

Distributed Immunisation Information System

Final Year Project

198860

BSc Computer Science

School of Informatics and Engineering

Supervised by: Dr. Imran Khan

University of Sussex

2021

Chapter 1

Statements

This report is submitted as part requirement for the degree of BSc Computer Science at the University of Sussex. It is the product of my own labour except where indicated in the text. The report may be freely copied and distributed provided the source is acknowledged.



Chapter 2

Summary

This project proposes, designs and implements a Distributed-Immunisation-Information-System (DIIS). Backed by a blockchain network, built with Hyperledger Fabric, the system includes a web application that facilitates confidential storage of immunisation information. The immunisation records are stored in the ledger of the blockchain network, via invocation of smart contracts. In Hyperledger Fabric smart contracts are called chaincode. Chaincode developed for the project enables submission and validation of immunisation records, which are off limits to untrusted participants. The aim of the system is to provide an example of how current electronic health records and systems can be improved upon with the aid of distributed ledger technologies.

Contents

1	Statements	2
2	Summary	3
3	Introduction	7
4	Professional Considerations	9
4.1	BCS Code of Conduct	9
4.1.1	Legislation	10
4.1.2	Addressing Considerations	11
5	Body of Report	13
5.1	Background Information	13
5.1.1	Hyperledger Fabric	14
5.2	System Design	17
5.2.1	Network	22
5.2.2	Chaincode	24
5.3	Implementation	25
5.3.1	Network	25
5.3.2	Chaincode	25
5.3.3	Web Application	26
6	Conclusion	35
6.1	Extensions	35
7	Appendices	45
A	Project Code	46
A.1	Web Application Code	46
A.1.1	db.js	46
A.1.2	app.js	47
A.1.3	Routes	47
A.1.4	Views	48
A.1.5	records.js	49
A.1.6	Model	58

A.2	Chaincode	59
A.3	Containerisation	64
A.3.1	Dockerfile	65
A.3.2	docker-compose.yaml	65
B	Hyperledger Fabric	67
B.1	Test network	67
B.2	Fabric version	67
B.3	Test Application	67
B.3.1	Utils	67
B.4	fabric-network Node.js SDK	67
B.5	Node.js fabric-ca-client	68
B.6	Node.js fabric-common	68
B.7	Node.js fabric-protos	69
B.8	Fabric's Performance Traffic Engine - PTE	69
B.9	Configtx.yaml	69
C	Development Tools	70
C.1	Docker	70
C.2	Docker Compose	70
C.3	PlantUML	70
C.4	Diagram Sprites	70
C.5	C4-PlantUML	71
C.6	ExpressJS	71
C.7	EJS	71
C.8	MongoDB	71
C.9	MongooseJS	71
C.10	UUID Node.js	71
C.11	Crypto	71
D	Logs	72
D.1	Test Network Logs	72
D.1.1	"./network.sh up createChannel -c mychannel -ca" Output	72
D.1.2	"./network.sh deployCC -ccn record -ccp ../app/chain- code -ccl go" Output	93

List of Figures

5.1	Order-execute architecture in replicated services	15
5.2	System context diagram for DIIS	18
5.3	A lower-level system view	19
5.4	Component diagram for the DIIS	21
5.5	Diagram of the underlying blockchain network	23
5.6	Peer admin from two organisations installs the chaincode package record_1.0	26
5.7	Records view	30
5.8	Form filled	30
5.9	Setup and InitLedger output	31
5.10	GetAllAssets output	31
5.11	CreateAsset output	32
5.12	ReadAsset output	32
5.13	AssetExists output	32
5.14	AssetValid output	32
5.15	Record validation form	33
5.16	ValidAsset output	33
5.17	ValidAsset output from an expired contract	33

Chapter 3

Introduction

Inspired by the ideas of using blockchain to benefit society through decentralised applications in [1], this project seeks to design and develop a distributed solution to aid current immunisation information systems. As mentioned in [2], current implementations of IISs, especially that in developing countries, are lacking in terms of technological proficiency. As stated in [3], IISs are centralised repositories of personally identifiable vaccination information for individual members of a served population. This project aims to produce a decentralised version of an IIS, utilising Decentralised Ledger Technology (DLT) which has been highly discussed since the wide adoption of blockchain technology introduced by [4] - As discussed in [5]. From research such as [6] and [7], it is believed that blockchain provides the required security and privacy necessary for implementing these types of systems. [7] states that blockchain technology can reform the interoperability of healthcare databases. The system this project proposes uses a permissioned blockchain, in which the health authorities of different nations are trusted nodes in the network. Each health authority will provide immunisation records of their population to the ledger – such data will only be accessible by the health authority that provided it as well as the individual the record belongs to. This shall be enforced by using asymmetrical cryptography. A UUID will be required to access these records, the key being held by the individual and their relevant health authority. This enables the individual's control over their immunisation records along with the ability to provide their UUID and verify their immunisations. This, as discussed in [3], will simplify the processes of vaccination verification that may be required in pandemic scenarios, registering for school, starting with a new employer, or crossing international borders. This system would benefit bodies striving to verify individuals' immunisations internationally. This report explores the required components of such a system, the professional considerations that are necessary for building this system such as legislation and ethical concerns, along with the requirements of an IIS and that of one implemented using blockchain.

Forgery and personal data security are dominant concerns of similar projects, but such problems are routinely solved for financial and other sensitive transactions. [8] This project attempts to utilise the techniques that achieve this, in other industries, to the vaccination passport problem.

As divulged in [5], DLT designs can be instantiated as a *public* or *private* distributed ledger [9], [10]. In public DLT designs, the underlying network allows arbitrary nodes to join and participate in the distributed ledger's maintenance. No registration or verification of the nodes' identities is required. Public DLT designs are often maintained by a large number of nodes, like with Bitcoin and Ethereum. The designs enable consistent high levels of availability. To allow for a high number of (arbitrary) nodes to find consensus, the designs for public DLT should be well scalable to not deter performance as more nodes join the network. [5]

In contrast, private DLT designs engage a defined set of nodes, with each node identifiable and known to the other network nodes. This means that private DLT designs require node verification when joining the distributed ledger, for example, by using Public Key Infrastructure (PKI). PKI comprises hardware, software, policies, procedures and roles that are used for the secure electronic transfer of data by means of an insecure network. A PKI manages the creation, distribution and revocation of digital certificates, which the use of public key cryptography requires. [5]

Blockchains can execute programmable transaction logic in the form of *smart contracts* as demonstrated by Ethereum [11]. The predecessor of smart contracts were the scripts in Bitcoin, introduced in [4]. Smart contracts function as *trusted distributed applications* and gains its security from the blockchain and the underlying consensus among peers. This is similar to the approach of building resilient applications with state-machine replication (SMR) [12]. Though, blockchains are different from traditional SMR with Byzantine

Chapter 4

Professional Considerations

The system proposed in this project will process personal data, in the form of immunisation records, so necessitates consideration of ethical and legal requirements.

4.1 BCS Code of Conduct

This project has been aligned with the BCS Code of Conduct; relevant sections are as follows:

1. Public Interest You shall:
 - a have a due regard for public health, privacy, security and wellbeing of others and the environment. Encryption methods will be used where necessary to ensure the confidentiality of information in the system.
 - b have due regard for the legitimate rights of Third Parties. This system will make use of asymmetric-key encryption as well as hash functions to protect data, including that of third parties.
 - c promote equal access to the benefits of IT and seek to promote the inclusion of all sectors in society wherever opportunities arise. This project proposes the system detailed be adopted by nations to provide a means of access to personal immunisation records, enabling ease of access for the population.
2. Professional Competence and Integrity You shall:
 - a only undertake to do work or provide a service that is within your professional competence.
 - b NOT claim any level of competence that you do not possess. This project has been thoroughly considered and the conclusion has been reached that it is within my professional competence.

- c develop your professional knowledge, skills and competence on a continuing basis. Maintaining awareness of technological developments, procedures, and standards that are relevant to your field. This project explores the current solutions for immunisation information systems, evolving my professional knowledge. This project displays an awareness of technological standards and procedures necessary for a distributed IIS.
- d ensure that you have the knowledge and understanding of Legislation and that you comply with such Legislation, in carrying out your professional responsibilities. This project includes a section exploring the legislation concerning this system. Specifically, the geographic area in which this system is being produced and how legislation governs the operations of such a system.
- e respect and value alternative viewpoints and, seek, accept and offer honest criticisms of work. This project shall regularly be shared with my project supervisor to gain alternative viewpoints and criticisms, which will be implemented.
- f avoid injuring others, their property, reputation, or employment by false or malicious or negligent action or inaction. The Background Information section of the project details the necessary security methods to maintain confidentiality, integrity and authenticity in the system. Briefly, the system utilises various techniques such as hashing data, Public Key Infrastructure and anonymous data verification to maintain confidentiality when handling Personally Identifiable Information (PII).

4.1.1 Legislation

The General Data Protection Regulation (GDPR) is a privacy and security law, passed by the European Union (EU), imposes obligations onto organizations that target or collect data related to EU citizens and residents. [13]

The GDPR sets out several key principles:

1. Lawfulness, fairness and transparency
2. Purpose limitation
3. Data minimization
4. Accuracy
5. Storage limitation
6. Integrity and confidentiality
7. Accountability

These stated principles are essential to our approach to processing personal data. Compliance with the principles established in the GDPR is fundamental to ensuring good practice in data protection. In Article 35(1) of the GDPR it states that a Data Protection Impact Assessment (DPIA) is required “Where a type of processing in particular using new technologies, and taking into account the nature, scope, context and purposes of the processing, is likely to result in a high risk to the rights and freedoms of natural persons, the controller shall, prior to the processing, carry out an assessment of the impact of the envisaged processing operations on the protection of personal data. A single assessment may address a set of similar processing operations that present similar high risks.”. This asserts that if the system being designed and developed in this project is implemented in the real world performing a DPIA is mandatory. Though, a DPIA will not be necessary for this undertaking. As the GDPR is mainly concerned with the European Economic Area (EEA), producing this system in the UK brings concerns. The UK is currently in a transition period until the 31st of December 2020. At the end of this transition period the UK will become a third country. Presently, the UK is seeking adequacy decisions from the European Commission. “The effect of an adequacy decision is that personal data can be sent from an EEA state to a third country without any further safeguard being necessary” because “The European Commission has the power to determine whether a third country has an adequate level of data protection.” [14]. If the adequacy decision is not secured, by the end of the transition period, the provisions set out in [15] will take effect.

4.1.2 Addressing Considerations

Because of Art. 17, GDPRs ‘right to erasure’ [16], The system has implemented chaincode to delete assets held on the network.

Medical records are pieces of personal, sensitive data which are stored, processed, processed and transmitted in healthcare systems. Though, recital 26 is not applicable to anonymous data, maintaining data anonymity is how compliance with GDPR shall be maintained.¹

Randomized hashing offers the signer additional protection by reducing the likelihood that a preparer can generate two or more messages that ultimately yield the same hash value during the digital signature generation process – even if it is practical to find collisions for the hash function.

Hash functions are formally defined in [17] as a function $h : D \rightarrow R$ where the domain $D = \{0, 1\}^*$ and the range $R = \{0, 1\}^n$ for some $n \geq 1$ using one way hash functions, as defined by Merkle in [18], are hash functions

Storing only hashes in the database, keeps data anonymous and is complacent with the GDPR. Also, this keeps the private data away from any verifiers etc, as all they check against is a hash.

”Issues of concern are falsified or counterfeit vaccine certificates” are described

¹<https://gdpr-info.eu/recitals/no-26/>

in [19], when considering digital vaccination passports. Blockchain enables an immutable store of data, rendering any attempt at providing a spoofed record would be nullified due to the data the individual provides not being in the ledger.

Chapter 5

Body of Report

5.1 Background Information

The proceeding section attempts to provide background information for this paper.

At its core, blockchains are decentralised, distributed systems. A distributed system is a computing paradigm whereby multiple nodes work together in coordination to achieve a common goal. [20]

[21] introduced distributed networks, using a new concept in which computers are connected to other stations rather than switching points like in a centralised system.

Distributed systems are defined as a system in which hardware or software components located at networked computers communicate and coordinate their actions via message passing. [22]

As stated in [22], a distributed system must accommodate heterogeneous hardware, operating systems and networks. The networks may differ widely in performance. Systems of widely differing scales, ranging from tens of computers to millions of computers, must be supported.

Replication is key to the effectiveness of distributed systems, it can provide enhanced performance, fault tolerance and high availability. [22]

nodes in distributed systems can be honest, faulty or malicious. [20] [22] defines arbitrary failures as follows. The term arbitrary or Byzantine failure is used to describe the worst possible failure semantics, in which any type of error may occur. For example, a process may set wrong values in its data items, or it may return a wrong value in response to an invocation.

In [23] the thought experiment of The Byzantine Generals Problem was proposed. The scenario depicts a group of army generals, leading different parts of the Byzantine army plan to attack or retreat from a city. The generals can only communicate with each other via messenger. Communication is necessary in agreeing on a strategy to avoid failure. The problem being that one or more of the generals may be traitorous and is attempting to prevent the

others from reaching an agreement. This necessitates a viable mechanism that enables agreement between actors in the presence of traitors. Analogously, distributed systems require agreement amongst nodes, whilst using a channel for communication, even in the presence of Byzantine nodes. This is known as the consensus problem. [24] The consensus problem occurs when attempting to achieve reliability in a distributed system, in the presence of faulty processes. A system requires processes to reach an *agreement* on a value after one or more processes propose what the value should be. In a system such that: each process p_i communicates with other processes via message passing (assuming communication is reliable), up to some number f of the N processes are faulty, the remainder of processes are correct.

Reaching consensus is achieved as follows. Every process p_i starts in an *undecided* state and *proposes* a single value v_i . The processes communicate with each other and exchange values. Each process then sets the value of a *decision variable*, d_i . In doing so the process enters the *decided* state and can no longer change d_i . [22]

1. Termination: Eventually each correct process sets a decision variable.
2. Agreement: The decision of all correct processes is the same.
3. Integrity: If the correct processes all proposed the same value, then any correct process in the *decided* state has chosen that value. [22]

The Byzantine Generals Problem was solved in [25], where the Practical Byzantine Fault Tolerance (PBFT) algorithm was introduced.

The first practical implementation of PBFT was produced with the inception of Bitcoin, in which the Proof-of-Work (PoW) algorithm was developed as a consensus mechanism. [4]

In [26], Raft was introduced as a consensus algorithm, which produced similar results to and was as efficient as Paxos - a family of protocols for solving consensus, presented in [27]. Raft was a successful attempt at restructuring a consensus mechanism in order to enhance understandability. Raft is crash fault tolerant, but lacks in Byzantine fault tolerance. [26]

5.1.1 Hyperledger Fabric

Hyperledger Fabric's key design features, as introduced in [28], are explored next. A channel, in Hyperledger Fabric, is a private "subnet" for communication between multiple organisations on the network [29]. Assets, defined by chaincode, are exchanged across the network. These are stored as a collection of key-value pairs, represented in binary and/or JSON, with state changes recorded as transaction on a channel ledger. Chaincode is Hyperledger Fabric's version of smart contracts. Chaincode is a program, which can be written in general-programming languages Go, JavaScript (for Nodejs runtime) or Java, that implements a prescribed interface.

Chaincode, the software defining assets, also defines the transaction instructions for manipulating the assets; chaincode is the business logic.

Invocation of chaincode is recorded in a ledger. Ledgers serve as a sequenced, tamper-proof record of all state transitions which are a result of transactions [30]. Ledgers are comprised of a blockchain "chain" to store the records in blocks and a state database [31]. The chain is transaction log, structured as hash-linked blocks where each block contains a sequence of N transactions [31]. Channels are used to communicate privately between organisations, enabling privacy. Each channel contains a ledger, for that channel, peers also maintain a copy of the ledger for each channel of which they participate. This is used to maintain consistency across peers.

Hyperledger projects follow a design philosophy, which includes a modular, extensible approach; enabling interoperability. Design puts an emphasis on secure solutions, crucial to systems using sensitive data. Hyperledger's token-agnostic approach simplifies bringing blockchain to business infrastructure. [32]

Hyperledger projects embrace security by design and follow the best practices specified by the Linux Foundation's Core Infrastructure initiative [32].¹ As discussed in [28], blockchain systems typically, both permissioned and permissionless, follow the order-execute architecture. This execution style involves ordering transactions first, using a consensus protocol, then executes them in the same order on all peers sequentially.²

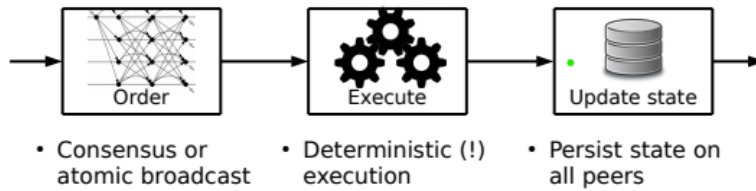


Figure 5.1: Order-execute architecture in replicated services

Source: [28]

Whilst the order-execute architecture is conceptually simple and therefore widely used, there are drawbacks which occur when using it for a general-purpose permissioned blockchain.[28] The most significant disadvantages are as follows: *Sequential execution*. Executing transactions sequentially on all peers limits the effective throughput that can be achieved on the blockchain [28]. One solution to this problem, used by Ethereum [33], utilises the concept of *gas* consumed by a transaction execution, which is converted at a *gas price* to a cost in the cryptocurrency and billed to the transaction submitter. This solution is excellent for use in permissionless

¹<https://www.coreinfrastructure.org/>

²This report follows the convention set in the Hyperledger Fabric whitepaper; naming the transaction execution (named "transaction validation" in blockchains such as Bitcoin [4]) *transaction execution* to bring the terminology together.

blockchains, though does not suffice when designing a general-purpose, permissioned, token-agnostic, blockchain like Hyperledger Fabric [28]. In Hyperledger business blockchain frameworks consensus is reached by performing two separate activities:

1. Ordering of transactions
2. Validating transactions

In the first step, the transactions are received from the client. An ordering service is used to order the transactions. To enable confidentiality, the ordering service may be agnostic to the transaction; that is, the transaction content can be hashed or encrypted.[32] This is extremely beneficial to this system, as maintaining confidentiality is essential to abiding by the GDPR [13].

Consensus in Hyperledger Fabric is further broken into 3 phases:

Endorsement, Ordering and Validation.

1. Endorsement is driven by policy (eg. m of n signatures) upon which participants endorse a transaction.
2. Ordering phase accepts the endorsed transactions and agrees to the order to be committed to the ledger.
3. Validation takes a block of ordered transactions and validates the correctness of the results, including checking endorsement policy and double-spending.

Hyperledger Fabric distributed applications consist of two parts: A smart contract, called *chaincode*, which is program code that implements the application logic and runs during the *execution phase*. As well as an *endorsement policy* that is evaluated in the *validation phase*. Endorsement only requires a subset of peers, this enables parallel execution and eliminates any non-determinism, as inconsistent results are filtered out before ordering. Because non-determinism has been eliminated, Fabric is the first blockchain technology that enables use of standard programming languages. [28] This proves useful in systems such as this, as adoption is easiest when current technologies can be used instead of domain-specific languages, like Solidity for the Ethereum blockchain [11].

The proposed system is utilising v2.3.2, at the time of writing, the latest release of Fabric. B.2 Fabric v2.0 delivers important new features and changes for users and operators. Namely, the addition of decentralised governance for smart contracts. A new implementation of the chaincode lifecycle allows multiple organisations to agree on parameters of chaincode, before it can be used to interact with the ledger. These parameters include: the chaincode package name, version, and endorsement policy [34]. Version 2.3 removed the requirement for a system channel when creating a channel, simplifying the administration process. [35]

In Fabric, to participate, every node and user that interacts with a network needs to be part of an organisation. [36]

Members of a network in Fabric enroll through a trusted Membership Service Provider (MSP). [37] An MSP serves as a trusted authority, that is in control of the governing of identities for its organisation.

