UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

A Digital Vault Solution for Banking Institutions

Vladyslav Yultyyev

Mestrado em Segurança Informática

Trabalho de Projeto orientado por:
Prof. Doutor Mário João Barata Calha
Eng. João Miguel Pereira Rodrigues da Securibox

2019

Acknowledgments

I would like to begin by expressing my special thanks of gratitude to my advisors PhD. Mário João Barata Calha and Eng. João Miguel Pereira Rodrigues from Securibox. Their support and expertise was crucial to develop and complete this project.

Also, I would like to thank my family for all their kindness and for always supporting me unconditionally.

Thanks to all my Securibox colleagues who provided an outstanding working environment and were always responsive to my questions. I will miss you!

I thank my love Marta Reis. Thank you for all your support, care and patience. Your company was immensely important to me, especially in this journey.

A special thanks to Nuno Tomás, for being a good friend and for always being there with his wise advices.

Thank you all!

Resumo

Este projeto surgiu no âmbito da necessidade que a empresa Securibox tem em fornecer um produto de armazenamento seguro compatível com o funcionamento na nuvem, para as instituições bancárias que operam no mercado francês.

Com o aparecimento da banca on-line e o intuito de atrair mais clientes, as instituições bancárias começaram a oferecer serviços que vão para além dos serviços convencionais deste setor. Muitas vezes esses serviços tratam ou armazenam dados sensíveis dos seus clientes e podem até incluir informação e documentos pessoais dos utilizadores que estão hospedados noutras entidades, tais como faturas eletrónicas, transações bancárias de outras instituições financeiras e recibos de vencimento. No entanto, sempre que for necessário armazenar informação dos clientes, este processo tem de respeitar um conjunto de boas práticas e normas do país onde a instituição opera, utilizando para o efeito um cofre digital. No caso do mercado francês, existem poucas soluções que satisfazem, parcialmente ou totalmente, as normas e a legislação respeitante aos cofres digitais e que sejam tecnicamente eficientes e competitivas.

O objetivo deste trabalho visou desenvolver uma versão inicial de uma solução que colmata a necessidade atual do mercado bancário francês relativo à área de armazenamento e manuseamento inteligente de dados.

Para satisfazer as normas da União Europeia e da França em particular, é necessário armazenar os ficheiros de forma cifrada, registar o seu formato, como, quando e por quem estes formas acedidos e os seus meta-dados de modo a garantir a sua preservação mesmo após a eliminação dos mesmos. Este desafio foi resolvido, e para se destacar das soluções atualmente existentes, foi construída a base para no futuro integrar esta solução com o serviço *Securibox ParseXtract*, que tem a capacidade de analisar e extrair informação importante do conteúdo dos documentos, de uma forma estruturada e precisa, recorrendo a aprendizagem automática.

Para o armazenamento dos documentos a solução adotada foi o *OpenStack Swift* – um software de código aberto, compatível com nuvens pública e privadas.

Uma vez que os documentos podem ser eliminados do sistema pelo utilizador, é necessário a existência de uma plataforma, separada do *OpenStack*, para armazenar os dados relativos aos meta-dados dos documentos e acessos ao sistema. A solução encontrada para o armazenamento destes dados, consiste no seu registo, através de *logs*, numa base de dados não relacional – o *MongoDB*, que é compatível com tecnologias em nuvem e é eficiente com grandes volumes de dados.

Para realizar a comunicação entre os vários componentes do cofre digital, foi criado um serviço que oferece uma *REST API*, o núcleo da solução. Nesta camada, os documentos são cifrados garantindo também a integridade, confidencialidade e o não-repúdio dos dados.

Por último, um servidor *Web* que comunica com a *REST API* foi criado para demonstrar todas as funcionalidades do cofre digital.

As principais vantagens desta solução consistem na utilização de tecnologias código aberto, na compatibilidade com o funcionamento na nuvem, na escalabilidade de todas as suas camadas, tais como o armazenamento de dados, *logs* e serviço *web* API, e numa melhor integração com outros produtos da Securibox, que deste modo reduzem o custo da solução para o cliente final.

Do ponto de vista conceptual, esta solução pode ser utilizada não apenas pelo sector bancário, mas também por qualquer outra área empresarial onde é necessário armazenar grandes volumes de dados em nuvem privada e pública em simultâneo, tendo como base uma solução facilmente escalável e onde todas as ações dos seus utilizadores são rastreáveis em conformidade com a legislação.

Palavras-chave: Caixa forte digital, banca online, armazenamento seguro em nuvem, extração automática de dados, privacidade.

Abstract

This project is a result of the Securibox need to provide a digital vault storage solution for some of their bank clients, operating in the French market.

Since electronic banking has emerged, banking institutions began to provide online services that go beyond conventional bank services to attract more users. Sometimes those services involve operations with personal data of their customers which can include data and documents from other services, entities and companies. All this information must be stored on the banking institution side, using a secure digital vault storage, while respecting the legislation of the country where the institution is located.

The goal of this work was to develop an initial solution, that would address the current needs of the French banking market, regarding intelligent data handling and storage.

To be compliant with the European Union and the French legislation it is necessary to ensure the security and the privacy of the costumers documents and data. To address those requirements a REST API solution was developed using .Net technology. This solution is divided in 3 layers. The document storage layer, the metadata and log storage layer and the core layer. The documents are encrypted and stored at the OpenStack Swift environment, while metadata is stored at the MongoDB database as journal log entries. The information processing and the communication between OpenStack and MongoDB occurs at the core layer.

This solution relies on open-source technologies, is easily scalable and compatible with other Securibox products. Conceptually it can be used, not only by banking institutions, but also by any organization or company that have to store and deal with large amounts of information.

Keywords: Digital vault, electronic banking, cloud storage, automatic data extraction, privacy.

Contents

List of Figures

List of Tables

List of Abbreviations

ACL – Access Control List

AFNOR – “Association Française de Normalisation”, translated as “French Standardization Association”

API – Application Programming Interface

CAPTCHA – Completely Automated Public Turing test to tell Computers and Humans Apart

CNIL – “Commission nationale de l'informatique et des libertés”, translated as “National Commission on Informatics and Liberty”

CRUD – Create, Read, Update, and Delete

EU – European Union

GDPR – General Data Protection Regulation

GUI – Graphical user interface

HDD – Hard Disk Drive

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

IEC – International Electrotechnical Commission

ISO – International Organization for Standardization

JSON – JavaScript Object Notation

MVC – Model-View-Controller

NF – “Norme français”, translated as “French Standard”

NoSQL – Not Only SQL

OAIS – Open Archival Information System

OWASP – Open Web Application Security Project

PDF – Portable Document Format

PEA – “Preuve, Echange, Archivage”, translated as “Proof, Exchange, Archiving”

RAM – Random Access Memory

REST – Representational State Transfer

SDK – Software Development Kit

SQL – Structured Query Language

USA – United States of America

UTC – Coordinated Universal Time

VPN – Virtual Private Network

WORM – Write Once Read Many

YAML – YAML Ain't Markup Language

Chapter 1 Introduction

A bank is a financial organization where the core business consists in offering its customers services and tools to manage their economic assets. Since the 1980s, the financial sector began shifting towards the digital technologies and with the spreading of the Internet the online banking, also known as home banking, emerged and changed the patterns of banking use.

The online banking gives customers the possibility to execute financial operation remotely (e.g.: transfer money from one account to another), without going to a bank branch. As a result, the need for physical channels has decreased, and the demand for online banking products and services has increased. [25]

To attract more customers, the banks began to offer new services and products, through the online banking systems, that goes beyond traditional financial services. One of these services consists in giving to the customers the ability to visualize all their bank accounts information, including bank accounts of other banks, such as transactions and account balance, at a single online banking service (e.g.: a bank website or a mobile application). Besides that, some banks are also offering the possibility to collect and store other documents of the customer, such as telecommunication or e-commerce invoices, payment notices, medical care documents, etc. Some banking institutions allow their customers to upload their own documents directly to their online banking system account. By aggregating all this information in one place, the customers can benefit from a better overview of their financial assets and operations. [20]

Regarding technical aspects, the service described above relies on a Web scrapper that automates the human interactions with a website by solving CAPTCHAs, providing credentials to a login form and collect specific information and files from it. Due to the complexity of developing a custom Web scrapper technology, often banks contract this service from third-party firms, such as Securibox.

Currently, the Securibox company is working with main leaders of banking companies in France by providing them a Web scrapper technology that besides the standard scrapping processes is also capable of extracting useful information from the documents itself.

After collecting costumer's transactions and documents, the bank is responsible for the privacy and safety of this stored data in accordance with the European Union and French law. [5]

Stored data can be generated by digital/computer systems or processes although it may also result from digitalization process of physical sources that rely on paper, audiotape, videotape, medical radiography and any other analogic media support that can be digitalized such as images, documents, audio, audio-visual content or biometric information. [33]

Due to the lack of solutions concerning secure storage products for French banking organizations, many of them had asked Securibox for a storage solution that could meet all legal and functional requirements – a digital vault.

The main reason why French banks cannot use already existing solutions used by other countries for that kind of problem, such as the USA, is because they not comply with all European legislation requirements [18]. Therefore banks may opt for a new solution from scratch or use a costume solution adopted from an open-source project such as Maarch [26], which does not support cloud storage and is outdated. Many banks choose to buy already existing proprietary solutions since it may require a lot of effort and costs to build a new or modify an open-source solution to make it compatible with all French legislation.

By analysing the target market in France regarding digital vault solutions for banking systems, it was possible to identify the most used product nowadays, which belongs to a French company named Cecurity [13]. Their product "PEA" has all required certifications and was built according to French standards and legislation. However, this solution is economically expensive, it is not based on open-source technologies and may involve extra effort to integrate services of data aggregation and information extraction.

The overall economic cost of the solution can be decreased by implementing open-source technologies. Besides that, it is possible to implement this solution in private and public clouds, since each module is scalable and cloud-friendly. In terms of future work and to attract more clients, this digital vault solution could be integrated with other Securibox services for data collection (Web scrappers) and document information extraction (ParseXtract – a machine-learning based solution).

1.1 Objectives

The main goal of this project is to develop the first version of a digital vault storage solution that can be used as a basis for a new Securibox commercial product for French bank organizations. This solution must be easily integrated with other Securibox services and respect all French and EU legal and technical requirements for digital storing solutions.

As a first version of the digital vault solution, it is expected that all core functionalities of a storage system will be implemented. Namely, the ability to scale any solution component, store large amounts of binary files (e.g.: documents) using private and public clouds and to have an auditable journal which would log any user's or solution's interaction with the system and its personal information.

To ensure user's data privacy and security, the binary files stored at the cloud should be encrypted using a strong encryption algorithm, while the safety of all encryption keys should also be guaranteed.

As an extra feature for this project, the ability to share documents between different users could be implemented. This should be done in a secure way, respecting all access control requirements and privilege restrictions.

To fit into Securibox development environment, the project core should be built using .Net technologies and some of the Securibox internal code libraries. Therefore, one of the goals is to develop a .Net C# SDK (Software development kit) for OpenStack Swift.

The solution should be suitable for different infrastructure environments and easily integrated with other existing Securibox services. For these reasons, the solution service should be offered as a REST (Representational State Transfer) API (Application programming interface).

1.2 Contributions

In this project, some of the contributions are especially worth to mention. Namely, the design of a document storage solution following security and privacy patterns, the vault's sharing feature and the .Net OpenStack Swift SDK.

During the design process of the storage component, different approaches were considered. By analyzing specific requirements and the environment of this project, a set of specific measures regarding encryption and loggings methods were adopted for each of the storing and processing modules.

The sharing feature, an extra implemented functionality (not imposed by Securibox nor any other entity), strictly follows all legal and technical requirements as the rest of the project which means that it is still possible to ensure privacy and safety of all its users and successfully pass an audit. This feature could be used in many different contexts, such as a file-sharing between different services or accounts, and it can be seen as a competitive advantage over other existing solutions.

The .Net OpenStack Swift SDK, which didn't exist before (at least for the recent versions of the OpenStack), was written from scratch by analyzing OpenStack documentation, especially the chapters regarding the API. To guarantee that this solution was working correctly, a set of unit tests were also written and executed.

As a result of this development, the Securibox developers can now use OpenStack Swift or SwiftStack on their .Net projects by simply importing this SDK into their solution.

1.3 Securibox

Securibox is a Fintech company whose mission is to enable people to have a clear picture of their data, by providing cloud-based technologies with tools to analyse, organize & visualize what really matters. [40]

One of the most successful Securibox' products is the "Securibox Cloud Agents". It is a web scrapping, API based, solution that is able to collect useful information and documents from more than 500 web services. To collect this data the scrapper has the ability to login into each of these services, bypass robot blocking technologies such as CAPTCHA, find and aggregate the data in a structured way. A free of charge web service that uses this solution is available at https://cube.securibox.eu (only for personal use).

Another important Securibox service is ParseXtract. This is a machine-learning based solution, which is able to automatically recognize a PDF or an image document by its pattern and extract its data. The machine learning algorithms are trained with known documents in order to achieve an acceptable accuracy level for the recognition and information extraction (about 95% success rate). All extracted information is retrieved in a structured way using JSON (JavaScript Object Notation). [39]

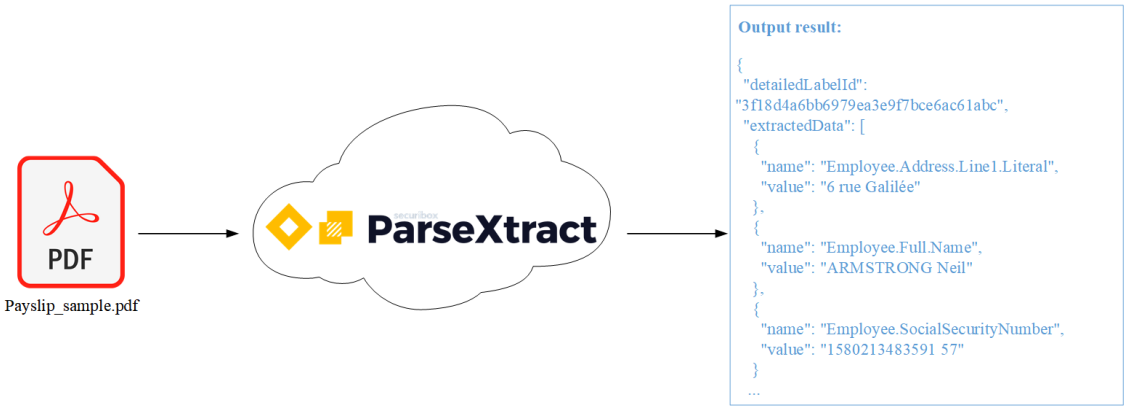

Figure 1.1: Securibox ParseXtract service example

The ParseXtract service will be used alongside the digital vault solution to extract information from PDF files' content before storing them, once this solution reaches the market. This feature can also be seen as a competitive advantage in a digital vault marketplace.

1.4 Work plan execution

In order to achieve the proposed goals in a structured and clear way, a work plan was established. This work plan consisted of 5 main steps described below:

1. Analysis and study of French and European standards relative to digital vault systems. (7 weeks)
2. Elaboration of solution's technical specifications. (4 weeks)
3. Solution's development (16 Weeks)
4. Solution's testing (functional and security) (4 Weeks)
5. Implementation of a simple proof of concept project (4 Weeks)

Overall, the work plan was followed fairly close. A notable exception, which took approximately more 8 weeks than initially expected, was the implementation of some of the core functionalities of the system (step 3.), such as the OpenStack Swift SDK, data encryption methods and the Web client application.

1.5 Document structure

This document is structured in 5 chapters. Each chapter will approach a specific stage of the project starting from the concept idea, main goals, legal and technical environment up to the conclusions and future work.

In Chapter 2, the documentation and implementation obligations for third-parties' entities that are going to use this solution are discussed. The most relevant software, methodologies and frameworks used during this project are introduced and briefly described. Also, some of the already existing and conceptually proposed solutions are discussed and compared.

In Chapter 3, system requirements and use cases are presented. Also, the system architecture is explained by clarifying the adopted design choices.

In Chapter 4, the implementation process is described in detail alongside functional, security and validation test results. Possible solutions are provided for the unexpected problems that appeared during the elaboration of this project.

Finally, in Chapter 5, the achievements of this project are analysed and the final considerations about possible future development are presented.

Chapter 2 Related work

This chapter begins with the description of the most important and relevant legal and technical standards applied to the French market and related, but not exclusively limited to the security and privacy of digital vault storage. Then, a brief analysis and overview of already existing solutions offered by Securibox market competitors are presented.

To conclude this chapter, some of the most pertinent frameworks and software tools used in the development of this project are described as well as the environment in which this solution is intended to be working and was implemented.

2.1 Standards

The digital vault solution will be deployed on third parties' entities, such as banks. To guarantee the correct functioning of the software/solution, not only the solution, but also those entities must respect some ISO and AFNOR standards and legislation as well. The AFNOR (*Association Française de Normalisation*), a member body of the ISO (International Organization for Standardization), is the French entity responsible for standardization, including the digital information domain.

During this project, the standard NF Z42-013 was used. However, there is also available its English version equivalent as ISO 14641-1 (Specifications concerning the design and the operation of an information system for electronic information preservation). [32]

In a simplified overview, the standard NF Z42-013 and NF Z42-020 (an extension of the NF Z42-013) are directly related with the development and implementation of a digital vault system, while other standards applied to this project do not focus at any specific system or solution. However, those standards are related to security and privacy of the specific vault's components or mechanisms.

The NF Z42-013/20 standards focus on both technical and organizational aspects of the digital vault. Namely, the data handling and its integrity at different states, the log system minimum requirements, the mandatory system documentation, the minimum set of digital vault solution's functions and the auditability of the system.

The standards, such as ISO 19005-1 (Files format), ISO 8601 (Time representation), ISO 15489 (Records Management) and ISO 14721 (Open Archival Information System - OAIS) are mentioned as a reference at the NF42-013/20 documents and provide a higher level of detail over specific topics.

The concepts, ideas, obligations and restrictions derived from GDPR (General Data Protection Regulation), ISO 27001 (Information security management) and ISO 27002 (information security controls) are traversal not only for this particular project, but must also be respected and implemented (where it applies) at the solution's host organization.

2.1.1 Standard NF Z42-013 (AFNOR)

This standard [33] establishes a set of specifications that the digital vault system must obey to be certified in France regarding the technical and organizational aspects, and measures to be taken, while handling with digital documents, to guaranty their integrity and non-repudiation while adding, storing, deleting, consulting or transmitting a document.

The key points of the standard focus on:

- The system's storage policy.
- The format of the data that is being stored to ensure its compatibility with at least some of the open-source formats.
- How to guaranty the integrity of a document, e.g.: fingerprints, signatures, etc.
- Logs requirements, regarding timestamps, user actions, document manipulations, etc.
- Logs storage policy.
- How to manage and support the migration or conversion of the digital vaults data format over time.
- Mandatory topics for the system's documentation regarding its management, support, and technical aspects.
- Auditability of the system.

It is important to mention that due to the complex process of the data format conversion and the existence of relatively cheap solutions to perform this task in a secure way, for example by Microsoft or LibreOffice, this feature will be out of the scope of the current project and hence not implemented.

2.1.2 Standard NF Z42-020 (AFNOR)

This standard can be seen as an extension for the NF Z42-013 that defines with a high level of detail the minimum functions of a digital-vault component and how they must be implemented to guarantee the integrity of the data over time. A component can be software running on a single or multiple hardware platforms or software which is embedded within hardware and behaves as a whole. The data of the system are objects, where each of them have a unique identifier and a hash, can be a file, a set of files compressed into a single file, such as zip file, a file with an electronic signature, a file with metadata or an encrypted file. [34]

Also, the digital vaults functional requirements are specified, namely, the functions call on the system. The minimum list of functions that should be available to the system is: add, read and delete object, read technical metadata (file format, size, etc.), control (verify the integrity of the file), read logs associated with a specific object, list the identifiers of stored objects, count the total amount of the stored objects.

