Reference Requirements for Records and Documents Management

Ricardo Jorge Seno Martins

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. José Luís Brinquete Borbinha

Examination Committee
Chairperson: Prof. António Manuel Ferreira Rito da Silva
Supervisor: Prof. José Luís Brinquete Borbinha
Member of the Committee: Prof. André Ferreira Ferrão Couto e Vasconcelos

May 2016
Acknowledgments

To my supervisor, Professor Doutor José Borbinha, for all the patience, the attention and the time spent helping me. The patience to show me the path that I couldn’t see, the attention given even when time was scarce, and all the help to finish this work.

To all the people involved in this work, especially António Higgs, that helped me greatly during the development every time I had problems and Denise Pedro that also helped me greatly.

To all my colleagues, Luís Antunes and António Matias, who despite being completely out of the scope of my thesis made an effort to help me when I needed. To my parents, that gave me the opportunity to proceed with my studies and become “someone” in this world. To Inês Almeida that helped me passing trough all of my fears and was always there for giving my strength. To all of you, thank you.
Abstract

When information systems started to be designed and implemented, each system was developed according to the needs of each organization. If the details of these systems were to be analyzed thoroughly, one would conclude that these details were very similar. This is due to the fact that the general goals were also very identical. Over time and gradually, this development became standardized, and with it came the need of creating requirements and good practices to build these systems. Reference requirements documents are an example of those practices, that guarantee a correct management of documents by organizations in their respective records management system.

Currently, these documents are used by system developers that are in the need of guidelines for developing their systems. The problem that arises is the fact that the documents were written by experts in the fields of records management, and generally system developers are not familiarized with this context. Therefore, there is a need of improving the presentation of these documents in order to simplify their comprehension.

The objective is to design a system capable of storing reference requirement documents by the means of a upload service, and allow an easy navigation through the documents in a view that would present, concurrently, the stored data all organized and the original document.

Keywords: Information Management, ISO, ISO standards, MoReq2010, Records Management, Records Management System
Resumo

Nos primórdios da concepção e implementação de sistemas de informação, cada sistema era desenvolvido de acordo com as necessidades de cada organização. Se os detalhes destes sistemas fossem analisados detalhadamente, rapidamente se iria concluir que os mesmos apresentavam várias semelhanças. Isto deve-se ao facto dos objectivos gerais de cada sistema serem também bastante idênticos. Gradualmente e com o decorrer do tempo, o desenvolvimento deste tipo de sistemas começou a seguir certos padrões, o que por sua vez levou à necessidade de criar requisitos e boas práticas para o desenvolvimento dos mesmos. Os documentos de referência de requisitos são um exemplo dessas práticas, pois garantem uma gestão de documentos correcta por parte das organizações nos respectivos sistemas de gestão de documentos de arquivos.

Actualmente, estes documentos são usados por programadores de sistemas que necessitam de uma certa orientação para desenvolver os respectivos sistemas. O problema que surge é o facto dos documentos terem sido escritos por especialistas na área da gestão documental, e geralmente, os programadores não estão familiarizados com este contexto. Deste modo, existe uma necessidade de melhorar a apresentação destes documentos para assim simplificar a sua compreensão.

O objectivo é conceber um sistema capaz de armazenar documentos de referência de requisitos através de um serviço de submissão, e permitir uma fácil navegação pelos mesmos numa vista que iria apresentar em simultâneo, os dados armazenados e organizados, bem como o documento original.

Palavras-Chave: Gestão de Informação, Gestão Documental, MoReq2010, Normas ISO,Sistemas de Gestão de Documentos de Arquivos
# Contents

<table>
<thead>
<tr>
<th>List of Tables</th>
<th>viii</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Figures</td>
<td>xi</td>
</tr>
<tr>
<td>Listings</td>
<td>xiii</td>
</tr>
<tr>
<td>Acronyms</td>
<td>1</td>
</tr>
</tbody>
</table>

## 1 Introduction

1.1 Problem and Motivation ........................................ 3
1.2 Objectives and Results ......................................... 4
1.3 Document Structure ............................................ 4

## 2 Problem Challenges and Related Work

2.1 The PDF Format .................................................. 5
2.2 The Challenges of PDF Data Extraction ...................... 6
2.3 PDF Data Extraction Techniques and Tools ................... 7
2.4 Technological Constraints ..................................... 9

## 3 ISO Standards as PDF Documents

3.1 ISO Standards Data ............................................... 18
3.2 Data Model .................................................... 20

## 4 Problem Analysis and Solution

4.1 Use Cases ..................................................... 21
4.2 System Architecture ........................................... 25

## 5 Implementation

5.1 Structure of the extracted files .............................. 29
5.1.1 XML structure ................................................ 29
5.1.2 TXT structure ............................................... 38
5.2 Structure of the JSON file ..................................... 40
5.3 Collection Viewer ............................................... 42
5.4 Document Viewer ................................................ 43
5.5 Document Uploader .............................................. 48
5.5.1 Data Processing ............................................... 50
6 Demonstration

6.1 "Reqs"Platform .................................................. 57
6.2 Upload Procedures .................................................. 60
6.3 "Reqs"Upload Service Performance .................................. 64

7 Conclusions and Future Work

7.1 Main issues faced during the development .......................... 67
7.2 Critical Analysis of the Work ........................................ 70
7.3 Future Work .......................................................... 71

Bibliography .................................................................. 72

A Implementation of the PDF rendering ............................... 75
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Browse Document</td>
<td>22</td>
</tr>
<tr>
<td>4.2</td>
<td>Browse Document Collection</td>
<td>22</td>
</tr>
<tr>
<td>4.3</td>
<td>Submit Document</td>
<td>23</td>
</tr>
<tr>
<td>6.1</td>
<td>ISO standards storing duration</td>
<td>65</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>PDF components</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>MEAN Stack</td>
<td>9</td>
</tr>
<tr>
<td>2.3</td>
<td>MongoDB Management Tool (Administration User Interface)</td>
<td>10</td>
</tr>
<tr>
<td>2.4</td>
<td>Documents Structure Model</td>
<td>12</td>
</tr>
<tr>
<td>2.5</td>
<td>Overview Page</td>
<td>13</td>
</tr>
<tr>
<td>2.6</td>
<td>MoReq specification in Reqs</td>
<td>14</td>
</tr>
<tr>
<td>2.7</td>
<td>MoReq Demo Page</td>
<td>14</td>
</tr>
<tr>
<td>2.8</td>
<td>MoReq Demo Page - Non-Functional Requirements</td>
<td>14</td>
</tr>
<tr>
<td>3.1</td>
<td>ISO 26122</td>
<td>17</td>
</tr>
<tr>
<td>3.2</td>
<td>ISO 15489-1</td>
<td>18</td>
</tr>
<tr>
<td>3.3</td>
<td>ISO 16175-2</td>
<td>18</td>
</tr>
<tr>
<td>3.4</td>
<td>MoReq 2010</td>
<td>19</td>
</tr>
<tr>
<td>4.1</td>
<td>System Use Cases</td>
<td>22</td>
</tr>
<tr>
<td>4.2</td>
<td>PDF conversion to XML</td>
<td>23</td>
</tr>
<tr>
<td>4.3</td>
<td>PDF conversion to TXT</td>
<td>23</td>
</tr>
<tr>
<td>4.4</td>
<td>Exporting the XML file from Adobe Acrobat Standard DC</td>
<td>24</td>
</tr>
<tr>
<td>4.5</td>
<td>Exporting the TXT file from Adobe Acrobat Reader DC</td>
<td>24</td>
</tr>
<tr>
<td>4.6</td>
<td>System Architecture</td>
<td>25</td>
</tr>
<tr>
<td>4.7</td>
<td>Document Uploader Workflow</td>
<td>27</td>
</tr>
<tr>
<td>5.1</td>
<td>Data Model applied to ISO-15489-x and ISO-26122</td>
<td>29</td>
</tr>
<tr>
<td>5.2</td>
<td>Data Model applied to ISO-18128</td>
<td>32</td>
</tr>
<tr>
<td>5.3</td>
<td>Data Model applied to ISO-16175-x</td>
<td>35</td>
</tr>
<tr>
<td>5.4</td>
<td>Collection Viewer</td>
<td>43</td>
</tr>
<tr>
<td>5.5</td>
<td>Document Viewer</td>
<td>44</td>
</tr>
<tr>
<td>6.1</td>
<td>Home page for Reqs Application</td>
<td>57</td>
</tr>
<tr>
<td>6.2</td>
<td>Overview page for Reqs Application</td>
<td>58</td>
</tr>
<tr>
<td>6.3</td>
<td>Document Viewer page for Reqs Application</td>
<td>58</td>
</tr>
<tr>
<td>6.4</td>
<td>Document Viewer in unsynchronized mode</td>
<td>59</td>
</tr>
<tr>
<td>6.5</td>
<td>Document selection in Document Viewer page</td>
<td>59</td>
</tr>
<tr>
<td>6.6</td>
<td>Document Upload page for Reqs Application</td>
<td>60</td>
</tr>
<tr>
<td>6.7</td>
<td>Uploading form</td>
<td>64</td>
</tr>
</tbody>
</table>
Listings

2.1 Text File Output example ............................................. 8
2.2 EXtensible Markup Language (XML) File Output example .......... 8
2.3 Route Structure .................................................... 10
5.1 XML example from ISO-26122 ....................................... 30
5.2 Figure example from ISO-26122 .................................... 31
5.3 Header example from ISO-26122 .................................... 31
5.4 Paragraph example from ISO-26122 ................................. 31
5.5 Index example from ISO-26122 ...................................... 31
5.6 Table example from ISO-26122 ...................................... 32
5.7 XML example from ISO-18128 ....................................... 33
5.8 Table of Contents example from ISO-18128 ......................... 33
5.9 Table of Contents example from ISO-18128 ......................... 34
5.10 Table of Contents example from ISO-18128 ....................... 34
5.11 Bookmark Tree example from ISO-16175-1 ......................... 36
5.12 Section without Sect element example from ISO-16175-1 ....... 37
5.13 Bullet list example from ISO-16175-1 .............................. 37
5.14 Index example from ISO-26122 ...................................... 38
5.15 Sections example from ISO-26122 .................................. 39
5.16 Bullet lists example from ISO-26122 ............................... 40
5.17 JSON example from ISO-15489-1 .................................. 41
5.18 Documents list in the Collection Viewer ............................ 42
5.19 Controller Request .................................................. 42
5.20 Routing Services Reply ............................................ 43
5.21 PDF viewer HTML .................................................. 44
5.22 Routing Services Response ......................................... 45
5.23 How the response is treated by Document Viewer Controller .... 45
5.24 PDF rendering ....................................................... 46
5.25 JSON file response .................................................. 47
5.26 Document Viewer Controller receiving the JSON file .............. 47
5.27 Document Uploader submission form ............................... 48
5.28 Document Uploader Controller request ............................ 49
5.29 Routing Services action upon upload ............................... 49
5.30 Document Uploader Controller receiving the response from the Server .................. 49
5.31 Parsing of the XML file ............................................. 50
5.32 Index Storing ....................................................... 50
5.33 Index Hierarchy ...................................................... 51
5.34 Creation of the XML object ........................................ 51
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.35 Populating the XML object with chapters and sections</td>
<td>51</td>
</tr>
<tr>
<td>5.36 Function populateDoc</td>
<td>52</td>
</tr>
<tr>
<td>5.37 Function populateDocTOCI</td>
<td>52</td>
</tr>
<tr>
<td>5.38 XPATH expression for obtaining the content from the xmlDoc object</td>
<td>52</td>
</tr>
<tr>
<td>5.39 Function populateDocBookmark</td>
<td>52</td>
</tr>
<tr>
<td>5.40 Reading the TXT file</td>
<td>53</td>
</tr>
<tr>
<td>5.41 Building the index from the TXT file</td>
<td>53</td>
</tr>
<tr>
<td>5.42 Populating the doc object with the data from the TXT file</td>
<td>53</td>
</tr>
<tr>
<td>5.43 Function for populating the XML object</td>
<td>54</td>
</tr>
<tr>
<td>5.44 Determining the type of element</td>
<td>55</td>
</tr>
<tr>
<td>5.45 Obtain all chapters</td>
<td>55</td>
</tr>
<tr>
<td>5.46 Import Component processing the Page content</td>
<td>56</td>
</tr>
<tr>
<td>6.1 Structured Index of International Organization for Standardization (ISO) 26122 XML File</td>
<td>62</td>
</tr>
<tr>
<td>6.2 Structured Index of ISO 26122 Text File</td>
<td>62</td>
</tr>
<tr>
<td>6.3 Structured Table in ISO 26122 Text File</td>
<td>63</td>
</tr>
<tr>
<td>6.4 Structured JSON of ISO 26122</td>
<td>63</td>
</tr>
<tr>
<td>7.1 Text file example from ISO 26122</td>
<td>68</td>
</tr>
<tr>
<td>7.2 Text file example from ISO 26122</td>
<td>68</td>
</tr>
<tr>
<td>7.3 Part of the XML file from ISO 26122</td>
<td>68</td>
</tr>
<tr>
<td>7.4 Part of the XML file from ISO 18128</td>
<td>69</td>
</tr>
<tr>
<td>7.5 ISO 26122 hierarchic structure example</td>
<td>69</td>
</tr>
<tr>
<td>7.6 ISO 18128 hierarchic structure example</td>
<td>70</td>
</tr>
<tr>
<td>A.1 Implementation of the PDF rendering</td>
<td>75</td>
</tr>
</tbody>
</table>
Acronyms

**BSON** Binary JavaScript Object Notation. [10]

**CSS** Cascading Style Sheets. [11, 26]

**E-Ark** European Archival Records and Knowledge Preservation. [3]

**HDD** Hard disks. [65, 71]

**HTML** Hypertext Markup Language. [11, 13, 26, 42, 43, 44, 48]

**HTTP** Hypertext Transfer Protocol. [11]

**ISO** International Organization for Standardization. [xii, 4, 5, 6, 7, 8, 14, 15, 17, 18, 19, 20, 21, 23, 38, 43, 48, 50, 56, 57, 60, 62, 63, 65, 67, 68, 70, 71]

**JSON** JavaScript Object Notation. [10, 12, 28, 40, 46, 47, 48, 63, 71]

**MoReq** Modular Requirements For Records Systems. [9, 12, 14, 19, 20, 23, 67, 71]

**MVC** Model View Controller. [11, 12]

**MVVM** Model–View–ViewModel. [11]

**PDF** Portable Document Format. [4, 5, 6, 7, 8, 12, 21, 23, 24, 25, 28, 29, 43, 44, 45, 46, 47, 48, 58, 60, 61, 63, 71]

**RAM** Random-access memory. [54]

**RDMS** Relational Database Management System. [10]

**SSD** Solid state drives. [65, 71]

**URL** Uniform Resource Locator. [42, 49]

**VPS** Virtual Private Server. [64]

**XML** EXtensible Markup Language. [xiii, xiv, 6, 7, 8, 12, 13, 14, 19, 20, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 35, 38, 40, 48, 49, 50, 51, 52, 53, 54, 55, 56, 62, 64, 65, 67, 68, 70]

**XPATH** XML Path Language. [8, 50, 52, 54, 55]
Chapter 1

Introduction

In the present chapter, the problem definition will be introduced, the motivation that leads to the development of a solution, the objectives of the project and the expected results, followed by a description of the structure of the document.

