

Masters in Informatics Engineering
Dissertation/Internship 2014/2015
Final Report

Enterprise Secure Cloud

José Ricardo Correia Miranda Ramos
jrramos@student.dei.uc.pt

Supervisors:

Pedro Pinto (WIT Software)
Cesar Teixeira (DEI)

July 2th, 2015



FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Masters in Informatics Engineering
Dissertation/Internship 2014/2015
Final Report

Enterprise Secure Cloud

José Ricardo Correia Miranda Ramos
jrramos@student.dei.uc.pt

Examiners:

Pedro Abreu
Marilia Curado

July 2th, 2015



FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Abstract

Companies are organized hierarchical structures made by teams of people sharing common goals. This social model requires a great amount of communicational skills and collaborative development capacity, trying to minimize the overhead required to update each member about the state of the ongoing projects. It is completely vital for the success of a project that all parties involved stay updated and aware about the latest developments.

The Internet growth and advances on telecommunications allowed companies to adopt management systems based on online platforms. During the last decade, new concepts such as “Cloud Computing”, allowed some companies to expand their infrastructure, add capacity on demand, or outsource the whole infrastructure, resulting in an improved flexibility, a wider choice of computing resources, and significant cost savings.

Despite all the benefits brought by this new paradigm, based on online solutions, the fact that information are stored inside an unknown infrastructure raises several security issues, namely regarding its privacy and confidentiality. In the last years, those concepts have been discussed, particularly about the exploitation of the data stored [1]. These aspects inspired some of the technical solutions of this work.

Among the most secure platforms one can identify those that adopt zero-knowledge principles. These platforms stand out by the fact that the stored contents are encrypted before being sent to the platform, according with a well established key schema. Thus, you can ensure that the remotely stored contents cannot be analyzed by external entities, providing confidentiality to that information.

This internship provides WIT Software with a new cloud storage platform, capable of storing company’s sensitive information with zero-knowledge warranties. Since this is an enterprise solution, it also provides a full set of tools capable of project, team and content management.

Keywords: Cloud Storage, Privacy, Confidentiality, Zero Knowledge, Project Management, Key Schema.

Sumário

As empresas são estruturas hierárquicas organizadas, compostas por equipas que partilham objectivos comuns. Este modelo social exige grandes capacidades comunicacionais e de desenvolvimento colaborativo, tentando minimizar a sobrecarga necessária para atualizar cada membro da equipa. É essencial para o sucesso do projeto que todas as partes envolvidas estejam contextualizadas e conscientes sobre os últimos desenvolvimentos.

Com o crescimento da internet, e conseqüente avanço das tecnologias de comunicação, as empresas começaram a adotar sistemas baseados em soluções online. Durante a última década, surgiram novos conceitos, como “Computação na Nuvem”, que permitiu às empresas expandir as suas infraestruturas, aumentar os seus recursos à medida das necessidades, ou mesmo transferir toda a sua infraestrutura para o exterior. Sendo assim, as instituições conseguiram melhorar significativamente a sua flexibilidade de recursos, ter uma escolha de aplicações mais ampla e reduzir despesas relativas às suas infraestruturas.

Apesar de todos estes benefícios trazidos por este novo paradigma computacional, baseado em soluções online, o facto que confiarmos numa infraestrutura externa, leva que sejam levantadas algumas questões de segurança, nomeadamente relativas à privacidade e confidencialidade. Nos últimos anos, algumas destas ideias foram debatidas, como por exemplo a exploração da informação armazenada[1]. Dada a relevância do tópico, estes conceitos formaram a base de inspiração para este trabalho.

Entre as plataformas mais seguras podemos identificar aquelas que adoptam princípios de “conhecimento zero”. Estas soluções destacam-se pelos seus conteúdos serem encriptados antes de serem enviados para a plataforma de armazenamento, de acordo com um esquema de chaves definido. Deste modo, podemos assegurar que os conteúdos remotos não podem ser analisados por entidades não autorizadas.

Este estágio tem como objectivo desenvolver um nova plataforma de armazenamento na nuvem para a empresa WIT Software, capaz de guardar informação sensível da empresa, assegurando garantias de “conhecimento zero”. Dado o contexto empresarial onde será introduzida, a solução será concebida de modo a fornecer um conjunto de ferramentas para de gestão de projetos, equipas e conteúdos.

Palavras Chave: Armazenamento na Nuvem, Privacidade, Confidencialidade, Conhecimento Zero, Gestão de Projetos, Esquema de Chaves.

Table of Contents

1. Introduction	19
1.1. Historical Background	19
1.2. Cloud Definition	20
1.3. Cloud Efficiency	21
1.4. Cloud Security	22
1.5. Motivation	22
1.6. Internship	23
1.7. Document Structure	24
2. State of the Art	25
2.1 Information Security	25
2.2 Security Mechanisms	27
2.2.1 Integrity	27
2.2.2 Non-Repudiation	27
2.2.3 Access Control	27
2.2.4 Database Encryption	28
2.2.5 Server Side Encryption	28
2.2.6 Client Side Encryption	28
2.2.7 Networks Encryption	28
2.2.8 Certificates	29
2.3 Cryptographic Algorithms	29
2.3.1 Hashing	29
2.3.2 Symmetric	32
2.3.3 Asymmetric	36
2.3.4 Key Derivation Functions	38
2.4 Data Deduplication	39
2.4.1 Fingerprinting	41
2.4.2 Chunking Mechanism	42
2.5 Existing Platforms	44
2.5.1 Features Description	44
2.5.2 Open Source Platforms	45
2.5.3 Close Source Platforms	49
2.5.4 Platforms Comparison Table	52
2.5.5 Costs Analysis	53
3 Implemented Approach	56
3.1 Scrum Methodology	56
3.1.1 How does it work?	56
3.1.2 Roles	57
3.1.3 Planning	58
3.1.4 User stories	58
3.1.5 Product Backlog	58
3.1.6 Sprint	59
3.2 Burndown Chart	60
3.3 Road Map	60
3.4 Risks	61
4 Implemented Architecture	63
4.1 Software Architecture	64
4.2 Key Schema	68
4.2.1 Algorithms	69
4.2.2 Entities	70
4.2.3 Data Key Schema	71
4.2.4 Content Sharing	72

4.3	Web Sockets	72
4.4	Deduplication.....	73
4.5	LDAP.....	75
4.6	Storage Model	76
4.6.1	Storage Server.....	77
4.6.2	Laptop Application.....	80
5	Final Results	83
5.1	Web Application	83
5.1.1	Project Management.....	83
5.1.2	Sharing.....	84
5.1.3	Chat.....	85
5.1.4	Auditing.....	86
5.2	Laptop Client.....	88
5.2.1	Authentication and Setup	88
5.2.2	Initial Synchronization	88
5.2.3	Live Updates	89
5.2.4	Resynchronization	91
6	Software Quality.....	93
6.1	Functional Tests.....	93
6.1.1	Acceptance Test.....	93
6.1.2	API Tests.....	94
6.1.3	Unit Tests	95
6.2	Quality Tests	96
6.2.1	Usability Tests	96
6.2.2	Software Performance.....	97
7	Conclusion Remarks and Future Steps	108
7.1	Completed Mission	108
7.2	Future Work	109
	Bibliography	111

Index of Figures

Figure 1 - Generic deduplication process [62].....	39
Figure 2 – Illustration of the phases during a Scrum methodology [91].....	57
Figure 3 - Gantt diagram describing the work done until the end of the first semester	60
Figure 4 - Gantt diagram describing the work done until the end of the second semester...	61
Figure 5 - Containers level description of the Wit Cloud platform.	65
Figure 6 – Web Application component level description. The arrow without connection establishes a communication to the servlet filter of the storage server.....	66
Figure 7 – Client Laptop component level description. The arrow without connection establishes a communication to the servlet filter of the storage server.....	67
Figure 8 – Storage server component level description.....	68
Figure 9 – Schema of the generation and storage process of the keys belonging to a single user (left) and groups (right). The links represent a relationship between the components, being the head of the arrow the target of the action, the tail represents the subject and the action is described with bold letter.....	70
Figure 10 - Schema describing the key derivation structure. The links represent a symmetric derivation relationship, where A->B means that B derive from A. The links that start or end nowhere indicate arbitrary numbers of connections to elements not shown in the figure, namely links to additional child folders and files.....	71
Figure 11 - Typical architecture of a default Spring STOMP message broker [97].....	73
Figure 12 – Illustration of the upload process. Green blocks represent processes occurred inside the client component, whereas blue elements describe operation inside the storage server.....	74
Figure 13 – Illustration of the download process. Green blocks represent processes occurred inside the client component, whereas blue elements describe operation inside the storage server.....	74
Figure 14 – Authentication process sequence diagram.....	74
Figure 15 - Relational database schema of the storage server.....	77
Figure 16 - Relational database schema of the storage server.....	82
Figure 17 - Screenshots of the interface used to manage and share contents using our platform.....	84

Figure 18 - Screenshots of the interface used to share a URL with a number of unlimited users. 85

Figure 19 - Screenshots of the developed chat, illustrating a communication between two users. 86

Figure 20 – Schema illustrating the auditing system. The blue boxes represent the elements of our cloud’s file hierarchy. The numbers over the arrows contain the creation order of the elements. The black boxes describe the operations present inside the auditing object corresponding to the blue element..... 87

Figure 21 – Screenshots of the laptop application interfaces. 88

Figure 22 – Schema illustrating the synchronization process 91

Figure 23 – Login and Dashboard pages YSlow’s overall score. The classification has a rank between A (highest classification) and F (lowest classification). 99

Figure 24 – Graphical representation of the information downloaded during the access to the login page, using a cached and not cached environments 99

Figure 25 - Graphical representation of the information downloaded during the access to the dashboard page, using a cached and not cached environments 100

Index of Tables

Table 1 - Comparison between MD5, SHA-0, SHA-1, SHA-2 and SHA-3, describing their key output size, internal state, block size, rounds, security and throughput. [39]	31
Table 2 - Comparison between DES, AES, 3DES and Blowfish, describing their key length, block size, rounds, scheme, security. [50]	35
Table 3 - Running Time, Total Chunks and Average Chunk-Size Comparisons for the BSW and the TTTD Algorithms [64]	43
Table 4 - The Maximum and Minimum Chunk-Size for the BSW and the TTTD Algorithm[64]	43
Table 5 - Comparison between platforms and checking some of their features. The “Yes” means that the feature is available for that platform, “No” otherwise.....	52
Table 6 - Cost analysis table. Description of conditions and plans offered by Wuala, Tresorit and SpiderOak.....	54
Table 7 - Acceptance tests results. This table describes the number of tests performed on each client and the results of the first and last tests. Red values represent the number of failed tests followed by the number of successful tests.	93
Table 8 – API tests results. This table describes the number of tests performed on each controller and the results of the first and last tests. Red values represent the number of failed tests followed by the number of successful tests.	94
Table 9 – Unit Tests table. This table contains all unit tests performed during the project. Each test identifies the tested class, the functionalities tested and the result obtained in the last execution.....	95
Table 10 – YSlow rules classification. This table contains the classification of the login and dashboard pages for each Yahoo!’s high performance web sites rules.	98
Table 11 - WIT Cloud Application page loading results. Each test performed 10.000 requests to the index page of the Web Application component, varying the number of concurrent connections. This tables describes information about the execution time, time per request, request per second and number of failed requests.	101
Table 12 - Own Cloud page loading results. Each test performed 10.000 requests to the index page of the Own Cloud, varying the number of concurrent connections. This table describes information about the execution time, time per request, request per second and number of failed requests.	101
Table 13 – WIT Cloud and Own Cloud upload performance results. Each row contains the size of the uploaded content and request completion time on each platform.....	103
Table 14 - WIT Cloud and Own Cloud content deletion performance. Each row contains the size of the removed content and request execution time on both platforms.....	104

Table 15 – Query performance table. This table describes all tested queries and their execution time (milliseconds). Each query was executed simultaneously with 200, 400, 800 and 1000 concurrent connections, in independent thread, requesting different information. 106

Glossary

API	An Application Programming Interface is a group of methods or access points that a software provides so that external parties can make use of their functionalities without any knowledge of the intrinsic processes.
Burndown Chart	A Burndown Chart is a graphical representation of a Team's outcome at a given point of a Sprint. It plots the evolution of the remaining points – Sprint's User Stories still opened – or the remaining hours – corresponding to the Tasks still not closed inside each User Stories.
DoD	The Definition of Done is a document that states the requirements that must be met before a Backlog User Story is marked as done.
Mockup	Regarding this project, a Mockup is both a graphical illustration of a feature's use cases and a proposal for the User Interface/User Experience of that feature in the current product.
Product Backlog	Product Backlog is an artifact that contains all the ideas for the project, in the form of User Stories. These stories are organized by the priority given by the Product Owner, and are also assigned an effort that is the estimate given by the Team Members.
Product Owner	In Scrum, the Product Owner is the person responsible for defining the User Stories that compose the Product Backlog and assign them a priority in order to conduct the Scrum Team's work. The Product Owner can change the priority of those User Stories in order to maximize the value of the work by the delivery date.
Scrum Master	Scrum Master is the responsible to help both Product Owner and Team Members. The Scrum Master helps the Product Owner to manage the Product Backlog, and helps the Scrum Team to mark all the User Stories in a Sprint as done, by removing impediments and keeping them focused and productive.
Sprint	A development Sprint is a fixed implementation period, where the Team Members focus their effort in developing the features that match the User Stories they compromised to. Sprint duration is usually between one to four weeks; in this project, the each Sprint lasts two weeks.

Sprint Backlog

Sprint Backlog is composed by the User Stories that are selected by the Team Members to be accomplished along a Sprint. Each of these User Stories is divided into smaller tasks, which details the steps the Team must take in order to fulfill a story.

Task

A Task is a smaller assignment, subset of a User Story, which states something that must be implemented as part of that feature. The tasks for each User Story may be defined when that story is assigned to a development Sprint, defining specifically what needs to be done to mark the story as closed.

Team Member

The Development Team Members are responsible to create the incremental changes on the product that match the User Stories assigned to each development Sprint. Team Members forecast the User Stories that they can deliver at the end of the Sprint, and also decide how they will be implemented.

User Story

A User Story defines a possible user action with the product in order to state a project requirement in the everyday language. A User Story can comprise multiple Tasks that define meticulously what must be created by the development Team Members in order to mark the story as done.

Acronyms

3DES	Triple Advanced Encryption Standard
AES	Advanced Encryption Standard
API	Application Programming Interface
CA	Certificate Authority
DES	Data Encryption Standard
DoD	Definition of Done
FIPS	Federal Information Processing Standard
GPG	GNU Privacy Guard
HMAC	Hash-based Message Authentication Code
IaaS	Infrastructure as a Service
IDEA	International Data Encryption Algorithm
MD5	Message Digest Algorithm
NaS	Network-Attached Storage
NIST	National Institute of Standards and Technology
PaaS	Platform as a Service
PBKDF2	Password-Based Key Derivation Function 2
PGP	Pretty Good Privacy
PKCS	Public Key Cryptographic Standards
PRNG	Pseudo Random Number Generation
REST	Representational State Transfer
RSA	Rivest Shamir Adleman
SaaS	Software as a Service
SHA	Secure Hash Algorithm
SSL	Secure Sockets Layer
TLS	Transport Layer Security
UI	User Interface
UX	User Experience
WS	Web Socket

1. Introduction

The present document results from the work conducted in the scope of the internship at WIT Software under the supervision of Cesar Teixeira, PhD Professor at the Department of Informatics Engineering of the University of Coimbra, and Engineer Pedro Pinto, Solution Architect at WIT Software.

This introductory chapter is organized in seven sections. The first section presents the context of this work in the area of Cloud Computing. The second, third and fourth sections introduce Cloud technologies, its impact in the industry and some security concerns regarding these new platforms. The fifth section presents the motivation that gave origin to this internship and the reasons of its importance. The sixth section contextualizes the proposed work in the company and the main goals set for this project. Finally, the seventh section describes the purpose of each chapter, describing as well all the documents that append this report.

1.1. Historical Background

In 1969, J.C.R. Licklider introduced an innovating concept, describing it as an “intergalactic computer network”. At this time, this psychologist and computer science visionary had a vision of a worldwide computer network, long before it was built, being one of the Internet pioneers. Besides the availability of an interconnected world, he forecasted the existence of a remote place, where everyone could store and create information, installing programs, accessing programs at any site. That futuristic ideas have in part become a reality. Margaret Lewis, product-marketing director at AMD, noticed “It is a vision that sounds a lot like what we are calling cloud computing”[2]. Other related concepts appeared, like the one from the computer scientist John McCarty, who proposed the idea of computation being delivered as a public utility, similar to the service bureaus which date back to the sixties.

At that time, the technology was not sufficiently developed to support the requirements demanded by this new paradigm of information interconnectivity, availability and accessibility. Some lines of code were written and the creation of a new Web 2.0 started to be announced. However, due to the technical difficulties already described, namely the reduced bandwidth, only in the nineties this concept started to gain some strength.

In 1999, the website salesforce.com was launched, which was the first company to offer enterprise applications via a simple website, and was the most important milestone in the cloud computing history[2]. Salesforce.com opened the way for both specialist and mainstream software companies to deliver applications over the Internet.

The next remarkable event was made in 2006, when Amazon announced its Elastic Compute Cloud (EC2)[2, 3] as a commercial web service that allows other companies and individuals to rent computational resources, on which can run their own computer applications. This concept allowed companies to expand their infrastructure, add capacity on demand, or outsource the whole infrastructure, resulting in greater flexibility, a wider choice of computing resources and significant cost savings.

Another important milestone came in 2009, when Web 2.0 hit its stride, with Google and other companies started to offer browser-based enterprise applications, through services

such as Google Apps[2, 4]. In the opinion of Jamie Turner, a chief technology officer at IT service provider Cobweb Solutions, the appearance of applications like Google Docs was possible due to the maturing of the virtualization technology, the development of universal high-speed bandwidth and universal software interoperability standards, such as HTML and JavaScript. Turner added, "As cloud computing extends its reach beyond a handful of early-adopter Google Docs users, we can only begin to imagine its scope and reach. Pretty much anything can be delivered from the cloud." [5].

1.2. Cloud Definition

Cloud computing, often referred to as simply "the cloud", is the delivery of on-demand computing resources, everything from applications to data centers, over the Internet, often on a pay-for-use basis.

This new concept results from the evolution of existing technologies and paradigms. Its main goal is to allow users to benefit from highly available and scalable services by abstracting the infrastructure underneath. The cloud aims to cut costs, and help the users focusing on their core business instead of being impeded by IT obstacles.

The National Institute of Standards and Technology's (NIST) identifies "five essential characteristics" in the definition of cloud computing: [6]

- ***On-demand self-service:*** A consumer can request resources on demand as needed, such as storage capacity or computational needs.
- ***Broad network access:*** The platform is highly available through standard mechanisms and across several platforms.
- ***Resource pooling:*** Platform Provider's resources are adjusted dynamically to consumer demands and as they are requested, without compromising the performance and availability of the service.
- ***Rapid elasticity:*** Platform resources provisioned and released dynamically, in some cases automatically, to scale quickly and respond to the demands of the client. To the consumer this process is totally transparent without any apparent limitation.
- ***Measured service:*** Cloud systems are optimized and controlled depending on the type of service provided. All resources must be monitored and controlled to give relevant information for both provider and consumer.

1.3. Cloud Efficiency

Cloud Computing has proven itself, and many IT professionals recognized its benefits, in terms of capacity, flexibility, and availability. Cloud service providers also argue that cloud solutions are cheaper and increase the performance of the company, being clearly two of the main reasons why companies move to the Cloud.

In fact, a study [7] showed that 66% of the companies found that cloud computing has reduced their IT costs, while only 17% said it failed to do so. Another survey [8], which runs 12 datacenters throughout the United States, suggests that 65% of the companies that said they are considering the use of cloud services, 41% expect them to reduce their costs.

The following list presents the most pointed reasons to increase company's cost savings: [9]

- ***An array of services at shared costs:*** Using a Cloud solution one can have multiple options for product and solutions without spending a huge amount of money in software or hardware product. One can share all these cost with other cloud clients, skip installation and maintenance costs and reduce the IT department.
- ***More with less:*** Cloud platforms usually take care of software maintenance, capacity issues, security problems or other daily functions. This way, cloud clients can focus on productivity goals rather than spending time doing these routines.
- ***Organizational agility:*** Companies archive higher flexibility adopting cloud structures, requesting resources as they are needed, without involving huge amount of money as they were investing in new equipment.
- ***Elimination of redundancy:*** Additional staff, extra equipment and redundant data processes are costly. They contribute to unrestrained expenses that can hamper an organization's potential for growth and affect profitability.

The same study[8] also showed new information about efficiency and performance while using cloud solutions. The report says that 54% of cloud services helped to accelerate IT project implementation, while 17% begged to differ. Furthermore, 49% said cloud computing helped grow their businesses, 21% seeing no such benefit and 30% are unsure. The main aspects that justify the increasing in performance are listed below:

- ***Efficiency:*** Computing technologies can improve performance and process efficiency, remove redundancy, increase productivity and availability, and help to achieve better results.
- ***Capacity:*** As stated before, companies can request resources, as they are needed, avoiding large investments upfront.
- ***Reliability:*** Reliability is predicated on stability, continuity, elasticity and flexibility of a service, removing some inconveniences using other solutions.

1.4. Cloud Security

Some concerns about the exposition of the information inside public infrastructures, such as the Internet and the necessity to trust on external entities, such as the cloud providers, raised some questions about the cloud technology security, namely its privacy.

In 2013, Edward Joseph Snowden, an American computer professional, leaked confidential information from the National Security Agency (NSA), revealing some of the biggest threats to our privacy, since the creation of Internet. Edward worked as a former system administrator for the Central Intelligence Agency (CIA) and a counterintelligence trainer at the Defense Intelligence Agency (DIA) and later on an NSA outpost in Japan. Edward came to international attention after the disclosure of thousands of confidential documents, containing information about global surveillance programs. One of those programs, PRISM, allowed the exploitation of private information stored inside cloud platforms, such as very well known companies as Google, Yahoo and Dropbox [10, 11], having full access to users data. He discourage the use of those platforms, calling them “hostile to privacy”, recommending the use of “zero knowledge” providers[1].

“Zero knowledge” platforms provide a set of security mechanisms to prevent the exploitation of our information inside untrustable infrastructures. These new platforms respond to privacy security concerns by ciphering those contents and the necessary credentials to decrypt them. Service providers never have access our information or the keys as plaintext. So, even with physical access to the database, the attackers cannot explore the contents stored inside the platform. This feature is attracting more IT companies to adopt cloud technologies for developing theirs business.

1.5. Motivation

As stated in the previous section, companies are adopting cloud technologies to manage their businesses, increasing profits and saving costs. However, some security concerns about having its data stored outside company’s facilities prevented some communities from adopting this new kind of services, namely because of privacy issues, waiting for a platform with greater security guarantees.

As a response to these problems “zero knowledge” platforms emerged, which aim to provide their users with a full privacy service. All the information and credentials needed to run the application are encrypted before leaving client’s facilities, in order to arrive at public network infrastructure and external services completely encrypted. The cryptographic operations only take place inside trustable environment, preventing its access from undesired entities. Thus, you can ensure total privacy, avoiding the fear of exploitation and data analysis.

WIT Software identified an opportunity for a cloud storage product, capable of storing, managing and sharing information, inside a business context, providing guarantees of a "zero knowledge" server. This internship aims the development and introduction of this new product inside the company, able to answer the management demands and security needs. Thus, it can be also installed in other facilities, ensuring the same quality and safety.

1.6. Internship

The internship took place at WIT Software, a company headquartered in Coimbra, Portugal, specialized in solutions for mobile telecommunication, cloud and television operators. WIT Software has several established companies in the cloud sector as customers, and is continuously developing innovative products and ideas that can provide them a competitive edge.

The main goals for this internship can be divided into two major overlapping themes:

- **Internship:** all the knowledge and experience learned during this internship, specially about cloud solutions, security mechanism and software engineering processes;
- **Project development:** which corresponds to the implementation of all the proposed features and capacity to enrich the product.

In this first point, this internship was useful to consolidate knowledge acquired during my academic path, more specifically about software engineering processes, since the planning phase of the project, until its final product.

I also gained experience on cloud storage models, security mechanisms about untrusted remote infrastructures and web application development.

Regarding the project, the main goal is to contribute with a new software solution to the company. To accomplish this, the product designed must be fully implemented and tested. Below is a summary of the most relevant features:

- **Project Management:** enable the project manager to create and manage projects. It allows him to change security accesses and to define workspaces, groups and their members;
- **File Management:** provide the user with a complete product to navigate and share the information stored inside the cloud platform.
- **Desktop Synchronization:** synchronize all information shared between different clients and platforms.
- **Zero Knowledge:** Create a zero knowledge server, providing the user with a trustable product without the risk of some information being exposed.

As stated in the previous section, this process requires a planning strategy in order to establish goals and processes to accomplish a well-developed product. This preparation includes the requirements analysis, so all the product requirements can be identified, described, discussed, prioritized, evaluated and measured. Then, the architecture definition allow us to understand which components needed to be created or modified, how they will communicate with each other and the production of the necessary documentation to explain the adopted strategy or other useful information for future use.

1.7. Document Structure

The present document is divided in the following chapters:

2. *State of the Art:* presents a brief description of security concepts, the most common types of cryptographic algorithms and an analysis of both direct and indirect competitors.

3. *Approach:* describes the development methodology followed in this project.

4. *Architecture:* presents architecture aspects regarding the proposed solution and a technical overview about the proposed key schema.

5. *Final Results:* Presents and describes the results obtained during the internship.

6. *Software Quality:* Describes all adopted strategies to evaluate, measure and assure the quality of the software developed.

7. *Concluding remarks and future steps:* Resumes the work done during all internship.

This report is followed with three appendix documents, containing a full description of the following aspect covered in this document:

- ***Approach (Appendix A):*** Provides information about the methodology used in this project, namely the full product backlog of the product, description of each sprint and its burndown chart. Complements Chapter 3 by detailing some aspects that were only briefly addressed;
- ***Architecture (Appendix B):*** This appendix contains a more complete discussion about our architectural decisions, REST API specifications and a fully detailed version of the key schema and database schemas conceived.
- ***Software Quality (Appendix C):*** This appendix contains an extended version of the quality tests approached in this project, namely a complete description of the acceptance and API tests developed and a fully detailed section with the definition, results and strategies adopted during the usability tests.

2. State of the Art

This chapter presents a complete version of the State of the Art, which comprises the following topics:

- Information security overview, in section 2.1;
- Analysis of the most popular security mechanisms, in section 2.2;
- Analysis of the most popular encryption mechanisms, in section 2.3;
- Analysis of the most popular chunking algorithms, in section 2.4;
- Analysis of the most popular cloud storage platforms, in section 2.5.

For each of these analyses it is provided a brief description of the platform/services, as well as some statistics of registered users and Alexa's global ranking. Alexa is a subsidiary company of Amazon that provides web traffic data and other global ranking information on 30 million websites. The traffic data is based on a panel of people which is a sample of all Internet users. The ranking position gives us a good indication of the platform's popularity.

It is important to clarify four aspects before proceeding to the next sections:

- The security mechanisms and detailed cryptographic algorithms described reflect the most relevant concerns to consider while developing a zero knowledge server. Other security issues may not be described in such detail.
- After the description of the most common algorithms of same cryptographic type, a comparison table reflecting the differences among those algorithms is presented. However, despite belonging to the same type, some algorithms can have different purposes and may not be comparable. In those cases a comparison table is not presented.
- Due to the amount of platforms and costs associated, not all platforms were tested and the described features reflect the information presented in the company's official website or official manuals.
- During the platform assessment and whenever possible, the number of active users is chosen as the main success metric. When this information is not available, only the information about the Alexa's page rank is presented. This metric is not so accurate because it measures the traffic registered inside the platform's website, logging users using Alexa's toolbar, do not counting the amount of users without the bar installed or using another type of connection (such as application or mobile).

2.1 Information Security

Security is very abstract concept, with different meanings depending on the context. In information security it is usual to describe security as the practice of defending information from unauthorized access, use, disclosure, disruption, modification, perusal, inspection, recording or destruction. The study and analysis of this concept resulted in the acronym CIA that stands for Confidentiality, Integrity and Availability[12]. There is a continuous debate about extending this classic trio and some revisions have been made over the past decades. In 2002, the OECD's Guidelines for the Security of Information Systems

and Networks proposed extended this principles to nine topics [13]. In 2004, the NIST's Engineering Principles of Information Technology Security proposed 33 principles [14]. Finally, in 2002, Donn Parker, a very well known information security researcher and consultant of the University of Minnesota, proposed a well accepted alternative to that model, containing the six atomic elements [15].

In the following list are described these key concepts, and some of the possible attacks:

- ***Integrity:*** This means that information maintains accuracy and consistency over its entire life cycle [16]. It prevents any kind of modification (physical) or correctness and rationality of the system's information (logical), from the time that was legitimately created to the time it was requested, even by an unauthorized and malicious entity, or by other type of unintentional anomaly, for example during its transmission (tampering).
- ***Availability:*** Availability means that the information must be available every time it is requested. The mechanisms associated with this concept should be able to cover all components, since platforms where the information resides, the security mechanism used to provide the information and all the networks used to transmit the information. One of the possible attacks used to affect a service availability is Denial of Service [17].
- ***Authenticity:*** When we try to communicate, we should have a mechanism to ensure that both parties are who they claim to be and extend this guaranty to our files and contents.
- ***Non-repudiation:*** This mechanism ensures that any other parties involved in a transaction are who they claim to be and cannot repudiate the action. Normally in digital contexts, this principle usually covers two concerns: a service that provides the integrity and origin of data; an authentication that can be asserted to be genuine with a high level of certainty [18].
- ***Confidentiality:*** Capacity to restrict the information accesses to the authorized entities (machines, processes and people).
- ***Access Control:*** Capacity to restrict information access and put boundaries to user control. Usually it includes operations of authorization, authentication, access approval and auditing.

