Freie Universität Berlin

Secure Data Exchange Using Distributed Ledger Technologies

*A **thesis** presented for the degree of **Bachelor of Science**.*

Lucas Antelo, Freie Universität Berlin, Germany

Matriculation number: 5094454

antelo.lucas@fu-berlin.de

21st October 2019

Supervisor:

Nicolas Lehmann[1], Freie Universität Berlin, Germany

Reviewers:

Prof. Dr. Agnès Voisard[2], Freie Universität Berlin, Germany

Prof. Dr. Matthias Wählisch[3], Freie Universität Berlin, Germany

Citation:

Lucas Antelo, *Secure Data Exchange Using Distributed Ledger Technologies*, Freie Universität Berlin, Bachelor Thesis, 2019

[1]Dept. of Computer Science and Mathematics, Databases and Information Systems Group
[2]Dept. of Computer Science and Mathematics, Databases and Information Systems Group
[3]Dept. of Computer Science and Mathematics, Internet Technologies Group

Statutory Declaration

I declare that I have developed and written the enclosed Bachelor thesis completely by myself, and have not used sources or means without declaration in the text. Any thoughts from others or literal quotations are clearly marked.

The Bachelor thesis was not used in the same or in a similar version to achieve an academic grading or is being published elsewhere.

Berlin (Germany), 21st October 2019

Lucas Antelo

Zusammenfassung

Im Deutschen Gesundheitswesen werden Daten zwischen Patienten, Gesundheitsdienstleistern, Krankenversicherungen sowie privaten und öffentlichen Instituten für die Verrichtung von Dienstleistungen, Forschung und der Begleichung von Rechnungen ausgetauscht. Diese Austausche werden standardmässig mit Papier vollbracht und kosten den Beteiligten Zeit, Ressourcen und Geld. In den letzten Jahren sind die Menschen mobiler geworden und ziehen häufiger von einer Stadt zur nächsten um. Patienten und Gesundheitsdienstleister arbeiten mit bruchstückhaften Gesundheitsakten, welche an verschiedenen Stellen im Gesundheitssystem verteilt sind. Folglich haben Gesundheitsdienstleister Schwierigkeiten die bestmögliche Therapie einzuleiten, die sie aufgrund von lückenhaften Informationen durch das Gespräch mit den Patienten vorschlagen. Dies führt dazu, dass das Gesundheitswesen nicht effizient und wirtschaftlich arbeiten kann. ”Distributed Ledger Technologies”(DLT) versprechen eine Lösung zu diesem Problem und somit ein Zeit-, Ressourcen- und Geldersparnis für den Austausch von Daten. In den letzten Jahren entstanden neue DLTs, welche für das Finanzwesen, die Versorgungsketten und das Gesundheitswesen entwickelt wurden und die Einführung von DLT in weiteren Geschäftsbereichen erleichtert. In der vorliegenden Arbeit werden die Gesetze, Vorschriften, Daten und Prozesse im Gesundheitswesen untersucht. Desweiteren wurde eine Anforderungsuntersuchung durch Gespräche mit einer Krankenversicherung, zwei Klienten und zwei Fachärzten gemacht, um eine Orientierung für die Untersuchung von geeigneten DLTs für das Gesundheitswesen zu schaffen. Die Resultate zeigen, dass Hyperledger Fabric (HLF) mit ihrer modularen Architektur eine Mehrzahl an Anforderungen erfüllt und mit den Gesetzen und Vorschriften im Einklang ist. Diese Technologie erlaubt den Entwicklern Zugriffsrechte auf Daten und Transaktionen zu definieren und anzupassen. Desweiteren erlaubt HLF die Automatisierung des sicheren Datenaustauschs mit Hilfe von Softwareprogrammen, welche als Chaincode bekannt sind. Diese These präsentiert einen Bauplan und Prototypen, die einen sicheren Datenaustausch im Gesundheitswesen ermöglichen. Der Prototyp basiert auf der 3-Schichten-Architektur und benutzt hauptsächlich HLF als Datenschicht und eine modulare NodeJS Webapplikation als Logikebene.

Schlagwörter Distributed Ledger Technology, Ethereum, Hyperledger Fabric, Gesundheitswesen, Datenaustausch, NodeJS, Express, Passport, MongoDB, Amazon AWS S3

Abstract

In the German healthcare sector, data are exchanged among patients, healthcare service providers and facilities, insurance companies as well as private and public institutions for services, research and billing. For the majority of these interactions, paper remains the de facto standard and costs all participants time, resources and money. Today, people are mobile and move more frequently from one city to another, where they usually seek healthcare related services. Patients and healthcare service providers possess a fragmented medical history, which is spread among many different locations. Consequently, healthcare service providers encounter difficulties concerning how to deliver the best possible treatment based on the incomplete information provided by the client during each visit. This impedes the healthcare system ability to be efficient and cost effective. Distributed ledger technologies (DLT) promise to solve this problem and consequently reduce time, resources and energy for the exchange of data. In recent years, new DLT solutions emerged that suit specific business use cases such as finance, supply chains and healthcare, which facilitate the adoption of this novel technology. In this thesis, the respective laws, regulations, data and processes in the healthcare industry are analyzed. In addition, a requirement analysis based on interviews with a health insurance company, two clients and two healthcare service providers are performed to build the foundation for the study of suitable DLT solutions in the healthcare industry. The thesis concludes that Hyperledger Fabric (HLF), with its modular architecture, fulfills the majority of the requirements as well as laws and regulations. This shows that HLF solves inherent problems found in the healthcare industry. This technology implements access control lists for data access, transactions, and moreover, allows the creation of software programs, known as chaincode, to automate secure data exchange procedures. This thesis results in the creation of a blueprint and a prototype as a data exchange grid for the healthcare industry. The prototype uses a three-tier software architecture with HLF as the main data layer and a modular web application as the logic layer using NodeJS.

Keywords Distributed Ledger Technology, Ethereum, Hyperledger Fabric, Healthcare, Data Exchange, NodeJS, Express, Passport, MongoDB, Amazon AWS S3

Table of Contents

List of Tables

List of Figures

List of Source Codes

1. Introduction

The section *Motivation* discusses distributed ledger technologies (DLTs) and its current implementations in the healthcare setting. Furthermore, the section dives into the problems of data silos that exist in several healthcare systems worldwide and how DLTs solve the connections among these silos by improving the availability and completeness of electronic healthcare records and consequently the quality of care. The second section, *Outline of Contribution*, gives an overview of the contributions from this thesis and the implications for the current German healthcare system. Finally, the last section, *Structure of the Thesis*, introduces the reader to the structure of this thesis.

1.1. Motivation

At the Cebit 2018 in Hanover, blockchain caught a lot of attention among visitors, and the speakers claimed that this technology solves trust related problems encountered in most industries. The presentations during this congress focused mainly on Bitcoin and how blockchain improves the business process for merchandisers, the trust between business partners, and finally, how such technologies facilitate business transactions. At the Future Blockchain Summit 2019 in Dubai, many companies presented DLT solutions on how to make bureaucracy more efficient [86] and showed that DLT is becoming more mature and thus attractive for businesses. Many sectors are currently struggling with the overwhelming amount of data generated by each individual in an ever-growing population. Many German institutions still provide many services in paper format without leveraging current and new technologies that could potentially improve the current workflow. Today, the healthcare sector has many problems due to shortages of healthcare professionals, outdated IT infrastructures and the burden of an aging society with a higher demand on medical care. And the number of problems are estimated to rise in the next two decades. Thus, the question emerges if DLT is able to at least solve some problems in this sector.

Unfortunately, actual DLT implementations within the healthcare industry remain quite obscure and inquiries over specific implementations inconclusive. Nevertheless, the hype around blockchain and healthcare does not wane and the fear of a bubble is not negligent. As of 2019, Estonia remains the only country in the world to have implemented DLT on a national scale. Estonia implemented this technology for the Estonian Electronic Healthcare Records in collaboration with Nortal, Helmes, Guardtime and INTELSYS [29, 26] in 2016, which allows patients to access their electronic health records and check for any inconsistencies. This makes tampering with the data more difficult to accomplish. For these reasons, the Estonian Government and Public Health continue their efforts in this field in collaboration with IT companies to secure their IT infrastructure with suitable technologies.

In the future, electronic health records (EHR) will be the norm in the United States due to political and societal pressures [24] and this is expected to be similar in Germany in the next two decades. The next step is to coordinate these data silos held by each healthcare facility such that patients and healthcare providers have access to a complete history of health records, improving the availability and completeness of their medical history, and consequently, the quality of care. These electronic health records form the basis for reimbursements, clinical research, teaching and legal disputes, to name a few. And soon, this data will be used to train artificial intelligence applications [67], improving the efficiency and quality of care as well as providing enhanced tools for future healthcare professionals.

In general, DLTs reliably and persistently store records of data and data transactions in a network among different parties. DLT guarantees data integrity by making these records immutable. On the other hand, by making these records publicly available, confidentiality concerns arise over the storage of data and transactions.

Fortunately, there exist several solutions concerning how to ensure privacy on a public ledger [7, 15, 17, 66], which need to be implemented on top of technologies such as Ethereum. Nevertheless, the General Data Protection Regulation (GDPR) and health laws impose further restrictions, like the right of correction and deletion of personal information, which is not trivial when using an immutable public ledger as a data structure. Based on these restrictions, new DLT solutions emerged as many businesses worldwide need to comply with GDPR and other regulations. Such new solutions are R3 by Corda Enterprise and the Hyperledger project by the Linux Foundation.

This thesis analyzes the environment for data exchanges for highly sensitive data in different public ledger settings and focuses on the question if DLT allows the secure data exchange among participants in the German healthcare industry.

In the current German healthcare system, health records are stored electronically or in paper format. These records are usually incomplete and not easily transferable between healthcare providers, making the access of such information difficult to accomplish. Such circumstances interfere with the quality of care and, moreover, their effectiveness. Beyond that, patients are more self-contained compared to the last century such that the business model is transitioning from a doctor-centered to a patient-centered business model, where the patient decides over the next steps. Consequently, DLT is the next trivial step where the information is distributed according to the data owner allowing for an extension of the new business model approach.

Today, people are mobile and move more frequently from one city to another, where they usually seek healthcare related services. Consequently, healthcare service providers encounter difficulties on how to deliver the best possible treatment based on the incomplete information provided by the client in each visit. In general, no complications are expected after a consultation, but this does not hold for emergency rooms, where treatment needs to be performed immediately. In such situations, many complications arise as demonstrated in the following example.

Example 1: *Mr. Nakamoto is 21 years old and has recently suffered a pneumonia. His general practitioner, Dr. Ripple, treated him with the appropriate antibiotic and anti-inflammatory drugs. During the treatment period, Mr. Nakamoto decides to visit his girlfriend in Zug, Switzerland. When he arrives at the train station in Zug, he starts to feel dizzy and loses consciousness shortly after. The ambulance brings him to the emergency department at the Zuger Kantonsspital, where healthcare professionals start to treat him based on the current clinical findings. Unfortunately, his health condition deteriorates and Mr. Nakamoto is put under artificial life support as his cardio-respiratory system is failing. His girlfriend was unaware of his current health condition nor does she know who his current general practitioner is. This situation could have been prevented, if Dr. Ripple had made specific information available and thus avoided the administration of a medication, which Mr. Nakomoto was allergic to and caused the further deterioration of his condition.*

This example shows a realistic scenario that happens on a daily basis worldwide in many healthcare facilities. Unfortunately, health records remain highly inaccessible in a highly mobile population causing fatal results. This asks for an appropriate infrastructure for the healthcare setting that makes critical information available to caretakers, improving the quality of care and the clinical outcome of their patients. DLTs enables the storage of such information making it easily accessible where and when needed. On the other hand, security and privacy concerns arise when using such a technology, which need to be addressed.

This thesis addresses the main problem in the current healthcare setting by showing how to make these data silos accessible with DLT. In theory, this technology facilitates the completion of individual health records, that allows healthcare professionals access to critical information in life-threatening situations. In addition, DLT provides patients the tools to keep a comprehensive health record on which better decisions can be made.

1.2. Outline of Contribution

The goal is to verify if a secure data exchange of sensitive data is possible using DLT under the many restrictions that apply. For this purpose, this thesis analyzes the respective laws, regulations, data and processes in the healthcare industry and DLT solutions for a secure data exchange among patients, healthcare providers, health insurances and institutions. The interviews with a health insurance company, two clients and two physicians give an overview of the current data exchange processes in the German healthcare setting and provide the requirements for the analysis of DLTs. Two candidates represented by Ethereum and Hyperledger Fabric (HLF) are examined based on the requirements as well as security parameters that include authenticity, privacy, anonymity, confidentiality, nonrepudiation and data integrity. These candidates are further analyzed under complementary criterias that facilitate the collaboration between different IT infrastructures run by the participants such as control,

openness, interoperability, reliability, auditability and accountability. From these investigations a suitable technology is chosen and used for the design of a blueprint facilitating the secure data exchange between two participants for sensitive medical data. In addition, a prototype is created based on the blueprint and tested along the implementation. The prototyping helps to understand the difficulties, disadvantages, advantages and finally the practicality of this technology.

To sum up, the reader is introduced to the current data exchange practices, the problems and the requirements for a secure data exchange of medical data next to the security and interoperability aspects. An introduction and presentation of DLT solutions allows the reader to understand this technology and the advantages and disadvantages for the German healthcare system. The analysis of HLF as a promising candidate helps the reader understand what difficulties software applications confront in the healthcare setting. The blueprint and prototyping provides an overview and insights on the implementation and complications of HLF with other technologies. At the end, the reader understands how the German healthcare system currently performs data exchanges, what DLT is and that there exist several different solutions. In addition, a publicly available prototype and source code showcase the result from this thesis.

1.3. Structure of the Thesis

This chapter, **Introduction**, presents the topic of the thesis organized in three sections. The first section, *Motivation*, presents my personal interests. The second section, *Outline of Contribution*, points to the value of this thesis and the third section, *Structure of the Thesis*, gives an orientation on this work.

The second chapter, **Background**, explains the definitions and introduces distributed ledger technology to the reader. The first section, *Definitions*, gives each definition used in this thesis a brief summary. In the second section, *Context of Work*, distributed ledger technology and two realizations are introduced, namely: Ethereum and Hyperledger Fabric. The third section, *Related Systems*, presents several DLT applications used in governments and the industry. Finally, the fourth section, *Related Work*, summarizes several research findings that showcase prototypes, examine security risks in current distributed ledger technologies and provide optional enhancements.

The third chapter, **Analysis of the Healthcare System**, presents in section *Health Ledger* the possibilities of DLT which are supplemented in the next section, *DLT in the Healthcare Industry*. The third section, *Data and Processes*, gives an overview of the current situation in the German healthcare system. In section *Security*, the primary requirements covering security aspects on data exchanges are set. The next section, *Laws and Regulations*, discusses the implications and defines legal requirements based on GDPR. In the sixth section, *Requirement Analysis*, user-defined re-

quirements are defined based on interviews with representatives from the healthcare industry.

The fourth chapter, **Design Decisions**, merges the findings from the previous chapters and defines the design used in the fifth chapter, *Implementation*. In section *Goal*, the desired outcome is defined and used as a landmark for the decisions on the design of the prototype. The second section, *Decision on DLT*, compares Ethereum with HLF and explains the decision behind HLF for the prototype. The last section, *Strategies*, resolves the elaborated requirements and provides solutions, assumptions and strategies to achieve the required outcomes.

The fifth chapter, **Implementation**, explains several aspects of the software application that represents the prototype. The first section, *Architecture*, introduces the applied architectural and design patterns. The second section, *Technologies*, presents the libraries, software tools and technologies that are used for the prototype. Finally, the last two sections, *Frontend* and *Backend*, present the actual implementation of the prototype and include views and excerpts from the source code, which implements the business logic.

The sixth chapter, **Evaluation**, analyzes the implementation of the requirements and the degree of accomplishment. The first section, *Security Aspects*, evaluates the implementation of the requirements that cover authentication, privacy, anonymity, non-repudiation, confidentiality and data integrity. The second section, *Legal Requirements*, evaluates a subset of the GDPR. Finally, the third section, *User-Defined Requirements*, analyzes the implementation of the requirements that originated from the interviews with representatives from the healthcare industry.

The seventh chapter, **Results**, summarizes the findings and results of the investigated topic in the first section, *Summary*. The second section, *Limitations*, presents inherent problems of DLTs and indicates the limitations of the prototype.

Finally, the last chapter, **Conclusion**, expresses my personal view on the results and concludes the thesis.

2. Background

The healthcare industry is diverse in its needs towards digital solutions. There exist several proprietary software solutions and data management systems to meet the requirements of each business and healthcare facility, which date back from a time where engineers did not anticipate the need for interoperability. This is reflected by the implementation of proprietary data structures and exchange protocols from each vendor that makes the integration of these systems so difficult to accomplish and consequently expensive. Participants are therefore not satisfied with the current situation that demand from software applications that they are open and capable of working with other systems. In addition, the number of clients demanding the access to their health data is increasing and posing further pressure on the development of new solutions in a restrictive industry. This thesis investigates the distributed ledger technologies in the context of healthcare related data exchanges under the following requirements such as

a) **control** over the data by the owner;

b) **openness**, and thus access to critical information,

c) **interoperability** between systems;

d) **reliability** of the database and application;

e) **auditability** and

f) **accountability** to comply with current regulations, and moreover,

g) **authentication and authenticity**;

h) **privacy** and **anonymity** of the user, as well as

i) **nonrepudiation**;

i) **confidentiality** of data exchanges;

k) **data integrity**, and finally,

l) **security**.

For this purpose, this chapter introduces important definitions, the distributed ledger technology, related systems and works discussed in the respective sections in order to follow and understand the analysis. The results are used for the prototyping of a secure data exchange software solution for the German healthcare system.

2.1. Definitions

Distributed Ledger Technology (DLT) is a technology stack in a distributed network of interconnected participants that share a copy of a data structure otherwise known as a ledger. The organization and structure of the ledger follow a well-defined protocol to validate operations onto and appendage of new data to the ledger. DLT uses several technologies and game theory as their underlying mechanisms and reflects an umbrella term for a huge variety of different solutions.

Blockchain belongs to the family of DLTs and uses a distributed ledger of transactions stored by each participating node in a peer to peer network (P2P). The ledger represents a linked list, where each block includes the reference to its previous block and thus equals a chain of blocks. The consensus method is usually competitive, based on proof-of-work but it may differ depending on the implementation of its underlying protocol. The main representatives of blockchain are Bitcoin and Ethereum [34].

Directed acyclic graph (DAG) represents a finite directed graph with no directed cycles and is used as a ledger in several DLT solutions. This graph consists of vertices and edges, which represent transactions (vertices) and hash values (edges) directing to the respective vertex. Here, the consensus method is usually cooperative and its main features consist in low energy consumption next to its absence of fees. The main representatives are IOTA, Byteball and Nano [34].

Hyperledger represents a family of open source blockchain technologies hosted by the Linux Foundation, which provide DLT solutions for finance, supply chains and healthcare. In addition, tools for the development of Hyperledger solutions allow developers to automate many tasks for the deployment, monitoring and debugging of a Hyperledger application. In general, Hyperledger uses blockchain for the ledgers as transaction log. The data is stored in one of two currently available database management systems. The database represents the world state. In addition, it uses a unique consensus mechanism that differs from other technologies and reflects the processes found in many enterprises [4, 39].

Smart Contracts are used in Ethereum. These are computer programs stored in the ledger and each participating node storing a replica of this ledger can run these programs. These contracts allow operations to be performed on a ledger and are capable of interacting with other contracts. Smart contracts may be written in any language, but Solidity is the de facto standard and therefore recommended for the creation of smart contracts [9, 37].

Ethereum Request Comments 20 (ERC-20) defines a technical standard for the definition of tokens in Ethereum. A token is defined by a smart contract and the standard facilitates the trade among participants by providing basic functionalities to transfer and approve these tokens [92].

Chaincode is the term for smart contracts in the Hyperledger family. Chaincode are computer programs stored in the participating nodes that include the smart contract,

configuration details and policies. The chaincode implements the business logic for put, create and delete operations that are recorded in the ledger of the respective channel and the system ledger. This fact is very different from the smart contracts used in Ethereum, where all transactions are always recorded in one ledger. Chaincode are written in Go, Java and Javascript [59].

Certificate Transparency (CT) is an internet security standard that helps increase transparency in the process of issuing certificates such that certificate authorities can be held responsible for their actions. Furthermore, bad certificates can be caught and revoked early on. This approach introduces three entities: a *log server* keeping track of issued certificates, an *auditor* responsible for keeping track of the presence or absence of certificates in the log server, and finally a *monitor* responsible for checking the quality of the certificates [16].

Decentralized Autonomous Organization (DAO) is an organization that acts autonomously and makes decisions based on the business logic in the software or through the vote of its members. In essence, it is a system of hard coded rules that define which actions an organization takes. The financial transaction record and program rules are maintained on a distributed ledger [9, 90].

General Data Protection Regulation (GDPR) specifies the rules on data protection and privacy in the European Union (EU). GDPR applies to all individuals within the EU and the European Economic Area (EEA). It specifies the export of personal data outside the EU and EEA and regulates the storage, access and processing of this data [19].

NoSQL stands for Not Only Structured Query Language and allows the storage and management of a wide variety of data models such as documents. NoSQL defines a database design that is flexible, as it allows for fast changes in the data schema and is moreover suitable for prototype projects [77].

Oblivious Random Access Memory Machine (ORAM) allows a client to store data on a trustless server without compromising safety requirements. The client performs, reads and writes remotely by mapping logical memory addresses to remote physical addresses. Freshness, integrity and confidentiality are ensured by using authenticated encryption and minimal local state. ORAM hides memory access patterns such that a malicious server cannot distinguish two client operations having the same length. In ORAM, only the client needs to verify the integrity [15].

Publicly-Verifiable Oblivious RAM Machine (PVORM) represents an extension of Oblivious Random Access Memory (ORAM). ORAM can be extended by defining a set of application-specific operations, legal and business requirements that are imposed on all updates. This extension of ORAM is defined as Publicly-Verifiable ORAM. PVROM allows users to hide account balances and data exchange details from non-transacting entities. These data exchanges are placed in the ledgers as cipher texts in order to impose strong confidentiality. Consequently, parties can only deduce the identities of the sending and receiving entity, but not of the explicit client

behind each entity nor its content. Not even the sending entity can deduce the receiving client behind the receiving entity and vice versa. In addition, the extension allows to publicly verify that such transactions have been correctly processed by both entities by an external institution [15].

RESTful API stands for Representational State Transfer (REST) and defines an architectural style for the design of network-based software architectures created by Roy Fielding [36]. RESTful Application Programming Interfaces (API) follow this architecture and provide the respective create, read, update and delete interface definition for the software application.

Trusted Execution Environment (TEE) is a secure enclave on a hardware with the guarantees that code and data loaded inside a processor are protected with respect to confidentiality and data integrity. These guarantees still hold when such an enclave is run on a hostile operating system. Nevertheless, availability cannot be guaranteed, as the hardware may shut down unexpectedly [15].

Web3.js is a Javascript library that allows a web application to interact with the Ethereum network using a hypertext transfer protocol (HTTP) or an interprocess communication (IPC) connection. It provides the interface to call smart contracts and execute transactions among other peers. This library needs a wallet and a storage medium for the private key, such as MetaMask [72], in order to execute the respective function calls [31].

X.509 certificate is a data structure for various public key protocols and allows the distribution of a single public key. The certificate contains several fields such as the distinguished names of the owner and the issuer, the public key, validity, serial number and the signature of the issuer [55].

Zero-knowledge proof uses mathematical concepts to construct a proof for given information without exposing the information. This allows a user, *u*, to another user, *v*, to proof that user *u* knows a given information without publishing the information by simply providing a secret information that allows user *v* to verify the proof. This procedure is based on simple computational assumptions without needing further knowledge nor assumptions [75].

2.2. Context of Work

This section gives a brief overview over DLTs and presents two representatives, namely: Ethereum and Hyperledger Fabric. These representatives allow the execution of complex operations on top of this technology.

2.2.1. Distributed Ledger Technologies

This technology uses a ledger that represents a data structure in a distributed network of interconnected participants, where a transaction represents the data generated from the execution of code. The ledger functions as a shared view of the current reality in the form of a replicated, shared, and synchronized data structure among peers, which makes this technology difficult to attack or hack since there is no single point of failure. The organization of the ledger and appendage of new data follow well-defined protocols and generally use hash functions to generate a reference value for each data entry. The references are used to interconnect the data entries along the ledger.

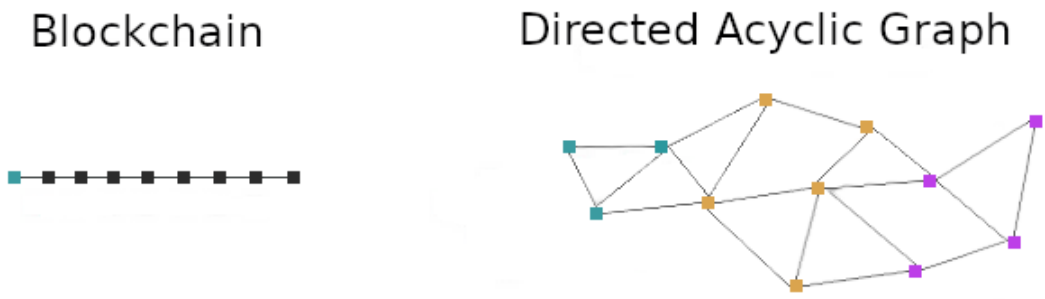

Figure 2.1.: This figure shows the differences between a blockchain and a directed acyclic graph. A blockchain ledger is a linked list, where data entries are represented as blocks containing transactions, metadata and code. The link is represented by the hash value of the previous block and ensures the data integrity and order of the ledger. In comparison, in a directed acyclic graph, the vertices represent the blocks that are connected to at least two blocks via the hash value of the respective blocks represented by the edges.

There exist two main implementations of the ledger consisting of a linked list of blocks called blockchain or a directed acyclic graph. In blockchain, transactions are accumulated until the storage capacity of the block is full. The data in each block defines the hash value and includes the hash value of the previous block. This allows participants to verify and detect any incongruities in the ledger and take actions accordingly. On the other hand, directed acyclic graph uses a finite directed graph with no directed cycles. This graph consists of vertices and edges, which represent transactions (vertices) and the hash value used as reference (edges).

The consensus mechanism defines how transactions are validated and how new blocks are appended to the ledger. These mechanisms are summarized as Proof-of-X, where X defines the specific implementation [10]. DLTs commonly use Proof-of-Work, Proof-of-Stake and Proof-of-Authority based on the trust model used for the network application.

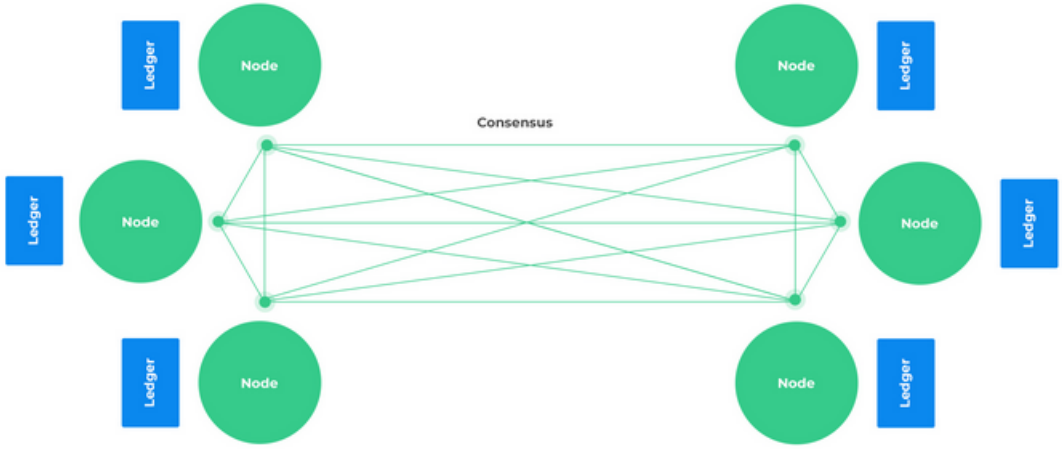

Figure 2.2.: In a DLT implementation, nodes are interconnected with each other. A node represents a server, desktop computer or a mobile device. Each node possesses a copy of the ledger that is replicated among all nodes. The consensus mechanism defines the structure of the ledger, the appendage of new data blocks and the validity of the ledger. Reprinted from [1]

- **Proof-of-Work (PoW)** scheme is a competitive protocol where nodes in the network need to solve a hash puzzle for the appendage of a new block. The hash puzzle is solved when the hash function results in a value below a certain threshold. The first published block with the correct solution is broadcast, validated by the peers and appended to the ledger [10].
- **Proof of Stake (PoS)** scheme provides a different approach to establish consensus. Nodes vote on new blocks weighted by their hash power, amount of currency or another suitable metric. Furthermore, PoS includes the random election of a leader among the stakeholders, that is allowed to announce new blocks which are appended to the ledger [10].
- **Proof-of-Authority (PoA)** is based on a predefined set of nodes that seal the blocks. In this consensus mechanism, no mining is performed and only the authorized nodes are allowed to announce new blocks to the network [8].

DLTs use either a public or private network. In a public or permissionless network, neither a central administrator nor a centralized data storage is needed. In this setting, the ledger is accessible to anyone who wants to participate and there is no need for authentication. By switching to a private or permissioned setting, the adaptation of the protocols enables a central administrator the power to modify the ledger and grant or revoke certain rights to users that need to be authenticated in the network.

Finally, this technology allows developers to implement software solutions on top of the network application using general-purpose or domain-specific programming languages. Turing-complete programs are not available on all technologies in order to prevent denial-of-service attacks from code snippets stored in the ledger.

2.2.2. Ethereum

Ethereum implements a blockchain as a ledger shared among all participants in the network. Ether is the inherent cryptocurrency needed to perform transactions and run applications on top of Ethereum. This technology is a general purpose blockchain platform and allows the implementation of Turing-complete software programs called smart contracts. These contracts allow operations to be performed on a ledger and are capable of interacting with other contracts and users. Solidity as a domain-specific programming language allows the creation of such smart contracts [9, 37], which are compiled to bytecode and stored as such in the ledger. The Ethereum Virtual Machine (EVM) executes the bytecode and is available on specific nodes in the network. For this purpose, a user needs an Ethereum account represented by a twenty byte long string created by applying a hash function twice in sequence on the respective public key. The associated private key is used to sign transactions. Finally, the user needs some Ether to pay for the transactions and consequently, the execution of a smart contract. These transactions include

- the recipient of the transaction,
- the signature of the sender,
- some Ether from the sender's account otherwise named as gas,
- an optional data field,
- a startgas field representing the maximum value the sender wishes to spend on the execution, and finally,
- the gasprice field the sender is willing to pay for each computational step performed by the node.

The startgas and gasprice fields prevent denial-of-service attacks, as smart contracts stop their execution whenever the gasprice times computational steps surpasses the amount of gas defined in startgas. In addition, these two fields guarantee that any smart contract execution finally ends and frees computational resources in the network [14]. Ethereum follows the order-execute architecture, where the consensus mechanism first orders the transactions and propagates them to all nodes and in a second step, needs each node to execute all the transactions in the given order . In other words, every node executes every transaction and all transactions need to be deterministic [4].

This technology provides a set of different protocols to be run in a public, permissioned or private setting to meet the requirements of the respective use case [68]. Decentralized Applications (DApps) provide the user interface to the smart contracts in the form of a mobile or web application [9]. Ethereum allows the implementation of several consensus mechanisms used in different public Ethereum networks such as PoW, PoS and PoA. The main network uses PoW currently known as the Ethereum network. In addition, there exist several test networks for development purposes that use PoW and PoA as consensus mechanism [37].