1.1 Problem and Motivation

Nowadays, organizations have to process a great amount of information, and accomplish it in the shortest time possible, in order to quickly decide to proceed or not in a business opportunity. To do this, it’s important to manage information in the most efficient way, something that can be provided by a records management system which in turn is much desired by organizations.

This kind of systems is a great advantage for organizations with record keeping obligations due to features that support dematerialization of processes and documentation, normalization of documents, centralized management of records, fast access & search of documents and more control of information flows. An example of that kind of systems already available in the market is for instance Microsoft Sharepoint.

Before these systems existed, organizations kept records in any way they saw fit, but with the creation of great amounts of information came the need of controlling this outbreak. Organizations started to implement systems to deal with this issue, and each solution would be created according to the needs of each organization, however records would still be kept in any suitable form, not following any kind of guidelines or formal information life-cycle procedures. To standardize these procedures and the best practices for records management, documents with reference requirements were created to assist in the validation and implementation of records management systems.

These documents can be extremely complex and when a stakeholder resorts to them for guidelines and requirements elicitation for developing it’s own system, he might select two requirements that appear in distinct documents and are written in different ways but actually have the same meaning, thus leading to requirements redundancy. All these aspects represent problems that affect the business.

The motivation for this work comes from a related project in the area that aims to provide a solution in the form of an information system for the stated problem: the European Archival Records and Knowledge Management Framework.

https://products.office.com/en-us/SharePoint/collaboration
"The project will create a methodology for electronic document archiving, synthesising existing national and international best practices, that will keep records and databases authentic and usable over time. This will be done by providing simple, efficient access to the workflows for the main activities of an archive which are acquiring, preserving and enabling re-use of information. The practices developed within the project will reduce the risk of information loss due to unsuitable approaches to keeping and archiving of records".

1.2 Objectives and Results

Having the problems stated in section 1.1 in mind, the objective of this work is to develop an information system to support the analysis of a set of requirements documents by a stakeholder, more precisely ISO standards documents.

ISO standards present quite a challenge due to the fact that:

- there are numerous standards therefore there is a great amount of data to store in the system;
- standards are very likely to be updated and so the system must be receptive to accept new standards on a more frequent basis;
- each standard has a complex structure, because each one is unique as well as its contents;
- each standard is in the Portable Document Format (PDF) format and the inherent data must be extracted.

Taking these aspects into account, the expected result would consist in a solution capable of extracting the data from the standard PDF, storing this data by the means of an upload service, and allow an easy navigation through the standard in a view that would present, concurrently, the stored data all organized and the original standard document.

1.3 Document Structure

This document comprises seven chapters. Chapter 2 contains the problem challenges and the related work, more precisely it explores the PDF format, the challenges of extracting data from PDF documents and its consequent manipulation, the existing PDF data extraction techniques and tools that were used in the project, the work that served as basis for this project. Chapter 3 presents the analysis of ISO standards as PDF documents and the derived Data Model. Chapter 4 explains the design of the information system and the system architecture. Chapter 5 contains implementation details, this is, how certain features were implemented, the roles of the existing components and the structure of important files to the application. Chapter 6 presents results of what was implemented of the information system as well and some tests that were developed. Chapter 7 depicts the problems encountered during the development as well as a critical analysis of the work and the future work that can be developed.
Chapter 2

Problem Challenges and Related Work

This chapter aims to resume the work developed concerning the areas that will be approached during this thesis. First one will present some concepts about the PDF format since its an important aspect of this dissertation, then the problems that the solution must solve will be presented, taking into account PDF documents and the manipulation of the respective data. The existing techniques and tools that provide data from PDF documents will be presented as well. Then the technology that is used in the dissertation is going to be depicted and also the work already developed which provides ground for this project.

2.1 The PDF Format

In order to work with PDF documents, one must determine how they are structured and what information they provide. For that purpose, in this section all the details and concerns about the PDF Format are going to be presented.

As the name implies, it's a data format that can be used to describe and present documents in a manner which is independent of software applications, hardware or even operating systems. Each PDF file encapsulates a complete description of a fixed-layout flat document, including the text, fonts, graphics, and other information needed to display it. It's a commonly used file format to exchange data, for example it’s the recommended format used for printing documents.

Adobe, the developers of PDF market software to create, edit and visualize PDF files. The specifications of the file format are publicly available and meanwhile even became an official ISO standard. A Public Patent License to ISO 32000-1 was published by Adobe, granting royalty-free rights for all patents owned by Adobe that are necessary to make, use, sell, and distribute PDF compliant implementations. Several other companies develop related software as well.

A PDF document consists in a collection of objects that together describe the appearance of one or more pages, possibly accompanied by additional interactive elements and higher-level application data. A PDF file contains the objects making up a PDF document along with associated structural information, all represented as a single self contained sequence of bytes. A document's pages (and other visual elements) can contain any combination of text, graphics, and images. A page’s appearance is described by a content stream, which contains a sequence of graphics objects to be painted on the page.
A PDF document can contain higher-level information that is useful for interchange of content among applications. In addition to specifying appearance, a document's content can include identification and logical structure information that allows it to be searched, edited, or extracted for reuse elsewhere.

According to [2] the PDF can be divided in four parts as can be seen in Figure 2.1.

**Figure 2.1: PDF components**

- **Objects** - A PDF document is a data structure composed from a small set of basic types of data objects such as boolean values, integer and real numbers, strings, names, arrays among others.
- **File structure** - The PDF file structure is independent of the semantics of the objects and determines how these are stored in a PDF file, accessed, and updated.
- **Document structure** - The PDF document structure specifies how the basic object types are used to represent components of a PDF document: pages, fonts and so forth.
- **Content stream** - A PDF content stream contains a sequence of instructions describing the appearance of a page or other graphical entity. These instructions, are also represented as objects, however they are conceptually distinct from the basic object types already mentioned.

With these components in mind, one must determine how to extract data from PDF documents, which data to extract from PDF documents and how to manipulate this data. Section 2.2 will provide some clarification in these matters.

### 2.2 The Challenges of PDF Data Extraction

In this section, all the problems that the solution must solve - considering PDF documents and the manipulation of its data - will be presented.

The ISO standard documents that must be uploaded into the system are all in the PDF format, which means that the data must be extracted in order to be embed in the system. The data must be extracted and it must comprise all the contents including paragraphs, images, bullet lists and tables. A certain
structure must be attached to the contents, and in a hierarchical fashion. For example, the index of the document must be extracted and grouped. Another example concerns the structure of chapters, sections, subsections and their respective contents, this is, a chapter should be comprised by its subsections, the subsections by the respective sub-subsections and so on. This relationship should be expressed in the extracted data. All of these concerns, map into the components presented in Figure 2.1. A mark-up language would be a suitable way of representing these aspects (for example XML), including the hierarchy of the chapters and the respective contents. However it would be necessary to use a tool or a technology of some kind to perform that process.

Besides the extraction, the data should be stored in a structured way, in the system. This workflow must be automated, starting by the extraction and ending in the storage. The visualization of the data consists in another issue, because the data must be retrieved, and presented simultaneously with the original document. When the data is retrieved, this is done taking into account the structure used previously in the storage.

Concerning the extraction of data from ISO standards, section 2.3 provides an insight on the existing techniques and tools for data extraction from PDF documents.

### 2.3 PDF Data Extraction Techniques and Tools

In the following section the most relevant techniques and tools for this project, that allow to extract the contents of a PDF are going to be presented, in order to have a suitable file that can be manipulated and embed in the system.

The Portable Document Format, PDF, is widely used nowadays, and the specification of ISO standards is no exception, each standard is available in this format. In the context of this dissertation, one of the steps to perform in order to have an ISO standard embed in the system, is to have a file with the contents of the standard that can be manipulated because the PDF itself is not sufficient for this purpose. A research was conducted in order to find an appropriate solution capable of extracting the contents of the PDF, if possible combined with a markup language - such as XML - in order to provide a data structure that organizes the contents.

Several solutions were available, however the majority consisted in websites that didn’t seem reliable and could stop providing their services at any moment thus compromising the work being developed, or in commercial software sold by vendors that didn’t provide at least a trial version to verify the utility and authenticity of the solution. Relatively to the websites, some of them were tested but the quality of the extracted contents was extremely low, in some cases it was practically unreadable. An attempt was also made using Microsoft Word that allows a document to be saved in the XML format, however the result was a very complex file with an enormous amount of elements and the text itself converted in a non readable format. Another kind of solutions consisted in libraries such as the Apache PDFBox that could extract the text from the PDF yet it didn’t provide the expected results. After some more research one started exploring the features from the Adobe tools, namely Adobe Acrobat Reader DC and Adobe Acrobat Standard DC. Both tools are from a reliable company and have been available for quite some time.

time. In the context of this dissertation they proved to add value:

Adobe Acrobat Reader DC consists on the PDF reader from Adobe, its freeware and gives read-only permissions to the user. The interesting feature that adds value to this dissertation, it’s the fact that it allows a user to save the contents of the PDF in a text file format, which is something that’s workable, despite the lack of structure that can be provided by a mark-up language. An example of the output provided by Adobe Acrobat Reader DC is presented in listing 2.1, where the contents from ISO 26122 were extracted.

Adobe Acrobat Standard DC consists on the PDF writer from Adobe that allows the user to create and edit PDF documents. Contrary to the PDF reader, this version is shareware, this is, it has a trial version but after some time the user has to buy the software in order to keep using it. In the context of this dissertation, the writer has a powerful feature, it allows the export of the data in the XML format. With this format the PDF contents are organized in a hierarchic fashion, which will simplify the process of embedding the ISO standard into the system. An example of the output provided by Adobe Acrobat Standard DC is presented in listing 2.2, where the contents from ISO 26122 were extracted.

Listing 2.1: Text File Output example

Listing 2.2: XML File Output example
As one can see, both approaches have their set of features and drawbacks. Adobe Acrobat Reader DC is free but the text file that is produced has a lack of structure due to the nonexistence of a mark-up language. Adobe Acrobat Standard DC is a commercial solution, despite having a trial version, but the XML file that is produced is far richer than the text file. When comparing the Text file example in listing 2.1 with the XML file example in listing 2.2 one can see that the difference resides in the existence of tags in the XML. These tags are an important advantage because the data can be accessed rapidly through XML Path Language (XPATH) expressions. For example if all chapters have a H3 element associated, an XPATH expression can be built to obtain them and the final result will be the index of the document. This can also be done with the Text file, but the file has to be parsed and some sort of algorithm has to be developed to achieve the same result. In chapter 6.2 the work developed with these solutions is further explained.

Given the context of this thesis and the features provided by the tools from Adobe, they proved to be the best option when it comes to extract the contents, despite being an outside service relatively to the system.

2.4 Technological Constraints

This section describes the technology that was used in the development of this dissertation as well as the work previously developed.

Previous to this dissertation, an information system was developed (Reqs), capable of supporting reference requirements documents[3], namely Modular Requirements For Records Systems (MoReq) 2010[4]. Since the objective consists on developing a service for this system, one must analyze and depict its technology. The Reqs system, consists on a Web Application, this is, a client-server software application which the client (or user interface) runs in a web browser. Reqs makes use of the MEAN stack, that consists in a collection of Javascript-based technologies comprised by (M)ongoDB, (E)xpress.js, (A)ngular.js, and (N)ode.js as can be seen in Figure 2.2.

![Figure 2.2: MEAN Stack](http://joaopsilva.github.io/talks/End-to-End-JavaScript-with-the-MEAN-Stack)
Concerning each technology:

**MongoDB** is an open-source document database that provides high performance, high availability, and automatic scaling. It's a NoSQL cross-platform document-oriented database that doesn't use the traditional table-based relational database. Records in MongoDB are documents (consist in a data structure composed of field and value pairs), which are similar to **JSON** objects. The values of the fields may include other documents, arrays, and arrays of documents. The documents stored by MongoDB are **Binary JavaScript Object Notation (BSON)** documents, which is a binary representation of JSON documents, though it contains more data types than JSON. One important feature that was used in the previous project is the Javascript execution in the server-side that consists in queries that are sent directly to the database to be executed. For administration and data viewing Robomongo was used to provide a GUI-style administrative interface, presented in Figure 2.3.

![Figure 2.3: MongoDB Management Tool (Administration User Interface)](image)

Figure 2.4 presents the several existing collections in Reqs, which is the equivalent to tables in a **Relational Database Management System (RDMS)**.

**Express.js** consists in a web framework for Node.js that aids in the organization of Reqs, namely on the server-side. It helps managing routes, this is **routing**. Routing determines how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, etc). A route takes the following structure:

```
app.METHOD(PATH, HANDLER)
```

**Listing 2.3: Route Structure**

Where:

- app is an instance of express.
- METHOD is an HTTP request method.
- PATH is a path on the server.
- HANDLER is the function executed when the route is matched.

