Blockchain-Based System for Document Storage and Signatures of Consenting Forms

University Fernando Pessoa

Ruben Pinto

Faculty of science and Technology

University Fernando Pessoa

A thesis submitted for the degree of

Master's in Computer Engineering,
Mobile Computing Branch

Professor Doutor Christophe Soares
Professor Doutor Ivo Pereira

24/09/2024

Abstract

Nowadays, decentralized models connecting various users and entities have gained prominence across the healthcare, finance, and Supply Chain Domains. Decentralized applications represent a transformational approach to data management and transaction execution, emphasizing security, data integrity, and transparency. At the core of these applications lies the blockchain system. This decentralized architecture supports a framework that guarantees data immutability and ensures network-wide transparency through consensus mechanisms.

This work aims to explore the application of a blockchain-based system for managing, storing, and signing consent forms within a decentralized framework. By leveraging smart contracts, the system facilitates the creation, modification, deletion, and storage of documents issued by authorized medical entities. Patients can sign these documents, with every alteration and transaction meticulously tracked and recorded, enhancing privacy and data integrity. In addition to these benefits, a private system with role-based access control restricts access to consent forms, as determined by the medical authority that created the documents.

The proposed project of this theses aims to leverage these benefits by implementing a Corda application, a blockchain-based solution designed for managing consent forms within the healthcare ecosystem. This solution will enable healthcare providers, patients, and other stakeholders to securely access, share, and manage sensitive medical data with full confidence in its integrity and privacy. By incorporating decentralized technology, the project seeks to create a system where patient consent is stored immutably on the blockchain, ensuring that no unauthorized modifications can be made.

Furthermore, the evaluation and testing section of this work reinforces the access security and permission enforcement mechanisms that are proposed and implemented. Rigorous tests and practical examples demonstrate the system's ability to protect patient data and uphold privacy standards, ensuring that only authorized users can interact with sensitive information.

Resumo

Hoje em dia, os modelos descentralizados que conectam vários utilizadores e entidades ganharam proeminência nos setores da saúde, finanças e nas cadeias de fornecimento. As aplicações descentralizadas representam uma abordagem transformativa na gestão de dados e na execução de transações, enfatizando a segurança, a integridade dos dados e a transparência. No centro destas aplicações está o sistema blockchain. Esta arquitetura descentralizada apoia um framework que garante a imutabilidade dos dados e assegura transparência na rede através de mecanismo de consenso.

Este trabalho tem o objetivo de explorar a aplicação de um sistema baseado em blockchain para gerir, guardar e assinar consentimentos informados dentro de uma framework descentralizada. Através do uso de smart contracts, o sistema permite a criação, atualização, eliminação e armazenamento de documentos emitidos por entidades autorizadas. Os pacientes podem assinar esses documentos, onde cada alteração e transação será rastreada e registada, garantindo a privacidade e a integridade dos dados. Em adição a estes benefícios, um sistema privado com um controle de acesso baseado em função restringe o acesso de consentimentos informados determinados pela autoridade médica que os criou.

A proposta desta dissertação implementa uma aplicação Corda, projetada para gerir consentimentos informados no setor da saúde. A solução possibilita que prestadores de cuidados, pacientes e outras partes interessadas acedam e partilhem dados médicos com segurança e total confiança na sua integridade. Com o consentimento armazenado de forma imutável no blockchain, evita-se qualquer modificação não autorizada.

Além disso, a secção de avaliação e testes deste trabalho reforça os mecanismos de segurança de acesso e aplicação de permissões, que foram propostos e implementados. Os testes rigorosos e exemplos práticos demonstram a capacidade do sistema de proteger os dados dos pacientes e manter padrões de privacidade, garantindo que apenas os utilizadores autorizados podem interagir com as informações confidenciais.

Acknowledgements

I would like to acknowledge the company ByMe that provided me with the opportunity to work with them, the professors who offered their guidance and support throughout my journey.

Contents

List of Figures

List of Tables

Nomenclature

ABAC Attribute-Based Access Control

ACM Association for Computing Machinery

API Application Programming Interface

CA Certificate Authority

CapBAC Capability-Based Access Control

CordApp Corda Application

CPB Corda Package Bundle

CPI Corda Package Installer

CPK Corda Package

CRUD Create Read Update Delete

DAC Discretionary Access Control

DApp Decentralized Application

DeFi Decentralized Finance

DLT Distributed Ledger Technology

EVM Ethereum Virtual Machine

HTTPS HyperText Transfer Protocol Secure

IEEE Institute of Electrical and Electronics Engineers

IoT Internet of Things

JAR Java Archive

JWT JSON Web Token

MGM Membership Group Manager

NFT Non-Fungible Token

PBFT Practical Byzantine Fault Tolerance

PoA Proof of Authority

PoET Proof of Elapsed Time

PoH Proof of History

PoS Proof of Stake

PoW Proof of Work

RBAC Role-Based Access Control

REST API Representational State Transfer Application Programming Interface

UCON Usage Control

URF User Requirement Functional

URNF User Requirement Non-Functional

URS User Requirement for Software and Hardware

VNodes Virtual Nodes

Glossary

Backend The server-side component of the application.

Base64 A binary-to-text encoding scheme that represents binary data in an ASCII string format.

Command An operation that can be performed on the Corda ledger, such as create, update, or delete.

Contract A set of rules and conditions used to validate transactions in Corda.

Decryption The process of converting encrypted data back into its original form.

DocumentState A state representing a document within the Corda ledger.

Encryption The process of converting data into a secure format to prevent unauthorized access.

Frontend The user interface layer of the application.

JWT A token used for user authentication and authorization in the application.

Notary An entity responsible for validating transactions in the Corda network.

RecordState The status of a document, which can be active or deleted.

Role A permission assigned to a user.

RolePermissions A set of rules determining what actions a role can perform on a state.

State A representation of data stored in the Corda ledger.

SubFlow A smaller unit of work executed within a larger flow in Corda.

Swagger A tool used to generate interactive API documentation for REST APIs.

Transaction An operation that involves one or more states in the Corda ledger.

UserState A state representing a user's registration and role in the application.

Virtual Node An isolated execution environment within the Corda network.

Workflow A flow that can be initiated by a client in the Corda network.

X500 Series of standards for directory services.

Chapter 1

Introduction

The use of digital program's and systems in the healthcare industry is a growing tread that is driven by the need for a more efficient, secure and engaging model of operation. As healthcare providers and institutions increasingly rely on digital solutions for managing patient data, medical records, and administrative tasks, concerns over data security, privacy, the integrity of sensitive information and access have also heightened (Stoumpos *et al.*, 2023; Yeung *et al.*, 2023).

Different solutions have emerged to quell these issues, resulting in a variety of innovative approaches aimed at creating a more sustainable solution that is able to address the issues highlighted. Among this, Blockchain-based technologies have offered a solution to maintain a consistent and reliable form of recording records. By utilizing a decentralized ledger system, blockchain ensures that data is not only securely stored but also immutable, meaning it cannot be altered without detection (Baskar & Gopirajan, 2023).

With its ability to ensure data consistency, blockchain represents a viable long-term solution to some of the most pressing challenges in the healthcare industry. As adoption grows, it has the potential to significantly improve how healthcare data is managed, protected, and utilized across various sectors.

1.1 Context

Blockchain-based technologies have been created and developed with the aim of creating a decentralized system where digital data is stored, encrypted using dual asymmetric key algorithms, and a hash function that binds each block with the history of the previous blocks (Centobelli *et al.*, 2021), thus guaranteeing a continuous line of a history of transactions that can be verified by the comparison of the previous block hash and the hash stored in the current hash. The following approach provides data integrity, trust, and transparency. By using consensus algorithms to distribute pending insertions on the blockchain, it provides several different entities to verify and decide the result (Tapscott & Tapscott, 2018).

According to a FinTech survey (Renduchintala *et al.*, 2022), the recent COVID outbreak has increased the interactions made between people and financial services, resulting in the development of blockchain-based applications to create and aid existing services, ease concerns about security, and resolve issues that the increased traffic created after the COVID pandemic. This development is not exclusive to the financial sector; the supply chain, health care, and IoT sectors (Ashraf & Heavey, 2022; Chen *et al.*, 2018) have also emerged as focal points for developing solutions using blockchain applications because of the common need to have a line of custody and history between participants while also keeping the same data secure from outside threats like counterattacks. According to a systematic literature review in the health care industry (Tandon *et al.*, 2020), these attacks pose a real and dangerous threat to the privacy of patients.

In the healthcare industry, blockchain technology offers transformative potential by addressing critical issues related to data security (Taylor *et al.*, 2020), patient privacy, and efficient data management (Yaqoob *et al.*, 2021). One of the primary applications is the secure and transparent management of patient records, especially medical histories and prescriptions executed (Jaya *et al.*, 2022; Haleem *et al.*, 2021). Traditional centralized systems often suffer from data breaches and unauthorized access, compromising patient confidentiality (Keshta & Odeh, 2021). Blockchain ensures that patient data is immutable and accessible only to authorized parties. This reduces the risk of data tampering and unauthorized access, thereby enhancing patient trust and data integrity.

Digital applications for electronic signatures have been developed by a variety of institutions, both public and private. Public institutions, such as the Portuguese state, have created digital signature applications utilizing identification cards (government, 2024), enabling secure and verified electronic signatures for official documents. On the other hand, private companies have developed signature pads (e-signature solutions, 2024) and REST APIs (digitalsign, 2024) that facilitate the capture of electronic signatures and the generation of certificates to authenticate signed documents. This application has been developed through the use of Rest APIs, in the case of the signing desk, or through integrating with an application, such as the Portuguese state solution.

1.2 Problem and Motivation

The problem that this work seeks to solve originated in the need to guarantee a secure, private, and integral system to store, manage, and access information that users create and view in the health care industry, based on the cross-domain design in (Kuo & Shieh, 2020), specifically documents signed by the participants. Current systems often fall short in providing the required security and privacy, making sensitive data vulnerable to unauthorized access and cyberattacks (Fekih & Lahami, 2020).

The motivation for this work and project is to aid in the development of new solutions

to the applications that are the current state of the art. In addiction, the project seeks to be implemented in an important sector of society with the collaboration of a company in this sector, ByMe. With this relationship, it is possible to present a more plausible application, reducing some of the technical limitations.

1.3 Proposed Work

The aim of this project is to develop an application, using the mentioned Corda framework, to be used in the creation and storage of consent forms, managing access to those documents for the selected participants chosen by the creator of the document, and keeping a history of transactions made between the creation and the last alteration made using blockchain technology in the form of ledgers. The system has users registered in nodes that represent them and are visible to other nodes. It uses RBAC (Sajal K. Das & Zhang, 2012) to control access and actions based on the role adopted on registration; then, the user can be authorized and receive a temporary token to interact with its virtual node and perform actions related to the document states. The project uses Kubernetes (Burns *et al.*, 2022) to create the basic building blocks to instantiate the database for persistence and the Kafta Messenger for communication and then uses Helm to install all modules that contain the Corda application.

1.4 Structure of the Work

The subsequent sections will present a detailed examination of the current state of blockchain technologies and their application developments, describe the proposed Corda application solution, analyze the implementation through testing scenarios and added features, and conclude with insights and recommendations for future developments in the five main sections.

Section 2 will detail the current state of blockchain technologies and application developments, providing a comprehensive overview of the latest advancements and use cases in various sectors.

Section 3 will introduce and elaborate on the proposed solution, focusing on the development and functionalities of the Corda application designed to meet the project's objectives.

Section 4 will analyze the implemented project through various testing scenarios, highlighting the added features and assessing the system's performance against the proposed objectives.

Section 5 will provide a conclusion summarizing the findings, discussing the implications of the results, and offering recommendations for future work and potential improvements.

1.5 Progress of the Work

To illustrate the progress of the project, a Gantt diagram has been created to present a visual representation of the project's activities, milestones, and their respective durations. It highlights the sequential order of tasks, phases, and overlaps between each flow.

Table 1.1: Task Schedule with Start, End, and Duration

Task	**Start Date**	**End Date**	**Duration (Days)**
Task: Topic Selection for Project	01-09-2023	15-09-2023	15
Task: Literature Review of Blockchain Technology	15-09-2023	01-11-2023	47
Task: Application Review of Blockchain Technology	01-11-2023	04-12-2023	34
Task: Proposal Writing for Corda Solution	01-12-2023	07-12-2023	7
Task: Proposal Approval by ByMe	07-12-2023	07-12-2023	0
Task: Set Requirements for Project	07-12-2023	16-12-2023	10
Task: Application Implementation	16-12-2023	01-07-2024	198
Task: Project Presentation to ByMe	01-07-2024	01-07-2024	0
Task: Testing Phase	01-07-2024	01-08-2024	32
Task: Review Phase	01-08-2024	31-08-2024	31
Task: Thesis Writing	01-09-2024	31-10-2024	61
Task: Project Completion	31-10-2024	31-10-2024	0

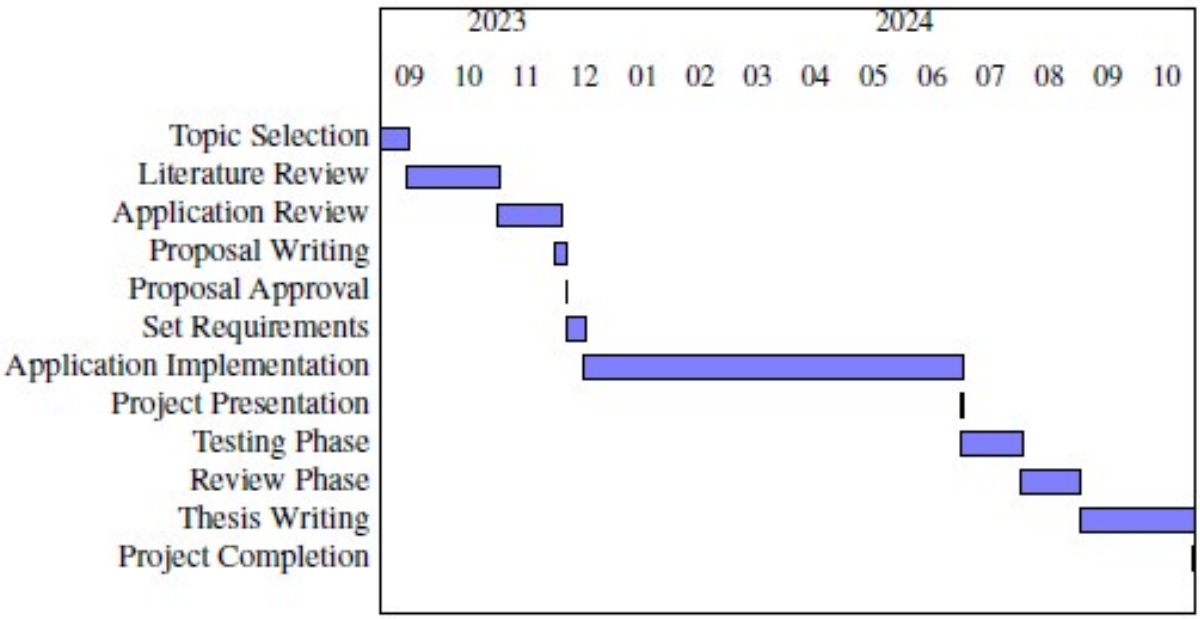


Figure 1.1: Gantt Chart for Task Schedule

The diagram 1.1 and the table 1.1 represent the 4 main phases of the project: the literature review, technology implementation, tests and review, and thesis writing. Each phase is divided into tasks that inform about each step of the completion of the project and the corresponding communication between the company and the project.

The literature review focuses on gathering and analyzing relevant research materials to establish a strong foundation for the project. This phase includes topic selection for the project, literary review for blockchain technology, application review of blockchain technology, and the writing and approval of the Corda Solution.

The technology implementation phase focuses on the implementation of the Corda application. This phase is divided on the setting requirements for the project, the application implementation, and the presentation of the solution to the company ByMe.

The tests and review phase focuses on validating the implementation and ensuring it meets the intended objectives with testing and review phases.

The final phase is the thesis writing and documentation of all findings, methodologies, and results to create a comprehensive thesis.

Chapter 2

State of the Art

In this section, the current state of the art involving blockchain-based models, frameworks, and applications developed will be presented. Additionally, this section outlines the selection process for the specific application utilized in this work, detailing the criteria and reasoning behind the choice.

2.1 Search Strategy

This study employs a systematic approach to gather and research literature relevant to blockchain technology and its applications. The methodology encompasses several detailed steps to ensure a thorough and rigorous examination of the topic.

The initial phase involved developing a comprehensive search strategy to identify relevant relevant scholarly literature. This strategy began with the identification of key terms to find pertinent information, including "blockchain", "blockchain healthcare", "private blockchain", "Hyperledger Fabric" and "applications blockchain".

Subsequently, a selection of academic databases was utilized to source the literature. These databases included IEEE Xplore (of Electrical & Engineers, 2024), ACM Digital Library (for Computing Machinery, 2024), ScienceDirect (Elsevier, 2024), and Google Scholar (Google, 2024b).

To ensure the relevance and currency of the information, the inclusion criteria specified the selection of articles, works, and books published between 2019 and 2024, focusing on the theoretical foundations, technical implementations, and practical applications of blockchain technology. Additionally, literature from the past twenty years was considered for topics such as security, encryption, and global analyses, provided they were written in English.

Following the literature search, a review was conducted that collected and analyzed key points made in various comparisons of blockchain-based applications. The review highlighted the benefits and the specific technical challenges faced by different systems, including security vulnerabilities, scalability issues, computational inefficiency, and the balance between privacy

and transparency.

To ensure the reliability and validity of the gathered literature, a critical appraisal was conducted. This involved evaluating the methodological rigor and credibility of the selected studies. The assessment considered whether the objectives defined initially were met, if the proposed frameworks had practical implementations, if they were analyzed using objective measurements, and if the results were significant.

The reviewed literature was systematically evaluated to understand how blockchain-based systems are applied and the specific benefits gained from various studies. The findings from these studies were then compared to identify commonalities and distinctions in their approaches and outcomes.

Based on this comparative analysis, a particular application was selected as the foundational element for the implementation of this work. This chosen application was selected for its demonstrated effectiveness and alignment with the goals defined in the work.

After the analysis and compilation, it was developed into a coherent narrative that involves points made in the cited work relevant to the work, detailing the evolution and status of the given technology.

2.2 Types of Blockchain

Blockchain technology, initially conceptualized as the underlying framework for cryptocurrencies, has rapidly evolved into a multifaceted innovation with far-reaching implications across industries. At its core, a blockchain is a distributed ledger system that enables the secure and transparent recording of transactions in a decentralized manner (J *et al.*, 2023; Cao *et al.*, 2020). As the technology matured, different types of blockchain emerged, each tailored to specific requirements, ranging from open and permissionless networks to controlled and permissioned environments (Zeba *et al.*, 2023; Monrat *et al.*, 2019).

2.2.1 Public Blockchain

Public blockchains are characterized by their open access towards any participants or public viewers that have access to the network, thus allowing them to access and create transactions and view the complete history of the ledger, or simply permissionless networks. Two commonly used consensus algorithms are Proof of Work (PoW) (Chandrasekaran *et al.*, 2024) and Proof of Stake (PoS) (Nguyen *et al.*, 2019) (Gans, 2023).

PoW requires participants to solve complex mathematical problems to validate transactions and add new blocks to the blockchain, while PoS selects validators based on the amount of resources they commit, or "stake," to the network. However, both consensus mechanisms face challenges, such as energy consumption in PoW and throughput limitations in PoS, which can affect the speed and efficiency of block creation (Ismail *et al.*, 2019; Monrat *et al.*, 2020).

Scalability is another issue, as an increase in transactions can lead to network congestion and slower processing times.

Despite these challenges, public blockchains are widely adopted and support a variety of applications across different sectors. For example, they are increasingly used in voting systems (Inayatulloh *et al.*, 2023), ensuring transparency and immutability in elections. Public blockchains also facilitate public audits (Shu *et al.*, 2022), allowing anyone to verify the integrity of records and transactions. Other sectors, such as supply chain management (Madumidha *et al.*, 2019), identity verification (Chowdhary *et al.*, 2021), and decentralized finance (DeFi), also leverage the decentralized and open nature of public blockchains, making them a versatile and robust solution for a wide range of use cases.

2.2.2 Private Blockchain

In contrast to public blockchains, private blockchains are, by definition, permissioned networks where access to the network is restricted to authorized participants (Wicaksana & Wira, 2022). This model prioritizes access control over the data in the ledger and the actions over it; thus, over time, several different models were developed to address and construct a way to restrict data to certain groups (Langaliya & Gohil, 2023).

Role-Based Access Control (RBAC) (Ali *et al.*, 2022), Attribute-Based Access Control (ABAC), and Discretionary Access Control (DAC) are traditional models used in security architectures, defining access based on roles, attributes, or the discretion of the resource manager (Penelova, 2021). While newer models like Usage Control (UCON) and capability-based access control (CapBAC) have been introduced, they have yet to deliver significant improvements in network access control. These centralized models contrast with public network models but offer greater scalability by limiting the number of participants(Uddin *et al.*, 2019; Mansour & Kamal, 2024).

The consensus mechanisms continue to play a crucial role in ensuring the integrity and security of blockchain networks. However, because control of access to the information is in place before the consensus mechanism is activated, the focus of the model is to prioritize efficiency, scalability, and control and not limit participation based on resource-intensive consensus mechanisms such as PoW or PoS.

In similarity to the public model, there were several different consensus models based on different aspects, for example, Practical Byzantine Fault Tolerance (PBFT), Raft Consensus, and Proof of Authority (PoA) (Tran *et al.*, 2023). The adaptation of this type is high, as there are many applications that are based on a private model, such as Hyperledger (Hyperledger, 2024), Multichain, Quorum (Quorum, 2024), and Corda.

2.2.3 Hybrid and Consortium Blockchain

A hybrid blockchain combines the features of both public and private blockchains in one system (J *et al.*, 2023). This type of blockchain architecture seeks to leverage the strengths of both models while mitigating their respective weaknesses. There are private and public components made to allow certain access to participants while allowing public access to other components.

This approach allows for the ability to improve security and scalability while keeping private data for certain participants and allowing interchanging transactions between the private and public components. Hybrid blockchains can utilize a variety of consensus mechanisms, depending on the requirements of each component. For example, the public component may use a consensus algorithm like PoW or PoS, while the private component may opt for a more efficient consensus mechanism such as Practical PBFT or Raft.

A consortium blockchain is similar to the hybrid system where both combine both public and private blockchains; the difference is that in a consortium, there is a small trusted consortium to collaborate with each other. There are several different applications based on this type of blockchain (Dong *et al.*, 2024; Zhu *et al.*, 2019) with the aim of improving some aspects of the underlying technology.

2.3 Existing Applications

Several applications have been developed to solve different issues. This section discusses their attributes, including modes of access, development tools, and their advantages and disadvantages (Jafar *et al.*, 2021; Kuo *et al.*, 2019).

