Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2018

Ricardo Pimentel Filipe

Sistema multidimensional de armazenamento e classificação de dados

Multidimentional system for storage and classification of data

Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2018

Ricardo Pimentel Filipe

Sistema multidimensional de armazenamento e classificação de dados
Multidimentional system for storage and classification of data

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requesitos necessários à obtenção do grau de Meste em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor José Luis Guimarães Oliveira, Professor associado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president	**Prof. Doutor José Manuel Matos Moreira** Professor Associado, Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro
vogais / examiners committee	**Prof. Doutor Rui Pedro Sanches de Castro Lopes** Professor Coordenador, Instituto Politécnico de Bragança
	Prof. Doutor José Luis Guimarães Oliveira Professor Associado, Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro (orientador)

agradecimentos / acknowledgements

Aos meus pais e irmão por todo o apoio e amor que me dão e pelos sacrificios que me permitiram concluir este percurso.
Ao professor José Luis Oliveira pela sua orientação, disponibilidade e apoio ao longo deste projeto.
À Nicole Hillenbrand por estar sempre presente para partilhar os bons momentos e apoiar nas dificuldades.
A todos os professores e colegas com os quais aprendi, pelo conhecimento partilhado.

Resumo

Com o avanço da tecnologia e a sua larga disponibilidade nos dias de hoje, a informação tem vindo a ser gerada digitalmente, quer se trate de documentos, fotos, vídeos criados por individuos, ou dados produzidos por dispositivos eletrónicos. Esta facilidade em gerar informação cria um enorme aumento na quantidade de dados que estão disponíveis, dificultando assim o acesso à informação pretendida e a associação entre diversos dados.

Esta dissertação tem como objetivo disponibilizar um repositório *online* no qual seja possível armazenar qualquer tipo de documento digital e associar informação, de forma automática e manual, de modo a que estes sejam encontrados com maior facilidade.

Abstract

Nowadays with the advancement of technology and its wide availability, information has been digitally generated, whether documents, photos, videos created by people or data files generated by electronic devices. This creates a huge amount of available data, which causes a burden when it comes to accessing to information and relating different pieces of data.

This dissertation aims to create an online repository that is capable of storing any type of digital file and associating information to them, both automatically and manually, so that they are found more easily.

Contents

List of Figures

List of Tables

Acronyms

ADF Application Development Framework. 27, 33, 34, 55

AMP Alfresco Module Package. 27, 39

API Application Programming Interface. 10, 27, 34

BLOB Binary Large OBject. 38

CMIS Content Management Interoperability Services. 26, 27, 32, 34

CPU Central Processing Unit. 38

CSV Comma Separated values. 16, 44

DBMS Database Management System. 7, 8, 38

HTML Hypertext Markup Language. 34

IdP Identity Provider. 29

MARC MAchine-Readable Cataloging. 6

NISO National Information Standards Organization. 5

OASIS Organisation for the Advancement of Structured Information Standards. 26

RAM Random Access Memory. 38

RBAC	Role Based Access Control. 32, 42
REGEX	Regular Expressions. 42
REST	Representational State Transfer. 10, 27, 34
SDK	Software Developer Kit. 27, 39
UI	User Interface. 27, 34, 39, 41, 42
URL	Uniform Resource Locator. 34
WYSIWYG	What You See Is What You Get. 42

Chapter 1

Introduction

1.1 Motivation

The volume of data has increased in the latest years. The advances in computer technology and its accessibility changed the way people store data. This was emphasised as documents, books, photos and videos began to exist solely in the digital form. This transformation also occurred in scientific fields with medical records, research data, astronomical data, biological and genomic data, to name a few. Storing this information digitally provides easy access to large quantities of data. The development of software applications to handle such data also contributed to the aforementioned shift. The solutions empowered researchers to be more productive and led to more complex investigations.

Managing this amount of data can get out of control easily. As the number of records escalate, it becomes increasingly difficult for a person to recall what data exists and how to access information for a particular topic, making it an impossible task to find the intended records and to identify similarities between different data.

Digital files often have additional information besides its focus content. This information is not mandatory for computers to understand how to interpret the data however, it provides clues and descriptions that can be explored to help discovering such files.

The main reason behind this work is to offer researchers, in this particular case, research work in the heart failure area, a platform where they can store and share with each other, any type of data related to their investigations. Also taking into account the facilitation of discovery and correlation of several pieces of data, by using data content to create automatic search mechanisms.

1.2 Dissertation outline

This dissertation is organized in four additional chapters.

- **Chapter 2** is aimed to contextualize the topics of discussion in this work.
- **Chapter 3** specifies the system requirements, presents work done in repository software and compares the studied platforms against the objectives of the required solution.
- **Chapter 4** details the architecture of the chosen platform, instruct on how to use and install, and presents the work developed to meet the system requirements described in the previous chapter.
- **Chapter 5** contains the conclusions of the work produced in this dissertation and suggestions on future work to further enhance the platform.