A route METHOD is derived from one of the [Hypertext Transfer Protocol (HTTP)] methods (GET, PUT, POST, etc), the PATH defines the endpoints (in the server-side) at which requests can be made and the HANDLER is the action to be executed combined with a response method. Routes are very helpful in handling requests, for instance in the case of Reqs are responsible for retrieving the data of the documents to be viewed. They also take place in error handling in order to prevent the server from stopping its execution.

Angular.JS is a front-end framework that provides a [Model View Controller (MVC)] and [Model–View–ViewModel (MVVM)] design patterns, in order to separate the presentation from the data and its manipulation as well as components commonly used in rich Internet applications. Angular.JS allows the usage of [Hypertext Markup Language (HTML)] as the template language as well as its extension to express the application’s components clearly and succinctly. The most important core features are the following:

- **Templates** - Consists in the rendered view with information from the controller and model. These can be a single file (like index.html) or multiple views in one page using **partials** (parts of the main view specified in another file).
- **Controller** - JavaScript functions that are bound to a particular scope.
- **Scope** - Objects that refer to the model and act as a glue between the controller and the view.
- **MVVM** - It’s similar to the **MVC** pattern, but given that Angular.JS provides a great degree of flexibility to nicely separate presentation logic from business logic and presentation state, the **MVVM** design pattern is the most appropriate to be considered. It comprises the Model and the View where the data and the presentation reside. The third component, ViewModel, is a result of the Data-binding feature that where the view is a projection of the model at all times.
- **Data-binding** - The automatic synchronization of data between model and view components.
- **Services** - JavaScript functions that are responsible to do a specific tasks only and normally injected using dependency injection mechanism of Angular.JS.
- **Dependency-Injection** - Software design pattern that deals with how components get hold of their dependencies. The Angular.JS injector subsystem is in charge of creating components, resolving their dependencies, and providing them to other components as requested.
- **Directives** - Markers on a DOM element (such as an attribute, element name, comment or [Cascading Style Sheets (CSS)] class) that tell Angular.JS’s [HTML] compiler to attach a specified behavior to that DOM element (e.g. via event listeners), or even to transform the DOM element and its children. Directives have the prefix “ng-” and Angular.JS comes with a set of these directives built-in, like ngBind, ngModel, and ngClass.

---

14 http://expressjs.com/en/api.html#res
15 www.angularjs.org
Node.js is a server platform that uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. It aids in the interaction with networks, file systems, or other I/O sources like databases for instance. Node can’t be considered as a web framework or a programming language, instead it's considered a middle term between these two because it's designed to be simple and easy to use as well as useful for I/O based programs that need to be fast and/or handle lots of connections. In the context of Reqs application, it must deal with requests from users and be fast in the interaction with the database, this is, MongoDB.

Using this kind of approach, brings several advantages:

- The code becomes isomorphic, this is, only one language is used in the whole application (Javascript). If a part of the application is implemented in the server-side and perhaps is better placed in the client-side, that part can be moved easily because the language is the same in either Node.js or Angular.JS;
- It supports the MVC pattern;
- JSON is used for transferring data;
- There is a big amount of modules provided by Angular.JS Node.js;
- MongoDB provides a lot of flexibility unlike MySQL's structure which is confining due to the existence of tables.

Regarding the context of this dissertation, certain features of the system were already developed. As mentioned, a previous project [3] managed to embed in Reqs the complete specification of MoReq 2010 [4]. The data regarding the MoReq 2010 [4] specification was already available in a workable format that could be integrated directly in the Reqs system, the XML format. The structure of the XML files dictated the database structure in MongoDB as it can be seen in Figure 2.4.

![Figure 2.4: Documents Structure Model](image)
Exploring the model, a Document can be the MoReq 2010 specification in the PDF format. The PDF has Pages, and a associated Structure that is unique, due to the amount of sections and subsections it may contain. This Structure comprises a set of sections, which in turn have Blocks that contain all elements such as Paragraphs, Bullet Lists, Tables and so on. All dependencies that are shown in Figure 2.4 are expressed in each document in MongoDB. Later in section 3.2 it is discussed how the model can be applied to the context of this dissertation.

To embed the document in the system, an Import component was developed in the server-side. This component parses the XML file by using the module libxmljs and stores its contents in the database. The file is parsed according to the structure, starting by the high level elements that correspond to chapters, and continuing through child nodes until paragraphs and text elements are found. To present the data stored in the database, a set of suitable views was designed. These views consist in pages in the web application, namely Overview and MoreqDemo. Figure 2.5 presents a screenshot of the Overview page.

![Overview Page](image)

Figure 2.5: Overview Page

In the Overview page, a user can navigate through the document with the help of an index. The index appears on the left panel of the screen, listing all the chapters of the document. After choosing an option on the left panel, the right panel shows information of the document regarding that option. When a user accesses the Overview page, the client (the browser), sends a request to the server that will retrieve the data of the selected document to the client.

At a more detailed level, the view which contains the HTML has an associated controller, which will use a service to get the document data. The purpose of the service is to provide a window to the client so that resources can be accessed. The request is sent with a specific url and the id of the document in it. In the server-side, using the routing mechanism, the server has a pre-defined action if a client tries to access that specific url, which is retrieving from the database the data from that specific document to the client. In this interaction, Node.js reads the content of the Database and serves it to the client in the most expeditious fashion. The rendering of data relatively to other views corresponds to the same process. Figure 2.6 shows an example of choosing an option on the left panel and the available information for that option appearing on the right panel.

https://github.com/polotek/libxmljs
The MoReq Demo page shown in Figure 2.7 has options to list some of the documents most important components: functional requirements, non-functional requirements, and information model.

The Intro tab presented in Figure 2.7 shows some brief information about how to navigate through MoReq Demo. The remaining tabs present content related to each component using the two panel visualization as can be seen in Figure 2.8.
The concern of this dissertation is to achieve the same result but using ISO standards instead of the MoReq specification. To achieve that purpose it seemed right to follow the same approach because the work already developed could be reused and adapted to the context of this thesis. For instance, the domain model presented in Figure 2.4 is capable of being reused as long as the ISO standard XML file is obtained first. Another example is the Overview page shown in Figure 2.6 that renders the MoReq content stored in the database. If an ISO standard is previously stored in the database, this page can also be reused to present that content.

The great challenge comes when one considers the data because there are a great amount of ISO standards and contrary to the work previously developed, there were no XML files already available to work with. This will be further analysed in Chapter 4.
Chapter 3

ISO Standards as PDF Documents

In this chapter one will approach ISO standards by presenting some concepts, analyzing the contents and structure of standards and determine a suitable Data Model.

ISO standards, are developed by ISO which stands for International Organization for Standardization. This organization is an independent, non-governmental international organization that, through its members, it brings together experts to share knowledge and develop voluntary, consensus-based, market relevant International Standards that support innovation and provide solutions to global challenges. A standard is a document that provides requirements, specifications, guidelines or characteristics that can be used consistently to ensure that materials, products, processes and services are fit for their purpose, safe, reliable and of good quality:

- For business, they are strategic tools that reduce costs by minimizing waste and errors, and increasing productivity and also help companies to access new markets.
- When products and services conform to International Standards consumers can have confidence that they are safe, reliable and of good quality.
- National governments can use ISO standards to support public policy, for example, by referencing ISO standards in regulations.

An example of such standards is presented in Figure 3.1.

![Figure 3.1: ISO 26122](image-url)
These requirements, specifications, guidelines or characteristics that comprise standards, are presented in a certain structure that must be thoroughly analyzed in order to embed the documents in the information system, Reqs. That structure is presented in section 3.1.

3.1 ISO Standards Data

One of the project's objectives is to embed documents, i.e. ISO Standards, into an information system, in order to perform some operations on its data. With several standards available, this specification comprises a sufficient amount of material to work with and test the basic functionality of the system.

First, the document must be seen exactly as what it is; a simple “document” or manual, with pages, sections, paragraphs, textual elements as titles and sentences, tables, and images. Several standards (ISO 15489-x[5][6], 16175-x[7][8][9], 18128[10], 26122[11]) were used for testing due to its complex structure and vast content. In all standards this type of document structure is used. The following Figures 3.2 and 3.3 present some excerpts of ISO standards 15489 - Part 1[5] and 16175 - Part 2[8].

![Figure 3.2: ISO 15489-1](image1)

![Figure 3.3: ISO 16175-2](image2)
As can be seen, the degree of resemblance between both examples is high, and this extends to all 
it’s structure, which includes tables, bullet lists, images, and others. The challenge, was to somehow, 
import these documents and all of it’s contents into Reqs. From previous works [3][12] there was a copy 
of MoReq 2010[4] in a XML document format stored in Reqs which contained all of the information 
of the document. After analyzing these files in the XML format, one could realize that a mapping could be 
done, this is, represent ISO standard documents in the same format.

Figure 3.4: MoReq 2010

In Figure 3.4, despite the fact that MoReq and ISO have different styles when it comes to the 
appearance, both types of documents can be represented in the same structure dictated in the XML files 
of MoReq. This is due to the fact that both documents have chapters, sections, bullet lists, tables and 
so on. When it comes to the XML documents, one could see that these had a structured way of rep-
resenting a document contents, making use of tags to represent pages, sections, images, texts, tables 
and structuring them in a hierarchic fashion.

Taking a step further in this analysis, the page tag, despite the definition of what a page is, represents 
a chapter with its title, and all of the following elements are children of this node. Inside the page tag, 
there are sections and these are divided in two kinds, the section container and the section content. The 
section container has also a title and is a parent section for other section contents. The section content 
consists in the last level child of a chapter, and it’s the parent of elements such as paragraphs, texts, 
etc. In Figure 3.2 for example, the chapter 6 maps to the page tag and sections 6.1 and 6.2 map into 
the section content tag since they have no more sections inside them.

Section contents contain paragraphs, images and tables. Paragraphs contain a text tag with the text 
existing in each paragraph. Paragraphs can also have the italics or bold tag designed for bold and italic 
format and the links tag for linking between different parts of the document. These last three types of 
tags were not taken into account, but this decision will be explained in chapter [5].
Contrary to MoReq documents, ISO documents weren’t available in the XML format from the start. In order to have them in Reqs, first a conversion would need to take place in order to obtain these XML files. That process will be explained in chapter 4.1. The structure of the database model was rapidly achieved through these elements, and it’s further explored in section 3.2.

### 3.2 Data Model

In the previous section 3.1, it was depicted the general structure of ISO standard documents, and how these could be represented by the same structure in the XML files of MoReq2010. In the present section one will see how a domain model for a database can be achieved, capable of storing documents and their inherent internal structures that consist, on pages, sections, paragraphs and other kinds of elements.

As stated in section 2.4, a previous project managed to embed the complete specification of MoReq2010 in Reqs and the domain model in Figure 2.4 resulted as the structure of the database for that work. When comparing ISO standards with the MoReq specification, it’s obvious that these are distinct documents, due to its presentation as well as contents. However, despite these differences, ISO standards also have paragraphs, images, tables and bullet lists as presented in Figure 2.4. For this reason, instead of developing a new model, the model presented in 2.4 was used as the required Data Model in this project that served as structure for the database.

For the purpose of this project not all the elements the Block comprises will be used, something that will be discussed in chapter 5 as well as the manipulation of these elements in order to embed ISO standards into Reqs.
Chapter 4

Problem Analysis and Solution

In the present chapter, it will be presented a description of the Problem Analysis and Solution. More precisely, it will be shown the proposed solution and what the system to build should be. A more detailed description is also provided in this chapter, by exhibiting internal schemes of the solution.

The proposed solution can be conceptualized as a system, presented to users as a web application, Reqs. As mentioned in section 2.4, the system was already developed, however, it didn’t take into account the rendering of ISO standard documents or even the extraction, manipulation and upload of its data. The documents in consideration should be available to a user trough the application interface in order to allow navigation inside their structure, as well as the upload service. The most relevant functionalities of the application are presented in the next section with the help of use cases. Still in this analysis, is available an overview of the System Architecture.

4.1 Use Cases

To develop the system it’s important to have a representation of the user's interaction with the system. To achieve that, use cases will be used, as well as the association with the respective users. When it comes to actors, there are three types that map into the expected users: the Expert and the User. The Expert has a deep knowledge on ISO standards and is well aware of every step that is required before submitting the PDF of the ISO standard into the system, Reqs. The User has "read-only" permissions, this is, has permission to browse the document collection as well as a single document. Expert inherits from User, which means that the Expert is involved in every use case, Expert is directly associated with the submission of the documents and has also access to the User's use case's, and User has only access to browsing functionalities.

In the early stages of the project, some of the use cases and actors were depicted, but due to the change of scope the diagram was restructured to accommodate the new needs. The previous use case "Create Item" corresponds to the present use case "Submit Document" and the previous "Browse and Explore Public/Private Views" corresponds to the present browsing use cases. Concerning the actors, the Passive End and Anonymous Users evolved into User, and the Operational involved into the Expert. Besides these, there are no more direct correspondences. In Figure 4.1 are proposed the new use cases of the system, and after that a detailed explanation of each one.
Figure 4.1: System Use Cases

Table 4.1: **Browse Document**

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Browse Document</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>Expert, User</td>
</tr>
<tr>
<td>Goal</td>
<td>Navigate through the document contents</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>The document is already submitted in the system</td>
</tr>
<tr>
<td>Main Scenario</td>
<td>1. The user accesses the application;</td>
</tr>
<tr>
<td></td>
<td>2. Chooses a document to be viewed;</td>
</tr>
<tr>
<td></td>
<td>3. The document is presented in a specific view.</td>
</tr>
</tbody>
</table>

Table 4.2: **Browse Document Collection**

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Browse Document Collection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>Expert, User</td>
</tr>
<tr>
<td>Goal</td>
<td>Present the list of documents in the system</td>
</tr>
<tr>
<td>Main Scenario</td>
<td>1. The user accesses the application;</td>
</tr>
<tr>
<td></td>
<td>2. The user selects the option of listing documents;</td>
</tr>
<tr>
<td></td>
<td>3. The view with the list of documents is presented.</td>
</tr>
</tbody>
</table>
Table 4.3: **Submit Document**

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Submit Document</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>Expert</td>
</tr>
<tr>
<td>Goal</td>
<td>Submit an ISO standard document into the system</td>
</tr>
</tbody>
</table>
| Pre-Conditions  | • The system is up and running;  
|                 | • The user submits the PDF and the XML or TXT file;  
|                 | • The XML or TXT file is structured. |
| Main Scenario   | 1. The user accesses the application;  
|                 | 2. The user submits the set of files into the system. |

In the use case presented in Table 4.3 there is a certain workflow that must be followed in order to obtain the XML or TXT file and structure it. That workflow is described in Figures 4.2 and 4.3.