2.2 Security Mechanisms

The following section contains a complete description of the most relevant mechanisms used to provide a protection layer regarding some security concerns described in the previous section. Each mechanism will be described, as well as the security issue addressed. It is also possible that one security concern has more than one mechanism associated.

2.2.1 Integrity

This is one of the most common guarantees assured by cloud storage platforms. This mechanism is present in several layers of the application. We are going to describe the most popular approaches that can be divided in these three types: checksum, redundancy and RAID storage.

Checksums are a calculated summary of data portions, computed with a collision-resistant hash function and extensively used to verify data integrity. Besides it is a cheap and widely used solution, in most of the cases it is just capable to detect the anomalies, but not to correct them. Some checksum algorithms are able to recover (simple) errors by calculating where the expected error must be and repairing it. However, Krioukov et al. show that checksums, if not carefully integrated into the storage system, can fail to protect against complex failures, such as lost writing operations and misdirected writing parity[19]. Further, checksums do not protect against corruptions that happen due to bugs in software, typically in large code bases [20, 21].

Redundancy is another mechanism capable to ensure integrity and recover some errors. For example, some B-Tree file systems, such as ReiserFS[22], store page-level information in each internal page inside the B-Tree. Thus, checking this page-level information catches a corrupted pointer that does not connect pages in adjacent levels. Similarly, ext2[23] and ext3[24] use redundant copies of superblocks and group descriptors to recover from corruptions.

RAID is another popular mechanism, including several versions, but not all of those versions allow us to guaranty data integrity and recovery from our losses. Different versions of RAID are designed to tolerate different kinds of failures (e.g. RAID 5 tolerates one disk failure and RAID 6 two) and may not be possible with RAID alone to identify accurately the corrupted block. However, these mechanisms have already showed some flaws[20] and sometimes it is not possible to include the amount of necessary disks to execute them.

2.2.2 Non-Repudiation

This property is usually provided by a mechanism of digital signatures. This mechanism also mixes the integrity concept, described in the last paragraph, and provides authentication. This property usually require the use of certificates signed by a trustable Certificate Authority (CA)[18].

2.2.3 Access Control

The access control protection is provided by a set of mechanisms, usually related with authentication and managing of privileges inside the platform. In an enterprise cloud

context, we can also question about the location of our cloud system, if it is placed inside or outside the company's infrastructure.

2.2.4 Database Encryption

Database encryption gives confidentiality properties to the stored information. This can also offer other different guarantees depending on the implementation, but in all cases prevents the exposure of the information, in case of being attacked (safety fail mechanism). In certain circumstances, we can also guarantee that even the service provider cannot access data. These two properties are always true when we refer to hashing mechanisms.

Database encryption can affect different communication levels. Sometimes those operations are performed at the application level. This requires intelligence at the application level, but no additional database features. Other databases have built-in encryption functions that can be used by applications to encrypt data as it is written, both explicitly by the programmers [25] or implemented transparently [26]. An encryption application can sit between the application and database, encrypting/decrypting information as it is written and read. This last option requires buying and installing additional software, but may not require modifications to the application or database.

There is another mechanism that improves our database confidentiality, which is called data sharding. This mechanism breaks our file into chunks, stored in different places of our database. Thus, our attacker will not have a direct access to the information if the database gets exposed.

2.2.5 Server Side Encryption

Server encryption happens when a server performs cryptographic operations to provide privacy to our information. These mechanisms are usually applied when want to save our data using external storage services (untrusted storage). Other example is when a server tries to communicate transmit encrypted data to a device with limited computational power, in that case, due to this limitation, the data usually is decrypted inside the server.

2.2.6 Client Side Encryption

Client side encryption is a confidentiality mechanism where both encryption and decryption operations are performed inside the client. This type of mechanism is usually related to another concept: zero knowledge. This concept refers to the incapacity of the server obtain information about the content stored in its hard drives, even with physical access to it.

2.2.7 Networks Encryption

Network encryption (sometimes called network layer, or network level encryption) is a security process that applies crypto services at the network transfer layer - above the data link level, but below the application level. The network transfer layers are layers 3 and 4 of the Open Systems Interconnection (OSI) reference model, being responsible for connectivity and routing between two end points. Using the existing network services and application software, network encryption is invisible to the end user and operates

independently of any other encryption processes used. Data is encrypted only while in transit, existing as plaintext on the originating and receiving hosts.

One of the most popular forms of this mechanism is Secure Sockets Layer (SSL)/Transport Layer Security (TLS), commonly used to encrypt web traffic in transit. SSL is regularly used to secure web applications that transmits or collects sensitive information. There are a number of other uses for SSL/TLS encryption, including authentication for email communication between clients and servers. SSL/TLS can also be used for "tunneling" effects (VPN networks).

Other common network encryption technology is Secure Shell (SSH), which is largely used for encrypted terminal connections (replacing telnet) and encrypted file transfers (SFTP replacing FTP). Like SSL/TLS, SSH can also be used for tunneling.

Finally, IP Security (IPSec) is other regular solution to encrypt network traffic, which operates at a more basic layer than SSL or SSH and can be applied to any network traffic. However, IPSec requires a common configuration between computers, so it is generally used within a company/department rather than across the Internet.

2.2.8 Certificates

Certificates are widely used in Internet secure communications, providing a set of guaranties to our application. The certificate is an electronic document, which proves the ownership of a public key (including information about the owner), key description and a digital signature of the authority. Mechanisms that encrypt information using public keys require a public key infrastructure and a certificate authority (CA) to generate, sign and maintain all certificates of our system. Without the CA we could not have guaranties about the source of the certificate and relate the entity to its public key[18].

2.3 Cryptographic Algorithms

This section describes the functions and cryptographic algorithms studied to design the platform's security schema. For each cryptographic algorithm will be included a brief description of its purpose and introduced some of the most relevant algorithms. There are four types of cryptographic functions: hashing, symmetric, asymmetric functions and key derivation functions.

2.3.1 Hashing

Hashing mechanisms rely on a potential injective function, which means that given an input, the function should generate a unique output, based on that parameter. Since the function's output has a limited number of bits, sometimes it is possible to generate some collisions, depending on the function strength. Hash functions shall not be bidirectional; this means that we cannot be able to infer function's input parameter given the hash value.

These functions are widely used on authentication systems; in this case the stored hash value is compared with the value introduced by the user after being hashed. Thus, even the database owners cannot extract the real password value. This brings another vantage: if

the system gets compromised, the attackers cannot access to the plain value of the passwords (fail safe mechanism).

Finally, as already described, this function can also be used for checking data integrity, adding the hash value with the rest of the message.

2.3.1.1 MD5

MD5 is a widely used cryptographic hash function, created by Ron Rivest in 1991[27], with a 128-bit output, is particularly used for password store and integrity check.

The algorithm starts processing and broke up the message into 512-bit chunk blocks. The message is padded if the last block length is not multiple of 512. The 128-bit output is calculated by changing some internal states, each one with a size of 32-bits. These state changes are made during the computation of each 512-bit block, a process composed by four similar stages, called rounds. Each round is composed by a group of 16 similar operations based on a non-linear function F, modular addition and a final left rotation[28].

The difference between each round consist in the non linear function (F) used:

- $F(B,C,D) = (B \text{ AND } C) \text{ OR } (\text{NOT}(B) \text{ AND } D)$
- $G(B,C,D) = (B \text{ AND } D) \text{ OR } (C \text{ AND } \text{NOT}(D))$
- $H(B,C,D) = B \text{ XOR } C \text{ XOR } D$
- $I(B,C,D) = C \text{ XOR } (B \text{ OR } \text{NOT}(D))$

Despite some previous problems have already been discussed[29], was in 2004 that a group of researchers, in a project called MD5CRK, started with the main goal of demonstrating that MD5 is practically insecure. This group performed a birthday attack to find a collision, given the small size of the hash[30]. In the same year was also demonstrated how to create different files sharing the same hash value (chosen-prefix collision attack)[31]. After that, the use of this hash function started to be questioned for applications like SSL or digital signatures[29].

In 2005, a group of researchers was able to create two X509 [32] certificates and two PostScript documents with the same hash value. In that same year, MD5's designer Ron Rivest said: "MD5 and SHA1 are both clearly broken (in terms of collision-resistance)"[33].

In 2008, a researchers group from the U.S., Switzerland and the Netherlands, created a fake certificate authority, with the same hash value as other authentic CA, exploring some of the MD5 collisions[34]. Since then, CDU Software Engineering Institute declared: "MD5 should be considerable cryptographically broken and unsuitable for further use"[35]. Thus, the application of this hash function is being abandoned and replaced by another algorithms, such as SHA, which stills more secure until now.

2.3.1.2 SHA

SHA (Secure Hash Algorithm) is another hash function algorithm, with 4 released versions. The first version is already considered obsolete and the last does not have yet a lot of applicability. The constant necessity to improve the complexity of the algorithms and, consequently, the length of the hash function output, reacts to the computation capacities advances, which, depending on the algorithm complexity, can find some algorithm fragilities or hash values collisions.

SHA0 was published in 1993 and did not get a lot of market applicability, which was dominated by MD5. It also had some issues related with some construction flaws[36], leading to the origin of SHA1. SHA 1 was released in 1995, with a similar algorithm than the previous version, however including some corrections on the related weaknesses. This version was the most used on protocol communications, including certificates. However in 2005, some investigations demonstrated that the algorithm complexity could be substantially reduced [37], and the new version, published in 2001, SHA2, came with substantial differences on the hash function construction and own a lot of interest. Despite there have not yet been reported attacks that compromise the integrity of this last algorithm, in 2012, the National Institute of Standards (NIST), a non-regulatory agency of the United States of America, selected the Keccak algorithm, in the NIST hash function competition, to be the next version of SHA [38].

2.3.1.3 Comparison Table

Table 1 contains a comparison between the hashing algorithms previously approached in this chapter. For each algorithm we describe the output size of the function, its internal state, size of each block processed, rounds number, security classification (according to the actual state of the art) and performances.

Algorithm	Output Size (bit)	Internal State (bit)	Block Size (bit)	Rounds	Security	Throughput (MiB/S)
MD5	128	128(3*32)	512	64	Vulnerable	335
SHA-0	160	160 (5*32)	512	80	Vulnerable	-
SHA-1	160	160 (5*32)	512	80	Theoretically	192
SHA-2	223,256 or 384	256 (8*32) or 512 (8*64)	1024	54 or 80	Secure	139, 154
SHA-3	128, 192 or 256	1600 (5*5*64)	1152, 1088, 832, 576, 1344 or 1088	24	Secure	-

Table 1 - Comparison between MD5, SHA-0, SHA-1, SHA-2 and SHA-3, describing their key output size, internal state, block size, rounds, security and throughput. [39]

As stated before, MD5 is a 128-bit output size algorithm using 512 bit blocks. This algorithm turned out to be very broken with regards to collisions. In fact, with the actual state of the art, we can produce a collision in a few seconds, exploring both the reduced size of the generated hash and design of the algorithm. Despite of having the smallest output size and a reduced internal state, the performance is also one of the handicaps of using this solution. Therefore, this algorithm is no longer suitable to implement in our applications.

By the same motivations, SHA-0 is also vulnerable and should not be used or trusted anymore. SHA-1 is one of the most commonly used algorithms and continues to be a secure choice. However, some new studies indicate possibilities of new theoretical attacks. Thus, the security margin left by SHA-1 is weaker than intended and its use is no longer recommended for applications that depend on collision resistance. Although SHA-2 bears some similarities to the SHA-1 algorithm, these attacks have not been successfully extended to this new algorithm. SHA-2 is considered a secure algorithm and even improved the performance of its precedent. SHA-3 is not meant to replace SHA-2, as no significant attack on SHA-2 has been demonstrated and the algorithm turned out to be more robust than expected, but can be a good candidate in the future. Since this last algorithm was announced less than a year ago, it will take more time before we get more documentation and specifications.

2.3.2 Symmetric

Symmetric mechanisms allow us to encrypt/decrypt information with just one key. Given this feature, two machines just need a secure way to share a symmetric key to start a secure communication [40]. Symmetric functions have a huge vantage comparing with the cryptographic functions previously described: they are pretty much faster on encryption/decryption operations. So it is common to apply a hybrid approach, combining both methods of communication, symmetric and asymmetric, using the second method to share the symmetric key, used during the communication. Some of the protocols implementing this method are the already referenced TLS and PGP.

2.3.2.1 DES

DES is a symmetric-key algorithm, developed by IBM[41], published as an official Federal Information Processing Standard (FIPS) for the United States, in 1977[42]. This solution has a 56-bit key size length (8 for parity purposes) and is an archetypal block cipher that takes as input a fixed-block length of 64 bits. The algorithm begins with an initial permutation, followed by a 16 iterations cycle, ending with a final permutation in a reverse order of the first one. Before each iteration, this block is divided into two blocks of 32-bit each, processed alternately, being this process known as the Feister scheme. This schema ensures that the encryption and decryption are similar processes, but the sub keys are applied in a reverse order.

The length of the key determines the number of key combinations and, due to the increasing of the computational power and the reduced key size length, brute force attack is a very feasible way to break the algorithm. In 1999 was proved its weakness by breaking the key in just 22 hours and 15 minutes [43]. Another attempts were succeeded in a small amount of time[44] and new approaches arrived, exploiting some cipher fragilities, although unfeasible to mount in practice[45].

These reported weaknesses withdrawn DES as a standard by the National Institute of Standard and Technology (NIST)[46] and was been superseded by the AES [47]. However, 3DES is an application of this algorithm and still remains secure, which will be described in a further section.

2.3.2.2 AES

Similarly to DES, AES is also a symmetric algorithm for electronic data encryption, based on a design principle know as a substitution-permutation network. Unlike DES, this algorithm does not use Feistel networks. AES is a variant of Rijndael, which has a fixed 128-bit block size and a key size of 128, 192 or 256 bits. The key size has a direct impact on the number of iterations of the algorithm. The National Institute of Standards and Technology (NIST) considered this algorithm as U.S. Federal Information Processing Standard (FIPS), in 2001[48].

AES operates on a bi-dimensional matrix of bytes, termed states, and several steps are performed during the execution of the algorithm:

- 1) **Key expansion:** Set of keys derived from a master key using Rijndael's key schedule. These keys are composed by a set of 128 bits in a number equivalent to the number of rounds plus one.
- 2) **Initial Round:** These steps are only performed on the first round and imply the combination of each byte of the state with the block of the round key using XOR operation.
- 3) **Rounds that include four stages:**
 - i. *Bytes Substitution:* Non-linear substitution step where each byte is replaced, accordingly to a lookup table.
 - ii. *Shift Rows:* Transposition step that shifts cyclically the last three rows of the stage a certain number of steps.
 - iii. *Mix Columns:* Mixing state columns, combining the four bytes of which column.
 - iv. *Add Round Key:* One of the keys is combined with the state with a XOR operation.
- 4) **Final round includes three stages:**
 - a. Bytes Substitution
 - b. Shift Rows
 - c. Combination with the key

Due to the fragilities associated with the key length of DES algorithm, AES is considered its natural successor [47]. The smallest key size of the algorithm is 128 bits, which is nowadays considered as a secure key, comparing with the actual computational power, having a prevision of some hundreds years for breaking a single key.

2.3.2.3 3DES

3DES provides a simple solution to the key size weaknesses of the, already described, DES. It uses three normal DES keys, 64 bit length each, with 8 bit for parity

purposes, archiving a total length of 168 bits. This encryption method begins with the data encryption using the first key, then decrypt with the second key, and finally encrypt again with the third key. Despite of the great security benefits, 3DES has some performance issues comparing with the normal DES or AES, which compromise its applicability[49].

2.3.2.4 Blowfish

Blowfish is another symmetric-key based algorithm, designed by Bruce Schneier, developed to solve some of the issues detected on other methodologies, like IDEA or DES [43, 44]. It achieves high performance on encryption and decryption operations compared with the previous related methods [50]. Schneier has stated that blowfish is available on public domain and can be freely used by everyone, independently of the purposes of its application [51].

This algorithm has a fixed 64 bit block size length, using key sizes from 32 up to 448 bit length and can be described in two parts: key expansion and data encryption. The first part is more complex and delayed, usually called as slow initial key setup, but it is a strong mechanism to prevent brute force attacks - the most common on symmetric key methods. The second parts can be described as a cycle of 16 iterations, requiring the transformation of a certain group of states. [51].

Key initialization begins with the population of a few matrixes, one named P and four S-Boxes. The P matrix is composed by a set of 18 keys (P1 to P18), 32 bits each, and the remaining by a set of 256 keys, also with 32 bits (S1 to S256). S-Boxes are initially filled with the decimal part of the Pi number, where the first hexadecimal value of the first key corresponds to the first hexadecimal value of the Pi number and so on. After that, are performed a set of XOR operations between the 32 bits of the key and the 32 bits of P1, P1 is combined with the 32 bits of P2, and so on. In case of the key size is not long enough for the operation, the first bits of the key are padded to the end of the message. Finally, a 64-bit all-zero block is encrypted with the algorithm as it stands. The resultant cipher text replaces P₁ and P₂. The same cipher text is then encrypted again with the new sub-keys, and the new cipher text replaces P₃ and P₄. This continues, replacing the entire P-array and all the S-box entries.

At the end of the process were executed 512 iterations, being recommendable to store these final state values, if they would be eventually requested again. The encryption methodology is based on the Feister scheme, with 16 iterations. Finally, a final operation uses the P matrix and the remaining two entries. The S-Box matrix is also used on the algorithm, generating a 32-bit output with an 8-bit input. The decryption of the information is performed by the same algorithm, but using a reverse order of iterations.

This algorithm is known to be particularly vulnerable to an attack called reflection attack and a limited number of weak keys, but does not compromise entirely the applicability of the algorithm[52, 53].

2.3.2.5 Comparison Table

Table 2 contains a comparison between the symmetric algorithms approached in this chapter. For each algorithm we describe the key length associated, the size of the processed block, the number of rounds, schema associated, security classification (according to the actual state of the art) and encryption and decryption performances.

Algorithm	Key Length(bit)	Block Size(bit)	Rounds	Scheme	Security	Throughput (Enc/Des)
DES	56	64	16	Feister	Vulnerable	4.01/6.347
AES	128,192 or 256	128,192 or 256	10,12 or 14	Rijdael	Considerable Secure	4.174/6.452
3DES	168(3*56) or 112(2*56)	64	48	Feister	One vulnerability reported	3.45/5.665
Blowfish	32- 448	64	16	Blowfish	Theoretically	25.892/18.72

Table 2 - Comparison between DES, AES, 3DES and Blowfish, describing their key length, block size, rounds, scheme, security. [50]

As stated before, DES encrypts a 64-bit block cipher using a 56-bit size key. A 56-bit key spaces generates approximately 72 quadrillion possibilities. Even though it seems large, according to today's computing power these keys might be vulnerable to a brute force attack. By this reason, DES is no longer appropriate for security reasons and should not be adopted by any application. As stated before, 3DES is a simple variation of the prior algorithm and was a quick solution for the vulnerabilities discovered. However, since this algorithm performs two encryption and one decryption operations, it consumes three times more CPU power than its predecessor. AES outperforms 3DES both in software and hardware and replaced 3DES as the standard encryption algorithm, according to NIST. This algorithm provides more security due to larger block size and longer keys. Finally, Blowfish, the 64-bit block cipher used to replace DES algorithm. This algorithm is one of the fastest block ciphers developed to date. However, suffers from weak key problems, still no attack is known to be success.

2.3.3 Asymmetric

Asymmetric cryptographic mechanisms are more suitable to secure communication contexts between two parties, without a previous direct key exchange. In this scenario are used two keys, a public and a private key. The first key is publicly available to other machines and the second just for the owner. Thus, if we want to grant confidentiality to a message, when the machine A want to talk to the machine B, the first machine has to encrypt the message with the B machine's public key, then the B can decrypt the message using its private key. Other possibility, introducing non-repudiation and authenticity to the information, is to encrypt the message with A's private key to be decrypted with its public key. This last mechanism is normally adopted to encrypt the hash value of the message, creating a digital signature. It is also applied in protocols, such as TLS[54], PGP[55] and GPG [56].

2.3.3.1 RSA

RSA is included on the asymmetric algorithms category, created by Ron Rivest, Adi Shamir and Leonar Adleman. Like other algorithms of the same type, it uses a public key, freely available, and a private key, secretly stored. The RSA design operates on three main steps: key generation, encryption, and decryption.

- **Key Generation:** Two distinct prime numbers are chosen, p and q , and, for security reasons, they should be considerably big, with a similar bit length and randomly chosen; then we calculate the product of these numbers, n , which will be used as the modulus for both public and private keys; compute $\varphi(n) = \varphi(p)\varphi(q)$, where φ is Euler's totient function; pick a number between $1 < e < \varphi(n)$, where e and $\varphi(n)$ are coprimes. This number will be used as the public key exponent and, despite being more efficient, should not have a small bit length and Hamming weight; finally we compute $d \equiv e^{-1} \pmod{\varphi(n)}$, where d is the private key exponent.
- **Encryption:** An element of a communication, A , receives the public key of other entity, B . When A want to send a message to B , assuring confidentiality to the message, the message has to be translated to a certain amount of numbers lower than a n value previously calculated, which can be easily achieved using a padding scheme. After that we just need to calculate $c = m^e \pmod{n}$.
- **Decryption:** The previous formula is applied to decrypt the message, but this time with the private key exponent (d).

As you could realize on the key generation section, components p , q and $\varphi(n)$ should be kept in secret to not compromise the private key.

The security associated to this algorithm is related with two mathematical problems: factoring large number and RSA problem. The most promising approach to solve both problems is through the factorization of the prime number n in p and q elements, calculating the remaining parameters. There was not yet found a polynomial formula to solve to the factorization of large numbers, however the opposite cannot also be proven. There were released other formulas to solve this problem, like the extended Riemann Hypothesis, which try to find d from n and e . However, its complexity is similar to the previous approach[57].

In 2010, the largest number factorized was 768 bit length and the whole process took around two years, by a state of the art distributed implementation, which used hundreds of computers[58]. Nowadays, the normal key size length is around a 1024 up to 2048-bit length and was not yet reported a single factorization of a key with that size. With the increasing of the computational processing power, some specialists believe to be feasible breaking 1024-bit size key in the near future [59].

2.3.3.2 Diffie-Hellman

Diffie-Hellman is one of the earliest examples of key exchanging protocols through a public infrastructure, such as the Internet. This protocol allows us to establish a shared key that will be used during the communication session, without any prior knowledge. The shared key is usually needed to perform encryption using symmetric key mechanisms.

The original implementation of this method is based on a simple mechanism. It uses a multiplicative group of integers modulo p , where p is a prime and g a prime root modulo p .

The following list resumes the fundamental steps to establish a communication between two parties using Diffie-Hellman. However, this mechanism can be extendable to more interlocutors:

1. The two members agree to use a finite cyclic group G and an element g in G .
2. One side of the communication choose a secret integer a , which is kept in secret, and then sends (g^a) .
3. The other side chooses a secret integer b , which is kept in secret, then sends (g^b) .
4. The first side computes $(g^{(b^a)})$.
5. The other side computes $(g^{(a^b)})$.

Therefore, if we increase the value of p , a and b , we also increase the possibilities of possible combinations and, consequentially, the number of prime numbers available. Note that the g cannot be that large because prime roots usually are quite numerous.

In order to assure the algorithm safety, both elements, G and g , have to be chosen properly. This phenomenon is related to the discrete algorithm problem, which consist on the computation of a and b numbers to solve the Diffie-Hellman problem, using, for example, a Pohlig-Hellman algorithm[60]. However, given the short time period of the session using the shared key, in most cases this is not an important issue.

This algorithm by itself does not provide authentication and stills vulnerable to a man-in-the-middle attack, masquerading both interlocutors. This attack can be archived by including a third element inside the communication channel and establishing two distinct key exchanges, one for the first element and other for the second element. This way, the attacker actuates as a server. His mission is to receive, decrypt, encrypt again and route all messages between these two interlocutors. To avoid this problem there are some variants, like STS protocol, capable of providing this characteristic to the mechanism.

2.3.4 Key Derivation Functions

Lastly, we analyze key derivation functions. These functions allow us to derive a group of keys from a “master key” or other piece of information, such as passwords or phrase keys. This key is usually combined with some other non-secret information to derive a new set of keys. This property allows us to create an access hierarchy to our files, with each key capable of deriving the next keys corresponding to the contents one level below. The mechanism is based on pseudo-random functions (ex: SHA or MD5) or symmetric algorithms (ex: AES).

Key derivation functions (KDF) are usually defined as KDF (Key, Salt, Iterations), where *Key* corresponds to the “master” key, *Salt*, as the public element combined with the key, and *Iterations*, the number of cycles that the sub-function will be executed.

2.3.4.1 PBKDF2

PBKDF2 is a key derivation function, succeeding the previous version PBKDF1, which could only produce derived keys up to 160 bit long. This function makes part of RSA laboratories’ Public Key Cryptography Standards (PKCS) and was also published as Internet Engineering Task Force’s RFC 2898[61].

As a key derivation function, PBKDF2 applies a pseudo-random function, such as a hashing function, cipher or HMAC, to an input or phrase key, along with an additional value (salt) and a random number of iterations. The most common pseudo-random function used is SHA1 [37], although some weaknesses already described on this report are questioning its applicability and choose to apply SHA2.

The hashing algorithm components and the number of iterations has to be considered in the system performance. The salt value, a random piece of data that is used as an additional input to a one-way function, works as an increasing of security to the hash function, avoiding rainbow tables attacks, with pre computed values. Thus, the number of combinations grow up exponentially, being recommendable to generate the salt value using a strong PRNG cryptography: the salt probability is sufficient low enough because the hash is sufficient long enough, which means with a minimum of 128 bits.

Relatively to the number of iterations, we should establish a compromise between security and performance, considering some parameters: attackers computational power; users average entropy password value; how long the attacker should not be able to obtain the password.

2.3.4.2 Bcrypt

Bcrypt is another key derivation function, designed by Niels Provos and Davis Mazières, presented to ISENIIX in 1999 and is based on the Blowfish algorithm. This algorithm works like the previous functions of this cryptographic family, with an input value, like a password or phrase key, and a salt value, iterating over a cryptographic function, in this case an adapted version of the blowfish algorithm.

Provos and Mazières used the characteristics of the already conceived blowfish and redesigned the key generation method, which was called as Eksblowfish. This change is connected to the capacity to integrate both salt and key values, in a configurable number of iterations, to generate a set of initial keys to the algorithm. The remaining operations are implemented as the original algorithm design.

2.4 Data Deduplication

Data deduplication is a mechanism for maximizing the utility of a given amount of storage. Similar to compression, deduplication encodes data to minimize disk usage. However, it is not restricted to a file set or tree. Deduplication exploits the similarities among different items to save disk space. It identifies duplicate sequences of bytes and only stores one copy of that sequence. If a chunk appears again in the same file (or in another file of the same file set), deduplication only stores a reference to this chunk instead of its actual contents. Depending on the amount and type of input data, this technique can significantly reduce the total amount of required storage.

The process to detect sequence of redundant information depends completely on the requirements of the application and characteristics of stored data. Deduplication processes breaks this files into individual pieces of small information, called chunks, and, depending on the chunking algorithm chosen, these methods can differ in the granularity of the resulting chunks as well as their processing time and I/O usage.

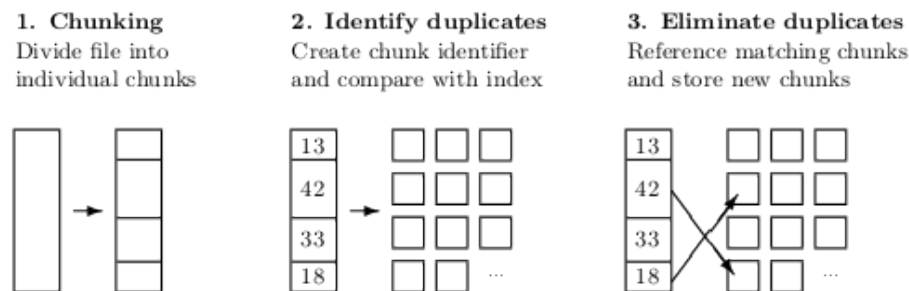


Figure 1 - Generic deduplication process [62]

Figure 1 illustrates a simple and generic case of data duplication algorithms. This process can be divided into three simple steps. First step, the file is broken into small pieces of information, called chunks. Step two, each of these chunks is hashed to create a unique identifier using a checksum algorithm, such as the described in section 2.3.1. These identifiers are then looked up in the chunk index – a database that stores chunk identifiers and their mapping to the individual files. Step 3, chunks already stored inside the system are referenced and new files are uploaded.

As we can easily understand, chunking methods that create coarse-grained chunks are usually less resource intensive than those that create fine-grained chunks. Coarse – grained chunks generate less quantity of indexes and fewer checksums that consequentially reduces the amount of lookups operations on the indexes that need to be performed. However, bigger chunks reduce the probability of redundancy detection, since the sequence of bytes is bigger, and often result in less space saving. Consequently, finer-chunks increase the odds to find similar sequences of information, but require more computational intensive operations.