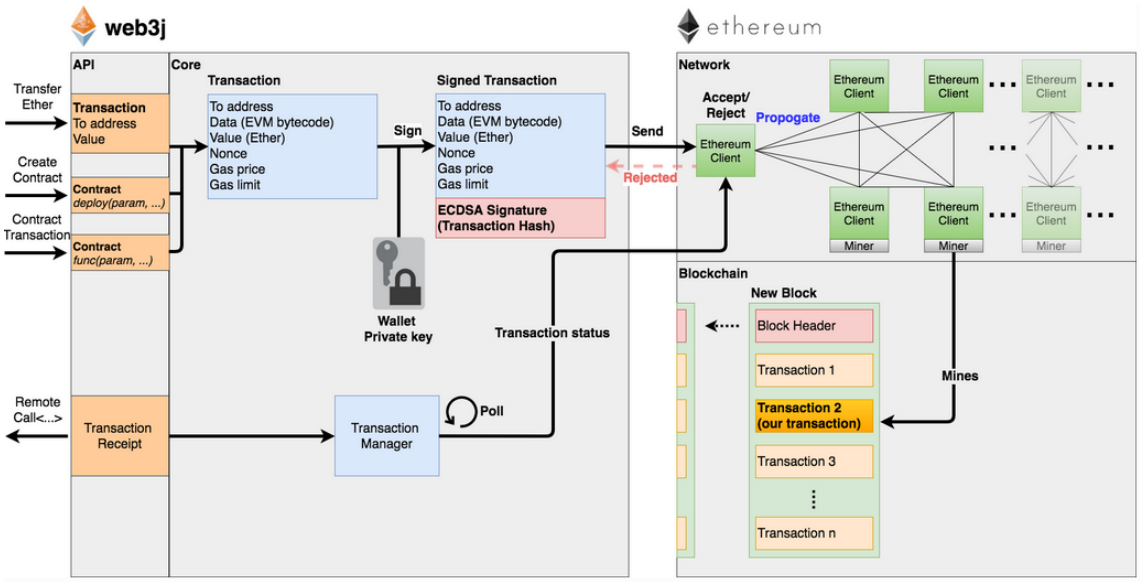

Figure 2.3.: In a web application using the web3.js library, a client can transfer Ether, create or call a smart contract by the given API in order to interact with the Ethereum network. Any transaction is signed with the client's private key stored in the wallet, such as MetaMask, using the Elliptic Curve Digital Signature Algorithm (ECDSA). The signed transaction is sent to one Ethereum client that either accepts or rejects the transaction. If the transaction is accepted, the Ethereum client propagates the transaction to all nodes in the network that is finally mined and broadcast via a new block. The transaction manager allows the client to follow the transaction status, which eventually terminates with the transaction receipt returned to the client [30]. Reprinted from [30].

2.2.3. Hyperledger Fabric

Hyperledger represents a family of open source blockchain projects hosted by the Linux Foundation and provide different distributions for finance, supply chains, healthcare and further use cases [39]. Hyperledger Fabric (HLF) is a distribution that was made publicly available on github on September 16, 2016 [54]. This distribution is extensible and uses a modular architecture such that consensus protocols can be tailored to specific trust models in a permissioned network. Chaincode specifies the transactions performed in the network that are written using general-purpose programming languages such as Java, Go and Javascript [59] without depending on a native cryptocurrency. Each chaincode is associated with a channel and the corresponding ledger. This ledger consists of two components and a copy is maintained by each peer in the channel [52]:

1. **World State Ledger** is a database that stores the unique and current values of the respective attributes.
2. **Transaction Log** uses an append-only blockchain ledger to record all transactions that resulted in the current world state.

Each chaincode implementation is associated with an independent append-only ledger that is only shared and accessible to a subgroup of peers, which HLF refers to as channel. In this channel, transactions can be performed independently from other channels such that HLF allows the parallel execution of several transactions. In addition, channels are only accessible to peers that are registered to the channel defined in the channel configuration file. This architecture follows the execute-order-validate architecture that separates transaction flow into three separate steps.

First, in the execution phase, the client signs and sends a transaction in the respective channel, where nodes simulate the execution of the request. Here, HLF differentiates between endorsing nodes, that execute and commit the transaction, and committing nodes, that simply commit the transaction. The endorsing nodes check the correctness of the simulation and its output is recorded in the read- and writeset. In addition, the client receives these results and signatures from all endorsing nodes in the channel. Second, in the ordering phase, the client sends the results and the signatures to a separate module called the ordering service that is responsible for the creation of blocks and the respective broadcasting. The ordering service does not validate nor execute any transactions as it is only responsible for the ordering and delivery of blocks in the network. Finally, in the validation phase, the ordering service broadcasts the next block including the transaction by the client to all nodes in the respective channel that perform the validation of the given endorsement policy. The endorsing nodes check the read- and writeset of the associated transactions and finalize the execution of the transaction by committing the new block to the ledger and consecutively updating the world state database, which is performed by all nodes in the respective channel. In HLF, every chaincode implicitly defines the endorsement policy for transactions and allows for modification to specific needs [4].

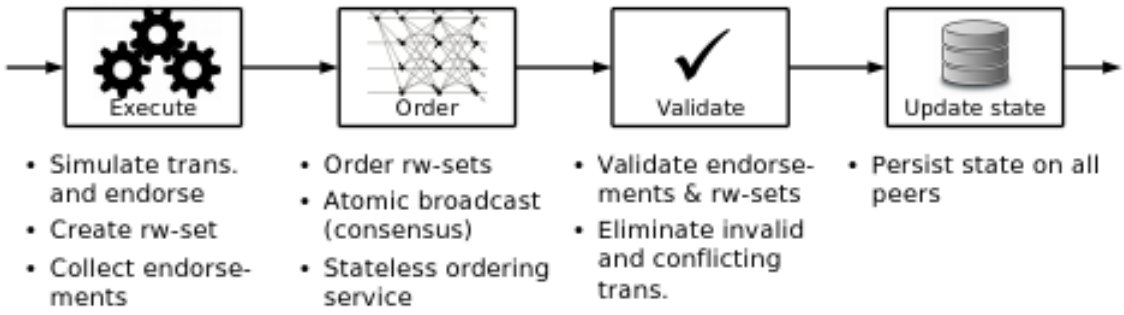

Figure 2.4.: The execute-order-validate architecture in HLF simulates the transaction in the endorsing nodes that create a readset and writeset representing the version dependencies and the state updates, respectively. Afterwards, the client sends the collected signatures to the ordering service, including the transaction in the next block and broadcasts to the network. In the final step, nodes in the channel validate the new block before updating the ledger and the respective data [4]. Reprinted from [4].

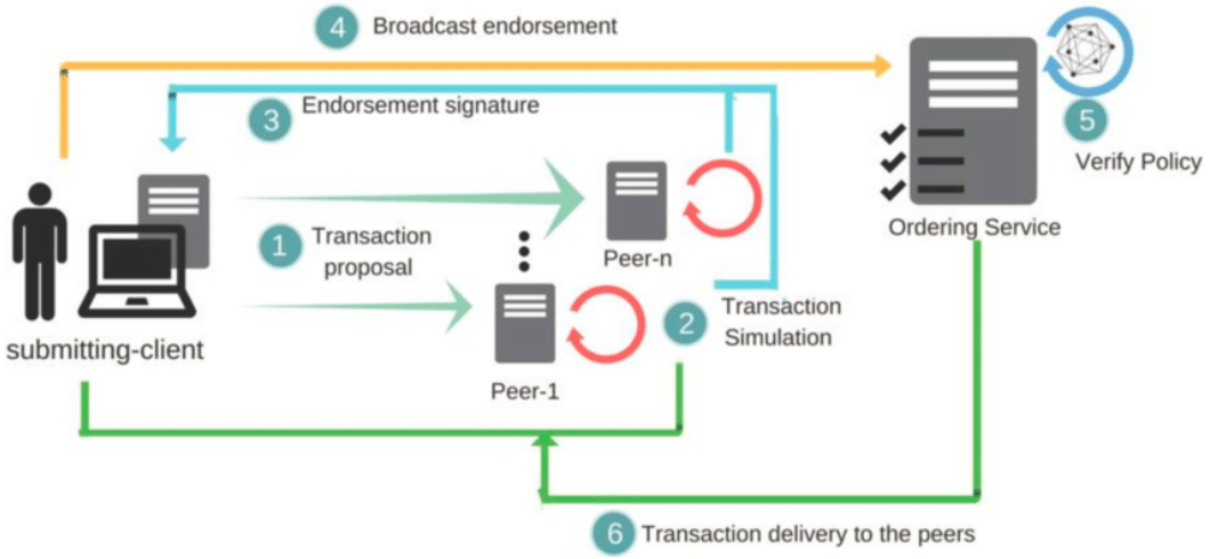

Figure 2.5.: In HLF, the client sends a transaction proposal to the endorsing nodes (1) represented by Peer-1 and Peer-n where n $\in \mathbb{N}$. The peers simulate the transaction and create a readset and writeset that are sent to the client with their respective signatures (3). Next, the client sends the collection of all endorsements to the ordering service (4), which verifies the consensus mechanism and endorsement policy by the given channel (5). Finally, the ordering service broadcasts a new block (6) including the transaction to all respective nodes and client [18]. Reprinted from [18].

A membership service provider (MSP) registers new nodes, as well as users. It creates the public and private keys that are used as credentials for authentication and authorization. Any transaction or message performed in the network is signed by the respective sender or node. The MSP maintains the identities and comprises a module at each node in order to authenticate and verify the integrity of transactions. In addition, this component signs and validates endorsements and is responsible for the validation performed in step three in the transaction flow explained in the previous paragraph. Finally, HLF can be integrated with different identity management systems and certification protocols making it flexible to a wide array of particular use cases. This DLT architecture aims at resiliency, scalability and confidentiality by following a novel paradigm for the distributed execution of code.

HLF differs from Ethereum as it allows the implementation of access control lists such that data access can be fine tuned to the respective data owners. Furthermore, it allows the creation of a private data collection that stores data in a separate database and includes only the hash value of the data in the transaction log. This ensures complete data privacy towards any other user except the data owner and the authorized organization. Consequently, businesses have the possibility to further enhance data privacy over their assets. For the future, zero-knowledge proofs are planned to be included by default, enhancing the current data privacy features of this technology [5].

2.3. Related Systems

There exist several DLT network applications for the exchange of data between different parties. The listing gives an overview of currently known applications used by companies and governments attacking the problem of securely exchanging health related data with DLT.

E-Estonia is the digital ecosystem for the Estonian government and their services, in collaboration with many diverse companies that allowed this country to become one of the world's most developed digital societies. The plan of the government is to digitize and automate basic services that are performed as e-services for their citizens as well as foreigners who applied for an e-residency. The IT infrastructure of Estonian's healthcare system allows their citizens, doctors, hospitals and the government to perform and deliver e-services. Each person in Estonia has an online e-Health record that can be accessed by their providers and tracked by the owner. This data is secured and access is granted by the electronic ID-card with the respective authorization. This allows data owners and authorized individuals to access critical information on the go. Keyless Signature Infrastructue (KSI) Blockchain technology by Guardtime is used to mitigate internal threats to the data and to ensure data integrity. Guardtime is an Estonian company, a team of over 150 cryptographers, developers and security architects with several decades of experience in cybersecurity. The KSI blockchain technology protects against data manipulation and guarantees data integrity in order

to guard data against any manipulations. The Estonian E-Health Foundation started the development to safeguard patient health records with blockchain technology in 2016 [26]. Today, the healthcare e-solutions are maintained and improved in collaboration with Nortal, Helmes, Guardtime, INTELSYS, Stacc and Quertec to ensure secure and reliable data storage and exchange [29, 41, 46].

MedicalChain is a London based company that uses blockchain to give patients access and full control over their health data. Patients grant access to their data through their personal mobile device and give healthcare providers immediate access on request. Their proprietary solution stores non-personal information that represents the pointer to personal encrypted health data stored in healthcare and research facilities as well as non-profit organizations complying with the regulations. Data owners can authorize access and allow the view of this data on the company's proprietary mobile interface. In addition, the incorporation of telemedicine for consultations and wearables for monitoring allow healthcare providers and patients to track the progress which facilitates the remote monitoring and communication. Medicalchain is built using a dual blockchain infrastructure with HLF and Ethereum as core technologies. HLF implements an access control list to allow data owners full disclosure, control and privacy. Ethereum incorporates the ERC20 tokens for the remuneration of services in the Medicalchain ecosystem. Identity management is performed in collaboration with Civic, which provides the user authentication services for this platform. Medicalchain guarantees the privacy of the sensitive data at all times and each interaction is secure, transparent and auditable to comply with the regulations in this industry [70, 71].

HSBlox and R3, two software companies, announced in June 2018 the launch of a blockchain initiative for the healthcare industry using the R3 Corda's Healthcare Community network. HSBlox provides patient-centric solutions by combining DLT and machine learning for the secure and real-time information sharing of medical data. R3 is a blockchain software company focused on the financial aspects of complex and highly-regulated markets such as the healthcare industry. R3 created the open source blockchain platform, Corda. Together, they work on the improvement and further development of Corda for the automation of multi-party transactions and secure data exchanges in compliance with the regulations in this industry. Patients are put in control of their data across multiple providers and smart contracts automate the data exchange in immutable records. Insurance companies can verify claims and automate processes for billing with patients and providers in compliance with their respective regulations. Finally, providers can exchange and access the necessary information by requesting access to the data owners with the help of Corda's global network, the Corda Healthcare Community [23, 50, 51, 74].

Iryo is a Slovenia-based enterprise creating a zero-knowledge health record storage platform that provides an interface for anonymous queries. EOS blockchain is used for the access control of health records and tokens are used as consensus mechanisms [13]. Their solution consists of an open platform using open standards to facilitate interoperability and transparency. The vision of Iryo is to build an ecosystem that allows the maintenance of a unified health record and takes the openEHR's approach for data modeling and exchange [73]. Their solutions allow users complete

control and access over their medical data secured and stored with cryptography and zero-knowledge. The platform is run in the cloud and data is stored securely using asymmetric key encryption. The owner keeps the private key used to encrypt personal data. Furthermore, the owner provides access to doctors with a re-encryption key added to the doctor's public key that is logged in the blockchain acting as an access control list [41, 60, 61]. In October 2018, Iryo cancelled their plans for an initial coin offering (ICO) and it is unclear if Iryo is still operational [62].

2.4. Related Work

This section provides an overview of related academic research papers and scientific books that deal with and provide a basis for the understanding of DLTs in the context of secure data exchanges. This section is organized in three separate parts namely: *Models*, *Security Threats* and *Optimizations*.

2.4.1. Models

Several countries around the world opened research facilities to investigate the possibilities of DLTs beyond the primary intention as a replacement of a fiat currency. These areas include bureaucracy, identity management and healthcare. The models introduced in this subsection present approaches and solutions for a secure data exchange among participants in the healthcare industry.

2.4.1.1. Ecosystem for Big Data Security

In [89], **Uchi Ugobame Uchibeke, Kevin A. Schneider, Sara Hosseinzadeh Kassani and Ralph Deters** showcase a software architecture and prototype of a blockchain access control ecosystem for data exchanges on top of HLF. In their work, they address several big data security and privacy issues that include

- data privacy preservation,
- authentication and authorization,
- identity and access control management,
- data ownership, and finally,
- policy management.

The data architecture uses a NoSQL database for both storing and querying in order to account for the semi-structured and unstructured data found in the healthcare industry. They suggest a blockchain access control ecosystem for the access control management for the data owners. This allows users to request, grant, revoke, verify and view transactions as well as assets based on the respective authorization.

They propose two different access control settings, one that is based on the identity and the other on the role of the current user. They explain how such data exchanges are performed in the respective setting. Figure 2.5 gives an example of an identity-based access-control solution. The application relies on HLF as the base layer with a middleware on top that acts as the server for the authentication before accessing the data stored in the ledger. The middleware represents a RESTful server created by the Hyperledger Composer tool set. The software architecture consists of a three-tier architecture that is natively provided by the Hyperledger development tool set.

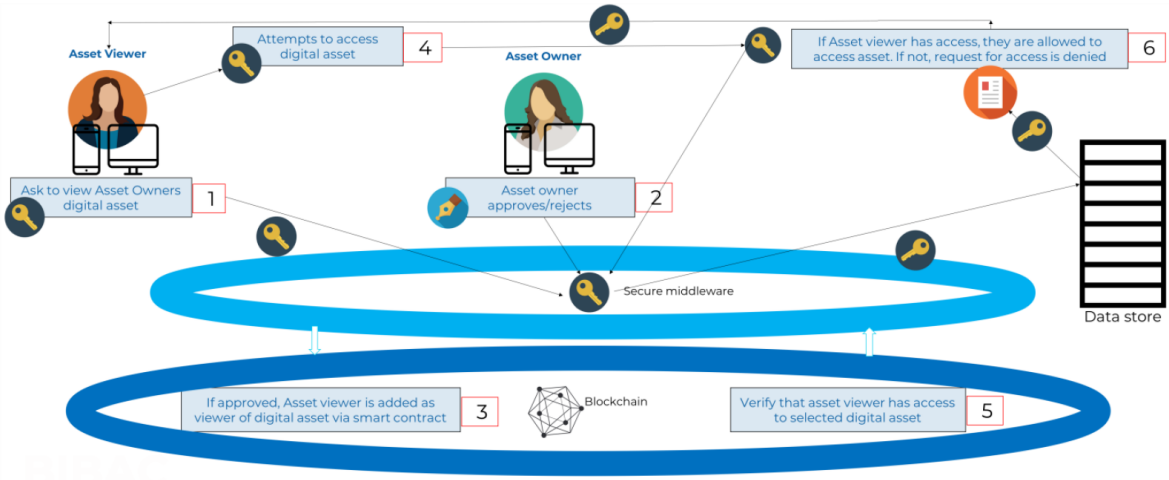

Figure 2.6.: In a blockchain identity-based access control ecosystem, each entity and node has a key with which transactions and further operations are signed. A user can request the access of a certain asset by sending a transaction that represents this request (1). The asset owner can view, grant or revoke the access (2). In both cases the transaction is recorded onto the ledger (3) and based on the result, the requester is given access to the asset (4) that is verified in HLF (5) before returning the requested data asset (6) to the respective user [89]. Reprinted from [89].

2.4.1.2. MedRec

In [27], **Ariel Ekblaw** presents MedRec, which is a decentralized record management system for electronic medical health records using blockchain technology. Her solution manages authentication, confidentiality, accountability and data sharing using a modular design, such that existing electronic health records can be integrated, which facilitates the interoperability between legacy systems. MedRec runs on top of the

Ethereum network and each node can be represented by a healthcare provider or a patient who controls the data in this ecosystem. MedRec addresses the problems with slow access to fragmented medical data, system interoperability, patient agency and finally data quality and quantity for medical research.

The architecture of MedRec consists of three smart contract modules that define how data is organized giving the control to the patient. The Registrar Contract (RC) maps participant identification strings to their Ethereum address. The Patient-Provider Relationship Contract (PPR) lists which nodes store and manage health records for the respective nodes. And finally, the Summary Contract (SC) functions as a repository for participants to localize their respective data [7].

A third party service provider stores encrypted data on a distributed hash table (DHT) that points to the data held by the user on its local device. A decentralized personal data management system allows users to own and control their data on their personal devices. They apply a protocol, that turns a blockchain into an automated access-control management system that ensures data ownership, transparency and auditability, as well as fine-grained access control [96].

In 2016, MedRec was tested as a web application on top of a private Ethereum network at the Beth Israel Deacon Medical Center (BIDMC) in Boston, Massachusetts [27, 28]. The test never touched any live medical records nor personally identifiable information and provided a learning opportunity for the team. In order to simplify the integration test, the scripts were modified to only expect medication records. In addition, the database columns in MedRec were adapted in accordance to the BIDMC's data schema. The integration test was accompanied by several technical problems such as the non-functioning of DHCP and bridging mode, which forced the team to revert the system to use static IP. Moreover, a simple escape character on one of the SQL server's password characters caused several connectivity issues. Finally, the test did not make use of over 200 records resulting in the separate patient-provider contracts for the respective data. Several key areas for improvement were identified such as:

- run-time performance: several seconds are needed to assign new contracts and respond with the contract information for the update of the logs;
- ether-generation mechanism: users run out of Ether and cannot perform any updates nor readings;
- maturity: the implementation needs further iterations and a complete review with a trained system architect and security consultant.

Unfortunately, problems concerning privacy and anonymity remain unresolved as the identification of interactions of entities can easily be inferred, as well as the identities of healthcare providers. In addition, several regulations are not completely integrated into the prototype and a global ID system needs to be in place in order to interconnect given data to a patient.

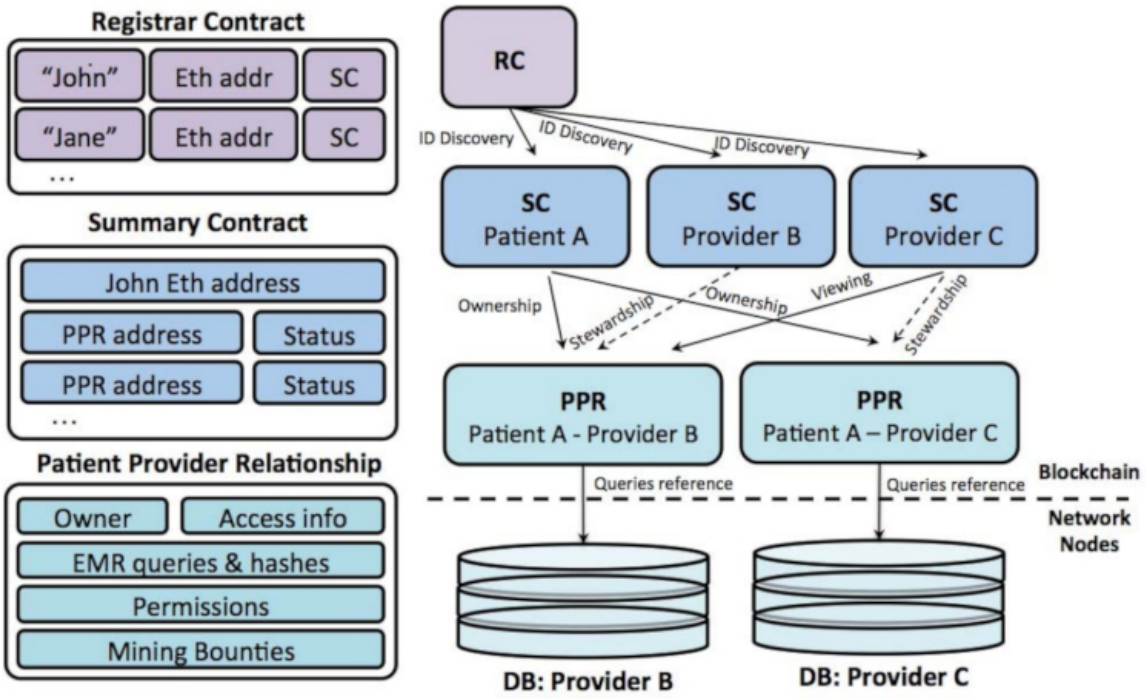

Figure 2.7.: The architecture of MedRec runs on top of Ethereum using several smart contracts for the data management and exchange. Three smart contracts control how data is organized. The Registrar Contract (RC) lists the identity of the participant to their Ethereum address. The Patient-Provider Relationship (PPR) Contract lists which providers store and manage health records for the respective patient. The Summary Contract (SC) maps all data managed by a provider to the respective patient and its respective status such as public or private [7, 27]. Reprinted from [7].

2.4.1.3. Hawk

In [66], **Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou** show how a decentralized smart contract system can be built using their smart contract compiler named Hawk, such that financial transactions are stored as encrypted entries in the ledger. The compiled smart contract does not store financial transactions in the clear, and consequently protects transactional privacy in a public ledger unless the contractual parties do disclose such information themselves. This transactional privacy is based on sending encrypted information to the blockchain and relies on zero-knowledge proofs to ensure the correctness of contract execution and money conservation. Current distributed ledger technologies lack such a privacy feature. Unfortunately, this solution does not allow for auditability, which is mandatory for insurance companies, healthcare providers and governmental institutions. Nevertheless, Hawk allows programmers to write smart contracts without having to deal with the implementations of cryptographic tools. The Hawk compiler generates an executable, which includes the cryptographic protocol between the blockchain and the users. This allows contractual parties to interact with the blockchain without disclosing their identity nor transactional information to the public.

They assume that the generalized consensus protocol is secure and that the blockchain can be trusted for correctness and availability, but not for privacy. They use a modified version of ZeroCash to achieve stronger security, which allows two parties to transfer a value among them without disclosing their identities to each other nor the amount transferred to external observers [78].

By interchanging financial data in a transaction with sensitive data representing a link as a string, a secure data exchange can be achieved arriving at the same conclusion as stated in the paper. These findings are useful in the healthcare setting in the creation of a web interface for the transaction of sensitive data.

2.4.1.4. Ekiden

In [17], **Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song** present Ekiden, a system that combines DLTs enabling smart contracts with TEE. This system allows smart contracts to run on any system without compromising confidentiality and security. TEE provides the necessary security features such that the application runs as an enclave on any system by protecting its data from any other application and operating system. Furthermore, this solution is applicable on any desired DLT. Ekiden helps to keep sensitive data confidential and smart contracts are endowed with strong confidentiality and integrity guarantees by using a hybrid architecture combining trusted hardware and the DLT. There are three types of entities when using this system: *clients*, *compute nodes* and *consensus nodes*.

Healthcare related data is mostly regarded as sensitive data which needs to be kept confidential if we want to enable the execution of smart contracts on such data. Even

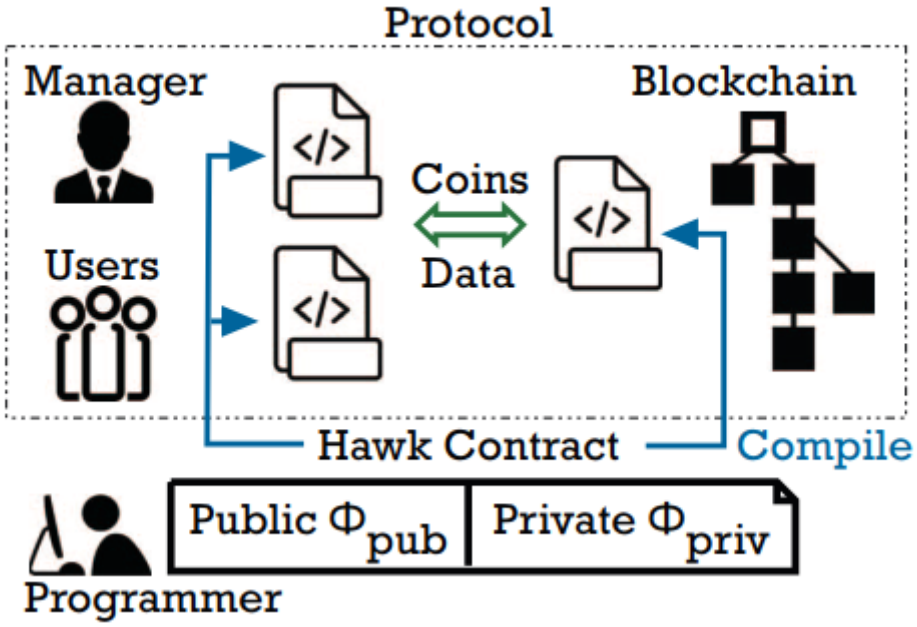

Figure 2.8.: The Hawk compiler generates a cryptographic protocol between the blockchain and the users. This protocol represents the program, which consists of a private Φ_{pub} and public Φ_{priv} component. The public component Φ_{pub} acts as the application interface. The private component Φ_{priv} takes the input data of all parties and performs the computation. This private component is meant to protect the transactional data and the exchange of value. The manager is a special facilitating party that has access to the inputs and is trusted not to disclose any private data and supervises the correct execution of the smart contract, such as in auctions [66]. Reprinted from [66].

if the system and the corresponding nodes are compromised, Ekiden protects the data from malicious activities. They used several models to evaluate Ekiden, as well as a machine learning model for detecting heart diseases in patients. They showed that the smart contracts can be used to train a shared machine learning model with sensitive data without exposing any plain text training data. Nevertheless, nodes may shutdown; therefore, the availability of such smart contracts cannot be guaranteed on any node.

2.4.1.5. Solidus

In [15], **Ethan Cecchetti, Fan Zhang, Yan Ji, Ahmed Kosba, Ari Juels, and Elaine Shi** present Solidus, a protocol that keeps transactions confidential on public DLT implementations using the concept of a Publicly-Verifiable Oblivious RAM Machine (PVROM). In the setting of the financial banking sector with several banks representing a modest number of clients, a client can request a secure transaction that remains confidential in a public ledger. The bank will perform the transaction on behalf of their clients without disclosing the identity of neither the sending nor receiving client represented by the other bank. The transaction is stored on a permissioned or private ledger shared by the participating banks. The bank internally maps the identity of its client with the identity used in the public ledger, without disclosing the true identity of its client to external observers nor other banks. Solidus therefore allows auditability that is mandatory for financial institutions without impacting the confidentiality towards their clients.

In a public ledger, a secure data exchange between participants needs to remain confidential in order to comply with the current data protection laws for highly sensitive data. By interchanging banks with healthcare providers and insurances, we can establish a similar scenario without having to heavily modify the protocol. If such a given infrastructure exists, insurance companies can be transformed into decentralized autonomous organizations allowing for a more cost efficient healthcare system in the future.

The amalgamation of Solidus with Hawk and Ekiden allows for a highly secure infrastructure for the secure data exchange of health related data. For this purpose, a public DLT such as Ethereum needs to modify several configurations that include TEE on every node in the network, application programming interfaces to interact with other parties and network settings that are accompanied by best coding practices. The degree of these modifications are proportional to the investment expenditures, time and complexity of the endeavour, which represent a barrier for the healthcare industry. Nevertheless, these findings show that a secure data exchange is theoretically possible using a DLT such as Ethereum.

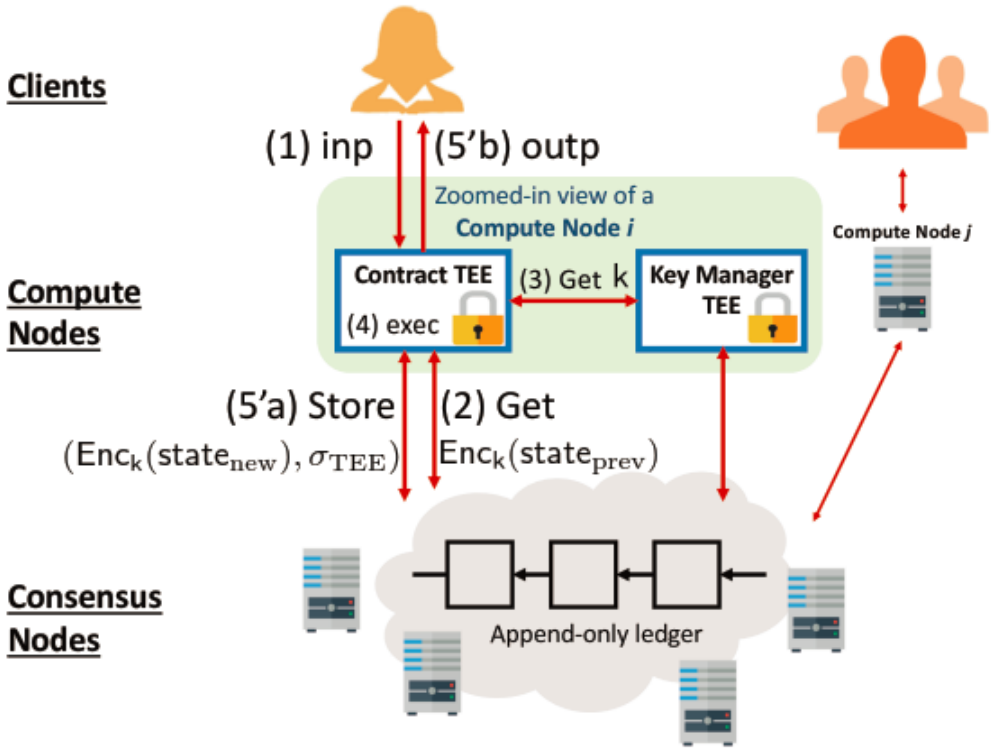

Figure 2.9.: In the architecture of Ekiden, there are clients, compute and consensus nodes. Clients send inputs (1) to smart contracts, which are executed within a TEE (4), preserving the confidentiality of the input on a compute node. The contract within the TEE retrieves the respective data from the ledger (2) in order to process the client's input. The key manager running within a separate TEE stores and provides the keys associated with the respective smart contract (3). The compute node executes the smart contract with the client's input. The smart contract stores the output encrypted with its respective keys in the ledger (5'a) and returns the output for the client encrypted with their respective public keys (5'b). The consensus nodes verify that the attestation σ_{TEE} is correct before appending the associated output to the ledger, which stores the encrypted contract state [17]. Reprinted from [17].