An MSP is a structure of folders added to the network configuration, to define the permissions and roles in an organisation. Certificate Authorities (CAs) generate the certificates that represent identities, the MSP is where these reside. Roles in organisations are assigned to the identities, by the MSP. MSPs also hold a copy of the revoked certificates, for the organisation, which is checked whenever the certificate is attempting to be used. [38] This list is called a Certificate Revocation List (CRL) [39] and is stored by the issuing CA. [40] MSPs come in two flavors, *local MSPs* have a scope limited to the organisation they are part of and reside on the organisation's node. Every node requires an MSP. *Channel MSPs* define administrative and participatory rights at a channel level. These channel MSPs include the MSPs of the organisations on the channel, enabling the control of relationships between the channel participants. [38]

Fabric CA is the default Certificate Authority in a Fabric network. Acting as a root CA, Fabric CA issues the certificates required for the MSPs to implement identities and roles. [40]

5.2 System Design

The following section will detail the design of the proposed system. Using the C4 model for visualizing software [41], diagrams shall be generated for visualisation of the system C.5. The C4 model employs an "abstraction-first" approach to diagramming architecture [41].

The system proposed in this paper is that of an Immunisation-Information-System (IIS), which utilises blockchain to supply a reliable and tamper-proof collection of Immunisation records.

Though medical records have evolved into electronic records, the immunisation systems around the globe have not evolved [42].

Evolving these systems is a necessity for the international community to possess a fair, transparent and safe IIS, with the capabilities for citizens access to their records as well as an avenue of confirming them for those who need the verification.

Hyperledger Fabric has been chosen as the framework to build this system. Because, as discussed in [43], Hyperledger Fabric brings fine-grained control to applications which is suitable for the healthcare industry.

The proposed Distributed-Immunisation-Information-System (DIIS) shall be comprised of both a web application and a Hyperledger Fabric network. The web application will facilitate communication from users to the underlying blockchain network.

Hyperledger Fabric enables organisations to collaborate in forming, running and utilising blockchain networks [44].

The main users using the web application will be those performing administrative processes - employees of health authorities, which are acting as the organisations.

The web application shall also enable citizens access to their immunisation records, that will be entered by the health authority in the citizen's nation. To ensure all records are that of fully vaccinated patients, entering the immunisation record information should be done after all required doses are administered.

If this requirement is followed correctly, all records in the system will be that of fully vaccinated patients.

The system shall not provide any assistance in managing records for patients who have had the first of multiple doses etc.

Additionally, immunisation verifiers, such as border agents, school administrators, venue admissions staff will be able to lookup a citizen's provided universally unique identifier (UUID) to confirm their status of immunisation.

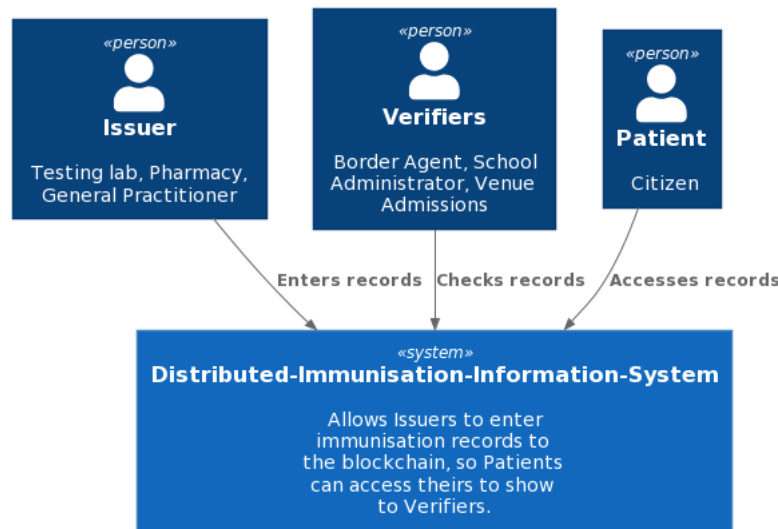


Figure 5.2: System context diagram for DIIS

Source: [28]

Figure 5.2 provides visual context to the system. As shown, immunisation record issuers are those who administer immunisations to patients. Issuers will enter the information of the immunisation record into the system, allowing patients constant access to their records. Verifiers, those who need verification of immunisation from citizens, shall be able to lookup information via credentials provided by citizens. This enables autonomy over a person's records.

Using Hyperledger Fabric maximises the potential for interoperability of Electronic Health Record (EHR) systems across borders, as it reduces the requirement for particular infrastructure. Discussed in [45], this is important to the EU as displayed in [46]. Maintaining this requirement is key for adoption of the system by EU nations.

A web application will facilitate communication with the underlying Fabric network. Organisations will have the option to develop their own applications, enabled by the nature of Open-source software. However, considerations have been made in selecting the default application SDK as the correct choice could simplify adoption. The default application will be built using the Hyperledger Fabric Nodejs SDK B.4. This decision is the result of the desire to maintain a flexible system, through ensuring the minimal requirements for adoption. Nodejs possesses the advantage in terms of ease of use, when compared to the other available SDKs such as Java and Go, as it enables the utilisation of JavaScript for both client and server-side scripting [47]. Also, Fabric has a Performance Traffic Engine tool (PTE), specific to Node B.8. Opting for the Node SDK enables the use of the PTE in testing the network.

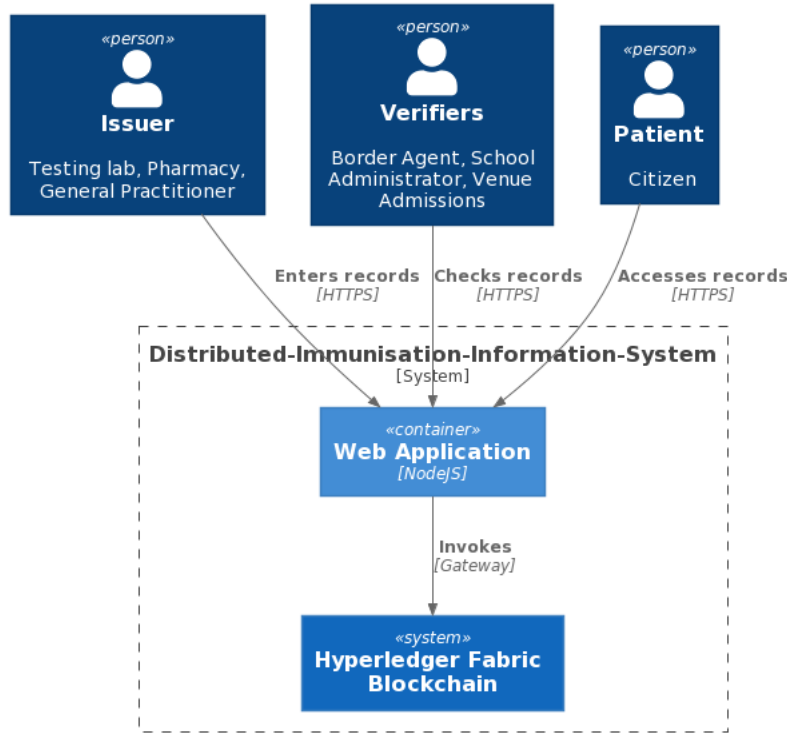


Figure 5.3: A lower-level system view

Figure 5.3 shows a lower-level view of the proposed system, including the web

application. This serves as a better picture to how this system communicates with the underlying blockchain network. All users will make use of the application as a means of either entering records, accessing records or validation of records. The web application takes advantage of a gateway to handle the interactions between the application and Fabric network B.4. A gateway is setup on the server-side of the application and uses a user's identity, stored in a wallet, to sign transactions [48]. Fabric has two types of gateway, static and dynamic. The latter shall be utilised in this application. This enables service discovery by the gateway, allowing it to find all connected peers and orderers within the network using the gossip protocol; Fabric's data dissemination protocol, which broadcasts information to and from peers in a channel [48].

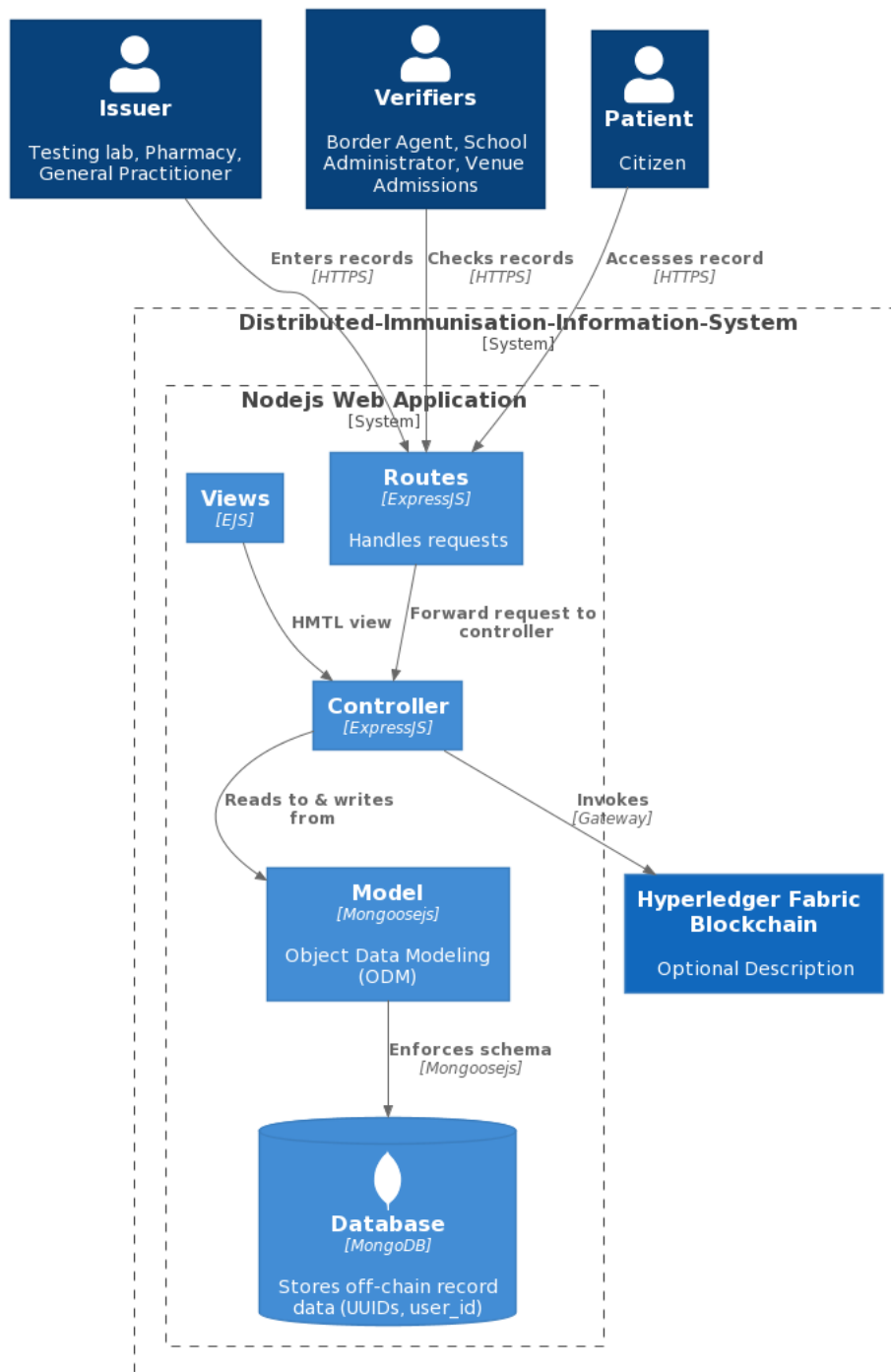


Figure 5.4: Component diagram for the DIIS

In figure 5.4 a component-level diagram is displayed, to show how the containers in figure 5.3 are constructed. Design of the web application used to communicate with the Fabric network has been achieved using the Model-View-Controller (MVC) architecture. The web application is separated into three main components, each built to handle specific tasks. This is to maintain a separation of concerns [49]. Designing this way ensures modularity [50]. Encapsulation of components is essential to ensuring minimal system-breaking bugs, that may occur when code is changed, through added flexibility **application’2001**. It also aids in keeping the system as infrastructure agnostic as possible. ExpressJS is the framework of choice, for building with Nodejs. It provides middleware that aids backend functionality by providing handling of routes, requests and views [51]. An array of template engines are available to use with Express, a selection has been made for this system to use EJS. EJS provides embedded JavaScript template functionality, using plain JavaScript enables speedy execution and simple debugging [52]. MongoDB will be used as a document-based database, enabling the use of JSON (JavaScript Object Notation [53]) to model data. Documents are stored in BSON (Binary JSON), which enables faster parsing [54]. Mongoose, an Object Data Modelling (ODM) library for MongoDB, will be used to model application data in Nodejs. Mongoose provides built-in validation and query building. [55]

Application will be containerised using Docker, building containers both for the application and database. Docker Compose allows the user to specify dependencies between services using the **depends_on** command [56]. This command shall be used to ensure the Nodejs container waits until the database container is "healthy" using a **healthcheck** [56].

Credentials for patients shall be provided in the form of a UUID (Universally Unique Identifier), defined in [57]. UUIDs will be created using a package for Node C.10. Using an encrypted email service to give patients their credentials for use in the application was debated, as the NHS already utilises this [58]. However, for the system to remain as infrastructure agnostic as possible, the credentials shall be received by patients in-person when the immunisation is administered.

5.2.1 Network

This section will explore, in detail, the backbone of the system; the Hyperledger Fabric blockchain network. The initial network for the system is based on the Fabric test network, provided in the official documentation for Hyperledger Fabric [36], contained in the "fabric-samples" repository B.1. It is kept to a limited configuration for ease of development:

1. It includes two peer organizations and an ordering organization.
2. For simplicity, a single node Raft ordering service is configured. Though, when in production the ordering nodes may also be implemented by

those organisations that wish to bypass relying on a central admin managing nodes.

3. A TLS Certificate Authority (CA) is deployed, for use with user identities.

A central, trusted authority, such as the World Health Organisation (WHO) shall maintain the ordering organisation.³

Each organisation in the network has control over its members, through Fabric's MSPs. This control also extends to the organisation's data, this shall not be shared with other organisations unless allowed by the organisation. Private channel capabilities ensure that data will only be shared with those authorised by the owning organisation.

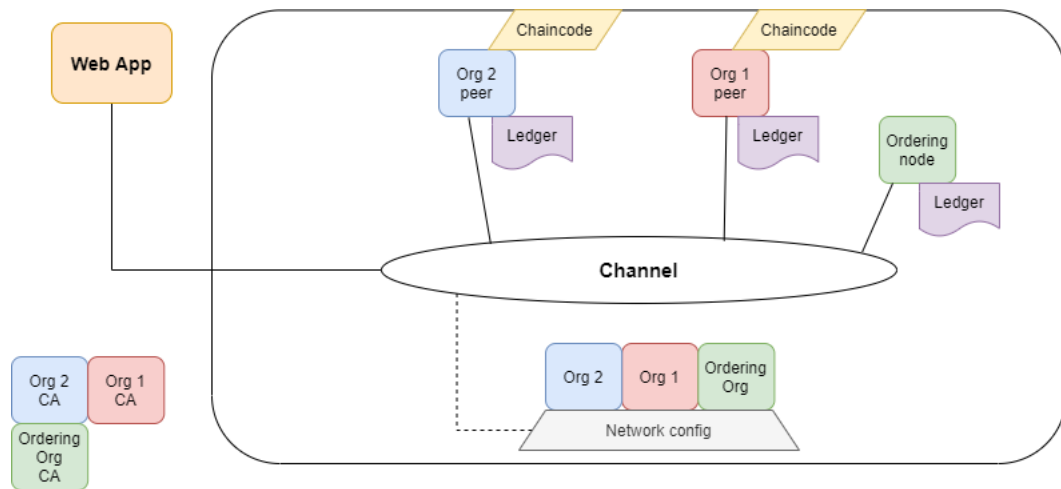


Figure 5.5: Diagram of the underlying blockchain network

Source: Adapted from [59]

Figure 5.5 shows the basic setup for the network. The organisations have established a network, which necessitates all organisations agreeing on a configuration. The configuration defines the roles each organisation plays in the network, more specifically; this is what makes the ordering organisation just that, instead of a participating organisation.

The channel configuration policy is defined in `configtx.yaml`, this can be found in the appendices B.9. Firstly, organisations are defined. With each definition for an organisation a default policy is also defined. Each policy has a type; either **Signature** or **ImplicitMeta**. **Signature** policies establish which type of user must sign for a policy to be satisfied. **ImplicitMeta** policies are valid only in the context of channel configuration, these policies

³<https://www.who.int/>

aggregate the policies defined by **Signature** policies. The implicit comes from the policy being constructed based on the policies defined in organisation definitions. **ImplicitMeta** policies are satisfied when all policies aggregated by it are satisfied. [60] When defining the orderer organisation for this network, the **Signature** policies shall require a member of the orderer organisation's MSP to sign. This is achieved using the "Rule" `"OR('OrdererMSP.member')"` in the **Signature** policy for the orderer organisation. When defining the application, policies are also defined. Utilising the **ImplicitMeta** policy type allows for reference of the sub-policies defined for each organisation [60]. Two important policies defined in the application section are **LifecycleEndorsement** and **Endorsement**. **LifecycleEndorsement** governs who needs to approve a definition for chaincode. **Endorsement** is the default chaincode endorsement policy. In this network both policies shall use the rule **MAJORITY Endorsement**, ensuring that the majority of peers belonging to the different organisations are required to execute and validate a transaction before the transaction is considered valid [60].

Figure 5.5 displays the basic setup for the network, including only two organisations for simplicity's sake. As shown, each organisation joins peers to the channel, this includes the ordering organisation. Each peer node shall contain a copy of the ledger of the channel [59]. Excluding the ordering organisation's peer, each peer also holds a state database. The state database, stores information on the world state. The world state holds the current value of attributes of the assets, this is useful as it negates the need to traverse the blockchain to gather an assets current value. [30] As shown, there will be one application utilised by the organisations to access the underlying network. All organisations possess their own CA, which generates the certificates for the nodes, admins and organisation definitions [59].

5.2.2 Chaincode

The following section will present the design of the chaincode for the system. [61] suggests that medical records should contain metadata of a patient-provider encounter (visit date/time, location, etc.), the data shall be stored off-chain and should be entered when the record is produced. [8] states that the duration of protection conferred by vaccines should be tied to expiry dates. To ensure immunisation records expire, the ledger must store metadata on the particular immunisation records to ensure they can be invalidated when the expiration is reached. This will be implemented using a timestamp, created on the client side, in the records stored on-chain.

Go has been selected as the language for writing chaincode, for this system. As discussed in [62], language choice has an impact on the latency of transactions. [62] found Go to be the best performing language available for writing Hyperledger Fabric chaincode.

The chaincode for this system will represent "Records", immunisation records, the initial chaincode package will be called "record_1.0". The implementation of this is documented in subsequent sections. Representing the records as

chaincode, or smart contracts, enables the manipulation of records by organisations that "own" them. The chaincode written for this system can be found in the appendix A.2.

5.3 Implementation

The following section will detail the implementation process of the system.

5.3.1 Network

explain bash scripts?
explain docker stuff
show generation of cryptographic things
bringing up certificate authorities
joining channels, peers contacting anchor peers (screenshots) <https://hyperledger-fabric.readthedocs.io/en/release-2.2/glossary.html>

5.3.2 Chaincode

Fabric networks use chaincode to initialise and manage the ledger state, through transactions submitted by applications. [63] To use chaincode in a network organisations need to agree to the parameters, the name of the chaincode, version and the endorsement policy. [63] explains that the agreement is reached using the following steps:

1. Package the Chaincode
2. Install the chaincode on peers: Each organisation that will utilise the specific chaincode, to endorse transactions or query ledgers, needs to execute this step.
3. Approve chaincode definition for organisation: This step also requires completion from each organisation that will use the chaincode. By default, the chaincode needs to be approved by atleast a majority of organisations before it can be deployed on a channel.
4. Commit the chaincode definition to the channel: After the chaincode is approved by a sufficient number of organisations, one organisation can commit the chaincode definition.

Chaincode needs to be packaged in a tar file for it to be installed onto peers [64]. Using the Fabric peer binaries, the following command can be used to skip manually adding the necessary "metadata.json" file which required in the tar file.

Listing 5.1: peer lifecycle chaincode package command

```
# create the chaincode package using the peer
  lifecycle chaincode package command
```

```
peer lifecycle chaincode package record.tar.gz --
  path ../asset-transfer-basic/chaincode-go/ --lang
  golang --label record_1.0
```

Though not necessary for Fabric networks, it would be useful for organisations to synchronise their labels for chaincode. This will be a standard for adopting this system.

Installation of chaincode on peers can be executed via CLI or an SDK [64]. This network relies on access via CLI, to install chaincode.