The following topics explain the most commonly used chunking algorithms, describing their advantages and disadvantages:

- ***Single Instance Storage:*** This chunk file mechanism does not break the file into smaller pieces of information. Unlike the other forms of deduplication, this mechanism only eliminates redundant files. Since we are not partitioning the file, this method has low CPU usage, reduced additional overhead and index performance. SIS mechanisms are highly inefficient in saving disk space, since a file that just changes a few bytes has to be entirely stored. This algorithm is suitable to perform backup operations, but might not be the best option when we are changing big files frequently.
- ***Fixed-Size Chunking:*** This approach breaks our files into equally sized chunks. This means that the position at which the file is broken up occurs at fixed offsets. After the chunking process, these pieces of information are identified by a content-based checksum. Fixed-Size chunking methodologies require more computational requirements than the previous method, generate a lot more indexes and perform more I/O operations. However, these algorithms are more efficient detecting data redundancy (depending on the chunk size chosen). This approach fails to detect redundant data if some bytes are inserted or deleted from the file. This means that if we delete a single byte at the beginning of the file, a new completely sequence of chunks is generated.
- ***Variable-Size Chunking:*** To address the problem described in the previous algorithm, variable-sized chunking breaks files when a certain condition becomes true. This methodology analyzes the file with a sliding window and calculates a fingerprint, at every offset of it. If the calculated fingerprint satisfies the established condition, we found a breaking point. Since we have to calculate the fingerprint at every offset of the file, the fingerprint algorithm must be very efficient, usually using rolling hash algorithms that use previous checksums as input value to calculate the new value - different from the hashing algorithm of the section 2.3.1. This mechanism is more computational intensive than the previous two categories but is highly efficient detecting information redundancy, since the file is broken dynamically when the pattern is matched.
- ***File Type Aware:*** This mechanism takes advantage of the format and specifications of the underlying data. The advantage is straightforward: if the chunking algorithm is aware of the file format, breakpoints are adjusted to it and thereby potentiates better space saving. However, the most immediately disadvantage is that we must be able to understand the format of each file and, due to the enormous amount of possible formats, is not usually the most realistic scenario. Other disadvantage happens when we save the same information using different file format, we are not able to detect the similarities.

All these mechanisms have their advantages and disadvantages and we must find the most suitable option for our specifications. First of all, we have to think about the focus of our project and adjust our choices based on that information.

2.4.1 Fingerprinting

While choosing a variable file-chunking methodology, we need to address two important topics: *fingerprinting algorithm* and *chunking algorithm*.

The fingerprinting algorithm (the algorithm by which the chunk boundaries are detected) is the parameter with the biggest impact on performance. They usually discard the usage of the hashing algorithms described in the section 2.3.1 due to performance reasons. This algorithm uses values previously obtained as the input to calculate the new hash value, improving the performance but decreasing the algorithm's reliability.

The following topics give an overview over the existing well-known fingerprinting algorithms and their characteristics:

- **Rabin:** Rabin fingerprints are calculated using random polynomials over a finite field – a field that contains a finite number of elements. This algorithm receives as input for the checksum calculation an irreducible polynomial in the form of an integer, selected by the user. Since the checksums are calculated using previous values, the fingerprint can be calculated efficiently. Other property of this algorithm is the detailed mathematical analysis of its collision probability.
- **Rolling Adler-32:** A checksum algorithm invented by Mark Adler in 1995, and is a modification of the Fletcher checksum. The checksum is obtained by calculating two 10-bit long values, A and B, and concatenating their bits leading to 32-bit integer. The A parameter is the sum of all bytes in the stream plus one, and B is the sum of individual values of A from each step. The sums are done module 65521 (the biggest number smaller than 2^{16}).
- **Pretty Light and Intuitive (PLAIN):** This algorithm is based on the previous one, also based on addition operations, being faster than the Rabin. The authors of this algorithm argue that the randomization produced in Rabin is redundant, since the data on which the algorithm is applied can be assumed fairly random and by eliminating this randomization, the algorithm speed can be improved.

Our evaluation of the algorithms was based on the experiments performed by Philipp C. Heckel, during his master thesis [63]. These tests wanted to measure the number of duplicated chunks found by each one of the algorithms, based on the Syncany's expected data file types, file sizes, the amount of files to be stored as well as the expected change patterns to the file. The CPU usage was also measured and the storage space reduction.

The results show that Adler-32 is the most efficient algorithm in terms of chunking efficiency in all databases. In the performed tests, Adler-32 archived a space reduction of 93.2%, the PLAIN algorithm archived 91.1% and, finally, Rabin, the worst of all four, with 90.3% of reduction.

However, Adler-32 as the mechanism with the highest average CPU usage, with a surprisingly value of 95.1%. Rabin was in the second place with 93.5%, followed by Plain with 92.4%. In the last place, as expected, was the fixed size algorithm with 90%.

In terms of chunking duration, the fixed-size algorithm outperformed any other candidate. As the results demonstrate[63], this method took about half the time than PLAIN algorithm and a less than a third using Rabin fingerprinting. Over all datasets and test runs, the average chunking duration for algorithms with fixed-offset chunking is 4.1 seconds, for PLAIN based algorithms it is 8.3 seconds, for Adler-32 based algorithms it is 10.0 seconds and for Rabin based algorithms it is 13.7 seconds.

2.4.2 Chunking Mechanism

The chunking algorithm is completely independent from the fingerprinting algorithm. As stated before, most of the variable chunking algorithms rely on a sliding window approach and relies on the fingerprinting algorithm to determine chunk boundaries, breaking the file only if the breaking condition matches. Therefore, chunk boundaries are defined based on the probability of certain fingerprinting occurs, conditioned by the size of the window and the divisor.

In the next two topics, we describe two of the most important variable-size chunking methods:

- *Two Threshold Two Divisor (BSW)*: This algorithm was purposed for a low bandwidth network file system (LBFS) by avoiding boundary-shifting problems. This algorithm needs three main parameters: window size (W), integer divisor (D), integer remainder (R). This algorithm works as follows:

- (1) The fixed size window (W) moves one byte at one time from the beginning of the file to the end of the file.
- (2) At every position (p) of W, this algorithm uses a fingerprinting algorithm to calculate the hash value (h) for the content of the current window.
- (3) If $h \bmod D = R$, p is a breakpoint boundary and the sliding window starts from this position. Otherwise this position is not a breakpoint and the window shifts

The expected chunk size depends on the D parameter. Each shifting, the checksum has the probability of $1/D$ to satisfy the condition, and, therefore, we expect to have a breaking point at every D bytes.

- *Two Threshold Two Divisor (TTTD)*: This chunking algorithm works similarly to the previous mechanism but allows us to ensure that the size of our chunks is bounded between two values, a minimum and a maximum size. In order to satisfy this condition, the algorithm ignores all information of the sliding window until a minimum size is reached. Between the minimum and the maximum size, the chunking algorithms uses one of the previous described fingerprinting algorithms to calculate the checksum, one byte at one time, and verifies if it matches the condition settled. To handle situation that exceeds the maximum chunk size, this algorithm generated two random values d and D' to be used as the new divisor. D' is smaller than D and, by this reason, is more likely to match the condition and find a breakpoint before reaching the maximum chunk size.

Data Set	Total Running Time (sec)		Total number of Chunks		Average Chunk-Size (bytes)	
	BSW	T'TTD	BSW	T'TTD	BSW	T'TTD
#1	2910	2885	172874	182582	1040	985
#2	10568	11011	391036	481963	1629	1321
#3	617	639	24692	32364	1532	1169
#4	381	398	17803	19590	1316	1196
Average	3619	3733	151601	179125	1379	1168

Table 3 - Running Time, Total Chunks and Average Chunk-Size Comparisons for the BSW and the T'TTD Algorithms [64]

The table 3 shows the results of some experiments[64] performed to these two algorithms, using the same dataset and window size. Firstly, in all cases, the total running time and the number of chunks was bigger in T'TTD algorithm than in BSW. In average, T'TTD running time took more 3% than the recorded using BSW and produces more 18% of the chunks. These results show that tradeoffs for the T'TTD algorithm are very small.

Data Set	Max Chunk-Size (bytes)		Min Chunk-Size (bytes)	
	BSW	T'TTD	BSW	T'TTD
#1	16442	2800	48	412
#2	154075	2800	8	8
#3	97168	2800	48	68
#4	68224	2800	48	62
Average	83977	2800	38	138

Table 4 - The Maximum and Minimum Chunk-Size for the BSW and the T'TTD Algorithm[64]

However, regarding the average chunk-size, the results showed that the T'TTD algorithm is much closer to the expected size (1000 bytes) than the BSW algorithm. Table 4 shows how better T'TTD algorithm controls the size variation of the chunk. Since 2800 is the maximum threshold settled to the T'TTD algorithm, this is the value obtained during all test regarding all datasets. BSW, however, has a huge variation among the tests. This difference is large enough to affect how average chunk-size closes to the expected chunk-size. Consequently, the average amount of information required to transmit some modification is also more expensive, because it needs to send more/bigger pieces of information.

2.5 Existing Platforms

Existing enterprise cloud storage platforms provide features that increase the user experience and security guaranties. Section 2.5.1 approaches the most relevant features to be considered in our analysis.

Section 2.5.2 and 2.5.3 contain the analysis of existing cloud open source and close source storage platforms, respectively. Finally, section 2.5.4 contains a table reflecting a comparative analysis of all described products and their respective features.

2.5.1 Features Description

Enterprise cloud storage platforms usually contain a broad set of relevant features, in order to provide us a better user experience, with a complete set of management tools and utilities, as well as some security concerns, providing better security mechanisms to our information. The following listing contains some of the features that enrich the existing enterprise cloud platforms:

- **Roles:** Enterprise environments usually have roles conceptualization associated. Different roles can have different access to features or contents of a project.
- **Access Rights:** Capacity to restrict data access (Read/Write) or contents.
- **Work Groups:** Capacity to create different workstation inside the platform.
- **Link Sharing:** Theses services are capable of sharing information with others through a direct link to the file/folder. This link can be protected with a password, restrict user's accesses and even be limited on time.
- **Mail notifications:** Changes, sharings and modifications are reported to the appropriate users.
- **External storage:** We evaluate if the platform supports compatibility with external storage platforms, like Amazon S3.
- **Logging:** All changes in the workstation are recorded into a log file, accessible for some members.
- **Integrity:** Integrity aims to prevent data corruption or data loss during over its entire life cycle.
- **Non-repudiation:** We want to certificate that all changes made to a certain file be associated with a unique person.
- **Secure data transmission:** These platforms support protocols, like TLS, to reinforce data security over the network.
- **Data Sharding:** We analyze if it is possible to chunk our data and store it at different places of our database.

- **Client Side Encryption:** This feature performs some of the cryptographic functions inside the client's side of the application. This feature becomes important when we use, for example, "untrusted storage" platforms.
- **Server Side Encryption:** Cloud service providers can perform some of the encryption operations. This feature becomes important when we, for example, try to create an API to deliver contents to mobile devices with small computational capacity.
- **Zero knowledge:** Server never knows the plaintext contents of the data it is storing.

2.5.2 Open Source Platforms

In this section of the report we will describe the seven most popular open source platforms, aiming to answer to the questions:

- If there are some Open Source platforms available, why develop a new product from scratch?
- Why we cannot adopt one of the existing products?

Although there are some very complete platforms, as we can see in the following analysis, most of them cannot assure some of the desired requirements for the new product, essentially considering our security concerns.

2.5.2.1 Pydio

Pydio is an open source sharing platform, published under a AGPL license. Previously known as AjaXplorer, started its development in 2009 and in 2003 became Pydio.



This platform can be installed in Network-Attached Storage(NAS), cloud Infrastructure as a Service(IaaS) or Platform as a Service(PaaS) servers. It claims to be a reliable alternative to SaaS Boxes and drives safety and privacy of information. Pydio is also used within leading global companies including Apple, Tecnica, Nikon, Crowe Horwath and Scanomat.

This project was developed using PHP and JavaScript languages, under a modular software philosophy, which makes Pydio a particularly friendly platform to produce additional desired features. Its development was made with strategic partnerships like Red Hat and ER, to deliver scalability and high availability software, as well as military grade security. Pydio also has a partnership with Lynkoa to deliver leading edge collaborative technology.

Current number of users: Around 680,504 (community downloads) [65]

Alexa Page Rank: 99.973 (Global Rank)

Main Features: [65-68]

- Sharing
- Scalability
- Features Extensible (Plugins)
- Logging
- RESTfull API

2.5.2.2 OwnCloud Inc.

OwnCloud Inc. was inspired by other popular open source project, OwnCloud, and, although it is available without any cost for a non-commercial use, it requires a license for any commercial activity. The platform was founded in 2011 to give corporate IT greater control of their data. Pydio's American headquarters is located in Boston, but is also installed in Nuremberg, Germany.



This platform's claims to be the most efficient platforms for file sync, share and view, offering a complete set of business features for a secure and end-user-friendly way to access and share files. Existing IT tools installed to secure, track and report data can also be integrated seamlessly. OwnCloud also established some partnerships with several companies, so they can extend the existing product by providing additional features, increasing the product value.

Current number of users: Around 1.3 million users [69]

Alexa Page Rank: 10.377 (Global Rank)

Main Features:[69-72]

- Sharing
- Collaborative Editing
- Encryption (Server Side)
- Versioning and Logging
- External Storage Compatibility
- RESTfull API

2.5.2.3 Open-Xchange

Open-Xchange offers a large set of tools to support our email, contacts, media and documents, optimizing workflows and productivity in this era of information overload.



In 2012, some key members from OpenOffice development team started developing a new cloud-based enterprise tool for text processing and file managing, using HTML5 and JavaScript languages. Two years later, in 2014, this cloud feature was added to a OX App Suit product, named as OX Drive, providing file syncing and collaboration tools, both browser based and mobile application.

Platform's backend was coded using Java language and has a different deployment model from Pydio and OwnCloud.

Current number of users: Around 60 million users[73]

Alexa Page Rank: 163.713 (Global Rank)

Main Features: [73-75]

- Sharing
- RMI interface
- Modularity
- HTTP API
- External Storage Compatibility

2.5.2.4 Syncany

Syncany was announced in 2011, by Philipp C. Heckel, offering a cloud storage service, similar to Dropbox, but leading it to another level, including storage flexibility (giving a choice between private service or external providers), intelligent versioning and promising security features, like client side encryption.



The announcement just wanted to stimulate the developers interest and attract some new public attention, not being actually released. It was very well succeeded, with a lot of public's positive feedback and, since that day, Philipp C. Heckel started developing his idea and already launched some versions. So far, the last version was published in February of 2014, but still is in an early development stage.

The application was coded using Java language and, because it was announced so recently, its features are not as enriched as the previous platforms.

Alexa Page Rank: 1.193.518 (Global Rank)

Main Features: [76, 77]

- File synchronization
- Client Side Encryption
- Large Storage Compatibility

2.5.2.5 Open Stack

OpenStack is a free and open-source cloud computing software platform, constituted by several modules, released under the terms of the Apache License.



All began in 2010, as a joint project between Rackspace Hosting and NASA, intended to help organizations offer cloud-computing services running on standard hardware. Nowadays, the OpenStack Foundation, a non-profit corporation established in 2012, manages it to promote OpenStack software and its community.

OpenStack project has around 200 partnerships, including some relevant companies such as AMD, Canonical, Dell, HP, Ericsson, IBM, Oracle, Red Hat and Intel. The partnership with Canonical showed some results in 2011 with OpenStack “Bexar”, initially released for Ubuntu 11.04 and latter extended to another versions. In 2012, RetHat announced its OpenStack distribution and, in the next year, started the commercial support. In 2013, NASA gives up of being an active developer of the project, after an internal audit

citing lack of technical progresses. Finally, in 2014, HP announced HP Helion as a public cloud service, operating since 2012.

Alexa Page Rank: 20.094 (Global Rank)

Main Features: [78]

- Object and block storage
- Redundancy and scalability
- Horizontal scalability
- Rest API, including compatibility with another platform like Amazon S3 and EC2
- Logging and monitoring

2.5.2.6 Riak CS

Riak CS (Cloud Storage) is a simple and open source storage software built on top of Riak, a distributed NoSQL key-value data store.



This platform installs an object storage layer that sits on top of the Riak key-value platform, and can be used as a public and private cloud, or as reliable storage to power applications and services. Although it is freely available for private purposes, it requires a license for any commercial activities.

This company is headquartered in Cambridge, Massachusetts, and has offices in London, San Francisco, Tokyo and Washington DC. Today it counts with over 110 people today employed.

Alexa Page Rank: 10.377 (Global Rank)

Main Features: [79]

- Fault tolerance
- Restful APIs, including the Map Reduce managing and external storage platforms
- Redundancy/Replication
- Monitoring
- Horizontal scalability
- Activities logging

2.5.2.7 SparkleShare

SparkleShare is an open source client software, started in 2012, that provides cloud storage and file synchronization services. SparkleShare is comparable to Dropbox, but the cloud storage infrastructure can be installed inside a private server or a hosted solution such as GitHub and Gitorious. The advantage of self-hosting is that the user retains absolute control over their own data.



Alexa Page Rank: 500.024 (Global Rank)

Main Features: [80]

- Versioning of the information

- Tracking and syncing files edited by multiple people
- Self Hosted
- Client Side Encryption

2.5.3 Close Source Platforms

In this section of the report we will describe the five most popular close source platforms. As we will see, these platforms offer a larger set of features than the previously described, essentially concerning about our security requirements.

2.5.3.1 SpiderOak



Ethan Oberman and Alan Fairless launched SpiderOak in 2007, with a mission to dispel the myth that once the information is published on the Internet is no longer be private.

This company developed a “zero knowledge” platform, offering a central and private place to costumers place theirs data. The information remains free from surveillance by private or public entities, with data encryption operations occurring inside the client side. All keys regarding cryptographic operations are not accessible to the storage server. In 2004, Edward Snowden even recommended SpiderOak over Dropbox, after one of the largest scandals involving Internet privacy. [1]

Current number of users: Around 1 million users [81]

Alexa Page Rank: 36.307 (Global Rank)

Main Features: [81, 82]

- Zero Knowledge
- Large Compatibility
- Versioning
- File Restore
- Share Content Privately

2.5.3.2 Tresorit

Tresorit was founded in 2011 by three programmers, Istvan Lam, Szilveszter Szebeni and Gyorgy, the current company’s CEO, CTO and COO, respectively.



They belied that people have a right to their privacy, even when they collaborate and share contents online. This platform aims to provide the benefits of the most popular storage services, like Dropbox, continuing to give a transparent, fluid and easy interface to the end user but including on the background all the necessary processes to archive full privacy.

This platform function is based on tresors, which are logical containers located in the cloud, containing our encrypted data. These tresors can be seen as a simple folder, but storing data only accessible through ours keys. The information about our tresors is stored

in a roaming profile, containing all the information needed to access our content's tree, using client side encryption.

Since April 10, 2013, Tresorit has hosted a hacking contest, offering \$10,000 to anyone who can hack their data encryption mechanisms and gain access to their servers. In 2013 the company increased the reward to \$25,000, challenging top hackers from institutions like Harvard, Stanford or MIT. The current reward is \$50,000[83].

Alexa Page Rank: 192.178 (Global Rank)

Main Features: [84]

- Client Side Encryption
- Workspaces(Tresors)
- Share Content Privately
- Crypto Mechanisms
- File Versioning

2.5.3.3 Wuala

Wuala is another secure cloud storage platform, made in Switzerland, covering all the essential needs expected by a storage cloud service for professional and personal use, such as file storage, synchronizing, versioning and backup service, but also provides strong security mechanisms.



It was founded in 2007 by Caleido Inc., which is now part of LaCie, a french manufacturer of well-design external storage devices. Wuala and LaCie are joined since 2009, sharing the vision of privacy as a right to every costumer who want to use and store contents inside cloud storage platforms.

As the previous mentioned cloud storage services, it uses client side encryption, based on an existing mechanism, named Cryptree[85]. This system operates over an “untrusted storage” environment, and brings some innovating concepts of key hierarchy, key regression and lazy revocation, reducing the computational effort of some operations, like customer revocation or changing contents location.

Alexa Page Rank: 34.242 (Global Rank)

Main Features: [86, 87]

- Client Side Encryption
- Redundancy
- Share Content Privately
- Backup and Versioning
- Synchronization

2.5.3.4 Box

Box, founded in 2005 by Aaron Levie and Dylan Smith, offers cloud storage and file-sharing services that enable you to securely share and access files online.



This platform aims to provide a full set of tools capable of joining online collaboration activities and data security. The service is available in three different plans: enterprise, business and personal. Each plan has different type of features, resources and costs associated.

In 2007, Box released OpenBox, allowing costumers to access theirs contents through a web based application and services. It also interacts with other platforms, such as EchoSign, Picnik, Twitter and Myxer. Besides being a closed source platform, it has available a public API interface, allowing developers to build custom applications.

We can distinguish some of their costumers, such as General Electric, Schneider Electric, and Procter & Gamble.

Alexa Page Rank: 722 (Global Rank)

Main Features: [88]

- File Sharing
- Online Collaboration
- File Synchronization
- Administration Control
- Security among communications
- Variety of Platforms
- Open API

2.5.3.5 Egnyte



Egnyte was founded in 2007, in Mountain View, California, by a team of high-tech experts, including former Valero partners Vinnit Jain, Amrit Jassal, Rajesh Ram and Plaris Partners.

This provider offers secure enterprise file services on premises and in the cloud to thousands of customers around the world, making the data available through a complete list of devices.

This company claims to provide deployment models capable of solving any use case including cloud file sharing, private file sharing, cross-office collaboration, and fast local file access.

Egnyte currently employs more than 200 employees worldwide, and hired at the rate of two to three employees per week as of May 2014 and some of company's customers include: Nasdaq, Home Depot, Ikea and Coach.

Main Features: [89]

- Local Storage, Hybrid and Cloud Storage Solutions
- File Synchronization and Sharing
- Cross Site Synchronization
- Global Namespace

2.5.4 Platforms Comparison Table

After identifying and describing our main direct competitors, table 5 shows us a comparison between them in terms of provided features and supported security mechanisms. This comparison is extremely important once it allows us to identify the common set of tools that companies would like and what features should be implemented in order to differentiate our implementation from our competitors. Therefore, it is an opportunity to reflect how we can produce an innovate product, capable of implementing differentiating features that have real and distinctive impact on the user experience of our customers.

Platform	Roles	Access Rights	Work Groups	Link Sharing	Mail Notifications	External Storage	Logging
Pydio	YES	YES	YES	YES	YES	NO	YES
OwnCloud	YES	YES	YES	YES	YES	YES	YES
Open-Xchange	YES	YES	YES	YES	YES	NO	YES
Syncany	NO	NO	NO	YES	NO	YES	YES
OpenStack	YES	YES	YES	YES	YES	YES	YES
RiakCS	NO	NO	NO	NO	NO	YES	YES
Sparkle Share	NO	YES	YES	NO	NO	NO	YES
SpiderOak	NO	NO	NO	YES	YES	NO	YES
Tresorit	YES	YES	YES	YES	YES	NO	YES
Wuala	YES	YES	YES	YES	YES	NO	YES
BOX	YES	YES	YES	YES	YES	NO	YES
Egnyte	YES	YES	YES	YES	YES	YES	YES

Platform	Integrity	Non-Repudiation	Secure Data Transmission	Data Sharding	Client Side Encryption	Server Side Encryption	Zero Knowledge
Pydio	YES	NO	YES	NO	NO	YES	NO
OwnCloud	YES	NO	YES	NO	NO	YES	NO
Open-Xchange	YES	NO	YES	NO	NO	NO	NO
Syncany	YES	NO	YES	YES	YES	NO	YES
OpenStack	YES	YES	YES	YES	YES	YES	YES
RiakCS	YES	NO	YES	YES	NO	YES	NO
Sparkle Share	YES	YES	YES	NO	YES	YES	YES
SpiderOak	YES	YES	YES	YES	YES	YES	YES
Tresorit	YES	YES	YES	YES	YES	YES	YES
Wuala	YES	YES	YES	YES	YES	YES	YES
BOX	YES	NO	YES	YES	NO	YES	NO
Egnyte	YES	NO	YES	YES	YES	YES	NO

Table 5 - Comparison between platforms and checking some of their features. The “Yes” means that the feature is available for that platform, “No” otherwise.

It is clearly visible in table 5 that not every competitor has the capability to implement and follow all security principles described in the *section 2.1.1 – Features Description*. In average most of the platforms implement the basic security principles, such as integrity, non-repudiation and logging, but only a few can assure zero-knowledge policies and client-

side encryption. This shows that covering all these security concerns can be a good opportunity and distinctive characteristic of our product.

Analyzing these zero knowledge services, most of them lack integration with other external storage providers, such as Amazon S3 or Dropbox. In fact, they do not need or want it since they offer their own storage services and, as we can see in the next chapter, most of their payment plans and consequentially their profit, are based on the storage capacity. Nevertheless, the presence of external cloud storage providers would be a good feature to be implemented since we could have the flexibility to choose a different storage server and, consequentially, opportunity of saving costs.


Syncany is currently the only product offering these two functionalities: external storage support and zero knowledge. However, this product lacks a lot of project management features and file sharing functionalities. Thus, a product covering all approached security concerns and still offers project managing and content sharing features, could bring significant value to our customers and distinguish us from our competitors.

Comparing all these platforms, we can also easily identify those with more similarities to the characteristics that we want to implement. SpiderOak, Tresorit and Wuala are three different products, especially conceived to preserve customer's privacy without limiting the capacity of sharing information with another customers. They had the ability to join two concepts in a single product, without compromising one of them: file management and security.

2.5.5 Costs Analysis

The following table (table 6) gives you a simple and quantitative approach for analyzing the costs associated with the three platforms stated as the main competitors of our product: Wuala, Tresorit and SpiderOak. This analysis is extremely importance once it can help us to understand the costs associated adopting this kind of solutions and analyze the profitability of developing our own product.

Therefore, in this section of the report we will identify and describe all different plans offered by these three platforms and establish a term of comparison. Since all products provide the same main set of features (table 5), the most important term of our analysis will be the storage capacity and other possible restrictions/specifications of the subscription plan. Since our platform can be easily adapted to an individual backup and personal sharing system, we also describe personal subscription plans.

Platform	Conditions	Price
	Wuala Personal:	5GB:
	<ul style="list-style-type: none"> • Storage Space: 5GB/20GB/50GB/100GB and more 	0.99€/month
	<ul style="list-style-type: none"> • Maximum File Size: unlimited 	2.99€/month
	<ul style="list-style-type: none"> • Maximum Size of Links: unlimited 	50GB:
	<ul style="list-style-type: none"> • Maximum Users: 1 	5.99€/month
		100GB:
		9.99€/month



	Wuala Business: <ul style="list-style-type: none"> • Storage Space: 100GB (extensible) • Maximum File Size: unlimited • Maximum Size of Links: unlimited • Maximum Users: 5 (extensible) 	Base: 389€/year + 100GB: 249€/year + 5 users: 99€/year
	Premium: <ul style="list-style-type: none"> • Storage Space: 100GB • Maximum File Size: 5GB • Number of Devices: 5 • Maximum Size of Links: 500MB • Download limits: 50 • Maximum Users: 1 	10€/month
	Business: <ul style="list-style-type: none"> • Storage Space: 1000GB • Maximum File Size: 10GB • Number of Devices: 10 • Maximum Size of Links: 10 • Download limit: 1000 	20€/month/person
	Enterprise: <ul style="list-style-type: none"> • Storage Space: Custom • Maximum File Size: 10GB • Number of Devices: unlimited • Maximum Size of Links: 1000MB • Download limits: 1000 	40€/month/person
	SpiderOak One: <ul style="list-style-type: none"> • Storage Space: 30GB/1TB/5TB • Maximum File Size: unlimited • Maximum Size of Links: unlimited • Maximum Users: 1 	30GB: 7\$/month 1TB: 12\$/month 5TB: 25\$/month
	SpiderOak Groups: <ul style="list-style-type: none"> • Storage Space: unlimited • Maximum File Size: unlimited • Maximum Size of Links: unlimited • Minimum Users: 10 	5\$/month/person

Table 6 - Cost analysis table. Description of conditions and plans offered by Wuala [90], Tresorit[91] and SpiderOak[92]

In the personal package category, Wuala is the platform with more variety of plans containing different storage capacities. This platform has a base plan for 5GB up to 2TB. Since we have a lot of storage alternatives, this platform is the most suitable option if the

user wants to store small amount of information, expending less than 1€/month. Tresorit has a base storage plan at 10\$/month for 100GB, an identical monthly payment as Tresorit for equal storage capacity. However, SpiderOak offers the most tempting plans. For less than 12\$/month, we can increase our space capacity 10x more and, comparing with the other competitors, this platform offers the best conditions on the market.

For collective plans, table 6 shows that we have two main alternative payment plans: pay per user; pay for group plan. The first method requires a payment for each account associated with the product. Unlike the first plan, this second method offers a base number of users that can be incremented affording additional group plans. Wuala has the most promising offers to manage small projects, if you do not require substantial amount of storage capacity. Comparing with the other products, this platform has a base monthly cost of 32,42€, with a 5 users package and 100GB of disk space. If you want to add more clients to the product you have to pay a 99€/year subscription per 5 users, that is equivalent to 1,65€/user/month. However, for each additional 100GB of disk capacity this bill increases 249€/year, or 20,75€ month. In other words, if we wanted to have the same conditions in Wuala as the ones offered by the Tresorit business pack, we would pay a bill of 43,83€/month/user, being the additional storage capacity the most expensive element of the equation. Finally, Wuala has the most promising group plans, since we can start using a group account, without spaces restrictions, maximum file size or number of links for just 5€/month/user.