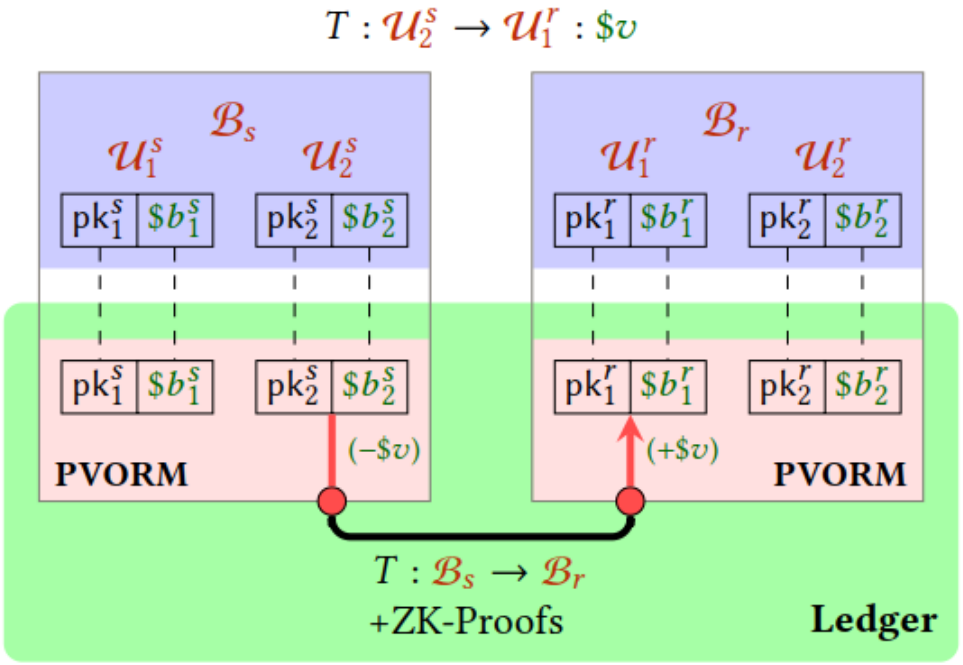

Figure 2.10.: This is an example transaction T in Solidus, where B_a represents a bank among a set of $a \in \mathbb{N}$ different banks and U_i^a represents a user of a bank B_a, which oversees $i \in \mathbb{N}$ different users. A user U_2^s at a bank B_s sends the amount of \$v $\in \mathbb{R}_{\geq 0}$ dollars to user U_1^r at the bank B_r. Both banks represent two users that are stored in a logical (plaintext) memory of each bank's PVORM represented as blue boxes and the lower pink boxes are the associated public (encrypted) memories. External observers only see that a user at a bank B_s sent money to a user at a bank B_r but nothing about the sending nor receiving user. After a successful transaction, both banks update their respective PVORMs correctly [15]. Reprinted from [15].

2.4.2. Security Threats

In [69], **Li Xiaoqi, Jiang Peng, Chen Ting, Luo Xiapu and Wen Quaoyan** systematically examine the security risks of DLT network applications such as Ethereum and Bitcoin. They divide the common risks into nine categories represented by

- 51% vulnerability,
- private key security,
- criminal activity,
- double spending,
- transaction privacy leakage,
- criminal smart contracts,
- vulnerabilities in smart contracts,
- under-optimized smart contracts and
- under-priced operations.

In addition, they explain and analyze attack cases in order to understand further security flaws in current DLT systems such as

- selfish mining attacks,
- DAO attacks,
- Border Gateway Protocol (BGP) hijacking attacks,
- eclipse attacks and
- balance attacks.

This systematic overview of security threats with current DLT using PoW and PoS as consensus mechanism shows that a wide variety of security aspects need to be addressed. Ethereum is susceptible to several of these threats for which preventive measures need to be implemented to allow secure data exchanges in the healthcare industry.

In [32], **Ittay Eyal and Emin Gün Sirer** demonstrate that the distributed protocol such as Bitcoin is not incentive-compatible and how an attack on the public ledger can successfully be centralized by a minority pool of colluding miners. These colluding miners need to run a modified bitcoin protocol that would favor their own mined blocks and ledger. By applying a stochastic strategy, these miners can put themselves into the most favorable position to get the rewards for mined blocks. They keep mined blocks secret as long as they are ahead of the public ledger and ignore any block mined outside their pool. By publishing their private branch, they are able to invalidate the public branch as new blocks are only appended on the longest chain, and thus claim the rewards for all privately mined blocks in the ledger. In case their branch is equal in length to the public branch, the colluding miners simply publish their branch hoping

that the majority adopts their previously private branch instead of the public one for the appendage of new blocks, and thus reap the rewards for the selfish pool. This strategy allows a minority of miners to reap a disproportionate amount of rewards and moreover, giving incentives to honest miners to join their pool further centralizing the mining hash power and consequently destroying the decentralized nature of Bitcoin. When these colluding miners reach the majority in the network, no new members are accepted making Bitcoin a centralized cryptocurrency.

2.4.3. Optimizations

In [10], **Shehar Bano,Alberto Sonnino,Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis** give a systematic overview of consensus protocols used in DLTs and show alternatives to the Proof-of-Work (PoW) consensus mechanism. These alternatives are summed up under the term Proof-of-X (PoX) and are presented next to hybrid consensus schemes that rely on single or multiple committees with different advantages and disadvantages.

The healthcare setting is already working on a given infrastructure with given laws and regulations, why PoW is not the best choice available when implementing a DLT solution, especially when trust in the infrastructure is enforced by law. Therefore, other consensus schemes may be preferred that can be implemented on top of the existing infrastructure.

In [16], **Melissa Chase and Sarah Meiklejohn** explain the implementation of a DLT without the need to store a complete ledger for each end user, which allows the elimination of hash-based mining. Their proposal relies on the assumption that one is willing to adopt a distributed, rather than a fully decentralized, DLT network by trusting any given set of nodes. Under this assumption, a certificate transparency (CT) solution is applied on top of a DLT, where certificates are issued and appended to the ledger as *dynamic list commitments*. This list represents a Merkle tree with committed events stored as certificates. This transparency overlay includes *log servers* (stores events as certificates in a publicly available Merkle tree), *auditors* (checks that specific events are in the log) and *monitors* (flags any problematic entries in the log). This allows users to request transactions without having to store the complete ledger. Users directly communicate with the log servers and act concurrently as an auditor and monitor. Consequently, users can autonomously check for correctness and finally ensure accountability in a distributed system.

Their proposal allows the implementation of an energy efficient DLT without the need of hash-based mining by using the underlying infrastructure that runs the healthcare setting in each country. Hence, this approach reduces the costs for maintaining a DLT and allows clients, insurance companies, as well as healthcare providers to check for the correctness of the ledger without having to rely on a central authority for bad certificates.

3. Analysis of the Healthcare System

The first section, *Health Ledger*, describes the vision of DLT in the healthcare industry for the management of health records. The second section, *DLT in the Healthcare Industry*, presents several business use cases in the healthcare industry using DLT for the data management and exchange. In contrast, the third section, *Data and Processes*, gives an overview of current data exchange procedures in the German healthcare system among three target groups represented by clients, healthcare providers and insurance companies. The fourth section, *Security*, complements the former findings with security aspects and represents the primary focus of this thesis. The fifth section, *Laws and Regulations*, lists current policies and regulations on medical data and analyzes the constraints for the data exchange. Finally, the sixth section, *Requirement Analysis*, summarizes the results of the requirement analysis with three representatives of the German healthcare industry.

3.1. Health Ledger

Imagine a world of distributed ledgers, where people possess their own little database with their own data, giving it to organizations to use when and where needed. This fundamentally reverses the current dynamic. In such a scenario, health records reside in their health ledger and different healthcare providers can access and update that single record, but only with the permission of the end user as the data remains theirs.

With DLT, there is no central facility nor healthcare provider that accumulates data and impedes their access with slow storage media such as paper. Transparency and security can be improved as data owners and participants are able to audit and verify the ledger. In addition, data owners are in possession of their own data with the power to grant and revoke their access. Healthcare records and medical data become accessible to data owners and authorized entities such as family members, healthcare and research facilities in a world of distributed ledgers. Medical histories from parents can be passed to their children and grandchildren such that genetic components can be found and treated early on without ever having to suffer from such a disease later in life. Healthcare providers can offer better treatment options and improve the quality of care. Research facilities are able to extend their database and generate more accurate results. And the best part is that all is stored in a public ledger that is available to all users without compromising security and privacy of the data owners.

Nevertheless, with every new technology, new investments have to be made. In addition, users need to understand how this technology works and what they can do with it. Paper is tangible, familiar and the de facto standard with how citizens proof their claims to institutions and healthcare providers. Culture and formalism is only

one aspect, because there exist further challenges in regard to the adoption of DLT as data exchange grid such as

- a public immutable ledger in the internet
- that is accessible to any authorized party.
- Public/private key encryption may fail
- or get lost making the data inaccessible.
- The huge size of the ledger
- that is redundantly copied to all users.
- Laws and regulations need to be met,
- which are expensive.
- Consequently, further research needs to be performed in this field
- without a guarantee of a return of investment.

To sum up, transparency and accessibility of data can be improved. Moreover, there is no single point of failure as data is replicated in the system. On the other hand, there exist several tensions such as the compliance with the GDPR and the immutability of the ledger. There are many advantages but also disadvantages and the question remains if a secure data exchange is possible among different parties using a new technology such as DLT.

3.2. DLT in the Healthcare Industry

Today, several countries and companies embrace DLT in their environment. In some countries, cryptocurrencies are accepted as a payment option for governmental services. In other countries, DLT ensures data integrity and helps to mitigate cyber attacks on their infrastructure. The influence of DLT is slowly and steadily invading aspects of our society and its use cases reach far beyond the initial intention of being an alternative to fiat currencies.

As of 2019, Estonia remains the only country in the world to have implemented DLT on a national scale for their e-health services. KSI Blockchain technology by Guardtime is currently used to mitigate internal threats to the data and to ensure data integrity of medical records. Guardtime is a general member of the Linux Foundation and actively contributes to the development of the Hyperledger Project. This company provides products and services, such as Guardtime Health, which allows the cross-collaboration of participants in the healthcare industry and transport of data through a secure data exchange grid on top of DLT [45]. In July 2019, the company announced a partnership with the Estonian Biobank to deploy Guardtime Helium. This project

intends to implement a comprehensive blockchain-based data access and management system for medical data [44]. This company shows that DLT can be used to exchange and manage data in the healthcare industry.

In Great Britain, Medicalchain uses blockchain to give patient access and full control over their health data by using Ethereum and HLF. They incorporated ERC20 tokens in Ethereum that are used for the remuneration of services such as remote consultations through telemedicine and data exchange services. The data management is performed by HLF, which clients can use from their personal mobile device.

Furthermore, the research community analyzed several use cases, on top of Ethereum and HLF, that resulted in the creation of interesting prototypes. MedRec is the result of a master thesis at the MIT written by Ariel Ekblaw and represents a health record management system on top of Ethereum. With MedRec, patients and healthcare providers can interact with each other using smart contracts and encrypted links stored in the ledger to access the data. Another paper presented by a team of researchers from the University of Saskatchewan showcased a similar prototype on top of HLF.

To sum up, these examples indicate that Ethereum and HLF can be used for the exchange of medical data. Nevertheless, business use cases show that HLF is used more extensively in the healthcare industry, whereas Ethereum primarily handles financial transactions in the form of tokens. Both technologies are backed by many influential companies and corporations that actively contribute to their development and signal their trust in these technologies. Today, HLF is the only DLT that provides a long-term support for its current version 1.4, whereas Ethereum is undergoing several changes with the upcoming hard fork named Istanbul [2, 39].

3.3. Data and Processes

An analysis of current data exchange procedures in the German healthcare industry is performed in order to design a suitable prototype using DLT as core technology. For this purpose, clients, healthcare service providers and a health insurance company were interviewed. In addition, further insurance companies, as well as public and private institutions were contacted to supplement and clarify the gathered findings. The nature of exchanged data is examined and categorized accordingly in order to establish a data model. The interactions among participants are analyzed in order to elaborate the priorities and the tasks for the data exchange. The results from this analysis show that data exchanges are time-consuming and inefficient, as they are generally performed on paper. Furthermore, healthcare service providers and health insurance companies are in constant need of data to perform their tasks, and consequently, invest a lot of time to retrieve and maintain the respective data.

3.3.1. Data

The healthcare industry is run by a diverse set of non-structured data that consists of pictures, videos, diagrams and text. Clients, healthcare service providers and health insurance companies need data to perform, provide and receive services, that is obtained and maintained according to their needs. Table 4.1 gives an overview of the data types and the need for each target group.

Health insurance companies and healthcare service providers need the following data to provide their services, such as

1 identifiable information including
   - name,
   - birth date and
   - contact details,

2 and non-identifiable information such as
   - current health status and
   - diagnosis, if available.

Data needed by Clients, Providers and Insurance Company			
Data Type	Client	Provider	Insurance
Name	YES	YES	YES
Birth Date	NO	YES	YES
Sex	NO	YES	YES
Contact Details	YES	YES	YES
Bank Details	YES	NO	YES
Insurance Policy	YES	YES	YES
Health Status	YES	YES	YES
Diagnosis	YES	YES	YES
Medical Report	YES	YES	NO
Imaging	YES	YES	NO
Drug List	NO	YES	NO

Table 3.1.: This table gives an overview of the data needed by the client, provider and insurance company.

Health insurance companies need the bank details of their clients and healthcare service providers for payments and direct debiting. The payment of medical bills on behalf of their clients are based on the diagnoses and associated bill provided by the attending healthcare service provider. The insurance policy defines the fraction of the amount that the insurance company pays on behalf of their client. Frequently, budget

estimations are needed from their client for the settlement of bills before the execution of the treatment plan. For the registration of new clients, they usually collect information from previous insurance companies, especially in case of a private insurance policy. In specific cases, they need a medical expert opinion on their clients that are commissioned on behalf of the company as well as summaries from the performed medical imaging.

Healthcare service providers require the public insurance card or the details from a private insurance company. Moreover, providers are focused on collecting data that are related to the current medical condition that is categorized as health related data. Important health related data include diagnosis, current medication, allergies, infection status, medical reports, lab results and if available previous imaging.

Clients usually do not collect identifiable information to such an extent, but still need the contact details of the healthcare service providers and insurance companies. This information is usually found in the internet and thus not collected nor maintained systematically. On the other hand, they collect and store prescriptions, sick certificates, lab results, medical imaging and reports providing the diagnosis, treatment plan and other health related information. Finally, the insurance policy, e-Health card and letters are needed for the settlement of bills and treatment plans with the health insurance company.

To sum up, all three groups need to collect the contact details in order to interact with each other, but clients collect and manage their information less systematically compared to healthcare service providers and insurance companies. Provider and insurance companies have to comply with more stringent regulations and perform more complex tasks with the data than clients. Moreover, healthcare providers and insurance companies need more information from the client than vice versa.

3.3.2. Processes

All three groups heavily rely on paper to perform data exchanges. In general, paper is used by mail or fax for any data request, which are then supplemented by phone calls and available digital channels. The frequency of interactions is especially high between healthcare service providers and insurance companies, as well as within both groups. On the other hand, the frequency is relatively low between clients and healthcare service providers or health insurance companies, as well as among clients. The storage of information is primarily done on desktop computers and servers within healthcare service providers and health insurance companies. Clients primarily store documents in a folder.

Healthcare service providers and clients exchange information during consultations and emergencies, if the client is conscious and oriented. Phone calls between healthcare providers and clients are used to communicate lab and further results from

other diagnostic procedures. In addition, documents are usually sent by mail or delivered personally during a consultation. These interactions are infrequent and increase with client's age and worsening health condition.

Health insurance companies and clients are in contact by mail, phone calls and digital channels through proprietary mobile apps and web applications. Clients usually request an update of their contact or bank details and the acceptance of a budget estimation or treatment plan provided by their attending healthcare service provider. In addition, insurance company branches allow for the discussion of personal concerns with a representative in person.

Healthcare service providers and health insurance companies are in constant exchange of data, which is primarily focused on settlements of bills and treatment plans. This is primarily performed on paper using the fax machine and mail, as well as through established digital communication channels in upgraded facilities. In addition, expertise and summaries of performed medical imaging are usually sent by mail on behalf of the insurance company.

Within each target group, we have to differentiate between clients and the other two groups. Clients do exchange information among other clients, but this is primarily done orally and through common messaging services such as WhatsApp. The frequency of interactions is moderate and usually performed for entertainment purposes. On the other hand, the communication and exchange of data within healthcare service providers and the health insurance company is primarily performed through digital communication channels. Interestingly, within healthcare service providers, the exchange of information usually falls back on phone calls and fax machines when medical reports are requested from external facilities. The degree of interactions within these two groups is very high, and to a certain degree, standardized using digital communication channels.

To sum up, among the three target groups, paper is primarily used for the exchange of data, but within each group digital communication channels are in place. The main storage medium are desktop computers and servers for healthcare service providers and health insurance companies. Clients, on the other hand, usually store the documents in folders.

3.4. Security

A DLT solution for the healthcare industry needs to address fundamental security threats that are the primary focus of this thesis. Aspects such as authentication, non-repudiation, privacy, confidentiality, and data integrity are fundamental requirements for the healthcare industry. For this purpose, the security parameters need to be clearly defined for the evaluation of the chosen DLT and the resulting prototype. The security parameters are represented by the following requirements:

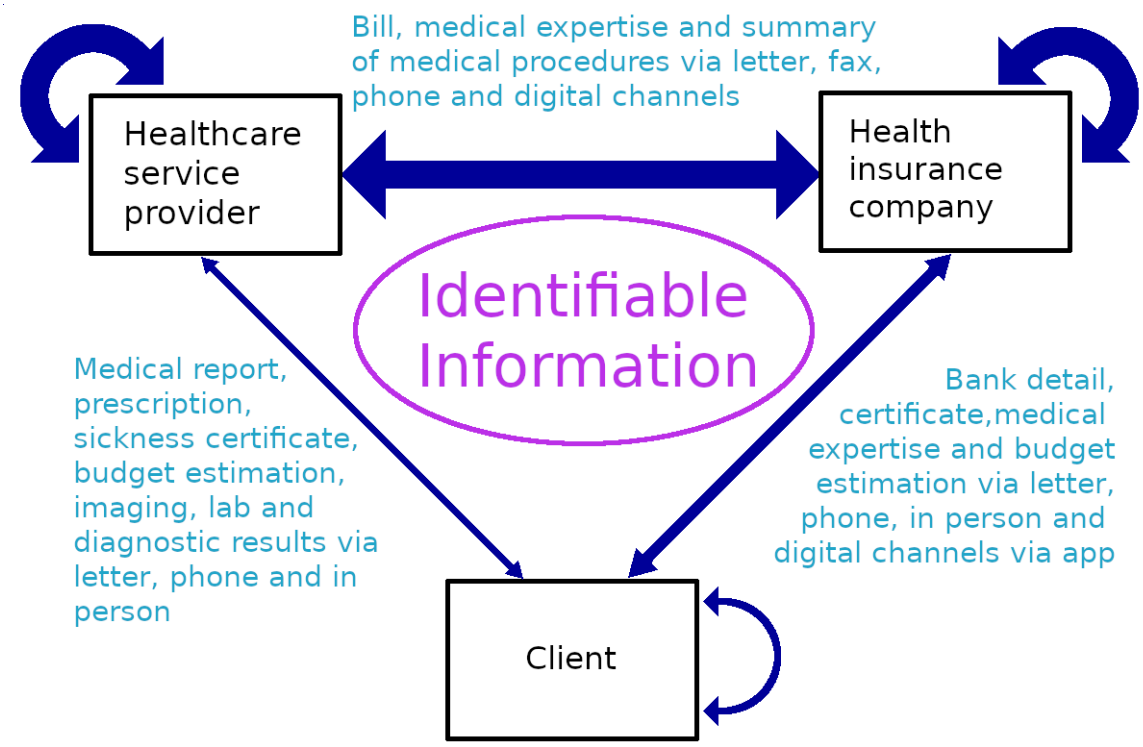

Figure 3.1.: The frequency of data exchanges is represented by the thickness of the respective arrows. Data is primarily exchanged among, and within, healthcare service providers and health insurance companies. On the other hand, the frequency with, and among, clients is low in comparison. All three groups require identifiable information such as contact details to exchange data.

S1: All participants undergo a registration procedure and receive a certificate used for the interaction in the network.

S2: Access is granted to registered and authenticated users and denied to anyone else.

S3: Participants are identifiable, such that nonrepudiation is not possible.

S4: Privacy of their identity and data is ensured.

S5: All interactions and data exchanges among participants are confidential.

S6: Data integrity is ensured and any non-authorized modification visible and restorable.

3.5. Laws and Regulations

In 2016, the European Union approved the General Data Protection Regulation (GDPR), which was enforced on May 25, 2018 [19]. This regulation applies to all businesses in the European Union, and consequently, the healthcare industry as well. This set of rules provide the citizens more control over their personal data and state that

- privacy and protection aspects are ensured by default.
- Second, businesses need to inform their clients about the purpose of the data collection, data transfers and how the data is processed. To all of these steps, citizens have the right to consent, contest or make further modifications.
- Third, businesses need to inform their clients in case of a data breach.
- Fourth, citizens have the right to access and move their data to another media platform.
- And finally, citizens have the right to be forgotten, and consequently, request the erasure of their data collected by the company with clear safeguards.

In addition, healthcare service providers and health insurance companies need to comply with additional laws and regulations such as the German Data Protection Act [63], and in particular, the Medicinal Devices Act [64] for software applications. In other words, there exist several laws and regulations that apply to the German healthcare industry that go far beyond the scope of this thesis. Consequently, this thesis focuses on the compliance with the fundamental subset of the GDPR and omits additional laws and regulations.

An infrastructure based on DLT allows clients, healthcare service providers and health insurance companies to access and exchange data without having to invest in new hardware. Moreover, it gives owners full control over their data. For this purpose, several aspects need to be analyzed:

- **Data Privacy**
  How to impose privacy and confidentiality in a ledger that each user in the ecosystem has access to and how to ensure that associations between parties cannot be deduced?
- **Data Exchange**
  How to secure the communication line between sender and receiver?
- **Access Authorization**
  How to grant or revoke access to data?
- **Data Modification**
  How to perform modifications on an immutable ledger?
- **Data Erasure**
  How to delete data stored in an immutable ledger?

A DLT implementation needs at least to comply with these aspects of the GDPR in order to provide a minimal viable solution. There exist several conflicts relative to data modifications and erasure on an immutable ledger and the GDPR, which need to be resolved in the creation of a prototype.

The following legal requirements conclude the analysis of this section:

L1: Privacy and protection aspects are ensured by default.

L2: Associations among participants are only visible to the network administrator.

L3: The data exchange is performed using a secure communication line.

L4: Data owner can grant or revoke access to data.

L5: Data owner can modify and delete corresponding data.

L6: Data owners can move their data to another platform.

3.6. Requirement Analysis

Interviews were performed with representatives of the healthcare industry. The analysis focused on three target groups, namely: health insurance companies, healthcare service providers and clients. These requirements are optional for the design and implementation of a user-friendly prototype. The questions cover their thoughts, frustrations of current data exchange procedures as well as expectations towards a new solution. During the interview, the representatives were expected to answer eight different questions in the following order:

1. What tasks have to be solved when exchanging health services related data with other parties and what ideas do you have concerning a solution?
2. Which problems and frustrations do you face along such data exchanges?

3 What health services related data do you consider to be the most sensitive?

4 And what of this data do you consider to be the most valuable for you personally?

5 In which cases would a new solution support you?

6 What security features should this new solution include: privacy; confidentiality; anonymity; data integrity; control?

7 What further features should this new solution include: openness; interoperability; reliability; accountability and auditability?

8 Should there be any exceptions to transfer certain data without your consent such as in emergencies, consultations among medical experts or insurance companies,and even family members or friends?

For the requirement analysis, five people were interviewed. The interviewees were represented by two clients and two healthcare service providers from the secondary care system and a representative of one health insurance company. All interviewees were between thirty and fifty years old. The following requirements summarize the survey data:

R1: Account settings allow the independent update of bank and contact details such as address and phone number.

R2: The identity of the user cannot be derived when examining the ledger.

R3: The prototype provides an overview of all available data to the user based on the identity and role such as client, healthcare service provider and health insurance company.

R4: The prototype provides an overview of all data requests from, and to, the current user.

R5: Data owners can instantaneously grant or revoke access to data.

R6: Authorized users can instantaneously access available data.

R7: All users are registered and authenticated in order to perform any exchange of data.

R8: Metadata on current user's data is generated only with explicit consent.

R9: Generated metadata is only accessible to the user.

R10: Sensitive medical data is accessible to all healthcare service providers given the user's explicit consent.

R11: Auditability and accountability features are implemented in the prototype.

R12: Essential sensitive medical data is accessible to healthcare providers in case of an emergency if and only if the user consented explicitly to provide such information in case of an emergency.

4. Design Decisions

This chapter merges the findings and requirements from the previous chapters for the design of the prototype. The first section, *Goal*, describes the primary focus and big picture of the functionalities of the prototype. The second section, *Decision on DLT*, provides a comparison between Ethereum and HLF, as well as the explanation for the decision behind HLF as core technology. The third section, *Resolving the Requirements*, describes the strategies and assumptions for the translation of the given requirements into the prototype.

4.1. Goal

The list of requirements provides an orientation on the priorities of the implementation of a prototype. Nevertheless, it is unrealistic to assume that all these requirements are implemented in the prototype. Consequently, this thesis focuses on the security aspects for a secure data exchange grid using DLT as core technology among representatives in the healthcare industry. The prototyping helps to increase the understanding of the possibilities, problems and constraints that this technology imposes.

The prototype represents a simplified version for the data exchange among and within the target groups in the healthcare industry. The proof of concept uses technologies that accelerates the implementation of a prototype. For this purpose, a web application allows the management of, and access to, data to all users represented by the clients, healthcare service providers and health insurance companies. The data in the healthcare system is reorganized according to the client such that all respective data can be managed by the client. Consequently, the databases among all participants need to be connected; a task which is performed using distributed ledger and web technologies. A client can request access to health records stored in the offices and facilities of various healthcare service providers. In addition, healthcare service providers can access or request data from any given patient in the system. The respective access can be managed by the client via the account settings and revoked at any time using the web interface. Finally, healthcare service providers can grant direct access to medical expertise performed on behalf of the health insurance company.

Healthcare service providers and health insurance companies act as registration authorities, and therefore register clients to the network with a unique ID, certificates and credentials for the log in. Users from all target groups access a web service that requires the authentication of the user. The logged in user should be able to store, delete, request, grant and revoke access to files using their unique ID in the system. A healthcare service provider and health insurance company can provide access to files that belong to a client. A client can request access to records held by healthcare service providers and vice versa. A user provides the ID to the attending healthcare

provider and vice versa to send requests. A user cannot be identified by a given ID, unless the true identity has been disclosed. For the sake of simplicity, it is assumed that the databases of healthcare service providers and health insurance companies are connected to the data exchange grid that is based on a DLT.

4.2. Decision on DLT

Ethereum and HLF are blockchain-based computing platforms or operating systems that include smart contracts or chaincode, respectively. Both technologies differ in their system architecture, design and features.

Ethereum is generally a public DLT where any user can read and make changes to the ledger. Participants in the network can remain anonymous and execute transactions without any restrictions as long as they can pay for it. All nodes in the network execute and validate every transaction stored in the ledger. The creation of new blocks are based on the chosen consensus protocol such as PoW, PoS or PoA. The smart contracts are written in a domain specific language called Solidity and need a cryptocurrency named Ether to be run.

In contrast, HLF is a permissioned DLT without an inherent cryptocurrency and not every node runs every transaction. Participants are authenticated and only allowed to execute authorized transactions. All transactions are signed by the certificates of the respective user. In addition, all nodes in the network are authenticated by the membership service provider and use the given certificates to sign executed transactions. This allows the creation of a network that consists only of partially trusted peers. The implementation of an access control list allows developers to restrict the access to the ledger and data stored in the network, which protects data from any unauthorized access.

The modular architecture of HLF allows system engineers to modify this platform based on specific trust models and use cases. There are several important components such as the membership service provider which maintains the identities of all nodes and participants. On the other hand, the ordering service is responsible for the creation of new blocks and the coordination of any update in the world state ledger. In addition, HLF enables several ledgers, on top of the world state ledger, that are bound to the chaincode and only accessible to peers that interact through the respective channel. This design allows transactions to run concurrently, which need only to be validated by a subset of peers. In other words, a client sends a transaction to a subset of peers and collects the signatures from the respective peers. The peers store the results from the execution without updating the ledger nor the data structures. The client sends these signatures together with the results from the simulated transaction to the ordering service. If the signatures and request meet the required conditions, a new block is created and broadcast to the network. This block is appended to the world state ledger shared by all peers and validated by the subset of

peers that executed the transaction. When the latter validation is successful, the respective ledger and data structures are updated and the transaction completed. This design mirrors the way businesses are run and requires less resources and energy to execute transactions.

Ethereum and HLF allow the exchange of data among participants, as well as automation of processes, but need different approaches. Businesses in the healthcare industry need to comply with regulations and keep sensitive data private. They are accountable for their actions and regularly perform audits on their business processes. Consequently, Ethereum represents a security risk as software may be run on hostile nodes. Moreover, anonymous participants can perform harmful transactions, causing financial losses to the business owners. In order to counter these risks, several layers of security need to be implemented as presented in subsection 2.2.1. These implementations cost time and money. In addition, Ethereum is currently undergoing several changes in their architecture and consensus mechanism such that efforts to adopt Ethereum as core technology should be postponed. On the other hand, the Linux Foundation announced their first long-term support distribution with Hyperledger Fabric v1.4 LTS on January 16, 2019.

Besides, cryptocurrencies have legal implications that need to be considered. In some countries, they are considered an asset and in others, a currency such that taxes apply. And in certain countries, cryptocurrencies are declared illegal. Thus, research on laws and implications needs to be performed for the adoption of Ethereum as a data exchange solution in order to avoid disastrous consequences. In sharp contrast, HLF does not need any cryptocurrencies and allows the circumvention of such legal implications [11]. Moreover, a native cryptocurrency for the execution of transactions imposes several constraints that need to be addressed. Otherwise, the execution of transactions are not possible, such as in the case of an insufficient balance as experienced along the field test of MedRec at the BIDMC in 2016 [27].

