesses much more efficient. Records of transactions can still be available if a single node fails as all peer nodes hold a copy of the state ledger.

Smart contracts can automatically enforce agreements between two or more parties without the need for an intermediary. The use of smart contracts is to initialize or perform all kinds of checks on transactions. Transactions are immutable and traceable in the open ledger. The ledger is also able to provide transparency among participants in the blockchain network. Each factoring company in the distributed network is a peer of its own. A consensus network removes the need for a central party or middle-person. When an invoice is added, it may be validated by all peers. As each peer holds a state and a copy of the ledger, this will aid in preventing double financing and discourage fraud. The success of this is, therefore, dependent on a community, the involvement of other factoring companies. For example, in the existing credit factoring system, a business can double finance an invoice by going to two different factoring companies. With the involvement of multiple factoring companies in a blockchain network, added invoices on the ledger shared among all peers can be checked. Invoice transactions can also be tracked in real time transactions to settle invoices more efficiently.

As more banks and finance institution get involved, the trade finance system will become its own marketplace. Verified invoices can be made available to all factors, and the banks and finance institution will be able to set advance rates based on the base, size and reputation of the business' customer. Based on the highest advance payment given or lowest charges, businesses may decide which proposals are to be accepted. The use of blockchain technology in smart contracts is thus able to change the entire business model of an existing system.

8. Reflection on Learning

While working on this project, I gained skills in project management and in working closely with clients. Before initiating the project, a client meeting was arranged to discuss the problem of credit factoring and how blockchain technology could enhance the existing system. From then onwards, I provided Equiniti with the design and specifications of the prototype as well as progress updates on the project. Throughout the project, I also worked on my communication skills by updating my supervisor on my progress and challenges faced during our weekly meetings. I was able to discuss problems and possible solutions, which is vital in managing the project.

During the project, I had the opportunity to meet with a blockchain developer from Equiniti. I was fortunate to be able to seek advice on which software and technologies to use for the project. I made sure to take notes so that I could always refer to it again. Discussions with someone with more experience are valuable as it enables us to avoid unnecessary problems by learning from their mistakes and thus save time. We also discussed potential solutions on using the Hyperledger. As Hyperledger is relatively new, there are few solutions available online. The developer was, however, able to provide me with useful tips such as how to inspect containers in Docker. The tips received helped me realise that the ports were incorrect for my containers and allowed me to fix the problem, as mentioned in section 4.1.

As Hyperledger, GoLang and NodeJS were all new things for me to learn, the best way was to go through the examples and demos provided by IBM to set up the prototype. At times, there were unexpected problems during the project and at times, a few days were needed to work out the solution for just a single problem. Nevertheless, while finding solutions to the problem, I found myself researching and learning more from it. With Hyperledger being a new and open source, there were frequent updates and therefore a lack of documentation. I spent a lot of time learning how to enrol members with the SDK, writing the query and invoke functions using the SDK and writing a NodeJS application. Besides meeting with technical challenges, fracturing my ankle two weeks before the dissertation was due, slowed me down quite a bit. Even so, with proper time management skills, I managed to stay on top of it. I knew from the start that working on this project can be a lot of taking on as all the technologies and tools I would be using will be new to me. Therefore, I made sure to follow up on the project daily right from the start, so I will not fall behind schedule.

It was definitely a steep learning curve in understanding how Hyperledger works, setting the environment of the network and for Go, writing chaincodes in GoLang and developing an app in NodeJS. Compared to when I first started, I have gained technical skills in both Go and JavaScript and improved my programming knowledge. The fundamental of programming is logic and development for the project was done based on a lot of basic logic: from getting a query, printing the results, formatting the results, identifying the format type that is needed and converting the results to the suitable format type. When developing a program, it is useful to log and print errors to debug errors. As I became more familiar with the tools, I was able to pick up some techniques and make improvements to my code, such as refactoring and reusability. Additionally, improvements were made to the performance of the application as mentioned in Section 5.