Chapter 2

Context

In this chapter we provide context to concepts that we found relevant within the scope of this dissertation.

2.1 Data storage

For many centuries mankind had the necessity to save information and knowledge acquired so it could be used to help future generations achieve scientific, sociological and political advancements, as well as saving personal experiences to relive and share multiple times. In the past, information was registered mainly in paper records and books, and was stored, protected and served by traditional libraries. The advancements in computer technology have shifted information from physical medium to digital. This is due to an increase in hard disk sizes that allows the same amount of data to be stored with a much smaller footprint, for less cost and easier replicating capability when compared to non-digital files.

Nowadays, due to the widespread of computers and the amount of computational power available, data is produced at extraordinary rates and in high capacities, making finding ways to effectively store it more and more critical. Even with the advancements in storage hardware and compression algorithms, there is still the need to plan where data is to be physically stored. Although the ratio between capacity and cost of hard drives, which are the most common medium to store digital files, has been increasing, it is not financial feasible to enterprises with massive quantities of data to only use this technology as their storage solution. In these scenarios, it is common practice to utilize hard drives to store data that is meant to be accessed regularly due to their speed, and adopt slower mechanisms, namely tapes, to archive historical data (i.e. data that is rarely used or has been updated by modified versions).

Any system that stores data must encompass structures to organize information in a logical and explicit manner in order to better keep track of the data gathered. The folder tree architecture [Figure 2.1], used by computers, has been around for many years, representing files in an intuitive hierarchical model that users find familiar and comprehend.

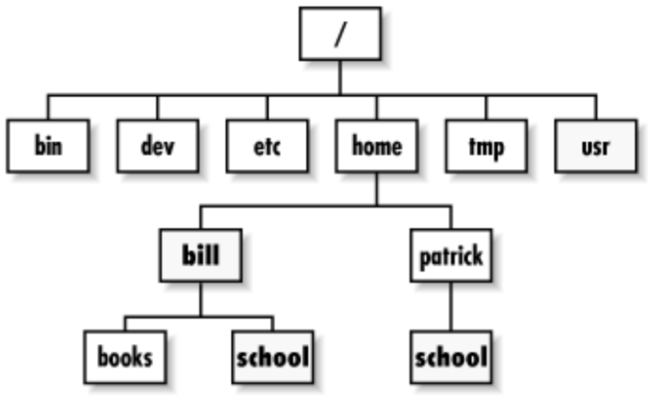


Figure 2.1: A hypothetical Linux directory tree [1]

2.2 Data and metadata

In the digital world, information can be stored in many different forms, depending on the type of content as well as the purpose and detail requirements of such content. In the same way that spoken language was developed to communicate and transmit information between people, computers need standard formats to create, consume and exchange data between them.

A file format specifies a structure for how information must be stored so that computers can read and understand its content. For the same type of information we can have different formats that are used to encode it, each with its strengths and weaknesses. For instance, we can have the same audio clip encoded in MP3 and FLAC formats; the first produces a smaller file by discarding some data, while the latter does not lose any data, producing larger files. Such differences in requirements also exist in other types of data, especially when transferring information over the internet.

Data can be stored in more or less structured formats. Formats such as CSV that represent data in a tabular manner focus primarily on the data itself, offering little information on what the data represents and are unable to portray data in a hierarchical manner. Complex data structures formats like XML and JSON are self-describing, meaning that they contain both the information and context for the data they store, in addition to being capable of storing hierarchical data.

When developing a digital data repository it is necessary to store information affiliated with such data in a systematic manner. Metadata is described as "a set of data that describes and gives information about other data". The purpose of metadata is to "facilitate search, evaluation, acquisition, and use of resources" [2].

Different authors segment metadata in different parts. Our definition derives from the concept provided by the National Information Standards Organization (NISO)[3] that separates metadata in three types: 1) Descriptive metadata, which covers information that identifies the resource, (i.e. name, description, creation date, author, etc.); 2) Structural metadata, consisting of information defining relations between different resources allowing the discovery of related resources; 3) Administrative metadata.

Administrative metadata is being comprised of several subtypes:

1. Technical metadata, containing information necessary to render and execute the resource i.e. information stored in actual file (bitrate in MP3 files, sensor data in image files);

2. Preservation metadata, which holds longstanding information to help use and main-

tain the resource in the future, such as instructions on how to use and checksums to assert that the resource has not been tampered with;

3. Rights metadata, which specifies information about the restrictions in regards to the use of the resource, as well as detailing the intellectual property holder.

2.2.1 Metadata Standards

Metatada in practice is a collection of elements that hold information about data [4]. As one of the principal objectives of metadata is to make the discoverability of resources easier, creating standard schemata that defines such elements becomes necessary, so they can be used systematically and promote interoperability between systems. As it as stated by Michele Hayslett[5]:

> "In order to be useful, metadata needs to be standardized. This includes agreeing on language, spelling, date format, etc. If everyone uses a different standard, it can be very difficult to compare data to other data."