2.1.3 General Data Protection Regulation (GDPR)

It is a set of rules governing the privacy and security of personal data laid down by the European Commission which came into effect on May 25th, 2018. [21]

The GDPR establishes strict global privacy requirements governing how you manage and protect personal data in the European Union and of its citizens, regardless of the company's location while respecting individual choice — independently where data is sent, processed, or stored.

This regulation applies to the processing of personal data wholly or partly by automated means and to the processing other than by automated means of personal data which form part of a filing system or are intended to form part of a filing system.

Personal data can be any information related to a natural person or 'Data Subject', that can be used to directly or indirectly identify the person. It can be anything from a name, a photo, an email address, bank details, posts on social networking websites, medical information, or a computer IP address.

The digital vault project must be fully compliant with the GDPR since all the sensitive data that could be linked to a physical person is stored in a secure way and all mechanisms that may use user's personal data must be presented to the user in a clear form as they are described at this document.

2.2 Overview of existing solutions

2.2.1 Cecurity

The Cecurity company provides different products and solutions regarding digital storage for French market. For the scope of this project, the focus will rely on the PEA ("Preuve, Echange, Archivage", translated as "Proof, Exchange, Archiving") and CecurCrypt solutions.

The PEA is an AFNOR certified storage solution, for medium and large enterprises, compatible with heterogeneous storage managing platforms such as EMC Centera, IBM Tivoli Storage Manager, NetApp SnapLock, Hitachi Content Platform, among others. This solution is fully installed on the client-side and offers a GUI client, which gives the possibility to configure the system in a user-friendly way, has a sophisticated search system based on the document's metadata and is also able to convert files formats and to define its retention period.

The CecurCrypt is a web-based solution, available for small enterprise companies (starting from a single user account), which offers the possibility to store encrypted documents with an auditable log journal system at the client's or Cecurity's servers. All stored documents are digitally signed and stored in folders with restricted access. To grant access to a folder, a password is required (the password is different for each folder). [13]

Since PEA is a proprietary solution that requires sophisticated infrastructure to be installed, it was not possible to analyze it in production. However, the CecurCrypt allows the creation of a demo account, which was created at https://www.cecurity.com/fr/essai-cecurcrypt, and made possible deeper analyzes of this product.

To understand how secure the CecurCrypt application is, the traffic generated by the client (Web Browser) was captured using Fiddler by Telerik software and then analyzed. Also, some of the client's

source script files were downloaded and analyzed. Since those scripts are not obfuscated it was possible to understand the key concepts behind some of the core functionalities, such as the upload. The overall product apparently has a strong security design and algorithms.

The document upload process is one of the most important functionalities in any digital vault system and it is imperative to ensure files security and integrity at every step (from the sender until the recipient). Due to that fact, it was analyzed how CecurCrypt upload methods work. The file is encrypted at the client's side using *javascript* "randomly" generated values and then is sent to the server.

It is important to refer, that with this analysis it is not possible to guarantee that the presented mechanism works exactly the way it was described in this document since some parts of the solution are hosted at the web server-side and can have other logical triggers and implementation behind.

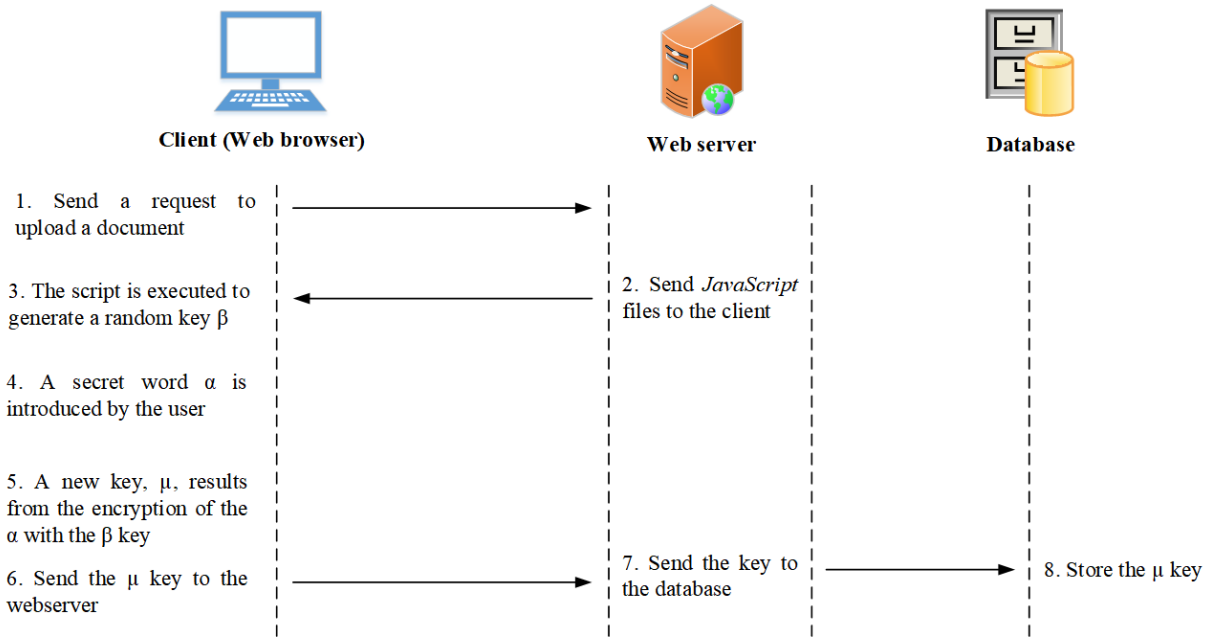

Figure 2.1: Cecurity document upload solution

Figure 2.1 illustrates what happens in the Cecurity solution when a file is sent to the digital-vault server. A temporary key generated on the webserver is sent to the client, in a secure way. Later, this key is used for encryption. After receiving the key, the file is split into several pieces (chunks). Those chunks are encrypted with a previously received random key (probably with the server public key as well) and then sent to the server one by one. Once the entire file is on the server it is encrypted with a key that is related in some way with the α key, and probably it is the μ key. A similar process occurs when a file is downloaded from the server, which makes the decryption possible only if someone knows the α key (a string).

It is also possible to set groups and share a key with other people, hence there has to be always an administrator for that vault, who is the only one who can change that key.

Overall, these solutions attend all essential market requirements for a digital vault system. However, since those are proprietary solutions, its implementation and maintenance cost could be high.

A solution that would be empowered by open source technologies could lower that cost. Also, this solution lacks the extraction capability like the Securibox ParseXtract service can provide.

2.2.2 MaarchRM

MaarchRM is an open-source solution that was developed in accordance with French standards, namely the NF Z42-013 and NF Z42-020, however it is not certified by AFNOR. The target market for this product are small companies that would use this solution as internal storage and document management system.

This solution offers all functions required by the NF Z42-020 and also provides a web interface that allows creating different user groups, with different roles and privileges. The demo version of the MaarchRM web service is available at https://demo.maarchrm.com/user/prompt.

Despite being a viable solution for a small organization, it is not an optimal solution for large scale organizations, since it relies on old technologies, such as PHP and Postgres SQL software, which originally were not designed to be scalable and work at the cloud infrastructure. Due to that fact, the setup and maintenance of this solution would result in extra non-trivial work and additional costs (e.g.: setup and configure a distributed SQL database; make the necessary modification to allow this solution to work on the cloud and ensure that it can pass the AFNOR certification process). Furthermore, the use of a SQL database for binary data storage can decrease the performance of the system in cases when there are millions of entries generated by thousands of users simultaneously.

2.3 Software frameworks

In this section, the frameworks that have been used in the project are briefly presented. Also, some additional software that could improve the current solution, such as SwiftStack, is presented as well.

The digital vault solution is a combination of multiple components, where each of them has its own role. Namely, a storage component and an information processing component.

The storage component has to be capable of storing binary files, such as documents, as well as the logs journal entries. Initially, a regular database was planned to be used for that purpose. However, due to the fact that this solution must handle large amounts of data and offer cloud support a NoSQL solution had to be adopted. The choice relied on the MongoDB since it was designed to work on top of cloud systems and is capable of handling large amounts of information with better performance than other similar database systems [2]. Since this component is serving two different proposes, a costumer's binary file storage, where the binary file's size could be too large for a database, and a journal's log storage, it was decided to divide it into two different modules. A module for the journal log system, using MongoDB, and a cloud-storage solution for the binary data.

For the binary data cloud storage, it was decided to use OpenStack Swift because it's Swift API is directly compatible with SwiftStack, which offers a highly automated environment for the infrastructure implementation and maintenance, and besides that, it already provides some additional features that can help to integrate the solution, for example, with Active Directory Services (Microsoft) among others. The SwiftStack is a well-known framework and technology for some of the Securibox

most influential clients, and therefore potential customers of it. Due to that fact, the choice relied on this technology. However, any other cloud solution with similar characteristics, such as cloud Stack, could be used instead.

Concerning the information processing component of the solution the .Net C# technology was chosen. In a technical perspective, this choice was made because by using .Net technology it is natively possible to export this project into the cloud (e.g.: Azure). In an organizational perspective, the Securibox has an experienced team of .Net developers who could offer any needed support during the development of this project and also guarantee this way its future development and maintenance.

2.3.1 OpenStack

OpenStack is a set of free and open-source software tools for building a large-scale virtualization environment and managing cloud computing platforms for public and private clouds capable of working in heterogeneous infrastructure [38]. This software controls large pools of computing, storage, and networking resources throughout a data centre and can be managed through a dashboard or via the OpenStack API, which is fully documented.

This solution is similar to Azure Cloud Services (Microsoft), Amazon Web Services, Google Cloud Platform, Rackspace Cloud, etc. Like any other cloud service solution, OpenStack can be used to create a scalable web service that will automatically add more computing power when needed, to work on complex problems using a distributed system.

The main components of the OpenStack are [38] :

- Nova – Computing engine, used for deploying and managing the instances to handle computing tasks.
- **Swift** – A storage system for objects and files. Rather than the traditional idea of a referring to files by their location on a disk drive, OpenStack gives each file or piece of information a unique identifier and decides where to store this information, making the scaling easy.
- Cinder – A block storage component, which works like a traditional storage system, where it is possible to access specific locations on a disk drive.
- Neutron – It provides the networking capability for OpenStack.
- Horizon – The dashboard, only graphical interface to OpenStack. Helps system administrator to look at what is going on in the cloud, and to manage it as needed.
- **Keystone** – It provides identity services for OpenStack. This service is responsible to grant permission to users or services to access the system, or part of it, according to their role, group, etc.
- Glance – It provides virtual-images services to OpenStack.
- Ceilometer – It provides telemetry services, by counting each user's usage of each component of the OpenStack cloud.
- Heat – Is the orchestration component, which allows storing the requirements of a cloud application in a file that defines what resources are necessary for an application.

2.3.1.1 Swift OpenStack

The OpenStack Object Store project [37], known as Swift, was built for scale and optimized for high durability, 99.99999999% of availability with 3 replicas (10-11 nines using the “nines” nomenclature), availability, and concurrency across the entire data set. It is used for redundant, scalable data storage using clusters of standardized servers to store large amounts (e.g.: petabytes) of accessible data. It is a long-term storage system for large amounts of static data which can be retrieved and updated. Object Storage distributed architecture with no central point of control, provide scalability, redundancy, and permanence. By writing objects into multiple hardware devices, the OpenStack software is responsible for ensuring data replication and integrity across the cluster. Storage clusters scale horizontally by adding new nodes. When a node fails, OpenStack works to replicate its content from other active nodes. Because OpenStack uses software logic to ensure data replication and distribution across different devices, inexpensive commodity hard drives and servers can be used instead of more expensive equipment. Swift is ideal for storing unstructured data that can grow without bound, since it provides a fully distributed API-accessible storage platform that can be integrated directly into applications.

For the purposes of this project, the architectural aspects of the Swift OpenStack will only focus on the accounts, containers, and objects.

An account/project/tenant represents the top-level of the hierarchy. The service provider (Administrator) creates the account for a user/service where the resources will be fully owned by him on that account. The account defines a namespace for containers. A container might have the same name on two different accounts and the number of containers per account is limitless.

A container defines a namespace for objects. Objects with the same name in two different containers represents 2 different objects. It is also possible to control access to objects, in a certain container, by using an access control list (ACL). However, it is not possible to store an ACL with individual objects. It is possible to set a storage policy on a container with predefined names and definitions from a specific cloud provider.

In order to delete a container, it is important to make sure that it does not contain any object, otherwise, it will be impossible to delete.

An object stores data. This data can be the content of a document, an image, and so on. It is also possible to store custom metadata with an object, container or account. The maximum size of the object can be configured and has no limit (theoretically). It is possible to compress files using content-encoding metadata, schedule objects for deletion, auto-extract archive files, generate a temporary URL that provides access to an object (via a GET request) and create symbolic links to other objects.

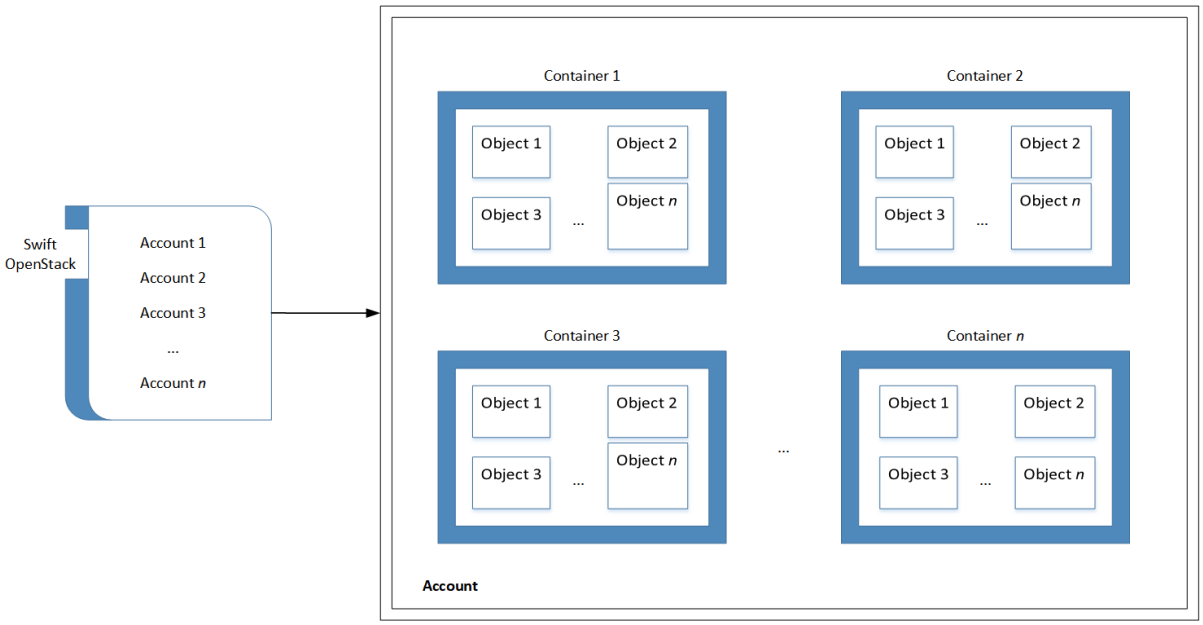

Figure 2.2: Basic Swift OpenStack scheme

Figure 2.2 illustrates what was described above about Swift OpenStack's components alongside its hierarchy and dependencies.

2.3.1.2 DevStack

The manual deployment of the OpenStack can be a very complex task. For that reason, a set of scripts that automatically install and set up a complete OpenStack **development** environment solution – DevStack – were developed by the authors of OpenStack. This solution is constantly updated and maintained by OpenStack at a public GitHub repository. It is available at https://git.openstack.org/cgit/openstack-dev/devstack.

A single Linux (Ubuntu Server 16.10) machine with DevStack software [9] was deployed and properly configured on a local Securibox server to perform any necessary tests with Swift OpenStack while developing the digital vault solution. More specifically an OpenStack's all in one solution was deployed, which means, that all services are running on the same (virtual) machine with a single HDD.

During the deployment, it was found that the essential component for this project, the **Swift** node, does not come by default, and has to be enabled and installed manually, by editing the DevStack configuration file (*local.conf*).

2.3.2 SwiftStack

The SwiftStack is an OpenStack Swift based product, that powers enterprise with a software-defined storage platform that's easy to deploy, scale, integrate with existing systems, and maintain over time. The SwiftStack platform contains an out-of-band (independent) Software-Defined Controller, SwiftStack Nodes, the Filesystem Gateway, and at its core, the OpenStack Swift object storage architecture. [17]

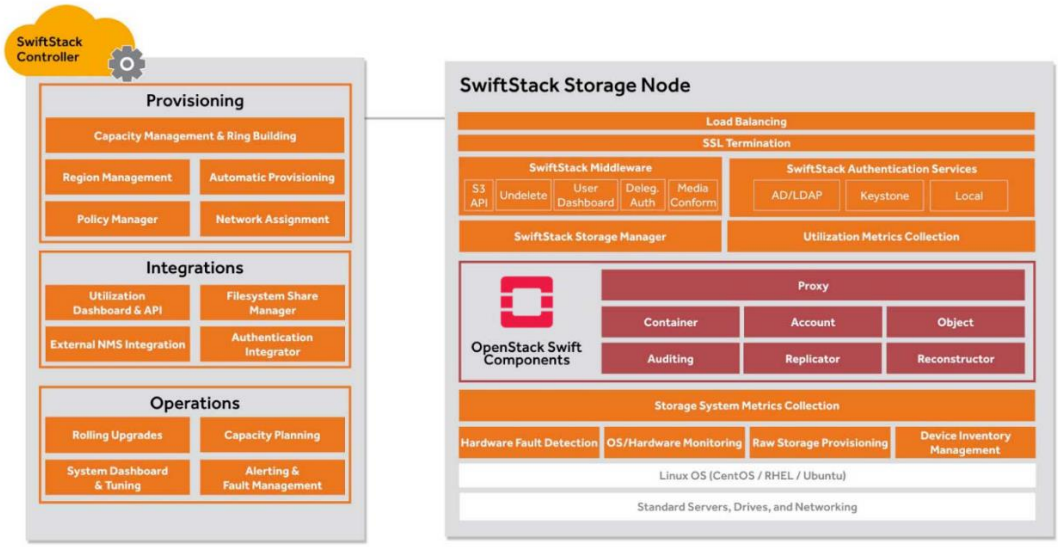

Figure 2.3: SwiftStack scheme [17]

2.3.3 MongoDB

MongoDB is an open-source NoSQL database that can works on top of geographically distributed systems, offering high availability and horizontal scaling.

A NoSQL (Not Only SQL) database, is a database without SQL interface [7]. Considering the case of MongoDB, the data is stored in JSON-like documents, rather than table entries like in a traditional relation database, allowing this way an easy and direct mapping with the application code. [30].

Besides standard database functionalities, MongoDB, also offers additional products that could reduce its implementation time and cost at large infrastructures, such as MongoDB Cloud Manager.

2.3.4 Postman

This software was designed to help the development and testing of web services, such as an API. It has an intuitive and easy to use graphical user interface which allows the creation of complex HTTP calls very quickly without worrying about authentication implementation or cookies handling. [42]

2.3.5 Fiddler

It is a free web debugging proxy which works with any browser, system or platform. This software uses the man-in-the-middle decryption technique to captures the traffic between the client and the browser/system and allows the manipulation of the HTTP requests at the real-time. [19]

By being a man-in-middle actor, between the web server and the client (e.g.: web browser), Fiddler allows the modification of the HTTP requests exchanged by them. Fiddler also supports SSL

encryption, and is capable to decode HTTPS requests to its clear-text form, since the connection is firstly established between the web server and Fiddler, and then between Fiddler and client. Basically, the traffic from the web server is being redirected to the client through the Fiddler application. When the traffic is being redirect to the client, a self-signed certificate is used.

