TECHNICAL UNIVERSITY OF CRETE

DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

Blockchain Support for a Personal Data Marketplace

Author
Alexandros Tragkas

Thesis Committee
Prof. Minos Garofalakis
(Supervisor)
Prof. Aggelos Bletsas
Prof. Sotirios Ioannidis

February 4, 2022

Abstract

Alexandros Tragkas
Thesis Supervisor: Prof. Minos Garofalakis

The revolution of blockchain technology has matured enough to substantially disrupt the healthcare sector, in terms of security, privacy, and data exchange. Citizens have ceased to trust third-party organizations for storing their personal data, due to the numerous data breaches and illegal exchanges occurring every year. These major issues have given birth to a brokerage and market platform for personal data utilizing blockchain technology, giving users the ability to buy and sell personal health and educational data in a permissioned network. Data owners set specific policies determining who can have access to their data, how, and for what purpose. Data exchange validations are handled by smart contracts reliably enforcing access control by verifying buyers' purposes of usage against data policies. The platform does not store sensitive data itself, inspiring trust to users by handing them exclusive control of their data. The smart contracts ensure that a user's sharing consent is always in control of the transaction output decision. As a result, users are incentivized to share their medical data for any purpose in a reliable and legal compliant manner.

Acknowledgements

After all these years of hard work and life teachings, I want to express my gratitude to everyone who walked beside me and assisted me throughout this journey.

I would like to acknowledge Davide Zaccagnini from Lynkeus for managing the development of this project as part of KRAKEN, funded by the European Union’s Horizon 2020 research and innovation programme.

I would like to thank my supervisor, Professor Minos Garofalakis, for granting me this massive opportunity and trusting me to work on a real blockchain use case with his team at Athena Research Center. I deeply thank the members of the team for assisting in the project’s development and for the motivating conversations.

I am eternally grateful to my family and close friends, for always supporting my decisions and dreams. I want to thank my mother, for teaching me an invaluable perspective in life that will guide me forever. I truly thank my father, for the selfless support and for teaching me virtue, respect, and perseverance. Of course, I can’t thank my brother and sister enough for continuously motivating me to overcome difficulties and achieve my goals.

Lastly, I am thankful to my beloved dogs and cats for the unconditional love they provided me. I dedicate this thesis to my cat, Storm, who passed away after 12 years of sweet companionship.

Contents

List of Figures

List of Tables

List of Abbreviations

HLF Hyperledger Fabric

SC Smart Contract

DB Database

CA Certificate Authority

PIMS Personal Information Management System

P2P Peer to Peer

DLT Distributed Ledger Technology

PoW Proof of Work

PoS Proof of Stake

MSP Membership Service Provider

CSR Certificate Signing Request

OU Organizational Unit

VM Virtual Machine

TPS Transactions per Second

TLS Transport Layer Security

CLI Command Line Interface

SDK Software Development Kit

SSI Self Sovereign Identity

GDPR General Data Protection Regulation

SMPC Secure MultiParty Computation

CRUD Create Read Update Delete

DPO Data Protection Officer

B2B Business to Business

B2C Business to Customer

Chapter 1

Introduction

Security and privacy of sensitive personal data are key requirements for citizens to interact safely in our Digital Society. As technology progresses, Personal Information Management Systems (PIMS) will become a standard as individuals demand trust and ownership of their data management and usage. These systems aim to return the control of the data back to the owner by designing an architecture where the owner can define extensive policies for the data usage as well as giving consent for access to specific parts of data. An architecture of this type integrated into sensitive, trust-requiring data sectors is incredibly innovative.

The data explosion in the healthcare sector in recent years has proved the importance and need for biomedical data [1]. In the emerging future, companies and research institutes will gradually be able to collect an exponentially increasing number of datasets both from the patients' medical records, as well as from the extensive use of IoT devices. However, ensuring data security and privacy of an individual's identity constructs an underlying technological and legal barrier between an organization and a data owner.

Data breaches and identity theft attacks on medical centers are particularly concerning and violate fundamental human rights to privacy [2]. Additionally, illegal exchanges of a patient's generated health data behind the scenes of organizations further render the system unreliable. Essentially, a user must have exclusive ownership of their personal data as defined through legal [3] and ethical considerations. This challenge can be overcome by leveraging a blockchain solution.

It is well established by now that blockchain implementations provide solutions to major problems. Distributed ledgers and peer-to-peer (P2P) networks open up possibilities for state-of-the-art architectures in many sectors where it is important to maintain transaction transparency, immutability, and security. Also, permissioned blockchain networks can set role-based access policies and data access permissions given the fact that members on the network are identifiable.

The idea of our proposal is a marketplace architecture that supports the exchange of personal data while enforcing access control policies using smart contracts on a permissioned blockchain network.

1.1 Objectives

Our work envisions a marketplace that grants the ability to a user to share or trade their personal data by setting a customized data access policy. Combining innovative cryptography technologies, marketing, and businesses in compliance with legal and ethical considerations, the objective lies in establishing a transnational personal data platform where citizens can actively sell their data to potential policy-matching buyers.

The overall architecture is designed to provide solutions to a number of platform issues, including:

1. A user must have the ability to trade personal data in legal and ethical compliance.
2. A user must be the exclusive owner and manage how their data are used.
3. Permissions must be set on a user identity level.
4. The network must be decentralized to remove a central point of trust.
5. Data access control logic cannot be tampered with or bypassed.
6. All transactions must be recorded in an immutable manner.
7. The platform should not host or store any sensitive data.

1.2 Our solution

By carefully inspecting the potential technologies, problems, and solutions, the architectural design of our proposal resolves these issues. Specifically, the core of our solution exploits smart contracts utilizing the data access control logic, also referred to as product or data policy. This policy mainly defines who, why, and for how long they have access to the data. The key characteristics of our solution are:

1. The network is decentralized and permissioned.
2. A seller can set their personal medical data for selling.
3. A seller can specify a policy regarding a piece of data.
4. A buyer can buy (have access to) a product if and only if they match the product's policy.
5. The data access control is realized using smart contracts on the blockchain network.
6. All product exchanges and payment transactions are recorded on the blockchain.
7. No sensitive data are stored on the blockchain. Only the owner holds the data.

Additionally, we have added a layer that functions as a filtering algorithm. Upon browsing the catalogue as a buyer, the algorithm filters the search query to show only products with policies that match the buyer′s as their credentials and purposes of buying are stored in the ledger when the user registers.

We will examine our design in-depth in the following chapters.

1.3 Thesis structure

In this section, we provide an outline of the thesis and the topics that we will cover.

In Chapter 2, we explain the key concepts of blockchain technology. We will explain the meaning of a distributed ledger, blockchain networks, smart contracts, and the frameworks that we use in our project.

In Chapter 3, we define our solution's architecture along with the application design elements. In addition, we state the project's development process that we followed to design and deploy the system. Lastly, we analyze the project's software and hardware requirements and the reasons for each decision.

In Chapter 4, we discuss the implementation of the project's blueprint. Specifically, we describe the following:

- The strategy upon which we decided to use the chosen frameworks, development and testing tools, programming languages, and platforms.
- The plan behind unit testing, validation, and integration. We describe how the testing is separated at a network and application level, why we took this approach.
- Rationale for choosing the various tools and technologies. The motivation and advantages of building on Hyperledger Fabric.

In Chapter 5, we conclude the thesis and provide our insights on the future development and upgrades of our platform.

Chapter 2

Background and Related Work

In this chapter, we will provide information regarding the literature and key concepts used in this project. We will look into multiple topics including blockchain technology, smart contracts, and frameworks.

2.1 Distributed Ledger Technology - Blockchain

Blockchain, a type of Distributed Ledger Technology (DLT), is a sequence of blocks of data records stored in a distributed, transparent, and immutable manner. Invented by Satoshi Nakamoto [4] with the Bitcoin use case, blockchain utilizes practically unbreakable cryptographic schemes, which can leverage secure architectures enforcing privacy while decentralizing the decision-making process. Essentially, applications built using blockchain realize a trust-less system that incentivizes each participant to make the right decision and not act with malicious intent.

Various ways have been developed as to how decentralization and trust is achieved in blockchain applications, called consensus mechanisms. In short, consensus mechanisms are protocols that ensure nodes are synchronized and agree on the validity of transactions that get appended to the blockchain. The first consensus mechanism, used in Bitcoin and later in many other projects, is known as Proof of Work (PoW). Other mechanisms have also been developed including Proof of Stake (PoS), Delegated Proof of Stake (DPoS), etc.

2.2 Blockchain networks

In the last decade, multiple big tech companies have been interested in blockchain technology. This has generated large amounts of capital in researches for new use cases and inventions in this sector [5]. Subsequently, this exploration has given birth to several types of blockchain networks varying depending on the use case's needs. For example, projects using sensitive data or decentralizing the supply chain management cannot be open to the public, nor participants can be anonymous. The two main types of blockchain networks are public or permissionless and private or permissioned.

2.2.1 Public Blockchain

Public blockchain network architectures constitute the high majority of the current deployed projects. In general, their key aspects consist mainly of these features:

- Participants are anonymous.
- Anyone can participate without a need of authorization or verification.
- There must be an incentive for the participant to utilize consensus.

The most common public blockchains are Bitcoin and Ethereum using native cryptocurrencies as an incentive integrated into their consensus protocols.

2.2.2 Private Blockchain

Private or permissioned networks offer blockchain solutions to use cases where a public blockchain would never sustain. Important core features of private blockchains include:

1. Participants need to be verified and approved to participate on the network
2. A member's identity is always known, acting as a malicious act blocker
3. There may or may not be a native currency, depending on the implementation

In essence, private networks take the core features of the blockchain, meaning data immutability, transparency and consensus, and apply them to a use case where privacy and high throughput in terms of transactions is more important. This is achieved by implementing alternative consensus mechanisms that are much faster compared to others used by permissionless networks such as PoW, given the fact that it is much more unlikely for an identified member to act maliciously, and even if they do, the issue would be recoverable in general terms.

2.3 Smart Contracts

A smart contract (SC) is a piece of self-executing code representing an agreement between the two sides of a transaction. Firstly introduced by Nick Szabo [6], smart contracts function as a digitally defined protocol that guarantees the outcome will be what it has promised. The substantial evolution of smart contracts came with Vitalik Buterin, founder of Ethereum [7], where he made possible the establishment of smart contracts on the chain.

This technology can be used to remove the middleman from many services while maintaining trust in the overall process. Also, every transaction is recorded on the blockchain immutably, making them traceable and irreversible. The most common use cases as examples include voting systems, auctions, assets exchanges and others.

2.4 Hyperledger Fabric

Hyperledger is a collaboration between the Linux Foundation and multiple big tech organizations including IBM, Intel, Consensys, and others [8]. They have created frameworks in the concept of blockchain technology providing innovative solutions.

One of these frameworks is Hyperledger Fabric (HLF) [9], a highly versatile modular infrastructure to build blockchain solutions for enterprise use cases. Providing tools to implement a permissioned network, Fabric allows for pluggable consensus mechanisms depending on the needs of the project. Its underlying architecture gives the ability to a business to customize multiple aspects of the network in terms of membership authorization and access control management while achieving great performance in terms of TPS.

In order to have a smooth transition of this chapter to the architecture of our project, we will briefly discuss some key concepts and terminology regarding Hyperledger Fabric.

Organization

Every entity inside the HLF network is an organization and every member making transactions must belong to one. Each organization can set up roles for their members, such as admin or peer, not only to split the tasks accordingly but also to enforce access control policies to Fabric's resources. Technically, an organization is represented by a folder with crypto material called Membership Service Provider (MSP). The MSP is the means by which validation is performed when organizations or members transact. Lastly, organizations form consortiums, set consensus policies, and decide on the network configuration.

Certificate Authority (CA)

A Certificate Authority is an entity issuing certificates to authenticate organizations or users so they can perform actions inside the blockchain network. In essence, CAs sign Certificate Signing Requests (CSR) submitted by members who have been first. registered by an admin. Also, most likely an organization has a dedicated CA which identify their own members. Other than that, they are responsible for checking if a certificate has expired, storing roles of members, etc.

