



**Daniel  
De Matos Dias**

**Plataforma para gestão de pacientes com doenças  
respiratórias crónicas**

**Platform for chronic respiratory disease patient  
management**





**Daniel  
De Matos Dias**

**Plataforma para gestão de pacientes com doenças  
respiratórias crónicas**

**Platform for chronic respiratory disease patient  
management**

Dissertation presented to the University of Aveiro to comply with necessary requirements to obtain the Masters Degree in Computer and Telematics Engineering, performed under scientific supervision of Prof. José Luís Oliveira and Dr. Sérgio Matos.



## **The jury**

The president

**Prof. Doutor Augusto Marques Ferreira da Silva**  
Professor Auxiliar, Universidade de Aveiro (Presidente)

Other members

**Prof. Doutor António Manuel de Jesus Pereira**  
Professor Coordenador, Departamento de Engenharia Informática  
da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Leiria

**Prof. Doutor José Luís Guimarães Oliveira**  
Professor Associado da Universidade de Aveiro (Orientador)

**Doutor Sérgio Aleixo de Matos**  
Investigador Auxiliar da Universidade de Aveiro (Co-Orientador)



## **acknowledgements**

I would like to express my gratitude to everyone that, directly and indirectly, made possible the achievement of this project.

To Prof. José Luís Oliveira and to Prof. Sérgio Matos my faithful thank you for your guidance, critic and suggestions presented during the development of this dissertation.

To my parents, family and friends, who always supported and incentivized me regardless of the effort given to assure the completion and success of this academic challenge, I kindly thank you.



**palavras-chave**

Monitorização de pacientes, aplicação Web, tosse crónica

**resumo**

As doenças respiratórias são uma das razões mais frequentes para consultas médicas e uma das causas de morte mais frequentes a nível mundial, representando gastos de vários milhões de euros. A frequência e grau de severidade de ocorrência de tosse é um dos principais indicadores a analisar, por ser este o sintoma mais frequente em grande parte destas doenças respiratórias e por ser também um bom indicador da evolução do estado do paciente.

Dada a importância deste sintoma, foi desenvolvido o Leicester Cough Monitor, uma aplicação que permite obter, de forma automática e não invasiva, dados quantitativos fiáveis relativos à frequência de ocorrência de tosse. Contudo, essa aplicação funciona apenas em modo local, tendo esta de estar instalada no computador onde se deseja realizar o estudo não havendo qualquer tipo de ligação e comunicação entre mais que um computador. Dado este facto, não é possível manter de forma eficiente um sincronismo de informação e de estudos realizados em mais que um computador ou em diferentes centros.

O trabalho descrito nesta dissertação teve como objectivo fundamental desenvolver uma plataforma Web que, através da integração da ferramenta LCM já existente, permita a realização de estudos sobre a evolução dos sintomas de tosse de vários pacientes. Foi então desenvolvida uma plataforma que permite ter informação organizada, sincronizada e reunida num único ponto estando esta acessível de qualquer local com acesso à Internet. A disponibilização das várias funcionalidades da ferramenta LCM numa vertente Web foi o foco principal, sendo que novas funcionalidades foram criadas de modo a permitir de uma forma organizada e controlada a gestão de toda a informação com a implementação de um sistema de gestão de utilizadores.



**keywords**

Patient monitoring, Web application, chronic cough

**abstract**

Respiratory diseases are one of the most frequent reasons for medical appointments and one of the main causes of death worldwide, accounting costs of several millions of Euros. The frequency and severity of occurrence of cough is one of the most important indicators, being the most frequent symptom in many of these diseases and also a good indicator of the evolution of the patient's disease state.

Given the importance of this symptom, the Leicester Cough Monitor, an application that allows obtaining, in an automatic and noninvasively way, reliable quantitative data on the frequency of occurrence of cough, was developed. However, this application only works locally, requiring that it is installed on the computer where we want to perform the study with no other kind of connection and communication between more than one computer. Given this fact, it is not possible to efficiently maintain synchronism of information and studies on more than one computer or between different centers.

The work described in this dissertation had as fundamental goal the development of a Web platform that, through the integration of the existing LCM tool, enables studies of the evolution of patients with various symptoms of cough. Thus, a platform was developed that allows having information organized, synchronized and collected at a single point being this accessible from any location with Internet access. The availability of the various functionalities of the LCM tool on a Web context was the main focus, and new features were created to allow an organized and controlled management of all information with the implementation of a user management system.



# Index

<b>Index</b> .....	<b>i</b>
<b>List of Figures</b> .....	<b>iii</b>
<b>List of Tables</b> .....	<b>v</b>
<b>Acronyms</b> .....	<b>vii</b>
<b>Chapter 1 Introduction</b> .....	<b>1</b>
1.1 Background .....	1
1.1.1 Leicester Cough Monitor (LCM).....	1
1.2 Objectives.....	3
1.3 Dissertation Structure.....	4
<b>Chapter 2 System Requirements</b> .....	<b>5</b>
2.1 Functional Requirements .....	5
2.1.1 System Interface .....	8
2.1.2 Non Functional Requirements.....	9
2.2 Architecture and Model.....	9
2.2.1 Centralized System .....	9
2.2.2 Distributed System .....	13
2.3 File Transfer.....	14
2.3.1 File Transfer via Web Services .....	14
2.3.2 File Transfer using Dropbox.....	15
2.3.3 File Transfer using HTTP File Upload.....	15
<b>Chapter 3 Technological Solutions</b> .....	<b>17</b>
3.1 LCM Integration.....	17

3.1.1 Java Native Interface (JNI) .....	17
3.1.2 Executable Files.....	18
3.2 Data Storage.....	19
3.2.1 SQL Databases.....	19
3.2.2 NoSQL Databases.....	20
3.2.3 MongoDB .....	22
3.2.4 JPA vs Native Driver .....	23
3.3 Web interface framework.....	24
3.4 Users Management with RBAC policy.....	25
3.4.1 Spring Security Framework.....	26
<b>Chapter 4 System Implementation.....</b>	<b>29</b>
4.1 Data Storage.....	30
4.2 Application structure .....	33
4.3 User Security and Authentication Management.....	35
4.4 File Management.....	37
4.5 Application Dependencies.....	38
4.6 JSF Client Server with Ajax.....	39
4.7 Web Application.....	41
4.7.1 User Management.....	42
4.7.2 LCM Data Processing.....	43
<b>Chapter 5 Conclusions and Future Work .....</b>	<b>51</b>
5.1 Conclusions .....	51
5.2 Future Work.....	52
<b>Bibliography.....</b>	<b>53</b>

## List of Figures

Figure 1.1 - Process to make recordings of audio samples [3] .....	2
Figure 1.2 - Overview of cough analysis system of LCM (adapted from [3]).....	3
Figure 2.1- Use Case Diagram .....	7
Figure 2.2 - Graphic Interface Prototype of Web Application .....	8
Figure 2.3- General Architecture of the Application .....	10
Figure 2.4 - Sequence diagram illustrating the creation of a new study.....	11
Figure 2.5- LCM Requests Attendance Queue.....	12
Figure 3.1 - JNI Lifecycle .....	18
Figure 3.2 - Work Flow of the system call .....	19
Figure 3.3 - Web Frameworks popularity .....	25
Figure 3.4 - JSF Framework popularity .....	25
Figure 3.5 - RBAC System .....	26
Figure 3.6 - Spring Security Work Flow .....	28
Figure 4.1 - Database simple architecture .....	32
Figure 4.2- MVC Simple LifeCycle.....	34
Figure 4.3 - Application Packages .....	34
Figure 4.4 - Application Dependencies at pom.xml.....	39
Figure 4.5 - Example of the interface .....	42
Figure 4.6 - User Creation Form.....	43
Figure 4.7 -Create New Study Form .....	43
Figure 4.8 - LCMEngine Processing .....	44
Figure 4.9 - Classify events as Cough .....	45
Figure 4.10 - LCMRefiner Processing.....	46
Figure 4.11- List of Studies .....	47
Figure 4.12- Study Comments.....	48
Figure 4.13 - Occurrences per Hour .....	48
Figure 4.14- Study Comparison about Occurrences per Hour .....	49



## List of Tables

Table 2.1- Requirements List .....	5
Table 3.1- Simple comparison between MongoDB and CouchDB[17] .....	21
Table 4.1 - MVC Concepts .....	33
Table 4.2 - AJAX Advantages and Disadvantages .....	40
Table 4.3 - Events.json fields from LCMEngine .....	46



# Acronyms

<b>Item/Term</b>	<b>Description</b>
AJAX	Asynchronous Javascript and XML
API	Application Programming Interface
BSON	Binary JavaScript Object Notation
CMS	Content Management System
CRM	Customer Relationship Management
CRUD	Create, Read, Update e Delete
FTP	File Transfer Protocol
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
JNI	Java Native Interface
JPQL	Java Persistence Query Language
JRE	Java Runtime Environment
JSF	Java Server Faces
JSON	JavaScript Object Notation
JSP	Java Server Pages
LCM	Leicester Cough Monitor
MP3	MPEG audio layer-3
MVC	Model View Controller
MVCC	Multiversion Concurrency Control
NoSQL	Not only SQL
P2P	Peer to Peer
POJO	Plain Old Java Objects
RBAC	Role Based Access Control
SQL	Structured Query Language
UUID	Universally Unique Identifier



# Chapter 1

## Introduction

### 1.1 Background

Respiratory diseases are one of the most frequent reasons for medical appointments and one of the main causes of death worldwide, accounting costs of several millions of Euros [1]. During monitoring and clinical management of patients with chronic respiratory diseases, several clinical studies are usually performed in order to obtain various indicators of the disease and its evolution. These indicators are used both for diagnosis and for determining the effectiveness of possible treatment phases. The frequency and severity of occurrence of cough is one of those indicators, being this the most frequent symptom in many of these diseases and also a good indicator of the changing state of the patient.[2].

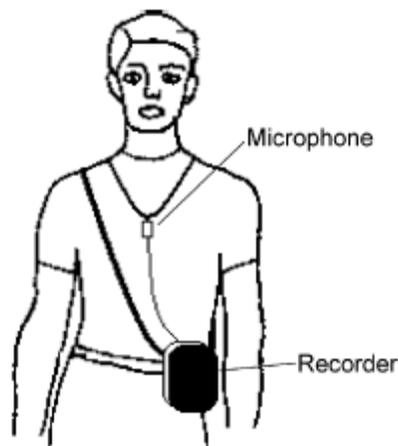
Given the importance of this symptom, some systems have recently been developed that automatically and non-invasively acquire reliable quantitative data concerning its frequency of occurrence. However, to obtain a patient's clinical evaluation, this data must be analyzed and understood in parallel with other studies and results. With this, arises the necessity to have one platform that allows grouping and processing large amounts of data, from several medical sources in order to enable the possibility to have collaborative studies between several doctors and medical centers. The collaborative studies emerged with the necessity of searching for new treatments and to provide accompaniment to patients with these pathologies [3-5].

#### 1.1.1 Leicester Cough Monitor (LCM)

In 2007 the Leicester Cough Monitor (LCM) arises in order to fill some of the necessities identified above. LCM is an automated tool and monitoring system of patients with chronic cough that uses Hidden Markov Models (HMM) to identify

possible candidate of cough events from previous recorded sound acquired from patients over a 24-hour period (free-field sound samples). This tool comes with a graphical interface which intends to facilitate its use. It is implemented in several clinical centers around the United Kingdom and it allows the ambulatory study and evaluation of the cough frequency, meaning, while patients execute their day-to-day tasks. With this system it is possible to study and monitor the symptoms of chronic cough associated to illnesses such as asthma, bronchitis or other lung diseases.

The system has some distinct phases. Before anything, the patients need to make a long time recording of their free-field sounds (usually 24 hours or more), using a portable digital audio recorder and a miniature condenser microphone close to their body (Figure 1.1)



*Figure 1.1 - Process to make recordings of audio samples [3]*

The audio is recorded in MPEG audio layer-3 (MP3) format. After collecting audio data from patients, the files can be submitted to the next phase that consists in trying to find possible cough event candidates through HMM analysis. The HMM analysis phase creates one small .wav file for each candidate, in order to facilitate reproduction of sound, keeping only the short sound segment where a candidate event was detected. Once the identification of candidate events is finalized, it is necessary to make a refinement phase. In this phase, a final binary classifier is used to differentiate, from the initial candidate events, which are real cough events and which are other noises. For this, the medical technicians will listen and classify some of the previously detected cough event candidates. These classifications will be used to refine the final classifier that is afterwards used to classify all remaining events. Figure 1.2 gives an overview of the multiple phases and actions performed by the LCM tool.

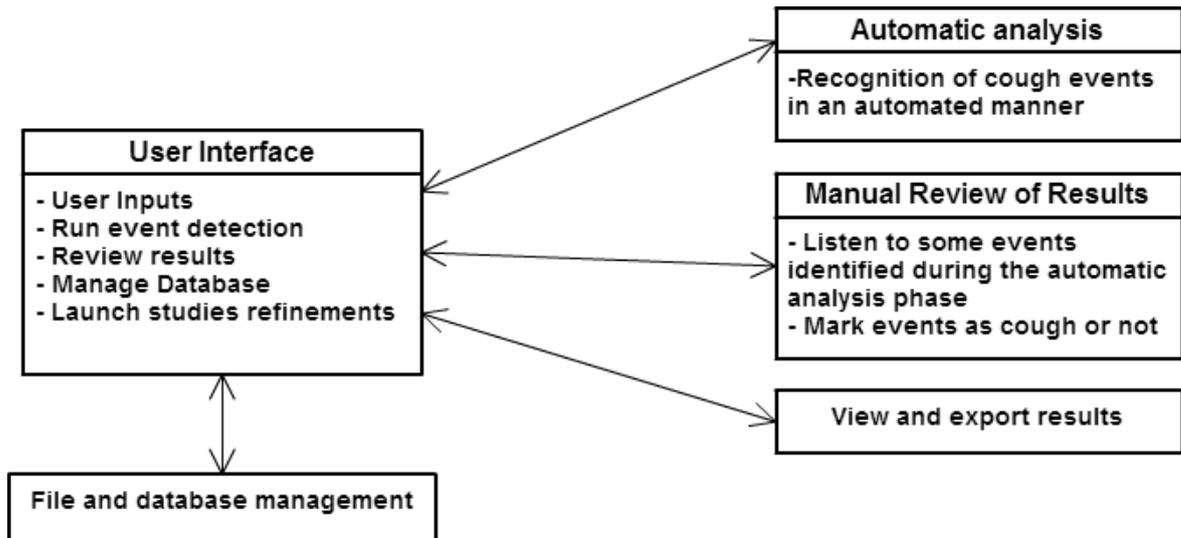


Figure 1.2 - Overview of cough analysis system of LCM (adapted from [3])

This tool needs to be integrated in the web application to be developed in this work. To do that, there are two possible technologies to re-utilize the application, without making a new full implementation of the LCM. Those technologies are explained in sections 3.1.1 and 3.1.2.