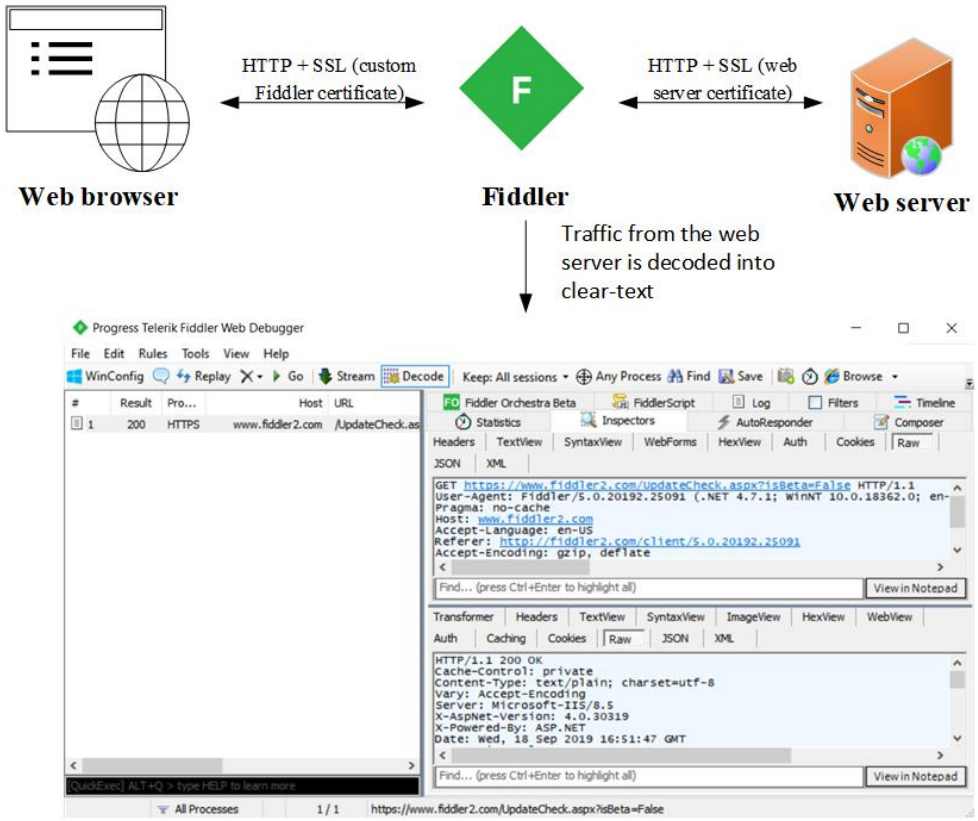

Figure 2.4: Fiddler functioning scheme

2.3.6 Burp Suite

Burp Suite is the world's most widely used web application security testing software, written in Java [11]. In terms of traffic interception, between the client and the web server, it works in the same way as Fiddler does.

Regarding its web testing possibilities, it comes with automatic tools that allows to perform several security testing operations over a selected target, and also offers the possibility of integration of custom addons. Those operations include web spidering, traffic analysis to detect headers and certificate related issues, payload injections, and other methods to detect OWASP TOP 10 attacks.

During this project, it was used as a security testing tool with the REST API server being a target, at a white box setup.

A white box setup means that the tester has knowledge about the testing target, and also has full access to the source code of the target application. [35]

2.3.7 Git

The Git is an open-source revision control system. It was used during this project as a code control system since it has direct integration with the Microsoft Visual Studio Online platform and is fairly easy to use. [14]

2.4 Bank Infrastructure

The digital vault solution requires a complex infrastructure to be operable and is mandatory to guarantee that the organization that will deploy the solution (e.g.: a Bank) must comply with a vast set of international and French standards and laws described in more detail at the Chapter 3**Erro! A origem da referência não foi encontrada.**. However, the main idea is that the infrastructure host is responsible and must assure the physical and virtual safety and security of the environment. For that reason, in the next chapters, it is assumed that all infrastructure's components and its communications, between each other, are treated as trusted hosts/services/etc. Any concerns about the system being compromised by physical or virtual means from the outside (without exploiting any possible zero-day vulnerability at any of the digital vault's solution components) are out of the scope for this project.

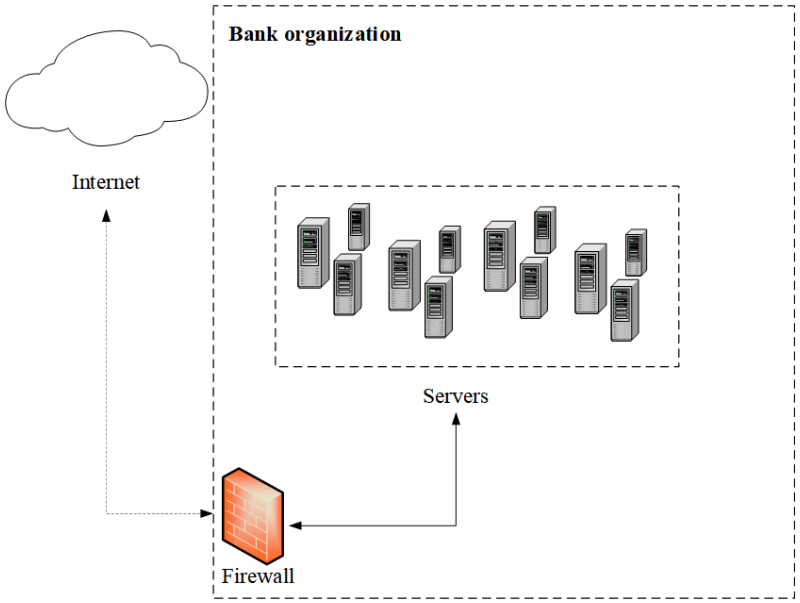

Figure 2.5: Bank required structure (simplified)

It is necessary to guarantee that the host organization can offer the required scalability and that it can handle large amounts of requests. It is also required from the solution's host organization to be compliant with the ISO/IEC 27001 and to be in charge of assuring all needed bandwidth, computational, hardware, documentation and human resource capacity. The organization security system must be separated from all other systems such as telecommunication systems (ISO 27001). The administration staff of the system must be identified and known to all other company staff that can deal with the digital vault solution.

The administration of the system must have a general security policy and rules that specify how the authorization system and intrusion detection system works, how the company's equipment, staff,

and software interact with the storage system (IEC 61000 regulations). While transmitting the data at the network use high-security measures (secure protocols and secure hardware/infrastructure) especially when an external service provider is being used. [24]

Chapter 3 System requirements and architecture

After analysing the standard documents related to digital vault's development it was possible to define the system's requirements.

A feature that could be potentially highly useful in real-life situations is the file-sharing system between different users. For that reason, a logical concept that would respect all security standards and regulations previously mentioned was designed. As a result, besides the architectural structure, functional and non-functional requirements, the rights and privileges of different roles that users can have when a vault is shared with them were specified.

To simplify solution's storage terminology, concepts like account, container and object from the Swift OpenStack were used.

3.1 Functional requirements

The main functionality of the digital vault is to store user's data, such as files or logs. Besides that, there is a list of other operations that the system must be able to perform:

- Register new user
- Authenticate user
- Create new digital vault for a specific user
- Create or delete a container for a specific digital vault
- Upload or remove an object from a container
- Share a specific digital vault with another user
- List files from a digital vault or a specific container
- Get file metadata or extracted content's information
- Get journal history
- Check journal integrity for a specific user

To guarantee privacy of system's users it should be possible to define user roles and set different ACLs (Access Control List) for user's objects.

A user must have a defined role for each vault and its container. Also, it should be possible to set different roles to a single user for each of its vaults or containers.

An ACL specifies the privacy settings of an object (document or set of documents and its metadata) for each user role's group. Any modification of the user role or object's ACL must be recorded in a log's journal.

To guarantee that the system is able to store data through long periods of time (years), it is necessary to ensure that it is compatible with different open-source document standards.

Table 3.1: Compatibility and durability requirements

Requirement	*Description*
Use formats that are standardized and support free/open-source standardized formats.	Be able to convert formats
Associate metadata with data	Use standard formats for metadata
Support media migration	
Format conversion	Check and convert the format of data if needed before it is stored. Warn in case of obsolete format. The conversion should be planned and traceable.

3.2 Non-functional requirements

Those requirements will mainly focus on aspects that guarantee integrity, availability, confidentiality, authentication, non-repudiation, and traceability of digital vault data.

In this solution, users cannot edit or replace the data once it is loaded to the system and the metadata related to all user's data and actions must be persistently stored even if the data itself was deleted, at an auditable log journal. Also, this system must respect the privacy of its users by guaranteeing durability, integrity and confidentiality of data, provide strong authentication methods, be able to handle different user roles with specific permissions and rights and record all user's actions history in a secure way.

The system must support standard file formats, such as PDF and be able to convert and work with them. (Out of the scope for the current project and hence not implemented)

Since it is possible that a digital storage will contain personal data it is mandatory to respect all rights and laws of "Digital Freedom" [36], in all electronic and digital operations. Personal data represent any information that may help or identify a specific person. That includes names, surnames, addresses, bank account numbers, phone numbers, driver license numbers, photo Id, etc. It is mandatory to satisfy legal requirements regarding control access policy and use of personal information as well as the minimum period that is mandatory to store the information and its level of security.

The software should be able to pass possible audits from the "Commission nationale de l'informatique et des libertés" – [16] as well as from an independent administrative authority that has the right to verify the security mechanisms of the data storage.

It is mandatory to respect and implement, at least, the following standards and regulations:

- AFNOR – NF Z42-013

- AFNOR – NF Z42-020
- *Délibération n° 2014-017 du 23 janvier 2014*
- *EU General Data Protection Regulation* (*GDPR*)
- ISO 19005-1 (Files format)
- ISO 8601 (Time representation)
- ISO 15489 (Records Management)
- ISO 14721 (Open Archival Information System - OAIS)
- ISO 27001
- ISO 27002

Regarding security, confidentiality, privacy and integrity, several requirements were mapped from the NF Z42-013.

Table 3.2: Confidentiality requirements

Requirement	*Description*
Encrypt files	The files should be encrypted using a complex and modern encryption algorithm.
Validate file's content	By analysing the logs it should be possible to guarantee that the file content was not edited. It can be implemented by storing the file's hash.

Table 3.3: Integrity requirements

Requirement	*Description*
The storage should guarantee:	
WORM – hardware	
WORM – software with:	
1. Event Log	
2. Mechanisms able to detect and prevent records from being edited or replaced.	
WORM logic for removable media	Enhanced security level Advanced security level
Rewrite (Standard security level)	Enhanced security level Advanced security level
Description of process for the data capture	
Warn before delete data	
Description of process for deleting data	Definition of each metadata that is stored and as well as it's period of storage. Store data metadata and logs after it were deleted.

Table 3.4: Security requirements

Requirement	*Description*
Authenticate users before grant access to the data	Strong authentication
Backup data	Use different media supports. Prevent data loss from natural disasters.
Exploitation and control of data. Identify user and create logs	Strong authentication Use different formats for consulting and inserting data.
Uninterrupted access to digital vault's data	
Timestamps	Every important action should be registered
Provide technical documentation: private policy, general terms, and conditions, operational and operating procedures, document's lifecycle	Adapt to client's needs
Keep logs for the lifecycle of all stored data (and each file) and an event log	Digital signature and timestamp for each operation or event: single document or set of documents Lifetime of logs

It is necessary to guarantee the existence of a unique time source for different system components, to ensure log's timestamps integrity and that system processes are synchronized. Timestamps must obey at least the following rules/recommendations:

1. Have a reference time source
2. Store timestamps for a specified period
3. Specify the adopted timestamp method in a technical documentation
4. Respect ISO 8601 for time representation: *YYYY-MM-DDThh:mm:ss.sTZD*

a. Example: 2007-08-29T09:36:30.45+02:00

5. Have a high accuracy level to avoid two events have equal timestamps
6. Use Coordinated Universal Time (UTC)

A record to the log's journal must be made (stored) for each event related to any operation of the system or with the life cycle of data. This process must be associated with a timestamp and performed automatically by the system.

It should be simple to read logs, but only authorized operators and users should have access to it. All operations performed by the log system, must be detailed in the technical documentation of the system and be compliant with AFNOR. The logs can be archived periodically and accordingly to the

archiving policy. At least, the journal system should guarantee durability and integrity of the logs for the same period as the document they are related to.

Archived files that reached the maximum period of retention must be destroyed (maximum period: specified by law). Once a document was deleted no one should have the ability to access it. However, it is possible to preserve the logs and metadata associated with them while respecting the archiving policy and user agreement.

3.3 Use cases

Based on the end-user perspective and to guarantee that the system satisfies all specified requirements, a set of use cases that involve the most important digital vault's features in different working conditions, were designed accordingly to its expected behaviour and outcome. All those cases were validated and its results are described at Chapter 4.5 .

3.3.1 User registration

For the user registration process, it was assumed 2 different scenarios. The first one is a regular case when a new inexistent user is registered. In the second, an attempt is made to register a user with an already registered username it the system.

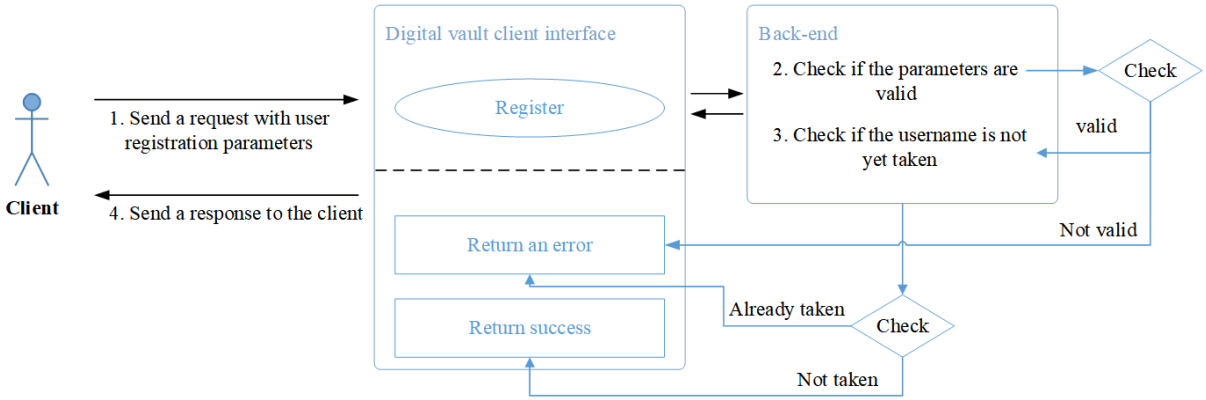

Figure 3.1: User registration use case

3.3.2 Containers manipulation

For this case, it is assumed that the client is a registered and authenticated user. For the container manipulation, it is possible to create and delete one or more containers (one by one). Note that when a container is deleted, all its files are also erased from the system, however, the information that was already recorded at the log system stays unchanged.

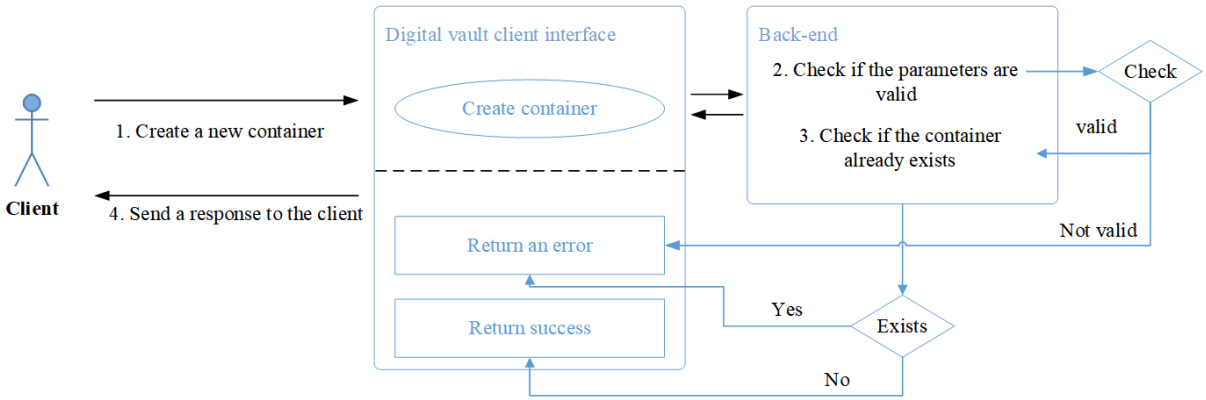

Figure 3.2: Container creation

Note that at the third step, if the container does not yet exist, a new one will be created at the cloud storage and all its associated information will be mapped at the database. The success message will be returned to the client only after the validation operation of container's creation/deletion.

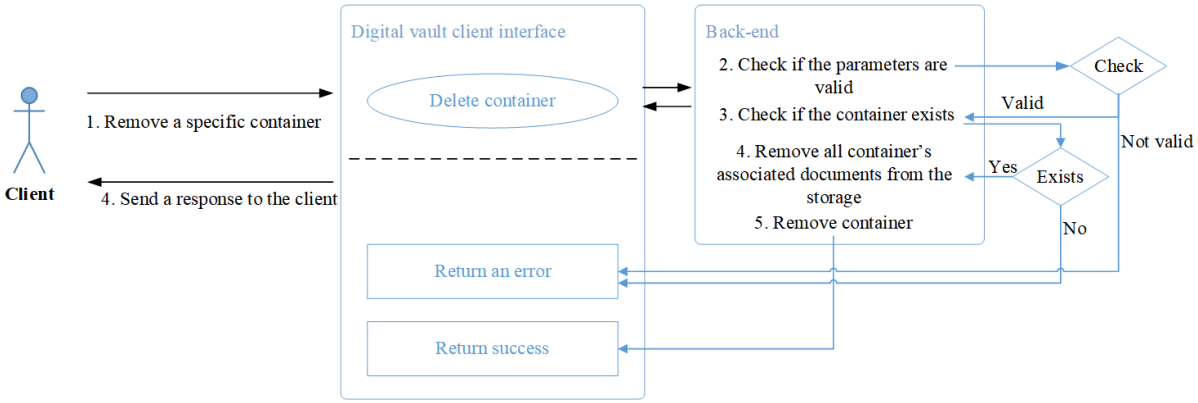

Figure 3.3: Container deletion

3.3.3 Document upload and download operations

When a user uploads a document, it must not replace any already existent document. For this reason, it is essential to verify if the system is able to handle files with the same names. When the document is encrypted by the solution's server, it should only be possible to retrieve back the clear-text version of it using the encryption key and guarantee that even if somehow, someone, has access to the encrypted document it will still be not possible to crack it (without using brute force techniques, which may require at least hundreds of years). [6]

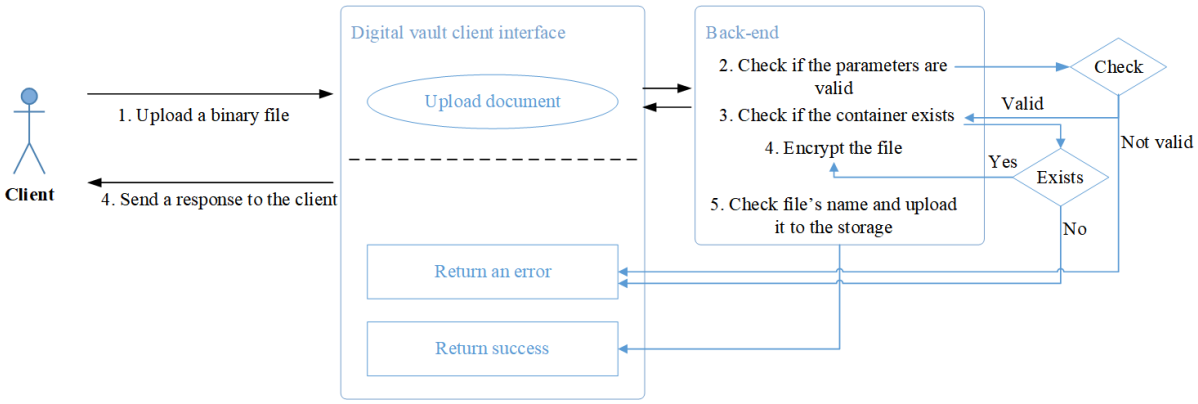

Figure 3.4: File upload

Note that at the third step (at Figure 3.4 and Figure 3.5), user privileges and ownership of the container are also evaluated in order to grant him access to this operation.