In the digital world many disciplines coexist, each with specific requirements, making it nearly impossible to create a universal metadata schema that satisfies the needs of a particular domain[5], thus the need of multiple metadata standards. There are numerous metadata standards[1], some are generic, meaning that they describe information that spans across multiple disciplines, while others detail the specificities of unique subjects.

The need to standardize metadata is not recent. In the 1960s Henriette Davidson Avram created the MAchine-Readable Cataloging (MARC) standard with the intent to unify the way libraries would catalogue books and make it possible to share bibliographic information digitally between institutions. Work in areas like biology, image files, geospatial data also developed standards to store metadata.

In the current era, due to the quantity of diverse file formats in computers, metadata standards emerge such as Dublin Core that can be applied to all types of data. This is

[1]http://www.dcc.ac.uk/resources/metadata-standards/list

accomplished by defining elements that exist for every resource (i.e. title, description, date, etc.) with the drawback that file characteristic information is not stored.

2.2.2 Metadata Extraction

Most files in a digital system have metadata associated to them, either being information that describes each file or information needed to properly execute it. Every file format defines a data structure upon which information is stored, so it can be encoded and decoded by suitable software.

Metadata extraction can be viewed as the process of extracting knowledge from files[6], however it is not the scope of this dissertation. We regard metadata extraction as the action of processing files and reading information that is stored in them.

There are several standalone tools, such as Metadata Extraction Tool[2] and Exiftool[3] that are able to parse files and read their data. Parsing, in the context of computer files, is the process of iterating through a file, being able to understand its structure and gather the information that it contains. Besides standalone tools there are also tool-kits developed in various programming languages that detect and read metadata from files. These tools implement parsers for each supported format and expose interfaces with the content extracted.

2.3 Searching data

Having large quantities of information has little purpose if finding and retrieving relevant data in run-time is difficult and time consuming. The performance of today's devices changed the way people perceived computers to machines that can execute tasks instantly, and such mentality also manifested in terms of finding data. Database Management System (DBMS) can store large amounts of data and include language syntax to perform queries to search information, however, in their day to day life people store their information in

[2]http://meta-extractor.sourceforge.net/
[3]https://www.sno.phy.queensu.ca/ phil/exiftool/

digital files rather than in DBMS systems as they are more convenient to create, edit, share and consume. The process of searching for information in files has evolved from matching relevant documents using their name, to matching information in the files content, to more complex techniques that analyze both file's content and query, to increase the chances of success.

A search action is triggered by a need of information[7] that, in most cases, needs to be fulfilled quickly and accurately given very few keywords, thus having methods to match such information need to relevant documents is essential. Information retrieval is the field that is responsible for processing documents and storing information so it is possible to retrieve them by knowing some of its content. A typical information retrieval system works by processing collections of data, and organising it in an inverted index, so when a user queries the system for a specific document, there is no need to iterate over all data in the system.

An inverted index [Figure 2.2] is a data structure that associates a term/token with all documents that include that term. Creating an inverted index is a computational demanding operation that requires the system to process each document and retrieve every term to create a dictionary, while shifting the cost from the query stage where the number of requests is larger and the time to process such requests needs to be as low as possible to the user.

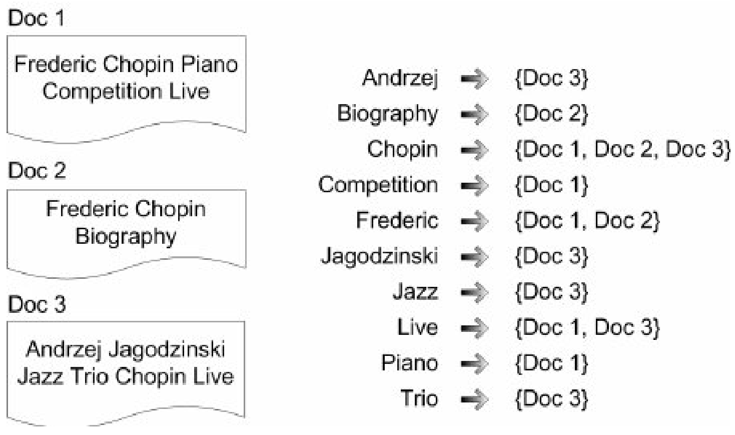

Doc 1
Frederic Chopin Piano Competition Live
Doc 2
Frederic Chopin Biography
Doc 3
Andrzej Jagodzinski Jazz Trio Chopin Live
Andrzej → {Doc 3}
Biography → {Doc 2}
Chopin → {Doc 1, Doc 2, Doc 3}
Competition → {Doc 1}
Frederic → {Doc 1, Doc 2}
Jagodzinski → {Doc 3}
Jazz → {Doc 3}
Live → {Doc 1, Doc 3}
Piano → {Doc 1}
Trio → {Doc 3}

Figure 2.2: Interface to associate key-value pairs to nodes.