## 1.2 Objectives

The aim of this project is to develop a Web platform for clinical management of patients with respiratory diseases that allows the integration of data and results from a cough frequency monitoring system, already implemented in LCM. The previous implementation of LCM must be integrated in the Web application and provide in a web context the same functionalities as the local application.

This platform must be based on a user interface and use a flexible information system, which includes various levels of users. The possibly to expand and integrate new sources of clinical data and kinds of exams making the system as modular system will also be considered.

### 1.3 Dissertation Structure

This dissertation is divided in five chapters with the following contents:

**Chapter 1 (this chapter):** Introduces the problem and objectives to achieve with this work.

**Chapter 2:** Presentation of the various system requirements to be considered for the implementation.

**Chapter 3:** Description of technological solutions concerning the scope of the solution to implement. Analysis of some existing solutions and technologies available to solve the problem.

**Chapter 4:** Presentation of the implemented data model and system. Explanation of the developed solution, such as how the various technologies were used. Details and description of web application development and demonstration of the final outcome of the application.

**Chapter 5:** Evaluation of work and knowledge acquired. Proposal for futures works.

# Chapter 2

## System Requirements

The main goal of this project is the development of a web platform that integrates the existing LCM application, and enables the creation of studies related to respiratory diseases. The studies are supported by audio free-field samples recorded from patients during approximately 24 hours. By analyzing the outcomes of a series of studies, it will be possible to follow the evolution of patients during larger periods of time. This allows assessing the evolution of the pathology with the aim to aid in establishing a diagnosis, measure improvements in the patients' clinical status, or even help in finding a cure. This tool will enable several ways of interaction with their users, associated with each one's allowance level within the system. In this chapter the requirements identified for the application, and that are intended to reach the objectives proposed for this project, will be presented.

### 2.1 Functional Requirements

To achieve successfully the objectives proposed for this project the requirements identified are listed in Table 2.1. This list also illustrates the main functionalities that the application must provide.

*Table 2.1- Requirements List*

<b>Requirement</b>	<b>Description</b>
Submission of audio files sized around 1GB	It should be possible to the application user to submit to the system audio files with samples of the 24 hours of free-field sound of a patient.
Creation of studies based on automatic analysis of	The user should be logged in the system and have an active session to be able to create new studies.

audio files.

---

Listen, classify and refine cough events detection.	The user with adequate permissions should be capable to listen candidate cough events identified by LCM, classify them as cough and make refinements.
Create and export statistics in several formats.	Users should be able to create and export various statistics associated to the studies carried out by the application to formats as Excel, PDF and CSV.
Create personal studies.	The user should have access to the possible creation of personal studies in an isolated manner from all other studies so they can work in a scientific area such as articles publications, etc.
Creation of grouped projects or co-operative studies between several clinical centers.	It should be possible to create collaborative studies between clinical centers, using patients from the several centers but keeping their privacy and data secure.

---

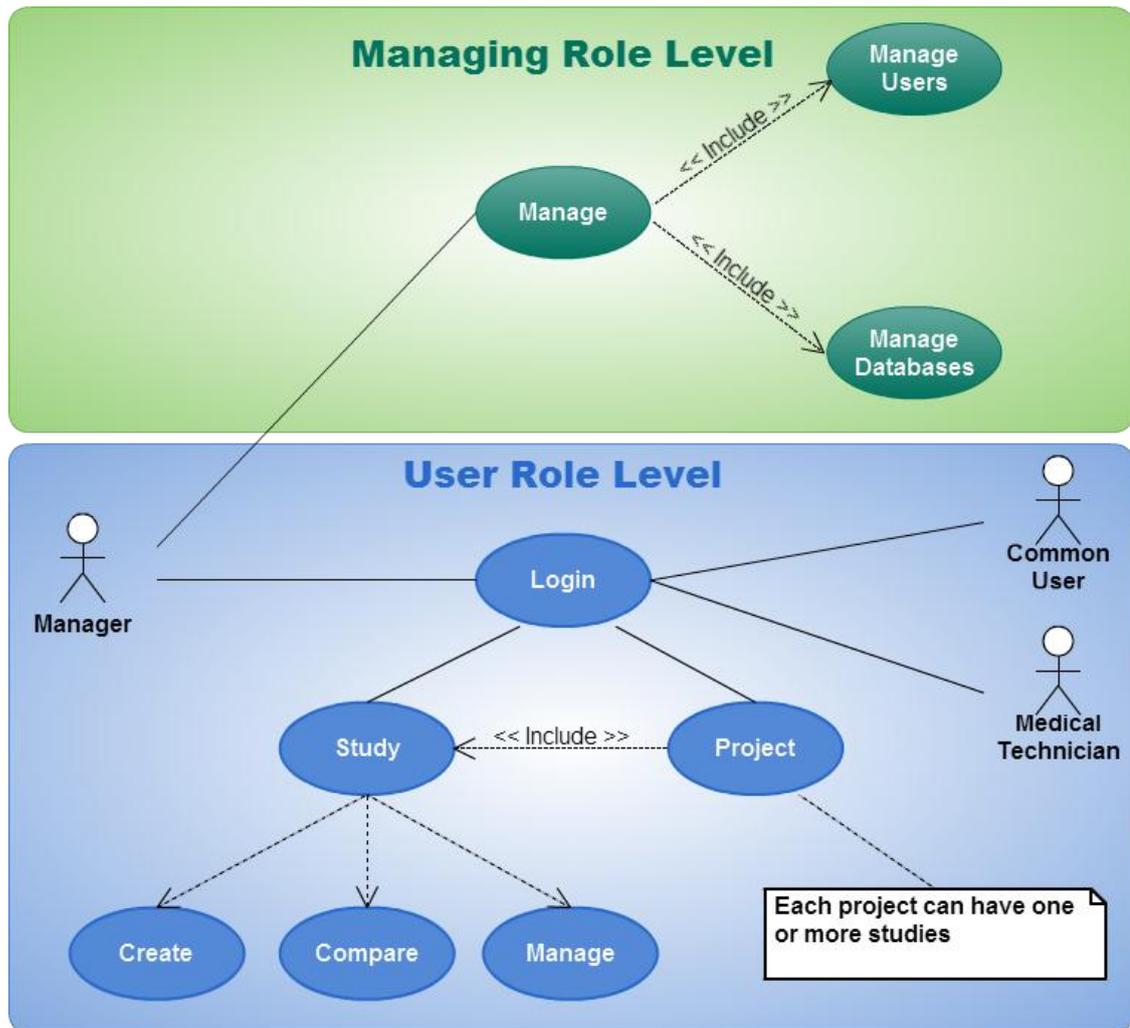


Figure 2.1- Use Case Diagram

The application to be developed must follow the requirements identified above in Table 2.1. The Use Case diagram in Figure 2.1 shows the interaction of the users with the application. The system must have three user levels to interact with the application.

The users with fewer privileges in the system are the common users. The users with this role can view existing studies, create and export statistics, compare results from existing studies, and submit new studies and audio files to be processed by the application. The users with role of medical technician can do the same things as the common users, but are also the ones responsible to listen previous detected candidate cough events, performing successive refinements. The users with more privileges are the Managers. These users are responsible for managing the system databases, assigning users to the same databases, creating new users and managing

existing ones assigning or removing roles to them. Besides those user privileges, they can also use all the application features accessible to the other user roles.

### 2.1.1 System Interface

The application to be developed will result in a Web application that will be accessed by its users through an Internet browser. This application was thought to have several levels of users. In order to facilitate the navigation to all distinct user levels at the application, although there are multiple user levels with access to distinct contents in the application, the GUI does not change in aspect or look. The GUI aspect of the application is kept for all the contents, changing only the number of accessible modules and contents to each kind of user's roles. The interface to develop should be simple and easy to navigate with intuitive and self-explanatory menus and surfing methods. With the aim to achieve the ideal interface, in the first phase of development, a prototype was designed with what should be the system interface. This interface must be built based on two information areas. In the left area of the application there will be the main menu of navigation as shown in Figure 2.2.

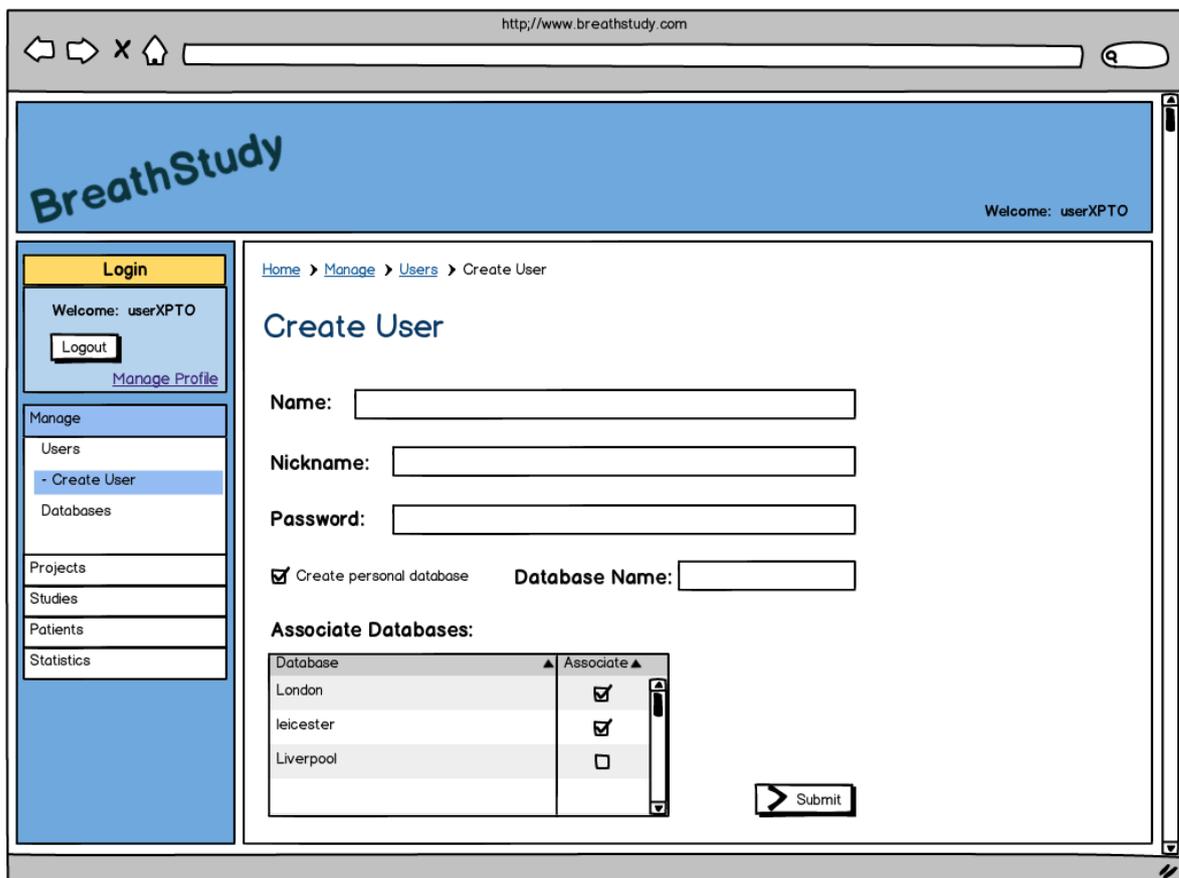


Figure 2.2 - Graphic Interface Prototype of Web Application

The right area will be the main area of the application. Here will be shown and processed the most part of the passive and active data. This area will be responsible to deal with the interaction and process of visualization, insertion, edition and searching of all the application data. The separation of the application in this two main areas allow an easy, quick and effective navigation between several features within the system, always with all options available in the left area.

### **2.1.2 Non Functional Requirements**

The application must be independent from the operative system where it is working and be able to run at the major operative systems like Microsoft Windows, Linux and OSX. So the application must be portable, and after compiled must work in the most operative systems as possible. Java is one language that after being compiled can run at any of the previous referred operative systems, as long as the Java Runtime Environment (JRE) is installed. Besides the JRE, the clients that will use the application need to have one Internet Browser to be able to connect and use the application. The application clients also need to have sound output on the computers and a way of listen the files in order to classify cough events.

## **2.2 Architecture and Model**

The application to be built in this project aims to enable the creation of studies about respiratory diseases, more specifically studies related to the objective monitoring of cough in patients that attend the several clinical centers that will possibly adopt this system in a near future. In order to achieve that, a web platform will be created to enable medical technicians from several medical centers to annotate information about their patients with the goal of performing the desirable studies. There will be two distinct ways of how the application will work and they are identified as Centralized System and Distributed System.

### **2.2.1 Centralized System**

This is the main approach of the system, contemplating all the possible functionalities of the application. In this case the system behaves analogously as shown in Figure 2.3. Besides having all the possible functionalities of the application, this version can have all the contents locally accessible. This can provide a better control and efficient performance to access and submit contents, such as large files, through the local network instead of the Internet connection. In this application variant that uses a centralized system, all the studies are analyzed and processed in the same location as the application is running.