In this section of report we will not approach the differences of experience, solutions and quality of the products. However, we strongly recommend the reader to deeply analyze the difference between each product before making any decision.

3 Implemented Approach

As stated earlier, this project aims to provide a new secure cloud storage product to WIT Software. As a software engineering project, different approaches can be followed in order to archive a product with high quality and that corresponds to the specified problem. In this section of the report will be described the iterative approach followed in the entire project, Scrum.

This chapter is divided into two different sections. The first section constrains an introduction and general considerations about the methodology used in this internship, followed by the description about the project's planning phase, with all the stages involved and risks.

In order to be short, this section contains only the most important aspects of the followed approach. For more details and other information, refer to *Appendix A – Approach*.

3.1 Scrum Methodology

This section describes general consideration about the methodology followed in this project, Scrum.

Scrum methodology, as another methodology, is a way for team to work together and develop a product. It is based on an interactive approach, challenging the traditional assumption based on a sequential model, with each new piece is build upon previously created pieces. Building products one small piece at a time encourages creativity and enables teams to respond to feedback and change, to build exactly and only what is needed. [93]

“Scrum methodology is often considered as a simple framework for effective team collaboration on complex projects. It provides a certain small set of basic rules just enough structure for teams to be able to focus their innovation on solving what might otherwise be an insurmountable challenge”[94]. Among other things, Scrum provides the necessary agility for projects like the one this internship is focused on: projects where the features can have a wide spectrum of possible requirements, and the technologies that are used are very complex. This methodology is also widely used in other software engineering projects at WIT Software, being a huge opportunity to be integrated in the methodologies used by the company.

3.1.1 How does it work?

In agile methodologies, as Scrum, projects are divided in cycles (usually between two and four weeks), also called Sprints. Each Sprint represents a “time box”, where a set of activities must be executed, adding value to a project. In Scrum, the features/functionalities are kept in a list called Product Backlog. At the beginning of each Sprint period is planned a Sprint Planning meeting in which the Product Owner can prioritize and/or changes the Product Backlog's items and the development team select the items to be implemented during the “time box” period. The selected tasks are transferred from the Product Backlog to the Sprint Backlog.

During the sprint period, all the scrum team convenes in a daily meeting, called Daily Scrum. The main goal of this meeting is to discuss the work done in the previous day and to define a new agenda for the current day.

At the end of each sprint period, the sprint backlog must be fully implemented and all the process is reviewed at a Sprint Review meeting. Finally, is prepared a Spring Retrospective meeting to discuss the plan for the new sprint period, and the cycle is repeated [93].

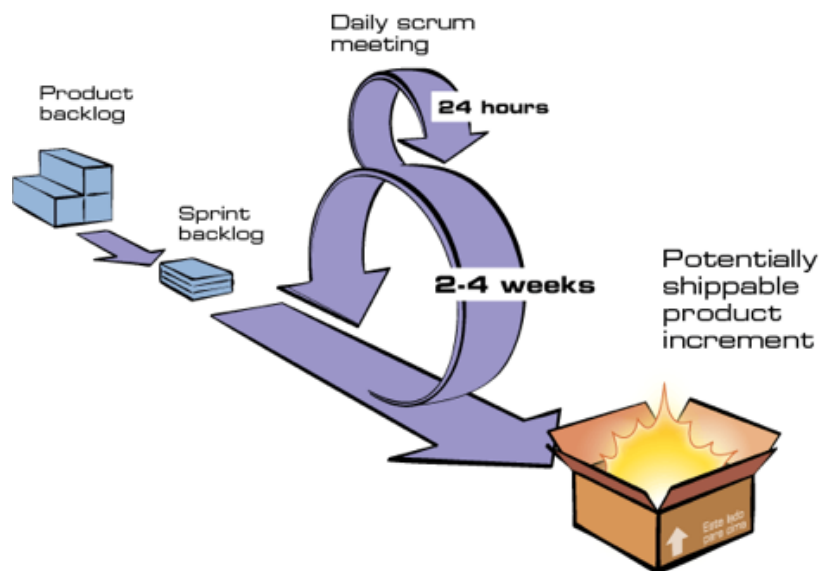


Figure 2 – Illustration of the phases during a Scrum methodology [95]

3.1.2 Roles

The following list contains a definition of roles present in this project, each with a well-defined purpose: [96, 97]

- **Product Owner:** The product owner is the person inside the Sprint project responsible to organize the activities inside the, already mentioned, Product Backlog. The prioritization aim to establish an order of importance of the tasks, risks to achieve the goals. This member is also responsible to clarify all the product backlog items, assuring that all Scrum Team members understand each element. Sometimes this responsibility can be delegated to other person, but this member still has all responsibility about the project.
- **Scrum Team:** This team does not include any of the traditional software engineering roles such as programmer, designer, tester or architect. Everyone on the project works together to complete the set of work they have collectively committed to complete within a sprint.
- **Scrum Master:** The Scrum Master is responsible for making sure that the team is as productive as possible. The Scrum Master does this by helping the team use the Scrum process, removing impediments to progress, protecting the team from outside, and so on.

Supervisor Pedro Pinto owns the Product Owner role, being responsible for the prioritization of the requirements. However, by considering important for the trainees to learn properly the Scrum methodology, he delegates some of his tasks to the Scrum team. Pedro also performs the Scrum Master role, arranging regular meetings with the Scrum Team to help solve some questions or other sort of impediment and checks the progress of the project. Me, José Ricardo Ramos, constitute the Scrum Team. I perform all the programmer, designer, tester or architect tasks, including all type of technical decisions. As already mentioned, I was also delegated to perform the Product Owner tasks.

3.1.3 Planning

The project-planning phase using a Scrum approach, like in any other development methodology, starts with the definition of the requirements. However, a requirement in this methodology has different name and motivations. The process adopted is based on the definition of user stories, described in the next section, containing the description of each story performed by an end-user. All user stories defined make part of the Product Backlog (section 4.6).

3.1.4 User stories

A user story is a tool used in Agile software development to capture a description of a software feature from an end-user perspective, in the everyday language, that describes the ‘who’, ‘what’, and ‘why’ of a requirement in a simple and concise way. A user story must contain the following information:

- **Role:** This describes who requires this story.
- **Goal:** A description of what is required. A statement of the problem (opportunity) to be solved, not the solution.
- **Benefit:** Description of why this story is needed. What is the business benefit if this problem is solved? It is imperative that this section of the card be completed so the team can focus the user on why he or she is asking for this feature/functionality.

A user story template often uses the following type of format:

As a <role>, I want <goal> in order to <benefit>.

User stories also drive the creation of acceptance tests and a time estimation to implement each story. One or more acceptance tests must be created to verify if the user story has been correctly implemented. The time estimation helps the scrum team to choose the user stories to be implemented during a Sprint period, depending on the available time.

3.1.5 Product Backlog

In the simplest definition the Scrum Product Backlog is simply a list of all things that needs to be done until the end of a project. It replaces the traditional requirements specification artifacts, usually expressed as Use Stories. The owner of the Scrum Product Backlog is the Product Owner, but the Scrum Master, the Scrum Team and other Stakeholders contribute it to have a broad and complete To-Do list. This is the only sort of

the requirement followed by Scrum methodologies and, unlike other methodologies can be dynamically changed after each Sprint, according to new requirements or other sort of mutation in the environment.

Due to the lack of space, the complete product backlog list is present in the *Appendix A – Approach*.

3.1.6 Sprint

After the definition of the product backlog, it is time to start specifying activities to be performed during a small period of time, usually called as Sprint. Sprint is a “time box”, made of short duration milestones, which allows the Scrum Team to develop a small portion of the project and produce a demonstrable product increment, the deliverable.

During this project, the defined sprint estimation was four weeks, with regular meetings between the Scrum Team and the Product Owner. This Sprint period was settled by the Product Owner and supervisor, Pedro Pinto, allowing a reasonable time to finish a considerable number of user stories and show a visible increment of value on the product, since the scrum team is composed by just one member.

Before each Sprint, during the Sprint Meeting, the Scrum Team member selects all user stories to be accomplished during the Sprint period, taking into account the priority and the effort of each story. The Sprint Backlog cannot be changed during the sprint period and, to prevent any unexpected situation, an implementation breakdown is made before each cycle. This breakdown of each story specifies all the implementation steps to complete the task.

A user story is only considered “Done” when it adheres to the team’s Definition of Done (DoD), meaning that all the team members recognize it as finished. For details about the definition of “Done” followed in this project, refer to Chapter 2 of *Appendix A - Approach*.

Chapter 4 of *Appendix A* contains an analysis and description of each completed sprint, contacting its user stories, implementation breakdown and estimation time.

3.2 Burndown Chart

During a Scrum project, the team tracks its progress against a Sprint plan on a Sprint Burndown Chart. The Sprint Burndown Chart is updated at the end of each day, according to the number of closed tasks. This tool helps the Scrum team to get a visual representation of the amount of work that still needs to be completed before the end of a the Sprint period.

This tools has an incredibly importance during Scrum projects, because it allows to calculate certain measures like the Burndown Velocity. The Burndown Velocity gives us the average rate of productivity for each day (based on the estimation effort given for each task). This type of values helps us to estimate a completion date for the sprint and correct some deviation. This chart also helps us to verify if we are estimating correctly time about of time needed to complete the tasks.

Appendix A - Approach contains a burndown chart corresponding on all sprints of the project as well as a brief discussion about the execution process of each spring.

3.3 Road Map

The following figures, figure 3 and 4, describe the road map followed during all execution of the project. As stated before this project followed an iterative approach, Scrum. Thus, the definition of user stories that integrate the Sprint backlog is defined as the project evolves, in a Sprint meting. During a Sprint meeting, the user stories to be part of the Sprint Backlog are defined depending on the company's requirements, stories priority and time available. Due to this fact, we cannot estimate the success of our plan based on a pre-established list of well-defined goals for this internship. However, all features and competences for this internship were completely archived. The product backlog also closed all user stories and, by these reasons, we can assume that the objectives for this internship were successfully archived.

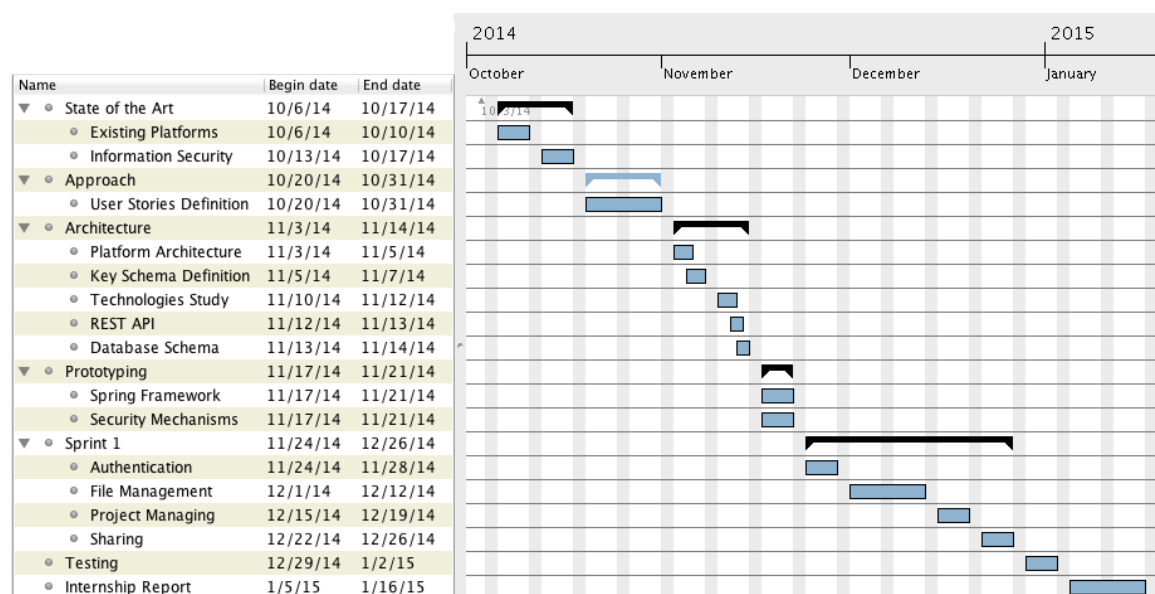


Figure 3 - Gantt diagram describing the work done until the end of the first semester

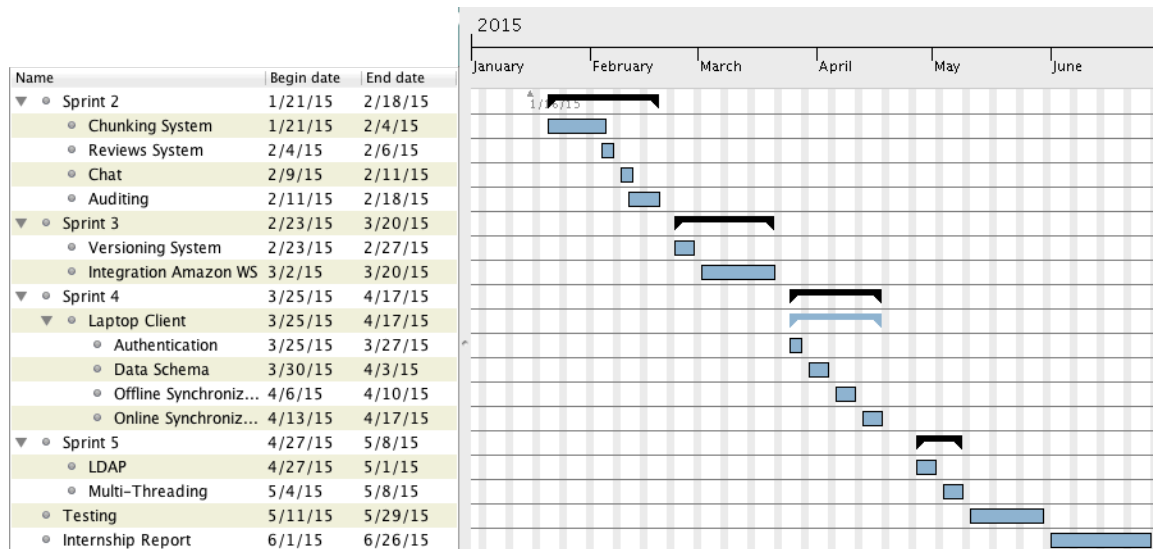


Figure 4 - Gantt diagram describing the work done until the end of the second semester

3.4 Risks

Risks are usually described as the task to identify, assess and prioritize the possibility of an uncertain event occurs and, if it occurs, the effect on the prospects of achieving the pretended project goal. This one of the most important tasks while developing a software project, allowing us to anticipate the occurrence of those events and prepare some strategies to overcome them (mitigation plan).

The following listing contains the risks, identified until the end of the internship that may affect this project, as well as some mitigation plans and their impact and probability in this stage of the project:

ID: 1

Type: Service Failure

Title: Offline APIs

Description: Once some features rely on third party APIs, it is not possible to guarantee that those services will be available all the time

Impact: Low

Probability: 10%

Mitigation Plan: The application will analyze all the responses provided by those HTTP requests in order to assess the availability of the services and produce a correct response.

ID: 2

Type: External Dependencies

Title: Modifications on the APIs

Description: Since some features depend from external APIs, there is a risk of those APIs being updated.

Impact: Moderate

Probability: 40%

Mitigation Plan: The application will analyze all the responses provided by those HTTP requests and log all unsuccessful cases to be analyzed later.

ID: 3

Type: Service Failure

Title: No Internet Connection

Description: An Internet connection is required to perform some of the operations, such as email validation, invitation notification or make request to the APIs.

Impact: Moderate

Probability: 40%

Mitigation Plan: Check the Internet connection before perform those operations and return an explicit message about the connection failure, explaining the impossibility to perform the operation, asking to come back later.

4 Implemented Architecture

This chapter contains a definition of the architectural aspects concerning to the project proposed for the internship. In order to define these architectures, the Scrum Team performed a deep analysis about the purpose of this product and its context, also based on the requirements.

The following list contains a description of all the components that make part of our platform and are defined in this section.

- ***Web Application Server:*** this component refers to the structure constructed to interact with the clients and establishes the connections with the storage component. As it will be described in the *section 4.1 – Software Architecture*, this component was conceived to be installed inside a secure facility, having access to all sensitive data.
- ***Laptop Client:*** this component refers to the client developed during the second semester of the project. Unlike the first developed client, this new component uses real time synchronization that cannot be supported using a client side protocol, such as HTTP. *Chapter 4.1 – Software Architecture* discusses and explains the purpose and mechanisms of this new component.
- ***Storage Server:*** this component refers to the structure constructed to receive all requests made by the client components and performs all the changes inside the storage tier. As it will describe in the *section 4.1 – Software Architecture*, the design allow us to assure 'zero knowledge' privacy.
- ***Database:*** the choices of the database paradigms will be discussed, as well as the arguments of the final decision. *Appendix B - Architecture* contains a full description of the schema, tables and fields of each database.

In order to be short, this chapter only provides a high-level overview about architectural aspects. This way, in order to get a closer look on the complete version of the defined architectures, as well as on some details about the APIs, database description, refer to *Appendix B – Architecture*.

4.1 Software Architecture

To address the described problem, as described in figure 5, our solution is based on a web application component, designed to be installed preferably inside the company's facilities, a Laptop Application, an application to run inside the client's computer, and a Storage Server component, a zero knowledge server, installed either inside or outside a trustable facility.

The web application component is responsible to interact with the clients and send the necessary requests to the Storage component. This element also performs all the necessary cryptographic operations, generating the necessary keys and certificates, preventing the cloud storage component of having any contact with the contents as plaintext or the responsible keys to decrypt that information. For that reason, this server must be stored inside a trustable facility.

The laptop client is a component developed to synchronize a folder inside the client's personal computer with his current WIT Cloud projects. This component allows us to access, store, update and remove our projects, workspaces, folder and files by performing these actions upon the contents inside the application's folder.

The cloud storage component is the only responsible to preserve all the application logic, but never having contact with the contents as plaintext, and store information inside databases.

The relational database is used to store all of the information about projects, workspaces and users. The relational paradigm makes sense in this case because it will not allocate a lot of information (so that we would need to expand horizontally), the table elements are extremely connected, can be requested frequently and its not expected to store huge amount of information. This way, if we were using a non-relational (NoSQL) database, data denormalization would be a performance and consistency issue in this scenario.

Besides the relational schema makes sense in this part of the application, for the file and folder storage does make so much sense to use this paradigm. The amount of content stored inside this kind of platforms is sometimes unpredictable, assuming huge proportions, and a distributed store system seems the best option. The denormalization of the database does not represent a problem in this scenario also. Due to these facts, we can adopt a NoSQL database. For the same reason, the log store is made inside a non-relational database system.

For more information about the data model, including a more detailed explanation about the two database paradigms (SQL and NoSQL) and the DynamoDB index management, please consult *chapter 4.6 – Storage Model*.

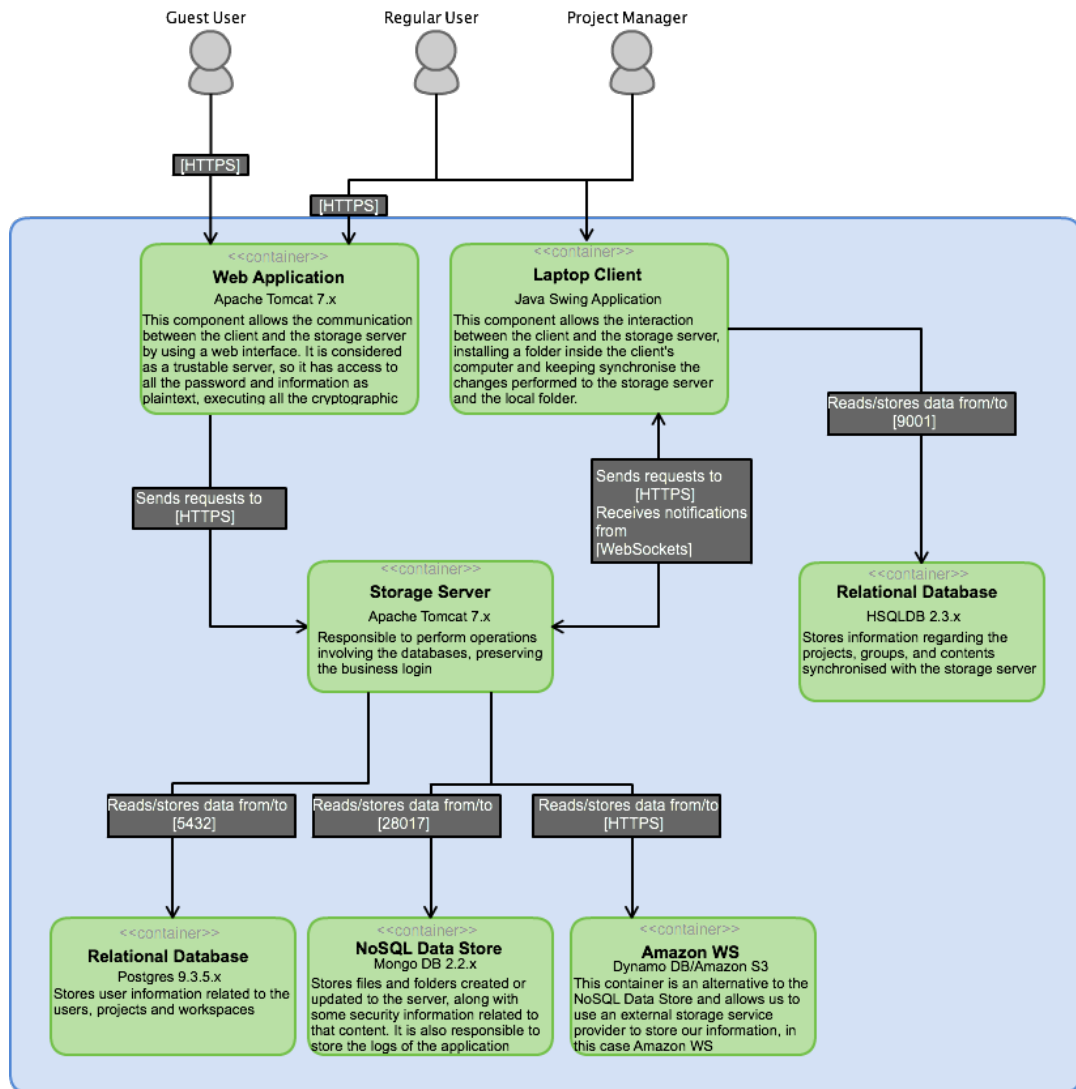


Figure 5 - Containers level description of the Wit Cloud platform.

Figure 6, 7 and 8, present in the next pages, describe the web application, laptop client and storage server at the component level.

We will start to approach the basic principles behind the web application component. As described before, the communication starts with an HTTPS request from our client's browser to the web application component. Servlet filters, implemented with the Spring Security framework, will analyze the requests, perform URL interceptions, verify required authorizations and redirect users to the proper page. During the sign in process, this filter uses a custom authentication provider to interact with the storage server, in order to authenticate the user. A LDAP connector connects this component to the company's LDAP system and registers users using information already stored inside this authentication service provider.

After the authentication process, all requests will be redirected to one of the controllers installed on this component, depending on the path of the request. This controller will produce all required solicitations to the storage server component and

perform all the cryptographic operations needed. Finally, it will be returned a page/information with the response of that request.

We also included an additional component, *Stomp Message Broker*, in order to receive and redirect messages between members of the same project, using Web Sockets as the protocol to establish the connection.

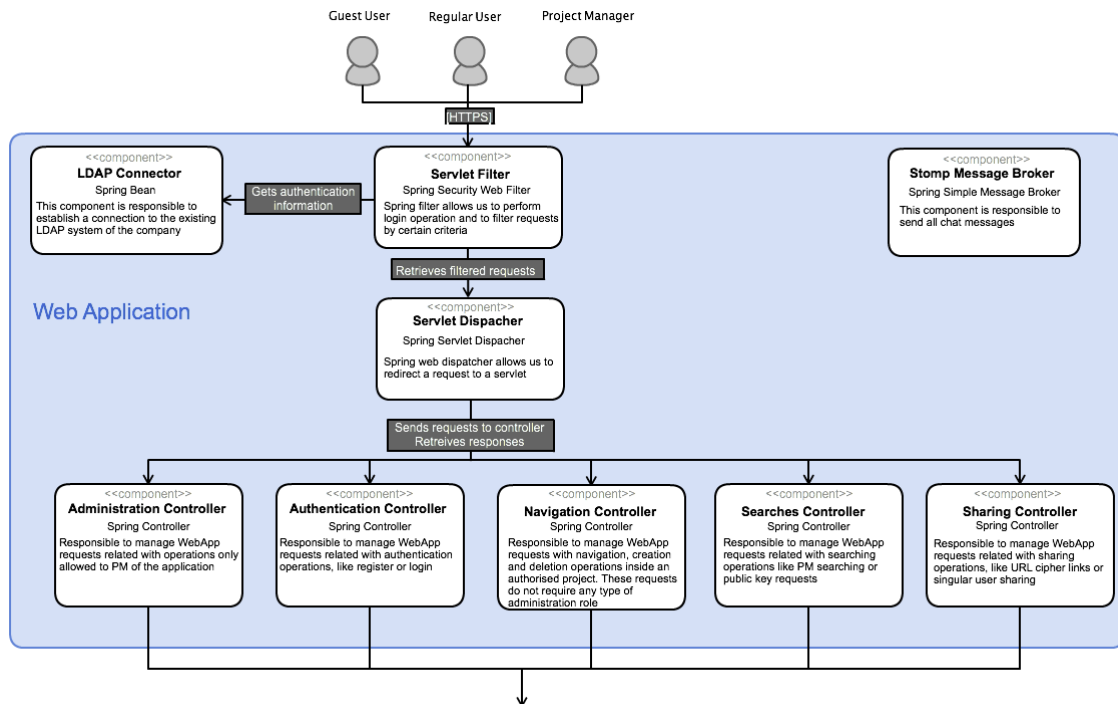


Figure 6 – Web Application component level description. The arrow without connection establishes a communication to the servlet filter of the storage server

This second element, Client Laptop, as stated before, is a Java application conceived to be installed inside our client's computer. This application starts with the instantiation of the *LaptopStartup* component, in order to check and read application's configuration files, or, in case of this information is not available, invalid or corrupted, create a new window to input authentication credentials and choose the directory to store the application's folder.

After the authentication process, application instantiates two new components, with references to each other: *StompWSCConnection* and *ProjectSync*. The former element, *StompWSCConnection*, starts the WS connection between the application and storage server. If the connection succeeds, this component sends a callback notification to the second element, *ProjectSync*. This last component will initiate the synchronization process between the local folder and the remote server. All local and remote updates will be notified and possible conflicts solved. All subscriptions to the storage server topics will also be performed during this synchronization process, using callback notifications to the *StompWSCConnection* component, in order to submit all subscriptions and manage topic's incoming messages. Figure 7 illustrates the schema of the Laptop Client at a component level.

All contents, projects and group will be referenced inside a local HSQLDB database, as described in the *section 4.6 – Data Model* of the report. This information will be needed during the synchronization process.

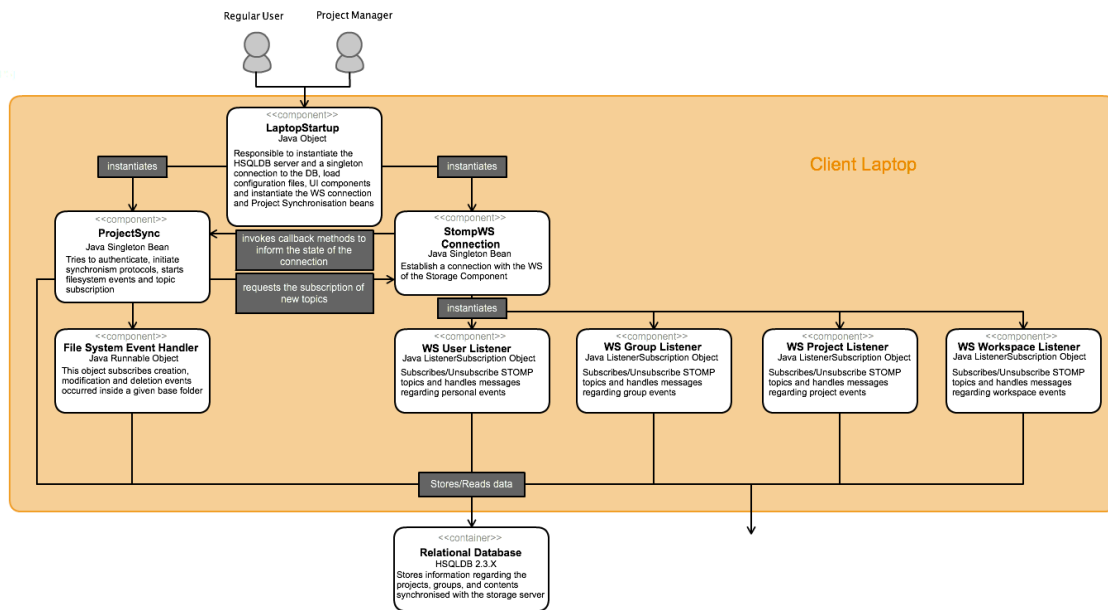


Figure 7 – Client Laptop component level description. The arrow without connection establishes a communication to the servlet filter of the storage server

The requests made to the cloud storage component will be initially filtered using a Spring Security Filter, which, as the one already described in the web application, makes use of the standards Servlet Filters. This filter is responsible to check the session's permission to the elements requested, perform authentication operations, define the entry point to the http elements, the authentication success handler and the authentication failure handler. If the user has the required authorizations, the request will be redirected to the controller responsible to handle the request.

