BlockCert: Blockchain in the supply infrastructure

J. M. Galjaard
W. R. Zonneveld
S. L. Maquelin
D. Hoonhout

TU Delft

BlockCert: Blockchain in the supply infrastructure

by

J. M. Galjaard
W. R. Zonneveld
S. L. Maquelin
D. Hoonhout

Contents

1 Introduction

This report regards the project *"BlockCert: Supply Chain Logistics"* as part of the Blockchain Engineering course (CS4160). For this project, a Proof of Concept (PoC) blockchain-based application for supply chain logistics in the agri-food business was designed and implemented. The project is based on a permissioned blockchain technology, Hyperledger Fabric.

This report is structured as follows. Firstly, the problem statement is analyzed in section 1.1. Thereafter, a general introduction on blockchain technology and Hyperledger Fabric are given in chapter 2. Chapter 3 outlines the PoC's system design, followed by chapter 4 which examines system's implementation details. Future work and limitations of the PoC are presented in chapter 5. In chapter 6 the project development from week to week is discussed. Lastly, a conclusion of the project as a whole is given in chapter 7.

1.1. Problem Analysis

The client *Spark! Living Lab* explores on behalf of Lamb Weston and IsaCert the possibility of a blockchain application, called BlockCert, for their supply chain. A high-level overview is provided in Figure 1.1. Lamb Weston is a producer of frozen potato products and IsaCert is a certification body for the agri-food business. Lamb Weston requires a valid certificate from the farmers to process their potatoes. At the moment, these certificates are sent periodically by mail from IsaCert to Lamb Weston. This is a labor-intensive process, and case studies preceding the BlockCert PoC showed that this process is error-prone.

The BlockCert PoC should showcase how blockchain technologies could address these problems. Furthermore, it should make the process more responsive, allowing the parties to see in real-time which certificates are valid. Moreover, the blockchain application ensures IsaCert to have transparent bookkeeping for the certificates, which improves the trust of the farmers in IsaCert. An added benefit of this is that this allows for a broader applicable PoC to extend to multiple certification bodies, as further discussed in chapter 5. Besides allowing for a transparent and responsive process, it also has the benefit of redundancy, protecting the accessibility to the data from a single point of failure.

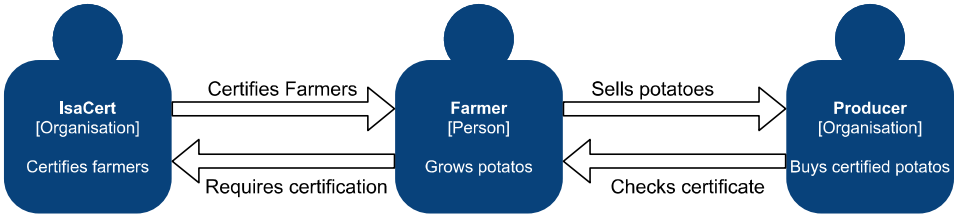

Figure 1.1: A high-level overview of the current process with the three actors, the Certification body IsaCert, the selling party (Farmer), and Producing party. To adhere to food safety regulations, the Producer requires a certificate from a third party (IsaCert) to process the goods of the selling party.

2

Blockchain Technology

Blockchain is a distributed ledger technology containing immutable records. This ledger holds a digital record of transactions, e.g. issuing a certificate to a specific farmer, as well as the current state which resulted from those transactions, e.g. the current set of issued certificates. Although different implementations of blockchain technology exist, they generally employ cryptographic techniques to record facts in a provably immutable manner. The blockchain is distributed on a network of peer nodes that each maintain a copy of the ledger. If a blockchain is permissioned, not just anyone can participate, but rather permissions are required.

2.1. Why use a blockchain technology?

Wüst and Gervais [1] discuss in *"Do you need a blockchain?"* the differences between centralized systems and distributed ledgers are described. There is a difference in public verifiability, transparency, privacy, integrity, and redundancy. A distributed ledger allows for better public verifiability as a transaction will only be recorded after verification of a configurable number of participants in the blockchain. In a centralized system, this is harder, as different observers may not have the same view of the state of the data. Public verifiability also protects the transparency and integrity of the data. A distributed ledger provides redundancy since there exist replications of the data. However, privacy is harder to achieve in a blockchain environment, but not impossible.