The process of matching a user query to documents in an index can follow several models. "A model of information retrieval predicts and explains what a user will find relevant, given a user query"[8]. Some models such as **Boolean Retrieval** perform simple algorithm's to determine if a document is relevant (e.g. either the term is in the document or not); meanwhile other models, such as **Vector Space model** use more advanced techniques as applying weights to different terms and cosine correlation to identify relevant documents. The processes in information retrieval can be summarized in Figure 2.3

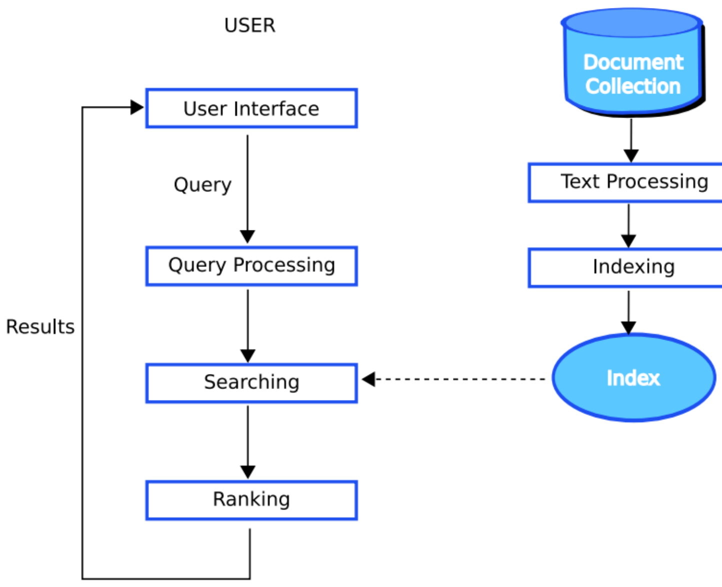


Figure 2.3: Information retrieval process [9]

Search engines are applications that implement information retrieval functionality and provide interfaces for creating indexes, and querying information. Modern search engines also add features like, stopwording, result ranking, stemming and query expansion.

Stopwording is a technique where, in the indexing stage, some words are ignored when processing the document. Pronouns and definite and indefinite articles are usually chosen to be ignored as most documents contain these words. Adding them to the index would degrade the result obtained as many unwanted documents would be evaluated as relevant, and would increase the size of the index.

Stemming is used to lower the number of similar terms in an index by reducing derivative words to their root[10]. One example of stemming is reducing the words fishing, fisher, fished to the stem fish. As stemming algorithms differ from language to language, it is important for the search engine to know which language is receiving. Reducing similar words to a common term enlarges the number of documents retrieved by a same query.

Query expansion is a manner of altering the query that the system receives to improve the results that are given, either in finding documents that otherwise would not be considered relevant or by altering a result rank. The most common and simple form of query expansion is to add synonyms to the base query.

Most search engines also have **ranking algorithms** in order to sort and present the most relevant results to the user.

There are several open-source search engines available on the market that provide libraries to index and search files so that application developers can focus on building their services. Many factors can play when deciding on a search engine to use in one's project. Besides the search capabilities previously mentioned that affect the quality of the results, other features such as the type of files they are prepared to *parse*, the ability to increment the index as new files are added and excerpt results (i.e. highlight the position where the query was matched) are important considerations.

The top 3 most popular search engines today are ElasticSearch, Apache Solr and Sphinx [11]. Both ElasticSearch and Solr are built on top of Apache Lucene, and extend its feature set by providing, among other features, result excerpt and exposing the system functionality over a REST API. Both Lucene and Sphinx have similar feature sets, being, arguably, the platforms built on top of Lucene easier to learn.

2.4 Cloud storage

The view of mobile devices being an essential good as well as effortless access to the internet, and the notion of the *always-connected* concept, affected in many ways, the relationships between people and computers, namely in the access to their data and files. In the past, digital files were stored in people's personal machines and could only be accessed directly through the machine or remotely via SSH shells or FTP servers, those being beyond reach for the common non-technical user.

As the advancements in storage hardware evolved, people began to be able to take their data anywhere with them. USB pen drives made possible to store and carry small amounts of data, with the later introduction of portable hard drives mitigating the issue. Nevertheless, it was still required that people add to manually update files stored in those devices, remember to carry those devices with them and ensure that they can connect to the desired system (e.g. smart-phones do not have USB Type-A ports that are still the most common physical port in portable hard-drives and USB pen drives).