These controllers will select/update/delete/create information from the databases. As described in the *section 4.6 – Data Model*, we choose to include two different database paradigms in this project, relational and non-relational. The relational database will be installed inside the storage server component and has a single connector to the PostgreSQL database. However, in order to improve the flexibility of our product, the non-relational database includes two different connectors: Mongo DB and Amazon WS. The prior component, Mongo DB, aims to provide a connection to the non-relational and document-based database, possibly installed on the same machine or network as the storage server. Therefore, all information regarding the platform will be stored without using any external storage services. On the other side, we have the Amazon S3 connector. This element establishes a connection with two different Amazon services: *Dynamo DB* and *Amazon S3*. The reasons that led us to choose this platform are also described in the *section 4.6* of the report. Figure 8 illustrates the schema of the Storage Server at a component level.

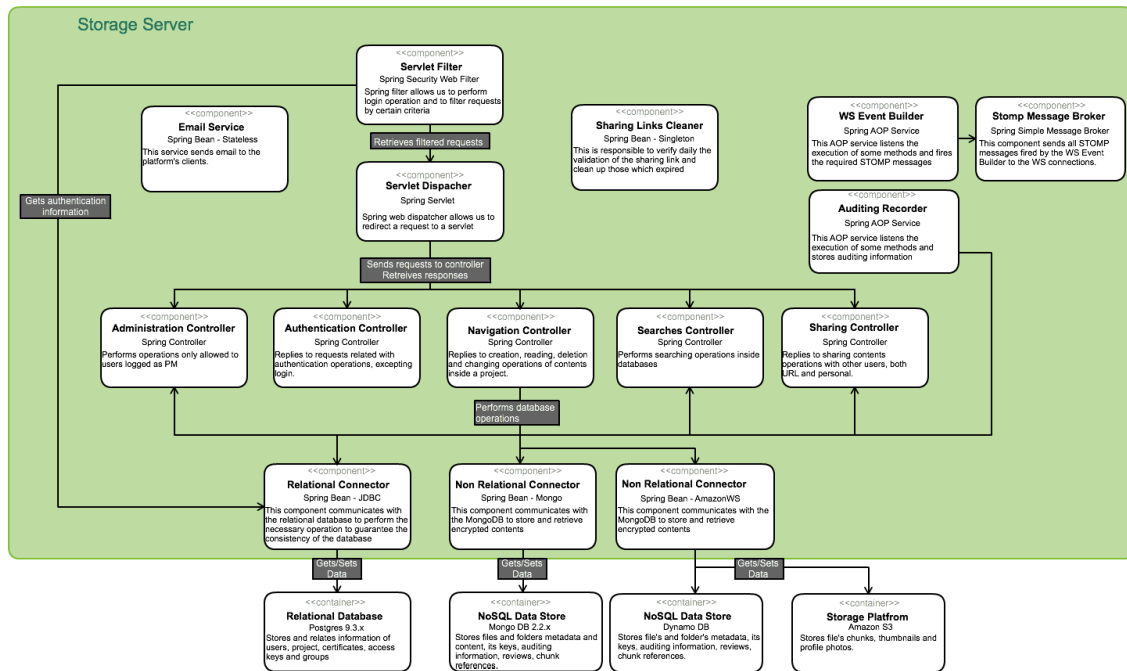


Figure 8 – Storage server component level description

4.2 Key Schema

The developed platform implements a combined encryption process based on both asymmetric and symmetric encryptions.

In the following list are described five important terms of our mechanisms, which are important to understand the encryption mechanism implemented:

- **Base Folder:** The uppermost folder of a project that a user has access.
- **File Key:** Set of symmetric keys used to encrypt or decrypt a file. Each file has its own unique and random set of file keys.
- **Users Keys:** Every user has his own pair of asymmetric keys (private and public).
- **Password Key:** A 256 bit long key, derived from the user original password using a key strengthening function.
- **Group Key:** Similarly to users, every group also has its own pair of asymmetric keys (private and public). Furthermore, each group entity has its unique and randomly generated membership key.

4.2.1 Algorithms

To perform all the cryptographic operations (with exception of HTTPS, that is automatically performed by the web server), we used the following algorithm:

- **AES-256:** AES-256 is currently a standard in symmetric encryption operations and considered as a U.S. Federal Information Processing Standard (FIPS) by the National Institute of Standards and Technology (NIST) [48].
- **PBKDF2 HMAC:** This algorithm performs all key strengthening operations. According to the NIST specifications [98, 99], we used a number of iterations between 64.000 to 128.000 and a random salt of 128 bits, produced by a strong pseudo random number generator function of the UUID class. This algorithm is also recommended by NIST to assure a high level of randomness and entropy of the stored passwords.
- **SHA-256:** This algorithm performs all the necessary hashing operations. Based on the studies already mentioned, SHA-256 is considered secure and adjusted to the required performance level.[39]
- **RSA 2048:** This algorithm performs all the necessary asymmetric encryption operations. The size was adjusted to support increasing of computational power and technological advances forecasted in a near future. [59].

4.2.2 Entities

In the introduction chapter of this work, we mentioned the importance of groups sharing a common goal. Therefore, we created two distinct entities: *Users* and *Groups*. The former entity represents a single customer that can be associated with several groups sharing common accesses. Each entity, user or group, has an associated RSA key, however generated and encrypted with different schemas.

- Users:*** Every time a client signs up, the web application component generates a key pair of private and public keys. The public key is sent as plaintext to the cloud storage server component and is stored inside the database, without requiring any encryption operation. The private key is encrypted using a derivation of the user's original password, with a random number of iterations and salt. To prevent the access to the strengthened key, it is hashed before leaving the web application component. A visual representation of this process can be found in figure 9.
- Groups:*** Every time a group is created, the web application component generates a key pair of private and public keys. The public key will be freely available inside the platform and is sent as plaintext to the storage server component. The private key has to be encrypted with the membership key before leaving the trustable infrastructure. To prevent the access to the membership key, this key is also encrypted using the group's creator public key. If we want to add more users to this group we just have to create a new entry of the membership key, but this time encrypted with the public key of the user that we want to associate. This processes is also represented in the schema of the figure 9.

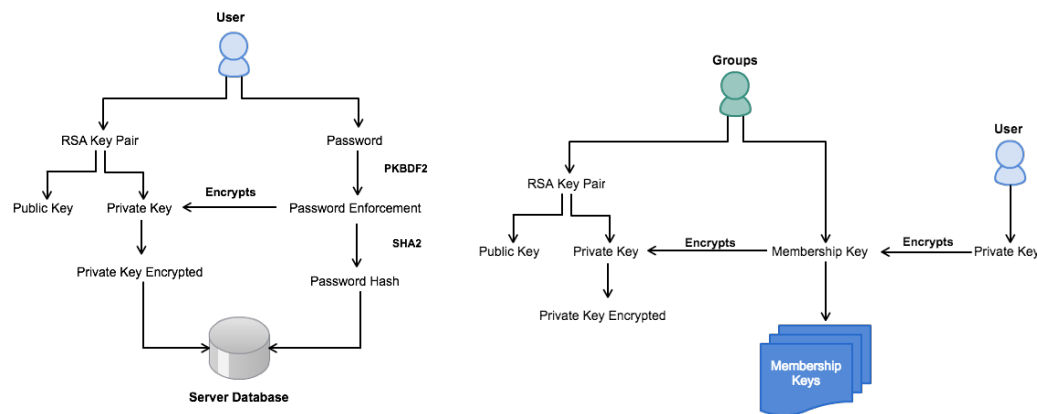


Figure 9 - Schema of the generation and storage process of the keys belonging to a single user (left) and groups (right). The links represent a relationship between the components, being the head of the arrow the target of the action, the tail represents the subject and the action is described with bold letter

4.2.3 Data Key Schema

The schema illustrated in figure 10 represents the solution designed to secure the information inside the database. The content encryption schema relies on a key derivation mechanism using symmetric links. When a user has full access to certain content inside our file hierarchy, he can derive the necessary keys to access the sub contents of that item. This process requires that each content, file or folder, contains 3 different types of keys:

- **Subfolder Key:** This key allows us to have access to the sub-contents of the present folder and derive the backlink key.
- **Backlink Key:** This key is necessary to derive the data key of the present content and to have access to the backlink key of the parent folder.
- **Data Key:** to encrypt all data needed to represent the folder and its content.

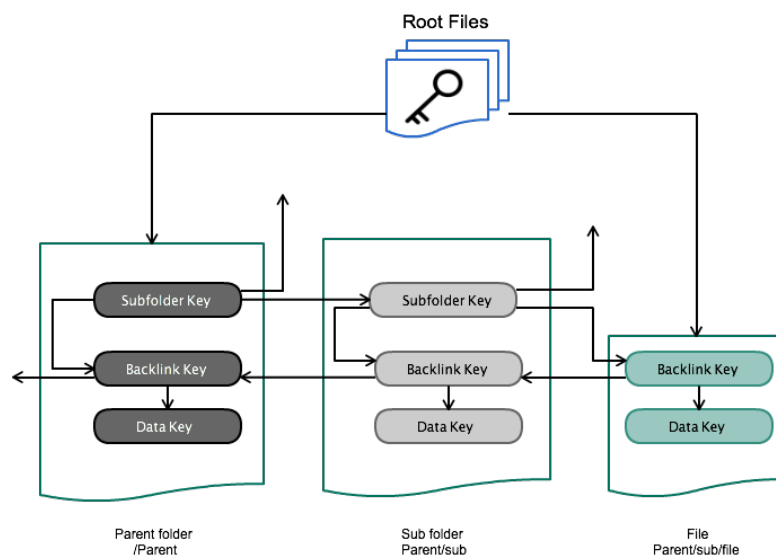


Figure 10 - Schema describing the key derivation structure. The links represent a symmetric derivation relationship, where A->B means that B derive from A. The links that start or end nowhere indicate arbitrary numbers of connections to elements not shown in the figure, namely links to additional child folders and files.

As described in figure 10, this schema requires a root file table (blue box with a key) to store all the access keys of a project. Each table entry is the *subfolder key* of the content encrypted using the public key of the associated user. This means that if we want to share a folder with N different entities, the root file table will have N different entries for the same key inside the table (not counting with the entries to the parent folders, which also grant access to this folder).

Besides the key derivation property, with this schema we also ensure that even if we only have access to a single file inside a project, we can access its complete location inside the cloud's file hierarchy by getting the *backlink key* of the parent folder. This key allows us to decrypt the parent's data key and get the metadata. This process of getting metadata of the parent folder can be repeated until we reach the base folder of the project. However, this

does not mean that we are granting access to more items beyond the ones shared with the user, since only *subfolder keys* can grant this type of privilege.

4.2.4 Content Sharing

When a user wants to share some information with other customer, the web application requests for the person's public key to the cloud storage component. This key is used to encrypt the content's *subfolder key*. The result is sent to the cloud storage component and inserted inside the root file table. Now, the added user can decrypt this root file key with his private key, getting access to the shared contents and all its sub-contents.

When we want to share content with a group, the process is not so simple. The web application makes an HTTP request to the cloud storage component for the group's public key. The *subfolder key* is encrypted with the requested key, sent back to the cloud storage component and added to the root files table. If a group member wants access that information, the web application has to request the storage server for the group's membership key entry that belongs to the customer. The user can decrypt this key using his private key. The membership key is used to decrypt the group's private key, allowing us to decrypt the subfolder key and get access the contents belonging to that group.

4.3 Web Sockets

Web socket protocol defines a new way of communication for web applications: full-duplex and two-way communication between client and server. It is designed to exchange events at high frequency and low latency making the web more interactive with a great UX. This protocol was designed to be implemented in web browsers and web servers but it can be used by any client or server application. However, in this last case, we can face some difficulties because there is not such support for libraries and framework capable of handling this protocol and all its potentialities. Since we also implemented this protocol in both web application and laptop application, all adopted strategies will be detailed later in this chapter.

In order to be short, this report will not provide a proper introduction to the Web Socket protocol. However, it is important to understand that HTTP is only used for the initial handshake, which relies on a mechanism built into HTTP to request a protocol switch. This protocol switching brought us some challenges to access the security context and authorize the connection to the channel. However, we extended the default Spring's *DefaultHandshakeHandler* in order to validate the creation of the WS. In this component, we are allowed to access the HTTP security context and authorize the protocol switching only if the user is authenticated. We also can transmit some information to be accessible in the WS context, in our case we send the *user id*.

Web Sockets have great potentialities, however lacks of a messaging protocol. It is just a very thin layer over TCP that transforms a stream of bytes into a stream of messages, leaving their content undefined. In order to address this problem we used STOMP protocol over the regular WS connection. STOMP was designed to address a subset of commonly used patterns in messaging protocols, such as the ability to intercept route, process, messages on both client and server side, using an enterprise message broker. Our message broker was implemented using a simple-broker provided by the Spring framework. This

component is responsible to receive all incoming messages and send messages back. The reader must visit [100].

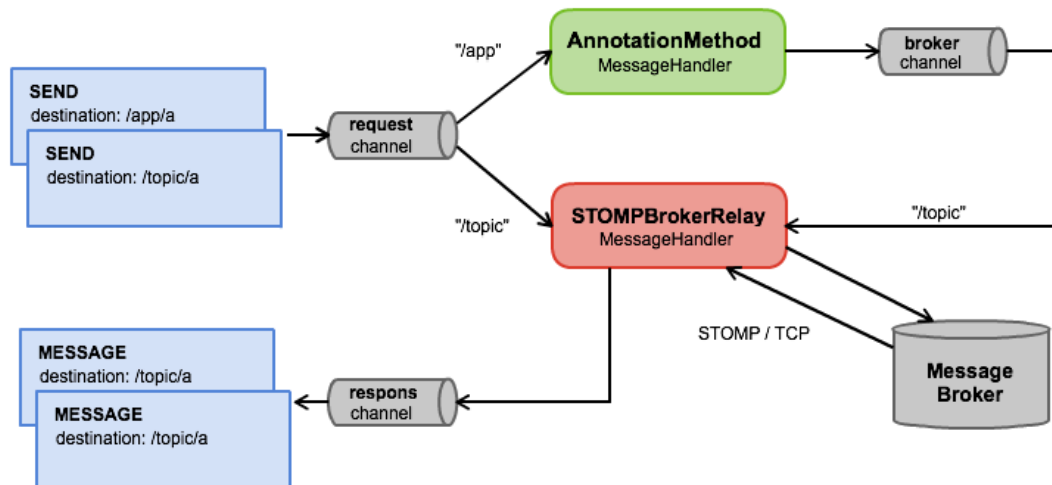


Figure 11 - Typical architecture of a default Spring STOMP message broker [101]

Figure 11 shows the typical architecture of a default Spring STOMP message broker. In this project we implemented two different WS, installed in two different components (web application and storage server). In both cases we needed to limit access to the inbound and outbound channel messages. This task was performed also extending the *InboundChannelInterceptor*. This component allows us to filter messages from the Web Socket connections. Every incoming Web Socket message carrying a STOMP frame is sent through this channel. Thus, we can verify the access to the channel and control the transmission, or denial, of the message. The web application performs two kinds of filters: Subscription and Topic Messages. Every time a user tries to subscribe a topic, this component access the user id (associated during the handshake) and validates his authorization to the topic. After a successful subscription, the project ID is associated to the session. Customers are only allowed to send messages to subscribed projects.

As said previously, the laptop application integration faced more challenges than the previous component. Unlike other programming languages, such as JavaScript, Java lacks a complete STOMP messaging framework in order to connect to WS, subscribe STOMP topics and send messages. Therefore, the most challenging task of WS integration with the laptop application was the study and development of our own framework to perform all these operations. This study required the exploration of STOMP documentation[100] and some reverse engineering techniques applied to the JavaScript framework. All this code was not already published in a public repository, but I intent to share it in the near future.

4.4 Deduplication

As stated before, deduplication is a suited technique for maximizing the utility of a given amount of storage. In this platform all files sent to the storage component are chunked into smaller pieces of information before getting transferred. This technique requires the definition of two important algorithms: chunking and fingerprinting. The decision of using TTTD and Adler-32 was based on the results described in *section 2.4* and

on the study published in [63]. However, the values presented in [63] for the minimum, average and maximum chunk sizes were impracticable for this project. Amazon S3 creates a new TCP connection and SSL handshake (HTTPS) for each request [102]. Therefore sending/fetching small size objects could incur a substantial overhead and have significant impact in our transfer throughput. Parallel connections for sending these chunks can also provide a boost in throughput. In order to address these problems, we defined an average chunk size of 512 KB and a 128-thread pool, basing on this article[103].

The transfer rate between our storage server and AWS was substantially improved, but we still have margin to optimize the upload/download processes inside the web application component. Thus, since chunking process is performed before cryptographic algorithms, we also use parallel computations to encrypt/decrypt our data. Figure 12 and 13 illustrated our download/upload processes using deduplication practices.

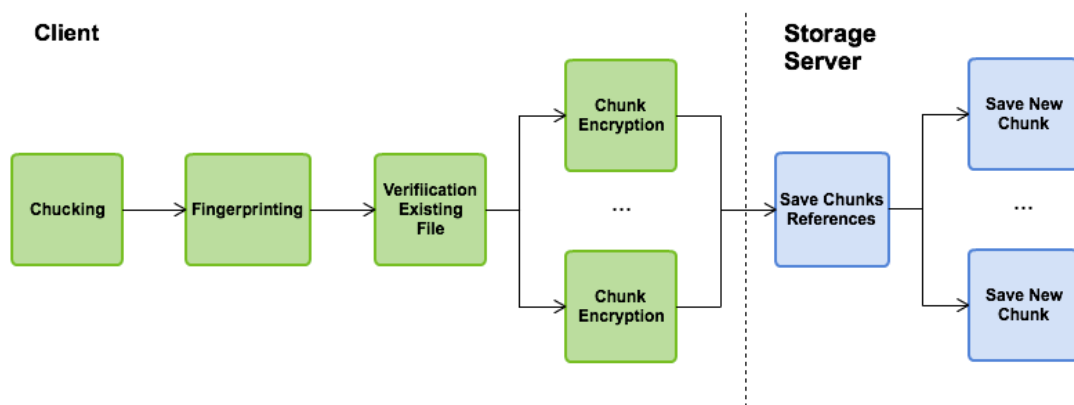


Figure 12 – Illustration of the upload process. Green blocks represent processes occurred inside the client component, whereas blue elements describe operation inside the storage server.

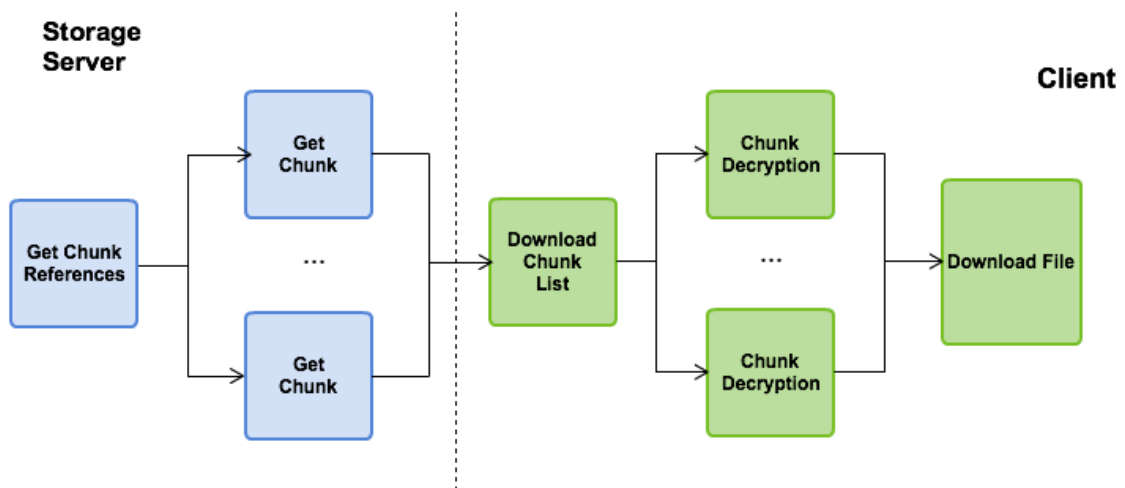


Figure 13 – Illustration of the download process. Green blocks represent processes occurred inside the client component, whereas blue elements describe operation inside the storage server.

4.5 LDAP

Lightweight Directory Access Protocol (LDAP) directories and LDAP authentication have become one of the enterprise user infrastructure cornerstones. This standard application protocol allows us to access a central directory inside the company to get information about users, system, networks and application.

Since some components of this platform were designed to be installed inside the company infrastructure, we could take advantage of this service. Single sign on - a property of access control of multiple related, but independent software systems - is a common usage of LDAP since we can authenticate a user in different services without any registration process and using the same pair of credentials.

In order to implement this strategy we connected our web application to WIT's LDAP system. This strategy allowed us to remove the registration process and automatically authenticate the user using his service username and password. The bellow diagram shows the authentication process.

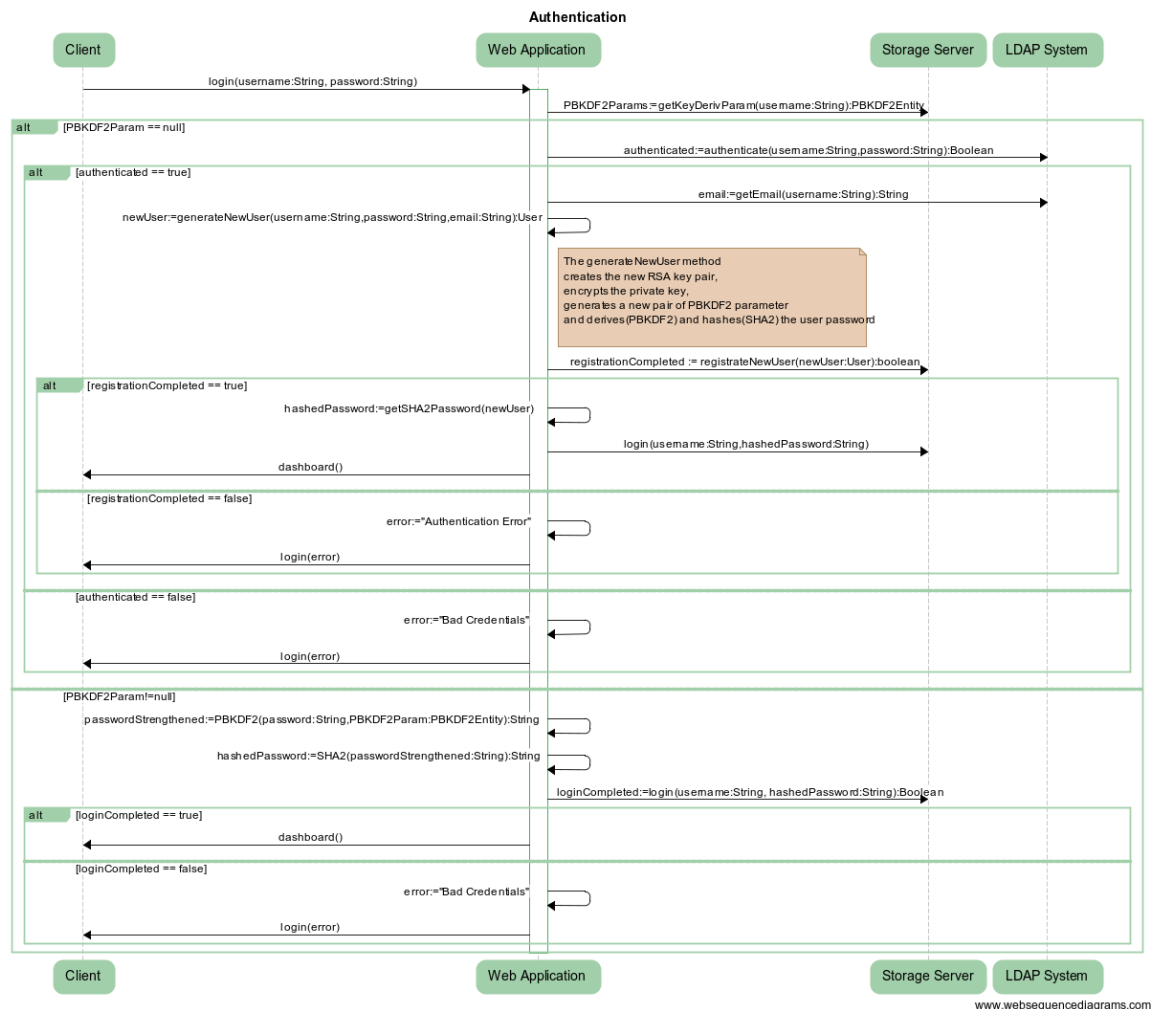


Figure 14 - Authentication process sequence diagram

4.6 Storage Model

This section of the report contains a full description of the database schemas and configuration files regarding the storage and laptop components. This section reports all implementation details about the data model conceived to manage information inside both components.

Relational databases (RDBMS) have been the primary data model for the last two decades. This database model uses Structured Query Language (SQL) as the standard programming language to communicate with the relational databases. SQL allows you to perform all different database operations, including managing, storing, and querying. These relational databases are installed inside a SQL server. They comprises a set of pre-defined tables and columns to build a data structure designed to store the desired information. These tables also store information regarding indexes and keys, to give consistency and improve queries performance.

However, besides all the well-know benefits brought by this database paradigm, most of the relational databases are intended to be installed in a single component, excepting some new engines, such as MySQL Cluster [104]. The only way to improve our storage capacity is to expand our database tier vertically, in other words, improving the quality of that component, buying expensive hardware components with a better performance. This limitation resides on the fact that, if we tried to expand our database horizontally, dividing our database across several database nodes, the relational database engine would not be capable to create the mapping between elements that could be inside different machines.

When we are handling a huge amount of information, the complexity of well-structured information and organization of RDBMS databases also slows the performance as data volume gets bigger and can be not scalable enough to meet the needs of Big Data.

This fact led to the emergence of NoSQL, commonly referred to as “Not Only SQL”. This framework was conceived for the high demands of big data and allows high performance and agile processing information at a much bigger scale. Due to the lack of structure and relationship between database elements, the information stored can be splitted across different database nodes, expanding transparently our database without application downtime, improving our storage and information handling capacities (ex: Hadoop [105]). The database can be easily adjusted to the platform needs by adding and removing storage nodes (more flexibility).

Besides all these benefits, NoSQL databases are still in very premature stages and many key features are yet to be implemented. The SQL language is a very expressive language, easy to program and require a less programming expertise, instead of NoSQL programming that required more significant amount of work and knowledge. And, finally, the performance of the NoSQL database is also significantly affected performing the JOIN operation, one of the most commonly instruction of RDBMS systems.

4.6.1 Storage Server

The storage server component has two kinds of database paradigms, relational and non-relational databases, conceived to address different kinds of problems.

4.6.1.1 Relational Schema

As stated before, relational database models are most suitable for aggregating information with a simple and tabular structure, non-dynamic data model, a well-determined number of relationships between tables and a predictable storage capacity.

Figure 15 describes the WIT Cloud’s relational database schema. This paradigm makes sense in this context since these database entities describe the most basic and simple elements of our platform, highly connected and with well-defined properties. This information will not change rapidly and do not require huge storage requirement (stored inside our storage server component) and the relations settled will extract some information regarding different database tables.

The figure number 15 describes the relational database schema of this platform. The schema allows us to manage accesses, certificates, sharing contents and groups, avoiding unnecessary relationships and an adjusted normalization of the information.

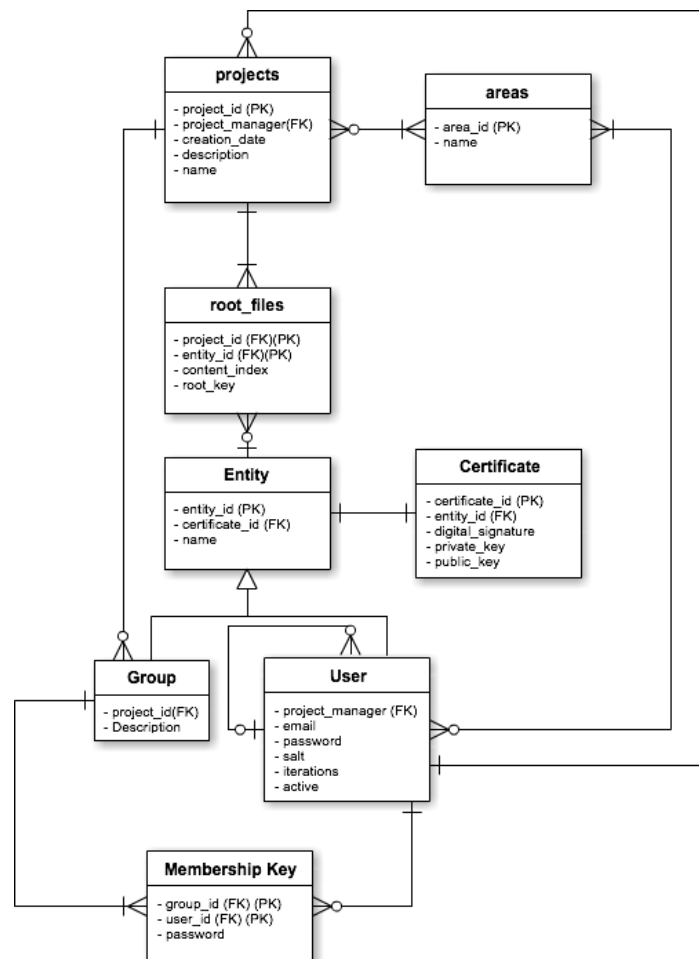


Figure 15 - Relational database schema of the storage server.