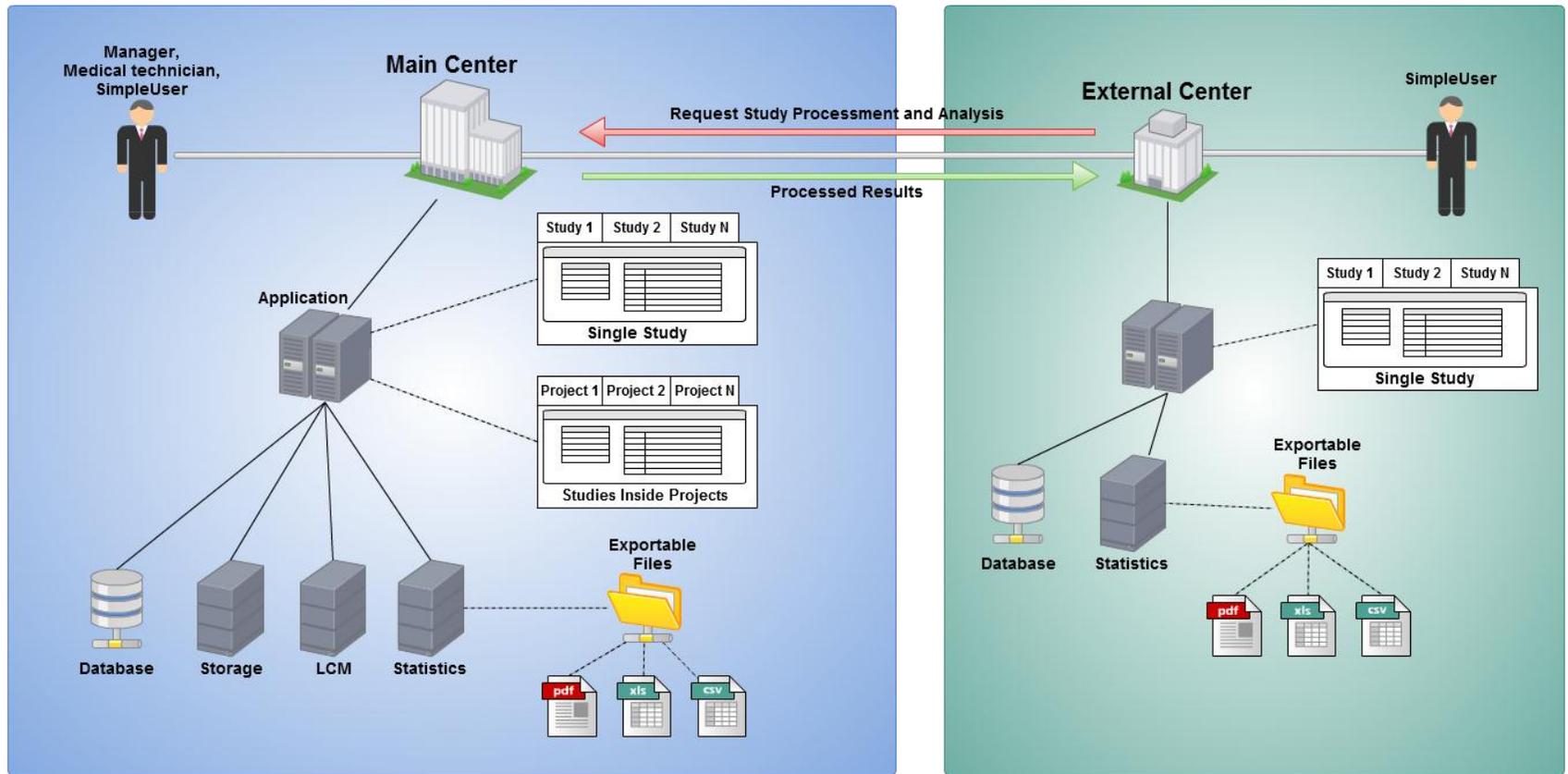


Figure 2.3- General Architecture of the Application

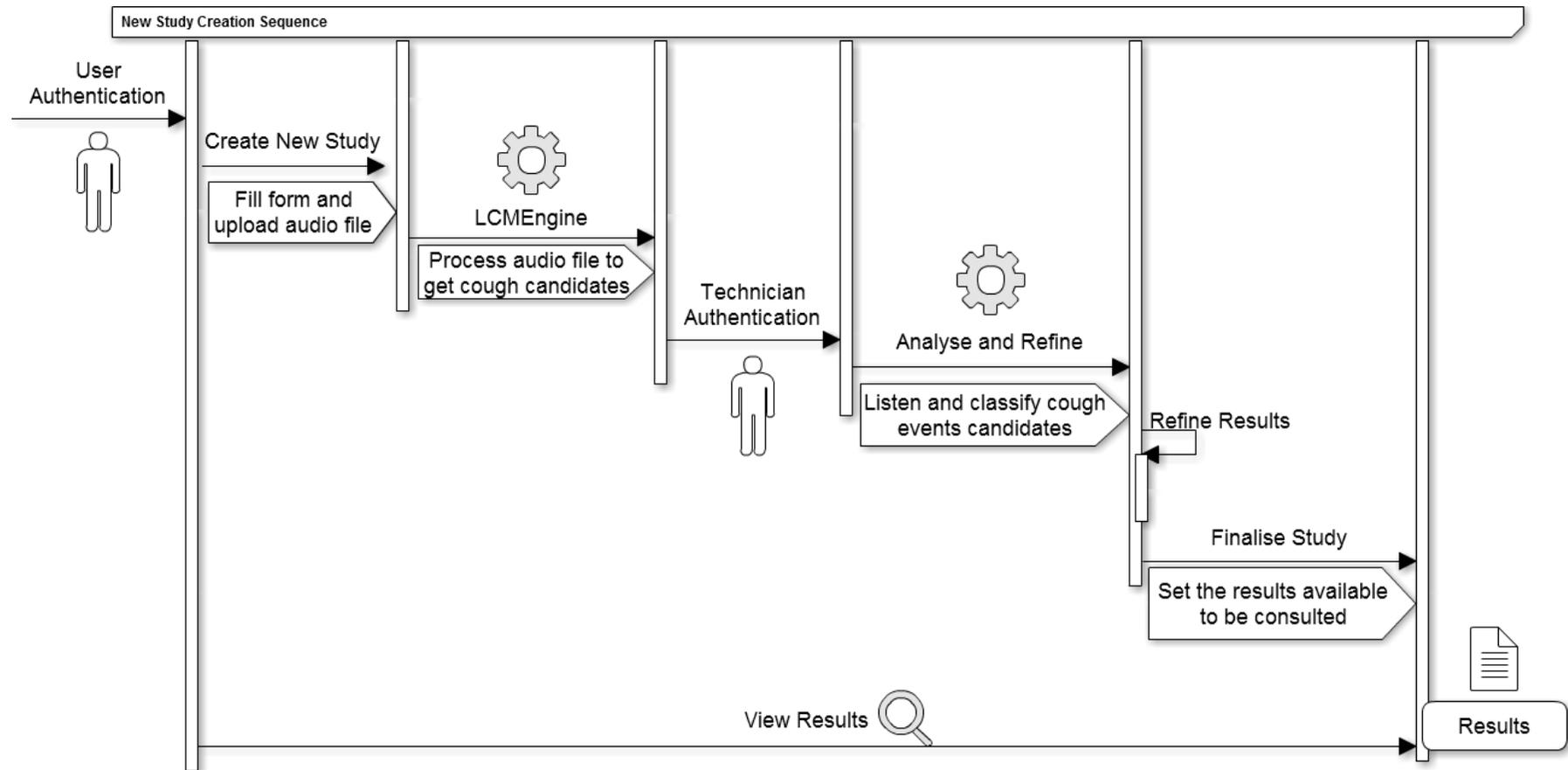


Figure 2.4 - Sequence diagram illustrating the creation of a new study

In the previous section about Functional Requirements, three distinct user levels were defined. The users with more privileges will be the system administrators having access to all the application contents. These users are allowed to manage user creation and studies. The users with medical technician role can run, analyze and refine studies performing improvements until arrives into a satisfactory results. All other users are only allowed to submit studies to be analyzed by other users with higher role levels and consult results from the existing and processed studies. The process of creating and fully analyze new studies, follows a set of steps described in the diagram of Figure 2.4.

This process starts with the authentication of some user at the application. After being validated in the system, the user creates a new study filling all the information in a web-form created to that effect and uploads an audio file with free-field sound recorded from one patient. After successful submission of the study, the sound recording will be placed in a queue to be processed by the LCMEngine, an automated process that analyses the free-field sound and identifies a list of candidate cough events. The system allows running four study analyses through the LCMEngine simultaneously, and the other requests will be kept in a queue. This queue will release the requests as soon as the system has one free slot to process the study analyses. The queue operates as illustrated in the Figure 2.5.

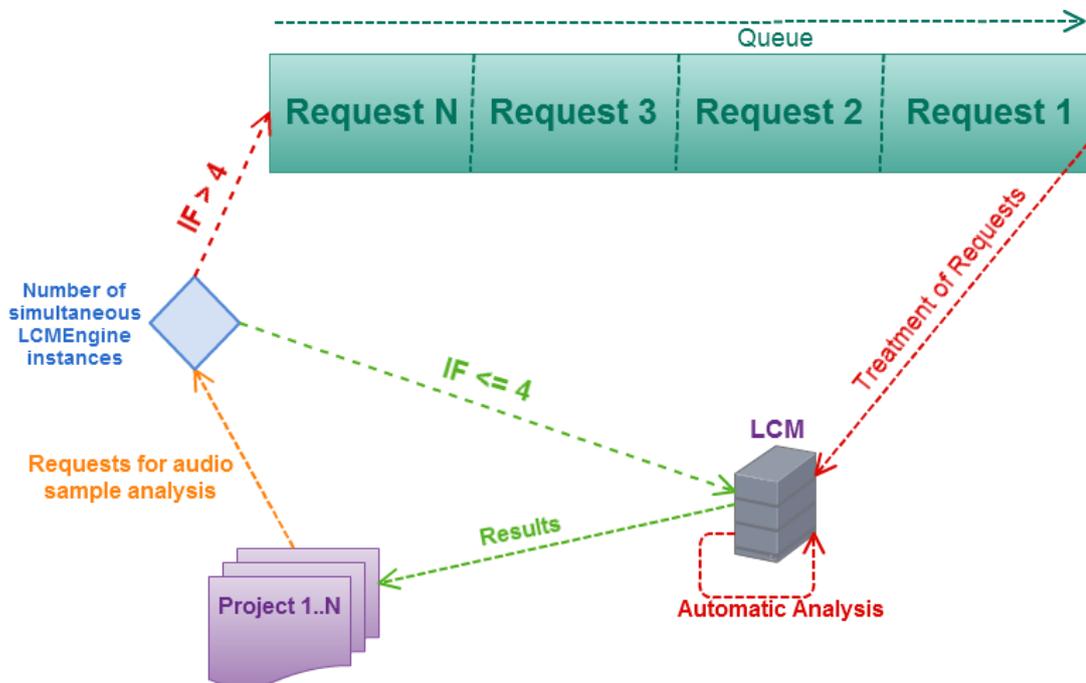


Figure 2.5- LCM Requests Attendance Queue

After the free-field sounds are processed by the LCMEngine, a user with at least technician role can evaluate and refine the study to get the desired results. This technician will be required to listen and classify some of the detected cough candidates as cough, not cough or unsure. While a medical technician classifies the candidate cough events, the application improves and adjusts the models of cough detection through the LCMRefiner. The LCMRefiner is a tool for cough event classification that works together with user inputs of cough event classification in order to refine the models to get desired results by the medical technicians.

The users of the application will be doctors in the majority of the cases and they can also add extra information to the studies such as prescriptions issued. With the information of the prescriptions and with the comparison of previous studies, the doctors can make their diagnosis and understand the evolution of their patients.

There is also the possibility of performing studies with collaborative projects between medical centers or hospitals being this process identical to the one described above. In this scenario there will be data from several different sources enabling the comparison of analysis results from patients subject to different environmental and climacteric conditions. The result of these collaborative studies may become quite interesting and richer if taken in consideration the diversity of distinct case study possibilities such as the environment.

### **2.2.2 Distributed System**

This is the second approach considered for the application. This approach intent to expand the application and gives the possibility to have it running in a context outside of the main center. In this scenario, the external centers will not have all the features running locally having some dependencies from the main center. In the external centers it will be possible to use all the application features as in the main center. However the study analysis of free-field sound recordings will always be performed at the main center. Each time a new study or project is uploaded the free field sounds will be updated to the main center. After successful file upload the studies will be stored in a queue to be analyzed by the LCMEngine tool. When available resources are in place the system will process the studies doing the automatic analysis in order to find candidate cough events. Then, a refinement must be done by one medical technician together with LCMRefiner tool. As soon as the free-field audio recording is successfully analyzed the results are sent to the external center that requested the study for analysis. Other functionalities such as view results, compare studies, manage patients or users can be done at the external centers. The aim of always performing analysis of free-field sound recordings at the main center is to guarantee

the system's quality and to ensure that each analysis is dealt by specialized technicians only.

## **2.3 File Transfer**

The file transfer is a process which must take into account several aspects in any application. In this particular case, a high speed Internet Broadband is required because generally the files of each study should be around 1GB size. It is also necessary to consider the security of the data which must be transferred in a secure and encrypted way, since medical ethics and confidentiality must be preserved. Another variable to consider is the limitation of open ports in the other hospital centers to communicate with the external environments. The limitation and restriction to use only the HTTP port for data transfer eliminate some of the possible forms of transferring data, such as FTP.

For this system it is required to assure a large storage space to be able to support the storage of tens or hundreds of gigabytes. The advisable manner to achieve this is through the use of cloud storage technology, which provides growing storage space based on the needs of the applications. On the other hand this technology becomes quite expensive which may affect the viability of the solution. Considering the facts mentioned above there are three possibilities of data transfer between hospitals, which are explained in section 2.3.1, 2.3.2 and 2.3.3.

### **2.3.1 File Transfer via Web Services**

One of the possible ways of data transfer is using WebServices. In this project and considering the size of the files to transfer it is necessary that the file transfer is somewhat similar to the P2P networks, with files of reduced size guaranteeing the capability of performing a continuous transfer in case of a fault occurrence. It is also needed to guarantee that the data travels between the client and the application server in a safe manner. The WebServices approach does not define a maximum limit of size per file. The file transfer depends on the performance of the machines that are processing the requests being highly recommended that files do not have huge size.

A possible approach to consider is to perform the transfers through http ports, taking into account the constraints of available ports for the transfers. WebServices supports the possibility to send a list of files instead of single files which allows the mechanism for chunking files with an automated file splitting system. Transfers of

smaller files together with prevision mechanisms and fault tolerance that supports resuming transfers, makes possible to have transfers in a lighter way and efficient system processing.

### **2.3.2 File Transfer using Dropbox**

The service provided by Dropbox [6] has been growing in popularity, being now-a-days one of the main companies that offer storage in a synchronized and secure manner between several computer terminals. This technology enables, in an easy, quick and economical way, the transfer of data between computer terminals and guarantees the security of the data when being transferred between terminals and when stored in the cloud. With this technology, it is possible to confirm that the data is perfectly synchronized between several terminals. Anytime a transfer is interrupted, for instance, by a Broadband fault or any other external factor, the data still will be transferred from the fault moment in an automatic, simple and transparent manner to the user. Its use through the HTTP port guarantees the transfer of data between hospitals without the need to open ports in the system that may compromise the security of the information systems in hospitals and medical centers. This way, it can be used through the platform offered by Dropbox or through its API. This is a simple, economical and efficient solution capable of eliminating the problem of files transfer between hospitals and medical centers.