The progression for smaller and lighter devices and adding to the points made previously, popularized the use of the cloud to store data in remotely. The high success of cloud storage solutions such as Dropbox[12], Google Drive[13], One Drive[4] accrues not only from the fact that users data is available anywhere but also the additional features provided. These solutions allow ease of sharing data, as users can grant access to folders and files to other users, and synchronize change in the stored files ensuring that the file is consistent across multiple devices. These services are scalable, meaning that users can increase or decrease storage capacity as they need and do not require an initial financial commitment. Furthermore, services that implement cloud storage have internal data replication and backup management capabilities ensuring that data is not lost while being transparent to the user.

Although cloud storage may bring many benefits, it also has some drawbacks. Since the data is stored in servers, users do not have full control of their files, requiring them to trust in the company that provides the service not to change, lost or share their files. Besides, in some cases there are legal repercussions that prohibit use of such solutions to store sensitive data. Nevertheless, such problems can be overcome by deploying private cloud storage solutions, which are maintained and configured by the organizations that want or need such control.

In short, the use of cloud storage is so inbred in today's society that storage capacity of

[4]https://onedrive.live.com/

devices that storage cap is not as important as previously had been. Google's Chromebook an example of a device that promotes access to remote storage as the primary storage mean.

Chapter 3

Related Work

In this chapter we studied several solutions in digital repository and content management domains, to determine whether their offers are valuable to our required solution. To parameterize our comparison points, we first specify system requirements, both function and non-function.

3.1 System Requirements

Our aim is to provide a storage solution that aggregates medical, patient and health research data. The capability of sharing and searching complex data is also essential. Although this work was inspired by the medical field, we want the final result to be adaptable to work with any type of data from varied areas.

This section specifies functional and non-functional requirements that the final system must comply. The specification of the functional requirements allows us to ensure that the expectations of the client are met and the non-functional requirements give directives on usability, performance and extensibility. In addition to the requirement specification, this section also lists some factors that would be preferable from the development scope.

3.1.1 Functional Requirements

- Users can access the system using a web browser.
- Users must be able to login and logout on the system.
- Users must be able to insert, modify and delete files.
- The system must be capable of storing any type of file format.
- Users must be able to search for documents in the repository.
- Users must be able to store their files privately, publicly or share with specific groups or users.
- The system must automatically extract metadata of supported files.
- Users must be able to edit and add metadata to documents.

3.1.2 Non-Functional Requirements

- The system must provide a REST API to enable programmable interaction.
- The system should be capable of extending support for future file formats.
- The system must be able to index stored documents.
- The system's user interface must be simple to use.
- The system must be developed using open-source software.

3.1.3 Development preferences

- Ideally, the system back-end should be codded in Java or Python
- If the system is based of an existing system, a mature system should be preferred.

3.2 System Evaluation

In the development of this work we first selected several repository solutions available on the open-source market to study. The candidates were discovered by searching the web using keywords as *data repository*, *repository metadata* and *data storage software*, as well as by exploring projects on GitHub.

Our analysis methodology consisted in evaluating system features, architecture, available documentation, as well extensibility. In addition, all systems were installed in a local machine to classify the installation process, and assess real user experience using the platforms.

3.2.1 Girder

Girder is an open-source web-based data management platform developed by Kitware as part of the Resonant ecosystem. Girder's aim is to provide a solid platform for building web applications that require: 1) data structuring and storage; 2) user management; 3) access management. All data in the platform is stored in collections. These collections represent the top level of the hierarchical structure and serve to associate data that shares a relation, such as belonging to the same project. Further structuring within the collection is obtainable by the addition of folders. The data is defined in items, which represent an abstraction of files, and have additional information - namely metadata - related to the specific file.

In regards to storing data, Girder offers the possibility to use several storing engines. System administrators can configure Girder to use the filesystem where the server is installed, but also have the possibility to store the files in remote Mongo database, Amazon S3 bucket. In addition, there is the possibility of using Hadoop Distributed Filesystems (HDFS) via the installation of additional plugins .

User management comprises all the user manipulation, from creating new users, authenticating them, to recovering passwords. Girder is capable of storing credentials, making accounts specific to the platform. However, it provides plugins that enable the use of third-party solutions such as OAuth and Lightweight Directory Access Protocol (LDAP), allowing institutions to use their own identity providers in a seamless way to manage users .

Girder allows both Discretionary Access Control (DAC) and Role-Based Access Control (RBAC) as ways to manage access to resources. These restrictions can be applied at collection or folder level, enabling finer regulation of access within a specific collection. Users are given access to a resource by getting permission from the resource owner or by belonging to a group that has access.

File searching is lacking in Girder. Although this feature exists it is very rudimentary, only being able to search resources through its name, neglecting the information present in the files content and metadata.

Architecturally, Girder is built with a strong decoupling of its logic and presentation, enabling system administrators to deploy its functionalities only via a RESTful API or by using the provided front-end client interface .