- Ethereum: This application is a successor to Bitcoin that looks towards building a better crypto currency that uses its predecessors achievements as a base to introduce smart contracts and decentralized applications (DApps) (Wood, 2024). Started in 2013, it introduced a Ethereum Virtual Machine (EVM) to be able to execute code mainly on Solinity but others as well. Its ability to deploy smart contracts to add a level of control enables it to be explored beyond simple peer-to-peer transactions. It builds on this by adding a cryptocurrency (ether) that enables participants to deploy contracts and execute transactions.

  Its development has enabled it to improve on its own; it currently uses POW, but it is moving towards POS in its Ethereum 2.0 upgrade (Kudzin *et al.*, 2022), aiming to improve scalability, security, and sustainability. Its difficulties range from scalability issues to high gas costs and network congestion when there's high demand (Ucbas *et al.*, 2023).

- Geth, short for Go Ethereum, is an Ethereum-based technology that is written in Golang. This application is a peer-to-peer network that focuses on sharing information directly between nodes rather than being managed by a central entity (Foundation, 2024a). The

architecture of the network is composed of Ethereum nodes that contain two clients: an executive and a consensus that handle different jobs in the node(Foundation, 2024b; Dhulavvagol *et al.*, 2020; Foundation, 2023).

Geth has problems with network security vulnerabilities and has been subject to various security risks, including network partitioning attacks such as Eclipse attacks (Henningsen *et al.*, 2019).

- Several projects focused on developing blockchain-based e-voting applications have highlighted significant challenges that need to be addressed for broader adoption, such as the Open Vote Network (OVN), the Decentralized Anonymous Transparent Electronic Voting System (DATE), the BSJC Proof of Completeness, the blockchain-based anti-quantum electronic voting protocol, and the blockchain-based electronic voting scheme (BES) (Jafar *et al.*, 2021).

  While these systems present innovative solutions, they each encounter distinct security challenges. Public e-voting systems are particularly vulnerable to attacks, whereas private systems often struggle with ensuring both accuracy and privacy for users.

- Stablecoins are a cryptocurrency developed for the purpose of allowing a stable exchange rate between currencies (Mita *et al.*, 2019; Heinonen, 2021). It provides a decentralized currency used with its fiat counter parts to provide its users with an alternative collateral option that would create a more diverse financial environment to engage with in transaction deals.

- IBM Food Trust (IBT) is a blockchain-based platform that manages the transactions between the stakeholders in the food industry, leveraging this technology to improve efficiency and transparency (Eletter *et al.*, 2022).

- Quorum: This application is an Ethereum-based solution designed to serve various sectors, ranging from finance to supply chain management, that employs smart contracts and a permission network to create a high-throughput and privacy-centered app (Kaur & Geeta, 2023; Harris, 2019). It is based on Geth technology and differs from the same by changing its Consensus algorithm, removing gas pricing, and changing its internal structure to improve the networks peer-to-peer communication (Consensys, 2024).

  Its main advantages are offset by the scalability issues evolving larger numbers, its maintenance requirements are higher, and its adoption is limited in the industries (Polge *et al.*, 2021).

- MedRec: This app is an blockchain solution in the healthcare industry with the objective to manage access to private and confidential patient information and records that are created and used by the providing party (Azaria *et al.*, 2016) (Shen *et al.*, 2019).

By leveraging this technology, it is able to follow current regulations in its country and provide a valuable service for patient privacy.

- Peer-to-peer energy ledger: This solution offers an electricity trading solution of interconnected peers that handle the power balance of a given area, thus insuring that power is distributed to regions that increase their consumption and deviate from areas that don't require as much energy (Ji *et al.*, 2022).

- Polygon is an blockchain app based on Ethereum that seeks to provide a second layer to its organization to improve the scalability issues found in Ethereum blockchain (Steinberg, 2023; Matic Network, 2024). It uses a modified POS consensus mechanism to be able to achieve the desired consensus among the network by using MATIC, an ERC-20 token.

  Notable projects developed on Polygon include the decentralized exchange QuickSwap (QuickSwap, 2024), the lending protocol Aave (Aave, 2024), and NFT platforms such as OpenSea (OpenSea, 2024), all of which contribute to its growing DeFi and NFT ecosystem.(Sujatha *et al.*, 2023).

- Solana: This is an blockchain technology that uses the consensus mechanism proof of history (POH) with the POS mechanism that aims to be a high-performance blockchain platform designed to support DApps and the transaction of cryptocurrencies. Solana's Proof of History system enables it to achieve high throughput and low transaction costs by providing a verifiable sequence of events to record the passage of time between transactions (Solana, 2024; Pierro & Tonelli, 2022).

  This public blockchain system is open and decentralized, meaning it is not a private blockchain and does not limit access to only authorized participants. Unlike private blockchains, which restrict access to pre-approved nodes, Solana allows any user or developer to participate in the network, validating transactions and interacting with decentralized applications without the need for centralized control or authorization.

- Edgence: This application is an Internet of Things (IoT) application that leverages blockchain technology to securely manage and facilitate transactions within IoT networks (Xu *et al.*, 2020). By integrating blockchain, Edgence ensures that all data exchanges between IoT devices are transparent, immutable, and resistant to tampering.

  The solution detailed in Edgence is primarily designed for IoT networks, which differ from the healthcare data-sharing context discussed in this work. Therefore, while valuable in its own right, it does not align with the specific goals and needs of the proposed implementation.

- MultiChain: This application is an open-source permissioned application that supports smart contracts. In its v2 version, it uses Round Robin validation and focuses on privacy

and security on transactions made (Greenspan, 2024). It is used in various industries for secure and private data exchange, such as finance, supply chain management, and healthcare. One of its key strengths is its flexibility in supporting multiple programming languages for smart contract development, including JavaScript, Python, and C++, making it accessible to a wide range of developers and allowing for integration into diverse technological ecosystems(Ismail *et al.*, 2023).

While it offers a range of programming languages to choose from, it suffers from some features not being available as others present (Kuo *et al.*, 2019).

- Hyperledger Fabric: This is the most popular currently developed application in the Hyperledger Foundation (Hyperledger, 2024), a permissioned platform that supports different languages (Go, Java, etc.) in the development of smart contracts. Its highly modular architecture allows organizations to customize and extend the platform to meet their specific requirements.

  In this architecture, it uses peers, channels for the peers to communicate, ordering services, and many other modules to allow the establishment of a blockchain that can guarantee stability, scalability, privacy, and data integrity. The platform utilizes databases such as LevelDB and CouchDB for state storage. The Linux Foundation oversees the Hyperledger project, ensuring its ongoing development and governance.

  However, the platform presents challenges related to network complexity, resource-intensive operations, and security concerns, which can complicate integration with other systems. These issues, as noted by (Brotsis *et al.*, 2020), create barriers to seamless interoperability and require advanced solutions to address the resource and security demands of the system.

- Hyperledger Sawtooth: This is another project under the Hyperledger Foundation, developed primarily for supply chain management and record keeping. It differs from Hyperledger Fabric in several ways, such as its consensus mechanism, which includes options like Proof of Elapsed Time (PoET) designed to work efficiently without needing heavy computational resources. Additionally, Hyperledger Sawtooth is able to achieve higher throughput with these consensus models compared to other models (Ampel *et al.*, 2019a).

  While Hyperledger Sawtooth offers significant benefits, it is important to note that the project has been archived, with active development passing to other entities, making it less popular and with significantly fewer resources (Hyperledger, 2024; Ampel *et al.*, 2019b).

- Corda: R3 Corda is a blockchain framework developed by the R3 Foundation (R3, 2024), developed in the financial sector for creating and securing transactions made by two or

more entities where they agree with the transaction. It uses Kotline for its smart contract software; Kubernetes has its building blocks to construct the database for storage and the Kafta (Apache Kafka, 2024) services for communication between modules; then it installs the rest of the application with other modules that will be used for cryptography, API rest, configuration settings, and membership modules. It additionally uses a unique mechanism to achieve consensus between the virtual nodes in a network (Gajić *et al.*, 2022). It is used in the financial and commerce industry (Pradhan *et al.*, 2023).

While it offers a wider set of benefits, it is resource-intensive, and it lacks adoption in the wider field.

Table 2.1 provides a comparative overview of several prominent blockchain platforms: Ethereum, MultiChain, R3 Corda, and Hyperledger Fabric. These platforms represent a spectrum of blockchain solutions, from public to permissioned networks, each tailored to different needs in terms of access control, scalability, development tools, consensus mechanisms, and use cases. They are the more promising among the already presented apps based on their benefits compared to the others and the availability of documentation. Each of the candidates was evaluated through testing in terms of their normal use, assessing functionality or/and documentation.

Table 2.1: Aspects of the different applications based on Blockchain technology

	Ethereum	**MultiChain**	**R3 Corda**	**Hyperledger Fabric**
Type of access	Public	Permissioned	Permissioned	Permissioned
Development Tools	Solidity	Several	JDK	Several
Scalability	Low	Low	High	High
Consensus Mechanism	POW/POS	Several	RAFT	PBFT, Kafka-based ordering service, Raft, Solo
Pros	Smart Contracts, Large Developer Community, EVM	Customization, Simple Deployment, Privacy, Confidentiality	Privacy, Confidentiality, Scalability, Interoperable, Smart Contracts, Legal Compliance (Finance), Identity Management, Governance, Membership	Modularity, Extensible, Smart Contracts (Chaincode), Performance, Scalability, Private Data, Channels, Identity Management, Governance, Membership
Cons	Scalability Challenges, Gas Fees, Upgrade Difficulties	Centralization Concerns, Scalability Challenges, Limited Smart Contract Support	Learning Curve, Resource Intensive, Adoption Lacks	Complexity, Resource Intensive, Development Learning Curve, Community and Ecosystem, Integration Challenges

2.4 Summary

All the projects previously discussed have been developed to achieve specific advantages in certain scenarios. However, this work focuses on finding a framework to build an application that can fulfill the objectives established in the introduction: a blockchain capable of creating and storing transactions, ensuring privacy, guaranteeing data integrity, allowing access only to authorized participants, and providing a high level of security.

In light of these requirements and after experimentation with the applications, Ethereum can be excluded as a suitable option. Ethereum's public access model, while robust and widely adopted for decentralized applications, is fundamentally incompatible with the need for restricted access and privacy. The public nature of Ethereum means that any transaction recorded on its blockchain is visible to all participants, which is a significant limitation for applications that require confidentiality and access control. Although there are ongoing efforts to improve privacy on Ethereum, such as Layer 2 solutions and Ethereum 2.0, these still do not fully address the core issue of restricting access to authorized participants. Therefore, Ethereum's open and decentralized infrastructure, while advantageous in many contexts, does not align with the specific needs of this project.

On the other hand, Hyperledger Fabric and Corda emerge as the two most viable options that meet the project's criteria. In contrast, MultiChain is not considered a suitable solution due to the lack of many features that both Hyperledger Fabric and Corda offer. Both platforms are designed for permissioned networks, allowing control over who can participate and view transactions. Additionally, both support peer-to-peer communications, ensuring that transactions can be conducted privately between parties without being broadcast across the entire network.

Given these considerations, the choice between Hyperledger Fabric and Corda ultimately depends on the specific requirements of the application being developed. The proposed project necessitates an application that can be integrated with different systems with flexible development for future expansion; the more development-friendly Corda achieves these better than Hyperledger Fabric.

Chapter 3

Proposed Project

As stated in the previous sections, Corda R3 is a permissioned blockchain project that aims to create a peer-to-peer network that communicates with each other to retrieve and create translations with the chosen participants. As a whole, it values privacy, data integrity, security, and interoperability between different external systems and builds on an API REST Port to be able to interact and change the application internally (R3, 2024, 2023b).

The functional, non-functional and software/hardware requirements for this system were defined to cover both essential features (functional) and the qualities that the system must maintain under various conditions (non-functional) approved by the company ByMe.

3.1 Functional, Non-functional and Software/Hardware Requirements

The functionalities that the system must be able to perform are detailed below:

- **URF001**: Register a user in a virtual Node.
- **URF002**: Authorize user.
- **URF003**: List all available document for the user.
- **URF004**: (Medical Authority) Be able to create new consent forms.
- **URF005**: (Medical Authority) Be able to update consent forms.
- **URF006**: (Medical Authority and Patient) Be able to view the history of transactions on a consent forms.
- **URF007**: (Medical Authority and Patient) Be able to sign consent forms.
- **URF008**: (Medical Authority and Patient) Be able to view the profile of the user.

- **URF009**: (Medical Authority) Be able to store files in the consent form.
- **URF010**: (Medical Authority) Be able to add participant in the creation of the document.
- **URNF011**: Be able to decrypt files.

The non-functional aspects that the system must meet are outlined below:

- **URNF001**: The files must be encrypted.
- **URNF002**: Operate with 50 users.
- **URNF003**: Operate asynchronous.
- **URNF004**: Operate continuously, day and night.

The software and hardware requirements for the system implementation are established below:

- **URS001**: Utilize Kotlin or Java, the primary programming languages for Corda application development.
- **URS002**: Implement Corda's Distributed Ledger Technology (DLT) using Corda's specific architecture and protocols for transaction validation and smart contract execution.
- **URS003**: Incorporate Corda's built-in security features for cryptographic authentication and authorization of network participants.
- **URS004**: Develop a web interface using Angular, allowing users to interact with the Corda cluster through a user-friendly graphical interface.
- **URS005**: Utilize Corda's API interface to enable communication between the web interface and the Corda nodes, facilitating actions such as querying node states, initiating transactions, and monitoring network activity.
- **URS006**: Utilize a reliable network infrastructure with low latency and high bandwidth to facilitate communication between Corda nodes within the cluster and ensure timely transaction propagation.

3.2 Application Architecture

This project builds on DLT (Anthony, 2023; Yap *et al.*, 2023), written in Kotlin; it used the standard Java development tools and packaged using Corda tooling; thus, it builds.JAR files with the desired code and bundles it to create a file in the format of a package that can be used in CordApp (Corda + App). Once packaged and deployed onto a Corda application, it allows ledger and application-based changes to occur (R3, 2024).

This system is composed of two layers:

- Orchestration layer: This layer handles peer-to-peer communication between participants in the form of flows. These flows are java/kotlin logic that is written in.java/kt files that contain the business logic of the application that can be accessed by the REST API (Fielding & Taylor, 2000) exposed. It allows developers to create code that allows participants to communicate with each other following a set of instructions.

- Consensus layer: This layer handles the creation of agreements between not-trusted participants about truths made in the applications, and, after being modified, it becomes a universal permanent truth that happened and everyone agrees. It does this by creating a validity and smart contract system that can rule transactions as able or unable to be agreed upon. By making the contract the arbiter of validity, it can enforce rules upon the transaction by attribute.

The CordApp Distributed ledgers, unlike public DLT´s where access is open to all, are restricted by the entities (known as network operators) operating on the network. The rules of engagement are decided by the network; as such, after it passes the verification and is permitted to join, the entity knows that the other entities also had their identities challenged by the same metric.

In the architecture of the network, it divides entities by members, privileged members that have discretion and special powers inside and non-members that do not have access to the network, and then a membership group manager (MGM) that manages membership inside the network, guaranteeing that the rules built are followed. One of them is the unique network ID and the X500 name that the participant chooses; the MGM does not allow duplicates and blocks any membership registration request that would result in that situation (R3, 2023a).

As the peers are validated and approved, they are allowed to communicate with each other, one-to-one, with their peers, but not allowed to communicate in broadcast, one to many, to not leak information to not intended parties. In addition to the communication, there is a global state that is composed of all states inside the application; this is not globally visible to all participants; only partial states are available to each participant because they are a direct participant in a mutation of the global state or were added as a participant. Each participant, with the data available to them, copies and stores it locally.

Corda's distributed architecture enables the system to scale by adding more nodes and workers as needed, while Kafka ensures that all components can exchange messages in real time with minimal latency. The REST API serves as the interface for external systems and applications, providing a standardized way to interact with the network and trigger actions such as creating or updating consent forms, retrieving document histories, and interacting with the virtual nodes in the Corda network.

The configuration of the network operator is built on a Corda Package Installer (CPI) file that is installed and validated by its subcomponents. The CPI has a Corda Package Bundle (CPB) file and a Corda Package (CPK) that has code-signing certificates and is validated by

a certificate authority (CA). The MGM root information is also established by this process in a static GroupPolicy.json file of the CPI, and then the CordApp updates the participant's virtual presence, Virtual Nodes, for the newer version of the CPI. In the configuration files, it is necessary to define trusted roots so that they can securely trust, for example, the messages sent by participants and the type of root used, the membership information.

After the network is configured and the CPI is installed, the virtual nodes can create flows based on the code added in the CPK file with the smart contracts in an HTTPS request (Zinedine *et al.*, 2024). This allows a way to create transactions and access information by executing code inside of the Corda app.

To interact with this workflow, an HTTPS request must be made, specifying the class name, request ID, and including the request body in the payload. Additionally, the authorization details (username and password) should be provided in the request headers. In a POST request, using the holdingHashID of the virtual node, the process will initiate the request. As exemplified in 3.1,

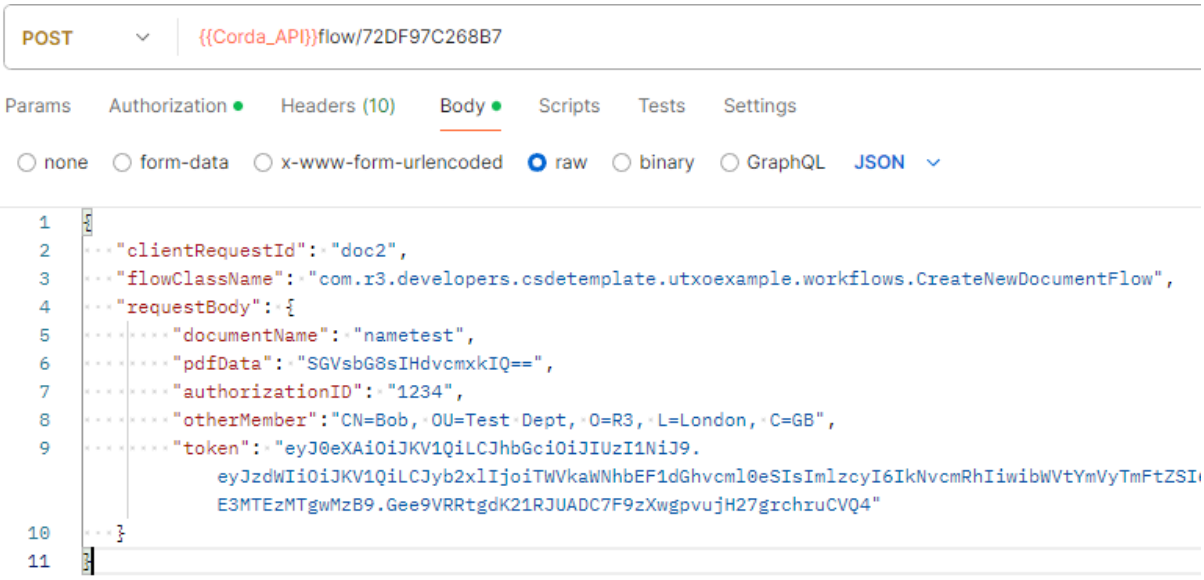


Figure 3.1: Example of a HTTPS request of the creation of Documents in Postman to the Corda application

The Request 3.1 is asynchronous, and its status can be checked by the key clientRequestId in the HTTPS request body and making a GET request. The clientRequestId acts as the unique identifier for tracking the request, which is set when the request is initiated. The key flowClassName refers to the class that handles the request, defined by the path of the file in which the class is written. The requestBody contains the necessary arguments or parameters used in the workflow, which are required to process the request.

Figure 3.2: Example of a Get HTTPS Request to get the status of the request in Postman

The Response 3.2 of the process by providing the current status of the asynchronous request initiated earlier.

The code processes a workflow, performing various operations that allow it to generate new transactions, which in turn generate new ledgers or add more transactions to already existing ledgers. In this project, when a document is created, it creates a ledger with an initial state based on the arguments given by the participant who initially requested its creation. This state could be multiple or single but is restricted by a contract that it belongs to; these restrictions can be universal, by the command issued and by the attributes of the input or the output of a given transaction.

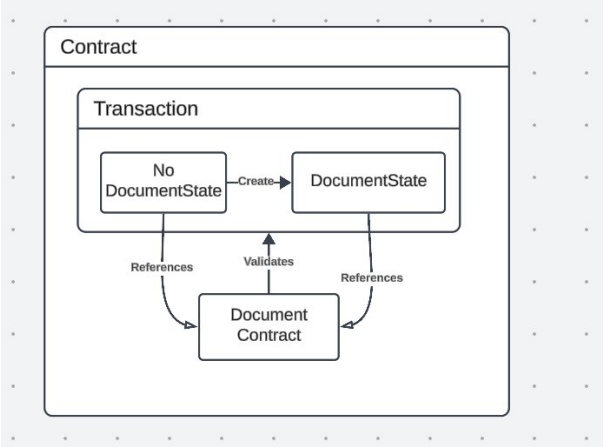


Figure 3.3: Structure of the Document Contract validating the Document State

In Figure 3.3, the states are depicted, with the input on the left side and the output on the right. The type of transaction being made is "create," which encapsulates this creation process. Additionally, the contract that validates the transaction and owns the document state is also shown.

3.3 Consent Forms CordApp

The application developed starts with the most basic layer in the development of a CordApp, the tools that were used. The table 3.1 that describes what tools were used for this implementation:

Table 3.1: Tools Description

Tool	**Description**
Windows 11	Operating system. Version: 11.
Docker Desktop	Docker Engine. Version: 26.1.1
Docker Compose	Tool for defining and running multi-container Docker applications. Version: v2.27.0-desktop.2.
Docker Credential Helper	Tool for securing access and storage of docker applications. Version: v0.8.1.
Kubernetes (Docker)	Kubernation integration of Docker Desktop. Version: v1.29.2
Git	Version control system for Github. Version: 2.45.0.windows.1
kubectl	command-line tool for interacting with Kubernetes clusters. Version Client: v1.30.0, Kustomize: v5.0.4
Helm	Package manager for Kubernetes. Version: v3.14.4
Minikube	Tool for running Kubernetes clusters locally. Version: v1.33.0
IntelliJ IDEA	Code Editor for Java or Kotlin. Version: 2024.1.1
JDK 17	Java development Kit. Version: Azul Zulu JDK 17
Visual Studio Community	Code Editor. Version: 2022
.NET Core and SDK	Development framework. Version: 8.0
Angular	Framework for client-side applications. Version: 15
Visual Studio Code	Code Editor.
Corda	Blockchain application. Version: v5.1
Chocolatey	Package Manager.
Corda CLI	Corda command line interface version: 5.1.0.0
PostgreSQL	Database.