![Figure 4.2: PDF conversion to XML](https://acrobat.adobe.com/pt/pt/products/pdf-reader.html)  
![Figure 4.3: PDF conversion to TXT](https://acrobat.adobe.com/pt/pt/products/acrobat-standard.html)

Before the user submits the ISO standard, the user must have in his possession the correspondent XML or TXT file. As it was mentioned in section 3.1 in the previous project[3] the MoReq 2010 [4] specification was available in the XML format, which in turn was structured in a hierarchic fashion. Regarding ISO standards, there weren’t any files available in this format. To solve this issue, one used the presented technology in section 2.3, namely Adobe Acrobat Reader DC[1] and Adobe Acrobat Standard DC[2]. One might think that only the second option would be enough because as depicted in section 2.3 it provides the contents in a more structured way, but the fact that this application is shareware, means that eventually for keep using it the user would have to buy the application. In order for not limiting users...
in such a manner, there is also the possibility of using the first option, despite the fact that the resulting
file does not have a mark-up language for structuring the contents.

Regarding Figure 4.2, the user opens the target PDF with Adobe Acrobat Standard DC and exports it as XML. In Figure 4.4 that process is presented.

Figure 4.4: Exporting the XML file from Adobe Acrobat Standard DC

Regarding Figure 4.3, the process is the practically the same. The user opens the target PDF with Adobe Acrobat Reader DC and saves it as text. In Figure 4.5 that process is presented.

Figure 4.5: Exporting the TXT file from Adobe Acrobat Reader DC
Even though both processes are very expedite there are some aspects that must be reviewed:

- Regardless of being the XML or TXT file, it must be checked if in the beginning of the file exists the Contents section with the titles of every chapter and section of the document spelled correctly, without extra spaces and each title in the same line, not in different lines. Every chapter or section that exists in the Contents must be spelled correctly across the document. These steps correspond to the Structure Index activity in Figures 4.2 and 4.3.

- In the case of the TXT file, since there is no mark-up language, the tables from the document can’t be reassembled because there is no associated structure. And so, the tables content must be identified with special characters in the beginning ("<table name="Table Name">
</table>") and in the end also ("<table>")

After the conversion of the PDF file, the document is ready to be uploaded, including the PDF and the XML or the TXT file.

### 4.2 System Architecture

In Figure 4.6 is presented the System Architecture diagram that comprises the several existing components of the web application, including the components that already existed, the components that were adapted in the context of this thesis and the components that arose from the existing needs of this dissertation and were entirely implemented.

![System Architecture Diagram]

Figure 4.6: System Architecture
Entering in further details, the several components have different colors that have the following meaning:

- The components and packages with the dark grey were previously developed and were reused.
- The components and packages in green, were previously developed also, however they were adapted to the needs of this thesis.
- The components and packages in white didn’t exist in previous versions of the system and were completely implemented during the course of this thesis.

Concerning each package presented in Figure 4.6:

**User Interface** The User Interface package is responsible for giving the user the needed information and also allow the user to interact with application whenever it’s necessary. In order for making the interaction possible, one uses the mechanisms provided by Reqs, i.e. HTML, CSS and AngularJS. It comprises components such as Collection Viewer, Document Viewer and Document Uploader which are very important in the interaction with the user:

- The Collection Viewer presents the list of existing documents in the system by requesting the list to the Collection Controller component, trough the Collection Retrieval interface.
- The Document Viewer, presents information about a previously selected document, comprised by the data stored in Reqs parallel to the original document. The Document Viewer Controller component provides all the required data trough the Document Retrieval interface.
- The Document Uploader consists in the new feature of the system, providing a interface with instructions for uploading a document and a form to be filled upon submission. It provides the Document Upload interface to the Document Uploader Controller component, which in turn will send the submitted files to the Server.

**Controller** The Controller package is where the business logic needed by the views in User Interface package reside. The package is comprised by the following components:

- The Collection Controller is responsible for providing the list of existing documents in the system trough the Collection Retrieval interface. It also has the Data Retrieval port for receiving the list of existing documents from the Data interface that exists in the Routing Services component.
- The Document Viewer Controller is responsible for providing the data from a specific document in the system trough the Collection Retrieval interface. It also has the Data Retrieval port for receiving the list of existing documents from the Data interface that exists in the Routing Services component.
- The Document Uploader Controller receives the files submitted in Document Uploader and uploads them to the Server, trough the File Storage interface, more precisely to Routing Services.

**Server** The Server package is responsible for providing and receiving resources to and from the client-side, as well as executing tasks like data processing from uploads. It also interacts with the Data Persistence package, where the data from each document is stored. It comprises the following components:
• The Routing Services component is responsible for dealing with the requests from the client-side. It receives the data from the Data Repository component through the Retrieve Data interface and provides the data to the controller components through the port Data. It receives the files from the Document Uploader Controller through the File Storage interface, stores the files in the server, and provides the XML or TXT file through the Parser Input interface to Document Parser which in turn resides in the Data Processing package.

• The Data Processing package is the most crucial component for the upload process, because it is where the data is processed and heuristics are applied for preparing a XML object that will be stored in the Data Repository:

  – The Document Parser component is where the XML or TXT files are analysed, heuristics are applied for extracting the data and the XML object required by the Data Repository is created. It receives from Routing Services, through the Parser Input interface, the XML file and provides the XML object, through the Document Import interface, to the Document Import component.

  – The Document Import component will receive through the Document Import interface the XML object that will be stored in the Data Repository component, through the Data Storing interface. The object is analysed from in a top-bottom approach, starting in the higher level elements and finishing in the lower level child elements.

Data Persistence It comprises the Data Repository component. This component consists in a NoSQL database (MongoDB), this is, it provides a mechanism for storage and retrieval of data which is modeled in means other than the tabular relations used in relational databases. The technology used in other components have the ability of communicating easily with MongoDB. The Data Repository component receives data from Document Import through the Data Storing interface, and provides data on a need basis to Routing Services through the Retrieve Data interface. It holds all the data from the existing documents in the system.

In Figure 4.7 is presented the workflow that takes place when a submission occurs.

![Figure 4.7: Document Uploader Workflow](image-url)
The sequence diagram presented in Figure 4.7 shows the interactions between objects in the sequential order that those interactions occur as clearly as possible.

As can be seen, the workflow begins with the User submitting a document in the Document Uploader view, comprised by the original document in PDF, the XML or TXT files and optionally the JSON file which is used for synchronizing the data stored in the system and the original document in the Document Viewer. The view sends these files to the Document Uploader Controller, which in turn will upload them to Routing Services. Routing Services will store the received files in the Server and as response the result of such operation which will be forwarded until it reaches the Document Uploader view that shows a confirmation message to the User as feedback.

Despite the storing of the files in the Server, the correspondent data in the XML or TXT file, still has to be stored in the Data Repository. For that purpose, the Routing Services send the file to the Document Parser which in turn will iterate upon it by, extracting the index, creating the required XML object by the Data Repository and filling it with data. Then the resultant XML object is sent to Document Import which will parse it in a top-down approach, starting in the higher level elements and finishing in the lower level child's, and store each element in an hierarchic fashion, in the Data Repository. When the storing is complete, a return message is forwarded until the Document Uploader Controller.

Aspects concerning the implementation of each component will be approached in Chapter 5.
Chapter 5

Implementation

In this chapter, all the decisions that were made during the development of the solution will be explained. Each section corresponds to the implementation details of the Collection Viewer, Document Viewer and Document Uploader as well as the structure of the files that the application deals with.

5.1 Structure of the extracted files

In this section, the structure of the files extracted, this is, the structure from the XML and TXT files that is used as input for the Document Parser component.

5.1.1 XML structure

During the development of this dissertation several standards were used for testing (see section 3.1). When using Adobe Acrobat Standard DC for extracting the XML from the PDF, the result was an XML file with a specific structure.

![Data Model applied to ISO-15489-x and ISO-26122](image)

Figure 5.1: Data Model applied to ISO-15489-x and ISO-26122
In all standards used for testing, the general structure was practically the same and the Data Model very similar to the one presented in Figure 2.4. However some variations occurred which influenced the way these files are analyzed by the Parser component. The most common structure found is in the XML files from standards ISO-15489-x[5][6] and ISO-26122[11], and the associated Data Model is presented in Figure 5.1.

Exploring the model in Figure 5.1 a Document has only an associated Structure and vice-versa. The Structure is composed by the Part element that comprises all the existing data. These relations are verified in all structure variations.

The Part element is divided in Sect elements, and might have several of them, but a Sect element is only associated to a Part element. The Sect element might be comprised by other Sects as well. The Part and Sect elements always have associated Content that can be of several types. An example in listing 5.1 demonstrates the mentioned elements in the XML file:

```
1 <Part>
2  <H1>TECHNICAL ISO/TR REPORT 26122 </H1>
3  <P>First edition 2008−06−15 </P>
4  <Sect>
5   <H3>Information and documentation – Work process analysis for records </H3>
6   <P>Information et documentation – Analyse du processus des <<records>> </P>
7   <Sect>
8     <H4>3 Terms and definitions </H4>
9     <P>For the purpose of this document, the terms and definitions given in ISO 15489−1 and ISO 15489−2, ISO 23081−1 and ISO 23081−2, and the following apply. </P>
10    <Sect>
11     <H5>3.1 documentation </H5>
12     <P>collection of documents describing operations, instructions, decisions, procedures and business rules related to a given function, process or transaction </P>
13    </Sect>
14  </Sect>
15  <Sect>
16   <H5>3.2 functional analysis </H5>
17   <P>grouping together of all the processes undertaken to achieve a specific, strategic, goal of an organization, which uncovers relationships between functions, processes and transactions which have implications for managing records </P>
18  </Sect>
19  <Sect>
20   <H5>3.3 sequence </H5>
21   <P>series of transactions connected by the requirement that undertaking a later transaction is dependent on completing earlier transactions </P>
22  </Sect>
23 </Sect>
24 </Part>
```

Listing 5.1: XML example from ISO-26122

The example in 5.1 is just a brief passage, however, allows the demonstration of such elements. Continuing with the analysis of the Data Model, the Part and Sect elements have Content of several types:

**Figure** - The Figure element consists in the images that the original document might have. Each Figure has only a ImageData element and vice-versa. The ImageData has the source attribute that consists in a String with the path for the image in the file system. The following listing presents an example of an image:
Listing 5.2: Figure example from ISO-26122

Header - The Header element consists in the title of a chapter, section or subsection and it's always the first element that occurs inside a Sect element. These elements are identified by the `<H>` tag. An example is presented in the following listing:

```
1 <Sect>
  2 <H3>Foreword</H3>
  3 </Sect>
```

Listing 5.3: Header example from ISO-26122

Paragraph - The Paragraph element is another type of Content very common in the XML file identified by the `<P>` tag. Is usually used for sentences and always occurs inside Part or Sect element, like in the following example:

```
1 <Sect>
  2 <H4>2 Normative references</H4>
  3 <P>The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies. </P>
  4 </Sect>
```

Listing 5.4: Paragraph example from ISO-26122

The Paragraph element is used for the index. The index is identified by a Sect with a specific Header element as can be seen in listing 5.5:

```
1 <Sect>
  2 <H3>Contents Page</H3>
  3 <P>Foreword</P>
  4 <P>Introduction</P>
  5 <P>1 Scope</P>
  6 </Sect>
```

Listing 5.5: Index example from ISO-26122

Table - The Table element is comprised by the TableRow element which in turn is comprised by the TableCell element. Each TableCell is only associated to a TableRow, and the latter is only associated to a Table. The TableRow is identified by the `<TR>` tag and the TableCell by the `<TH>` or `<TD>` tags. The following listing presents a case of Table:
Besides the structure presented in Figure 5.1, another type of structure was found, in the extracted XML file from ISO-18128[10]. The Data Model associated is presented in Figure 5.2.