### **2.3.3 File Transfer using HTTP File Upload**

The HTTP file upload transfers can be done using our own implementation or using third party API's. The Apache file commons IO library offers an API to manage and handle file transfers. The File Upload from the referred API can be used in multiple ways, depending on the requirements of each user and application. The most common approaches are the all-in-one and the multipart approaches. In case of all-in-one, the file is processed in a unique and simple request, transferring the entire file between the source and destiny without separate the file in a list of smaller files. For every request, the entire file is processed one-time. The multipart approach is different. It automatically divides the file to be transferred in smaller files creating a list of files to be transferred. So, it cuts the file in smaller pieces and deals with all the process of file transferring. The user must define the max size of each file piece and the API does the rest of the work. While the file is being transferred, it is saved in a temporary folder. At the end, in case of successful transfer the file, all the file pieces are joined and stored at the destiny folder desired. This is a good option the use in this work, because it uses the port 80 and deals with file splitting and transferring issues.



# Chapter 3

## Technological Solutions

The technologies and frameworks studied in order to try to find a good solution to solve the problems of the proposal of this dissertation are presented in this chapter. The discussion about database storage technology, data security and web development frameworks are the main focus of this chapter.

### 3.1 LCM Integration

The LCM needs to be integrated in the application to be developed in this work. In order to not reinvent the wheel, technologies that allow the reutilization of the previous developed application should be considered. The next two subchapters present two possible solutions identified to deal with this process of code reutilization.

#### 3.1.1 Java Native Interface (JNI)

Java Native Interface is one framework that allows reutilization of code made in other languages such as C, C++ or Matlab. Consequently it is possible to map the functions developed in another programming environment into Java or vice-versa. This is done to allow having part of a system already developed and stable, working in a different language and therefore avoiding re-implementation of the whole system completely. Through the use of DLL's functionalities and JNI[7], it is possible to redefine the features we want to use. To achieve that, the process to follow is illustrated by Figure 3.1.

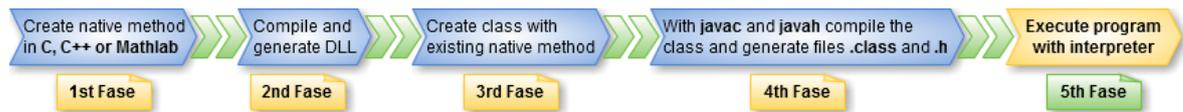


Figure 3.1 - JNI Lifecycle

At first instance of this process, all methods are built in their native language, for example C, C++ or Matlab. On a second phase it is required to compile the whole application or methods that we intent to use in Java afterwards, followed by the generation of a DLL with all compiled information. In the third phase, in Java, it is necessary to create the methods that we envisage to use in relation to the originally created methods in a different language. All of these methods must be native type to allow the correct interpretation by the compiler. In the last phase, once all desired and targeted classes and methods are built, originally in C, C++ or Matlab, it is necessary to compile the files to generate files with extension .class and extension .h that possess the headers required for the mapping to the native programming language methods.

### 3.1.2 Executable Files

Executable files are one of the possible solutions for the reutilization of code developed in programming languages like C, C + + or Matlab. For this case, executable files were created that offer the functionalities of the original application developed in Matlab. The working process follows a similar approach to all programs that run on Windows operating system environments. Whenever we want to get data from the application supported by the executable, a system call is made with some input parameters and the application will run in system background until it finishes processing the information required. In the end, the desired outputs are generated and the process that had been running in background is finished releasing the allocated memory and processor resources. This approach brings an advantage comparing with JNI, which is their portability for use in another language different from JAVA. It supports the possibility of calls made through the operating system. At Figure 3.2 the workflow of one system call from the client to the server is represented.

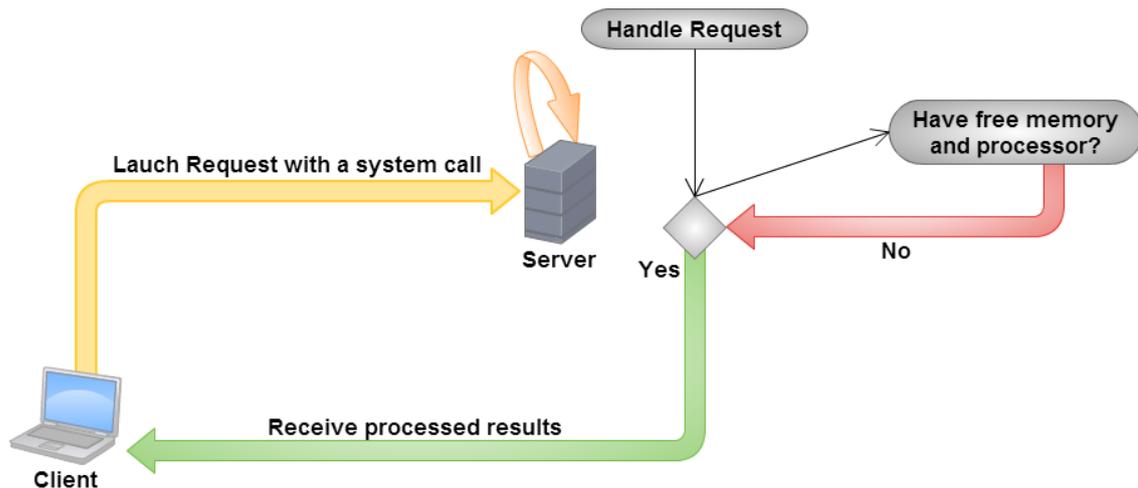


Figure 3.2 - Work Flow of the system call

Taking in consideration the portability offered, this technology was the chosen one to integrate the LCM in the web application. Two executable files were generated with the necessary functionalities to be used in the application that will be developed for this project. The first one, LCMEngine, is responsible for handling new study requests and for providing results based on automatic analysis of the audio free-field samples from patients. The second executable, LCMRefiner, is the tool to be used during the process of manual identification of cough events by a medical technician. This process reduces the error margin of cough event detection in each study by adapting the classification models according to the manual cough event identification performed by the medical technicians

## 3.2 Data Storage

The data storage system is one of the main issues to consider when designing an information system which uses web based platform. We should then evaluate diverse aspects to choose the correct database technology to be applied. In this particular case two different technologies were studied and considered, namely the traditional relational databases and the NoSQL databases, both using tools with free licenses. NoSQL databases represent the most recent database technology at present and it has a great worldwide expansion.

### 3.2.1 SQL Databases

The SQL relational databases are still the most common solution used for permanent data storage. When there is a well-known structure with minimal variance in how the data is stored, keeping data free from inconsistencies, this is the

technology to be considered to use. The relational databases have shown to their users that are very secure, fast and trustworthy when designed and used accordingly with a properly defined database model.

For this type of database technology there are multiple options such as the commercial licensed model of Microsoft SQL Server [8] and Oracle [9] or free license databases like MySQL [10]. From the range of technologies presented above only the free licensed databases technologies were considered, excluding this way the Microsoft SQL Server and Oracle and adopting the MySQL databases.

In some cases, the use of software with free license may bring some problems due to a possible lack of support or the discontinuance of applications. However, MySQL technology has proven to be good and reliable solution, with several improvements over the years both in terms of quality or reliability. It has been improving since first implemented, competing closely with commercial solutions such as Microsoft SQL Server. However, despite being a free, robust and reliable solution, it does not solve all the necessities required for this particular project. The database system to be used should be a model of variable data storage, being built over time with the most diverse information of studies, not having a fixed structure as the relational databases normally require for their proper functionality.

The indexing of metadata and documents is one of the requirements to be considered and the relational databases do not support them in an efficient way.

### 3.2.2 NoSQL Databases

NoSQL databases are the most recent database technology available and the number of companies that develop such database system has increased considerably, providing a wide range of products to the users. This type of database offers great scalability allowing having high sized registries and that they are distributed through machines all over the Internet. Their interconnection with the Cloud database technology is supported by the majority of the enterprises that offer Cloud storage. NoSQL databases are essentially divided into three distinct major groups:

- **Document-oriented:** Allows the data to be stored as documents, making it easy to export and import metadata. The MongoDB [11] and CouchDB [12] are two examples of NoSQL databases that work as Document-Based paradigm.
- **Key-value:** Allows data to be stored in distributed hashtables, storing the objects by keys. This method supports the search of objects through their

keys. Hirabi and MemcacheDB[13] are examples of technologies that use key-value format to store information.

- **Column-oriented:** Allow data from several fields to be stored in distinct columns, making the writing process very efficient. This model is similar to the relational model, because it needs a way of identification between columns which will create correspondences of between data. BigTale[14] and Cassandra[15] are two well-known examples of Column-oriented databases being used by Google and Facebook[16] respectively.

In this particular project, the metadata can be important and relevant data. To deal with that, the Document-Oriented databases were considered in order to explore the functionalities of data exportation and metadata. Table 3.1 shows a comparison of some aspects of MongoDB and CouchDB.

*Table 3.1- Simple comparison between MongoDB and CouchDB[17]*

	<b>MongoDB</b>	<b>CouchDB</b>
<b>Initial Release</b>	2009	2005
<b>Major Users</b>	Craigslist, Foursquare, Shutterfly, Intuit	LotsOfWords.com
<b>License Type</b>	AGPL, OpenSource	Apache, OpenSource
<b>Implementation Language</b>	C++	Erlang
<b>Example Use</b>	CMS system, comment storage, voting	CRM, CMS systems
<b>Best Use</b>	Dynamic queries, frequently written, rarely read statistical data	Accumulating, occasionally changing data with pre-defined queries
<b>Key Points</b>	Retains some properties of SQL such as query and index.	DB consistency, easy to use
<b>Concurrency Control</b>	Locks	MVCC
<b>Replication</b>	Async	Async
<b>Operative System</b>	Linux, OSX, Windows	Linux, OSX, Windows
<b>Characteristics</b>	Consistency, Partition Tolerance, Persistence	High Availability, Partition Tolerance, Persistence
<b>Data Model</b>	Document-Oriented (BSON)	Document-Oriented (JSON)
<b>Object Storage</b>	Database contains Collections	Database contains Documents

	Collections contain Documents	
<b>Query Method</b>	Map/Reduce (javascript) creating collections + Object-Based query language	Map/Reduce (javascript + others) creating View + Range queries

After evaluating, testing and comparing the features from MongoDB and CouchDB, it was decided to use the MongoDB to do this work. Both of these databases are good possible solutions for this work, but after testing and exploring the functionalities of both databases, the MongoDB offers more functionalities to the users. The way to interact programmatically with the database is easier, comparing to CouchDB. The popularity between these two solutions, gives advantage to MongoDB too. This is one of the most used NoSQL database technologies at present. The next section describes this database technology in more detail.

### 3.2.3 MongoDB

Created in 2009, the MongoDB NoSQL database technology was thought and developed for documents-oriented databases in a C++ programming. It is very commonly used at present for its much appreciated features within the programmer's community. The main characteristics are the scaling which allows the change-over from a single server system to more complex systems with multiple servers with distributed information. It is possible to create indexing similar to relational databases which accelerate the documents search.

Instead of storing data in rows and columns as relational database, MongoDB stores a binary form of JSON documents (BSON) which makes it easier to read, navigate and export to other data formats. MongoDB doesn't impose flat or rigid schemas across many tables. It's an agile database that allows schemas to vary across documents and to change quickly as applications evolve [18].

This technology comes with drivers compatible with the most popular programming languages becoming attractive to all programmers. Being one of the most used NoSQL databases it also comes available in the majority of the cloud storage systems having this way a broader range of platforms that support its usage. This database technology was the chosen to be studied with more intensity between the two document-oriented databases referred in Table 3.1. It is also the chosen database technology to be used in this project.

Data in MongoDB are stored in documents, which in turn are organized in collections. The collections are physically created as soon as the first document is created in it. Documents are not identified by a simple ID, but by an object identifier type, optimized for storage and indexing. It uses machine identifier, timestamp and process id to be reasonably unique. That's the default approach, but the user is free to assign any other value he wishes as a document's ID. Binary data is serialized in little-endian. Every document gets a default index on the id attribute, which also enforces uniqueness. However it is possible and recommended to index any other attributes thought by the programmer to be queried or sorted, in order to speed up the processes of searching. Indexes can also be created on multiple attributes, additionally specifying a sort order like in relational databases.

On the next code board, is an example of the structure of a MongoDB document. Like referred before, it follows a similar approach to JSON files with the same hierarchical organization but in binary files (BSON).

```
{
  "id": "51daf9156a2eb8d5fae91ad8",
  "name": "Daniel Dias",
  "number": "123456789",
  "birthdate": "29/05/1986",
  "city": "Aveiro"
}
```

### 3.2.4 JPA vs Native Driver

For the project in hands the Java Persistence API (JPA) [19] technology was considered to be a possible solution to be used. JPA is a Java API to deal with data persistence, which facilitates the way connection and communication with databases is made. This technology was designed and created to make the access and communication with databases independent of the database technology used. It can be used with Plain Old Java Object (POJO) objects in order to interact with the J2EE beans or with traditional relational databases models.

Many libraries allow and support usage of JPA or similar technologies with the same theoretical and practical concepts to interact with databases. The most used to interact with MongoDB are EclipseLink [20], DataNucleus [21], Morphia [22],

Kundera [23] and SpringMongoDB [24]. One of the main advantages of using JPA is the possible portability to another database technology in a simple manner. Another good advantage is the consistent programming style and architecture used in all JPA systems. It is independent from the database being used, which makes it easy to use without needing to know too many details about the original database implementations. Despite this approach sometimes not having all the necessary functions to interact with database, it supports extension and overriding to the existing implementation turning possible to expand the original available solutions of JPA.

The native driver offers all the possible interactions to each database, turning possible optimizations in the database assessments and insertions. In terms of performance, it can have a better performance comparing with JPA. However this is not linear, because it depends on the programmers implementations.

Both approaches were studied and tried in the beginning phase of the project and both of them could be adapted to this project. Although JPA may have some minimal loss in performance compared to the original driver, JPA is compensated with methods already studied and tested with the best possible performance. It has also a good advantage that consists in possibility of portability to another database without necessity of changing the entire system. In case of using a native driver, if the system needs to be changed to another database system it will bring a lot of changes over all the entire application. The performance is always important in any application, but evaluating the pros and cons of both solutions, it appears that JPA approach brings more benefits compared with native driver having in mind possible application changes or future expansions.

### **3.3 Web interface framework**

The Primefaces [25] framework was selected for implementing the web interface of this application. Primefaces is an open-source framework for Web environment programming of JavaServer Faces providing a wide group of components with diverse features such as dynamic tables, charts, dynamic menus, etc. It is thought and designed for an event-oriented approach such as the majority of the Web applications. This framework uses an MVC architecture and it groups all the required components and features regarding the web interface necessary to build this project. Components in Primefaces were developed with the design principle that *“A good UI component should hide complexity but keep the flexibility”*. According to numbers from devrates.com and from the Primefaces official website, this framework takes the lead

as the developers' favorite framework to create rich user web interfaces[25] as shown in Figure 3.3.