As stated in the figure 3.4, this application has three parts. The Frontend, Backend/Rest API, and the CordApp itself; the Frontend is built in Angular 15 (Google, 2024a), and the Backend/Rest API is built in. NetCore in C sharp (Microsoft, 2024), because the company ByMe (ByMe, 2024) uses these tools and code programs in their products and to keep in line with their reality. Additionally, there is also a Kotlin project that does decryption of client data with his private key.

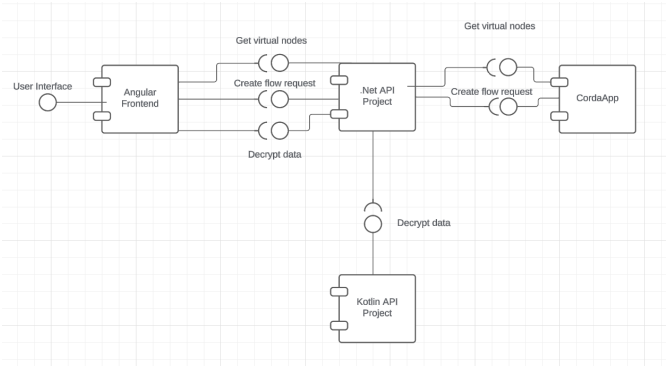


Figure 3.4: Component Diagram of the Structure of Application

The diagram 3.4 presents the projects as components that interact with each other through the interface symbols: Provided Interface and Required Interface, where one component provides a service or functionality that is required by another component.

In this architecture, the Angular Frontend relies on the .NET Backend primarily to handle the processing of requests rather than executing the core functionality itself. The .NET backend acts as an intermediary, routing requests for services such as the creation of the workflow, decryption of data, and retrieval of the virtual node to other services, such as Kotlin API and the Corda Application.

A Contract, in the Corda application, is a class created in a kotlin file, that implements an interface given in the Corda library, that defines the commands that the contract allows and restricts each type of command to a set of built-in or universal rules that are set outside the given command restriction. Its restrictions aim to limit what changes are made to the ledger containing previous transactions, ensuring a level of data integrity is maintained. Each transaction is designed to alter one or more states by validating them.

```
class DocumentContract : Contract {  40115 *

    class Create: Command  40115

    class Update: Command  40115

    class Delete: Command  40115
```

Figure 3.5: Class of the Document Contract with the declared commands.

In the figure 3.5, there is a class that is named as DocumentContract, it implements the interface Contract, and then it defines 3 Command classes: Create, Update and Delete, which represent the 3 main commands that are going to be restricted.

```
override fun verify(transaction: UtxoLedgerTransaction) {  40115*

    val command = transaction.commands.singleOrNull() ?: throw CordaRuntimeException(REQUIRE_SINGLE_COMMAND)

    OUTPUT_STATE_SHOULD_ONLY_HAVE_TWO_PARTICIPANTS using {
        val output = transaction.outputContractStates.first() as DocumentState
        output.participants.size== 2
    }

    TRANSACTION_SHOULD_BE_SIGNED_BY_ALL_PARTICIPANTS using {
        val output = transaction.outputContractStates.first() as DocumentState
        transaction.signatories.containsAll(output.participants)
    }
```

Figure 3.6: Universal rules of the Document Contract

The lines of code in 3.6 define the function `verify`, which is overwritten by the function that is defined in this contract. This verifies the transaction that is being processed by the consensus mechanism and validates the transaction, its type, and attributes of the state. There is a validation in the beginning to restrict the command to only one command being made per transaction, followed by two universal conditions: restricting the number of participants and comparing the participants. Afterwards, it provides the restriction by the type and then by attributes given by the participant that created the transaction.

```
when(command) {
    is Create -> {
        CREATE_COMMAND_SHOULD_HAVE_NO_INPUT_STATES using (transaction.inputContractStates.isEmpty())
        CREATE_COMMAND_SHOULD_HAVE_ONLY_ONE_OUTPUT_STATE using (transaction.outputContractStates.size == 1)
    }
    is Update -> {
        UPDATE_COMMAND_SHOULD_HAVE_ONLY_ONE_INPUT_STATE using (transaction.inputContractStates.size == 1)
        UPDATE_COMMAND_SHOULD_HAVE_ONLY_ONE_OUTPUT_STATE using (transaction.outputContractStates.size == 1)

        val input = transaction.inputContractStates.single() as DocumentState
        val output = transaction.outputContractStates.single() as DocumentState
        UPDATE_COMMAND_ID_SHOULD_NOT_CHANGE using (input.id == output.id)
        UPDATE_COMMAND_PARTICIPANTS_SHOULD_NOT_CHANGE using (
                input.participants.toSet().intersect(output.participants.toSet()).size == 2)
    }
    is Delete -> {
        DELETE_COMMAND_SHOULD_HAVE_ONLY_ONE_INPUT_STATE using (transaction.inputContractStates.size == 1)
        DELETE_COMMAND_SHOULD_HAVE_ONLY_ONE_OUTPUT_STATE using (transaction.outputContractStates.size == 1)

        val input = transaction.inputContractStates.single() as DocumentState
        val output = transaction.outputContractStates.single() as DocumentState
        DELETE_COMMAND_ID_SHOULD_NOT_CHANGE using (input.id == output.id)
        DELETE_COMMAND_PDF_SHOULD_NOT_CHANGE using (input.pdfData.contentEquals(output.pdfData))
        DELETE_COMMAND_PARTICIPANTS_SHOULD_NOT_CHANGE using (
                input.participants.toSet().intersect(output.participants.toSet()).size == 2)
        DELETE_COMMAND_PARTICIPANTS_SHOULD_BE_RECORDSTATE_DELETED_BEFORE_CHANGE using (input.recordState == RecordState.Active)
        DELETE_COMMAND_PARTICIPANTS_SHOULD_BE_RECORDSTATE_DELETED_AFTER_CHANGE using (output.recordState == RecordState.Deleted)
    }
    else -> {
        throw CordaRuntimeException(UNKNOWN_COMMAND)
    }
}
```

Figure 3.7: Commands rules of the Document Contract

Beyond these general validations in the figure 3.7, more specific rules come into play based on the type of transaction. These rules govern the precise actions that can be taken, such as:

1. **Transaction Type-Specific Rules**:
    - **Create Transactions**:
        - **No Input States**: The create command should have no input states, ensuring that it is introducing a new document or entity into the ledger.
        - **Single Output State**: The create command should have only one output state, maintaining simplicity and clarity in the creation process.
    - **Update Transactions**:
        - **Single Input and Output State**: The update command should have only one input state and one output state to ensure a clear and direct modification process.
        - **ID Consistency**: The ID of the document should not change during the update, maintaining the continuity of the document's identity.
        - **Participants Consistency**: The participants involved in the document should remain unchanged, ensuring the same parties are involved post-update.
    - **Delete Transactions**:
        - **Single Input and Output State**: The delete command should have only one input state and one output state, similar to the update command, for a straightforward deletion process.
        - **ID Consistency**: The ID of the document should not change, ensuring the deleted document can be traced back to its original state.
        - **PDF Data Consistency**: The PDF data of the document should remain unchanged, confirming the content of the document being deleted.
        - **Participants Consistency**: The participants involved should remain the same before and after deletion, verifying that the same entities approve the deletion.
        - **Record State Validation**: The record state should change from active to deleted, reflecting the accurate status of the document in the ledger.
2. **Ledger Update Restrictions**:
    - **Integrity Checks**: Every update to the ledger must pass integrity checks to prevent tampering and ensure data consistency. This includes verifying the hash values, ensuring that the ledger remains immutable, and that any changes are appropriately logged and auditable.
    - **Consistency Checks**: Ensuring that updates do not conflict with existing entries in the ledger. This involves cross-referencing new entries with existing ones to detect and resolve any discrepancies.

- **Approval Requirements**: Some updates may require approval from a certain number of participants or specific roles within the network. This ensures that critical updates have been reviewed and sanctioned by trusted entities.

3. **List of Participants Modification**:

   - **Addition of Participants**: Adding new participants is restricted to the CREATION, the other types of commands adds unnecessary access to documents.

On the other side of the contract section, the state is defined by a data class, that implements the interface ContractState and a notation to define what contract it belongs to and restrict it. This interface allows freedom to store all different types of data, but it has to have a function to getparticipants to have a list of members by their public keys in the application.

This classes are set-up to represent different data points that are needed to be applied, it has different proprieties and, for each of those, different types that reflect the data type stored.

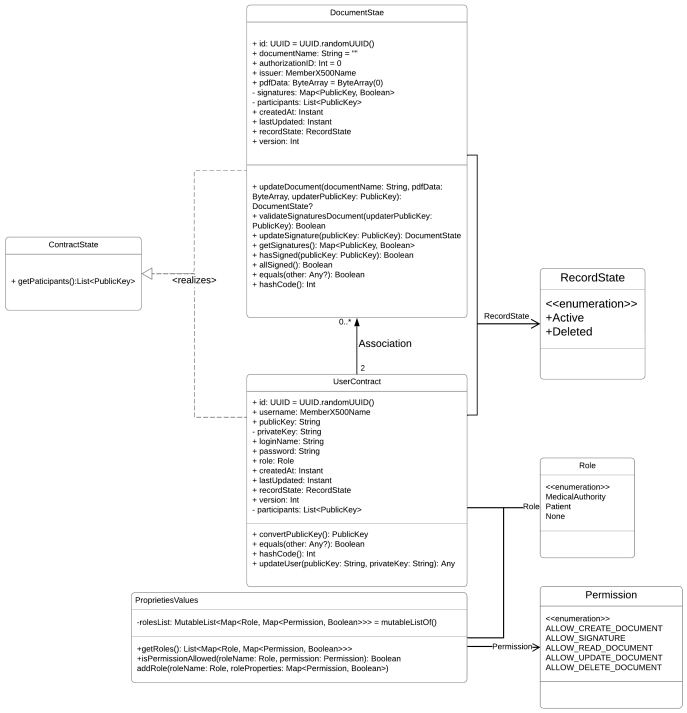


Figure 3.8: UML Class Diagram Showing the Structure and Relationships of the classes present in the Application

The class diagram in the figure 3.8 illustrates the structure and relationships of classes involved in representing different data points and contracts within the application. Each class is designed to encapsulate specific properties and data types necessary for the application's

functionality. The States that are constricted with their contracts, the enums that are present at least in one of the states and finally their attributes. Each state has a recordState, to determinate if its active or deleted, and the user has a role attribute to present the type of user that the user is.

```
@BelongsToContract(DocumentContract::class)  40115
@CordaSerializable
data class DocumentState (
    val id: UUID = UUID.randomUUID(),
    val documentName: String = "",
    val authorizationID: Int = 0,
    val issuer: MemberX500Name,
    val pdfData: ByteArray = ByteArray( size: 0),
    private val signatures: Map<PublicKey,Boolean>,
    private val participants: List<PublicKey>,
    val createdAt: Instant,
    val lastUpdated: Instant,
    val recordState: RecordState,
    var version: Int
) : ContractState {

    override fun getParticipants(): List<PublicKey> {
        return participants
    }
```

Figure 3.9: Data Class for Document State with its corresponding attributes and its Document Contract

The figure 3.9 contains information about the ID of the state, the information of the document in itself, the issuer, the state of the document (RecordState), the participants, who signed, the creation and last update date, and the current version. It highlights important information to keep track of during the creation of a new transaction.

In parallel to the document ledger, there is a UserState and a UserContract that have similar objectives and structures, where the UserState belongs to the UserContract and restricts transactions, and the new state is created by adding conditions like the contract for the documents.

```
@BelongsToContract(UserContract::class)  40115
@CordaSerializable
data class UserState (
    val id: UUID = UUID.randomUUID(),
    val username: MemberX500Name,
    val publicKey: String,
    val privateKey: String,
    val loginName: String,
    val password: String,
    val role: Role,
    val createdAt: Instant,
    val lastUpdated: Instant,
    val recordState: RecordState,
    var version: Int,
    private val participants: List<PublicKey>,
    ) : ContractState {

    override fun getParticipants(): List<PublicKey> {
        return participants
    }
```

Figure 3.10: Data Class for User State with its corresponding attributes and its User Contract

The user state in the figure 3.10 is to allow a type of registration inside the workflows, outside the membership, and permit registration and login with more control over developing

the application than at the network level, while it would be possible and useful to use it in other scenarios.

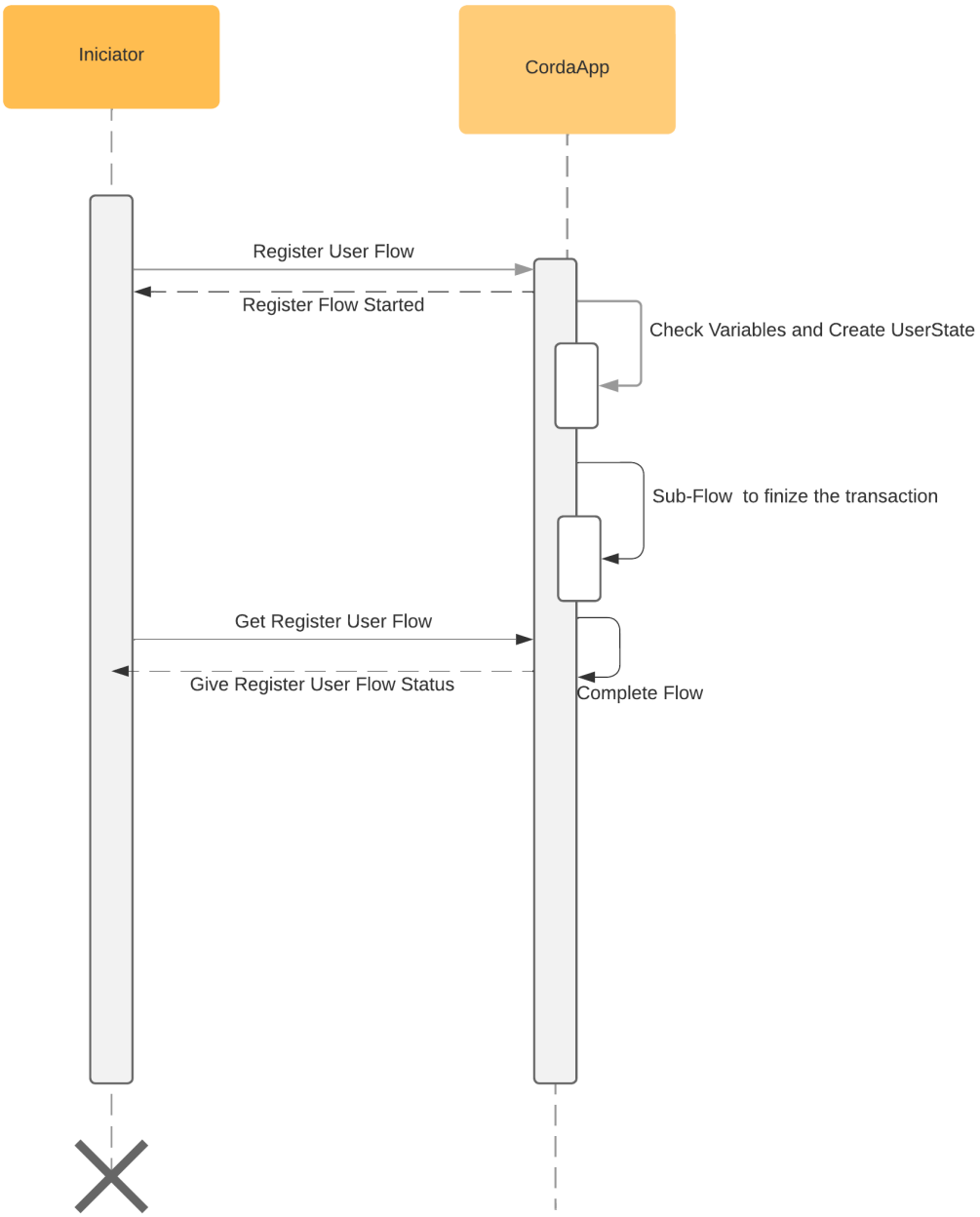


Figure 3.11: Sequence diagram of the Registration of a User in a Vnode

In the example 3.11, it demonstrates a basic flow between an initiator (participant) and the CordApp. It starts with the initial HTTPS request with the basic authorization and the body request with the needed arguments.

It returns the return message with the message that the "START_REQUEST" state. The request is validated by the network, which checks if the virtual node shortholdindhashId exists,

the basic authorization credentials, and then the workflow code. In this case, it validates the arguments given, validates if the virtual node mentioned already has a user state, then creates a transaction in which the only one that is a participant is the issuer, and concludes the flow. If, by this point, the user requests the status, it will display the results of the workflow.

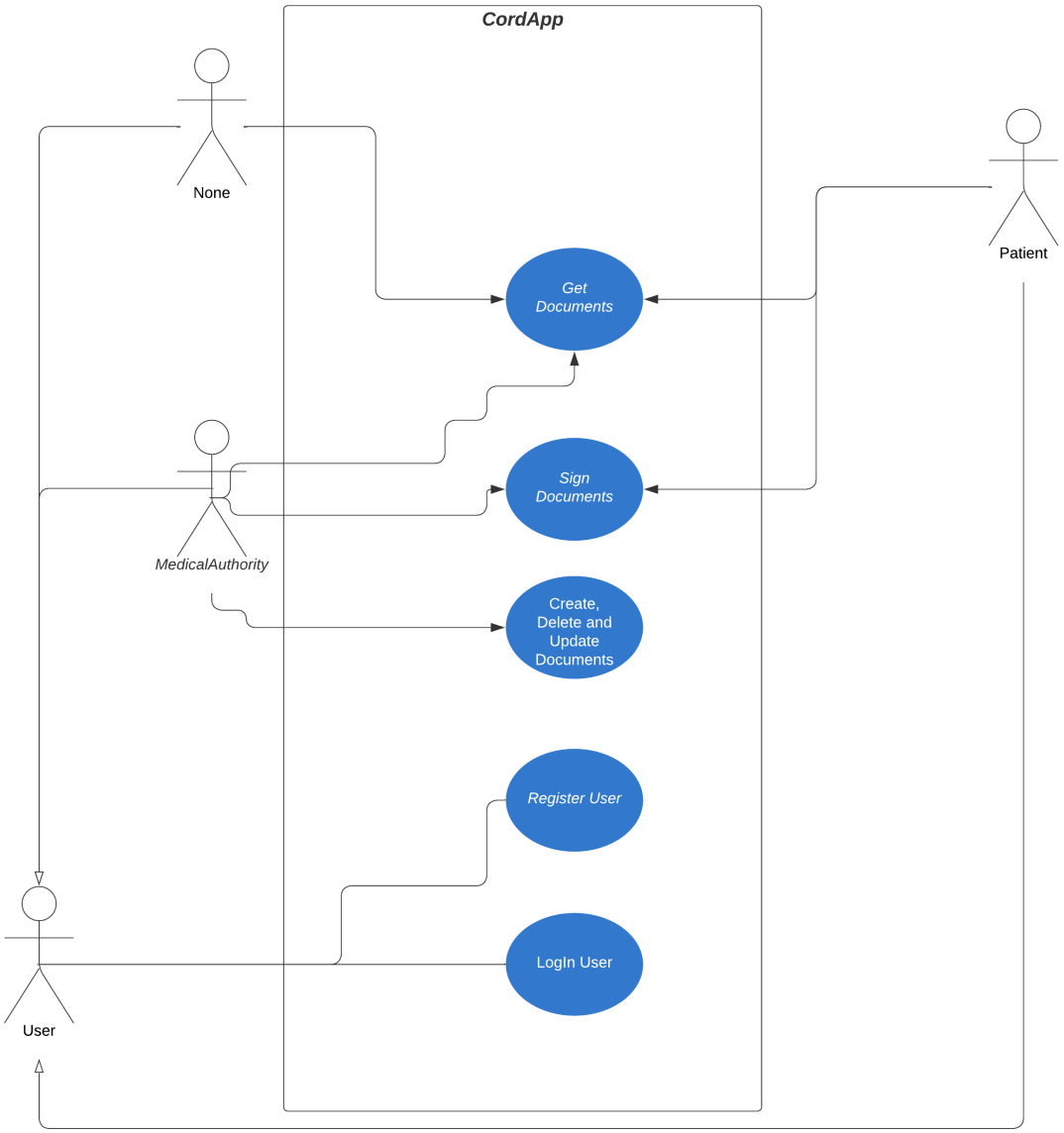


Figure 3.12: User Use Case Diagram Demonstrating Interactions and Functionalities

The registration also involves the determination of a role; the CordaApp has 3 roles: None, MedicalAuthority, and Patient.

None has any permission to do anything except view the history of the document, like the other roles. The medical authority is allowed to create, update, delete, sign, and view the history of documents, while the patient is allowed to sign and view documents. After all members sign the documents, it is prohibited to update or delete the document in any way.

In the diagram 3.12, it shows the allowed actions of each actor by interacting with the Corda application. The workflows related to the document state involve more than one participant: the issuer, the medical authority, and the other user who will sign or not. The roles are given by the participant role, and that attribute is then used for permissions inside the given flow.

```
class proprietiesValues {  40115 *
    companion object {  40115 *
        private val rolesList: MutableList<Map<Role, Map<Permission, Boolean>>> = mutableListOf()

        init {  40115 *
            addRole(Role.MedicalAuthority, mapOf(Permission.ALLOW_CREATE_DOCUMENT to true, Permission.ALLOW_SIGNATURE to true,
                Permission.ALLOW_READ_DOCUMENT to true, Permission.ALLOW_UPDATE_DOCUMENT to true,
                Permission.ALLOW_DELETE_DOCUMENT to true))
            addRole(Role.Patient, mapOf(Permission.ALLOW_CREATE_DOCUMENT to false, Permission.ALLOW_SIGNATURE to true,
                Permission.ALLOW_READ_DOCUMENT to true, Permission.ALLOW_UPDATE_DOCUMENT to false,
                Permission.ALLOW_DELETE_DOCUMENT to false))
            addRole(Role.None, mapOf(Permission.ALLOW_CREATE_DOCUMENT to false, Permission.ALLOW_SIGNATURE to false,
                Permission.ALLOW_READ_DOCUMENT to true, Permission.ALLOW_UPDATE_DOCUMENT to false,
                Permission.ALLOW_DELETE_DOCUMENT to false))
        }

        private fun addRole(roleName: Role, roleProperties: Map<Permission, Boolean>) {  40115
            rolesList.add(mapOf(roleName to roleProperties))
        }

        fun getRoles(): List<Map<Role, Map<Permission, Boolean>>> {  40115
            return rolesList
        }

        fun isPermissionAllowed(roleName: Role, permission: Permission): Boolean {  40115
            val role = rolesList.find { it.containsKey(roleName) }
            return role?.get(roleName)?.get(permission) ?: false
        }
    }
}
```

Figure 3.13: Class of the proprietyValues

In this class 3.13, it receives the role from the argument from the function and checks it. For the type of action and the type of role, it will give a boolean if it can and cannot do said action. The type of permission used is a list with two things inside: the role, and then a map with two things inside: the key (an enum with the type of permission) and the value with the boolean. This way, it offers a list of roles, where each has permissions and values presented in the table 3.2.