Girder offers a plugin framework enabling developers to extend its feature set. Plugins can add functionality on the back-end side, such as the authentication methods described previously in this section as well as adding features on the presentation side, for example rendering tables from CSV files.

The user interface (Figure 3.1) has an always-present menu on the side providing access to the main areas of interest. The majority of interactions occurs on the right side of the menu, changing its content based on the area selected. The actions available are hidden by a minimal design approach, which adds steepness to the learning curve but are consistent within the whole application, making its effects less predominant as users use the application over time.

Girder's community is vast and active. The latest stable release was published in November 2017, with developers currently adding new features for the next release. The code is available in the Girder's repository and includes both a Docker image and Vagrant file for easy deployment. Girder is written in Python making it possible to develop and deploy in Windows, Linux and many other operating systems.

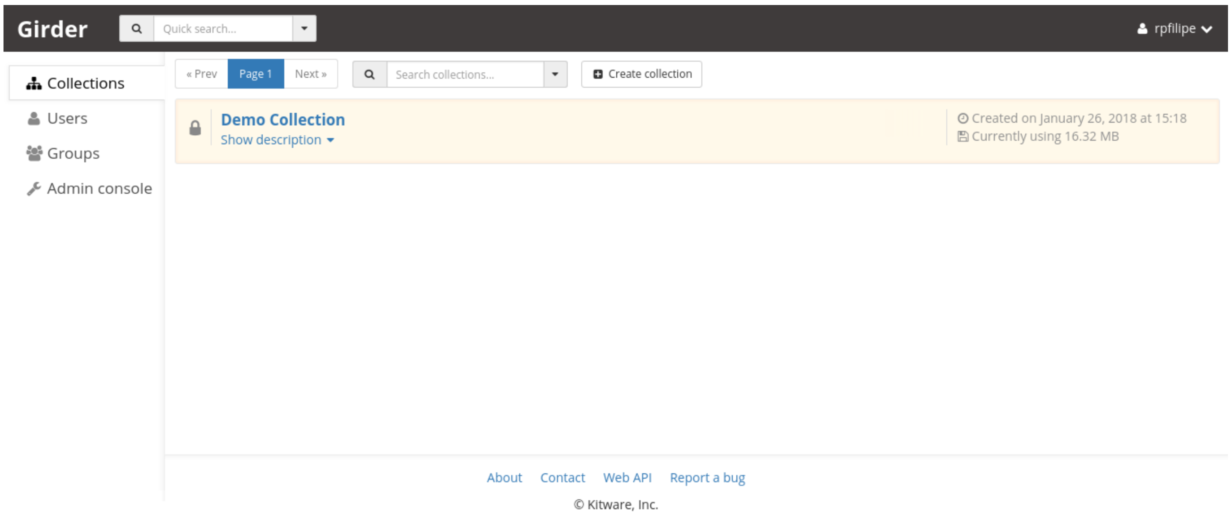


Figure 3.1: Girder's web client interface

3.2.2 CKAN

CKAN (or Comprehensive Knowledge Archive Network) is an open source data management system created by the Open Knowledge International, a non-profit organization whose objective is to promote open access to data by providing infrastructures to easily share and store content. CKAN is wildly used as a backbone to power open data repositories, being adopted by both institutions and governments .

As CKAN was designed to be a data portal for accessing data, its architecture is based on organizations. Organizations are the responsible entities for creating and managing datasets. Each organization can have several datasets, for example, a sexual transmitted diseases organization can have datasets referring to many different diseases or years. In addition to storing data, datasets have forms that can hold additional information such as license, version of the data, and description among other properties.

CKAN lacks a RBAC approach. Although promoting open access to data, enables organizations to create private datasets its contents are limited to only users within the organization. The notion of groups is present in CKAN but not functioning as a RBAC. Groups are a means to aggregate datasets that share a common factor and may be from different organizations.