While taking up this project, I learned about distributed ledgers and how the smart contracts can be used. I also had the chance to work on a range of technologies and develop different technical skills despite the short amount of time. In the future development of Hyperledger blockchain network, there will be tools such as Fabric Composer which will not only simplify the development of Hyperledger Fabric blockchain applications [25], but save a lot of development time. Still, it was useful to understand how the components in the Hyperledger network work and how to set things up manually. Even though there were changes that needed to be done to the existing app when upgrading from v0.6 to v1.0, I was able to quickly grasp an understanding compared to when I first started with v0.6.

Even though the project went well according to the initial plan, I learned that I could set better goals for future projects by having better-directed goals. At the start of the project, I was more focused on trying to understand what Equiniti wanted from the project. Towards the end, I learned the importance of making some decisions myself. As my clients, Equiniti may have the ideas on what blockchain technology and what a smart contract can do and how it could be used to improve problems. Nevertheless, being the one managing the project, I needed to decide on the best solutions to their problem. Sometimes, the clients might not know what they want. Alternatively, they might be able to come up with an idea of how a certain technology could be used to solve a problem, but might not know the logic behind it and how to make it happen. Therefore, my role would be to explain why a solution would work and convince the clients to accept it. It is up to the person designing the system to understand the purpose of the use case and make decisions based on requirements.

Towards the end of my project, I had the chance to demonstrate my prototype to Equiniti to gain some feedback. During the demonstration, I realised that I still need to improve my presentation skills by gaining more confidence, finding ways to calm my nervousness and by learning to speak at a slower pace. When presenting a system, it may be useful to have a storyline for the demonstration, especially when there is a group among the audiences who have little to no knowledge of the presentation content. I also realised that I should not begin by presenting all the functionalities of a system as it might be confusing to the audience. When giving a presentation, it would be clearer and better understood by beginning with the main functionality of a system. For example, it is better to say, 'This system allows businesses to send invoices to customers,' instead of, 'Businesses are able to add customers and send invoices using this system'.

Overall, I am thankful for having received good feedback from my demo of the prototype to Equiniti who seemed impressed with what I have produced and for covering all the requirements discussed. The feedback from Equiniti is as attached in Appendix 9.2. The prototype would potentially be used in another demo to the Equiniti Risk Factor Solutions who were originally the one that came up with the trade finance proof of concept.