For a complete description of the table and its respective fields, the reader must consult *Appendix B – Architecture*

4.6.1.2 Non Relational Database

The non-relational schema was used to represent file’s metadata, reviews, chunk references and auditing information either inside our non-relational database, Mongo DB, or using an external service, such as Amazon Dynamo DB. In the prior storage platform, the file’s chunks and thumbnails are also stored inside the non-relational database.

The choice of using a non-relational database engine to store this information is due to the fact the amount of data is expected to grow fast and we need to scale efficiently. In contrast to the relational information, this data is more dependent on the user’s activity and can scale in an unpredicted way. The schema also needs to be more flexible (especially on tables storing file’s information) and the data is more “independent” than in the relational paradigm (usually accessed directly using indexes, instead of intensive queries).

However, non-relational databases are not based on a single model, unlike the relational model, and each database concept, depending on their target-functionality, adopt a different one. We can distinct four different categories:

- **Key-Value:** It is considered the most basic and backbone implementation of NoSQL. The main idea is using a hash table where a unique key points to a particular item of data. This data is considered to be inherently opaque to the database.
- **Column Based:** The columns can be compared with the columns used in a relational system and are arranged by category. We also use the concept of keys but, instead of pointing to a single element, this key can point to multiple columns. Since the columns can be stored in different machines this model is very efficient to store and process large amounts of data.
- **Document-Based:** A document-based database is inherently a subclass of the key-value store. The concept has the basic mechanism as the key-value, but can associate a model and meaning to the data stored. The semi-structured documents are stored in formats like JSON.
- **Graph Database:** This concept uses a graph model to store and scale the information across multiple machines. These databases are commonly used by applications whereby clear boundaries for connections are necessary to establish.

Comparing all non-relational database types, the document-based model seems the best to be the best solution to map our ORM objects, using indexed information to access them and preserve the information using the JSON structure. Mongo DB is a cross-platform, open-source, document-oriented and non-relational database, released under a combination of the GNU Affero General Public License and the Apache License, being complete free and without use restrictions.

As stated in the *section 4.1 - Software Architecture*, this platform implements a connector to establish a connection with Amazon WS. Amazon WS offers a broad set of

products and services to reinforce company's computational power, storage capacities, databases, data analytics and tools that help organizations to move faster, reduce costs and scale applications. The decision to implement this connector was motivated by the possibility of reducing the amount of data stored inside the storage server infrastructure, which could require the acquisition of new hardware equipment to store and balance the information, and use a scalable, consistent, available and fast service to manage our data.

Therefore, we selected two Amazon WS products, with distinct purposes, as we will describe in the next two topics:

- ***Dynamo DB***: This is a fast and flexible NoSQL database engine, completely developed by Amazon, with high levels of consistency, latency at any scale. It is a fully managed database and supports both document and key-value data models.

Despite of having the concept of a document-based database, the concept of keys is substantially different than we used in Mongo DB. As in the previous database, tables have mandatory and unique key values that identify each item, but these keys can belong to two different categories: Hash Primary Key; Hash and Range Primary Keys;

The hash primary key is made of a single attribute, a hash attribute. Dynamo DB uses this field to build an unordered hash index. Unlike the first index, Hash and Range Primary Keys, are made of two attributes. Dynamo DB constructs an unordered hash index on the first field and a sorted range index on the second attribute. The uniqueness of the elements depends on these two components and the queries can perform different types of filters based on the range attribute. As you can observe in the field description table in *Appendix B - Architecture*, hash and range indexes were very useful to create the tree relationship, search by review's and auditing date and query a specific version of a content, without requiring additional queries or "composed fields".

In order to access the table efficiently, Dynamo DB allows us to create additional indexes for the primary key attribute (local indexes) or a new hash and range index that can differ completely of those on the table. These indexes were also used to improve the access using other field of the contents store when another index type was already in use.

Amazon Dynamo DB allows you to automatically generate your IDs using the *DynamoDBAutoGeneratedKey* annotation. However, it is limited to String values and changing its format will require more space allocated and changing a lot of components of the application. To address this problem, we implemented an atomic counter, using databases consistent readings and object versioning, in order to read and change the database consistently.

- ***Amazon S3***: Mongo DB stores all information regarding metadata, auditing, reviews, chunk references and chunk information inside the non-relational database. However, this connector uses an alternative mechanism to store the information of the files. Amazon S3 provides a simply, easy to use, secure durable and high-scalable object storage, with a simple web service interface to retrieve and store any amount of data. Some of the main differences between the two products described in these three points:

- **Size Limitation:** All items stored inside Dynamo DB cannot exceed 400KB, including attribute name binary length and attribute value lengths.
- **Data Structure:** Amazon Dynamo DB stores structured data, indexed by primary key, and allows low latency read and write to items ranging from 1 byte up to 400KB. Amazon S3 stores unstructured blobs and suited for storing large objects up to 5 TB.
- **Costs:** Amazon Dynamo DB and Amazon S3 prices are substantially different. Since Amazon S3 is a storage platform, the ratio price/(GB stored) is almost insignificant compared with the Dynamo DB. As described in the official website[106], the price in the first terabyte of information cost 0.300\$, using the standard storage. The most expensive parts of the service are the costs per number of requests and per data transferred to/from Amazon S3. In the case of Dynamo DB, the price per gigabyte increases to 0.25\$ month, assuming US East region[107].

However, the price/request is a considerable smaller using Dynamo DB. The cost of making 1 million write and read operations per day using Dynamo DB would be 7.50\$/month and using Amazon S3 approximately 162\$ (wherein 92% of the cost are related to write operations).

The size limitations and usage costs led us to choose Amazon S3 as the service to store the information regarding the stored files.

For a complete description of the tables/collection and its respective fields of the non-relational database, the reader must consult *Appendix B - Architecture*.

4.6.2 Laptop Application

This section of the report refers to the laptop application data models. These models are divided into two different kinds of information: application preferences; and WIT Cloud content's information.

In the first point, application preferences, we aim to produce a simple and effective way to save and load information regarding some preferences of the application. These preferences are defined during the first run of the application, configuring the account associated with the program and folder to store the contents downloaded from the storage server. Since these preferences are easily defined in four distinct fields, we choose to store this information inside two files, as we describe in the *section - 4.6.2.1* of this report, placed inside a hidden folder in the same directory where the *.jar* package is running.

The second type of information stored, WIT Cloud content's information, stores all information regarding the contents downloaded/uploaded from/to the storage server. Although this information seems to be of the same kinds as the described in the *section 4.6.1.2*, since this information regards to our personal contents, with a relative reduced storage weight, we choose to use a relational model, using HSQLDB. HSQLDB[108] is a high performance java relational database engine that can be included and instantiate in our application *.jar*, bringing a more friendly database language.

4.6.2.1 Configuration Files

After the user signs in the laptop application successfully for the first time and choose the folder where he wants to store the contents downloaded, the platform creates a new folder and stores two new files inside it. The new folder, *.config*, is hidden, by default, on the most of file explorer programs, preventing it to be deleted or changed inappropriately. This folder is created inside the path where the program was deployed.

These new files added, *configurations.json* and *credentials.json*, allow us to store /access some application configurations, loaded during the application start up. This information, as the names say, allows us to fetch information about the folder containing our WIT Cloud's projects, and credentials to authenticate with the storage server.

Both files have the information described in JSON format, as described in the following example:

Credentials.json	Configurations.json
<pre>{ "id":3, "name":"jrramos", "password":"Authentication_Password", "privateKey":"Private_Key" }</pre>	<pre>{ "directory":"/Users/jrramos/Documents/WIT_Cloud/" }</pre>

The first file, *credentials.json*, is opened and mapped to a *Credentials* object at the application startup. This object is a singleton bean, available during all application lifecycle. The *name* and *password* information allow us to authenticate the user automatically and get a session Cookie, sending these pair of credentials to the storage server without requiring any user's input. The *privateKey* field is a PKCS#8 Base64 representation of the user's private key. This key is required to access all our personal projects, personal sharing *rootfiles* and access to the group's membership keys, as described in the *section 4.2* of the report.

If any of the files is corrupted, some information is invalid or there was an error during the authentication process, the application is restarted and all current information is dumped. The user will have to reintroduce all authentication information, choose the WIT Application folder and download all projects' information.

4.6.2.2 Laptop Client Database

The mapping and synchronization processes between the storage server and the local client is archived by using a local HSQLDB. This database stores information regarding all projects, groups and contents currently shared with the costumer. The application needs to maintain updated all relationships between these elements and perform all required database modifications in response to file system events or STOMP messages, keeping track of the file versions and comparing all these recorded information with the data collected from the server. For more information about the laptop client application, it is highly recommend reading the *section 5.2 – Laptop Client* of this report, where details about this component are given.

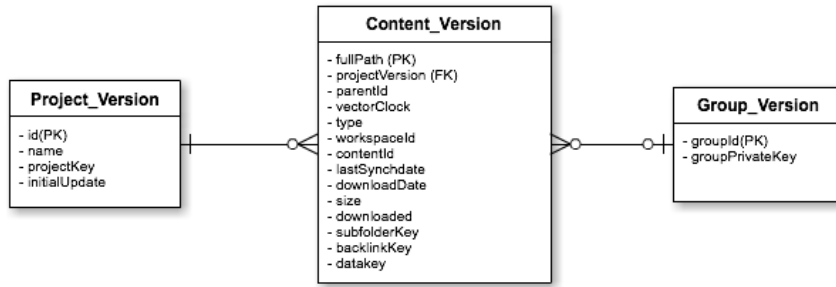


Figure 16 - Relational database schema of the storage server

The figure number 16 describes the relational database schema of this platform. The schema allows us to manage accesses, certificates, sharing contents and groups, avoiding unnecessary relationships and an adjusted normalization of the information. For additional information regarding the table and its fields, the reader is advised to consult *Appendix B – Architecture*.

5 Final Results

The current chapter resumes all the work produced during this internship. We divided this work into two main sections: web application and laptop client.

Each chapter contains a text explanation, as well as some screenshots or schemas to illustrate the obtained results. *Appendix-A - Approach* contains a full description of the User Stories implemented during the first and second sprints of the final product.

5.1 Web Application

This section presents the most important features developed concerning the web application. For each feature we describe and specify some implementation details about each feature, its mainly challenges and faced problems.

5.1.1 Project Management

After the key schema established, the first step was to develop some of the most basic features of a project-managing platform. These tasks include mostly content management, administration and authentication features. The following list has a description of the implemented features concerning the project management tasks:

- ***Project Management:*** The ability to create new projects and remove undesired contents.
- ***Group Management:*** Capacity to create new groups, expressing different roles inside the project.
- ***Workspace Management:*** Capacity to create different areas, expressing different activities of interest inside the same project. The project manager has the capacity to manage roles and their accesses.
- ***File Navigation:*** Includes the capacity to navigate through the files and folders stored inside the cloud platform.
- ***Content Download:*** All contents, both files and folder, can be downloaded using a simple button.
- ***File Upload:*** Capacity to upload new contents into the cloud's file hierarchy.
- ***Folder Creation:*** In order to organize the information, a user is able to create new folder.
- ***Contents deletion:*** All contents, projects, workspaces, files, and folders, can be deleted.
- ***Reviews:*** A review system allows all customers to comment and see other client reviews about the stored content.

- **Versioning System:** Capacity to store and access different version of the same file, maximizing the utility of the space occupied.

The following figure (Figure 17) illustrates some of the features previously described and a high level vision of the appearance of the product.

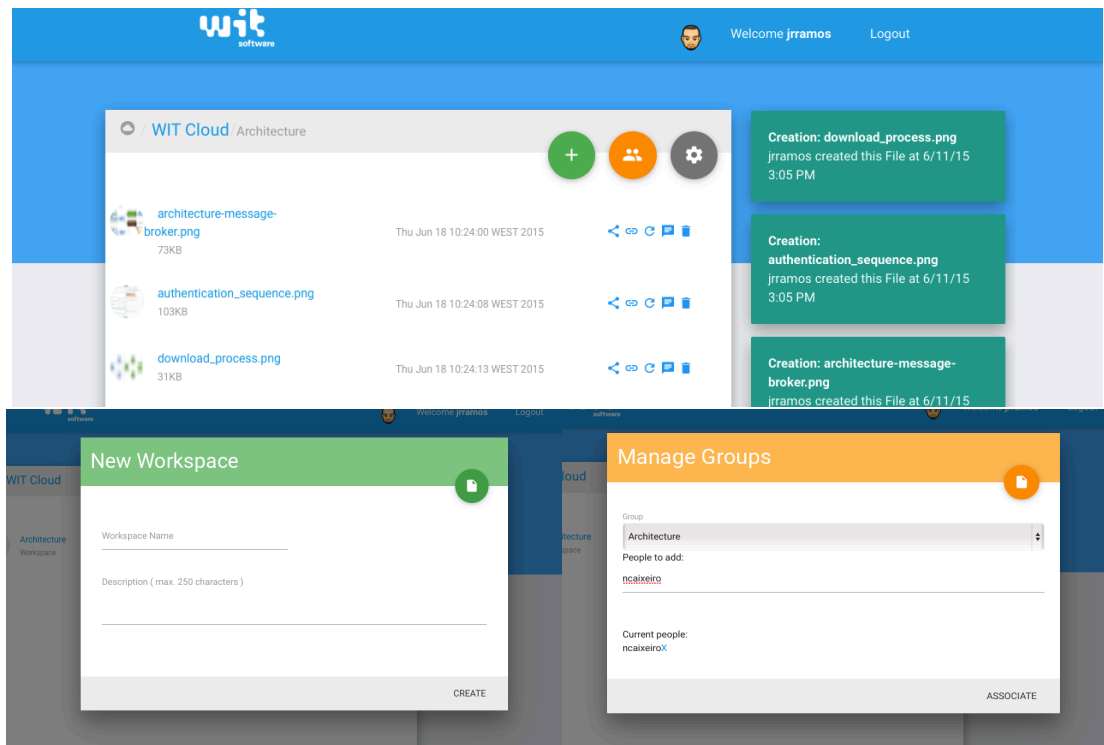


Figure 17 - Screenshots of the interface used to manage and share contents using our platform.

5.1.2 Sharing

One of the most relevant features regarding cloud storage platforms is the ability to share our cloud contents with other persons or groups, registered or not inside the platform. As specified in the *Attachment A – Approach*, the developed product aimed to provide a transparent and simple process to share our content in three different ways: personal sharing, group shared, URL links.

The personal sharing feature allows the user to share a specific content, either folder or file, with a specific person. The email is used to identify the invitee user. After associating a person, this customer is able to see the shared project in the dashboard page, or, if the project is already associated with the user, the contents shared are added to the previously available contents. Inside the project's folder, the customer must be capable of accessing the shared contents.

However, sometimes we might want to share our some information with other entities outside the company. In order to address this problem, the web application signs up a “fake” user. This process requires the following steps:

1. The web application generates a new RSA key pair.
2. The private is encrypted using a new generated 32-bit password.
3. This new password is hashed using SHA-256.
4. The “fake” registry is created by sending the following information:
 - I. Email
 - II. Hashed password;
 - III. Public key;
 - IV. Encrypted private key;
 - V. Digital signature.

The password used to encrypt the RSA private key is showed to the user whom performs the invitation. This password has to be securely transmitted to the end user in order to decrypt the RSA key and setup his new password.

The group-sharing feature allows a project owner to share a specific workspace with an existing group. This feature introduced the capacity to create different project areas and share them to a limited group of elements, without exposing all information stored. The project owner is the only responsible to concede and restrict all accesses, which can be changed at any time of the application.

Sometimes we need to expose some information to an unlimited number of persons, even not registered inside the platform. Thus, sharing a simple URL provides a simple way to grant read-only accesses to an unlimited number of users. This URL can also be password protect, being necessary to introduce a secret value to decrypt the information, and have an expiration date.

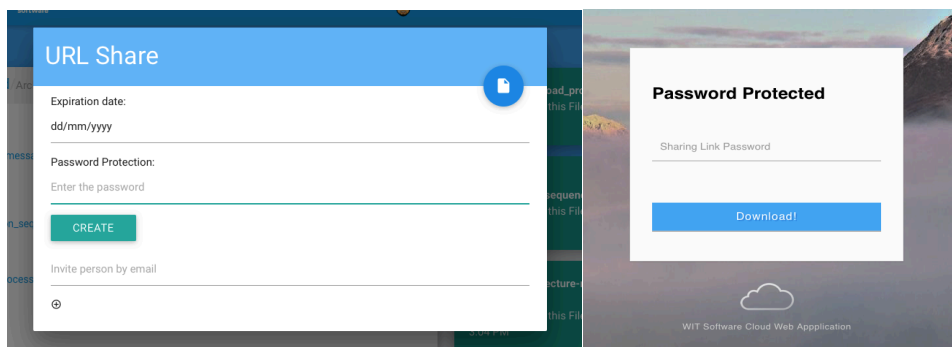


Figure 18 - Screenshots of the interface used to share a URL with a number of unlimited users.

5.1.3 Chat

This product was designed for collaborative work. Communication is a crucial process in order to organize and manage team members and the work in progress. This feature enables the possibility of exchanging messages between members of the same project. A discreet and simple button in the bottom-right corner of the interface opens a new menu that allows us to view and send new messages. However, this simple-looking feature turned out to be very complex in many aspects.

The communication starts with the establishment of the Web Socket between the browser and web application. During this process, the web application intercepts the handshake between both parties in a form of an HTTP UPDATE request. This request is intercepted by Spring and we can override the default procedure. Since we still have access to the HTTP context, we can get the session properties, deny every unauthenticated user and set attributes to the WS context. The connection to the WS does not grant full access to the subscription channels and, consequentially, deny the possibility to read other users messages. After establishing the connection, the user has to subscribe the topics of projects that he wants to receive messages. A message broker installed inside the web application is responsible to manage all incoming and outgoing messages of the WS, using STOMP to provide a meaningful structure to the information. An inbound messages interceptor, installed inside the message broker, intercepts all incoming messages. This interceptor validates each subscription request sent to the server. If the requester user has the required permissions to access the project, he will received all new messages sent to the project's topic. Otherwise, the server generates an exception and denies the subscription request. All other STOMP message types are only accepted if the user already has access to the topic of the project.

Figure 19 shows an example using this feature. As we can observe in these images, depending if we have the chat window opened, the incoming messages can be directly added to our chat menu or shown inside a pop-up window, on the top-right corner of the dashboard page. These pop-up allow us to visualize all incoming messages for a limited short time period, without requiring our chat window to be open.

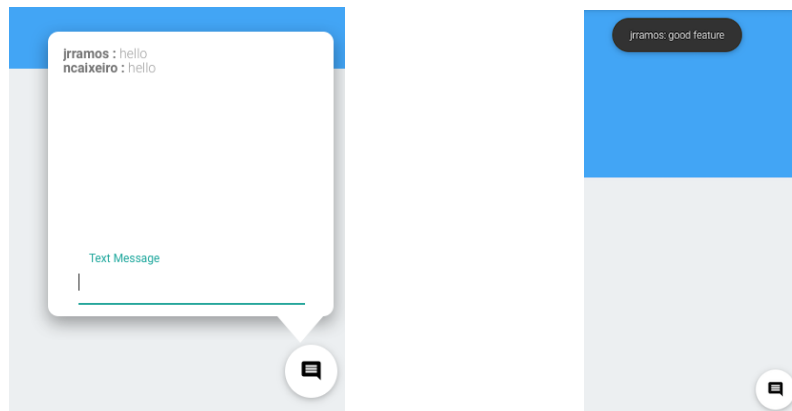


Figure 19 - Screenshots of the developed chat, illustrating a communication between two users.

5.1.4 Auditing

The auditing system allowed us to record and display the operations performed inside a specific level of our cloud's folder hierarchy. However, this simple-looking feature turned out to be very complex in many aspects. Firstly, we had to design a model following the cloud's folders hierarchy. Each level of the tree give us access to the last fifteen auditing operation recorded inside all nodes bellow that node. Then, we needed to conceive a key schema that could encrypt this information.

For this feature, four types of actions are recorded: creation, update, deletion and sharing. Each action changes the way the operation is displayed in the web application.

In order to address the first problem, we designed a solution based on the following principles, as illustrated in figure 20: 1) Each workspace, folder and file have an individual object to store its last fifteen operations or regarding the nodes below it. 2) These operations are added asynchronously to the previous structure after the execution of some methods. This information is not just added to the target node but also in the uppermost elements of the cloud's hieratical structure. The main objective behind this “proliferation” schema was to access directly to all operations inside a workspace/folder, without accessing individually to the operations of each element. This situation would require a signification amount of database queries and consequently increasing the execution time of this operation. However, the process of adding operations is slow, since we have to access every level above the content. Although, this process does not require any synchronization mechanism and can be easily executed using background threads.

Other major challenge regarding this auditing system was the key schema required to encrypt this information. All auditing information is added inside the storage server, which does not have access to any key as plaintext. However, we require to access several different levels of our cloud's files hierarchy in order to associate this operation with all these levels. Thus, the *data key* of the base content is not the most suitable key to encrypt this information, since it is attached to the content of the operation. Thus, we needed to establish a common key accessibly through all access levels of a project. The backlink key allows us to access the backlink key of the folder immediately above the content. Consequentially, we can apply this property until we get the project's root content *backlink key*. Using this key we decrypt the *data key* of the same content. This *data key* will be used to encrypt every content's names inside that specific project, when a workspace /folder/file gets uploaded/created along with the rest of it metadata. Then, we can use this value to identify the content in the auditing system, and every person with access to the project can access it.

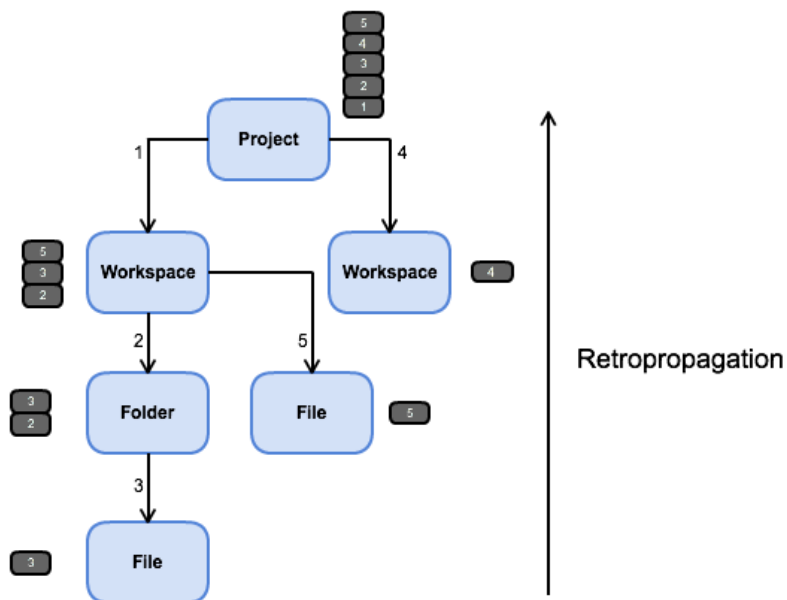


Figure 20 – Schema illustrating the auditing system. The blue boxes represent the elements of our cloud's file hierarchy. The numbers over the arrows contain the creation order of the elements. The black boxes describe the operations present inside the auditing object corresponding to the blue element.

5.2 Laptop Client

This section presents the most important features developed concerning the laptop application component. For each feature we describe and specify some implementation details, its mainly challenges and faced problems.

5.2.1 Authentication and Setup

After the interface design process, the first step was to start developing some of the most basic features regarding the client authentication and application preferences. This application was developed to be the least intrusive to the customer. Thus, we minimized the number of interfaces displayed and saved all its preferences during his first sign in process. Figure 21 displays the interfaces displayed during the application's first instantiation.

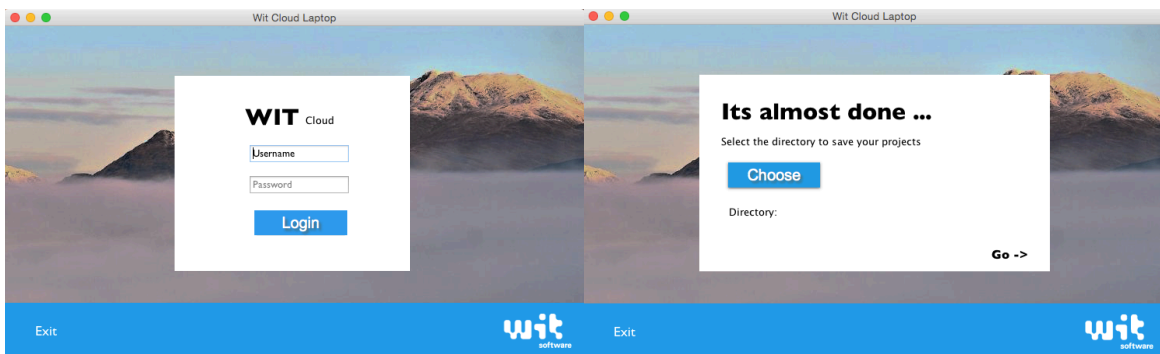


Figure 21 – Screenshots of the laptop application interfaces.

As we can easily observe in the above pictures, the first sign in process can be divided into two fundamental steps: user authentication and local folder specification. The first step, user authentication, is similar to the sign in process of the web application. The user inputs his credentials that are validated by the storage server components. If the authentication succeeds, the user needs to specify the folder where he wants to store the downloaded information regarding his cloud projects.

This data is stored inside two JSON files along with other information, as described in *section 4.6.2*. This file allows us to authenticate the user and automatically contextualize the application about the application's folder, without requiring any input. Therefore, after the first sign in operation, the application starts in “background” without displaying any interface window to the user.

5.2.2 Initial Synchronization

After the first authentication process has been accomplished, the application downloads all projects associated to the authenticated user. As the previous task, we need to establish a connection between the laptop client and storage component.

This initial synchronization process can be divided into three fundamental steps:

- **Get list of projects:** It starts by making a request in order to get metadata information regarding all projects, including the type of access (personal or group) - the same request performed when we access the web application dashboard page.
- **Access project root files:** Firstly, the client downloads the root files of each project. Root files, as specified in section 4.7 – *Storage Server*, contain information about the keys and base content ids. Depending on whether the user has personal or group accesses, the application uses his private or group key, respectively, to decrypt the base key for that specific content
- **Download of each content:** The application downloads specific information inside our projects. Starting on the index of the content that is referenced by the root file, we download the element's metadata. Then, depending on the type of content we have downloaded, one of the following operations is performed:
 - If the content is a workspace or folder, we create a folder inside our local file system and download the metadata of its sub-contents and the process is repeated.
 - Otherwise, the file is downloaded and decrypted.

During this initial download, some specific information regarding these contents, projects and groups downloaded is stored inside a HSQLDB, as explained in the *section - 4.6.2*. This information will be used to perform live updates, as well as in the resynchronization process at the application startup. For this reason and some other performance issues, the project cannot be completely downloaded, as we do when we want to download a folder using the web application, because it lacks of metadata information, so each content is downloaded independently.

This process uses the same API interface methods as the Web Application to access the storage server component in order to access the list of projects, get the base contents of each project, access folders/workspaces and download items. Since the download does not follow a specific order, this process cannot be interrupted. In this case, the application has to restart all the process, checking individually the content already downloaded.

5.2.3 Live Updates

As stated in the *section 4.3 – Web Sockets*, we used Web Sockets with STOMP messages in order to perform real time updates on the laptop application. These messages are generated by the storage component when some specific actions are executed. Then, a message broker is responsible to send them to a list of subscribers. This list contains all events responsible to generate these messages:

- New or deleted project;
- New or deleted workspaces;
- Group association/dissociation;
- New, updated or deleted file/folder;

Since these STOMP messages are consequence of the execution of these methods and not part of their process, we used AOP principles to implement all code regarding the sending process, so that they could be programmed independently and fired asynchronously.

In addition to the structure provided by the STOMP layer, we also enrich the message body following a specific JSON format, containing the following fields: *content id*, *operation* and *project id*. The *operation* field expresses the type of action performed (ex: creation, deletion or modification), the *content id* references the target content of the operation (workspace, content or group), and the *project id*, as the name says, references the project where the action was executed. Independently of the operation, these fields give us enough information to perform all required request to the REST interface of the storage server.

Depending on the nature of the event, this message has to be sent to a determined topic. Each topic expresses a different level of specification on the operation's target. For example, if some new workspace is created inside a project, the project owner should be the only customer to receive this notification. Other customers, independently of their access level, do not have immediate permission to the created workspace and should not be alerted. However, other events can be sent to a large number of customers, such as the upload to a workspace.