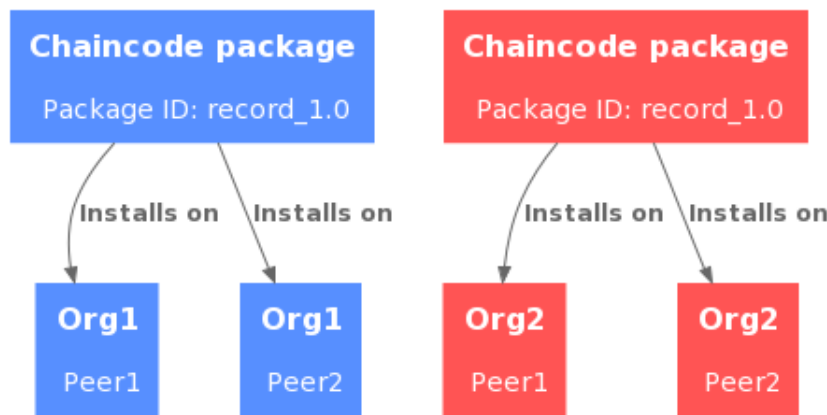


Figure 5.6: Peer admin from two organisations installs the chaincode package record_1.0

Source: Adapted from [64]

In this system it is not necessary to set the endorsement policy for the chaincode as it will be utilising the default value, which requires that a majority of organisations endorse a transaction. [64]

5.3.3 Web Application

The following section shall cover the implementation of the web application, of which the design was explored previously. Development was aided by the supplied test-application in the documentation for Hyperledger Fabric B.3. This provided sample, application uses JavaScript to interact with the underlying network via Fabric's Node SDK B.3.

The complete code for the web application can be found in the appendix A.1. Development of the application was utilising the Fabric test network B.1, provided with the official documentation for testing applications and chaincode. All that is required to use the test network are the Hyperledger Fabric Docker images and samples. [36] The topology of the test network has been explored previously.

JavaScript's "strict mode" is used throughout the application. Strict mode ensures cleaner code, by preventing the use of undeclared variables [65]. Strict mode is supported by all modern browsers, except Internet Explorer 9. Though, "use strict" is just a string, so IE 9 will not throw an error even if it does not understand it [66].

Network Deployment

The sample test network comes with provided scripts to deploy the network B.3.

To bring up the network the shell script **network.sh** is executed with the command **up**. A channel also needs creating, as does a CA for the network, so we pass the command **createChannel -c mychannel** and the flag **-ca**. The **createChannel** command also ensures the organisations' peers join the created channel [36]. The full command is

```
./network.sh up createChannel -c mychannel -ca.
```

During this step, an admin user for the organisations is bootstrapped with the CA.

Next, the chaincode, of which the implementation was detailed previously, is installed. This is performed using, again, the "network.sh" shell script. The command **deployCC** is passed, with the **-ccn** flag to specify the name of the chaincode, which is **record**. Additionally the **-ccp** flag is supplied with the command with the location of the chaincode package **../app/chaincode**, as well as the **-ccl** flag with the option **go** to indicate the language the chaincode was written in. The full command used is

```
./network.sh deployCC -ccn record -ccp ../app/chaincode -ccl go.
```

The full logs for process can be found in the appendix D.1.

Application Details

Once the test network is running, the application can be deployed.

lstinlinenode app.js starts the application, from the commandline.

The MongoDB database is connected at startup, using the **db.js** file A.1.1.

First, an object for connection to the database via mongoose is instantiated.

Connection options, for the database, are setup as constants and then provided in the url passed as a parameter to the **mongoose.connect()** function.

The main file for the app, **app.js**, imports necessary objects at the beginning of the file and can be found in the appendix A.1.2. The Express api is imported as an object, **express**, this is how an Express application is created.

[51] The Express version used in this application is 4.16.4. A router is also imported, this is how Express applications handle how the applications endpoints (URIs) respond to client requests [67]. A path for the views is then defined, next a port to open the application via http. Next, the template engine, **C.7**, is defined. This application uses EJS to dynamically serve HTML files. Using a template engine simplifies passing data to the client view from the backend. The app finally listens for connections on the previously defined port.

Routes A folder named `routes` contains two files dictating the routes for the application. `index.js`, found in appendix A.1.3, imports a router object from Express and defines the index route. The script then sends the `index.html` file as a client view. The module is then exported as `router`. The file `records.js`, found in appendix A.1.3, also imports the router object and exports the module as `router`. This file defines routes necessary for GET and POST requests used to add records, generate UUIDs, check records and serve an index page for the record manipulation functionality. These routes are responsible for calling the functions, dependant on which request is sent, defined in `records.js` A.1.5 - the file is located in the controller directory.

Controller The file `records.js` represents the controller for the application. Imports are made at the top of the file, notable imports follow. `Record` model is imported in the form of an object, for use by the controller to structure data. A package for generating UUIDs C.10. The `Gateway` and `Wallets` classes are imported from '`fabric-network`', information on these can be found in the appendices; B.4, B.4. `FabricCAServices` is an API, imported from '`fabric-ca-client`' B.5, enabling CA functionality.

Controller utility files Utility files, provided with the Hyperledger Fabric test application, have been utilised in the controller B.3.1. The file `CAUtil.js` provides functions for building a CA client, enrolling the admin user and registering as well as enrolling users. The function `registerAndEnrollUser()` first checks if the user is already enrolled, next uses an admin user to build a user object for authentication with the CA, then registers and enrolls the user. An admin account, bootstrapped with the `network.sh` up script, is used as the registrar for the CA. `enrollAdmin()` is a function that, when called, generates, locally, a private key, public key, and X.509 certificate for the admin. As the admin account was bootstrapped at startup, the admin only needs to be enrolled. Using a Certificate Signing Request, described in [68], the locally generated public key is sent to the CA which returns an encoded certificate. [69] These credentials are stored in a wallet, which is passed as a parameter along with the CA instance previously generated and the MSP name. The second utility file, `AppUtil.js` B.3.1, defines functions for building connection profiles and wallets. `buildCCPOrg1()` and `buildCCPOrg2()` both locate the relevant common connection profile for the organisation, parse the contents into a JSON object and return the built connection profile. `buildWallet()` creates a new wallet. Wallets are used to manage identities in Fabric. [70] The wallets are produced by calling `buildWallet`, with parameters: 'Wallets', a class imported from '`fabric-network`' B.4, along with a path where the wallet shall reside. B.4 If no path is provided the wallet will be created in-memory. However, providing a path is suggested as in-memory wallets are volatile and so will be lost when an application ends normally or crashes. [70] The controller builds a network configuration or 'connection profile', as an object in memory, by calling `buildCCPOrg1`. This configuration is then used as

a parameter to build an instance of the 'fabric-ca-client' B.5, which is used to enroll users. Once the admin is enrolled, the application is able to use the admin to register and enroll an application user, used to interact with the network. `registerAndEnrollUser()` is called, passing as parameters the CA instance, wallet, MSP name, userID for the organisation and affiliation label. As with the admin enrollment, this function uses a Certificate Signing Request to register and enroll the user. For simplicity's sake, these generated credentials are stored in the same wallet as the admin user. In a production environment the wallets would be held by the individuals part of the organisations, in file systems or even as Kubernetes secrets as shown in [71]. The generated credentials permission the application user to interact with chaincode functions. For this, a reference to the channel name and contract name is necessary. These requirements are fulfilled using the class Gateway, imported from 'fabric-network' B.4. The connection configuration specifies only the peer from the users' own organisation. With the setting `discovery` set to true, the node client SDK is instructed to use service discovery. This runs on the peer and fetches other peers that are currently online, along with metadata such as relevant endorsement policies and any static information it would have otherwise needed to communicate with the rest of the nodes. The `asLocalhost` settings set to true ensures connection as localhost, since the client is running on same network as the other fabric nodes. In deployments where clients are not running on the same network as the other fabric nodes, such as production environments, the `asLocalhost` option would be set to false. [69]

The controller performs these actions when using the two main functions; `submit(record)` and `check(record)`. Submitting records is done by users from the example organisation `Org1`, whereas checking records is done by a user from `Org2`. This is the separation of `Issuers` and `Verifiers` which is displayed in the design documents. In both of these instances, a network instance is gathered using the `getNetwork()` function, of the `gateway`, passing the `channelName` as a parameter. Then, the contract object is gathered using the `getContract()` function, of the `network` object, by passing the `chaincodeName` as a parameter. This contract object is used to call chaincode functions, defined in the chaincode which can be found in the appendix A.2. Two main views are used in the application, both can be found in the appendix A.1.4. `index.html` serves as an index page for the app, this is the entrance point for users on the client-side, the page includes a button for navigation to the records page. `Issuers` utilise the `records` view, to fill out a form with immunisation information and submit the supplied data to the network.

Figure 5.7: Records view

Figure 5.7 displays the **records** view, in this view there is a button to generate a UUID. When clicked, this button sends a **GET** request to **/records/genuuid**, a route that calls the **createuuid** function in the controller which returns a UUID to be used in the form.

The file **records.js**, located in the **models** directory, defines a schema for Records. The schema maps to the MongoDB collection and defines the shape of the documents in the collection [55]. The file can be found in the appendix A.1.6.

As shown in figure 5.7, the input boxes for the dates are HTML date pickers, restricting the input to that of a date format.

Figure 5.8: Form filled

Figure 5.8 shows the records form filled with data, ready to be submitted. These form values are supplied in a **POST** request, to **/addrecord**, the route handling the request calls the **create()** function which is imported to the routes files from the controller. The function is called with the parameters for the request and result, enabling the create function in the controller to instantiate a new object based on the **Record** schema. This object's **timestamp** and **expiration** attributes are parsed using the **Date.parse()** function [72], so that the Date types are replaced by the number of milliseconds since January 1, 1970. This simplifies the comparison of dates for checking expiration of immunisation records. A hash of the UUID is saved to the database and the **submit()** function in the controller is called, with the newly created object passed as a parameter. As explained previously, the setup required for producing user credentials, registering and enrolling the user is then executed. The function then, using the contract that was obtained using the gateway and then network objects, invokes the chaincode. A few functions are called, firstly, the **InitLedger** function, which sets up some dummy data in the network. The function utilises the **evaluateTransaction** function provided by the contract API. The output from this chaincode invocation, as well as the previous function calls, is displayed in figure 5.9.

```

Built a CA Client named ca-org1
Built a file system wallet at /home/email/go/src/github.com/kalne/fabric-samples/app/wallet
Successfully enrolled admin user and imported it into the wallet
Successfully registered and enrolled user appUser and imported it into the wallet

--> Submit Transaction: InitLedger, function creates the initial set of assets on the ledger
*** Result: committed

```

Figure 5.9: Setup and InitLedger output

The chaincode is available in the appendix A.2.

After initialisation of the ledger is complete, the `GetAllAssets` function is called. Again, this uses the `evaluateTransaction` function supplied by the contract API. The function returns all data initialised by the previous function, `InitLedger`. The output can be seen in figure 5.10

```

--> Evaluate Transaction: GetAllAssets, function returns all the current assets on the ledger
*** Result: [
  {
    "uuid": "0a970bbe-b436-4601-87d2-becd3bf84054",
    "dateTime": "1511382400000",
    "owner": "Jane Doe",
    "expiration": "1655510400000"
  },
  {
    "uuid": "183ce652-0788-4ea7-896b-93d6036db83e",
    "dateTime": "1611382409500",
    "owner": "Pierre Paul",
    "expiration": "1655510400000"
  },
  {
    "uuid": "3519bdc7-eb13-4ac0-bd63-137241af313b",
    "dateTime": "1621382448000",
    "owner": "Max Mustermann",
    "expiration": "1655510400000"
  },
  {
    "uuid": "6d8670a9-8550-40b9-9c57-30bb33a4d6f4",
    "dateTime": "1621382402100",
    "owner": "Zhang San",
    "expiration": "1655510400000"
  },
  {
    "uuid": "8e0c5ba3-4dfe-41fc-b454-26a3a276ac92",
    "dateTime": "1511382400120",
    "owner": "Wang Wu",
    "expiration": "1655510400000"
  },
  {
    "uuid": "f57d594d-3a88-4fcc-80a2-4147d23923e3",
    "dateTime": "1721382400000",
    "owner": "Ajay Singh",
    "expiration": "1655510400000"
  }
]

```

Figure 5.10: GetAllAssets output

Next, the **CreateAsset** function is called, with the necessary data from the form in the **records** view passed as parameters. This time, the **submitTransaction** function from the contract API is used. This processes commits the transaction, with the newly created asset, to the ledger. The output for this function can be seen in figure 5.11.

```
--> Submit Transaction: CreateAsset, creates new asset with UUID, timestamp, owner and expiration arguments
*** Result: committed
```

Figure 5.11: CreateAsset output

The function **ReadAsset** is next to be called, this function takes the record UUID as a parameter. This value is used to lookup the corresponding record in the ledger. The **ReadAsset** function uses the context supplied by the contract API to find the record and returns it to the controller. The results retrieved are shown in figure 5.12.

```
--> Evaluate Transaction: ReadAsset, function returns an asset with a given assetID
*** Result: {
  "uuid": "1319a349-e2c9-49ee-a6fa-1f9a22d882d9",
  "dateTime": "1621296000000",
  "owner": "Greg",
  "expiration": "1655510400000"
}
```

Figure 5.12: ReadAsset output

Following is the call to the **AssetExists** function, which checks a record UUID against the ledger to find if the corresponding asset is present. A boolean is returned, dependant on if the asset was found. The output for this command is shown in figure 5.13.

```
--> Evaluate Transaction: AssetExists, function returns "true" if an asset with given assetID exist
*** Result: true
```

Figure 5.13: AssetExists output

Finally, the function **AssetValid** is called, this function takes a UUID as an input parameter, finds the relevant asset and unmarshals the JSON representation of the asset. Then, the values for **timestamp** and **expiration** are converted to type **Uint64**, compared to check that the timestamp is smaller than the expiration and returns true if so. This function is the way records are checked for validity, to ensure that the immunisation record has not expired. The output for this function is shown in figure 5.14.

```
--> Evaluate Transaction: AssetValid, function returns "true" if an asset with given assetID exist
*** Result: true
|
```

Figure 5.14: AssetValid output

Utilising the **AssetValid** function from the chaincode is key in verifying records, without the need for the verifier to ever handle the information in the ledger. Patients with immunisation records in the ledger can provide their UUID for verifiers to enter in the alternative form on the records view. For demonstration purposes an existing record, produced by the **InitLedger** function shall be used to check the validity of the record. The view, with an entered UUID can be seen in figure 5.15.

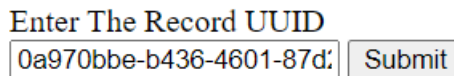
A web form titled "Enter The Record UUID" in blue text. Below the title is a text input field containing the UUID "0a970bbe-b436-4601-87d". To the right of the input field is a button labeled "Submit".

Figure 5.15: Record validation form

On submit of the form, a **POST** request is made to **/checkrecord** with the data in the request body. The route handling this request calls the **check** function, imported from the controller. This function first creates a new **Record** object, utilising the schema described previously, using the data from the request body. The object is used as a parameter to the **find** function, provided by Mongoose [73], to check the record is held in the database. Next, the **checkRecord()** function is called, passing the newly generated **Record** object as a parameter. A hash of the UUID is stored in the database. Again, setup is performed via the formerly described sequence of functions for enrolling and registering a user and instantiating a gateway object. These credentials are then used to connect to the gateway, enabling access to the underlying network via chaincode invocation. The same functions ran previously are again called, the important one being the **AssetValid** which looks up the record corresponding to the provided UUID in the ledger, then returns a boolean value showing the validity of the immunisation record. A simple comparison is made to check that the **timestamp** date is before the **expiration** date. The output generating by this function is displayed in figure 5.16.

```
--> Evaluate Transaction: AssetValid, function returns "true" if an asset with given assetID exist  
*** Result: true
```

Figure 5.16: ValidAsset output

To demonstrate the result gained if an expired immunisation record is attempted to be verified, another call to the function is made using a record which is expired. The resulting output is available in figure 5.17. As shown in the figure, the function has responded with **false** to indicate that the expiration date on the record has been reached.

```
--> Evaluate Transaction: AssetValid, function returns "true" if an asset with given assetID exist  
*** Result: false
```

Figure 5.17: ValidAsset output from an expired contract

Containerisation Code for the files utilised by docker during the containerisation of the web application can be found in the appendix A.3. Containersation of the web application running on Node was successful, the files in the appendix can be utilised to compose all necessary services for the web application. This includes the MongoDB database. However, interaction with the Hyperledger Fabric network was not implemented and so the application is relying on the CLI instead.

Chapter 6

Conclusion

The key objectives for this project have been met. Though, complete oversight of the complexities of some of the components, coupled with poor project management, has resulted in a simplified version of the original proposal. The system does maintain confidentiality of information, to abide by the previously discussed requirements such as the GDPR [13]. Personally identifiable information, stored in the database, has been hashed first and all necessary references and comparisons to this information utilise the hash instead of the plaintext data.

Even though there is additional functionality missing, the project has completed what it set out to achieve. The system designed and implemented in this paper may be simple but it meets all the requirements for the basis of a Distributed-Immunisation-Information-System.

A key feature which was discarded, due to improper planning for development, is an avenue for patients to report adverse effects of their immunisations. Literature such as [74], made clear the advantages to including this in an immunisation information system. This feature is crucial to harvesting information for use in data analysis.

Neglecting important parts of the initial interim report submission, such as the gant chart for planning of design and implementation, has lead to a lack of structure in terms of planning in the project. Ensuring proper scheduling for development would have helped the project tremendously. This also could have been avoided if communication with the project supervisor was not completely lacking.

6.1 Extensions

Unfortunately, due to poor project management and communication, additional features were not implemented or detailed in the report and are instead included in the following text as extensions to the project.

Private data collections were neglected during implementation, this is a must

for any evolution on this system. Whilst penetration testing Hyperledger Fabric, [75] found that there is a high severity issue; the potential to guess the content of a Private Data Collection. This is achieved using SHA256 as an oracle. To ensure this vulnerability is not available in the system, it is necessary to ensure any private data is salted before being hashed. Containersation of a Hyperledger Fabric network is an important extension, to utilise the containerised web application.

Adverse report avenues via mobile application, or similar, should be implemented as an extension.

Currently, medical data has the HL7 standard as a data exchange format, but the exchange of medical records between countries is not widespread [76].

FHIR, the current standards for the sharing of healthcare information should be implemented in this system [77]. This would provide a simple format of data, for organisations joining the network, that health authorities are already familiar with.

Deploying a production scale network and application, using Blockchain-as-a-Service (BaaS) platforms is left as an extension. To ensure all organisations can setup the necessary infrastructure, it would be most simple to use a BaaS platform. IBM Blockchain Platform has excellent documentation and provides code patterns for a range of different tasks such as performance analytics.¹

Implementing Zero Knowledge Proofs (ZKP) would be left as an extension, as the complexity of this feature was overlooked early on. The reason for this is that Fabric uses Idemix², but the technology supports only a limited amount of attributes. So implementation would take a considerable amount of time to get right using this. Otherwise, there are technologies that can facilitate ZKP with some customisation such as Hyperledger Indy³.