To determine the need for a blockchain, certain characteristics of the problem described in section 1.1 must be considered. First of all, the problem dictates the need for a system to store state, namely the certificates, hence the use of a database or ledger system is appropriate. Second, the problem for the PoC has that from a theoretic perspective, only one party should be able to update the blockchain state. Whereas the other parties should function in a *read-only* manner, e.g. see the status of the certificate(s). However, the PoC should be extensible to a scenario with multiple certification bodies. Such a scenario would have to allow multiple parties to have write access. If there is only one writer, a blockchain solution is not necessary. For example, the certificate body could maintain a database system to record facts regarding certificates, and provide an API to parties that require access to the stored data. In this scenario, the other parties must trust the curator of the database system to be trustworthy. As the solution needs to be extensible, a blockchain solution may still be warranted. Third, the participating parties feel there is no trusted third party that is always online that can be used. Fourth, all writers are known by all parties. However, not all writers are fully trusted by the other parties. Namely, the farmers do not fully trust IsaCert not to make any mistakes when creating certificates. Lastly, the data should not be public, only authorized entities should be able to have access to specific data. Based on these characteristics of the problem, [1] proposes that blockchain technology is warranted and to choose a private permissioned blockchain solution.

2.2. Hyperledger Fabric

This section provides a brief overview of some key concepts in Hyperledger Fabric (HLF). The contents of this section are based on the documentation of HLF which can be consulted for more information

[2].

Hyperledger Fabric offers a permissioned blockchain technology ensuring confidentiality through its channel architecture and offers flexibility through its modular architecture. Being a permissioned blockchain technology, HLF is very useful in industry where organisations do not fully trust one another.

2.2.1. Network

Consider a high-level layout of an HLF network, as depicted in Figure 2.1. An explanation on each of the components in the network follows in the rest of this chapter. A HLF network consists of a number of peer nodes which are connected and can communicate over channels. These channels allow peer nodes to communicate in private with others in the same channel. A peer node is owned by an organisation and hosts both instances of ledgers and instances of chaincode. The chaincode is both installed on peers and defined on a channel, thereby defining *who* is allowed to do *what* within the channel. A client application is also associated with an organisation and can access the ledger directly via a peer of the organisation. This access is managed by the chaincode installed on the peer. The chaincode helps generate transactions which are distributed over the channel.

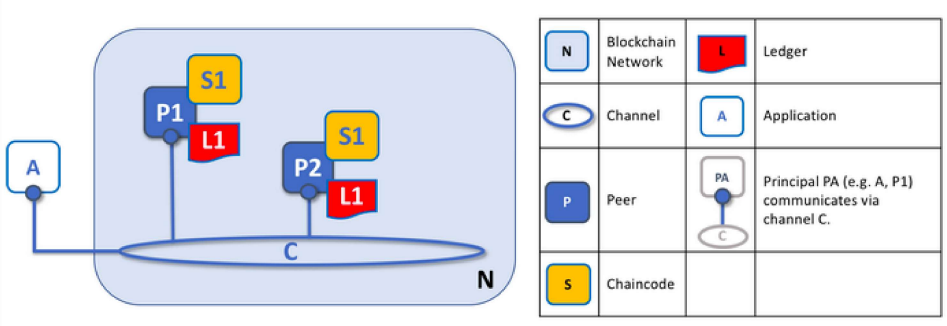

Figure 2.1: High-level architecture overview taken from the HLF documentation [3].

Note, however, that this scenario sketch omits certain components, e.g. the ordering service and certificate authorities. These components were left intentionally out of the equation to provide a clear image. These components will not be discussed in the report as they are not a key aspect of our implementation. Nonetheless, they play an important role in the HLF blockchain technology and more information can be found in the HLF documentation [4], [5].

2.2.2. Peer nodes

Every organisation that needs to be included in the HLF blockchain should have at least one peer node. On this peer node a copy of the ledger is kept as well as instances of chaincode. A peer can be part of multiple channels and has a ledger for each channel in which it participates.

2.2.3. Channels

Via a channel multiple peers can communicate with each other. If a transaction is done on one peer, the transaction is distributed over the channel to the other peers, using a so-called gossip protocol. This means that if your peer is not part of a channel, you also do not have the ledger associated with it. Thus one can share information with others in a channel without sharing it with every organisation taking part in the blockchain. In addition, although this feature is not utilized in the PoC, HLF allows the use of Collections or Private Data. This allows peers to securely share data with a subset of the peers within a channel.