The topic also reflects the type of operation to be performed by the application. In order to address this problem, we created 4 different types of topics:

- **User:** this topic receives messages about personal events, more specifically group associations/dissociations. The JSON body of the message identifies the target group and, depending on the type of operation, the application performs one of the following actions:
 - *Group association:* In this case, the application downloads the membership key of the user, decrypts it using the user's private key, adds this information to the database and subscribes the group's topic.
 - *Group dissociation:* In this case, the group is removed from the database, all workspaces associated exclusively to that workspace are deleted and the topic unsubscribed.
- **Group:** This topic receives messages regarding new or removed workspaces from a specific group. The JSON body of the message identifies the target workspace and, depending on the type of operation, the application performs one of the following actions:
 - *Workspace association:* This operation verifies if the workspace is already downloaded or not and, if not, downloads the metadata of the content, decrypts the keys using the group key stored inside the HSQLDB, downloads it and subscribes the workspace's topic. Otherwise, if the workspace is already downloaded, the association between the workspace and the group is created inside HSQLDB.
 - *Workspace dissociation:* This operation eliminates the association between a workspace and a group. This relationship is removed from our HSQLDB and the workspace topic is unsubscribed if there are no other groups with access to that workspace.
- **Project:** This topic is exclusive to the project owner. This topic receives notifications of created and deleted workspaces in our personal projects. Therefore, the JSON body identifies the workspace created or removed and project. Depending on the operation, the following operations can be performed:

- **Workspace creation:** The created workspace is downloaded and performed a new subscription to its topic.
- **Workspace:** This topic receives information about new files or folders uploaded to our projects. The JSON body also identifies the project and workspace. The following option can be performed, depending on the operation:
 - *File/Folder download:* The operation alerts the application about the existence of a new file/folder inside that workspace. The file's metadata gets downloaded and the *parentId* field gives us information about the new content's path.
 - *File update:* When a new file/folder gets updated, the storage server sends a notification regarding this event, this notification has the id of the modified content and its project. Then, a new complete version of the file gets downloaded and the local database is updated with the actual version stored.
 - *File/Folder deletion:* This event is generated when a new file/deleted gets deleted. This files is removed both inside database and file system.

5.2.4 Resynchronization

This section of the report will approach the resynchronization mechanism between storage server and laptop application. This process aims to reproduce all modifications performed inside the storage server to the local file system and vice-versa, after the application start up. The figure bellow, figure 22, gives you a high level schema of the process:



Figure 22 – Schema illustrating the synchronization process

This schema illustrates the synchronization process that can be divided into three fundamental steps:

- **Accesses Verification:** The application verifies the user's accesses for each project. This information is downloaded from the storage server and is compared with the actual data of our local HSQLDB. If a project does not exist, it is downloaded, decrypted and added to our local database and local folder, as in the process of the *section 5.2.2 – Initial Synchronization*. For each

existing project, the application verifies the current accesses of the user, including current groups, the accesses of each group and current workspaces. Finally, all projects no longer available are removed.

- **Auditing Verification:** The application fetches the most recent upload date from our HSQLDB. All information before this timestamp should not be relevant since we are already updated. This information will be used to download auditing information for each workspace currently accessible. This auditing information, as explained in *section 5.1.4 - Auditing*, contains information regarding all operation performed inside a specific level of our file hierarchy. The *last_modified* field will be used to give us element from our database after the specified data. Then, all modified/created/deleted files are downloaded/removed, if the vector clock is bigger than the one present in our HSQLDB.
- **Local Changes:** All files stored inside our workspace folder are scanned. For each file, the application checks the existence of the element in our HSQLDB and, if not, the file gets uploaded to our remote storage server. If the file already exists, we verify the modification date and compare it to the value stored in our HSQLDB. If the file is modified it is uploaded. All removed files are also verified and deleted inside the remote server.

Some of the readers might notice that some remote updates are not downloaded to our local file system, while the resynchronization process. Due to versions incompatibility, we cannot perform any updates without checking if our file is a previous version of the storage server's file. If not, during the *local changes* phase, we upload our local version to the remote server. Therefore, we can have access to these two conflicted versions using the Web Application platform.

6 Software Quality

The validation process of the developed product is a fundamental step on any software project. It is desired that the application is delivered to the user with zero bugs, and with the best UI/UX possible in order to make them enjoy and, consequently, use it.

In this various tests were designed and executed in order to validate the implemented features. All the types of testing that took place in this internship are described in the following sub- sections. The reading of *Appendix C – Software Quality* is highly recommended once it contains the complete set of functional and non-functional tests that were designed and performed.

6.1 Functional Tests

This section of the report describes all functional tests, to validate and ensure the software's quality, using the usually called *black-box* tests.

6.1.1 Acceptance Test

The following test suit contains all acceptance tests developed and performed in order to guarantee that the functional requirements are met. For this motivation, all written tests do not go into great detail since they are intended to be performed by a software quality team. This team aims to validate, in a simple and easily perceptible way, the correct functionality of the developed features. The tester must be aware about what should be the system response for a determined action, discarding all internal logic of the procedure. In other words, what is evaluated is the correctness of the system reaction during our operations.

Despite the acceptance tests are supposed to be performed by a software quality team, as stated before, all the described test were performed by me, by schedule incompatibility motivations.

A total of fifty-nine functional tests were designed and executed in order to check the behavior of all the implemented features within the WIT Cloud platform. These test aimed to test all functionalities implemented in both client components: web application and client laptop.

The testing process has been continuous, meaning that it went along all the implementation. Thus, it is natural that the tests were executed multiple times until all the bugs were corrected. Table 7 describes the results obtained. For details about the acceptance tests performed, refer to *Appendix C – Software Quality*.

Client	Total Tests	First Result	Last Result
Web Application	35	7/28	0/35
Storage Server	25	8/17	0/25

Table 7 - Acceptance tests results. This table describes the number of tests performed on each client and the results of the first and last tests. Red values represent the number of failed tests followed by the number of successful tests.

6.1.2 API Tests

API (Application Programming interface) testing is in many respects like testing software at the user-interface level, only instead of testing by means of standard user inputs and outputs, you use software to send calls to the API, get output, and log the system's response.

The job of the API tester is to test the software knowing only what a user is likely to know. Besides being one of the most common and important test produced during the development of a project, Unit testing aims to make sure that a particular component of a software unit works properly, but, since it is not a complete black-box testing (we see through the code), it is likely to miss something that may be obvious using another type of tests.

API tests test the unit as a part of the system; we are not focused on the detail but in the entire platform as a whole. To produce API tests it is necessary to send a specific call to the API, then compare and log the expected and the obtained results. These tests are usually composed by a set of tests, executed sequentially, and each test should capture API outputs produced during the calls, so we can store these values into several variables and use them in the following test calls. These practices allow us to test more sophisticated environments, such as dynamic environments, where each call usually can depends on previous server's responses.

For this purpose, we used an API test tool named Runscope. Stating Ola Wiberg, company's co-founder and VP of Engineering, "*Runscope verifies that Human API is returning the right data, even for multi-step workflows and chained requests.*"[109], allowing us to build and run our tests using this online tool. This platform also offers several important features, such as saving returned data into variables and use them to make new requests, schedule our tests to be automatically executed, send the obtained results to our email account and managing and adjust the amount of request per second performed to the platform.

Table 8 shows the API testing results on the first and last test runs for all the implemented features. For each test performed, we evaluated both header parameters HTTP body of the server's response. For details about the API tests performed, refer to *Appendix C – Software Quality*.

Controller	Total Tests	First Result	Last Result
Authentication	45	18/27	0/45
Administration	65	27/38	0/65
Navigation	69	21/38	0/69
Sharing	23	6/17	0/23
Search	10	0/10	0/10

Table 8 – API tests results. This table describes the number of tests performed on each controller and the results of the first and last tests. Red values represent the number of failed tests followed by the number of successful tests.

6.1.3 Unit Tests

As stated before, unit test, as the name says, is an excellent method to quickly verify the state of a piece of code (unit) and its interactions in isolation, or in another words, disconnected from the rest of the system. This methodology is widely used to test the smallest amount unit of work, reducing as much as possible the amount of dependencies with another resources, as another system or resources, and then checks a single assumption about the behavior of that unit of work.

Since this platform is installed in an extremely dynamic environment, with a high number of interactions between different parts of the system, we used another kind of functional tests, such as the API, to assure the quality of our system. However, as the development occurs, we performed this kind of tests to validate specific small pieces of code.

The following table contains a full description of the test performed during the project developments:

Class	Functionalities to test	Last Result
AESFunctions	• Encryption	OK
	• Decryption	OK
RSASFunctions	• Encryption	OK
	• Decryption	OK
PBKDF2Functions	• Password derivation	OK
SHA1Functions	• Hashing	OK
CertificateGenerator	• Asymmetric password generation	OK
	• Digital signature generation	OK
ContentGenerator	• Key generation	OK
	• Key encryption	OK
	• Data encryption	OK
Base62	• Base 10 to Base 62	OK
	• Base 62 to Base 10	OK
	• UTF-8 to Base62	OK
	• Base62 to UTF-8	OK
ContentUtil	• Key decryption	OK
TTTDChunker	• Data chunking	OK

Table 9 – Unit Tests table. This table contains all unit tests performed during the project. Each test identifies the tested class, the functionalities tested and the result obtained in the last execution.

6.2 Quality Tests

The functional tests evaluated the behavior of the application by analyzing the system response to our action or requests. In this section we are more interested in evaluating the performance and usability of the application, and thus is more focused on the non-functional requirements.

6.2.1 Usability Tests

In this chapter of the report we will describe and analyze the usability tests performed regarding the user interface (UI) of the platform's web application component.

Usability testing is a technique focused on the evaluation of a product. These tests are done by real users on a controlled environment and represent very useful information, once they give us feedback on how the community will use the designed system.

Firstly, we performed these tests using a population of six individuals and the obtained results were subject of our personal opinion, commenting all gathered observations and tests results. In a second level, we made an introspection exercise about usability, using the ten fundamental usability heuristics, named as 10 heuristics of Nielsen. This last process helped us to detect usability problems related with the UI design, analyzing its compliance with the usability principles. Finally, we repeated the first test using a final version of our product and analyzed the effectiveness of our improvements.

The following topics resumes some of the observations gathered during the first usability tests. All tasks regarding these tests are listed, described, and analyzed in *Appendix C – Software Quality*.

The first prototype test showed that not all tasks were easily completed and some of them could not even be finished. There were some difficulties understanding some of the menus and follow the required steps in each task. The concept behind group and workspaces also was not immediately understood and should be improved. Therefore, the most difficult tasks regarding this first test were related with group operations and access to previous versions of a file. The difficulties in the group management tasks were caused by lack of visibility of the menus to realize the pretended actions. The access to previous version was affected by lack of association between icon and the action and missed an icon to download the content.

The Nielsen principles also showed other associated issues related essentially with some mapping, mental concept and feedback problems. The mapping was mainly affected by lack of association between the pointer type and our application objects. Clickable object should have “clickable” pointers or may not be understood by the clients. All icons also must be identified with a title describing its purpose. The application did not always give feedback of the operations. In fact, some users did not understand if the application was performing some action or the action was not received by the platform. This situation took some users to repeat the operation more than one time or confirmed the operation.

Thus, we developed a second interface model with three main improvements: introduction of a title and “clickable” pointer in every icons; feedback messages whenever it is possible; new group menu. Full a full description about the performed improvements, refer to *Appendix C – Software Quality*.

The second prototype gave us more confidence about the UX/UI of our platform. In fact the results improved substantially, essentially a set of three tasks that revealed to cause some difficulties to the users. Since these results were considered as satisfactory and we did not could identify new major problems, this last prototype was included in the final version of the product.

6.2.2 Software Performance

This section of the report is intended to present a detailed performance analysis of some of the platform's components. All performed test were executed using two virtual machines, as described in the next subsection.

6.2.2.1 *Virtual Machine Configuration*

Web Application

OS: Ubuntu Server Edition v14.04.1 – 64 Bits
CPU: 1 CPU Core Intel Xeon X3363 @ 2.83GHz
RAM: 1 GB

Storage Server

OS: Ubuntu Server Edition v14.04.1 – 64 Bits
CPU: 1 CPU Core Intel Xeon X3363 @ 2.83GHz
RAM: 1 GB

Client

OS: OSX Yosemite v10.10
CPU: Intel Core 2 Duo @ 2.66GHz
RAM: 8GB

The reader is advised to read appendix C- Architecture, in order to fully understand the deployment and interaction between the components of the platform.

All the following tests were performed with a cable connection to the Internet and inside WIT Software's intranet.

6.2.2.2 *Web Application*

This chapter relates to the tests performed to the web application component.

6.2.2.2.1 *YSlow Rules*

“YSlow analyzes web pages and why they are slow based on Yahoo!'s rules for high performance web sites”[110] .This platform measures the performance of a page, examining all components on the page, including those components dynamically included by using JavaScript.

Yahoo!'s rules for high performance web site are composed by 34 topics, but only 23 of these are testable. Each topic can have different impact on the web page response time and consequently have different impact on the overall page's classification

The following table contains a list of the Yahoo!'s topics followed by the YSlow's classification to the login and dashboard pages. These rules are listed in order of importance and effectiveness and each topic has a classification between "A" (highest rank) and "F" (lowest rank):

#	Rule	Login	Dashboard
1	Minimize HTTP Requests	A	B
2	Use a Content Delivery Network	F	F
3	Avoid Empty <i>src</i> or <i>href</i>	A	A
4	Add an Expires or a Cache-Control Header	A	A
5	Gzip Components	F	F
6	Put Style Sheets at the Top	A	A
7	Put Scripts at the Bottom	A	A
8	Avoid CSS Expressions	A	A
9	Make JavaScript and CSS External	A	A
10	Reduce DNS Lookups	A	A
11	Minify JavaScript and CSS	A	A
12	Avoid Redirects	A	A
13	Remove Duplicate Scripts	A	A
14	Configure ETags	A	A
15	Make AJAX cachable	A	A
16	Use GET for AJAX Requests	A	A
17	Reduce the Number of DOM Elements	A	A
18	No 404s	A	A
19	Reduce Cookie Size	A	A
20	Use Cookie-Free Domains for Components	D	E
21	Avoid Filters	A	A
22	Do Not Scale Images in HTML	A	A
23	Make favicon.ico Small and Cachable	A	A

Table 10 – YSlow rules classification. This table contains the classification of the login and dashboard pages for each Yahoo!'s high performance web sites rules.

There are three rules that deserve our special attention, rules number two, five and twenty, once they have a bad score. Regarding the rule number two, *Use a Content Delivery Network*, scored with F on both pages, the solution to solve the problem involves additional costs for optimization, since it requires to deploy content across multiple geographically dispersed servers and, due to this fact, it is out of our scope.

The rule number five, *GZIP components*, allow us to reduce the response time of the page by compressing the size of the HTTP responses, using GZIP as the compression algorithm. Despite all these benefits, there are several vulnerabilities associated with this technique, such as CRIME and BREACH, which could be explored in HTTPS connections. By this reason, this rule was also not deeply explored.

Finally, the rule number twenty, *Use Cookie-Free Domains for Components*, is related to the fact that when we are accessing static server resources we are also transmitting the

session cookie, which is ignored by the server and increases the network traffic. To workaroud this problem, is necessary to create a subdomain and host those static components there. Since this procedure requires a significant amount of work and the overhead introduced by the cookie is considerably low, we choose to not implement the necessary changes.

Regarding the overall score of the two pages tested, it can be said that the grade is very optimistic and represents a good optimization effort done (Figure 6).

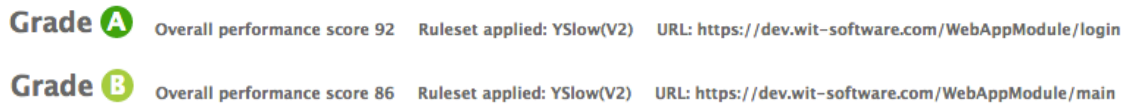


Figure 23 – Login and Dashboard pages YSlow’s overall score. The classification has a rank between A (highest classification) and F (lowest classification).

6.2.2.2.2 HTTP Requests

This section of the report related to the analysis made to the HTTP requests in a cached and not cached environment for the login and dashboard pages.

The following graphs represent information about the amount of data downloaded during the *login* and *dashboard* pages loading (Figure 24 and 25).

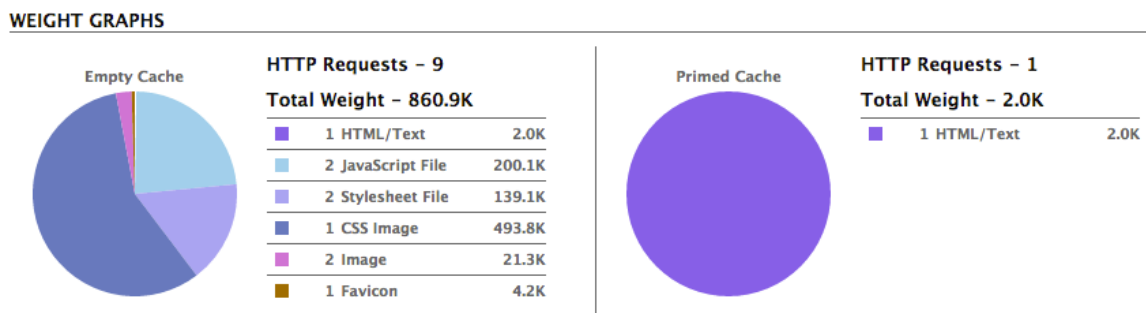


Figure 24 – Graphical representation of the information downloaded during the access to the login page, using a cached and not cached environments

WEIGHT GRAPHS

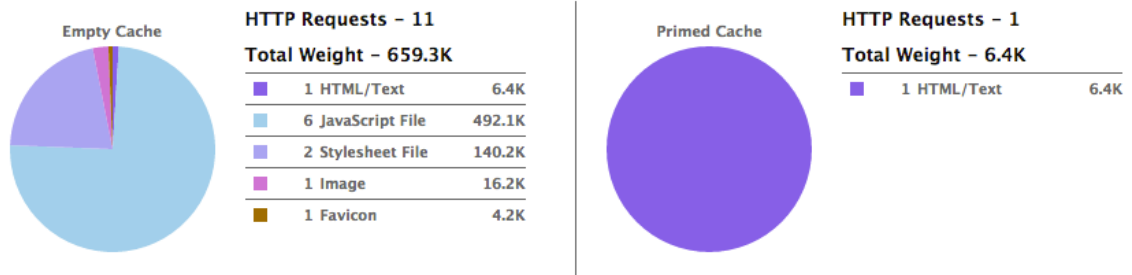


Figure 25 - Graphical representation of the information downloaded during the access to the dashboard page, using a cached and not cached environments

By analyzing the graph of the figure 24 and 35, we can see that by caching all components, except the HTML information that is mandatory, there is a gain of 430,45% on the number of information transferred from the Web App component, comparing with the non-cacheable version, being a good performance improvement. Regarding the dashboard page, the improvement was not so significant being around the 103,02%.

All requests performed during the AJAX calls are not described in the previous graph, however the content fetched during those calls is mostly non cacheable because it is constantly changing.

6.2.2.2.3 Benchmarking

This section of the report relates to the benchmark tests developed to collect some information about the performance of our product. In a way to compare the obtained results, we also tested the WIT Software's current solution: *OwnCloud*. *OwnCloud* was installed in a virtual machine with the same hardware and default apache configurations as specified in the *section 6.2.2.1*.

To evaluate the performance of the Web Application component, we execute three different benchmarking tests, pursuing different motivations: 1) Page loading times; 2) Upload performance; 3) Removal performance. The next three chapters of this report will describe the purpose and results for each of these tests.

6.2.2.2.3.1 Page Loading Times

This section of the report aims to benchmark the server's response for the login page requests. To perform this test we used the Apache Benchmark Tool (*ab*) to create the requests, with the specific number of concurrent connections, and analyze the server's performance for each of the tested platforms. The test followed the following principles:

- The total amount of request for each test was 10.000.
- The number of concurrent connection was increased during the tests.
- Each test was repeated five times and the values displayed represent the average of those values.

The following tables (table 11 and 12) present the benchmark results obtained from the benchmark test.

WIT Cloud Web Application

Concurrent Connections	Total Time (sec)	Time / Request (ms)	Requests / Second (#/sec)	Failed Requests
20	6,739	13,477	1484,00	0
50	5,849	29,244	1709,73	0
100	4,647	46,471	2151,86	0
200	5,581	111,625	1819,47	0
500	21,906	1095,303	456,49	15
700	23,590	1651,322	423,90	21
1000	23,652	2365,220	422,79	173
2000	27,108	5421,566	368,90	360
4000	30,268	12107,246	330,38	1301

Table 11 - WIT Cloud Application page loading results. Each test performed 10.000 requests to the index page of the Web Application component, varying the number of concurrent connections. This tables describes information about the execution time, time per request, request per second and number of failed requests.

Own Cloud

Concurrent Connections	Total Time (sec)	Time / Request (ms)	Requests / Second (#/sec)	Failed Requests
20	10,561	21,122	946,86	0
50	4,503	22,517	2220,51	0
100	4,186	41,865	2388,64	0
200	4,384	87,678	2281,07	0
500	16,368	818,378	610,96	0
700	24,571	1719,952	406,991	156
1000	24,465	2446,495	408,75	241
2000	25,389	4077,898	393,86	917
4000	24,435	9773,822	409,26	2316

Table 12 - Own Cloud page loading results. Each test performed 10.000 requests to the index page of the Own Cloud, varying the number of concurrent connections. This table describes information about the execution time, time per request, request per second and number of failed requests.

As we can observe by analyzing the previously table, both platforms had similar time responses and can be divided in two distinct phases, depending on the level of concurrence. Using default apache configuration the first 4 levels of concurrent requests, 20 – 200, are taking advantage of the apache thread pool, which gets expanded until a maximum of 200 threads. The maximum throughput happens in the third level of concurrence, with the requests getting returned without require expanding the base pool.

Between the forth and fifth request (200 and 500 concurrent connections) tomcat initiates 400 new threads, being used as its full strength (1 thread per request), the

throughput reduces substantially, as we can see in the fifth pair of columns of the image 9. The reason for this substantial reduction of performance is due to the server starts to have some difficulties dealing with this many threads concurrently, expanding and reducing the pools dynamically as the requests happens.

As we increase the number of threads, the throughput decreases even more (increasing the response time), since there are not enough threads to handle all incoming requests and they stay in an idle state until another request is freed up. All these processes of pool expansion/reduction are the main responsible for this decrease of performance to answer all requests. This difficulty to answer to all these requests also influences the amount of failed request, starting to appear after the 700 concurrent connections.

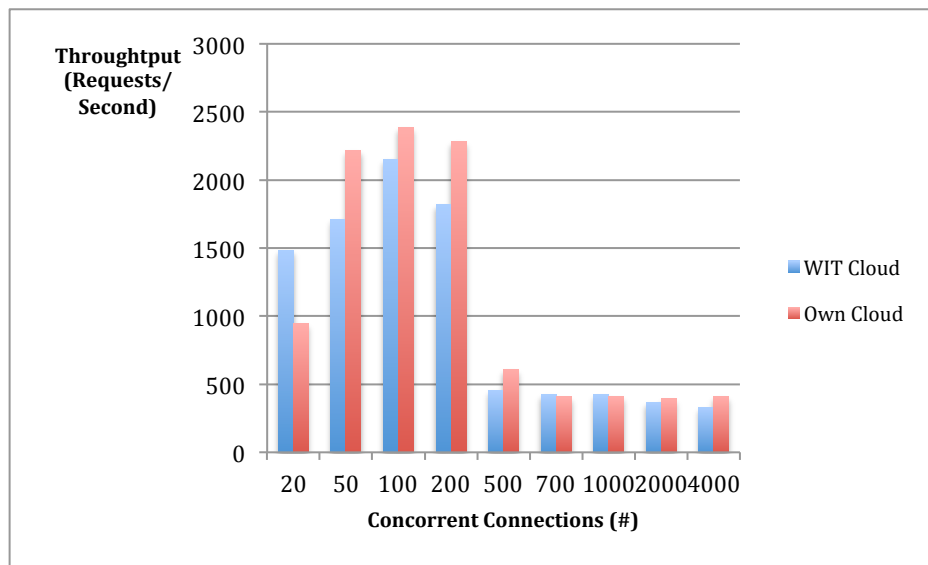


Figure 26 - Graphical representation of WIT Cloud's and Own Cloud's page loading performance tests. Each test performed 10 000 request to the server, with a variable number of concurrent connections. The x-axis describes the number of concurrent connections of the test, and the y-axis the throughput as the number of request (10 000) divided by the execution time of the request.

6.2.2.2.3.2 Upload Performance

Since one of the main features of this project is the possibility to store information using an external storage service without compromising our data, the upload benchmark tests used the Amazon Web Services as an external storage platform on both tested products, WIT Cloud and Own Cloud.

In the *Own Cloud*'s case, the Amazon connector uses a single service, Amazon S3, to manage the uploaded information inside the server. Therefore, it is not required a database service since all information, including content's metadata and folder navigation, is fetched from the S3 web service. In our case, as explained on the section 4.1, we use the Dynamo DB beyond the Amazon S3. For more information about the storage database schema information stored, please consult Appendix B - Architecture.

Finally, we also activated the *Own Cloud* server side encryption using the schema [111], being very close to the one used in our platform. Each test was repeated five times and the values displayed represent the average of those values.

The following table presents the results for the upload test:

File Size (KB)	WIT Cloud (Seconds)	OwnCloud (Seconds)
500	16,203	31,294
1 000	12,121	32,474
10 000	16,949	93,517
30 000	42,105	123,663
50 000	48,273	192,076
100 000	67,490	> 200

Table 13 – WIT Cloud and Own Cloud upload performance results. Each row contains the size of the uploaded content and request completion time on each platform.

The previously described table shows that, in all tested scenarios, our platform outperformed completely the concurrent platform, Own Cloud. We also performed other tests, running the same examples but with Own Cloud's Server-Side Encryption app disabled, presenting a time reduction of 172%, 188% and 357% on the first, second and third requests, respectively. These results revealed an extreme reduction of performance running Own Cloud with the data server encryption daemon activated[112]. Own Cloud uses a similar key schema as we do, but do not implement data chunking and multithreaded encryption and upload.

6.2.2.2.3.3 Removal Performance

The following table presents the recorded times from the file removal tests. The execution of these tests involved the deletion of six files, with different sizes: 500KB; 1 000 KB; 10 000KB; 30 000KB; 100 000KB. Each test was repeated five times and the values displayed represent the average of those values.

File Size (KB)	WIT Cloud (Seconds)	OwnCloud (Seconds)
500	4,045	33,082
1 000	3,525	29,267
10 000	4,751	30,331
30 000	5,315	99,511
50 000	4,713	116,759

Table 14 - WIT Cloud and Own Cloud content deletion performance. Each row contains the size of the removed content and request execution time on both platforms.

As we can observe by examining the previous table, WIT Cloud performed all database deletion operations with a lot more efficiency than the tested concurrent platform. Examining the deletion process of both platforms, we can easily understand this significant difference between them. As was already mentioned before, these platforms use different Amazon services and this fact have huge implication how the information is handled by both platforms. Whereas Own Cloud submit synchronous requests directly to the Amazon S3, waiting until the Web Server removes all the requested contents and returns the proper answer, in our case the operation is divided in two different steps. When the request reaches our server, the DynamoDB reference between the parent folder and the removed content is deleted and the operation returns immediately. All contents stored inside the Amazon S3 service or other possible files inside the removed content are deleted asynchronously and do not have impact on the response time. Therefore our implementation has an execution time independent of the content size and all “size dependent” operation, such as the deletion request to the Amazon S3 Web Service, is executed on the background.

6.2.2.2.3.4 Additional remarks

Other problems related with the use of Amazon S3 as the external storage provider for Own Cloud is the absurd amount of requests required to sync files between all Own Cloud client applications[113]. Since Amazon S3 charges for the amount of requests performed, this situation is not ideal and is avoided in our application by using Web Sockets to sync all clients.

Some Own Cloud forums[114] also described some problems during the folder navigation experience using Owncloud’s WebUI, mainly caused by the Amazon S3 inefficiency for that purpose.

6.2.2.3 Storage Server

In this section of the report we will analyze the performance of our storage server during the execution of queries to the relational database. To measure this component we selected ten queries, with a substantial level of complexity and frequently requested during the server's activity. This information allows us to understand if the database schema was designed properly, with the correct relations between tables and the right selection of indexes. Therefore, these tests allow us to assure a certain level of confidence on the performance of the storage component in a highly stressful situation, with a lot of simultaneous and concurrent connections, and different accesses to the same table, requesting different information.

To perform these tests the database was populated with pseudorandom generated information, simulating a context with 4000 users, 4000 groups and 1000 projects, with a ratio of 5 users per project. Each project also has an average of 5 individual and 5 group accesses to its contents. These tests executed all different queries sequentially and each query had a variable number of 200, 400, 800 and 1000 concurrent connections. Each test was repeated three times and the presented results contain the average value of those results.

6.2.2.3.1 Queries performance

The following table describes the ten queries selected to perform the storage server database performance tests, as well as their execution time, in milliseconds, for 200, 500, 800 and 1000 concurrent threads. These queries were executed under the conditions specified in the *section 6.2.2.1* of the current report.