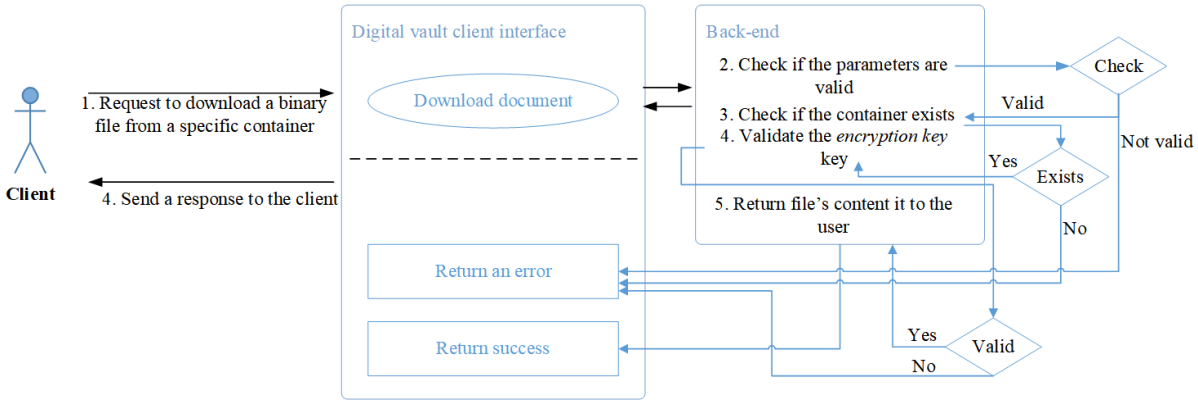

Figure 3.5: File download

3.3.4 Database malfunction

In this case, a temporary crash at the database connection has a presence. It is expected that the system will hang all user's operation until it gets back online.

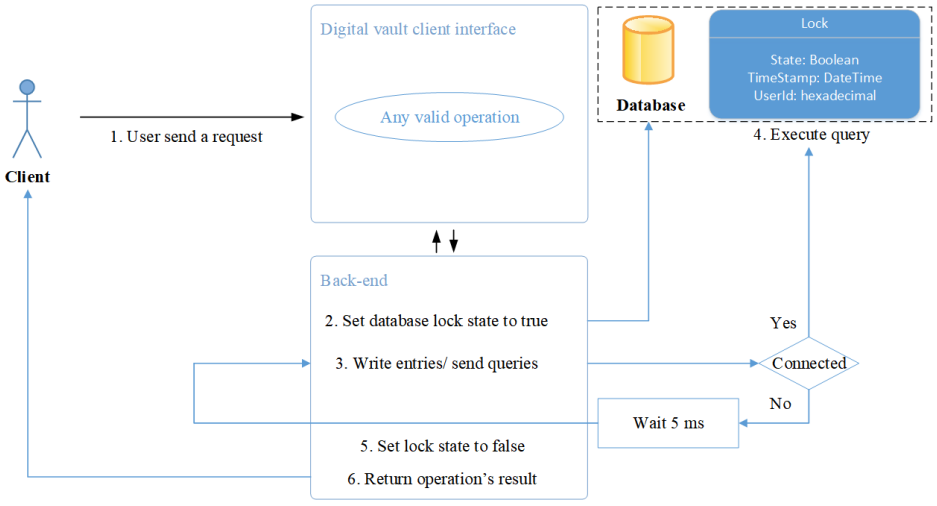

Figure 3.6: Database connection failure

3.4 System architecture

The main scheme of the solution was designed by having in mind the main technologies that were chosen to develop this project (.Net, OpenStack and MongoDB), and also the business and legal requirements.

Some of the system component can be implemented on a public cloud. For that reason, it is necessary to ensure that the most critical services and data, such as data storage and database, is kept on a safe and controlled environment [12]. Furthermore, to reduce the risk of a single environment compromising, virtual or physical separation of different system components must take place once the system is implemented [29].

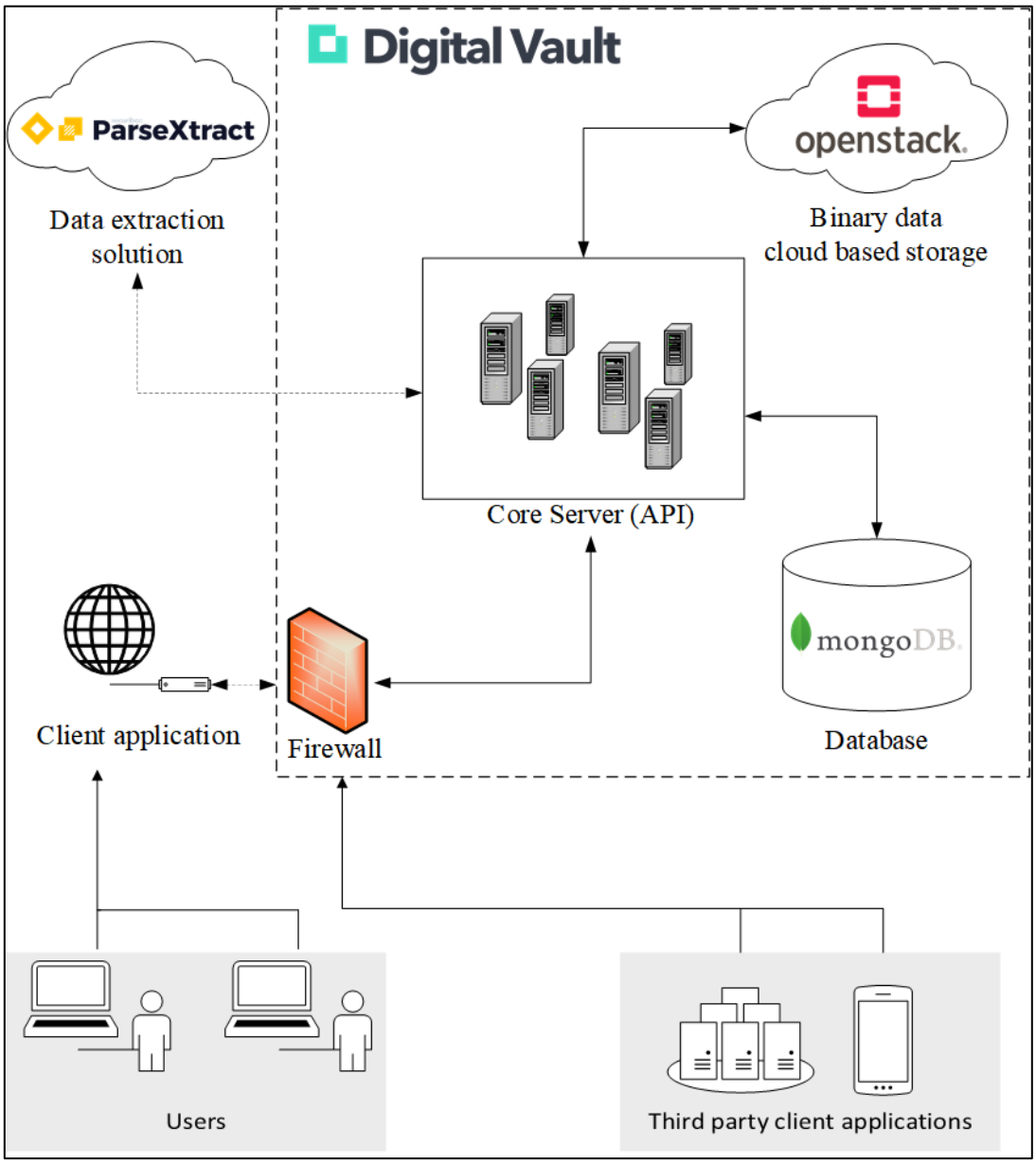

Figure 3.7: Basic solution schema

The digital vault solution is a conglomerate of several, easily scalable and redundant, components. The binary data storage is used to store a user's uploaded data, such as documents, images or any other type of files. The database is used to store user's *log* files and user personal information, such as username, password hash, authentication certificate's information, etc.. The ParseXtract solution can be used as an additional feature for this project.

The main functionality of the core is to provide the interoperability between all of the different components and to handle the data sent by the client to the solution. As a result, the core should be able to process client's data and perform a set of operations in accordance with the standards, such as client authentication, data encryption, file's integrity validation, etc.

Any client application that interacts with the solution will never have direct access to the binary or database storage and will be limited to the set of functions offered by the API core, in order to prevent any possible unauthorized access to other users' data. [34]

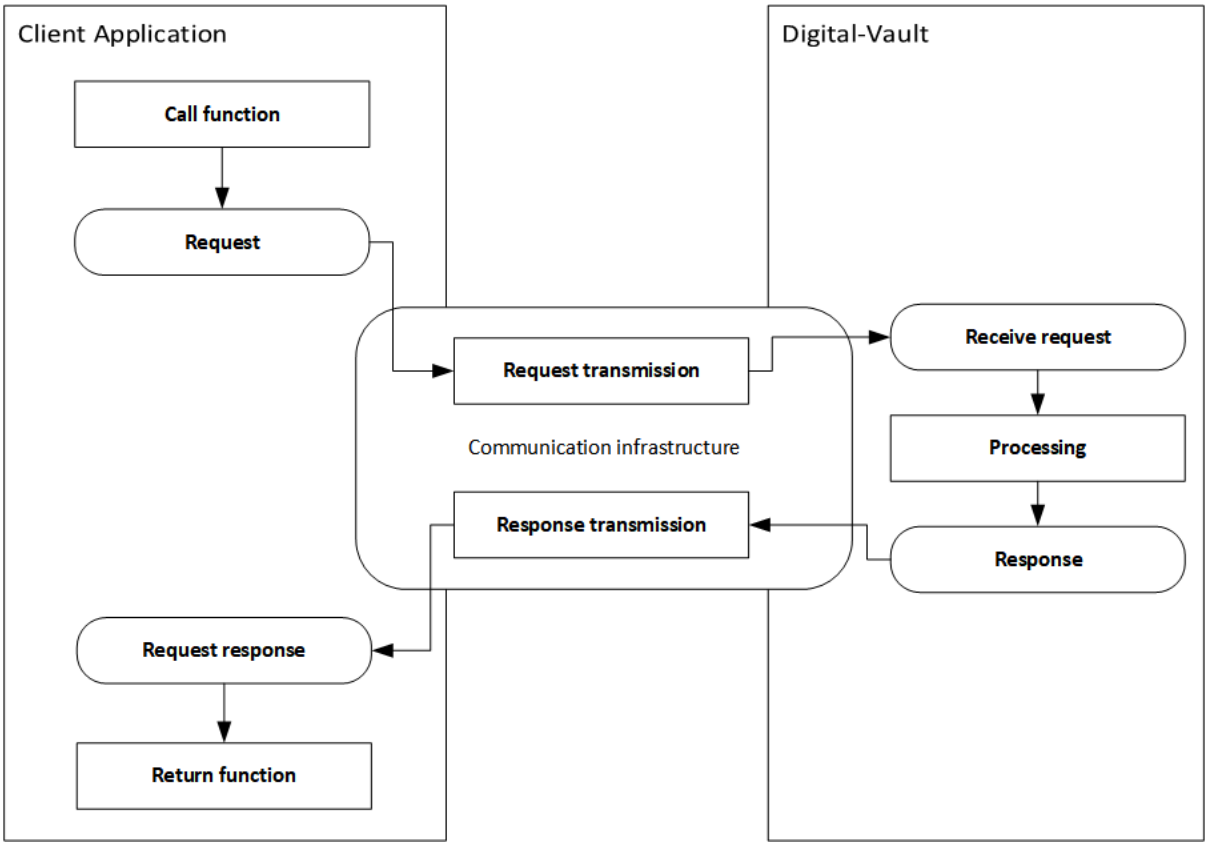

Figure 3.8: Function call diagram [34]

Figure 3.8 Figure 3.8: Function call diagram [34] illustrates the default path and order that a request has to take from the client to the digital vault service and vice versa.

To guarantee that only authorized users have access to their respective content, a secure authentication mechanism, such as mutual authentication, should exist between the client and the digital vault solution. In mutual authentication, both, client and server can prove their authenticity to each other, eliminating this way possible MIM(man-in-the-middle) attacks. [15]

The database will be used to store all user-related data, except its documents. Unlike relational database design, where the goal is to describe existent relations between different entities, in the NoSQL an aggregation design takes place. [28]

In the aggregation design, first, a conceptual representation of the system's entities is built. Then, the data that is related to each other, and consequently, that would be requested together (a join query in a relation database) is mapped into a collection. When a specific data, or set of data, is requested in different contexts, it is possible to have that same data mapped, simultaneously, into multiple collections. [10]

A collection is a set of documents, that are sharing the same structure, defined as a JSON (JavaScript Object Notation). In order to add a document (entry) to a collection, it is necessary to ensure that it follows collection's structure.

One of the challenges of using NoSQL database is related to the concurrency problems. To guarantee the integrity of the log's journal it is mandatory to have atomic commits. Since, it is possible for multiple clients to read and write the same data at the same time, consistency problems may appear. The implementation of this solution must address this problem, for example by implementing a lock system. [41]

The database is also highly connected with the journal solution's architecture, since journal's data is stored on it. The journal is a collection of log's entries. Each entry represents a past action and contains information about the author of the action, the action itself and the successfulness of the performed action. These entries will follow the ISO 8601 timestamp standard and are created during the execution of any kind of operations, since its intention towards its execution.

For instance, when a user is trying to login into the system, a log is created registering this intention. After that, user's credentials are validated, and during the validation, logs are created for each step of the validation process. By implementing this logic it is possible to trace whether the operation was successful or at which point of the process has been a failure.

To satisfy the journal durability and integrity requirements each user of the system has the right to request their own entire journal history or a subset of it, and also has the right to check journal's integrity, by executing a validation method at the server-side.

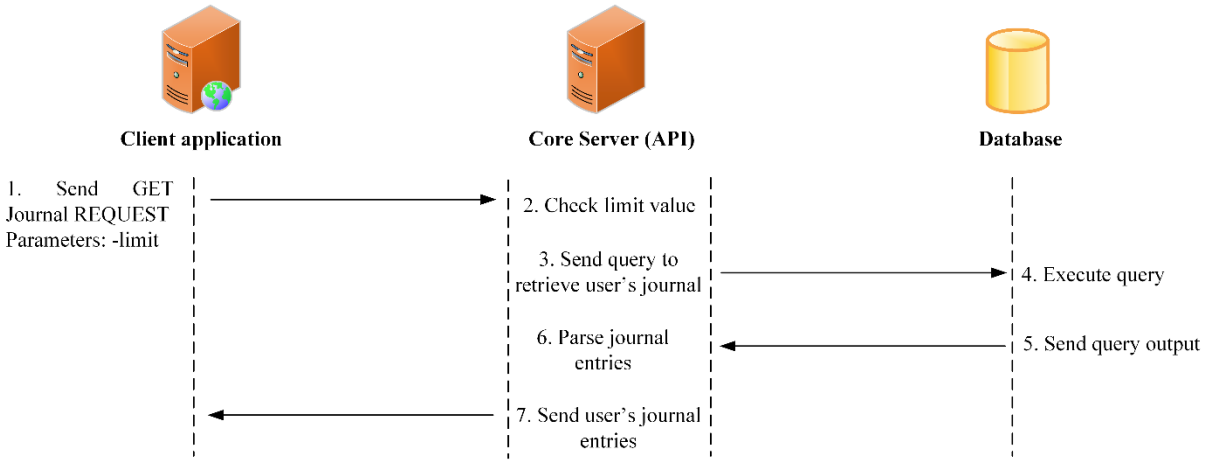

Figure 3.9: Get journal entries for a user

When a new user is registered into the system, an associated binary cloud storage account is automatically created with a randomly generated credentials. These credentials are stored in the database, and are requested by the client whenever it is necessary to access costumer's binary files.

The access to the database is restricted to the core server. However, even if the database is going to be compromised, and somehow, a malicious actor will gain direct access to the binary data storage platform, it is still possible to ensure user's data privacy, since it is encrypted.

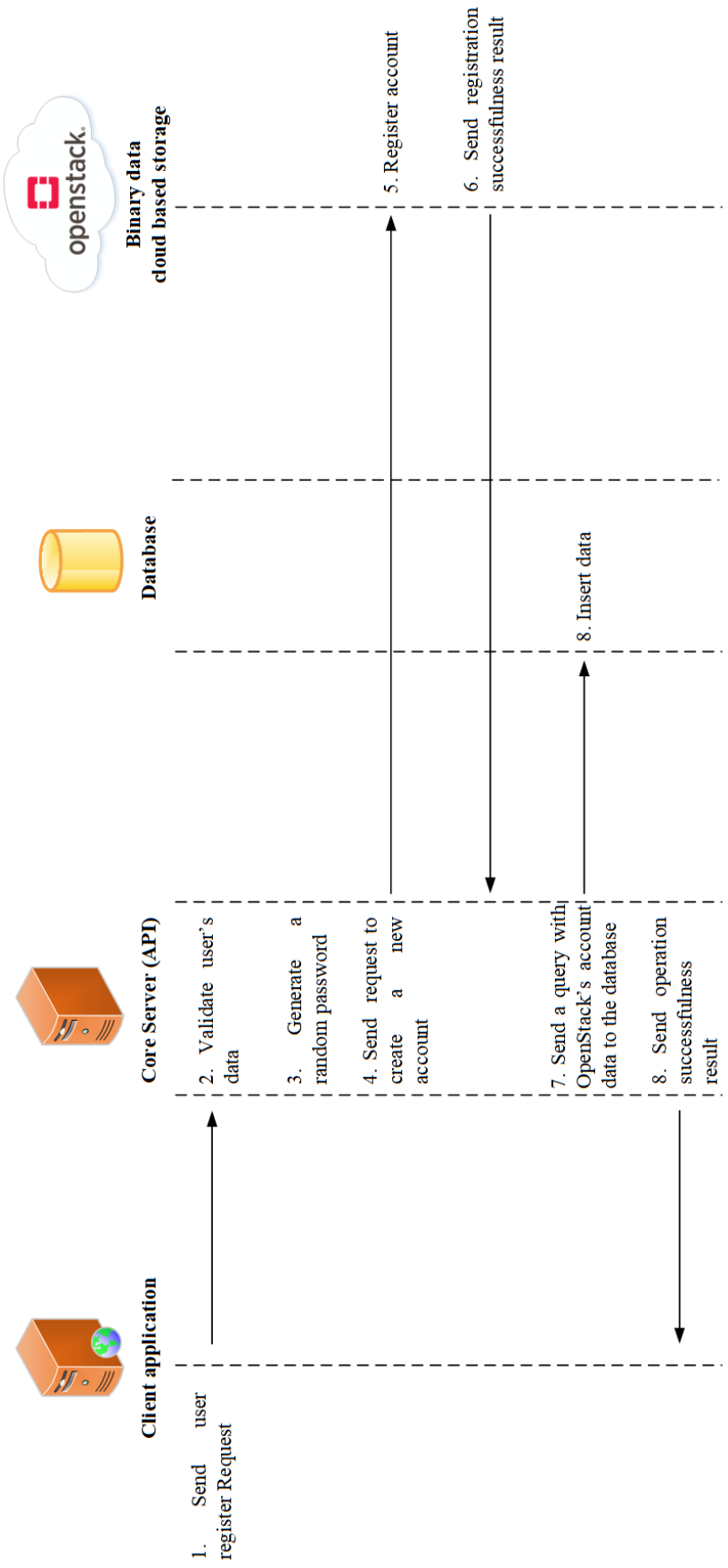

Figure 3.10: User registration scheme

All costumers documents, before stored, will be encrypted using a strong encryption algorithm, at the server side. The files are not encrypted at the client side, because if an information extraction service is used, such as Securibox ParseXtract, it is necessary to guarantee its ability to read the document content in a clear-text mode before it is encrypted.