Digital signatures should have been detailed and implemented. Using the NIST Special Publication "Randomized Hashing Digital Signatures". This documents the basic randomization technique for use with digital signatures, which should have been included in the report.⁴

¹<https://developer.ibm.com/patterns/use-db2-and-sql-to-perform-analytics-on-blockchain-transactions/>

²<https://www.cise.ufl.edu/~nemo/anonymity/papers/idemix.pdf>

³<https://www.hyperledger.org/blog/2019/04/25/research-paper-validating-confidential-blockchain-transactions-with-zero-knowledge-proof>

⁴<http://csrc.nist.gov/publications/drafts/Draft-SP-800-106/Draft-SP800-106.pdf>

Bibliography

- [1] S. Huckle, R. Bhattacharya, M. White, and N. Beloff, “Internet of things, blockchain and shared economy applications,” *Procedia Computer Science*, vol. 98, pp. 461–466, Dec. 31, 2016. DOI: 10.1016/j.procs.2016.09.074.
- [2] W. H. Organization. (May 2013). Better_immunization_information_systems.pdf, [Online]. Available: https://www.who.int/immunization/programmes_systems/supply_chain/optimize/better_immunization_information_systems.pdf?ua=1 (visited on 11/18/2020).
- [3] (). The digital immunization system of the future: Imagining a patient-centric, interoperable immunization information system - katherine m. atkinson, salima saleem mithani, cameron bell, taylor rubens-augustson, kumanan wilson, 2020, [Online]. Available: <https://journals.sagepub.com/doi/10.1177/2515135520967203> (visited on 05/18/2021).
- [4] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” Manubot, Nov. 20, 2019, Publication Title: Manubot. [Online]. Available: <https://git.dhimmel.com/bitcoin-whitepaper/> (visited on 11/18/2020).
- [5] A. Sunyaev, “Distributed ledger technology,” in *Internet Computing: Principles of Distributed Systems and Emerging Internet-Based Technologies*, Cham: Springer International Publishing, 2020, pp. 265–299, ISBN: 978-3-030-34957-8. DOI: 10.1007/978-3-030-34957-8_9. [Online]. Available: https://doi.org/10.1007/978-3-030-34957-8_9.
- [6] N. Elisa, L. Yang, F. Chao, and Y. Cao, “A framework of blockchain-based secure and privacy-preserving e-government system,” *Wireless Networks*, Dec. 3, 2018, ISSN: 1572-8196. DOI: 10.1007/s11276-018-1883-0. [Online]. Available: <https://doi.org/10.1007/s11276-018-1883-0> (visited on 11/10/2020).

- [7] S. Tanwar, K. Parekh, and R. Evans, "Blockchain-based electronic healthcare record system for healthcare 4.0 applications," *Journal of Information Security and Applications*, vol. 50, p. 102407, Feb. 1, 2020, ISSN: 2214-2126. DOI: 10.1016/j.jisa.2019.102407. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S2214212619306155> (visited on 11/18/2020).
- [8] C. Dye and M. C. Mills, "COVID-19 vaccination passports," *Science*, vol. 371, no. 6535, pp. 1184–1184, Mar. 19, 2021, Publisher: American Association for the Advancement of Science, ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.abi5245. [Online]. Available: <https://science.sciencemag.org/content/371/6535/1184> (visited on 04/23/2021).
- [9] K. Yeow, A. Gani, R. W. Ahmad, J. J. P. C. Rodrigues, and K. Ko, "Decentralized consensus for edge-centric internet of things: A review, taxonomy, and research issues," *IEEE Access*, vol. 6, pp. 1513–1524, 2018, Conference Name: IEEE Access, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2779263.
- [10] X. Xu, I. Weber, M. Staples, L. Zhu, J. Bosch, L. Bass, C. Pautasso, and P. Rimba, "A taxonomy of blockchain-based systems for architecture design," in *2017 IEEE International Conference on Software Architecture (ICSA)*, Apr. 2017, pp. 243–252. DOI: 10.1109/ICSA.2017.33.
- [11] (). Ethereum whitepaper, ethereum.org, [Online]. Available: <https://ethereum.org> (visited on 04/22/2021).
- [12] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: A tutorial," *ACM Computing Surveys*, vol. 22, no. 4, pp. 299–319, Dec. 1, 1990, ISSN: 0360-0300. DOI: 10.1145/98163.98167. [Online]. Available: <https://doi.org/10.1145/98163.98167> (visited on 04/22/2021).
- [13] (). General data protection regulation (GDPR) – official legal text, General Data Protection Regulation (GDPR), [Online]. Available: <https://gdpr-info.eu/> (visited on 04/04/2021).
- [14] (Sep. 11, 2020). Information rights at the end of the transition period - frequently asked questions. Publisher: ICO, [Online]. Available: <https://ico.org.uk/for-organisations/data-protection-at-the-end-of-the-transition-period/information-rights-at-the-end-of-the-transition-period-frequently-asked-questions/> (visited on 11/17/2020).
- [15] *Agreement on the withdrawal of the United Kingdom of Great Britain and: Northern Ireland from the European Union and the European Atomic Energy Community*. 2019, OCLC: 1232719720, ISBN: 978-1-5286-1639-3.

- [16] (). Art. 17 GDPR – right to erasure (‘right to be forgotten’), General Data Protection Regulation (GDPR), [Online]. Available: <https://gdpr-info.eu/art-17-gdpr/> (visited on 05/18/2021).
- [17] B. V. Rompay and K. U. L. Esat, “Analysis and Design of Cryptographic Hash Functions, MAC Algorithms and Block Ciphers,” p. 259,
- [18] R. C. Merkle, “Secrecy, authentication, and public key systems.,” AAI8001972, PhD thesis, Stanford University, Stanford, CA, USA, 1979, 187 pp.
- [19] P. Schlagenhauf, D. Patel, A. J. Rodriguez-Morales, P. Gautret, M. P. Grobusch, and K. Leder, “Variants, vaccines and vaccination passports: Challenges and chances for travel medicine in 2021,” *Travel Medicine and Infectious Disease*, vol. 40, p. 101996, 2021, ISSN: 1477-8939. DOI: 10.1016/j.tmaid.2021.101996. [Online]. Available: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7899929/> (visited on 04/03/2021).
- [20] I. Bashir, *Mastering Blockchain*. Packt Publishing Ltd, Mar. 17, 2017, 531 pp., Google-Books-ID: urkrDwAAQBAJ, ISBN: 978-1-78712-929-0.
- [21] P. Baran, “On distributed communications networks,” *IEEE Transactions on Communications Systems*, vol. 12, no. 1, pp. 1–9, Mar. 1964, Conference Name: IEEE Transactions on Communications Systems, ISSN: 1558-2647. DOI: 10.1109/TCOM.1964.1088883.
- [22] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, *Distributed Systems: Concepts and Design*. Pearson Education, Nov. 21, 2011, 1066 pp., Google-Books-ID: 3ZouAAAAQBAJ, ISBN: 978-0-13-300137-2.
- [23] L. Lamport, R. Shostak, and M. Pease, “The byzantine generals problem,” *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, Jul. 1, 1982, ISSN: 0164-0925. DOI: 10.1145/357172.357176. [Online]. Available: <https://doi.org/10.1145/357172.357176> (visited on 04/10/2021).
- [24] M. J. Fischer, “The consensus problem in unreliable distributed systems (a brief survey),” in *Foundations of Computation Theory*, M. Karpinski, Ed., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 1983, pp. 127–140, ISBN: 978-3-540-38682-7. DOI: 10.1007/3-540-12689-9_99.
- [25] M. Castro and B. Liskov, “Practical byzantine fault tolerance,” in *Proceedings of the third symposium on Operating systems design and implementation*, ser. OSDI ’99, USA: USENIX Association, Feb. 22, 1999, pp. 173–186, ISBN: 978-1-880446-39-3. (visited on 04/24/2021).
- [26] D. Ongaro and J. Ousterhout, “In search of an understandable consensus algorithm,” in *Proceedings of the 2014 USENIX conference on USENIX Annual Technical Conference*, ser. USENIX ATC’14, USA: USENIX Association, Jun. 19, 2014, pp. 305–320, ISBN: 978-1-931971-10-2. (visited on 04/24/2021).

- [27] M. Pease, R. Shostak, and L. Lamport, “Reaching agreement in the presence of faults,” *Journal of the ACM*, vol. 27, no. 2, pp. 228–234, Apr. 1, 1980, ISSN: 0004-5411. DOI: 10.1145/322186.322188. [Online]. Available: <https://doi.org/10.1145/322186.322188> (visited on 04/24/2021).
- [28] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, and J. Yellick, “Hyperledger fabric: A distributed operating system for permissioned blockchains,” *Proceedings of the Thirteenth EuroSys Conference*, pp. 1–15, Apr. 23, 2018. DOI: 10.1145/3190508.3190538. arXiv: 1801.10228. [Online]. Available: <http://arxiv.org/abs/1801.10228> (visited on 11/19/2020).
- [29] (). Channels — hyperledger-fabricdocs master documentation, [Online]. Available: <https://hyperledger-fabric.readthedocs.io/en/release-2.2/channels.html> (visited on 05/15/2021).
- [30] (). Ledger — hyperledger-fabricdocs master documentation, [Online]. Available: <https://hyperledger-fabric.readthedocs.io/en/release-2.2/ledger.html> (visited on 05/14/2021).
- [31] (). Hyperledger fabric model — hyperledger-fabricdocs master documentation, [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.2/fabric_model.html (visited on 04/25/2021).
- [32] “Hyperledger architecture, volume 1,” Aug. 2017. [Online]. Available: https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf (visited on 04/11/2021).
- [33] D. G. Wood, “ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER,” p. 39,
- [34] (). What’s new in hyperledger fabric v2.x — hyperledger-fabricdocs master documentation, [Online]. Available: <https://hyperledger-fabric.readthedocs.io/en/release-2.2/whatsnew.html> (visited on 04/22/2021).
- [35] (). Create a channel using the test network — documentation hyperledger-fabricdocs master, [Online]. Available: https://hyperledger-fabric.readthedocs.io/fr/release-2.2/create_channel/create_channel_test_net.html (visited on 05/17/2021).

- [36] (). Using the fabric test network — hyperledger-fabricdocs master documentation, [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/latest/test_network.html (visited on 04/23/2021).
- [37] (). Introduction — hyperledger-fabricdocs master documentation, [Online]. Available: <https://hyperledger-fabric.readthedocs.io/en/release-2.2/blockchain.html> (visited on 04/25/2021).
- [38] (). Membership service provider (MSP) — hyperledger-fabricdocs master documentation, [Online]. Available: <https://hyperledger-fabric.readthedocs.io/en/release-2.2/membership/membership.html> (visited on 04/25/2021).
- [39] D. Cooper. (). Internet x.509 public key infrastructure certificate and certificate revocation list (CRL) profile, [Online]. Available: <https://tools.ietf.org/html/rfc5280> (visited on 04/24/2021).
- [40] (). Identity — hyperledger-fabricdocs master documentation, [Online]. Available: <https://hyperledger-fabric.readthedocs.io/en/release-2.2/identity/identity.html> (visited on 04/24/2021).
- [41] (). The c4 model for visualising software architecture, [Online]. Available: <https://c4model.com/> (visited on 05/14/2021).
- [42] A. Abbas and S. U. Khan, “A review on the state-of-the-art privacy-preserving approaches in the e-health clouds,” *IEEE journal of biomedical and health informatics*, vol. 18, no. 4, pp. 1431–1441, Jul. 2014, ISSN: 2168-2208. DOI: 10.1109/JBHI.2014.2300846.
- [43] H. Yu, H. Sun, D. Wu, and T.-T. Kuo, “Comparison of smart contract blockchains for healthcare applications,” *AMIA Annual Symposium Proceedings*, vol. 2019, pp. 1266–1275, Mar. 4, 2020, ISSN: 1942-597X. [Online]. Available: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7153130/> (visited on 04/03/2021).
- [44] (). Blockchain network — hyperledger-fabricdocs master documentation, [Online]. Available: <https://hyperledger-fabric.readthedocs.io/en/release-2.2/network/network.html#network-summary> (visited on 05/14/2021).
- [45] P. Kierkegaard, “Electronic health record: Wiring europe’s healthcare,” *Computer Law & Security Review*, vol. 27, no. 5, pp. 503–515, Sep. 1, 2011, ISSN: 0267-3649. DOI: 10.1016/j.clsr.2011.07.013. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0267364911001257> (visited on 02/24/2021).

- [46] *Directive 2011/24/EU of the european parliament and of the council of 9 march 2011 on the application of patients' rights in cross-border healthcare*, Code Number: 088, Apr. 4, 2011. [Online]. Available: <http://data.europa.eu/eli/dir/2011/24/oj/eng> (visited on 05/15/2021).
- [47] Node.js. (). Documentation, Node.js, [Online]. Available: <https://nodejs.org/en/docs/> (visited on 05/15/2021).
- [48] (). Gateway — hyperledger-fabricdocs master documentation, [Online]. Available: <https://hyperledger-fabric.readthedocs.io/en/release-2.2/developapps/gateway.html> (visited on 05/15/2021).
- [49] M. Richards, *Software Architecture Patterns*. O'Reilly Media, Incorporated, 2015, book, Google-Books-ID: AtF2nQAACAAJ, ISBN: 978-1-4919-7143-7.
- [50] P. A. Laplante, *What Every Engineer Should Know about Software Engineering*. CRC Press, Apr. 25, 2007, 330 pp., Google-Books-ID: pFHYk0KWAEgC, ISBN: 978-1-4200-0674-2.
- [51] (). Express middleware, [Online]. Available: <https://expressjs.com/en/resources/middleware.html> (visited on 05/14/2021).
- [52] (). EJS – embedded JavaScript templates, [Online]. Available: <https://ejs.co/#about> (visited on 05/14/2021).
- [53] (). JSON, [Online]. Available: <https://www.json.org/json-en.html> (visited on 05/12/2021).
- [54] (). JSON and BSON, MongoDB, [Online]. Available: <https://www.mongodb.com/json-and-bson> (visited on 05/12/2021).
- [55] (). Mongoose v5.12.9: Schemas, [Online]. Available: <https://mongoosejs.com/docs/guide.html> (visited on 05/18/2021).
- [56] (May 10, 2021). Compose file version 3 reference, Docker Documentation, [Online]. Available: <https://docs.docker.com/compose/compose-file/compose-file-v3/> (visited on 05/12/2021).
- [57] (). Rfc4122, [Online]. Available: <https://datatracker.ietf.org/doc/html/rfc4122> (visited on 05/12/2021).
- [58] (). Guidance for sending secure email (including to patients), NHS Digital, [Online]. Available: <https://digital.nhs.uk/services/nhsmail/guidance-for-sending-secure-email> (visited on 05/12/2021).
- [59] (). How fabric networks are structured — hyperledger-fabricdocs main documentation, [Online]. Available: <https://hyperledger-fabric.readthedocs.io/en/latest/network/network.html> (visited on 05/15/2021).

- [60] (). Policies — hyperledger-fabricdocs master documentation, [Online]. Available: <https://hyperledger-fabric.readthedocs.io/en/release-2.2/policies/policies.html> (visited on 05/16/2021).
- [61] S. Alexaki, G. Alexandris, V. Katos, and N. E. Petroulakis, “Blockchain-based electronic patient records for regulated circular healthcare jurisdictions,” in *2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, ISSN: 2378-4873, Sep. 2018, pp. 1–6. DOI: 10.1109/CAMAD.2018.8514954.
- [62] L. Foschini, A. Gavagna, G. Martuscelli, and R. Montanari, “Hyperledger fabric blockchain: Chaincode performance analysis,” in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, ISSN: 1938-1883, Jun. 2020, pp. 1–6. DOI: 10.1109/ICC40277.2020.9149080.
- [63] (). Writing your first chaincode — hyperledger-fabricdocs master documentation, [Online]. Available: <https://hyperledger-fabric.readthedocs.io/en/release-2.2/chaincode4ade.html> (visited on 04/26/2021).
- [64] (). Fabric chaincode lifecycle — hyperledger-fabricdocs master documentation, [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.2/chaincode_lifecycle.html (visited on 04/26/2021).
- [65] (). ReferenceError: Assignment to undeclared variable "x" - JavaScript — MDN, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Errors/Undeclared_var (visited on 05/10/2021).
- [66] (). JavaScript "use strict", [Online]. Available: https://www.w3schools.com/js/js_strict.asp (visited on 05/10/2021).
- [67] (). Express routing, [Online]. Available: <https://expressjs.com/en/guide/routing.html> (visited on 05/18/2021).
- [68] (). Rfc2986, [Online]. Available: <https://datatracker.ietf.org/doc/html/rfc2986> (visited on 05/10/2021).
- [69] (). Running a fabric application — hyperledger-fabricdocs main documentation, [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/latest/write_first_app.html (visited on 05/17/2021).
- [70] (). Wallet — hyperledger-fabricdocs master documentation, [Online]. Available: <https://hyperledger-fabric.readthedocs.io/en/release-2.2/developapps/wallet.html> (visited on 05/10/2021).

- [71] (). Use kubernetes secrets as hyperledger fabric wallet using fabric java SDK, IBM Developer, [Online]. Available: <https://developer.ibm.com/solutions/security/patterns/use-kubernetes-secrets-as-hyperledger-fabric-wallet-using-fabric-java-sdk/> (visited on 05/18/2021).
- [72] (). Date.parse() - JavaScript — MDN, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/parse (visited on 05/18/2021).
- [73] (). Mongoose v5.12.9: API docs, [Online]. Available: https://mongoosejs.com/docs/api.html#model_Model.find (visited on 05/18/2021).
- [74] European Centre for Disease Prevention and Control., *Designing and implementing an immunisation information system: a handbook for those involved in the design, implementation or management of immunisation information systems*. LU: Publications Office, 2018. [Online]. Available: <https://data.europa.eu/doi/10.2900/349426> (visited on 05/18/2021).
- [75] G. Shaw, “Penetration testing technical report,” p. 20, Aug. 8, 2019.
- [76] H. H. Kung, Y.-F. Cheng, H.-A. Lee, and C.-Y. Hsu, “Personal health record in FHIR format based on blockchain architecture,” in *Frontier Computing*, J. C. Hung, N. Y. Yen, and J.-W. Chang, Eds., ser. Lecture Notes in Electrical Engineering, Singapore: Springer, 2020, pp. 1776–1788, ISBN: 9789811532504. DOI: 10.1007/978-981-15-3250-4_237.
- [77] (). Overview - FHIR v4.0.1, [Online]. Available: <https://www.hl7.org/fhir/overview.html> (visited on 05/14/2021).

Chapter 7

Appendices

Appendix A

Project Code

A.1 Web Application Code

The full code for the web application developed for the project; uses Node, ExpressJS, MongoDB. ¹

Listing A.1: Web application for DIIS.