9. Appendix

9.1 Code

9.1.1 Docker-Compose.yaml

```
version: '2'

services:

  # Certificate Authority
  ca:
    container_name: ca
    image: hyperledger/fabric-ca:latest
    environment:
      - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
      # NOTE: This line must be changed when new crypto material is generated
      # to match the new key file
      - FABRIC_CA_SERVER_CA_KEYFILE=/etc/hyperledger/fabric-ca-server-
      config/99ceb551cc66cb166b712c634fd1efd88d905c0205cef06101a77b484b31830c_sk
      - FABRIC_CA_SERVER_CA_CERTFILE=/etc/hyperledger/fabric-ca-server-
      config/peer0rg1-cert.pem
    ports:
      - 7040:7054
    volumes:
      - ../crypto/peerOrganizations/peer0rg1/ca:/etc/hyperledger/fabric-ca-
      server-config
    command: sh -c 'fabric-ca-server start -b admin:adminpw' -d

  # Orderer
  orderer:
    container_name: orderer
    image: hyperledger/fabric-orderer:latest
    environment:
      - ORDERER_GENERAL_LOGLEVEL=debug
      - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
      # Genesis
      - ORDERER_GENERAL_GENESIMETHOD=file
      - ORDERER_GENERAL_GENESISFILE=/opt/gopath/src/mnt/gen/orderer.block
      # TLS
      - ORDERER_GENERAL_TLS_ENABLED=false
      # MSP
      - ORDERER_GENERAL_LOCALMSPID=OrdererOrg
      - ORDERER_GENERAL_LOCALMSPDIR=/opt/gopath/src/mnt/msp/
    volumes:
      - ../crypto/ordererOrganizations/ordererOrg1/orderers/ordererOrg1orderer1:/opt/gopa
      th/src/mnt/msp/
      - ../gen:/opt/gopath/src/mnt/gen/
    ports:
      - 7050:7050
    working_dir: /opt/gopath/src/github.com/hyperledger/fabric
```

```

    command: orderer

# CouchDB for Peer0
couchdb0:
    container_name: couchdb0
    image: hyperledger/fabric-couchdb:latest
    ports:
        - 7002:5984

# Peer 0 (App)
peer0:
    container_name: peer0
    image: hyperledger/fabric-peer:latest
    environment:
        - CORE_PEER_ID=peer0
        #- CORE_PEER_ADDRESS=peer0:7051
        - CORE_PEER_ADDRESS=172.18.0.5:7051
        - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0:7051
        - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
        - CORE_LOGGING_LEVEL=info # error | info | debug
        - CORE_NEXT=true
        - CORE_PEER_ENDORSER_ENABLED=true
        - CORE_PEER_PROFILE_ENABLED=true
        # MSP
        - CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/mnt/msp/
        - CORE_PEER_LOCALMSPID=Org1
        # Gossip
        - CORE_PEER_GOSSIP_ORGLEADER=false
        - CORE_PEER_GOSSIP_USELEADERELECTION=true
        # TLS
        - CORE_PEER_TLS_ENABLED=false
        # CouchDB
        - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
        - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb0:5984
    volumes:
        - /var/run:/host/var/run/

- ../crypto/peer0organizations/peerOrg1/peers/peerOrg1Peer1:/opt/gopath/src/mnt/msp/

    ports:
        - 7000:7051
        - 7001:7053
    depends_on:
        - ca
        - orderer
        - couchdb0
    working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
    command: peer node start --peer-defaultchain=false

# Command Line Interface
cli:

```

```

container_name: cli
image: hyperledger/fabric-peer:latest
tty: true
environment:
  - GOPATH=/opt/gopath
  - CORE_PEER_ADDRESSAUTODETECT=true
  - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
  - CORE_LOGGING_LEVEL=DEBUG
  - CORE_NEXT=true
  - CORE_PEER_ID=cli
  - CORE_PEER_ENDORSER_ENABLED=true
  - CORE_PEER_ADDRESS=peer0:7051
  - CORE_PEER_GOSSIP_IGNORESECURITY=true
  - CORE_PEER_LOCALMSPID=Org1
  - MNT=/opt/gopath/src/mnt
volumes:
  # Docker socket
  - /var/run:/host/var/run/
  # Chaincode
  - ../chaincode:/opt/gopath/src/mnt/chaincode/
  # Scripts
  - ../scripts/container:/opt/gopath/src/mnt/scripts/
  # Crypto Material
  - ../crypto:/opt/gopath/src/mnt/crypto/
  # Generated Material
  - ../gen:/opt/gopath/src/mnt/gen/
  # Logs
  - ../logs:/opt/gopath/src/mnt/logs/
  # Data persistence
  #- ../data:/var/hyperledger
depends_on:
  - ca
  - orderer
  - peer0

# Start in the scripts folder and run the init.sh script
working_dir: /opt/gopath/src/mnt/scripts/
#command: /bin/bash -c 'while true; do sleep 1000; done'
command: /bin/bash -c 'sleep 2; source ./init.sh; while true; do sleep
1000; done'

```

9.1.2 Chaincode Query()

```

// Query - query chaincode
func (t *SimpleChaincode) Query(stub shim.ChaincodeStubInterface, function string,
args []string) ([]byte, error) {
    fmt.Println("query is running " + function)

    // Handle different functions
    if function == "read" { //read a variable
        return t.Read(stub, args)
    }
}

```

```

    fmt.Println("query did not find func: " + function)

    return nil, errors.New("Received unknown function query: " + function)
}