**Listing 5.6: Table example from ISO-26122**

```xml
<Table>
  <TR>
    <TH>Term</TH>
    <TH>Source</TH>
    <TH>Example 1 (in a university)</TH>
    <TH>Example 2 (in a medical practice)</TH>
  </TR>
  <TR>
    <TD>Function</TD>
    <TD>ISO/TS 23081−2:2007</TD>
    <TD>Research</TD>
    <TD>Patient services</TD>
  </TR>
  <TR>
    <TD>Aggregate of processes</TD>
    <TD>This Technical Report</TD>
    <TD>Funding of research</TD>
    <TD>Examination, diagnosis and treatment of patients</TD>
  </TR>
  <TR>
    <TD>Process</TD>
  </TR>
</Table>
```

**Figure 5.2: Data Model applied to ISO-18128**
As can be seen in Figure 5.2, the Data Model is very similar, however with some changes. In this model, the same rules regarding the associations of Document, Structure, and Part elements are applied, but when comparing with the previous model in Figure 5.1, it's easily concluded that there are no Sect elements as well as Header elements that identify the titles of chapters, sections, or subsections. Without Sect elements, the notion of hierarchy between chapters, sections, and subsections is severely compromised. The solution for this problem will be explained in section 5.5.1. An example of the structure is presented in the following listing:

```
1 <Part>
2 <P>Foreword</P>
3 <P>ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.</P>
4 <P>Introduction</P>
5 <P>All organizations identify and manage the risks to their functioning successfully. Identifying and managing the risks to records processes and systems is the responsibility of the organization’s records professional.</P>
6 <P>This Technical Report is intended to help records professionals and people who have responsibility for records in their organization to assess the risks related to records processes and systems.</P>
7 </Part>
```

Listing 5.7: XML example from ISO-18128

In this excerpt, it is possible to note that all the content is enclosed in the Part element. In this model, there are some new elements like the TableOfContents. The TableOfContents element comprises the index, and is identified by the tag `<TOC>`. Each chapter, section, or subsection of the index occurs in a TableOfContentsElement identified by the tag `<TOCI>`. An excerpt is presented in listing 5.8:

```
1 <TOC>
2 <TOCI>Contents Page</TOCI>
3 <TOCI>Foreword</TOCI>
4 <TOCI>Introduction</TOCI>
5 <TOCI>1 Scope</TOCI>
6 <TOCI>2 Normative references</TOCI>
7 <TOCI>3 Terms and definitions</TOCI>
8 </TOC>
```

Listing 5.8: Table of Contents example from ISO-18128

Contrary to the model presented in Figure 5.1, the index is presented in a distinct way, something that is predicted by the Parser component and explained in section 5.5.1. Another new element in Figure 5.2 is the existence of the BulletList element, which as the name suggests, identifies bullet lists. The BulletList can have several BulletListItem's but the latter only is associated to a BulletList. Each BulletListItem is associated with a BulletListBody which comprises the data. The BulletList is identified by the `<L>` tag, the BulletListItem by the `<LI>` tag and the BulletListBody by the `<LBody>` tag as can be seen in listing 5.9:
a) the nature and types of consequences to be included and how they will be measured;

b) the way in which probabilities are to be expressed;

c) how a level of risk will be determined;

d) the criteria by which it will be decided when a risk needs treatment;

e) the criteria for deciding when a risk is acceptable and/or tolerable;

Listing 5.9: Table of Contents example from ISO-18128

Regarding the Paragraph and Table elements, all the stated associations depicted in Figure 5.1 are still maintained, except for the fact that inside each TableCell, the data is represented by a Paragraph element. Each TableCell has zero or more Paragraphs inside, and a certain Paragraph is only associated with one TableCell. An example can be seen in listing 5.10:

Listing 5.10: Table of Contents example from ISO-18128
The third and last variation of the structure from the extracted XML files, is presented in Figure 5.3 that concerns ISO-16715-x standards.

![Data Model applied to ISO-16175-x](image)

**Figure 5.3: Data Model applied to ISO-16175-x**

This Data Model is more similar to the one presented in Figure 5.1, however it comprises some elements from the model presented in Figure 5.2 although with some differences.

The model presents the Document, Structure, Part, Sect and Content elements that have the same associations and rules between them. However, the model has a element called BookmarkTree that occurs at the same level as the Structure element.

Similarly to the model in 5.2, it stores the index. The title of each chapter, section or subsection is in the source attribute of the Bookmark element, which in turn is associated to the BookmarkTree element. A BookmarkTree always has one or more Bookmark, and the latter is only associated with the Bookmark element. Inside the Bookmark there is another element, but since it has no significant data, it's disregarded. An example of the BookmarkTree is presented in listing 5.11.

In this model although Sect elements are used and there is a hierarchy between sections due to them, some sections are not comprised in a Sect element. An example can be seen in listing 5.12, where the chapters title "5 OTHER FUNCTIONAL REQUIREMENTS REFERENCED AND EVALUATED" should be comprised by a Sect element that would include all the data concerning the chapter. Instead the chapter title is in a Paragraph element followed by other elements of the same kind that represent the chapters data.
Listing 5.11: Bookmark Tree example from ISO-16175-1
Each organizational and juridical environment likely has established processes designed to ensure the financial and organisational stability of any capital investment. Although potentially conceptually over-simplified, the totality of analyses comprising a business case can be thought of as the collective means by which an organisation ensures this stability in the case of an IT investment, such as records software.

In its simplest form, a business case articulates a variety of analyses that substantiate an acquisition proposal for the expenditure of an organisation’s capital in accordance with its capital asset strategy and inventory control of such investments. In the case of records software acquisition, such a business case might consist of:

- Establish clear performance objectives and evaluation criteria;
- Involve and continually encourage pilot project participants to use the system;
- Perform prototype work sessions with the software before customising it;
- Finalise system design;

The aim of this project is to harmonise multiple existing jurisdiction-specific digital records software specifications in a manner that complies with the general requirements set forth in the International Standard on Records Management, ISO 15489, Parts 1 and 2 (2001), and the International Standard on Records Management Processes – Metadata for Records, Part 1 – Principles and Part 2 – Conceptual and Implementation Issues, ISO 23081 (2006 and 2009). The jurisdiction-specific functional requirements considered in preparing these modules are as follows:
The three data models presented in this section have their own aspects that must be considered upon the data extraction. Such aspects were taken into account in the development of the Parser component that extracts the data from the tags in each structure and is explained in section 5.5.1.

### 5.1.2 TXT structure

As it was explained in section 4.1, besides the XML extraction, there was also the possibility of extracting the contents of an ISO standard to a TXT file, by using the Adobe Acrobat Reader DC which is going to be explained in this section.

Comparing to the richness and context provided by the extraction with Adobe Acrobat Standard DC, the TXT file obtained with Adobe Acrobat Reader DC is much poorer since it hasn’t a mark-up language attached to provide hierarchy amongst sections as well as other aspects such as directly identifying bullet lists for instance. Without the tags provided by the XML there is still an advantage, the file becomes more readable and the parsing task is less complex. Starting by the index, in TXT files, the index is always present in the beginning of the file and can be identified because it always starts with the title Contents:

```
1 Contents Page
2 Foreword
3 Introduction
4 1 Scope
5 2 Normative references
6 3 Terms and definitions
7 4 Undertaking work process analysis
8 4.1 General
9 4.2 Records dimension of work process analysis
10 4.3 Scope and scale of work process analysis
11 4.4 Participants and validation
12 4.5 Responsibilities
13 5 Contextual review
14 5.1 General
15 5.2 Outcomes of the contextual review
16 6 Functional analysis
17 6.1 General
18 6.2 Analysis of the functions
19 7 Sequential analysis
```

Listing 5.14: Index example from ISO-26122
In the listing 5.14 example, only a part of the index is showed, but as can be seen the contents are all there suitable of being stored in a data structure in the Parser component.

With the index stored, is possible to parse the rest of the file in order to find the content of each section. These contents occur in the same order presented in the original document until the end of the file, which in turn guarantees a linear search and identification of sections:

Listing 5.15: Sections example from ISO-26122
Since there are no tags for identifying bullet lists, they must be extracted using other methods. The occurrence of symbols such as '-' or textual points "a)" indicate the existence of bullet lists:

```plaintext
1. Work process analysis is the foundation needed for the following processes used for creation, capture and control of records:
   2. a) identification of records requirements to document a function or other aggregates of processes;
   3. b) development of function-based classification schemes for identification, location and linking of related records;
   4. c) maintenance of links between records and the context of their creation;
   5. d) development of naming and indexing rules and conventions to ensure maintenance of identification of records over time;
   6. e) identification of ownership of records over time;
   7. f) determination of appropriate retention periods for records and development of records disposition authorities;
   8. g) analysis of risk management in records system context;
   9. h) determination of appropriate security protection for records and development of access permissions and security levels.
```

Listing 5.16: Bullet lists example from ISO-26122

However elements like images or tables can’t be extracted because there is no structure that gives context to the Parser component identify and make the extraction. Instead the recommendations in section 4.1 are followed and a reference is used instead, supported by the original standard document. More details will be presented in section 5.5.1.

## 5.2 Structure of the JSON file

In this section is going to be explained the structure of the JSON file used for synchronizing the stored data in the system and the original standard document being displayed in the Document Viewer component.

The structure of the file is very straightforward. It consists in a array with multiple objects. Each object is a record containing the name of the chapter/section and the page number that corresponds to it. The name of the chapter/section must match the one existing in the XML or TXT files. An example of its structure is presented in the following listing:
Listing 5.17: JSON example from ISO-15489-1

How this file is used for its purpose, taking into account where its stored and how its served to the Document Viewer component, will be explained in section 5.4.

5.3 Collection Viewer

As mentioned before, the Collection Viewer presents the list of existing documents in the system.

The Collection Viewer itself consists in a HTML view, and when the view is accessed, the associated controller which is the Collection Controller will deal with the fetching of the list of documents. The part in the view that concerns the list of available documents is the following:

```html
<div class="featurette">
  <h1>Available Documents:</h1>
  <div class="text-center">
    <div class="list-group" style="margin: 10px">
      <button type="button" class="list-group-item ng-repeat="doc in home.docsData ui-sref='anon.base.overview1params({doc: doc.uriId})'">
        {{doc.title.content}}
      </button>
    </div>
  </div>
</div>
```

Listing 5.18: Documents list in the Collection Viewer

Each document is presented in a list, and by selecting one of them, the user will be redirected to the documents page. Through the ng-repeat directive, the view has access to documents data existing on the controller and for each document a button is created with its title.

The controller will send a request to the Server, more precisely to the Routing Services component, trying to access the document list:

```javascript
function getAllDocuments() {
  return ReqServices.getAllDocuments()
    .then(function(docs){
      vm.docsData = docs.data;
    });
}
```

Listing 5.19: Controller Request

The Routing Services has a defined route in such a manner that when the client (browser), tries to access the Uniform Resource Locator (URL) of the Collection Viewer, the Collection Viewer Controller sends the request for the list of documents in the system, and the pre-defined action of Routing Services is to communicate with the Data Repository to consult the list of existing documents.
The request is sent with a specific [URL] and in the Routing Services component, using Express.js, the Server that pre-defined action is executed:

```javascript
publicRoute.get('/docs', function (req, res) {
    DBAccess.documents.getAllDocuments(function (docs) {
        res.send(docs);
    });
});
```

Listing 5.20: Routing Services Reply

After obtaining the information, the Server will return the requested information, through the Routing Services to the Collection Controller. The controller will then supply the data that will populate the view, Collection Viewer as can be seen in Figure 5.4.

![Available Documents:](image)

Figure 5.4: Collection Viewer

## 5.4 Document Viewer

In the current section its presented how the data is retrieved from the Data Repository and presented to the user, concurrently with the original ISO standard PDF in the Document Viewer.

In order to view a document, one must access the Document Viewer or simply select a document to be viewed. Once the document is selected, the client (the browser), sends a request to the Server that will retrieve the data of the selected document to the client.

At a more detailed level, the view which contains the HTML has a associated controller, the Document Viewer Controller which will use a route from the Routing Services, to get the document data. Just like in section 5.3 there is a pre-defined action which in this case is retrieving from the Data Repository a specific document to the client. Not all the data is retrieved because if so, it would take too long to present the data from the Data Repository. Instead, the information is loaded on a need basis, this is, only the chapters of the document are presented as well as the current section that is selected and all of its contents. In case another chapter is selected and that chapter has sections, the list of sections that the chapter comprises will be presented and the content from the first section will be presented as well.

Similarly to Collection Viewer in section 5.3, the HTML view in the Document Viewer also has several ng-repeat directives that are used for showing the documents chapters, sections and subsections as well as the respective content by accessing the data retrieved by the Document Viewer Controller.
The list of existing documents in Reqs is also present in this page, and allows a user to select other
document to be viewed. If such action is done, the rendering process will be repeated. To complete
the Document Viewer, the correspondent PDF of the standard is also presented simultaneously with the
data and synchronized, as it can be seen on Figure 5.5.

![Figure 5.5: Document Viewer](image)

For rendering the PDF, the PDF.js library was used. This library parses raw arrays of bytes into
streams of PDF “bytecode”, compiles the bytecode and then the canvas from HTML is used to draw
the PDF all implemented in the Document Viewer Controller component. Following the best practices,
the PDF files are stored in the server-side, not on the client-side and are served by Routing Services.

The parsing and rendering of the PDF is implemented in the client-side, more precisely in the Doc-
ument Viewer Controller. Features like zoom, previous/next buttons and the current page that is being
viewed had to be also implemented because the library only provided the parsing and rendering of the
PDF. The following listing comprises the correspondent HTML of the view:

```html
<md-whiteframe layout="column" flex="30">
  <div id="buttons">
    <button id="prev">Previous</button>
    <button id="next">Next</button>
    &nbsp; &nbsp;
    &nbsp; &nbsp;
    <span>Page: <span id="page_num"></span> / <span id="page_count"></span></span>
    &nbsp; &nbsp;
    &nbsp; &nbsp;
    <span>Zoom</span>
    <button id="zoomOut">-</button>
    <button id="zoomIn">+</button>
    <span ng-if="docViewer.unsyncmode">Unsync Mode</span>
  </div>
  <div id="viewer" ng-if="!docViewer.nopdfmode">
    <canvas id="the-canvas" style="border:1px solid black;"/>
  </div>
  <div ng-if="docViewer.nopdfmode" style="border:1px solid black; height: 600px;">
    No Pdf available.
  </div>
</md-whiteframe>
```

Listing 5.21: PDF viewer HTML

[https://mozilla.github.io/pdf.js/](https://mozilla.github.io/pdf.js/)
The div element with the several button elements is where the features like zoom, previous/next buttons and the current page that is being viewed are presented. In the div element with the canvas element, is where the PDF is being presented. The process of retrieving the PDF from the server consists in a service implemented in the client-side. The Document Viewer Controller, makes use of this service by sending a request to Routing Services. Since the PDF is stored in the Server itself and not in the Data Repository, therefore there is no need to communicate with the database in this scenario and Routing Services will search for the file and send a response that contains a String with Base64 encoding to the Document Viewer Controller:

```
publicRoute.get('/pdfs/pdfFile/:pdfFilename', function (req, res) {
  var pdfFilename = req.params.pdfFilename;
  var totalPath = dirname + '/../resources/documentPdfs/' + pdfFilename + '.pdf';
  fs.readFile(totalPath, function (err, data) {
    try {
      var cfile = data.toString('base64');
      response.succeeded = true;
      response.response_data = cfile;
      res.send(response);
    } catch (err) {
      response.succeeded = false;
      response.message = err;
      res.send(response);
    }
  });
});
```

Listing 5.22: Routing Services Response

In the controller, the response is analyzed and if succeeded the String is converted into a Uint8Array that is required by PDF.JS, which in turn will use it for rendering purposes. In case the response doesn’t contain the required data for rendering the PDF, the “no pdf mode” is activated and instead of rendering the PDF, the canvas element presented in listing 5.21 will present a “No pdf is available.” message:

```
function getPDF(isoName) {
  return DOCService.getPDF(isoName).then(function (res) {
    if (!res.data.succeeded){
      vm.nopdfmode = true;
      console.log("No pdf is avialable.");
    } else{
      vm.nopdfmode = false;
      var pdf = base64ToUint8Array(res.data.response_data);
      PDFJS.getDocument(pdf).then(function (pdfDoc, ) {
        pdfDoc = pdfDoc;
        if (!document.getElementById('page_count'))
          return;
        document.getElementById('page_count').textContent =pdfDoc.numPages;
        renderPage(pageNum);})
    }
  });
}
```

Listing 5.23: How the response is treated by Document Viewer Controller
The `getPDF()` function is executed when the user accesses the Document Viewer view, and receives as argument the name of the ISO standard document which was selected to be viewed. The rendering of the PDF is responsible by the `renderPage()` function that receives as argument the number of the PDF page that is going to be presented, which is set to the first page by default:

```javascript
function renderPage(num) {
    if (!document.getElementById('the-canvas'))
        return;
    var canvas = document.getElementById('the-canvas'),
        ctx = canvas.getContext('2d');
    pageRendering = true;
    // Using promise to fetch the page
    pdfDoc.getPage(num).then(function (page) {
        var viewport = page.getViewport(scale);
        canvas.height = viewport.height;
        canvas.width = viewport.width;
        // Render PDF page into canvas context
        var renderContext = {
            canvasContext: ctx,
            viewport: viewport
        };
        var renderTask = page.render(renderContext);
        // Wait for rendering to finish
        renderTask.promise.then(function () {
            pageRendering = false;
            if (pageNumPending !== null) {
                // New page rendering is pending
                renderPage(pageNumPending);
                pageNumPending = null;
            }
        });
    });
    // Update page counters
    document.getElementById('page_num').textContent = pageNum;
}
```

**Listing 5.24: PDF rendering**

The `renderPage()` function:

- gets the canvas element by searching for its id;
- gets the page from the PDF with the number that is passed as argument;
- resizes the canvas with a pre-defined scale;
- uses render the page function provided by PDF.JS;
- and then updates the page counter present in the view that informs which page is currently being viewed.