A.1.1 db.js

```
const mongoose = require('mongoose');

// require('dotenv').config()

/*
const {
  MONGO_USERNAME,
  MONGO_PASSWORD,
  MONGO_HOSTNAME,
  MONGO_PORT,
  MONGO_DB
} = process.env;
*/

const MONGO_USERNAME = 'user';
const MONGO_PASSWORD = 'userpassword123';
const MONGO_HOSTNAME = '127.0.0.1';
const MONGO_PORT = '27017';
const MONGO_DB = 'diis';
```

¹Aided by official documentation for Hyperledger Fabric "https://hyperledger-fabric.readthedocs.io/en/latest/write_first_app.html"

```

const options = {
  useNewUrlParser: true,
  reconnectTries: Number.MAX_VALUE,
  reconnectInterval: 500,
  connectTimeoutMS: 10000,
};

const url = 'mongodb://${MONGO_USERNAME}:${MONGO_PASSWORD}@${MONGO_HOSTNAME}:${MONGO_PORT}/${MONGO_DB}?authSource=admin';

mongoose.connect(url, options).then(function () {
  console.log('MongoDB is connected');
})
  .catch(function (err) {
    console.log(err);
  });

```

A.1.2 app.js

```

const express = require('express');
const app = express();
const router = express.Router();
const db = require('./db');
const recordRouter = require('./routes/records');

const path = __dirname + '/views/';
const port = 3000;

app.engine('html', require('ejs').renderFile);
app.set('view engine', 'html');
app.use(express.urlencoded({ extended: true }));
app.use(express.static(path));
app.use('/records', recordRouter);

app.listen(port, function () {
  console.log('Example app listening on port 3000!');
});

```

A.1.3 Routes

index.js

```

const express = require('express');
const router = express.Router();

```



```

const path = require('path');

router.use (function (req,res,next) {
  console.log('/') + req.method);
  next();
});

router.get('/',function(req,res){
  res.sendFile(path.resolve('views/index.html'));
});

module.exports = router;

```

records.js

```

const express = require('express');
const router = express.Router();
const record = require('../controllers/records');

router.get('/', function(req, res){
  record.index(req,res);
});

router.post('/addrecord', function(req, res) {
  record.create(req,res);
});

router.get('/genuuid', function(req, res) {
  record.createuuid(req,res);
});

router.post('/checkrecord', function(req, res) {
  record.check(req,res);
});

module.exports = router;

```

A.1.4 Views

This section of the appendix displays the code used to produce the client views, using a template engine.

index.html

```

<!DOCTYPE html>
<body>
  <a href="/records" role="button">Records</a>

```

```

    </body>
</html>

```

records.html

```

<!DOCTYPE html>
<form action="/records/genuuid" method="get">
    <button type="submit">Get UUID</button>
</form>
<form action="/records/addrecord" method="post">
    <div class="caption">Enter Your Record</div>
    <input type="text" placeholder="Record UUID"
        name="uuid" <%=records[i].uuid; %>
    <input type="date" placeholder="Record
        timestamp" name="timestamp" <%=records[i].
        timestamp; %>
    <input type="text" placeholder="Record owner"
        name="owner" <%=records[i].owner; %>
    <input type="text" placeholder="Record
        expiration" name="exp" <%=records[i].exp;
        %>
    <button type="submit">Submit</button>
</form>

<form action="/records/checkrecord" method="post">
    <div class="caption">Enter The Record UUID</
    div>
    <input type="text" placeholder="Record UUID"
        name="uuid" <%=records[i].uuid; %>
    <button type="submit">Submit</button>
</form>
</html>

```

A.1.5 records.js

The following code is that of the controller for the application.

```

'use strict';

const path = require('path');
const Record = require('../models/records');
const uuid4 = require('uuid').v4;

const {
    createHash,
} = require('crypto');

const hash = createHash('sha256');

```

```

const { Gateway, Wallets } = require('fabric-network');
const FabricCAServices = require('fabric-ca-client');
const { buildCAClient, registerAndEnrollUser,
  enrollAdmin } = require('../utils/CAUtil.js');
const { buildCCPOrg1, buildWallet, buildCCPOrg2 } =
  require('../utils/AppUtil.js');

const channelName = 'mychannel';
const chaincodeName = 'record';
const mspOrg1 = 'Org1MSP';
const mspOrg2 = 'Org2MSP';
const walletPath = path.join(__dirname, '..', 'wallet');
const org1UserId = 'appUser';
const org2UserId = 'verifier';

function prettyJSONString(inputString) {
  return JSON.stringify(JSON.parse(inputString),
    null, 2);
}

/**
 * @param record A record produced using the body
 * of the request in the create function
 */
async function submit(record) {
  try {
    // build an in memory object with the
    // network configuration (also known as a
    // connection profile)
    const ccp = buildCCPOrg1();

    // build an instance of the fabric ca
    // services client based on
    // the information in the network
    // configuration
    const caClient = buildCAClient(
      FabricCAServices, ccp, 'ca.org1.example.com');

    // setup the wallet to hold the credentials
    // of the application user

```

```

const wallet = await buildWallet(Wallets,
    walletPath);

// in a real application this would be done
// on an administrative flow, and only once
await enrollAdmin(caClient, wallet, mspOrg1)
    ;

// in a real application this would be done
// only when a new user was required to be
// added
// and would be part of an administrative
// flow
await registerAndEnrollUser(caClient, wallet
    , mspOrg1, org1UserId, 'org1.department1
    ');

// Create a new gateway instance for
// interacting with the fabric network.
// In a real application this would be done
// as the backend server session is setup
// for
// a user that has been verified.
const gateway = new Gateway();

try {
    // setup the gateway instance
    // The user will now be able to create
    // connections to the fabric network and
    // be able to
    // submit transactions and query. All
    // transactions submitted by this
    // gateway will be
    // signed by this user using the
    // credentials stored in the wallet.
    await gateway.connect(ccp, {
        wallet,
        identity: org1UserId,
        discovery: { enabled: true,
            asLocalhost: true } // using
            asLocalhost as this gateway is
            using a fabric network deployed
            locally
    });
}

```

```

// Build a network instance based on the
// channel where the smart contract is
// deployed
const network = await gateway.getNetwork
(channelName);

// Get the contract from the network.
const contract = network.getContract(
chaincodeName);

// Initialize a set of asset data on the
// channel using the chaincode '
// InitLedger' function.
// This type of transaction would only
// be run once by an application the
// first time it was started after it
// deployed the first time. Any updates
// to the chaincode deployed later would
// likely not need to run
// an "init" type function.
console.log('\n--> Submit Transaction:
InitLedger, function creates the
initial set of assets on the ledger')
;
await contract.submitTransaction('
InitLedger');
console.log('*** Result: committed');

// Let's try a query type operation (
// function).
// This will be sent to just one peer
// and the results will be shown.
console.log('\n--> Evaluate Transaction:
GetAllAssets, function returns all
the current assets on the ledger');
let result = await contract.
evaluateTransaction('GetAllAssets');
console.log('*** Result: ${
prettyJSONString(result.toString())
}');

var stamp = Date.parse(record.timestamp)
var exp = Date.parse(record.exp)

// Now let's try to submit a transaction
.

```

```

// This will be sent to both peers and
// if both peers endorse the transaction
// , the endorsed proposal will be sent
// to the orderer to be committed by
// each of the peer's to the channel
// ledger.
console.log('\n--> Submit Transaction:
CreateAsset, creates new asset with
UUID, timestamp, owner and expiration
arguments');
result = await contract.
submitTransaction('CreateAsset',
record.uuid, stamp, record.owner, exp
);
console.log('*** Result: committed');
if ('${result}' !== '') {
    console.log('*** Result: ${
        prettyJSONString(result.toString
        ())}');
}

console.log('\n--> Evaluate Transaction:
ReadAsset, function returns an asset
with a given assetID');
result = await contract.
evaluateTransaction('ReadAsset',
record.uuid);
console.log('*** Result: ${
    prettyJSONString(result.toString())
}');

console.log('\n--> Evaluate Transaction:
AssetExists, function returns "true"
if an asset with given assetID exist
');
result = await contract.
evaluateTransaction('AssetExists',
record.uuid);
console.log('*** Result: ${
    prettyJSONString(result.toString())
}');

console.log('\n--> Evaluate Transaction:
AssetValid, function returns "true"
if an asset with given assetID exist
');

```

```

        result = await contract.
            evaluateTransaction('AssetValid',
                record.uuid);
        console.log('*** Result: ${
            prettyJSONString(result.toString())
        }');

    } finally {
        // Disconnect from the gateway when the
        // application is closing
        // This will close all connections to
        // the network
        gateway.disconnect();
    }
} catch (error) {
    console.error('***** FAILED to run the
        application: ${error}');
}
}
exports.submit = submit;

/**
 * @param record A record produced using the body
 * of the request in the check function
 */
async function checkRecord(record) {
    try {
        // build an in memory object with the
        // network configuration (also known as a
        // connection profile)
        const ccp = buildCCPOrg2();

        // build an instance of the fabric ca
        // services client based on
        // the information in the network
        // configuration
        const caClient = buildCAClient(
            FabricCAServices, ccp, 'ca.org2.example.
            com');

        // setup the wallet to hold the credentials
        // of the application user
        const wallet = await buildWallet(Wallets,
            walletPath);
    }
}

```

```

// in a real application this would be done
// on an administrative flow, and only once
await enrollAdmin(caClient, wallet, mspOrg2)
;

// in a real application this would be done
// only when a new user was required to be
// added
// and would be part of an administrative
// flow
await registerAndEnrollUser(caClient, wallet
, mspOrg2, org2UserId, 'org2.department1
');

// Create a new gateway instance for
// interacting with the fabric network.
// In a real application this would be done
// as the backend server session is setup
// for
// a user that has been verified.
const gateway = new Gateway();

try {
  // setup the gateway instance
  // The user will now be able to create
  // connections to the fabric network and
  // be able to
  // submit transactions and query. All
  // transactions submitted by this
  // gateway will be
  // signed by this user using the
  // credentials stored in the wallet.
  await gateway.connect(ccp, {
    wallet,
    identity: org2UserId,
    discovery: { enabled: true,
      asLocalhost: true } // using
      asLocalhost as this gateway is
      using a fabric network deployed
      locally
  });

  // Build a network instance based on the
  // channel where the smart contract is
  // deployed

```



```

const network = await gateway.getNetwork
(channelName);

// Get the contract from the network.
const contract = network.getContract(
chaincodeName);

// Initialize a set of asset data on the
channel using the chaincode '
InitLedger' function.
// This type of transaction would only
be run once by an application the
first time it was started after it
// deployed the first time. Any updates
to the chaincode deployed later would
likely not need to run
// an "init" type function.
console.log('\n--> Submit Transaction:
InitLedger, function creates the
initial set of assets on the ledger')
;
await contract.submitTransaction('
InitLedger');
console.log('*** Result: committed');

// Let's try a query type operation (
function).
// This will be sent to just one peer
and the results will be shown.
console.log('\n--> Evaluate Transaction:
GetAllAssets, function returns all
the current assets on the ledger');
let result = await contract.
evaluateTransaction('GetAllAssets');
console.log('*** Result: ${
prettyJSONString(result.toString())
}');

console.log('\n--> Evaluate Transaction:
ReadAsset, function returns an asset
with a given assetID');
result = await contract.
evaluateTransaction('ReadAsset',
record.uuid);
console.log('*** Result: ${
prettyJSONString(result.toString())
}');

```

```

    }');

    console.log('\n--> Evaluate Transaction:
        AssetExists, function returns "true"
        if an asset with given assetID exist
    ');
    result = await contract.
        evaluateTransaction('AssetExists',
            record.uuid);
    console.log('*** Result: ${
        prettyJSONString(result.toString())
    }');

    console.log('\n--> Evaluate Transaction:
        AssetValid, function returns "true"
        if an asset with given assetID exist
    ');
    result = await contract.
        evaluateTransaction('AssetValid',
            record.uuid);
    console.log('*** Result: ${
        prettyJSONString(result.toString())
    }');

    } finally {
        // Disconnect from the gateway when the
        // application is closing
        // This will close all connections to
        // the network
        gateway.disconnect();
    }
    } catch (error) {
        console.error('***** FAILED to run the
            application: ${error}');
    }
    }
    exports.checkRecord = checkRecord;

    exports.index = function (req, res) {
        res.sendFile(path.resolve('views/records.html'))
        ;
    };

    exports.check = function (req, res) {
        hash.update(req.body.uuid);

```

```

    var hashedRec = new Record({uuid : hash.digest('
      hex')}});
    var newRecord = new Record(req.body);
    Record.find(hashedRec).exec(function (err) {
      if (err) {
        res.status(400).send('Unable to find
          record in database');
      } else {
        checkRecord(newRecord);
      }
    });
  });
};

exports.create = function (req, res) {
  hash.update(req.body.uuid);
  var hashedRec = new Record({uuid : hash.digest('
    hex')}});
  var newRecord = new Record(req.body);
  hashedRec.save(function (err) {
    if (err) {
      res.status(400).send('Unable to save
        record to database');
    } else {
      submit(newRecord);
    }
  });
};

exports.createuuid = function (req, res) {
  var uuid = uuid4();
  res.send(uuid);
};

```

A.1.6 Model

Code for the model that handles data formatting, for this Mongoose is used as the ORM for MongoDB.

```

const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const Record = new Schema ({
  uuid: { type: String, required: true },
  timestamp: { type: Date, required: false },
  owner: { type: String, required: false },
  exp: { type: Date, required: false },

```

```
});

module.exports = mongoose.model('Record', Record)
```

A.2 Chaincode

The chaincode developed for the DIIS, representing immunisation records.²

Listing A.2: Chaincode representing immunisation records.

```
package main

import (
    "encoding/json"
    "fmt"
    "log"
    "strconv"

    //"time"

    "github.com/hyperledger/fabric-contract-api-go/
        contractapi"
)

// Record provides functions for managing an Asset
type Record struct {
    contractapi.Contract

    // medical records should contain metadata of a
    // patient-provider encounter (visit date/time,
    // location, etc.),

    // Asset describes basic details of what makes up a
    // simple asset
    type Asset struct {
        UUID          string `json:"uuid"`
        Timestamp     string `json:"dateTime"`
        Owner         string `json:"owner"`
        Expiration    string `json:"expiration"`
    }

    // InitLedger adds a base set of assets to the
    // ledger
```

²Adapted from tutorial provided in official documentation for Hyperledger Fabric
<https://hyperledger-fabric.readthedocs.io/en/release-2.2/chaincode4ade.html>

```

func (s *Record) InitLedger(ctx contractapi.
    TransactionContextInterface) error {
    assets := []Asset{
        {UUID: "0a970bbe-b436-4601-87d2-becd3bf84054",
            Timestamp: "1511382400000", Owner: "Jane Doe"
            , Expiration: "1655510400000"}, // add an
            attribute for Immunisation, representing the
            disease + variant?
        {UUID: "f57d594d-3a88-4fcc-80a2-4147d23923e3",
            Timestamp: "1721382400000", Owner: "Ajay
            Singh", Expiration: "1655510400000"},
        {UUID: "6d8670a9-8550-40b9-9c57-30bb33a4d6f4",
            Timestamp: "1621382402100", Owner: "Zhang San
            ", Expiration: "1655510400000"},
        {UUID: "3519bdc7-eb13-4ac0-bd63-137241af313b",
            Timestamp: "1621382448000", Owner: "Max
            Mustermann", Expiration: "1655510400000"},
        {UUID: "183ce652-0788-4ea7-896b-93d6036db83e",
            Timestamp: "1611382409500", Owner: "Pierre
            Paul", Expiration: "1655510400000"},
        {UUID: "8e0c5ba3-4dfe-41fc-b454-26a3a276ac92",
            Timestamp: "1511382400120", Owner: "Wang Wu",
            Expiration: "1655510400000"},
    }

    for _, asset := range assets {
        assetJSON, err := json.Marshal(asset)
        if err != nil {
            return err
        }

        err = ctx.GetStub().PutState(asset.UUID,
            assetJSON)
        if err != nil {
            return fmt.Errorf("failed to put to world
                state. %v", err)
        }
    }

    return nil
}

// CreateAsset issues a new asset to the world state
    with given details.
func (s *Record) CreateAsset(ctx contractapi.
    TransactionContextInterface, uuid string,

```

```

        timestamp string, owner string, expiration string
    ) error {
exists, err := s.AssetExists(ctx, uuid)
if err != nil {
    return err
}
if exists {
    return fmt.Errorf("the asset %s already exists",
        uuid)
}

asset := Asset{
    UUID:      uuid,
    Timestamp: timestamp,
    Owner:     owner,
    Expiration: expiration,
}
assetJSON, err := json.Marshal(asset)
if err != nil {
    return err
}

return ctx.GetStub().PutState(uuid, assetJSON)
}

// ReadAsset returns the asset stored in the world
// state with given uuid.
func (s *Record) ReadAsset(ctx contractapi.
    TransactionContextInterface, uuid string) (*Asset
    , error) {
assetJSON, err := ctx.GetStub().GetState(uuid)
if err != nil {
    return nil, fmt.Errorf("failed to read from
        world state: %v", err)
}
if assetJSON == nil {
    return nil, fmt.Errorf("the asset %s does not
        exist", uuid)
}

var asset Asset
err = json.Unmarshal(assetJSON, &asset)
if err != nil {
    return nil, err
}

```

```

    return &asset, nil
}

// DeleteAsset deletes an given asset from the world
// state.
func (s *Record) DeleteAsset(ctx contractapi.
    TransactionContextInterface, uuid string) error {
    exists, err := s.AssetExists(ctx, uuid)
    if err != nil {
        return err
    }
    if !exists {
        return fmt.Errorf("the asset %s does not exist",
            uuid)
    }

    return ctx.GetStub().DelState(uuid)
}

// AssetExists returns true when asset with given
// UUID exists in world state
func (s *Record) AssetExists(ctx contractapi.
    TransactionContextInterface, uuid string) (bool,
    error) {
    assetJSON, err := ctx.GetStub().GetState(uuid)
    if err != nil {
        return false, fmt.Errorf("failed to read from
            world state: %v", err)
    }

    return assetJSON != nil, nil
}

// AssetValid returns true when an asset's timestamp
// is less than the expiration
func (s *Record) AssetValid(ctx contractapi.
    TransactionContextInterface, uuid string) (bool,
    error) {
    assetJSON, err := ctx.GetStub().GetState(uuid)
    if err != nil {
        return false, fmt.Errorf("failed to read from
            world state: %v", err)
    }

    var asset Asset
    json.Unmarshal(assetJSON, &asset)

```

```

var stamp, error = strconv.ParseUint(asset.
    Timestamp, 10, 64)
if error != nil {
    return false, fmt.Errorf("failed to Validate
        asset %v", error)
}

var exp, anotherErr = strconv.ParseUint(asset.
    Expiration, 10, 64)
if anotherErr != nil {
    return false, fmt.Errorf("failed to Validate
        asset %v", anotherErr)
}

return stamp < exp, nil
}

// TransferAsset updates the owner field of asset
// with given uuid in world state.
func (s *Record) TransferAsset(ctx contractapi.
    TransactionContextInterface, uuid string,
    newOwner string) error {
    asset, err := s.ReadAsset(ctx, uuid)
    if err != nil {
        return err
    }

    asset.Owner = newOwner
    assetJSON, err := json.Marshal(asset)
    if err != nil {
        return err
    }

    return ctx.GetStub().PutState(uuid, assetJSON)
}

// GetAllAssets returns all assets found in world
// state
func (s *Record) GetAllAssets(ctx contractapi.
    TransactionContextInterface) ([]*Asset, error) {
    // range query with empty string for startKey and
    // endKey does an
    // open-ended query of all assets in the chaincode
    // namespace.

```



```

resultsIterator, err := ctx.GetStub().
    GetStateByRange("", "")
if err != nil {
    return nil, err
}
defer resultsIterator.Close()

var assets []*Asset
for resultsIterator.HasNext() {
    queryResponse, err := resultsIterator.Next()
    if err != nil {
        return nil, err
    }

    var asset Asset
    err = json.Unmarshal(queryResponse.Value, &asset
    )
    if err != nil {
        return nil, err
    }
    assets = append(assets, &asset)
}

return assets, nil
}

func main() {
    assetChaincode, err := contractapi.NewChaincode(&
        Record{})
    if err != nil {
        log.Panicf("Error creating record chaincode: %v"
            , err)
    }

    if err := assetChaincode.Start(); err != nil {
        log.Panicf("Error starting record chaincode: %v"
            , err)
    }
}

```

A.3 Containerisation

Here are the files used to containerise the web application and database.