Channel

A channel is a sub-network inside the system network which allows for private communication between organizations. Additionally, organizations can specify their own rules for how the channel will be operated. Every channel may be created for a dedicated purpose and run specific smart contracts, depending on the use case's needs of setting access control policies both on a channel but also at a chaincode level. Also, each channel maintains a separate ledger.

Ledger

A ledger is a journal recording the history of transactions. The HLF ledger consists of the blockchain, which is an immutable transaction log, and a database called World State.

World State

The World State is a database keeping the current (latest) values for all the keys on the blockchain. The World State can be directly determined by reviewing the ledger and getting the latest state. The need for this database is for effective chaincode operations in terms of performance.

Chaincode

A chaincode is a package containing multiple smart contracts. Access control policies can be enforced inside the chaincode to ensure the eligibility of an entity performing read and write operations on the ledger.

Peer

Peers are nodes responsible for storing ledgers, running chaincodes, and endorsing transactions. When a transaction is submitted to the network, it is firstly handled by peers who execute the transaction proposal and endorse it if it produces a valid outcome. The endorsements must satisfy the channel policy for the transaction to be considered valid and subsequently get included in a block. After the endorsements are collected, the proposal is passed to the orderers.

Orderer

Orderers are nodes who pack transactions in order and include them into blocks after validating that they satisfy the channel policies. In essence, the process of executing and validating transactions is split in Fabric with the purpose of reducing work done by individual nodes and thus achieving greater performance and TPS. When blocks are created, orderers disseminate them to peers who in turn append them to the chain.

2.5 Related Work

This thesis aims to be an innovative approach to safe and ethical healthcare and educational data exchange. Current architectures are flawed in multiple aspects such as security of data, ownership, control and trust, which result in a lot of aftereffects. Some of these include:

1. Lack of connection of user-generated and healthcare professional data

2. Lack of secure data aggregation

3. Lack of an independent and neutral institution

4. No secure compliant way to exchange data with end-users
5. Lack of user control sharing consent
6. Little support for real-time data

We will briefly explore some case studies related to our architecture.

Case Study: Healthbank

HealthBank is a cooperative based in Switzerland. In a cooperative, the business is owned by its members, the people who invest in the platform, and those that use it to store their health data. The primary benefit for members is that they are empowered to take control of their health data. It is built on a neutral, independent, and safe data infrastructure.

The platform has implemented a Business-to-Customer (B2C) and Business-to-Business (B2B) scenario which facilitates data exchange between parties. Users can collect data in any format and give consent for access to others.

In particular, they are incentivized to sell their data to third parties securely using personalized policies. Any data sharing can be ceased upon demand of the owner. Additionally, users can see the details of any transaction and decide to approve or reject. On top of this, they have the option to share anonymized data protecting their identity.

The major issue of this architecture is that all the user data are uploaded on central databases. This poses a threat to a user's privacy since centralized servers are prone to attacks. Also, users rely on the platform to store their data and have no other option than to trust the system. Lastly, there is only support for batch data.

Case Study: Health Wizz

Health Wizz is an innovative technology company with the mission to enable people to securely and efficiently manage their health information. It is a mobile application platform that allows people to aggregate, organize and share their health records from multiple sources including wearables, Electronic Health Records (EHR) Systems, and their genome.

Health Wizz is a decentralized application and all data is securely collected and stored on the blockchain. Users have complete control of their data and decide who it is shared with. By using the free Health Wizz app, users can collect, aggregate, and store health data from devices, hospitals, clinical records, and more on a "Digital File Cabinet" which allows users to assume complete possession of their medical records. Users can control, review, and track their health status, moving this information with them wherever they go.

On the other hand, Data Seekers, Research Organisations, Pharmaceuticals Companies, etc., get identities on blockchain and publish contracts soliciting health records matching criteria for their research. Individual users submit anonymized records matching the contract. Data Seekers can provide monetary incentives for the data (e.g. for non-anonymized records), which can be fueled with rewards points. OmPoints is a digital currency, a reward-point system for incentivizing data exchange contracts between users ("data providers") and "Data Seekers." Users can also create gaming contracts with OmPoints as rewards or prizes offered by a promotor or by other participants. Participants submit health records which are shared, and rewards are distributed by the contractor or validator.

Case Study: Datapace

Datapace [10] is a marketplace for IoT sensor data. Built in Hyperledger, it utilizes a permissioned network where users are verified to ensure the credibility of data sources. The data is collected and shared as an input stream while the transactions are handled by the smart contract logic. The platform does not store any data, but only the hash of the data so that it can be verified from the buyer. Additionally, a token is used to simplify the data exchanges and payments.

Case Study: MyHealthMyData

The My-Health-My-Data project [11] was established to interconnect individuals, hospitals, research centres, pharmaceutical companies, universities, among others, within a biomedical information network that allows them to control access to healthcare data for research purposes in line with the GDPR.

It relies on a distributed secured blockchain network based on Hyperledger Fabric, instead of a central authority, in order to appropriately manage the data-sharing pipeline and regulate data access on the basis of user-defined permission/consent settings. It uses smart contracts to enforce different complex business rules according to the data category, security level, purpose of use, network members and GDPR requirements. It includes the My-Health-My-Data mobile application which acts as a dynamic consent interface and enables data owners to allow, refuse and withdraw access to their data according to different types of potential usage.

In the next chapter, we discuss the architectural design elements and requirements of our project.

Chapter 3

System Design

3.1 Proposal analysis

The system consists mainly of three components: network, application, and smart contracts. The network acts as a base layer to deploy the marketplace, while the application holds the back-end logic of the software. Essentially, the application handles user requests and database management and acts as middleware for the user-blockchain interaction.

The whole architecture and the application connection are seen on the following schema.

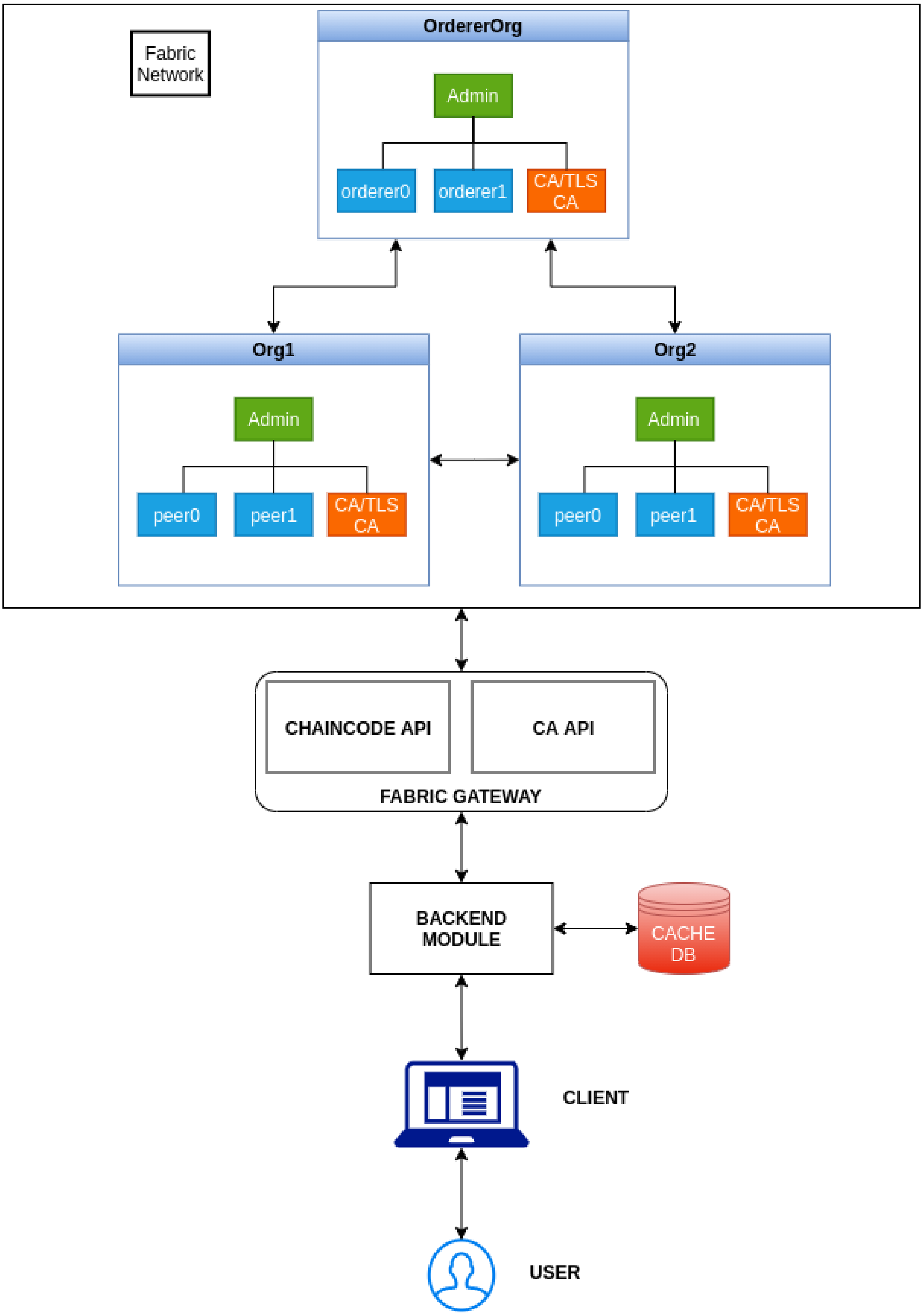

Figure 3.1: Architectural Overview

In order to maintain security and limit exposure, the nodes are exposed only to the rest of the network and the application. As observed in the figure, there is no direct connection between the user and the blockchain network. Consecutively, all of the requests are passed to the back-end and forwarded to the relevant nodes.

The first obstacle to be encountered with this approach is often the handling of the transactions with the intention of enforcing decentralization. In the standard implementation, the Fabric SDK both acts as a client and signs transactions on behalf of the user. This approach assumes that crypto material is centralized, or that the application has access to a user's secret key. Therefore, since transactions are directed from the application, the user must have already signed the transactions with their secret key and only use the application as a client.

To solve this problem, we have designed a solution where the user, as the only holder of their private key, signs a transaction manually and forwards the transaction to the application. This way the network maintains trust, security, and privacy, while the user can safely assess that they and only they can alter their protected data.

We examine both the network and the application in the following sections.

3.2 Network Architecture

Overview

The distributed network, built on Hyperledger Fabric, aims to be the pillar of the whole marketplace infrastructure. The pilot deployment consists of 2 peer organizations, namely Org1 and Org2, that represent the initial consortium. Also, an additional organization which we name OrdererOrg has the role of the Ordering Service in the network. OrdererOrg is not a distinct organization but is represented as such for the sake of clarity and role separation. The organizations are joined to a channel in which a chaincode is deployed that handles the user and data logic. The overall architecture is also seen in the schema below:

