View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Repositório Institucional da Universidade de Aveiro

Rui André
Cruz Lebre

Mecanismo de Gestão de Áreas de Utilizador para Repositórios de Imagem Médica

Accounting Mechanism for Shared Medical Imaging Repositories


Universidade de Aveiro
2017

Departamento de Eletrónica,
Telecomunicações e Informática

Rui André
Cruz Lebre

Mecanismo de Gestão de Áreas de Utilizador para Repositórios de Imagem Médica

Accounting Mechanism for Shared Medical Imaging Repositories

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Carlos Manuel Azevedo Costa, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Prof. Doutor Joaquim Manuel Henriques de Sousa Pinto

Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor José Paulo Lousado

Professor Adjunto, Dep. de Informática, Comunicações e Ciências Fundamentais, da Escola Superior de Tecnologia e Gestão de Lamego do Instituto Politécnico de Viseu

Prof. Doutor Carlos Manuel Azevedo Costa

Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro (orientador)

agradecimentos /
acknowledgements

Gostava de agradecer, em primeiro lugar ao meu orientador, Carlos Costa, pela oportunidade, orientação e apoio durante o mestrado. Os seus conselhos, recomendações e discussões foram fundamentais para a realização deste trabalho. Um agradecimento especial também ao Luís Bastião pelo apoio fundamental quase diário durante a realização da dissertação. Quero também deixar uma palavra de apreço a todos os elementos do grupo de Bioinformática por proporcionarem um ambiente de entreajuda e divertimento dentro e fora espaço aberto dedicado ao grupo.

Quero também agradecer aos meus amigos Helder, Joana e Xavier, não esquecendo a Mélanie, o João e o Jerónimo pelo apoio e amizade, passando também pelo André, a Raquel, o Tiago, a Rita e a Soraia.

Finalmente, um agradecimento especial aos meus pais, Mário Lebre e Julieta Lebre, e ao meu irmão, Sérgio Lebre por todo o suporte.

Palavras Chave

Informática Médica, PACS, Imagem médica, Armazenamento Cloud, Computação Cloud, Repositórios Partilhados, DICOM.

Resumo

A imagem médica em formato digital é um elemento presente nas mais variadas instituições prestadoras de cuidados de saúde, afirmando-se como um imprescindível elemento de suporte ao diagnóstico e terapêutica médica. Nesta área, os formatos e processos de armazenamento e transmissão são definidos pela norma internacional DICOM. Um ficheiro deste tipo contempla, para além da imagem (ou vídeo), um conjunto de meta-dados que incluem informação dos pacientes, dados técnicos relativos ao estudo, dose de radiação, relatório clínico, etc.

Um dos maiores problemas associados aos repositórios de imagem médica está relacionado com a grande quantidade de dados produzidos que impõe desafios acrescidos ao armazenamento e transporte da informação, em particular em cenários distribuídos e de grande produção de estudos imagiológicos.

Esta dissertação tem como objetivo estudar e explorar soluções que permitam a integração do conceito de pertença e controlo de acesso em arquivos de imagem médica, possibilitando a centralização de múltiplas instâncias de arquivos. A solução desenvolvida permite associar permissões a recursos e delegação a terceiras entidades. Foi desenvolvida uma interface programática de gestão da solução proposta, disponibilizada através de web services, com a capacidade de criação, leitura, atualização e remoção de todos os componentes resultantes da arquitetura.

Keywords

Medical Informatics, Medical Imaging, PACS, Cloud Storage, Cloud Computing, Accounting, DICOM, Shared Repositories.

Abstract

The production of medical images in digital format has been growing in the most varied health care providers, representing at this moment an important and indispensable element for supporting medical decisions. In medical imaging area, the formats and transmission processes are defined by the international DICOM standard. A file in this format contains image pixel data but also a set of metadata, including information about the patient, technical data related to the study, dose of radiation, clinical report, etc.

One of the biggest problems associated with medical imaging repositories is related to the large amount of data produced that poses additional challenges to the transport and archive of information, particularly in distributed environments and laboratories with huge volume of examinations. This dissertation aims to study and explore solutions for the integration of ownership concept and access control over medical imaging resources, making possible the centralization of multiple instances of repositories. The proposed solution allows the association of permissions to repository resources and delegation of rights to third entities. It was developed a programmatic interface for management of proposed services, made available through web services, with the ability to create, read, update and remove all components resulting from the architecture.

Contents

List of Figures

List of Tables

Acronyms

CT Computed Tomography

CR Computed Radiography

MR Magnetic Resonance

US Ultrasounds

XA X-Ray Angiography

ECG Electrocardiogram

IT Information Technology

PACS Picture Archive and Communication System

DICOM Digital Imaging and Communications in Medicine

UID Unique Identifier

SOP Service-Object Pair

LAN Local Area Network

WAN Wide Area Network

HIS Hospital Information System

RIS Radiology Information System

NEMA National Eletrical Manufactures Association

ACR American College of Radiology

HIPAA Health Insurance Portability and Accountability Act

DIM DICOM Information Model

SOP Service-Object Pair

TLV Tag-Length-Value

VR Value Representation

AETitle Application Entity Title

TCP Transmission Control Protocol

IP Internet Protocol

SCU Service Class User

SCP Service Class Provider

WADO Web Access to DICOM persistent Objects

STOW-RS STore Over the Web

QIDO-RS Query based on ID for DICOM Objects

HTTP HyperText Transfer Protocol

HTTPS HyperText Transfer Protocol Secure

VPN Virtual Private Network

ePR electronic Patient Record

SDK Software Development Kit

URI Uniform Resource Identifier

URL Uniform Resource Locator

API Application Programming Interface

AWS Amazon Web Services

REST Representational State Transfer

RESTful Representational State Transfer (web services implementing REST)

JSON JavaScript Object Notation

SOAP Simple Object Access Protocol

SSL Secure Sockets Layer

DES Data Encryption Standard

AES Advanced Encryption Standard

JPA Java Persistence API

JDBC Java Database Connectivity

DBMS Database Management System

CRUD create, read, update and delete

XML Extensible Markup Language

IdP Identity Provider

CHAPTER 1

INTRODUCTION

This chapter provides an introduction to the thesis, such as the concepts of medical imaging and gives also a glimpse on Dicoogle platform. Finally, the main goals of this thesis will be presented and immediately after, all outline described.

1.1 OVERVIEW

Digital Medical Imaging has seen its presence strengthened in healthcare institutions. It provides a great support to medical staff in terms of diagnosis and further decisions. Because of that, healthcare industry has been following the general evolutionary tendencies in IT technologies. These institutions have been increasingly providing new services to improve patients well-being, like telemedicine or electronic Patient Record (ePR).

At institutional management level, healthcare systems had been improved. Hospital Information System (HIS) and Radiology Information System (RIS) are examples of the use of information systems in medical environment.

DICOM is a standard that defines how communications, data format and storage should be accomplished in the digital medical imaging field. This medical data is agglutinated on one or multiple files called DICOM object(s), which contains, besides images, metadata related to reports, study or even patient and healthcare institution information.

PACS stands for Picture Archiving and Communication System. It is a system composed by one or more archives. Besides this archiving task, Picture Archive and Communication System (PACS) is associated with another one: the distribution of images, since some faculty do not practice in the same department where images are acquired and even archived. This system supports the DICOM files archiving.

Medical image acquisition produces a huge amount of data [1] and its storage and distribution are associated with significant financial charges. Over time, the amount of data will tend to increase exponentially and even small institutions can produce a large amount gigabytes of data. This issue is a key concept in PACS with direct impact in archive, distribution and workflow performance.

PACS outsourcing is a good solution and current trend because of the lowest institutional budget consumption. However, maintaining PACS over cloud may increase the communication latency to retrieve medical studies.

The environment of PACS use is the clinical. This institutional clinical environment is divided into departments with different areas of activity. Therefore, the studies are also of different modalities. However, in addition to a departmental organization, there is also a need for division of infrastructures, i.e. multiple files belonging to the same organization.

Over the last years, an open source PACS system has been continuously developed at UA.PT Bioinformatics Group. It is a research group from Institute of Electronics and Informatics Engineering of Aveiro, whose headquarters are in the University of Aveiro and propose the project "Dicoogle P2P Network". Dicoogle is a PACS archive supported by a document based indexing system and distributed engines that can be easily installed on a server or workstation capable of storing medical DICOM images/files. Dicoogle is a platform able to extend by plugins since its documented architecture and SDKs are provided.

However, the Dicoogle platform does not support the multi-archive and multi-user paradigm. Since Dicoogle is a system that allows rapid development making use of the possibility of extension via plugins and the available SDK, it was the choice for the development support of this thesis.

1.2 GOALS

The actual conjuncture of cloud computing providers is a good opportunity to reduce the costs of acquisition and maintenance of hardware and software. I.e., the outsourcing of IT medical storage system allows medical facilities to reduce the costs of investing in infrastructure, trained personnel and licensed software. Besides that, cloud computing provides mechanisms to increase computing power and storage, so healthcare institutions only pay the capacity needed to production.

However, the current PACS Dicoogle solution does not support the multi-archive paradigm. The purpose of this thesis is to study the state of the art solutions for accounting management, aiming to support multiple users with different access permissions. This will open doors to a new paradigm of shared medical imaging repositories.

At the end of this work, it is expected to have a unified, scalable and reliable information system. The information system must implement an access control mechanism that can be integrated with the Dicoogle platform. Parallel to the programmatic library developed to support this platform, an abstraction layer must also be developed in the form of Representational State Transfer (REST) services. This REST layer will allow the development of other applications that could, for example, implement a storage system in the existing cloud services, using for this the management of users and sharing of resources.

1.3 OUTLINES

This thesis is divided in 5 chapters. A brief description will be following presented.

- ***Chapter 2:*** provides a description of the scenario where this thesis is inserted and also a description of the state of the art, including a description of digital medical laboratories, DICOM standard and an overview over PACS and its existing services. Finally, there is a brief description of technologies related to this thesis.
- ***Chapter 3:*** gives the description of the requirements expected to be fulfilled at the end of the development of this thesis. The section presents the functional and non-functional requirements are presented.
- ***Chapter 4:*** in this section it is presented and described the thesis practical development: Architecture and Implementation. It is shown the design and all the forward implementation details, as well as the data model developed and services available.
- ***Chapter 5:*** of this chapter, nominated Results and Discussion, is presented the validation of the system. Tests are shown and discussed, evaluating the impact of the solution in a matter of scalability and time to be performed. There is also presented guidelines to use services.
- ***Chapter 6:*** finally, on the last chapter, it is presented conclusions and future work that can be developed as a result of the work of this thesis.

CHAPTER 2

State of the Art

In this chapter is presented a detailed analysis of the state of the art in systems and technologies related to medical imaging such as PACS and DICOM. The reading of this chapter is fundamental to understand the background and environment in which this document is inserted.

2.1 OVERVIEW

In the last decades, the healthcare institutions have been adopting technologies and information systems to diagnose and treat patient diseases [2]. Medical imaging is one of these technologies and it is defined as the production of the visual representation of the Human body for use of clinical diagnosis [3].

Before the digital era, medical images were acquired on analogue equipment and printed in films. Therefore, archive and distribution entailed some constraints, like the maintenance cost and examination retrieval time. This last one issue, access time, had been particularly important due the real time access to studies by healthcare physicians

Nowadays, due the proliferation of Information Technology (IT) associated to medical equipment, the number of imaging-based procedures is increasing, resulting in a speed-up of the workflows and redution of costs to healthcare institutions.

Among medical imaging areas, subareas such Radiology and Nuclear Medicine are very popular. The most common modalities are Computed Tomography (CT), Computed Radiography (CR), Magnetic Resonance (MR) and Ultrasounds (US).

Small Sized healthcare institutions are benefiting of the increasing availability of medical equipment, as referenced above, at lower costs. They are acquiring more devices, with higher resolutions. The amount of images produced by these acquisition devices has opened the door to PACS distributed environments and cloud services outsourcing, allowing it to scale up to supporting a growing amount of medical data and metadata.

According to Frost & Sullivan Principal Analyst Nadim Daher [1], "even if diagnostic imaging volumes continue to plateau around the 600 million procedures per year mark, overall storage and

archiving volume requirements for U.S. medical imaging data will cross the 1-exabyte mark by 2016". For reference, 1 exabyte is 1,000 petabytes or 1,000,000 terabytes. This report, authors say that 1-exabyte mark defines the "medical imaging's (...) entry into Big Data territory".

2.2 PICTURE ARCHIVE AND COMMUNICATION SYSTEM

Over the last years, health care institutions have made great investments in IT infrastructure to maintain medical imaging laboratories. The amount of data generated in these laboratories is huge and for that reason, a key issue.

The volume of data generated is very high in modalities like, for instance, in X-Ray Angiography (XA), US, multi-slice CT and specially digital mammography [4]. So, to keep these data, it is primordial to create robust and efficient storage and communication infrastructures to ensure full availability [1][5], even without requiring major upgrades and overhauls that increase the cost over time [1].

PACS are medical systems composed by a set of hardware and software that processes, stores, distributes and provides medical images or a portion of them to, or from, a health care institution [4][6]. They comprise modalities, digital image acquisition devices; digital image archives, storage to acquired images; and workstations, devices to view those images [6].

Those components communicate typically , through network (Local Area Network (LAN) or subnet in Wide Area Network (WAN)) [4] . The usage of PACS reduces the retrieve time of exams and the need to use film jackets [4], but also the probability of losing studies.

PACS workflow has the following major steps (Figure 2.1: acquisition, distribution and displaying [2][7]:

1. **Acquisition:** the process of acquiring or capturing the image in a digital codification, creating a representation similar to reality. Those images can be acquired by two methods, they are: scanning directly from digital equipment through examination procedures, or scanning from analogical films, produced by early equipment, to keep compatibility between archives.

2. **Distribution:** mentioned in [2] as the process of moving images and metadata from PACS to another node outside the sector. Most of medical imaging studies need to be moved from acquisition (performed by modalities) to network workstations that will perform visualization. Over PACS, the loss of studies in this process can be avoided regarding the traditional film system. Additionally, PACS can also allow sharing studies among institutions in contrast to sharing studies only inside the institution.

3. **Visualization:** is the process of viewing the medical images. Commonly, it is achieved using workstations. These workstations include, among others, display and processing software, allowing the user to search, retrieve, visualise, manipulate and share medical images [8].

By stating these processes, we can infer that PACS turns the workflow easier in medical imaging environments, allowing clinics, physicians and technicians to access the data quickly.

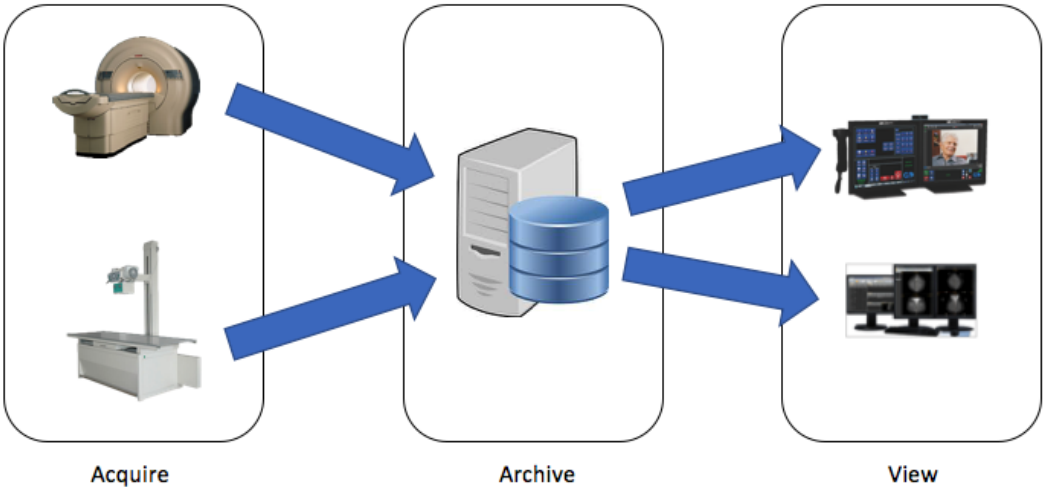

Figure 2.1: Major PACS Components. On the left, image acquisition devices (modalities). Those modalities store acquired images on a digital archive (center). From there images are accessed by radiologists at the viewing workstations (right). Adapted from [6]

Over time, medical imaging laboratories trend to generate a large amount of data [1] that is an issue that PACS Archives have to deal with. PACS needs to store and retrieve this big volume of data and, at the same time, keep the communication delays acceptable to medical diagnosis. These constraints lead to a problem in small medical institutions due to the economic limitations for creating and maintaining the IT infrastructure. As a result, and joining the fact that PACS must have security and reliability (such backup, redundancy and crash reports), PACS outsourcing is rising in the last years.

PACS workflows comprise the examination procedure, image viewing, reporting and image archiving, but also the patient registration on HIS and RIS. In *PACS and Imaging Informatics: Basic Principles and Aplications* [7], the author proposes the general architectures that fit in different kind of workflows. Those are the stand-alone, client-server and web-based model.

- The stand-alone architecture approach has a central repository where the images acquired from modalities, after the procedure, are immediately sent to it and then forwarded to previously registered workstations. This workflow involves a store and forward approach. Despite of the architecture mentioned has several benefits (because of the allowance to perform Query/Retrieve operations and the modality can send images directly to workstations), there are some problems related to study loss and studies that are being reviewed by more than one radiologist. This happens because studies are transmitted without asking to a workstation.
- On client-server approach, studies are stored in a central repository from acquisition equipment. Then, technicians use the worklist to retrieve the expected exams only when needed. After image analysis, since workstations don't have local storage, they are discarded. Workstations retrieve studies without any pre-fetching strategies or Query/Retrieve. This can be a constraint since PACS is single-point-of-failure and bandwidth has to be fast to not introduce a big delay. Nonetheless, this architecture introduces a more efficient control than stand-alone architecture.

- Web-based architecture follows the current trend in PACS architectures. To the review process, workstations just need Internet access and a web browser/client. This architecture provides a front-end for operating with images stored on a remote storage. It is the most efficient architecture in terms of bandwidth, portability and reliability. Although, there are some limitations related to performance due to the computation power of client and the possible bottleneck when many clients are accessing at the same time [9].

Nonetheless, there can arise several security issues related to the deploying of this architecture on public cloud environments. So, patient privacy and security of sensitive patient data must be ensured.

2.3 DIGITAL IMAGE COMMUNICATIONS IN MEDICINE

DICOM stands for Digital Imaging and COmmunications in Medicine [6]. It is a standard created by a consortium founded in the 80s formed by National Eletrical Manufactures Association (NEMA) and American College of Radiology (ACR). DICOM Specifies a non-proprietary medical data interchanging protocol, data format and file structure for medical images and its associated metadata [10].

With the uprising of digital medical imaging, manufacturers of medical imaging equipment started to develop equipment able to acquire, store and transfer data across medical devices. The vendors had developed their own protocols so, communication between devices from different manufacturers started to be a real challenge.

To solve this problem, the consortium mentioned above, in the mid-80s, released the ACR/NEMA-300. At the end of the same decade, it was released the second version of the protocol, rectifying several non-clear and contradicting the original text. In the 90s, the consortium created the third version of the protocol with the name changed to DICOM. Officially, this is the latest version.

This last version named at the time DICOM 3.0, has been constantly updated and extended since 1993, its release year. DICOM has a continuous process of development and is constantly updated and extended to face the most recent issues in the medical imaging field. Although, most of the changes are forward and backwards compatible. Presently, DICOM is the most important standard in medical imaging and used by almost PACS [8] [11].

Over the years, the proliferation of DICOM compliant equipment enabled the exchange of data between medical imaging devices and triggered the implementation of PACS. Nowadays, DICOM is a recognized standard around the world [6].

2.3.1 DICOM INFORMATION MODEL

DICOM Standard seeks to represent the real-world. The DIM defines the structure and organisation of information [12] representing items that match real life objects and describing their relationship. DICOM hierarchy is Patient-Study-Series-Image. This represents real world organization on medical facilities. A patient can have multiple studies. Each study can have multiple series from different modalities and each modality can produce a different number of images. Figure 2.2 represents that hierarchy.

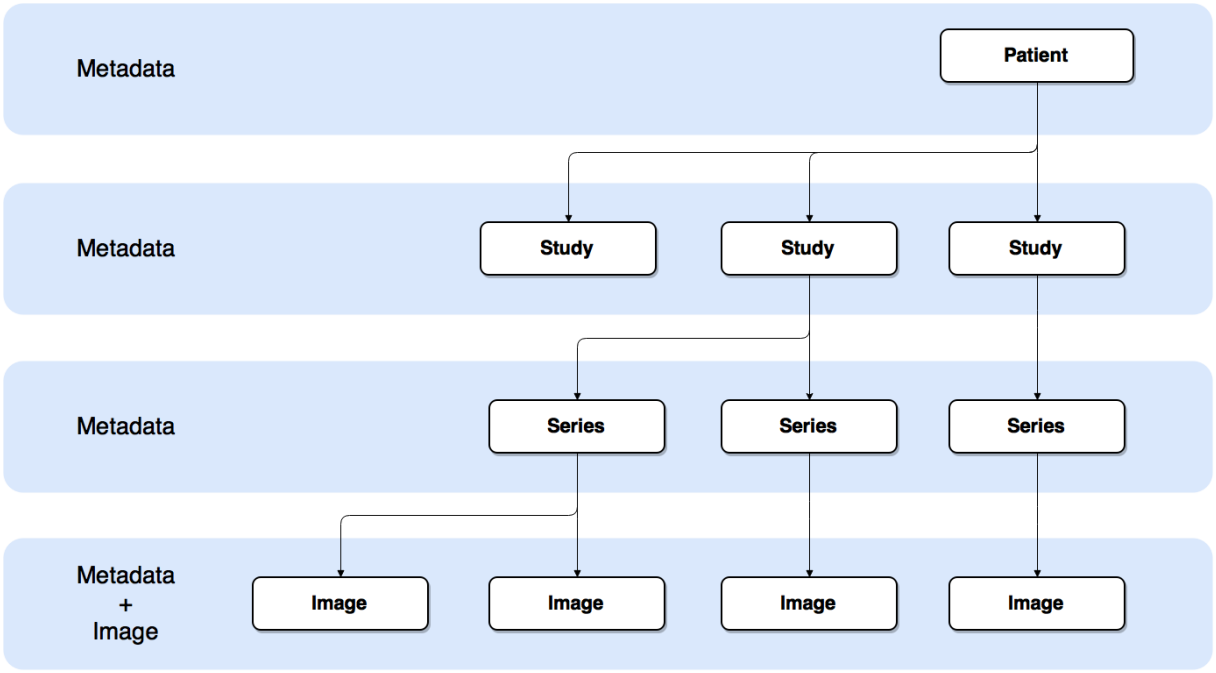

Figure 2.2: DICOM Information Hierarchy. A patient can have multiple studies. Each study can include various series. Each serie has one or more images.

Each hierarchy level has associated a unique ID. In the first level, there is Patient ID, followed by Study Instance UID to study level. In the series level, there is the identifier Series Instance UID and in the last level, image level, the identification is made with a Service-Object Pair (SOP) Instance UID.