```

9.1.3 Chaincode Init()

```

// Init - init function
func (t *SimpleChaincode) Init(stub shim.ChaincodeStubInterface) pb.Response {
    fmt.Println("Init")
    var err error

    // Initialize the chaincode
    // Set the business index
    var business KeyIndex
    business.ObjectType = "BusinessIndex"
    businessAsBytes, _ := json.Marshal(business)

    // Write business index to ledger
    err = stub.PutState(businessIndexStr, businessAsBytes)
    if err != nil {
        return shim.Error(err.Error())
    }

    // Set the trading index
    var trading KeyIndex
    trading.ObjectType = "TradingIndex"
    tradingAsBytes, _ := json.Marshal(trading)

    // Write trading index to ledger
    err = stub.PutState(invoicesTradeStr, tradingAsBytes)
    if err != nil {
        return shim.Error(err.Error())
    }

    // Add business index and trade index to bank user
    var bank Bank
    bank.Object = "Bank"
    bank.User = "BankA"
    bank.BusinessIndex = businessIndexStr
    bank.InvoiceIndex = invoicesTradeStr
    bankAsBytes, _ := json.Marshal(bank)
    err = stub.PutState(bank.User, bankAsBytes)
    if err != nil {
        return shim.Error(err.Error())
    }

    fmt.Println("Initialization done!")
    return shim.Success(nil)
}

```

9.1.4 Chaincode Invoke()

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

    if function == "init" {
        return t.Init(stub)
    } else if function == "initUser" {
        return t.InitUser(stub, args)
    } else if function == "addCustomer" {
        return t.AddCustomer(stub, args)
    } else if function == "addInvoice" {
        return t.AddInvoice(stub, args)
    } else if function == "updateInvoice" {
        return t.UpdateInvoice(stub, args)
    } else if function == "query" {
        return t.query(stub, args)
    } else if function == "getInvoices" {
        return t.GetInvoices(stub, args)
    }

    return shim.Error("Invalid invoke function name. Expecting \"initUser\" \"addCustomer\" \"addInvoice\" \"updateInvoice\" \"query\" \"getInvoices\"")
}
```

9.1.5 Main()

```
func main() {
    err := shim.Start(new(SimpleChaincode))
    if err != nil {
        fmt.Printf("Error starting Simple chaincode: %s", err)
    }
}
```

9.1.6 SDK Init

```
// dependencies
var hfc = require('fabric-client');
var EventHub = require('fabric-client/lib/EventHub.js');
var ca = require('fabric-ca-client/lib/FabricCAClientImpl.js');
var utils = require('fabric-client/lib/utlis.js');

var config = require("../config.json");
var user = require('fabric-client/lib/User.js');
var admin, chain, client;
var ca_client;
var setup = require("../setup.json");
```



```
// Init function
exports.init = () => {
  return new Promise((resolve, reject) => {
    try {
      // Create client object
      client = new hfc();

      // Create chain object
      chain = client.newChain(config.app.channelId);

      // Add peer, orderer, ca and eventhub
      var peer = client.newPeer(config.peer.protocol + "://" +
config.peer.host + ":" + config.peer.port);
      chain.addPeer(peer);

      var orderer = client.newOrderer(config.orderer.protocol + "://" +
config.orderer.host + ":" + config.orderer.port);
      chain.addOrderer(orderer);

      var eventhub = new EventHub();
      eventhub.setPeerAddr(config.event.protocol + "://" + config.event.host
+ ":" + config.event.port);
      eventhub.connect();

      var caUrl = config.ca.protocol + "://" + config.ca.host + ":" +
config.ca.port;
      ca_client = new ca(caUrl);
    } catch (err) {
      console.log(err);
    }
    // Set Key/Val store
    return hfc.newDefaultKeyValueStore({
      path: config.app.kvsPath
    }).then((store) => {
      client.setStateStore(store);
      // Enroll admin
      return initAdmin();
    }).then((member) => {
      admin = member;