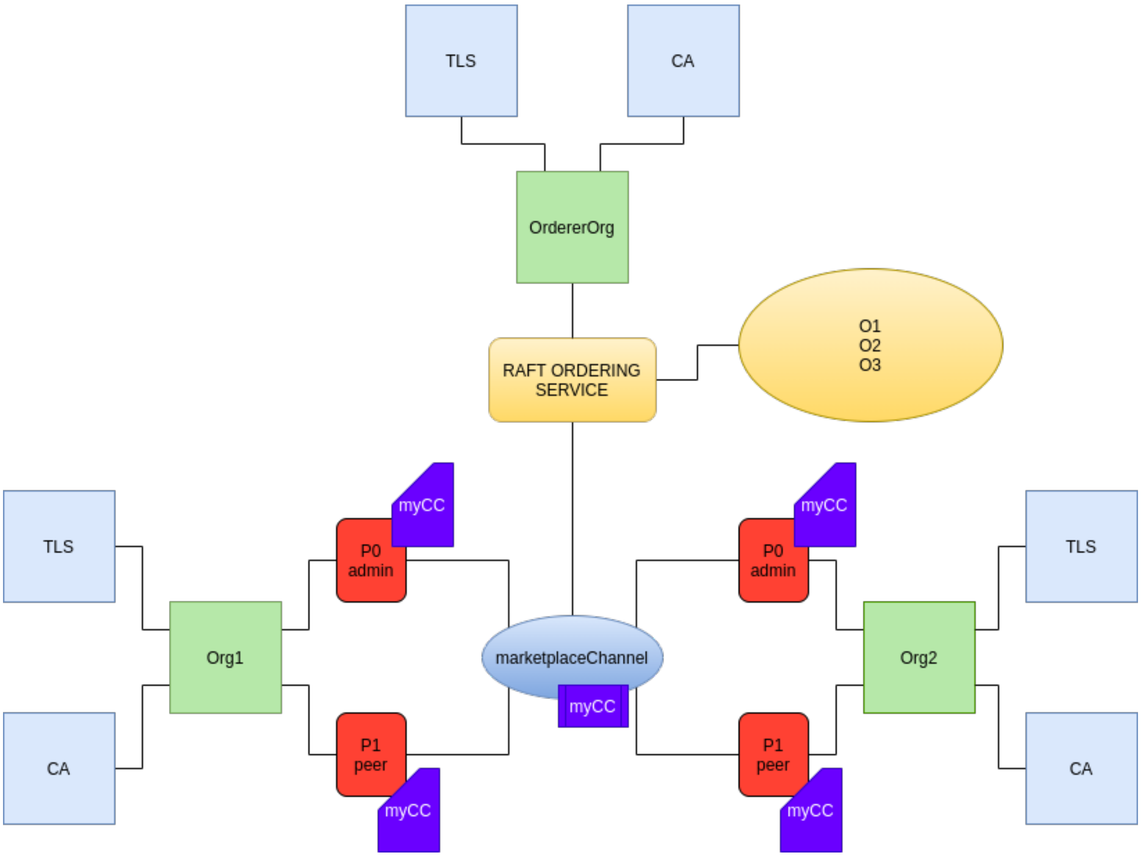

Figure 3.2: The Basic Consortium on Hyperledger Fabric Network

Components

Each organization has the following components:

- Identity CA
- TLS CA
- Organization's MSP
- Peers/Orderers

The **identity CA** is used as the root certificate authority which issues enrollment certificates. These certificates form the identities of all the nodes and users in the network and guarantee that they are approved by a consortium member. Moreover, each certificate holds information regarding the role of identities. These different types of roles are called Node OUs (Organizational Units) and currently, the following four roles are accepted on HLF: admin, peer, orderer, client. Lastly, using this role distinction design we are able to create special access control policies and separate the network's actors.

The **TLS CA** is mandatory to enforce security on prone to attacks distributed network. This CA is always deployed first and provides the certificate for all the communications between the nodes in the network. All nodes are deployed with two-way (server and client) authentication, adding an extra layer of security.

Next, we have the most important component of an organization, the **organization's MSP**, which forms its identity and allows for participation in the channel. Under each MSP, all nodes and user transactions are verified against the Ordering Service where it checks two conditions:

1. The transactor is a member of an organization joined in the channel.

2. The transactor is eligible to perform the intended action based on the channel's policies.

Both the TLS and Identity root certificates are stored in the channel configuration, so any member of the channel can message and authenticate messages and actions of other members.

Each peer organization hosts 2 peers in the initial deployment. While an organization can be a member of the channel, it is not necessary to host peers. It is important for organizations that endorse transactions based on the channel policy to have peers joined to the channel. For that reason, each of the initial organizations deploys 2 peers for the role of endorsing transactions. In the following sections, we will present the channel policies set for the most relevant resources.

Lastly, the Orderer organization, which deploys the ordering service, hosts 3 orderers. Being responsible for the block dissemination to peers, if the ordering service does not form a quorum, the network stops producing blocks and thus transactions cannot be committed. In our case, 3 orderers provide fault tolerance of 1 node.

Deployment

Each of the components mentioned above represents a different physical or virtual node. In our case, each node is deployed as a docker container inside a Virtual Machine (VM) hosted in cloud services. The TLS CA is only used for the enrollment of nodes, so after the initial enrollment, it will be down as long as another node will not be joined to the network. Thus, we have used one VM for both of the CAs for better resource management. The individual network nodes can be viewed in the figure below.

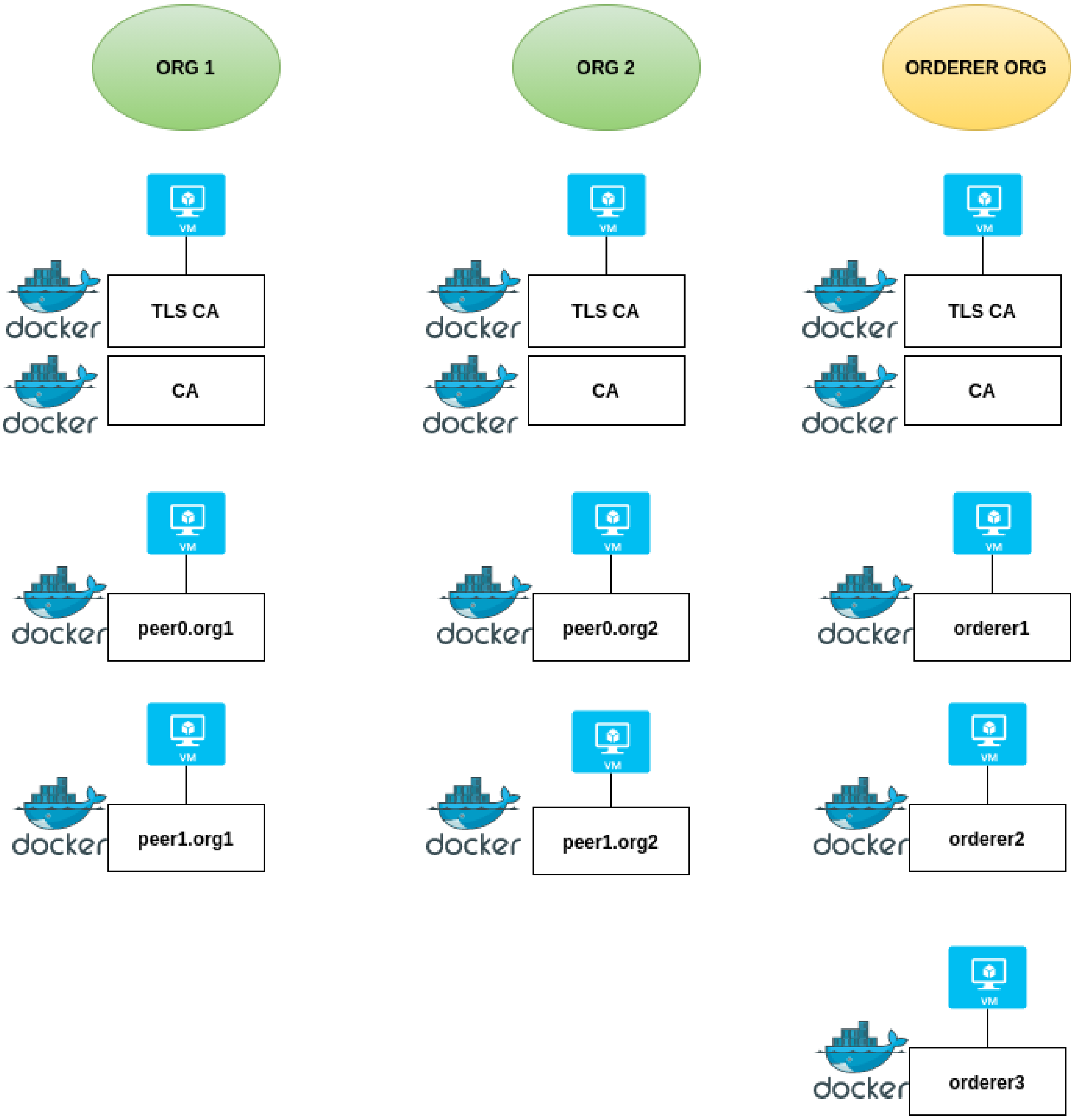

Figure 3.3: Network nodes represented as docker containers

Channel

Upon deployment, the organizations are joined on a channel as seen in figure 3.2. The Hyperledger Fabric network is based on a governance infrastructure in which the members have the power to decide and establish policies imposing access control rules at an organizational level but also at an identity level. As the last step, the organizations approve the chaincode definition and commit it to the channel, making it possible to transact on the network.

In the following section, we will look into how the application connects to the network nodes and perform user registration and smart contract operations.

3.3 Application Architecture

The application is the bridge between the user and the data marketplace. Decisions regarding the architectural design are taken based on user privacy and security, high performance and throughput, and enforcement of decentralization while sustaining a design that is manageable in terms of maintenance and continuous integration.

How do we fulfill these requirements effectively? The answer to this question resides in the implementation of the following application elements:

1. Cache database

2. User registration

3. Transaction flow

4. Data exchange

5. Smart contracts

We will discuss each of these components for the rest of this section.

Cache Database

The greatest bottleneck of a blockchain network is scalability. The extreme effort of broadcast and validation of a block by the whole network results in a low transactions per second meter which is one of the most important indicators of a blockchain.

The architecture of HLF partially solves this problem to a great extent with the Ordering Service handling block validation and dissemination. Also, peer databases are integrated so that better time complexities can be achieved when querying data on the ledger. However, it is still not performance-wise to query big amounts of data and generally performing database operations directly on the peer's database.

This creates a need for a back-end Cache DB to replicate the ledger's data and also store additional metadata used by the application. For this reason, we have developed a MongoDB which is automatically updated by listening to blockchain events. All of the operations which do not require using smart contracts utilize the Cache DB, thus reducing the peer's effort substantially while achieving faster results.

User registration

One of the core features of this project is the realization of a system where the user is the only holder of their key and their data. We want to protect our privacy and our sensitive data as individuals, so we tend to trust systems that take this approach as their main goal.

Since it all begins when the user registers and creates an account on the platform, we must make sure that the application has no access to the auto-generated keys and certificates. We have carefully designed and implemented this feature.

The goal is achieved by generating keys on the client side and creating a Certificate Signing Request (CSR) that is forwarded to the organization's Enrollment CA. The user's identity is constructed on the client side after successfully fetching the CA's response. We will thoroughly examine this operation in Chapter 4.

Transaction flow (Blockchain)

It is already stated that the user demands guaranteed security and privacy of their secret credentials. On most public blockchains, the standard procedure is to sign a transaction and broadcast to the peers/validators of the network. However, this comes with a heavy drawback which is the exposure of the nodes to the whole internet. While a public blockchain would not suffer from an attack on individual nodes, only the nodes themselves, an attack on a Fabric node would be disastrous.

A worst-case scenario would be an attack on a root CA. If the root CA is compromised, the entire trust domain would collapse. Healing the network would require the creation of a new CA and the subsequent registration/enrollment of all the nodes and users. The implications are obviously catastrophic so our approach is to limit the network exposure by making users able to transact via the application acting as a middleman.

We achieve this by allowing the user to manually sign a transaction on the client side. Consecutively, the signed transaction is forwarded to the blockchain nodes where it gets validated and committed. The technicalities of this matter are further explained in Chapter 4.

Transaction flow (Data)

A standard marketplace operates as a medium of exchange for digital or physical data between two parties. The prime attribute of our platform is the existence of a penetration-resistant layer of security actualized by smart contracts. Committed on a blockchain ledger as immutable code, their main function is validating the legality of data exchanges. On the basis of this architecture, the transaction flow of the data is split into four distinct steps.

Firstly, the buyer selects a product willing to buy. At this step, they are required to state the intention of use which if fed as input to the smart contract. The contract checks the intention of use and matches it against the policy of the product specified by the seller, producing the boolean outcome regarding the eligibility of this transaction. Since a rejection from the contract stops the transaction flow, we assume that the buyer is eligible to proceed in which case the transaction details are stored with the status stating that the buyer is eligible at the current state.

Secondly, the buyer proceeds to the payment process. Payments are accepted on the Ethereum platform, so the buyer is asked to send ETH to the seller's address. If this is done successfully, the transaction status on the fabric blockchain is updated to state that the buyer has paid for the product and awaits the key to access it.

Thirdly, the key is given to the buyer, and upon verification of proof of delivery, the transaction status is updated to state that the buyer has currently access to the product.

Lastly, when the time of access has expired the transaction status is updated by the application to state the end of the transaction flow.