2.3.2 DICOM DATA FORMAT

DICOM standard has support for many kinds of information from different modalities, reports and waveforms (such as CT, XA, CR or Electrocardiogram (ECG)). Each DICOM file has metadata headers related to the image (pixel data), such information is related to the patient, modality, institution, radiation dosage and many other. The DICOM metadata header fields may vary according to the modality of study. However, there are certain required fields. Those fields (data elements) are defined in the previously introduced DICOM DIM [12] (Figure 2.3).

Data elements that compose a DICOM file follows a Tag-Length-Value (TLV) structure as shown in figure 2.4. *Tag* is a pair of two values representative of group and element. It is represented by a 16-bit unsigned integer in hexadecimal. Exemplifying, Patient Name is identified by the group 0x10 and element 0x10. So, Patient Name is (0010, 0010). The second field, *Length*, represents, in bytes, the length of the value field. Finally, the *Value* field contains the binary data of *Tag*, for instance, the patient name, institution name or even pixel data.

An optional field is Value Representation (VR) that specifies how values are encoded. There are 27 ways to do that encoding. For example, PN for Patient Name or UI for Unique Identifier (UID). This label is optional because the type can be reached using DICOM dictionary. In other words, the

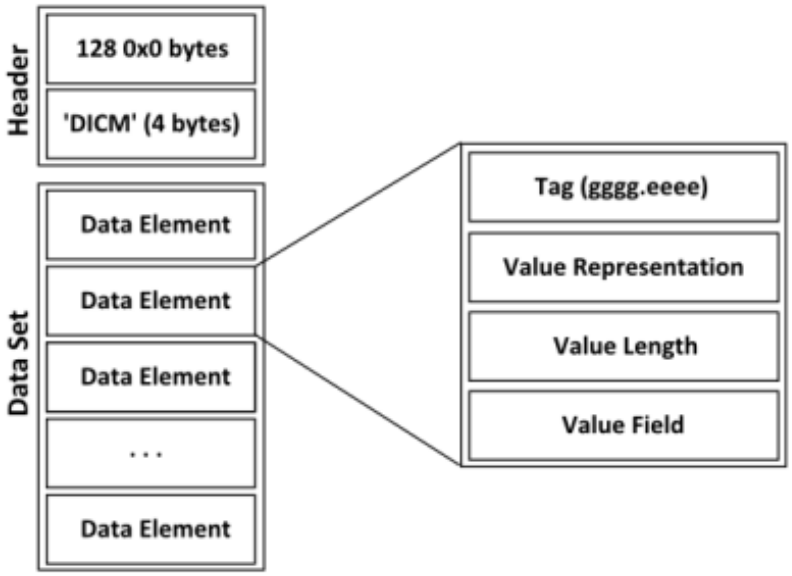

Figure 2.3: DICOM file format. Acquired from [13]

dictionary defines for what the *Tag* stands for.

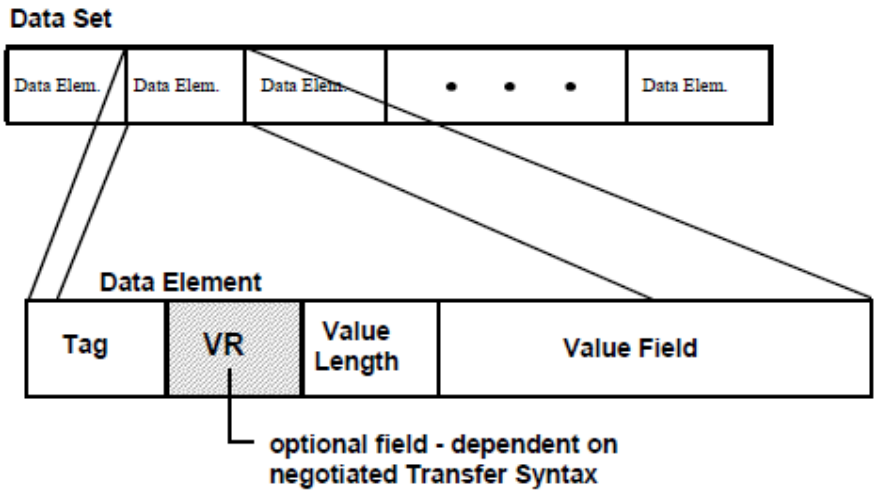

Figure 2.4: DICOM Data Element structure. Adapted from [13]

Because of the TLV structure, new tags can be defined in DICOM objects, increasing flexibility in communication and functionality. DICOM parsers can be generic due to this dynamic structure, decoding any kind of DICOM file.

2.3.3 DICOM services

DICOM defines a set of fundamental services for interoperability between application nodes. This protocol is based on Transmission Control Protocol (TCP)/Internet Protocol (IP) and has the same

structure as DICOM objects. In other words, messages are also encapsulated as TLV elements.

Each service can be used between an SCU and an SCP following a client/server architecture. SCP stands for Service Class Provider and represents a DICOM node providing services. PACS Archive is an example of an SCP. SCU stands for Service Class User and is the node that consumes the DICOM service. Workstations are examples of Service Class User (SCU) when using PACS Archive Server services to retrieve DICOM images. Shortly, Service Class Provider (SCP) offers a service and SCU consumes that service.

A DICOM Service is identified by an Application Entity Title (AETitle). Therefore, applications are identified by IP, Port and AETitle which allows to run different DICOM Services in the same host device.

The standard defines also the procedure to communicate with the DICOM device. The first step is the proposal of an exchange process. That process is called DICOM Association. In this procedure, devices negotiate some parameters to exchange that will take place, like image compression type, data encoding, or even what kind of information will follow. After this negotiation, the service negotiated may take place [14].

Each DICOM service has several DICOM commands associated. The most important ones are Verification (C-Echo), Storage (C-Store), Query/Retrieve (C-Find, C-Move/C-Get) and Worklist Management (C-Find). Verification service allows a user/device to check end-to-end communications, i.e. to verify connectivity between SCP and SCU.

STORAGE SERVICE

Usually, storage service is provided by PACS Archives. The service is called when a node needs to store an image in a repository. In practice, when SCU (modality or the image generator) sends files to PACS Archive, it uses a C-Store request message for each image. The workflow, graphically represented in figure 2.5, is as follows:

1. SCU sends C-Store request to SCP with the image metadata and pixel data
2. SCP replies with a C-Store response acknowledging the receiving status

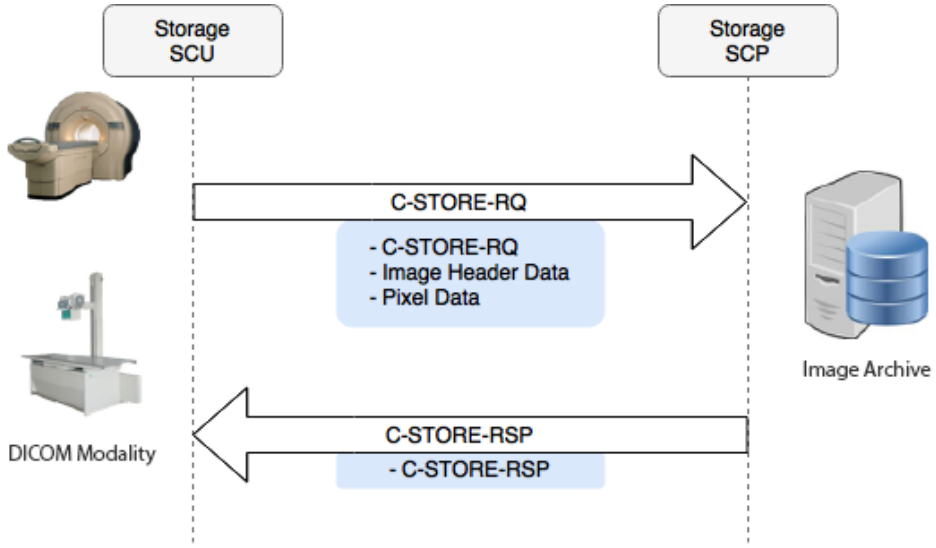

Figure 2.5: DICOM Storage Service

QUERY/RETRIEVE SERVICE

Query/Retrieve is composed by two commands. This service is often used to search and download studies from PACS Archive, usually to be reviewed by a technician in visualisation workstations.

Query Service allows SCUs to search for objects with parameters like name, date, modality and so on, using C-Find command. One or more C-Find-response will be sent to the SCU, one per each matched object (Figure 2.6).

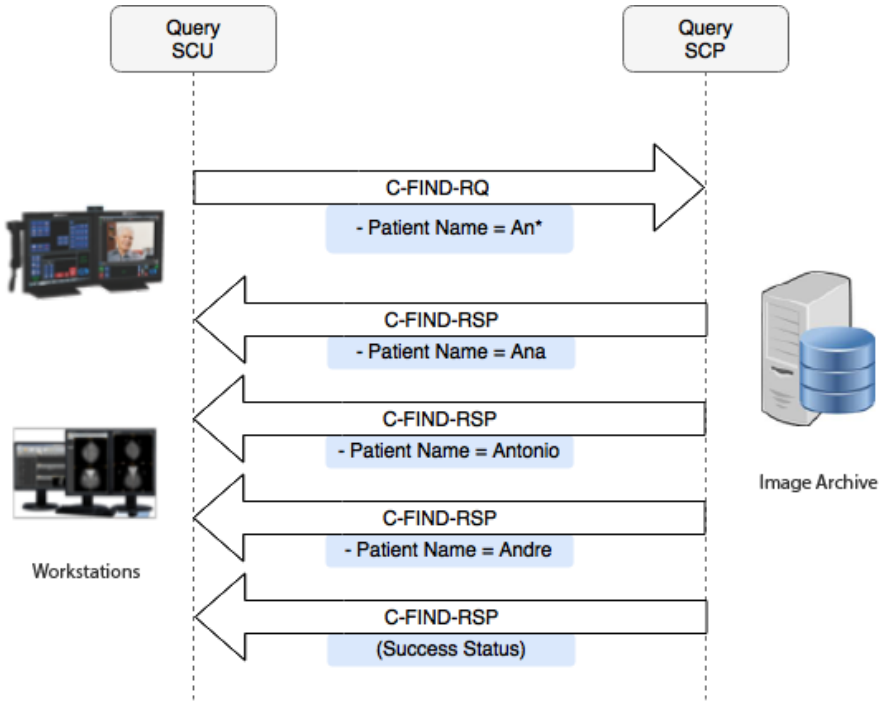

Figure 2.6: DICOM Query Service

This is the first of the two phases. After the response, it begins the second step, the retrieval phase. Retrieve allows the SCU to get or move files from a SCP. It uses the command C-Move or C-Get. Retrieve procedure starts with a C-Move-request message to SCP, identifying the desired objects previously searched.

C-Move does not download images instantly by itself. Instead of that, the SCP invokes a C-Store command for each requested object from SCU. The C-Store is a request meaning that the image archive is ready to transfer the study to a specific workstation/location. After the last object is transferred, a C-Move-response is sent to acknowledge the conclusion of the process (Figure 2.7).

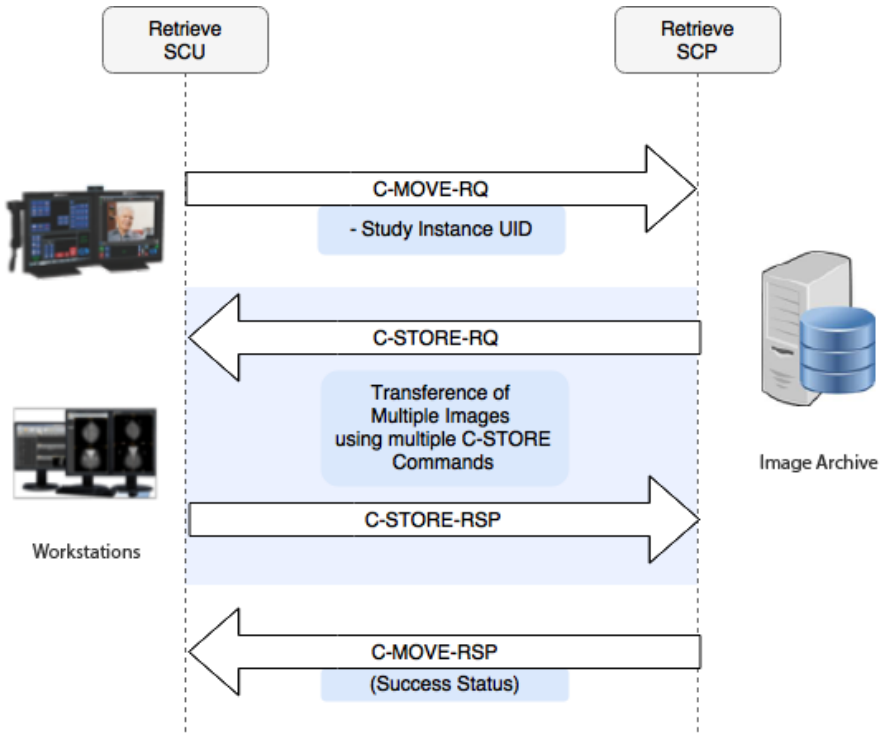

Figure 2.7: DICOM Retrieve Service

Therefore, SCU that will receive the images needs to support a Storage Service in order to receive those objects.

It is important to note that a C-Store command usually is performed for each requested study, as it might not be the most effective way to transfer a large number of studies [8]

2.3.4 DICOMWEB

Initially, DICOM Services were only supported over TCP/IP protocol [15]. Although, with the rising of the popularity of Web Service technologies, there was the need to create a web-version of the most common services: WADO-RS for searching using key parameters, QIDO-RS for retrieving objects and STOW-RS for web storage [16][17].

DICOMweb is the web standard for medical imaging. It defines a set of services that compose

the family of Representational State Transfer (web services implementing REST) (RESTful) DICOM services responsible for storing, retrieving and querying for medical data and metadata.

The intent of this standard is to provide a REST interface for developers to implement mechanisms for accessing healthcare images since this standardized methods provide access to underlying imaging systems that speak DICOM.

Usually, network administrators tend to implement measures related to security and policies like firewalls. DICOM Protocol is often not recognised [8] by them so, the content is blocked. However, using DICOMweb services, that content is recognized as HyperText Transfer Protocol (HTTP) and therefore, allowed.

WADO

WADO is a recent web-based extension to DICOM Protocol [18]. It allows the access to DICOM objects like, for instance, images and reports. WADO extension allows DICOM persistent objects to be encapsulated in a HTTP/HyperText Transfer Protocol Secure (HTTPS) connection and so can be integrated on a web application.

WADO uses HTTP or HTTPS, and this protocol is normally accepted by network firewalls. WADO is a retrieval service similar to C-Move and C-Get. However, it does not provide discovery analogous to C-Find over PACS Archive. So, to download DICOM Objects, users need to know the resource identifier.

WADO-RS

WADO-RS is a RESTful version of WADO that enables the retrieving of specific studies, series, instances or frames by reference through HTTP messaging. DICOM standard defines these action types and its implementation should support the following types [19] represented in table 2.1.

Action	Description	Example
RetrieveStudy	Retrieves a set of DICOM instances with the requested Study Unique Identifier (UID). The response can be DICOM or bulk data.	GET /studies/StUID
RetrieveSeries	Retrieves a set of DICOM instances with the requested Study and Series UID. The response can be DICOM or bulk data.	GET /studies/StUID/series/SeUID
RetrieveInstance	Retrieves a set of DICOM instances with the requested Study, Series and SOP Instance UID. The response can be DICOM or bulk data.	GET /studies/StUID/series/SeUID/instances/InUID
RetrieveFrames	Retrieves a set of DICOM instances with the requested Study, Series, SOP Instance UID. The response is pixel data.	GET /studies/StUID/series/SeUID/instances/InUID/frames/FrameList
RetrieveBulkdata	Retrieves the bulk data for a requested BulkDataURI.	GET BulkDataURL
RetrieveMetadata	Retrieves the DICOM instances presented as the study, series, or instance metadata with the bulk data removed.	GET /studies/StUID/metadata

Table 2.1: WADO-RS action types. Adapted from [19]

The requests examples in table 2.1 may contain some query parameters like the following ones:

- **accept** ...?"accept=" 1#media-type
- **charset** ...?"charset=" 1#charset

Besides the query parameters, some Headers may compose the GET RESTful request. The request may contain various representation schemes requests, separated by a comma, in preference order:

- **multipart/related; type="application/dicom" [dcm-parameters]** Specifies that the response can be DICOM instances. If transfer-syntax is not specified in "dcm-parameters", the response will be in Explicit VR Little Endian Transfer Syntax for each instance.
- **multipart/related; type="application/octet-stream" [dcm-parameters]** Specifies that the response can be Little Endian
- **multipart/related; type="media-type" [dcm-parameters]** Specifies that the response can be compressed pixel data encoded

Note: Schemes above were adapted from [19].

After the request, the server should provide the desired document(s). If the document cannot be returned, an error code will be sent. Supposing that the server cannot convert the requested data

to any of the requested media types/Transfer Syntaxes, the error code should be a "Not Acceptable" response if no data is returned, or a "Partial Content" response if only some data is returned [19].

STOW-RS

STOW-RS allows to store specific DICOM instances to the server. DICOM standard defines these action types and its implementation should support the following action [20] as shown in Table 2.2.

Action	**Description**	**Example**
Store Instances	Creates new resources for the given SOP Instances or appends to existing resources on the Server.	POST /studies/StUID or POST /studies/

Table 2.2: STOW-RS action type. Adapted from [20]

QIDO-RS

QIDO-RS enables the searching for studies, series and instances by patient ID, and receive their unique identifiers for further usage (i.e., to retrieve their rendered representations). DICOM standard defines this action types and its implementation should support the following types[21].

Action	**Description**	**Example**
SearchForStudies	Searches for DICOM Studies that match the query parameters and returns a list of matching studies as well as the desired attributes for each one of it	GET /studies?StudyInstanceUID
SearchForSeries	Searches for DICOM Series that match the query parameters and returns a list of matching series as well as the desired attributes for each one of it	GET /series?SeriesInstanceUID
SearchForInstances	earches for DICOM Instances that match the query parameters and returns a list of matching instances as well as the desired attributes for each one of it	GET /instances?...

Table 2.3: QIDO-RS action types. Adapted from [21]

Alongside the query parameters exemplified on Table 2.3, some Headers should compose the GET RESTful request, indicating to the server the reponse formats preferred:

- **multipart/related; type="application/dicom+xml** Specifies that the results should be DICOM PS3.19 XML
- **application/dicom+json** Specifies that the results should be DICOM JSON
- **Cache-control: no-cache** If this field is included, the search results should be the current ones and not the cached ones

Note: Schemes above were adapted from [21].

The origin server would after perform the query indicated in the request. The response is determined as follows:

- If there were no matches, a 204 (No Content) response shall be returned with an empty payload.
- If there were matches, a 200 (OK) response shall be returned with a payload containing results.
- If there were results remaining since last response, the response shall include a Warning header field like: "Warning: 299 +service: There are <remaining> additional results that can be requested"

2.4 SECURITY

2.4.1 PRIVACY AND CONFIDENTIALITY

Medical imaging and patient confidentiality is an important social and medical legal issue when data is transmitted across public networks and stored on cloud [22]. Therefore, medical data and metadata are considered valuable to many parties like hospitals, researchers, insurance companies, etc [23]. Because of its value, medical data need to be secured. However, healthcare institutions cannot sustain the responsibility of maintaining a secure storage. Usually, outsourcing is a solution to this problem. Outsourcing is described as the contracting of various information system functions such hardware support, software maintenance, network or managing of data centre among others [24].

In this document, it is described more strictly as exporting medical records to third party companies whose core business is to provide computing power and storage to costumers. This makes telemedicine and remote diagnosis possible or, at least, easier. However, medical institutions need to guarantee the privacy which means that third party companies cannot be trusted since someone can have unauthorised access to data.

There are different aspects to take care like in storage field, where medical images should only be accessed by authorised personnel, maintaining privacy. When medical images are being transmitted, the transferred data should be confidential [25], avoiding attacks like, for instance, *man in the middle* attack. When those aspects are guaranteed, there is an issue related to processing: query requests and results should be ciphered.

The outsourcing to public providers is dependent on country laws [2]. There are some bureaucracy and political issues to take care of. In the USA, there were already mandates for ensuring medical data security issued by the federal government. For instance, Health Insurance Portability and Accountability Act (HIPAA), where health care institutions are obligated to ensure that patient data is only provided to authorised personnel [22].

Permanence of patient data is another topic that must be considered [2]. Data protection laws, in some countries, require knowing where data is stored. In some countries, like Spain or France, storage in the cloud may be difficult due to health sensitive data storage on third party services law.

In these cases, encryption is the most useful approach to ensure data security [22]. This technique can be used not only in storage but also in transmission through public communications. Nowadays, in cases of telemedicine, Virtual Private Network (VPN) are being used for confidentiality purposes.

2.4.2 AUTHENTICATION MECHANISMS

A study was made of some of the authentication mechanisms that currently exist. This study aims to understand how web authentication is processed as well as the different types of approximation.

It was decided to address standards such as OAuth, SAML and OpenID in this document. This approach allows better information about the future choice in the authentication system to be developed.

OAUTH

OAuth was created to solve the problem of secure authorization when OpenID system was adopted [26]. OpenID allows authentication in the sense that a user can have login into a website by being verified by another service (OpenID trusted provider), in other words, it is an authorization model based on an IETF Standard [27]. However, the process forces the user to provide account credentials to an application, so that application can be able to access resources from content provider [28]. For instance, in the case of Google Drive API not requiring OAuth, the only way to a user provide delegated access was to give that application his or her credentials, and only after it the application would communicate with Google's proprietary ClientLogin protocol. This mechanism leads to unnecessary security vulnerabilities [29]:

- Third-party applications are required to store user's credentials for future use if needed, and typically the password store is in clear-text
- Servers are required to support password authentication regarding the security problems inherited from passwords
- Third-party access gains full access to owner's protected resources, leaving the resource owners without the ability to restrict content access
- Resource owners cannot revoke access to third-party individual applications without revoking the access to all the applications using the credentials. Even to do that, user must change the third party's credentials
- The compromising of any application that has access to user's credentials results in the compromising of those credentials and all the content associated with it

Based on these considerations, OAuth authentication protocol was born [26]. First initialized by Blaine Cook and Chris Messina and then delegated to Internet Engineering Task Force (IETF), OAuth allows a secure API authorization in a simple and standardised method [30].

The protocol allows a user to grant access to third-party client applications to access protected resources on its behalf. That can be done without requiring user's credentials [31]. To achieve authorization, the user must log in on OAuth service which on it hand provides an access token containing information about permission grants like specific scope, lifetime and other attributes. The third party then uses that token to access the protected resources [26][29]. In conclusion, instead of using resource owner's credentials, the client simply obtains and uses the access token to access protected resources hosted.

Coming back to the previous example, in the case of Google Drive API had support to OAuth, an end-user could grant a third-party application access to his/her personal cloud storage, for instance, only a couple of hours, without sharing his/her credentials. The user simply authenticates on the trusted server, which on it hand issues the access token that would be used by the third-party application to access shared data.