More details about the PDF rendering, concerning features like zoom, previous/next buttons and the current page are presented in Appendix A.

Another feature, is the synchronization between the PDF and the data that is in the system, this is, if one selects a section of the document (in Figure 5.5 the selected section is "Cover" in the left side), the PDF will change to the correspondent page. This is achieved through a JSON file that uses the structure presented in section 5.2. Just like the PDF file, the JSON file is also stored in the server-side and its
served to the client in the same fashion, through a service implemented in the client-side. The Document Viewer Controller sends a request to Routing Services. Since the JSON is stored in the Server itself and not in the Data Repository like the PDF, therefore there is no need to communicate with the database in this scenario and Routing Services will search for the file and send a response that contains the data of the JSON file. In the routing services the action is very similar to the PDF case. Routing Services has a pre-defined action when the JSON file for a specific document is requested:

```javascript
publicRoute.get('/pdfs/pdfIndex/:pdfFilename', function (req, res) {
  try {
    var pdfFilename = req.params.pdfFilename;
    var pdfIndexData = require('./resources/documentIndexes/' + pdfFilename);
    response.succeeded = true;
    response.response.data = pdfIndexData;
    res.send(response);
  } catch (err) {
    response.succeeded = false;
    response.message = err;
    res.send(response);
  }
});
```

Listing 5.25: JSON file response

The file is obtained and as response the data is sent to the Document Viewer Controller which receives it:

```javascript
function getIndex(isoName) {
  if (!vm indx) {
    if (!vm.unsyncmode)
      return DOCService.getPDFIndex(isoName)
        .then(function (index) {
          if (!index.data.succeeded) {
            vm.unsyncmode = true;
            console.log("The sync with the pdf is not available.");
          } else {
            vm.unsyncmode = false;
            vm indx = index.data.response.data;
            setPDFPageAux();
          }
        })
      else
        return // This document is in unsync mode: no index was available from server.
    
    else
      setPDFPageAux();
  }

  function setPDFPageAux() {
    for (var i = 0; i < vm indx.length; i++) {
      if (vm indx[i].Part === vm. selection. model. title. content) {
        setPage(vm indx[i].Page);
        break;
      }
    }
  }
};
```

Listing 5.26: Document Viewer Controller receiving the JSON file
In the controller the flow is pretty straightforward. By receiving the response from the Server, it’s determined whether the JSON is available or not (it could be not available because when the upload of a document takes place, this file is optional). In case the file isn’t available the “unsynchronized mode” will be activated, and on top of the PDF that will be reported to the user. In case it is available whenever a section of the document is selected, the Document Viewer Controller makes use of the index provided by the PDF to synchronize the PDF through the setPDFPageAux() function. The function tests if the selected section title exists in the JSON file, and in case it does, the setPage() function is used by changing the page that is currently being viewed to the page with the number that is received as argument.

5.5 Document Uploader

In this section the uploading process of an ISO standard into the information system Reqs is going to be described. It will be explained how such feature was implemented by exploring the necessary components to make it work.

For the uploading process, a view (Document Uploader) was created with the upload instructions that the user must follow in order to submit a ISO standard into the system. The view also contains a text field that must be filled with the standard name and a button to select the files for submission. In the view the range of files that is accepted is limited to PDF, JSON, TXT and XML files, thanks to the extension of HTML provided by Angular.JS. Regarding the instructions for submission, the review recommendations mentioned in section 4.1 must be applied in the XML or TXT file, and the JSON file that is used for synchronization between the PDF and existing data of the standard in the system must follow the mentioned structure in section 5.4. The most important part of the view is the submission form that is shown in listing 5.27.

```html
<form name="myForm">
  <legend><h4>Please select the pdf, xml/txt and optionally the json file</h4></legend>
  Standard name (i.e. ISO-26122):<br />
  <input type="text" name="standardName" ng-model="standardName" size="31" required>
  <br />
  <button ngf-select type="files" name="files" ng-model="files" multiple ngf-accept=".pdf,.xml,.json,.txt" required>Select Files</button>
  <br />
  Files:<br />
  <ul>
    <li ng-repeat="f in files" ng-show="f != null">{{f.name}} {{f.errorParam}}
      <button ng-click="removeFile(f)">Remove</button>
      <span class="progress" ng-show="f.progress >= 0">= {{f.progress}}%</span>
      <div style="width:{{f.progress}}%" ng-bind="f.progress + '%'"></div>
    </li>
  </ul>
  <li ng-repeat="f in errFiles">{{f.name}} {{f.error}} {{f.errorParam}}</li>
  <ul>
  </ul>
  <button ng-disabled="!myForm.valid" ng-click="uploadFiles(files)">Submit</button>
</form>
```

Listing 5.27: Document Uploader submission form
When a submission occurs, a upload request (using the ng-file-upload[^] library directive) is sent by Document Uploader Controller to the server, more precisely to Routing Services.

```javascript
angular.forEach(files, function(file) {
    file.upload = Upload.upload({
        url: '/api/public/docs/docImport/',
        file: file,
        data: {standardName: scope.standardName}
    });
});
```

Listing 5.28: Document Uploader Controller request

For each file, a request is sent together with the ISO standard name, and in Routing Services, the server recognizes that a client is trying to upload a standard because there is a pre-defined action for the POST method in that specific [URL](https://github.com/danialfarid/ng-file-upload) which is storing each file in different directories, according with the file extension:

```javascript
publicRoute.post('/docs/docImport/', function(req, res) {
    if (req.files != undefined) {
        var newPath = null;
        if (req.files.file.extension == 'pdf')
            newPath = _dirname + '/../resources/documentPdfs/' + req.body.standardName + "." + req.files.file.extension;
        else if (req.files.file.extension == 'xml')
            newPath = _dirname + '/../resources/documentXMLs/' + req.body.standardName + "." + req.files.file.extension;
        else if (req.files.file.extension == 'txt')
            newPath = _dirname + '/../resources/documentTXTs/' + req.body.standardName + "." + req.files.file.extension;
        else if (req.files.file.extension == 'json')
            newPath = _dirname + '/../resources/documentIndexes/' + req.body.standardName + "." + req.files.file.extension;
    }
});
```

Listing 5.29: Routing Services action upon upload

After the storage of the files in the Server, The Routing Services component will send a response that is received by the Document Uploader Controller:

```javascript
file.upload.then(function (response) {
    if (!response.data.succeeded) {
        console.log("An error occurred in the upload.");
    } else if (response.data.succeeded) {
        res = true;
    }
}
if (res)
    alert("The files have been successfully uploaded.");
```

Listing 5.30: Document Uploader Controller receiving the response from the Server

If the storage of the files was successful the user will be notified of such fact by receiving a message saying that the submission was successful. In the case of XML or TXT file, after its stored, the Parser component, will create the [XML](https://www.w3.org/XML) object that is required by the Import component.

[^]: https://github.com/danialfarid/ng-file-upload
5.5.1 Data Processing

In this section the components that comprise the Data Processing package will be presented, including the Parser and Import components.

In order for the ISO standard document to exist in Reqs, Reqs must have a XML object that follows the structure presented in Figure 2.4 because the Data Repository is organized in this fashion and needs to receive such object in order to store it. The Parser component, presented in section 4.2, is responsible for this task, converting the XML or TXT into a XML object, by using the module libxmljs.

The Parser component is implemented in the server-side by using Node.js, thus relieving the client-side from that burden plus the interaction with the Data Repository. The Parser receives as argument the path to the XML or TXT file stored in the server-side provided by the Routing Services component. There are two versions of the Parser implemented, one for each file format. This is due to the fact that the structure is different, which means that the data has to be manipulated in distinct ways.

Regarding the XML Parser, the three Data Model’s presented in section 5.1.1 were taken into account when parsing and extracting the data:

- First the file is opened, parsed and stored using the libxmljs module:

```
1 filesystem.readFile(path, 'utf8', function(err, data) {
2   var xmlDoc = libxmljs.parseXml(data);
3 });
```

Listing 5.31: Parsing of the XML file

- Then the index is obtained according to each Data Model and stored in a vector:

```
1 var index = [];
2 // According to each case
3 index = xmlDoc.getElementsByTagNameXML(contains(.//H3, 'Contents'));
4 index = xmlDoc.getElementsByTagNameXML('TOCI');
5 index = xmlDoc.getElementsByTagNameXML('bookmark');
```

Listing 5.32: Index Storing

These get’s and find’s are provided by the libxmljs module API. They receive as argument an XPATH expression and return XML elements. After the data being stored in the index vector, the getIndex(index) is executed and deletes from the vector unnecessary data.

- After obtaining the index, it will be organized hierarchically in a associative array, that in each position will have as key the title of the chapter or section and, as value, a record with its title and its content, which consists in an array with the subsections (in case there are any). The function responsible for this action is the sepSections(index) which receives the index as argument and organizes the data in the vector "sections". An example of the contents of this vector might be the following:

https://github.com/polotek/libxmljs
Listing 5.33: Index Hierarchy

- With the index organized in the sections data structure the XML object is created:

```javascript
var sections = [
    { title: 'Foreword', content: [] },
    { title: 'Introduction', content: [] },
    { title: '1 Scope', content: [] },
    { title: '2 Normative references', content: [] },
    { title: '3 Terms and definitions', content: [] },
    { title: '4 Undertaking work process analysis', content: [
        { title: '4.1 General', content: [] },
        { title: '4.2 Records dimension of work process analysis', content: [] },
        { title: '4.3 Scope and scale of work process analysis', content: [] },
        { title: '4.4 Participants and validation', content: [] },
        { title: '4.5 Responsibilities', content: [] }
    ]};
```

Listing 5.34: Creation of the XML object

And in a recursive fashion, formDoc(doc.get('//document')) function (receiving as argument the root of the created XML object), will be executed upon the sections vector and will populate the doc object according to the data in each entry. With the Data Model of Figure 2.4 in mind, that is used as basis for the Data Repository, the elements are created in the XML object:

```javascript
function formDoc(element){
    for(var i = 0; i < sections.length; i++)
        if(Object.keys(sections[i].content).length > 0)
            element = auxFormDoc(sections[i], element, 'page', {id: i+1, title: sections[i].title});
        else
            element.node('page').attr({id: i+1, title: sections[i].title.trim()});
    return element;
}
```

Listing 5.35: Populating the XML object with chapters and sections
In the formDoc() function, the sections associative array is analyzed. Each position is accessed, and in case the chapter has no sections the Page node is created in the XML object, else the auxFormDoc() function is called for dealing with the sections. In the auxFormDoc() function the same rule is applied and in case there are more subsections the function is called in a recursive fashion for the remaining subsections. This is done until all entries of the sections vector are present in the XML object.

- With the XML object properly organized the content of each chapter, section and subsection can be extracted from the xmlIDoc object and populate the doc object. With the three Data Models presented in section 5.1.1 three different methods had to be implemented to correspond to the needs of each structure:

  - Taking into account the model in Figure 5.1 the following method was implemented:
    
    ```
    populateDoc(xmlDoc, doc);
    
    Listing 5.36: Function populateDoc
    ```
    
    This method, populates the doc object in an recursive fashion. It relies in the Sect elements that comprise each chapter, section or subsection. It searches the index vector that contains all chapters, sections and subsections, and for each title tries to find it in the xmlIDoc through an XPATH expression. If it finds the respective title, another XPATH expression is used to obtain the node in the doc object and that node is filled with the respective contents. If a Sect element is found, the auxiliary method populateDocAux() is called to deal with that Sect element. In the auxiliary method the same rule applies, and if a new Sect is found, the method is called recursively. This will last until the end of the index vector is reached and the doc object is fully populated.

  - With the model in Figure 5.2 in mind, the following method was implemented:
    
    ```
    populateDocTOCI(xmlDoc, doc);
    
    Listing 5.37: Function populateDocTOCI
    ```
    
    The method iterates over the index vector and the contents from the xmlIDoc. The contents of the xmlIDoc are obtained using the following XPATH expression:
    
    ```
    xmlDoc.find('//@Part/TOC/following-sibling::*');
    
    Listing 5.38: XPATH expression for obtaining the content from the xmlIDoc object
    ```
    
    Since there are no Sect elements, the data occurs linearly, and if the current element matches the position in the index vector, the following elements are going to populate the doc object in the respective section until another match is verified. This is done until the end of the index array.