The keys for files' encryption and decryption should be retrieved by the core server, only, when a respective request comes from an authenticated and authorized client. Also, those keys shall never be stored outside of the RAM. A possible way to achieve this, is by using a scheme where user-provided passwords, designated as master-keys, are used to encrypt file encryption keys [1] and where each container can be associated with a different master-key. This way, even when a master-key is compromised, the file's encryption key still safe. This also means, that the master-keys can be changed at any time without affecting the file's encryption key. [8]

The binary data storage account information will be stored at the database and will be only accessed when needed, exclusively by the core server. That way, the authentication information regarding binary storage component, will be independent and unknown to the end-user, which also permits the use of this encryption technique to share vaults between different users.

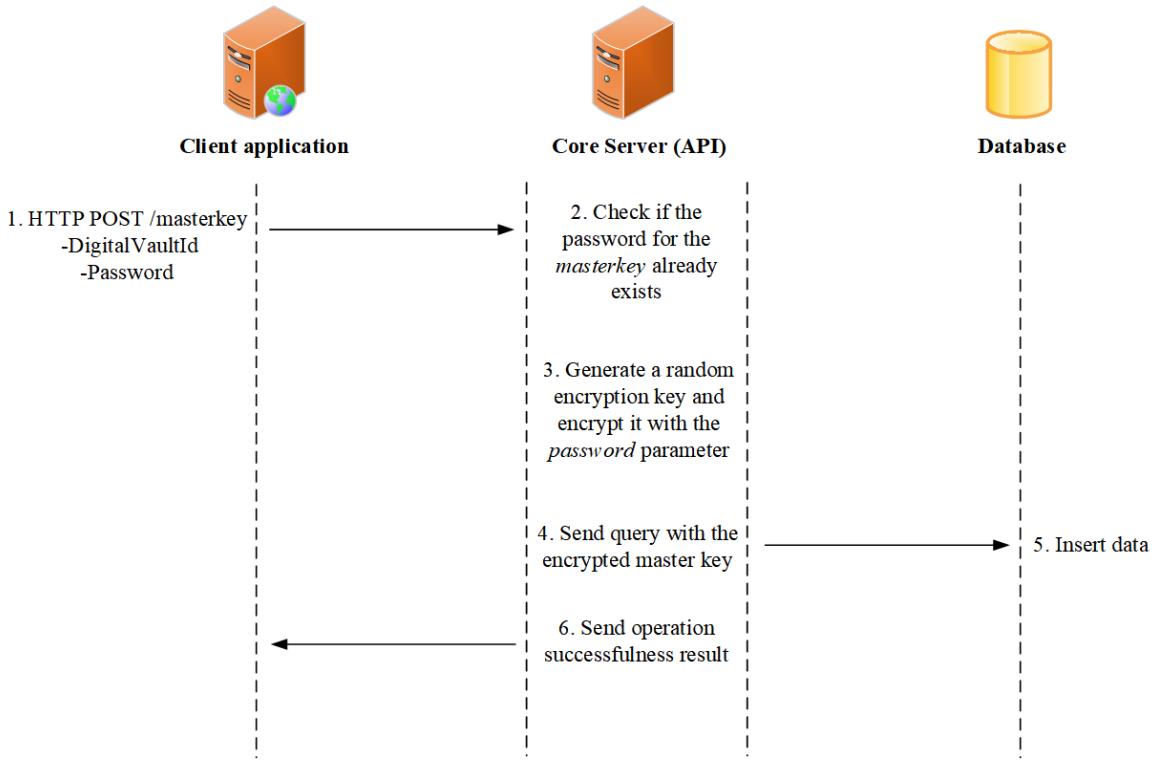

Figure 3.11: Master key creation

The encryption scheme can be divided in the following steps:

1. A strong and random encryption key is generated at the server side (Core API).
2. The key generated at the first step is encrypted with a user-provided password (master-key).
3. Whenever the user uploads or downloads a file, it is necessary to provide the master-key from the send step.

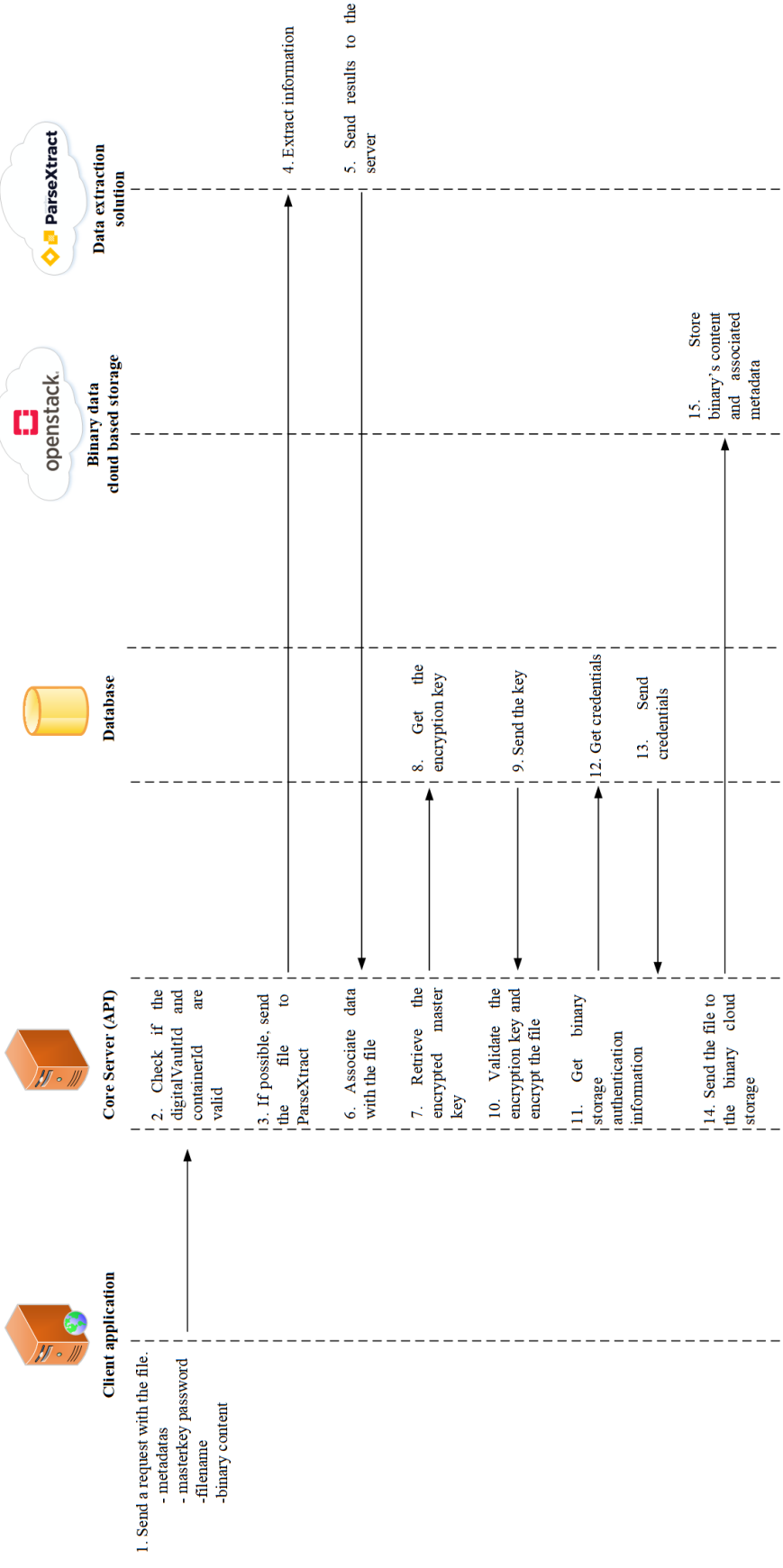

Figure 3.12: Upload a document to the digital vault

In order to download a file from the system, a similar to file's upload request has to be sent to the API server. The file is then decrypted at the server, using the master key password provided by the user and the encryption key itself from the database. Lastly, a clear-text document is sent to the client. Also, it is possible to access just the file's metadata by sending the respective request to the server.

When a user wants to share access to his digital vault with someone else, he must give the respective master key to another user. This can be achieved if each user of the system will have an associated public user id, which could be shared with other members without revealing any costumer's personal data.

To satisfy the requirements related to user roles and ACLs, the solutions will adopt, three different roles. However, it is possible to add more roles with customizable settings if requested by the client.

A role can be applied to a user or to an entity:

Table 3.5: Digital vault user roles

Role	*Modify an archiving role*	*Insert new data*	*Consult data*	*Set the lifetime of documents/data*	*Delete a document in advance*
Admin	x	x	x	x	x
Service		x	x	x	x
User		x	x		

3.5 API specification

Each method of the API endpoint was specified in order to attend a specific solution requirement. By specifying the methods, it is possible to ensure that no requirement will be left aside during the implementation.

To save development time later, the Swagger Editor was used at this stage. The Swagger is an online platform which offers multiple tools for API specification and implementation. By using the YAML (YAML Ain't Markup Language) specification language it is possible to automatically generate the API source code into multiple programming languages. The generated source code, contains fully functional HTTP request methods[1] and the structure of previously specified methods (the methods are empty and must be implemented manually).

Overall, 27 methods were described at the Swagger API specification. The specification contains the name of the methods, its parameters, a description and possible HTTP status response codes. [23]

[1] HTTP request methods: GET, HEAD, POST, PUT, DELETE, OPTIONS, TRACE, PATCH

Table 3.6: Core server API endpoints

HTTP Method	*Url*	*Description*
GET	/digitalvaults/{digitalVaultId}	Get the list of files associated with a specific digital vault
POST	/digitalvaults/{digitalVaultId}	Upload a file to the main container
GET	/digitalvaults/{digitalVaultId}/containers/{containerId}	Get the list of files associated with a specific digital vault
POST	/digitalvaults/{digitalVaultId}/containers/{containerId}	Upload a file to an containerId
GET	/digitalvaults/{digitalVaultId}/files	Get file's content
DELETE	/digitalvaults/{digitalVaultId}/files	Delete a file from the digital vault main container
GET	/digitalvaults/{digitalVaultId}/containers/{containerId}/files	Get file's content
DELETE	/digitalvaults/{digitalVaultId}/containers/{containerId}/files	Delete a file from a specific container
GET	/digitalvaults/{digitalVaultId}/files/metadata	Get file's metadata
GET	/digitalvaults/{digitalVaultId}/containers/{containerId}/files/metadata	Get file's metadata
POST	/container	Create new container.
DELETE	/container	Delete user container
POST	/sharevault	Share digital vault or container with other user
POST	/deleteshare	Remove the sharing from a specific user
POST	/listshareusers	Check who has access to a specific share
POST	/register	Register new user
POST	/updatepassword	Update user’s password
GET	/getuserinfo	Retrieve user information
POST	/login	Login with username credentials
DELETE	/deleteaccount	Delete account
GET	/journalhistory	Get last journal entries for the current user

GET	/checkjournalintegrity	Check the journal integrity
GET	/masterkey	Get encryption key
POST	/masterkey	Create master key for a specific digital vault
POST	/updatemasterkey	Update master key's password for a specific digital vault
GET	/masterkeyCheck	Verify if a specific digitalVault already has a masterKey encryption key
GET	/verifyMasterKey	Verify if the masterKey password is correct for a specific digitalVault

Chapter 4 Implementation and validation

4.1 API Server Implementation

For the system database, the MongoDB was chosen, because it supports large sets of data and is a scalable solution. For the binary storage, OpenStack was chosen, since it is a very powerful solution in terms of private and public cloud storage. Besides that, the OpenStack manages the layer responsible for the data replication.

The OpenStack and MongoDB were installed and configured at a Securibox local server running Linux operative system (Ubuntu server v16.10), accessible at a LAN level or by using Securibox VPN. The API and Web server are running using a Microsoft Windows machine since they were developed using .NET technologies and require .NET libraries and Microsoft Windows services to run properly.

The digital vault core server is in charge of communication between OpenStack and MongoDB and also responsible for processing of user's requests. Due to that, the project was divided into three stages of development:

1. OpenStack communication layer development stage
2. MongoDB communication layer development stage.
3. REST API - A combined layer in which both services are being used at the same time to attend user's requests

To make the development easier the Postman and Fiddler software were used to send manual HTTP requests to the server and OpenStack API.

The overall .Net solution has approximately 4.2k lines of code and a maintainability index of 77.7 (0-100 - higher the better).

To provide a better overview of the project implementation in terms of flows and coding, a pseudo-code algorithms for each use case are presented. Namely, the user registration, container creation and deletion, and file upload and download.

User registration:

Main method:

```
If validParameters(username, password, ...) != true:
        Return

success = DigitalVault.InsertNewUser(username, password, ...)
Journal.AddEntry("User registration operation", success.errorCode, username)
```

```
Insert User:
    DigitalVault.InsertNewUser(username, password){

        If UserAlreadyExists(username) == true:
            Journal.AddEntry("Existent user registration attempt: check user name",
            errorCode, username)
            Return success(code=errorCode)

        InitializeOpenStackAccount(username)
        InitializeLogJournal(username)
        User = new User(openstackAccount, username, password hash, ...)
        UsersDatabase.AddNewUser(User)
        Return success
    }
```

Container creation:

```
Main method:
    If validParameters(username, password, ...) != true:
        Return

    Journal.AddEntry("Container creation attempt", username)
    success = DigitalVault.InsertNewContainer(username, containerId)
    Journal.AddEntry("Container creation operation", success.errorCode, username, containerId)

Create Container:
    DigitalVault.InsertNewContainer(username, containerId){

        If Authenticated(User) == false:
            Journal.AddEntry("Container creation attempt: authentication failed",
            errorCode, username)
            Return

        OpenStack.Authenticate(username)
        Journal.AddEntry("Container creation attempt: OpenStack layer", username,
        containterId)
        success = OpenStack.AddContainer(username, containerId)

        if success == false:
            Journal.AddEntry("Container creation result: OpenStack layer",
            success.errorCode, username, containterId)
            Return success(errorCode)

        UserDatabase.AddContainerInformation(username, containerId)
        Return success
  }
```

Container deletion:

Main method:

```
If validParameters(username, password, ...) != true:
        Return

Journal.AddEntry("Container deletion attempt", username, containerId)
success = DigitalVault.DeleteContainer(username, containerId)
Journal.AddEntry("Container deletion operation", success.errorCode, username, containerId)
```

Delete Container:

```
DigitalVault.DeleteContainer(username, containerId){

        If Authenticated(username) == false OR Authorized(username) == false:
                Journal.AddEntry("Container deletion attempt: authentication failed",
                errorCode, username)
                Return

        If User.ContainerExists(containerId) != true:
                Journal.AddEntry("Container deletion attempt: not found", errorCode,
                username, containterId)
                Return

        OpenStack.Authenticate(username)

        If User.ContainerContainsDocuments(containerId) == true:
                DeleteAllDocuments(containerId)

        success = OpenStack.RemoveContainer(username, containerId)

        if success == false:
                Journal.AddEntry("Container creation result: OpenStack layer",
                success.errorCode, username, containterId)

        UserDatabase.UpdateContainerInformation(username, containerId:removed)
        Return success
}
```

File upload:

Main method:

```
If validParameters(username, password, ...) != true:
        Return

Journal.AddEntry("File upload attempt", username, containerId, fileId)
success = DigitalVault.UploadFile(username, containerId, filleId, fileContent, masterKey)
Journal.AddEntry("File upload operation", success.errorCode, username, containerId, fileId)
```

Upload:

```
DigitalVault.UploadFile(username, containerId, fileId, fileContent){

        If Authenticated(username) == false OR Authorized(username) == false:
                Journal.AddEntry("File upload attempt: authentication failed", errorCode, username)
                Return

        If User.ContainerExists(containerId) != true:
                Journal.AddEntry("File upload attempt: container not found or not authorized", errorCode, username, containterId)
                Return

        If User.PermissionToUploadToContainerl(containerId) == false:
                Journal.AddEntry("File upload attempt: not enough permissions to execute this operation", errorCode, username, containterId)
                Return

        If CheckMasterKey(containerId, masterKey) == false:
                Journal.AddEntry("File upload attempt: wrong or non-existent master key", errorCode, username, containterId)
                Return

        If Container.HasDocument(fileId) == true:
                fileId = fileId + "(n)" //n is an integer number, starting from 1

        File = File(fileId, fileContent)
        File.additionalMetadata = ParseXtract.Analyze(fileContent) //not implemented
        encryptionKey = Database.RetriveEncKey(username, containerId, masterKey)
        File = Encrypt(Algorithm, encryptionKey, fileContent)
        Journal.AddEntry("File upload start: OpenStack")
        success = OpenStack.Upload(username, containerId, File)
        Return success
}
```

File download:

Main method:

```
If validParameters(username, password, ...) != true:
        Return

Journal.AddEntry("File download attempt", username, containerId, fileId)
success = DigitalVault.DownloadFile(username, containerId, filleId, masterKey)
Journal.AddEntry("File download operation", success.errorCode, username, containerId, fileId)
```

```
Download:
    DigitalVault.DownloadFile(username, containerId, fileId){

            If Authenticated(username) == false OR Authorized(username) == false:
                Journal.AddEntry("File download attempt: authentication failed", errorCode,
                username)
                Return

            If User.ContainerExists(containerId) != true:
                Journal.AddEntry("File download attempt: container not found or not
                authorized", errorCode, username, containterId)
                Return

            If User.PermissionToDownloadFromContainerl(containerId) == false:
                Journal.AddEntry("File download attempt: not enough permissions to execute
                this operation", errorCode, username, containterId)
                Return

            If Container.HasDocument(fileId) == false:
                Journal.AddEntry("File download attempt: file not found",

            If CheckMasterKey(containerId, masterKey) == false:
                Journal.AddEntry("File download attempt: wrong master key", errorCode,
                username, containterId)
                Return

            encryptionKey = Database.RetriveEncKey(username, containerId, masterKey)
            Journal.AddEntry("File download start: OpenStack")
            success = OpenStack.Download(username, containerId, fileId).Decrypt()
            Return success
    }
```

4.1.1 OpenStack layer

Since it was not possible to find a good and working SDK for the .Net project a new one was created. The communication with OpenStack Swift is done using the HTTP protocol. To develop the SDK, the OpenStack Swift API (https://developer.openstack.org/api-ref/object-store) was analysed and written into swagger (https://editor.swagger.io) to get a fully functional C# library with all HTTP core methods, such as GET, POST, PUT, DELETE already implemented for the specified endpoints as synchronous and asynchronous methods.