Figure 3.3 - Web Frameworks popularity

Even comparing this framework with his JSF framework competitors, according to data from Primefaces website, Primefaces is the current market leader for JSF framework development [25, 26] as shown in Figure 3.4.

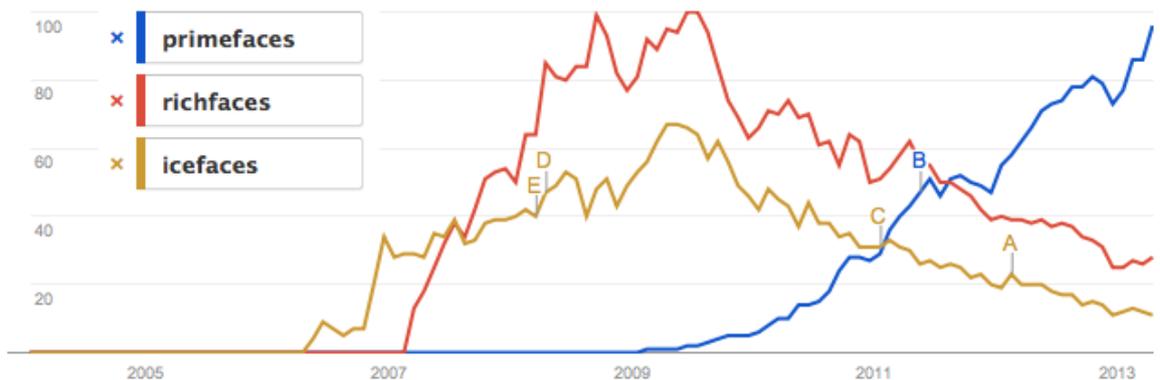


Figure 3.4 - JSF Framework popularity

### 3.4 Users Management with RBAC policy

The policy for users' management based on Role Based Access Control enables, in a simple and organized manner, to define which areas of the application are accessible to each role. In this application it is required to have several access levels for all users.

Taking in consideration that in the future, new features may be implemented and added to the application associated with distinct access levels, this is a possible way to deal with users information access control. Considering the MongoDB database technology and Primefaces framework, two possible solutions for access control

based on roles were studied, namely Spring Security [27] and Primefaces EL Extensions that allow the control of the contents accessible for each role type at the web interface level.

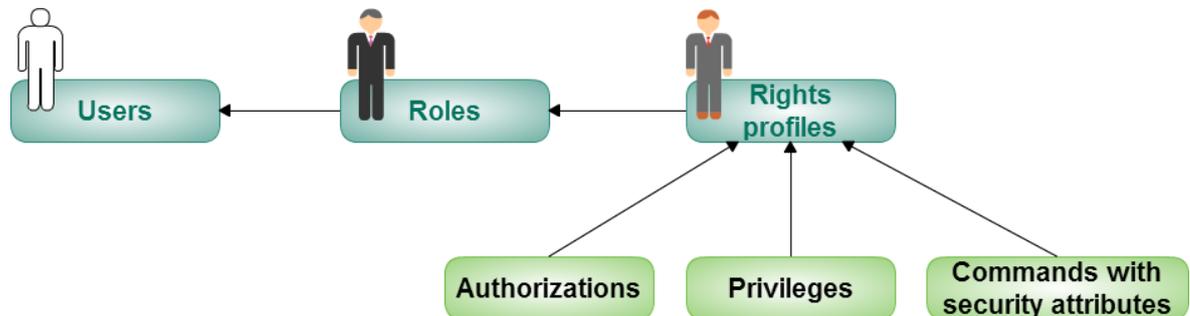


Figure 3.5 - RBAC System

In RBAC, access to a resource by a user is permitted only if all of the following conditions are met:

- the permission required for access to the resource is assigned to a role;
- the role is assigned to the user requesting access to the resource;
- the role is activated in the user's session.

A role may be used to give permissions or grant access to a set of operations or data associated with that role. It has users associated, and each user can have more than one role. One of the greatest advantages of RBAC is that it allows the access privileges provided to individuals to be reconfigured. For example, every time users change their job requirements or company position it is easy to change their roles. This can be simply made by deleting one's original assignment to a first role and adding one to the new role [28].

### 3.4.1 Spring Security Framework

Spring Security is a highly customizable authentication and access-control framework. Although initially thought only for Spring projects, it was rapidly expanded to be used in projects which are not native Spring projects, such as Java/J2EE projects for example. It can be applied on any Java project and database. If we are dealing with relational databases, which are generally the most used in the vast majority of projects, all the necessary settings to make the connection between the application and the database using spring security are simple and available. These settings are made through XML configuration files. In case of non-relational databases

such as MongoDB it is necessary to make our own implementation to support the link between documents that contain users, their roles and SpringSecurity.

This framework enables the validation and authentication at the web page as a whole, or at the form level. With form level, it is possible to give access to certain content only to users or user levels as desired. The main settings are made at the level of XML configuration files. The authentication flow of Spring Security follows the flux represented in Figure 3.6.

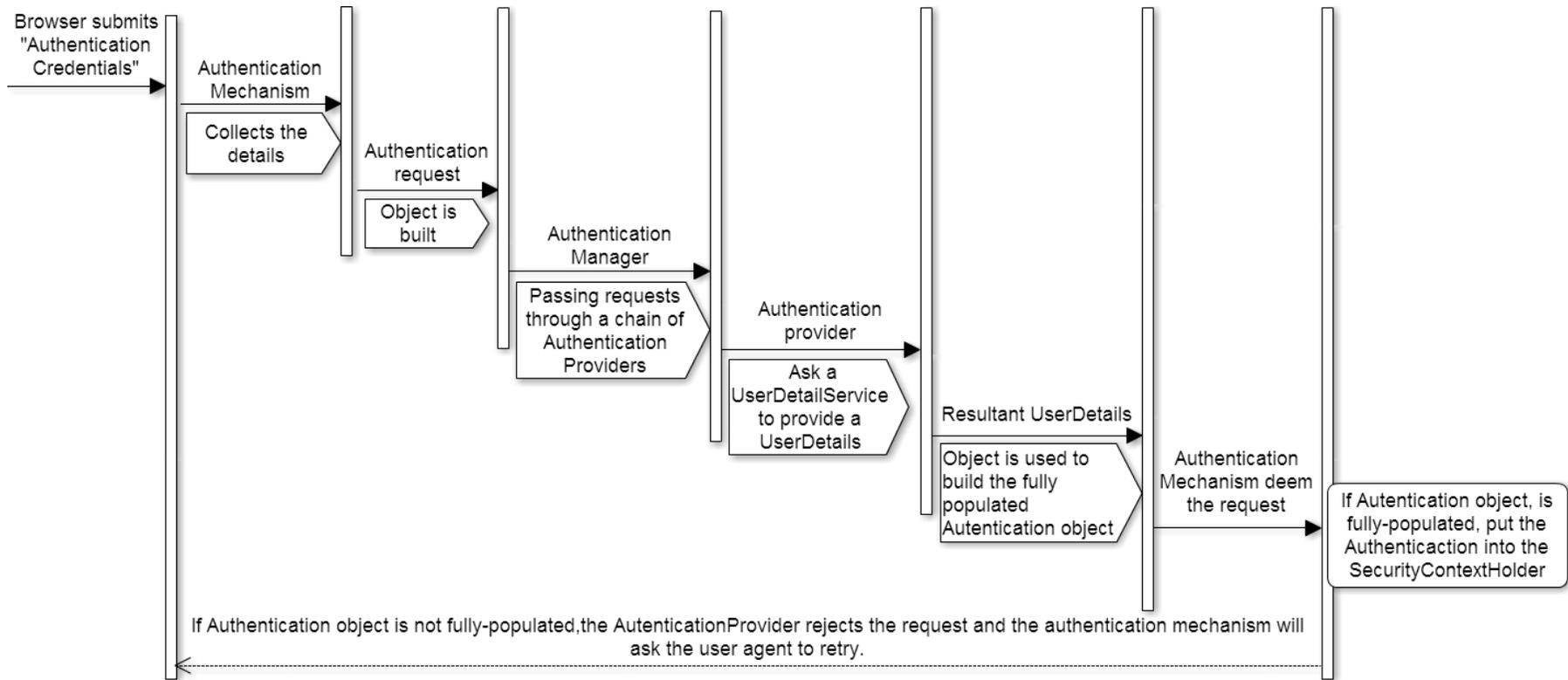


Figure 3.6 - Spring Security Work Flow

## Chapter 4

# System Implementation

This chapter details the development phase of the designed system. This description is accompanied by the engineering solutions adopted, and how they were used. For the implementation phase, JAVA programming language was the chosen language and Primefaces JSF was the framework to meet the need of the graphical application. With Java, it is possible to guarantee the independence of the application regarding operating systems or infrastructure in the Cloud. MongoDB was the chosen database to store data objects in documents and Glassfish 3.1.2 the application server used to deploy the multiple versions of the application in all the development process.

Using Java as a development tool it is possible to create multiplatform applications that can run independently of the operative system in which they are deployed, such as Windows or Linux. This way, any system manager can easily configure the application, leaving it ready for distribution in any production environment.

After evaluating technological solutions in Chapter 3 of this work, some technologies were chosen to fill the needs of the application in multiple fields. For security and authentication issues, the Spring Security framework was chosen to ensure safety in the forms of application access with a role based policy. For database storage, MongoDB NoSQL database was the chosen technology.

The development phase had two distinct and essential moments to the development of the application. As a first stage the structure was developed based on the application with access to the database and implementation of security. The second phase was the integration of LCM technology with the web application.

The NetBeans IDE, was used during all entire project development. However, in order to facilitate the transition between developments environments dealing with all dependencies associated, the project was created using the Maven tool.

## 4.1 Data Storage

In the previous chapter of technological solutions, MongoDB was the studied database and the chosen technology to use in this project. It has a document-oriented database philosophy and it makes possible a future integration with Solr or Lucene to index some metadata from MongoDB documents.

Spring Data is a Spring API that aims to facilitate the process to deal with databases communication. It provides generic methods to interact with the most common operations of relational databases such as insert, delete and query operation through JPA style. However, this technology was extended to be used with some of the most popular non-relational databases using similar approaches to JPA, but adapted to each database type. This technology has an implementation for MongoDB, named Spring Data MongoDB that was the solution used in this project.

The Spring Data MongoDB provides integration with the MongoDB document databases. The key functional areas of Spring Data MongoDB are the POJO centric model for interacting with MongoDB collections and the easy way of writing a Repository style data access layer.

The Repository style intents to provide easy and automatic implementations for the custom finder methods to query the databases. The process is simple, it is only necessary to implement the repository interface classes of Spring. Once there is the SpringDataMongo behind, it provides access to the finder methods already implemented by Spring. The programmer only needs to call those methods, or override it to create more personalized implementations.

Spring Data Mongo, has also the MongoTemplate class that increases productivity performing the common Mongo operations. It has included an integrated object mapping between documents and POJOs. MongoTemplate methods have the native Java driver operations running in their background. Thus, although it does not use the native driver in a direct manner, it is used as one superior code layer with all the main methods already implemented, studied and validated by Spring developers to interact with databases.

This approach requires less code comparing with native implementations, because this technology has already implemented most of the common methods to deal and interact with databases. The use of the SpringRepositories together with MongoTemplates is a powerful tool to deal with database management and assessments using Java and MongoDB.

Despite the native MongoDB driver for Java/J2EE approach had been tried initially, with good results in terms of efficiency, the Spring Data MongoDB approach with MongoTemplates brings enormous benefits. The Spring Data MongoDB is simpler and offers a safe and good mapping between the domain objects and MongoDB documents. It also offers an easy way to use operations such as create, update, delete and queries MongoDB documents.

One of the biggest advantages to take in consideration is the possibility of database portability to another database technology as long as it is supported by Spring. In this case, the changes to take in consideration are minimal, since that it is only necessary to adapt some parameters of interaction with databases according with the technology that is being used.

MongoDBTemplate provides a good way to make searching processes through the implementation of multiple Criteria system technology. After parameterizing the criteria, the database queries can be easily made. The Criteria is an API from JPA for constructing queries that standardizes many of the programming features that exists in proprietary persistence products. More than just a literal translation of Java Persistence Query Language (JPQL) to programming interface, it also adopts programming best practices of proprietary models, such as method chaining, and full use of the Java programming language features [29].

The process of making searches with multi criteria method is simple. It is necessary to have a class responsible to fill and store the object with all the fields we want to be used on possible searches. For example, if we want to have one search process to find patients based on their name, number and city, one *PatientSearchCriteria* must be created with these three fields and having respective getters and setters. The fields to be used in these searches can be filled on JSF pages through the javabeans. After the user fills the form, a generic dynamic query will be created and criteria parameters will be added as inputs to the query. There is the possibility to use regex expressions to increase the range of options to get more detailed queries. After having the query filled with the criteria parameters, one of the methods offered by MongoTemplate can be used to proceed with searches. The query

to be done and the class to be queried will be the inputs of the MongoTemplate finder methods.

The searching criteria process has an implementation similar to the one at bottom code board.

```

public List<Patient> searchByCriteria(PatientSearchCriteria criteria) {
    Query query = new Query();
    if (StringUtils.hasText(criteria.getName())) {
        Criteria c = Criteria.where("name").regex(".*" + criteria.getName() + ".*", "i");
        query.addCriteria(c);
    }
    if (StringUtils.hasText(criteria.getNumber())) {
        Criteria c = Criteria.where("number").is(criteria.getNumber());
        query.addCriteria(c);
    }
    if (StringUtils.hasText(criteria.getCity())) {
        Criteria c = Criteria.where("city").regex(".*" + criteria.getCity() + ".*", "i");
        query.addCriteria(c);
    }
    return mongoTemplate.find(query, Patient.class);
}

```

The architecture of the database created for this project is shown in Figure 4.1.