Table 3.2: Permissions for Different Roles

Permission	**Medical Authority**	**Patient**	**None**
CREATE	true	false	false
SIGN	true	true	false
READ	true	true	true
UPDATE	true	false	false
DELETE	true	false	false

The workflows with the registration and logins contain arguments to register and login

members with the virtual nodes; the register returns their keys used for decrypting data that is particularly sensible, and the login returns a token used to interact with workflows that interact with the DocumentState.

The interactions with the DocumentState have 4 types of interactions with the state: create, update, delete, and read.

Create: In the creation, the workflow validates the arguments, validates the token, validates if the role and the permission are true, allowing creation of the document, then creates a new transaction that creates a ledger with an initial DocumentState. While creating the transaction, it creates sub-flows where the participants in the list of participants validate the transaction, and, if it exists, the notary, after they validate and sign it, concludes the transaction by returning the transactionID.

UPDATE: In the UPDATE, the workflow validates the arguments, validates the token, validates if the role and the permission are true, allowing alteration of the document, then creates a new transaction that copies the previous documentState and changes the attributes, thus creating a new updated DocumentState. While creating the transaction, it creates sub-flows where the participants in the list of participants validate the transaction, and, if it exists, the notary, after they validate and sign it, concludes the transaction by returning the transactionID.

DELETE: In the DELETE, it uses the UPDATE method but updates the RecordState to DELETED while validating all other steps. This will be useful for the validation of getting the RecordState from the Virtual Node.

READ: In the READ, the workflow validates the arguments, validates the token, validates if the role and the permission are true for getting the records of the document, then returns the request with a JSON (Bourhis *et al.*, 2020) with the documents.

Table 3.3: Interactions with DocumentState

Description	**CREATE**	**READ**	**UPDATE**	**DELETE**
Validates arguments	✓	✓	✓	✓
Validates token	✓	✓	✓	✓
Checks role & permission	✓	✓	✓	✓
Creates transaction	✓		✓	✓(sets to DELETED)
Creates sub-flows	✓		✓	✓
Notary validates transaction	✓		✓	✓
Concludes transaction	✓		✓	✓
Returns request with JSON		✓		

The project is built using this 3.3 type of control over data access and privacy to secure the data and validate changes to the ledgers.

The workflows are composed by a class that implements the ClientStartableFlow interface, that contains a call function, and that, after the verification with the network, starts the code here with an object containing the body of the request. In the class, there are also containers that

provide features such as member lookup, logging, marshaling, and ledger services that allow marshaling of the request body and get states available to the VNodes. It also has a notary lookup, useful to use the notary to act on transactions, and a flow engine that allows the start of sub-flows in the middle of a flow.

3.4 Installing, deploying, and configurations of CordApp

As referenced in the previous chapters, this application uses Kunernetes to establish the application in a container in Docker (Docker, 2024).

In the deployment phase, it involves creating a cluster with pods that are installed with a pre-installation phase with Helm, and then, using a values.yaml file with a template of what set values the Corda application will have, installing Corda on the cluster. Afterward, expose port 8888 to the system to be able to interact with the Swagger web page made to help make requests (Arora, 2022).

1. **Windows Terminal:**

   1.1 `docker context use default`
   Sets Docker context to default.

   1.2 `minikube start -memory 8000 -cpus 6 -driver=docker`
   Starts Minikube instance using Docker driver with 8GB RAM and 6 CPUs.

   1.3 `minikube kubectl - get pods -A`
   Retrieves pods in Minikube instance using `kubectl`.

   1.4 `kubectl create namespace corda-dev`
   Creates 'corda-dev' namespace in `kubectl`.

   1.5 `kubectl config set-context -current -namespace=corda-dev`
   Sets 'corda-dev' namespace as current in `kubectl` context.

   1.6 `helm install prereqs -n corda-dev`
   `oci://registry-1.docker.io/corda/corda-dev-prereqs`
   `-timeout 10m -wait`
   Installs Corda development prerequisites from Docker registry with a timeout of 10 minutes.

2. **Directory:**

   2.1 `mkdir test`
   Create directory.

   2.2 `cd test`
   Move to the directory.

2.3 `echo. > values.yaml`
Create an empty `values.yaml` file that will contain the Corda configuration in the `test` directory.

2.4 `helm install -n corda-dev corda oci://registry-1.docker.io/corda/corda -version 5.1.0 -f values.yaml`
Installs Corda version 5.1.0 in the 'corda-dev' namespace using the specified configuration from `values.yaml`.

After installing the prerequisites in step 1.6, the terminal will display a template of the 'values.yaml' file. This file can be modified to adjust specific pod resources. For this project, no modifications were made to this file.

```
bootstrap:
  kafka:
    replicas: 1
db:
  cluster:
    host: "prereqs-postgres"
    username:
      value: "corda"
    password:
      valueFrom:
        secretKeyRef:
          name: "prereqs-postgres"
          key: "corda-password"
kafka:
  bootstrapServers: "prereqs-kafka:9092"
  sasl:
    enabled: true
    mechanism: "PLAIN"
    username:
      value: "admin"
    password:
      valueFrom:
        secretKeyRef:
          name: "prereqs-kafka"
          key: "admin-password"
  tls:
    enabled: true
```

```
    truststore:
      valueFrom:
        secretKeyRef:
          name: "prereqs-kafka"
          key: "ca.crt"
```

The last lines show the deployment status and instructions for using Corda 3.14:

```
Pulled: registry-1.docker.io/corda/corda:5.1.0
Digest: sha256:adb96642f43bef1bc93016ff0f0f626f1d87fa0d67e710cbed762eb7fe59c1d9
NAME: corda
LAST DEPLOYED: Thu May  2 16:48:33 2024
NAMESPACE: corda-dev
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
1. Extract the username and password for the REST API admin created during bootstrap:
kubectl get secret corda-rest-api-admin --namespace corda-dev -o go-template='{{ .data.username |
base64decode }}'
kubectl get secret corda-rest-api-admin --namespace corda-dev -o go-template='{{ .data.password |
base64decode }}'
2. Expose the API endpoint on localhost by running this command:
kubectl port-forward --namespace corda-dev deployment/corda-rest-worker 8888
3. The API endpoint definition can then be accessed via: https://localhost:8888/api/v1/swagger

For more information, see the Corda 5 documentation at docs.r3.com.
```

Figure 3.14: Deployment status and instructions for using Corda

The message 3.14 gives a brief overview of the installation of Corda 5.1.0 with the cluster; the username and password can be defined in the values.yaml file, the template uses the username admin and a random password that can be retrieved by extracting them in the terminal. To check that the pods are completed or running correctly, minikube allows to get the correct status of pods by the *'minikube kubectl – get pods -A'* command.

After deploying the Corda Cluster, the projects can interact with the API by exposing a specific port. To register VNodes with a CPI file during the creation phase, it is necessary to provide the CPI targetcpifilechecksum, which acts as an ID. If the VNodes already exist and you are upgrading the CPI, change the VNodes' status to MAINTENANCE and upload the new CPI.

In this project, the CPI is composed of a CPB bundle with 2 signed CPKs that contain the workflow grandle project, the contracts grandle project, and the group-policy.json that defines the policy that the group follows as presented in the figure 3.15. In this case, it defines a static network with defined initial VNodes and a Notary that also has a CPI file to deploy it, containing all the information and configuration necessary.

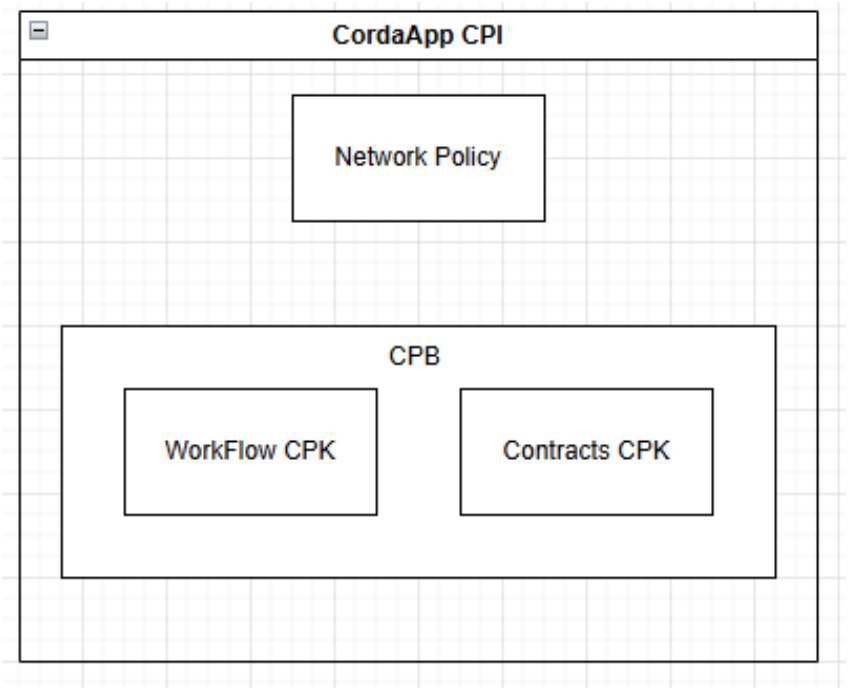


Figure 3.15: Structure of the CPI file

After registration of the Vnodes, it is possible and has been made necessary to change the values of the configuration on some sections of the CordApp. This is made possible through a PUT request made to the Corda app "/config," where the parameters are defined: section, config, schemaVersion, and version. The schemaVersion and the version are the type of version and the current version, and the section and the config are the name of the section being changed and the new configuration.

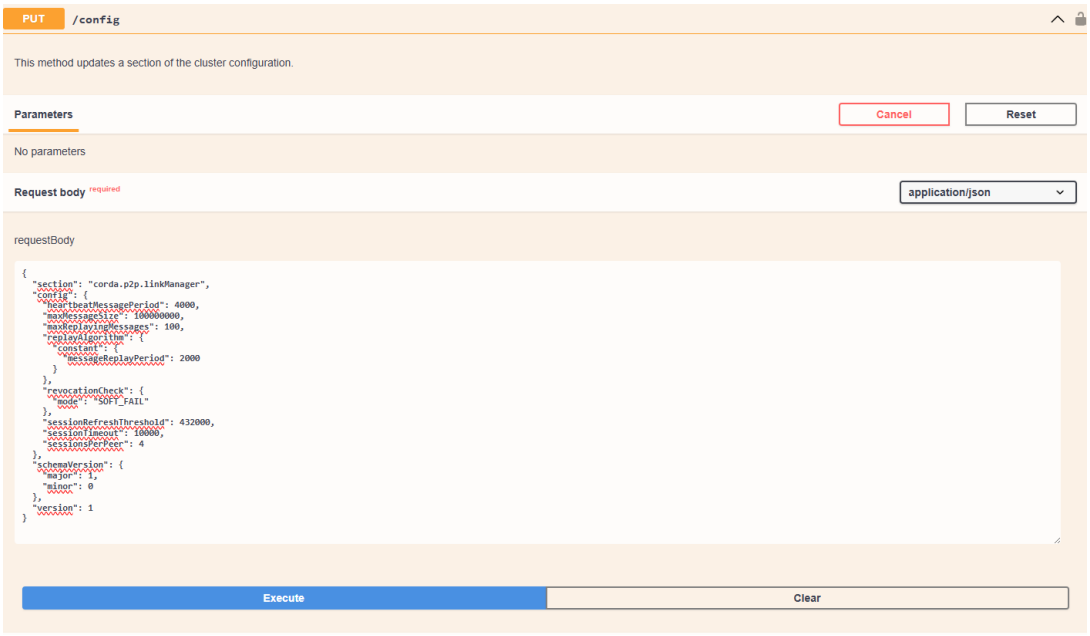


Figure 3.16: Example of a configuration change in the link Manager config.

In this example 3.16, there is a change being made in the link manager of the application, the section that is in charge of peer-to-peer communication. This change seeks to change the maximum message size from the default value of 1000000 bytes (1Mb) to 100000000 (100Mb) bytes and the heartbeatMessagePeriod from 2000 to 4000. This change enables more in-depth analyses between smaller vs. larger payloads being sent through the system and uploading pdf´s. Afterwards, the new configuration can be view through a GET Request "/config/section" where section is the altered section as the example in the figure 3.17 shows.

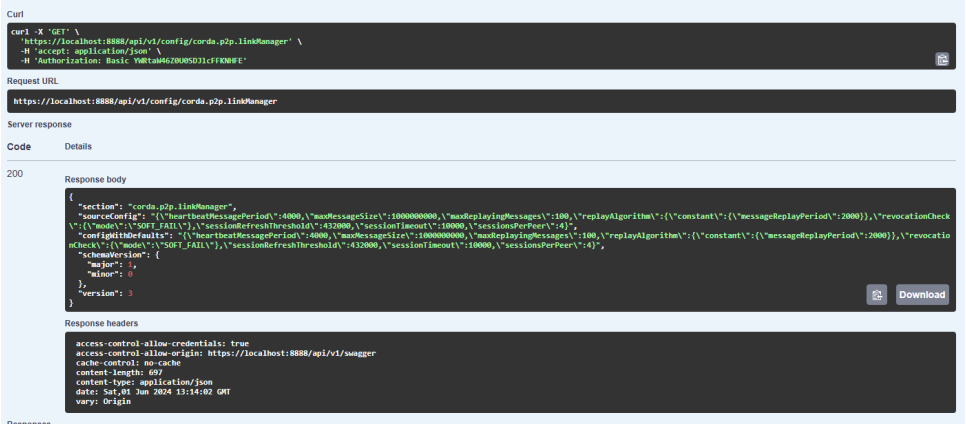


Figure 3.17: Result of the Get request to receive the configuration of a section.

In the response 3.17, it has all the previous arguments mentioned before, plus the default value; if it only has that argument and an empty sourceConfig, it will use those values as

needed. If the CPI needs to be changed with new code or group network policy, then the process to create a new CPI with a new version needs to be followed, except for the publishing of the certificates.

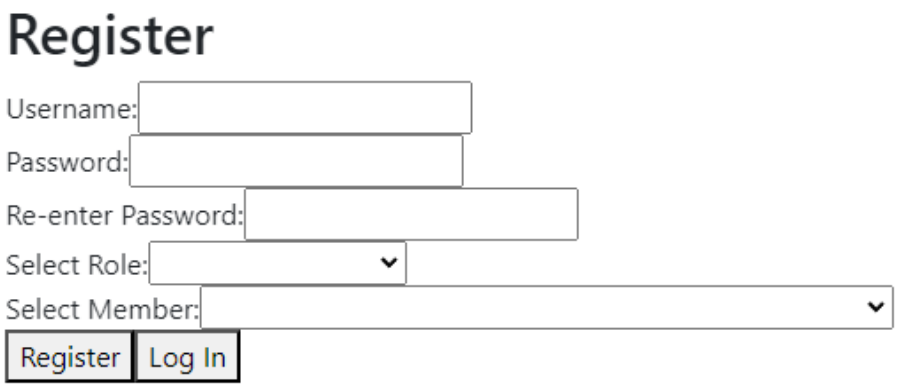


Figure 3.18: Registration page in Angular.

In the Angular project, it starts with the login page and with a register button to go to a register page. In the registration, it contains the inputs necessary to register the member with the VNode selected, thought getting the X500 Names (International Telecommunication Union, 2019) from a GET request to collect the necessary information as the figure 3.18 shows.

After filling the necessary information, it will make a request to the REST API C sharp project, which is responsible for handling the synchronous and asynchronous requests to check the status of requests made to Corda and return their results. In this case, if it's successful, it will return the transaction ID, its private and public keys, with the latter displayed here for the user to download the information 3.19.

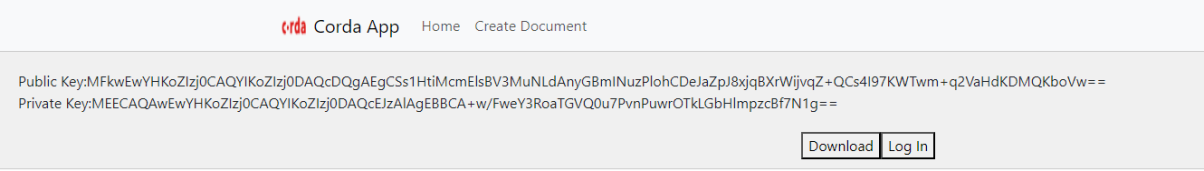


Figure 3.19: Public/Private Keys in RSA/ECB/PKCS1 Padding encryption of the user after successful registration.

Afterward, the user can go and Login through the button on the button or the button on the right-hand side, replacing the profile button because the user isn't logging in 3.20.

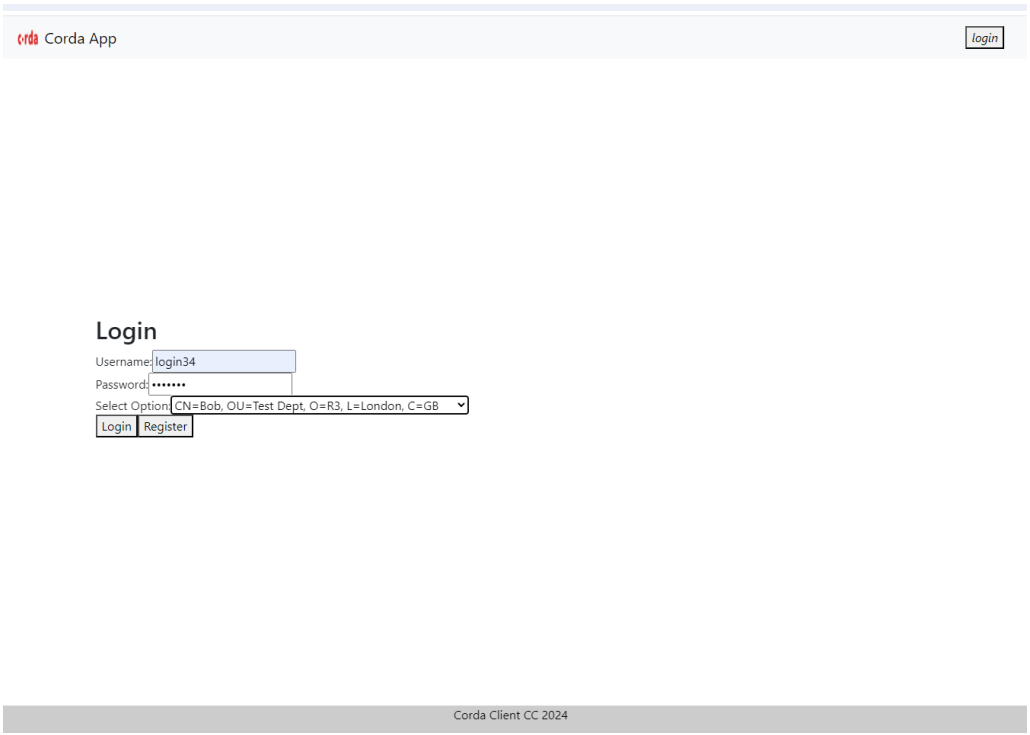


Figure 3.20: Authorization page.

After adding the newly created credentials, the user can make a request similar to the registration request, but with the classname to get the JWT token (Jones *et al.*, 2015) used for authentication. Afterward, the Angular project stores the token given by the result of the request in session storage and navigates to a new page, Home 3.21.

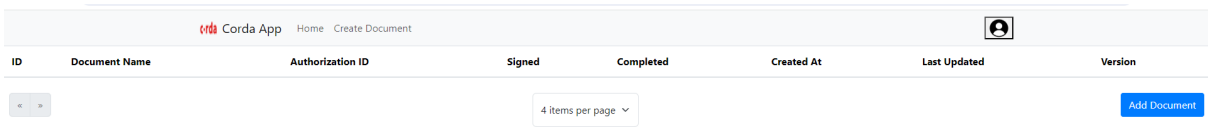


Figure 3.21: Home Page Displaying Documents Accessible to the Authenticated User.

The homepage has a list, initially empty, of documents that the user has access to. In the initialization phase, it will construct a request to receive a list of documents, parse through the information received, and display it in a table where the user can view the current, unconsumed version of the document. If the user has the required role, they can create, sign, update, and view the history of the document. In this case, the user is a medical authority, and the button to create documents is available. After clicking on the button, it presents all the values to create a document 3.22.

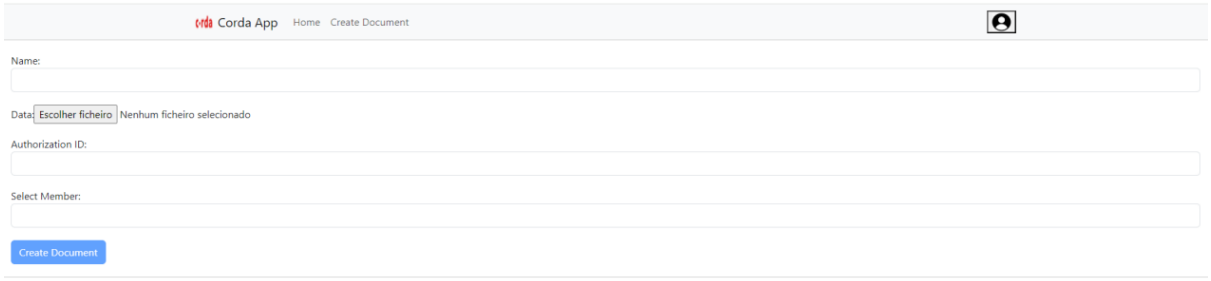


Figure 3.22: Document Creation Page with Form for Adding New Documents.

After successfully filling out and sending the request, it will then redirect to the home page, where the document is available with additional actions to update, view, and delete the current row and document 3.23.

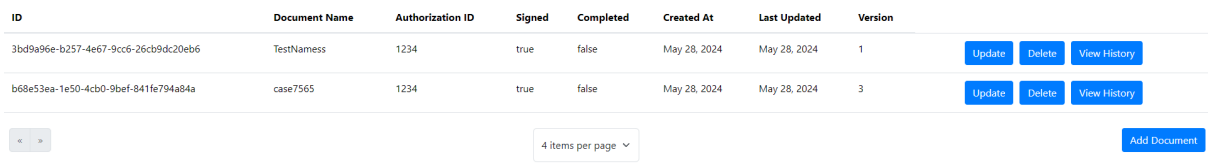


Figure 3.23: Home Page with the documents that user has access.

If the user requires to update the record, then after selecting the document update button, it presents the document's basic information with inputs that allow edition, then after pressing update, gather the values and send them in a request to start a flow that allows its update as outlined in the Corda project 3.24.

Figure 3.24: Update Page with the document information.