      // Initialise chain object
      return chain.initialize();
    }).then(() => {
      // get setup for users
      // test invoke
      /*
      var getUsers;
      var users = setup.initUser;
      for(user in users){
```

```

        var setUser = getArgs(users[user]);

        exports.invoke("initUser", setUser);
        console.log("Added user: " + users[user].username);
    }
    */
    //return exports.invoke("updateInvoice", ["in1235"]);
}).then(() => {
    // test query
    //return exports.query("query", ["businessuser"]);
}).then((response_payloads) => {
    return resolve(response_payloads);
}).catch((err) => {
    throw reject(err);
});
});
};

function initAdmin() {
    return new Promise((resolve, reject) => {
        console.log("Attempting to enroll: " + config.admin.usr);
        var member;
        // Enroll the client with the CA server
        return ca_client.enroll({
            enrollmentID: config.admin.usr,
            enrollmentSecret: config.admin.pwd
        }).then((enrollment) => {
            member = new user(config.admin.usr, client);
            return member.setEnrollment(enrollment.key, enrollment.certificate,
config.chain.org);
        }).then(() => {
            return client.setUserContext(member);
        }).then(() => {
            console.log("- Success!", true);
            return resolve(member);
        }).catch((err) => {
            console.log("- Failed", false);
            throw reject(err);
        });
    });
}

```

9.1.7 SDK Query

```

exports.query = (func, args) => {
    return new Promise((resolve, reject) => {

        var nonce = utils.getNonce();

        tx_id = hfc.buildTransactionID(nonce, admin);
    });
}

```

```

var request = {
  chaincodeId: config.app.chaincodeId, // mycc
  chainId: config.app.channelId,      // mych
  txId: tx_id,
  nonce: nonce,
  fcn: func,
  args: args
};
return chain.queryByChaincode(request)
  .then((query_result) => {
    return resolve(query_result);
  }).catch((err) => {
    throw reject("Error");
  });
});
};

```

9.1.8 SDK Invoke

```

exports.invoke = (func, args) => {
  var eventhubs = [];

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

    var nonce = utils.getNonce();

    tx_id = hfc.buildTransactionID(nonce, admin);
    utils.setConfigSetting('E2E_TX_ID', tx_id);

    var request = {
      chaincodeId: config.app.chaincodeId,
      chainId: config.app.channelId,
      txId: tx_id,
      nonce: nonce,
      fcn: func,
      args: args
    };
    return chain.sendTransactionProposal(request)
      .then((results) => {
        var proposalResponses = results[0];

        var proposal = results[1];
        var header = results[2];
        var allGood = true;
        // Verify proposal response from the chain
        for (var i in proposalResponses) {
          let oneGood = false;
          let proposalResponse = proposalResponses[i];
          if (proposalResponse.response &&
proposalResponse.response.status === 200) {

```

```

        console.log("Transaction proposal has response status of
good");
        oneGood = chain.verifyProposalResponse(proposalResponse);
        if (oneGood) {
            console.log("Transaction proposal signature and
endorser are valid");
        }
    } else {
        console.log("Transaction proposal failed");
    }
    allGood = allGood & oneGood;
}
if (allGood) {
    // Check all the read/write sets to see if the same, verify
that each peer
    // got the same results on the proposal
    allGood =
chain.compareProposalResponseResults(proposalResponses);
    console.log("CompareProposalResponseResults execution did not
throw an error");
    if (allGood) {
        console.log("All proposals have a matching read/writes
sets");
    } else {
        console.log("All proposals do not have matching read/writes
sets");
    }
}
if (allGood) {
    // Check to see if all results match

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

    // Set the transaction listener and set a timeout of 30sec
    // if the transaction did not get committed within the timeout
period,