A.3.1 Dockerfile

```
FROM node:latest

# LABEL maintainer="emailkaine@gmail.com"

# Create app directory
WORKDIR /usr/src/app

#COPY test-network ./

#COPY chaincode ./

COPY package*.json ./

RUN npm install

# Bundle app source
COPY . .

EXPOSE 3000

ENTRYPOINT [ "node", "app.js" ]
```

A.3.2 docker-compose.yml

```
version: '3'

services:
  nodejs:
    build:
      context: .
      dockerfile: Dockerfile
    image: nodejs
    container_name: nodejs
    env_file: .env
    environment:
      - MONGO_USERNAME=$MONGO_USERNAME
      - MONGO_PASSWORD=$MONGO_PASSWORD
      - MONGO_HOSTNAME=db
      - MONGO_PORT=$MONGO_PORT
      - MONGO_DB=$MONGO_DB
    ports:
      - "3000:3000"
```

```

depends_on:
  db:
    condition: service_healthy
volumes:
  - ./home/node/app
  - node_modules:/home/node/app/node_modules
networks:
  - app-network
command: ENTRYPOINT [ "node", "app.js" ]

db:
  image: healthcheck/mongo
  container_name: db
  restart: unless-stopped
  env_file: .env
  environment:
    - MONGO_INITDB_ROOT_USERNAME=$MONGO_USERNAME
    - MONGO_INITDB_ROOT_PASSWORD=$MONGO_PASSWORD
  volumes:
    - dbdata:/data/db
  networks:
    - app-network

networks:
  app-network:
    driver: bridge

volumes:
  dbdata:
  node_modules:

```

Appendix B

Hyperledger Fabric

B.1 Test network

Used for initial testing of chaincode, available on GitHub ¹

B.2 Fabric version

Version 2.3.2 of Fabric is, at the time of writing, the latest release, bringing many improvements which are detailed in the release notes. ²

B.3 Test Application

Used for initial testing and development of chaincode to application communication and vice versa, available on GitHub. ³

B.3.1 Utils

Provided with the test application, two utility files: `AppUtil.js` and `CAUtil.js` have been used in the application to handle building connection profiles, wallets, CA clients and the functionality that goes with the clients; enrolling admins and registering as well as enrolling users. ⁴

B.4 fabric-network Node.js SDK

A package that encapsulates the APIs required when connecting to a Fabric network, submitting transactions, performing queries against the ledger, and

¹<https://github.com/hyperledger/fabric-samples/tree/main/test-network>

²<https://github.com/hyperledger/fabric/releases/tag/v2.3.2>

³<https://github.com/hyperledger/fabric-samples/tree/main/test-application>

⁴<https://github.com/hyperledger/fabric-samples/tree/main/test-application/javascript>

listening for or replaying events. ⁵

fabric-network.Wallets

A factory for creating wallets backed by default store implementations. ⁶

fabric-network.Gateway

A package that manages network interactions on behalf of an application. A connection profile is provided as a parameter, in this case as a label matching an identity within the wallet supplied as an option B.4. ⁷

fabric-network.Wallet

A package to store information for use when connecting to a Gateway. The wallet utilises a store which handles persistence of identity information. B.4⁸

fabric-network.WalletStore

Interface for store implementations that provide backing storage for identities in a Wallet. ⁹

B.5 Node.js fabric-ca-client

A package that encapsulates the APIs to interact with the Fabric CA. Used to manage the certificates lifecycle of registering, enrolling, renewing and revoking. ¹⁰

B.6 Node.js fabric-common

A package that encapsulates common code used by the 'fabric-ca-client' and 'fabric-network' packages. ¹¹

⁵<https://www.npmjs.com/package/fabric-network>

⁶<https://hyperledger.github.io/fabric-sdk-node/release-2.2/module-fabric-network.Wallets.html>

⁷<https://hyperledger.github.io/fabric-sdk-node/release-2.2/module-fabric-network.Gateway.html>

⁸<https://hyperledger.github.io/fabric-sdk-node/release-2.2/module-fabric-network.Wallet.html>

⁹<https://hyperledger.github.io/fabric-sdk-node/release-2.2/module-fabric-network.WalletStore.html>

¹⁰<https://www.npmjs.com/package/fabric-ca-client>

¹¹<https://www.npmjs.com/package/fabric-common>

B.7 Node.js fabric-protos

A package that encapsulates the Protocol Buffer files and generated JavaScript classes for Fabric. ¹²

B.8 Fabric's Performance Traffic Engine - PTE

A tool that uses SDKs to interact with Fabric networks by sending requests and receiving responses to see how a network performs. ¹³

B.9 Configtx.yaml

Configuration for the network, adapted from the sample provided in the documentation for Hyperledger Fabric. ¹⁴

¹²<https://www.npmjs.com/package/fabric-protos>

¹³<https://github.com/hyperledger/fabric-test/tree/main/tools/PTE>

¹⁴<https://github.com/hyperledger/fabric-samples/blob/main/test-network/configtx/configtx.yaml>

Appendix C

Development Tools

C.1 Docker

Used to containerise the web application and database for the system, as well as containers from Hyperledger Fabric images. Containers package up all code and necessary dependencies so applications can run reliably on differing systems. ¹

C.2 Docker Compose

A tool used to define and run multi-container Docker applications. YAML files are used to configure application's services. Used in this project to build the web application and database together. ²

C.3 PlantUML

An open-source tool that allows users to create diagrams from text. Used to generate diagrams throughout this report, using both PNGs generated on the PlantUML web server ³ and the tex package ⁴ used with Latex.

C.4 Diagram Sprites

Repository of sprites, used with PlantUML to easily identify technologies in diagrams. ⁵

¹<https://docs.docker.com/>

²<https://docs.docker.com/compose/>

³<http://www.plantuml.com/plantuml>

⁴<https://ctan.org/pkg/plantuml?lang=en>

⁵<https://github.com/tupadr3/plantuml-icon-font-sprites>

C.5 C4-PlantUML

A tool to facilitate building C4 model diagrams in PlantUML. ⁶

C.6 ExpressJS

A web framework for Nodejs, provides HTTP utility methods and middleware to aid backend development. ⁷

C.7 EJS

A simple templating language to generate HTML markup using JavaScript. ⁸

C.8 MongoDB

A general purpose, document-based, database for web applications. ⁹

C.9 MongooseJS

An Object Data Modelling (ODM) library for MongoDB. It provides a schema-based method of modelling application data. ¹⁰

C.10 UUID Node.js

A package for creating RFC4122 UUIDs. ¹¹

C.11 Crypto

Crypto module, provides cryptographic functionality, used for hashing data before entering in database. ¹²

⁶<https://github.com/plantuml-stdlib/C4-PlantUML>

⁷<https://expressjs.com/>

⁸<https://ejs.co/>

⁹<https://www.mongodb.com/>

¹⁰<https://mongoosejs.com/>

¹¹<https://www.npmjs.com/package/uuid>

¹²<https://nodejs.org/api/crypto.html>

Appendix D

Logs

D.1 Test Network Logs

Output produced when test network startup scripts are run.

D.1.1 `”./network.sh up createChannel -c mychannel -ca”` Output

```
Creating channel 'mychannel'.
If network is not up, starting nodes with CLI
  timeout of '5' tries and CLI delay of '3'
  seconds and using database 'leveldb with crypto
    from 'Certificate Authorities'
Bringing up network
LOCAL_VERSION=2.3.2
DOCKER_IMAGE_VERSION=2.3.2
CA_LOCAL_VERSION=1.5.0
CA_DOCKER_IMAGE_VERSION=1.5.0
Generating certificates using Fabric CA
Creating network "fabric_test" with the default
  driver
Creating ca_org1      ... done
Creating ca_org2      ... done
Creating ca_orderer  ... done
Creating Org1 Identities
Enrolling the CA admin
+ fabric-ca-client enroll -u https://admin:
  adminpw@localhost:7054 --caname ca-org1 --tls.
  certfiles /home/email/go/src/github.com/ka1ne/
  fabric-samples/test-network/organizations/
  fabric-ca/org1/tls-cert.pem
```

```

2021/05/17 17:19:18 [INFO] Created a default
configuration file at /home/email/go/src/github
.com/kalne/fabric-samples/test-network/
organizations/peerOrganizations/org1.example.
com/fabric-ca-client-config.yaml
2021/05/17 17:19:18 [INFO] TLS Enabled
2021/05/17 17:19:18 [INFO] generating key: &{A:
ecdsa S:256}
2021/05/17 17:19:18 [INFO] encoded CSR
2021/05/17 17:19:18 [INFO] Stored client
certificate at /home/email/go/src/github.com/
kalne/fabric-samples/test-network/organizations
/peerOrganizations/org1.example.com/msp/
signcerts/cert.pem
2021/05/17 17:19:18 [INFO] Stored root CA
certificate at /home/email/go/src/github.com/
kalne/fabric-samples/test-network/organizations
/peerOrganizations/org1.example.com/msp/cacerts
/localhost-7054-ca-org1.pem
2021/05/17 17:19:18 [INFO] Stored Issuer public
key at /home/email/go/src/github.com/kalne/
fabric-samples/test-network/organizations/
peerOrganizations/org1.example.com/msp/
IssuerPublicKey
2021/05/17 17:19:18 [INFO] Stored Issuer
revocation public key at /home/email/go/src/
github.com/kalne/fabric-samples/test-network/
organizations/peerOrganizations/org1.example.
com/msp/IssuerRevocationPublicKey
Registering peer0
+ fabric-ca-client register --caname ca-org1 --id.
name peer0 --id.secret peer0pw --id.type peer
--tls.certfiles /home/email/go/src/github.com/
kalne/fabric-samples/test-network/organizations
/fabric-ca/org1/tls-cert.pem
2021/05/17 17:19:18 [INFO] Configuration file
location: /home/email/go/src/github.com/kalne/
fabric-samples/test-network/organizations/
peerOrganizations/org1.example.com/fabric-ca-
client-config.yaml
2021/05/17 17:19:18 [INFO] TLS Enabled
2021/05/17 17:19:18 [INFO] TLS Enabled
Password: peer0pw
Registering user
+ fabric-ca-client register --caname ca-org1 --id.
name user1 --id.secret user1pw --id.type client

```

```

--tls.certfiles /home/email/go/src/github.com/
kaïne/fabric-samples/test-network/organizations
/fabric-ca/org1/tls-cert.pem
2021/05/17 17:19:18 [INFO] Configuration file
location: /home/email/go/src/github.com/kaïne/
fabric-samples/test-network/organizations/
peerOrganizations/org1.example.com/fabric-ca-
client-config.yaml
2021/05/17 17:19:18 [INFO] TLS Enabled
2021/05/17 17:19:18 [INFO] TLS Enabled
Password: user1pw
Registering the org admin
+ fabric-ca-client register --caname ca-org1 --id.
name org1admin --id.secret org1adminpw --id.
type admin --tls.certfiles /home/email/go/src/
github.com/kaïne/fabric-samples/test-network/
organizations/fabric-ca/org1/tls-cert.pem
2021/05/17 17:19:19 [INFO] Configuration file
location: /home/email/go/src/github.com/kaïne/
fabric-samples/test-network/organizations/
peerOrganizations/org1.example.com/fabric-ca-
client-config.yaml
2021/05/17 17:19:19 [INFO] TLS Enabled
2021/05/17 17:19:19 [INFO] TLS Enabled
Password: org1adminpw
Generating the peer0 msp
+ fabric-ca-client enroll -u https://peer0:
peer0pw@localhost:7054 --caname ca-org1 -M /
home/email/go/src/github.com/kaïne/fabric-
samples/test-network/organizations/
peerOrganizations/org1.example.com/peers/peer0.
org1.example.com/msp --csr.hosts peer0.org1.
example.com --tls.certfiles /home/email/go/src/
github.com/kaïne/fabric-samples/test-network/
organizations/fabric-ca/org1/tls-cert.pem
2021/05/17 17:19:19 [INFO] TLS Enabled
2021/05/17 17:19:19 [INFO] generating key: &{A:
ecdsa S:256}
2021/05/17 17:19:19 [INFO] encoded CSR
2021/05/17 17:19:19 [INFO] Stored client
certificate at /home/email/go/src/github.com/
kaïne/fabric-samples/test-network/organizations
/peerOrganizations/org1.example.com/peers/peer0
.org1.example.com/msp/signcerts/cert.pem
2021/05/17 17:19:19 [INFO] Stored root CA
certificate at /home/email/go/src/github.com/

```

```

    ka/ine/fabric-samples/test-network/organizations
    /peerOrganizations/org1.example.com/peers/peer0
    .org1.example.com/msp/cacerts/localhost-7054-ca
    -org1.pem
2021/05/17 17:19:19 [INFO] Stored Issuer public
    key at /home/email/go/src/github.com/ka/ine/
    fabric-samples/test-network/organizations/
    peerOrganizations/org1.example.com/peers/peer0.
    org1.example.com/msp/IssuerPublicKey
2021/05/17 17:19:19 [INFO] Stored Issuer
    revocation public key at /home/email/go/src/
    github.com/ka/ine/fabric-samples/test-network/
    organizations/peerOrganizations/org1.example.
    com/peers/peer0.org1.example.com/msp/
    IssuerRevocationPublicKey
Generating the peer0-tls certificates
+ fabric-ca-client enroll -u https://peer0:
    peer0pw@localhost:7054 --caname ca-org1 -M /
    home/email/go/src/github.com/ka/ine/fabric-
    samples/test-network/organizations/
    peerOrganizations/org1.example.com/peers/peer0.
    org1.example.com/tls --enrollment.profile tls
    --csr.hosts peer0.org1.example.com --csr.hosts
    localhost --tls.certfiles /home/email/go/src/
    github.com/ka/ine/fabric-samples/test-network/
    organizations/fabric-ca/org1/tls-cert.pem
2021/05/17 17:19:19 [INFO] TLS Enabled
2021/05/17 17:19:19 [INFO] generating key: &{A:
    ecdsa S:256}
2021/05/17 17:19:19 [INFO] encoded CSR
2021/05/17 17:19:19 [INFO] Stored client
    certificate at /home/email/go/src/github.com/
    ka/ine/fabric-samples/test-network/organizations
    /peerOrganizations/org1.example.com/peers/peer0
    .org1.example.com/tls/signcerts/cert.pem
2021/05/17 17:19:19 [INFO] Stored TLS root CA
    certificate at /home/email/go/src/github.com/
    ka/ine/fabric-samples/test-network/organizations
    /peerOrganizations/org1.example.com/peers/peer0
    .org1.example.com/tls/tlscacerts/tls-localhost
    -7054-ca-org1.pem
2021/05/17 17:19:19 [INFO] Stored Issuer public
    key at /home/email/go/src/github.com/ka/ine/
    fabric-samples/test-network/organizations/
    peerOrganizations/org1.example.com/peers/peer0.
    org1.example.com/tls/IssuerPublicKey

```

```

2021/05/17 17:19:19 [INFO] Stored Issuer
    revocation public key at /home/email/go/src/
    github.com/kalne/fabric-samples/test-network/
    organizations/peerOrganizations/org1.example.
    com/peers/peer0.org1.example.com/tls/
    IssuerRevocationPublicKey
Generating the user msp
+ fabric-ca-client enroll -u https://user1:
    user1pw@localhost:7054 --caname ca-org1 -M /
    home/email/go/src/github.com/kalne/fabric-
    samples/test-network/organizations/
    peerOrganizations/org1.example.com/users/
    User1@org1.example.com/msp --tls.certfiles /
    home/email/go/src/github.com/kalne/fabric-
    samples/test-network/organizations/fabric-ca/
    org1/tls-cert.pem
2021/05/17 17:19:19 [INFO] TLS Enabled
2021/05/17 17:19:19 [INFO] generating key: &{A:
    ecdsa S:256}
2021/05/17 17:19:19 [INFO] encoded CSR
2021/05/17 17:19:19 [INFO] Stored client
    certificate at /home/email/go/src/github.com/
    kalne/fabric-samples/test-network/organizations
    /peerOrganizations/org1.example.com/users/
    User1@org1.example.com/msp/signcerts/cert.pem
2021/05/17 17:19:19 [INFO] Stored root CA
    certificate at /home/email/go/src/github.com/
    kalne/fabric-samples/test-network/organizations
    /peerOrganizations/org1.example.com/users/
    User1@org1.example.com/msp/cacerts/localhost
    -7054-ca-org1.pem
2021/05/17 17:19:19 [INFO] Stored Issuer public
    key at /home/email/go/src/github.com/kalne/
    fabric-samples/test-network/organizations/
    peerOrganizations/org1.example.com/users/
    User1@org1.example.com/msp/IssuerPublicKey
2021/05/17 17:19:19 [INFO] Stored Issuer
    revocation public key at /home/email/go/src/
    github.com/kalne/fabric-samples/test-network/
    organizations/peerOrganizations/org1.example.
    com/users/User1@org1.example.com/msp/
    IssuerRevocationPublicKey
Generating the org admin msp
+ fabric-ca-client enroll -u https://org1admin:
    org1adminpw@localhost:7054 --caname ca-org1 -M
    /home/email/go/src/github.com/kalne/fabric-

```

```

samples/test-network/organizations/
peerOrganizations/org1.example.com/users/
Admin@org1.example.com/msp --tls.certfiles /
home/email/go/src/github.com/kalne/fabric-
samples/test-network/organizations/fabric-ca/
org1/tls-cert.pem
2021/05/17 17:19:19 [INFO] TLS Enabled
2021/05/17 17:19:19 [INFO] generating key: &{A:
ecdsa S:256}
2021/05/17 17:19:19 [INFO] encoded CSR
2021/05/17 17:19:19 [INFO] Stored client
certificate at /home/email/go/src/github.com/
kalne/fabric-samples/test-network/organizations
/peerOrganizations/org1.example.com/users/
Admin@org1.example.com/msp/signcerts/cert.pem
2021/05/17 17:19:19 [INFO] Stored root CA
certificate at /home/email/go/src/github.com/
kalne/fabric-samples/test-network/organizations
/peerOrganizations/org1.example.com/users/
Admin@org1.example.com/msp/cacerts/localhost
-7054-ca-org1.pem
2021/05/17 17:19:19 [INFO] Stored Issuer public
key at /home/email/go/src/github.com/kalne/
fabric-samples/test-network/organizations/
peerOrganizations/org1.example.com/users/
Admin@org1.example.com/msp/IssuerPublicKey
2021/05/17 17:19:19 [INFO] Stored Issuer
revocation public key at /home/email/go/src/
github.com/kalne/fabric-samples/test-network/
organizations/peerOrganizations/org1.example.
com/users/Admin@org1.example.com/msp/
IssuerRevocationPublicKey
Creating Org2 Identities
Enrolling the CA admin
+ fabric-ca-client enroll -u https://admin:
adminpw@localhost:8054 --caname ca-org2 --tls.
certfiles /home/email/go/src/github.com/kalne/
fabric-samples/test-network/organizations/
fabric-ca/org2/tls-cert.pem
2021/05/17 17:19:19 [INFO] Created a default
configuration file at /home/email/go/src/github
.com/kalne/fabric-samples/test-network/
organizations/peerOrganizations/org2.example.
com/fabric-ca-client-config.yaml
2021/05/17 17:19:19 [INFO] TLS Enabled

```

```

2021/05/17 17:19:19 [INFO] generating key: &{A:
    ecdsa S:256}
2021/05/17 17:19:19 [INFO] encoded CSR
2021/05/17 17:19:19 [INFO] Stored client
    certificate at /home/email/go/src/github.com/
    kaune/fabric-samples/test-network/organizations
    /peerOrganizations/org2.example.com/msp/
    signcerts/cert.pem
2021/05/17 17:19:19 [INFO] Stored root CA
    certificate at /home/email/go/src/github.com/
    kaune/fabric-samples/test-network/organizations
    /peerOrganizations/org2.example.com/msp/cacerts
    /localhost-8054-ca-org2.pem
2021/05/17 17:19:19 [INFO] Stored Issuer public
    key at /home/email/go/src/github.com/kaune/
    fabric-samples/test-network/organizations/
    peerOrganizations/org2.example.com/msp/
    IssuerPublicKey
2021/05/17 17:19:19 [INFO] Stored Issuer
    revocation public key at /home/email/go/src/
    github.com/kaune/fabric-samples/test-network/
    organizations/peerOrganizations/org2.example.
    com/msp/IssuerRevocationPublicKey
Registering peer0
+ fabric-ca-client register --caname ca-org2 --id.
    name peer0 --id.secret peer0pw --id.type peer
    --tls.certfiles /home/email/go/src/github.com/
    kaune/fabric-samples/test-network/organizations
    /fabric-ca/org2/tls-cert.pem
2021/05/17 17:19:20 [INFO] Configuration file
    location: /home/email/go/src/github.com/kaune/
    fabric-samples/test-network/organizations/
    peerOrganizations/org2.example.com/fabric-ca-
    client-config.yaml
2021/05/17 17:19:20 [INFO] TLS Enabled
2021/05/17 17:19:20 [INFO] TLS Enabled
Password: peer0pw
Registering user
+ fabric-ca-client register --caname ca-org2 --id.
    name user1 --id.secret user1pw --id.type client
    --tls.certfiles /home/email/go/src/github.com/
    kaune/fabric-samples/test-network/organizations
    /fabric-ca/org2/tls-cert.pem
2021/05/17 17:19:20 [INFO] Configuration file
    location: /home/email/go/src/github.com/kaune/
    fabric-samples/test-network/organizations/

```

```

    peerOrganizations/org2.example.com/fabric-ca-
    client-config.yaml
2021/05/17 17:19:20 [INFO] TLS Enabled
2021/05/17 17:19:20 [INFO] TLS Enabled
Password: user1pw
Registering the org admin
+ fabric-ca-client register --caname ca-org2 --id.
  name org2admin --id.secret org2adminpw --id.
  type admin --tls.certfiles /home/email/go/src/
  github.com/kalne/fabric-samples/test-network/
  organizations/fabric-ca/org2/tls-cert.pem
2021/05/17 17:19:20 [INFO] Configuration file
  location: /home/email/go/src/github.com/kalne/
  fabric-samples/test-network/organizations/
  peerOrganizations/org2.example.com/fabric-ca-
  client-config.yaml
2021/05/17 17:19:20 [INFO] TLS Enabled
2021/05/17 17:19:20 [INFO] TLS Enabled
Password: org2adminpw
Generating the peer0 msp
+ fabric-ca-client enroll -u https://peer0:
  peer0pw@localhost:8054 --caname ca-org2 -M /
  home/email/go/src/github.com/kalne/fabric-
  samples/test-network/organizations/
  peerOrganizations/org2.example.com/peers/peer0.
  org2.example.com/msp --csr.hosts peer0.org2.
  example.com --tls.certfiles /home/email/go/src/
  github.com/kalne/fabric-samples/test-network/
  organizations/fabric-ca/org2/tls-cert.pem
2021/05/17 17:19:20 [INFO] TLS Enabled
2021/05/17 17:19:20 [INFO] generating key: &{A:
  ecdsa S:256}
2021/05/17 17:19:20 [INFO] encoded CSR
2021/05/17 17:19:20 [INFO] Stored client
  certificate at /home/email/go/src/github.com/
  kalne/fabric-samples/test-network/organizations
  /peerOrganizations/org2.example.com/peers/peer0
  .org2.example.com/msp/signcerts/cert.pem
2021/05/17 17:19:20 [INFO] Stored root CA
  certificate at /home/email/go/src/github.com/
  kalne/fabric-samples/test-network/organizations
  /peerOrganizations/org2.example.com/peers/peer0
  .org2.example.com/msp/cacerts/localhost-8054-ca-
  -org2.pem
2021/05/17 17:19:20 [INFO] Stored Issuer public
  key at /home/email/go/src/github.com/kalne/