The delete button makes a request with the ID and token that creates a new transaction with the record State Deleted, and then, after processing the return request, the user will see the list

without the document, as only active records are given to the user. The View History displays all the changes made over time, as well as the data that the home page list doesn't display 3.25.

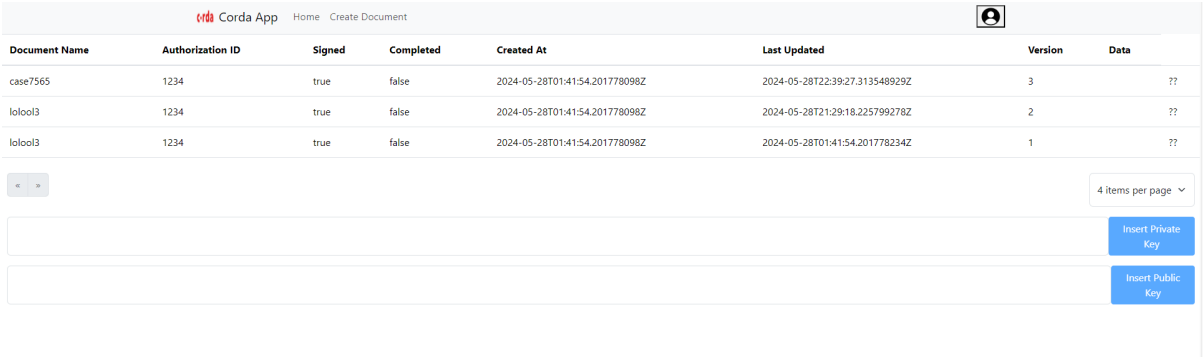


Figure 3.25: Documents History Page without Keys submitted.

That data is unavailable to be downloaded because the keys weren't given. The user can give its keys to decrypt the data received from the cordApp. The files are encrypted using an RSA Encryption with Electronic Codebook Mode and PKCS 1 v1.5 Padding (RSA/ECB/PKCS1Padding) (Galla *et al.*, 2016) and then dencrypted by a Kotlin project available to the user through the REST API. After these processes, if the data is larger than 30 characters, the user is given a button to download the file 3.26.

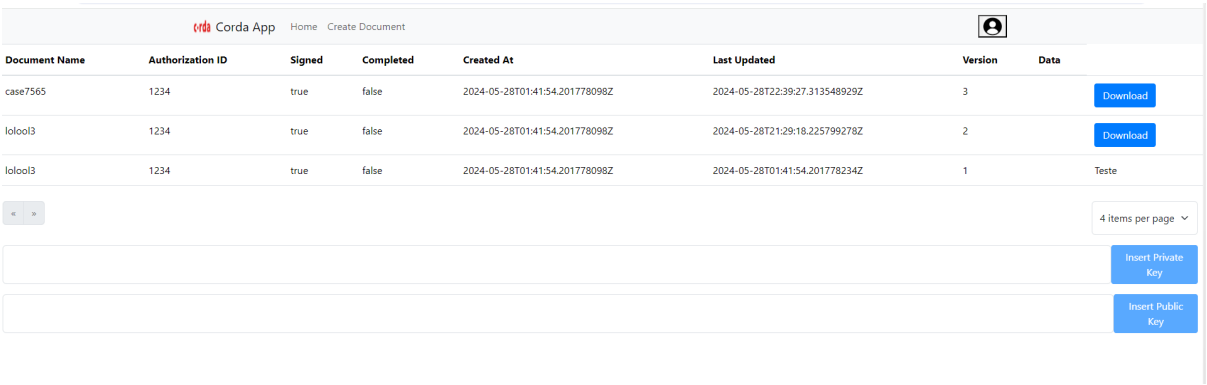
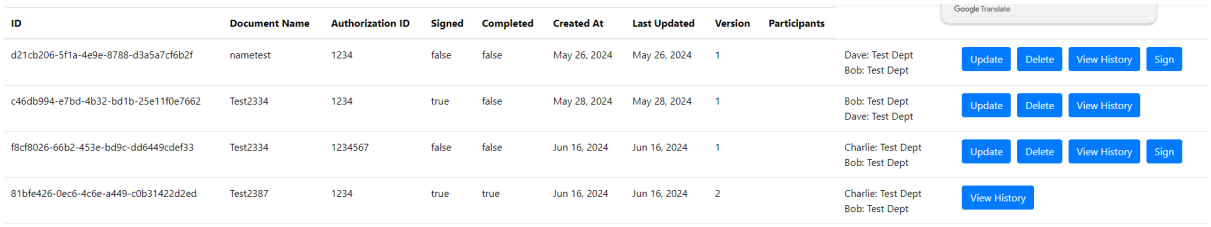


Figure 3.26: Documents History Page with Keys submitted.

On the other participant's homepage, there will be a document available that requires their signature. Participants can only perform actions that their role allows them to. When a participant presses the "Sign" button, it triggers a specific action for that participant to sign the document, similar to updating the document, and initiates a workflow.

Once a participant signs the document, their signature is recorded. The document remains editable until all required participants have individually signed it. After each required participant has signed, the document will become inedible, preventing any further changes. This

approach ensures that each participant's signature is collected separately before finalizing the document 3.27.



ID	Document Name	Authorization ID	Signed	Completed	Created At	Last Updated	Version	Participants		
d21cb206-5f1a-4e9e-8788-d3a5a7cf6b2f	nametest	1234	false	false	May 26, 2024	May 26, 2024	1		Dave: Test Dept Bob: Test Dept	Update Delete View History Sign
c46db994-e7bd-4b32-bd1b-25e11f0e7662	Test2334	1234	true	false	May 28, 2024	May 28, 2024	1		Bob: Test Dept Dave: Test Dept	Update Delete View History
f8cf8026-66b2-453e-bd9c-dd6449cdef33	Test2334	1234567	false	false	Jun 16, 2024	Jun 16, 2024	1		Charlie: Test Dept Bob: Test Dept	Update Delete View History Sign
81bfe426-0ec6-4c6e-a449-c0b31422d2ed	Test2387	1234	true	true	Jun 16, 2024	Jun 16, 2024	2		Charlie: Test Dept Bob: Test Dept	View History

Figure 3.27: Home Page after signing a document.

The profile page contains basic information about the user 3.28.

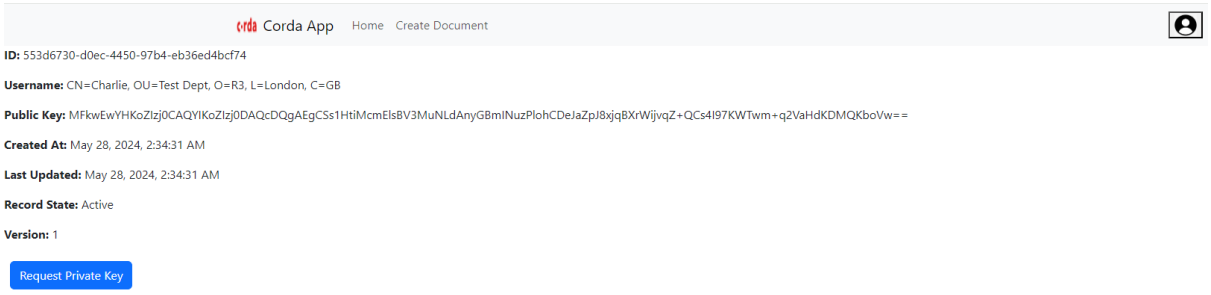


Figure 3.28: Profile History Page.

The profile history page, depicted in Figure 3.28, displays a comprehensive overview of the user's profile information with the added option of requesting the keys in case the user requires them.

3.5 Summary

Throughout this study, the deployment and application of Corda Distributed Ledger Technology (DLT) within a clustered network environment have been explored. The deployment process involved setting up and configuring Corda modules within the cluster, facilitated by Helm for orchestration and management. Additionally, the configuration of cluster modules was streamlined using the Corda Package Installer (CPI).

Looking forward, the next chapter will conduct a comparative analysis between the proposed architecture and its practical implementation. This analysis will evaluate how well the deployed Corda DLT solution aligns with initial design goals, identify any deviations or challenges encountered during deployment, and explore opportunities for further optimization. By

examining these aspects, valuable insights into the practical application of Corda DLT within clustered environments and its implications for future deployments in enterprise settings will be provided.

Corda DLT has been successfully deployed and used in a clustered network, demonstrating its ability to enable safe, authorized interactions and improve the efficiency of distributed applications. The comparative analysis in the upcoming chapter will offer deeper insights into the operational effectiveness of Corda DLT and inform strategies for enhancing its deployment in similar clustered environments.

Chapter 4

Analyses and Results

The project sought and accomplished its main objectives: implementing a private blockchain solution using Corda. This section presents the detailed analyses and results of the implementation process, focusing on the technical aspects, performance metrics, and overall outcomes of deploying the private blockchain.

The assessment of the application encompasses four key aspects: privacy, security, data integrity, and scalability. Each is critical to a private blockchain-based network.

Privacy is assessed through tests that evaluate the effectiveness of algorithms that restrict the access of the records that exist within the system. These tests are mainly focused on verifying that only authorized parties have access to ledgers that they are included in through the participant lists and internal Corda methods to determine the virtual node identification.

Security is assessed through tests that verify the use of cryptography in the application within the internal mechanism of the Corda application with the pod "crypto" and the encryption of data in the document state.

Data integrity is assessed through tests that ensure that the validations present in the contracts are correctly used and that participants have access to the ledgers records and the validations of the participants and notary.

Scalability is assessed through the system's ability to efficiently manage growth by limiting the number of participants associated with each ledger.

4.1 Improvements Made

After considerations by the company ByMe, several changes were made to the application to improve the user interface and overall user experience. These enhancements are categorized as follows:

4.1.1 User Interface Enhancements

- **Participants**: The participants involved in the ledger were added to the document list on the homepage.
- **Tooltips**: Tooltips were added to improve readability to the contents and buttons.

These enhancements were implemented by updating the Frontend components and Backend services, ensuring a more intuitive and user-friendly experience. Additionally, these changes were supported by creating a new version of the API to streamline data retrieval and presentation, enhancing overall user satisfaction and efficiency.

4.1.2 Performance Improvements

- **Consistent Code**: A consistent methods to improve code readability and remove redundancy.
- **Optimized Corda Workflows**: Enhanced the performance of Corda by optimizing workflows.

The application underwent significant improvements in performance, focusing on optimizing both code efficiency and data retrieval speeds.

To enhance code maintainability and readability, consistent coding methods were implemented throughout the application. These improvements ensure that the codebase is standardized, making it easier to understand, modify, and extend in the future.

Additionally, substantial efforts were made to optimize Corda's database queries. This involved cleaning some unnecessary code and filtering out documents that are not to be presented to the user.

4.1.3 Future Improvements

- **Enhanced Data Control**: Enable users to further customize their tables by adding, removing, and rearranging filters to suit their preferences.
- **Advanced Search Functionality**: Upgrade the search functionality to support more advanced filters, facilitating quicker and more precise data retrieval.
- **Document-Based Notifications**: Implement notifications that are triggered by changes or activities related to specific documents within the Corda App. This will keep users informed about important updates to relevant documents in real-time without requiring them to request the changes.
- **Collaboration Tools**: Introduce new collaboration tools to allow users to share and work on documents, including features like chat, comments, version control, and reviews.

In the future, the application is set to undergo several enhancements aimed at further enriching user experience and functionality. Plans include expanding user control over data tables and allowing for customization through interactive tables that enable the addition, removal, and rearrangement of filters based on individual preferences. This enhancement aims to provide users with more tailored and efficient data management capabilities.

Real-time notifications are also slated for integration, ensuring users stay informed about important updates and activities instantly without the need for manual refreshes.

Furthermore, new collaboration tools will be introduced to enable seamless real-time project collaboration. Features such as comments and file previewer are expected to enhance the user experience and efficiency in the application.

These forthcoming enhancements are designed to elevate the application's utility, responsiveness, and user satisfaction, supporting smoother operations and enriched collaborative capabilities.

4.2 Time Performance Analysis

To provide a thorough understanding of the system's behavior under different scenarios, there are three primary scenarios: authorization in, creating a document, and updating a document.

Each scenario will be evaluated based on the time required to complete the respective operations through HTTPS requests to the Corda application with postman. These requests will utilize the holding identity short hash of the user Dave, as detailed in the test group outlined in table 4.5.

This evaluation will provide valuable insights into the system's performance and responsiveness.

4.2.1 Scenario 1 - Authorization

This scenario involves the process of authenticating users into the system. Users will provide their credentials (such as loginName and password) to access the system, and upon successful authentication, they will be granted access to their respective virtual nodes and functionalities.

The time performance in this scenario primarily measures how long it takes from the moment the user submits their credentials until they gain a JWT token to gain access to the system. Factors influencing this time include the efficiency of the authentication service, network latency, and the complexity of security protocols involved.

4.2.2 Scenario 2: Create Document

In this scenario, a user creates a new document ledger that contains the necessary information for the document state. Users with the appropriate permissions can initiate this process by

filling out the requestBody with the necessary arguments to create the document.

If authorized, the system processes the create and commits the new transaction to a new document ledger. The time performance in this scenario measures the duration from the user submitting the create request until the create is confirmed with a Get request, with potential delays arising from network latency or the complexity of the transaction.

4.2.3 Scenario 3: Update Document

This scenario involves updating an existing document by creating a new transaction within the document's ledger. Users with the appropriate permissions can add to an existing document ledger by creating a new state with the updated information, ensuring that all changes are securely recorded and auditable within the Corda application.

If authorized, the system processes the update and commits the new transaction to the document ledger. The time performance in this scenario measures the duration from the user submitting the update request until the update is confirmed with a Get request, with potential delays arising from network latency or the complexity of the transaction.

4.2.4 Time Analysis

The table above presents the time taken to complete each scenario, measured in milliseconds (ms). This analysis helps in identifying any performance bottlenecks and provides a baseline for future optimizations.

It is important to note that the initial requests for authorization, creation, and update are asynchronous. Consequently, subsequent requests to check the status of the workflow and obtain the duration are based on the timestamps recorded at the initial timestamp obtained after the creation of the request (example in the figure 4.1) and the timestamp on completion of each workflow.

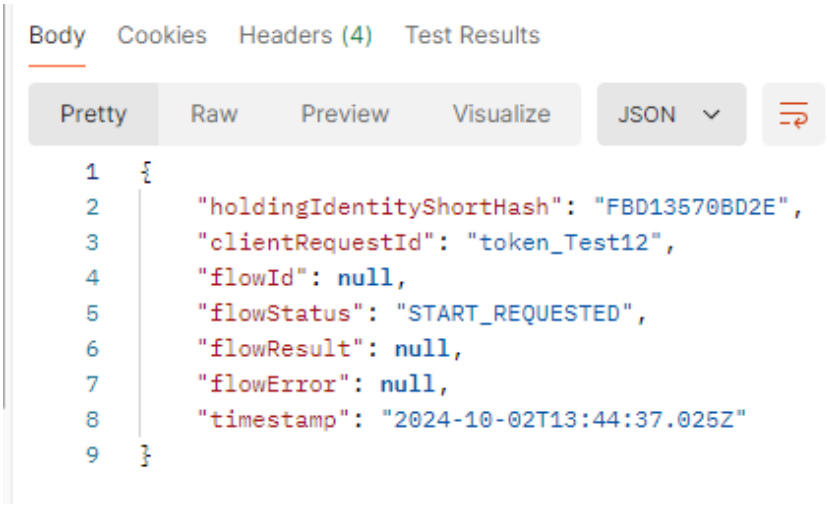


Figure 4.1: Example of a return message of a request for the creation of a workflow.

The duration taken for these workflows was calculated using the difference between the timestamps, ensuring an accurate measurement of the overall workflow duration, and the timestamps presented were altered to remove the date and preserve just the hours, minutes, seconds, and milliseconds.

Table 4.1: Time Analysis

Scenario	Initial Time	Completion Time	Time (ms)
Scenario 1	10:15:08.684	10:15:08.792	108
Scenario 2	10:32:38.828	10:32:39.351	523
Scenario 3	10:39:26.606	10:39:27.171	565

The performance analysis in 4.1 of the Corda application, as presented in the table, reveals varying time taken to complete different scenarios, measured in milliseconds (ms).

Scenario 1, which completes in 108 ms, is the fastest among the three, indicating it likely involves less complex operations, such as an initial login request or a simple data retrieval.

Scenario 2 takes 523 ms, nearly five times the time of Scenario 1, suggesting additional processing is required, possibly for creating a ledger in the Corda global state.

Scenario 3, with a completion time of 565 ms, is the most time-consuming. This significant duration likely reflects more complex operations, such as conducting transactions involving multiple validation steps and computations with the addiction of a new state in the ledger.

The noticeable differences in completion times across the scenarios highlight potential areas for performance optimization, particularly for more complex operations, to enhance the overall efficiency and responsiveness of the Corda application.

In a study examining time performance with Hyperledger Sawtooth (Moschou *et al.*, 2020), the findings revealed that the scenario involving 10 transactions and 5 validators achieved a total processing time of 6787.2 ms, resulting in an average of 678.72 ms per transaction.

Comparably, even the more complex workflows, with a processing time of 565 ms, exhibited faster performance than this already efficient scenario. In contrast, scenarios with additional validators tended to increase the time per transaction, while those with a higher number of transactions remained even more time-consuming.

This test indicates that while simple operations in the Corda application are performed relatively quickly, more complex tasks can significantly increase processing time. Optimizing these processes, especially the more time-intensive ones, could substantially improve the application's performance and user experience.

4.2.5 Discussion of Results

In the Corda application, there are three primary types of requests: retrieval, creation, and update. Each of these types involves different processes and complexity levels.

- **Retrieval Requests**

  Retrieval requests involve fetching information from the system, such as logging in or getting documents. These requests are generally fast because they do not require consensus from other nodes and mainly involve querying the database.

- **Creation Requests**

  Creation requests involve adding new records to the blockchain. This process starts with an initial retrieval of information, followed by the creation of a new ledger relevant to the request. It involves the consensus layer, where other participants and a notary validate the new entry. This step ensures that the new record is accurate and secure.

- **Update Requests**

  Update requests are the most complex and time-consuming. They start with an initial retrieval of information and then proceed with additional steps to fetch the current ledger state. A new state is then created, and the previous state is marked as consumed. This ensures the immutability and accuracy of the blockchain, but it also requires more processing and validation, leading to longer completion times.

As demonstrated in the time analysis for each scenario, the varying complexity of these operations leads to different execution times, ultimately influencing the overall system performance.

By understanding these different types of requests and their respective complexities, it can better optimize the system's performance and improve overall efficiency for future development.

4.3 Project Analysis: Proposed vs Implemented

This section focuses on analyzing which components were implemented as part of the project compared to those proposed initially.

In this analysis, it will specifically identify and discuss the components that were created during the implementation phase of the project compared to what was originally proposed. This evaluation aims to highlight any differences between the initial plan and the actual execution in terms of component creation.

Table 4.2: Implementation Status of User Requirement Features (URF)

Requirement	**Status**
URF001: Register a user in a virtual Node.	Done
URF002: Authorization a user.	Done
URF003: List all available documents for the user.	Done
URF004: (Medical Authority) Be able to create new consent forms.	Done
URF005: (Medical Authority) Be able to update consent forms.	Done
URF006: Be able to view the history of transactions of a consent form.	Done
URF007: Be able to sign consent forms.	Done
URF008: Be able to view the profile of the user.	Done
URF009: Be able to store files in the consent form.	Done
URF010: (Medical Authority) Be able to add participants in the creation of the document.	Done

The table 4.2 presents the current implementation status of the User Requirement Features, detailing various functionalities and their respective statuses within the system. As presented, the implementation of the proposed project presents all of the user requirements, from the client-side logic that the user directly underacts with to the corda-side implementation that fulfills those requirements.

Table 4.3: Implementation Status of User Requirements (URNF)

Requirement	**Status**
URNF001: The files must be encrypted.	Done
URNF002: Be able to decrypt files.	Done
URNF003: Operate with a 50 users.	Done
URNF004: Operate asynchronously.	Done
URNF005: Operate continuously, day and night.	Done

This table 4.3 summarizes the implementation status of user requirements for the project, provided the scalability, security, and availability. Key requirements such as file encryption, decryption, and asynchronous and continuous operation have all been successfully implemented.

These functionalities ensure that sensitive data remains secure through encryption measures while enabling the system to handle a significant user load efficiently.

Table 4.4: Implementation Status of System Requirements (URS)

Requirement	**Status**
URS001: Utilize Kotlin or Java for Corda application development.	Done
URS002: Implement Corda's distributed ledger technology (DLT).	Done
URS003: Incorporate Corda's built-in security features.	Done
URS004: Develop a web interface using React.js or Angular.	Done
URS005: Utilize Corda's API interface for communication.	Done
URS006: Utilize a reliable network infrastructure with low latency.	Done

This table 4.4 summarizes the implementation status of the software and hardware requirements for the project, all of the points are realized and implemented in the parameters defined.

4.4 Performance Testing and Evaluation

Validation functions are necessary for maintaining the security and integrity of applications, particularly when it comes to managing user access and protecting sensitive data. These functions ensure that only registered users can be authorized in, access specific states, or perform certain actions within the Corda application. By validating user credentials, permissions, and tokens, the application safeguards against unauthorized access and potential security breaches.

4.4.1 Introduction

In these tests, the system's ability to detect and respond to unauthorized access attempts is checked and rejected. Scenarios such as incorrect authorization credentials, expired or invalid tokens, and unauthorized attempts to access restricted states are simulated. Additionally, these tests will verify that the application is correctly processing authorized requests and providing the expected results.

The objective is to assess the effectiveness of the validation mechanisms in identifying and acting on these potential compromises, ensuring that only authorized users can interact with the system in accordance with the defined security policies.

The tests include virtual nodes that represent a user who is registered in the Corda app and serve as the subjects of the various test scenarios. These users are assigned different roles and permissions to evaluate how the Corda app enforces access control and manages authorized actions.

Table 4.5: Test Virtual Nodes in Corda App

X500 Name	**Holding Identity Short Hash**	**Role**
CN=Dave, OU=Test Dept, O=R3, L=London, C=GB	FBD13570BD2E	Medical Authority
CN=Bob, OU=Test Dept, O=R3, L=London, C=GB	54CC7B7B6F91	Patient
CN=Alice, OU=Test Dept, O=R3, L=London, C=GB	72DF97C268B7	Medical Authority
CN=Charlie, OU=Test Dept, O=R3, L=London, C=GB	986A6712407A	Patient

Table 4.5 provides an overview of the user roles within the Corda application. Each virtual node is identified by their X500 name and a unique holding identity short hash used for requests. The table distinguishes between two primary roles: medical authority and patient. Users assigned the role of Medical Authority, such as Dave and Charlie, are responsible for overseeing and managing sensitive medical data. In contrast, users like Bob and Alice, who are assigned the role of patient, have limited access, typically to their own medical records.