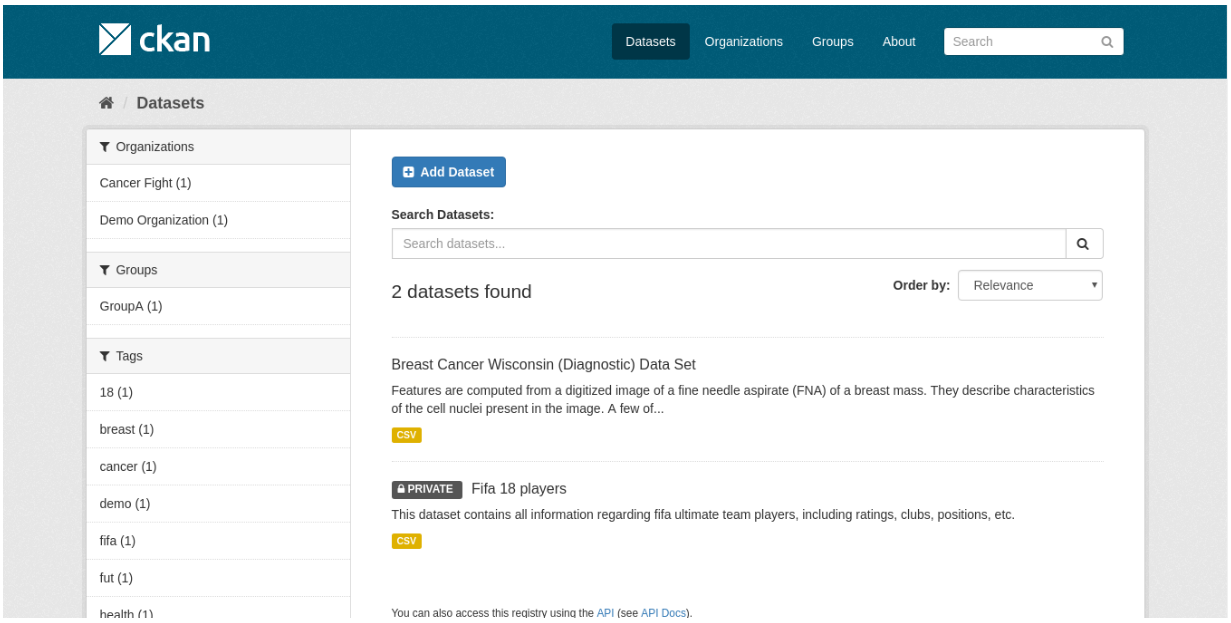
CKAN manages its users internally, i.e. does not have third-party authentication methods. User's primary function is to create, manage and access private datasets from an organization, considering public datasets can be accessed without logging in the platform.

A dataset in CKAN can contain any number of files of any type. It comes with solid visualization capabilities for dataset related formats (such as CSV and XML Spreadsheets (XLS)) offering tabular, graph presentation of data. It is possible to extend presentation formats by enabling and/or creating plugins.

As CKAN is designed to be a data portal for accessing data from multiple organizations, searching is a key feature. It uses Apache Solr as its index and searching engine, allowing the discovery of datasets by its metadata defined by the user upon its creation. Such metadata is obtained in a form containing properties like description, author, filetypes, tags, among others. Any metadata contained in the files is not extracted, however, dataset managers can add key-value pairs that can be used in the search. It is possible to customize the core metadata associated with a dataset, but requires coding of new forms instead of an approach tailored to the consumer.

In regards to development and extendibility CKAN provides solid documentation, detailing the system's API endpoints and giving practical examples in developing new functionalities. Developers can access most of CKAN's functionality via the RESTful API, however system administrator tasks such as managing plugins and configuration options, are only available by accessing the server where the instance is installed; having to manually edit the configuration files and use the command line interface (CLI), adding an additional layer of security. Extensions add additional features to CKAN's core functionality. Some extensions are designed and supported by the CKAN developer team, with many more created by community members and institutions that use CKAN to power their data repositories.

CKAN's interface (Figure 3.2) is very tailored to the discovery of data. The top menu holds access to the datasets, organizations, groups and the searching functionality. The left-side panel shows information related with the datasets found, such as the organization they belong, tags, data formats, etc. This proves helpful as it gives users feedback that can be used to tune or restrict the results. The main portion of the interface holds primary content of the sections, both search results and user, dataset and organization details.



2 datasets found

Breast Cancer Wisconsin (Diagnostic) Data Set

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. A few of...

Fifa 18 players

This dataset contains all information regarding fifa ultimate team players, including ratings, clubs, positions, etc.

You can also access this registry using the API (see API Docs).

Figure 3.2: CKAN's web client interface

CKAN is still in development, having the latest stable release in April 2017. The development team encourages community contributions by creating small tasks aimed to developers with low understanding of the system's architecture to start getting familiarity with the platform. The community management team hosts monthly private meetings with its members as well as public discussions via meet-ups and forums. CKAN has a defined road-map giving stakeholders and developers insight to the development path of the platform. The software is available in CKAN's github repository and has an online demo available.

3.2.3 DSpace

Dspace is an open source digital repository developed by the Massachusetts Institute of Technology (MIT) in partnership with Hewlett-Packard (HP) Labs back in 2002 under the

BSD license. As Dspace was growing MIT and HP Labs created the DSpace Foundation to manage and provide support to the project. Currently DSpace is managed by DuraSpace, a non-profit organization emerged from the growing synergy between the Dspace Foundation and Fedora Commons organization.

DSpace is an institutional repository, meaning that each institution will run and manage its own DSpace instance independently. The information model that the system uses is based in communities that can be understood as parts of the institution, for example, departments of a university. Each community is self-managed. Communities can create sub-communities, define policies, and create collections. Collections group data that has a common relation, for example, a department can have a collection for PhD thesis and another for Masters. DSpace defines items as the units of data and must belong to a specific collection.