2.2.4. Identities

A digital identity in the form of a cryptographically validated digital certificate that complies with X.509 standard is issued by a Certificate Authority (CA). In this certificate a public-private key pair is defined. When a user is registered with a Fabric CA, a role of admin, peer, client, orderer, or member must be associated with the user. Thus not only a client and an admin have an identity, but also the peer has

an identity which for instance can be used to determine it's rights in a channel. A network can have multiple CA's which establishes a chain of trust for any certificate issued by a CA in the chain.

A Membership Service Provider (MSP) is used for authentication. An MSP contains a list of permissioned identities, which determine the administrative or participatory rights of the identities. Thus an MSP turns an identity into a role by identifying specific privileges an actor has on a node or channel. Each organisation should have their own MSP to manage their members. This MSP can then be used to link an identity to an organisation.

2.2.5. Chaincode

A smart contract defines the transaction logic that controls the life cycle of a business object contained in the world state. Note that a transaction is not necessarily a transfer of ownership. A transaction can for example also be the creation of an object. This creation is denoted by a transaction on the blockchain. As a matter of fact, an object does not even need an owner.

There are two types of transactions. A transaction can alter the world state or it can query the blockchain. Together the world state and the blockchain form the ledger. The world state can be seen as a database that holds current values of a set of ledger states and the blockchain as a transaction log that records all changes that have resulted in the current world state.

Chaincode is the packaged smart contract, which can be installed on peer nodes and defined on a channel. The terms chaincode and smart contracts are used interchangeably in the HLF documentation.

Access control on the assets in the ledger can be enforced in the chaincode. As a transaction is done by an identity, the chaincode can use the certificate of the identity to check information such as its organisation in order to decide if the identity is allowed to make the transaction. For this type of access control, assets may need to include some information to indicate who is allowed access.

To further ensure confidentiality of the data, the ledger data can be encrypted via file system encryption on the peer and data in-transit can be encrypted via TLS. The file system encryption is, however, not included in the developed application.

2.2.6. Applications

An application is used to interact with the ledger. To access the ledger the application utilizes the chaincode to do transactions and queries. In order to access the ledger one needs an identity which is saved in ones wallet. A wallet holds one or more identities and can just be a folder on a computer containing these identities. With the information of the wallet, the application connects to a gateway which gives access to one or more peers in the network. The application now has access to all channels on these peers. One of the channels is chosen as well as one of the contracts defined in the chaincode of that channel. After these steps, the application is ready to submit transactions to the network and process the responses.

3

System design

This chapter dives further into the system design of the developed PoC. It regards the configuration of the HLF network and the chaincode runs on it. In addition, the developed applications are discussed. For this chapter a visual depicted was found useful to quickly graps the envisioned system. As such, Figure 3.1 was created to aid to the reader. It showcases the different nodes that make up the developed PoC. Notice the similarities with the simple network as depicted in Figure 2.1.

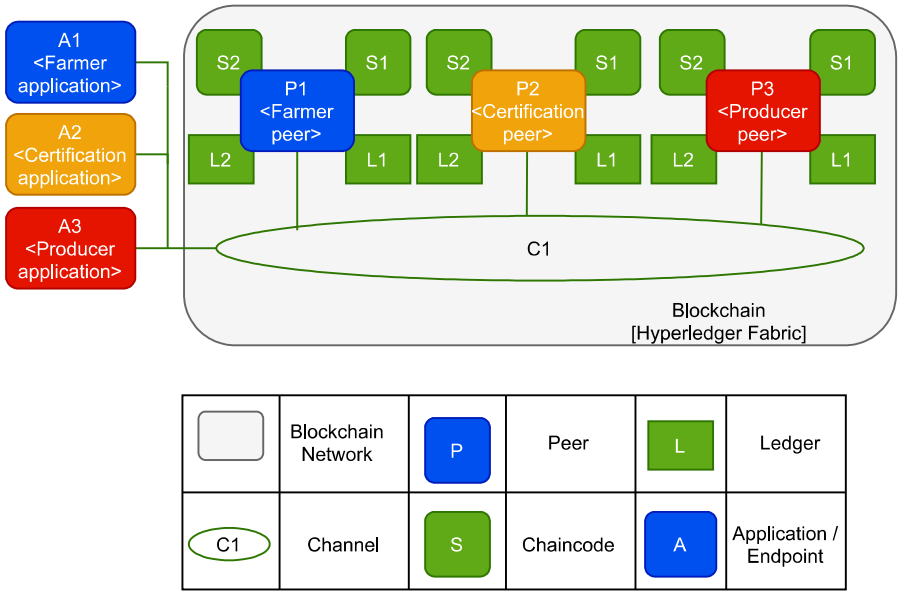

Figure 3.1: HLF network architecture of BlockCert, based on the architecture diagrams used in the HLF documentation.