To sum up, HLF provides several privacy features that facilitate the fulfillment of several elaborated requirements along this thesis. In addition, there is less overhead for the implementation of data privacy. Its latest release provides a long-term support such that it is more fruitful to develop on this platform compared to Ethereum, which is undergoing several changes in its architecture making previous efforts potentially obsolete. Consequently, the design of the prototype focuses on the implementation of HLF as core technology for the secure data exchange grid.

4.3. Strategies

The primary goal of this thesis is the evaluation of the security parameters. Consequently, legal and user-defined requirements are considered optional for the implementation of the prototype. The strategies, solutions and assumptions are stated in the following sections for each requirement.

Comparison between Ethereum and HLF		
Characteristic	Ethereum	HLF
Description	Generic blockchain platform	Modular blockchain platform
Progamming Language	Domain Specific	General Purpose
Throughput	Low	High
Ledger Size	Very high	High
Configurability	Low	Very High
Complexity	Moderate	Very High
Currency	Yes	No *
Mining Possible	Yes	No
Susceptibility to 51% Attacks	Yes	No
Auditability	Yes	Yes
Accountability	No	Yes
Control	Yes	Yes **
Interoperability	Yes	Yes
Openness	Yes	No ***
Reliability	High	Moderate
Scalability	Low	High
Authentication	No ****	Yes
Privacy	Moderate	Very High
Anonymity	Yes	No
Confidentiality	Moderate	Very High
Nonrepudiation	No	Yes
Data Integrity	Yes	Yes
Immutability	Yes	Yes
Transparency	Yes	Yes *****

Table 4.1.: This table compares Ethereum with HLF and shows the advantages and disadvantages of the respective technologies in regard to data exchange in the healthcare industry [14, 35, 91]. The discrete distinctions consist of *Very High*, *High*, *Moderate*, *Low*, *Yes* and *No*.
* Cryptocurrency is not included in the standard implementation, but can be added via chaincode.
** Via configuration, chaincode and access control list.
*** In general, HLF networks are private, but can be configured to be open.
**** Authentication needs to be implemented on top of Ethereum with other technologies
***** Depending on the configuration, transparency features can be overwritten that restricts the access to the ledger.

4.3.1. Security Aspects

S1: All participants undergo a registration procedure and receive a certificate used for the interaction in the network.

All potential users undergo a registration procedure by the registration authority (RA), which are represented by healthcare service providers, insurance companies or any other suitable institutions. The RA verifies the identity and requests the creation of a certificate for the given user. The certificate authority (CA) creates, signs and stores the respective certificate that is issued to the user. In HLF, this certificate follows the X.509 standard provided by the Fabric CA, but can be replaced by any other standard and furthermore externalized to any other suitable and trustworthy service provider. The user needs the certificate for the authentication and signature for any requests and transactions within the HLF network. The authentication is performed by the membership service provider and can optionally be enhanced by a web interface that connects to the network. This strategy represents a public key infrastructure that represents the recommended approach in HLF [33]

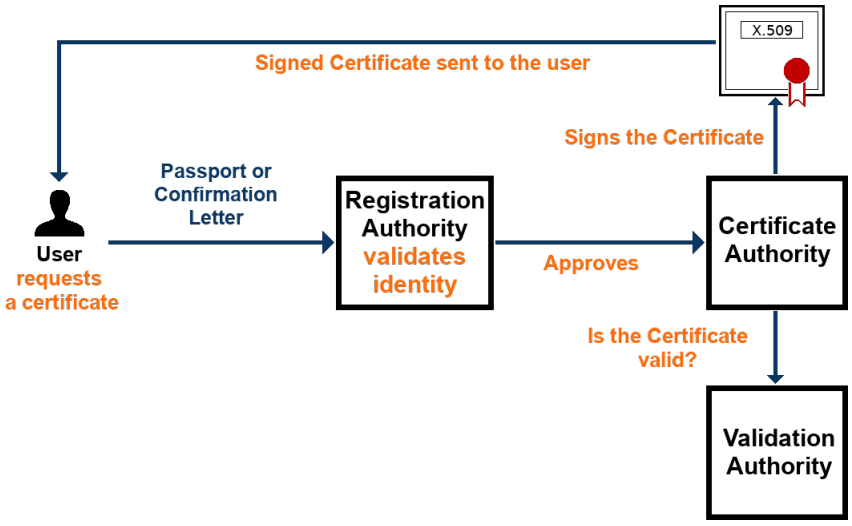

Figure 4.1.: Healthcare service providers and insurance companies represent the registration authority in the network. In order to interact with the network, all users must undergo a registration procedure for the issuance of a certificate. This certificate is signed and stored by a certificate authority that is appointed by any health insurance company but can be delegated to another suitable organization. The membership service provider authenticates the user and acts as the validation authority whenever a user interacts with the HLF network.

S2: Access is granted to registered and authenticated users and denied to anyone else.

After successful verification, the account is activated and the user can log in. A web application provides the user interface for the HLF network via a REST API. Without a successful verification, the access is denied. We assume that a user can only interact with the HLF network via the web application. This ensures that the access to the network is granted if and only if the user has been successfully verified and authenticated.

S3: Participants are identifiable such that repudiation is not possible.

From the assumptions made in requirement S1, a user requires a X.509 certificate to log in and interact with the HLF network, which is issued if the user has undergone the registration procedure and has been successfully verified. In addition, from the assumptions made in requirement S2, a user can interact with the HLF network if and only if the user has been successfully verified and authenticated. In HLF, any transaction and request is signed by the respective user using the given certificate. Any peer and authorized user can validate the signature via the public key infrastructure and thus identify the respective sender.

S4: Privacy of their identity and data is ensured.

For this purpose, the findings from Solidus presented in [15] are used, where an institution performs any requests and transactions on behalf of the user without disclosing its real identity to the HLF network. Consequently, the assumptions made in requirement S1 need to be modified, such that the true identity is not disclosed to the certificate authority and is only known to the registration authority. Therefore, the X.509 certificate includes a representative such as a healthcare service provider, insurance company or another trustworthy institution that performs the requests and transaction on the user's behalf. This approach is similar to the transactions found in the financial industry, where transactions are performed using the IBAN. In several countries, the IBAN is managed by a financial institution and does not disclose the true identity of the client, but provides only a weak measure to ensure privacy on the identity of the user.

Data is kept private by defining the access control list such that only the data owner can read, write and delete the respective data in the HLF network. In addition, HLF provides further tools to enhance privacy, such as private data collections that restrict the access to data to predefined organizations such as healthcare facilities [5].

S5: All interactions and data exchanges among participants are confidential.

HLF allows the definition of access control lists such that conditions ensure that data access is only granted if and only if the data was generated by the user or

belongs to the user. This is similar to performed transactions, where the view access to the transaction log can be restricted accordingly. In addition, chaincode provides a further layer of security by abstracting view requests into transactions. The chaincode checks the identity of the user and the given data and grants the view access only if the user owns the data. HLF provides a feature to enhance data privacy via private data collections. On the other hand, the world state ledger, transaction log and the ledgers to the associated channels are only visible to a subset of authorized users for auditability and accountability purposes. Finally, channels ensure confidentiality by restricting the access to the world state database and transaction log only to the participating peer nodes.

S6: Data integrity is ensured and any non-authorized modification is visible and retraceable.

The ledger is shared among all peers in the network. Any user can request an audit on the data integrity, which is performed by the network administrator or the administrator who creates the network administrator if the view access has been restricted. The network and peer administrator have full access to the network, in case the access control list does not deny access to the ledgers. The consensus mechanism and certificates make sure that any modifications are easily verified and managed. Finally, HLF provides data integrity by design and any finalized transaction cannot be modified.

4.3.2. Legal Requirements

L1: Privacy protection aspects are ensured by default.

Non-authenticated users have no access to the network. Access control lists and chaincode hide any information that does not concern the authenticated user. In addition, the channel configuration file defines the organizations that are allowed to process the requests, which provide the respective user interface. Finally, private data collections allow for further restriction of the data availability to a subset of members in the channel. Otherwise, data is encrypted and stored publicly in the channel using the private key of the data owner provided after the successful registration process.

L2: Associations among participants are only visible to the network administrator.

The access control list allows access to the ledger to the network administrator such that any access is denied to anyone else.

L3: The data exchange is performed using a secure communication line.

All network connections are secured following best practices. Consequently, the network, channel and further configuration files are set accordingly to reflect these

best practices. In addition, the smart contracts follow best coding practices using defensive programming paradigm.

L4: Data owner can grant or revoke access to data.

The chaincode provides the transactions to grant and revoke the access to data owned by the respective authenticated user. In addition, the web application provides the user interface to execute the transactions based on the respective business logic.

L5: Data owner can modify and delete corresponding data.

The chaincode provides the transactions to update and delete data owned by the respective authenticated user. In addition, the web application provides the user interface to execute the transactions based on the respective business logic.

L6: Data owners can move their data to another platform.

The web application provides the user interface to execute this request. All data of the respective user stored in the web application and network are retrieved and summarized to an appropriate data format. The user is notified for the download of the respective data in order to upload the data to another platform.

4.3.3. User-Defined Requirements

R1: Account settings allow the independent update of bank and contact details such as address and phone number.

The web user interface allows for the update of the respective information that is stored in the database of the web application, but not in the HLF network.

R2: The identity of the user cannot be derived when examining the ledger.

Based on the strategy for requirement S4, the identity of the user cannot be derived when examining the ledger.

R3: The prototype provides an overview of all available data to the user based on the identity and role, such as client, healthcare service provider and health insurance company.

The respective queries are defined in the business network model and accessible via the REST API, which the institution can call on behalf of the user. The business logic in the web application and HLF ensures that the user receives only

authorized results of the queries reflected by the respective certificate.

R4: The prototype provides an overview of all data requests from, and to, the current user.

The respective queries are defined in the business network model and accessible via the REST API, which the institution can call on behalf of the user. The business logic in the web application and HLF ensures that the user receives only authorized results of the queries reflected by the respective certificate.

R5: Data owners can instantaneously grant or revoke access to data.

The chaincode provides the transactions to grant and revoke the access to data owned by the respective authenticated user. In addition, the web application provides the user interface to execute the transactions based on the respective business logic.

R6: Authorized users can instantaneously access available data.

The business network model defines the event such that the respective user is notified whenever the data request has been authorized. The authenticated user can access the data using a web interface provided by the respective institution that validates the authorization via the HLF network.

R7: All users are registered and authenticated in order to perform any exchange of data.

See strategies for S1 and S2.

R8: Metadata on current user's data is generated only with explicit consent.

The web application provides an account settings where the user can choose from several options. The metadata is generated in the web application, whereas there is no metadata generated in the HLF network.

R9: Generated metadata is only accessible to the user.

The web application set the business logic to ensure that only authorized users have access to the generated metadata. This is performed by checking the identity of the authenticated user with the metadata of the respective user. There is no metadata stored in the HLF network in order to prevent the generation of metadata in the network that is accessible to organizations in the channel.

R10: Sensitive medical data is accessible to all healthcare service providers given the user's explicit consent.

The web application provides an interface where the user can upload the respective data and allow access to healthcare service providers by generating a data item in the HLF network. The access control list and chaincode ensure that the requesting user has the authorization to access this data based on its role.

R11: Auditability and accountability features are implemented in the prototype.

The network administrator has full access to the transaction logs and can consequently perform the audit. Based on laws and regulations, the network administrator can delegate the access to a subset of organizations. For this purpose, the access control list needs to be updated and extended to allow the access to additional users. In addition, zero-knowledge proofs can hide data that is not needed for these organizations. Accountability features are available in HLF by design, as all members and infrastructure components possess a X.509 certificate used to sign any interaction in the network.

R12: Essential sensitive medical data is accessible to healthcare providers in case of an emergency if and only if the user consented explicitly to provide such information in the case of an emergency.

See strategy for R10.

5. Implementation

The first section, *Architecture*, describes the applied architectural and design patterns. The second section, *Technologies*, presents and describes the technologies used for the front- and backend. In the last two sections, *Implementation of the Fronted* and *Implementation of the Backend*, the actual implementation is shown and highlighted by relevant code snippets. The source code for the implementation is provided on the attached CD and available on Github in the repositories *Mana*, *HLF-Mana* and *Manangular* at https://github.com/basacul.

5.1. Architecture

This section is divided into two parts. The first part describes the architectural pattern and provides a general overview of the software architecture. The second part presents several design patterns that are applied in the prototype.

5.1.1. Architectural Pattern

The implementation uses a layered architecture, namely the three-tier software architecture, such that each tier can scale horizontally. This is primordial for a distributed system, where several organizations rely on shared interfaces for the distribution of data. The presentation tier runs on a web browser that provides the user interface to interact indirectly with the HLF network. The logic tier processes end user requests and applies the business logic to create, read, update and delete data in the corresponding databases based on the given policies. In addition, the logic layer processes the transactions and performs the requested operations on the data tier, which stores the information needed for the application. The data tier represents several databases, where different organizations cooperate in a distributed system. Consequently, each database belongs to the corresponding organization in order to share data in the system, which is connected via the HLF business network.

5.1.2. Design Patterns

The design patterns applied in the implementation consist of

- the Model-View-Controller (MVC),
- the Abstract Factory,
- Flyweight,
- Proxy and

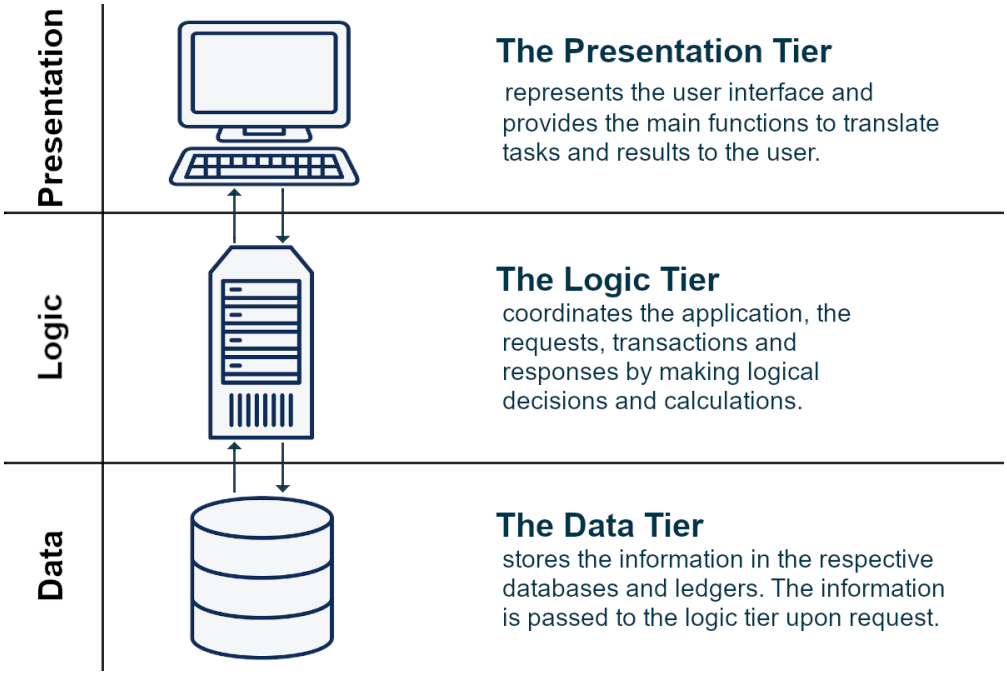

Figure 5.1.: General overview of the three-tier software architecture.

- Publish-Subscriber pattern [95].

The MVC pattern separates the client, the servers and the databases into different components that allow different organizations and developers to work on the same HLF business network and the respective business projects without conflicting with each other. In MVC, the controller manages the requests from the client by applying the business logic and interacts with the model to generate the view. It represents the business logic implemented in the web server in the logic tier. The model represents the databases. The view is served by the web server as static HTML5 files and displayed on the web browser.

The Abstract Factory pattern maintains the evolvability of the application while minimizing the complexity. Healthcare facilities run a diverse set of software applications using proprietary data models that create several difficulties to ensure the interoperability among these applications. With the creation of a data structure common to these applications, the integration complexity is minimized and the extensibility of the data model ensured.

The Flyweight pattern minimizes the data storage requirements for the organizations. The majority of the data is recorded and maintained by the ledger, which is replicated and distributed among the participating nodes in the HLF network. By sharing common data structures, the data redundancy is minimized in an immutable blockchain and the respective databases. For example, each participant in the network possesses a role, which defines the data access authorization by role in the distributed system. These roles include client, healthcare service provider and insurance company. In addition, insurance policies, diagnoses, diagnostic procedures

and treatment plans are limited to a given set of possible values, which can be shared by applying the Flyweight pattern.

The Proxy pattern allows to enhance the control of the access to sensitive data and improves the security by using an additional layer of indirection [95]. The healthcare industry relies on a diverse set of data formats, that range from a few kilobytes to several terabytes in size. By writing this data onto the replicated ledger, the size of the blockchain increases substantially. Furthermore, the data persists in an immutable ledger that is available to several organizations. Data owners, therefore, loose their control over this data, which is not GDPR-compliant. Encryption of this data in an immutable ledger does not provide full protection, as the encryption can theoretically break. Consequently, the Proxy pattern applies an additional layer of security by only storing a pointer in the ledger. This pointer redirects the requester to another location, usually a web server, that includes further security features.

Publisher-Subscriber pattern helps to keep medical health records up-to-date by tracking relevant changes across the population in the system. Any updates on the ledger emit an event, that notifies the subscribers to update the given data. This minimizes the workload for healthcare service providers, who spend a substantial amount of time maintaining electronic health records. In addition, the pattern minimizes the workload in a distributed system and improves the accuracy of the available data among clients, healthcare service providers and health insurance companies.

5.2. Technologies

This section introduces the technologies and tools used for the implementation of the prototype. The main technologies represent Bootstrap version 4.3.1, NodeJS, Express, Passport, MongoDB Atlas, AWS S3, Hyperledger Composer version 0.20.9 and HLF version 1.2.1.

5.2.1. Frontend

In the presentation tier, the frontend provides the user interface needed in order to interact with the HLF network. The frontend relies on HTML5, which is created in the NodeJS web server. The views are stored as embedded Javascript template files and compiled with ejs (embedded Javascript) version 2.6.2 library to static HTML5 files upon request [25]. The web application primarily targets mobile devices and adopts a responsive design, namely the mobile first design. In order to achieve a responsive layout, Bootstrap version 4.3.1 [87] is used as CSS3 framework and supplemented with proprietary CSS3 styling rules. Bootstrap requires two Javascript libraries, namely: jQuery version 3.3.1 [40] and bootstrap.js. Otherwise, no further Javascript frameworks nor Javascript files are used to improve the user experience, except for the Angular application.

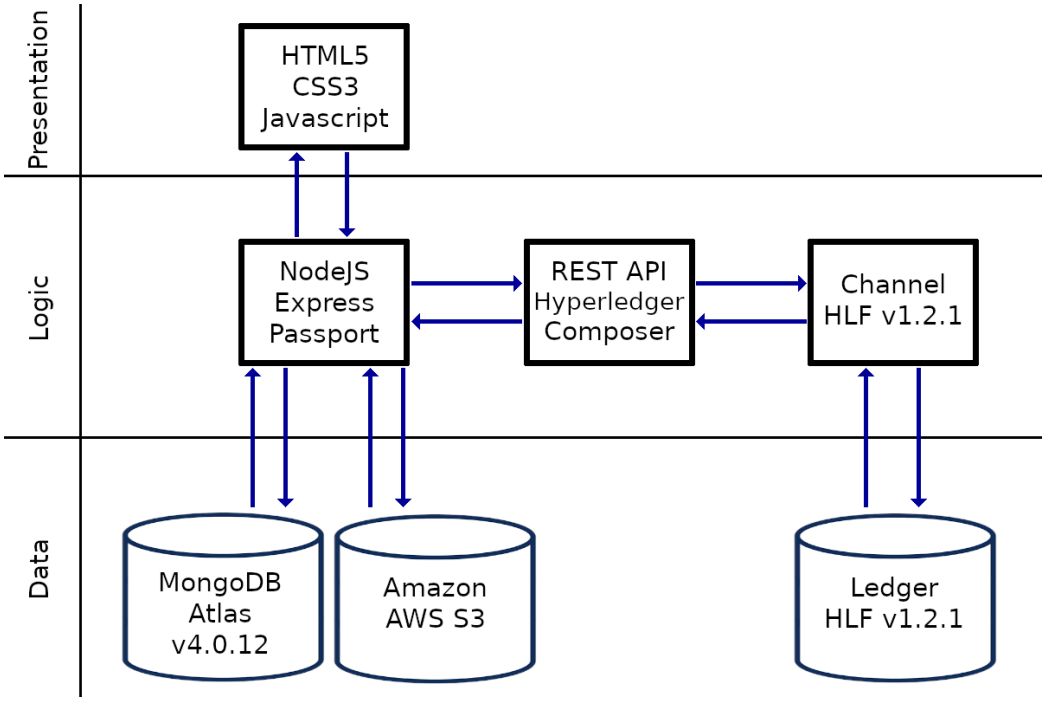

Figure 5.2.: Overview of the technologies used in the three-tier software architecture.

An Angular version 5.6.0 application [42] is furthermore served by the web server and includes several Javascript libraries. This application was automatically generated with Yeoman, a web-scaffolding tool that provides generators, and consequently, accelerates the development process in order to test the HLF network [88].

5.2.2. Backend

Several servers are used for the logic and data tier that use a vast array of different technologies to host, run and serve the responses upon request. The logic tier consists of three different servers: the NodeJS server hosts the web application in Virginia, U.S.A., which requests data from the MongoDB Atlas and the AWS S3 Bucket both hosted in Frankfurt, Germany, and finally via the REST API hosted in a proprietary server in Zurich, Switzerland. The REST API provides the interface for the HLF network application that is hosted on a proprietary server in Zurich, Switzerland. The HLF network application uses Docker [56] to simulate the organizations and ordering service in one physical server.

5.2.2.1. Web Application

The development of the web application was performed on Goorm, which provides an integrated development environment hosted on Amazon Web Services in South Korea [43]. Goorm accelerates the development process, as it allows code changes

to be tested in real-time in the internet. The web application relies on NodeJS version 10.15.3 in addition to specific Javascript libraries. NodeJS is an asynchronous event-driven runtime environment that supports Javascript for the implementation of the business logic [38]. The web application runs on top of NodeJS that is hosted on Heroku, a cloud application platform [20], and uses Express version 4.17.1 as web application framework [81]. The Javascript library Passport version 0.4.0 manages and performs the authentication of the users.

Express is a popular, minimalist and open source framework for the development of NodeJS web applications [81]. The framework provides HTTP utility methods and the middleware for the development that are simple to configure. The web application uses Express version 4.17.1 that runs on top of NodeJS version 10.15.3. Express supports several template engines for the compilation of HTML5 files such as ejs [25], which is used to compile embedded Javascript template files to static HTML5 files. In addition, several Javascript libraries enrich the features of this web application framework, which help to sanitize user input and generate a session ID for the interactions with a given end user.

Passport performs the authentication on the web application. This library represents a middleware for NodeJS [48] that integrates with an Express-based web application. Passport supports several strategies for the authentication such as Google, Facebook and Twitter, which redirects the user to the respective site to perform the authentication for the web application. The prototype uses Passport version 0.4.0 with the local strategy that stores the credentials in the MongoDB Atlas database and requires additional Javascript libraries to perform this strategy [49, 93]. The passwords of the users are stored as hash values using a salt to include some randomness.

In addition, the web application uses several additional Javascript libraries to implement the business logic, such as

- **aws-sdk v2.510.0**, that allows the connection with the AWS S3, which stores the actual files. It provides the methods to create, read, update and delete these files [79].
- **axios v0.19.0** is a HTTP client to perform http requests [6].
- **connect-flash v0.1.1** is a middleware to store messages that are displayed once to the user in order to provide a feedback on performed actions [47].
- **crypto-random-string v3.0.1** is used to create the tokens to perform explicit actions in the web application [80].
- **ejs v2.6.2** compiles embedded Javascript template files to static HTML5 files. The template files interweave Javascript expressions in a HTML5 file [25].
- **express-sanitizer v1.0.5** helps to sanitize user input in order to prevent malicious code injections [3].
- **express-session v1.16.2** creates a session ID in order to give HTTP requests some context and to transport data between the server and the client [82].

- **method-override v3.0.0** supports the declaration of PUT and DELETE requests in HTML5 forms as these are not supported by default [83].
- **mongoose v5.6.0** facilitates the connection with the MongoDB Atlas database and provides the methods to create, read, update and delete user-related data [57].
- **morgan v1.9.1** logs HTTP requests [84].
- **multer v1.4.2** handles the upload of files to the server [85].
- **nodemailer v6.3.0** provides the methods to send emails from the web application [76].
- **passport-local v1.0.0** defines the authentication strategy for passport, where the user needs a username and password to log in. The credentials are stored in the MongoDB Atlas database [49].
- **passport-local-mongoose v5.0.1** simplifies the integration of the local strategy with MongoDB Atlas [93].
- **winston v3.2.1** is a universal logging library and manages the log configurations in the web application [94].

5.2.2.2. HLF and REST API

HLF and the REST API run on top of an Ubuntu Server 16.04 LTS that is hosted on a proprietary web server in Zurich, Switzerland. In order to install HLF, Hyperledger Composer version 0.20.9 is used as development tool that provides the utilities to set up the business network application and the REST API as mid-tier for the web application. As of August 29, 2019, Hyperledger Composer has been deprecated and is not recommended for the set up of a HLF network application.

Hyperledger Composer provides the essential tools to create a HLF network application and the REST API via the command line. This tool creates the NodeJS project for the HLF network application that consists of several files but are mainly defined by:

- a business network application file, with the extension bna, that represents the model. It allows the definitions of resources such as participants, assets, transactions and events. The resources use the respective keywords that consist of *participant*, *asset*, *transaction* and *event*.
- Second, the access control list file, with the extension acl, defines the conditions for the access authorization to the respective and predefined resources.
- Third, the predefined queries served by the REST API are stored within a file with the extension qry.

- And fourth, the chaincode written in Javascript defines the business logic for the transactions. The transaction definitions use annotations to connect the respective transaction to the respective resource defined in the model.

The files for the model, access control list and queries use domain specific programming languages that are similar to related programming languages such as SQL, Javascript and Java. Several programming paradigms such as object-oriented programming and inheritance are featured in the definition of business network models, such that the Abstract Factory design pattern applies. Furthermore, the keywords *enum* and *concept* allow the definitions of enumerations and specific classes, respectively.

Finally, Hyperledger Composer facilitates the deployment of the business network application on a running HLF network application. It compiles the business network application and installs the executable onto the network. Furthermore, it creates a REST API, which simplifies the integration of the network with an external web server.

Yeoman is an open-source web-scaffolding tool that provides a command-line interface. It extends Hyperledger Composer by providing generators for a given set of templates, such as an Angular application. Moreover, it features several tools to optimize and test the HLF network application, which accelerate the creation and testing of the prototype.

HLF version 1.2.1 is used for the development of the HLF network application, which needs to be installed on a server. The manual set up of a HLF network application requires a deeper understanding, as HLF relies on several technology stacks and configuration files [65]. In order to accelerate the development process, a generic solution provided by the Linux Foundation is used to set up a HLF network application on a server running Ubuntu 16.04 LTS. Before installing HLF, several prerequisites need to be fulfilled, which include the installation of

- Ubuntu Linux 14.04/16.04 LTS or Mac OS 10.12,
- Docker engine,
- Docker-Compose,
- NodeJS 8.9 or higher, though version 9 is not supported,
- npm version 5.x,
- git version 2.9.x or higher and
- Pyhton 2.7.x [21].

It is important to note that the specifications need to be fulfilled as stated above. Otherwise several issues arise due to incompatibilities that are time-consuming to resolve. Next, additional components need to be installed before starting the HLF network, such as essential client tools, utilities and Yeoman.

In the final step, a github repository [53] consisting of essential files to install and run a HLF network is cloned and installed on the server [22]. The network is started by

running a bash file, that sets the configurations for the Docker containers, network, channel, public key infrastructure and the ordering service. At the end, one organization is instantiated to start the development on a running HLF network application.

5.3. Implementation of the Frontend

The user interface manages four different tasks. These tasks cover the authentication, the account settings, the private data on the web server and the data exchange on top of HLF. For non-authenticated users, the login page is displayed, from which the user has access to the login, registration, password reset page and the contact form. The authenticated user has access to the tasks for the account settings, private data and data exchange, which are accessible from the navigation bar. Flash messages provide an immediate feedback to performed tasks. Error messages are displayed in a red box and success messages in a green box as an overlay at the top of the view. In addition, an automatically generated Angular application is provided and used for testing purposes accessible in the navigation bar represented by the option *Manangular*.

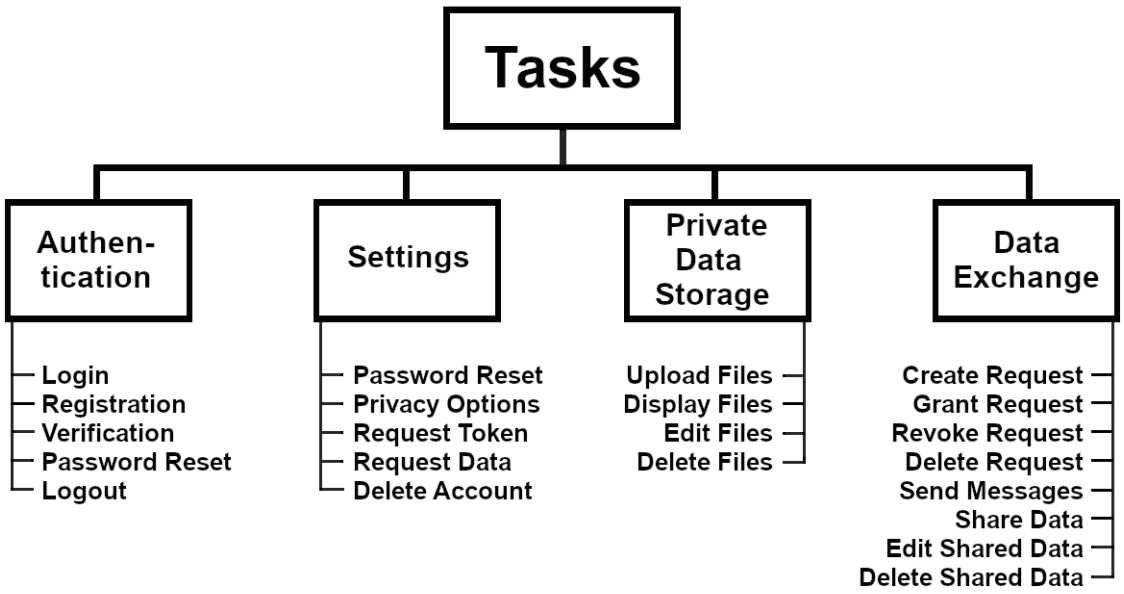

Figure 5.3.: Overview of the four main tasks and the respective sub-tasks provided by the user interface.

Bootstrap allows the creation of web content that implements a responsive layout for several end user devices. It improves the usability of the web application and the user experience. The web content is generated using embedded Javascript template files that add the necessary data to be displayed to the end user. For this purpose, the

```
<% include partials/header %>
...
<% if (files.length > 0) { %>
...
<ul class="list-group...">
    <% files.forEach(function(file){ %>
    <a href="/private/<%=file._id%>">
        <% if(file.accessible || file.authorized.length > 0){ %>
        <li class="list-group-item...">
        <% }else{ %>
        <li class="list-group-item...">
        <% } %>
            <span>
                <%=file.filename%>
            </span>
            <span class="float-right">
                <%=file.date.toDateString()%>
            </span>
        </li>
    </a>
    <% }) %>
</ul>
...
<% } %>
...
<% include partials/footer %>
```

Source Code 5.1: Excerpt of an embedded Javascript template file for displaying available files to the user. This code snippet interweaves Javascript expressions and includes information of existing files that are fetched from the database. After the compilation, the user receives a static HTML5 file upon request.

web server compiles the embedded Javascript template files upon request and sends a static HTML5 file as response. The compilation is performed using the library ejs version 2.6.2. The embedded Javascript expressions are enclosed by specific tags in order to instruct the compiler. In general, the expressions are enclosed by '<%' and '%>'. The equal sign in the tag, '<%=', furthermore signals the compiler to retrieve and include the value of the expression in the static HTML5 file. The header and footer of the views are reused in several templates and included with the according statement in order to minimize code duplication and to increase maintainability of the frontend.