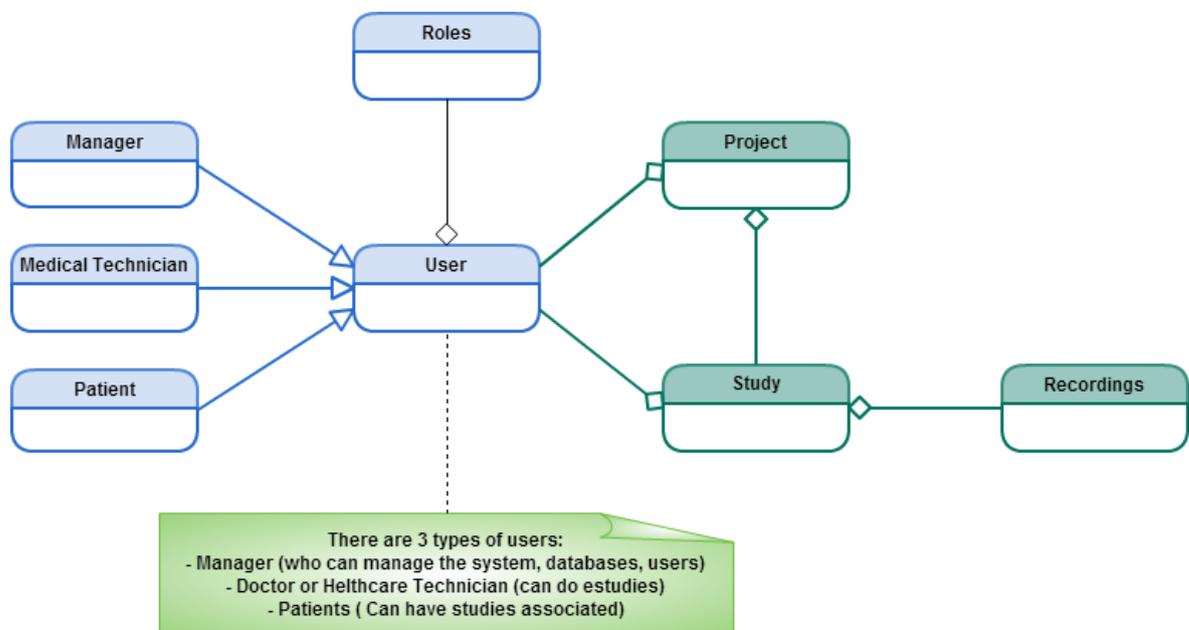


Figure 4.1 - Database simple architecture

The database stores objects as documents in BSON files. The documents can grow with time, with details associated to each study. The database can be divided in two main areas. The user and authentication section and the studies section, being both connected. With exception to patients, that are singular persons from who studies are created, each user has one or more roles associated to work with the application. Each study has patients and medical technicians associated and each project can include a group of studies. Besides patients and medical technicians, each study has audio recordings and some comments by the medical technician about the study or information about prescribed medicines.

## 4.2 Application structure

The web interface of the application was developed using the Java Server Faces technology. This tool is an MVC framework especially designed to build web interfaces. Taking advantage of a MVC model it is possible to develop the logic and the application interface in completely independent ways. According to Table 4.1 it is possible to understand which resources constitute an application that follows this model.

*Table 4.1 - MVC Concepts*

<b>Concept</b>	<b>Description</b>
View	The JSF pages which have JSF components are known as the GUI of the MVC model.
Controller	The Faces Servlet accompanying the implementation of the JSF is responsible for managing all requests for JSF pages. This entity carries the appropriate views and interacts with the model to process each response.
Model	The ManagedBeans correspond to the model of the application. These entities are JavaBeans whose state is managed by the JSF framework.

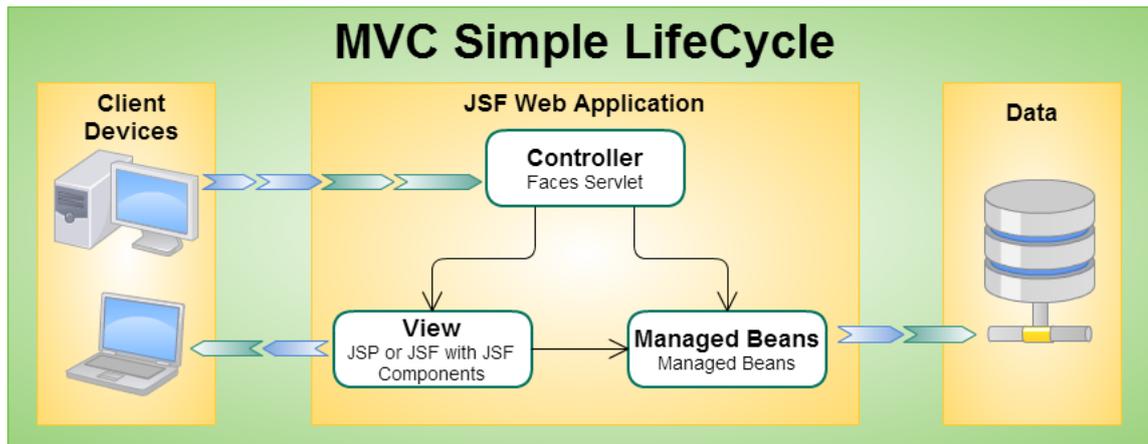


Figure 4.2- MVC Simple LifeCycle

All the application follows one Spring Repository philosophy to organize the code. So the application has five main packages being them domain, repository, service, views and LCMEngine.

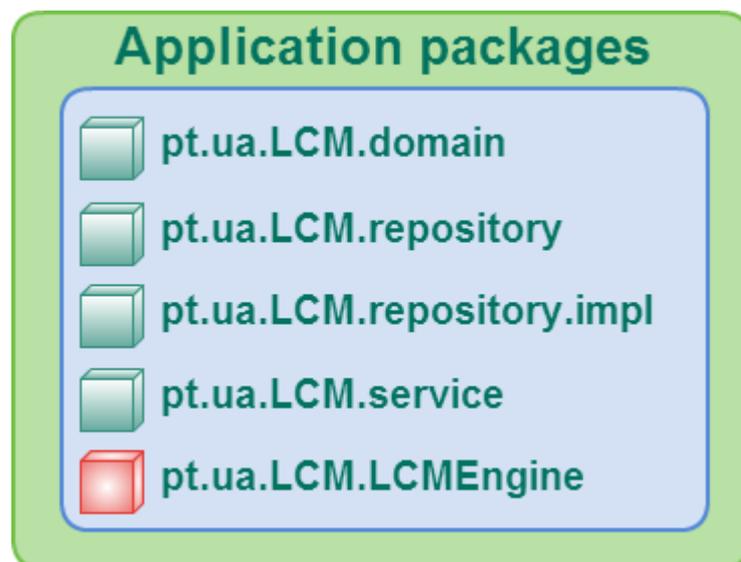


Figure 4.3 - Application Packages

At the domain package there are only normal classes that are constructed based on primitive data types such as Patients, Users, Studies with their getters and setters.

The repository package is divided in two sub-packages. At the first one, are the interfaces that interact with the database extending features from Spring, which allows the use of CRUD operations in an easy way. There is also the function prototypes to interact with database, having more specific and personalized options as searches based on Spring criteria. The second one has the implementation of the previous defined interfaces. In this case, it uses the multi-criteria system from Spring

criteria to allow searching the distinct objects with combined or single parameters on queries. The multi-criteria search uses MongoTemplate functions to interact with the database.

The Service package is responsible for using the repositories packages function, i.e. through the repositories previously declared where is now possible to search the database and realize CRUD operations. These classes use the @Service annotation from Spring to be recognized as a service in the application.

The view package uses all the previous packages to make the interaction between the JSF client and the java server through javabeans.

The LCMEngine package has all the necessary implementations to deal with LCMEngine tools. This package is also responsible to interact with JSON files in order to manipulate the results from studies created by the LCMEngine tool, as described below.

### 4.3 User Security and Authentication Management

As mentioned Spring Security framework, was the chosen technology to deal with user authentication process in the application. Although the chosen database was MongoDB which doesn't have an immediate integration with Spring Security. Thus, as we are not dealing with the native approach of Spring Security that was developed for the relational databases and for simple JAVA POJO's, it is necessary to make some changes in order to allow to with other database environments. The implementation of Spring Security together with MongoDB follows in the next set of steps.

The first thing to do is defining the filter chains with role access levels in the application, i.e., it is necessary to stipulate which contents are accessible by each user level. A springsecurity.xml file is created to achieve the above, that stores SpringSecurity configurations. The file is added to the web.xml as a context-param entry with the path to the springsecurity.xml. Then, contents that define the areas accessibility to each user role level are saved as filter chains. These configurations are similar to the ones illustrated by the next code board.

```
<http use-expressions="true">
  <intercept-url pattern="/Manager/**" access="hasRole('Manager')" />
  <intercept-url pattern="/Study/**" access="hasRole('Technician','Manager',)" />
      ....
      ....
  <intercept-url pattern="/Patient/**" access="hasRole('Technician','Manager', 'User')"
/>
```

```
<form-login default-target-url="/index.xhtml" always-use-default-target="true" login-  
page="/login.xhtml"/>  
<logout delete-cookies="JSESSIONID"/>  
</http>
```

Having the filter chains configured, it is necessary to know where to check users to make the user authentication in the application. The SpringSecurity class responsible to deal with user authentication is the UserDetailsService. If this class were not re-implemented, the system will check an xml configuration file with hardcoded users with their passwords and roles. In this project it is desirable to have all the contents in the MongoDB database, even the system users. To do that, it is required to implement the UserDetailsService with our own method to deal with user authentication. This implementation has a set of methods similar to the ones referred on the next code board. We achieve this by associating the system user class with a Spring user and the loadUserByUsername method is responsible to make that association. The GrantedAuthority is the list of allowed role levels recognized in the context of SpringSecurity.

```
public class CustomUserDetailsService implements UserDetailsService {  
  
    private MongoTemplate mongoTemplate;  
  
    public UserDetails loadUserByUsername(String username)  
        throws UsernameNotFoundException {  
        User user = getUserDetail(username);  
        org.springframework.security.core.userdetails.User userDetails = new  
org.springframework.security.core.userdetails.User(user.getUsername(), use  
r.getPassword(), true, true, true, true, getAuthorities(user.getRole()));  
        return userDetails;  
    }  
  
    @Autowired  
    public void setMongoTemplate(MongoTemplate mongoTemplate) {  
        this.mongoTemplate = mongoTemplate;  
    }  
  
    public List<GrantedAuthority> getAuthorities(Integer role) {  
        List<GrantedAuthority> authList = new ArrayList  
<GrantedAuthority>();  
        if (role.intValue() == 1) {  
            authList.add(new SimpleGrantedAuthority("Manager"));  
            authList.add(new SimpleGrantedAuthority("Technician"));  
        } else if (role.intValue() == 2) {  
            authList.add(new SimpleGrantedAuthority("Manager"));  
        } else if (role.intValue() == 3) {  
            authList.add(new SimpleGrantedAuthority("User"));  
            authList.add(new SimpleGrantedAuthority("Technician"));  
        } else if (role.intValue() == 4) {  
            authList.add(new SimpleGrantedAuthority("User"));  
        }  
    }  
}
```

```

    }
    return authList;
}

public User getUserDetail(String username){
    MongoOperations mongoOperation = (MongoOperations)mongoTemplate;
    User user = mongoOperation.findOne(
        new Query(Criteria.where("username").is(username)),
        User.class);
    return user;
}
}
}

```

The user roles are considered authorities in the Spring applications context. So the list of authorities of the system is saved at the Granted Authorities list.

After that, we have to map the path with the class that will deal with the user authentication in the system on the springsecurity.xml file. The configuration desired is similar to the referred in the next code board. The last step is to make a webpage for user authentication.

```

<beans:bean id="customUserDetailsService"
class="pt.ua.LCM.domain.CustomUserDetailsService"/>

<authentication-manager>
<authentication-provider user-service-ref="customUserDetailsService"/>
</authentication-manager>

```

## 4.4 File Management

One of the concerns to take in consideration in this project was the limitation of the ports that can be used to make the file transfers between application clients and server. The file transfer is a complex and sensitive part of the application to take into consideration. Although, after research for some technologies, it was decided to use the Primefaces file upload technology that provides a multi-part transfer system. Internally it creates multiple chunk files and sends them through the browser using the port 80. The Primefaces file upload technology is implemented using the apache commons-fileupload and commons-io libraries.

In order to make easier the process of folder identification from each study, the Universally Unique Identifier (UUID) library is used to generate a random unique value inside of the application scope. This value combined with a fixed temporary folder, is used as name of the study folder location with uploaded files. This process makes the identification and location from each study simple and effective.

After the application processes the audio files, the output values are stored in another folder with a different location from the temporary folder. This location is composed by a default path joined with the previous generated UUID making it unique in the entire data repository. The use of generated UUID's makes easy to create a good and robust relation between the locations of the processed files and files to be processed. The application keeps in the storage repository the original audio files and the processed results. This makes possible to run multiple studies from the same original audio files, and successive refinements of the results. However if the necessity of cleaning oldest original audio files emerges, for lack of space, the original audio files can be discarded. The only ones that need to be kept in the system are the processed files that make possible to get results from the studies.

## 4.5 Application Dependencies

The developed application depends on several external frameworks. In order to facilitate the integration of libraries and to export the application to other environments, a Maven project has been created which hosts all application dependencies. Figure 4.4 shows the main application libraries needed to this project declared in the Maven pom.xml configuration file. The dependencies are grouped into four major categories. The category shown in green depicts the libraries responsible for the web interface of the application and the interaction with the application functionality through the beans of Java Server Faces. The one in purple represents the libraries responsible for the interaction with the MongoDB database through its native driver for Java and integration with Spring which facilitates and speeds up the process of insertion and searching at the database. The libraries shown in gray are responsible for the processing of multiple files handled by the application. The *json-lib* library deals with creating, reading and changing JSON files containing the results of studies processed by LCMEngine, and *commons-fileupload* deals with uploading process of the audio files to the application studies. The yellow category depicts the libraries of the various components used in the application security process at the safety management and authentication of users in a role based policy.