3.1. Network design

The network is designed according to the wishes of the client and so it can easily be extended for more organisations in the future. An overview of the components is found in Figure 3.1.The applications are not part of the blockchain network, but are connected to the channel as they can invoke the chaincode which is installed on the peer nodes.

3.1.1. Farmer peer

The organisations in the scenario are the producer Lamb Weston, the certification body IsaCert and every farmer. The farmers will share a peer to reduce their expenses and so they do not need any technical expertise. The farmers could turn to an online hosting platform which takes care of the technical setup of the blockchain. Thus the system has 3 peers in total (P1-P3). Every peer has the same

smart contracts (S1 & S2) installed on it and has a copy of the ledgers (L1 & L2).

3.1.2. Channels

The peers are all connected on the same channel (C1). Additional channels can be added when a certification body wants to join the blockchain. If a new certification body joins the blockchain, the second channel would connect the farmer peer, the second certification body peer and the producer peer.

As the ledger of a channel is only available of those peers on the channel, the channels are confidential. The certificates created by IsaCert should be accessible for the producer and the farmer that acquires the certificate. A new channel for a new certification body or producer makes sure there is no business-sensitive information shared. On the other hand, the farmers are sharing a peer and therefore extra measures for confidentiality must be taken. To make sure the farmers can only request their own certificate, access control is built into the chaincode.

3.2. Chaincode and applications

In the certificate chaincode (S1), the certificate data is defined, as well as the logic related to the certificates. Certificates can be created, updated and deleted using transactions. The certificates can also be queried by state, certificate acquirer or a combination of the two and by the certification body that created the certificate. In the farmer chaincode (S2), the farmer data is defined, such as the farmers first and last name and his address. The applications use the chaincodes to access the ledgers. There are 3 applications (A1-A3), one for each party. Below, the functionality available in each application is described and how this is enforced.

3.2.1. Access control

Not all functionality described above should be available to every organisation. The farmers should only be able to request their own certificate. Producers should only be able to query certificates. The certification body is allowed to do all of the above. The access control to the functionalities is managed in the chaincode. The identity of the one requesting some functionality is available in the chaincode and can be used to enforce access control. The identity contains information on the organisation it belongs to as well as a unique identifier.

3.2.2. Front-end architecture

The three applications are each connected to a front-end application which provides a graphical interface for invoking functionality and providing feedback. In the front-end application for the farmers, all certificates of a farmer are shown and the data fields can be inspected. The front-end application for the certification body provides an overview of all certificates and farmers, as well as an interface for creating updating and deleting certificates and farmers. The second chaincode with the farmer assets are useful for IsaCert to create certificates. Now they only need to add information for a farmer once, which can be used for future certificates. This feature is especially useful in the future when farmers will have the same client number (id) when new certification bodies are added. The producer's front-end application provides an interface to search for certificates given a client number of an acquirer and shows a certificate if the farmer posses a valid certificate. It is necessary for the producer to receive the entire certificate so the producer can show the certificates to an external quality controller when it gets audited. The producer has also read access to the farmers. The farmers themselves can only see their own certificates. In the future, the farmers could get the rights to view and update their information.

3.2.3. State of certificates

Each certificate has a state field which indicates if the certificate is valid. The certification body has the right to revoke a certificate and therefore change the state. As the certificate also contains an end date, the state should be changed accordingly. Such an update requires the inspection and possibly alteration of all certificates and can only be performed by the certification body. The check should be performed every day. To make it fail safe, the end date is checked against the current date in the front-end applications as well.

4

Implementation

This chapter describes the important implementation details and discusses the different implementation choices made in this project. The implementation of the final product can be divided into four parts: smart contracts, applications, front-end, and deployment. The requirements for the final product were setup using the MoSCoW method and are listed in Appendix A.