All views of the user interface are shown in the Appendix under A. Frontend.

5.3.1. Authentication

The login page is accessible to any user, where the user can furthermore reset the password and contact the administrator via the contact form. In addition, the user interface provides the views for the registration and verification procedure. These views are accessible to all non-authenticated users.

5.3.2. Account Settings

The account settings enable the user to perform several account related requests and to configure the privacy settings. All privacy related options are turned off by default, such that the user needs to give explicit consent to data collection and sharing. These options can be accessed from the navigation bar represented by the menu with the current user's username and allows the user to

- change the password,
- configure the privacy options such as email notifications, personalization of the user experience and data sharing with public research institutes, healthcare service providers or insurance companies,
- request the current verification token,
- request a copy of all the data stored on the web server with the option to only send a summary that excludes all the files, and finally,
- delete the account.

When requesting the verification token, the user receives an email to the user's email address. The email includes the verification token needed to request a copy of the data and to delete the account.

5.3.3. Private Data

The web application enables the user to store files in the cloud. This feature is accessible from the navigation bar via the menu E-Record. The web server stores metadata in the MongoDB Atlas database and the actual files in the AWS S3 bucket. The required information to retrieve the file is stored as metadata of the respective file object in the MongoDB Atlas database. The user interface provides the respective methods to upload, edit and delete the files that are managed by the web application.

The user interface provides an overview of all available files. The coloring of the border of the file items provide an additional feedback to the user. Shared files in the HLF network are tagged by a green border, as shown in Figure A.9. By clicking on the respective file item, the user is redirected to the show view, where additional

information is displayed. The user can download, edit and delete the file in the show view, as shown in Figures A.10, A.11 and A.12.

5.3.4. Data Exchange

The menu E-Record provides the user interface to request and exchange data, which is subdivided into two sub-menus, that handle the data requests and items, respectively. The index page of E-Record provides an overview of all requests that are pending, approved or need approval. These requests are tagged accordingly with a red or green border, as shown in Figure A.13. The approved data requests are tagged with a green border. Requests that are pending or need the approval of the current user, are tagged with a red border. In addition, all available data items are enlisted and tagged with a green border. The access to data items is defined by the role. Each user has a role, that can take the value of a client, healthcare service provider or an insurance company. These roles are declared in the business model as an enumeration. By default, all registered users are defined as client. The network administrator updates the role to provider or insurance company upon request after the successful verification.

The sub-menu, Associations, welcomes the user with an overview of all concerning data requests, as shown in Figure A.14. The user can send data access requests to all available participants in the HLF network and add additional messages to the request, as shown in Figure A.15. From the show page of a given data request, the user can send messages and delete the request, as shown in Figure A.16. The data owner can furthermore approve or revoke the request. Once approved, the requester can download the data from the show page of the respective data request. Based on the Abstract Factory pattern, the data requests are defined as associations in the HLF business network application that can be extended in future iterations. Consequently, the name was retained to signal the development stage of the prototype.

The sub-menu, Items, provides an overview of all owned and available data items in the HLF network, as shown in Figure A.20. The data owner shares documents based on the role of the participants. This allows the user to share important information by authorizing the access to healthcare service providers for emergency situations. The authorization can also be assigned to clients or insurance companies. All authorized users can download the data, whereas only the data owner can edit or delete the item.

The data is represented by the files, which are managed by the web application. Whenever a document is shared in the HLF network, the respective file is tagged with a green border. This helps the user to easily detect shared files in the private data storage, which is managed from the menu, Private.

5.3.5. Angular Application

An Angular version 5.6.0 application is furthermore served by the web server, which provides a reactive user interface for the HLF network application as shown in Figure A.24. This application was automatically generated using Yeoman and integrated into the web application to accelerate the development process. The Angular application connects to the REST API and provides all functionalities such as create, read, update and delete participants and assets. Moreover, queries and transaction requests are also provided in order to perform specific tests. This component is primarily used for the development process and to manually update the role of the user in the HLF network. The application is accessible from the navigation bar labeled as Manangular.

5.4. Implementation of the Backend

The backend consists of the logic and data tier, which take the role of the controller and model, respectively, according to the MVC design pattern. In the logic tier, the NodeJS web server implements the business logic for the authentication, account settings, private data management and data exchange, as shown in Figure 5.3. The channel implements the business logic for the HLF business network application. The REST API provides the interface for the NodeJS web application to interact with the channel.

The data tier is represented by three databases. MongoDB Atlas manages the users, privacy options, files and the mapping of the user with the respective ID in the HLF network application. AWS S3 stores the actual uploaded files and simulates the setting of an external database, which can be maintained by an institution such as a healthcare facility. The file object stored in MongoDB Atlas contains the information to access the file in the AWS S3 bucket. The NodeJS web server is connected to the MongoDB Atlas and AWS S3 bucket using additional Javascript libraries. The third database represents the ledger, which consists of the world state database and the blockchain. The world state stores the resources such as the users, associations, items and messages that are managed by CouchDB. This database management system stores resources as JSON objects and allows the execution of complex queries. The history of all performed database manipulations are stored in the transaction log as blockchain.

The first part, *Data Tier*, introduces the data model. The second part, *Logic Tier*, shows the implementation and underlying mechanisms of the business logic for the four tasks that are handled by the web application server. In addition, it showcases the HLF business network application and the implementation of the business model, the access control list, the queries and the chaincode.

5.4.1. Data Tier

The data model is defined for all three databases, which are logically interconnected. The ledger is only accessible by the channel via the predefined transaction and query definitions. MongoDB represents the database attached to the NodeJS web application. The Proxy design pattern introduces a layer of indirection by storing a pointer in the respective entries. The file object in MongoDB, as well as the associations and items in the ledger, contain a pointer to the actual file, which is stored in the AWS S3 bucket. This prevents the storage of data in the ledger, that is replicated among all peer nodes in the channel. Consequently, the Proxy design pattern minimizes the size of the ledger and enhances data privacy on sensitive data without compromising the efficiency of the distributed network.

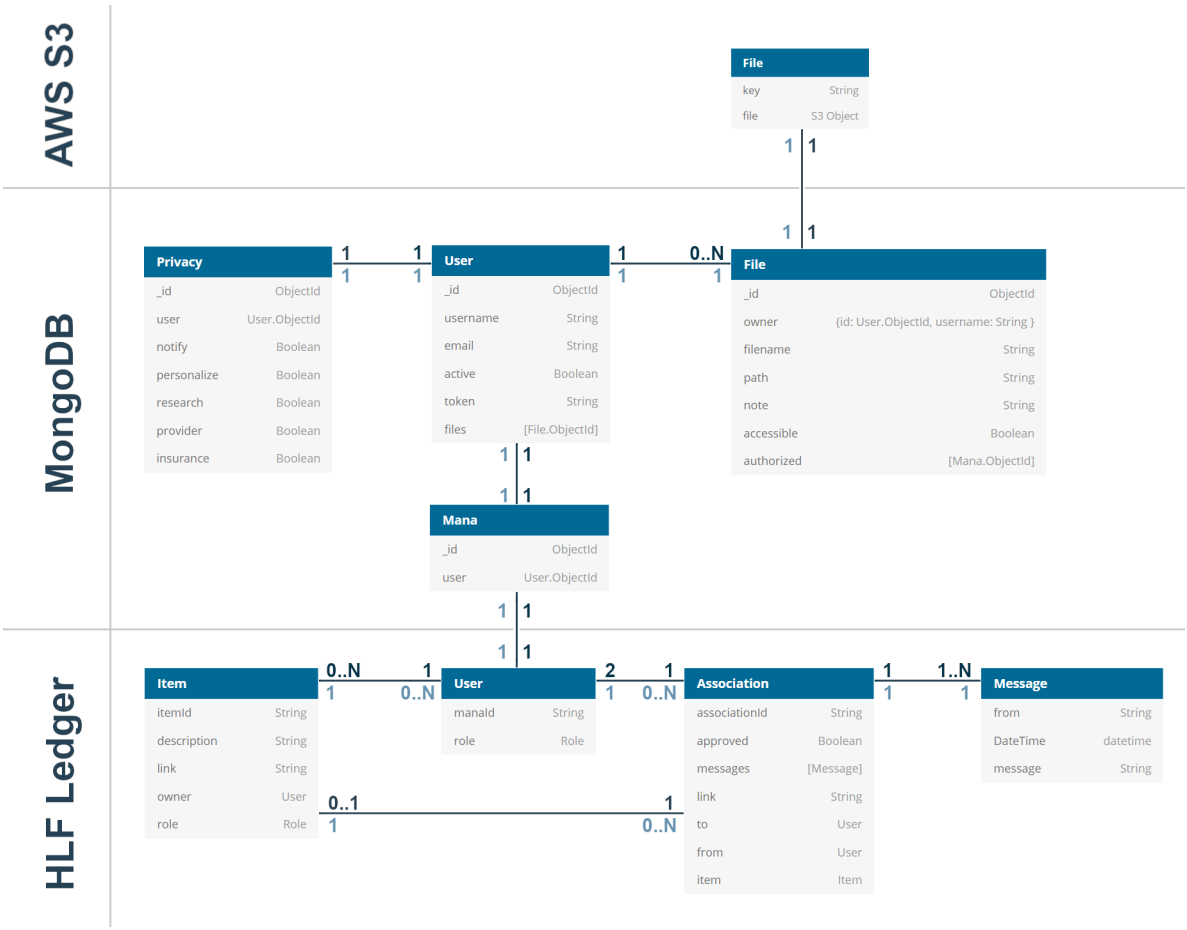

Figure 5.4.: Overview of the data model used in the data tier.

The Javascript library mongoose provides the tools for the definition of the model and management of the MongoDB Atlas database. These models are stored in separate files on the web application server and are reused for the interactions between the web application and the MongoDB Atlas database. Source Code B.1 showcases the model definition for the user, which implements further methods for the authentication using the Javascript library passport-local-mongoose.

In HLF, the data model is defined in the respective business network application file. It defines several resources, the transactions and events, that are emitted in the network application. The syntax is similar to Java and follows object-oriented design patterns such as inheritance. The file defines the user using the keyword participant, the association and item using the keyword *asset*, as well as the transactions and events using the respective keywords. The compilation of the business network application creates the executable from the query definition file, access control list and the chaincode, which needs to be installed in the respective channel. Source Code B.6 shows an excerpt from the business network application file, which defines all the resources needed for the application.

5.4.2. Logic Tier

The logic tier represents the main engine of the prototype and implements the business logic. The NodeJS web server and the channel represent the controller in the MVC design pattern. The web server handles the authentication, account settings, private data storage and the data exchange. The channel manages the transaction logic that are performed by the peer nodes, the ordering service and the membership service provider. The REST API wraps the interface of the channel and provides a standardized interface for the web application in order to interact with the HLF network application.

This part is divided into two subparts. The first subpart presents the implementation of the web application and the second subpart the channel and REST API.

5.4.2.1. Web Application Server

The web application server handles several requests and uses specific Javascript libraries to process the given tasks. Express handles the HTTP requests and is configured in the file app.js, that defines the routes, additional configurations and the business logic. For all tasks provided by the frontend, specific routes are set. Each route is associated with the respective embedded Javascript template file, that is compiled and sent upon request. A default route is defined in case the requested URL does not exist, which redirects the user to the login or the home page depending on the authentication status.

Authentication

A user can log in if and only if the user has performed the registration and verification procedure, which activates the account and generates a mapping of the user with a given ID and a role in the HLF network. The role is set to CLIENT by default. Consequently, any registered user has performed

```
1 router.get("/", function (req, res) {
2     if (req.isAuthenticated()) {
3         res.redirect("home");
4     } else {
5         res.render('auth/login');
6     }
7 });
```

Source Code 5.2: The definition of the route and the business logic for the index page of the web application, which is performed on the web server. If the user is already authenticated, Express redirects the user to the home page. Otherwise, the server renders the embedded Javascript template for the login page. The function parameter req represents the user's request and res the server's response.

1 a registration, that asks for a preferred username, the email address for the verification token and a password for the login.

2 Next, the user receives an email with the verification token in order to finalize the registration. For this purpose, the user is redirected to the verification page, where the user inputs the chosen username and the provided verification token.

3 Finally, the user account is activated and the respective participant is created in the HLF network application with a randomly generated ID that the web application maps to the user.

These steps merge the procedures performed by the 'Techniker Krankenkasse' in Germany and the research findings of the prototype Solidus [15].

The Javascript library Passport implements the authentication, which is processed by the web application by using the local strategy. This strategy configures Passport to use the MongoDB Atlas database in order to verify the username and the provided password. MongoDB stores the password as a hash value together with the salt in the respective user object.

The web application implements a middleware that validates the authorization of the user's request. The respective middleware is defined in one file that is stored in the folder *middleware* of the web application. All routes that manage the account settings, private data storage and data exchange include the middleware, *isLoggedIn*. This middleware function primarily checks if the user is authenticated by calling the function, *isAuthenticated()*, on the request body. The function, isAuthenticated(), is provided by the Javascript library passport-local-mongoose and implemented in the mongoose model definition for the user as shown in Source Code B.1.

```
middleware.isLoggedIn = function (req, res, next) {
    if (req.isAuthenticated()) {
        return next();
    } else {
        req.flash('error', 'Please, Login First');
        res.redirect('/');
    }
};
```

Source Code 5.3: The object middleware includes several function definitions. The definition, *isLoggedIn*, verifies if the user is authenticated. If a user is authenticated, the function calls next(), which refers to the next parameter in the router definition handled by Express. Otherwise, the user is redirected to the login page and provides a message with instructions that is displayed in a red box to the end user.

```
router.get("/", middleware.isLoggedIn, function (req, res) {
    res.render("app/home");
});
```

Source Code 5.4: Express handles the HTTP requests defined in the routes and configured in the Javascript file app.js. Each route definition represents a URL that implements the respective business logic. The middleware is included as parameter in several route functions. The middleware function, *isLoggedIn*, calls the anonymous function(req, res), if and only if the user is authenticated. Otherwise the user is redirected to the index page of the web application and the execution of the current route definition is cancelled.

Account Settings

For all options provided by the view for the account settings, the routes are predefined. They handle the respective requests by updating the respective variables in the privacy object stored in the MongoDB Atlas database or by sending the requested data by email. The Javascript library nodemailer provides the functions to send emails to the user. The library relies on an API that is provided by Mailgun as intermediary web service [58]. When the user requests the verification token, the email is sent with a delay in order to ensure that the user has enough time to inform the network administrator in case the credentials have been stolen. In addition, the user needs to provide the verification token if the user wishes to receive a copy of the data stored in the network application as well as for the deletion of the account. The verification token represents an additional security mechanism against malicious and unintentional activities.

Private Data

The uploaded files form the basis for the data exchange among participants in the HLF network application. The files are available on the AWS S3 bucket and the metadata of the file is stored in the respective file object in the MongoDB Atlas database. All features for the private data storage require the user to be authenticated.

```
1  const storage = multer.diskStorage({
2    destination: function (req, file, cb) {
3      cb(null, 'temp')
4    },
5    filename: function (req, file, cb) {
6      const hash = crypto.createHmac('sha256', req.user.username)
7                  .update(Date.now()
8                  .toString())
9                  .digest('hex');
10     const format = file.originalname.split('.').pop()
11    cb(null, `${hash}.${format}`);
12   }
13 });

15 middleware.upload = multer({ storage: storage });
```

Source Code 5.5: The Javascript library multer handles the upload of the file to the web server. The file is temporarily stored in the temp folder. In addition, the middleware definition creates a hash value as filename. This function definition for the hash value can be replaced by another definition that creates a hash value from the data of the file. This modification enhances the data integrity feature of the web application. Unfortunately, this was dismissed in order to reduce the number of computations on the hosted web server and to avoid the associated delay.

The end user uploads the file using the respective HTML5 form. The Javascript library multer provides the interface for the upload and is implemented as a middleware function, as shown in Source Code 5.5. The respective function handles the upload from the HTML5 form and generates the filename. Next, the uploaded file is passed as parameter to the API provided by the Javascript library aws-sdk, which uploads the current file to the AWS S3 bucket. Thereafter, a file object in MongoDB is saved. This object includes the values to retrieve the file from the bucket. Finally, the uploaded file stored in the temp folder is deleted. For all these steps, a flash message is generated, which provides the user immediate feedback if something failed. Otherwise, a success message is displayed. The business logic for the file upload is shown in Source Code B.9.

The filename is simply a random hash value parameterized with the username and the current date, but can be modified, such that it generates a hash value from the

given data. This would enhance the data integrity, as authorized users could easily check for the data integrity of the file. Unfortunately, this was dismissed in order to reduce the number of computations on the hosted server.

Data Exchange

The data exchange uses all three databases and servers to process the requests by the end user. The REST API provides the interface to the HLF network application to create, read, update and delete resources, as well as to process transactions and queries. The Javascript library axios is used to integrate the REST API with the web application by implementing the required functions. The file hyperledger.js is stored in the folder utils hosted by the web application, which includes a subset of all available API calls. This subset suffices for the implementation of the prototype based on the given requirements.

Using the Proxy design pattern, the ledger acts as a repository and includes a pointer to the file that redirects the user to the web server, where the validation on the access request is performed. For this purpose, the web application fetches the associated resource from the ledger as well as the metadata of the respective file and the manaId of the authenticated user from the MongoDB Atlas database. The validation includes the following steps:

1. Fetch the associated resource from the ledger.
2. Fetch the manaId of the current authenticated user.
3. Fetch the respective file object.
4. Check if the file exists.
5. Check if the current manaId equals the requester's manaId defined in the resource.
6. Check if the current manaId is enlisted as authorized user in the file object.

```
1  hyperledger.grantAssociation = function(associationId,
2                                          message,
3                                          manaId,
4                                          link){
5  let associationObject = {};

7  associationObject.$class = `${config.namespace}.` +
8                             `GrantAssociation`;

10 associationObject.association = `resource:`+
11                 `${hyperledger.namespaces.association}#` +
12                 `${associationId}`;

14 associationObject.from = manaId;
15 associationObject.message = message;
```

```
16 associationObject.link = link;

18 const url=`${config.url}/${config.namespace}.GrantAssociation`;

20 return axios.post(url, associationObject);

22 };
```

Source Code 5.6: This function definition takes an ID (associationId) for the data request, the message and the mapped ID (manaId) in the HLF network application of the current user, as well as the link to the file as function parameters. An empty object named *associationObject* is created and populated with the necessary attributes. The Javascript library axios provides the interface to perform the required HTTP request. The HTTP post request takes two parameters. The first parameter represents the URL of the API to grant the data access. The second parameter represents the populated associationObject to update a given data request.

If any condition fails, the request is denied and the user is redirected to the previous view. Otherwise, the access is granted and the user can download the file directly from the AWS S3 bucket. The download is processed by the web application, which provides the associated view.

5.4.2.2. HLF and REST API

The HLF network application is defined by the model, the chaincode, the access control list and the queries file. Hyperledger Composer compiles these files and merges them into an executable, which is installed onto the running HLF network. In addition, Hyperledger Composer provides the tools to set up a REST API based on the executable. The set up is performed using the X.509 certificate of the network administrator. Consequently, all transactions and API calls are signed with the respective certificate.

Chaincode

The chaincode implements the logic for the available transactions, which are defined in the model. Annotations are used to reference the function definitions to the respective transaction definitions. These annotations are essential for the compilation of an executable, which is performed by Hyperledger Composer. The annotation, @param, specifies the respective resource in the model file and the parameter name for the associated function definition. The value for the parameter is passed by the web application, which calls the function via the REST API. These calls are performed by the Javascript library axios as http post request. Source Code 5.7 shows an example of a function definition in the chaincode implementation.

An event is emitted at the end of the transaction that needs to be stated in the chaincode function definition. For all transactions, the respective event is defined in the model. This represents the Publish-Subscriber design pattern and allows the web application server to listen for updates on subscribed resources. Unfortunately, this feature is not implemented in the web application due to time constraints. In order to implement this feature, the web application needs to connect to the respective web socket on the proprietary server, which hosts the REST API. Consequently, the business logic needs to be defined, such that the web application listens on the web socket and handles the events accordingly.

The transactions allow the interface to perform operations in the ledger and implement a supplemental security layer by defining a controlled pathway to the database specific operations. All transactions are signed by the authenticated caller, which integrate conditions and validation procedures to database updates such as create, update and delete. Consequently, transactions secure the ledger from unauthorized operations. Furthermore, all performed transactions are signed with the the certificate of the caller and logged in the transaction log, which can be used for internal and external audits.

```
/**
 * Create an item
 * @param {care.openhealth.mana.CreateItem} itemData
 * @transaction
 */
async function createItem(itemData) {

  // 1. create the resource Item instance
  let itemId = itemData.itemId;

  let item = factory.newResource(namespace, 'Item', itemId);
  item.description = itemData.description;
  item.role = itemData.role;
  item.link = itemData.link;
  item.owner = itemData.owner;


  // 2. Add new item to the ledger
  let itemRegistry= await getAssetRegistry(`${namespace}.Item`);
  await itemRegistry.add(item);


  // 3. Emit an event that a new item was created
  let event = factory.newEvent(namespace, 'ItemCreatedEvent');
  event.itemId = item.itemId;
  emit(event);
}
```

Source Code 5.7: This code shows an excerpt of the chaincode implementation, namely for the transaction CreateItem. The @param annotation refers to the resource in the model and defines the parameter name for the function definition. The parameter @transaction signals to the compiler that this function defines a transaction that is available in the model file. The implementation follows the Abstract Factory pattern. Consequently, the function, *getFactory()*, creates the respective resource, which is populated with the given values in the function parameter itemData. Finally, the resource is stored in the world state and the performed transaction is logged in the blockchain. An event is emitted at the end and published by the server, which hosts the REST API.

Access Control List

The access control list consists of a collection of rules that define the conditions for the access of resources. These rules are defined in the file permissions.acl. Source Code 5.8 shows an example for the access of data requests in the ledger. A list of key value pairs such as description, participant, operation, resource, condition and action specify the rules. The value for the condition represents a Boolean expression, which is applied whenever a participant requests the access to the respective resource. Based on the output of the condition, the access is granted or denied.

All access requests to the resources are performed using the X.509 certificate of the network administrator, who has full access to all resources defined in the model. This simplifies the development of the prototype and accelerates the implementation. Nevertheless, this is not the recommended approach in production mode.

```
1  rule ConcernedCanModifyAssociation {
2    description:      "Allow concerned participants to
3                       perform transactions on Associations
4                       that concern them"
5    participant(p):   "care.openhealth.mana.User"
6    operation:        CREATE, READ, UPDATE
7    resource(r):      "care.openhealth.mana.Association"
8    condition:   (r.from.getIdentifier() === p.getIdentifier() ||
9                  r.to.getIdentifier() === p.getIdentifier())
10   action:           ALLOW
11 }
```

Source Code 5.8: This rule defines the participant for the requested resource *Association*. In general, the rule includes a description, the participant and the database operations from a given set of values such as create, read, update and delete. The condition defines the requirement for the authorized access. It allows the inclusion of references, such as p for the participant and r for the respective resource. The condition is evaluated for every request to the specified resource and returns a Boolean value after the evaluation. Finally, if the condition is true, the key action defines if the access is allowed or denied. This key takes either the value ALLOW or DENY.

Queries

Finally, the file *queries.qry* predefines the read operations on the ledger, which are served by the REST API. Consequently, the web application can query the ledger by sending a http get request to the respective API, which returns a JSON object with the desired results wrapped in an array. These queries are defined for the participants and the respective assets, *Item* and Association. The keyword *query* defines the read operation for a specific resource and the syntax is similar to a regular SQL statement.

```
query selectItemByRole {
  description: "Select items for given role"
  statement:
    SELECT  care.openhealth.mana.Item
    WHERE   (role == _$role)
}
```

Source Code 5.9: This query defines the read operation for the resource Item. The parameter role is retrieved from the http get request performed by the web application. A list of items is returned, where the parameter *role* equals the value role defined in the resource Item

6. Evaluation

In the third chapter, *Analysis of the Healthcare System*, the requirements were determined in order to define the evaluation parameters for the prototype. The fourth chapter, *Design Decisions*, established the strategies for each requirement. In this chapter, all requirements are evaluated based on the degree of accomplishment for the security aspects, the legal and user-defined requirements, which are presented in the respective order. For the evaluation, we assume that all servers and communication channels are secured and encrypted by configuring the web servers, communication channels, network and channel configuration files accordingly. From twenty four different requirements, eight were not fulfilled by the prototype as some aspects require a deeper understanding of HLF, have not been implemented or resulted from the implementation of fulfilled requirements. In addition, no X.509 certificates are issued upon the registration and verification of new users. Consequently, the prototype makes nonrepudiation and non-retraceable modifications possible to end users.

6.1. Security Aspects

The prototype does not fulfill the requirements S1, S3 and S6, as these implementations require a deeper understanding of HLF. In addition, the validation of the data integrity on documents is not performed. These documents are stored in an external web server and are outside the control of the HLF network application.

S1: All participants undergo a registration procedure and receive a certificate used for the interaction in the network.

The web application provides the views for the registration and verification, which are handled by the web server. Upon a registration request, the user receives a verification token in order to verify the email address and to activate the user account. If the verification is successful, a participant with the associated manaId is created in the HLF network application without creating a new X.509 certificate. In order to implement the issuance of a new certificate for the respective user, the configurations for the REST API and the public key infrastructure need to be upgraded, which requires a deeper understanding of HLF. Consequently, the prototype does not fulfill this requirement.

S2: Access is granted to registered and authenticated users and denied to anyone else.

We assume that the web server requires an API key to connect to the REST API, and thus, with the HLF network application. In this setting, the user is unable to interact with the network application, unless the user is logged in. Consequently, the user is registered and authenticated in order to access the distributed system,

which fulfills the requirement.

S3: Participants are identifiable, such that repudiation is not possible.

The interactions with the HLF network application are signed using the X.509 certificate of the network administrator. The web application does not store a log on performed activities, such that all transactions, queries and interactions with the HLF network application cannot be mapped onto specific end users whenever the user deletes the account. In this situation, an investigator cannot reconstruct the identity of the deleted user from the associated manaId used in the ledger. Thus, the prototype does not fulfill this requirement.

S4: Privacy of their identity and data is ensured.

We assume that the web server and HLF network application implement the findings from Solidus [15]. In this setting, participants cannot deduce the true identity of the end user by examining the ledger, as the transactions are performed on behalf of the institution via their web services. Once the end users publish the associated manaId with their true identity, privacy is no longer ensured . In addition, the Proxy design pattern imposes the storage of data off the ledger. The data is therefore stored in external web servers that perform the validation of the data access. To sum up, the prototype fulfills the requirement based on the assumption, as long as the users do not publish their manaId in conjunction with their true identity to external observers.

S5: All interactions and data exchanges among participants are confidential.

This requirement is fulfilled based on the assumption that the web servers, communication channels, network application and HLF channels are secured and encrypted.

S6: Data integrity is ensured and any non-authorized modification is visible and retraceable.

The files are stored in external databases and the ledger offers the respective link. The network application does not perform any validation on the data integrity, and consequently, does not adhere to the requirement. In addition, as requirement S3 is not fulfilled, an investigator cannot retrace any non-authorized modifications on the file. Consequently, the prototype does not fulfill this requirement.

6.2. Legal Requirements

All interactions with the HLF network application use the X.509 certificate of the network administrator. Authenticated end users interact with the ledger via this certifi-

cate, and consequently, have the same rights as the network administrator. This allows them to deduce associations among participants in the network, as all users have full access to the ledger. Consequently, the requirement L2 is not fulfilled. In addition, requirement L6 is not implemented in the prototype, as the business logic to send a copy of the data upon request has not been implemented.

L1: Privacy protection aspects are ensured by default.

This requirement is fulfilled because privacy is ensured on their identity and data as stated in the fulfilled requirement S4.

L2: Associations among participants are only visible to the network administrator.

Based on requirement S4, this requires the update of the access control list, such that the network administrator has full access to the ledger. This update has been implemented in the prototype. In addition, all interactions in the prototype are signed with the X.509 certificate of the network administrator. The access control list is configured to allow full access to resources that were created by the owner. The owner represents the respective user of the web application, but all users sign the respective transactions with the same certificate. Consequently, all users have the same right. In conclusion, the network administrator and all users have full access to all resources, such that associations among participants are not only visible to the network administrator. To sum up, the prototype does not fulfill this requirement.

L3: The data exchange is performed using a secure communication line.

Based on the assumptions, this requirement is fulfilled.

L4: Data owner can grant or revoke access to data.

The web application provides the views to read, grant and revoke data access requests that are handled by the web server. The Proxy design pattern imposes the storage of data off the ledger. The data is therefore stored in the web server, which performs the validation of the data access. Consequently, the prototype fulfills this requirement.

L5: Data owner can modify and delete corresponding data.

The web application provides the views to create, read, update and delete corresponding data that are handled by the web server. By applying the Proxy design pattern, the corresponding data is stored in an AWS S3 bucket, which is only accessible to the data owner. Authorized users can read, but not modify nor delete the data. Thus, the prototype fulfills this requirement.

L6: Data owners can move their data to another platform.

The web application provides the view to request a copy of the data and to delete the account. The web server only handles the deletion of user related data in the web server and the AWS S3 bucket, but not in the ledger. Nevertheless, the pointers in these resources show a non-existing file after the execution of the delete request. On the other hand, the web application does not implement the logic to collect, package and send the desired data upon request. To sum up, the prototype does not fulfill this requirement.

6.3. User-Defined Requirements

The prototype does not fulfill the requirements R1, R8 and R9 because the web server does not implement the views, nor the business logic, to update bank or contact details. Furthermore, the web application does not generate any metadata on the user, such that the user cannot consent on the generation of metadata nor have access to non-existing metadata.

R1: Account settings allow the independent update of bank and contact details such as address and phone number.

This requirement is not fulfilled, as the web application does not provide the views nor does the web server handle these requests.

R2: The identity of the user cannot be derived when examining the ledger.

This requirement is fulfilled because privacy is ensured on their identity and data, as stated in the fulfilled requirement S4.

R3: The prototype provides an overview of all available data to the user, based on the identity and role such as client, healthcare service provider and health insurance company.

The web application provides the view, which is handled by the web server by fetching the respective data from the ledger. Consequently, the prototype fulfills this requirement.

R4: The prototype provides an overview of all data requests from and to the current user.

The web application provides the view, which is handled by the web server by fetching the respective data from the ledger. Consequently, the prototype fulfills this requirement.

R5: Data owners can instantaneously grant or revoke access to data.

The web application provides the view, which is handled by the web server to perform the specific updates in the ledger. Consequently, the prototype fulfills this requirement.

R6: Authorized users can instantaneously access available data.

The web application provides the view, which is handled by the web server. The business logic validates the data access request. If successful, the data is available for download to the authorized user. Consequently, the prototype fulfills this requirement.

R7: All users are registered and authenticated in order to perform any exchange of data.

This requirement is fulfilled based on the implementation of the fulfilled requirements S2, R3, R4, R5 and R6.

R8: Metadata on current user's data is generated only with explicit consent.