Each request made in the testing will contain a body in JSON format using postman, including a clientRequestId, which is unique for that virtual node, a flowClassName indicating a workflow, and a requestBody containing the arguments accepted in the workflow.

Ultimately, these tests will validate the robustness of the Corda App's security framework in a decentralized environment, ensuring that user roles and permissions are enforced appropriately.

4.4.2 Unauthorized login

This test focuses on assessing the Corda App response when incorrect authorization credentials are provided. The objective is to ensure that the application can accurately detect and prevent unauthorized attempts.

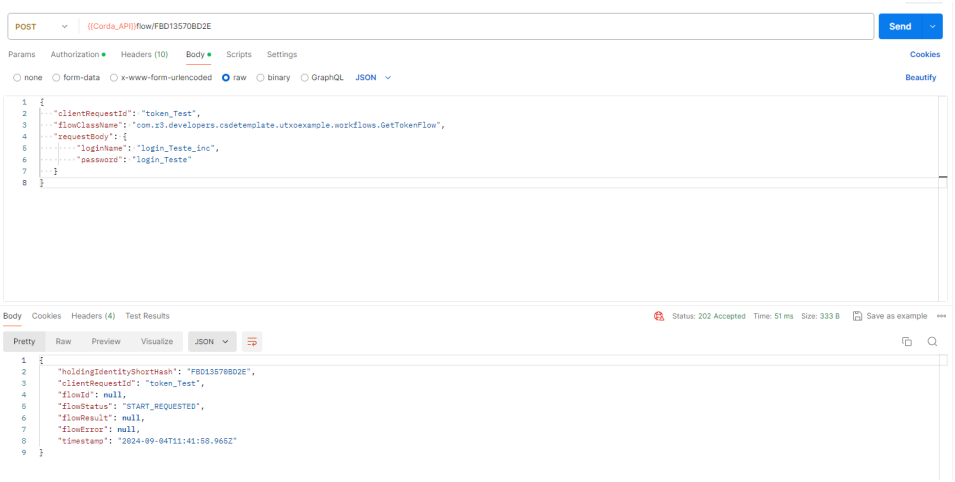


Figure 4.2: Authorization request with incorrect credentials.

The first figure 4.2 shows the authorization request with invalid details, specifically using the short hash of the virtual node of the user Dave (registered with the name and password `login_Test`). However, the credentials are intentionally altered by appending the extra characters `_inc` in the loginName, making them incorrect. The system attempts to validate this request and, as expected, the request is rejected.

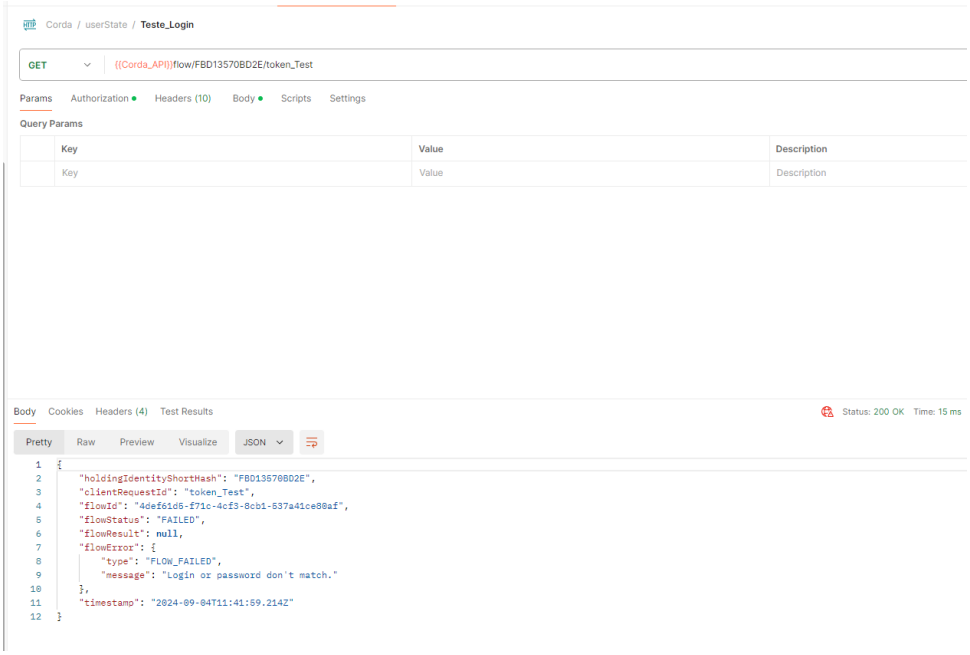


Figure 4.3: Authorization response with incorrect credentials.

The second figure 4.3 illustrates the system's response, in which access is denied. It returns an error message under the key flowError, indicating a failure to authenticate due to incorrect login credentials with the type `FLOW_FAILED` and the message "Login or password don't

match."

This response includes additional details such as the holdingIdentityShortHash, which identifies the user (Dave); the clientRequestId, generated for this specific request; the flowId, representing an internal Corda application ID; the flowStatus, indicating that the request failed; the flowResult, reflecting the outcome of the workflow; and the timestamp, recording when the request was made.

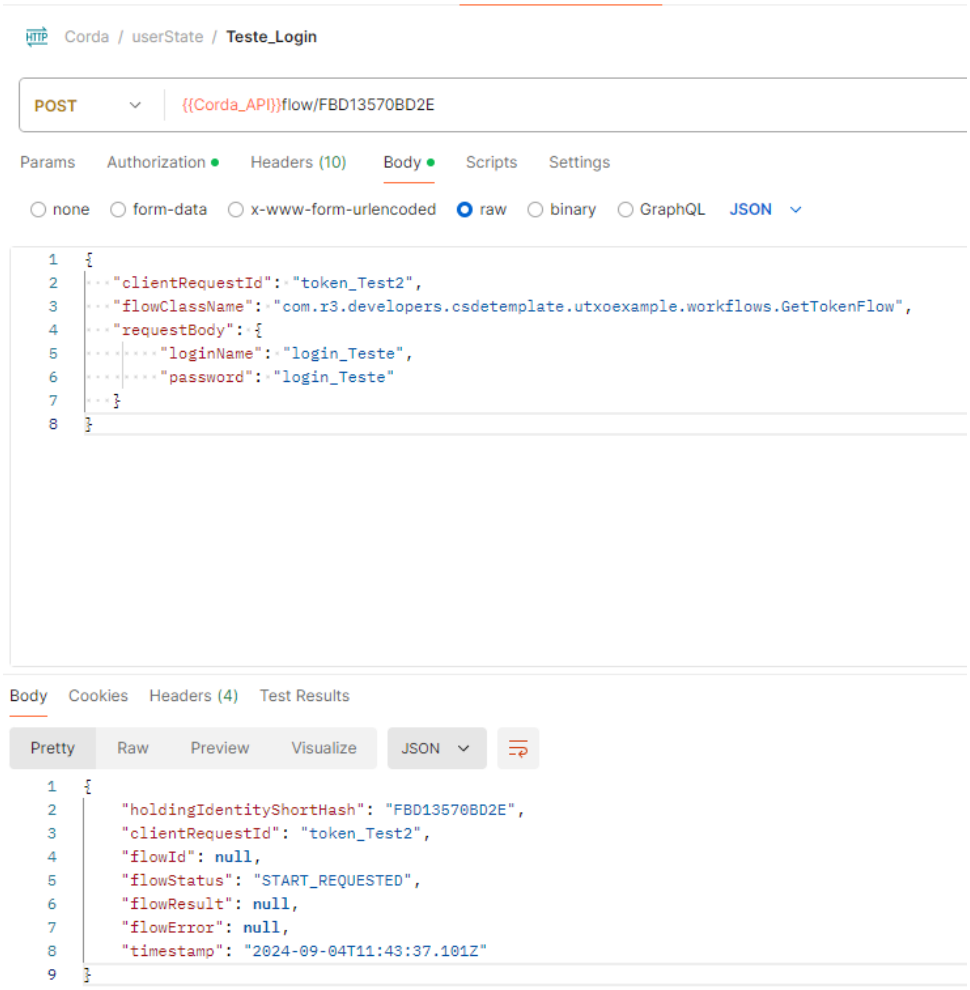


Figure 4.4: Authorization request with correct credentials.

The first figure, 4.4, illustrates the authorization request made with valid credentials without appending the extra characters `_inc` in the loginName. This request includes all necessary details that authenticate the user and facilitate access to the system.

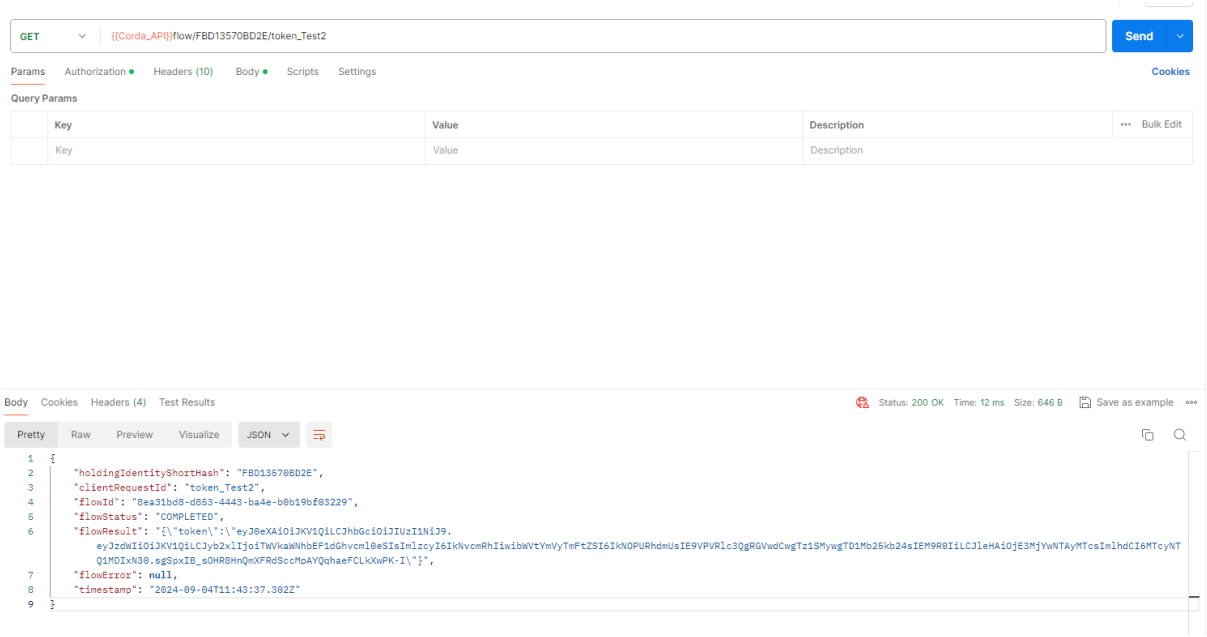


Figure 4.5: Authorization response with correct credentials.

The second figure, 4.5, presents the authorization response from the system. In this case, the system grants access by returning a JSON Web Token (JWT) in the key flowResult and presenting a "Completed" message in the key flowResult.

The token enables secure interactions with the system and access to the application through the virtual nodes, demonstrating the system's effective handling of authorization requests. Additionally, the flowStatus indicates that the request was completed successfully, and there were no errors in the flowError.

4.4.3 JWT Manipulation

This test is designed to assess the Corda App against tampering or manipulation of JWTs. These tokens are commonly used for authentication, and their integrity is critical to ensuring secure access control. Any unauthorized modification of JWT could potentially allow attackers to gain access to protected states and resources.

For this test, the JWT received by an authorization request of the user Dave will be used to make a request to retrieve the documents that they have access to. By using this token, the system can verify whether the JWT has been tampered with or altered. The integrity of the JWT is ensured by checking its values against the expected values for the member, thereby confirming that only unmodified and legitimate tokens are accepted.

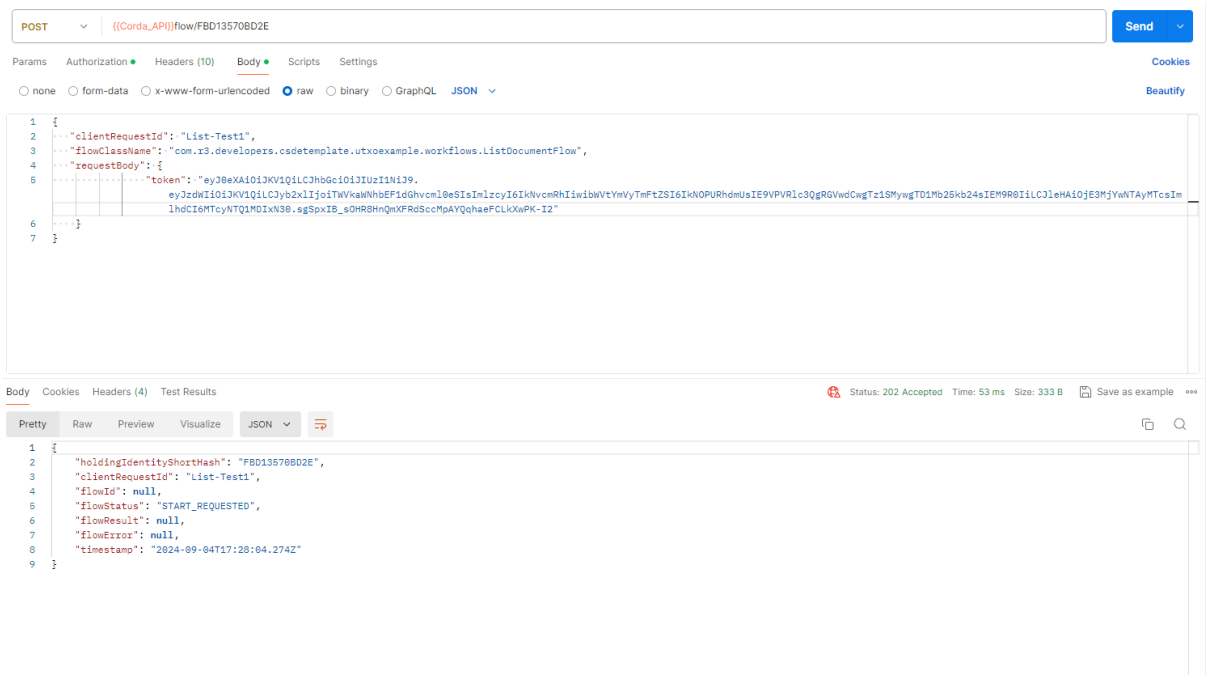


Figure 4.6: An request that requires an JWT that is incorrect.

The first figure, 4.6, illustrates the submission of the authorization request with a tampered JWT in the requestBody that has been deliberately manipulated by adding a '2', it receives the JSON file that informs the user the request has started.

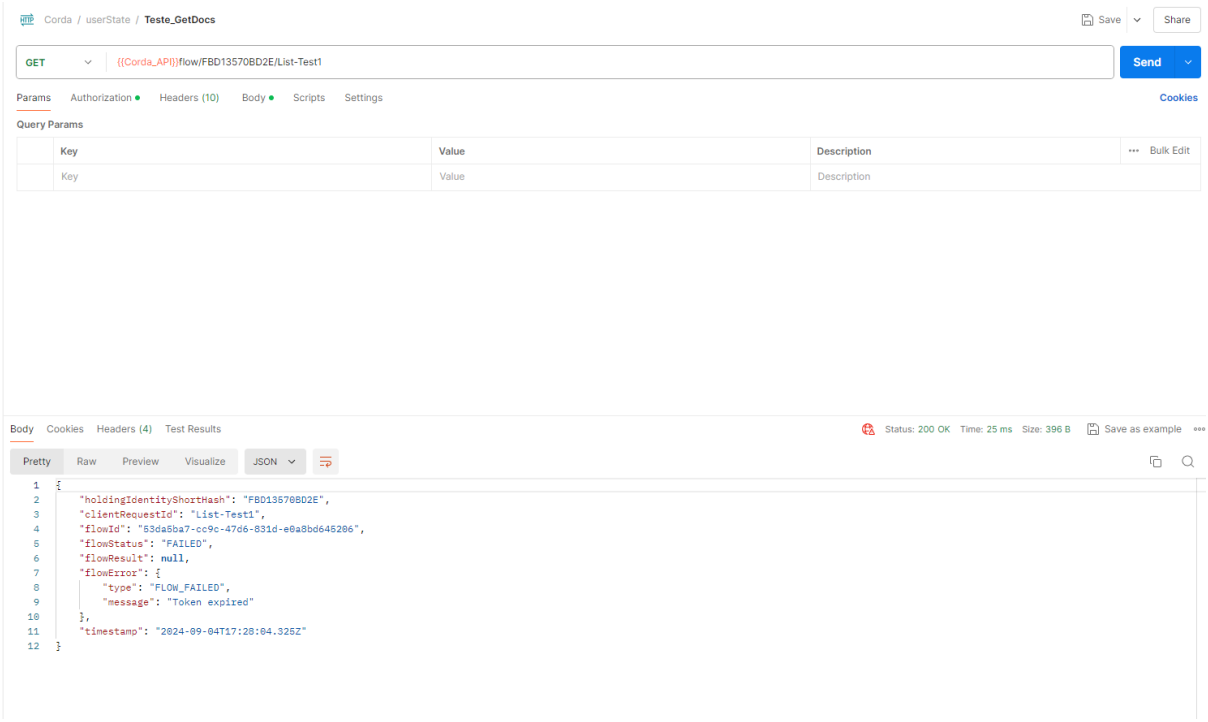


Figure 4.7: An response that requires an JWT that is incorrect.

The second figure, 4.7, shows the system's response to the request. This figure includes detailed information about the response structure, which indicates that the system has correctly identified the manipulated token.

The response includes a flowStatus marked as failed and an error message in the flowError key, explicitly stating that the token is expired. This error message serves as a generic JWT error response, effectively preventing unauthorized access to protected resources by notifying the user that the provided token is invalid.

Together, these figures demonstrate the system's robust error-handling capabilities and its commitment to maintaining security by rejecting requests with invalid or tampered tokens.

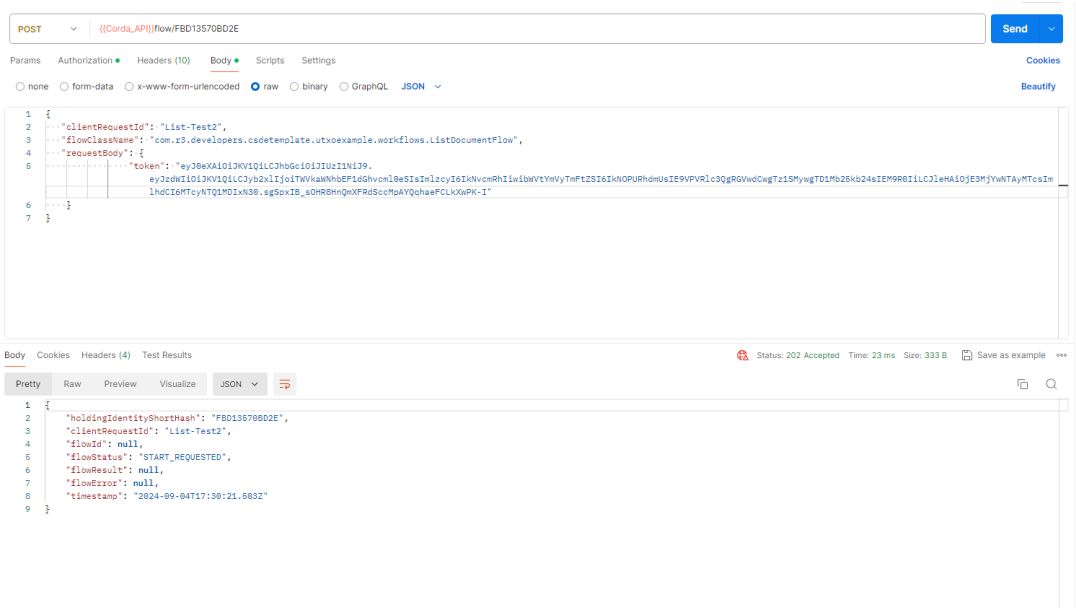


Figure 4.8: An request that requires an JWT that is correct.

The first figure, 4.8, illustrates the submission of the request with the correct token included in the requestBody. This JWT has been properly generated by the system during the authorization process and remains unaltered, ensuring its validity.

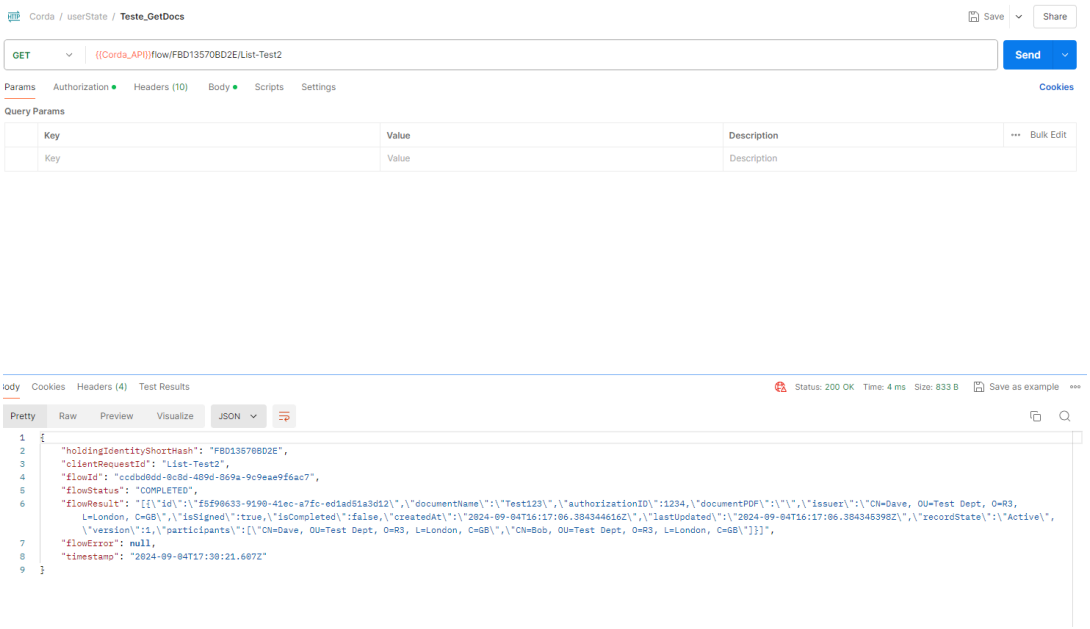


Figure 4.9: An response that requires an JWT that is correct.

The second figure, 4.9, presents the system's response to the authorization request. The figure details the response structure with the flowResult containing the list of the document this user has access and the flowStatus being "Completed".

This response confirms that the request has been successfully authorized, granting the user access to a list of documents formatted in JSON within the flowResult key.