After, a mapper was developed to read JSON formatted response as well as some headers from OpenStack Swift API endpoint transforming the data into .Net objects such as:

- SwiftAccount
- SwiftAuthentication
- SwiftContainer

- SwiftObject
- SwiftProject
- SwiftUser

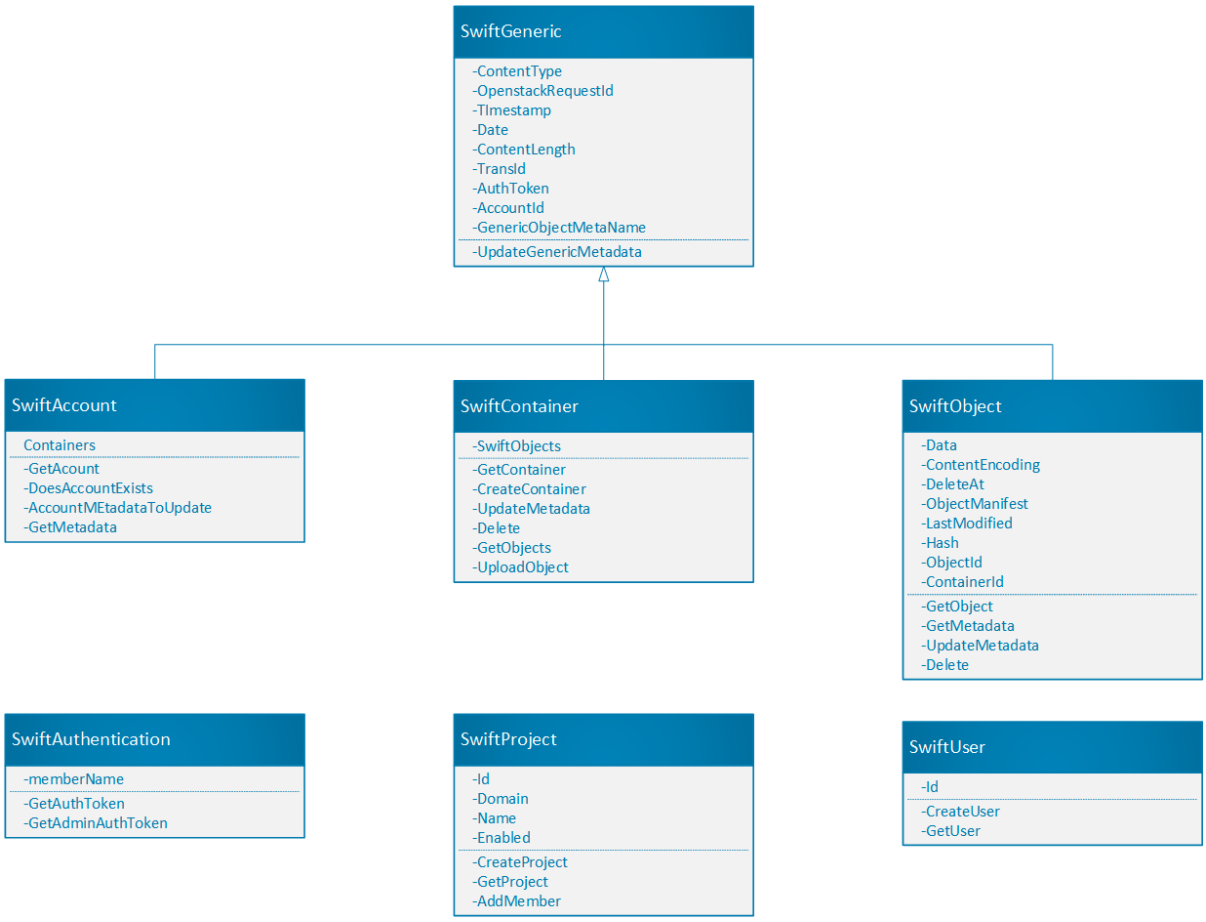

Figure 4.1: .Net OpenStack Classes with proprieties and methods

After the mapper implementation, some of the methods to interact with OpenStack API generated by the swagger were modified and a new exception handler with costume exceptions were implemented. To allow the creation of the new accounts via API it is necessary to authenticate with administrator rights to the OpenStack. Due to that fact, a method to log in as administrator was developed. This method reads the administrator credentials from the project's local configuration file, and then sends them to the OpenStack which returns the session token that is used to create a new request to create a new account. This way, all communication between the API Server and the OpenStack are totally neutral and absent to the client.

4.1.2 MongoDB layer

Following the methodology described at the system's architecture, a conceptual design of the database was built by having in mind digital vault solution's requirements and architectural decisions.

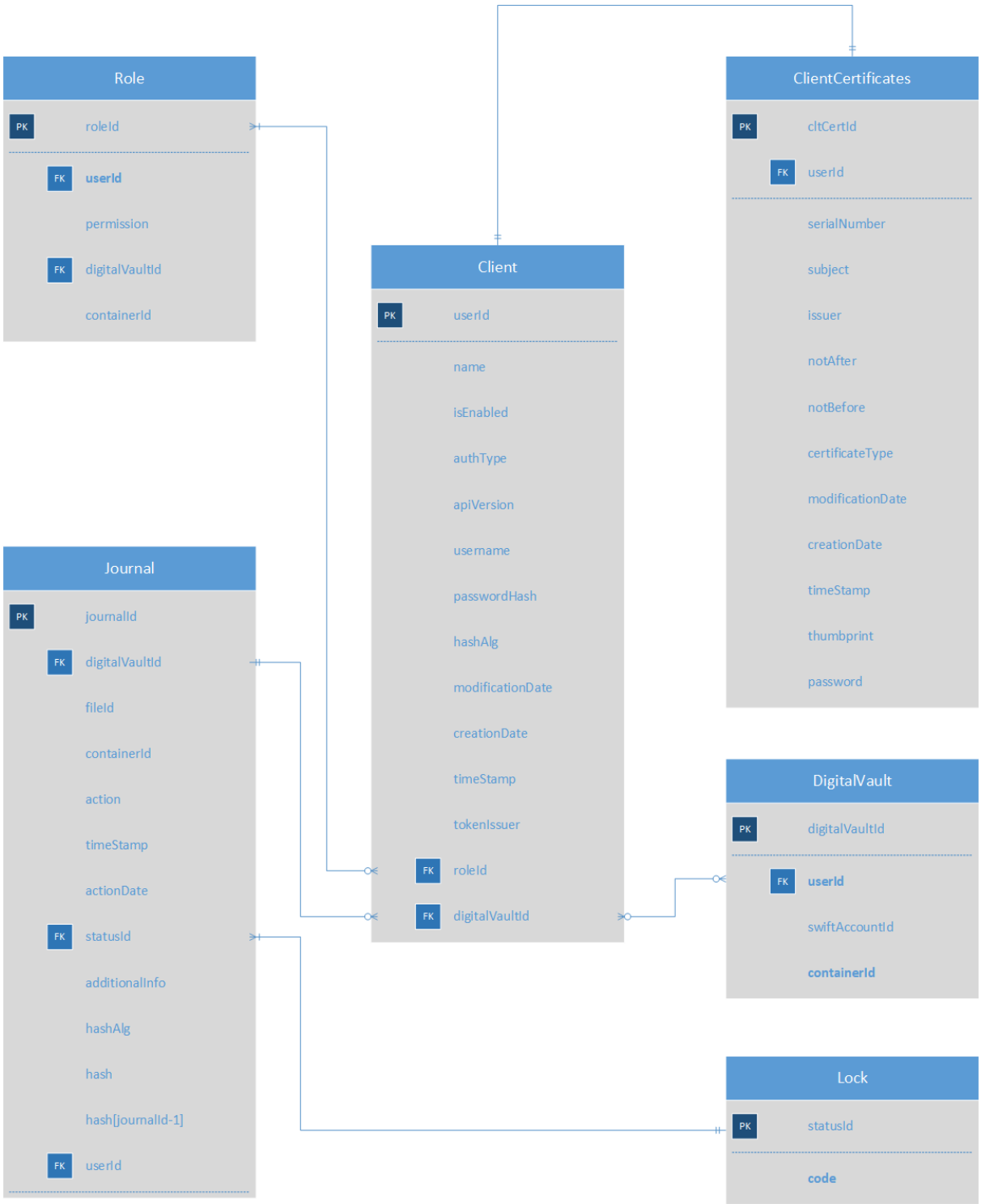

Figure 4.2: Database conceptual model

Then, the data was aggregated by considering how frequent each attribute of the database model can be requested. After, using JSON (JavaScript Object Notation) the collections document's structures were specified. Overall, the database has 7 collections, presented below:

- User information collection.
- Journal collection.

- User encryption key collection.
- OpenStack account information.
- Users password salt collection.
- Two-phase commit collection.
- User sharing information.

To make calls to the database the MongoDB, the Driver library (https://mongodb.github.io/mongo-csharp-driver/) was used. This library allows to make asynchronous calls to the database and to translate its objects into .Net objects automatically.

In the .Net project classes were created to establish the connection with each collection from the database, namely:

- MongoDBJournalInstance
- MongoDBMasterKeyInstance
- MongoDBOpenStackAccountInstance
- MongoDBSaltInstance
- MongoDBTwoStepInstance
- MongoDBUserInstance
- MongoDBUserShareInstance

First, all collection's elements were defined as static classes, in order to match the database collection elements names so it would be possible to load any entry from any collection. Then, asynchronous methods were implemented with CRUD functionality to each collection.

A possible scenario for a concurrency problem can be found when a read operation is performed to get the user's last log entry hash. A simple lock system is described at MongoDB official documentation. [31]

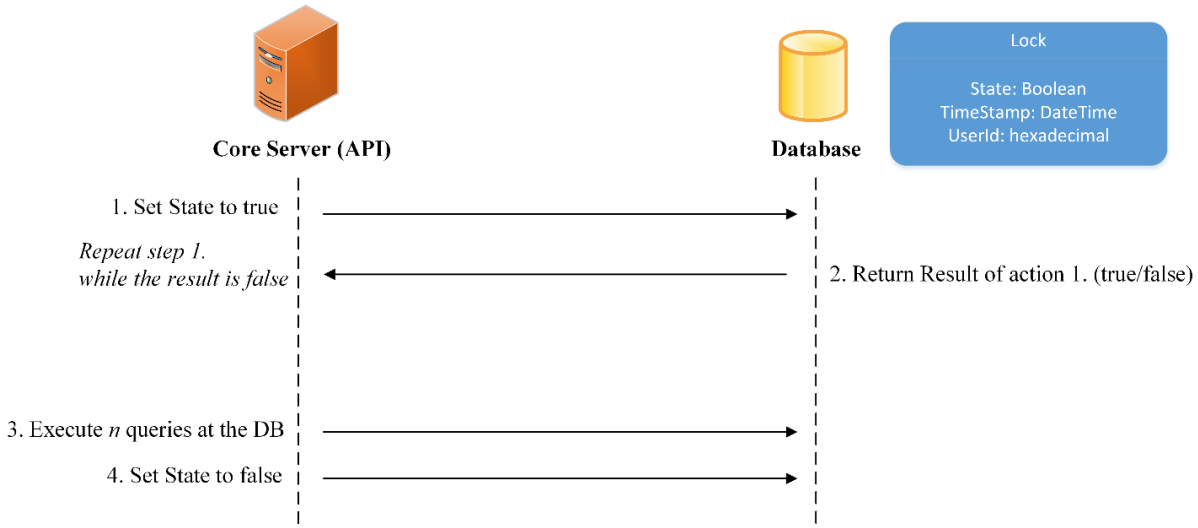

Figure 4.3: NoSQL two-phase commit

The two-phase commit consists of creating an additional collection to store the "lock" state. The lock collection will have objects with the following attributes: a timestamp, a lock state and a user Id.

The collection will contain a single object per user, that way before attempting any action on the webserver a request for the lock database will be sent in order to change its state to "busy", which can be achieved using a Boolean type variable (*state*), which in case of a successful state change will return *true*. After locking the state, a read-on user's log will be made in order to get the last entry hash and only after completing an action, successfully or not, the lock state will be reset allowing new actions on the system. If the lock's value is already set as true, the system waits 5 ms before attempting the operation execution again (while loop).

In the scenario that the connection to the database would be lost for a certain period of time, to satisfy the journal auditable and integrity requirement, the solution would be inoperable for the users that had their lock entry set to true, for an indefinite time. Since the MongoDB, when properly configured offers redundancy and availability, eventually, the solutions (core API) will be able to contact the database and unlock the system by changing the lock entry state.

The journal logs are stored at the MongoDB database, and for each entry request it is necessary to ensure that it came from an authorized system, use secure data transmission protocols, guarantee data integrity and confidentiality, and entry uniqueness. [3]

The secure logging protocols are divided in 4 main classes [4]. Namely, the syslog, Schneier/Kelsey, Ma/Tsudik and Encrypted search. At this project, a syslog-based protocol was implemented, since it can respond to all presented challenges.

Since logs requests are sent by the core server, internally on the system without leaving the organization network (at least virtually), it is possible to ensure the confidentiality of the sent data. To guarantee the uniqueness of an entry, the lock system, described above, was implemented.

To guarantee data integrity, once the entry is stored, the entry can be signed. Moreover, after the first entry, it is possible to include the information (hash) from the previous entry to the new one, and by that create a blockchain, where each entry is signed. This approach is described at the syslog-sign secure logging protocol [4]. By that, a chain of logs is established, and it is possible to validate all entries, by calculating its hash.

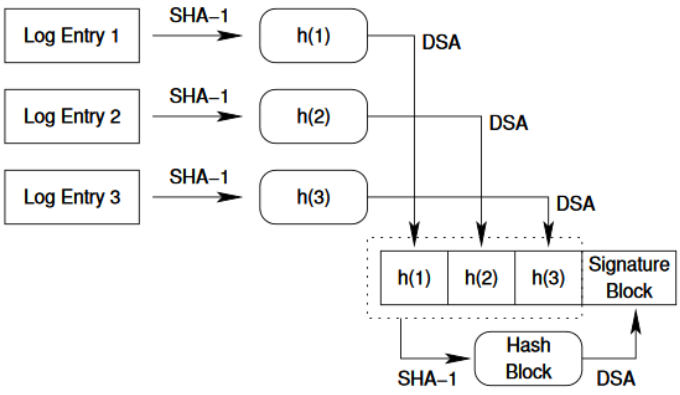

Figure 4.4: Creation of signature blocks in syslog-sign [3]

It is important to emphasize that in the current solution's implementation blocks are not signed, although its source code was written in a way that it would requires little effort to start using an external trusted service authority that would sign these blocks (also, the Securibox has its own signing solution). This feature was not implemented at this stage of the project for economic reasons since each signature involve monetary costs. Also, to reduce costs involving signatures it is possible to sign only every n^{th} entry, instead of all log entries. Since this is a blockchain solution, if it is possible to guarantee the integrity of the n^{th} entry, considering that the hashes of all previous not signed entries were successfully validated, it is possible to assure the integrity of the entries previous to the n^{th}.

4.1.3 Authentication

Since this is a REST API project, the authentication is a must before the execution of any request. The authentication is made by using SSL certificates issued by the host of the digital vault. Once the certificate is validated, the server contacts the MongoDB's OpenStack collection and retrieves the credentials for this specific user to authenticate at the OpenStack service. After, some claims, stored at the server's RAM are used to save the user's username and OpenStack's authentication token for the lifetime of the operation execution.

In the case of the certificate based authentication, the user's certificates must be signed by the company which is operating the digital vault. This way it is possible to ensure, that not only the user authenticates the server, by receiving its certificate, but also the server guarantee the user's authenticity. Also, by implementing this authentication method, it is possible for the solution's host organization to easily adopt this project to implement smart-card (bank card) based certificate signing solution for authentication. [22]

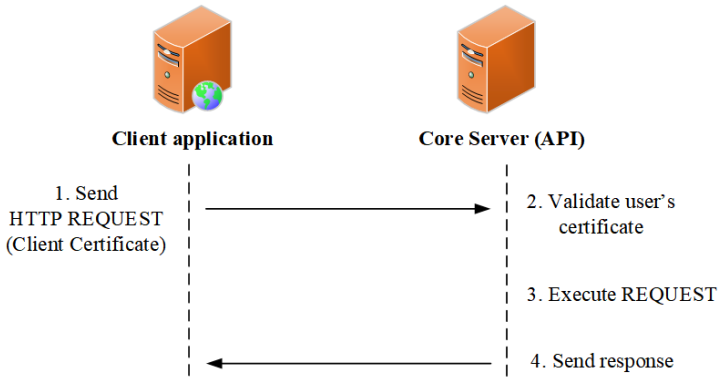

Figure 4.5: Certificate authentication scheme

Also, basic authentication was implemented to make the development process easier. In the basic authentication, the username and the password are encoded into base64 (not secure to use in production).

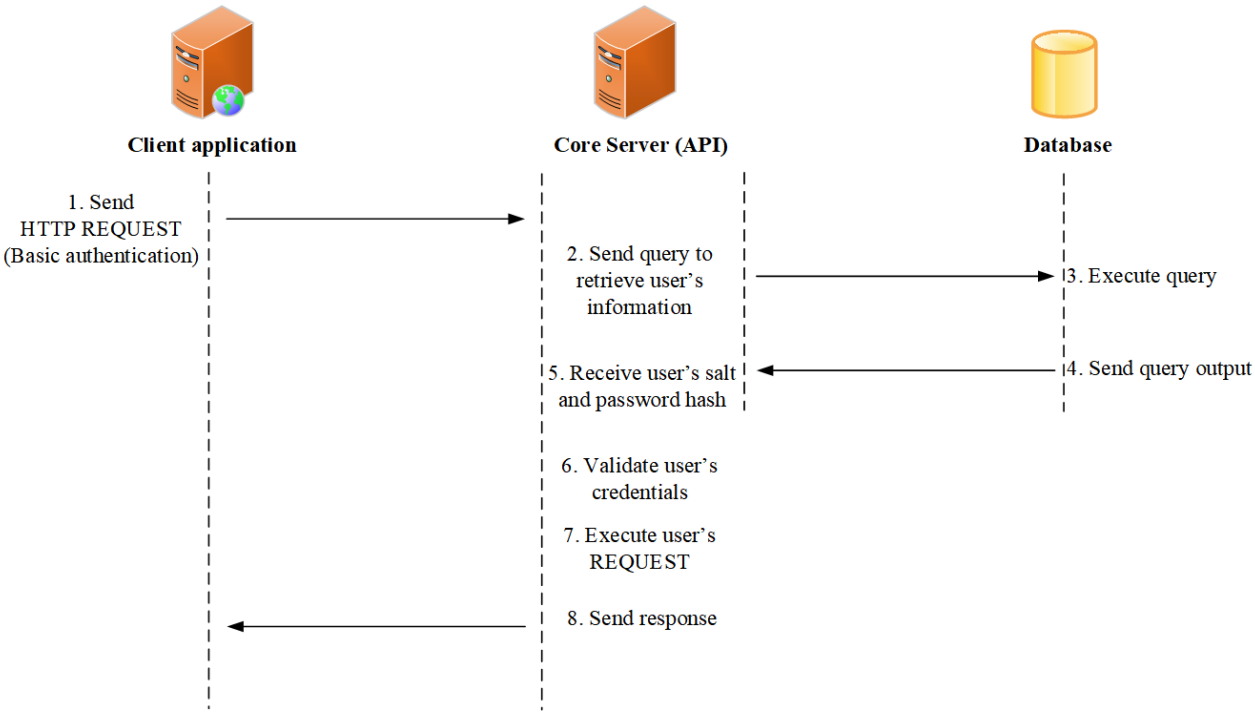

Figure 4.6: Basic authentication scheme

4.1.4 Core Server layer

To simplify the process of the communication between all solution's nodes a REST-API interface and all methods regarding data manipulation were implemented, resulting in the core server layer from the architecture scheme.

One important fact to mention that may affect the performance of the API is that for each received concurrent call a new thread is created. For that reason, even if only one user is using the system, the number of threads that serves his requests may not equal to one. [27]

In the Visual Studio solution, controllers regarding different operations were separated into different files, in order to add more organizational logic to the project, namely:

- FilesController
- UserController
- MasterKeyController

Since the journal has to keep track of everything that user's do inside of the system, a method to add entries to the journal was developed. This method, called *AddJournal()*, gets the previous entry hash and then calculates the new journal entry hash. This way it is possible to guarantee not only the integrity of data but also that the order of journal entries corresponding to the order of server actions. Besides that, the action, its description, an error code and additional data can be added to the entry. An entry will

be added to the journal after each user request, at least, at the beginning and at the end of the operation, to make possible to know if the operation went well or no.

Another important method is the file's content encryption/decryption method. Any time the user wants to get access to the file's content or to upload a new document to the vault, first user's permissions are validated and then a key (a *master Key's password*) introduced by the user at the header of the request is used to retrieve the encryption key. The encryption key is a randomly generated 256 bit string. The key also contains salt and has an initialization vector. The generated key is encrypted with the user's *master key* hash value. The downside of having a mechanism such this is that if the user forgets the *master Key* password it would be almost impossible to retrieve the encryption key again.

By implementing this logic, it is possible to ensure that even if someone, somehow, has direct access to the OpenStack Swift storage the files will be encrypted with a complex algorithm (AES-256 or superior) and by that ensure the necessary confidentiality level to the user's files.

To validate if the *master key* value is correct, a known string, encrypted with the encryption key generated during the creation of the container, is stored at the database. This way, by encrypting the known string with an encryption key derived from the user provided *master key*, it is possible to ensure the correctness of it by comparing two encrypted strings.

Since the file is being encrypted at the server-side, it is important to refer that is the responsibility of the solution's host to ensure all kinds of infrastructure security measures. By that, it is assumed that the solutions running environment is never compromised and there is no risk to send a clear-text document to the server over a secure HTTPS connection (man-in-the-middle attacks are out-of-the scope).