Data exchange

As a blockchain platform in compliance with GDPR, we intend to hold only parameters that define the policies set by the seller for their products. Therefore, the storage and exchange of the products after the successful validation from the smart contract logic, which will be discussed in the next section, it is up to the seller-buyer communication to establish the way of their out-of-band data exchange.

In our implementation the user stores data in a cloud environment of preference, which is encrypted with a secret key. Upon successful payment, the key is transferred to the buyer giving them the ability to download and decrypt the data to use as defined in the agreement.

On the expiration of the date specified by the policy regarding the eligibility of access to the data, the buyer is informed by the platform about the transaction status and is asked to delete it. Any use of the product from the buying party after the expiration date is deemed illegal and they will be held accountable under GDPR laws and citizen privacy protocols.

The platform bears no responsibility for the actions taken by the participants in terms of legality.

Smart Contracts

The decentralized code governing authorization between the users lies in the smart contract logic. Packed in a chaincode, it is split into 3 contracts:

- User Credentials
- Data Catalogue
- Agreements

Essentially, each one holds the operations for the respective actions taken inside the marketplace. The User Credentials and Data Catalogue contracts handle CRUD operations for the respective categories, while the Agreements contract enforces policy matching and validation in addition to storing transactions and tracking their state during the product's transaction flow.

Having described all of the foundation elements that synthesize this project, we present the requirements of this project.

3.4 Requirements

Building a distributed network requires connecting multiple nodes together to achieve the desired results. Security and durability of blockchain nodes are essential to any blockchain network and must be carefully designed to avoid issues in the future.

Hardware

In terms of hardware in the system, as it can be observed in figure 3.3 we need individual machines for every node except CA/TLS for the pilot network. Also, each machine's resources must reflect and execute the tasks assigned easily and effortlessly for long-time periods. To be more specific, the standard requirements for our project are summarized below:

- Peers must execute chaincode invocations/queries with ease.
- Peers must hold enough storage for the increasing size of the blockchain and world state.
- CAs do not need heavy CPU due to a small number of requests, but enough storage for the database.
- Orderers need less CPU than peers as they handle block ordering and store the ledger and same storage.

In addition to the above, Hyperledger Fabric experts provide a rule of thumb to the approximate resources of each node [12]. It is stated that orderers should have about 1/3 of peer's resources and CAs should have 1/10 of peer's resources.

All in all, we concluded the following hardware specifications.

Software

Software-level requirements revolve around the operation of nodes and the application. We present the specifications for all nodes below:

- Fabric Binaries: 2.2
- Fabric CA: 1.4.8
- OS: Ubuntu 20.04
- Docker: 20.10.1
- Docker-compose: 1.27.0
- GO: 1.15.2

For the application:

- Npm: 7.6.3
- MongoDB: 3.6.8

With this, the chapter on application design elements is closed and we will discuss implementations and testing procedures in the next chapter.

Chapter 4

Implementation

In this chapter, we present the development process that we followed along the lines of this project. Additionally, we provide the implementation of all the distinct modules explaining the course of action, tools used, and testing routines.

4.1 Development Process

The development process of this project is framed on the Agile philosophy providing simplicity and great results as a standard process [13]. Agile is proven to be one of the greatest choices for blockchain projects [14] as its iterative and incremental nature allows for a gradual build-up of prioritized functionalities.

Building a blockchain project requires the integration of many distributed components creating the need for a well-structured unit build and test approach. Additionally, individual components must be continuously integrated into the overall architecture and ensure there are no inconsistencies in the various data exchanged on the network.

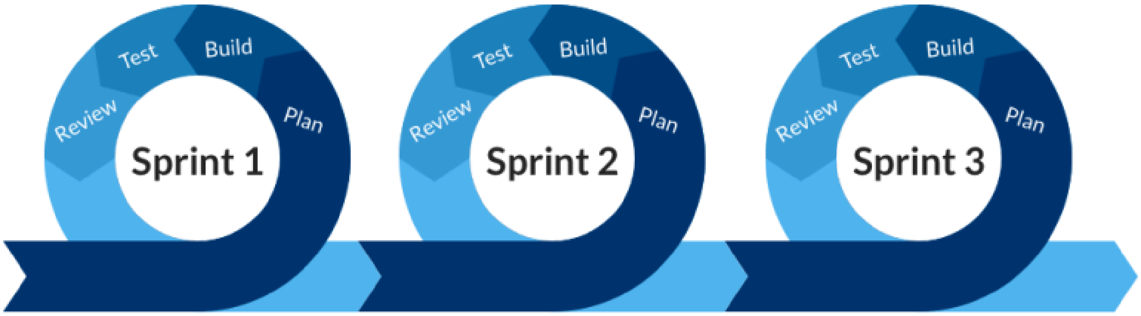

Figure 4.1: Agile Development Process

Starting with the overall concept in mind, we divided the tasks or user stories into individual sprints of a 3-week time frame. The sprints can be seen in the table below as well as the tasks implemented on each sprint.

Sprint	General task
1	Create and bring up a basic HLF network on test environment.
2	Setup a channel and deploy a test chaincode. Test channel and chaincode policies.
3	Code User/Data smart contracts and implement CRUD operations.
4	Store legal parameters inside the ledger via contracts. Implement policy matching algorithm for parameter validation.
5	Implement backend API and create application - network operations to smart contracts.
6	Create Cache Database as an offchain solution which replicates the ledger by listening to events. Create API calls to the DB on the backend.
7	Automate network deployment with scripts creating organizations, nodes, and scripts to set up connections.
8	Deploy a pilot network on individual VMs and test user-blockchain interaction.

Table 4.1: Sprint planning

4.2 Blockchain Network

The development of the HLF network as a whole is a process that requires several modules to be built. Firstly, nodes should be implemented to go online and perform many tasks easily in order to be used by a network operator with the help of well-structured documentation. Secondly, the chaincode handling the logic of user and data-related operations must be built and deployed in the channel.

Nodes

The process of bringing up an organization is automated using scripts. The scripts handle the registration and enrollment of nodes to their respective CA, as well as creating docker files with node-specific variables for automatic deployment.

Additionally, we have created APIs implementing peer and CA functionalities so an organization admin or network operator can use them seamlessly to connect and perform tasks on the nodes.

It is important to keep the file structure consistent and in a specific format. For this reason, we provide tools that handle file and system operations, such as setting up all the software requirements on the VM, importing/exporting an organization's crypto material, or integrating hostnames and IPs of other nodes to enable communication.

Some general notes on the node implementation are:

- All messages between nodes are authenticated via TLS protocol.
- The Ordering Service is based on a RAFT Protocol.
- Peers use CouchDB as World State.

Smart Contracts

The fundamental objects of our use case that SCs handle are users and products. The core operation in general terms needed to be secure, transparent, and immutable is the eligibility of data exchange by the participants. We present our data model in a UML diagram so objects and properties can be viewed in a more clear manner.

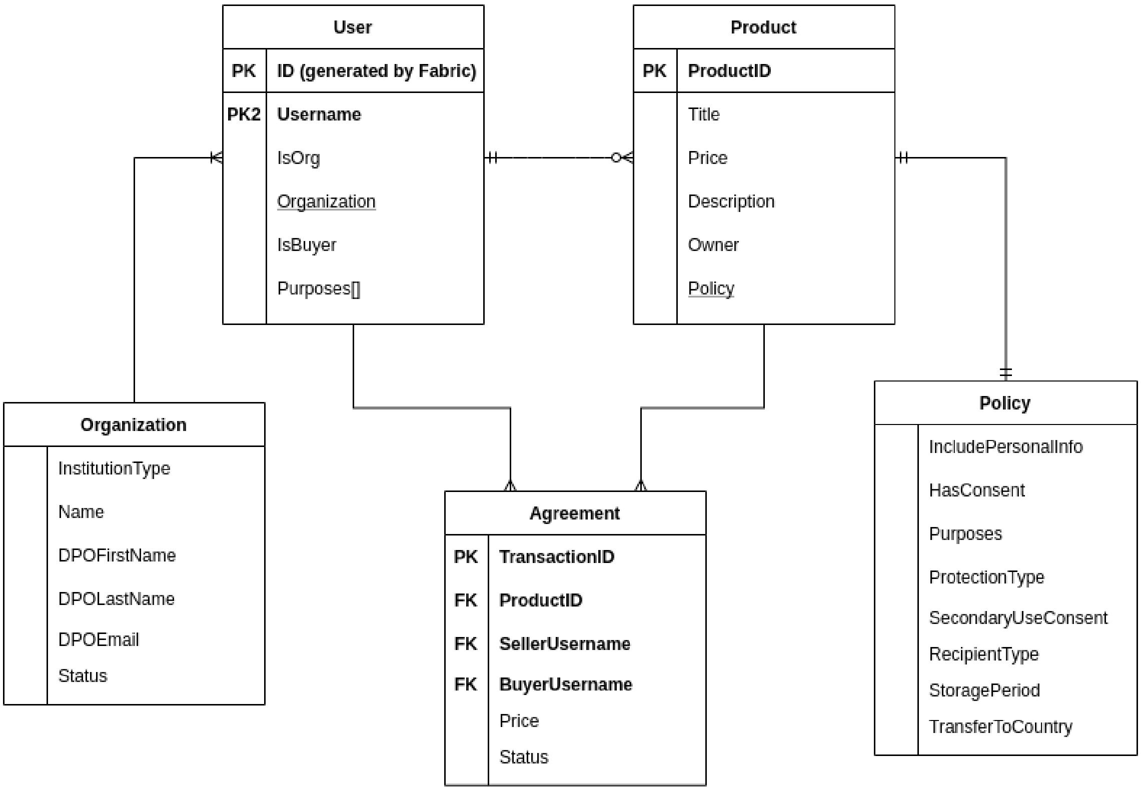

Figure 4.2: Data Model in World State

We should note here that most of the attributes regarding users and products are used to enforce compliance with GDPR and will not be discussed in this document.

Our chaincode implementation is a package hosting 3 Smart Contracts. The SCs interact with the ledger with mainly 3 operations: Insert, Read, Delete which are named as PutState, GetState, DeleteState respectively. State, in the abstract, is key-value pair stored in the ledger. Furthermore, we can group together multiple states to create a logical state list by creating composite keys. For example, two different user entries would be stored on the ledger as:

- Key: “user:Bob”, Value: “data1”
- Key: “user:Alice”, Value: “data2”

Querying the key: "user" would return both users Bob and Alice and their data. This way we can form namespaces and interact with objects of the same type. We define unique namespaces for states and their respective keys as:

State List	Namespace/Type	Key	Value
Users	user	$username	User (Object)
Products	product	hash($username + $salt)	Product (Object)
UserIDs	userID	$userID	$username
User Inventories	inventory	$username	UserInv (Object)
Agreements	agreement	$transactionID	Agreement (Object)

Table 4.2: Ledger Key-Value Arrangement

The User and Product States are self-explanatory. The UserID State is a mapping between an ID and a Username. The User Inventory State stores for each user their product IDs, the number of products, and a salt number for the productID generation. The Agreements State stores the lifecycle of a product purchase in the 3 phases of policy matching, payment, and access. Having defined the States, keys, and values we can define the logical flowcharts of each SC operation.

The User Credentials SC provides CRUD functionality for users (See Appendix A). The key note to consider with respect to the user credentials is the purposes of buying parameter which the user gives as input. Examples for these purposes include Marketing, Publicly Funded Research, Business, and others. These values are used from the application to filter the catalogue that a buyer browses.

The Data Catalogue SC provides CRUD functionality for the products of the users (See Appendix B). In addition to the basic operations, this contract also invokes the Agreements contract when the user requests to buy a product. Each product is associated with a strictly selected policy which in turn defines its lifecycle. The seller inserts the purposes that allow for use by another user. Although a user as a buyer has stated reasons for buying at their credentials, it is mandatory to state the reason for buying at each individual product they wish to buy, which is then handled by the Agreements contract.

The Agreements SC is invoked on the buy product operation to decide if a user is eligible to access a specific product by matching the product's policy with the user's reason for buying (See Appendix C). Specifically, the buyer's given use case is legally bound and recorded permanently on the ledger. If the agreement results eligible, a transaction called agreement is stored on the ledger stating that the buyer has the authority to buy this product. Lastly, functionality is provided that allows for updating this transaction status via the application in the subsequent steps that follow the data lifecycle as was discussed previously in Chapter 3.