```



```

fabric-samples/test-network/organizations/
peerOrganizations/org2.example.com/peers/peer0.
org2.example.com/msp/IssuerPublicKey
2021/05/17 17:19:20 [INFO] Stored Issuer
revocation public key at /home/email/go/src/
github.com/kalne/fabric-samples/test-network/
organizations/peerOrganizations/org2.example.
com/peers/peer0.org2.example.com/msp/
IssuerRevocationPublicKey
Generating the peer0-tls certificates
+ fabric-ca-client enroll -u https://peer0:
peer0pw@localhost:8054 --caname ca-org2 -M /
home/email/go/src/github.com/kalne/fabric-
samples/test-network/organizations/
peerOrganizations/org2.example.com/peers/peer0.
org2.example.com/tls --enrollment.profile tls
--csr.hosts peer0.org2.example.com --csr.hosts
localhost --tls.certfiles /home/email/go/src/
github.com/kalne/fabric-samples/test-network/
organizations/fabric-ca/org2/tls-cert.pem
2021/05/17 17:19:20 [INFO] TLS Enabled
2021/05/17 17:19:20 [INFO] generating key: &{A:
ecdsa S:256}
2021/05/17 17:19:20 [INFO] encoded CSR
2021/05/17 17:19:20 [INFO] Stored client
certificate at /home/email/go/src/github.com/
kalne/fabric-samples/test-network/organizations
/peerOrganizations/org2.example.com/peers/peer0
.org2.example.com/tls/signcerts/cert.pem
2021/05/17 17:19:20 [INFO] Stored TLS root CA
certificate at /home/email/go/src/github.com/
kalne/fabric-samples/test-network/organizations
/peerOrganizations/org2.example.com/peers/peer0
.org2.example.com/tls/tlscacerts/tls-localhost
-8054-ca-org2.pem
2021/05/17 17:19:20 [INFO] Stored Issuer public
key at /home/email/go/src/github.com/kalne/
fabric-samples/test-network/organizations/
peerOrganizations/org2.example.com/peers/peer0.
org2.example.com/tls/IssuerPublicKey
2021/05/17 17:19:20 [INFO] Stored Issuer
revocation public key at /home/email/go/src/
github.com/kalne/fabric-samples/test-network/
organizations/peerOrganizations/org2.example.
com/peers/peer0.org2.example.com/tls/
IssuerRevocationPublicKey

```

```

Generating the user msp
+ fabric-ca-client enroll -u https://user1:
  user1pw@localhost:8054 --caname ca-org2 -M /
  home/email/go/src/github.com/kalne/fabric-
  samples/test-network/organizations/
  peerOrganizations/org2.example.com/users/
  User1@org2.example.com/msp --tls.certfiles /
  home/email/go/src/github.com/kalne/fabric-
  samples/test-network/organizations/fabric-ca/
  org2/tls-cert.pem
2021/05/17 17:19:20 [INFO] TLS Enabled
2021/05/17 17:19:20 [INFO] generating key: &{A:
  ecdsa S:256}
2021/05/17 17:19:20 [INFO] encoded CSR
2021/05/17 17:19:20 [INFO] Stored client
  certificate at /home/email/go/src/github.com/
  kalne/fabric-samples/test-network/organizations
  /peerOrganizations/org2.example.com/users/
  User1@org2.example.com/msp/signcerts/cert.pem
2021/05/17 17:19:20 [INFO] Stored root CA
  certificate at /home/email/go/src/github.com/
  kalne/fabric-samples/test-network/organizations
  /peerOrganizations/org2.example.com/users/
  User1@org2.example.com/msp/cacerts/localhost
  -8054-ca-org2.pem
2021/05/17 17:19:20 [INFO] Stored Issuer public
  key at /home/email/go/src/github.com/kalne/
  fabric-samples/test-network/organizations/
  peerOrganizations/org2.example.com/users/
  User1@org2.example.com/msp/IssuerPublicKey
2021/05/17 17:19:20 [INFO] Stored Issuer
  revocation public key at /home/email/go/src/
  github.com/kalne/fabric-samples/test-network/
  organizations/peerOrganizations/org2.example.
  com/users/User1@org2.example.com/msp/
  IssuerRevocationPublicKey
Generating the org admin msp
+ fabric-ca-client enroll -u https://org2admin:
  org2adminpw@localhost:8054 --caname ca-org2 -M
  /home/email/go/src/github.com/kalne/fabric-
  samples/test-network/organizations/
  peerOrganizations/org2.example.com/users/
  Admin@org2.example.com/msp --tls.certfiles /
  home/email/go/src/github.com/kalne/fabric-
  samples/test-network/organizations/fabric-ca/
  org2/tls-cert.pem

```

```

2021/05/17 17:19:20 [INFO] TLS Enabled
2021/05/17 17:19:20 [INFO] generating key: &{A:
    ecdsa S:256}
2021/05/17 17:19:20 [INFO] encoded CSR
2021/05/17 17:19:21 [INFO] Stored client
    certificate at /home/email/go/src/github.com/
    kalne/fabric-samples/test-network/organizations
    /peerOrganizations/org2.example.com/users/
    Admin@org2.example.com/msp/signcerts/cert.pem
2021/05/17 17:19:21 [INFO] Stored root CA
    certificate at /home/email/go/src/github.com/
    kalne/fabric-samples/test-network/organizations
    /peerOrganizations/org2.example.com/users/
    Admin@org2.example.com/msp/cacerts/localhost
    -8054-ca-org2.pem
2021/05/17 17:19:21 [INFO] Stored Issuer public
    key at /home/email/go/src/github.com/kalne/
    fabric-samples/test-network/organizations/
    peerOrganizations/org2.example.com/users/
    Admin@org2.example.com/msp/IssuerPublicKey
2021/05/17 17:19:21 [INFO] Stored Issuer
    revocation public key at /home/email/go/src/
    github.com/kalne/fabric-samples/test-network/
    organizations/peerOrganizations/org2.example.
    com/users/Admin@org2.example.com/msp/
    IssuerRevocationPublicKey
Creating Orderer Org Identities
Enrolling the CA admin
+ fabric-ca-client enroll -u https://admin:
    adminpw@localhost:9054 --caname ca-orderer --
    tls.certfiles /home/email/go/src/github.com/
    kalne/fabric-samples/test-network/organizations
    /fabric-ca/ordererOrg/tls-cert.pem
2021/05/17 17:19:21 [INFO] Created a default
    configuration file at /home/email/go/src/github
    .com/kalne/fabric-samples/test-network/
    organizations/ordererOrganizations/example.com/
    fabric-ca-client-config.yaml
2021/05/17 17:19:21 [INFO] TLS Enabled
2021/05/17 17:19:21 [INFO] generating key: &{A:
    ecdsa S:256}
2021/05/17 17:19:21 [INFO] encoded CSR
2021/05/17 17:19:21 [INFO] Stored client
    certificate at /home/email/go/src/github.com/
    kalne/fabric-samples/test-network/organizations
    /ordererOrganizations/example.com/msp/signcerts

```

```

/cert.pem
2021/05/17 17:19:21 [INFO] Stored root CA
certificate at /home/email/go/src/github.com/
kaïne/fabric-samples/test-network/organizations
/ordererOrganizations/example.com/msp/cacerts/
localhost-9054-ca-orderer.pem
2021/05/17 17:19:21 [INFO] Stored Issuer public
key at /home/email/go/src/github.com/kaïne/
fabric-samples/test-network/organizations/
ordererOrganizations/example.com/msp/
IssuerPublicKey
2021/05/17 17:19:21 [INFO] Stored Issuer
revocation public key at /home/email/go/src/
github.com/kaïne/fabric-samples/test-network/
organizations/ordererOrganizations/example.com/
msp/IssuerRevocationPublicKey
Registering orderer
+ fabric-ca-client register --caname ca-orderer --
id.name orderer --id.secret ordererpw --id.type
orderer --tls.certfiles /home/email/go/src/
github.com/kaïne/fabric-samples/test-network/
organizations/fabric-ca/ordererOrg/tls-cert.pem
2021/05/17 17:19:21 [INFO] Configuration file
location: /home/email/go/src/github.com/kaïne/
fabric-samples/test-network/organizations/
ordererOrganizations/example.com/fabric-ca-
client-config.yaml
2021/05/17 17:19:21 [INFO] TLS Enabled
2021/05/17 17:19:21 [INFO] TLS Enabled
Password: ordererpw
Registering the orderer admin
+ fabric-ca-client register --caname ca-orderer --
id.name ordererAdmin --id.secret ordererAdminpw
--id.type admin --tls.certfiles /home/email/go
/src/github.com/kaïne/fabric-samples/test-
network/organizations/fabric-ca/ordererOrg/tls-
cert.pem
2021/05/17 17:19:21 [INFO] Configuration file
location: /home/email/go/src/github.com/kaïne/
fabric-samples/test-network/organizations/
ordererOrganizations/example.com/fabric-ca-
client-config.yaml
2021/05/17 17:19:21 [INFO] TLS Enabled
2021/05/17 17:19:21 [INFO] TLS Enabled
Password: ordererAdminpw
Generating the orderer msp

```

```

+ fabric-ca-client enroll -u https://orderer:
  ordererpw@localhost:9054 --caname ca-orderer -M
  /home/email/go/src/github.com/kalne/fabric-
  samples/test-network/organizations/
  ordererOrganizations/example.com/orderers/
  orderer.example.com/msp --csr.hosts orderer.
  example.com --csr.hosts localhost --tls.
  certfiles /home/email/go/src/github.com/kalne/
  fabric-samples/test-network/organizations/
  fabric-ca/ordererOrg/tls-cert.pem
2021/05/17 17:19:21 [INFO] TLS Enabled
2021/05/17 17:19:21 [INFO] generating key: &{A:
  ecdsa S:256}
2021/05/17 17:19:21 [INFO] encoded CSR
2021/05/17 17:19:21 [INFO] Stored client
  certificate at /home/email/go/src/github.com/
  kalne/fabric-samples/test-network/organizations
  /ordererOrganizations/example.com/orderers/
  orderer.example.com/msp/signcerts/cert.pem
2021/05/17 17:19:21 [INFO] Stored root CA
  certificate at /home/email/go/src/github.com/
  kalne/fabric-samples/test-network/organizations
  /ordererOrganizations/example.com/orderers/
  orderer.example.com/msp/cacerts/localhost-9054-
  ca-orderer.pem
2021/05/17 17:19:21 [INFO] Stored Issuer public
  key at /home/email/go/src/github.com/kalne/
  fabric-samples/test-network/organizations/
  ordererOrganizations/example.com/orderers/
  orderer.example.com/msp/IssuerPublicKey
2021/05/17 17:19:21 [INFO] Stored Issuer
  revocation public key at /home/email/go/src/
  github.com/kalne/fabric-samples/test-network/
  organizations/ordererOrganizations/example.com/
  orderers/orderer.example.com/msp/
  IssuerRevocationPublicKey
Generating the orderer-tls certificates
+ fabric-ca-client enroll -u https://orderer:
  ordererpw@localhost:9054 --caname ca-orderer -M
  /home/email/go/src/github.com/kalne/fabric-
  samples/test-network/organizations/
  ordererOrganizations/example.com/orderers/
  orderer.example.com/tls --enrollment.profile
  tls --csr.hosts orderer.example.com --csr.hosts
  localhost --tls.certfiles /home/email/go/src/
  github.com/kalne/fabric-samples/test-network/

```

```

        organizations/fabric-ca/ordererOrg/tls-cert.pem
2021/05/17 17:19:21 [INFO] TLS Enabled
2021/05/17 17:19:21 [INFO] generating key: &{A:
    ecdsa S:256}
2021/05/17 17:19:21 [INFO] encoded CSR
2021/05/17 17:19:21 [INFO] Stored client
    certificate at /home/email/go/src/github.com/
    kaïne/fabric-samples/test-network/organizations
    /ordererOrganizations/example.com/orderers/
    orderer.example.com/tls/signcerts/cert.pem
2021/05/17 17:19:21 [INFO] Stored TLS root CA
    certificate at /home/email/go/src/github.com/
    kaïne/fabric-samples/test-network/organizations
    /ordererOrganizations/example.com/orderers/
    orderer.example.com/tls/tlscacerts/tls-
    localhost-9054-ca-orderer.pem
2021/05/17 17:19:21 [INFO] Stored Issuer public
    key at /home/email/go/src/github.com/kaïne/
    fabric-samples/test-network/organizations/
    ordererOrganizations/example.com/orderers/
    orderer.example.com/tls/IssuerPublicKey
2021/05/17 17:19:21 [INFO] Stored Issuer
    revocation public key at /home/email/go/src/
    github.com/kaïne/fabric-samples/test-network/
    organizations/ordererOrganizations/example.com/
    orderers/orderer.example.com/tls/
    IssuerRevocationPublicKey
Generating the admin msp
+ fabric-ca-client enroll -u https://ordererAdmin:
    ordererAdminpw@localhost:9054 --caname ca-
    orderer -M /home/email/go/src/github.com/kaïne/
    fabric-samples/test-network/organizations/
    ordererOrganizations/example.com/users/
    Admin@example.com/msp --tls.certfiles /home/
    email/go/src/github.com/kaïne/fabric-samples/
    test-network/organizations/fabric-ca/ordererOrg
    /tls-cert.pem
2021/05/17 17:19:21 [INFO] TLS Enabled
2021/05/17 17:19:21 [INFO] generating key: &{A:
    ecdsa S:256}
2021/05/17 17:19:21 [INFO] encoded CSR
2021/05/17 17:19:22 [INFO] Stored client
    certificate at /home/email/go/src/github.com/
    kaïne/fabric-samples/test-network/organizations
    /ordererOrganizations/example.com/users/
    Admin@example.com/msp/signcerts/cert.pem

```

```

2021/05/17 17:19:22 [INFO] Stored root CA
    certificate at /home/email/go/src/github.com/
    kaïne/fabric-samples/test-network/organizations
    /ordererOrganizations/example.com/users/
    Admin@example.com/msp/cacerts/localhost-9054-ca
    -orderer.pem
2021/05/17 17:19:22 [INFO] Stored Issuer public
    key at /home/email/go/src/github.com/kaïne/
    fabric-samples/test-network/organizations/
    ordererOrganizations/example.com/users/
    Admin@example.com/msp/IssuerPublicKey
2021/05/17 17:19:22 [INFO] Stored Issuer
    revocation public key at /home/email/go/src/
    github.com/kaïne/fabric-samples/test-network/
    organizations/ordererOrganizations/example.com/
    users/Admin@example.com/msp/
    IssuerRevocationPublicKey
Generating CCP files for Org1 and Org2
Creating volume "docker_orderer.example.com" with
    default driver
Creating volume "docker_peer0.org1.example.com"
    with default driver
Creating volume "docker_peer0.org2.example.com"
    with default driver
WARNING: Found orphan containers (ca_org2, ca_org1
    , ca_orderer) for this project. If you removed
    or renamed this service in your compose file,
    you can run this command with the --remove-
    orphans flag to clean it up.
Creating peer0.org1.example.com ... done
Creating peer0.org2.example.com ... done
Creating orderer.example.com ... done
Creating cli ... done
CONTAINER ID    IMAGE
COMMAND
CREATED
STATUS
PORTS
NAMES
a8ce24d4ecd4    hyperledger/fabric-tools:latest
    "/bin/bash"    1 second ago
    Up Less than a second
    cli
1e1bc66d10f6    hyperledger/fabric-peer:latest
    "peer node start"    2 seconds ago
    Up 1 second    7051/tcp,
    0.0.0.0:9051->9051/tcp, :::9051->9051/tcp

```

```

peer0.org2.example.com
50fd73a87292 hyperledger/fabric-orderer:latest
    "orderer" 2 seconds ago Up
    1 second 0.0.0.0:7050->7050/tcp,
    :::7050->7050/tcp, 0.0.0.0:7053->7053/tcp,
    :::7053->7053/tcp orderer.example.com
ad6ad67f0b97 hyperledger/fabric-peer:latest
    "peer node start" 2 seconds ago
    Up 1 second 0.0.0.0:7051->7051/tcp,
    :::7051->7051/tcp
peer0.org1.example.com
be9a25f3b104 hyperledger/fabric-ca:latest
    "sh -c 'fabric-ca-se..." 10 seconds
    ago Up 8 seconds 7054/tcp,
    0.0.0.0:8054->8054/tcp, :::8054->8054/tcp
ca_org2
9670b428fa28 hyperledger/fabric-ca:latest
    "sh -c 'fabric-ca-se..." 10 seconds
    ago Up 8 seconds 7054/tcp,
    0.0.0.0:9054->9054/tcp, :::9054->9054/tcp
ca_orderer
547c49cecac0 hyperledger/fabric-ca:latest
    "sh -c 'fabric-ca-se..." 10 seconds
    ago Up 8 seconds
    0.0.0.0:7054->7054/tcp, :::7054->7054/tcp
ca_org1
Generating channel genesis block 'mychannel.block'
/home/email/go/src/github.com/kayne/fabric-samples
/test-network/./bin/configtxgen
+ configtxgen -profile TwoOrgsApplicationGenesis -
  outputBlock ./channel-artifacts/mychannel.block
  -channelID mychannel
2021-05-17 17:19:25.735 BST [common.tools.
  configtxgen] main -> INFO 001 Loading
  configuration
2021-05-17 17:19:25.761 BST [common.tools.
  configtxgen.localconfig] completeInitialization
  -> INFO 002 orderer type: etcdraft
2021-05-17 17:19:25.764 BST [common.tools.
  configtxgen.localconfig] completeInitialization
  -> INFO 003 Orderer.EtcdRaft.Options unset,
  setting to tick_interval:"500ms" election_tick
  :10 heartbeat_tick:1 max_inflight_blocks:5
  snapshot_interval_size:16777216
2021-05-17 17:19:25.764 BST [common.tools.
  configtxgen.localconfig] Load -> INFO 004

```



```

    Loaded configuration: /home/email/go/src/github
    .com/kalne/fabric-samples/test-network/configtx
    /configtx.yaml
2021-05-17 17:19:25.772 BST [common.tools.
    configtxgen] doOutputBlock -> INFO 005
    Generating genesis block
2021-05-17 17:19:25.773 BST [common.tools.
    configtxgen] doOutputBlock -> INFO 006 Creating
    application channel genesis block
2021-05-17 17:19:25.775 BST [common.tools.
    configtxgen] doOutputBlock -> INFO 007 Writing
    genesis block
+ res=0
Creating channel mychannel
Using organization 1
+ osnadmin channel join --channelID mychannel --
    config-block ./channel-artifacts/mychannel.
    block -o localhost:7053 --ca-file /home/email/
    go/src/github.com/kalne/fabric-samples/test-
    network/organizations/ordererOrganizations/
    example.com/orderers/orderer.example.com/msp/
    tlscacerts/tlsca.example.com-cert.pem --client-
    cert /home/email/go/src/github.com/kalne/fabric
    -samples/test-network/organizations/
    ordererOrganizations/example.com/orderers/
    orderer.example.com/tls/server.crt --client-key
    /home/email/go/src/github.com/kalne/fabric-
    samples/test-network/organizations/
    ordererOrganizations/example.com/orderers/
    orderer.example.com/tls/server.key
+ res=0
Status: 201
{
    "name": "mychannel",
    "url": "/participation/v1/channels/
        mychannel",
    "consensusRelation": "consenter",
    "status": "active",
    "height": 1
}

Channel 'mychannel' created
Joining org1 peer to the channel...
Using organization 1
+ peer channel join -b ./channel-artifacts/
    mychannel.block