    // fail the test
    var deployId = tx_id.toString();

    var eventPromises = [];
    eventhubs.forEach((eh) => {
        console.log("deployId" + deployId.toString());

        var txPromise = new Promise((resolve, reject) => {
            var handle = setTimeout(reject, 30000);

            eh.registerTXEvent(deployId.toString(), (tx, code) => {

```

```
        clearTimeout(handle);
        eh.unregisterTxEvent(deployId);
        if (code !== 'VALID') {
            console.log('The balance transfer transaction
was invalid, code = ' + code);
        } else {
            console.log('The balance transfer transaction
has been committed on peer ' + eh.ep._endpoint.addr);
            resolve();
        }
    });
    });
    eventPromises.push(txPromise);
});

var sendPromise = chain.sendTransaction(request);
return Promise.all([sendPromise].concat(eventPromises))
    .then((result) => {
        console.log('Event promise all complete and testing
complete');
        return result[0];
    }).catch((err) => {
        console.log('Failed to send transaction and get
notifications within the timeout period.');
```

```
    });
    }

    }).then(() => {
        return resolve();
    })
    .catch((err) => {
        throw reject('Error');
    });
});
});
};
```

9.2 Feedback from Equiniti

Trade Finance Demo



Foley, Michael <Michael.Foley@equiniti.com>

Today, 4:04 PM

Deborah Khoo; Davison, Peter <peter.davison@equiniti.com> ↵



Reply all | ▾

Dear Deborah,

Thank you very much for the demonstration yesterday of the Trade Finance System that you have implemented on a Hyperledger Fabric Blockchain. I was very impressed with the work, you have covered all of the requirements that we discussed resulting in a good example of how a trade finance system running on a Distributed Ledger could look.

I wish you well for the future.

Kind regards

Dr Michael Foley

Chief Strategist

Mobile +44 (0)7469 852022

Equiniti

6th Floor, The Broadgate Tower, 20 Primrose Street, London EC2A 2EW

1 Driscoll Buildings, Capital Quarter, Ellen St, Cardiff, CF10 8BP

michael.foley@equiniti.com

www.Equiniti.com

Glossary

Block

A set of transactions that is cryptographically linked to the previous block in a blockchain.

Consensus

Agreement and verification of transactions in a block.

Factoring

Short-term, non-bank financing of accounts receivable.

Factoring Company

A finance institution in which a business sells its accounts receivable to at a discount.

Hyperledger Fabric

Open source private blockchain network.

Ledger

The blockchain and current state data maintained by each peer in a peer-to-peer network.

Node

A connection point in a network.

Protocol Buffers

A method of serialising structured data.

Table of Abbreviations

AJAX

Asynchronous JavaScript and XML

API

Application Programming Interface

CA

Certificate Authority

CORS

Cross-Origin Resource Sharing

GRPC

Google Remote Procedure Call

HFC

Hyperledger Fabric Client

LDAP

Lightweight Directory Access Protocol

REST

Representational State Transfer

SDK

Software Development Kit

Works Cited

- [1] Bitcoin, "Open Source P2P money," Bitcoin, 2009. [Online]. Available: <https://bitcoin.org/en/>. [Accessed 20 3 2017].
- [2] A. Narayanan, "Cryptography and Cyptocurrencies," in *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*, E. F. A. M. S. G. Joseph Bonneau, Ed., Princeton, Princeton University Press, 2016, p. 11.
- [3] A. Narayanan and A. Miller, "Cryptocurrencies, Blockchains, and Smart Contracts," *ACMQUEUE*, vol. 60, no. 5, p. 49, 2017.