OAuth 2.0 RFC [29] defines certain roles on a typical framework use as shown in Table 2.4:

Role	**Description**
Resource Owner	Entity responsible for grant or deny access to a protected resource.
Resource Server	The resource server is the hosting server, capable of, by the mean of access token use, provide access to Resource Owner's protected resources.
Client	The client application that wants to access Resource Owner's restricted content. It needs Resource Owner's access grant to do it.
Authorization Server	The server responsible to issuing access tokens to clients after proper successfully resource owner authentication and authorization.

Table 2.4: OAuth 2.0 Roles. Adapted from [29]

The OAuth 2.0 RFC describes the interaction between these four roles (Figure 2.8):

1. **Authorization Request:** Client requests authorization from Resource Owner to access content.
2. **Authorization Grant:** Client receives authorization grant from Resource Owner, which is a credential representing that authorization.
3. **Authorization Grant:** The client requests an access token from Authorization Server, presenting the authorization grant.
4. **Access Token:** Authorization Server issues an access token if authorization grant is valid.
5. **Access Token:** Client requests the access to the protected content in Resource Server, providing the access token.
6. **Protected Resource:** If the access token is valid, Resource Server provides the requested content.

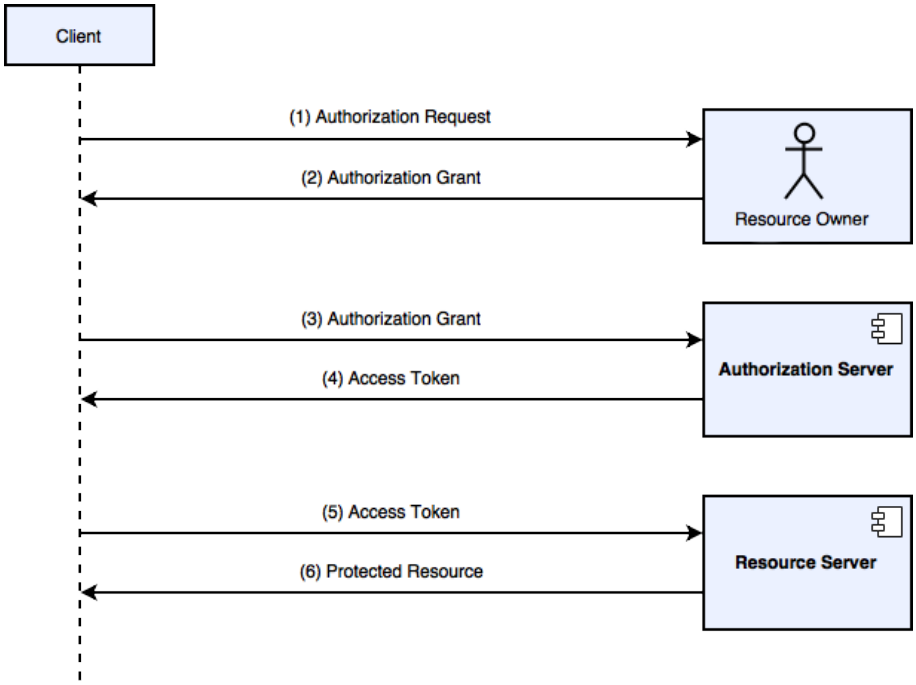

Figure 2.8: OAuth 2.0 flow chart. Adapted from [29].

SAML

SAML stands for Security Assertion Markup Language [32]. It is a standard which goal is to exchange authentication and authorization between subject (user), a service provider and a identity provider.

SAML defines encoding security assertions in a XML-based format [33]. SAML assertions are statements defined in a Extensible Markup Language (XML) schema. For instance, a assertion can state that a user (subject) "John Doe" has the email john@example.com and is member of the "general" group.

The security of SAML relies on trust relationships among involved parties and in the security of the transport protocols used for message exchanging. In figure 2.9 it is shown how the SAML standard workflow is.

1. **Request resource:** Client requests resource from Resource Server to access content.
2. **Redirect to SSO:** However, client is not authenticated. Resource server redirects client to an Identity Provider (IdP) so client can get authenticated.
3. **Request SSO Service:** The client requests an access token from Identity Provider.
4. **Request Credentials:** IdP issues an challenge that can be, for instance, username and password.
5. **Login:** Client responds to IdP challenge.
6. **Signed Access Token:** If client response is valid, IdP sends back a signed access token
7. **Access Token:** Client requests the access to the protected content in Resource Server, providing the access token.

8. **Protected Resource:** If the access token is valid, Resource Server provides the requested content.

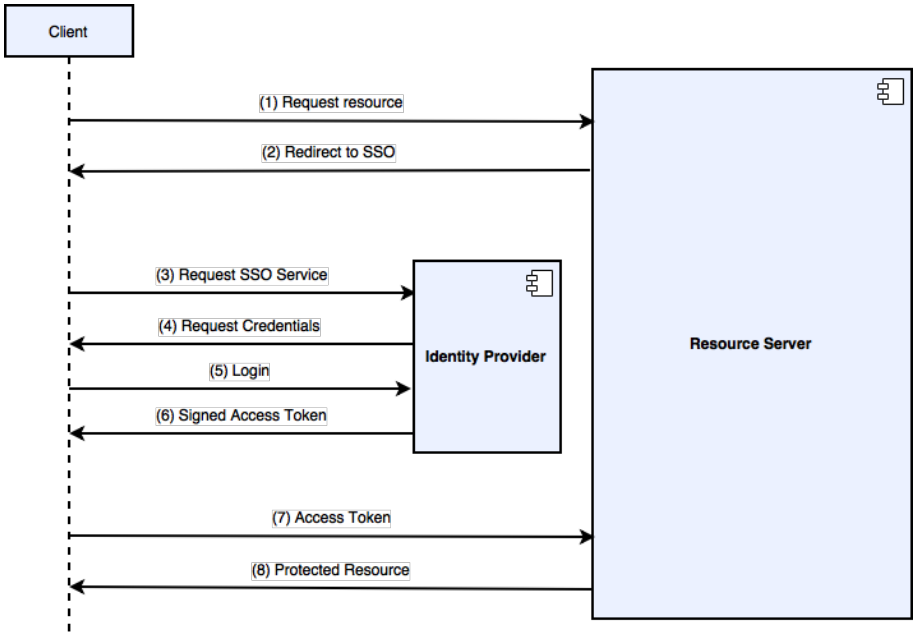

Figure 2.9: SAML flow chart. Adapted from [34].

SAML standard is useful when multiple services with multiple login credentials needs to centralize login service.

2.4.3 IMAGE ENCRYPTION

Nowadays, the transmission of medical images via the worldwide network, the Internet, has grown rapidly to a daily routine [35]. Nevertheless, computer networks are complex and espionage is a potential risk [36][37]. Moreover, the medical images are often distributed in the network of a healthcare institution or Internet with a lot of metadata related to patients' privacy [38][39]. So, we face a security issue when transmitting data over network [37].

For ethical reasons, medical images should not be sent in a clear text way so, each one has to be better protected [36]. Among the three pillars of security services (integrity, confidentiality and availability), confidentiality is an essential feature related to medical imaging [38]. Encryption is the best form and effective way of protection in cases like this [36][40].

As [38] states, image encryption is one of the most important fields of cryptography and Data Encryption Standard (DES) is one of the most algorithms is this area.

Nonetheless, other agents apply different techniques. In [35], author mention a solution which is based on a system that can partially encrypt i.e. "encrypts only the smallest portion of the data that makes the entire data set unusable". In same article [35], the author proposes a new method that makes selective encryption for JPEG images. This method uses Advanced Encryption Standard (AES) cypher.

DICOM standard does not specify the encryption mechanisms to be used. However, it refers to other encryption algorithms such as AES, DES, 3DES to encode the data [6]. DICOM recently released PS3.15 supplement for the standard that specifies in more detail some security measures to take care of.

2.5 DICOOGLE

Dicoogle is a PACS archive supported by a document-based indexing system and by peer-to-peer (P2P) protocols [41]. It has a modular concept given that it provides a software framework that allows developers and researchers to quickly develop a new functionality.

The platform replaces the traditional relational database with a more agile process of indexing and retrieval mechanism [42]. Dicoogle was designed to extract, index and store all the metadata presented in DICOM medical files, including private tags, without any engineering or configuration process [43][41].

The modular concept is provided by the plugin-based architecture (Figure 2.10) enabling different features to be separately developed and easily integrated [44]. The architecture allows developers and users to add new extractions like, for instance, storage plugins to deal with new necessities and without changing the core software [42].

Dicoogle has a Software Development Kit (SDK) created in order to simplify the development of new features [45] by third parties and assure compatibility. To develop, programmers need to implement the available interfaces and move the built package to Dicoogle Plugins directory. After this process, Dicoogle will automatically load the new modules on startup. Dicoogle SDK makes immediately available all operations related to storage, querying and indexation via its internal API [42].

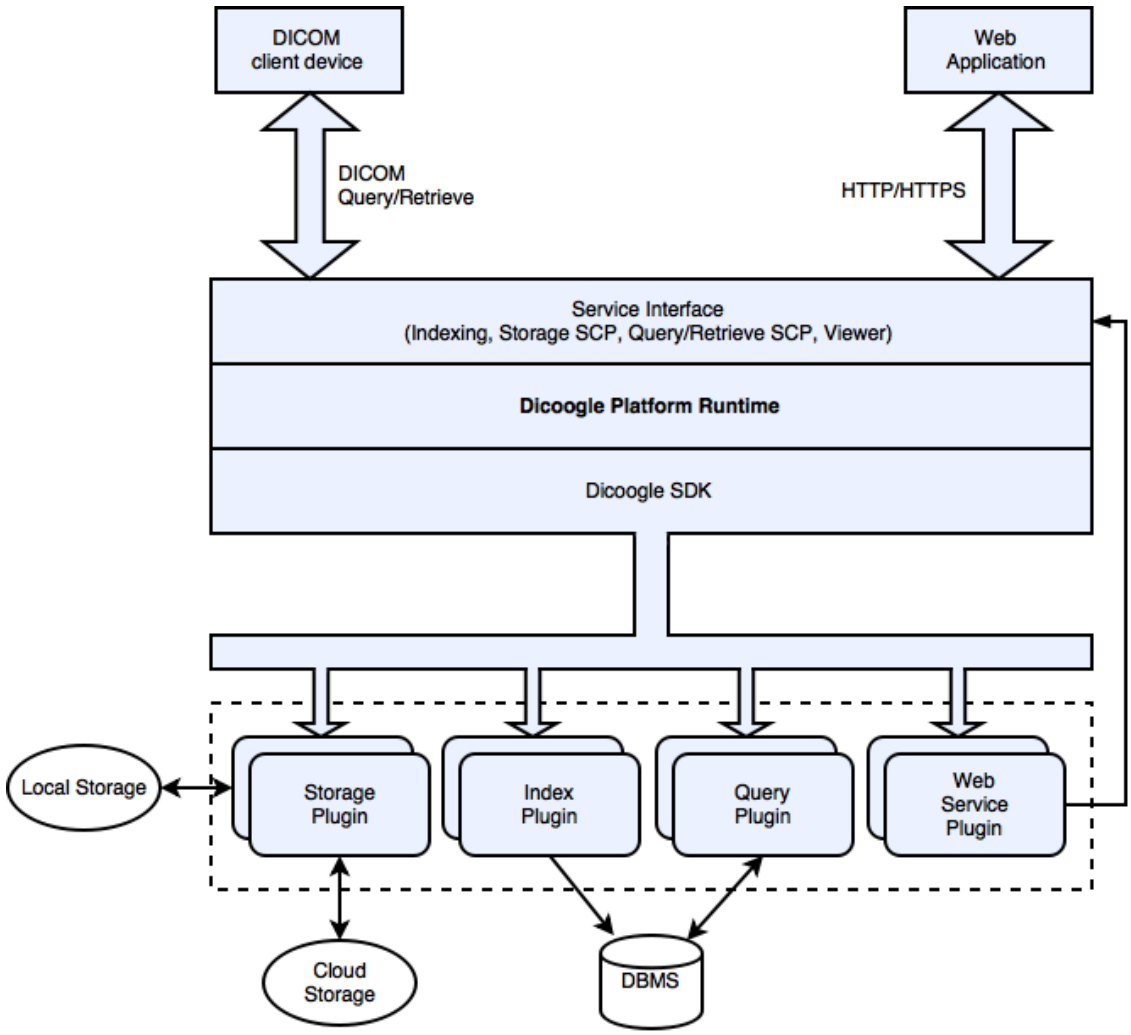

Figure 2.10: Dicoogle general architecture. Adapted from [42].

2.5.1 STORAGE PLUGINS

Storage plugins are responsible for storing and retrieving data [46]. They provide the persistence mechanisms and are triggered every time a storage operation is requested. At a more low level, storage plugins have the methods for retrieving the Uniform Resource Identifier (URI) (or Uniform Resource Locator (URL)), identification of DICOM Files as well as, the methods to retrieve a DICOM object matching a certain URL.

A simple implementation of this plugin can keep the relevant files in the local filesystem. However, a more complex development can extend Dicoogle to support remote storage services like, for instance, at cloud storage providers (e.g. Dropbox, Google Drive, Onedrive or Amazon S3).

2.5.2 INDEX PLUGINS

Index plugins are triggered every time a file is added to the system, just like storage plugins. In practice, an Index Plugin can be activated in two distinct ways: the first one is through the Application Programming Interface (API) (web services); the second one is through the web interface, where a

user can provide a path and select DICOM files to perform index operations.

The purpose of this type of plugins is to organise data in a format that allows its quick manipulation, including processes of storing and extracting data. At the end of the indexing process, a log file is presented, showing up, for instance, the number of errors and number of successfully indexed DICOM Objects.

A fully deployed instance of Dicoogle should have at least one Dicoogle Index Plugin [46].

2.5.3 QUERY PLUGINS

Query plugins allow querying to seek and access indexed data through retrieve methods. They have the mission to convert the data that fills the user search requirements in a given query to data representation compatible with Dicoogle web interface. Often, there is a query plugin coupled to each index plugin and bundled in the plugin set, precisely to make the conversion mentioned above.

2.6 CLOUD STORAGE SERVICES

With the growing market of cloud storage triggered by the advent of the network and global connectivity, there was a huge demand for outsourcing of storage to remote locations. Nowadays there are a lot of providers like Microsoft, Google or Amazon offering these services at a acceptable storage capacity/price ratio. Besides the price, these services offer horizontal scaling and great availability.

One of the great advantages of cloud services is its resilience. I.e., the system is prepared to, if a machine fails, readjust itself so the user never know that any machine failed[47].

Moreover, the healthcare institution saves budget that would be spent on a local data center infrastructure like hardware, software licensing air conditioning, fire alarms, physical security, electricity consumption and IT updates, for instance.

2.6.1 GOOGLE DRIVE

Google provides a file storage and synchronisation service not only for personal usage (with free plans) but also for enterprise application.

The service, which name is Google Drive, offers every user 15GB of free storage, shared among other Google apps like Gmail, the mail service, or Google Photos, the photo backup service. Besides the free plan, the user can purchase additional monthly or yearly storage space plans, as can be seen in Table 2.5.

Total Storage	Monthly price (euros)
15 GB	Free
100 GB	1.99
1 TB	9.99
10 TB	99.99
20 TB	199.99
30 TB	299.99

Table 2.5: Google Drive storage plans pricing. Adapted from [48], as it is in 29/01/2017

Apart from storage amount, Google Drive offers to developers a REST API, client libraries and documentation to help on Google Drive applications development.

The Drive REST API allows not only the basic operations of downloading and uploading files but also search for files, manage files metadata, create files, store application data and sharing personal files[49][50], for instance.

2.6.2 GOOGLE STORAGE

Google Storage is a RESTful online file storage service similar to Google Drive. However, Google Storage focus is on enterprise business since[51]:

- Space is unlimited
- Advanced storage modes: data can be stored in "cold" mode (rarely accessed) to save money
- REST API support for advanced business applications
- Allowance of resume a data transfer after failure
- Support for streaming

Storage Type	GB per month price (euros)
Multi-Regional	0.026
Regional	0.02
Nearline	0.01
Coldline	0.007

Table 2.6: Google Cloud Platform storage plans pricing. Adapted from [51], as it is in 29/01/2017

2.6.3 ONE DRIVE

Similar to Google Drive, Microsoft offers too a cloud personal storage starting with 5GB for users with the free plan. Paid plans increase the storage amount limit and give other benefits like access to proprietary Microsoft applications. In table 2.7 is the price list:

Total Storage	Monthly price (euros)
5 GB	Free
50 GB	1.99
1 TB	~5.83
5 TB	~8.33

Table 2.7: OneDrive storage plans pricing. Adapted from [52], as it is in 29/01/2017

Analogously to other user-level cloud storage services, Microsoft released a set of APIs for OneDrive to allow developers to develop web services and client applications. The development of these applications allows users of these web services and client apps to browse, view, upload or edit files stored on OneDrive. These APIs were made available via a SDK that is available for .NET Framework, iOS, Android and Python with a limited set of API for web apps and Windows.

2.6.4 AWS s3

AWS S3 (Simple Storage Service) is a service provided by Amazon with a novel storage utility with a "pay-as-you-go" charging model[53]. This object storage service comes with a web service interface (REST, Simple Object Access Protocol (SOAP)) to store and retrieve any amount of data from any part of the world. Amazon claims its 99.99% availability[54] due to its support by a large power of computation distributed across multiple data centers[55].

Standard Storage	Monthly price (dollars)	Infrequent Access Storage
First 50 TB	0.023 per GB	0.0125 per GB
Next 450 TB	0.022 per GB	0.0125 per GB
Over 500 TB	0.021 per GB	0.0125 per GB

Table 2.8: AWS S3 plans pricing. Adapted from [54], as it is in 30/01/2017

Amazon S3 distinguishes its service as a complete storage platform. It claims several key features[54] like, for instance:

- **Simplicity** - Providing a web-based management console, mobile app, REST APIs and SDKs for easy integration with third party applications
- **Durability** - Providing the service in several regions of world, AWS S3 includes redundancy and the possibility to replicate data across multiple data centers
- **Scalability** - AWS S3 allows to store as much data as user requirements. Users can scale up and down as their needs at the time so require.
- **Security** - AWS S3 supports data transfer over Secure Sockets Layer (SSL) and automatic encryption when data is stored

2.6.5 DROPBOX

Dropbox is a file storage service that offers cloud storage, file synchronisation, personal cloud and a client software application among other permissions like sharing files and folders[56].

Similarly to previous services, Dropbox provides paid plans: Pro, Business and Enterprise (Table 2.9) and a API permitting developer's third party applications to access the stored data.

Dropbox uses AWS S3 services as third party cloud to store files[57].

Total Storage	**Monthly price (euros)**
2 GB	Free
1 TB	9.99
∞	12

Table 2.9: Dropbox storage plans pricing. Adapted from [58], as it is in 29/01/2017

2.7 ACCESS CONTROL MECHANISMS

Access control is the protection of resources from unauthorized agents [59]. The use of system resources is granted by a mean of permission, that process is called Authorization.

As the proposed system was an access control mechanism, a study of the relevant tools for this project was made. It is important to emphasize that this study was carried out taking into account its integration in the Dicoogle platform.

2.7.1 OACC

The Object Access Control (OACC) [60] is an application security framework that provides authentication and authorization services. This framework focuses on providing a fully implemented Java API to create security in accessing resources in a single access control paradigm. This API facilitates the process of authenticating users and controlling access to resources in an application.

OACC needs a DBMS-backed data store. It can be MySQL.

2.7.2 APACHE SHIRO

Apache Shiro [61] is an opensource project under the Apache Software Foundation. It is a Java security framework that provides developers services to secure the applications developed: authentication, authorization, cryptography and session management. It can be used in any type of application (mobile, web, command line or even enterprise).

However, Apache Shiro is not what Dicoogle platform needs. Due its complexity and difficulty settings during its test, it was decided that Apache Shiro will not be used to provide authentication and authorization on a multi-archive paradigm of Dicoogle.

CHAPTER 3

System Wide Requirements

In this chapter there is the overall system architecture and implementation of the developed plug-ins, focusing on system functional and non-functional requirements that the proposed system must accomplish.

3.1 system wide requirements

On software engineering field, the software plan has some software requirements specification that the a software system must satisfy after concluded the development. It is usually divided into functional and non-functional requirements and may include a set of real-world actions called use cases. On one hand, the functional requirements define what the system should do, on the other hand, the non-functional describe how the system should behave[62]. This chapter illustrates the most important ones, taken into account while developing the proposed solution.

3.1.1 functional requirements

In this project, it is possible to enhance some crucial functional requirements to satisfy the objective of this work and to keep compatibility with Dicoogle SDK and DICOM standard.

interface to external services

The system should provide an interface to be used by external services. This interface should be accessible through REST API. Besides the REST API the system should have a programatic Java API so developers can use the interfaces available.

The provision of the interface allows external services or applications to access to the management system of accounts, permissions and resources required for availability in a health care organisation. The API delegates this management issues to the present system rather than assigning management to these external applications.

MANAGE ORGANISATIONS

The system must provide both REST and programatic API of services to manage the organisation level on the hierarchy model. It should be possible to add and remove a user from an organisation but also create and remove the organisation itself once a new or old organisation is added or removed to and from the multi archive system.

MANAGE FACILITIES

New facilities belonging to an organisation should have both REST and programatic API services that gives allowance to add user to that facility and add resources belonging to that facility. In addition, a Facility will belong to an Organization. It should therefore be possible to assign an Organization to a Facility, or vice-versa, in the services available in the API.

MANAGE USERS

The system must have a user management subsystem and provide REST and Java API services to access it. Operations like create, edit and remove a user must be possible. However, those operations are not enough. The User must belong to an organisation and a facility, and the resources uploaded/sent to the system must belong to that specific user and all users of the user facility. Having the "User" entity in the system allow us to give different permissions to each one.

MANAGE PERMISSIONS

Permissions are the entity that will allow a user to have access to a resource. A permission should be composed by a category, operation and in some cases, a resource. An REST and Java API to manage those permissions is a requirement.

SHARE RESOURCES

Users that owns a resource or a batch of resources must have the ability to share those resources. This means that a user A that owns the resource 1, must be able to give permission to user B (that doesn't have the allowance to access resource 1) to access the resource.

DICOOGLE COMPATIBILITY

The system has the capability of being configured using a file that contains user specified settings. This method is already being used by Dicoogle default available plugins. Values like a flag mentioning if multiple archive system is or is not activated or even which is the DICOM model level by default when indexing new studies (we will address further into this subject) should be easily configured. The settings exemplified above are set after the plugin initialization, i.e. in run-time.

INTEGRITY CONTROL

The first one is the capability of being configured using a file containing user specified settings. Values like data folder location, databases names, and other important application/environment settings, need to be present on a file using XML notation. These setting are configured after the plugin initialization. Another important requirement is the use of transactions in order to keep the data integrity, assuring that the DICOM information model is secure, for instance, if an error occurs while inserting a given study, the images of that study can not be indexed/stored because all images require a study parent.