Having examined the smart contracts, we proceed to discuss the deployment flow.

Deployment

Deploying nodes and exposing the network is a distributed effort that includes mandatory file exchange and therefore out-of-band communication. Files to be transferred include chaincode packages and TLS crypto material, among other important data. We present the overall process of deploying the network which includes all steps up to deploying the chaincode.

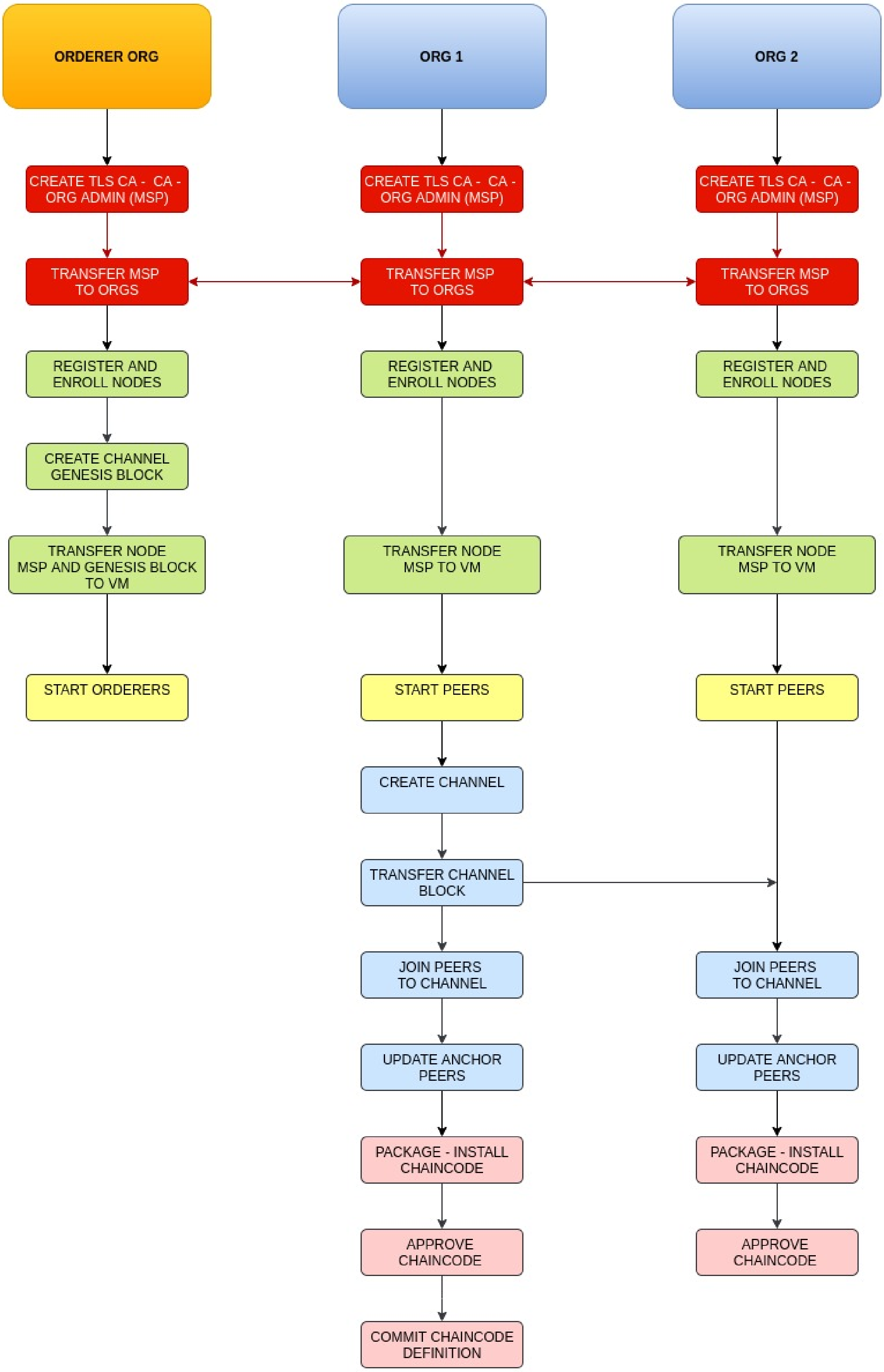
ORDERER ORG
ORG 1
ORG 2
CREATE TLS CA - CA - ORG ADMIN (MSP)
CREATE TLS CA - CA - ORG ADMIN (MSP)
CREATE TLS CA - CA - ORG ADMIN (MSP)
TRANSFER MSP TO ORGS
TRANSFER MSP TO ORGS
TRANSFER MSP TO ORGS
REGISTER AND ENROLL NODES
REGISTER AND ENROLL NODES
REGISTER AND ENROLL NODES
CREATE CHANNEL GENESIS BLOCK
TRANSFER NODE MSP AND GENESIS BLOCK TO VM
TRANSFER NODE MSP TO VM
TRANSFER NODE MSP TO VM
START ORDERERS
START PEERS
START PEERS
CREATE CHANNEL
TRANSFER CHANNEL BLOCK
JOIN PEERS TO CHANNEL
JOIN PEERS TO CHANNEL
UPDATE ANCHOR PEERS
UPDATE ANCHOR PEERS
PACKAGE - INSTALL CHAINCODE
PACKAGE - INSTALL CHAINCODE
APPROVE CHAINCODE
APPROVE CHAINCODE
COMMIT CHAINCODE DEFINITION

Figure 4.3: Deployment Process

Distinct phases on the deployment flow are represented by different colors.

Also, actions taken by peer organizations are different from the orderers as seen on the diagram.

Upon committing the chaincode definition on the channel, the next step is establishing the connection with the application, which is discussed in the next section.

4.3 Application

The backend application is written in Node.js and is built mainly using HLF's Node SDK which utilizes functionalities to provide connection and interaction with the blockchain network. However, in order to achieve some of the goals we had to reform some of these processes and adapt them to our needs. Essentially the application is the middleware between the network and the user and must be optimized accordingly to achieve high throughput, security, and privacy. In this section, we will examine in depth the implementation of the design elements that were mentioned in the previous chapter.

Cache Database

The Cache Database is an off-chain solution to provide faster performance as well as greater flexibility on query handling. Since the platform can be used for data analytics, it is important to build a viable solution that allows for seamless additions in the future, further enhancing the platform's functionality and value.

First of all, the cache is implemented as a MongoDB which replicates the data on the ledger via block event listening. As most of the blockchain platforms with smart contracts, HLF allows setting events inside the contracts to notify applications of updates on the ledger. Thus, a block listener updates the database with the data associated with the event and is also used to provide updates on the client side of the application.

Additionally, our cache implementation allows for the reconstruction of the database from block 0 instantly without facing issues such as race conditions or data mismatches. Due to the asynchronous nature of such functionalities, it is essential to keep the data synchronized and inserted in the DB in the correct order. In order to accomplish this, we have created a mutual exclusion (mutex) implemented with queues on every user key which locks every time there is an ongoing operation on the DB. The key is unlocked once the DB finishes the operation. The diagram below illustrates the queues in a more clear manner.

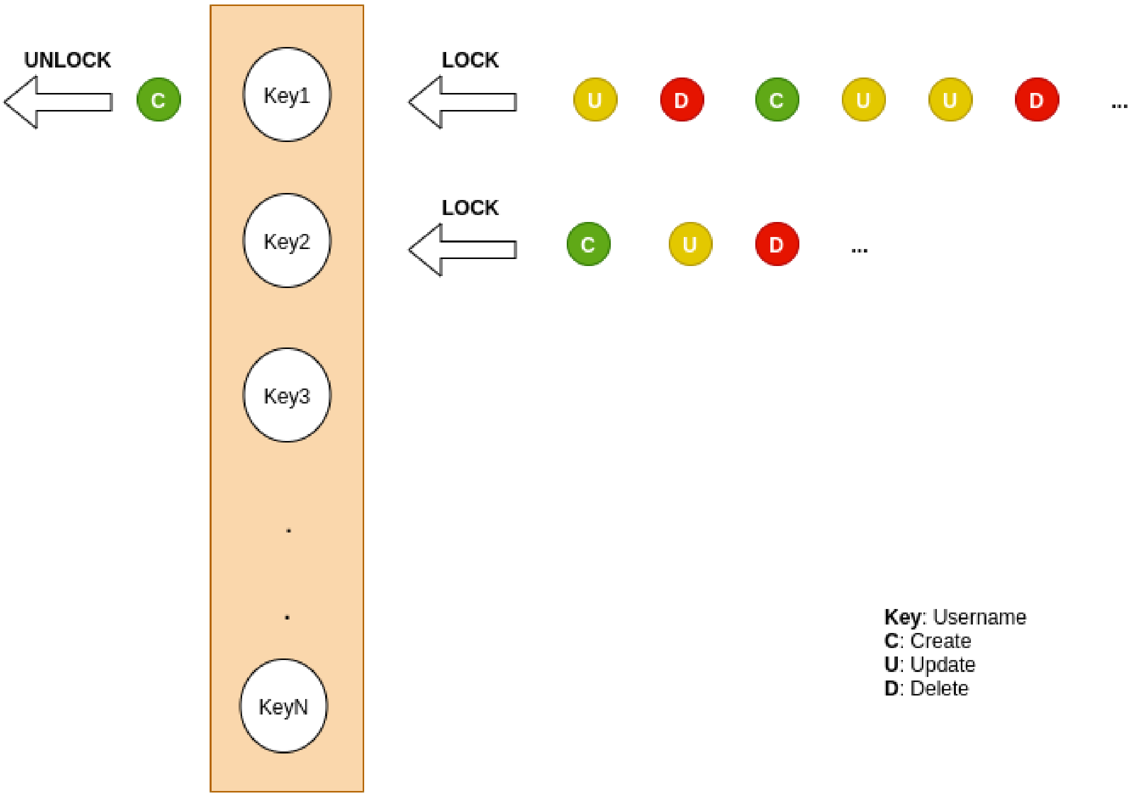

Figure 4.4: Database key locking

The last thing to note is that the DB is only updated via the block listener and exposes an API for querying operations.

User registration and enrollment flow

During the registration and enrollment process, a user must be guaranteed they are the only party that has had access to their secret key. By default, the API exposed by Fabric uses a wrapper for this process in which a client connects directly to the network, keys are generated locally, a CSR is created and proceeded to the CA where it is validated. Since the architecture of this platform prohibits users to connect directly to the network, we have split this process to ensure that the application does not have access to the user's secret key.

To achieve this requirement, we have manually implemented the key generation and the CSR on the client side from where it is forwarded to the backend and then to the relevant CA. The registration process is made clear on the sequence diagram 4.5. Note that in Fabric, registration and enrollment are two different processes but we are referring to the overall implementation of a user being a member of the network by getting a certificate.