#	Query	Concurrent Threads / Execution time (ms)	
		200	275
1	SELECT project.id FROM Projects project WHERE project.name=:name	200	275
		400	434
		800	857
		1000	931
2	SELECT group_entity.id, group_entity.name FROM Projects project, Groups group_entity WHERE project.id=:projectId and group_entity.project=project	200	1296
		400	2589
		800	5453
		1000	6442
3	SELECT rootfile.fileIndex FROM RootFiles rootfile, Entities entity, Projects project WHERE rootfile.entity=entity and rootfile.projects=project and project.id=:idProject and entity.id=:idEntity and project.projectManager=entity	200	211
		400	481
		800	877
		1000	913
4	SELECT user_entity.id,user_entity.name FROM Groups group_entity,MembershipKeys membershipKeys, User user_entity, Projects project WHERE group_entity.project=project and project.id=:projectId and membershipKeys.usersId=user_entity and membershipKeys.groupsId=group_entity and group_entity.id=:groupId and user_entity.id!=:userId	200	685
		400	1453
		800	2741
		1000	3019

5	SELECT membershipKeys FROM Groups group_entity,MembershipKeys membershipKeys, User user_entity, Projects project WHERE group_entity.project=project and project.id=:projectId and membershipKeys.usersId=user_entity and membershipKeys.groupsId=group_entity and group_entity.id=:groupId and user_entity.id=:userId	200	1998
		400	1260
		800	1873
		1000	1349
6	SELECT project.id, project.name, groups.id, groups.name FROM Projects project,Entities user, Entities groups, MembershipKeys membership WHERE user.id=:idPerson and membership.usersId = user and membership.groupsId = groups and groups.project = project	200	2961
		400	773
		800	1165
		1000	1831
7	SELECT rootFile FROM RootFiles rootFile, Projects project,Entities group_entity WHERE rootFile.projects=project and rootFile.entity=group_entity and rootFile.fileIndex=:fileIndex and project.id=:projectId and group_entity.id=:groupId	200	1075
		400	1155
		800	1257
		1000	1128
8	SELECT membershipKeys.password FROM MembershipKeys membershipKeys, User user, Groups group_entity, Projects project WHERE membershipKeys.usersId=user and membershipKeys.groupsId=group_entity and user.id=:idUser and group_entity.project=project and project.id=:idProject and group_entity.id=:groupId	200	291
		400	669
		800	972
		1000	1014
9	SELECT DISTINCT project.id, project.name, project.creationDate, project.projectManager FROM Projects project, RootFiles rootFiles, Entities user WHERE project = rootFiles.projects and user.id=:idPerson and rootFiles.entity = user	200	1311
		400	2357
		800	4747
		1000	5432
10	SELECT DISTINCT project.id, project.name, project.creationDate, groups.name, groups.id, project.projectManager FROM Projects project, RootFiles rootFiles,Entities user, Entities groups, MembershipKeys membership, User pm WHERE project = rootFiles.projects and project.projectManager = pm and user.id=:idPerson and membership.usersId = user and membership.groupsId = groups and rootFiles.entity = groups and pm!=user)	200	2336
		400	4441
		800	9218
		1000	11120

Table 15 – Query performance table. This table describes all tested queries and their execution time (milliseconds). Each query was executed simultaneously with 200, 400, 800 and 1000 concurrent connections, in independent thread, requesting different information.

As we can deduce by analyzing the previously described results, the query with more execution time was the tenth task, with 11,120 seconds and 1000 concurrent connections, an average time of 0,01120 seconds per connection ($TOTAL_EXECUTION_TIME / NUMBER_OF_CONCORRENT_CONN$). This query was already expected as one of the most expensive, since it relates information present in four different tables (although all entities fetched are using indexed fields) and in the case of the *entities* table is fetching three different rows: *groups*, *user* and *pm*.

Since this time does not represent a substantial amount of time and consequently a possible bottleneck of the quality of service provided by our platform, we considered these results as acceptable and did not perform any changes on the database schema.

7 Conclusion Remarks and Future Steps

This final chapter of the reports marks the end of my internship at WIT Software. It is also a good opportunity to make an overview about my work developed during this nine-months project. We approach two different analyses regarding this project.

In a first section we will describe how this product addresses the problems identified in the beginning of this report. Then, we will be present some suggestions about the work that can still be done, underlying some new features that could enrich the product and improve some of the developed work.

7.1 Completed Mission

The main purpose of this project was to address some company's security concerns about having data stored outside their own facilities. Our product should also provide a complete set of tools to help them to manage their projects efficiently, namely to: have access control, sharing and activity logging control. With this goal in mind, our first step was to design a web client and a storage server, based on a flexible model, capable of implementing a complete list of features in order to store, manage and control all information stored inside our platform, improve it with other communicate features, without losing any confidentiality. In order to make this platform even more complete, we added a new laptop client capable of turning available and manage all information regarding the projects inside your local file system.

This list contains a description of some of the most relevant functionalities developed during this project:

- **Key Schema:** This schema was one of the most important functionalities developed during all project. It guaranties all security concerns regarding our information and can transmit some potential limitation of the entire application. These limitations can be performance issues or even incapacity to develop some features of the application due to schema incompatibilities.
- **File Management:** This was the first feature to be implemented. It allows us to upload, download and navigate through our cloud's files hierarchy. The concept behind this feature seems to be very straightforward. However, we faced several problems to construct a suitable schema in order to keep our file tree structure, encrypt information, derive required keys, specify the required metadata and manage fault tolerance situations. This was also my first contact with non-relational databases, in this case MongoDB, and Spring framework.
- **Project Management:** This feature was also introduced during the first semester and allowed us to enrich the product with several new concepts. We enabled the creation and management of different groups and workspaces inside a project. This stage allowed us to accomplish some of the most relevant functionalities of this entire project: extend the number of collaborators in a given project; specify each collaborator's access level; group identical access levels into the same role; differentiate access types.

- **Sharing:** Unlike the previous features, this new functionality allows us to share our information with unregistered customers. What we thought to be a relatively easy collection of features to implement, turned out to be one of the most difficult in the entire project. In order to allow us sharing invitations with an unregistered client, we had to develop a new kind of intermediary registry. This registry allowed us to create the “fake registry”, already described in this report that can be decrypted using an access token, without disclosure any information inside the storage server.
- **Auditing:** All actions performed inside the cloud’s projects are tracked by an auditing mechanism. The information recorded shows us the activity of a specific folder inside our cloud’s file hierarchy. The encryption of this information as well as the structure behind this concept brought us some complex challenges.
- **Chunking, Parallelism and Versioning:** The main purpose of these tasks aimed to improve the performance, effectiveness and robustness of our product. The introduction of chunking mechanisms allowed us to maximize the utility of a given amount of storage. The parallelism optimizes the time spent to performing some operations. Finally, the versioning took advantage of the previously developed chunking system and let us get access to previous version of a file. Since I had no previous experience in none of the topic, it required the study of some techniques and approaches to apply in this project
- **Laptop:** At last, but not the least, the laptop application. This application used the well defined REST API and WS communication in order to perform real time synchronizations and start synching tasks. This application allows us to manage our projects, folders, workspaces and files using two distinct alternatives: web application client and laptop client. This work allowed me to consolidate some knowledge regarding synchronizations mechanisms, asynchronous messaging technologies and study some of the new solutions brought by Java NIO.

Finally, we can also refer the results obtained during our experiments comparing with company’s present solution: Owncloud. In fact, we improved significantly the content browsing operations, as well as the new uploads to the cloud’s files hierarchy. However, these results are the output of a very hard growth and learning process in order to study different approaches to the problem, specially regarding the and key schema data structure and technologies involved. Consequently, we can proudly say: mission completed!

7.2 Future Work

The platform had a significantly reduced development period compared with the competitors and only one team member composed the development team. Thus, it is not expected that this platform have a comparable set of features than other projects.

Thus, there are some opportunities to improve this product. A mobile application could allow customers to navigate through the cloud’s file system hierarchy and share some of its contents. We also could also introduce a preview system. The main objective behind this last features is to allow the visualization of some specific formats, such as text files or PDFs.

Some of the developed features could also or may be improved. The file upload and download processes are significantly memory expensive. All operations performed in these

two processes are executed using exclusively volatile memory. Thus, the size of the file is limited to the JVM space and memory already occupied by other Java processes. Considering the upload process, the space occupied can exceed 2.5x more than the original size of the content. Since this application is supposed to be installed inside a small/medium company, with a significant number of concurrent connections, this can be considered a severe limitation of the application.

Finally, one last improvement regarding the development process. This application did not use a build automation system, such as Gradle or Maven. These systems can help the developer to declare and execute all tasks necessary to compile, test and package an application. Thus, they help us to reduce the quantity of time spent in those tasks and focus on improving our application.

Bibliography

1. Guardian, T. *Snowden: Dropbox is hostile to privacy, unlike 'zero knowledge' Spideroak*. 2013 [cited 2015 25/6/2015]; Available from: <http://www.theguardian.com/technology/2014/jul/17/edward-snowden-dropbox-privacy-spideroak>.
2. Locke, J. *The Roots of Cloud Computing*. 2013 [cited 2015 25/6/2015]; Available from: <http://www.servercloudcanada.com/2013/10/the-roots-of-cloud-computing/>.
3. Amazon. *What's New | 2006*. 2006 [cited 2015 25/6/2015]; Available from: <http://aws.amazon.com/pt/about-aws/whats-new/2006/>.
4. Google. *Google Apps highlights 11/6/2009*. 2009 [cited 2015 25/6/2015]; Available from: <http://googleblog.blogspot.pt/2009/11/google-apps-highlights-1162009.html>.
5. Techbyt. *History of Clouds*. 2012 [cited 2015 25/6/2015]; Available from: <https://techbyt.wordpress.com/2012/03/07/history-of-clouds/>.
6. NIST. *The NIST Definition of Cloud Computing*. 2011 [cited 2015 25/6/2015]; Available from: <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>.
7. Rackspace. *Open Cloud Technologies Spark Innovation*. 2013 [cited 2015 25/6/2015]; Available from: <http://www.rackspace.com/blog/open-cloud-technologies-spark-innovation/>.
8. Internap. *Cloudy Colo: The Data Center Services Landscape*. [cited 2015 25/6/2015]; Available from: <http://www.internap.com/resources/cloudy-colo/>.
9. Blog, G. *How Cloud Computing Reduces Costs and Increases Value*. 2012 [cited 2015 25/6/2015]; Available from: <http://www.golime.co/blog/bid/136271/How-Cloud-Computing-Reduces-Costs-and-Increases-Value>.
10. Post, T.W. *NSA infiltrates links to Yahoo, Google data centers worldwide, Snowden documents say*. 2013 [cited 2015 25/6/2015]; Available from: http://www.washingtonpost.com/world/national-security/nsa-infiltrates-links-to-yahoo-google-data-centers-worldwide-snowden-documents-say/2013/10/30/e51d661e-4166-11e3-8b74-d89d714ca4dd_story.html.
11. Post, T.W. *NSA slides explain the PRISM data-collection program*. 2013 [cited 2015 25/6/2015]; Available from: <http://www.washingtonpost.com/wp-srv/special/politics/prism-collection-documents/>.
12. Perrin, C. *The CIA Triad*. 2008 9/10/2014 [cited 2015 25/6/2015]; Available from: <http://www.techrepublic.com/blog/it-security/the-cia-triad/>.
13. OECD. *OECD Guidelines for the Security of Information Systems and Networks: Towards a Culture of Security*. [cited 2015 25/6/2015]; Available from: <http://www.oecd.org/internet/ieconomy/15582260.pdf>.
14. NIST. *Engineering Principles for Information Technology Security (A Baseline for Achieving Security), Revision A* 9/10/2014; Available from: <http://csrc.nist.gov/publications/nistpubs/800-27A/SP800-27-RevA.pdf>.
15. Kim, S.F.Y.a.P.T.-h. *IT Security Review: Privacy, Protection, Access Control, Assurance and System Security* 9/10/2014; Available from: http://www.sersc.org/journals/IJMUE/vol2_no2_2007/2.pdf.
16. Boritz, J.E., *IS practitioners' views on core concepts of information integrity*. International Journal of Accounting Information Systems, 2005. 4(6): p. 260-279.
17. Oke, G.L.a.G., *Protection against Denial of Service Attacks: A Survey*. The Computer Journal, 2010. 53(7): p. 1020-1037.
18. Caelli, A.M.a.W., *Non Repudiation in Digital Enviroments* First Monday, 2000. 5(8).

19. Andrew, K., et al., *Parity lost and parity regained*, in *Proceedings of the 6th USENIX Conference on File and Storage Technologies*. 2008, USENIX Association: San Jose, California. p. 1-15.
20. Andy, C., et al., *An empirical study of operating systems errors*. SIGOPS Oper. Syst. Rev. %@ 0163-5980, 2001. **35**(5): p. 73-88.
21. Junfeng, Y., S. Can, and E. Dawson, *EXPLODE: a lightweight, general system for finding serious storage system errors*, in *Proceedings of the 7th symposium on Operating systems design and implementation* %@ 1-931971-47-1. 2006, USENIX Association: Seattle, Washington. p. 131-146.
22. Vijayan, P., C.A.-D. Andrea, and H.A.-D. Remzi, *Analysis and evolution of journaling file systems*, in *Proceedings of the annual conference on USENIX Annual Technical Conference*. 2005, USENIX Association: Anaheim, CA. p. 8-8.
23. R. Card, T.T.o., and S. Tweedie. *Design and Implementation of the Second Extended Filesystem*. 1994 [cited 2015 2576/2015]; Available from: <http://e2fsprogs.sourceforge.net/ext2intro.html>.
24. Timothy, E.D., C.A.-D. Andrea, and H.A.-D. Remzi, *Journal-guided resynchronization for software RAID*, in *Proceedings of the 4th conference on USENIX Conference on File and Storage Technologies - Volume 4*. 2005, USENIX Association: San Francisco, CA. p. 7-7.
25. MySQL. *MySQL Encryption Functions*. 2014 [cited 2014 4/10/2014]; Available from: <http://dev.mysql.com/doc/refman/5.5/en/encryption-functions.html>.
26. Oracle. *Oracle TDE*. 2014 [cited 2014 4/10/2014]; Available from: <http://www.oracle.com/technetwork/database/security/twp-transparent-data-encryption-bes-130696.pdf>.
27. Patel, D.R., *Information Security: Theory and Practice*. 2008.
28. Rivest, R., *The MD5 Message-Digest Algorithm*. 1992, RFC Editor.
29. Dobbertin, H., *The Status of MD5 After a Recent Attack*. CryptoBytes, 1996. **2**(2).
30. Xiaoyun, W. and Y. Hongbo, *How to break MD5 and other hash functions*, in *Proceedings of the 24th annual international conference on Theory and Applications of Cryptographic Techniques* %@ 3-540-25910-4, 978-3-540-25910-7. 2005, Springer-Verlag: Aarhus, Denmark. p. 19-35.
31. J. Black, M.C.a.T.H., *A Study of the MD5 Attacks: Insights and Improvements*. 2004.
32. Benne de Weger, X.W.a.A.L., *Colliding X.509 Certificates based on MD5-collisions*. 2006.
33. Rivest, N. *hashlib - faster md5/sha, adds sha256/512 support*. 2005 [cited 2015 25/6/2015]; Available from: <https://mail.python.org/pipermail/python-dev/2005-December/058850.html>.
34. Poulsen, K. *Researchers Use PlayStation Cluster to Forge a Web Skeleton Key*. 2008 [cited 2015 25/6/2015]; Available from: <http://www.wired.com/2008/12/berlin/>.
35. CERT. *Vulnerability Note VU#836068*. 2008 [cited 2015 25/6/2015]; Available from: <http://www.kb.cert.org/vuls/id/836068>.
36. Florent, C. and J. Antoine, *Differential Collisions in SHA-0*, in *Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology* %@ 3-540-64892-5. 1998, Springer-Verlag. p. 56-71.
37. Schneier, B. *Cryptanalysis of SHA-1*. 2005 [cited 2015 25/6/2015]; Available from: http://www.schneier.com/blog/archives/2005/02/cryptanalysis_o.html.
38. Schneier, B. *NIST Hash Workshop Liveblogging*. 2005 [cited 2015 25/6/2015]; Available from: http://www.schneier.com/blog/archives/2005/11/nist_hash_works_4.html.
39. Cryptopp. *Crypto++ 5.6.0 Benchmarks*. 2009 [cited 2015 25/6/2015]; Available from: <http://www.cryptopp.com/benchmarks-amd64.html>.
40. Knebl, H.D.a.H., *Introduction to Cryptography*. 2007.

41. IBM. *Java security: Java security, Part 1: Crypto basics*. 2002 [cited 2014 4/10/2014]; Available from: <http://www.ibm.com/developerworks/java/tutorials/j-sec1/j-sec1.html>.
42. FIPS. *DATA ENCRYPTION STANDARD (DES)*. 1999 [cited 2015 25/6/2015]; Available from: <http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf>.
43. Foundation, E.F. *RSA Code-Breaking Contest Again Won by Distributed.Net and Electronic Frontier Foundation (EFF)*. 1999 [cited 2014 4/10/2014]; Available from: https://w2.eff.org/Privacy/Crypto/Crypto_misc/DESCracker/HTML/19990119_deschallenge3.html.
44. SciEngines. *Break DES in less than a single day*. 2008 [cited 2015 25/6/2015]; Available from: <http://www.sciengines.com/company/news-a-events/74-des-in-1-day.html>.
45. Page, D., *Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel*. IACR Cryptology ePrint, 2002: p. 169.
46. Baum, M. *NIST Withdraws Outdated Data Encryption Standard*. 2005 [cited 2015 25/6/2015]; Available from: http://www.nist.gov/itl/fips/060205_des.cfm.
47. NIST. *Announcing development of a federal information processing standard for advanced encryption standard*. 1997; Available from: http://csrc.nist.gov/archive/aes/pre-round1/aes_9701.txt.
48. NIST. *Announcing the ADVANCED ENCRYPTION STANDARD (AES)* 2001 [cited 2015 26/5/2015]; Available from: <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>.
49. IBM. *Triple DES Encryption*. 2010 [cited 2015 25/6/2015]; Available from: http://www-01.ibm.com/support/knowledgecenter/SSLTBW_1.12.0/com.ibm.zos.r12.csfb400/tdes1.htm%23tdes1.
50. Kesarwani, M.M.a.A., *Comparison Between DES , 3DES , RC2 , RC6 , BLOWFISH and AES* CIBIMA, 2009. **8**(8).
51. Schneier, B., *Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)*. 1993.
52. Gonzalez, T., *A Reflection Attack on Blowfish*. Latex Class Files, 2007. **6**(1).
53. Manap, O.K.a.C., *A New Class of Weak Keys for Blowfish*. Lecture Notes in Computer Science, 2007. **4593**: p. 167-180.
54. Microsoft. *Overview of SLS/TLS Encryption*. 10/4/2014; Available from: [http://technet.microsoft.com/pt-pt/library/cc781476\(v=ws.10\).aspx](http://technet.microsoft.com/pt-pt/library/cc781476(v=ws.10).aspx).
55. pgp.net. *Chapter 8. Public Key Servers*. 4/10/2014; Available from: <http://www.pgp.net/pgpnet/pgp-faq/pgp-faq-keyservers.html>.
56. GNU. *The GNU Privacy Handbook*. 4/10/2014; Available from: <https://http://www.gnupg.org/gph/en/manual.html>.
57. Miller, G.L., *Riemann's Hypothesis and Tests for Primality*. 1976.
58. bit-tech.net. *Researchers crack 768-bit RSA*. 2010; Available from: <http://www.bit-tech.net/news/bits/2010/01/13/researchers-crack-768-bit-rsa/1>.
59. Joppe W. Bos, M.E.K., Thorsten Kleinjung, Arjen K. Lenstra, and Peter L. Montgomery, *On the Security of 1024-bit RSA and 160-bit Elliptic Curve Cryptography*. 2010.
60. Stiglic, J.-F.R.a.A. *Security Issues in the Diffie-Hellman Key Agreement Protocol*. 2002 [cited 2015 25/6/2015]; Available from: http://www.researchgate.net/publication/2401745_Security_Issues_in_the_Diffie-Hellman_Key_Agreement_Protocol.
61. Laboratories, R. *Password-Based Cryptography Specification*. 2000 [cited 2015 25/6/2015]; Available from: <http://tools.ietf.org/html/rfc2898>.

62. Heckel, P. *Generic deduplication process*. 2013 [cited 2015 17/6/2015]; Available from: <http://cdn1.philippheckel.com/wp-content/uploads/2013/05/thesis-figure-3-1.png>.
63. Heckel, P.C. *Minimizing remote storage usage and synchronization time using deduplication and multichunking: Syncany as an example*. 2013 [cited 2015 17/6/2015]; Available from: <http://blog.philippheckel.com/2013/05/20/minimizing-remote-storage-usage-and-synchronization-time-using-deduplication-and-multichunking-syncany-as-an-example/6/-Experiments>.
64. Chang, B., *A Running Time Improvement for Two Thresholds Two Divisors Algorithm*. 2009.
65. Pydio. *Welcome on Pydio/AjaXplorer Community*. [cited 4/10/2014; Available from: <https://pyd.io/>].
66. Pydio. *Administrator Guide | Pydio, formerly AjaXplorer*. [cited 2014 4/10/2014]; Available from: <https://pyd.io/administrator/>.
67. Pydio. *Open source file sharing platform*. [cited 2014 4/10/2014]; Available from: <https://pydio.com/solution>.
68. Pydio. *Pydio Product Tour*. [cited 2014 4/10/2014]; Available from: <http://pt.slideshare.net/axeladida/pydio-product-tour>.
69. OwnCloud. *ownCloud Enterprise Edition*. 2014 [cited 2014 4/10/2014]; Available from: https://owncloud.com/wp-content/uploads/2013/09/ownCloud_Enterprise_Edition+_Subscriptions.pdf.
70. OwnCloud. *OwnCloud Features*. 2014 [cited 2014 4/10/2014]; Available from: <https://owncloud.org/features/>.
71. OwnCloud. *ownCloud Developer Manual*. 2014 [cited 2014 4/10/2014]; Available from: http://doc.owncloud.org/server/7.0/developer_manual/.
72. OwnCloud. *ownCloud User Manual*. 2014 [cited 2014 4/10/2014]; Available from: http://doc.owncloud.org/server/7.0/user_manual/.
73. Xchange, O. *OX App Suite*. [cited 2014 4/10/2014]; Available from: http://oxpedia.org/wiki/index.php?title=AppSuite:Main_Page_AppSuite_information.
74. Xchange, O. *OX App Suite Synchronizing Data with OX Drive*. [cited 2014 4/10/2014]; Available from: <http://software.open-xchange.com/products/appsuite/doc/drive/OX-OXDrive-User-Guide-English-v1.2.0.pdf>.
75. Xchange, O. *OX App Suite User Guide*. [cited 2014 4/10/2014]; Available from: <http://software.open-xchange.com/products/appsuite/doc/OX-App-Suite-User-Guide-English-v7.6.0.pdf>.
76. Heckel, P.C. *Syncany explained: idea, progress, development and future (part 1)*. 2014 [cited 2014 4/10/2014]; Available from: <http://blog.philippheckel.com/2013/10/18/syncany-explained-idea-progress-development-future/>.
77. Heckel, P.C. *Deep into the code of Syncany – command line client, application flow and data model (part 2)*. 2014 [cited 2014 4/10/2014]; Available from: <http://blog.philippheckel.com/2014/02/14/deep-into-the-code-of-syncany-cli-application-flow-and-data-model/>.
78. Stack, O. *OpenStack Object Storage ("Swift")*. [cited 2014 4/10/2014]; Available from: <https://wiki.openstack.org/wiki/Swift>.
79. RiakCS. *RiakCS Technical Overview*. [cited 2014 4/10/2014]; Available from: <http://info.basho.com/RiakCSTechnicalOverview.html>.
80. Share, S. *SparkleShare - Self hosted, instant, secure file sync*. 4/10/2014; Available from: <http://sparkleshare.org/>.
81. SpiderOak. *The SpiderOak User Manual*. 2014 [cited 2014 4/10/2014]; Available from: <https://spideroak.com/user-manual/>.

82. SpiderOak. *SpiderOak Features Overview*. [cited 2014 4/10/2014]; Available from: <https://spideroak.com/features/>.
83. Tresorit. *Hack us!* [cited 2014 4/10/2014]; Available from: <https://tresorit.com/contest/hacking-challenge>.
84. Tresorit. *Access and share content easily and securely - Tresorit Features*. [cited 2014 4/10/2014]; Available from: <https://tresorit.com/features>.
85. Dominik, G., et al., *Cryptree: A Folder Tree Structure for Cryptographic File Systems*, in *Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems* %@ 0-7695-2677-2. 2006, IEEE Computer Society. p. 189-198.
86. Wuala. *Wuala - Features - Secure Cloud Storage - Backup. Sync. Share. Access Everywhere*. [cited 2014 4/10/2014]; Available from: <https://http://www.wuala.com/en/learn/features>.
87. Wuala. *Wuala - Technology - Secure Cloud Storage - Backup. Sync. Share. Access Everywhere*. 4/10/2014]; Available from: <https://http://www.wuala.com/en/learn/technology>.
88. BOX. *Product Features Overview | Box*. [cited 2014 4/10/2014]; Available from: <https://http://www.box.com/business/products-and-features/>.
89. Engyte. *Egnyte Feature List*. [cited 2014 4/10/2014]; Available from: https://http://www.egnyte.com/corp/pdfs/Egnyte_Feature_List.pdf.
90. Wuala. *Wuala - Secure Cloud Storage - Pricing*. 2015 [cited 2015 3/7/2015]; Available from: <https://http://www.wuala.com/en/pricing/>.
91. Tresorit. *Get 1TB of secure cloud storage*. 2015 [cited 2015 3/7/2015]; Available from: https://tresorit.com/pricing?utm_expid=64271970-48.HDmMWMfZTuSzq89CvQtaag.0&utm_referrer=https%3A%2F%2Ftresorit.com%2Fsecure-cloud-storage.
92. SpiderOak. *Complete price list*. 2015 [cited 2015 3/7/2015]; Available from: <https://spideroak.com/about/price-list>.
93. Scrum.org. *What is Scrum?* [cited 2015 25/6/2015]; Available from: <https://http://www.scrum.org/resources/what-is-scrum/>.
94. Kokko, A. *Improving requirements management practices in agile software development environment*. 2013 [cited 2014 4/10/2014]; Available from: http://www.theseus.fi/bitstream/handle/10024/68392/Kokko_Antti.pdf?sequence=1.
95. desenvolvimentoagil.com.br. *SCRUM*. [cited 2015 25/6/2015]; Available from: <http://www.desenvolvimentoagil.com.br/scrum/>.
96. Agile42. *Scrum Roles*. [cited 2015 25/6/2015]; Available from: <http://www.agile42.com/en/agile-info-center/scrum-roles/>.
97. Methodology, S. *The Scrum Team Role*. 2015 [cited 2015 25/6/2015]; Available from: <http://scrummethodology.com/the-scrum-team-role/>.
98. OWASP. *Password Storage Cheat Sheet*. 7/1/2015]; Available from: https://http://www.owasp.org/index.php/Password_Storage_Cheat_Sheet.
99. NIST. *Recommendation for Password-Based Key Derivation*. 2010 [cited 2015 7/1/2015]; Available from: <http://csrc.nist.gov/publications/nistpubs/800-132/nist-sp800-132.pdf>.
100. STOMP. *STOMP Protocol Specification, Version 1.2*. 2013 [cited 2015 25/6/2015]; Available from: <http://stomp.github.io/stomp-specification-1.2.html>.
101. Stoyanchev, R. *WebSocket Apps Spring Framework 4 (part 1)*. 2015 [cited 2015 18/6/2015]; Available from: <http://rstoyanchev.github.io/springx2013-websocket/-1>.
102. S3, A. *Class AmazonS3Client*. 2013 [cited 2015 17/6/2015]; Available from: <http://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/com/amazonaws/services/s3/AmazonS3Client.html>.

103. Rasmussen, M.S. *Pushing the Limits of Amazon S3 Upload Performance*. 2011 [cited 2015 17/6/2015]; Available from: <http://improve.dk/pushing-the-limits-of-amazon-s3-upload-performance/>.
104. MySQL. *Guide To Scalling Web Databases With MySQL Cluster* 2015 [cited 2015 17/6/2015]; Available from: <https://http://www.mysql.com/products/cluster/>.
105. Apache. *Welcome to Apache™ Hadoop®!* 2015 [cited 2015 18/6/2015]; Available from: <https://hadoop.apache.org>.
106. Amazon. *Amazon S3 Pricing*. 2015 [cited 2015 18/6/2015]; Available from: <http://aws.amazon.com/s3/pricing/>.
107. Amazon. *Amazon DynamoDB Pricing*. 2015 [cited 2015 18/6/2015]; Available from: <http://aws.amazon.com/dynamodb/pricing/>.
108. HSQLDB. *HSQLDB - 100% Java Database*. 2015 [cited 2015 3/7/2015]; Available from: <http://hsqldb.org>.
109. Runscope. *Runscope*. 2015 2/6/2015]; Available from: <http://www.runscope.com>.
110. YSlow. *YSlow*. 2015 [cited 2015 2/6/2015]; Available from: <http://yslow.org/>.
111. OwnCloud. *OwnCloud's Data Encryption Model*. 2014 [cited 2015 2/6/2015]; Available from: https://owncloud.com/wp-content/uploads/2014/10/Overview_of_ownCloud_Encryption_Model_1.1.pdf.
112. TechFindings. *Testing OwnCloud Performance*. 2014 2/6/2015]; Available from: <http://zo0ok.com/techfindings/archives/1505>
113. bueker. *External Storage S3 Very High Requests*. 2015 [cited 2015 2/6/2015]; Available from: <https://github.com/owncloud/core/issues/13147>.
114. OwnCloud. *OwnCloud Performance Tweaks*. 2014 [cited 2015 2/6/2015]; Available from: <https://forum.owncloud.org/viewtopic.php?t=10692&p=80618>