Together, these figures demonstrate the seamless interaction between the user and the Corda application, highlighting the effective use of valid JWTs to facilitate secure access to information.

4.4.4 Decryption failure with incorrect key

This test is designed to ensure the security of data received from the Corda App by verifying that it is properly encrypted and can only be decrypted with the correct keys provided to the user during registration. The integrity of this process is crucial for safeguarding sensitive information.

This test highlights that any attempt to decrypt the data with incorrect or unauthorized keys should fail, thereby ensuring that only users with the correct keys can access the protected content.

The user David sends in two requests with the necessary arguments to decrypt the data in a specific document. One request includes an extra character in the private key to simulate an incorrect key, while the other contains the correct key without the extra character. Both requests are sent to the API and subsequently to the Kotlin project for decryption.

These tests occur in the document history, which requests and receives the history of the

document along with the encrypted data for each record, further ensuring the security and integrity of the information.

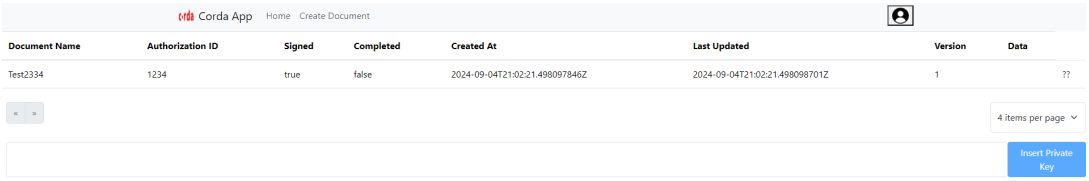


Figure 4.10: A history of a document with the name "Test2334" without key.

Figure 4.10 shows the history page of a document with the name Test2334 where no key is available for decryption. The list shows that there is no key stored for decrypting the data, indicating that access to the encrypted content is not possible without a valid key.

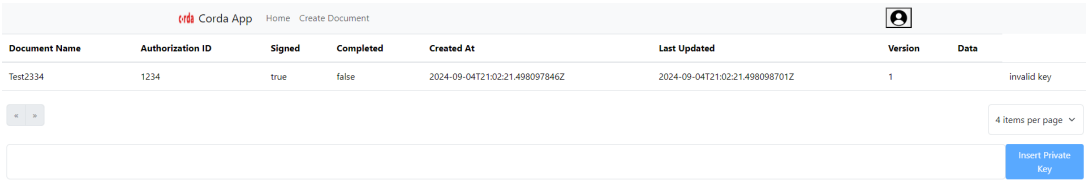


Figure 4.11: A history of a document with the name "Test2334" with an incorrect key.

Figure 4.11 demonstrates the result when an incorrect key is used to attempt decryption of the document Test2334. The response shows that the decryption attempt with the incorrect key fails, confirming that unauthorized keys cannot decrypt the data.

The request sends the encrypted data along with the private key provided by the user, illustrating the importance of using the correct key for successful decryption.

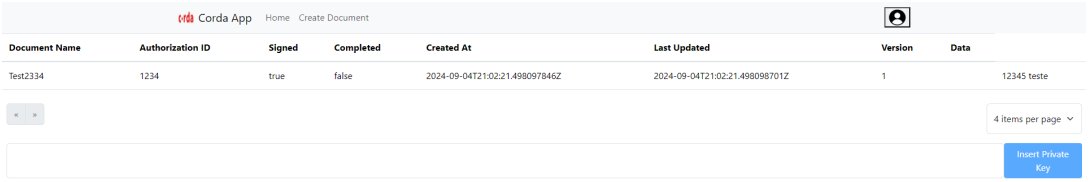


Figure 4.12: A history of a document with the name "Test2334" with a correct key.

Figure 4.12: This figure displays the history page of the document named "Test2334" when accessed with a correct key. Unlike the previous scenarios, this figure shows a successful decryption attempt. The document data is displayed, indicating that the correct key was used to decrypt the data in the Kotlin API.

This confirms that the system properly allows access to the content only when the correct key is provided, ensuring secure and authorized data retrieval.

4.4.5 Inaccessible Document

This test is designed to ensure the security of data received from the Corda App by verifying that users that do not have access to documents aren't able to get them through requests to the Corda app. By simulating attempts to access documents that weren't shared with different users through the participant list with the ID of the document, this test can evaluate the way access control is handled.

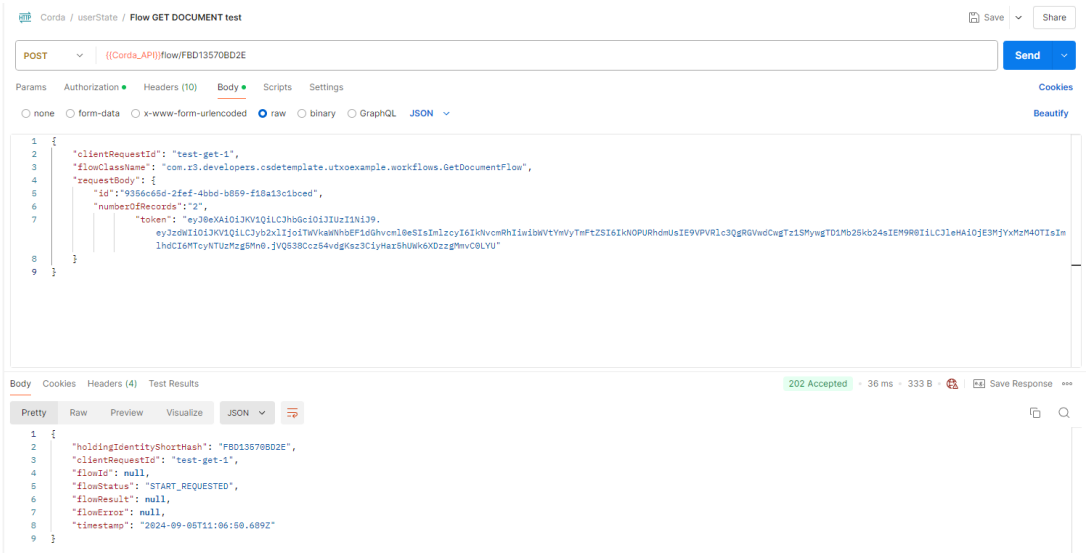


Figure 4.13: A request to attempt to access a document with a specific ID without access.

In the first figure 4.13, the request is made to access the document by specifying its unique ID, but since the user Dave lacks the necessary access, the request is expected to fail.

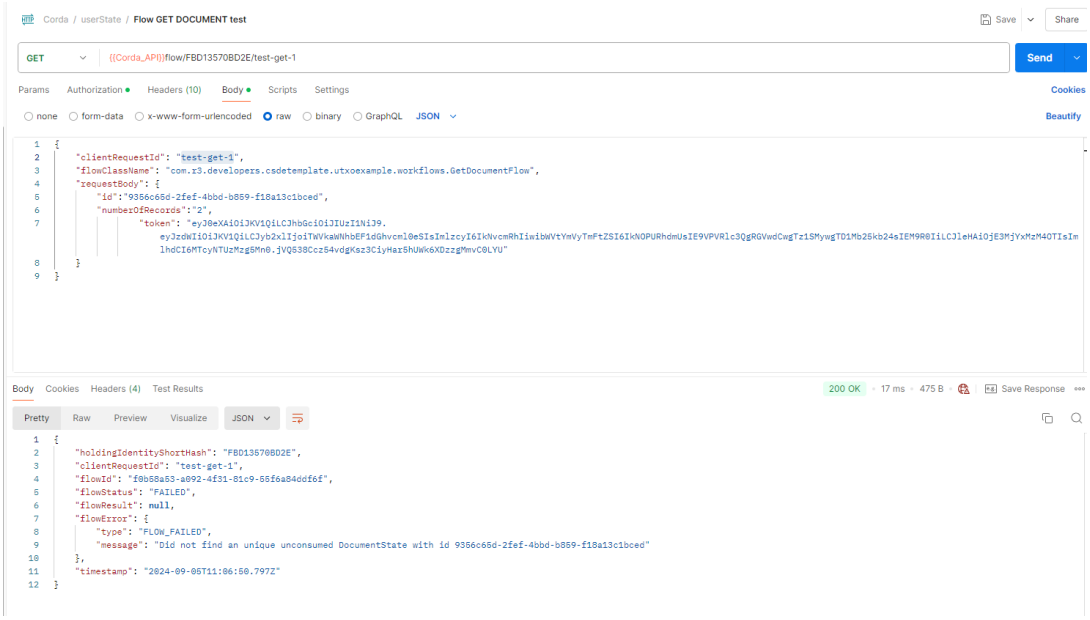


Figure 4.14: The response to attempt to access a document with a specific ID without access

The second figure 4.14 shows the corresponding response from the system, which denies access to the user Dave and returns an error message. This confirms that the Corda App correctly enforces access control policies by ensuring that only authorized users can view or retrieve documents.

The workflow performs a search for the accessible documents that the user has. In this case, the document does not contain the X500 name of the user in the list of participants and then returns an error with the message "Did not find an unique unconsumed DocumentState with the id..." with the id of the requested document.

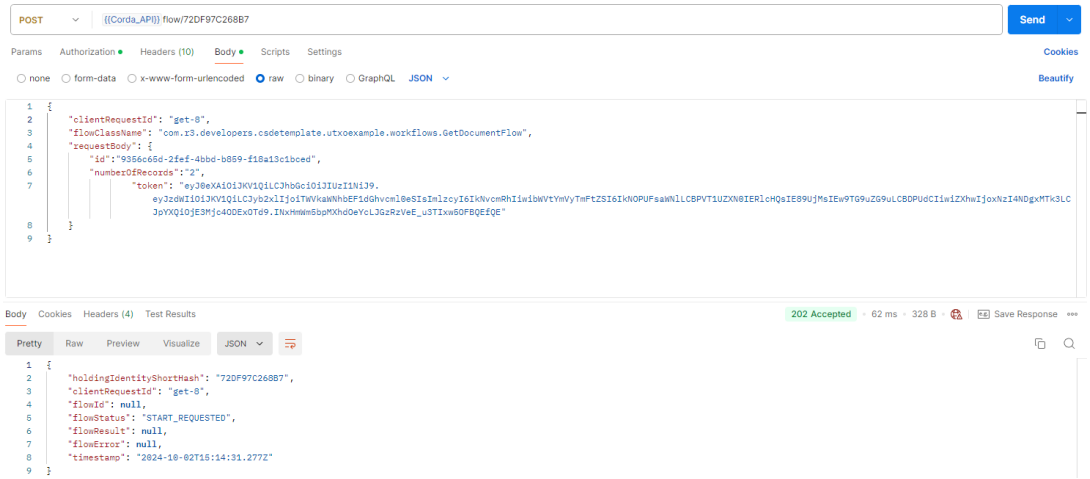


Figure 4.15: A request to attempt to access a document with a specific ID with access.

In the first figure 4.15, the request is made to access the document by specifying its unique

ID, and since the user Alice is a participant in the state of the document and she has the necessary access, the request is expected to succeed.

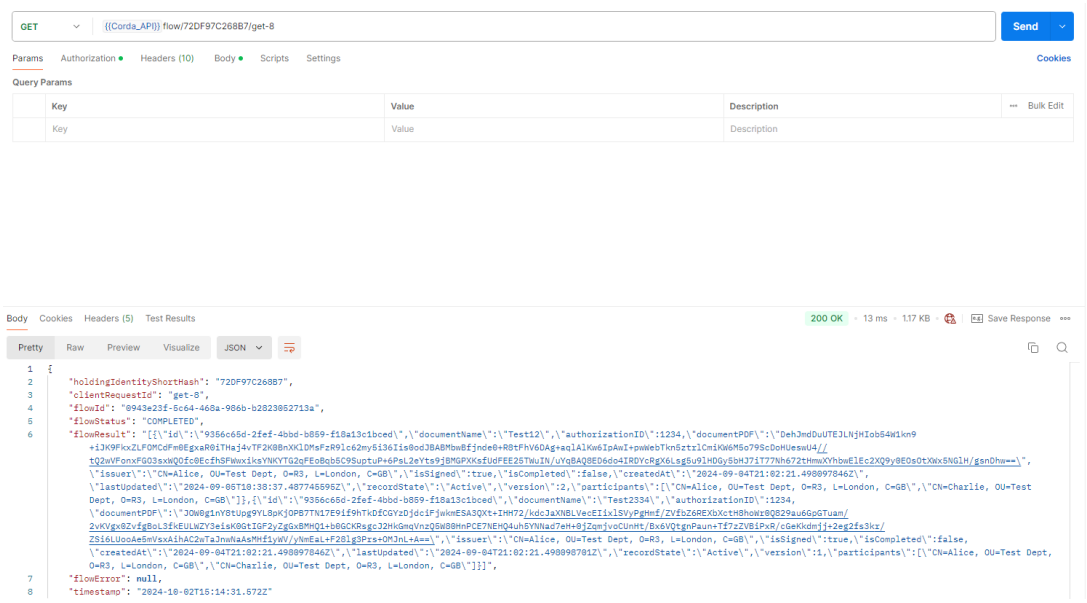


Figure 4.16: The response to an attempt to access a document with a specific ID with access.

The figure 4.16 shows the response to an attempt by the user Alice to access a document with a specific ID, this time with the correct access permissions.

Unlike the previous attempt, where access was denied because the user Dave was not included in the participant list, this request pertains to a document in which the user Alice is a participant. This response indicates that Alice successfully retrieved the document, as her name is included in the participant list selected by the document's creator.

The positive outcome of this request demonstrates that the Corda App effectively grants access to users who are participants, confirming that the access control mechanisms are functioning correctly for both authorized and unauthorized scenarios.

4.4.6 Executing workflows without permission

This test is designed to ensure that the Corda App enforces permission controls by preventing users from performing workflows they are not authorized to execute. The permissions are related to user roles, which determine what actions each user can perform.

In this test, attempts are simulated to update a document using a valid JWT and the required arguments sent in the request body. Two different roles are tested: the patient, user Charlie, who is not authorized to execute this workflow, and the medical authority, user Alice, who is allowed to perform the update. The test checks whether the system correctly denies the update attempt for the patient while allowing it for the medical authority as described in the table 3.2.

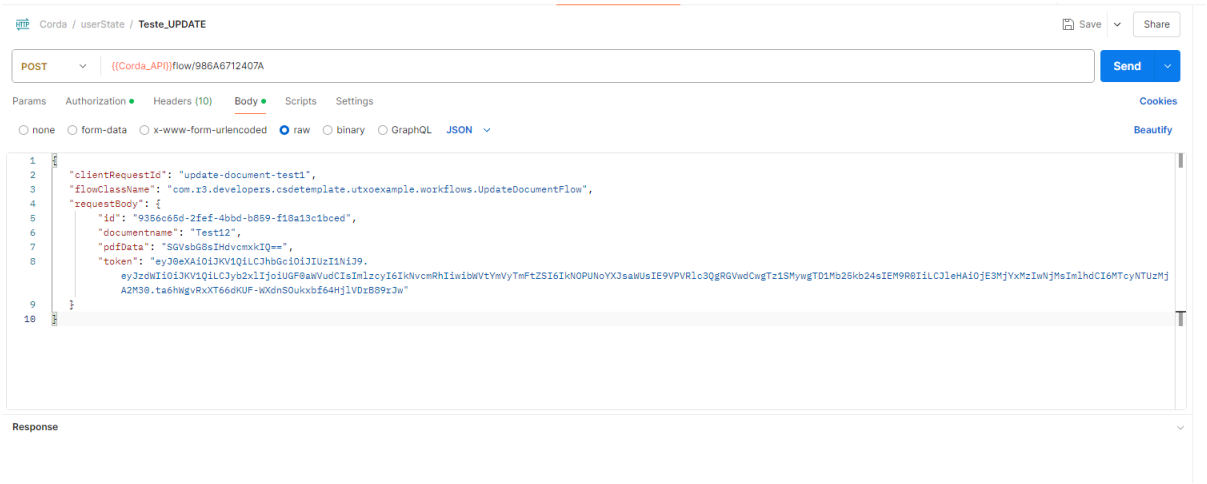


Figure 4.17: The user Charlie, attempts to submit a request to add a new document state to an existing ledger.

In the first figure 4.17, the user Charlie initiates a request to perform this action, despite lacking the necessary permissions that the role medical authority holds.

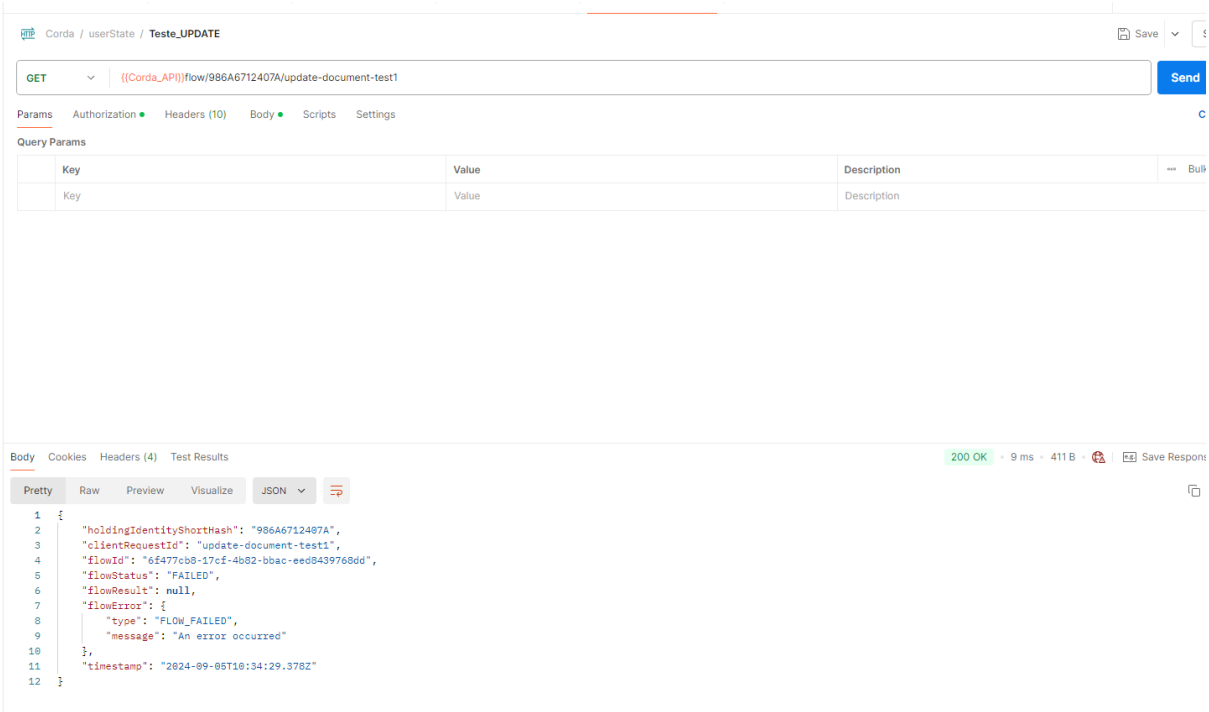


Figure 4.18: Response to the request that Charlie made for insufficient permission.

The second figure 4.18 shows the Corda app response, which correctly denies the request because the workflow checks the role in the user state of the user and checks if the action is allowed. As the user Charlie doesn't have permission, it returns an error message in the flowError with a generic message "An error occurred.".

This demonstrates that the Corda App's access control mechanisms are functioning as expected, preventing unauthorized users from making specific requests to the global state. This is essential for maintaining the integrity of the system and ensuring that only authorized users can perform specific workflows.

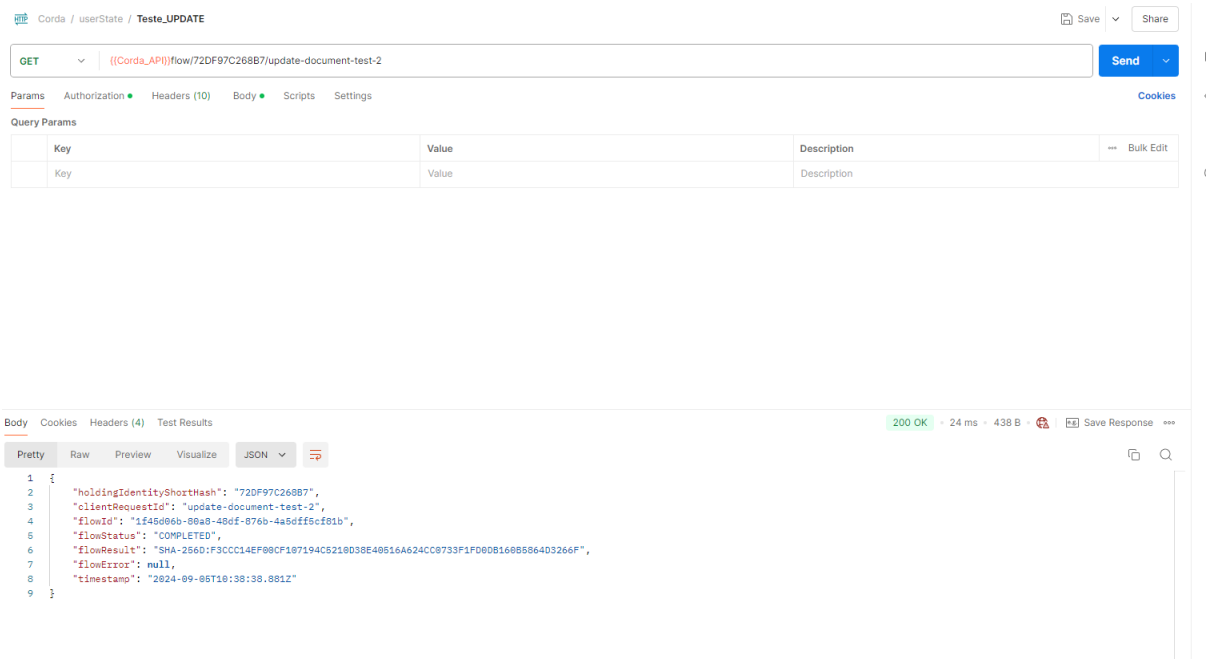


Figure 4.19: Successful response to a request made by Alice.

The figure 4.19 showcases the successful response to a request made by Alice, who holds the role of Medical Authority. The system correctly processes Alice's request, acknowledging her permission to execute the action by checking the user's role in the virtual node and concluding that the action has sufficient permission to be executed and, after executing the workflow, returns the internal Corda id of the transaction.

This response demonstrates the Corda App's ability to validate and authorize actions based on the user's assigned role, ensuring that only authorized users, like Alice, can perform specific workflows.

4.5 Summary

The performance tests successfully demonstrated the Corda applications ability to handle unauthorized access attempts and manipulation. Through various simulated scenarios, the system consistently detected and rejected these actions. Additionally, the tests confirmed that the application responded correctly to valid requests, ensuring that authorized users were granted appropriate access and received accurate results.