ACCESS CONTROL

The authentication and authorisation usage of API should only be given to authorised personnel from the organisation which owns the resource required.

In addition, authorisation should be given only if the applicant is allowed to carry out the intended operation in the required category. Therefore, a user who wishes to obtain a resource must, by necessity, have read permissions for the resource category and permission to access the particular resource.

3.1.2 NON-FUNCTIONAL REQUIREMENTS

PRIVACY AND CONFIDENTIALITY

Personal medical data privacy is a requirement of great importance and is a very delicate issue. Patient data must be handled carefully and its storage on third parties and transmission across public networks should be handled carefully. The storage of medical information on cloud providers encompasses the file transmission over public (and unsafe) networks. Additionally, institutions have no guarantee that cloud providers ensure the data privacy. To take full advantage of cloud computing and storage, like the scalability, redundancy and performance, there is the challenge to protect the privacy of the patient's medical data.

PORTABILITY

The system should support different personal cloud providers. As stated in Chapter 2, there are multiple cloud providers that offer free plans for personal cloud storage (Google Drive, OneDrive and Dropbox, for instance). So, supporting portability is a key point.

Besides the multiple cloud providers support, the system should also allow different implementations. So, providing a library would allow the development of more applications that could port the system to different platforms.

PERFORMANCE AND ROBUSTNESS

Medical institutions and staff require almost a full availability of the system, a good performance and robustness due to the sensitivity of the healthcare services provided. For instance, the system

must be available when a physician must perform a study and cannot have a significant delay or a system crash. Therefore, minimal services must be provided.

ACCESSIBILITY AND AVAILABILITY

It is a requirement that the services should be accessible from anywhere with a high ratio of availability. However, security measures must be taken into consideration. This requirement is relevant when a physician wants to access information from another location (telemedicine scenarios).

Thus, availability and accessibility are important issues to make the system efficient and able to meet the healthcare organisations interests.

3.1.3 USE CASES

During the project planning, several use cases scenarios were identified. Bellow, it is a description of each one in more detail. In figure 3.1 it is shown the use case diagram of all the following diagrams.

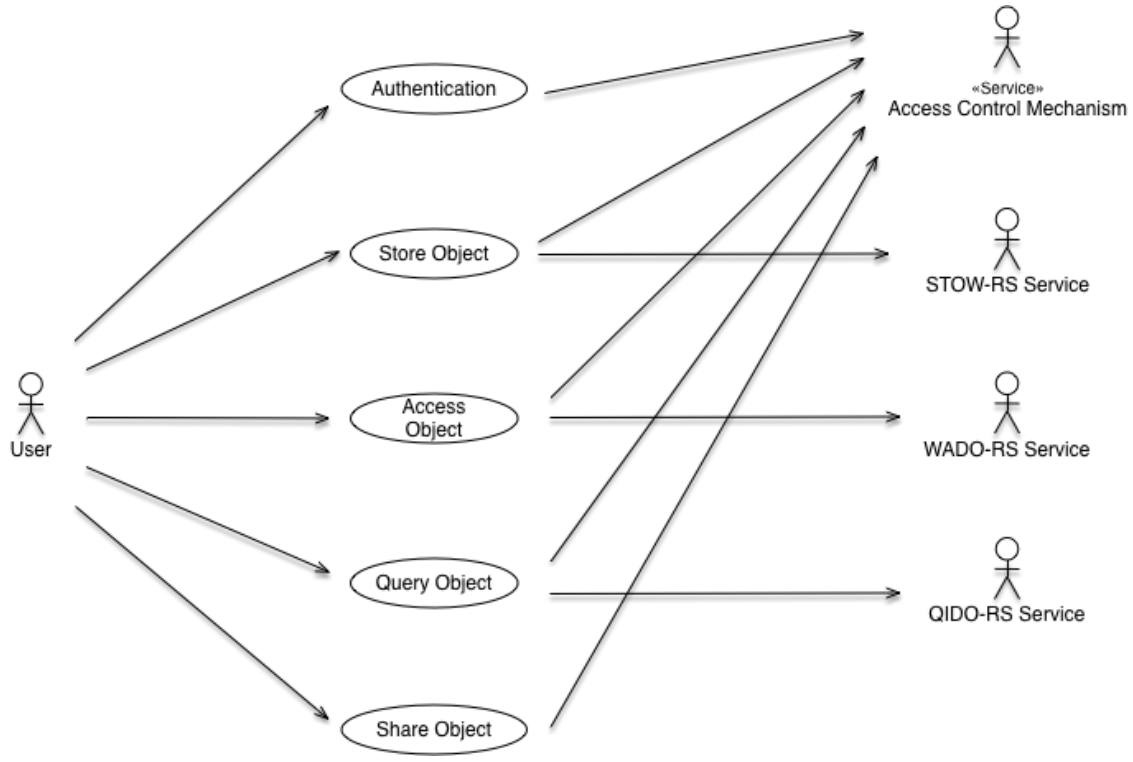

Figure 3.1: Use case diagram of the system

REVOKING UNAUTHORISED USERS

With this system, the web access via API is always protected. A nonexistent user cannot have access to resources. On the same line, a user that exists and is logged in the system but has not permission to perform certain types of operations on certain types of resources as well as have access or permission to see through the requested operation.

STORE OBJECTS

A user or modality may need to store objects via REST services. To do that, they may use STOW-RS. When the request is made, its requester needs to provide authentication data. By using that authentication data, the system will perform or not the operation, based on the authorisation clearance of the requester.

ACCESS OBJECTS

Web access to DICOM objects is done via WADO. So, in this service, access permissions should be verified in a similar way as the previous item "Store objects".

SEARCH OBJECTS

The Dicoogle platform supports query over DICOM objects, following the QIDO-RS standard. Once again, queries must be protected: only authorised personnel should have access to query services.

SHARE OBJECTS

The information system should allow a sharing mechanism. A user, who has permissions to do so, must be able to share access permissions to another user on the system. This feature will allow sharing of resource access, in this case DICOM objects, between facilities within the same organization or even between organizations, to persons belonging to different Roles.

CHAPTER 4

Architecture and Implementation

In this chapter is presented the overall architecture and implementation of the proposed system in order to provide a functional access control mechanism for a multi-archive PACS. Since accessing speed and usability were strong factors to take care of, technologies and implemented methods were carefully chosen. Its description is also present in this chapter.

4.1 INTRODUCTION

In the present thesis, it is expected an extension to the Dicoogle PACS system that can handle a multi-user environment. State of the art review shows that there is a gap of integrated multi-archive solution. The current solution does not allow the same archive to serve multiple organisations, installations, users, and permissions.

The developed solution aims to allow, for example, the singular sharing of instances to another user belonging to the system and, simultaneously, to one of the organisations constant in the archive, as well as to allow the integrated management of the PACS at the organisational level.

Taking into account the current state of the art and the Dicoogle system in the present state of production, it was considered the development of a access control mechanism system. The development option arises from the fact that, among the existing solutions, none is adequate to the current constraints with regard to requirements and integration of technologies.

4.2 PROPOSAL

In this thesis it is proposed a access control mechanism system that can manage a multi-archive PACS in a simple instance.

Dicoogle is a PACS archive that does not support the multi-archive environment. However, this PACS solution supports the development of plugins (written in Java) so a developer can extend the main functions of the existing software. The system already has a user authentication mechanism used in the web interface.

It is proposed the development of a library that interacts with the Dicoogle PACS and with Dicoogle SDK in order to allow the restriction of access to resources, based, for this, on a permission managing information system.

The development of this library allow that web requests can be authenticated including services like, for instance, QIDO-RS, WADO or STOW-RS. Besides these obvious services, it will also be allowed the addition of arbitrary permissions over resources like, for example, permissions for editing the data of a facility or even the edition of information about a user.

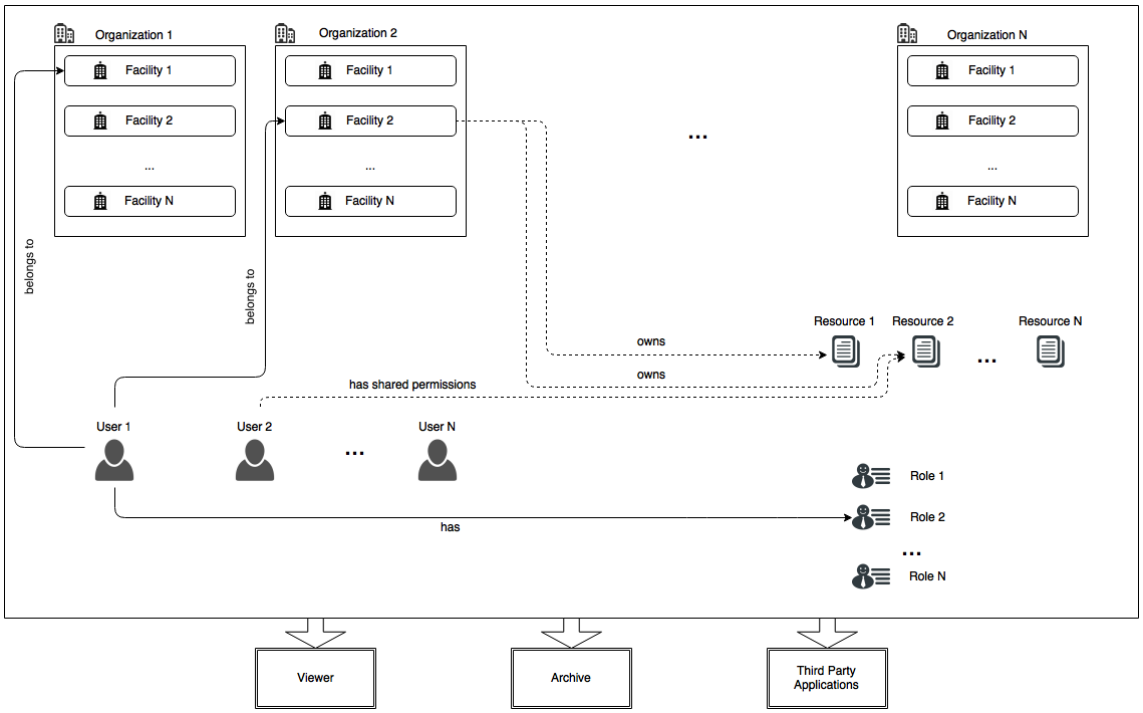

Figure 4.1: General entities in proposed system

In general, at the end of this thesis, a software as-a-service should be available, allowing extensions and application development to manage the access control mechanism (Figure 4.1) service in a multi-archive Dicoogle PACS context.

In the following sections, it will be revealed the architecture and implementation of these systems, which in turn will be explained in more detail.

The figure 4.2 illustrates an overview of the proposed system modules and its interaction with Dicoogle core.

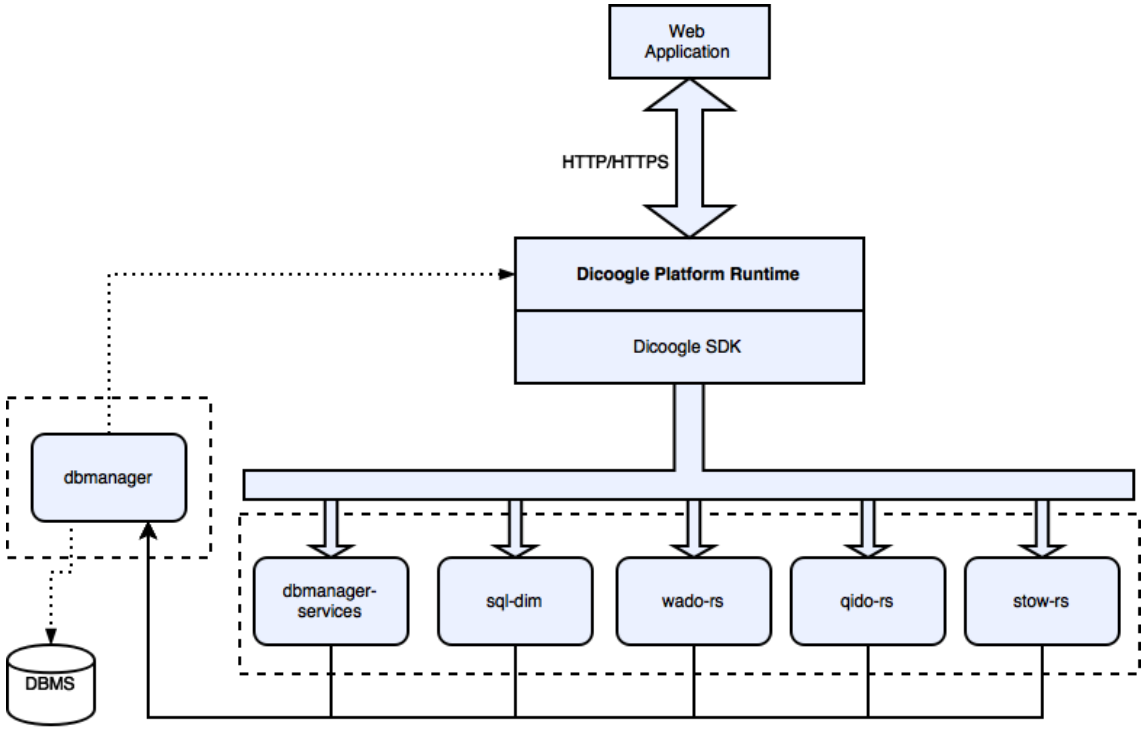

Figure 4.2: Overview of proposed system modules

4.3 DATA MODEL

In the context of the requirement analysis, a data model was developed.

First of all, the data model should incorporate the DICOM Data Model or, at least, some attributes containing the most relevant data. Entities like Patient, Study, Series or Instance are essential (Figure 4.3). Besides, these entities must be interconnected with the access control system.

The purpose of the existence of this model is to index all the relevant attribute information of the DICOM metadata. Bellow, there is a description of those attributes. However, not all of them are presented for a question of relevance to the developed work.

- **Patient:** In this entity, there is reference to the DIM levels of PatientID, PatientAge, PatientName, PatientSex and PatientBirthDate.

- **Study:** The Study entity is the level immediately after Patient. Besides the natural StudyInstanceUID, it also has a reference to its parent ID, in this case, PatientID.

- **Series:** Following the same reasoning, Series is the level immediately after Study and has also the unique identifier SeriesInstanceUID and the parent id connection (StudyInstanceUID).

- **Instance:** In this last entity, there is information about the Instance of the DICOM Object. Its parent id is related to the previous entity Series and is SeriesInstanceUID.

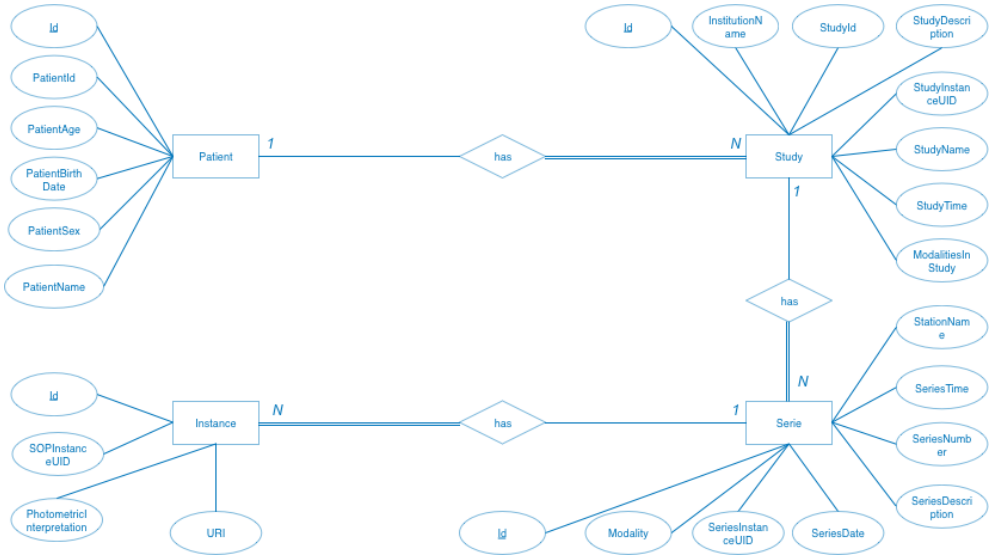

Figure 4.3: DIM Entity relationship diagram proposal for DIM storage in database.

Secondly, the system should manage the access control system (Figure 4.4). To do so, a model was developed based on permissions granting or denial. After the requirement analysis, some entities were classified as required to a multi-archive system.

DIM Entities (Figure 4.3) are interconnected with the access control mechanism via Entity "Resource". A resource is identified by the UID of the Patient, Study, Series or Instance. By default, the level is Study, i.e. when a file is added to the system, the Resource created will be of type Study and identified by the StudyInstanceUID. All the permissions given to the resource will be at Study level. However, that parameter can be changed and the default level can be Patient, Series or Instance, besides Study.

On a multi-archive PACS, the system should support multiple Organizations. Each organisation has one or more Facilities with employees or staff (Users) associated. A user can have associated one or more Roles and each Role one or more Permissions.

This first approach allows the system admins to assign roles with different permissions to different users. For instance, a physician can have, simultaneously, the role that gives permissions to perform exams and the role to review those exams. However, another different user, that has only the administration (or bureaucratic) role, must not have permissions to access those exams.

A permission must have information about what it represents. The entity Permission is a "super-entity" that is related with entities like Resource, Category or Operation. Resource is the access control system representation of a DICOM Object on the access control system. Category is used to identify the type of permission that we are dealing with. For instance, a Category can be an Object, Operation, Organization, Role, and so on. Operation is the entity responsible for storing the possible operations that will be available in the archive, like "Read", "Write", "List", and more.

So, at the end, Facilities have Resources (DICOM Objects) associated and Users associated with that Facility have access to those resources since they have permission to perform the desired operations.

Additionally, the data model must support the Resource sharing requirement. To do so, it was added a connection directly from User to resource. This allows a User to share a Resource with another user from another different Facility and give it a specific permissions.

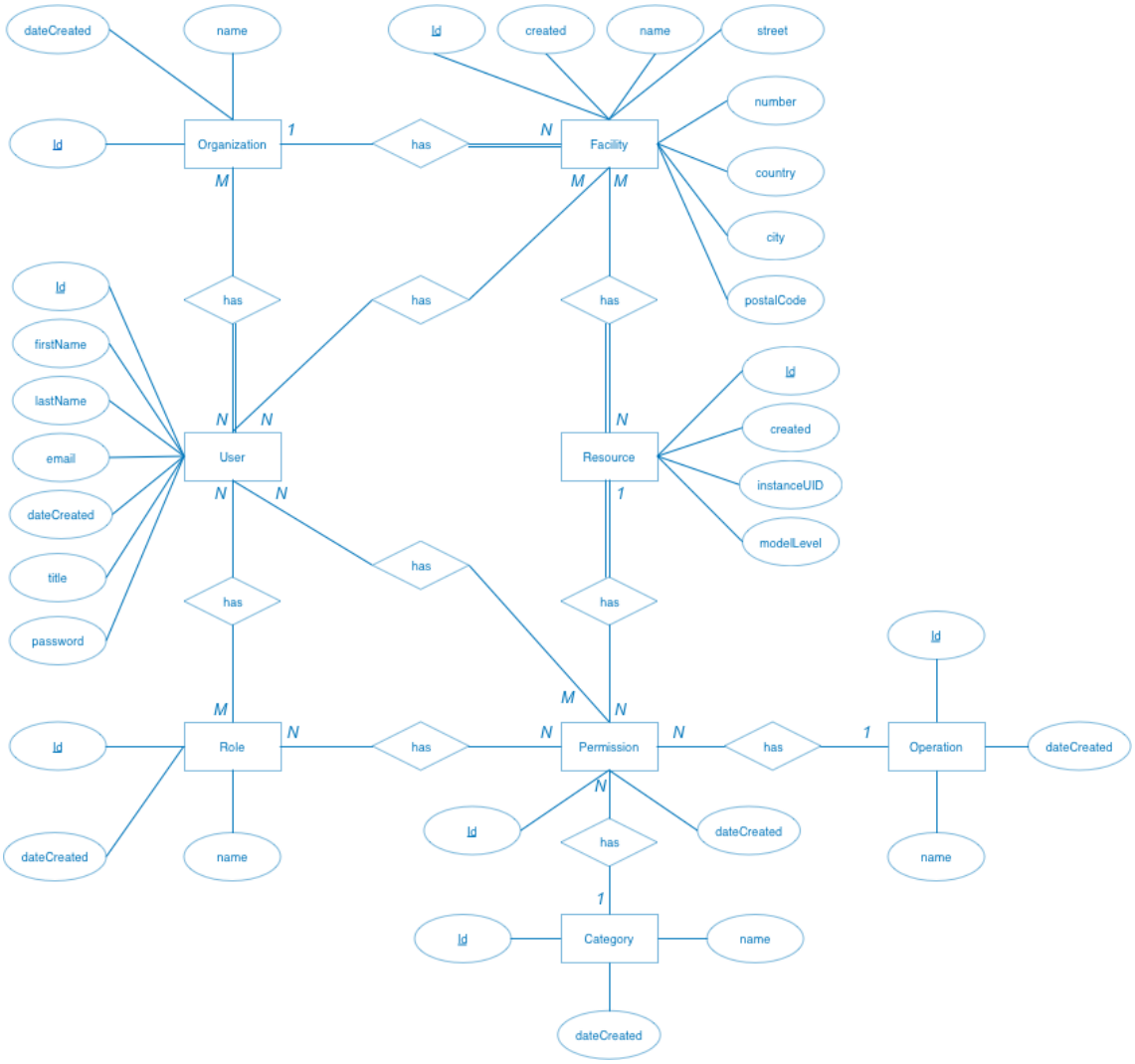

Figure 4.4: Access control mechanism entity relationship diagram proposal.