4.1. Blockchain choice

From the problem analysis it was quickly derived that a private blockchain was needed instead of a public one. This is because a public blockchain provides no control on who joins the network, makes transactions, or the cost of transactions (proof of work). A private blockchain does provide these functionalities and therefore only private blockchains needed to be considered. Two widely used private blockchains were compared, namely Hyperledger Fabric (HLF) and Corda. Corda is specialized in the financial sector, and gives strong security guarantees [6]. However, only recently did Corda start expanding outside of the financial domain. HLF however is modularized (see chapter 2), which gives this option a lot of extensibility and is therefore used in a variety of use cases. As the team was unsure which features would all be needed in the project, the extensibility of HLF would be the deciding factor and therefore HLF was chosen as the blockchain to develop the application on.

4.2. Smart contracts

As explained in chapter 2, smart contracts describe the transactions possible on the ledger. In Hyperledger Fabric, two runtimes for smart contracts are supported: JVM and Node.js. This means smart contracts can thus be developed in languages which compile to those runtimes. The main candidates are Javascript, Typescript, and Java. Java offers a lot of structure, but has as a downside that its code is quite verbose compared to the other languages. Javascript lies on the other end of the spectrum, where you need relatively little code but offers very little in terms of structure (e.g. no type variables). Typescript lies in the middle of the two, since it does offer typed variables and its code is still compact. The team therefore decided to implement the smart contracts in Typescript as this allows for rapid prototyping, yet still providing the developers with desired structural properties.

The implemented transactions in the smart contracts can be divided into two categories, one which alters the contents of the ledger, and one which merely queries its content. For the first category the following operations are implemented: Creating a certificate, modifying an existing certificate, and deleting certificates. Querying of certificates can be done in several ways. The most trivial way is to request a certificate with a specific ID. However, it is also possible to query the ledger based on certain attributes of a certificate. Currently querying of certificates based on `Acquirer`, `RegistrationNr`, and `Status` is implemented.

4.2.1. Data structure

The ledger of the blockchain contains two types of data, one corresponding to food certificates, and one to farmers. The fields of the food certificates, were identified by a document shared with the team

from the client. The identified fields are as follows:

- **ID** A unique ID which is used to identify certificates
- **CertNr** The certification number of the certificate. Where the ID is unique, the CertNr is only unique when combined with the RegistrationNr.
- **Start date** Indicative of the date at which the farmer became certified.
- **End date** Indicative of the end date of the certification.
- **AcquirerID** The ID of the owner of the certificate, i.e. a famer.
- **RegistrationNr** The number of the certification body which issued the certificate
- **CertificateURL** This field is for a future feature and will link to the original certificate pdf version.
- **Status** Used to indicate the validity of a certificate, either `ISSUED`, `EXPIRED` or `REVOKED`. This field allows a certification body to retract an issued certificate, without the need of removing it.

The contents for farmers consist of an ID (corresponding to **AcquirerID** in a certificate), a first and last name, and an address. These fields were identified from the same document mentioned before. However now these fields belong to the farmer instead of the certificate itself to avoid redundancy.

4.2.2. Access control

Hyperledger Fabric has the capabilities to give different permissions to different users. Access control can be regulated based on the client identity itself, the organisation an identity belongs to, or a certain attribute of an identity. In consultation with the client, this functionality was used to create the following access control rules:

- Farmers can see their own certificates and their current status (whether or not they currently have a valid certificate).
- Producers can see the valid certificates of all farmers, and information of those farmers.
- The certification body can see all certificates, and can create, modify, and delete certificates. Furthermore the certificate body can create, modify and delete information about farmers.

To determine to which organisation a user belongs, the X.509 certificate (default digital certificate used in Hyperledger Fabric) belonging to the user performing an action is used. In this certificate the MSP (Member Service Provider) which granted the certificate is checked. Since each organisation (farmer, producer, and the certificate body), has an MSP, this can be used to determine to which organisation a user belongs. The X.509 certificate also contains a user-id, which is used to identify individual farmers. This is then used to ensure farmers only gain insight into their own food certificates.

4.3. Applications

Three applications were developed, one for farmers, one for producers, and one for the certification body. Each application only implement functionality which the respective users are allowed to access. However this does not serve as access control as in theory someone could deploy a different application on their peer without the other parties' agreement. For this reason, the access control is built into the chaincode itself (which cannot be altered by just one party), and not in the applications. Like the smart contracts, the applications are implemented in Typescript as the team would then only use a single language to develop the back-end of the application (as well as getting the benefits described in section 4.2).