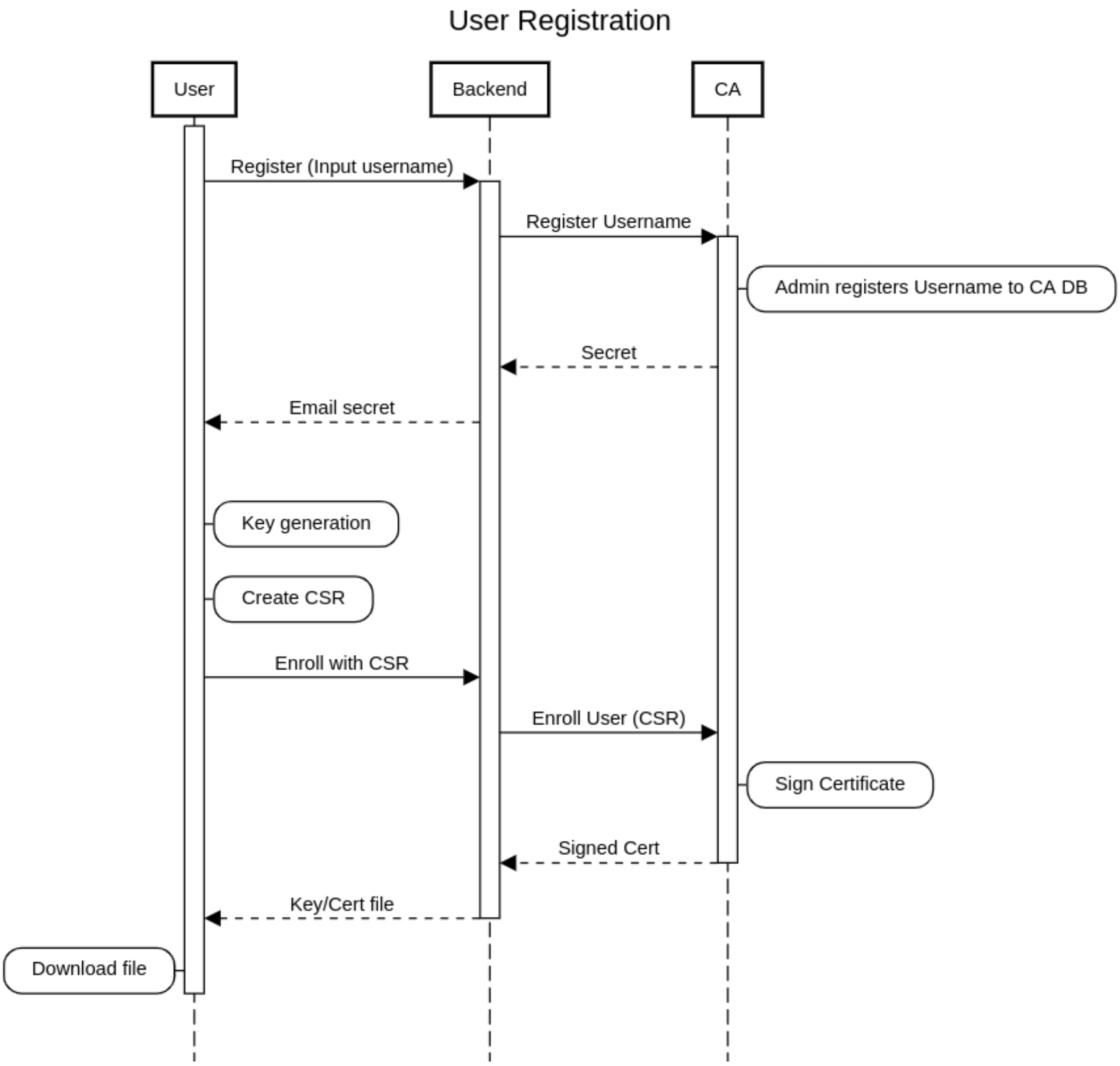

Figure 4.5: User registration flow

User transaction flow

In the same manner, as the registration and enrollment process, the SDK wrapper reads the signer's key imported from the file system and uses it to sign transactions and broadcast them to the network. Since the user communicates via the application, transactions must be signed locally on the client side and handled by the application, consequently preserving the privacy of their crypto material.

Essentially, the tasks of Fabric's transaction flow must be split. The sequence of the transaction lifecycle is:

1. The user imports the secret key on the browser where they sign the initial proposal which contains the transaction willing to make, such as creating a new product.

2. The transaction is forwarded to the peers, which validate and endorse the proposal.

3. The endorsed proposal is returned to the user who signs a commit proposal.

4. The commit proposal is forwarded to the orderers who in turn validate the policies and append the transaction to a block.

5. The block is disseminated to peers, added to the ledger, and the application handles the response.

The offline-signing transaction flow is also illustrated on the sequence diagram 4.6 below.

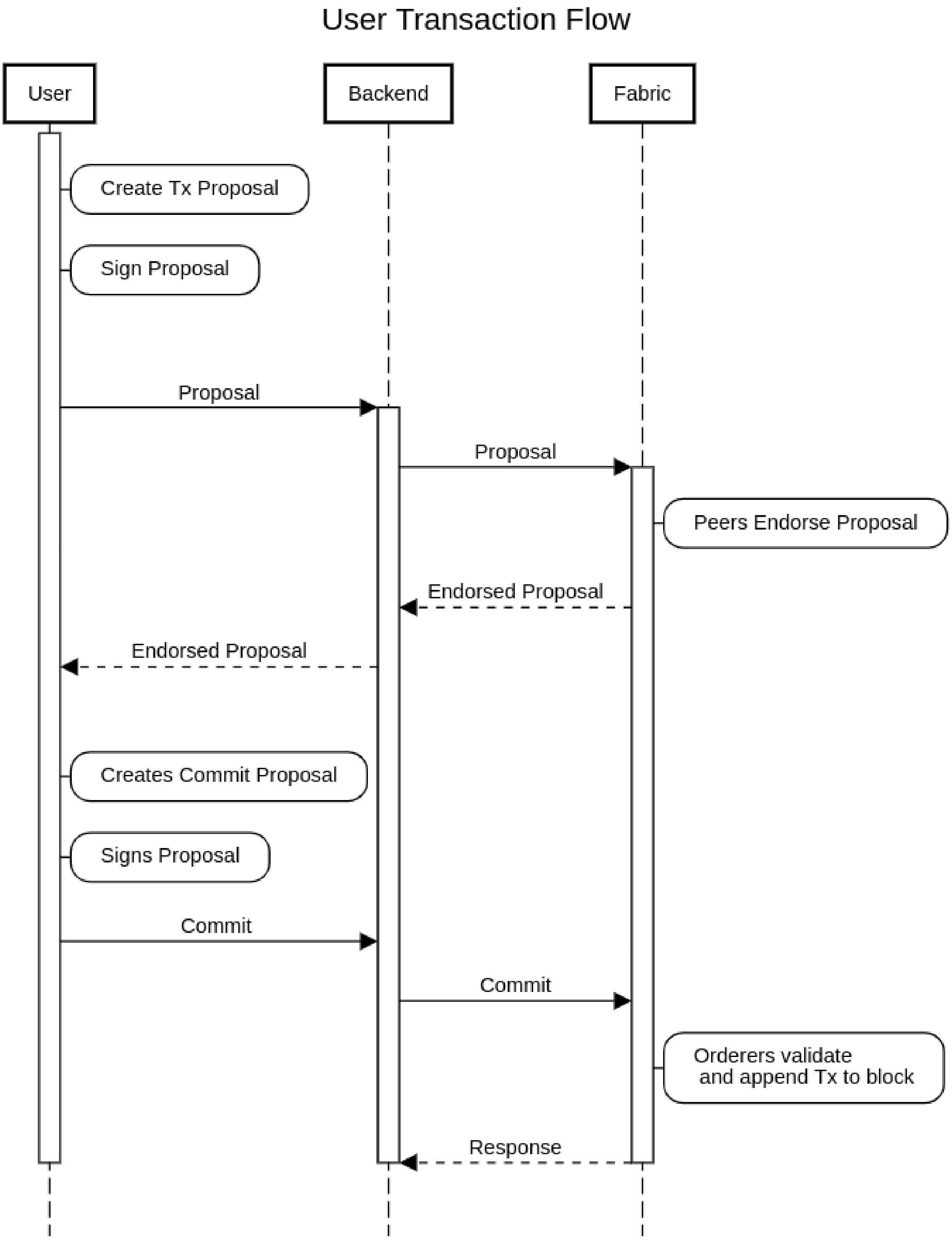

Figure 4.6: User offline-signing transaction flow

Filtering Algorithm

A user has the ability to browse the data marketplace and select products to buy. However, since buyers have predefined purposes of buying data products, it is essential to present only the products that the user may be eligible to buy. For this reason, we have implemented a filtering algorithm.

Upon selection to browse the product catalogue, the application fetches the user's credentials from the database. The purposes property from the user credentials is checked against the products' purposes and filters all the products whose purposes do not match the user's preferences. Thus, the data marketplace is more relevant to the buyer.

Monitor

It is of great importance to set up an infrastructure that monitors nodes in a distributed system with ease. Additionally, the application must provide simple and well-defined logging controls to make the process of error handling simpler and faster.

In terms of the network nodes, logs are monitored via docker containers. Any fatal error that occurs is discovered by the application and directs us to the relevant nodes where the error occurred.

Regarding the application, logging control is applied on the code via the Node.js winston package. We have split logging levels to differentiate between the importance of the message. Specifically, there is an option to set the level at which the information should be logged to files or console. Debugging a project with multiple files and structures is a tedious process, so it is important to create logs that direct us exactly to the line of error. Therefore, the format of logs we chose is: colored(Time LEVEL File) – function – message, where color depends on the level.

As for the database, we use a script with continuously compares all the data on the ledger and the database to make sure it is the same. A mismatch on the database is a fatal error and must be corrected immediately.

4.4 Testing

Developing and deploying a blockchain network requires excessive testing and monitoring. Both security and connection of nodes, as well as establishing a channel and committing chaincodes to it, are issues that generate multiple bugs. Since each bug is most of the time fatal, we have a step-by-step testing procedure of the overall lifecycle. Also, during all the testing and debugging procedures, there was efficient logging that simplified the process.

Network

The development of the platform was implemented in a localhost environment. All nodes communicate via docker containers on the host machine. The process of creating the CAs, nodes, docker containers, channel, and deploying the chaincode is automated using scripts. Also, the lifecycle was split into subsequent steps in order to simplify testing and debugging.

In most cases, docker containers save data in volumes which poses a problem when the nodes need to be restarted. For this reason, the standard process is deleting the whole network and starting again. This prevents existing persistent volumes to hold crypto material that later gets invalidated in the new network.

When debugging channel creation and chaincode definition commit, changing name or sequence respectively was enough. Generally, some modifications required to restart the network except the CAs which would not be needed to be deleted.

Smart Contracts

The development of smart contracts was made simple with the development environment that HLF provides. Essentially, it works as a simple network used only for testing smart contract implementation that does not require performing the chaincode lifecycle process. This tool alone is not enough to debug a contract due to the fact that every function call must be passed as a CLI command, and also JSON objects must be stringified using tools. We used the dev environment to reach the desired state which can be tested at the application level.

In this case, the core functionalities are implemented and unit tests in the application test every operation in detail. If errors occurred, we fixed them in the dev environment and repeat the process by deploying the chaincode to the channel and perform unit tests in the application.

Cache Database

The database should always be replicating the ledger. In order to ensure that the event handler worked correctly and all the data operations in the database were performed in order at all times, the database-ledger validating function was used in the whole process. Most of the issues were caused by the asynchronous nature of the code, meaning that events were continuously coming to the event handler but the database operations had different times for each of them.

This posed a problem since the events were not handled in order, but using the mutual exclusion implementation and carefully test each operation it was made possible to overcome this obstacle.

4.5 Tools & Technologies

In this section, we will describe the thought process for the choice of various tools and technologies used in this project.

Private Network

Our use case demands a permissioned network to leverage an enterprise solution. The launch of this platform is a joint effort by multiple organizations aiming to maintain and expand the core functionalities of a decentralized marketplace, such as integrating Self Sovereign Identities (SSI) and data analytics via Secure Multiparty Computation (SMPC).

At the time of this writing, only Hyperledger provides frameworks with this project's requirements. Additionally, the Hyperledger partnership lifts its products to very high standards further enhancing reliability and robustness.

Hyperledger provides two frameworks that could adjust to our platform's needs, Fabric and Sawtooth. While Sawtooth could also be our choice, Fabric achieves two times more transactions per second (TPS). Also, Sawtooth grants the ability to create both a permissioned and a permissionless network, but this feature would not apply to our use case. Lastly, after inspecting both source codes, Fabric's implementation and code structure were more appealing to the standpoint of a developer.

Cache Database

As an off-chain solution, we had multiple options but the two main ones were MongoDB and RedisJSON. Since the World State is implemented in a CouchDB (using JSON documents), it would make sense to choose between NoSQL databases similar to CouchDB.

We chose to use MongoDB due to the high scalability and flexibility in terms of querying and indexing. Also, being the most popular, there are multiple use cases and solved issues on the internet which accelerates the process of building, testing, and debugging. Lastly, even though RedisJSON is also a popular JSON solution, its size is limited by the amount of RAM in the machine which creates a bottleneck.