After this analysis and planning, there was created all the entities needed. The technology used was Java with Hibernate Framework [63], an implementation of the native Java Persistence API (JPA). The created entities are as follows, with the example of the source code developed to support them:

- **Organization:** An Organization is the entity that wants to have a multi-archive. It has a set of Facilities and Users (or staff) associated with it.

```
public class Organization implements Serializable {
    @Id
    @GeneratedValue
    private Long id;

    @Column(nullable = false)
    private Date createdDate;
```

```
    @Column(nullable = false)
    private String name;

    @OneToMany(mappedBy = "organization", cascade =
        CascadeType.ALL, fetch = FetchType.EAGER)
    private List<Facility> facilities;

    @ManyToMany(mappedBy = "organizations", cascade =
        CascadeType.ALL, fetch = FetchType.EAGER)
    private List<User> organizationUsers;

        ...
```

Listing 1: Organization entity implementation

- **Facility:** A Facility is the physical structure that performs studies to patients. It has users (staff) and Resources associated to it. Besides these connections, the entity "Facility" has attributes like name, contact, address and a main Organization.

```
public class Facility implements Serializable {
    @Id
    @GeneratedValue
    private Long id;
    @Column(nullable = false)
    private Date createdDate;
    @Column(unique = true, nullable = false)
    private String name;
    @Column(nullable = false)
    private String street;
    @Column(nullable = false)
    private String number;
    @Column(nullable = false)
    private String postalCode;
    @Column(nullable = false)
    private String city;
    @Column(nullable = false)
    private String country;
    @ManyToOne
    @JoinColumn(name = "organization_id")
    private Organization organization;
    @ManyToMany(mappedBy = "facilities", cascade = CascadeType.ALL,
        fetch = FetchType.EAGER)
```

```
    private List<User> facilityUsers;
    @ManyToMany(mappedBy = "facilities", cascade = CascadeType.ALL,
        fetch = FetchType.EAGER)
    private List<Resource> resources;

        ...
```

Listing 2: Facility entity implementation

- **User:** User is the entity on which every agent is inserted. User must have attributes like first and last name, email and password. A User belongs to a Organization and assigned to a Facility. The entity has a set of Roles and Permissions of shared Resources.

```
public class User implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    @Column(nullable = false)
    private Date createdDate;
    @Column(nullable = false)
    private String title;
    @Column(nullable = false)
    private String firstName;
    @Column(nullable = false)
    private String lastName;
    @Column(unique = true, nullable = false)
    private String email;
    @Column(nullable = false)
    private String password;
    @Column(nullable = false)
    private Boolean hidden = false;
    @ManyToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
    private List<Facility> facilities;
    @ManyToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
    private List<Organization> organizations;
    @ManyToMany(mappedBy = "users", cascade = CascadeType.ALL,
        fetch = FetchType.EAGER)
    private Set<Role> roles;
    @ManyToMany(mappedBy = "users", cascade = CascadeType.ALL,
        fetch = FetchType.EAGER)
    private List<Permission> sharedPermissions;
```

```
    ...
```

Listing 3: User entity implementation

- **Role:** A Role is an entity that has a set of permissions and a name. For instance, the hypothetical role FACILITY_ADMIN should have permissions to manage the Facility attributes, Resources and Users. Besides, the Role has the list of the Users associated with it.

```
public class Role implements Serializable {
    @Id
    @GeneratedValue
    private Long id;
    @Column(nullable = false)
    private Date created;
    @Column(unique = true, nullable = false)
    private String name;

    @ManyToMany(mappedBy = "roles", cascade = {CascadeType.REMOVE,
        CascadeType.PERSIST, CascadeType.REFRESH,
        CascadeType.DETACH}, fetch = FetchType.EAGER)
    private List<Permission> permissions;

    @ManyToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
    @JoinTable(name="Role_User")
    private List<User> users;

    ...
```

Listing 4: Role entity implementation

- **Permission:** Permission is the entity consulted to check if a User can access to a Resource or perform an Operation. It has connections to Operation, Resource and Category. A Permission is not obligated to have a connection to Resource: there are some cases in which there is not a Resource associated. Giving an example, a Permission to allow the "Edit" Operation on the Category "Facility" does not have a Resource. Additionally, Permission entity has a set of users that have the shared Permission and the set of Roles on which is associated.

```
public class Permission implements Serializable {
    @Id
    @GeneratedValue
    private Long id;
```

```
    @Column(nullable = false)
    private Date created;
    @ManyToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
    @JoinTable(name="Permission_Role")
    private List<Role> roles;
    @ManyToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
    @JoinTable(name="Permission_User")
    private List<User> users;
    @ManyToOne
    private Operation operation;
    @ManyToOne
    private Resource resource;
    @ManyToOne
    private Category category;

    ...
```

Listing 5: Permission entity implementation

- **Resource:** A Resource can be any level of the DICOM Information Level, such as Patient, Study, Series or instance. It has a UID attribute and a field indicating from which level of the DICOM does it belong.

```
public class Resource implements Serializable {
    @Id
    @GeneratedValue
    private Long id;
    @Column(nullable = false)
    private Date created;
    @Column(nullable = false, unique = true)
    private String instanceUID;
    @Column(nullable = false)
    private String modelLevel;
    @OneToMany(mappedBy = "resource", cascade = CascadeType.ALL)
    private Set<Permission> permissions;
    @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    private List<Facility> facilities;

    ...
```

Listing 6: Resource entity implementation

- **Category:** This entity has the attribute name that indicates if the Permission is to a Facility, Organization, Resource, or many others.

```
public class Category implements Serializable {
    @Id
    @GeneratedValue
    private Long id;
    @Column(nullable = false)
    private Date created;
    @Column(nullable = false, unique = true)
    private String name;
    @OneToMany(mappedBy = "category", cascade = CascadeType.ALL)
    private Set<Permission> permissions;

    ...
```

Listing 7: Category entity implementation

- **Operation:** The Operation entity defines which operation is the Permission referenced for.

```
public class Operation implements Serializable {
    @Id
    @GeneratedValue
    private Long id;
    @Column(nullable = false)
    private Date created;
    @Column(unique = true, nullable = false)
    private String name;
    @OneToMany(mappedBy = "operation", cascade = CascadeType.ALL)
    private Set<Permission> permissions;

    ...
```

Listing 8: Operation entity implementation

On figure 4.5 is presented the database diagram generated by MySql Workbench, a visual tool for MySql.

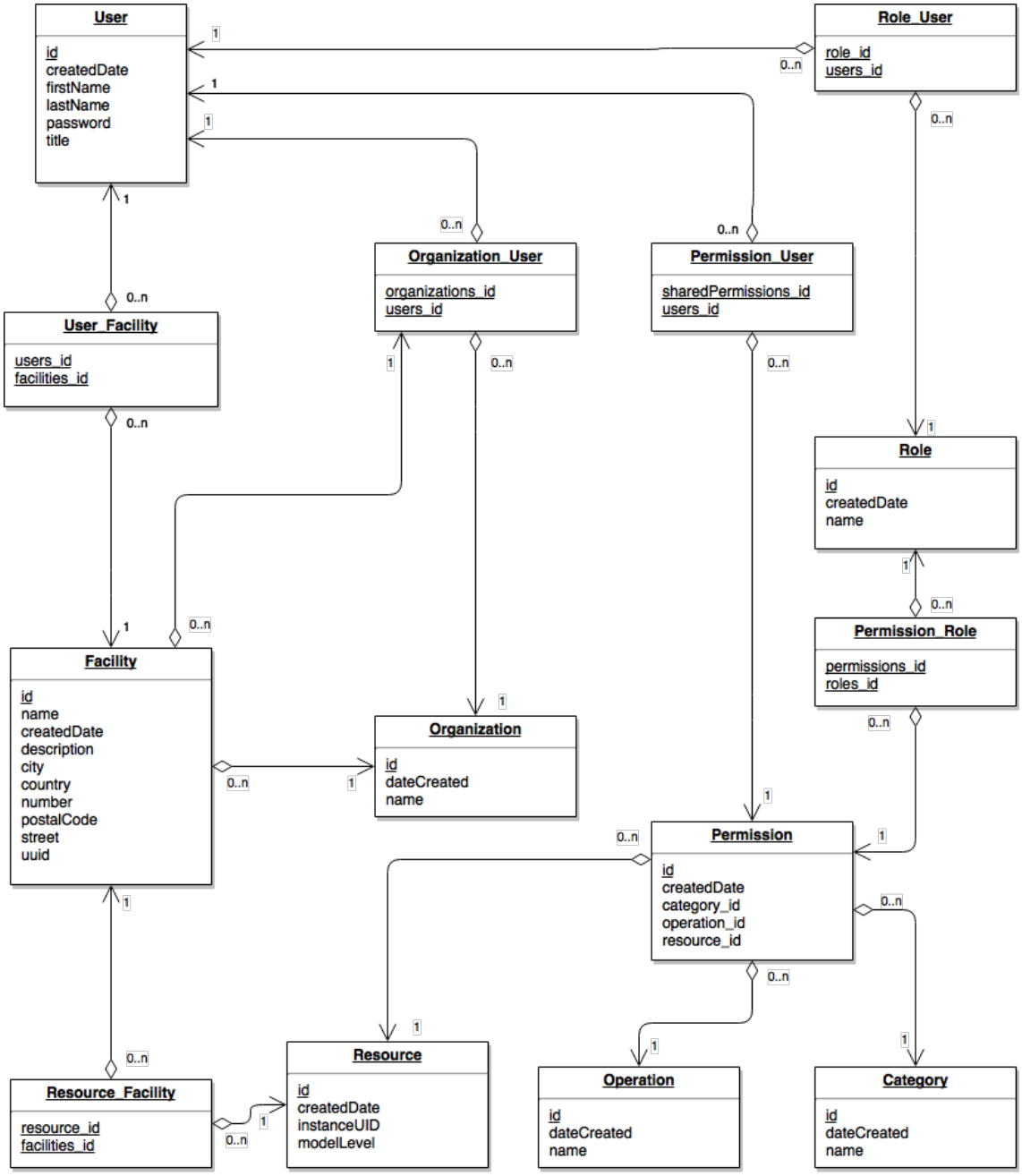

Figure 4.5: Database diagram proposal.

4.4 DATA PERSISTENCE

4.4.1 SERIALIZERS

For each of the entities described above, Serializers have been created so that future users of the developed library can obtain required objects (such as Organization, User, Permission) in the JavaScript Object Notation (JSON) format.

Listing 9 Serializer source code for the Facility entity, with the JSON object returned for a sample Facility object. It should be noted that the code and output (Listing 10) presented are mere examples. There are Serializers for all entities described above, where the relevant attributes of each entity are

explained.

```
public class FacilitySerializer extends JsonSerializer<Facility> {
    @Override
    public void serialize(Facility facility, JsonGenerator jgen,
        SerializerProvider provider)
            throws IOException, JsonProcessingException {
        jgen.writeStartObject();
        jgen.writeNumberField("id", facility.getId());
        jgen.writeStringField("createdDate",
            facility.getCreatedDate().toString());
        jgen.writeStringField("name", facility.getName());
        jgen.writeStringField("number", facility.getNumber());
        jgen.writeStringField("postalCode", facility.getPostalCode());
        jgen.writeStringField("street", facility.getStreet());
        jgen.writeStringField("uuid", facility.getUuid());
        jgen.writeStringField("city", facility.getCity());
        jgen.writeStringField("country", facility.getCountry());

        if (facility.getOrganization() != null) {
            jgen.writeStringField("organization_id",
                facility.getOrganization().getId().toString());
        }

        jgen.writeEndObject();
    }
}
```

Listing 9: Serializer example for Facility entity

```
{
    "id": 132,
    "createdDate": "2017-04-09 00:23:42.0",
    "name": "RBACTest",
    "number": "1",
    "postalCode": "4000",
    "street": "rbac street",
    "uuid": "1001",
    "city": "rbac city",
    "country": "rbac country"
```

```
}
```

Listing 10: Serialization of a Facility object example

4.4.2 MANAGERS

After the development of the entities and their integration with Java Database Connectivity (JDBC), there was a need for a layer of abstraction that facilitated the use of the library. The manager layer was created for that purpose. The use of this layer of abstraction obligated to delegate procedures to operations on the database.

After analysing the necessary and potentially most used operations, we opted for the development of two managers: UserManager and ResourceManager.

The UserManager is responsible for checking a user's authorisation and permissions, as well as obtaining a user based on the session token. Contains methods such as userHasAuthorization (checking a user's authorisation to perform an operation), userHasPermissions (checking permission to perform an operation in a certain category or perform an operation on a specific resource or resource set), getUser (obtain a User object based on session token).

On the other hand, there is the ResourceManager, responsible for creating resources and associated permissions in the database. Contains methods such as createResource based on the resource's DICOM object, createResourceWithUser for resource creation with associated permissions, or even createResourceWithFacility for creating a feature and association belonging to the Facility in question.

The existence of these layers of abstraction aims at allowing simplification of the development and logic associated with documented operations.

4.5 SERVICES

In order to provide an interface for developers, it was developed a set of services that allows the interaction with the database. Those services are provided by a REST API. All the services were developed as a Dicoogle plugin (Jetty service plugin) and the main goal was to have access to the basic create, read, update and delete (CRUD) functions of the proposed access control mechanism system.

However, those services must be protected against unauthorised accesses. To do so, it was developed a filter, called CheckPermissionFilter, that works as a barrier between the requester of the operation and the operation itself.

Once the reception of the request takes place, the filter checks which permissions are needed to perform the requested operation. After that procedure, as shown in figure 4.6, the CheckPermissionFilter searches for the HTTP request header attribute "Authorization" where must be in a Dicoogle session token, represented by an alphanumeric string of characters.

A connection to Dicoogle's core is made to get data about the token such as the user that is logged in. Since there are distinct accounting systems, it is required to verify, first of all, if the user is a member of the access control system. Secondly, the system will check 1) if one of the user's

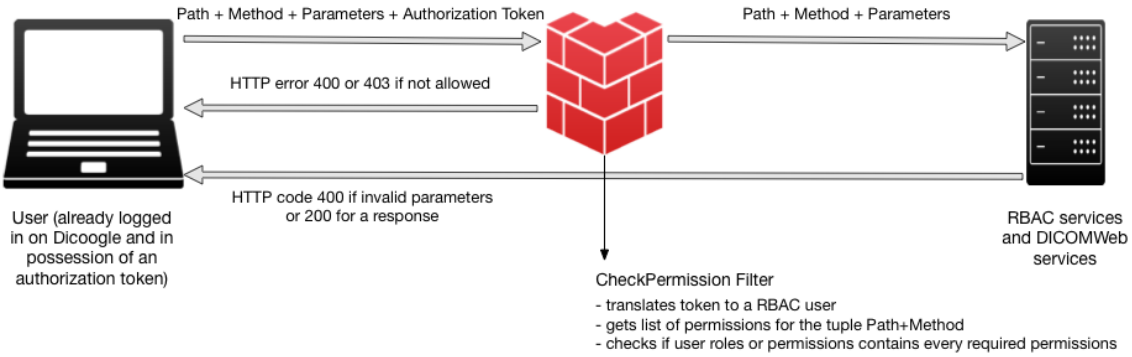

Figure 4.6: CheckPermissionFilter usage scheme

roles contains one permission or, if it is the case, all the permissions to perform the operation; 2) if a permission was shared by another user that allows the requester to accomplish the operation.

After the authentication and authorisation testing, there are two results possible: the user is able to perform the operation and the user is not authorised. In the first case, the requester gets a 400 (Bad Request) HTTP error code when is accessing to an invalid service, or a 403 (Forbidden) HTTP error it has not authorisation to perform the operation. In the second case, CheckPermissionFilter allows the user to access the required path.

The following subsections will describe the services developed. On each category, there is a table representing the endpoint of each service, allowed methods and the expected response.

Note that in all the services, except the Login service (Section 4.5.1) there is required that a HTTP Header "Authorization" must be sent and contain the session token relative to the requesting user, and the token must be valid. That token is obtained after a successful request to Login service.

4.5.1 LOGIN

The Login service allows a client to authenticate to the Dicoogle platform. After this authentication, a session token in JSON format is returned and can then be used to perform other operations in the system.

- **Endpoint** {server}/multi_archive/login
- **Description** This endpoint allows to Login on Dicoogle. Providing a username and a password to this service will allow to obtain a access token required to almost every service above.

Method	**Parameters**	**Response**
POST	username, password	HTTP 200 Response Code and JSON with the attributes *token* and *status*, providing the status with "success" or "error"

Table 4.1: Method allowed in Login webservice, with its required parameters.

4.5.2 LOGOUT

This service is used to log off the user.

- **Endpoint** {server}/multi_archive/logout
- **Description** This endpoint allows to Logout from Dicoogle. Providing the access token, when login was made, to this service will logout and make that token deprecated.

4.5.3 MANAGE USERS

The user management service allows the creation, edition, and removal of users. However, if the user does not have permissions to perform operations in this service, the service will not execute the request.

- **Endpoint** {server}/multi_archive/users
- **Entities** User
- **Description** This endpoint allows the CRUD operations related to the entity User.

Method	**Parameters**	**Response**
GET	id or email or no parameters	JSON containing *id, title, firstName, lastName, email, createdDate* or HTTP 400 Error Code when user not found
POST	id, firstName, lastName, email, password	JSON containing *id, title, firstName, lastName, email, createdDate*, which can be a user created at the time or earlier
DELETE	id or email	HTTP 200 Response Code with no data or HTTP 400 Error Code when user not found

Table 4.2: Method allowed in User webservice, with its required parameters.

4.5.4 MANAGE FACILITIES

This service allows the creation, edition and removal of facilities on the system.

- **Endpoint** {server}/multi_archive/facilities
- **Entities** Facility
- **Description** This endpoint allows the CRUD operations related to the entity Facility.

Method	Parameters	Response
GET	id or name or no parameters	JSON containing *id, name, number, postalCode, street, uuid, city, country, createdDate* or HTTP 400 Error Code when facility not found
POST	name, number, postalCode, street, uuidAtCP, city, country	JSON containing *id, name, number, postalCode, street, uuidAtCP, city, country, createdDate*, which can be an organization created at the time or earlier
DELETE	id or name	HTTP 200 Response Code with no data or HTTP 400 Error Code when facility not found

Table 4.3: Method allowed in Facility webservice, with its required parameters.

4.5.5 MANAGE ORGANIZATIONS

The Organizations service lets a User add or modify an Organization as long as the user have authorisation to perform the operation.

- **Endpoint** {server}/multi_archive/organizations
- **Entities** Organization
- **Description** This endpoint allows the CRUD operations related to the entity Organization.

Method	Parameters	Response
GET	id or name or no parameters	JSON containing *id, name, createdDate* or HTTP 400 Error Code when organization not found
POST	name	JSON containing *id, name, createdDate*, which can be an organization created at the time or earlier
DELETE	id or name	HTTP 200 Response Code with no data or HTTP 400 Error Code when organization not found

Table 4.4: Method allowed in Organization webservice, with its required parameters.

4.5.6 MANAGE OPERATIONS

The manage operations service lets authorised Users to perform CRUD (Create, Read, Update and Delete) operations on the entity Operation.

- **Endpoint** {server}/multi_archive/operations
- **Entities** Operation
- **Description** This endpoint allows the CRUD operations related to the entity Operation.

Method	Parameters	Response
GET	id or name or no parameters	JSON containing *id, name, createdDate* or HTTP 400 Error Code when operation not found
POST	name	JSON containing *id, name, createdDate*, which can be an operation created at the time or earlier
DELETE	id or name	HTTP 200 Response Code with no data or HTTP 400 Error Code when operation not found

Table 4.5: Method allowed in Operation webservice, with its required parameters.

4.5.7 MANAGE CATEGORIES

This service allows the creation, edition and removal of categories of "resources" on the system. However, if the user does not have permissions to perform operations in this service, the service will not execute the request.

- **Endpoint** {server}/multi_archive/categories
- **Entities** Category
- **Description** This endpoint allows the CRUD operations related to the entity Category.

Method	Parameters	Response
GET	id or name or no parameters	JSON containing *id, name, createdDate* or HTTP 400 Error Code when category not found
POST	name	JSON containing *id, name, createdDate*, which can be an category created at the time or earlier
DELETE	id or name	HTTP 200 Response Code with no data or HTTP 400 Error Code when category not found

Table 4.6: Method allowed in Category webservice, with its required parameters.

4.5.8 MANAGE PERMISSIONS

This service allows the creation, modification and removal of permissions. This service has a particularity in relation to the previous ones: the creation of a new permission, obligates to have previously created a operation and a category, as well as the existence of a Resource if the category is, precisely, Resource.

- **Endpoint** {server}/multi_archive/permissions
- **Entities** Permission
- **Description** This endpoint allows the CRUD operations related to the entity Permission.

Method	**Parameters**	**Response**
GET	resourceName and operationName and categoryName or id or no parameters	JSON containing *id, name, createdDate* or HTTP 400 Error Code when permission not found
POST	resourceName and operationName and categoryName or resourceId and operationId and categoryId	JSON containing *id, resource, category, operation, createdDate*, which can be an permission created at the time or earlier
DELETE	resourceName and operationName and categoryName or id	HTTP 200 Response Code with no data or HTTP 400 Error Code when permission not found

Table 4.7: Method allowed in Permission webservice, with its required parameters.

4.5.9 MANAGE ROLES

Similarly to the services already mentioned, the managing role Role allows the creation, editing and removal of objects from the Role entity. This service assumes the existence of permissions to be associated with Role, although it is not mandatory.

- **Endpoint** {server}/multi_archive/roles
- **Entities** Role
- **Description** This endpoint allows the CRUD operations related to the entity Role.

Method	Parameters	Response
GET	id or name or no parameters	JSON containing *id, name, createdDate* or HTTP 400 Error Code when role not found
POST	name	JSON containing *id, name, createdDate*, which can be an role created at the time or earlier
DELETE	id or name	HTTP 200 Response Code with no data or HTTP 400 Error Code when role not found

Table 4.8: Method allowed in Role webservice, with its required parameters.

4.5.10 APPEND/REMOVE USER-FACILITY

Append/Remove User-Facility service allows a user to join an facility. It assumes that both objects are already in the system. Permissions are also required to effect the request, for whatever the intended operation, in the User and Facility categories.

- **Endpoint** {server}/multi_archive/userToFacility
- **Entities** User, Facility
- **Description** This endpoint allows to append a User to a Facility or remove a User from a Facility.

Method	Parameters	Response
POST	idUser, idFacility	HTTP 200 Response Code with no data or HTTP 400 Error Code when some error occurred with the parameters provided
DELETE	idUser, idFacility	HTTP 200 Response Code with no data or HTTP 400 Error Code when some error occurred with the parameters provided

Table 4.9: Method allowed in UserToFacility webservice, with its required parameters.

4.5.11 APPEND/REMOVE USER-ORGANIZATION

Append/Remove User-Organization service allows a user to join an organization. It assumes that both objects are already in the system. Permissions are also required to effect the request, for whatever the intended operation, in the User and Organization categories.

- **Endpoint** {server}/multi_archive/userToOrganization

- **Entities** User, Organization
- **Description** This endpoint allows to append a User to a Facility or remove a User from a Facility.

Method	Parameters	Response
POST	idUser, idOrganization	HTTP 200 Response Code with no data or HTTP 400 Error Code when some error occurred with the parameters provided
DELETE	idUser, idOrganization	HTTP 200 Response Code with no data or HTTP 400 Error Code when some error occurred with the parameters provided

Table 4.10: Method allowed in UserToOrganization webservice, with its required parameters.

4.5.12 APPEND/REMOVE PERMISSION-ROLE

Append/Remove Permission-Role service allows to make a connection between a Role and a Permission. It assumes that both objects are already in the system. Permissions are also required to effect the request, either the operation is creation or removal, in the Permission and Role categories.

- **Endpoint** {server}/multi_archive/permissionToRole
- **Entities** Permission, Role
- **Description** This endpoint allows to append a Permission to a Role or remove a Permission from a Role.

Method	Parameters	Response
GET	idRole	JSON containing the list of permissions of the role with the id idRole or HTTP 400 Error Code when invalid or not found Role
POST	idRole, idPermission	HTTP 200 Response Code with no data or HTTP 400 Error Code when some error occurred with the parameters provided
DELETE	idRole, idPermission	HTTP 200 Response Code with no data or HTTP 400 Error Code when some error occurred with the parameters provided

Table 4.11: Method allowed in PermissionToRole webservice, with its required parameters.