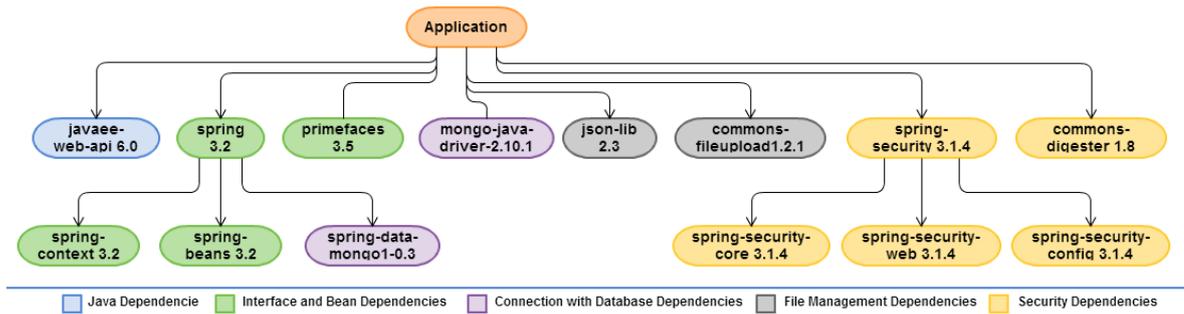


Figure 4.4 - Application Dependencies at pom.xml

## 4.6 JSF Client Server with Ajax

Some years ago, the first web applications were developed following approaches with static contents, in other words, to make some actualization at web clients, at each request every page needed to be fully reloaded. This approach works, but if the application needs to make lot of requests it will compromise the performance of the application. So, AJAX emerged to solve this problem and with a couple of technologies it provides solutions to develop dynamic applications, that respond to certain situations asynchronously without need to refresh or update all the application content with requests to the server. With AJAX, the web applications can share information between server and client in a highly responsive and efficient way. The JSF framework has AJAX technology fully integrated with its components, and it has two distinct ways to deal with components state. It can save the components state at the client or at the server. At the client approach, the state persists on the client side and is serialized, encoded at base64 and will remain at an HTML-input-hidden field on the generated page. At the server approach, the state persists at user session on the server. Both approaches have advantages and disadvantages referred at Table 4.2. The configuration is made and parameterized on the server on the web.xml file and it is shown at bottom code board. Changing the approach from client-side to server-side is easy, requiring only changing the param-value from client to server or vice-versa.

```

<context-param>
  <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
  <param-value>client</param-value>
</context-param>

```

Table 4.2 - AJAX Advantages and Disadvantages

<b>Server Side</b>	
<b>Advantages</b>	<b>Disadvantages</b>
Low network bandwidth needed	There is a higher memory consumption on the server because all states are being kept in session per user
Low memory consumption on the client	Problems with the back button of the browser (requires more memory consumption to solve this problem at server side)
Low CPU usage on the server	Move the session to another node in the application (many data in the session);
Improves communication via AJAX because there is no need to resend the view state to the server	
<b>Client Side</b>	
<b>Advantages</b>	<b>Disadvantages</b>
Less use of session and consequently less memory consumption on the server side	Increased memory consumption on the client side;
Without consuming memory between requests	High-bandwidth network consumption
Corrects refresh pages problems and back button keeping state at client	Depending on the complexity and quantity of components, the pages can load a little bit slower
Improves communication via AJAX because there is no need to resend the view state to the server	
Resolve concurrency problems	
Improves the application scalability	

After evaluating the advantages and disadvantages, both approaches are possible solutions to use in this project, but it was decided to follow the client-side approach. The views of the same page have a large number of components and some data need to be kept in memory at the client's side to deal with constant navigation between pages and the necessity of keeping the component's status. This relieves the server memory usage and improves the scalability of the application giving the option to

have more clients at the same time connected to the application. AJAX improved a lot the performance and decreased the requests to the server to get data into page views.

## **4.7 Web Application**

During the development of this work, a combination of several technologies studied was made. The various aspects related to requirements were taken into account trying always to find the best possible solution for each case. As a final result, a Spring MVC model was used combined with JSF and Primefaces libraries. The GUI of the application combines a standard Primefaces template with some personal customizations and changes in the template modules location.

As the initial contact with the system, the user needs to complete his login so he can access the contents of the application. Triggered the login process, the user is redirected to a landing page that will allow him to browse all the contents on the application to which his role will give him access.

In this landing page, the user has two main areas in the application that are the same as those designed in the prototype mockups. In the left side of the application are located the menus for various application features. In the central part is where all the data are shown after clicking on the menu links. In Figure 4.5 is possible to see the interface where the user can search patients on the application.

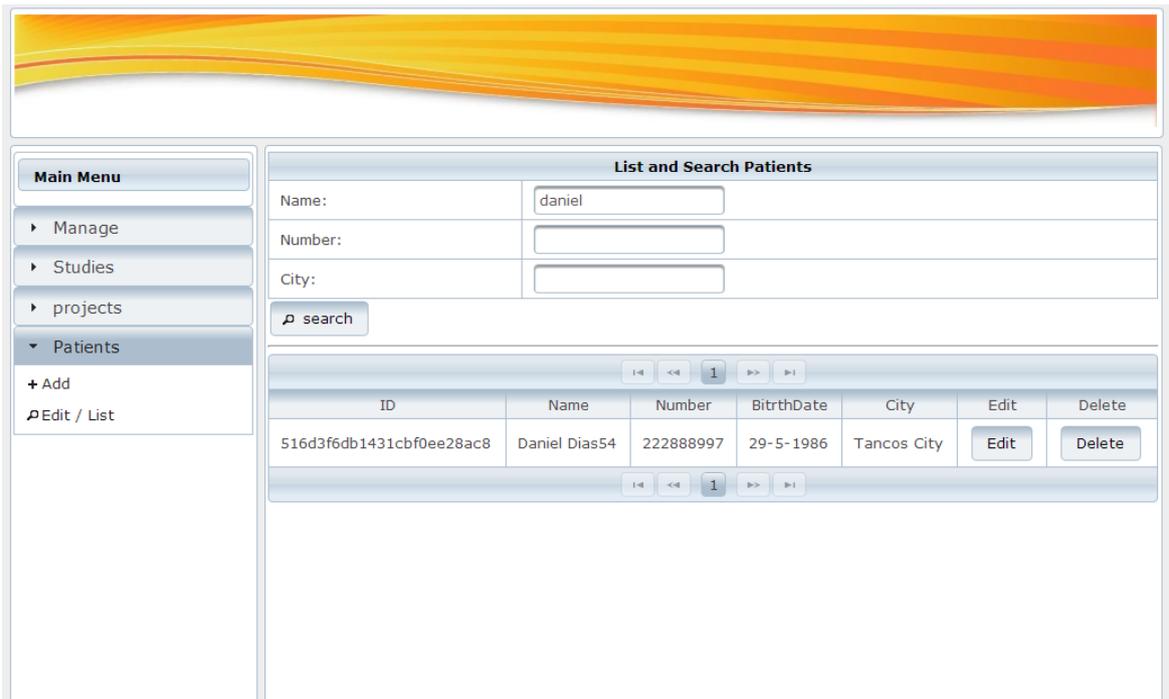


Figure 4.5 - Example of the interface

### 4.7.1 User Management

The user management is a section from the application that only users with role manager are allowed to have access. Once one of the managers is logged in, he will have access to the Manager menu and is able to create and manage users, including himself. The interface of the user management follows one of the standard approaches used by other systems of user management. The manager creates new users filling all the form associating respective roles according to the necessities and capacities. Then, sends the credentials to the user that will use the application. The Figure 4.6 shows the simple form where managers can create users.

Figure 4.6 - User Creation Form

### 4.7.2 LCM Data Processing

The LCMEngine is the main core of the application and everything revolves around the LCM. Every time a user wants to create a new study at the application, after a valid login session, they should go to the menu navigation and choose the option to create a new study as shown in Figure 4.7.

Figure 4.7 -Create New Study Form

The user must fill all data in the form and upload a file with the free-field sound recording from one patient and then submit the new study. After successfully submitting the study, a new LCMEngine process is launched at the webserver, running in the background until all data has been processed by the process of automatic cough event detection. This process is formed by a set of steps described below in Figure 4.8.



Figure 4.8 - LCMEngine Processing

Once LCMEngine tool starts processing the audio file, it will detect candidate cough events using HMM models, and for each detected event one small audio .wav file is created with the corresponding piece of sound. This process continues until all recording is processed. At the end, a json file is generated with fields that identify the detected events with an association to the time instant that the event occurs. After this process finishes, the study is marked as processed and is ready to be analyzed by a technician. Some of the cough events detected by the LCMEngine will be listened and classified by the technician together with an automated process (LCMRefiner) that refines the models according with the user classification inputs.

The process of refinement with the manual analysis, together with automated adjustments of the models by the LCMRefiner, starts with the classification of at least 20 events. The medical technician needs to listen and identify these 20 events as cough events or not cough and then, an automated system call is made in order to adjust and refine the models to discard previous false positives detected by the LCMEngine. This process is made in a screen with an interaction like the one shown in Figure 4.9.

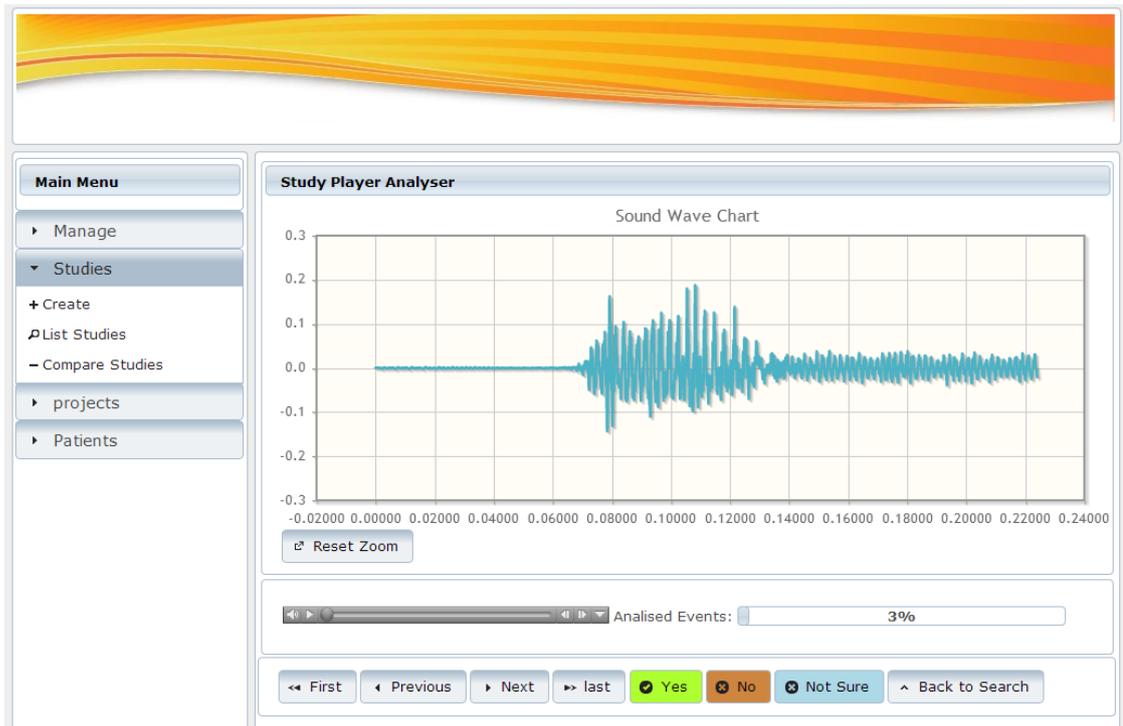


Figure 4.9 - Classify events as Cough

On this classification screen, the user can see a chart with the sound waveform of each candidate event, listen to the sound in an embedded player on the browser and should then based on his perception of the sound, classify the event as cough, not cough or unsure.

After LCMRefiner processes the given inputs of cough event classification by the medical technician, the fields in the json file are updated. The *iscoughevent* field will receive a classification as cough or not based on calculations at the user inputs and the updating models process of the LCMRefiner. The *dist* field will receive a value that needs be checked every time LCMRefiner runs. If this field is less or equal to a certain pre-defined value the study is considered to be sufficiently processed and analyzed. The sorting field will also be updated every time the LCMRefiner runs, sorting the candidate events starting with the ones more difficult to automatically classify as cough or not cough. Figure 4.10 illustrates the automated process of the LCMRefiner. The fields that compose the json file are described in Table 4.3.

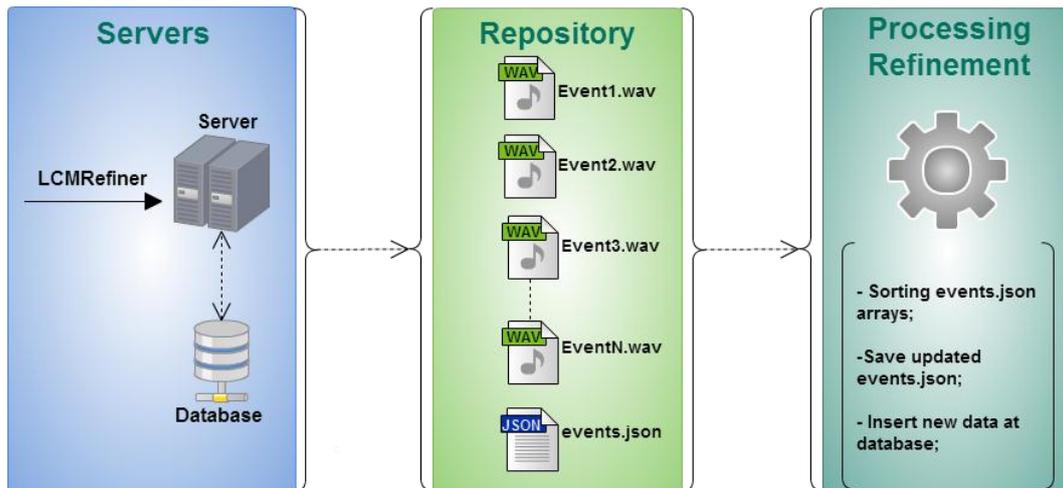


Figure 4.10 - LCMRefiner Processing

Table 4.3 - Events.json fields from LCMEngine

Events.json Fields	Description
Count	Represents the total number of events detected by the LCMEngine.
Time	Array with the start time of each detected event, counted in seconds from the start of the recording.
Duration	Array with the duration of each detected event, in seconds.
Reviewed**	Array that marks the events listened by the Medical Technician.
Classified**	Array that saves the cough event classification by the Medical Technician. The value is 1 if the event is marked as cough, -1 if is marked as not cough and 0 if is classified as unsure.
Sorting *	Array with cough events in the order they will be shown to the Medical Technician. After each LCMRefiner step the models are updated according to the user inputs and this sorting order is changed.
Refined*	This field is 1 if the study has already been analyzed and refined and 0 if the refinement step is not yet done.
Dist *	Represents the precision of the event analysis. If this value is less than a pre-defined value the Medical Technician can stop making refinements.
Iscoughevent *	Array with the final classification for each cough event. This value is calculated based on the user classification inputs and the LCMRefiner process.

The fields marked with \* in the table are updated every time the LCMRefiner runs. The fields marked with \*\* are updated on every cough event analysis by the Medical Technician.

After the first step of refinement is completed, it will continue with the same philosophy as the beginning. But now, the medical technician only needs to classify 10 events in each iteration. This iterative process will be repeated with the same process of the first step, until the stopping condition defined by the LCMRefiner tool is met. After the study is finished, the results are available to be consulted and the medical technician can write some notes in comments on each study. These comments can be useful for example if a patient is being studied for some time and to check his evolution checking old analysis and comments. In Figure 4.11 can be seen a list of studies in the application and in Figure 4.12 can be seen the screen where the technician can make comments about the studies.