- [4] L. Luu, Chu, D-H., Olickel, H., P. Saxena and A. Hobor, "Making Smart Contracts Smarter," *In ACM SIGSAC Conference on Computer and Communications Security*, p. 254, 24-28 10 2016.
- [5] G. Volpicelli, "How the blockchain is helping stop the spread of conflict diamonds," 2017. [Online]. Available: <http://www.wired.co.uk/article/blockchain-conflict-diamonds-everledger>. [Accessed 20 4 2017].
- [6] A. Narayana and A. Miller, "Cryptocurrencies, Blockchains, and Smart Contracts," *ACMQUEUE*, vol. 60, no. 5, p. 50, 2017.
- [7] World Trade Organization, "WTO | Trade Finance," 2017. [Online]. Available: https://www.wto.org/english/thewto_e/coher_e/tr_finance_e.htm. [Accessed 1 5 2017].
- [8] A. Berke, "How Safe Are Blockchains? It Depends.," 7 3 2017. [Online]. Available: <https://hbr.org/2017/03/how-safe-are-blockchains-it-depends>. [Accessed 2 5 2017].
- [9] Staff, Entrepreneur, "Trade Credit," 13 2 2017. [Online]. Available: <https://www.entrepreneur.com/encyclopedia/trade-credit>.
- [10] Investopedia, "Factor," 2017. [Online]. Available: <http://www.investopedia.com/terms/f/factor.asp>. [Accessed 15 2 2017].
- [11] Investopedia, "Invoice Financing," 2017. [Online]. Available: <http://www.investopedia.com/terms/i/invoice-financing.asp>. [Accessed 15 2 2017].
- [12] Hyperledger, "Meet Hyperledger: An "Umbrella" for Open Source Blockchain & Smart Contract Technologies," 2016. [Online]. Available: <https://www.hyperledger.org/blog/2016/09/13/meet-hyperledger-an-umbrella-for-open-source-blockchain-smart-contract-technologies>. [Accessed 20 3 2017].
- [13] IBM Blockchain, "The Hyperledger Project," 2016. [Online]. Available: <https://www.ibm.com/blockchain/hyperledger.html>. [Accessed 12 2 2017].
- [14] Hyperledger Fabric, "Fabric-SDK-Node/web-app-developer.png," 2016. [Online]. Available: <https://github.com/hyperledger/fabric-sdk-node/blob/master/docs/images/web-app-developer.png>. [Accessed 20 2 2017].
- [15] Hyperledger-fabricdocs, "Architecture Explained," 2017. [Online]. Available: <http://hyperledger-fabric.readthedocs.io/en/latest/arch-deep-dive.html>. [Accessed 15 3 2017].
- [16] Hyperledger, "Getting Started," 2017. [Online]. Available: http://hyperledger-fabric.readthedocs.io/en/latest/getting_started.html. [Accessed 26 3 2017].

- [17] NodeJS, "Node.js," 2017. [Online]. Available: <https://nodejs.org/en/>. [Accessed 20 3 2017].
- [18] Docker, "What is Docker?," 2017. [Online]. Available: <https://www.docker.com/what-docker>. [Accessed 20 3 2017].
- [19] Hyperledger, "Setting Up a Network," 2017. [Online]. Available: <http://hyperledger-fabric.readthedocs.io/en/v0.6/Setup/Network-setup.html>. [Accessed 20 2 2017].
- [20] Hyperledger Fabric, "End-to-End Flow," 2017. [Online]. Available: https://github.com/hyperledger/fabric/blob/master/examples/e2e_cli/end-to-end.rst. [Accessed 10 4 2017].
- [21] Hyperledger-fabricdocs, "The Fabric Model," 2017. [Online]. Available: http://hyperledger-fabric.readthedocs.io/en/latest/fabric_model.html#assets. [Accessed 5 4 2017].
- [22] IBM-Blockchain, "Learn how to write chaincode," 2016. [Online]. Available: <https://github.com/IBM-Blockchain/learn-chaincode>. [Accessed 25 3 2017].
- [23] Hyperledger-fabricdocs, "What is chaincode?," 2017. [Online]. Available: <http://hyperledger-fabric.readthedocs.io/en/latest/chaincode.html#chaincode-interfaces>. [Accessed 26 3 2017].
- [24] Google, "GRPC," Google, 2015. [Online]. Available: <http://www.grpc.io/about/>. [Accessed 15 4 2017].
- [25] Hyperledger, "Hyperledger Composer," Hyperledger, 2017. [Online]. Available: <https://github.com/hyperledger/composer>. [Accessed 25 4 2017].