4.4. Front-end

Although code from the chain-code can be invoked via the command line, it is unsuited for the intended users to interact with the system this way. As such, different user interfaces were developed to allow the different parties to interact with the system according to their respective role. The front-end user interface is built on the Angular[1], a framework created and maintained by Google. This was chosen as it offers a variety of nice features. It allows web-pages to be dynamic and has a easy to use framework for REST-API's. This front-end application is generic for all of the three organizations and it exists of five pages.

First there is a login page. This component exist of an username field and a wallet-key field and is used to identify users who have access to the system. During login the application will check whether the identity exists and whether the private wallet-key also corresponds to that same user. If all credentials are correct a token will be generated and stored in the session-storage. When requesting actions on the asset the user will use this token to identify itself.

The remaining pages are used to modify and query assets. Two pages for the creation of the two assets and two pages to give insight in all existing assets and modifying existing assets. Depending on which application is running it will either return all data or only data for a specific farmer. Furthermore there is also a search box in order to find specific certificates or farmers. This will make it a lot easier when working on a system with hundreds of certificates.

4.5. Deployment

The configuration to deploy the blockchain network is located in several yaml[2] files. Currently the network is configured to run with three *example* organisations, each with their own Membership Service Provider (MSP), Orderer, and Certificate Authority (CA). Besides the CAs belonging to the organisations, there is a root (TLS) CA. Through a trust of command, this CA enables the different entities to interact with eachother by having a common point of trust. Furthermore, the network is set up to run on a single device, by means of Docker compose[3], and thus not in a truly distributed environment.

Taking this next step, i.e. bringing the network to a production network, was left as open work. The rationale behind this is set further apart in section 5.1. Nonetheless, the repository contains documentation and modified Helm charts by Owkin[4] that were created during an early attempt to setup a production network. The team is confident that a next iteration can combine the structure of the test network with the deployment charts. Furthermore, these charts allow for *cross-cluster* deployment, i.e. letting the different organizations manage their peer themselves.

[1] https://angular.io/
[2] Yet Another Markup Language
[3] https://docs.docker.com/compose/
[4] https://github.com/owkin/charts

5 Future work

Throughout the work, the team has aimed to create an application that allows showcasing the use case of Blockchain. The developed PoC allows to demonstrate this, however, aspects were identified that would benefit from more iterations. Moreover, certain features were explicitly left as future work, to properly scope the developed PoC, see also chapter 6.

5.1. Future steps

Production network: Currently, the blockchain is deployed in a test mode on a single machine. A future goal would be to run the blockchain on separate machines (production network). Nonetheless, the chaincode developed on a test network can directly be used (except for some configuration options) in a production setting.

Security: The deliverable uses a TLS Certificate Authority, allowing for encrypting the data in transit. The ledger data on the peer node is not encrypted at the moment. Encryption of this data could be achieved in multiple ways, e.g. using file system encryption or using a public-private key combination.

Access control: The chaincode takes care of access control on the ledger data. The next step is to install access control on the front-end applications as well by using some form of authentication. This feature is most important for the farmer application, as the applications for the producer and certification body can be used within their secured network environment.

Extra types of certificates: Currently, a single certificate type is implemented. The implemented certificate allows for a farmer to be certified as a whole. This was deemed a good step for the prototype, however, an interview with IsaCert made clear that different granularities in this certification are required for production. A prime example is that sometimes only a part of an overarching entity is to be certified.

Enhanced certificate control: Currently, the PoC does not allow the farmer to decide on whom contracts should be shared. E.g. a farmer may only want to share this information with a subset of the producers that accept certificates from a certification body. A future step would enhance the current chaincode to enable the farmer to have this control.

Event listeners: During meetings with both IsaCert and Lamb Weston, it became evident that certain organization processes would benefit from notifications or other interactions based on the ledger state. To this end, HLF allows for the creation of event listeners on channels[1]. Further requirement engineering would allow to better assess the needed types of event listeners to make the PoC fit better in the process flow.

5.1.1. Future course of action

For the transition phase from the current systems to a blockchain application, extra functionality might be necessary such as an exporting functionality of the certificate data. Of course, this means when taking the data from the blockchain the data could become stale.