Each digital vault has its owner which is represented by the public digital vault user id. When a user is attempting to access a digital vault, first the system will check if that user has the correct privileges and access rights to that container/vault. After that, the system will use the digital vault's owner id to authenticate with OpenStack in order to get access to the files. However, even if the user has access to the vault, he won't be able to get any correct file's content if a wrong master key is used.

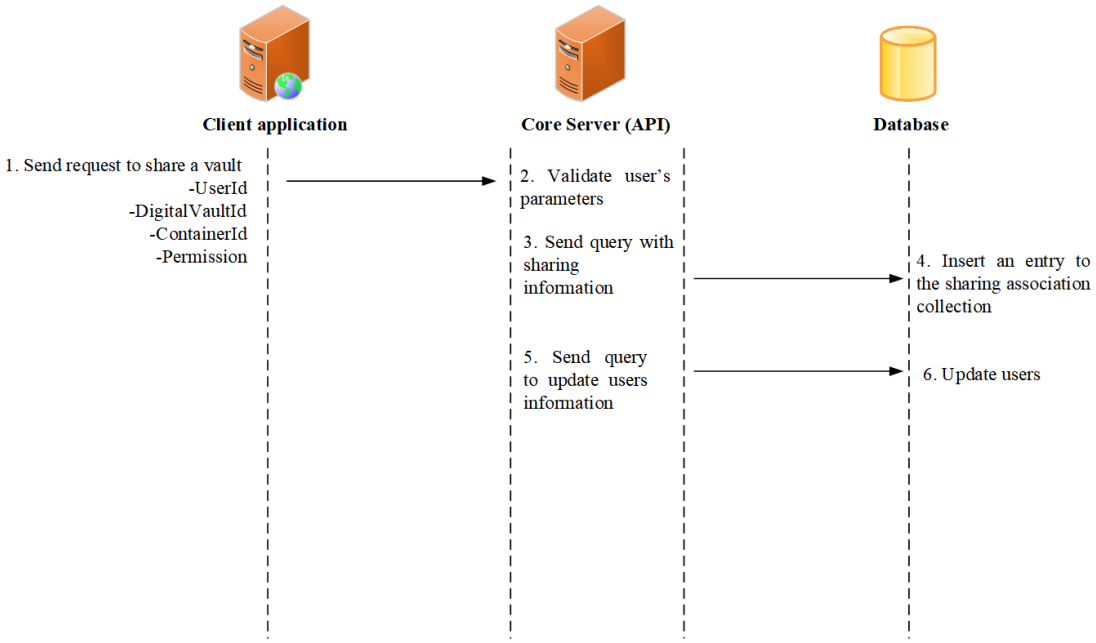

Figure 4.7: Share a digital vault with another user

4.2 Client implementation

A simple web client with GUI using the MVC paradigm and Microsoft .Net technology was designed and implemented. This interface allows the user to make calls to the Server API in a user-friendly way. However, any entity who will use this vault can develop and implement its own client.

A user that uses the web client is able to:

- Create an account
- Update the account password
- Login into the system
- Create new containers
- Set and update the master key password of a container
- Upload files
- Remove files
- Share containers with other users using different roles
- Check the journal
- Check the journal integrity

4.3 Functional Tests

For each stage of development, a set of unit tests was implemented. A unit test is a test in which the tester knows from the begging what should be the result of the test. That way, it can be assumed that it works similar to an *assertion* method. Also, some of these tests were performed in a row, in order to simulate a real-world situation.

```
[TestMethod]
public async Task TestMethod1()
{
    await OpenstackAuthenticationPerProjectTest();

    User

    Test Container

    #region Account
    await GetAccount();
    await AccountNotFound();
    await UpdateAccount();
    await AccountMetadata();
    #endregion

    SwiftObject
}
```

Figure 4.8: Unit tests sequence example

```
[TestMethod]
private async Task GetAccount()
{
    try
    {
        var newAccount = await SwiftAccount.GetAccount(account: adminAccountId
        foreach (var cont in newAccount.Containers)
        {
            var containerObjects = await cont.GetObjects();
            foreach (var obj in containerObjects)
            {
                var objData = obj.GetData();
            }
        }
        var msg = await SwiftAccount.DoesAccountExists(account: adminAccountId
        Assert.AreEqual(true, msg);
    }
    catch (Exception)
    {

    }
}
```

Figure 4.9: Unit test code example

At the OpenStack communication layer, several tests were written in order to test its API and to find bugs across the .Net project. By following this secure development approach, it is possible to guarantee, that future modifications of the project wouldn't impact the good functioning of the already existing code.

OpenStack User:

- Create user
- Authentication

OpenStack Container:

- Create container
- Get container
- Update container
- Delete container
- Delete container which is not empty (first, it is necessary to remove all objects)
- Get container metadata

OpenStack Account:

- Get account
- Account not found
- Update account
- Get account metadata

OpenStack Swift object:

- Get object
- Upload object
- Update object metadata
- Delete object

Then, a set of unit tests were written to test MongoDB collections.

MongoDB:

- Insert new user
- Get user
- Get users with specific attributes
- Delete User
- Add digital vault
- Update user's digital vault collection
- Add container
- Delete container
- Delete digital vault
- Share digital vault

After having a functional communication layer with OpenStack and MongoDB, some unit tests were written in order to test system behaviour when calling both services. These tests were later used to develop the core server API layer. In these tests, for example, after creating an account at the OpenStack, a journal entry was added to the MongoDB. This way it was possible to test how asynchronous services can handle multiple requests at the same time.

4.4 Security Tests

For the security tests, the scope was the REST API server, since it's the component that is serving the client and accepts the client's input. For the purpose of the tests, the Burp Suite was used, alongside with the OWASP REST security cheat sheet, available at https://cheatsheetseries.owasp.org/cheatsheets/REST_Assessment_Cheat_Sheet.html.

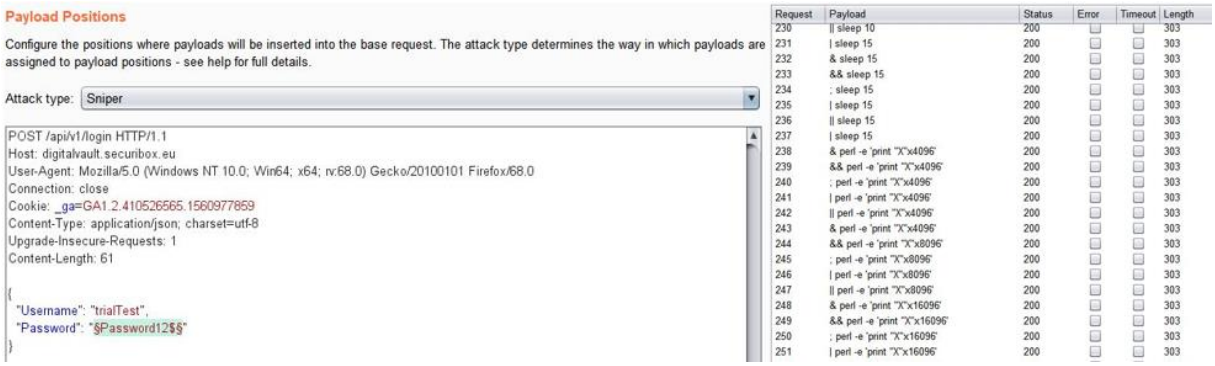

Figure 4.10: A Security test example

The scope of the tests was the core server API endpoint, detailed at the API specification chapter.

4.5 Use cases validation

All use cases from Chapter 3.3 were validated using the user's web client. However, to test the database malfunction, the project client was running in debug mode and the connection with the database was intentionally disabled and the system showed the expected behaviour, during the "blackout" and after the database went online.

Regarding the data storage, it is interesting to observe the difference between a simple *.txt* file in clear-text mode and after the vault's encryption, as it is stored at the OpenStack:

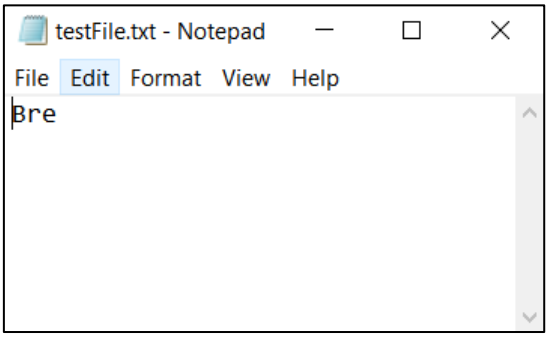

Figure 4.11: Clear-text ".txt" document

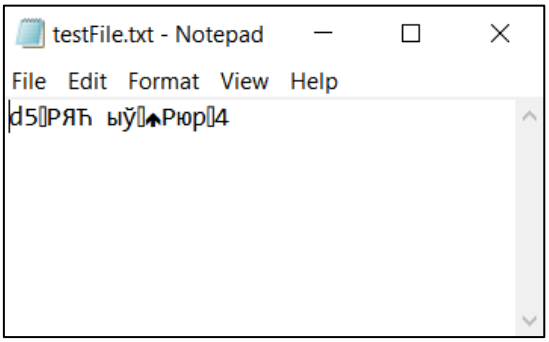

Figure 4.12: Encrypted ".txt" document

4.6 Discussion

It is important to emphasize, that due to the initial conditions of the technological Securibox's environment, some of the technologies were adopted because they were more familiar to its development team members. And its use would increase the development speed, overall project quality and leave open the opportunity for future solutions' customization and adaptations.

During the development stage, the available infrastructure was mostly virtualized, and due to that fact it was not possible to take full advantage of all used software/framework solutions, especially regarding the scalability and data replication features (OpenStack and MongoDB).

Chapter 5 Conclusion

The main goal of this project was achieved and the core functionalities of the digital vault were designed and developed with success. It were implemented in a scalable system, capable of storing large amounts of encrypted data using cloud systems where an auditable journal logs any user's or system's operations. The functionality, regarding document sharing between several users, was also implemented without violating any legal or technical requirement constraint.

This project was developed using the .Net technologies and is offered as an API system. Due to this, it is suitable for different technical and organizational environments and can be easily integrate with other Securibox products and services. Also, a .Net C# SDK was developed for the most recent version of the OpenStack Swift, since it was not possible to find any working solution at the public domain.

The functional tests results were satisfying and worked as expected. Also, by replicating the use-cases described in this document using solution's web client, it was possible to demonstrate how this solution behaves in a real world situation.

The functionalities offered in this solution may seem to be very close to other digital storage solutions. However, since the future goal is to integrate other Securibox services with it, or even apply this solution to other areas, I have a strong believe that it can have a great competitive advantage over existing solutions. It was also demonstrated that by using open-source technologies it is possible to create a fully operational software solution, compliant with standards and regulations, offering as many functionalities as a closed-source based one.

Regarding future improvements, it is expected the ParseXtract service integration into this project. Also, the functionalities related with document format conversion can be implemented by using third-party solutions. This way, this project solution can evolve into a more intelligent and complete digital vault. Other possibility, is to adapt it for a general-propose digital vault storage solution.

As a final though, I hope that this work will encourage and inspire more companies to invest in open-source technologies.

Bibliography

[1] Abadi, M., & Warinschi, B. Password-based encryption analyzed. In: Caires L., Italiano G.F., Monteiro L., Palamidessi C., Yung M. Automata, Languages and Programming. ICALP 2005. Lecture Notes in Computer Science, vol 3580, 2005. ISBN: 978-3-540-31691-6. DOI: 10.1007/11523468_54

[2] Abramova, V., Bernardino, J., & Furado, P. Which NoSQL database? A performance overview. *Open Journal of databases*. Vol. 1, issue 2, pp. 17-24, 2014, Coimbra, Portugal. ISSN 2199-3459

[3] Accorsi, R. Log data as digital evidence: what secure logging protocols have to offer?. In *Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference, COMPSAC 2009*, Seattle, Washington, United States of America, 2009. DOI: 10.1109/COMPSAC.2009.166

[4] Accorsi, R. Safekeeping Digital Evidence with Secure Logging Protocols: State of the Art and Challenges. In *Fifth International Conference on IT Security Incident Management and IT Forensics*, Stuttgart, Germany, 2009. DOI: 10.1109/IMF.2009.18

[5] Aladwani, A. M. Online banking: a field study of drivers, development challenges, and expectations. *International Journal of Information Management*, 2001, vol. 21, no.3, pp. 213–225, Kuwait. DOI:10.1016/s0268-4012(01)00011-1

[6] Arora, M. *How secure is AES against brute force attacks?*. United States of America: EE|Times, 2012. Available at https://www.eetimes.com/document.asp?doc_id=1279619

[7] Bach, M., & Werner, A. Standardization of NoSQL Database Languages. In *International Conference: Beyond Databases, Architectures and Structures*, Silesian University of Technology, Poland, 2014. DOI: 10.1007/978-3-319-06932-6_6

[8] Bellovin, S. M., & Merrit, M. Augmented encrypted key exchange: a password-based protocol secure against dictionary attacks and password file compromise. In *ACM Conference on Computer and Communications Security CCS-1*, Virginia, United States of America , pp. 244–250 , 1993. DOI:10.1145/168588.168618

[9] Boudjnah, C., Troyer, D., Kotton, G., Wienan, I., Blair, J. & Dague, S. DevStack. Accessed at https://wiki.openstack.org/wiki/DevStack in January 2018

[10] Bugiotti, F., Cabibbo, L., Atzeni, P., & Torlone, R. Database design for NoSQL systems. In *International Conference on Conceptual Modelling*, Atlanta, United States of America, pp. 223 - 231, 2014. DOI:ff10.1007/978-3-319-12206-9_18

[11] Burp Suite. Frequently asked questions. Accessed at https://portswigger.net/burp in September 2019

[12] Carrol, M., van der Merve, A., & Kotze, P., Secure cloud computing: Benefits, risks and controls. In *Conference: Information Security South Africa (ISSA)*, Johannesburg, South Africa 2011. DOI: 10.1109/ISSA.2011.6027519

[13] Cecurity corporation. Proof, Exchange, Archiving – Electronic Archival Storage. Accessed at https://www.cecurity.com/en/products/pea in November 2018

[14] Chacon, S., & Straub, B. *Pro Git*. [epub, pdf, mobi]. 2nd Edition. United States of America: Software Freedom Conservancy, 2014. Available at: https://git-scm.com/book/en/v2

[15] Chen, Z., Guo, S., Duan, R., & Wang, S. Security analysis on mutual authentication against man-in-the-middle attack. In *First International Conference on Information Science and Engineering*, Nanjing, China, 2009.DOI: 10.1109/ICISE.2009.1051

[16] CNIL. Commission Nationale de l'Informatique et des Libertés. Accessed at http://www.cnil.fr in Sepetember 2017

[17] Crump, G. *OpenStack Swift – Should enterprise customers DIY or use SwiftStack?*. Switzerland: Storage Switzerland, LLC, 2015. Accessed at https://storageswiss.com/2015/01/22/should-enterprise-customers-diy-or-swiftstack/ in January 2018

[18] FedISA, *Coffre-fort électronique: Livre Blanc*. Version 2. Luxembourg: Fedisa Luxembourg, 2014. Available at: https://www.leslivresblancs.fr/livre/informatique-et-logiciels/archivage-electronique/coffre-fort-electronique-version-2

[19] Fiddler. Telerik Fiddler. Accesses at https://www.telerik.com/fiddler in February 2019

[20] Floh, A., & Treiblmaier, H. What Keeps the E-Banking Customer Loyal? A Multigroup Analysis of the Moderating Role of Consumer Characteristics on E-Loyalty in the Financial Service Industry. SSRN Electronic Journal. 2006,Austria. DOI:10.2139/ssrn.2585491

[21] GDPR. EUGDPR: EU General Data Protection Regulation. Accessed at http://www.eugdpr.org, European Union in January 2017

[22] Hiltgen, A., Kramp, T., & Weigold, T. Secure Internet banking authentication. *IEEE Security & Privacy Magazine*, vol.4, no.2, pp. 21-29, 2006, ISSN 1540-7993. DOI: 10.1109/msp.2006.50

[23] IANA. *Hypertext Transfer Protocol (HTTP) Status Code Registry*. United States of America: IANA, 2018. Available at: https://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml, United States of America

[24] ISO/IEC 27002:2013. Information technology – security techniques – code of practice for information security controls. 2nd edition, 2013

[25] Jayawardhena, C., & Foley, P. Changes in the banking sector – the case of Internet banking in the UK. *Internet Research: Electronic Networking Applications and Policy*, 2000, vol. 10, no. 1, pp. 19–31, United Kingdom, ISSN 1066-224. DOI:10.1108/10662240010312048

[26] Maarch Courrier. Maarch solution wiki: Document management system. Accessed at https://sourceforge.net/p/maarch/wiki/Home/ in October 2017

[27] Marquardt, T. L. *ASP.NET Thread Usage on IIS 7.5, IIS 7.0, and IIS 6.0*. United States of America: Microsoft Corporation, 2007. Available at: https://blogs.msdn.microsoft.com/tmarq/2007/07/20/asp-net-thread-usage-on-iis-7-5-iis-7-0-and-iis-6-0/

[28] McDonald, C. *Data Modelling Guidelines for NoSQL JSON Document Databases*. United States of America: MAPR, 2017. Available at: https://mapr.com/blog/data-modeling-guidelines-nosql-json-document-databases/

[29] Mishra, A., Mathur, R., Jain, S., & Rathore, J. S. Cloud computing security. *International Journal on Recent and Innovation Trends in Computing and Communication*. vol. 1, no. 1, pp. 36-39, 2013, ISSN2321-8169

[30] MongoDB. Introduction to MongoDB. Accessed at https://docs.mongodb.com/manual/introduction/ in May 2018

[31] MongoDB. *Perform Two Phase Commits*. United States of America: MongoDB Inc., 2018. Available at: https://docs.mongodb.com/v3.4/tutorial/perform-two-phase-commits/

[32] NF 461. Norme française: Règles de certification de la marque. AFNOR, 2012

[33] NF Z42-013. Norme française: Spécifications relatives à la conception et à l'exploitation de systèmes informatiques en vue d'assurer la conservation et l'intégrité des documents stockés dans ces systèmes. AFNOR, 2009

[34] NF Z42-020. Norme française: Spécifications fonctionnelles d'un composant Coffre-Fort Numérique destiné à la conservation d'informations numériques dans des conditions de nature à en garantir leur intégrité dans le temps. AFNOR, 2012

[35] Nidhra, S., & Dondeti, J. Black box and white box testing techniques – a literature review. *International Journal of Embedded Systems and Applications (IJESA)*. vol.2, no.2, pp. 29-50, 2012. DOI : 10.5121/ijesa.2012.2204

[36] NOR CNIX1402990X. Délibération nº 2014-017 du 23 janvier 2014 portant adoption d'un référentiel pour la délivrance de labels en matière de services de coffre-fort numérique. Journal official de la République Française, 2014

[37] Opensource.com. What is OpenStack?. Accessed at https://opensource.com/resources/what-is-openstack in September 2017

[38] OpenStack. Open source software for creating private and public clouds. Accessed at https://www.openstack.org in October 2017

[39] Securibox. ParseXtract: The data extraction toolkit. Accessed at https://px.securibox.eu in June 2018

[40] Securibox. Securibox: Empowering through data aggregation. Accessed at https://www.securibox.eu/about.html in September 2017

[41] Sun, C., & Sosič, R. Optional locking integrated with operational transformation in distributed real-time group editors. In *Proc. of ACM 18th Symposium on Principles of Distributed Computing*, Atlanta, United States of America, 1999

[42] Wagner, J. Review: *Postman Client Makes RESTful API Exploration a Breeze*. United States of America: Programmable Web, 2014. Available at https://www.programmableweb.com/news/review-postman-client-makes-restful-api-exploration-breeze/brief/2014/01/27

Appendix

Database collections

The following documents contain data that was used during the development.