Smart Contracts

Regarding the choice of programming language for the contracts, at the time of this writing HLF grants the options: GO, Javascript, Typescript. Our choice is GO for these reasons:

1. GO has seen massive growth and is the choice of both HLF and Ethereum (Ethereum is written in C++ also).

2. HLF updates are always first in the GO libraries.

3. Most of the documentation, samples, and examples are written in GO.

4. GO is very strict in terms of syntax and logic and this is of great importance when writing smart contracts.

Javascript is also an attractive choice due to the simplicity and wide use, but for the reasons above GO far exceeds the requirements of a programming language for robust and reliable smart contract development.

Documentation

The last topic to discuss is code documentation. Regarding the smart contracts, the code is documented using godoc, using their standard code documentation practices. In respect of the Node.js application, we used the tool JSDoc, given that it is well designed and widely used.

Chapter 5

Conclusions

In this thesis, we developed a permissioned blockchain network utilizing smart contract logic to build a marketplace for personal data exchange while storing transactions in an immutable ledger.

5.1 Summary

Blockchain technology has marked the beginning of a new era backed up by democracy, privacy, and security. Complete ownership of assets revolutionizes the way people think of proceeding with technology in the future. Our blockchain solution is fundamentally built on the intention of creating a reliable data marketplace. Citizens are granted the ability to safely buy and sell sensitive data via a privacy-preserving and GDPR compliant platform.

Any data seller has the option to specify a legally bound policy that determines the way their data will be used by buyers, either individual citizens or institutions. Trust in this system originates from the fact that the logic for matching a product's policy with the buyer's intent of use is governed by smart contracts.

Our implementation ensures a user is the sole holder of their private crypto material and no other member on the network can execute transactions on their behalf. Also, a seller always knows who is using their data and for what purpose, thus being in a safe environment of sensitive data exchange. Therefore, this use case incentivizes citizens to sell their medical data to researchers who continuously need data to make substantial progress.

5.2 Future work

Regarding future development, the extent to which this network can expand is very wide. Multiple components can be integrated easily and other use cases can be added as simply as creating a new chaincode in another channel.

One of the main features to be added is an SSI module that validates a user's credentials upon registration and subsequent logins. SSI gives the ability to a user to verify their identity without anyone having access to their personal information. It is a state-of-the-art technology which is being developed in recent years and will be used widely in the future.

Another key component is the integration of an SMPC handling data analytics. Specifically, an option will determine the seller's data protection type, e.g. SMPC or anonymization. Depending on the option, their data will be handled accordingly and buyers will be able to purchase outcomes of aggregated data while preserving privacy.

Bibliography

[1] Eric J. Topol. *The patient will see you now: the future of medicine is in your hands.* 2015.

[2] Adil Hussain et al. Seh. *Healthcare Data Breaches: Insights and Implications.* 2020.

[3] *Ethics and data protection.* European Commission, 2018. URL: https://ec.europa.eu/info/sites/info/files/5._h2020_ethics_and_data_protection_0.pdf.

[4] Satoshi Nakamoto. *A peer-to-peer electronic cash system.* 2008.

[5] "Tech Giants' Blockchain Projects". In: *www.e-zigurat.com* (2019). URL: https://www.e-zigurat.com/innovation-school/blog/tech-giants-blockchain-projects/.

[6] N Szabo. "Formalizing and Securing Relationships on Public Networks." In: Vol. 2 no. 9 (Sept. 1997). DOI: 10.5210/fm.v2i9.548.

[7] Vitalik Buterin. *Ethereum: A next-generation smart contract and decentralized application platform*. 2014. URL: https://github.com/ethereum/wiki/wiki/White-Paper.

[8] *An Introduction to Hyperledger.* The Linux Foundation, 2018. URL: https://www.hyperledger.org/wp-content/uploads/2018/08/HL_Whitepaper_IntroductiontoHyperledger.pdf.

[9] Elli Androulaki et al. "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains". In: *Proceedings of the Thirteenth EuroSys Conference*. EuroSys '18. Porto, Portugal: Association for Computing Machinery, 2018. ISBN: 9781450355841. DOI: 10.1145/3190508.3190538. URL: https://doi.org/10.1145/3190508.3190538.

[10] Drasko Draskovic and George Saleh. "Decentralized data marketplace based on blockchain". In: *White Paper* (2017).

[11] "MyHealthMyData". In: *www.myhealthmydata.eu* (2019). URL: http://www.myhealthmydata.eu/why-mhmd/.

[12] C. Beleites. *Deploying a production network*. 2013. URL: https://hyperledger-fabric.readthedocs.io/en/release-2.2/deployment_guide_overview.html.

[13] Hoda Rashina, Noble James, and Marshall Stuart. "Agile Project Management". In: June 2008. ISBN: 978-3-540-26277-0. DOI: 10.1007/11499053_47.

[14] Marchesi Michele, Marchesi Lodovica, and Tonelli Roberto. "An Agile Software Engineering Method to Design Blockchain Applications". In: Oct. 2018, pp. 1–8. DOI: 10.1145/3290621.3290627.

Appendices

Appendix A

User Operations Flowchart

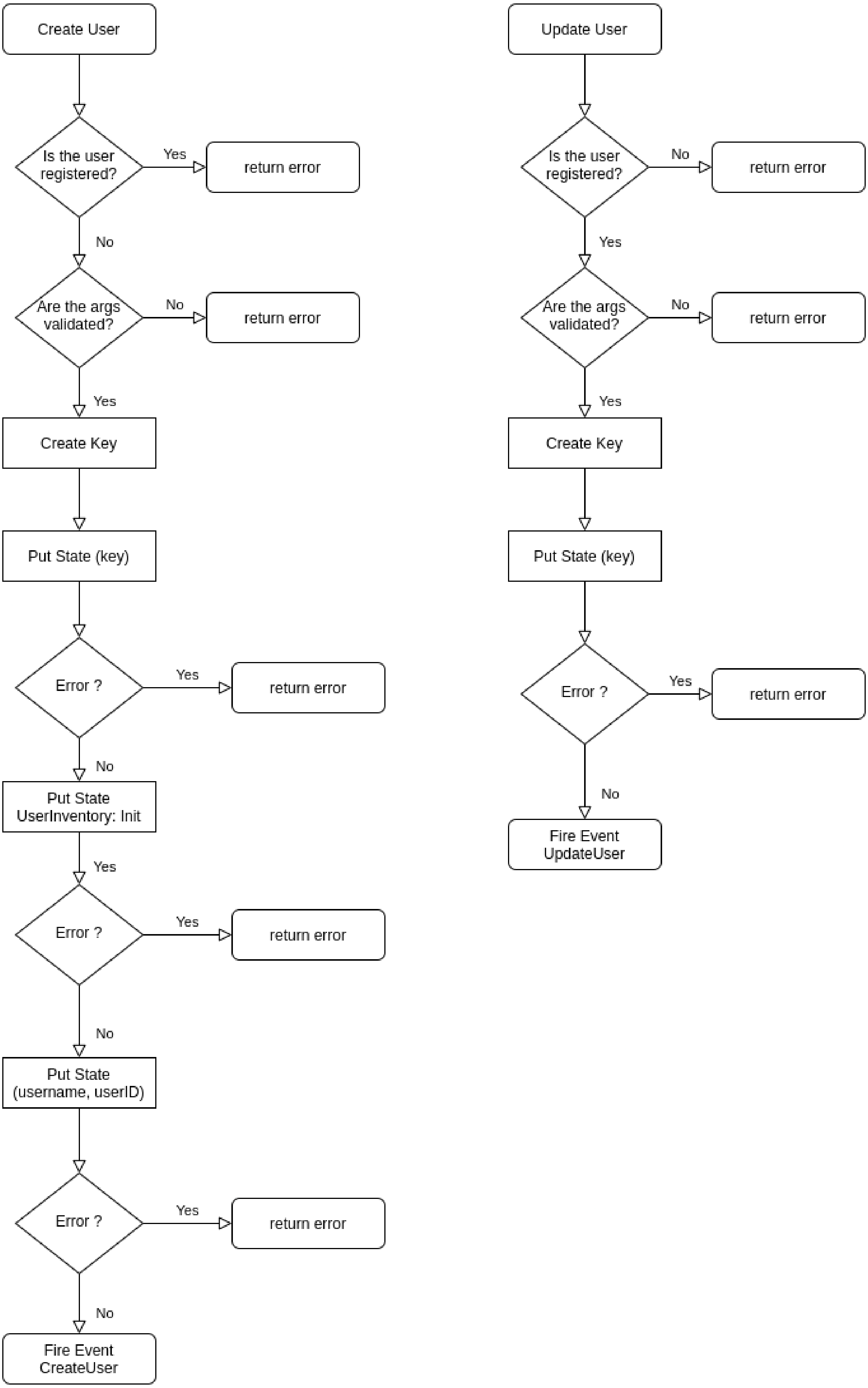

Figure A.1: User operations flowchart (Create, Update)

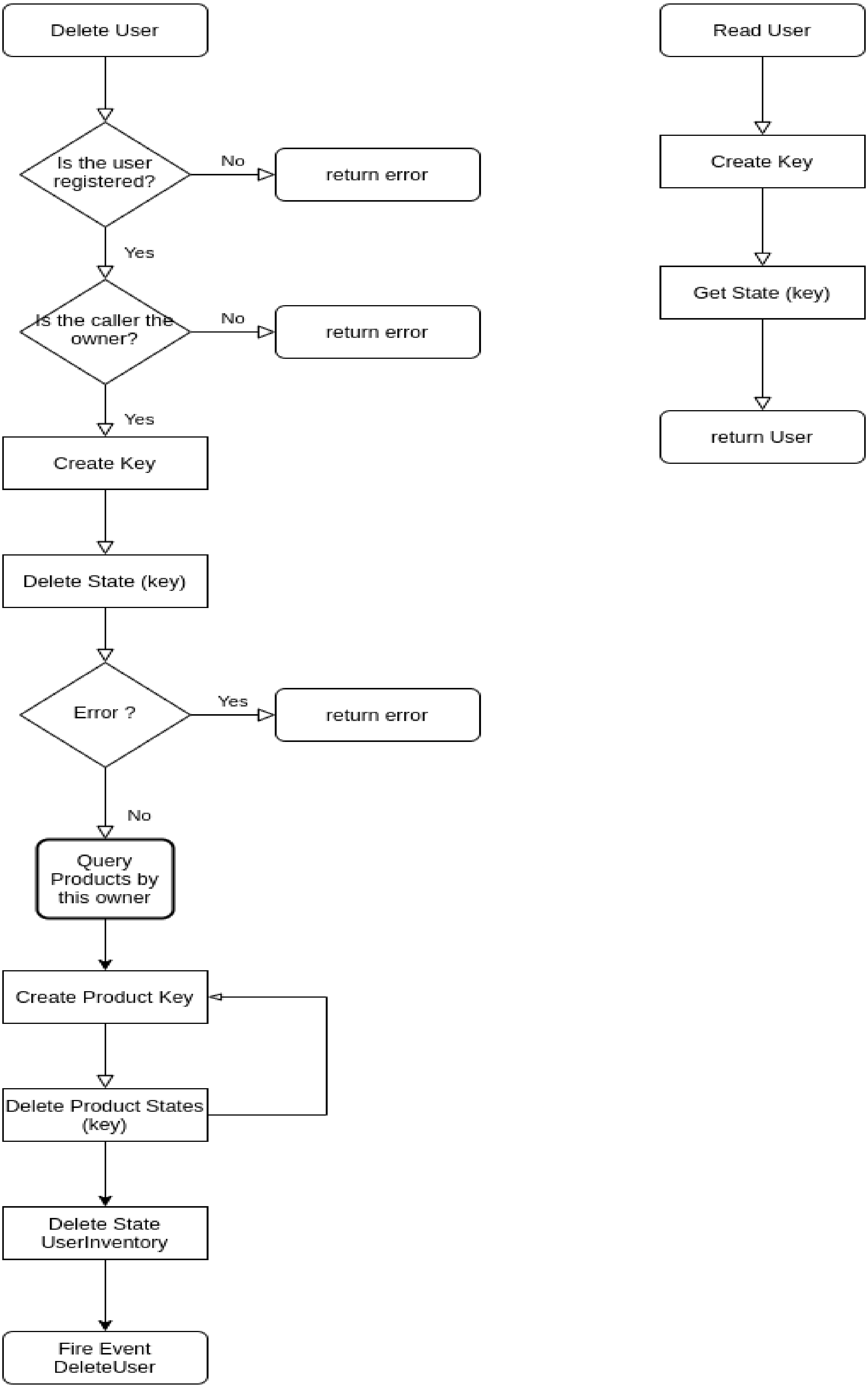

Figure A.2: User operations flowchart (Delete, Read)