4.5.13 APPEND/REMOVE FACILITY-ORGANIZATION

Append/Remove Facility-Organization service allows to assign a Facility to an Organization or vice-versa. It assumes that both objects Facility and Organization are already in the database. Permissions are also required to effect the request, either the operation is creation or removal, in the Facility and Organization categories.

- **Endpoint** {server}/multi_archive/facilityToOrganization
- **Entities** Facility, Organization
- **Description** This endpoint allows to append a Facility to a Organization or remove a Facility from a Organization.

Method	**Parameters**	**Response**
GET	idOrganization	JSON containing the list of facilities belonging to the Organization with the id idOrganization or HTTP 400 Error Code when invalid idOrganization or not found Organization
POST	idFacility, idOrganization	HTTP 200 Response Code with no data or HTTP 400 Error Code when some error occurred with the parameters provided
DELETE	idFacility, idOrganization	HTTP 200 Response Code with no data or HTTP 400 Error Code when some error occurred with the parameters provided

Table 4.12: Method allowed in FacilityToOrganization webservice, with its required parameters.

4.5.14 APPEND/REMOVE CATEGORY-PERMISSION

Append/Remove Permission-Category service makes available methods to remove or create associations of Permission to/from Category. It assumes that both objects are already in the system. Permissions are also required to effect the request, either the operation is creation or removal, in the Permission and Category categories.

- **Endpoint** {server}/multi_archive/categoryToPermission
- **Entities** Category, Permission
- **Description** This endpoint allows to append a Category to a Permission or remove a Category from a Permission.

Method	Parameters	Response
GET	idPermission	JSON containing the Category of the given Permission with id idPermission or HTTP 400 Error Code when invalid idPermission or not found Permission
POST	idPermission, idCategory	HTTP 200 Response Code with no data or HTTP 400 Error Code when some error occurred with the parameters provided
DELETE	idPermission, idCategory	HTTP 200 Response Code with no data or HTTP 400 Error Code when some error occurred with the parameters provided

Table 4.13: Method allowed in CategoryToPermission webservice, with its required parameters.

4.5.15 APPEND/REMOVE OPERATION-PERMISSION

Append/Remove Permission-Operation service makes available methods to remove or create associations of Permission to/from Operation. It assumes that both objects are already in the system. Permissions are also required to effect the request, either the operation is creation or removal, in the Permission and Operation categories.

- **Endpoint** {server}/multi_archive/operationToPermission
- **Entities** Operation, Permission
- **Description** This endpoint allows to append a Operation to a Permission or remove a Operation from a Permission.

Method	Parameters	Response
GET	idPermission	JSON containing the Operation of the given Permission with id idPermission or HTTP 400 Error Code when invalid idPermission or not found Permission
POST	idPermission, idOperation	HTTP 200 Response Code with no data or HTTP 400 Error Code when some error occurred with the parameters provided
DELETE	idPermission, idOperation	HTTP 200 Response Code with no data or HTTP 400 Error Code when some error occurred with the parameters provided

Table 4.14: Method allowed in OperationToPermission webservice, with its required parameters.

4.5.16 SHARE

Share endpoint gives the user the ability to give their permissions to third users. Those users are already on the system and do not need to have special permissions to receive the shared permissions. In addition, the requester should submit the user ID to which the resource should be shared. Likewise, the resource ID must be in the HTTP request as well.

Given the assumptions, it is assumed that both objects are already in the system. Permissions are also required to effect the request, either the operation is creation or removal, in the Permission and Operation categories.

- **Endpoint** {server}/multi_archive/share
- **Entities** Resource, User, Permission
- **Description** This endpoint allows to append a Operation to a Permission or remove a Operation from a Permission.

Method	**Parameters**	**Response**
POST	idUser, idResource	HTTP 200 Response Code with no data or HTTP 400 Error Code when some error occurred with the parameters provided
DELETE	idPermission, idUser, idResource	HTTP 200 Response Code with no data or HTTP 400 Error Code when some error occurred with the parameters provided

Table 4.15: Method allowed in Sharing webservice, with its required parameters.

4.6 DICOMWEB

In order to protect the access to DICOMWeb standard services, it was applied the same concept as the one previously mentioned. On each DICOMWeb service, it is extracted the Authorization token that is part of the HTTP header. Next, the authorisation token is passed to the database manager library that contains methods to evaluate the authorisation clearance of the user.

4.6.1 WADO-RS

In the WADO-RS service, similarly to the previously described services, the Dicoogle session token is obtained through the HTTP Header "Authorization".

Subsequently, using the dbmanger library, the authorisation check is performed for the user in question. The permission requirement in WADO-RS is the GET operation of the "Resource" category. This verification is done using the abstraction provided by the UserManger class, a class designed to simplify the verification of authorisations and permissions.

If the user does not have permission from Resource GET or does not have permission to access the requested resource, a response will be sent with the HTTP 403 Forbidden error code. On the contrary, if the user is authorised and has all the necessary permissions, the whole process will proceed normally as a WADO-RS service following the DICOM Standard should proceed.

The figure 4.7 represents a sequence diagram demonstrating the workflow of an HTTP request to the WADO-RS plugin. In the points below, the steps of this workflow are deepened. Note that it will be assumed that the session token sent in the request is valid and the corresponding user has permissions to perform the requested operations.

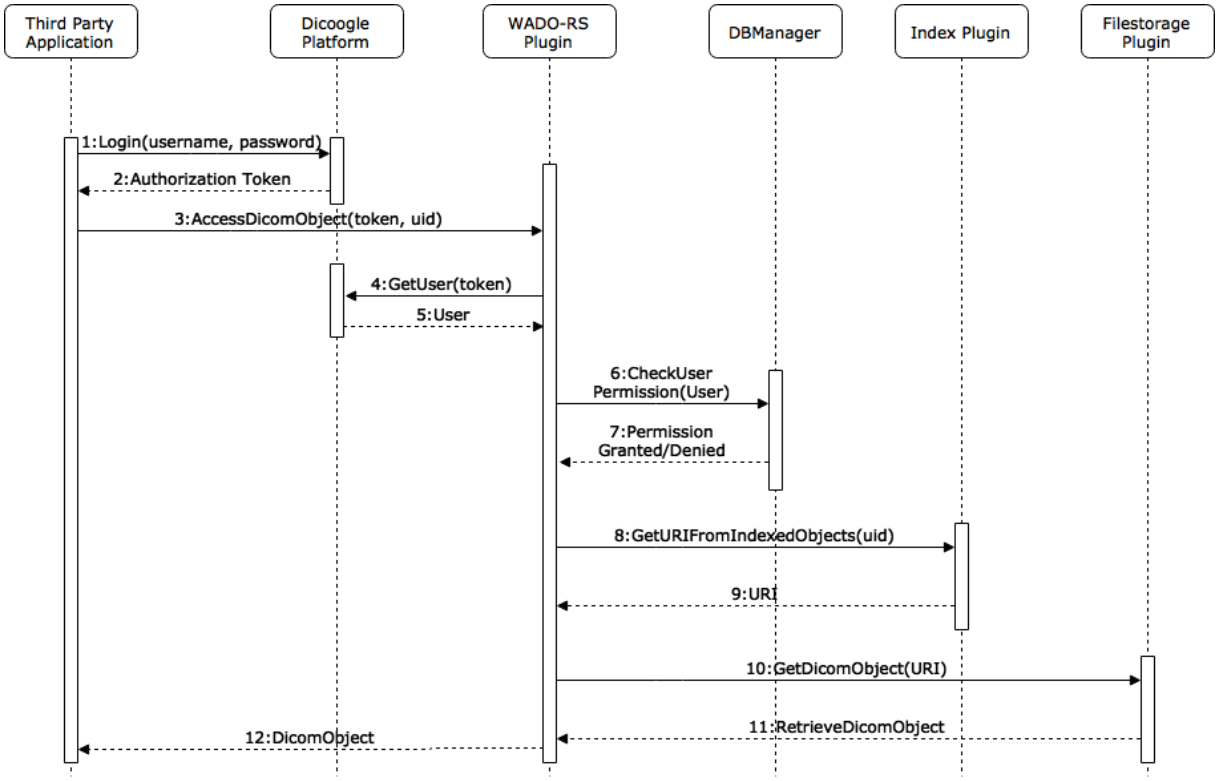

Figure 4.7: Sequence diagram when accessing WADO from a third party viewer.

1. **Login** - Login via HTTP REST service host/multi_archive/login described in section 4.5
2. **AuthorizationToken** - After phase 1, when successful, the session token that will be used to gain access in the following phases is returned
3. **AccessDicomObject** - After phases 1 and 2, all requirements to perform an order to the WADO-RS service are completed. At this stage, an HTTP request must be made with the POST method where the HTTP "Authorization" header with the session token obtained in the previous steps must be included. If the token is not sent, the user/client application will get an error code from the WADO-RS service. In addition to this requirement, the request must also include the UID of the intended object
4. **GetUser** - After receiving the Authorization Token, it is made verification of compliance with the cache of users logged into the system
5. **User** - If the token matches one of the users currently connected to the system, information about that user will be returned

6. **CheckUserPermission** - Once the user has been obtained, it is checked if the user is authorized and allowed to perform resource creation operations (necessary permission to access the WADO service)
7. **Permission Granted / Denied** - The permission to continue to carry out the operation is granted or denied (we will assume granted)
8. **GetURIFromIndexedObjects** - To get the file, it is required to know its location. This procedure is done by querying the index plugin about the URI where the file is located
9. **URI** - If the DICOM object exists, the URI corresponding to its location is returned
10. **GetDicomObject** - The filestorage plugin is requested from the file present in the URI returned in the previous phase
11. **RetrieveDicomObject** - DICOM file is returned, in case that URI is valid
12. **DicomObject** - File is sent to client over HTTP

4.6.2 STOW-RS

The STOW-RS service, in addition to the usual Dicoogle authorisation token verification, has undergone other changes.

Based on the change in the Authorization for access to the service, similarly to the service previously mentioned (WADO), it is used the session token for obtaining the user. For the STOW-RS service, the Add permission in the Resource category is required.

Using the ResourceManager abstraction level, the Resource is created in the database.

This feature will be created according to the model level of the DICOM standard contained in the STOW-RS plugin settings. For example, if the DICOM standard model level is by default Study, each time a DICOM file is stored, a resource will be created whose default model level will be Study and the ID will be the StudyInstanceUID. In a final phase, the facilities to which the user is associated are obtained and, to each of them, the new Resource is associated, for the purposes of access or listing permissions.

The workflow of the DICOM STOW-RS service plugin with the active multi-file option, where it is assumed that the access permission is guaranteed, is demonstrated in the sequence diagram shown in figure 4.8. Below the steps are deepened, case by case:

1. **Login** - Login via HTTP service Rest host/multi_archive/login described in section 4.5
2. **AuthorizationToken** - After phase 1, when successful, the session token that will be used to gain access in the following phases is returned
3. **StoreDicomObject** - After the token has been obtained, it is already possible to request the operations to the STOW-RS plugin. Through an HTTP REST request with the POST method in the host/ext/stow service where the HTTP "Authorization" header with the session token obtained in phase 1 and 2 must appear, the DICOM file is sent to the store operation on the Dicoogle platform

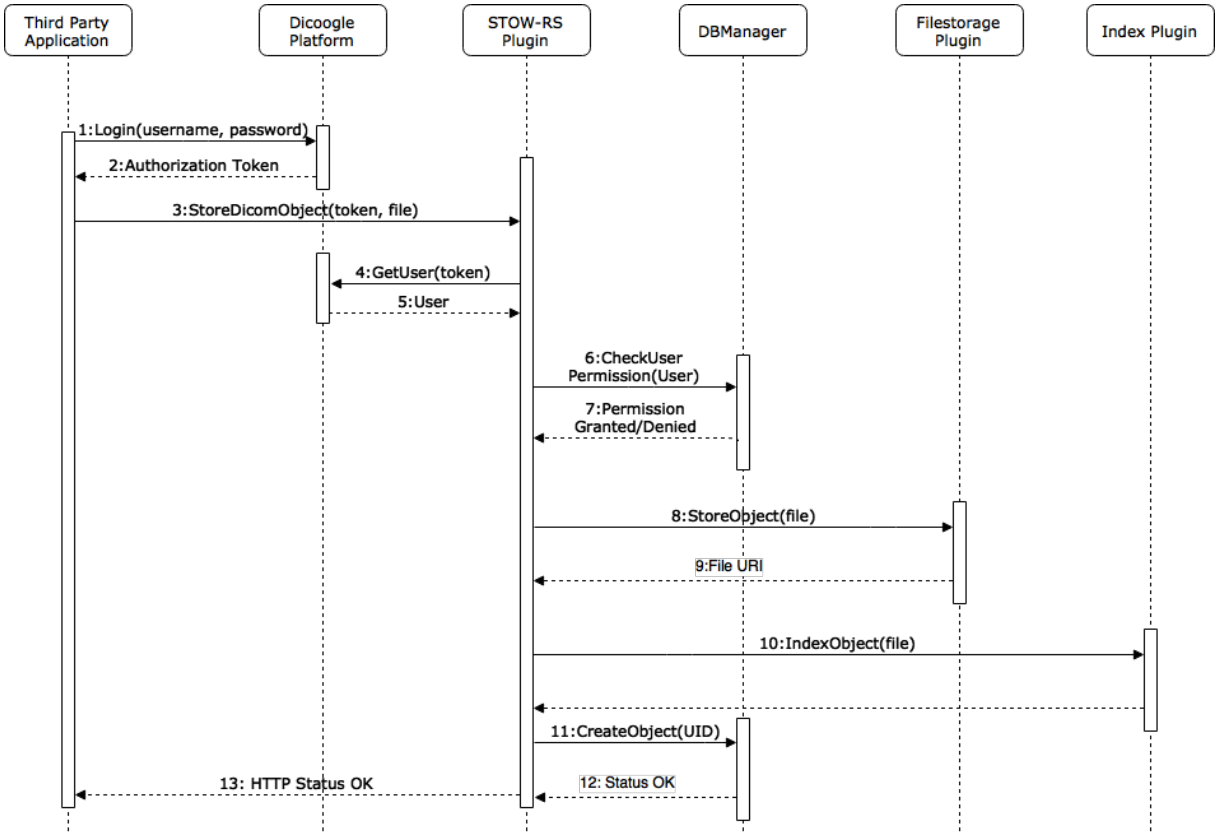

Figure 4.8: Sequence diagram storing file STOW-RS from a third party application.

4. **GetUser** - After receiving the Authorization Token, it is made verification of compliance with the cache of users logged into the system
5. **User** - If the token matches one of the users currently connected to the system, information about that user will be returned
6. **CheckUserPermission** - Once the user has been obtained, it is checked if the user is authorized and allowed to perform resource creation operations (necessary permission to access the STOW-RS service)
7. **Permission Granted / Denied** - The permission to continue to carry out the operation is granted or denied (we will assume granted)
8. **StoreObject** - After the permissions are verified, the DICOM file is sent to storage, from where a URI is returned
9. **URI** - The URI corresponding to the location of the DICOM file is returned after storage operation
10. **IndexObject** - All relevant attributes of the DICOM object are indexed. In the case of the present thesis, in a MySql Database Management System (DBMS)
11. **CreateObject** - Once indexed, it is necessary to create a Resource entry in the resource access control system, with the resource identification (UID), with the category (Resource) and the corresponding DIM level (by definition, Study, being possible that would be Patient, Series or Instance)

12. **Status** - Return of the created Resource object, if it was created successfully or already exists in the system
13. **HTTP Status** - the error code is returned to the application as soon as there is an error or the resource is successfully added to the DBMS and there is a Dicoogle platform

4.6.3 QIDO-RS

The development of authentication and filtering in QIDO-RS has become cumbersome. One more time, REST requests must contain the HTTP "Authentication" Header. Subsequently, the Dicoogle core search method is invoked, which sends the query request to the plugins. That query contains the session token.

The query submitted is processed by a particular plugin, sql-dim. In this plugin, it is done the Authorization and permission check of the List operation for the Resource category on the specified user. After verification, one of the following two options may occur: the user does not have the authorisation to perform queries on the system and an empty result list is returned; or, on the other hand, the user has permissions and the execution proceeds.

In the latter case, in which the execution proceeds, the SQL query is performed. The **sql!** (**sql!**) statement is the result of the transformation performed on the search parameters. The query filters the results to only authorised objects appear in result list. Only features that meet one of the following requirements will be returned:

1. the resource belongs to one of the Permissions of one of the User Roles
2. the resource belongs to one of the Facilities to which the User is associated
3. the resource belongs to one of the Permissions shared with the User

Below, in Figure 4.9, the sequence diagram representing the workflow of the query operation whose request is made via HTTP to the QIDO-RS plugin is shown, assuming that the user who performs the query holds the permissions required to the service:

1. **Login** - Login via HTTP service Rest host/multi_archive/login described in section 4.5
2. **AuthorizationToken** - After phase 1, when successful, the session token that will be used to gain access in the following phases is returned
3. **QueryForDicomObjects** - Once the token is obtained, it can be included in the HTTP REST request, in the HTTP "Authorization" header of the POST method. The inclusion of the token is critical for results to be returned: if the token is not sent or otherwise invalid, the query is not executed and an empty result set is returned. In addition to the token and the query itself, it is necessary to send the expected attributes of the objects found in the query. The endpoint to which the request is to be sent is host/multi_archive/qido
4. **Search** - After validation, the QIDO-RS plugin delegates the query to all query plugins. Among them, there is SQL-DIM (see section 4.7). This plugin is prepared for the multi-file paradigm and will return only DICOM objects that are contained in one of the user's permission sets that requests the QIDO-RS plugin

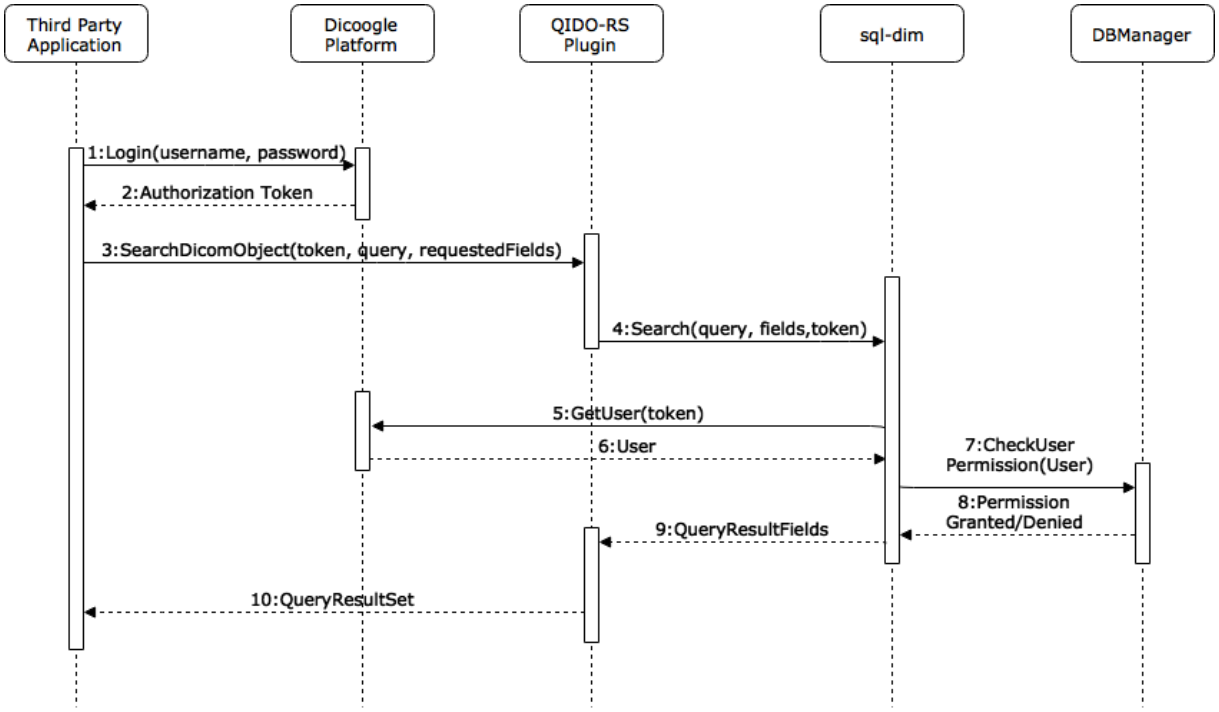

Figure 4.9: Sequence diagram when querying QIDO-RS from a third party viewer.

5. **GetUser** - After receiving the Authorization Token, it is made verification of compliance with the cache of users logged into the system
6. **User** - If the token matches one of the users currently connected to the system, information about that user will be returned
7. **CheckUserPermission** - Once the user has been obtained, it is checked if the user is authorized and allowed to perform resource creation operations (necessary permission to access the QIDO-RS service)
8. **Permission Granted / Denied** - The permission to continue to carry out the operation is granted or denied (we will assume granted)
9. **QueryResultFields** - All results already filtered against the user are returned to the QIDO-RS plugin
10. **QueryResultSet** - QIDO-RS sends the HTTP response in JSON format with the result of the query. The result is represented in a list of JSON objects corresponding to each result returned in the previous point

4.7 SQL-DIM

Dicoogle's architecture is based on the development of index and filestorage plugins so that basic DICOM index and retrieval service operations are ensured. Besides these types, there is also a third one relevant for the well funtioning of Dicoogle platform, there is the query plugins.

Usually, the plugin that supports the index operations is also responsible for querying. On the main page of the public repository of the Dicoogle project [64], an index/query plugin, the Lucene plugin, is available. Lucene plugin is a plugin based on Apache Lucene Library. It supports index and query on DICOM meta-data.

However, in the development of this thesis, it was used another index/query plugin, the sql-dim. Instead of Apache Lucene library, Sql-dim uses the JPA framework. So, sql-dim allows the index and querying of DICOM files in a MySql engine.

The use of this plugin over Lucene plugin allows integration with MySql and unification of the access control system. This integration is noticed in the query function of this plugin. The SQL query developed for searching the indexed attributes is not limited to the search parameters search. In order to protect resources, the agent performing the query operation must provide its session token for identification. Resource filtering is delegated to the database management system via SQL query.

The query developed is a query that, in addition to searching the required resource and returning the requested parameters, combines the list of results with other constraints. These constraints focus on filtering the results list. Only results with one of the following conditions will be returned, in which the resource(s):

1. appears in one of the permissions of one of the roles of the requesting user
2. belongs to one of the facilities to which the user is associated
3. is one of the resources shared with the user

CHAPTER 5

Results and Discussion

The Results and Discussion chapter presents the work done and its validation. The test methodology is presented, the results are presented and discussed. In addition, it is demonstrated the use of the services described in the previous chapter.

5.1 RESULTS

The addition of an user management mechanism introduces temporal delay on the execution supported. This impact arises from the need to verify, for each of DICOMWeb services, the entity making the request and, in addition, verify its authorization and permissions for the service in question.