[1] https://hyperledger-fabric.readthedocs.io/en/release-2.2/peer_event_services.html

To start real-life deployment using the blockchain application, integration must be provided for systems certification bodies and producers are working with. For instance, IsaCert desires to fill out the certificates with their application, and the data should be ported to the blockchain application to prevent double work on their side. In a later phase, it would be desirable for the farmers to get functionality for requesting certification and handing in necessary documents on the same platform as on which they can view their certificates.

The extensibility of the application, as discussed before in this report, was kept in mind during the creation of the project. Conversations with the client indicated that other types of organizations, such as transport companies, may also grow interested in becoming part of the blockchain solution in later stages.

6

Development process

This chapter discusses the development process and all the investigated topics that did not make it into the PoC. The following steps were made per week:

1. Week 1 was used to start up the course, and read into the basics of Blockchains. This time period was also used by the team to get to know each other and to decide on BlockCert. Furthermore, preliminary requirements were engineered to allow to determine the aspects of the application deemed important by the client in the coming week.

2. Week 2 was the kickoff of the project with the product owners. After the first meetings, in which the need for a private permissioned blockchain technology was discussed, the two blockchain technologies Hyperledger Fabric and Corda were investigated. This included, and was not limited to:

- Reading into the documentations of the different platforms.
- Investigate applied versions of the application.
- Setup development environment to test different pieces of example code.
- Work through sections of the getting started guides.

This research was done throughout the rest of the project. Further mentioned research will be mentioned if it required *significant* effort by the team.

3. Week 3 was used to decide on a preliminary architecture design and the underlying blockchain technology. Hyperledger Fabric was chosen and the team continued to set up a test network example from HLF, making sure everyone was familiar with the basics at the end of the second week.

During this time the client indicated interest in a production network, i.e. a network that runs on multiple machines, preferably in the cloud.

4. Week 4 was a continuation of the efforts started in week 2. The team spend the third week on trying to set up a production network, but ran into multiple problems. Two approaches for setting up a production network were taken in parallel.

- Using the HLF documentation. The documentation provided was a guide on which elements should be considered when setting up a production network but no actual instructions were given. The team looked into using Digital Oceans to set up a cluster for resources. However, setting up a CA proved very difficult and the team ran into multiple problems during the setup of the production network.
- Using prior-art using Kubernetes (K8s) and Helm[1], a deployment manager for K8s. Although limited prior experience with K8s or Helm was present in the team, with relative ease early steps in the setup process were made. However, this process started to dwindle as more features were required and limited documentation resulted in a high development time cost. The results of this are documented in the repository deliverable, which also provides an insight in the undertaking that was done.

[1] This process was documented and is available in the repository in the `charts` directory

The hands-on experience with HLF allowed to re-evaluate the preliminary design that was created in earlier weeks.

5. Week 5 Using the lessons learned from the preceding weeks, the team decided to reduce the man-power in setting up the production network. This meant that the team communicated the unsuccessful attempt at production to the clients, and a test-network was created in agreement with the client. The main advantage was that this allowed the team to develop code that is also suitable for deployment on a production network.

In addition, the team split up, to also focus development effort at experimenting with the creation and usage of chaincode functionality in HLF and TypeScript.

The development of the chaincode was impeded as the team started to build on example code provided by HLF that turned out not to work. After this discovery, the team investigated if the problems were caused by an oversight on their side and since this did not seem to be the case, tried to remedy the problems in the given code. Unfortunately this was not possible, setting back the production of the chaincode by a week.

6. Week 6, the test network that was written was further updated to allow for three different organizations to be spun up. In addition, errors with spinning up the CouchDB and TLS CA nodes were resolved to allow for a more realistic test network.

This meant that the start of the development of the chaincode became eligible to be tested. Which took of from the experiences of the preceding week and the finalized test-network.

The chaincode was developed from a more basic starting point because of the encountered problems in the previous week.

Some research was done into the topics of collections of data, certificate based access control, asset based access control and private data. Even though not all of these topics were used in the design, it helped the team with design choices on confidentiality of the data.

7. Week 7, in the meeting with Lamb Weston, the producer indicated that a search functionality was very important to them in the front-end applications.

Furthermore, chaincode based access control was investigated. Two different concepts were considered in depth, Certificate Atribute Based and Attribute Based Access Control (CABAC and ABAC respectively). The latter was chose as it allowed for a more natural progression in the work, proved to be more flexible in the implementation. In addition the application code was updated to make it more scalable and allows for the parallel execution of the different web applications.