The web application does not generate any metadata on the users with or without consent. Consequently, the prototype does not fulfill this requirement.

R9: Generated metadata is only accessible to the user.

This requirement is not fulfilled as the web application does not generate any metadata.

R10: Sensitive medical data is accessible to all healthcare service providers given the user's explicit consent.

The web application provides the view to create items in the ledger, which is handled by the web server. The user can choose a document from the private data storage, which the data item refers to, and authorize the access to participants with the respective role in the HLF application network. All healthcare service providers can have access to this sensitive medical data after a successful validation procedure performed by the web server. Consequently, the prototype fulfills this requirement.

R11: Auditability and accountability features are implemented in the prototype.

Auditability feature is implemented in HLF by design, but end users cannot be made accountable for their actions, as repudiation is possible. This is based on the fact that all end users sign their interactions in the HLF network application with the X.509 certificate of the network administrator. Thus, the prototype does

not fulfill this requirement.

R12: Essential sensitive medical data is accessible to healthcare providers in case of an emergency, if and only if the user consented explicitly to provide such information in case of an emergency.

This requirement is based on the requirement R10. The web application does not differentiate between the case of an emergency and the general case for the data access to sensitive medical data. Nevertheless, the end user can create a data item that refers to a document with essential sensitive medical data for the case of an emergency, which is accessible to all healthcare service providers in general. Items include a description to the document, which helps healthcare service providers to find the essential information. Based on this fact, the prototype fulfills this requirement.

7. Results

The first section, *Summary*, merges the findings and generated results from the investigation of the question: Is a secure data exchange using DLT possible? The second section, *Limitations*, presents the inherent problems of current DLTs, Ethereum, HLF and, furthermore, indicates the limitations of the prototype.

7.1. Summary

DLT is a tool that allows several parties to collaborate with each other. The healthcare industry runs on such a collaboration between clients, healthcare service providers and health insurance companies. Ethereum and HLF represent two DLTs that differ in their software architecture and design and allow the management and exchange of data between these parties. Both technologies securely store information in a distributed ledger and admit the implementation of an automated secure data exchange grid for the healthcare industry. The implementation in Ethereum requires several modifications in order to integrate data privacy for a secure data exchange. On the other hand, HLF provides data privacy by design, which is used by a number of companies in the healthcare industry and the public health of Estonia.

In the German healthcare industry, the interactions are marked by the high frequency between, and among, healthcare service providers and health insurance companies that rely on data to perform their services. This data is generated because, and mainly on behalf of, the client, and is primarily exchanged by mail, fax or personally as paper documents. Health insurance companies provide digital channels to exchange data with both parties, whereas healthcare services providers do not generally provide such channels to their clients and coworkers from external healthcare facilities. In addition, the service providers in this industry need to comply with laws and regulations, as the German healthcare industry is heavily regulated. There exist several laws and regulations that define, manage and clarify aspects of medical software and data management. These laws and regulations restrict the freedom of action to perform data exchanges on sensitive data that runs the healthcare industry. These restrictions are summarized as requirements covering the security and primarily GDPR related legal aspects that are complemented by user-defined requirements. In total, twenty four requirements are defined and used for the design, implementation and evaluation of the resulting prototype on top of HLF.

The implementation is based on a three-tier software architecture and applies several design patterns, which include MVC, Abstract Factory, Flyweight, Proxy and Publish-Subscriber pattern. The main technologies used in the prototype are Bootstrap version 4.3.1, NodeJS, Express, Passport, MongoDB, AWS S3, Hyperledger Composer version 0.20.9 and HLF version 1.2.1, which are organized accordingly to the software architecture shown in Figure 5.2. The frontend represents the presentation tier

and provides the views as static HTML5 files for the four main tasks that handle authentication, account settings, private data storage and data exchange. The views are primarily designed for mobile devices and are responsive to different screen sizes. Alongside, the backend covers the logic and data tier. In the logic tier, the prototype uses a web server and a proprietary server for the REST API and HLF network application. The web application server compiles the embedded javascript template files for the respective views and implements the business logic to handle the tasks. The REST API provides the interface for the web application to communicate with the HLF network application. In the data tier, the MongoDB database maintains the data to run the web application. The files are stored in a AWS S3 bucket, which is managed by the web server, that validates the data access requests. Finally, the ledger is maintained by the HLF network application and functions as a repository to perform the data exchanges, which implement the Proxy design pattern.

The evaluation concludes that the prototype fulfills sixteen out of twenty four requirements. But, it also shows inherent flaws. Eight requirements are not fulfilled as these require a deeper understanding of HLF, have not been implemented in the prototype or resulted from the implementation of the fulfilled requirements. The prototype does not issue X.509 certificates upon the registration and verification of new users that use the certificate of the network administrator to interact with the HLF network application. This makes nonrepudiation and non-retraceable modifications possible to end users in the HLF network application and consequently, in the prototype. Nevertheless, the evaluation shows that a secure data exchange is possible using DLT in the healthcare industry.

7.2. Limitations

Ethereum and HLF are young technologies and were made public in 2015 and 2016, respectively. Both technologies are undergoing fast paced improvements such that it is difficult to stay up to date. As of August 29, 2019, Hyperleger Composer is deprecated and the Hyperledger Project recommends HLF version 1.4 LTS instead. Nevertheless, the prototype is based on deprecated technology, because at the time of this writing, there existed better content for the development with Hyperledger Composer using HLF version 1.2.1 than with HLF version 1.4. In addition, the goal of this thesis was to implement a prototype within the given time frame in order to evaluate the technology under an elaborated set of requirements. Consequently, a pragmatic decision was made to use the best available and explained content for the development to achieve this goal.

The implementation of the prototype omitted the network and channel configuration as well as the set up of a public key infrastructure. Therefore, a github repository [54] was cloned, which manages the configurations and the set up. In addition, the repository provides the tools to generate the certificate for the peer admin, who instantiates the HLF network application. The peer admin creates the network administrator with the

respective X.509 certificate using Hyperledger Composer, which provides the tools to set up the REST API. The set up of the mid-tier interface requires a certificate to connect the API with the HLF network application. The prototype uses the certificate of the network administrator such that all requests from the web server are signed with the respective key and does not fulfill essential requirements for a secure data exchange. The solution consists in setting up of a public key infrastructure and an authentication system that issues the certificate for each user in the network application. Finally, these certificates need to be stored in, and managed by, the web server or the server, that runs the REST API. But, the manual set up of an infrastructure using HLF as core technology simply requires a deeper understanding and further research [65].

The validation of the data access request to the documents stored in the AWS S3 bucket requires that the web server has access to the metadata of the respective file stored in the MongoDB database. Consequently, the pointer stored in the ledger needs to redirect the user to the respective web server that has access to the required information. This issue was anticipated by creating a configuration file that allows developers to set the URL accordingly. Nevertheless, the business logic needs to be updated such that the validation is performed in the setting of a truly distributed network application. The update requires the definition of routes for data access requests from users that are unknown to the web application and maps the requested URL internally to the metadata of the respective file. Furthermore, a public key infrastructure or a trusted web service is needed to perform the authentication. Currently, the web server denies access to the files to non-authenticated, and thus, non-registered users.

Finally, the prototype implements shortcuts to reduce the number of computations needed to handle the requests and to accelerate the development of the prototype. The views for the data exchange include hidden input tags, that provide the manaId and further information needed to perform the requests. This represents a real security threat as a malicious user can scroll through the history of the web browser and deduce the true identity for a retrieved manaId. Medical staff do not always adhere to best practices to protect the access to the IT infrastructure in the healthcare facility. Consequently, external visitors can theoretically access the HTML5 file and map the manaId with the true identity of the user. In this case, anonymity and privacy are not guaranteed. The simple solution consists in updating the business logic to perform the computations to retrieve the required information, and moreover, to delete the hidden input tags in the related views to restore anonymity and privacy.

8. Conclusion

The final chapter explains my personal view and concludes the thesis. This thesis focused on the question, is a secure data exchange using DLT possible? The findings and results indicate that it is indeed possible, but not feasible in the German healthcare industry for a foreseeable future. DLT is a young technology and currently maturing in such a fast pace that it is difficult to stay up to date. Several nations and companies worldwide seek to implement software applications using this technology. Unfortunately, the German healthcare industry is not competitive enough to recruit experienced engineers and developers to set up a highly complex infrastructure and secure data exchange interfaces in a heavily regulated system. The healthcare system is struggling to recruit medical personnel, such as nurses and healthcare service providers, based on a stringent budget plan and lacks the political will to improve the situation, especially as there is no monetary incentive. Consequently, the priorities to implement such a technology are situated at the bottom of the respective agendas.

The development of the prototype showed that the complexity is not negligent. Even though HLF provides privacy by design, it still requires a deep knowledge and years of experience to implement this technology [65]. HLF uses several technologies and touches many different areas, such that it becomes highly complex and requires a team of experienced engineers and developers. With such a team, HLF provides the necessary tools and solutions to implement a secure data exchange grid that is compliant with the laws, regulations and the expectations of the participants in the healthcare system.

To sum up, DLT makes it possible to improve the collaboration between clients, healthcare service providers and health insurance companies by making the required information easily accessible. This allows the healthcare system to run more efficiently, as DLT performs time-consuming administrative tasks and allows healthcare service providers to focus on their clients with a complete and coherent health record. On the other hand, this technology requires a team of experienced engineers and developers to successfully implement a secure data exchange grid for the healthcare industry.

9. References

[1] **AnalystPrep**. *Financial Applications of Distributed Ledger Technology*. https://analystprep.com/cfa-level-1-exam/portfolio-management/describe-financial-applications-of-distributed-ledger-technology/ [Online: last accessed 2019-10-20 21:48]. 2019.

[2] **anditgetseverywhere**. *Istanbul*. https://eth.wiki/en/roadmap/istanbul [Online: last accessed 2019-09-17 16:49]. 2019.

[3] **Mark Andrews**. *express-sanitizer*. https://github.com/markau/express-sanitizer [Online: last accessed 2019-09-23 14:51]. 2019.

[4] **Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al.** *Hyperledger fabric: a distributed operating system for permissioned blockchains*. In: Proceedings of the Thirteenth EuroSys Conference. ACM. 2018, p. 30.

[5] **Elli Androulaki, Sharon Cocco, and Chris Ferris**. *Private and confidential transactions with Hyperledger Fabric*. https://developer.ibm.com/tutorials/cl-blockchain-private-confidential-transactions-hyperledger-fabric-zero-knowledge-proof/ [Online: last accessed 2019-09-15 19:42]. 2018.

[6] **axios**. *axios*. https://github.com/axios/axios [Online: last accessed 2019-09-23 14:27]. 2019.

[7] **Asaph Azaria, Ariel Ekblaw, Thiago Vieira, and Andrew Lippman**. *Medrec: Using blockchain for medical data access and permission management*. In: 2016 2nd International Conference on Open and Big Data (OBD). 2016, pp. 25–30.

[8] **Timur Badretdinov**. *Clique: cross-client Proof-of-authority algorithm for Ethereum*. https://medium.com/@Destiner/clique-cross-client-proof-of-authority-algorithm-for-ethereum-8b2a135201d, [Online: last accessed 2019-08-25 07:55]. 2018.

[9] **Joseph J Bambara, Paul R Allen, Kedar Iyer, Rene Madsen, Solomon Lederer, and Michael Wuehler**. *Blockchain: A practical guide to developing business, law, and technology solutions*. McGraw Hill Professional. 2018.

[10] **Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis**. *Consensus in the age of blockchains*. arXiv preprint arXiv:1711.03936. 2017.

[11] **Salman A Baset, Luc Desrosiers, Nitin Gaur, Petr Novotny, Anthony O'Dowd, Venkatraman Ramakrishna, Weimin Sun, and Xun (Brian) Wu**. *Blockchain Development with Hyperledger*. Packt Publishing. 2019.

[12] **Moritz Becker**. *Understanding users' health information privacy concerns for health wearables*. In: Proceedings of the 51st Hawaii International Conference on System Science. University of Hawaii. 2018, pp. 3261–3270.

[13] **Block.one**. *EOSIO*. https://eos.io/ [Online: last accessed 2019-08-25 21:04]. 2019.

[14] **Vitalik Buterin**. *A Next-Generation Smart Contract and Decentralized Application Platform*. https://github.com/ethereum/wiki/wiki/White-Paper [Online: last accessed 2019-09-15 10:37]. 2019.

[15] **Ethan Cecchetti, Fan Zhang, Yan Ji, Ahmed Kosba, Ari Juels, and Elaine Shi**. *Solidus: Confidential distributed ledger transactions via PVORM*. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM. 2017, pp. 701–717.

[16] **Melissa Chase and Sarah Meiklejohn**. *Transparency overlays and applications*. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. ACM. 2016, pp. 168–179.

[17] **Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song**. *Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contract execution*. arXiv preprint arXiv:1804.05141. 2018.

[18] **Olivia Choudhury, Hillol Sarker, Nolan Rudolph, Morgan Foreman, Nicholas Fay, Murtaza Dhuliawala, Issa Sylla, Noor Fairoza, and Amar Das**. *Enforcing Human Subject Regulations using Blockchain and Smart Contracts*. Blockchain in Healthcare Today. 2018.

[19] **European Commission**. *2018 reform of EU data protection rules | European Commission*. https://ec.europa.eu/commission/priorities/justice-and-fundamental-rights/data-protection/2018-reform-eu-data-protection-rules_en [Online: last accessed 2019-08-24 11:04]. 2018.

[20] **Salesforce Company**. *Cloud Application Platform | Heroku*. https://www.heroku.com/ [Online: last accessed 2019-09-22 20:40]. 2019.

[21] **Hyperledger Composer**. *Installing pre-requisites*. https://hyperledger.github.io/composer/v0.19/installing/installing-prereqs.html [Online: last accessed 2019-09-23 19:51]. 2019.

[22] **Hyperledger Composer**. *Installing the development environment*. https://hyperledger.github.io/composer/v0.19/installing/development-tools.html [Online: last accessed 2019-09-23 20:04]. 2019.

[23] **Corda**. *Corda | Industry*. https://www.corda.net/discover/industry.html [Online: last accessed 2019-09-04 10:01]. 2018.

[24] **Dr. Fred Davis**. *Physicians Under Pressure To Sell Practices: How to Remain Independent*. https://physiciansnews.com/2015/01/27/physicians-under-pressure-to-remain-independent/ [Online: last accessed 2019-09-04 09:22]. 2018.

[25] **Matthew Eernisse**. *EJS*. https://github.com/mde/ejs [Online: last accessed 2019-09-22 19:45]. 2019.

[26] **Taavi Einaste**. *Blockchain and healthcare: the Estonian experience*. https://nortal.com/blog/blockchain-healthcare-estonia/ [Online: last accessed 2019-09-14 09:21]. 2018.

[27] **Ariel Caitlyn Ekblaw**. *Medrec: blockchain for medical data access, permission management and trend analysis*. Massachusetts Institute of Technology, Master thesis. 2017.

[28] **Ariel Ekblaw, Asaph Azaria, John D Halamka, and Andrew Lippman**. *A Case Study for Blockchain in Healthcare:"MedRec" prototype for electronic health records and medical research data*. In: Proceedings of IEEE open & big data conference. Vol. 13. IEEE. 2016, p. 13.

[29] **Enterprise Estonia**. *e-Health Records - e-Estonia*. https://e-estonia.com/solutions/healthcare/e-health-record/ [Online: last accessed 2019-09-05 15:37]. 2019.

[30] **ethereum**. *Transactions - web3js*. https://web3j.readthedocs.io/en/latest/transactions.html [Online: last accessed 2019-09-15 17:30]. 2019.

[31] **ethereum**. *web3.js - Ethereum JavaScript API*. https://github.com/ethereum/web3.js/ [Online: last accessed 2019-09-15 17:15]. 2019.

[32] **Ittay Eyal and Emin Gün Sirer**. *Majority is not enough: Bitcoin mining is vulnerable*. In: Communications of the ACM. Vol. 61. 7. ACM. 2018, pp. 95–102.

[33] **Hyperledger Fabric**. *Identity*. https://hyperledger-fabric.readthedocs.io/en/release-1.4/identity/identity.html [Online: last access 2019-09-19 20:07]. 2019.

[34] **Pietro Ferraro, Christopher King, and Robert Shorten**. *Distributed Ledger Technology, Cyber-Physical Systems, and Social Compliance*. arXiv preprint arXiv:1807.00649. 2018.

[35] **Chris Ferris and Davis Enyeart**. *Introducing Hyperledger Fabric 1.4 LTS!* https://www.hyperledger.org/blog/2019/01/10/introducing-hyperledger-fabric-1-4-lts [Online: last accessed 2019-08-26 05:34]. 2019.

[36] **Roy T Fielding and Richard N Taylor**. *Architectural styles and the design of network-based software architectures*. University of California, Irvine Doctoral dissertation. 2000.

[37] **Ethereum Foundation**. *Ethereum Project*. https://www.ethereum.org/ [Online: last accessed 2019-09-04 09:50]. 2018.

[38] **Node.js Foundation**. *About | Node.js*. https://nodejs.org/en/about/ [Online: last accessed 2019-09-22 20:41]. 2019.

[39] **The Linux Foundation**. *Hyperledger - Open Source Blockchain Technologies*. https://www.hyperledger.org/ [Online: last accessed 2019-08-25 07:20]. 2019.

[40] **The jQuery Foundation**. *jQuery*. https://jquery.com/ [Online: last accessed 2019-09-22 20:15]. 2019.

[41] **The Medical Futurist**. *Top 12 Companies Bringing Blockchain To Healthcare*. https://medicalfuturist.com/top-12-companies-bringing-blockchain-to-healthcare [Online: last accessed 2019-09-04 09:59]. 2018.

[42] **Google**. *Angular*. https://angular.io/ [Online: last accessed 2019-09-22 20:30]. 2019.

[43] **Goorm**. *goormIDE - No installation required*. https://ide.goorm.io/ [Online: last accessed 2019-09-22 21:27]. 2019.

[44] **Guardtime**. *Guardtime and Estonian Biobank announce a partnership to deploy Guardtime Helium*. https://guardtime.com/blog/guardtime-and-estonian-biobank-announce-a-partnership-to-deploy-guardtime-helium [Online: last accessed 2019-08-26 09:04]. 2019.

[45] **Guardtime**. *Guardtime Health*. https://guardtime.com/health [Online: last accessed 2019-08-29 09:01]. 2019.

[46] **Guardtime**. *Technology*. https://guardtime.com/technology [Online: last accessed 2019-08-25 20:19]. 2019.

[47] **Jared Hanson**. *connect-flash*. https://github.com/jaredhanson/connect-flash [Online: last accessed 2019-09-23 14:33]. 2019.

[48] **Jared Hanson**. *Passport*. http://www.passportjs.org/ [Online: last accessed 2019-09-23 15:15]. 2019.

[49] **Jared Hanson**. *Passport*. https://github.com/jaredhanson/passport-local [Online: last accessed 2019-09-23 15:17]. 2019.

[50] **HSBlox**. *Blockchain Healthcare and Distributed Ledger Tech Solutions*. https://hsblox.com/solutions/ [Online: last accessed 2019-09-04 09:57]. 2018.

[51] **HSBlox**. *HSBlox*. https://hsblox.com/ [Online: last accessed 2019-08-25 21:54]. 2019.

[52] **Hyperledger**. *A Blockchain Platform for the Enterprise*. https://hyperledger-fabric.readthedocs.io/en/release-1.4/ [Online: last accessed 2019-09-20 19:09]. 2019.

[53] **Hyperledger**. *fabric-dev-servers*. https://github.com/hyperledger/composer-tools/tree/master/packages/fabric-dev-servers [Online: last accessed 2019-09-23 20:05]. 2019.

[54] **Hyperledger**. *Releases*. https://github.com/hyperledger/fabric/releases?after=v0.6.1-preview [Online: last accessed 2019-08-26 05:30]. 2019.

[55] **IBM**. *X.509 certificates*. https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.2.0/com.ibm.zos.v2r2.icha700/xcerts.htm [Online: last accessed 2019-09-19 21:47]. 2019.

[56] **Docker Inc**. *Enterprise Container Platform | Docker*. https://www.docker.com/ [Online: last accessed 2019-09-22 20:57]. 2019.

[57] **Automattic Inc.** *Mongoose*. https://mongoosejs.com/ [Online: last accessed 2019-09-23 14:59]. 2019.

[58] **Mailgun Technologies Inc.** *Transaction Email API Service For Developers | Mailgun*. https://www.mailgun.com/ [Online: last accessed 2019-09-27 04:30]. 2019.

[59] **Ledger Insights**. *Hyperledger Fabric integrates Ethereum smart contracts*. https://www.ledgerinsights.com/hyperledger-fabric-integrates-ethereum-smart-contracts-evm-blockchain/ [Online: last accessed 2019-08-20 21:01]. 2018.

[60] **Iryo**. *IRYO Global participatory healthcare ecosystem*. https://iryo.io/iryo_whitepaper.pdf [Online: last accessed 2019-08-25 20:23]. 2017.

[61] **Iryo**. *IRYO.IO*. https://iryo.io/, [Online: last accessed 2019-08-25 20:22]. 2018.

[62] **Iryo.network**. *Iryo is cancelling plans for an ICO*. https://medium.com/iryo-network/iryo-is-cancelling-plans-for-an-ico-fb32504222e6 [Online: last accessed 2019-08-25 21:20]. 2018.

[63] **Bundesministerium der Justiz und für Verbraucherschutz**. *Bundesdatenschutzgesetz*. https://www.gesetze-im-internet.de/bdsg_2018/index.html [Online: last accessed 2019-08-31 07:04]. 2018.

[64] **Bundesministerium der Justiz und für Verbraucherschutz**. *Medizinproduktegesetz*. http://www.gesetze-im-internet.de/mpg/index.html [Online: last accessed 2019-08-31 14:34]. 2013.

[65] **Tomasz Klim**. *A practical guide to Hyperledger Fabric security*. https://espeoblockchain.com/blog/a-practical-guide-to-hyperledger-fabric-security/ [Online: last accessed 2019-09-23 19:42]. 2019.

[66] **Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou**. *Hawk: The blockchain model of cryptography and privacy-preserving smart contracts*. In: 2016 IEEE symposium on security and privacy (SP). IEEE. 2016, pp. 839–858.

[67] **Kai-Fu Lee**. *AI Superpowers: China, Silicon Valley, and the New World Order*. Houghton Mifflin Harcourt. 2018.

[68] **Antony Lewis**. *A gentle introduction to Ethereum*. https://bitsonblocks.net/2016/10/02/gentle-introduction-ethereum/ [Online: last accessed 2019-08-25 07:05]. 2016.

[69] **Xiaoqi Li, Peng Jiang, Ting Chen, Xiapu Luo, and Qiaoyan Wen**. *A survey on the security of blockchain systems*. In: Future Generation Computer Systems. Elsevier. 2017.

[70] **Medicalchain**. *Medicalchain - Blockchain for electronic health records*. https://medicalchain.com/en/ [Online: last accessed 2019-09-04 10:05]. 2019.

[71] **Medicalchain**. *Medicalchain Whitepaper 2.1*. https://medicalchain.com/Medicalchain-Whitepaper-EN.pdf [Online: last accessed 2019-08-25 22:33]. 2018.

[72] **MetaMask**. *MetaMask*. https://metamask.io/ [Online: last accessed 2019-09-15 17:20]. 2019.

[73] **openEHR**. *What is openEHR*. https://www.openehr.org/ [Online: last accessed 2019-08-25 20:35]. 2019.

[74] **R3**. *Enterprise Blockchain Platform*. https://www.r3.com/ [Online: last accessed 2019-08-25 21:51]. 2019.

[75] **Koshik Raj**. *Foundations of Blockchain*. Packt Publishing. 2019.

[76] **Andris Reinmann**. *Nodemailer*. https://nodemailer.com/about/ [Online: last accessed 2019-09-23 15:09]. 2019.

[77] **Margaret Rouse**. *NoSQL (Not Only SQL database)*. https://searchdatamanagement.techtarget.com/definition/NoSQL-Not-Only-SQL [Online: last accessed 2019-09-16 17:27]. 2011.

[78] **Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza**. *Zerocash: Decentralized anonymous payments from bitcoin*. In: 2014 IEEE Symposium on Security and Privacy. IEEE. 2014, pp. 459–474.

[79] **Amazon Web Services**. *AWS SDK for JavaScript*. https://github.com/aws/aws-sdk-js [Online: last accessed 2019-09-22 21:07]. 2019.

[80] **Sindre Sorhus**. *crypto-random-string*. https://github.com/sindresorhus/crypto-random-string [Online: last accessed 2019-09-23 14:37]. 2019.

[81] **StrongLoop, IBM, and other**. *Express - Node.js web application framework*. http://expressjs.com/ [Online: last accessed 2019-09-23 14:44]. 2019.

[82] **StrongLoop, IBM, and other**. *express-session*. https://github.com/expressjs/session [Online: last accessed 2019-09-23 14:53]. 2019.

[83] **StrongLoop, IBM, and other**. *method-override*. https://github.com/expressjs/method-override [Online: last accessed 2019-09-23 14:57]. 2019.

[84] **StrongLoop, IBM, and other**. *morgan*. https://github.com/expressjs/morgan [Online: last accessed 2019-09-23 15:23]. 2019.

[85] **StrongLoop, IBM, and other**. *Multer*. https://github.com/expressjs/multer [Online: last accessed 2019-09-23 15:07]. 2019.

[86] **Future Blockchain Summit**. *CNN live stream from the Future Blockchain Summit 2019 - Dubai*. https://www.youtube.com/watch?v=ilyh_53_KWo [Online: accesse 2019-10-11 10:43]. 2019.

[87] **Bootstrap Team**. *Bootstrap*. https://getbootstrap.com [Online: last accessed 2019-09-22 20:10]. 2019.

[88] **The Yeoman Team**. *The web's scaffolding tool for modern webapps*. https://yeoman.io/ [Online: last accessed 2019-09-22 20:33]. 2019.

[89] **Uchi Ugobame Uchibeke, Kevin A Schneider, Sara Hosseinzadeh Kassani, and Ralph Deters**. *Blockchain access control Ecosystem for Big Data security*. In: 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData). IEEE. 2018, pp. 1373–1378.

[90] **Universa**. *Decentralized autonomous organization—What is a DAO company?* https://medium.com/universablockchain/decentralized-autonomous-organization-what-is-a-dao-company-eb99e472f23e [Online: last accessed 2019-09-04 10:33]. 2017.

[91] **Martin Valenta and Philipp Sandner**. *Comparison of ethereum, hyperledger fabric and corda*. Frankfurt School, Blockchain Center. 2017.

[92] **Fabian Vogelsteller and Vitalik Buterin**. *ERC-20 Token Standard*. https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md [Online: last accessed 2019-09-17 16:34]. 2015.

[93] **Christoph Walcher**. *Passport-Local Mongoose*. https://github.com/saintedlama/passport-local-mongoose [Online: last accessed 2019-09-23 15:21]. 2019.

[94] **winstonjs**. *winston*. https://github.com/winstonjs/winston [Online: last accessed 2019-09-23 15:26]. 2019.

[95] **Peng Zhang, Jules White, Douglas C Schmidt, and Gunther Lenz**. *Applying software patterns to address interoperability in blockchain-based healthcare apps*. arXiv preprint arXiv:1706.03700. 2017.

[96] **Guy Zyskind, Oz Nathan, and Alex 'Sandy' Pentland**. *Decentralizing privacy: Using blockchain to protect personal data*. In: 2015 IEEE Security and Privacy Workshops. IEEE. 2015, pp. 180–184.

Appendices

A. Frontend

Figure A.1.: The view of the login page. The user needs to input the username and the password. In addition, the buttons, *Register?* and *Password?*, redirect the user to the registration view or password reset view, respectively. The link, *Contact Us*, redirects the user to the contact form.

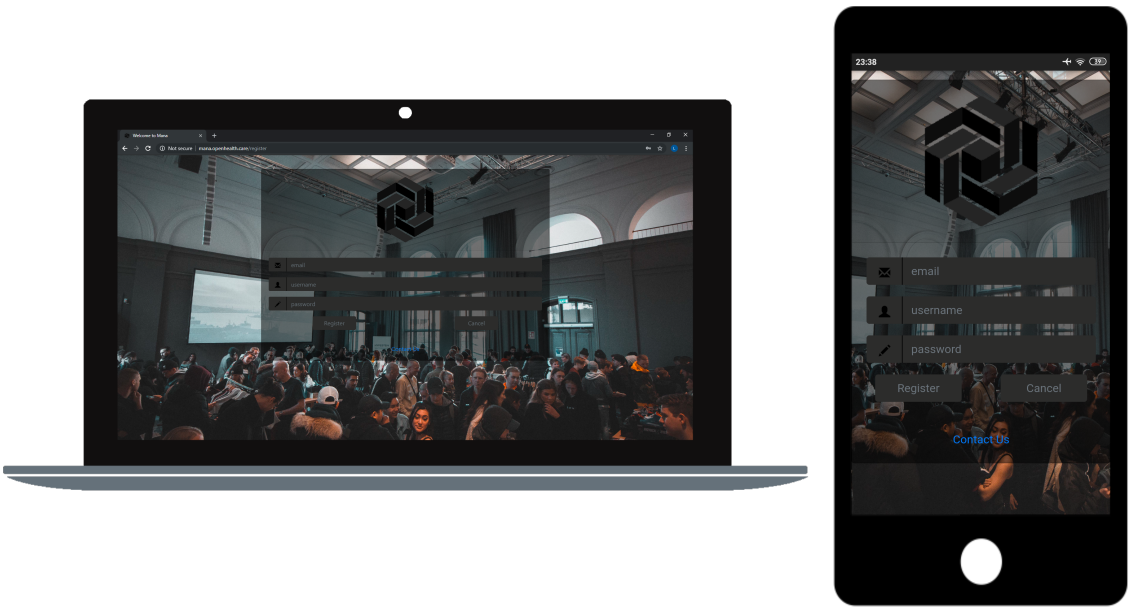

Figure A.2.: The view of the registration page requires a valid email address, a username and a password. By clicking the button, *Register*, the web server validates the input and sends the verification token to the given email address, if the chosen username is available. Otherwise, the user needs to provide a different username.

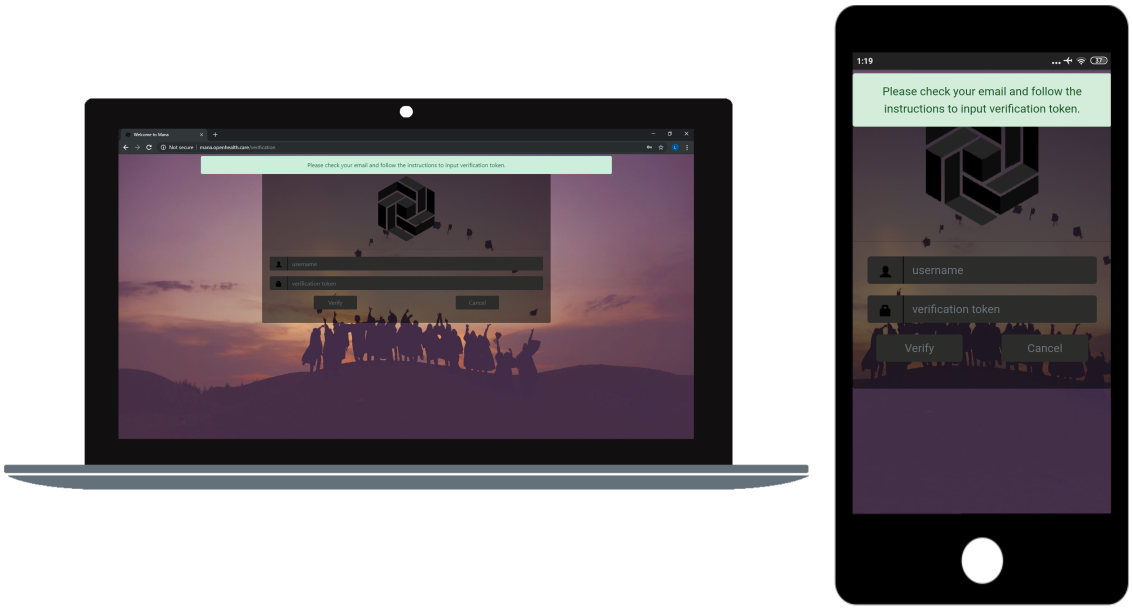

Figure A.3.: The view of the verification page is shown, when the user has performed the registration. A flash message provides the instructions in order to finalize the registration and activate the account. The user needs to input the provided username and the verification token that was sent by email.