  - Concerning the model in Figure 5.3 another method was developed:
    
    ```
    populateDOCBookmark(xmlDoc, doc);
    
    Listing 5.39: Function populateDOCBookmark
    ```
This method makes use of the Sect elements also, but in this case not all chapters, sections or subsections have a Sect associated, this is, multiple sections might occur inside the same Sect element. Similarly to Function populateDoc() in listing 5.36, the Sect elements are analyzed but in an iterative way with the help of an auxiliary method, the populateDOCBookmarkAux() that uses the matching system in Function populateDOCBookmark mentioned in listing 5.37

- For converting the contents from the xmlDoc to the format in 2.4, the auxPopulateDoc() function. The function determined according to the used tags, the right template to be used.

Regarding the TXT Parser, the structure presented in 5.1.2 was taken into account when parsing and extracting the data:

- First the file is "read" trough a read stream which is provided by a function from the file system:

```
var readline = readline.createInterface({input: filesystem.createReadStream(path)});
```

Listing 5.40: Reading the TXT file

- Each line from the file is "read", and while on reading, the index array is built by identifying the "Contents" section in the buildIndex() function:

```
rl.on('line', function (line) {
  buildIndex(line);
});
```

Listing 5.41: Building the index from the TXT file

- After obtaining the index, it will be organized hierarchically in a associative array, just like in listing 5.33 and using the same function that receives the index, sepSections(index).

- The XML object, doc, is created just like in listing 5.34 and is populated recursively using the same method, formDoc(doc.get('//document')) that receives as argument the root node of the doc object.

- With the XML object properly organized the content of each chapter, section and subsection can be extracted from the TXT file and populate the doc object. This is done when the read stream is about to be closed:

```
rl.on('close', function (line) {
  populateDoc(file, doc, index);
});
```

Listing 5.42: Populating the doc object with the data from the TXT file

The populateDoc() function receives as argument the file array that in each position has a line from the TXT file, the doc XML object with the chapters, sections and subsections all structured and the index array that was previously populated.
function populateDoc(originalDoc, newDoc, index) {
    var element = null,
        paragraph = null,
        k = 0;
    for (var i = 0; i < index.length; ){
        var found = false,
            inTable = false;
        for (; ; k++) {
            if (index[i] == undefined)
                break;
            if (existsSection(index[i], originalDoc, k, originalDoc[k])) {
                if (found)
                    break;
                found = true;
                var name = index[i];
                if (i < index.length - 1)
                    i++;
                element = newDoc.get('//document//*[@title="' + name + '" ]');
                element = element.node('paragraphs');
            } else if (originalDoc[k].trim() != "" && found){
                if (originalDoc[k].startsWith('<table')){
                    inTable = true;
                    populateDocAux(originalDoc[k].split('""') [1], paragraph, element);
                }
                if (inTable){
                    if (originalDoc[k].trim() == "</table>")
                        inTable = false;
                } else
                    populateDocAux(originalDoc[k], paragraph, element);
            }
            i++;
        }
    }
}

Listing 5.43: Function for populating the XML object

The function iterates the index array and the file array that contains all the contents from the TXT file. The function will compare by string matching, each position in the file array with the current position in the index array. If a match is obtained, the respective node in the doc object is obtained through an XPath expression and the node will be populated with the contents of that respective sections until the next match is obtained. Then the process is repeated until the end of the index and file arrays.

- The populateDoc() function does not consider Tables because there is no structure that allows their construction. Instead, according to the recommendations in section 4.1, Table content is identified with special delimiters and when that is verified in the function, only a reference to the table (the table name) is placed in the doc object.

- The populateDocAux() function is used for providing the template that is going to fill the doc object. It’s determined by string matching whether a line is a paragraph or a bullet from a bullet list:
function populateDocAux(vec, el1, el2) {
    if (vec.charAt(0) == "-" || vec.charAt(1) == ")")
        el1 = el2.node('bullet');
    else
        el1 = el2.node('paragraph');
    el1.node('text', vec.trim());
    el1.node('links');
    el1.node('bolds');
    el1.node('italics');
    el2.addChild(el1);
}

Listing 5.44: Determining the type of element

By order, the function receives as argument the line from the TXT file, the node to be created and
the parent node. If the current line is a bullet it might have those special characters, else it's a
paragraph.

Each chapter section, or subsection comprises a set of content that can go from simple paragraphs
with text, to bullet lists, tables or images. To identify each one a certain line of thought had to be used.
Generally, when the Parser starts analysing the content of a chapter/section or subsection, elements are
also considered to be paragraphs with text. Bullet Lists, Tables and Images, in the XML file case, are
always identified with a specific element, which allows the direct mapping into the structure presented
in Figure 2.4 In the TXT case, since there is no mark-up language, when it comes to tables, there
is no structured associated, thus there is no way of knowing how many rows or columns a table has
and to which cells the content belongs without human intervention. In this case, the recommendation
concerning the tables content is applied and the parser will not consider the content, it will only place
the table name in the XML object. Still in the TXT case, bullet lists are identified due to the existence of
special symbols like ")" or "-". After identifying which kind of element the Parser is dealing with, the data
is extracted to the correct element template that is stored afterwards in the XML object.

Regarding Figure 2.4, there are elements which are impossible to extract due to the lack of context,
for example if a certain word is a link, written in bold or in italic. Therefore, these are not considered.
As one can see, the Parser relies strongly in the Contents section, it's used as basis for populating the
XML object. This is the reason why the recommendations presented in section 4.1 must be followed.

After the Parser has completed its task and the XML object is fully populated, the Document Import
component is used to store the object in the Data Repository component. The process of storing is
pretty forthright:

- The object is received as argument, provided by the Document Parser component;
- The object is parsed (again with the help of the libxmljs module) and the major element of the
  object that comprises all chapters is obtained with the following XPATH expression:

```javascript
var docEls = mainNode.find(‘document’);
```

Listing 5.45: Obtain all chapters

- The document is created in the Data Repository and in an asynchronous way the Import com-
  ponent will access every page element (represents the main chapters of the document) and will
  explore its child nodes (other sections and content);
• While the Document Import is moving through the nodes, it’s storing each one of them in a hierarchical fashion until the lowest level that exists;

• After creating an entry for the Document, then will create entries for each Page as well as the association with the Document.

• Inside pages, there can be “Section Containers” (sections with subsections inside), "Section Contents” (only have paragraphs elements) and paragraphs (can contain, several paragraph elements, images, bullet lists and tables):

```javascript
switch(childName){
  case 'sectionContainer':
    processSectionContainer();
    break;
  case 'sectionContent':
    processSectionContent();
    break;
  case 'paragraphs':
    processParagraphs();
    break;
  default:
    console.log('[INFO] - ProcessPage: Ignoring unknown tag found -' + childName);
    finishChild();
    break;
}
```

Listing 5.46: Import Component processing the Page content

Since the structure of the object, matches the structure of the Data Repository component (see Figure 2.4), every node is added without any issue and the ISO standard is finally embed in the system. When the storing ends, the Routing Services component is alerted and communicates to the controller that the storage was successful.
Chapter 6

Demonstration

In this chapter are presented the results of the implementation of the proposed solution depicted in section 1.2. With the help of screenshots several details of the application are shown, including the integration of ISO standards data and the respective correspondence with the original document. The performance of the ISO standards upload service will also be approached.

6.1 "Reqs" Platform

As mentioned before in section 2.4, a web application was already built, "Reqs", and the work of this dissertation was developed on top of such application. The application provides navigation into the several ISO standards that are stored in the system. During the development this dissertation, several standards were uploaded into the system, namely: ISO-15489-1, ISO-15489-2, ISO-16175-1, ISO-16175-2, ISO-16175-3, ISO-18128 and ISO-26122. The application has several sections, but the ones that are related to this dissertation are: Home, Overview, Document Viewer and Document Uploader. The Home page can be seen in Figure 6.1.

![Figure 6.1: Home page for Reqs Application](image)
The Home page presents the list of documents that are available in the system for the user to browse. A click in any of the documents redirects the user to the Overview page, presented in Figure 6.2.

**Figure 6.2: Overview page for Reqs Application**

The Overview page allows for a user to navigate through the document with the help of an index that appears on the left panel of the screen, listing all the chapters of the document. After choosing an option on the left panel, the right panel shows information of the document regarding that option. A similar page to this one is presented in Figure 6.3, the Document Viewer page.

**Figure 6.3: Document Viewer page for Reqs Application**

The Document Viewer page, just like the Overview page, allows a user to navigate through the document with the help of an index in the left panel and the correspondent data to the section that is currently selected in the middle panel. In figure 6.3 is presented an example in which chapter "1. Scope" is selected and in the middle panel the correspondent data is presented. In the right panel is available the original document that is currently selected together with features such as zoom, previous and next.
page and the current page being viewed. Still in this Figure, one can assess the consistency of the data stored in the database with the original document and as it can be seen the paragraphs and bullet lists are exactly the same. To improve the users experience, when a user selects a section in the left panel, the data is presented in the middle panel and in the right panel the view synchronizes to the correspondent PDF page. If synchronization isn’t possible (because no JSON file was provided in the upload process), the user is alerted of such fact as it can be seen in Figure 6.4 (unsynchronized mode is activated).

Figure 6.4: Document Viewer in unsynchronized mode

While in Document Viewer, its also possible to select which document will be viewed as it can be seen in Figure 6.5

Figure 6.5: Document selection in Document Viewer page
The Document Upload page is presented in Figure 6.6.

Figure 6.6: Document Upload page for Reqs Application

In this page, all the instructions necessary to upload a standard are provided as well as examples of the content of the files to be uploaded. A form is also present in this view, in which a user provides the standard name and selects the required files for the submission. In case the standard name isn’t written according to the guidelines a message is provided to the user, in order to correct the name. In the selection of files, if files with other formats are selected or no files are selected at all, a message is also provided to the user with suggestions of what can be done in order to successfully submit a standard. After the files have been successfully uploaded, a message is provided as feedback to the user.

6.2 Upload Procedures

In this section is presented an example of the uploading procedures in order to submit a document - ISO 26122 in this case - into the Reqs system.

1. The PDF of the standard to be submitted shouldn’t be locked (without attached restrictions). If it is, the solution would be to unlock it here.

2. The PDF must be opened in Adobe Acrobat Standard DC, by going to File->Export to->XML and saved like in the following figure.
In case the application isn’t available, the **PDF** must be opened in Adobe Acrobat Reader DC, by going to File->Save as Other->Text and saved like in the following Figure.

3. The following aspects must be complied:

   - The index contains all the chapters (and subchapters as well) without spelling errors, extra spaces or in different lines just like in the following listings.
Listing 6.1: Structured Index of ISO 26122 XML File

1. Contents Page
2. Foreword
   1. Introduction
3. 1 Scope
4. 2 Normative references
5. 3 Terms and definitions
6. 4 Undertaking work process analysis
7. 4.1 General
8. 4.2 Records dimension of work process analysis
9. 4.3 Scope and scale of work process analysis
10. 4.4 Participants and validation
11. 4.5 Responsibilities
12. 5 Contextual review
13. 5.1 General
14. 5.2 Outcomes of the contextual review
15. 6 Functional analysis
16. 6.1 General
17. 6.2 Analysis of the functions
18. 7 Sequential analysis
19. 7.1 General
20. 7.2 Identifying the sequence of transactions in a process
21. 7.3 Outcomes of the analysis of the sequence of transactions in a process
22. 7.4 Identifying and analysing the variations to the process
23. 7.5 Outcomes of the analysis of variations to the process
24. 7.6 Establishing the rules governing the identified constituent transactions
25. 7.7 Outcomes of the analysis of the rules base for transactions

Listing 6.2: Structured Index of ISO 26122 Text File

1. Contents Page
2. Foreword
3. Introduction
4. 1 Scope
5. 2 Normative reference
6. 3 Terms and definitions
7. 4 Undertaking work process analysis
8. 4.1 General
9. 4.2 Records dimension of work process analysis
10. 4.3 Scope and scale of work process analysis
11. 4.4 Participants and validation
12. 4.5 Responsibilities
13. 5 Contextual review
14. 5.1 General
15. 5.2 Outcomes of the contextual review
16. 6 Functional analysis
17. 6.1 General
18. 6.2 Analysis of the functions
19. 7 Sequential analysis
20. 7.1 General
21. 7.2 Identifying the sequence of transactions in a process
22. 7.3 Outcomes of the analysis of the sequence of transactions in a process
23. 7.4 Identifying and analysing the variations to the process
24. 7.5 Outcomes of the analysis of variations to the process
25. 7.6 Establishing the rules governing the identified constituent transactions
26. 7.7 Outcomes of the analysis of the rules base for transactions
4. A JSON file (optional) can be provided in order to synchronize the view between the PDF and the data in the system. If it's provided, it must follow the example in listing 6.4.

```json
[
   { "Part": "Cover", "Page": 1 },
   { "Part": "Contents", "Page": 2 },
   { "Part": "Foreword", "Page": 4 },
   { "Part": "Introduction", "Page": 5 },
   { "Part": "1. Scope", "Page": 7 },
   { "Part": "2. Normative references", "Page": 8 },
   { "Part": "3. Terms and definitions", "Page": 10 },
   { "Part": "4. Undertaking work process analysis", "Page": 11 },
   { "Part": "5. Record dimension of work process analysis", "Page": 12 },
   { "Part": "6. Scope and scale of work process analysis", "Page": 13 },
   { "Part": "7. Participants and validation", "Page": 14 },
   { "Part": "8. Responsibilities", "Page": 15 },
   { "Part": "10. General", "Page": 17 },
   { "Part": "11. Outcomes of the contextual review", "Page": 18 },
   { "Part": "12. Functional analysis", "Page": 19 },
   { "Part": "15. General", "Page": 22 },
   { "Part": "16. Sequential analysis", "Page": 23 },
   { "Part": "17. General", "Page": 24 },
   { "Part": "18. Identifying the sequence of transactions in a process", "Page": 25 },
   { "Part": "19. General", "Page": 26 },
   { "Part": "20. Analysis of the sequences of transactions in a process", "Page": 27 },
   { "Part": "22. Identifying the variations to the process", "Page": 29 },
   { "Part": "23. General", "Page": 30 },
   { "Part": "24. Outcomes of the analysis of variations to the process", "Page": 31 },
   { "Part": "25. General", "Page": 32 },
   { "Part": "26. Establishing the rules governing the identified constituent transactions", "Page": 33 },
   { "Part": "27. General", "Page": 34 },
]
```

Listing 6.4: Structured JSON of ISO 26122

The counting starts from page 1, and the page of a chapter or sub-chapter is the page where it appears in the PDF.
5. In the Document Uploader page in Reqs web application, the form must be filled with the standard name, and the files must be selected and submitted just like in the following Figure.