8. Week 8, half of the team started work on the report, to allow for some leeway, in case aspects of the application required improvement based on future feedback. Furthermore, the team focused on iterating on the web application, e.g. providing more feedback to the user, extracting configuration files, code base refactors and the writing of unit tests for the chaincode functionality. This last part was deemed important by the team to verify assumptions made during the process, and to allow for future iterations to have tested source code.

Additionally, a meeting with IsaCert was scheduled, which provided valuable points of feedback, and suggestions. These points were addressed in the report, as well as updating certain features in the GUI. The most notable change was the decision in the underlying chaincode, to allow for a link to a (hosted) document that corresponds to the certificate. This decision was in line with the topics of the meeting with Lamb Weston, which also concerned the integration of a product like BlockCert into the current certification process.

9. Week 9. In the last week of the project, the team worked on the report, documentation, and making the code ready to be transferred to a new team. This included the additional documentation of work that did not make the cut in the handover. Two more features were added. A feature for the certification body to add farmer assets to the blockchain. And a login page was created for the front-end applications.

7

Conclusion

The developed blockchain application provides our client with a PoC which can be used to test the feasibility of such an application in business. The application comes with three front-end applications, which can be used by a farmer to view his certificates; by Lamb Weston to search for valid certificates; and by IsaCert to create, update and delete certificates. Upon creation, the certificates are stored on the blockchain.

The blockchain solution provides real-time insight into the process of certification of Supply Chain Logistics in the setting of potato farmers and potato producers. The product allows gaining insight into the validity of certificates while allowing parties to gain trust in each other and the data by the use of blockchain technology. As a result, the developed PoC allows showcasing how blockchain technology can be used to modernize the problem, as described in section 1.1, while also offers more resistance in terms of availability. It is an improvement on the current system, in which certificates are sent by mail and it is hard to communicate the revocation of certificates.

Hyperledger Fabric makes it easy to add additional organizations by adding peer nodes as well as channels to ensure the confidentiality of business-sensitive information. The chaincode can be deployed as-is on any additional channels and the front-end applications can be reused as well. Therefore making the application highly extensible.

Future features are discussed and are based on conversations with the producer Lamb Weston and the certification body IsaCert. The project will be handed off to another team that may implement some of these features.

A

MoSCoW

This appendix provides the original MoSCoW requirements that were engineered based on meetings with the different stakeholders. They describe the wanted functionality of the developed PoC. For the PoC, all the *must haves* must be present in the final product. *Should haves* are features with high priority after the requirements of the MVP have been achieved. *Could haves* are desirable features but have a low priority in this project. To limit the scope of this project, *Won't haves* are features which are excluded at the start of this project.

A.1. Must haves

- An authorized certification body must be able to issue certificates;
- An authorized certification body must be able to revoke issued certificates;
- Only authorized parties must be able to issue certifications;
- Accessible interface for users (certification body, farmers, producers) to interact with certificate information;
- All involved parties must be able to join and be identified by the network;
- Only parties related to a certification must be able to view a certification's contents and corresponding addenda;
- The product must be implemented using Blockchain technology.

A.2. Should haves

- Farmers should receive a notification when their certificate status is changed;
- The application should be GDPR compliant[1];
- The product should have scalability for more farmers, certificate authorities, and producers.

A.3. Could haves

- The farmer could be able to upload documents to get certified;
- The farmer could be able to share additional information such as an audit report.

A.4. Won't haves

- The developed application won't have advanced/polished User Interface design.

[1] Based on a best effort basis by the team

Bibliography

[1] K. Wüst and A. Gervais, “Do you need a blockchain?” In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, 2018, pp. 45–54. DOI: 10.1109/CVCBT.2018.00011.

[2] Hyperledger Fabric, *A blockchain platform for the enterprise*¶. [Online]. Available: https://hlf.readthedocs.io/en/v2.3.1/.

[3] ——, *A blockchain platform for the enterprise - peers and channels*¶. [Online]. Available: https://hlf.readthedocs.io/en/v2.3.1/peers/peers.html#peers-and-channels.

[4] ——, *A blockchain platform for the enterprise - the ordering service*¶. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.2/orderer/ordering_service.html.

[5] ——, *A blockchain platform for the enterprise - certificate authorities*¶. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.2/identity/identity.html?#certificate-authorities.

[6] D. Newton, *What is corda?* Acessed: 2021-04-03, Jul. 2020. [Online]. Available: https://www.corda.net/blog/what-is-corda/.