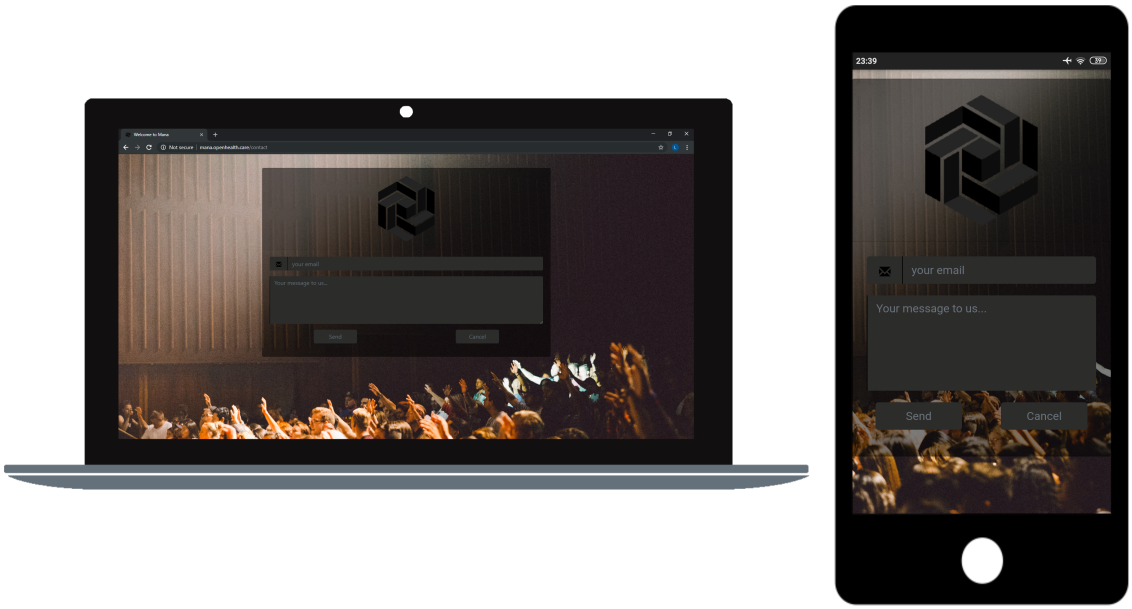

Figure A.4.: The view of the contact form is accessible to non-authenticated users.

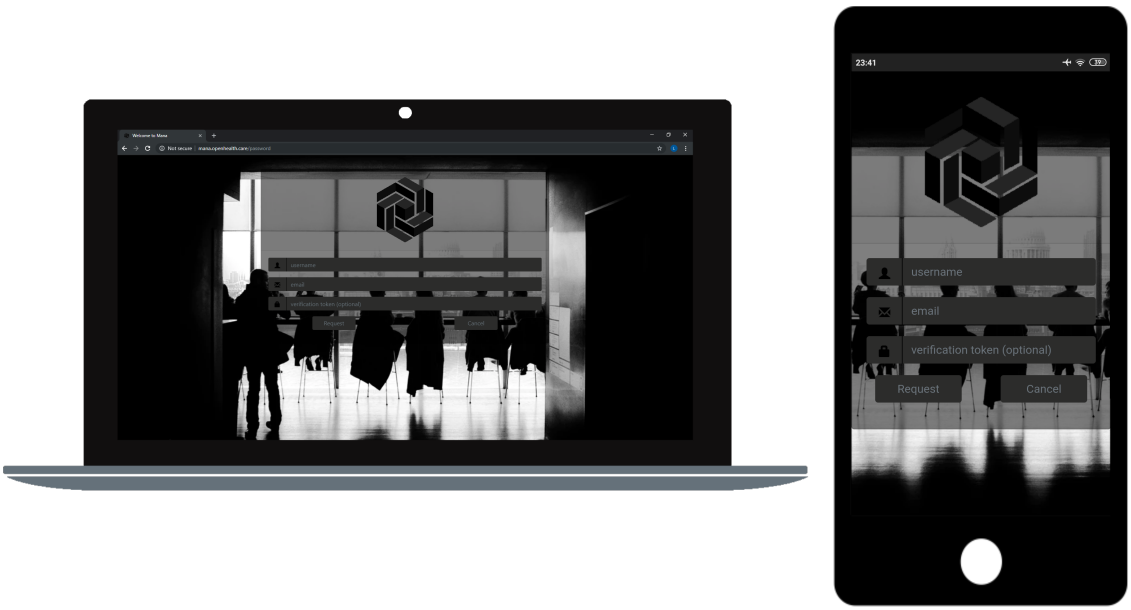

Figure A.5.: The view to request a password reset requires the username, the associated email and optionally the verification token. If the current verification token is provided, the account remains active. Otherwise, the account is deactivated and a new verification token is sent for the password reset.

Figure A.6.: The view to reset the password is shown after the successful password reset request and requires the username, a new password and the verification token.

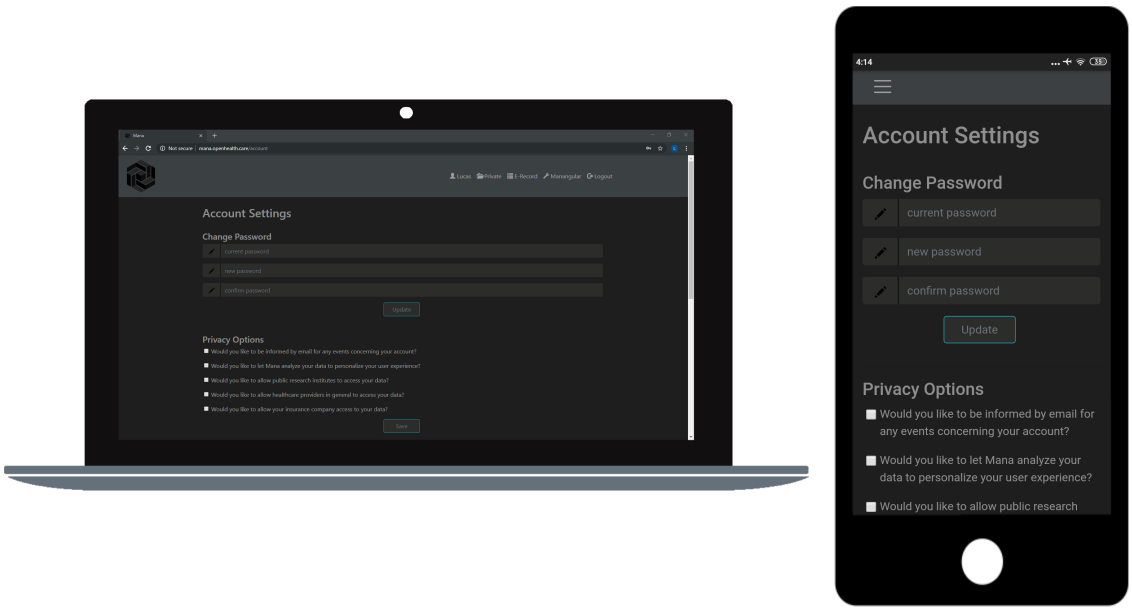

Figure A.7.: The view of the first half of the account settings includes the password reset and the privacy options. The account settings is accessible to all authenticated users.

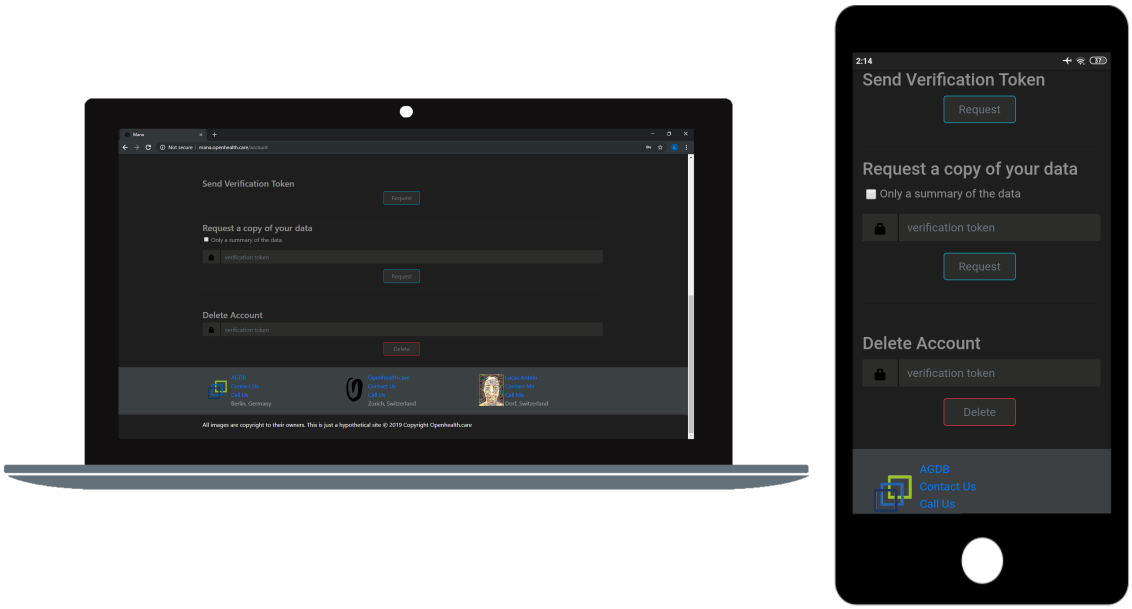

Figure A.8.: The view of the second half of the account settings provides the user interface to request a new verification token, to request a copy of the data associated to the current user and finally to delete the account. The last two options require the current verification token to perform the request.

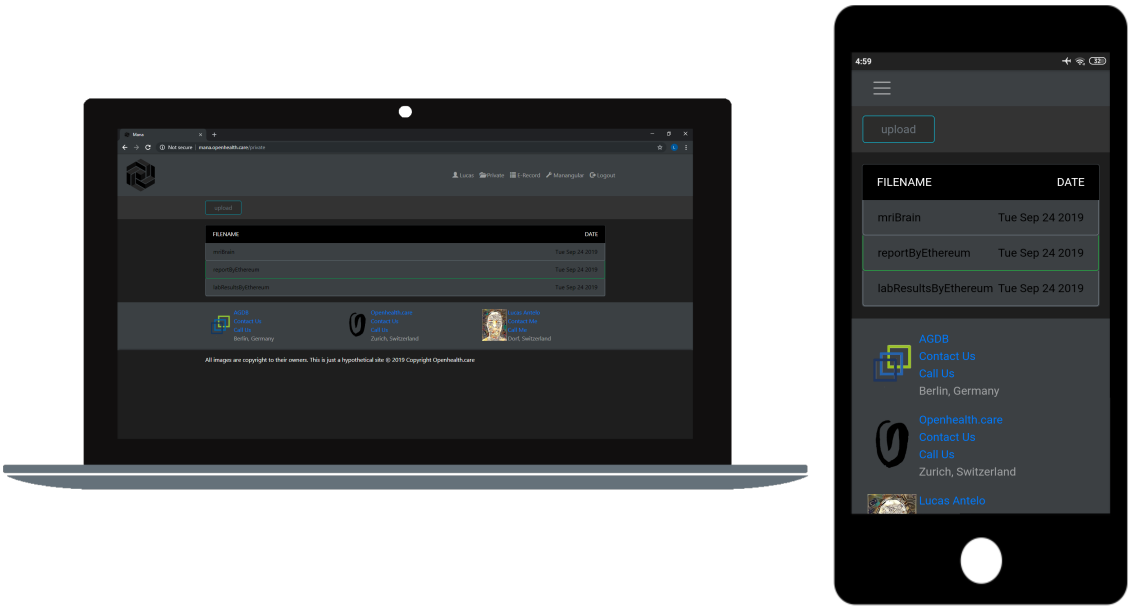

Figure A.9.: The view of the index page of the menu, *Private*. This menu manages the uploaded files that are stored in the AWS S3 bucket. The button in the navigation bar, *upload*, opens a dialog box to upload a file.

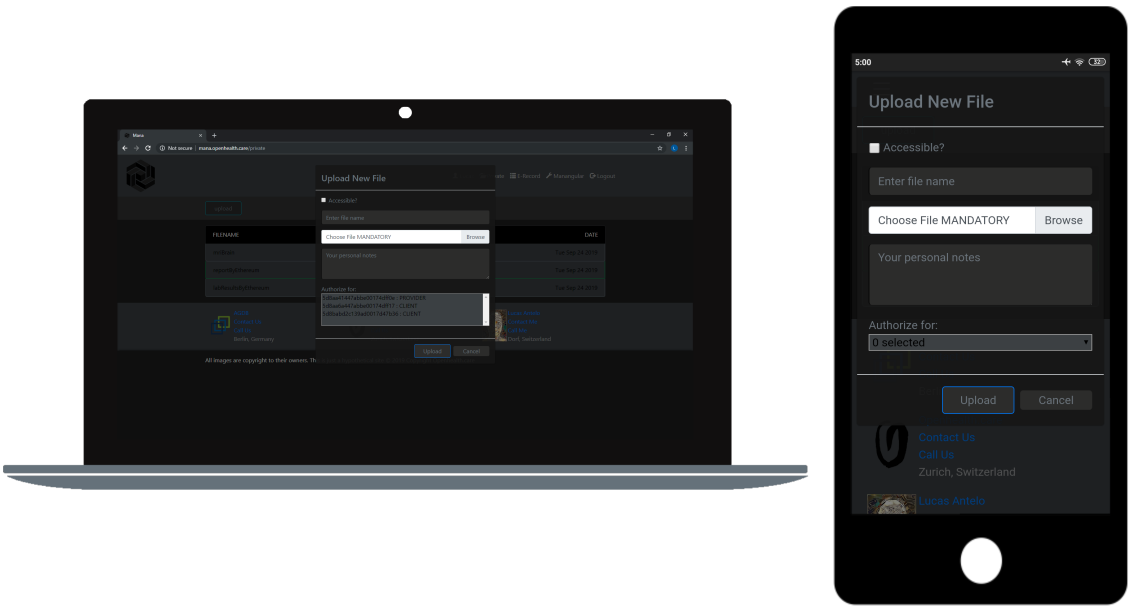

Figure A.10.: The view of the index page of the menu, *Private*, with a dialog box to upload a file. The checkbox, *Accessible?*, allows the user to publish the file in the HLF network application. In addition, the text area, *Your personal notes*, allows the user to include further information. Finally, the data owner can authorize the access to a set of users from the list, *Authorize for:*.

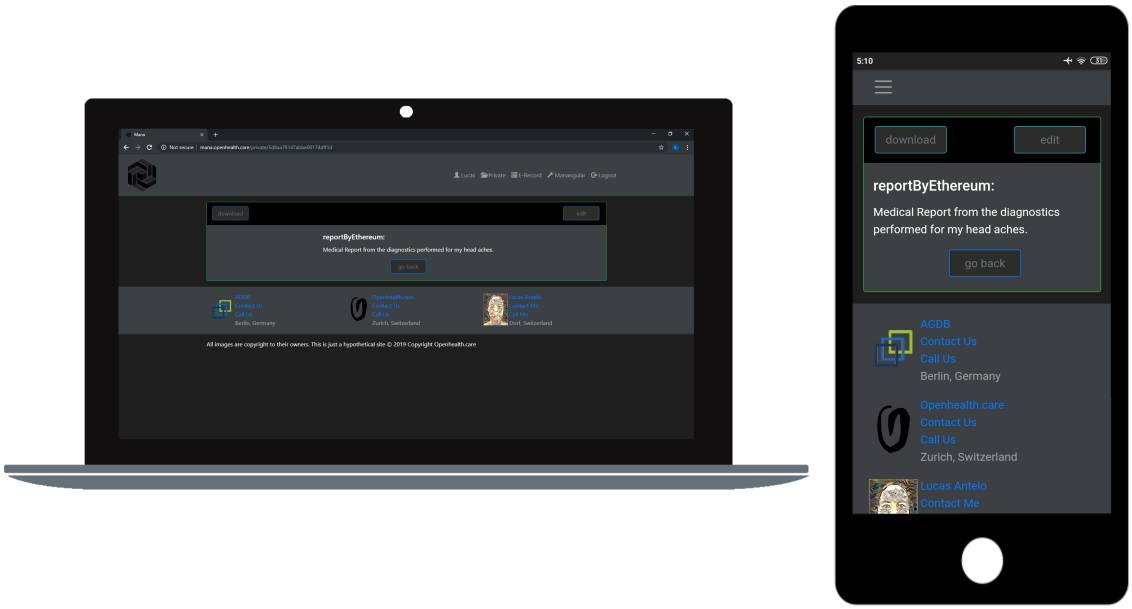

Figure A.11.: The show page of a given file in the menu, *Private*, allows the user to download the file and edit the associated information.

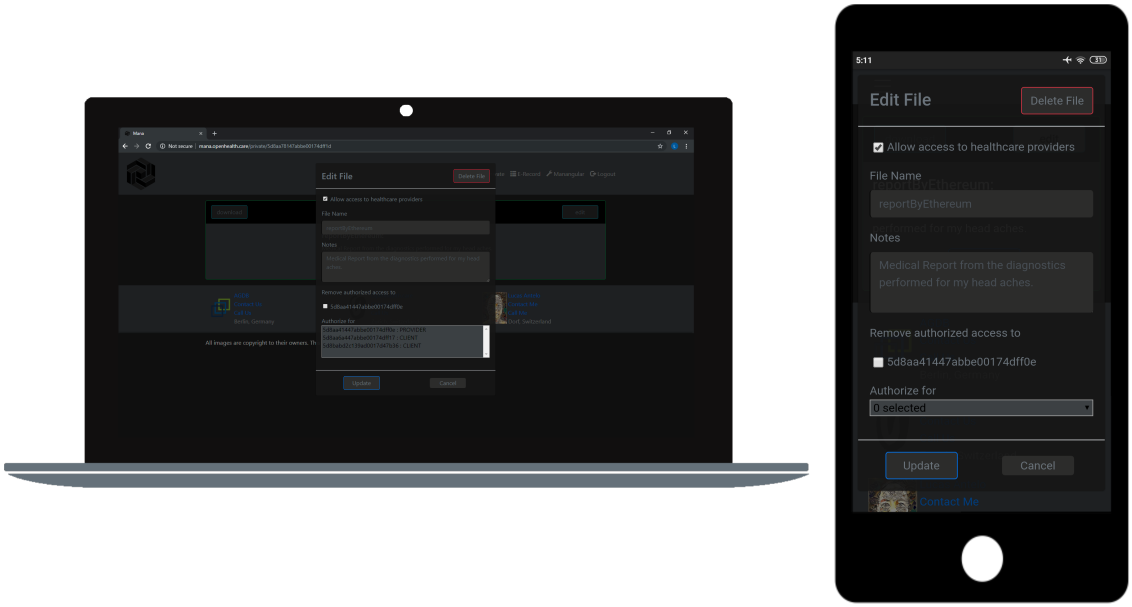

Figure A.12.: The view of the private storage show page in the menu *Private* with a dialog box to edit the respective file.

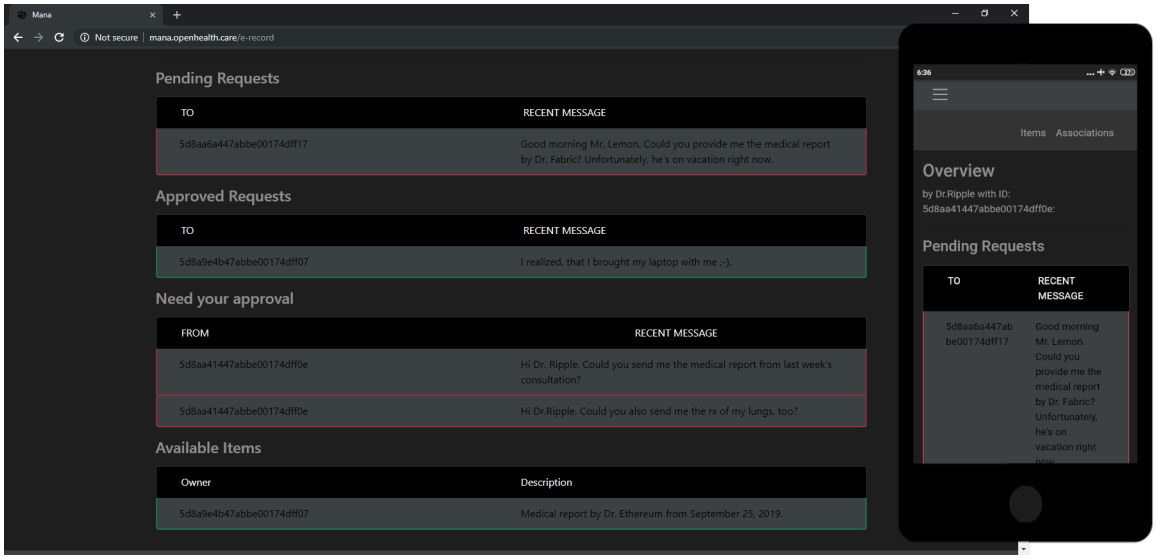

Figure A.13.: The index page of the menu, *E-Record*, gives an overview of all available requests and authorized data items in the HLF network application. The red border signals to the user the need for approval and the green border signals approved requests and accessible data items.

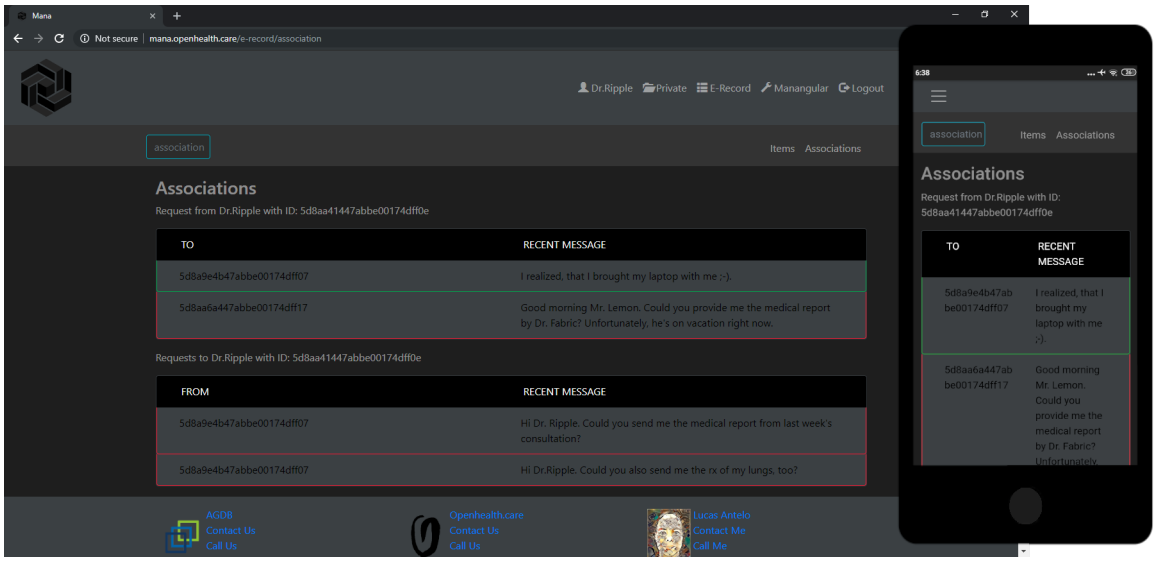

Figure A.14.: The index page of the sub-menu, *Associations*, gives an overview of all available data access requests from and to the current user in the first and second half of the view, respectively.

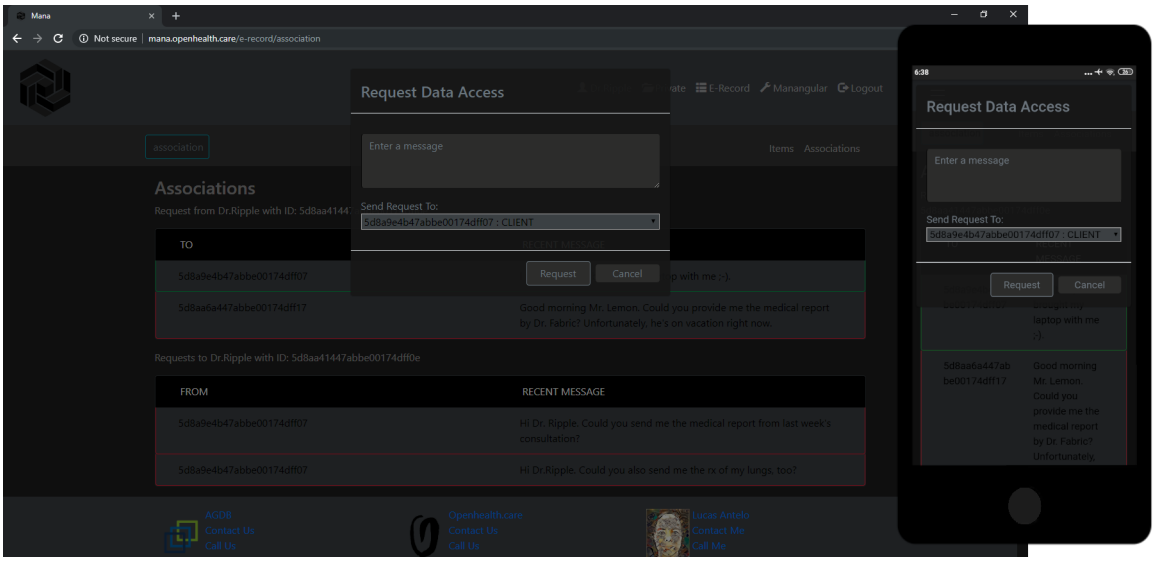

Figure A.15.: The view of the sub-menu, *Associations*, with a dialog box for creating a data access request. The dialog box requires a message and a receiver from the list of available users in the HLF network application.

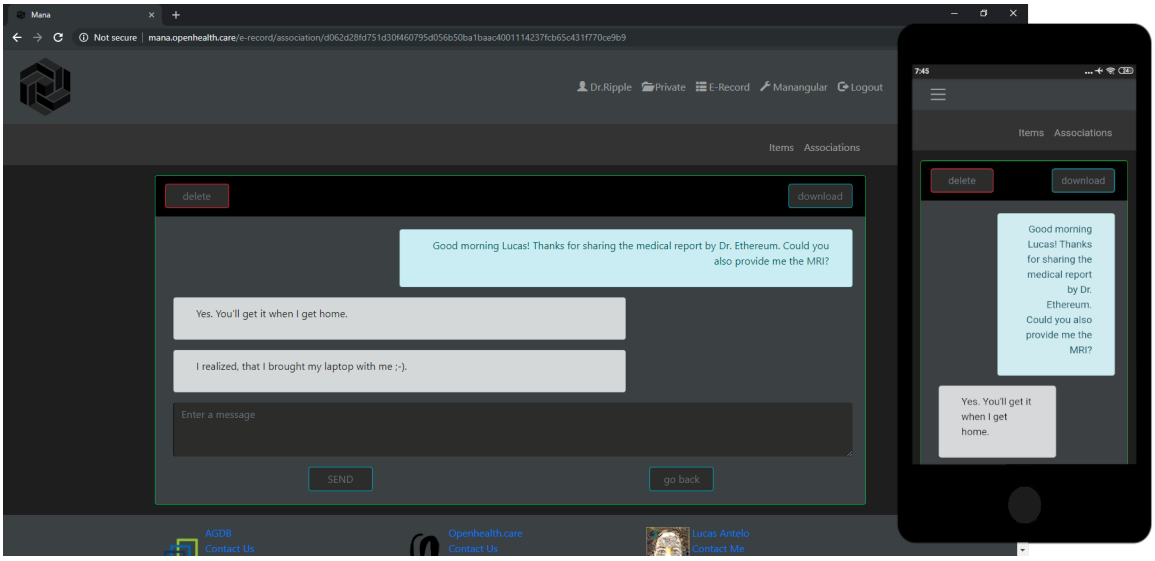

Figure A.16.: The show page in the sub-menu, *Associations*, for an approved data access request from the perspective of a requester. The buttons *download* and *delete*, allows the requester to download the file and to delete the request, respectively. In addition, a messaging service provides a communication channel between the data owner and the requester.

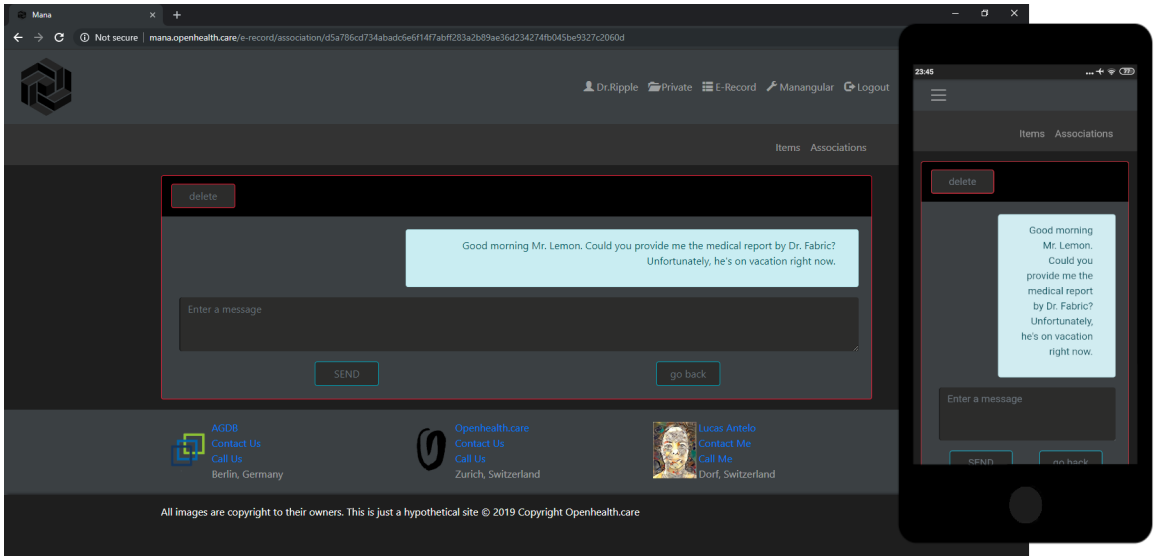

Figure A.17.: The show page in the sub-menu, *Associations*, for a data access request that needs approval from the perspective of a requester and lacks the download option.