Appendix B

Data Operations Flowchart

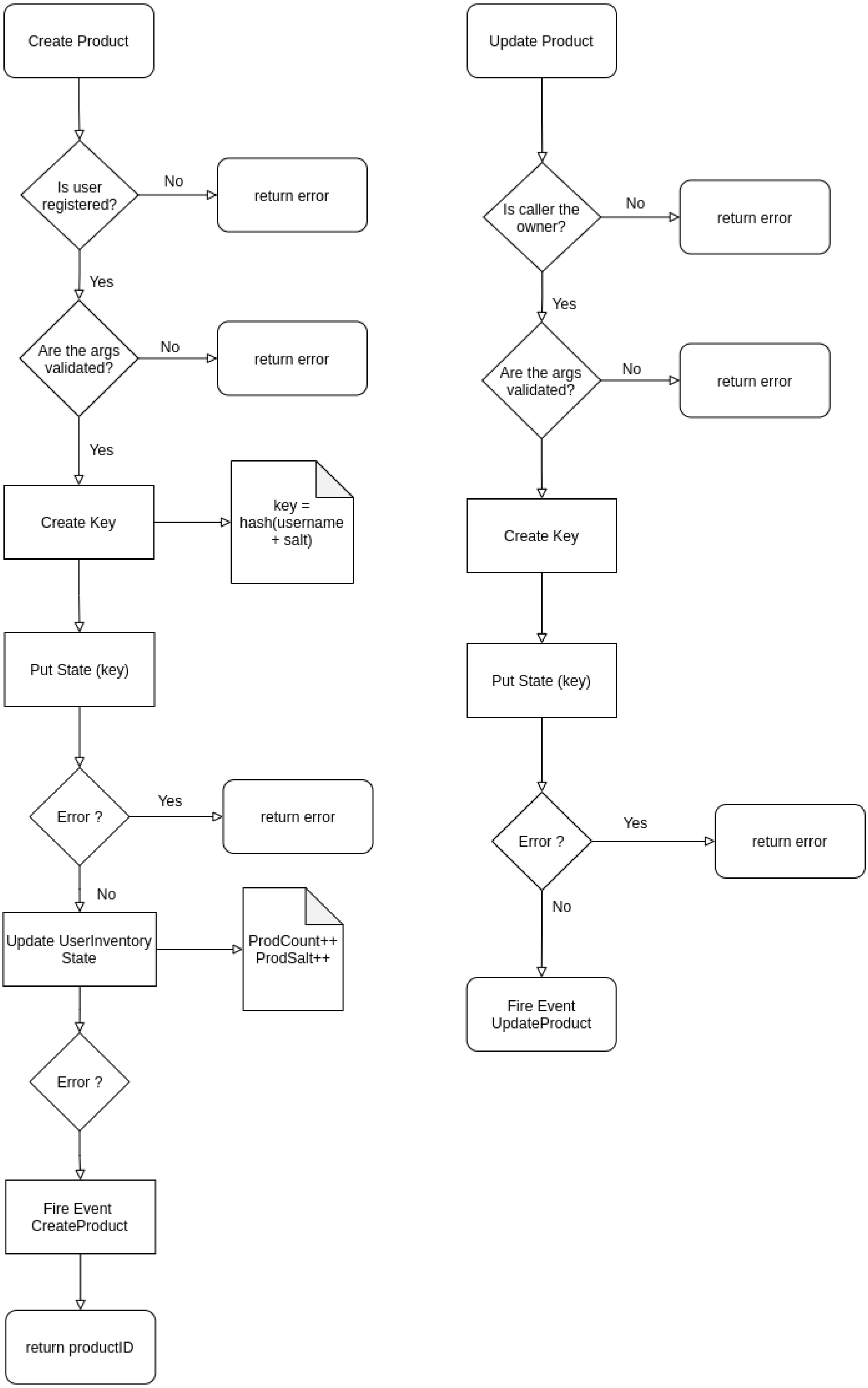

Figure B.1: Data operations flowchart (Create, Update)

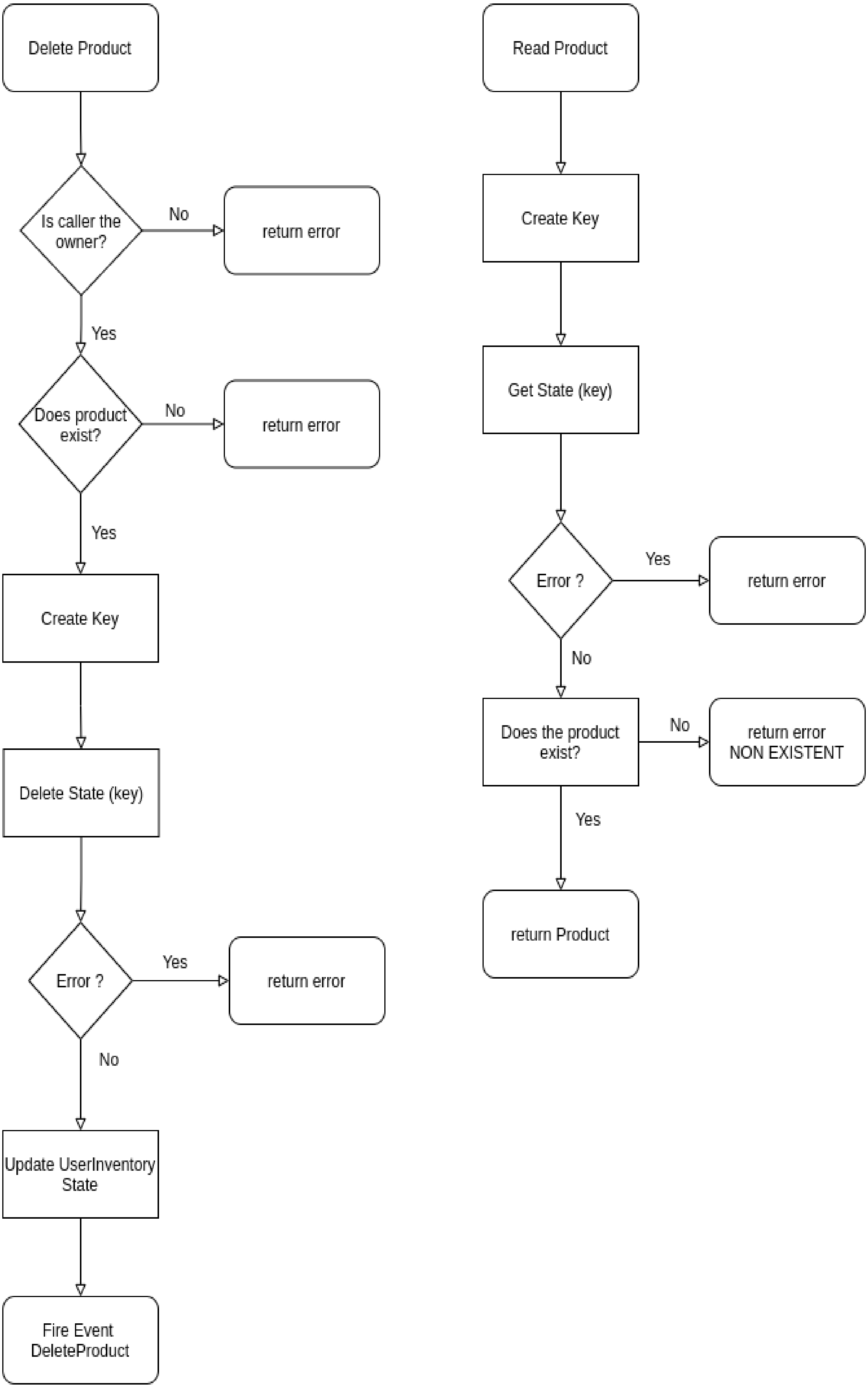

Figure B.2: Data operations flowchart (Delete, Read)

Appendix C

Agreement Operations Flowchart

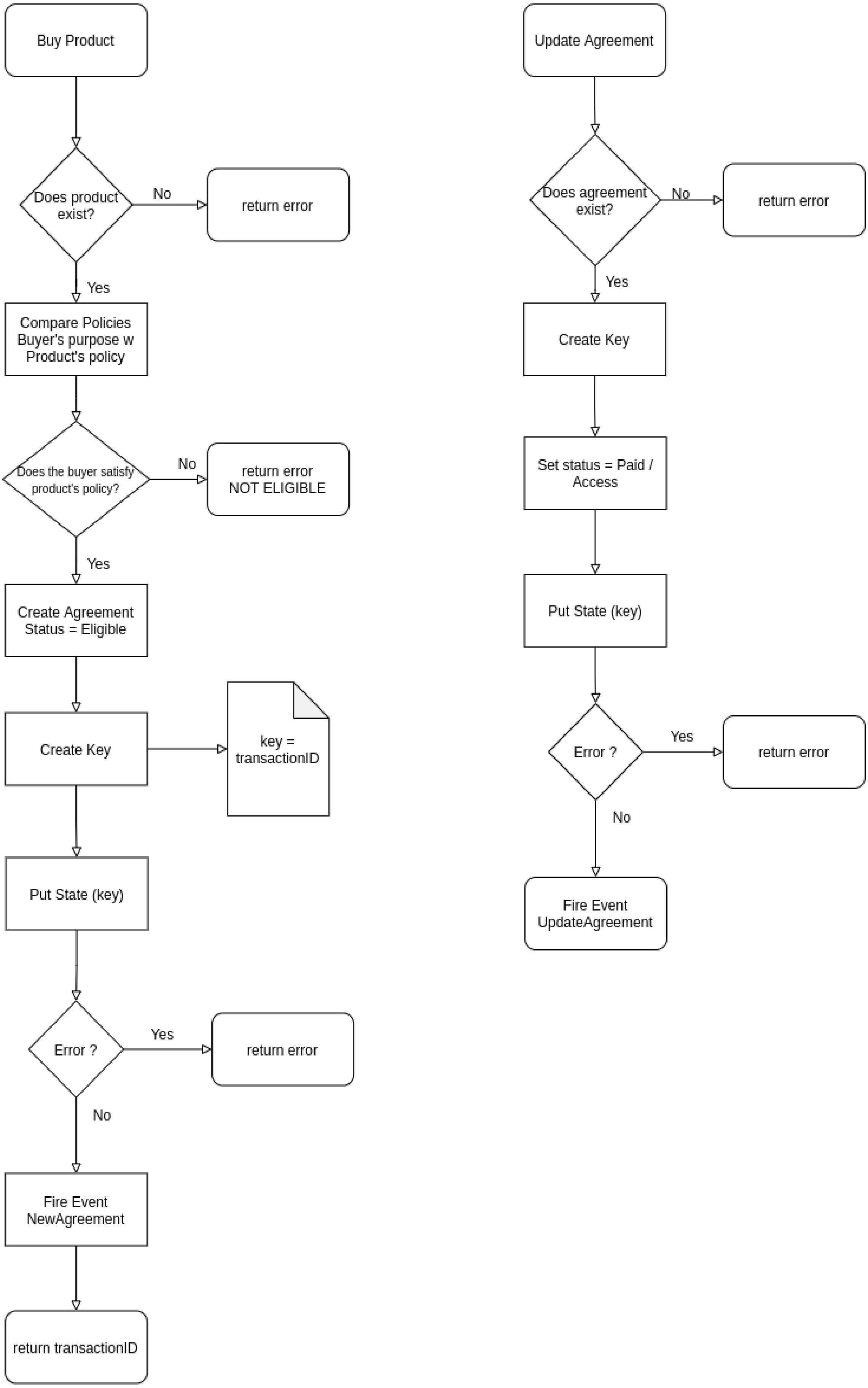

Figure C.1: Agreement operations flowchart