In order to evaluate the performance impact, several automatic tests were performed. These tests simulate the massive usage of QIDO-RS, STOW-RS and WADO services. For a most accurate analysis, all benchmarks on performance impact (DICOMWeb services over Dicoogle vs. DICOMWeb services over Dicoogle with user management support) must have the same execution conditions as for example, the amount of RAM, CPU processing speed or even the processor load.

After the tests have been carried out, a processing and analysis of the data obtained will be presented, as well as a results representation in the form of graphs.

Finally, the results will be discussed and interpreted as much as possible.

5.2 TEST ENVIRONMENT

The tests were carried out in an instance of macOS Sierra 10.11.5 operating system which specifications are presented in Table 5.1

CPU	Intel Core i5 2.7GHz with 3MB shared L3 cache
RAM	8 GB 1867 MHz DDR3
ROM	SSD 128 GB PCIe-based

Table 5.1: Equipment specifications

To ensure maximum testing equality, time measurements were performed under the following conditions:

- Same software version including operating system or even DBMS (MySQL 5.7.17)
- Same amount of RAM
- Same DICOM objects dataset
- Same system load and number of active applications and services
- Similar test replications, the only change factor being the authorization and provision of an access token

5.3 TEST METHODOLOGY

Two types of tests were developed on various components of the system. These two types are divided into load tests and unit/validation tests.

With regard to unit tests or validation tests, 3 tests were written. These three tests aim to validate the DICOMWeb services (QIDO-RS, STOW-RS and WADO-RS). In the case of the QIDO-RS tests, queries were constructed at each level of DICOM information (DIM): Patient, Study, Series and Instance, with search parameters such as Modality, SOPInstanceUID, SeriesInstanceUID, StudyInstanceUID or PatientID. The queries were repeated 400 times.

Attempting to STOW-RS plugin tests, it used 5694 DICOM files (instances) with the default level of resources creation of "Study". The tests were performed 5 times in order to obtain the most reliable results possible. The distribution of files per size is described in table 5.2

Number of files	Size (Kilobytes)
2224	131
1120	290
960	163
417	132
368	514
352	394
156	515
78	130
2	7360
2	14340
2	17106
2	8947
2	15449
2	7357
1	14339
1	14693
1	14341
1	7366
1	6054
1	7358
1	16671

Table 5.2: File size of each DICOM file

Finally, to test the impact of adding the user management mechanism to the WADO-RS plugin, a list of 900 instances (SOPInstanceUID) was obtained. With these 900 identifiers, requests were made to the WADO-RS service with the request of the first frame of the object. All 900 instances were in the list of permissions associated with the test user. This set of tests was repeated 4 times.

For the entire set of tests, the Unirest library was used. This library allows the invocation of REST services. The temporal duration of service,from the time of the creation of the request until the answer to that same request, was recorded. Values such as Status code of the response, name of the file to be stored (STOW-RS), query (QIDO-RS), SOPInstanceUID (WADO-RS), wait time and response body were saved in a logfile.

Note that the tests were done in the system without authentication and in the system with authentication. These tests were identical, only with one change, in the case of the system with authentication: the addition of the session token in each request made. Before the service tests starts, a request was made to the login service in order to obtain the service access token.

Finally, a set of tests were performed to evaluate the solution scalability. There were 7 tests in which all of them ended up reaching the goal of 20000 requests. In each of the following cases, there was a hatch rate. This is the rate that the Locust.io spawns a virtual user. For instance, in the first case, a user was spawned every second until a total of 1000 users. The test ended up when the users performed 20000 requests in total.

- 1000 users with 1 hatch rate
- 10000 users with 1 hatch rate
- 50000 users with 1 hatch rate
- 1000 users with 10 hatch rate
- 10000 users with 10 hatch rate
- 50000 users with a rate of 10 hatches/s
- 100000 users with a rate of 50 hatches/s and a total of 50.000 orders

These load tests were performed taking advantage of the Locust.io tool. This tool allows the creation of tasks in which it is possible to test the endpoints. The endpoints and operations tested were:

- GET /multi_archive/categories
- GET /multi_archive/facilities
- POST /multi_archive/facilities
- GET /multi_archive/facilities?id=1124
- DELETE /multi_archive/facilityToOrganization
- POST /multi_archive/facilityToOrganization
- POST /multi_archive/login
- GET /multi_archive/operations
- GET /multi_archive/organizations
- POST /multi_archive/organizations
- GET /multi_archive/permissions
- GET /multi_archive/roles
- GET /multi_archive/users
- GET qido-rs series

5.4 TEST RESULTS

5.4.1 STOW-RS

In this section is presented the comparative result related to the STOW-RS plugin. About 22776 time measurements were taken from a set of 5694 different files. The table and graph below show average values:

-	**Without protection Mechanism**	**With protection Mechanism**
Average (ms)	51	483

Table 5.3: Average values of time measured of 22776 files storage requests

As we can see in the table, the storage over web operation grows 10 times comparatively between the original STOW-RS plugin and the STOW-RS plugin with the authentication mechanism developed. Despite the relatively high value (500 ms), an equally large temporal increase was expected.

In this plugin there are considerably more operations to the database than in the previous system without multi-archive support, enumerating:

1. authorization check
2. permission check
3. storage operation (common between solutions)
4. resource creation
5. association of resource to set of user facilities or, in case the user does not belong to any facility
6. creating access permissions on the part of the user in particular to the created resource
7. operations of creation of permissions
8. user query when verifying the access token

However, even considering the factors listed above, it exists a considerable difference in this operation and, in a future work, an algorithmic revaluation must be done in order to reduce the time of each operation. However, the dataset used for testing consisted essentially of files of small size (85% of files had less than 290 kB and 99% less than 515 KB). This factor contributes to the fact that the discrepancy between performances is so different. In an actual environment of use, since DICOM objects will be larger, this discrepancy should be diluted. I.e., performance values will increase in the storage operation without access control mechanism, which will make the increase of time due to this mechanism no longer so relevant.

5.4.2 QIDO-RS

Turning now to the case of the QIDO-RS plugin, Table 5.5 shows the proposed solution performance versus the original implementation. In this case, 26 queries were developed that were executed a total of 500 times, on each ou the four DIM levels, resulting in a total of 52000 queries to the QIDO-RS service, and the averages presented:

-	**Without protection Mechanism**	**With protection Mechanism**
Average (ms)	21	109
Average w/ response (ms)	-	97
Average wo/response (ms)	-	128

Table 5.4: Average values of time measured of 52000 query requests.

Particularly in the test of this service, in the analysis of the data collected during the tests, it was decided to present, in the case of the solution with the permissions management mechanism, both the global average and the average referring to the cases in which the server returned empty responses in contrast to the media in which the server returns composite responses with results. In the cases where there was no response, it was due to lack of user access permissions.

As expected, an empty response due to lack of permissions takes less time to get, since less query is required to the database. The addition of 29 ms is due to the need to execute another query to get the resources requested in the DBMS.

In relative terms between the system without multi-file support and with multi-file support, the temporal execution time is respectively 21 ms and 109 ms. Thus, there is an increase of approximately 5 times the original value, an acceptable value for the operations in question.

5.4.3 WADO-RS

In order to test the remaining service, the WADO-RS, a set of 988 SOPInstanceUIDs was obtained and, for each of them, it was invoked the REST service. All instances belong to studies whose user has authorization of access.

Four replicates were performed, totaling around 3930 time measurements. The averages are shown below:

-	**Without protection Mechanism**	**With protection Mechanism**
Average (ms)	42	311

Table 5.5: Average values of time measured of 27800 files storage requests

As in previous cases, there was an increase in the execution time of requests for web access to DICOM objects. In this case, the increase corresponds to approximately 7 times the original value.

Again, the explanation for this event is in the need to verify the authorization and permissions relating to a particular operation over a category or access to a resource.

These operations include the verification of the existence of permissions in one of the user's roles, the user's shared permissions list, or the resources's membership in one of the facilities where the user is associated. All these operations are intensive in requests of access to the database, reason why the relative temporal increase is expected.

In the case where the access times are smaller, the difference becomes more noticeable, except in cases where more queries/writing are required in the database.

5.4.4 SCALABILITY

Several load tests have been performed in order to validate server behavior and its responsiveness when there is a high affluence.

The tests performed using the Locust.io tool, in which it is possible to test endpoints and obtain statistics on the operations performed. It allows individual testing of a set of arbitrary endpoints. After starting execution indicating the host on which the tests will fall, the test phase can be started. The tool simulates users with actual requests. In the test presented below in the 5.6 table, a total of 20.000 orders were made. These requests were made by a set of 100.000 virtual users at a rate of 50 user spawns per second. Requests are parallelized at the time of execution.

Name	# reqs	# fails	Avg (ms)	Min (ms)	Max (ms)	Median (ms)	req/s
GET /categories	5162	0	1232	4	95403	96	3
GET /facilities	1668	0	1354	75	92591	200	1
POST /facilities	1100	0	1316	8	49920	100	0.2
GET /facilities ?id=1124	4836	0	1256	4	95664	88	2.5
DELETE /facilityToOrganization	3436	0	1161	9	83023	99	2.1
POST /facilityToOrganization	3000	0	1256	9	60154	99	1.1
POST /login	698	0	9	3	197	6	0.1
GET /operations	5714	0	1271	4	101390	90	2.4
GET /organizations	3766	0	1386	73	101395	170	2.2
POST /organizations	1552	0	1633	7	93983	110	0.7
GET /permissions	4648	0	1422	90	92518	190	2.7
GET /roles	4441	0	1287	72	95675	180	2.4
GET /users	3943	0	1154	72	95018	170	2.1
GET qido-rs /series	6036	0	2333	5	92390	310	3.5
Total	20000	0	1291				26

Table 5.6: Scrutiny of requests made to REST webservices performed by Locust.io

As is perceptible in table 5.6, there was no failures in the 20.000 orders, meaning that all orders were completed with valid response.

On average, a total of 26 requests were made per second, which translates into an average response time of around 1291 ms, with the QIDO-RS service being the slowest, with a mean of 2333 ms. However, this service was also the most requested with 6036 requests, which may have resulted in an overload. It should be noted that both the minimum and the median response times are far below the average (between 3 and 90 ms and between 6 and 200 ms, respectively). It is possible to conclude that the

average response time was very influenced by the requests that represent the maximum value obtained from the response time.

For each service, the percentage of requests completed in the time presented is displayed. Taking the case of GET multi_archive/users, 100% of requests were answered in less than 95018 ms. However, 99% of total requests were answered at 16000 ms, which is a considerable drop in values. With the help of table 5.7 we verify that 50% of requests were answered in less than 170 m.

Name	# reqs	50%	66%	75%	80%	90%	95%	98%	99%	100%
GET /categories	5162	96	230	440	700	2400	6800	13000	20000	95403
GET /facilities	1668	200	350	540	800	2700	6700	13000	16000	92591
POST /facilities	1100	100	200	380	650	3300	9200	14000	21000	49920
GET /facilities?id	4836	88	220	430	650	2400	6900	13000	22000	95664
DELETE /facilityToOrganization	3436	99	240	460	700	2500	6700	12000	18000	83023
POST /facilityToOrganization	3000	99	230	410	670	2700	7400	14000	21000	60154
POST /login	698	6	8	9	11	18	33	52	69	197
GET /operations	5714	90	230	430	670	2600	7400	13000	20000	101390
GET /organizations	3766	170	290	470	710	2400	7400	15000	23000	101395
POST /organizations	1552	110	300	610	1000	5400	9700	15000	18000	93983
GET /permissions	4648	190	330	540	810	2900	7400	14000	23000	92518
GET /roles	4441	180	310	540	790	2700	6800	13000	20000	95675
GET /users	3943	170	290	470	700	2200	5900	12000	16000	95018
GET qido-rs /series	6036	310	1100	2300	3300	7200	12000	18000	24000	92390

Table 5.7: Percentage values of requests completed in the given time

CHAPTER 6

Conclusions and Future Work

6.1 conclusion

This dissertation main objective was to study and develop a user accounting mechanism for a DICOM based PACS archive. The idea was to support a new concept of multi-archive medical imaging repository that aggregates and manages several instances, resources, users and permissions. Therefore, the work began by studying and analysing the existing access control mechanisms and permissions management systems, in order to understand how state of the art solutions could be adapted to solve the challenge. It was concluded that no existing solution satisfied the project requirements and a new system was designed, implemented and validates in the scope of this dissertation.

It is possible to identify several contributions of the work described in this document. First, an accounting mechanisms for medical imaging repositories; Secondly, the integration and evolution of an open source platform, i.e. the Dicoogle PACS; Thirdly, the development of a Web API of services for management of proposed architecture and transparent integration with third applications; Finally, developed services were integrated with most recent DICOMWeb standard, supporting the STOW-RS, QIDO-RS and WADO services.

The proposed architecture was validated through exhaustive tests and the results show that temporal overhead introduced by services proposed is acceptable in a real-world environment.

6.2 future work

Although the contributions of this dissertation work are significant, there is still room for improvements and implementation of new functionalities. Next, we will present two future work paths assuming the existence of a multi-archive solution:

- The implementation of a multi-user medical imaging archive integrated with personal cloud storage services like, for instance, Dropbox, Google Drive or others;
- A web portal for end-user management of proposed architecture, consuming the services available through Rest API.

CHAPTER 7

ATTACHMENTS

7.1 USER GUIDE

In this section a user guide will be presented, which will not only show the creation from the zero of the database system, creating users and managing permissions, but will also show the use of available DICOMWeb services (WADO, QIDO-RS and STOW-RS).

The user and permissions management platform has a default root user so that the first instance management would be possible. Otherwise, all management attempts would be blocked on filters by lack of authorization.

The following steps represent a workflow of operations in order to have permissions to perform DICOMWeb requests.

1. Login
   By invoking this step you can get the session token required for the next steps. We will use username: dicoogle, password: dicoogle (root account)

```
curl --request POST --url http://localhost:8080/multi_archive/login
    --header "content-type: application/x-www-form-urlencoded"
    --data "username=dicoogle&password=dicoogle"
```

Listing 11: Login webservice call

```
{"status":"success","token":"31737d53-0ea8-4507-8ccc-899d037750e9"}
```

Listing 12: Login webservice response

2. Create Organization
   The following request allows the creation of an Organization with the name "Demo Organization".

```
curl --request POST --url
    http://localhost:8080/multi_archive/organizations --header
    "authorization: 31737d53-0ea8-4507-8ccc-899d037750e9" --header
    "content-type: application/x-www-form-urlencoded" --data
    name=Demo\%20Organization
```

Listing 13: Create Organization webservice call

```
{"id":1105,"createdDate":"Thu Jun 15 01:48:37 WEST
    2017","name":"Demo Organization"}
```

Listing 14: Create Organization webservice response

3. Create Facility
   In this procedure it is possible to create a Facility with the following attributes:
   - Name: Demo Facility
   - Number: 0
   - PostalCode: 0000-000
   - Street: Demo Street
   - uuidAtCP: 0000
   - City: Demo City
   - Country: Demo Country

```
curl --request POST --url
    http://localhost:8080/multi_archive/facilities --header
    "authorization: 31737d53-0ea8-4507-8ccc-899d037750e9" --header
    "content-type: application/x-www-form-urlencoded" --data
    "name=Demo\%20Facility&
    number=0&postalCode=0000-000&street=Demo\%20Street&
    uuidAtCP=00000000&city=Demo\%20City&country=Demo\%20Country"
```

Listing 15: Create Facility webservice call

```
{"id":1106,"createdDate":"2017-06-15 01:55:33.0","name":"Demo
    Facility","number":"0","postalCode":"0000-000","street":"Demo
    Street","uuid":"00000000","city":"Demo City","country":"Demo
    Country"}
```

Listing 16: Create Facility webservice response

4. Append Facility – Organization
   After executing the following request the “Demo Facility” will have “Demo Organization” has its main organization.

```
curl --request POST --url
    http://localhost:8080/multi_archive/facilityToOrganization
    --header "content-type: application/x-www-form-urlencoded"
    --data "idOrganization=1105&idFacility=1106" --header
    "authorization: 31737d53-0ea8-4507-8ccc-899d037750e9"
```

Listing 17: Append Facility – Organization webservice call

5. Create User

   A User will be created after the following procedure with the following attributes:

   - firstName: DemoFirstName
   - lastName: DemoLastName
   - email: demo@email.com
   - title: mr
   - password: demopassword

```
curl --request POST --url http://localhost:8080/multi_archive/users
    --header "content-type: application/x-www-form-urlencoded"
    --header "authorization: 31737d53-0ea8-4507-8ccc-899d037750e9"
    --data "firstName=DemoFirstName&lastName=DemoLastName
    &email=demo\%40email.com&title=mr&password=demopassword"
```

Listing 18: Create User webservice call

```
{"id":1107,"createdDate":"2017-06-15 11:19:25.0",
    "firstName":"DemoFirstName","lastName":"DemoLastName",
    "title":"mr","email":"demo@email.com"}
```

Listing 19: Create User webservice response - A JSON containing all information about the recently created User object but not password.

6. Append User to Organization

   After this step, User created with ID 1107 will be associated to Organization with ID 1105.

```
curl --request POST --url
    http://localhost:8080/multi_archive/userToOrganization --header
    "content-type: application/x-www-form-urlencoded" --data
    "idUser=1107&idOrganization=1105" --header "authorization:
    16c92604-fc20-4f1e-a53e-6df4e2dbff57"
```

Listing 20: Command to append a user to an organization. Only error code 200 if it was successful.

7. Append User to Facility

   After this step, User created with ID 1107 will be associated to Facility with ID 1106.

```
curl --request DELETE --url
    http://localhost:8080/multi_archive/userToFacility --header
    "content-type: application/x-www-form-urlencoded" --data
    "idUser=1107&idFacility=1106" --header "authorization:
    96fe14a1-a5dc-4861-89b8-69b5c2a27e7d"
```

Listing 21: Command to append a user to an facility. Only error code 200 if it was successful.

8. Create Role
   In this step, we will create the first Role in the system. The role will be called DemoRole.

```
curl --request POST --url http://localhost:8080/multi_archive/roles
    --header "content-type: application/x-www-form-urlencoded"
    --data name=DemoRole --header "Authorization:
    96fe14a1-a5dc-4861-89b8-69b5c2a27e7d
```

Listing 22: Create Role webservice call

```
{"id":1108,"createdDate":"Thu Jun 15 23:34:03 WEST
    2017","name":"DemoRole"}
```

Listing 23: Create Role webservice response

9. Create Categories
   We need to create Categories. The main and required categories on the system are *Organization*, *Facility*, *Category*, *Operation*, *Permission*, *Role*, *User*, *Share* and *Resource*. Each one of the categories above will be created, in the same order.

```
curl --request POST --url
    http://localhost:8080/multi_archive/categories --header
    "content-type: application/x-www-form-urlencoded" --data
    name=Organization --header "Authorization:
    96fe14a1-a5dc-4861-89b8-69b5c2a27e7d"

{"id":8,"createdDate":"2017-04-08 19:42:23.0","name":"Organization"}

curl --request POST --url
    http://localhost:8080/multi_archive/categories --header
    "content-type: application/x-www-form-urlencoded" --data
    name=Facility --header "Authorization:
    96fe14a1-a5dc-4861-89b8-69b5c2a27e7d"

{"id":5,"createdDate":"2017-04-08 19:42:23.0","name":"Facility"}
```

```
curl --request POST --url
    http://localhost:8080/multi_archive/categories --header
    "content-type: application/x-www-form-urlencoded" --data
    name=Category --header "Authorization:
    96fe14a1-a5dc-4861-89b8-69b5c2a27e7d"

{"id":4,"createdDate":"2017-04-08 19:42:23.0","name":"Category"}

curl --request POST --url
    http://localhost:8080/multi_archive/categories --header
    "content-type: application/x-www-form-urlencoded" --data
    name=Operation --header "Authorization:
    96fe14a1-a5dc-4861-89b8-69b5c2a27e7d"

{"id":7,"createdDate":"2017-04-08 19:42:23.0","name":"Operation"}

curl --request POST --url
    http://localhost:8080/multi_archive/categories --header
    "content-type: application/x-www-form-urlencoded" --data
    name=Permission --header "Authorization:
    96fe14a1-a5dc-4861-89b8-69b5c2a27e7d"

{"id":9,"createdDate":"2017-04-08 19:42:24.0","name":"Permission"}

curl --request POST --url
    http://localhost:8080/multi_archive/categories --header
    "content-type: application/x-www-form-urlencoded" --data
    name=Role --header "Authorization:
    96fe14a1-a5dc-4861-89b8-69b5c2a27e7d"

{"id":11,"createdDate":"2017-04-08 19:42:24.0","name":"Role"}

curl --request POST --url
    http://localhost:8080/multi_archive/categories --header
    "content-type: application/x-www-form-urlencoded" --data
    name=User --header "Authorization:
    96fe14a1-a5dc-4861-89b8-69b5c2a27e7d"

{"id":13,"createdDate":"2017-04-08 19:42:24.0","name":"User"}
```

```
curl --request POST --url
    http://localhost:8080/multi_archive/categories --header
    "content-type: application/x-www-form-urlencoded" --data
    name=Share --header "Authorization:
    96fe14a1-a5dc-4861-89b8-69b5c2a27e7d"

{"id":12,"createdDate":"2017-04-08 19:42:24.0","name":"Share"}

curl --request POST --url
    http://localhost:8080/multi_archive/categories --header
    "content-type: application/x-www-form-urlencoded" --data
    name=Resource --header "Authorization:
    96fe14a1-a5dc-4861-89b8-69b5c2a27e7d"

{"id":10,"createdDate":"2017-04-08 19:42:24.0","name":"Resource"}
```

Listing 24: Create Categories webservice call. It is shown a command that invokes the web service and then the respective response (JSON)

10. Create Operations
    The are some fundamental Operations that are needed so the general users can perform the basic procedures, like adding a file or list a file. The basic operations that we will address are: *Add*, *Delete*, *Get*, *List* and *Update*.

```
curl --request POST --url
    http://localhost:8080/multi_archive/operations --header
    "content-type: application/x-www-form-urlencoded" --data
    name=Add --header "Authorization:
    96fe14a1-a5dc-4861-89b8-69b5c2a27e7d"

{"id":14,"createdDate":"2017-04-08 19:42:34.0","name":"Add"}

curl --request POST --url
    http://localhost:8080/multi_archive/operations --header
    "content-type: application/x-www-form-urlencoded" --data
    name=Delete --header "Authorization:
    96fe14a1-a5dc-4861-89b8-69b5c2a27e7d"

{"id":15,"createdDate":"2017-04-08 19:42:35.0","name":"Delete"}

curl --request POST --url
```

```
    http://localhost:8080/multi_archive/operations --header
    "content-type: application/x-www-form-urlencoded" --data
    name=Get --header "Authorization:
    96fe14a1-a5dc-4861-89b8-69b5c2a27e7d"

{"id":16,"createdDate":"2017-04-08 19:42:35.0","name":"Get"}

curl --request POST --url
    http://localhost:8080/multi_archive/operations --header
    "content-type: application/x-www-form-urlencoded" --data
    name=List --header "Authorization:
    96fe14a1-a5dc-4861-89b8-69b5c2a27e7d"

{"id":17,"createdDate":"2017-04-08 19:42:35.0","name":"List"}

curl --request POST --url
    http://localhost:8080/multi_archive/operations --header
    "content-type: application/x-www-form-urlencoded" --data
    name=Update --header "Authorization:
    96fe14a1-a5dc-4861-89b8-69b5c2a27e7d"

{"id":18,"createdDate":"2017-04-08 19:42:35.0","name":"Update"}
```