Overall, the testing validated the system's ability to assure privacy, data integrity, and security. These results confirm that the system is well-equipped to handle both operational demands and potential security threats, meeting the goals set for the evaluation.

By reviewing these sections, it becomes evident that the project has successfully translated initial proposals into comprehensive implementations across all specified requirements. The thorough testing and analysis have confirmed that the application maintains a system where it ensures privacy, data integrity, and security through the use of workflows that validate and secure the restrictions that were implemented.

Chapter 5

Conclusion

This chapter provides a comprehensive summary of the project, detailing the achievements, evaluation, areas for improvement, and future prospects. The implementation of the decentralized application using Corda blockchain has yielded significant insights and advancements in the realm of data management. This project highlights the potential of blockchain technology to address issues of privacy, data integrity, and transaction validation in the healthcare sector. The following sections will elaborate on these aspects, outlining the accomplishments, evaluations, and suggestions for future work.s

5.1 Discussion of Results

The development and implementation of a decentralized application for the management, storage, and signature of consent forms using Corda blockchain have demonstrated significant advancements in enhancing privacy and data integrity for the healthcare sector. The use of this blockchain technology, with its inherent characteristics of immutability, transparency, and security, addresses critical issues related to data management and transaction validation with the added benefit of provided privacy. By leveraging smart contracts and the RAFT consensus, the application ensures that each transaction, whether it involves the creation, alteration, or deletion of the documents state, is meticulously validated, maintaining a comprehensive and tamper-proof history of all changes.

The private network framework provided by Corda has proven to be robust and efficient, meeting both functional and non-functional requirements. The evaluation of the system through rigorous testing has shown that it performs well in key metrics, including security, reliability, and scalability. This project not only aligns with the current state-of-the-art technologies but also sets a precedent for future applications in various sectors that require stringent data management and integrity standards.

5.2 Limitations and future work

While there are many positive aspects, certain areas could benefit from further improvement. For instance, the user interface could be enhanced to provide a more intuitive and seamless experience for both medical personnel and patients. Additionally, the system's integration with existing healthcare IT infrastructure needs refinement to ensure smoother interoperable communication and data exchange with other systems that can guarantee that changes made to other systems can accuracy be considered and acted upon by the CordApp. Scalability also remains a critical area for ongoing development, particularly as the volume of participants and transactions increases.

Ultimately, this project exemplifies how decentralized models can revolutionize data management practices, offering a promising avenue for further innovation. The successful deployment of this blockchain-based application in healthcare can inspire similar implementations in other fields, fostering a more secure and transparent digital environment. The tests and analyses underscore the potential and practicality of decentralized applications in real-world scenarios, paving the way for broader adoption and refinement of these technologies.

References

AAVE (2024). Aave. Accessed: 26 August 2024. 11

ALI, M., DOLUI, K. & ANTONELLI, F. (2022). A decentralized and secure privacy-preserving framework for healthcare data using blockchain and role-based access control. *IEEE Transactions on Industrial Informatics*, **18**, 1234–1242, accessed: 2024-06-01. 8

AMPEL, B., PATTON, M. & CHEN, H. (2019a). Performance modeling of hyperledger sawtooth blockchain. In *2019 IEEE International Conference on Intelligence and Security Informatics (ISI)*, 59–61. 12

AMPEL, B., PATTON, M.W. & CHEN, H. (2019b). Performance modeling of hyperledger sawtooth blockchain. *2019 IEEE International Conference on Intelligence and Security Informatics (ISI)*, 59–61. 12

ANTHONY, B. (2023). Deployment of distributed ledger and decentralized technology for transition to smart industries. 16

APACHE KAFKA (2024). Apache kafka. `http://kafka.apache.org`, accessed: 2024-06-01. 13

ARORA, A. (2022). Corda 5: Develop, deploy, and dominate. `https://medium.com/@asharora78/corda-5-develop-deploy-and-dominate-2b433910e26d`, accessed: 2024-06-01. 30

ASHRAF, M. & HEAVEY, C. (2022). A prototype of supply chain traceability using solana as blockchain and iot. *Procedia Computer Science*, **217**, 1. 2

AZARIA, A., EKBLAW, A., VIEIRA, T. & LIPPMAN, A. (2016). Medrec: Using blockchain for medical data access and permission management. In *2016 2nd International Conference on Open and Big Data (OBD)*, 25–30. 10

BASKAR, S. & GOPIRAJAN, P. (2023). Application of blockchain in digital healthcare. In *2023 International Conference on Intelligent and Innovative Technologies in Computing, Electrical and Electronics (IITCEE)*, 591–595. 1

BOURHIS, P., REUTTER, J.L. & VRGOČ, D. (2020). Json: Data model and query languages. *Information Systems*, **89**, 101478. 29

BROTSIS, S., KOLOKOTRONIS, N., LIMNIOTIS, K., BENDIAB, G. & SHIAELES, S.N. (2020). On the security and privacy of hyperledger fabric: Challenges and open issues. *2020 IEEE World Congress on Services (SERVICES)*, 197–204. 12

BURNS, B., BEDA, J. & HIGHTOWER, K. (2022). *Kubernetes Up and Running: Dive into the Future of Infrastructure*. O'Reilly Media, 3rd edn. 3

BYME (2024). Byme page. `https://www.byme.pt/index.html`, accessed: 2024-06-01. 20

CAO, B., WANG, X., ZHANG, W., SONG, H. & LV, Z. (2020). A many-objective optimization model of industrial internet of things based on private blockchain. *IEEE Network*, **34**, 78–83. 7

CENTOBELLI, P., CERCHIONE, R., ESPOSITO, E. & OROPALLO, E. (2021). Surfing blockchain wave, or drowning? shaping the future of distributed ledgers and decentralized technologies. *Technological Forecasting and Social Change*, **165**, 120463. 1

CHANDRASEKARAN, K.S., MAHALAKSHMI, V. & ANANTHAPADMANABHAN, M.R. (2024). Exploring the proof of work based block chain algorithm for pharmaceutical industry supply chain innovation. *Journal of Advanced Research in Applied Sciences and Engineering Technology*, **35**. 7

CHEN, W., XU, Z., SHI, S., ZHAO, Y. & ZHAO, J. (2018). A survey of blockchain applications in different domains. In *Proceedings of the 2018 International Conference on Blockchain Technology and Application*, ICBTA '18, 17–21, Association for Computing Machinery, New York, NY, USA. 2

CHOWDHARY, A., AGRAWAL, S. & RUDRA, B. (2021). Blockchain based framework for student identity and educational certificate verification. In *2021 Second International Conference on Electronics and Sustainable Communication Systems (ICESC)*, 916–921. 8

CONSENSYS (2024). Quorum github development implementation. `https://github.com/Consensys/quorum`, accessed: 2024-09-12. 10

DHULAVVAGOL, P.M., BHAJANTRI, V.H. & TOTAD, S.G. (2020). Blockchain ethereum clients performance analysis considering e-voting application. *Procedia Computer Science*, **167**, 2506–2515, international Conference on Computational Intelligence and Data Science. 10

DIGITALSIGN (2024). Signingdesk. `https://www.signingdesk.com/home`, accessed: 2024-04-10. 2

DOCKER (2024). Docker documentation. https://docs.docker.com/, accessed: 2024-06-17. 30

DONG, Y., LI, Y., CHENG, Y. & YU, D. (2024). Redactable consortium blockchain with access control: Leveraging chameleon hash and multi-authority attribute-based encryption. *High-Confidence Computing*, **4**. 9

E-SIGNATURE SOLUTIONS, S. (2024). Signotec pads. https://en.signotec.com/solutions/electronic-signature/signature-pads/, accessed: 2024-04-10. 2

ELETTER, S.F., ELREFAE, G.A., YASMIN, T., QASEM, A., ALSHEHADEH, A.R. & BELARBI, A. (2022). Leveraging blockchain-based smart contracts in the management of supply chain: Evidence from carrefour uae. In *2022 International Arab Conference on Information Technology (ACIT)*, 1–5. 10

ELSEVIER (2024). Science direct webpage. https://www.sciencedirect.com/, accessed: 2024-02-10. 6

FEKIH, R.B. & LAHAMI, M. (2020). *Application of Blockchain Technology in Healthcare: A Comprehensive Study*, vol. 12157. Springer International Publishing. 2

FIELDING, R.T. & TAYLOR, R.N. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Ph.D. thesis, University of California, Irvine, aAI9980887. 17

FOR COMPUTING MACHINERY, A. (2024). Acm digital library webpage. https://dl.acm.org/, accessed: 2024-02-10. 6

FOUNDATION, E. (2023). go-ethereum: Official go implementation of the ethereum protocol. https://github.com/ethereum/go-ethereum, accessed: 2024-09-16. 10

FOUNDATION, E. (2024a). Geth: Getting started. https://geth.ethereum.org/docs/getting-started, accessed: 2024-08-14. 9

FOUNDATION, E. (2024b). Geth: Node architecture. https://geth.ethereum.org/docs/fundamentals/node-architecture, accessed: 2024-08-14. 10

GAJIĆ, D.B., PETROVIC, V.B., HORVAT, N., DRAGAN, D., STANISAVLJEVIĆ, A.M., KATIĆ, V. & POPOVIC, J. (2022). A distributed ledger-based automated marketplace for the decentralized trading of renewable energy in smart grids. *Energies*. 13

GALLA, L.K., KOGANTI, V.S. & NUTHALAPATI, N. (2016). Implementation of rsa. In *2016 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, 81–87. 38

GANS, J. (2023). *Proof of Work Versus Proof of Stake*, 69–83. Springer International Publishing, Cham. 7

GOOGLE (2024a). Angular 15 documentation. https://v15.angular.io/docs, accessed: 2024-04-1. 20

GOOGLE (2024b). Google scholar webpage. https://scholar.google.com/, accessed: 2024-02-10. 6

GOVERNMENT, P. (2024). Authenticator app. https://www.autenticacao.gov.pt/, accessed: 2024-04-10. 2

GREENSPAN (2024). Enterprise blockchain platform page. https://www.multichain.com/blog/, accessed: 2024-06-01. 12

HALEEM, A., JAVAID, M., SINGH, R.P., SUMAN, R. & RAB, S. (2021). Blockchain technology applications in healthcare: An overview. *International Journal of Intelligent Networks*, **2**, 130–139. 2

HARRIS, C.G. (2019). The risks and challenges of implementing ethereum smart contracts. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 104–107. 10

HEINONEN, H.T. (2021). On creation of a stablecoin based on the morini's scheme of inv'i&'sav wallets and antimoney. In *2021 IEEE International Conference on Blockchain (Blockchain)*, 409–416. 10

HENNINGSEN, S.A., TEUNIS, D., FLORIAN, M. & SCHEUERMANN, B. (2019). Eclipsing ethereum peers with false friends. *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS'I&'PW)*, 300–309. 10

HYPERLEDGER (2024). Hyperledger. http://www.hyperledger.org, accessed: 2024-06-01. 8, 12

HYPERLEDGER (2024). Hyperledger sawtooth core. https://github.com/hyperledger-archives/sawtooth-core, accessed: 2024-08-14. 12

INAYATULLOH, HARTONO, I.K. & KUSUMASTUTI, D.L. (2023). The blockchain conceptual model to prevents fraud and increase transparency in the presidential election in indonesia. In *2023 IEEE 9th International Conference on Computing, Engineering and Design (ICCED)*, 1–6. 8

INTERNATIONAL TELECOMMUNICATION UNION (2019). Information technology - open systems interconnection - the directory: Overview of concepts, models and services. ITU-T X.500 Series Recommendations, accessed: 2024-06-01. 35

ISMAIL, L., HAMEED, H., AISHAMSI, M., AIHAMMADI, M. & AIDHANHANI, N. (2019). Towards a blockchain deployment at uae university: Performance evaluation and blockchain taxonomy. In *Proceedings of the 2019 International Conference on Blockchain Technology*, vol. Part F148153, 30–38, Association for Computing Machinery. 7

ISMAIL, S., REZA, H., ZADEH, H.K. & VASEFI, F. (2023). A blockchain-based iot security solution using multichain. In *2023 IEEE 13th Annual Computing and Communication Workshop and Conference (CCWC)*, 1105–1111. 12

J, A., ISRAVEL, D.P., SAGAYAM, K.M., BHUSHAN, B., SEI, Y. & EUNICE, J. (2023). Blockchain for healthcare systems: Architecture, security challenges, trends and future directions. 7, 9

JAFAR, U., AZIZ, M.J.A. & SHUKUR, Z. (2021). Blockchain for electronic voting system—review and open research challenges. *Sensors*, **21**. 9, 10

JAYA, R.M., RAKKHITTA, V.D., SEMBIRING, P., EDBERT, I.S. & SUHARTONO, D. (2022). Blockchain applications in drug data records. *Procedia Computer Science*, **216**, 739–748. 2

JI, H., JIAN, J., YU, H., JI, J., WEI, M., ZHANG, X., LI, P., YAN, J. & WANG, C. (2022). Peer-to-peer electricity trading of interconnected flexible distribution networks based on distributed ledger. *IEEE Transactions on Industrial Informatics*, **18**, 5949–5960. 11

JONES, M.B., BRADLEY, J. & SAKIMURA, N. (2015). Json web token (jwt). Internet Requests for Comments, accessed: 2024-06-01. 36

KAUR, A. & GEETA (2023). Revolutionizing nanocommunication networks: A review of bio-inspired synchronization techniques. In *2023 2nd International Conference on Computational Modelling, Simulation and Optimization (ICCMSO)*, 89–94. 10

KESHTA, I. & ODEH, A. (2021). Security and privacy of electronic health records: Concerns and challenges. *Egyptian Informatics Journal*, **22**, 177–183. 2

KUDZIN, A., TOYODA, K., TAKAYAMA, S. & ISHIGAME, A. (2022). Scaling ethereum 2.0s cross-shard transactions with refined data structures. *Cryptography*, **6**. 9

KUO, T.T., ZAVALETA ROJAS, H. & OHNO-MACHADO, L. (2019). Comparison of blockchain platforms: a systematic review and healthcare examples. *Journal of the American Medical Informatics Association*, **26**, 462–478. 9, 12

KUO, Y.J.J. & SHIEH, J.C. (2020). Cross-domain design of blockchain smart contract for library and healthcare privacy. In *Proceedings of the 4th International Conference on Medical and Health Informatics*, ICMHI '20, 122–126, Association for Computing Machinery, New York, NY, USA. 2

LANGALIYA, V. & GOHIL, J.A. (2023). Innovative and secure decentralized approach to process real estate transactions by utilizing private blockchain. *Discover Internet of Things*, **3**. 8

MADUMIDHA, S., RANJANI, P.S., VARSINEE, S.S. & SUNDARI, P. (2019). Transparency and traceability: In food supply chain system using blockchain technology with internet of things. In *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, 983–987. 8

MANSOUR, S. & KAMAL, N.F. (2024). Decentralized private peer-to-peer energy trading using public blockchains. In *2024 4th International Conference on Smart Grid and Renewable Energy (SGRE)*, 1–6. 8

MATIC NETWORK (2024). Matic network whitepaper. Accessed: 26 August 2024. 11

MICROSOFT (2024). c sharp documentation. `https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-8.0`, accessed: 2024-04-1. 20

MITA, M., ITO, K., OHSAWA, S. & TANAKA, H. (2019). *What is Stablecoin?: A Survey on Price Stabilization Mechanisms for Decentralized Payment Systems*. Institute of Electrical and Electronics Engineers Inc. 10

MONRAT, A.A., SCHELÉN, O. & ANDERSSON, K. (2019). A survey of blockchain from the perspectives of applications, challenges, and opportunities. *IEEE Access*, **7**, 117134–117151. 7

MONRAT, A.A., SCHELÉN, O. & ANDERSSON, K. (2020). Performance evaluation of permissioned blockchain platforms. In *2020 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)*, 1–8. 7

MOSCHOU, K., THEODOULI, A., TERZI, S., VOTIS, K., TZOVARAS, D., KARAMITROS, D. & DIAMANTOPOULOS, S. (2020). Performance evaluation of different hyperledger sawtooth transaction processors for blockchain log storage with varying workloads. In *2020 IEEE International Conference on Blockchain (Blockchain)*, 476–481. 45

NGUYEN, C.T., HOANG, D.T., NGUYEN, D.N., NIYATO, D., NGUYEN, H.T. & DUTKIEWICZ, E. (2019). Proof-of-stake consensus mechanisms for future blockchain networks: Fundamentals, applications and opportunities. *IEEE Access*, **7**. 7

OF ELECTRICAL, I. & ENGINEERS, E. (2024). Ieeexplore webpage. `https://ieeexplore.ieee.org/Xplore/guesthome.jsp`, accessed: 2024-02-10. 6

OPENSEA (2024). Opensea. Accessed: 26 August 2024. 11

PENELOVA, M. (2021). Access control models. *Cybernetics and Information Technologies*, **21**. 8

PIERRO, G.A. & TONELLI, R. (2022). Can solana be the solution to the blockchain scalability problem? In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 1219–1226. 11

POLGE, J., ROBERT, J. & LE TRAON, Y. (2021). Permissioned blockchain frameworks in the industry: A comparison. *ICT Express*, **7**, 229–233. 10

PRADHAN, N.R., SINGH, A.P., SUDHA, S.V., REDDY, K.R.K. & ROY, D.S. (2023). Performance evaluation and cyberattack mitigation in a blockchain-enabled peer-to-peer energy trading framework. *Sensors (Basel, Switzerland)*, **23**. 13

QUICKSWAP (2024). Quickswap. Accessed: 26 August 2024. 11

QUORUM (2024). Quorum. Accessed: 2024-06-01. 8

R3 (2023a). Application networks in corda. Accessed: 2024-08-13. 17

R3 (2023b). Why corda? Accessed: 2024-08-13. 15

R3 (2024). Corda documentation page. `https://docs.r3.com/`, accessed: 2024-06-01. 12, 15

R3 (2024). Cordapps fundamentals. Accessed: 2024-08-13. 16

RENDUCHINTALA, T., ALFAURI, H., YANG, Z., PIETRO, R.D. & JAIN, R. (2022). A survey of blockchain applications in the fintech sector. *Journal of Open Innovation: Technology, Market, and Complexity*, **8**. 2

SAJAL K. DAS, K.K. & ZHANG, N. (2012). Introduction. *Handbook on Securing Cyber-Physical Critical Infrastructure*, 549–550. 3

SHEN, B., GUO, J. & YANG, Y. (2019). Medchain: Efficient healthcare data sharing via blockchain. *Applied Sciences*. 10

SHU, J., ZOU, X., JIA, X., ZHANG, W. & XIE, R. (2022). Blockchain-based decentralized public auditing for cloud storage. *IEEE Transactions on Cloud Computing*, **10**, 2366–2380. 8

SOLANA (2024). Solana. Accessed: 26 August 2024. 11

STEINBERG, D. (2023). Time100 most influential companies 2023. *TIME*, retrieved 17 March 2024. 11

STOUMPOS, A.I., KITSIOS, F. & TALIAS, M. (2023). Digital transformation in healthcare: Technology acceptance and its applications. *International Journal of Environmental Research and Public Health*, **20**. 1

SUJATHA, E., KUMAR, G.N., VIKAS, N.V. & REDDY, M.Y. (2023). Decentralized portfolio tracking application on a polygon layer 2 blockchain. In *2023 International Conference on Networking and Communications (ICNWC)*, 1–5. 11

TANDON, A., DHIR, A., ISLAM, N. & MÄNTYMÄKI, M. (2020). Blockchain in healthcare: A systematic literature review, synthesizing framework and future research agenda. *Computers in Industry*, **122**. 2

TAPSCOTT, D. & TAPSCOTT, A. (2018). *Blockchain Revolution: How the Technology Behind Bitcoin and Other Cryptocurrencies Is Changing the World*. Penguin. 1

TAYLOR, P., DARGAHI, T., DEHGHANTANHA, A., PARIZI, R. & CHOO, K.K. (2020). A systematic literature review of blockchain cybersecurity. *Digit. Commun. Networks*, **6**, 147–156. 2

TRAN, T.N.T., FELFERNIG, A. & LE, V.M. (2023). An overview of consensus models for group decision-making and group recommender systems. *User Modeling and User-Adapted Interaction*. 8

UCBAS, Y., ELEYAN, A., HAMMOUDEH, M. & ALOHALY, M. (2023). Performance and scalability analysis of ethereum and hyperledger fabric. *IEEE Access*, **11**. 9

UDDIN, M., ISLAM, S. & AL-NEMRAT, A. (2019). A dynamic access control model using authorising workflow and task-role-based access control. *IEEE Access*, **7**. 8

WICAKSANA, A. & WIRA, J.C. (2022). Security analysis of private blockchain implementation for digital diploma. *International Journal of Innovative Computing, Information and Control*, **18**. 8

WOOD, G. (2024). Ethereum: A secure decentralised generalised transaction ledger. Technical Report 705168a, Ethereum & Parity, Paris, version 705168a. 9

XU, J., WANG, S., ZHOU, A. & YANG, F. (2020). Edgence: A blockchain-enabled edge-computing platform for intelligent iot-based dapps. *China Communications*, **17**, 78–87. 11

YAP, K.Y., CHIN, H.H. & KLEMEŠ, J.J. (2023). Blockchain technology for distributed generation: A review of current development, challenges and future prospect. *Renewable and Sustainable Energy Reviews*, **175**, 113170. 16

YAQOOB, I., SALAH, K., JAYARAMAN, R. & AL-HAMMADI, Y. (2021). Blockchain for healthcare data management: opportunities, challenges, and future recommendations. *Neural Computing and Applications*. 2

YEUNG, A., TORKAMANI, A., BUTTE, A., GLICKSBERG, B.S., SCHULLER, B., RODRIGUEZ, B., TING, D.S.W., BATES, D., SCHADEN, E., PENG, H., WILLSCHKE, H., VAN DER LAAK, J., CAR, J., RAHIMI, K., CELI, L.A., BANACH, M., KLETEČKA-PULKER, M., KIMBERGER, O., EILS, R., ISLAM, S.M.S., WONG, S.T., WONG, T.Y., GAO, W., BRUNAK, S. & ATANASOV, A.G. (2023). The promise of digital healthcare technologies. *Frontiers in Public Health*, **11**. 1

ZEBA, S., SUMAN, P. & TYAGI, K. (2023). *Chapter 4 - Types of blockchain*. Academic Press. 7

ZHU, S., HU, H., LI, Y. & LI, W. (2019). Hybrid blockchain design for privacy preserving crowdsourcing platform. In *2019 IEEE International Conference on Blockchain (Blockchain)*, 26–33. 9

ZINEDDINE, A., CHAKIR, O., SADQI, Y., MALEH, Y., SINGH GABA, G., GURTOV, A. & DEV, K. (2024). A systematic review of cybersecurity assessment methods for https. *Computers and Electrical Engineering*, **115**, 109137. 18