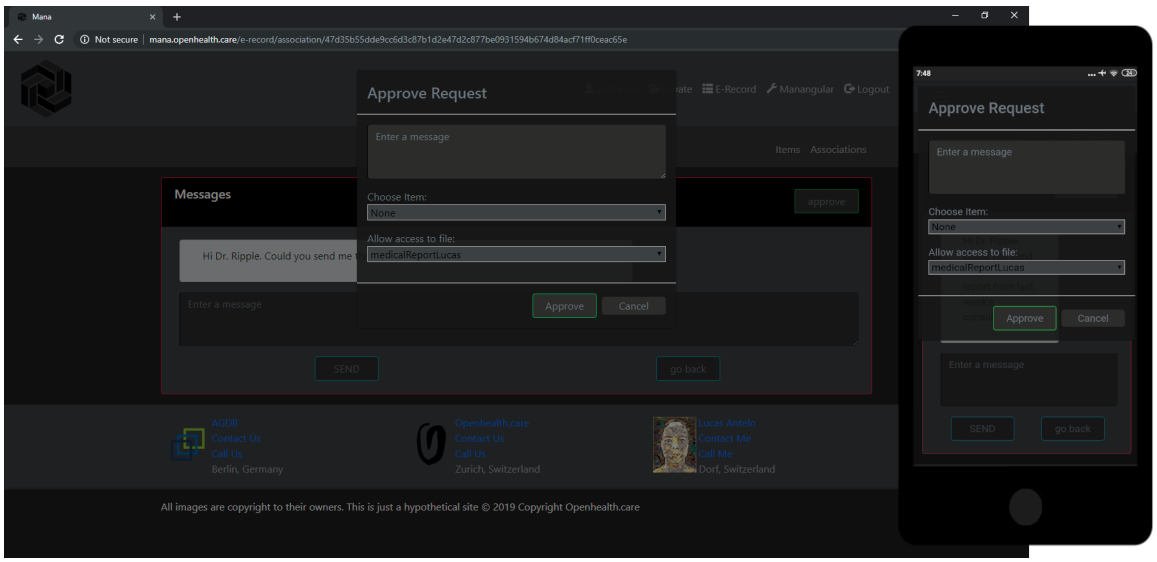

Figure A.18.: The show page in the sub-menu, *Associations*, for a data access request that needs approval from the perspective of the data owner. The button, *approve*, opens a dialog box and requires a message and the respective file. Optionally, the data owner can choose from the list of owned data items instead of a file, but this option is not functional in the prototype, yet.

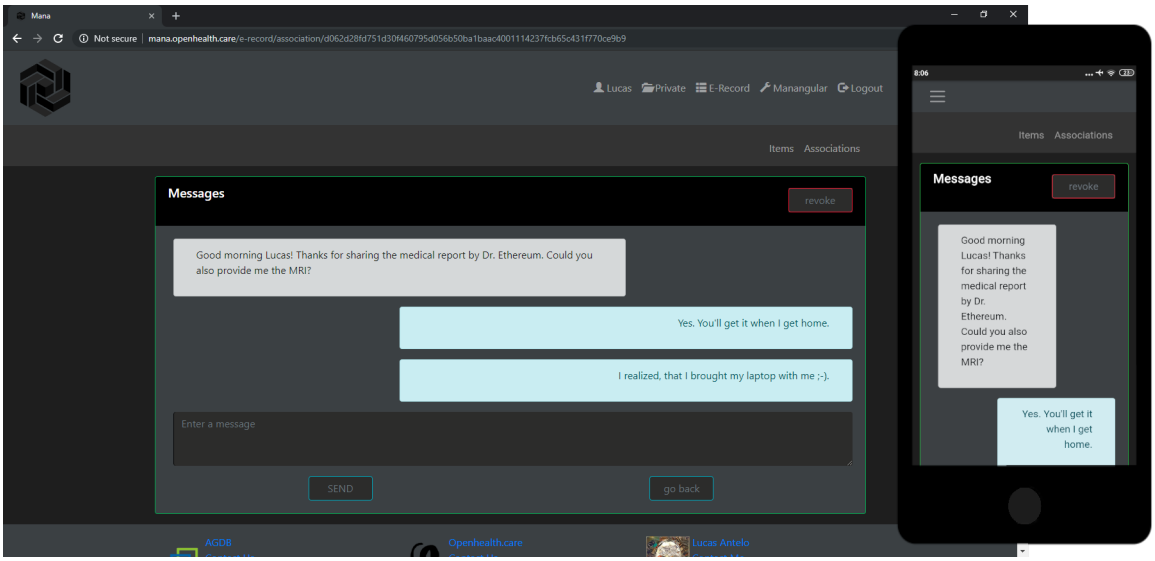

Figure A.19.: The show page in the sub-menu, *Associations*, for an approved data access request from the perspective of the data owner. The button *revoke* allows the data owner to revoke the access authorization, previously granted to the requester.

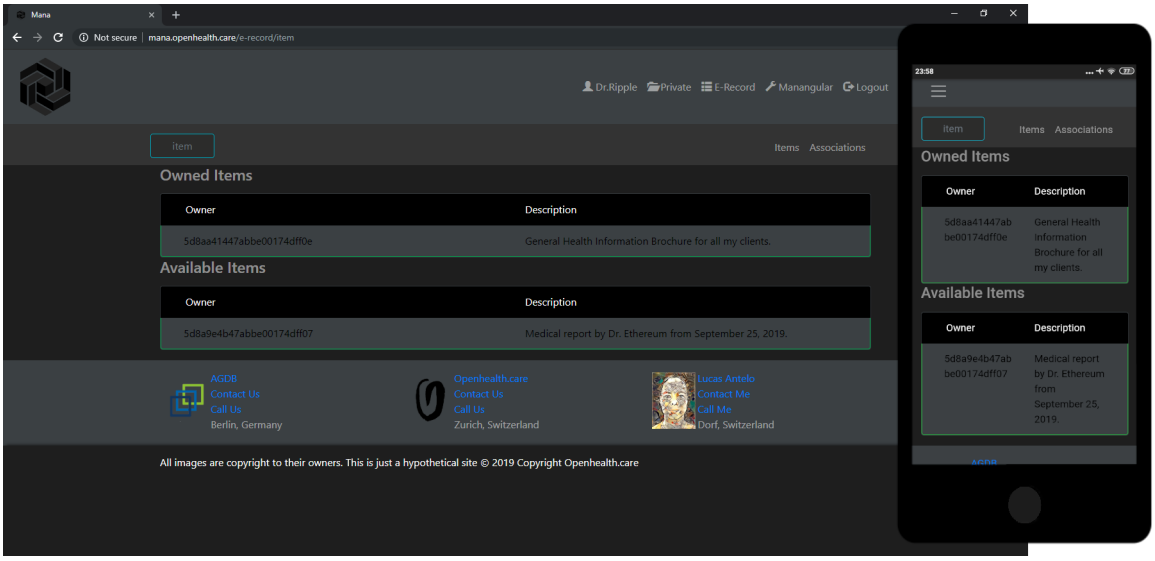

Figure A.20.: The index page of the sub-menu, *Items*, gives an overview of all owned and authorized data items in the HLF network application. The button *item* opens a dialog box in order to publish an owned file in the HLF network application.

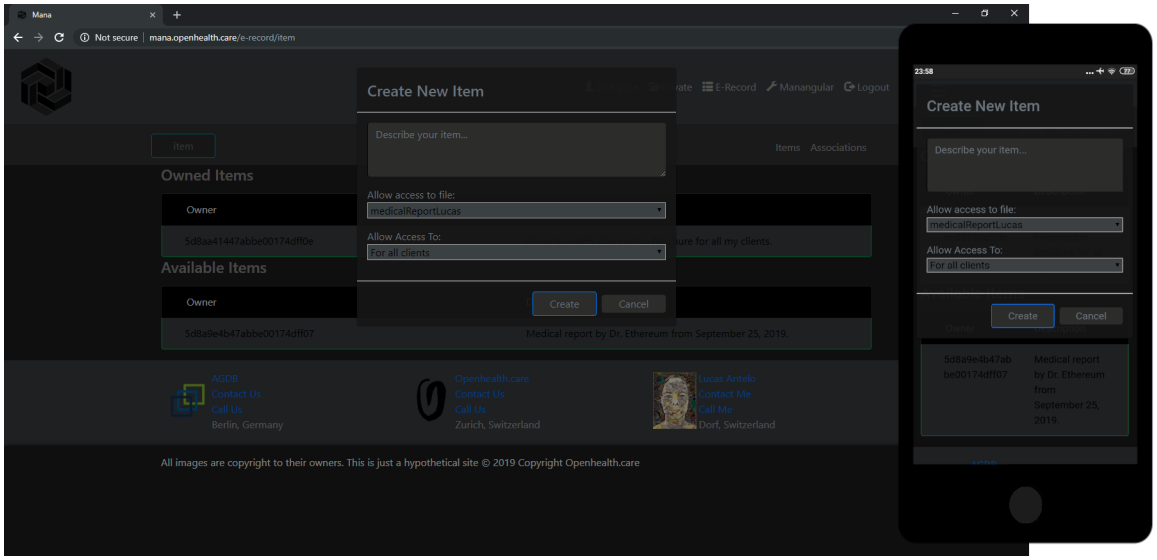

Figure A.21.: The index page of the sub-menu, *Items*, with a dialog box in order to publish a file in the HLF network application. The dialog box requires a description, a file from the list of owned files and a role from the list. The list provides three options, namely: *CLIENT*, *PROVIDER* and *INSURANCE*. The data item is only accessible to users with the given role.

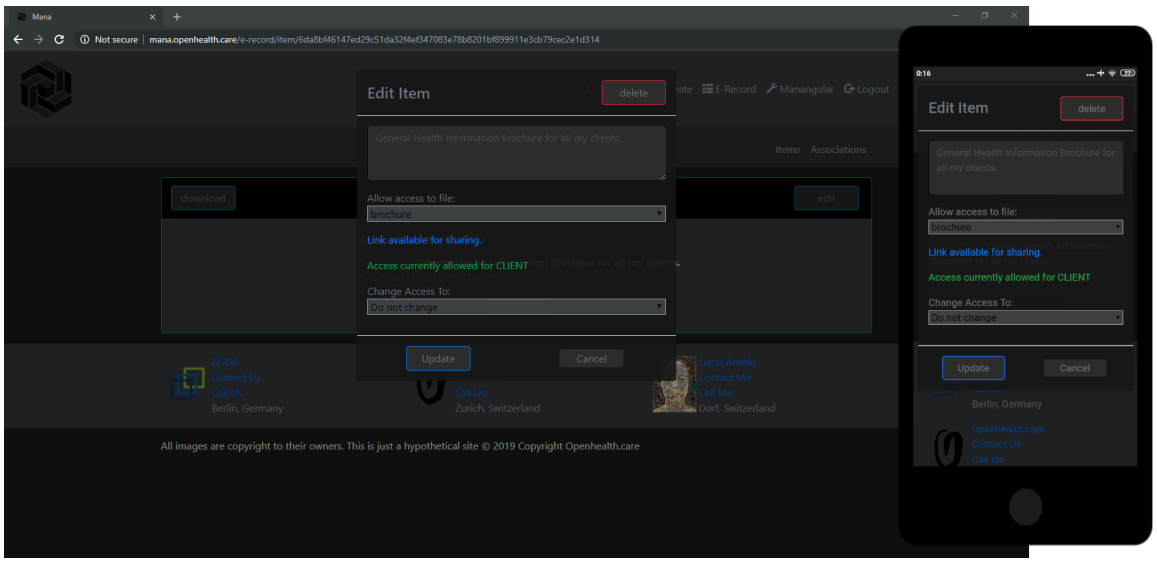

Figure A.22.: The show page in the sub-menu, *Items*, for an owned data item with a dialog box in order to edit the current item. The dialog box presents the description, published file from the list of owned files and further information. In addition, the data owner can change the authorized role from the given list in *Change Access To:*. Finally, the data owner can delete the data item from the HLF network application by clicking the button *delete*.

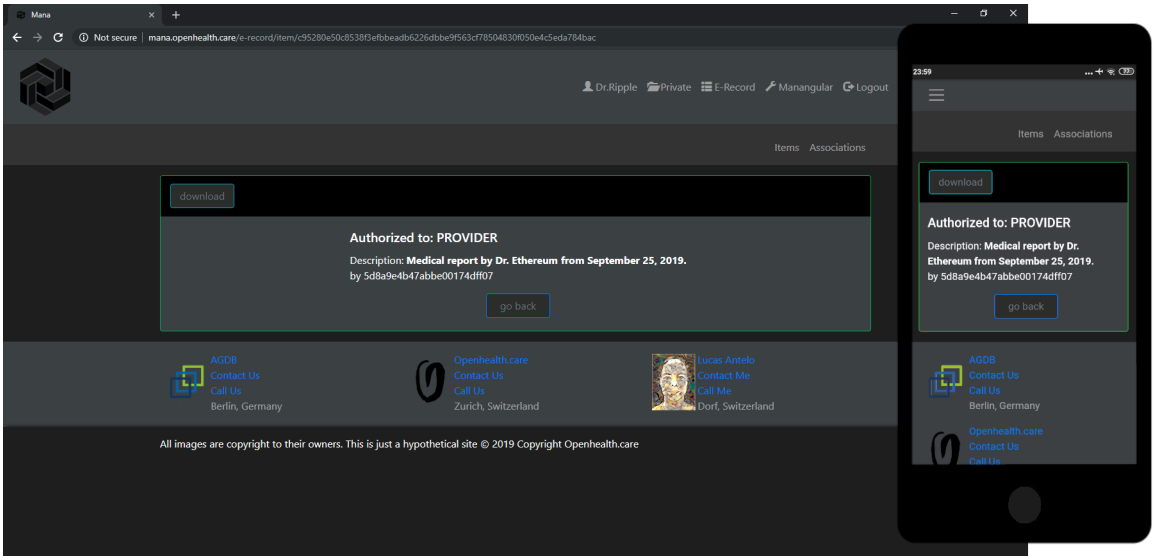

Figure A.23.: The show page in the sub-menu, *Items*, for an authorized data item from the perspective of user that does not own the file. The button, *download*, provides the only option to the user.

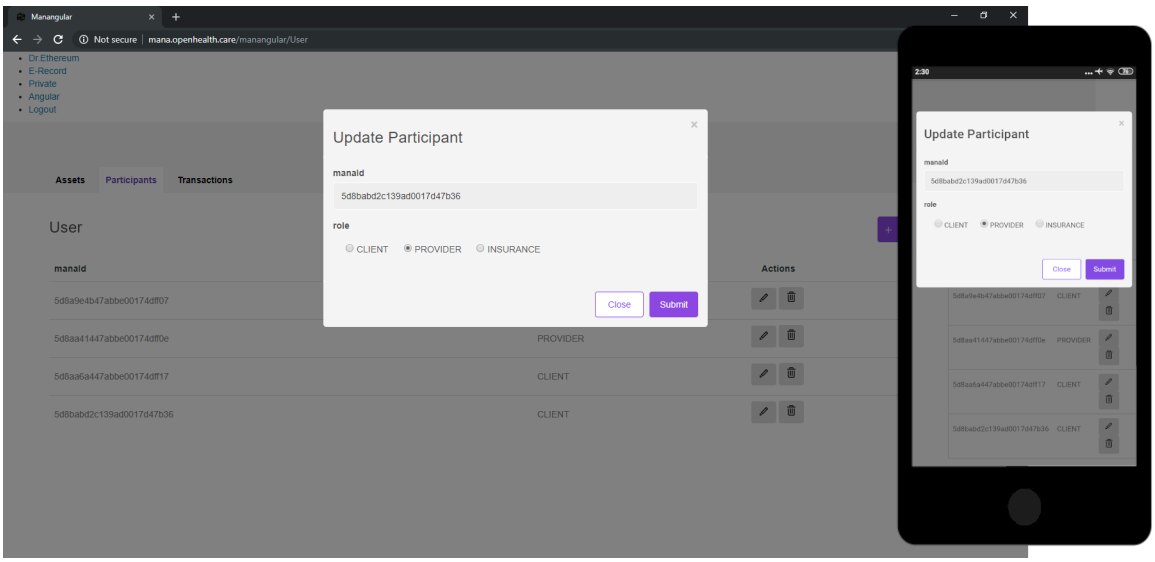

Figure A.24.: The Angular application provides the interface to create, read, update and delete resources and perform transactions in the ledger. From the navigation bar, the menu *Manangular* redirects the user to the Angular application, which is primarily used to update the role and furthermore to test the prototype. This application was automatically generated with Yeoman.

B. Backend

B.1. Data Tier

B.1.1. MongoDB

```
const mongoose = require('mongoose'),
passportLocalMongoose = require('passport-local-mongoose');

const userSchema = new mongoose.Schema({
    username: {type: String, unique: true, required: true },
    email: {type: String, unique: true, required: true},
    active: Boolean,
    token: String,
    files: [{
            type: mongoose.Schema.Types.ObjectId,
            ref: "File"
    }]
});

userSchema.plugin(passportLocalMongoose);
module.exports = mongoose.model("User", userSchema);
```

Source Code B.1: The model definition for the user model is defined in a Javascript file stored on the web server. The Javascript library mongoose provides the methods to define and manage the model in the database. The library passport-local-mongoose adds additional methods to the model used for the authentication and password management.

```
{
    _id: ObjectId("5d8a9e3547abbe00174dff05"),
    files: [ ObjectId("5d8aa75547abbe00174dff1b"), ... ],
    username: "Lucas",
    email: "antelo.lucas@outlook.com",
    active: true,
    salt: "b03df1a...b7c256",
    hash: "f14012...84b585",
    __v: 6,
    token: "NJ2wXG/SlvW2BwGhV+Z4z06aDSKajplB"
}
```

Source Code B.2: A user object stored in the collection *users* in the MongoDB Atlas database. The object includes the hash value of the password and the respective salt for the authentication, an array of available files, the token and additional attributes needed for the prototype.

```
{
    _id: ObjectId("5d8a9e4b47abbe00174dff07"),
    participant: null
    user: Object("5d8a9e3547abbe00174dff05"),
    __v:0
}
```

Source Code B.3: A mana object stored in the MongoDB Atlas database. The attribute *id* represents the used identification string in the HLF network for the respective attribute *user* stored in the collection *users*. The attribute *participant* is a relic from the development and does not fulfill any function. The collection *manas* represents the links between the web application and the HLF network application for the respective users and allows the anonymization of user credentials in the ledger.

```
{
    _id: ObjectId("5d8aa78147abbe00174dff1d"),
    accessible: true,
    ETag: ""d55ac3b3ac3cb44debc819aa9dbd645c"",
    authorized: [ObjectId("5d8aa41447abbe00174dff0e")],
    filename: "reportByEthereum",
    note: "Medical Report from the diagnostics performed
          for my head aches.",
    date: 2019-09-24T23:32:17.696+00:00,
    __v: 1,
    owner : {
        id: ObjectId("5d8a9e3547abbe00174dff05"),
        username: "Lucas"
    },
    path: "Lucas/f24cd3...34459a.pdf"
}
```

Source Code B.4: A file object stored in the collection *files* in the MongoDB Atlas database that includes the owner and the respective path. The attribute *path* represents the key for the respective file stored in Amazon AWS S3. The attribute *owner* includes the id and the associated username, that was retained to simplify the development of the prototype. The attribute *ETag* is a value from the AWS S3 bucket and was retained in order to verify, that the file has been successfully uploaded.

B.1.2. AWS S3 Bucket

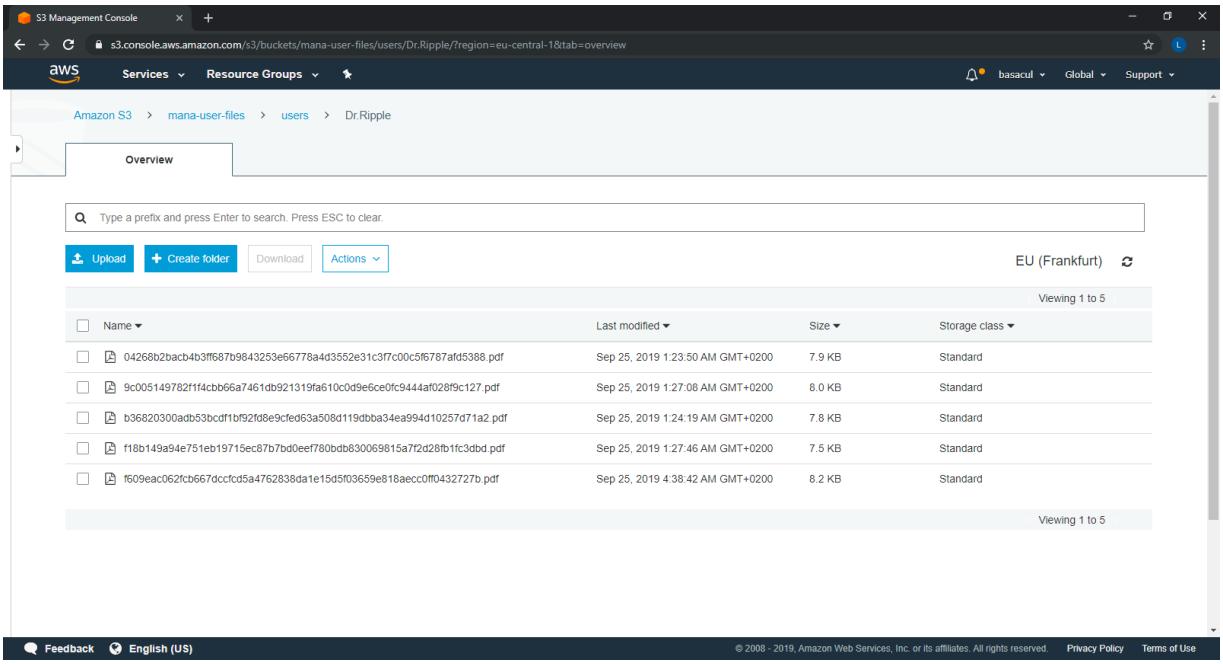

Figure B.1.: The view of the AWS S3 console displays the files that are managed by the user Dr.Ripple. The file names are randomly generated hash values. The key value to retrieve these files is stored in the respective file object in the MongoDB Atlas database and consist of the username and the hash value.

```
router.post('/:id', middleware.isLoggedIn,
            middleware.checkOwnership, (req, res) => {

    File.findById(req.params.id, (error, file) => {
        // middleware.checkOwnership already checks
        // for error and null

        // DOWNLOAD FROM AMAZON AWS S3
        const downloadObject = aws.s3
            .getObject(aws.paramsDownload(file.path))
            .createReadStream();

        const filename = file.path.split('/')[1];

        res.attachment(filename);
        downloadObject.on('error', err => {
            winston.error(err.message);
            req.flash('error', 'File could not ...');
            res.redirect('back');
```

```
20          }).pipe(res);
21      });
22  });
```

Source Code B.5: The source code implements the business logic to download owned files in the menu *Private*. First, the route definition validates, if the user is authenticated and owns the respective file given by the parameter *:id*. Second, it retrieves the file object from the MongoDB Atlas databases and uses the Javascript library aws-sdk to download the document from the AWS S3 bucket. Third, the file is prepared to be served to the user. Finally, the user is redirected back to the previous view, where a window pops up to download the document.

B.1.3. Ledger

```
1  namespace care.openhealth.mana

3  participant User identified by manaId {
4    o String manaId
5    o Role role
6  }

8  /**
9  link remains unencrypted to facilitate prototyping
10  */
11 asset Item identified by itemId {
12   o String itemId
13   o String description
14   o Role role
15   o String link
16   --> User owner
17 }

19 enum Role {
20   o CLIENT
21   o PROVIDER
22   o INSURANCE
23 }

25 /**
26 link remains unencrypted to facilitate prototyping
27  */
28 asset Association identified by associationId {
29   o String associationId
```

```
30    o Boolean approved
31    o Message[] messages
32    o String link optional
33    --> User to
34    --> User from
35    --> Item item optional
36  }

38  concept Message {
39    o String from // represents the manaId
40    o DateTime date
41    o String message
42  }

44  transaction CreateAssociation {
45    --> User to
46    --> User from
47    --> Item item optional
48    o String message
49    o String associationId
50  }

52  event AssociationCreatedEvent {
53    o String associationId
54  }
```

Source Code B.6: Excerpt of the business model definition of the HLF network application. The domain specific programming language is similar to Java and includes distinct features such as *o*, that represents the letter 'o' for owned. Moreover, the arrow –> represents a pointer to the entry in the world state managed by CouchDB. The keywords *asset*, *participant*, *transaction* and *event* are essential for the definition of the business model.

B.2. Logic Tier

B.2.1. Account Settings

```
router.put("/privacy", middleware.isLoggedIn, (req, res)=> {
  let privacy_settings = sanitize_privacy(req);
  Privacy.findOneAndUpdate({user: req.user._id},
        privacy_settings, (err, privacy) => {
    if(err){
      winston.error(err.message);
      req.flash('error', 'Could not update privacy settings.');
    }else{
      winston.info('Privacy settings were updated.');
      req.flash('success', 'Privacy settings updated.');
    }
    res.redirect('/account');
  })
});
```

Source Code B.7: Express handles the route to update the privacy options. The middleware validates, if the user is authenticated. Then it sanitizes the input from the user and updates the respective privacy object in the MongoDB Atlas database. A flash message is generated that informs the user, if the request was successful or not. Finally, the user is redirected to the index view of the account settings.

```
const nodemailer = require('nodemailer');
const config = require('../config/mailer');

const transport = nodemailer.createTransport({
  service: 'Mailgun',
  auth: {
    user: process.env.MAILGUN_USER || config.MAILGUN_USER,
    pass: process.env.MAILGUN_PASS || config.MAILGUN_PASS
  },
  tls: {
    rejectUnauthorized: false
  }
});

module.exports = {
    sendEmail(from, to, subject, html){
      return new Promise((resolve, reject) => {
        transport.sendMail({from, subject, to, html},
```

```
19              (err, info) => {
20                if(err){
21                  reject(err);
22                }else{
23                  resolve(info);
24                }
25            });
26          });
27        }
28  };
```

Source Code B.8: The web application defines the function to send emails in a separate file stored in the folder *utils* found in the web server. This feature requires the library nodemailer and the configuration file defined in the folder */config/mailer* or in the environment variables defined in the web server.

B.2.2. Private Data Storage

```
1  router.post("/", middleware.isLoggedIn,
2                   middleware.upload.single('upload'),
3                   (req, res, next) => {
4
5    sanitize_text(req);
6
7    const file = req.file;
8
9    if (!(file && req.body.file.filename)) {
10
11     const error = new Error('Provide file and file name.');
12         winston.error(error.message);
13         req.flash('error', error.message);
14         res.redirect('/private');
15
16   } else {
17
18     File.create(req.body.file, function (err, newFile) {
19
20       if (err) {
21
22         winston.error(err.message);
23                 req.flash(err.message);
24         res.redirect("back");
25
26       } else {
27
28     newFile.owner.id = req.user._id;
29     newFile.owner.username = req.user.username;
30     newFile.path = `${req.user.username}/${req.file.filename}`;
31
32     const uploadObject = aws.s3.putObject(aws.params(newFile.path)).promise();
33
34     uploadObject.then(data => {
35       winston.info('File upload to s3 successful.');
36       newFile.ETag = data.ETag.toString();
37
38       if (req.body.authorizedUser) {
39         newFile.authorized = req.body.authorizedUser.map(item => {
40                 return mongoose.Types.ObjectId(item.split(':')[0].trim());
41         });
42       }
43
44
45       User.findById(req.user._id, function (errUser, user) {
46         if (errUser) {
47           winston.error(err.message);
```

```
48                 req.flash('error', err.message);
49                 res.redirect('/');
50           } else {
51             // in case no user with given id there is simply no error thus null testing
52             if(user){
53                 newFile.save();
54                 user.files.push(newFile);
55                 user.save();
56                 winston.info('File Upload successfully completed.');
57                 req.flash('success', 'New file uploaded');
58             }else{
59                 winston.error('User account NOT found to push file id to files array.');
60                 req.flash('error', 'User not found to push file!');
61             }
62             winston.info('New file was uploaded by user to its respective folder.');
63             res.redirect('/private');
64           }
65         });
66       }).catch(error => {
67         winston.error(error.message);
68         req.flash('error', 'File could not be uploaded on amazon aws S3');
69         res.redirect('back');
70       }).finally(function() {
71         fs.unlink(`temp/${file.filename}`, (err) => {
72           if (err) {
73             winston.error(err.message);
74           }else{
75             winston.info('File successfully removed from webserver');
76           }
77         });
78       });
79         }
80       });
81     }
82   });
```

Source Code B.9: This code snippet represents the business logic for the file upload managed by the web server. A file object is generated in the MongoDB Atlas database and the values are updated accordingly. Next, the file is uploaded to the AWS S3 bucket. If the upload is successful, then the current user object is updated and the temporary file is removed from the web server. This route definition is a good example of a callback hell, that is caused by the sequential execution of several Promises. A Promise is a Javascript object, that provides the methods for the execution of asynchronous tasks.

B.2.3. Data Exchange

```
1   middleware.checkIfAuthorizedAssociation = function (req, res, next) {
2   let association;
3
4     //retrieve respective association from HLF
5     hlf.getAssociationById(req.params.associationId).then(responseAssociation => {
6
7       association = responseAssociation.data[0];
8
9       // create file Id in order to retrieve the respective file object
10      const fileId = mongoose.Types.ObjectId(association.link.split('files/')[1]);
11
12
13      // retrieve ManaId of logged in user
14      Mana.findOne({user: req.user._id}, (errorMana, mana) => {
15
16        if(errorMana){
17
18          winston.error(errorMana.message);
19          req.flash('error', 'Could not find Mana');
```

```
20          res.redirect('back');
21        }else{
22
23          File.findById(fileId, (errorFile, file) => {
24
25            if(errorFile){
26
27              winston.error(errorFile.message);
28              req.flash('error', 'File not found');
29              res.redirect('back');
30
31            // check if current user's manaID equals the requester's manaId in association
32            }else if(mana._id.toString() != association.from.split('#')[1]){
33
34              winston.error('Current user s manaId and authorized manaId do not match');
35              req.flash('error','Mana ID mismatch.');
36              res.redirect('back');
37
38            //check if user by manaId is in the authorized array
39            }else if(!file.authorized.includes(mana._id)){
40
41            //else flash message and redirect
42              winston.error('User is not authorized to access file.');
43              req.flash('error', 'You are not authorized.');
44              res.redirect('back');
45
46            }else{
47
48              // otherwise user is authorized : next
49              // pass file path to call aws method. Done via request body
50              req.filePath = file.path;
51              next();
52            }
53          });
54        }
55      });
56    }).catch(error => {
57      winston.error(error.message);
58      req.flash('No HLF Connection.')
59      res.redirect('back');
60    });
61  };
```

Source Code B.10: The middleware checks, if the user is authorized to access the file from a given data request. First, it fetches the respective data request from the ledger and retrieves the link. Second, it retrieves the current user's manaID from the MongoDB Atlas database. Third, it retrieves the associated file object and checks, if the user is enlisted as authorized user. Only if the validation is successful, the access to the file is granted.

```
2   /**
3    * Grant existing Association by providing a message or even a link
4    * @param {care.openhealth.mana.GrantAssociation} associationData
5    * @transaction
6    */
7   async function grantAssociation(associationData) {
8       let association;
9
10      // 1. Get respective Association
11      let associationRegistry = await getAssetRegistry(`${namespace}.Association`);
12      let associationExists = await associationRegistry.exists(associationData.association.associationId);
13
14      if (associationExists) {
15
16          association = await associationRegistry.get(associationData.association.associationId);
17          association.approved = true;
18
19          // 2. Create concept Message
20          let message = factory.newConcept(namespace, "Message");
21          message.from = associationData.from;
22          message.message = associationData.message;
23          message.date = new Date(Date.now());
```

```
25          // 3. Unshift message to respective Association
26          association.messages.unshift(message);
27
28          // 4. Update link in Association
29          association.link = associationData.link;
30
31          // 5. Update item if it exists
32
33          if (associationData.item) {
34              association.item = associationData.item;
35          }
36
37          // 6. Update Association Registry
38          associationRegistry.update(association);
39      }
40
41
42      // 7. Emit AssociationEvent
43      let event = factory.newEvent(namespace, 'AssociationGrantedEvent');
44      event.associationId = association.associationId;
45      emit(event);
46  }
```

Source Code B.11: The function in the chaincode definition defines the transaction for granting access requests based on a given association. The association represents the data access request by the requester.