Table 0.1: Database Journal collection document structure


```
{
 "_id" : ObjectId("5cf6f8dba4aee14ee8247b9a"),
 "username" : "trialTest",
 "dvUserId" : "736c8c7d39834e6090ec85753f39585b",
 "fileId" : "",
 "swiftAccountId" : {
  "_csharpnull" : true
 },
 "digitalVaultId" : {
  "_csharpnull" : true
 },
 "containerId" : "",
 "action" : "new OpenStack user created",
 "actionDate" : ISODate("2019-06-04T23:03:55.467Z"),
 "code" : 0,
 "message" : "",
 "description" : "",
 "additionalInfo" : [],
 "hashAlg" : "SHA512",
 "hash" :
"Q55/Xe3i86LupubUnOT5W+mlH5B8Ff0EvgtitR0/lXrxoR6R/OGt7n8p7FFdfpxB9FcxtkPJbK8Q
Q6hDyP02Qg==",
 "hashPrevious" : ""
}
```


Table 0.2:Database MasterKeyCollection document structure


```
{
 "_id" : ObjectId("5cf6f91da4aee14ee8247baa"),
 "digitalVaultId" : "5cf6f8dba4aee14ee8247b9c$newContainer",
 "salt" : "61jzIw1ydJEQqOQXYlEFHGubkgad3nBjjXictXbfPGw=",
 "initializationVector" : "dYkFYpn2b0E/V+s4",
 "masterKey" :
"OSFkl+wPXdmyg1FjlqJlJwRpNgHRBU2CoSS+n4xJ6ZxiQhWby/Gv58uMkG0hW6lB",
 "controlHash" :
"b6yKupaLQAY2724x+2d2iqCSqQNxYf/nsheZV+BpRuGGxSjczxjTLNkehVFZOEI1JN+VVBzh
8iE2HEcqb8VA7Q=="
}
```

Table 0.3: Database OpenStackAccount collection document structure


```
{
  "_id" : ObjectId("5cf6f8d8a4aee14ee8247b99"),
  "key" :
"hjQNt9ajhy7Sz3Wkzb673C4OsQFKBJcEHXdV29gc24k=L9K2DE4DKxt+MeN7pGgX1iLvPu6N
+CcPO4iowOhQPptc71BwE8VL98vIJdjw3bsN++Qe8Uz+LrdWznpTUxxsCg==",
  "thumbprint" : null,
  "username" : "trialTest"
}
```


Table 0.4: Database SaltCollection document structure


```
{
  "_id" : ObjectId("5cf6f8dba4aee14ee8247b9d"),
  "dvUserId" : "736c8c7d39834e6090ec85753f39585b",
  "salt" : "In1q+oO4vd59XkZ2gTv/2IUBeMOpzwF3LSDQ8a7qMek="
}
```


Table 0.5: Database SharedCollection document structure


```
{
  "_id" : ObjectId("5cf94e70a4aee1576076f0dd"),
  "digitalVaultId" : "5cf6f964a4aee14ee8247bb3",
  "sharings" : [
    {
      "containerId" : "testcontainer",
      "entries" : [
        {
          "dvUserId" : "736c8c7d39834e6090ec85753f39585b",
          "permission" : "Member"
        }
      ]
    }
  ]
}
```


Table 0.6: Database TwoStepCommit document structure


```
{
  "_id" : ObjectId("5cf6f8dba4aee14ee8247b9b"),
  "dvUserId" : "736c8c7d39834e6090ec85753f39585b",
  "flag" : false,
  "timestamp" : Timestamp(1567788264, 1)
}
```

Table 0.7: Database User collection document structure


```
{
  "_id" : ObjectId("5cf6f8dba4aee14ee8247b9e"),
  "dvUserId" : "736c8c7d39834e6090ec85753f39585b",
  "name" : "Trial Test",
  "isEnabled" : false,
  "authType" : "Pswd",
  "apiVersion" : "dev",
  "username" : "trialTest",
  "passwordHash" :
"z9kLLYatNcnUPwpotp5NCWvQ3zL0QawcmPAzkD8Kknt1Cif4E0occF0Y8Jyk2FStFoMuvKwIp
bRnE5+kKDddsQ==",
  "hashAlg" : "Sha512",
  "modificationdate" : ISODate("2019-06-04T00:00:00.000Z"),
  "creationdate" : ISODate("2019-06-04T23:03:55.581Z"),
  "timeStamp" : Timestamp(363, 616307133),
  "tokenIssuer" : "securibox",
  "digitalVaults" : [
    {
      "digitalVaultId" : "5cf6f8dba4aee14ee8247b9c",
      "digitalVaultName" : "main",
      "swiftNodes" : [
        {
          "swiftAccountId" : "08a1741f30fc48f6bc0d7bac28e05885",
          "containerId" : "main",
          "permission" : "3cb6e9de9b0342efbdb44a23707798b7"
        }
      ],
      "digitalVaultOwner" : "736c8c7d39834e6090ec85753f39585b"
    },
    {
      "digitalVaultId" : "5cf6f964a4aee14ee8247bb3",
      "digitalVaultName" : "5d04a805cf5f45598bf0e5ce886486fd",
      "swiftNodes" : [
        {
          "swiftAccountId" : "ecabbf295fcb41029e0e3ca0ab9c0e00",
          "containerId" : "testcontainer",
          "permission" : "5b1bfbcdc0eb42eab1823fc015cbe7b1"
        }
      ],
      "digitalVaultOwner" : "5d04a805cf5f45598bf0e5ce886486fd"
    }
  ],
  "certificates" : []
}
```

Swagger API specification

The presented file was originally written in YAML using Swagger Editor. However, due to its size it was converted to JSON. The Swagger editor offers the possibility to convert JSON back to YAML automatically. The digitalvault.securibox.eu host, presented at the document, is local and is only available at the Securibox intranet.

{"swagger":"2.0","info":{"description":"API for DigitalVault","version":"1.0.0","title":"Securibox DigitalVault","contact":{"email":"fc50118@alunos.fc.ul.pt"}},"host":"digitalvault.securibox.eu","basePath":"/api/v1","schemes":["http","https"],"tags":[{"name":"Containers"},{"name":"Files"},{"name":"User"},{"name":"Journal"}],"securityDefinitions":{"BasicAuth":{"type":"basic"}},"paths":{"/container":{"post":{"tags":["Containers"],"summary":"Create new container.","description":"A new container will be created for user. Also, an swift container is made.","parameters":[{"in":"path","name":"containerId","required":true,"type":"string","minimum":1,"description":"container name."}],"produces":["string"],"responses":{"200":{"description":"OK"},"400":{"description":"Internal server error"}}},"delete":{"tags":["Containers"],"summary":"Delete user container","description":"delete user's container","parameters":[{"in":"query","name":"digitalVaultId","required":true,"type":"string","minimum":1,"description":"user's digital vault id."},{"in":"query","name":"containerId","required":true,"type":"string","minimum":1,"description":"digital vault's container id."}],"produces":["string"],"responses":{"200":{"description":"OK"},"400":{"description":"Internal server error"}}}},"/digitalvaults/{digitalVaultId}":{"get":{"tags":["Files"],"summary":"Get the list of files associated with a specific digital vault","description":"Get files from the 'main' container","parameters":[{"in":"path","name":"digitalVaultId","required":true,"type":"string","minimum":1,"description":"digitalVault id"}],"produces":["json"],"responses":{"200":{"description":"OK"},"400":{"description":"The digitalVaultId is wrong! Internal server error"}}},"post":{"tags":["Files"],"summary":"Upload a file to the main container","parameters":[{"in":"path","name":"digitalVaultId","required":true,"type":"string","minimum":1,"description":"digital vault Id"},{"in":"header","name":"metadata","required":false,"type":"array","items":{"type":"string"},"description":"Each metadat should start with X-Object-Meta-"},{"in":"header","name":"masterKey","required":true,"type":"string","description":"the encryption key will be based on the masterKey"},{"in":"header","name":"x-dv-file-idcustom","required":true,"type":"string","description":"file name"}],"produces":["string"],"responses":{"200":{"description":"OK"},"400":{"description":"The digitalVaultId is wrong! Internal server error You don't have permission to upload to this vault."}}}},"/digitalvaults/{digitalVaultId}/containers/{containerId}":{"get":{"tags":["Files"],"summary":"Get the list of files associated with a specific digital vault","description":"Get files from the 'containerId' container","parameters":[{"in":"path","name":"digitalVaultId","required":true,"type":"string","minimum":1,"description":"digital vault Id"},{"in":"path","name":"containerId","required":true,"type":"string","minimum":1,"description":"Container

Id"}],"produces":["application/json"],"responses":{"200":{"description":"OK"},"400":{"description" :"The digitalVaultId is wrong! Internal server error"}}},"post":{"tags":["Files"],"summary":"Upload a file to an containerId","parameters":[{"in":"path","name":"digitalVaultId","required":true,"type":"string","mi nimum":1,"description":"digital vault Id"},{"in":"path","name":"containerId","required":true,"type":"string","minimum":1,"description":"c ontainer Id"},{"in":"header","name":"metadata","required":false,"type":"array","items":{"type":"string"},"de scription":"Each metadat should start with X-Object-Meta-"},{"in":"header","name":"masterKey","required":true,"type":"string","description":"the encryption key will be based on the masterKey"},{"in":"header","name":"x-dv-file-idcustom","required":true,"type":"string","description":"file name"}],"produces":["string"],"responses":{"200":{"description":"OK"},"400":{"description":"The digitalVaultId is wrong! Internal server error You don't have permission to upload to this vault."}}}},"/digitalvaults/{digitalVaultId}/files":{"get":{"tags":["Files"],"summary":"Get file's content","parameters":[{"in":"path","name":"digitalVaultId","required":true,"type":"string","minimu m":1,"description":"digital vault Id"},{"in":"path","name":"fileId","required":true,"type":"string","minimum":1,"description":"File Id"}],"produces":["application/octet-stream"],"responses":{"200":{"description":"OK"},"400":{"description":"The digitalVaultId is wrong! Internal server error"}}},"delete":{"tags":["Files"],"summary":"Delete a file from the digital vault main container","parameters":[{"in":"path","name":"digitalVaultId","required":true,"type":"string","mini mum":1,"description":"digital vault Id"},{"in":"path","name":"fileId","required":true,"type":"string","minimum":1,"description":"File Id"}],"produces":["application/octet-stream"],"responses":{"200":{"description":"OK"},"400":{"description":"The digitalVaultId is wrong! Internal server error"}}}},"/digitalvaults/{digitalVaultId}/containers/{containerId}/files":{"get":{"tags":["Files"],"su mmary":"Get file's content","parameters":[{"in":"path","name":"digitalVaultId","required":true,"type":"string","minimu m":1,"description":"digital vault Id"},{"in":"path","name":"containerId","required":true,"type":"string","minimum":1,"description":" Container Id"},{"in":"path","name":"fileId","required":true,"type":"string","minimum":1,"description":"File Id"}],"produces":["application/octet-stream"],"responses":{"200":{"description":"OK"},"400":{"description":"Internal server error"}}},"delete":{"tags":["Files"],"summary":"Delete a file from a specific container","parameters":[{"in":"path","name":"digitalVaultId","required":true,"type":"string","mini mum":1,"description":"digital vault Id"},{"in":"path","name":"containerId","required":true,"type":"string","minimum":1,"description":" Container Id"},{"in":"path","name":"fileId","required":true,"type":"string","minimum":1,"description":"File Id"}],"produces":["application/octet-stream"],"responses":{"200":{"description":"OK"},"400":{"description":"Internal server error"}}}},"/digitalvaults/{digitalVaultId}/files/metadata":{"get":{"tags":["Files"],"summary":"Get file's

*metadata","parameters":[{"in":"path","name":"digitalVaultId","required":true,"type":"string","mini mum":1,"description":"digital vault Id"},{"in":"path","name":"fileId","required":true,"type":"string","minimum":1,"description":"File Id"}],"produces":["application/json"],"responses":{"200":{"description":"OK"},"400":{"description" :"file not found Internal server error"}}}},"/digitalvaults/{digitalVaultId}/containers/{containerId}/files/metadata":{"get":{"tags":["F iles"],"summary":"Get file's metadata","parameters":[{"in":"path","name":"digitalVaultId","required":true,"type":"string","mini mum":1,"description":"digital vault Id"},{"in":"path","name":"containerId","required":true,"type":"string","minimum":1,"description":" Container Id"},{"in":"path","name":"fileId","required":true,"type":"string","minimum":1,"description":"File Id"}],"produces":["application/json"],"responses":{"200":{"description":"OK"},"400":{"description" :"file not found Internal server error"}}}},"/sharevault":{"post":{"tags":["User"],"summary":"Share digital vault or container with other user","description":"Share digital vault or just a container with other user.","parameters":[{"in":"query","name":"UserId","required":true,"type":"string","minimum":1,"m aximum":1,"description":"Digital vault will be shared with the UserId"},{"in":"query","name":"DigitalVaultId","required":true,"type":"string","minimum":1,"maxi mum":1,"description":"Digital vault Id"},{"in":"query","name":"ContainerId","required":true,"type":"string","minimum":1,"maximum":1, "description":"Container Id"},{"in":"query","name":"Permission","required":true,"type":"string","minimum":1,"maximum":1," description":"Permission possible values are *:* Admin, Member, Service"}],"produces":["string"],"responses":{"200":{"description":"OK"},"400":{"description":"Per mission not found User with this ID doesn't exist"}}}},"/deleteshare":{"post":{"tags":["User"],"summary":"Remove the sharing from a specific user","description":"Remove the rights of access to a specific vault or a container from a specific user","parameters":[{"in":"query","name":"UserId","required":true,"type":"string","minimum":1,"m aximum":1,"description":"Digital vault will be shared with the UserId"},{"in":"query","name":"DigitalVaultId","required":true,"type":"string","minimum":1,"maxi mum":1,"description":"Digital vault Id"},{"in":"query","name":"ContainerId","required":true,"type":"string","minimum":1,"maximum":1, "description":"Container Id"}],"produces":["string"],"responses":{"200":{"description":"OK"},"400":{"description":"Permissi on not found User with this ID doesn't exist"}}}},"/listshareusers":{"post":{"tags":["User"],"summary":"Check who has access to a specific share","description":"Check who has access to your vault","parameters":[{"in":"query","name":"DigitalVaultId","required":true,"type":"string","minimu m":1,"maximum":1,"description":"Digital vault Id"}],"produces":["string"],"responses":{"200":{"description":"OK"},"400":{"description":"Permissi on not found User with this ID doesn't exist"}}}},"/register":{"post":{"tags":["User"],"summary":"Register new user","description":"Create new user. An OpenStack will be also automatically created.","parameters":[{"in":"query","name":"Name","required":true,"type":"string","minimum":1, "description":"User's name"},{"in":"query","name":"Username","required":true,"type":"string","minimum":1,"maximum":*

1,"description":"User's username"},{"in":"query","name":"Password","required":true,"type":"string","minimum":1,"maximum":1,"description":"User's password"}],"produces":["string"],"responses":{"200":{"description":"OK"},"400":{"description":"Internal server error"}}}},"/updatepassword":{"post":{"tags":["User"],"summary":"Update user's password","description":"A new password will be set to the account","parameters":[{"in":"query","name":"PasswordOld","required":true,"type":"string","minimum":1,"description":"User's current password"},{"in":"query","name":"Password","required":true,"type":"string","minimum":1,"maximum":1,"description":"User's new password"}],"produces":["string"],"responses":{"200":{"description":"OK"},"400":{"description":"Internal server error"}}}},"/getuserinfo":{"get":{"tags":["User"],"summary":"Retrieve user information","description":"Retrieve currently logged user information.","produces":["string"],"responses":{"200":{"description":"OK"},"400":{"description":"Internal server error"}}}},"/login":{"post":{"tags":["User"],"summary":"Login with username credentials","description":"Can be used to validade if user credentials are ok","parameters":[{"in":"query","name":"Username","required":true,"type":"string","minimum":1,"maximum":1,"description":"User's username"},{"in":"query","name":"Password","required":true,"type":"string","minimum":1,"maximum":1,"description":"User's password"}],"produces":["string"],"responses":{"200":{"description":"OK"},"400":{"description":"Internal server error"}}}},"/deleteaccount":{"delete":{"tags":["User"],"summary":"Delete account","description":"The account and all its associated files (except logs) will be removed from the system.","produces":["string"],"responses":{"200":{"description":"OK"},"400":{"description":"Internal server error"}}}},"/masterkey":{"get":{"tags":["Master Key"],"summary":"Get encryption key","description":"It's only possible to get the real encryption key if the masterKey is correct","parameters":[{"in":"query","name":"Password","required":true,"type":"string","minimum":1,"maximum":1,"description":"MasterKey"},{"in":"query","name":"DigitalVaultId","required":true,"type":"string","minimum":1,"maximum":1,"description":"MasterKey"}],"produces":["string"],"responses":{"202":{"description":"OK"},"400":{"description":"Internal server error"},"405":{"description":"Impossible to retrieve masterKey for that DigitalVaultId"}}},"post":{"tags":["Master Key"],"summary":"Create master key for a specific digital vault","description":"Create master key that will be used to encrypt files. Note:* it will be used only as one of the parts of the encryption key.","parameters":[{"in":"query","name":"Password","required":true,"type":"string","minimum":1,"maximum":1,"description":"MasterKey"},{"in":"query","name":"DigitalVaultId","required":true,"type":"string","minimum":1,"maximum":1,"description":"MasterKey"}],"produces":["string"],"responses":{"200":{"description":"OK"},"400":{"description":"Internal server error"},"405":{"description":"Impossible to create masterKey for that DigitalVaultId"},"409":{"description":"MasterKey already exists"}}}},"/updatemasterkey":{"post":{"tags":["Master Key"],"summary":"Update master key’s password for a specific digital vault","description":"It's only possible to get the real encryption key if the masterKey is correct","parameters":[{"in":"query","name":"Password","required":true,"type":"string","minimum":1,"maximum":1,"description":"MasterKey current password"},{"in":"query","name":"NewPassword","required":true,"type":"string","minimum":1,"maximum":1,"description":"new MasterKey*

*password"},{"in":"query","name":"NewPasswordRepeat","required":true,"type":"string","minimum":1,"maximum":1,"description":"new MasterKey password"},{"in":"query","name":"DigitalVaultId","required":true,"type":"string","minimum":1,"maximum":1,"description":"MasterKey"}],"produces":["string"],"responses":{"202":{"description":"OK"},"400":{"description":"Internal server error"},"405":{"description":"Impossible to retrieve masterKey for that DigitalVaultId"}}}},"/masterkeyCheck":{"get":{"tags":["MasterKey"],"summary":"Verify if a specific digitalVault already has a masterKey encryption key","description":"The encryption key itslef is not retrieved","parameters":[{"in":"query","name":"DigitalVaultId","required":true,"type":"string","minimum":1,"maximum":1,"description":"MasterKey"}],"produces":["string"],"responses":{"202":{"description":"OK"},"400":{"description":"Internal server error"},"405":{"description":"Impossible to retrieve masterKey for that DigitalVaultId"}}}},"/verifyMasterKey":{"get":{"tags":["MasterKey"],"summary":"Verify if the masterKey password is correct for a specific digitalVault","description":"Verify if the masterKey password is correct for a specific digitalVault, without retrieving the key itself","parameters":[{"in":"query","name":"DigitalVaultId","required":true,"type":"string","minimum":1,"maximum":1,"description":"MasterKey"}],"produces":["string"],"responses":{"202":{"description":"OK"},"400":{"description":"Internal server error"},"405":{"description":"Impossible to retrieve masterKey for that DigitalVaultId"}}}},"/journalhistory":{"get":{"tags":["Journal"],"summary":"Get last journal entries for the current user","description":"By default it will only return the last entry. Max number of entries is
250","parameters":[{"in":"query","name":"limit","required":false,"type":"integer","minimum":1,"maximum":1,"description":"number of the entries"}],"produces":["application/json"],"responses":{"200":{"description":"OK"},"400":{"description":"Internal server error"}}}},"/checkjournalintegrity":{"get":{"tags":["Journal"],"summary":"Check the journal integrity","description":"All logs hash is calculated again and compared with the ones that are stored.","produces":["application/json"],"responses":{"200":{"description":"OK"},"400":{"description":"Internal server error"}}}}}*