**Main Menu**

- Manage
- Studies
  - Create
  - List Studies
  - Compare Studies
- Project
- Patients

**List and Search Studies**

Patient Name:

**Search Studies**

	Name	Number	Date Submitted	Processed	Re-Run Study	Delete
1	Kyle	333444333	23-7-2013 17:27	false	<input type="button" value="Launch"/>	<input type="button" value="Delete"/>
2	Kyle	333444333	23-7-2013 16:40	true	<input type="button" value="Launch"/>	<input type="button" value="Delete"/>

Figure 4.11- List of Studies

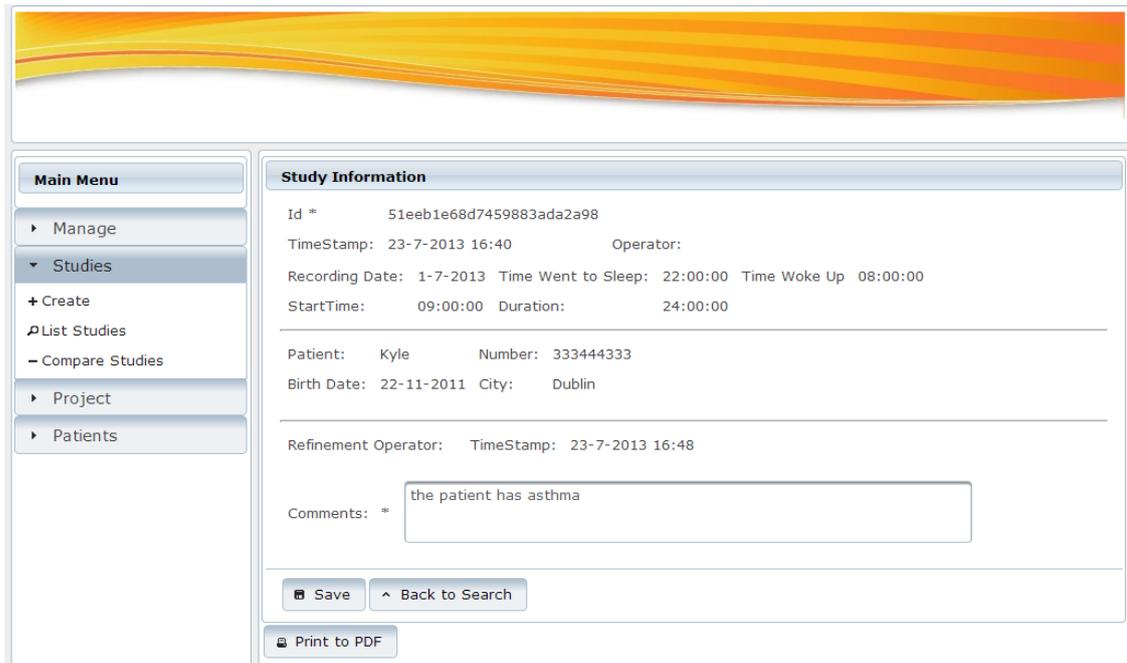


Figure 4.12- Study Comments

The results can be viewed in charts containing the occurrences of cough events per hour. Figure 4.13 illustrates an example of a processed and final result of the number of cough events detected per hour.

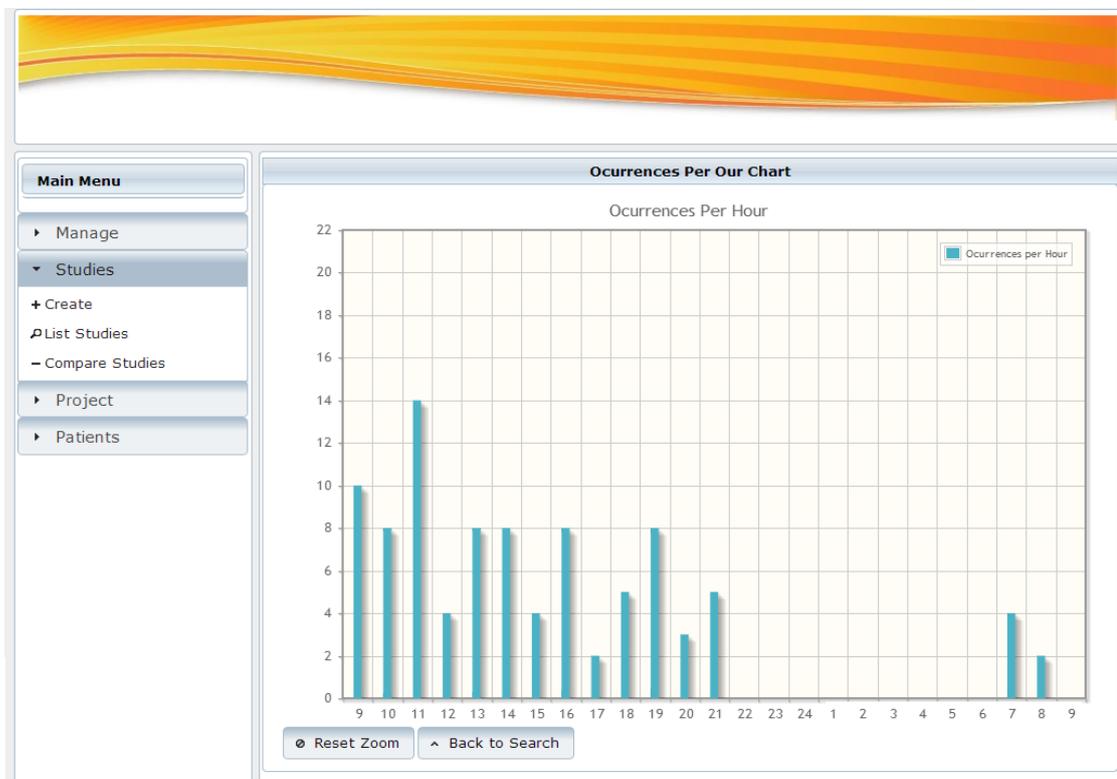


Figure 4.13 - Occurrences per Hour

Besides the screen of the results about one study, a comparison between studies can be made and seen in a similar bar chart as the one above but with the comparison of the occurrence per hour of multiple studies. It can be seen in Figure 4.14.

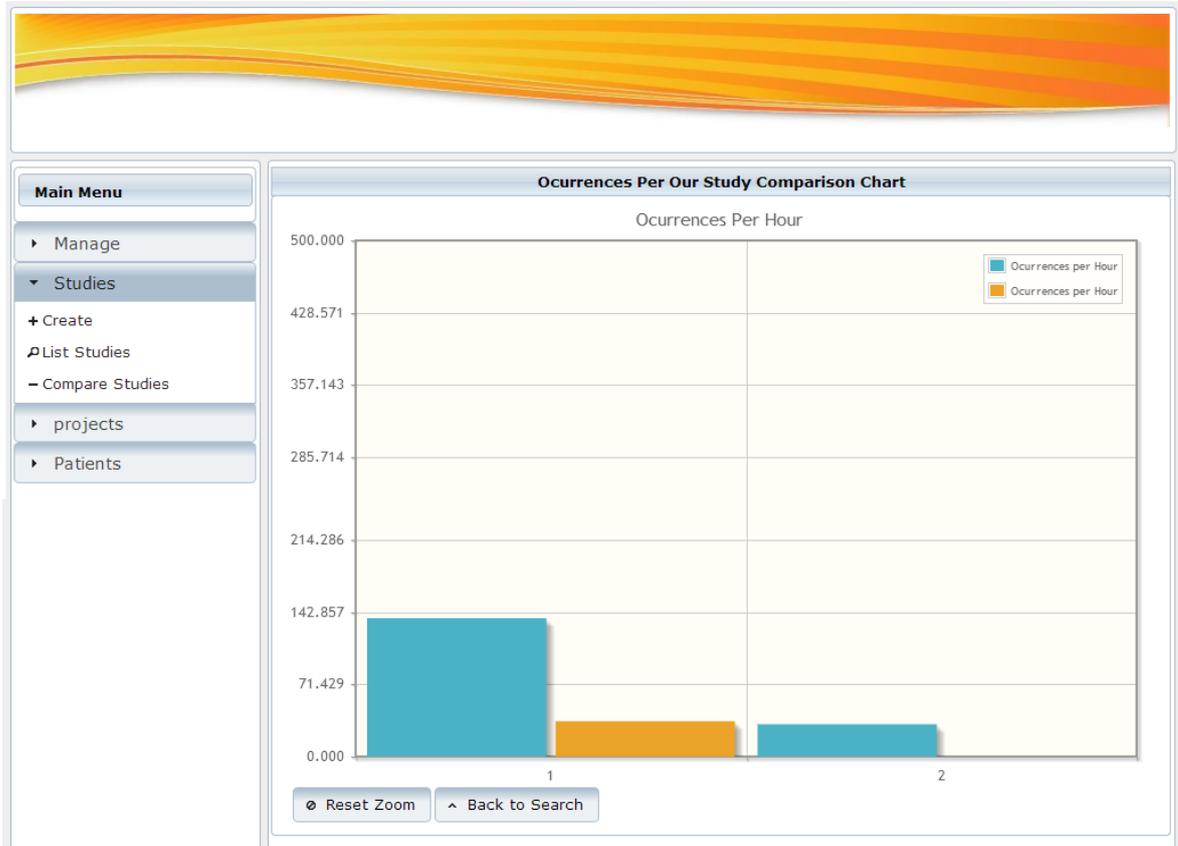


Figure 4.14- Study Comparison about Occurrences per Hour



# Chapter 5

## Conclusions and Future Work

### 5.1 Conclusions

With the work described in this dissertation a web platform was developed using the LCM tool to create studies about cough frequency occurrence in people with symptoms of cough. This document presents the theoretical basis of the problem, the study of technologies and practical developments made.

The objectives defined during the modeling system were generally met with success. The features implemented shows that the final system is a web tool with great potential to grow up in the field of objective cough frequency monitoring. However, other features could have been explored, naturally with different results.

The reuse of the LCM tool previously developed in another programming language shows that the reuse of code previously developed and tested, can be a good solution when it is possible to be made. It avoids the need for re-testing the application looking for possible problems. The LCM tool has been operating during the last few years and has been improved over time with successful results.

Through this work it was possible to explore new technologies unknown to me, and become richer in my level of knowledge. The programming paradigm for web application follows a slightly different approach comparing with local applications. The cares to have with a web platform are of greater complexity, because the platform works based on a client-server model that should work in multiple browsers that do not always meet the norms of the respective standards.

## 5.2 Future Work

Decisions taken during both requirements and implementation phase determined the final outcome of the project. Although the data and functionality that the system provides can be sufficiently beneficial, several improvements can still be made.

Relatively to the data and audio files, an external repository could be created for storing files without limitation of space. This approach will end the possible necessity to rationalize the space with cleaning techniques of older raw audio files in case of necessity of space.

File transfer can also be optimized in order to use a file transfer system with fault prediction, recovery of the same in case of necessity and synchronization and sharing between multiple physical locations. The solution studied at the section 2.3.2 that uses the Dropbox API can be a good approach because it ensures the total space that is required as well as data security.

The Solr or Lucene can be explored and integrated with this application, in order to take in advantage possible indexing of metadata.

Although studied and planned, the system operating as a distributed system were not fully implemented, though it would be a great asset for the application and could be used in a larger scale by several medical centers spread across the world.

## Bibliography

- [1] E. L. Foundation. "Lung Diseases," 18-Feb-2013; <http://www.european-lung-foundation.org/16-european-lung-foundation-elf-lung-diseases.htm>.
- [2] K. F. Chung, "Measurement and Assessment of Cough," *Cough: Causes, Mechanisms and Therapy*, pp. 57-73: Blackwell Publishing Ltd, 2003.
- [3] S. Matos *et al.*, "An automated system for 24-h monitoring of cough frequency: the leicester cough monitor," *Biomedical Engineering, IEEE Transactions on*, vol. 54, no. 8, pp. 1472-1479, 2007.
- [4] Y. Hiew *et al.*, "DSP algorithm for cough identification and counting." pp. IV-3888-IV-3891.
- [5] M. A. Coyle *et al.*, "Evaluation of an ambulatory system for the quantification of cough frequency in patients with chronic obstructive pulmonary disease," *Cough*, vol. 1, no. 1, pp. 3, 2005.
- [6] "Dropbox," 21-May-2013; <https://www.dropbox.com/>.
- [7] "JNI," 21-05-2013, 2013;  
<http://docs.oracle.com/javase/6/docs/technotes/guides/jni/spec/jniTOC.html>.
- [8] "Microsoft SQL Server," 22-Feb-2013; <http://www.microsoft.com/sqlserver>.
- [9] "Oracle," 22-Feb-2013; <http://www.oracle.com/>.
- [10] "MySQL," 22-Feb-2013; <http://www.mysql.com/>.
- [11] "MongoDB," 22-Feb-2013; <http://www.mongodb.org/>.
- [12] "Apache CouchDB," 20-Feb-2013; <http://couchdb.apache.org/>.

- [13] "MemcachedDB," 21-May-2013; <http://memcachedb.org/>.
- [14] "BigTable," 22-Feb-2013; <https://www.google.com>.
- [15] "Cassandra," 21-May-2013; <http://cassandra.apache.org/>.
- [16] "Facebook," 22-Feb-2012; <http://www.facebook.com/>.
- [17] "MongoDB and CouchDB comparison," 04-June-2013;  
<http://nosql.findthebest.com/compare/1-3/MongoDB-vs-CouchDB>.
- [18] "MongoDB Overview," 04-June-2013.
- [19] "JPA," 21-May-2013;  
<http://www.oracle.com/technetwork/articles/javaee/jpa-137156.html>.
- [20] "EclipseLink," 21-May-2013; <http://www.eclipse.org/eclipselink/>.
- [21] "DataNucleus," 21-May-2013; <http://www.datanucleus.org/>.
- [22] "Morphia," 21-May-2013; <https://code.google.com/p/morphia/>.
- [23] "Kundera," 21-May-2013; <https://code.google.com/p/kundera/>.
- [24] "SpringMongoDB," 21-May-2013; <http://www.springsource.org/spring-data/mongodb>.
- [25] "Primefaces overview," 21-May-2013;  
<http://primefaces.org/whyprimefaces.html>.
- [26] "DevRates," 21-May-2013; <http://devrates.com/stats/index>.
- [27] "Spring Security," 21-May-2013; <http://static.springsource.org/spring-security/site/articles.html>.
- [28] J. Barkley, "Workflow management employing role-based access control," Google Patents, 2000.
- [29] M. Keith, and M. Schincariol, *Pro JPA 2: Mastering the Java™ Persistence API*: Apress, 2009.