![Figure 6.7: Uploading form](image)

6.3 "Reqs" Upload Service Performance

In this section, some results will be presented, namely the upload time of the standards that were used for testing, in order to assess the performance of the developed solution.

There are several aspects that influence the uploading time of a standard into Reqs, for example the time that takes for a file to be stored in the server or to store data in the Data Repository. In order to assess the performance of Reqs when importing a document into the system, some tests were made using several documents presented in Table 6.1.

In order to test it, Reqs was running in `localhost` and the Data Repository in a Virtual Private Server (VPS). The `localhost` consists in a machine with an i5 processor with 4 cores and 8GB of Random-access memory (RAM). The VPS was comprised by a processor with 8 cores and 16GB of RAM.

In Table 6.1, are presented the results from the executed tests, concerning the storing time of the data into the Data Repository. The Table is composed by the standards name that were tested, the approximate storing duration in the database, and the number of elements that each respective XML object contained.
Table 6.1: ISO standards storing duration

<table>
<thead>
<tr>
<th>ISO Standard Name</th>
<th>Storing Time</th>
<th>Number of Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISO 15489-1</td>
<td>± 10 min.</td>
<td>2743</td>
</tr>
<tr>
<td>ISO 15489-2</td>
<td>± 52 min.</td>
<td>7292</td>
</tr>
<tr>
<td>ISO 16175-1</td>
<td>± 7 min.</td>
<td>1322</td>
</tr>
<tr>
<td>ISO 16175-2</td>
<td>± 50 min.</td>
<td>7188</td>
</tr>
<tr>
<td>ISO 16175-3</td>
<td>± 43 min.</td>
<td>6606</td>
</tr>
<tr>
<td>ISO 18128</td>
<td>± 25 min.</td>
<td>4818</td>
</tr>
<tr>
<td>ISO 26122</td>
<td>± 11 min.</td>
<td>2894</td>
</tr>
</tbody>
</table>

As can be seen in Table 6.1, there is a correlation between the storing time and the number of elements to be stored in the database, this is, as the number of elements rises, time also increases. The several factors that influence the upload service in Reqs are:

- the time that a file takes to be stored in the server;
- the conversion time in order to obtain the XML that will be stored in the database;
- and the storing time that is analyzed in Table 6.1.

From the mentioned aspects, the first two are almost immediate. From the results that are presented in Table 6.1, the storing time in the Data Repository is the bottleneck, this is, the aspect that has more impact on the total uploading duration and the one that needs to be improved. Considering the results, the storing isn’t the most expeditious process and in operational terms the time spent in storage isn’t acceptable. It should be faster and up to ten minutes.

The following improvements could reduce the storing time in the Data Repository:

**Hardware** - The capability of the storage system creates some important physical limits that might influence the performance of write operations provided by the Data Repository (MongoDB). Many unique factors related to the storage system of the drive affect write performance, including random access patterns, disk caches, disk readahead and RAID configurations. Using Solid state drives (SSD) instead of Hard disks (HDD) could contribute to an improvement in performance.

**Journaling** - To provide durability in the event of a crash, the Data Repository (MongoDB) uses write ahead logging to an on-disk journal. It writes the in-memory changes first to the on-disk journal files. If MongoDB should terminate or encounter an error before committing the changes to the data files, MongoDB can use the journal files to apply the write operation to the data files. If the journal and the data files reside on the same block device, the data files and the journal may have to contend for a finite number of available I/O resources. Moving the journal to a separate device may increase the capacity for write operations.
Chapter 7

Conclusions and Future Work

In the following chapter the main obstacles faced during the development are going to be presented, as well as a critical analysis of the developed work including the limitations of the application and the future work to be implemented.

7.1 Main issues faced during the development

In the beginning of the development the obstacles began with the application previously developed, Reqs\[3\]. In order to understand how the system worked, one had to learn the inherent technology - which had a very sharp learning curve - and the already developed components such as the rendering of the MoReq data and the import process into the database.

The first major challenge was to obtain the XML files from the ISO standards used for testing and mentioned in section 3.1. At the time, the Parser component mentioned in section 4.2 had not yet been developed, and so, there wasn’t a automated procedure to obtain the XML with the structure that was required by the database. The solution was to:

- Obtain the XML by using Adobe Acrobat Standard DC;
- Execute Xquery\[1\] expressions upon the XML to automate some parts of the process, for example obtain all section titles or all bullet lists;
- Manually construct the XML required by the database, using the XML from MoReq as guideline.

As one can imagine, this is a great error-prone task, and the standards used for testing and mentioned in section 3.1 were converted to the required format. The average time to convert a document was 3 - 4 days and there were a lot of errors, but those were amended upon revision.

Another obstacle was the quality of the extracted data by the Adobe tools, Adobe Acrobat Reader DC for the Text file and Adobe Acrobat Standard DC for the XML file. In both applications, the output file contained unnecessary content, for example trademarks and others. An example of that is presented in listing 7.1.

To discard such cases the solution was to simply verify in the Parser component - while the file was being analyzed - if that sequence of words or numbers existed. If it did, the Parser component would just ignore the line and move on in the task of analysing the file. Another problem similar to his one, was the fact that the chapter title would appear in different lines of the text file instead of one. For example, in ISO 26122, instead of having "7 Sequential analysis" - which is a chapter of the document - in the same line, it would appear in different lines, like in listing 7.2:

```
7 Sequential analysis
```

The solution was to adapt the Parser component for predicting these cases, by merging the content of different lines in the file. Also in the uploading page that is presented in section 6.1 there are instructions related to this issue that a user must follow before uploading the respective document. For example, analyze the file and make sure that the index contains all chapters, sections and sub-sections, and that each one occurs in the file without any extra spaces or in different lines.

Still related with the quality of the data extracted from ISO standards, more precisely with the Adobe Acrobat Standard DC that extracts the XML, is the fact that the structure of the file isn’t always the same. For instance, considering two distinct documents that were used for testing, ISO 26122[11] and ISO 18128[10] a part of the respective XML files is presented in listings 7.3 and 7.4 relatively to the index of each document.

```
1  <Sect>
2   <H3>Contents Page </H3>
3  
4   <P>Foreword </P>
5  
6   <P>Introduction </P>
7  
8   <P>1 Scope</P>
9  
10  <P>2 Normative references </P>
11  </Sect>
```

Listing 7.3: Part of the XML file from ISO 26122
Both listings present part of the respective index for both documents, however, the elements that hold the textual content have distinct names, fact that was taken into account in the implementation of the Document Parser by considering both structures. Another case related to the structure is the hierarchy of the sections. For example, the general structure might be as it follows:

```
1 <Sect>
  2 <H4>4.4 Participants and validation</H4>
  3 ...
  4
  5 ...
  6 <P>Work process analysis for the purposes of creation, capture and control of records is specific. It describes and analyses processes taking place in organizations in real time and is dependent on accurate information gathering. The participants in the work process are a key source of that information and an important reference for validation of its accuracy.</P>
  7 </Sect>
```

Listing 7.4: Part of the XML file from ISO 18128

```
1 <Sect>
  2 <H4>4.5 Responsibilities</H4>
  3 ...
  4
  5 ...
  6 <P>The head of an organization is responsible for the performance of the organization and for how the organization undertakes its business and conducts its work processes.</P>
  7 </Sect>
```

Listing 7.5: ISO 26122 hierarchic structure example
However, in some XML files, the following example might occur:

```xml
1 <H2>
2 4.2 Risk criteria
3 </H2>
4
5 <P>
6 Regarding the nature and types of consequences to be included in the risk assessment of records processes and systems, there is a general starting point which applies to all organizations. Records which are authentic, reliable, have integrity and are useable for as long as they are required will support the needs of the organization. Risks are identified based on their potential to undermine those general characteristics of records which would make them fail to meet the purposes for which they are created.
8 </P>
```

Listing 7.6: ISO 18128 hierarchic structure example

In listing 7.5 every chapter, section or subsection is included a `<Sect>` element along with the respective title and contents, rule that is always applied in the document. However, as can be seen in listing 7.6 chapters, sections and subsections might not have a `<Sect>` element. The Document Parser, was also adapted to these cases, by including heuristics that predict these structures.

7.2 Critical Analysis of the Work

Throughout this document, all phases of the development of a solution capable of aiding stakeholders with their needs, were presented and discussed. First, one defined the problem as well as the objectives for the project. The second phase consisted in the proposal of a system capable of storing by upload ISO standards, the navigation amongst these documents through a structured index and the visualization of the uploaded data parallel to the original document. The most challenging phase was the development phase, starting by the visualization and manipulation of data parallel to the development of the Import component to store data in the Data Repository. Finally, the upload service was achieved, by developing the Parser component and its interaction with the Import component, as well as the respective view.

Although this thesis is based in another dissertation, the addition of this new way of visualizing the data together with the original ISO standard, as well as the automation of the file embedding process in the system, brought added value to the existing system. In fact - despite not being the main objective - other types of reference requirements documents that a stakeholder needs to perform his work, can also be uploaded to the system as long as the rules for uploading are followed as well as the structure of the files regarding the index part.

The most significant limitation in the application, resides on the time that a ISO standard takes to be uploaded in the system. As explained in section 5.5.1 there is a certain process to be followed in
order for a document to be uploaded in the system. In that process there is a final task in which the Import component stores the data from the document - the XML object - in the Data Repository, and that’s where the bottleneck resides. When the Import component is storing the data in the Data Repository, a XML object that contains all this data is parsed and each element that this object contains is being stored. If the document to upload is large, for example 100 pages, the XML object will have more elements to be parsed and consequently more time the upload will take.

All objectives have been met and the solution works as it should. Additionally, the obtained results looked promising. The required time for embedding a standard in the system has been greatly reduced - from days to minutes - and the process is no longer error-prone.

### 7.3 Future Work

Even though the objectives have been accomplished there’s room for improvements and addiction of functionalities that would make the solution more robust and complete, but couldn’t be implemented, mainly due to time constraints.

The first improvement is related to the upload service, more precisely to the JSON files that provide synchronization between the original document and the data retrieved from the Data Repository. Instead of providing a JSON file for these matters that might contain errors, the system would show to the user, after he submitted the data and the PDF, a form with the index of the document where the page numbers would be inserted, thus improving the users experience. Another functionality, would be a management log page of the ongoing uploads in the system. Such log would have options to resume, pause or remove the current upload, thus adding more robustness to the application. Still regarding the upload, there is the storing time of the data into the Data Repository which can be considerable according to the size of each document. The storage performance could be increased by using SSD instead of HDD. To provide durability in the event of a crash, the Data Repository (MongoDB) uses write ahead logging to an on-disk journal, and by moving the journal to a separate block device may improve performance.

Also, it would be interesting to relate ISO standards amongst themselves and with the MoReq2010 specification in order for a stakeholder to compare their contents, more precisely requirements. For example, a stakeholder would search for a certain requirement and the system would provide as result a list that would contain the name of the documents where that requirement occurred, thus saving time for the stakeholder.
Bibliography


Appendix A

Implementation of the PDF rendering

```
var pdfDoc = null,
pageNum = 1,
pageRendering = false,
pageNumPending = null,
scale = 0.625;

/**
 * Get page info from document, resize canvas accordingly, and render page.
 * @param num Page number.
 */
function renderPage(num) {
  if (!document.getElementById('the-canvas'))
    return;
  var canvas = document.getElementById('the-canvas'),
      ctx = canvas.getContext('2d'),
      pageRendering = true;
  // Using promise to fetch the page
  pdfDoc.getPage(num).then(function(page) {
    var viewport = page.getViewport(scale);
    canvas.height = viewport.height;
    canvas.width = viewport.width;
    // Render PDF page into canvas context
    var renderContext = {
      canvasContext: ctx,
      viewport: viewport
    };
    var renderTask = page.render(renderContext);
    // Wait for rendering to finish
    renderTask.promise.then(function () {
      pageRendering = false;
      if (pageNumPending !== null) {
        // New page rendering is pending
        renderPage(pageNumPending);
        pageNumPending = null;
      }
    });
  });
  // Update page counters
  document.getElementById('page-num').textContent = pageNum;
}
```
function queueRenderPage(num) {
    if (pageRendering)
        pageNumPending = num;
    else
        renderPage(num);
}

/*
 * Displays previous page.
 */
function onPrevPage() {
    if (pageNum <= 1) {
        return;
    }
    pageNum--;
    queueRenderPage(pageNum);
}
document.getElementById('prev').addEventListener('click', onPrevPage);

/*
 * Displays next page.
 */
function onNextPage() {
    if (pageNum > pdfDoc.numPages) {
        return;
    }
    pageNum++;
    queueRenderPage(pageNum);
}
document.getElementById('next').addEventListener('click', onNextPage);

/*
 * Sets the current Page number to num.
 * @param num Page number.
 */
function setPage(num){
    pageNum=num;
    queueRenderPage(pageNum);
}

function zoomOut(){
    if (scale == 0.625)
        return 0;
    else{
        scale -= 0.2;
        queueRenderPage(pageNum);
    }
}
document.getElementById('zoomOut').addEventListener('click', zoomOut);
function zoomIn()
{
    if (scale == 1.3)
        return 0;
    else
    {
        scale += 0.2;
        queueRenderPage(pageNum);
    }
}

document.getElementById('zoomIn').addEventListener('click', zoomIn);

Listing A.1: Implementation of the PDF rendering