Listing 25: Create Operations webservice call. It is shown a command that invokes the web service and then the respective response (JSON)

11. Create Permissions
    Creating permissions is needed to let user created above do something. For this guide we will create the following permissions:
    - Add Resources
    - List Resources
    - Get Resources

```
curl --request POST --url
    http://localhost:8080/multi_archive/permissions --header
    "Authorization: 96fe14a1-a5dc-4861-89b8-69b5c2a27e7d" --header
    "content-type: application/x-www-form-urlencoded" --data
    "operationName=Add&categoryName=Resource"

{"id":36,"createdDate":"2017-04-08
    19:44:37.0","resource":null,"operation":
    {"id":14,"createdDate":"2017-04-08
```

```
    19:42:34.0","name":"Add"},"category":
    {"id":10,"createdDate":"2017-04-08
    19:42:24.0","name":"Resource"}}

curl --request POST --url
    http://localhost:8080/multi_archive/permissions --header
    "Authorization: bedb9e12-93ea-4e3d-a517-7e9ba21c25a8" --header
    "content-type: application/x-www-form-urlencoded" --data
    "operationName=List&categoryName=Resource"

{"id":1109,"createdDate":"Fri Jun 16 02:09:31 WEST
    2017","resource":null, "operation":
    {"id":17,"createdDate":"2017-04-08 19:42:35.0","name":"List"},
    "category": {"id":10,"createdDate":"2017-04-08
    19:42:24.0","name":"Resource"}}r

curl --request POST --url
    http://localhost:8080/multi_archive/permissions --header
    "Authorization: bedb9e12-93ea-4e3d-a517-7e9ba21c25a8" --header
    "content-type: application/x-www-form-urlencoded" --data
    "operationName=Get&categoryName=Resource"

{"id":1110,"createdDate":"Fri Jun 16 02:10:22 WEST
    2017","resource":null,"operation":
    {"id":16,"createdDate":"2017-04-08
    19:42:35.0","name":"Get"},"category":
    {"id":10,"createdDate":"2017-04-08
    19:42:24.0","name":"Resource"}}
```

Listing 26: Create Permissions webservice call. It is shown a command that invokes the web service and then the respective response (JSON)

12. Append Permissions to Role
    After the permissions are created we will associate them to DemoRole.

```
curl --request POST --url
    http://localhost:8080/multi_archive/permissionToRole --header
    "Authorization: bedb9e12-93ea-4e3d-a517-7e9ba21c25a8" --header
    "content-type: application/x-www-form-urlencoded" --data
    "idRole=1108&idPermission=36"

curl --request POST --url
```

```
    http://localhost:8080/multi_archive/permissionToRole --header
    "Authorization: bedb9e12-93ea-4e3d-a517-7e9ba21c25a8" --header
    "content-type: application/x-www-form-urlencoded" --data
    "idRole=1108&idPermission=1109"

curl --request POST --url
    http://localhost:8080/multi_archive/permissionToRole --header
    "Authorization: bedb9e12-93ea-4e3d-a517-7e9ba21c25a8" --header
    "content-type: application/x-www-form-urlencoded" --data
    "idRole=1108&idPermission=1110"
```

Listing 27: Command to append a paermission to a role. No JSON returned but only error code 200 if it was successful.

13. Append Role to User
    For now, Demo User will have only one Role: DemoRole. All associated permissions will be permissions of Demo User.

```
curl --request POST --url
    http://localhost:8080/multi_archive/roleToUser --header
    "Authorization: bedb9e12-93ea-4e3d-a517-7e9ba21c25a8" --header
    "content-type: application/x-www-form-urlencoded" --data
    "idRole=1108&idUser=1107"
```

Listing 28: Command to append a user to a role. Only error code 200 if it was successful.

14. Logout
    Now, we will logout the dicoogle (root) account.

```
curl --request POST --url
    http://localhost:8080/multi_archive/logout --header
    "content-type: application/x-www-form-urlencoded" --header
    "Authorization: bedb9e12-93ea-4e3d-a517-7e9ba21c25a8"
```

Listing 29: Logging out from the system using the access token.

```
{"status":"success"}
```

Listing 30: Create Facility

15. Login with DemoUser
    In this step, we will get the session token to the recently created DemoUser.

```
curl --request POST --url http://localhost:8080/multi_archive/login
    --header "content-type: application/x-www-form-urlencoded"
    --data "username=demo@email.com&password=demopassword"

{"status":"success","token":"f3346243-16ee-482f-9ad3-f285bf208c80"}
```

Listing 31: Create Facility

16. Request to STOW-RS using DICOM sample file
    For user to request the storage of a DICOM file over the service STOW-RS, additionally to the request without the accounting platform, the HTTP header Authorization must be sent with the session token.

```
curl -i -X POST -H "Content-Type: multipart/form-data" --header
    "authorization: f3346243-16ee-482f-9ad3-f285bf208c80" -F
    "data=@/Users/rui/Downloads/ThesisDataset/xr_chicken2.dcm;
    type=application/dicom" http://localhost:8080/ext/stow
```

Listing 32: STOW-RS

```
HTTP/1.1 100 Continue

HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: X-Requested-With, Content-Type,
    Accept, Origin, Authorization, Content-Length
Access-Control-Allow-Methods: GET,POST,HEAD,PUT,DELETE
Date: Fri, 16 Jun 2017 09:14:16 GMT
Accept-Ranges: bytes
Server: Restlet-Framework/2.1.2
Vary: Accept-Charset, Accept-Encoding, Accept-Language, Accept
Content-Length: 0
```

Listing 33: Create Facility

17. Querying with QIDO-RS
    Only users with the permission to list resources are available to perform this step. So, because we gave that permission above, DemoUser is able to querying using QIDO-RS.

```
curl --request GET --url
    "http://localhost:8080/ext/patient?PatientID=CHICKEN" --header
    uthorization: 5aac7479-71a7-4610-b00f-1f1d030b1042"
```

Listing 34: Create Facility

```
{"studies": [{"attributes": {"StudyInstanceUID":
    "1.2.392.200036.9125.2.36232624471.64658633050.171611"},
    "series": [{"attributes":{"Modality":"CR", "SeriesInstanceUID":
    "1.2.392.200036.9125.3.36232624471.64658633051.171614"},
    "images":[{}]},
    {"attributes":{"Modality":"CR","SeriesInstanceUID":
    "1.2.392.200036.9125.3.36232624471.64658633051.171615"}
    ,"images":[{}]}]}]}
```

Listing 35: Create Facility

18. Web access using WADO-RS
    In this step, we will request the access over web to a resource that we previously sent to Dicoogle platform using STOW-RS.

```
curl --request GET --url http://localhost:8080/ext/instance/
    1.3.12.2.1107.5.4.5.35017.4.0.494841222911107.512/frame/1
    --header "Accept: multipart/related" --header "authorization:
    5c719df1-6e77-4f55-b439-5b77f7e0c7d2" --header
    "type:application/dicom"
```

Listing 36: Create Facility

Bibliography

[1] A. F. S. H. Article, *Prepare for Disasters & Tackle Terabytes When Evaluating Medical Image Archiving*, 2008. [Online]. Available: http://ww2.frost.com.

[2] L. A. B. Silva, "Medical Imaging Services Supported on Cloud", PhD thesis, 2011.

[3] S. Parker, *Mcgraw-hill dictionary of scientific and technical terms*. McGraw-Hill Education, 2003.

[4] E. Iadanza and J. Dyro, *Clinical engineering handbook*, ser. Biomedical Engineering. Elsevier Science, 2004, ISBN: 9780080476575.

[5] C. Costa, J. L. Oliveira, A. Silva, V. G. Ribeiro, and J. Ribeiro, "Design, development, exploitation and assessment of a Cardiology Web PACS", *Computer Methods and Programs in Biomedicine*, vol. 93, no. 3, pp. 273–282, 2009, ISSN: 01692607. DOI: 10.1016/j.cmpb.2008.10.015.

[6] O. S. Pianykh, *Digital imaging and communications in medicine: A practical introduction and survival guide*, 1st ed. Springer Publishing Company, Incorporated, 2008.

[7] H. Huang, *Pacs and imaging informatics: Basic principles and applications*. Wiley, 2004.

[8] T. M. Godinho and C. Costa, "Distributed pacs: Performance and availability", p. 83, 2013.

[9] J. Zhang, J. Sun, and J. N. Stahl, "PACS and Web-based image distribution and display", *Computerized Medical Imaging and Graphics*, vol. 27, no. 2, pp. 197–206, 2002, ISSN: 08956111. DOI: 10.1016/S0895-6111(02)00074-5.

[10] W. D. Bidgood, S. C. Horii, F. W. Prior, and D. E. Van Syckle, "Understanding and Using DICOM, the Data Interchange Standard for Biomedical Imaging", *Journal of the American Medical Informatics Association*, vol. 4, no. 3, pp. 199–212, 1997.

[11] P. Mildenberger, M. Eichelberg, and E. Martin, "Introduction to the DICOM standard", *European Radiology*, vol. 12, no. 4, pp. 920–927, 2002.

[12] NEMA, *Digital Imaging and Communications in Medicine (DICOM) Part 3: Information Object Definitions*. 2004.

[13] B. Kristianto, C. Wen-yaw, and T. Yuh-show, "DICOM Waveform Generator", 2008.

[14] NEMA, *Digital Imaging and Communications in Medicine (DICOM) Part 7 : Message Exchange*. 2003, vol. 3.

[15] A. P. Alves, T. M. Godinho, and C. Costa, "Assessing the relational database model for optimization of content discovery services in medical imaging repositories", in *2016 IEEE 18th International Conference on e-Health Networking, Applications and Services (Healthcom)*, IEEE,

Sep. 2016, pp. 1–6, ISBN: 978-1-5090-3370-6. DOI: 10.1109/HealthCom.2016.7749484. [Online]. Available: http://ieeexplore.ieee.org/document/7749484/.

[16] L. A. B. Silva, C. Costa, and J. L. Oliveira, "Semantic search over DICOM Repositories", in *Proceedings - 2014 IEEE International Conference on Healthcare Informatics, ICHI 2014*, IEEE, Sep. 2014, pp. 238–246, ISBN: 9781479957019. DOI: 10.1109/ICHI.2014.41. [Online]. Available: http://ieeexplore.ieee.org/document/7052496/.

[17] G. V. Koutelakis and D. K. Lymperopoulos, "PACS through web compatible with DICOM standard and WADO service: Advantages and implementation", in *Annual International Conference of the IEEE Engineering in Medicine and Biology - Proceedings*, IEEE, Aug. 2006, pp. 2601–2605, ISBN: 1424400325. DOI: 10.1109/IEMBS.2006.260761. [Online]. Available: http://ieeexplore.ieee.org/document/4462329/.

[18] NEMA, *Digital Imaging and Communications in Medicine (DICOM) Part 18 : Web Access to DICOM Persistent Objects (WADO)*. 2009, vol. 3, pp. 1–21.

[19] *6.5 WADO-RS Request/Response*. [Online]. Available: http://dicom.nema.org/dicom/2013/output/chtml/part18/sect%7B%5C_%7D6.5.html (visited on 01/26/2017).

[20] *6.6 STOW-RS Request/Response*. [Online]. Available: http://dicom.nema.org/medical/dicom/current/output/chtml/part18/sect%7B%5C_%7D6.6.html (visited on 01/26/2017).

[21] *6.7 QIDO-RS Request/Response*. [Online]. Available: http://dicom.nema.org/medical/dicom/current/output/chtml/part18/sect%7B%5C_%7D6.7.html (visited on 01/26/2017).

[22] F. Cao, H. K. Huang, and X. Q. Zhou, *Medical image security in a HIPAA mandated PACS environment*, 2003.

[23] W. K. Seng, M. H. Kim, R. Besar, and F. Salleh, "A Secure Model for Medical Data Sharing", *International Journal of Database Theory and Application*, no. January 2008, pp. 45–52, 2006.

[24] A. Chaudhury, K. Nam, and H. R. Rao, "Management of information systems outsourcing: A bidding perspective", *Journal of Management Information Systems*, vol. 12, no. 2, pp. 131–159, 1995.

[25] A. Al-Haj, "Providing Integrity, Authenticity, and Confidentiality for Header and Pixel Data of DICOM Images", *Journal of Digital Imaging*, vol. 28, no. 2, pp. 179–187, Apr. 2015.

[26] S. Abolfazli, Z. Sanaei, M. H. Sanaei, M. Shojafar, and A. Gani, *Encyclopedia of Cloud Computing*. 2015, pp. 1–15, ISBN: 9781118821978.

[27] E. Hammer-Lahav, "The oauth 1.0 protocol", RFC Editor, RFC 5849, Apr. 2010, http://www.rfc-editor.org/rfc/rfc5849.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5849.txt.

[28] R. Boyd, *Getting Started with OAuth 2.0*. O'Reilly, 2012, p. 82, ISBN: 9781449311605. [Online]. Available: http://shop.oreilly.com/product/0636920021810.do.

[29] D. Hardt, "The oauth 2.0 authorization framework", RFC Editor, RFC 6749, Oct. 2012, http://www.rfc-editor.org/rfc/rfc6749.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6749.txt.

[30] "A primer to the oauth protocol", *Linux J.*, vol. 2011, no. 206, Jun. 2011, ISSN: 1075-3583. [Online]. Available: http://dl.acm.org/citation.cfm?id=1992856.1992864.

[31] K. Liu and K. Xu, "OAuth based authentication and authorization in open telco API", in *Proceedings - 2012 International Conference on Computer Science and Electronics Engineering, ICCSEE 2012*, vol. 1, IEEE, Mar. 2012, pp. 176–179, ISBN: 9780769546476. DOI: 10.1109/ICCSEE.2012.275. [Online]. Available: http://ieeexplore.ieee.org/document/6187855/.

[32] OASIS, *OASIS Security Services (SAML) TC | OASIS*, 2008. [Online]. Available: https://www.oasis-open.org/committees/tc%7B%5C_%7Dhome.php?wg%7B%5C_%7Dabbrev=security%20http://www.oasis-open.org/committees/tc%7B%5C_%7Dhome.php?wg%7B%5C_%7Dabbrev=security (visited on 07/11/2017).

[33] A. Armando, R. Carbone, L. Compagna, J. Cuellar, and L. Tobarra, "Formal analysis of SAML 2.0 web browser single sign-on", in *Proceedings of the 6th ACM workshop on Formal methods in security engineering - FMSE '08*, 2008, pp. 1–10, ISBN: 9781605582887. DOI: 10.1145/1456396.1456397.

[34] N. Ragouzis, J. Hughes, R. Philpott, E. Maler, P. Madsen, and T. Scavo, "Security Assertion Markup Language (SAML) V2.0 Technical Overview (OASIS)", *May*, no. February, p. 50, 2007.

[35] W. Puech and J. M. Rodrigues, "Crypto-Compression of Medical Images by Selective Encryption of DCT", *EUSIPCO'05: European Signal Processing Conference*, p. x, 2005. [Online]. Available: http://hal-lirmm.ccsd.cnrs.fr/lirmm-00106485.

[36] W. Puech, "Image encryption and compression for medical image security", in *2008 1st International Workshops on Image Processing Theory, Tools and Applications, IPTA 2008*, 2008, ISBN: 9781424433223. DOI: 10.1109/IPTA.2008.4743800. [Online]. Available: https://hal-lirmm.ccsd.cnrs.fr/lirmm-00371814.

[37] A. Kanso and M. Ghebleh, "An efficient and robust image encryption scheme for medical applications", *Communications in Nonlinear Science and Numerical Simulation*, vol. 24, no. 1-3, pp. 98–116, 2015, ISSN: 10075704. DOI: 10.1016/j.cnsns.2014.12.005. [Online]. Available: http://dx.doi.org/10.1016/j.cnsns.2014.12.005.

[38] M. Sokouti, A. Zakerolhosseini, and B. Sokouti, "Medical Image Encryption: An Application for Improved Padding Based GGH Encryption Algorithm", *The Open Medical Informatics Journal*, vol. 10, no. 1, pp. 11–22, 2016, ISSN: 1874-4311. DOI: 10.2174/1874431101610010011. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/27857824%20http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC5090780%20http://benthamopen.com/ABSTRACT/TOMINFOJ-10-11.

[39] L.-b. Zhang, Z.-l. Zhu, B.-q. Yang, W.-y. Liu, H.-f. Zhu, and M.-y. Zou, "Medical Image Encryption and Compression Scheme Using Compressive Sensing and Pixel Swapping Based Permutation Approach", vol. 2015, 2015.

[40] W. Cao, Y. Zhou, C. Chen, and L. Xia, "Medical image encryption using edge maps", *Signal Processing*, vol. 132, no. September 2016, pp. 96–109, 2017, ISSN: 01651684. DOI: 10.1016/j.sigpro.2016.10.003. [Online]. Available: http://dx.doi.org/10.1016/j.sigpro.2016.10.003.

[41] C. Costa, C. Ferreira, L. Bastião, L. Ribeiro, A. Silva, and J. L. Oliveira, "Dicoogle - An open source peer-to-peer PACS", *Journal of Digital Imaging*, vol. 24, no. 5, pp. 848–856, Oct. 2011. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/20981467%20http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC3180530%20http://link.springer.com/10.1007/s10278-010-9347-9.

[42] E. Pinho, T. Godinho, F. Valente, and C. Costa, *A Multimodal Search Engine for Medical Imaging Studies*, 2016. DOI: 10.1007/s10278-016-9903-z.

[43] C. Costa, F. Freitas, M. Pereira, A. Silva, and J. L. Oliveira, "Indexing and retrieving DICOM data in disperse and unstructured archives", *International Journal of Computer Assisted Radiology and Surgery*, vol. 4, no. 1, pp. 71–77, Jan. 2009. DOI: 10.1007/s11548-008-0269-7. [Online]. Available: http://link.springer.com/10.1007/s11548-008-0269-7.

[44] F. Valente, C. Costa, and A. Silva, "Dicoogle, a Pacs Featuring Profiled Content Based Image Retrieval", *PLoS ONE*, vol. 8, no. 5, P. V. Benos, Ed., e61888, May 2013, ISSN: 19326203.

DOI: 10.1371/journal.pone.0061888. [Online]. Available: http://dx.plos.org/10.1371/journal.pone.0061888.

[45] C. Viana-Ferreira, C. Costa, and J. L. Oliveira, "Dicoogle relay - A cloud communications bridge for medical imaging", in *Proceedings - IEEE Symposium on Computer-Based Medical Systems*, IEEE, Jun. 2012, pp. 1–6, ISBN: 9781467320511. DOI: 10.1109/CBMS.2012.6266402. [Online]. Available: http://ieeexplore.ieee.org/document/6266402/.

[46] *Dicoogle plugin development*. [Online]. Available: https://github.com/bioinformatics-ua/dicoogle/wiki/Plugin-Development (visited on 01/24/2017).

[47] L. A. B. Silva, "Federated architecture for biomedical data integration", 2015.

[48] Google, *Google Drive storage plans & pricing*, 2016. [Online]. Available: https://support.google.com/drive/answer/2375123?hl=en (visited on 01/29/2017).

[49] S. Devarakonda, P. Sevusu, H. Liu, R. Liu, L. Iftode, and B. Nath, "Real-time air quality monitoring through mobile sensing in metropolitan areas", *Proceedings of the 2nd ACM SIGKDD International Workshop on Urban Computing - UrbComp '13*, p. 1, 2013. DOI: 10.1145/2505821.2505834. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2505821.2505834.

[50] I. Claros, R. Cobos, E. Guerra, J. De Lara, A. Pescador, and J. Sanchez-Cuadrado, "Integrating open services for building educational environments", *IEEE Global Engineering Education Conference, EDUCON*, pp. 1147–1156, 2013, ISSN: 21659559. DOI: 10.1109/EduCon.2013.6530253.

[51] Google, *Cloud Storage - Online Data Storage - Google Cloud Platform*, 2016. [Online]. Available: https://cloud.google.com/storage/ (visited on 01/30/2017).

[52] Microsoft, *Microsoft OneDrive*, 2014. [Online]. Available: https://onedrive.live.com/about/en-US/plans/ (visited on 01/29/2017).

[53] M. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel, "Amazon S3 for Science Grids: a Viable Solution", in *Proc. ACM Int. Workshop on Data-aware Distributed Computing*, 2008, pp. 55–64, ISBN: 978-1-60558-154-5. DOI: http://doi.acm.org/10.1145/1383519.1383526.

[54] Amazon, *Amazon Simple Storage Service (Amazon S3)*, 2011. [Online]. Available: https://aws.amazon.com/s3/%20http://aws.amazon.com/s3/ (visited on 01/30/2017).

[55] S. S. Garfinkel, "Commodity grid computing with Amazon's S3 and EC2", *Usenix*, pp. 7–13, 2007.

[56] D. Quick and K. K. R. Choo, "Dropbox analysis: Data remnants on user machines", *Digital Investigation*, vol. 10, no. 1, pp. 3–18, 2013, ISSN: 17422876. DOI: 10.1016/j.diin.2013.02.003.

[57] I. Drago, M. Mellia, M. M Munafo, A. Sperotto, R. Sadre, and A. Pras, "Inside dropbox", *the 2012 ACM conference*, p. 481, 2012. DOI: 10.1145/2398776.2398827. [Online]. Available: http://traces.simpleweb.org/dropbox/%20http://dl.acm.org/citation.cfm?doid=2398776.2398827%7B%5C%%7D5Cnpapers3://publication/doi/10.1145/2398776.2398827.

[58] Dropbox, *Dropbox Business*, 2014. [Online]. Available: https://www.dropbox.com/?landing=dbv2/ (visited on 01/29/2017).

[59] R. W. Shirey, *Internet Security Glossary, Version 2*, RFC 4949, Aug. 2007. DOI: 10.17487/RFC4949. [Online]. Available: https://rfc-editor.org/rfc/rfc4949.txt.

[60] *OACC | Java Application Security Framework*. [Online]. Available: http://oaccframework.org/ (visited on 07/09/2017).

[61] *Apache Shiro | Simple. Java. Security.* [Online]. Available: https://shiro.apache.org/ (visited on 07/09/2017).

[62] A. Alves, "DICOOGLE: No-SQL to support big data environments", 2016.

[63] Hibernate, *Hibernate. Everything data. - Hibernate*, 2015. [Online]. Available: `http://hibernate.org/` (visited on 07/07/2017).

[64] U. Bioinformatics, *Dicoogle - Open Source PACS*, 2017. [Online]. Available: `https://github.com/bioinformatics-ua/dicoogle` (visited on 07/08/2017).