```

```

+ res=0
2021-05-17 17:19:32.379 BST [channelCmd]
    InitCmdFactory -> INFO 001 Endorser and orderer
    connections initialized
2021-05-17 17:19:32.537 BST [channelCmd]
    executeJoin -> INFO 002 Successfully submitted
    proposal to join channel
Joining org2 peer to the channel...
Using organization 2
+ peer channel join -b ./channel-artifacts/
    mychannel.block
+ res=0
2021-05-17 17:19:35.622 BST [channelCmd]
    InitCmdFactory -> INFO 001 Endorser and orderer
    connections initialized
2021-05-17 17:19:35.686 BST [channelCmd]
    executeJoin -> INFO 002 Successfully submitted
    proposal to join channel
Setting anchor peer for org1...
Using organization 1
Fetching channel config for channel mychannel
Using organization 1
Fetching the most recent configuration block for
the channel
+ peer channel fetch config config_block.pb -o
orderer.example.com:7050 --
ordererTLSHostnameOverride orderer.example.com
-c mychannel --tls --cafile /opt/gopath/src/
github.com/hyperledger/fabric/peer/
organizations/ordererOrganizations/example.com/
orderers/orderer.example.com/msp/tlscacerts/
tlsca.example.com-cert.pem
2021-05-17 16:19:36.247 UTC [channelCmd]
    InitCmdFactory -> INFO 001 Endorser and orderer
    connections initialized
2021-05-17 16:19:36.266 UTC [cli.common] readBlock
-> INFO 002 Received block: 0
2021-05-17 16:19:36.267 UTC [channelCmd] fetch ->
INFO 003 Retrieving last config block: 0
2021-05-17 16:19:36.270 UTC [cli.common] readBlock
-> INFO 004 Received block: 0
Decoding config block to JSON and isolating config
to Org1MSPconfig.json
+ configtxlator proto_decode --input config_block.
pb --type common.Block
+ jq '.data.data[0].payload.data.config'
```

```

+ jq '.channel_group.groups.Application.groups.
  Org1MSP.values += {"AnchorPeers":{"mod_policy":
    "Admins","value":{"anchor_peers": [{"host": "
      peer0.org1.example.com","port": 7051}]}}',"
    version": "0"}}' Org1MSPconfig.json
Generating anchor peer update transaction for Org1
on channel mychannel
+ configtxlator proto_encode --input Org1MSPconfig
.json --type common.Config
+ configtxlator proto_encode --input
  Org1MSPmodified_config.json --type common.
  Config
+ configtxlator compute_update --channel_id
  mychannel --original original_config.pb --
  updated modified_config.pb
+ configtxlator proto_decode --input config_update
.pb --type common.ConfigUpdate
+ jq .
++ cat config_update.json
+ echo '{"payload":{"header":{"channel_header":{"
  channel_id:"mychannel", "type":2},"data":{"
  config_update":{"' '"channel_id":"' '"mychannel
  ","' '"isolated_data":"' '},' '"read_set":"' '}'
  '"groups":"' '{' '"Application":"' '{' '"groups
  ":"' '{' '"Org1MSP":"' '{' '"groups":"' '{' '"
  mod_policy":"' '","' '"policies":"' '{' '"Admins
  ":"' '{' '"mod_policy":"' '","' '"policy":"' null,
  '"version":"' '"0"' '},' '"Endorsement":"' '{'
  '"mod_policy":"' '","' '"policy":"' null, '"
  version":"' '"0"' '},' '"Readers":"' '{' '"
  mod_policy":"' '","' '"policy":"' null, '"version
  ":"' '"0"' '},' '"Writers":"' '{' '"mod_policy":"'
  '","' '"policy":"' null, '"version":"' '"0"' '}'
  '},' '"values":"' '{' '"MSP":"' '{' '"mod_policy
  ":"' '","' '"value":"' null, '"version":"' '"0"'
  '}' '},' '"version":"' '"0"' '}' '},' '"
  mod_policy":"' '","' '"policies":"' '{' '"
  values":"' '{' '"version":"' '"0"' '}' '},' '"
  mod_policy":"' '","' '"policies":"' '{' '"
  values":"' '{' '"version":"' '"0"' '}' ','"
  write_set":"' '{' '"groups":"' '{' '"Application
  ":"' '{' '"groups":"' '{' '"Org1MSP":"' '{' '"
  groups":"' '{' '"mod_policy":"' '"Admins',' '"
  policies":"' '{' '"Admins":"' '{' '"mod_policy":"'
  '","' '"policy":"' null, '"version":"' '"0"'
  '},' '"Endorsement":"' '{' '"mod_policy":"' '","'

```

```

        "policy":' null, "version":' '"0"' '},' '
Readers":' '{' "mod_policy":' '"', ' "policy
":' null, "version":' '"0"' '},' "Writers":'
'{' "mod_policy":' '"', ' "policy":' null, "
version":' '"0"' '}, '},' "values":' '{' '
AnchorPeers":' '{' "mod_policy":' '"Admins",'
"value":' '{' "anchor_peers":' '[' '{' "host
":' "peer0.org1.example.com",' "port":' 7051
'},' '},' "version":' '"0"' '},' "MSP":'
'{' "mod_policy":' '"', ' "value":' null, "
version":' '"0"' '}, '},' "version":' '"1"'
'},' "mod_policy":' '"', ' "policies":'
'{}', ' "values":' '{}', ' "version":' '"0"' '},
'},' "mod_policy":' '"', ' "policies":' '{}',
' "values":' '{}', ' "version":' '"0"' '}, '}}}',
+ configtxlator proto_encode --input
  config_update_in_envelope.json --type common.
  Envelope
2021-05-17 16:19:36.793 UTC [channelCmd]
  InitCmdFactory -> INFO 001 Endorser and orderer
  connections initialized
2021-05-17 16:19:36.812 UTC [channelCmd] update ->
  INFO 002 Successfully submitted channel update
Anchor peer set for org 'Org1MSP' on channel '
  mychannel'
Setting anchor peer for org2...
Using organization 2
Fetching channel config for channel mychannel
Using organization 2
Fetching the most recent configuration block for
  the channel
+ peer channel fetch config config_block.pb -o
  orderer.example.com:7050 --
  ordererTLSHostnameOverride orderer.example.com
  -c mychannel --tls --cafile /opt/gopath/src/
  github.com/hyperledger/fabric/peer/
  organizations/ordererOrganizations/example.com/
  orderers/orderer.example.com/msp/tlscacerts/
  tlscacert.pem
2021-05-17 16:19:37.225 UTC [channelCmd]
  InitCmdFactory -> INFO 001 Endorser and orderer
  connections initialized
2021-05-17 16:19:37.230 UTC [cli.common] readBlock
  -> INFO 002 Received block: 1
2021-05-17 16:19:37.232 UTC [channelCmd] fetch ->
  INFO 003 Retrieving last config block: 1

```

```

2021-05-17 16:19:37.234 UTC [cli.common] readBlock
-> INFO 004 Received block: 1
Decoding config block to JSON and isolating config
to Org2MSPconfig.json
+ configtxlator proto_decode --input config_block.
pb --type common.Block
+ jq '.data.data[0].payload.data.config'
Generating anchor peer update transaction for Org2
on channel mychannel
+ jq '.channel_group.groups.Application.groups.
Org2MSP.values += {"AnchorPeers":{"mod_policy":
"Admins","value":{"anchor_peers": [{"host": "
peer0.org2.example.com","port": 9051}]}},"
version": "0"}}' Org2MSPconfig.json
+ configtxlator proto_encode --input Org2MSPconfig
.json --type common.Config
+ configtxlator proto_encode --input
Org2MSPmodified_config.json --type common.
Config
+ configtxlator compute_update --channel_id
mychannel --original original_config.pb --
updated modified_config.pb
+ configtxlator proto_decode --input config_update
.pb --type common.ConfigUpdate
+ jq .
++ cat config_update.json
+ echo '{"payload":{"header":{"channel_header":{"
channel_id":"mychannel", "type":2}},"data":{"
config_update":{"' '"channel_id":"' '"mychannel
", ' '"isolated_data":"' '},' '"read_set":"' '}'
'"groups":"' '{' '"Application":"' '{' '"groups
":' '{' '"Org2MSP":"' '{' '"groups":"' '{' '"
mod_policy":"' '"', ' '"policies":"' '{' '"Admins
":' '{' '"mod_policy":"' '"', ' '"policy":"' null,
'"version":"' '"0"' '},' '"Endorsement":"' '{'
'"mod_policy":"' '"', ' '"policy":"' null, '"
version":"' '"0"' '},' '"Readers":"' '{' '"
mod_policy":"' '"', ' '"policy":"' null, '"version
":' '"0"' '},' '"Writers":"' '{' '"mod_policy":"'
'"', ' '"policy":"' null, '"version":"' '"0"' '}'
'}, ' '"values":"' '{' '"MSP":"' '{' '"mod_policy
":' '"', ' '"value":"' null, '"version":"' '"0"'
'}' '},' '"version":"' '"0"' '},' '"
mod_policy":"' '"', ' '"policies":"' '{' '"
values":"' '{' '"version":"' '"0"' '},' '"
mod_policy":"' '"', ' '"policies":"' '{' '"

```

```

values":' '{},' "version":' '0' '},' "
write_set":' '{' "groups":' '{' "Application
":' '{' "groups":' '{' "Org2MSP":' '{' "
groups":' '{},' "mod_policy":' "Admins",' "
policies":' '{' "Admins":' '{' "mod_policy":'
''," ' "policy":' null, "version":' '0'
'},' "Endorsement":' '{' "mod_policy":' '",'
' "policy":' null, "version":' '0' '},' "
Readers":' '{' "mod_policy":' '",' "policy
":' null, "version":' '0' '},' "Writers":'
' '{' "mod_policy":' '",' "policy":' null, "
version":' '0' '},' "values":' '{' "
AnchorPeers":' '{' "mod_policy":' "Admins",'
' "value":' '{' "anchor_peers":' '[' '{' "host
":' "peer0.org2.example.com",' "port":' 9051
'}' '},' "version":' '0' '},' "MSP":'
' '{' "mod_policy":' '",' "value":' null, "
version":' '0' '},' "version":' "1"
'}' '},' "mod_policy":' '",' "policies":'
' '{},' "values":' '{},' "version":' "0" '},'
'},' "mod_policy":' '",' "policies":' '{},'
' "values":' '{},' "version":' "0" '}' '}}}'
+ configtxlator proto_encode --input
  config_update_in_envelope.json --type common.
  Envelope
2021-05-17 16:19:37.489 UTC [channelCmd]
  InitCmdFactory -> INFO 001 Endorser and orderer
  connections initialized
2021-05-17 16:19:37.503 UTC [channelCmd] update ->
  INFO 002 Successfully submitted channel update
Anchor peer set for org 'Org2MSP' on channel '
  mychannel'
Channel 'mychannel' joined

```

D.1.2 “./network.sh deployCC -ccn record -ccp ../app/chaincode -ccl go” Output

```

deploying chaincode on channel 'mychannel'
executing with the following
- CHANNEL_NAME: mychannel
- CC_NAME: record
- CC_SRC_PATH: ../app/chaincode
- CC_SRC_LANGUAGE: go
- CC_VERSION: 1.0
- CC_SEQUENCE: 1
- CC_END_POLICY: NA

```

```

- CC_COLL_CONFIG: NA
- CC_INIT_FCN: NA
- DELAY: 3
- MAX_RETRY: 5
- VERBOSE: false
Vendoring Go dependencies at ../app/chaincode
~/go/src/github.com/kalne/fabric-samples/app/
  chaincode ~/go/src/github.com/kalne/fabric-
    samples/test-network
~/go/src/github.com/kalne/fabric-samples/test-
  network
Finished vendoring Go dependencies
+ peer lifecycle chaincode package record.tar.gz
  --path ../app/chaincode --lang golang --label
  record_1.0
+ res=0
Chaincode is packaged
Installing chaincode on peer0.org1...
Using organization 1
+ peer lifecycle chaincode install record.tar.gz
+ res=0
2021-05-17 17:21:56.582 BST [cli.lifecycle.
  chaincode] submitInstallProposal -> INFO 001
  Installed remotely: response:<status:200
  payload:"\nKrecord_1.0:
  c42923bce628c5d8e7c879210bd7b94181334bcbe3e0fe67341e7f3bac72d075
  \022\nrecord_1.0" >
2021-05-17 17:21:56.583 BST [cli.lifecycle.
  chaincode] submitInstallProposal -> INFO 002
  Chaincode code package identifier: record_1.0:
  c42923bce628c5d8e7c879210bd7b94181334bcbe3e0fe67341e7f3bac72d075

Chaincode is installed on peer0.org1
Install chaincode on peer0.org2...
Using organization 2
+ peer lifecycle chaincode install record.tar.gz
+ res=0
2021-05-17 17:22:10.686 BST [cli.lifecycle.
  chaincode] submitInstallProposal -> INFO 001
  Installed remotely: response:<status:200
  payload:"\nKrecord_1.0:
  c42923bce628c5d8e7c879210bd7b94181334bcbe3e0fe67341e7f3bac72d075
  \022\nrecord_1.0" >
2021-05-17 17:22:10.686 BST [cli.lifecycle.
  chaincode] submitInstallProposal -> INFO 002
  Chaincode code package identifier: record_1.0:

```

c42923bce628c5d8e7c879210bd7b94181334bcbe3e0fe67341e7f3bac72d075

Chaincode is installed on peer0.org2

Using organization 1

+ peer lifecycle chaincode queryinstalled

+ res=0

Installed chaincodes on peer:

Package ID: record_1.0:

c42923bce628c5d8e7c879210bd7b94181334bcbe3e0fe67341e7f3bac72d075
, Label: record_1.0

Query installed successful on peer0.org1 on
channel

Using organization 1

+ peer lifecycle chaincode approveformyorg -o
localhost:7050 --ordererTLSHostnameOverride
orderer.example.com --tls --cafile /home/email/
go/src/github.com/ka1ne/fabric-samples/test-
network/organizations/ordererOrganizations/
example.com/orderers/orderer.example.com/msp/
tlscacerts/tlsca.example.com-cert.pem --
channelID mychannel --name record --version 1.0
--package-id record_1.0:
c42923bce628c5d8e7c879210bd7b94181334bcbe3e0fe67341e7f3bac72d075
--sequence 1

+ res=0

2021-05-17 17:22:12.994 BST [chaincodeCmd]

ClientWait -> INFO 001 txid [82468

da003dd0b559266d1106e0feacbe27c6da30b5ee17656db183db55462cf

] committed with status (VALID) at localhost
:7051

Chaincode definition approved on peer0.org1 on
channel 'mychannel'

Using organization 1

Checking the commit readiness of the chaincode
definition on peer0.org1 on channel 'mychannel'
,...

Attempting to check the commit readiness of the
chaincode definition on peer0.org1, Retry after
3 seconds.

+ peer lifecycle chaincode checkcommitreadiness --
channelID mychannel --name record --version 1.0
--sequence 1 --output json

+ res=0

```
{  
  "approvals": {  
    "Org1MSP": true,  

```



```

        "Org2MSP": false
    }
}
Checking the commit readiness of the chaincode
  definition successful on peer0.org1 on channel
    'mychannel'
Using organization 2
Checking the commit readiness of the chaincode
  definition on peer0.org2 on channel 'mychannel'
    ...
Attempting to check the commit readiness of the
  chaincode definition on peer0.org2, Retry after
    3 seconds.
+ peer lifecycle chaincode checkcommitreadiness --
  channelID mychannel --name record --version 1.0
  --sequence 1 --output json
+ res=0
{
    "approvals": {
        "Org1MSP": true,
        "Org2MSP": false
    }
}
Checking the commit readiness of the chaincode
  definition successful on peer0.org2 on channel
    'mychannel'
Using organization 2
+ peer lifecycle chaincode approveformyorg -o
  localhost:7050 --ordererTLSHostnameOverride
  orderer.example.com --tls --cafile /home/email/
  go/src/github.com/kaine/fabric-samples/test-
  network/organizations/ordererOrganizations/
  example.com/orderers/orderer.example.com/msp/
  tlscacerts/tlsca.example.com-cert.pem --
  channelID mychannel --name record --version 1.0
  --package-id record_1.0:
  c42923bce628c5d8e7c879210bd7b94181334bcbe3e0fe67341e7f3bac72d075
  --sequence 1
+ res=0
2021-05-17 17:22:21.327 BST [chaincodeCmd]
  ClientWait -> INFO 001 txid [54501
  cbfc4d34d751800aa8c873e2c4da20bf2fecc5665c60f03764cd00cd790
  ] committed with status (VALID) at localhost
  :9051
Chaincode definition approved on peer0.org2 on
  channel 'mychannel'

```

```

Using organization 1
Checking the commit readiness of the chaincode
    definition on peer0.org1 on channel 'mychannel'
    ...
Attempting to check the commit readiness of the
    chaincode definition on peer0.org1, Retry after
    3 seconds.
+ peer lifecycle chaincode checkcommitreadiness --
    channelID mychannel --name record --version 1.0
    --sequence 1 --output json
+ res=0
{
    "approvals": {
        "Org1MSP": true,
        "Org2MSP": true
    }
}
Checking the commit readiness of the chaincode
    definition successful on peer0.org1 on channel
    'mychannel'
Using organization 2
Checking the commit readiness of the chaincode
    definition on peer0.org2 on channel 'mychannel'
    ...
Attempting to check the commit readiness of the
    chaincode definition on peer0.org2, Retry after
    3 seconds.
+ peer lifecycle chaincode checkcommitreadiness --
    channelID mychannel --name record --version 1.0
    --sequence 1 --output json
+ res=0
{
    "approvals": {
        "Org1MSP": true,
        "Org2MSP": true
    }
}
Checking the commit readiness of the chaincode
    definition successful on peer0.org2 on channel
    'mychannel'
Using organization 1
Using organization 2
+ peer lifecycle chaincode commit -o localhost
    :7050 --ordererTLSHostnameOverride orderer.
    example.com --tls --cafile /home/email/go/src/
    github.com/kaine/fabric-samples/test-network/

```

```

organizations/ordererOrganizations/example.com/
orderers/orderer.example.com/msp/tlscacerts/
tlsca.example.com-cert.pem --channelID
mychannel --name record --peerAddresses
localhost:7051 --tlsRootCertFiles /home/email/
go/src/github.com/kaine/fabric-samples/test-
network/organizations/peerOrganizations/org1.
example.com/peers/peer0.org1.example.com/tls/ca
.crt --peerAddresses localhost:9051 --
tlsRootCertFiles /home/email/go/src/github.com/
kaine/fabric-samples/test-network/organizations
/peerOrganizations/org2.example.com/peers/peer0
.org2.example.com/tls/ca.crt --version 1.0 --
sequence 1
+ res=0
2021-05-17 17:22:29.844 BST [chaincodeCmd]
ClientWait -> INFO 001 txid [1
d868b11788fa57d3800618e2e4597087825dc43f00341cc7727ff936fbeed91
] committed with status (VALID) at localhost
:9051
2021-05-17 17:22:29.854 BST [chaincodeCmd]
ClientWait -> INFO 002 txid [1
d868b11788fa57d3800618e2e4597087825dc43f00341cc7727ff936fbeed91
] committed with status (VALID) at localhost
:7051
Chaincode definition committed on channel '
mychannel'
Using organization 1
Querying chaincode definition on peer0.org1 on
channel 'mychannel'...
Attempting to Query committed status on peer0.org1
, Retry after 3 seconds.
+ peer lifecycle chaincode querycommitted --
channelID mychannel --name record
+ res=0
Committed chaincode definition for chaincode '
record' on channel 'mychannel':
Version: 1.0, Sequence: 1, Endorsement Plugin:
escc, Validation Plugin: vscc, Approvals: [
Org1MSP: true, Org2MSP: true]
Query chaincode definition successful on peer0.
org1 on channel 'mychannel'
Using organization 2
Querying chaincode definition on peer0.org2 on
channel 'mychannel'...
Attempting to Query committed status on peer0.org2

```

```
, Retry after 3 seconds.
+ peer lifecycle chaincode querycommitted --
  channelID mychannel --name record
+ res=0
Committed chaincode definition for chaincode '
  record' on channel 'mychannel':
Version: 1.0, Sequence: 1, Endorsement Plugin:
  escc, Validation Plugin: vscc, Approvals: [
  Org1MSP: true, Org2MSP: true]
Query chaincode definition successful on peer0.
  org2 on channel 'mychannel'
Chaincode initialization is not required
```