DSpace defines items using the Dublin Core metadata standard, providing an uniform information representation among heterogeneous data. Such metadata records are indexed making them discoverable by the search engine. DSpace uses Apache Solr as its search engine. DSpace allows system administrators do define additional metadata schemata or alter existent ones making them available for describing items.

This enables collection managers to define item templates depending on the needs of a particular collection, increasing the items detail. Any changes to metadata schemata have to be manually performed, as DSpace is not able to import schemata via XML Schema Definition (XSD) files. Changing item templates does not affect the form displayed when creating a new item nor making the new values available when performing a search. This makes it necessary to access the server where the software is deployed, edit the form configuration files and re-build the search engine. DSpace can also perform full text indexing on files, making its content available to the search engine. The software ships with several filters to process the most common file types (PDF, HTML, Word documents) and allows the creation of custom parsers for handling other file formats. DSpace is capable of filtering search results from title, author, subject, date and file existence.

DSpace offers a range of authentication methods. The application can manage users internally or use third-party providers that support Shibboleth Authentication or LDAP. It is also possible to authenticate in the software through X.509 Certificates and explicit IP addresses. The software also calculates and stores file checksums on upload enabling users to validate its integrity.

The platform implements a RBAC model to control access to its resources. It is possible to control access at every level of the hierarchy, meaning that communities, sub-communities, collection, items and files have individual policies.

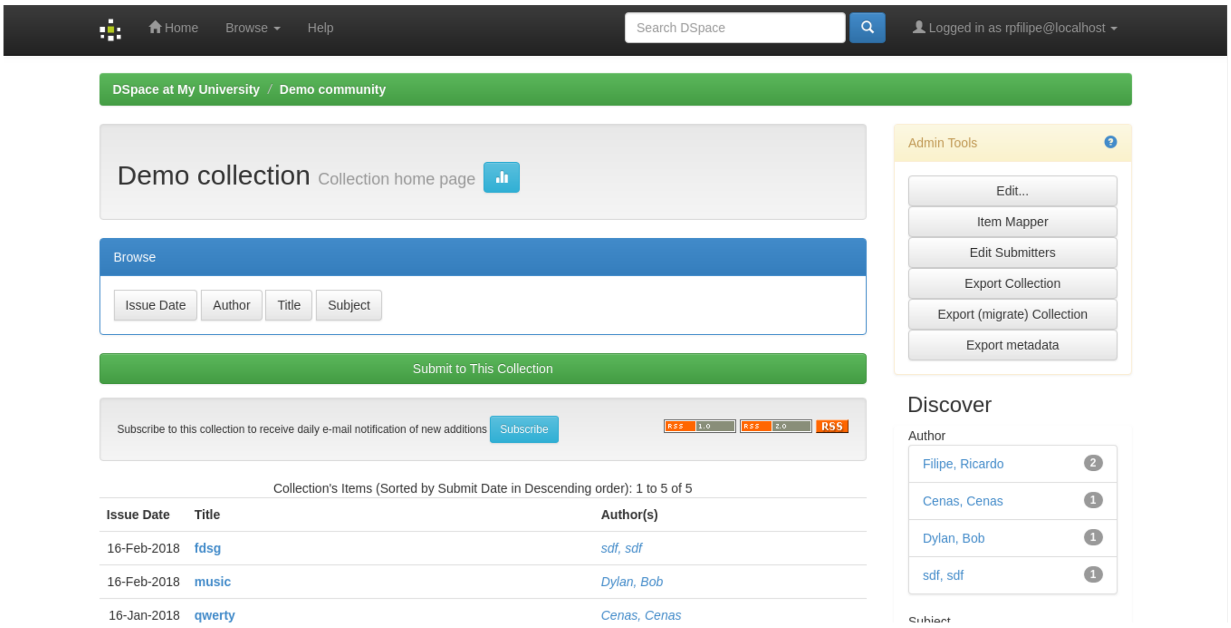
DSpace also has document management features. Collection administrators can define submission workflows to control revised submissions before making them openly available to the users. This means that the platform can accept submissions automatically or, upon receiving a new submission, delegate tasks to selected users to examine the work.

The software ships with two user interfaces, one using Java Server Pages (JSP) technology (Figure 3.3) and another using the Apache Cocoon framework (Figure 3.4). The JPS interface has a more modern look using a top bar to hold user navigation and search query entry and a main panel that displays the content of the active context area and where users can apply filters to the retrieved items. There is also an additional panel on the right side that is shown to logged in community and collection managers. The interface is built with a responsive design suitable for mobile devices.

The Apache Cocoon interface organizes its content in a dual panel mode. The left panel holds the active context, with all the navigation, search and management functionality located on the right panel. DSpace 7.0, scheduled to be released in late 2018 or early 2019, will have a new RESTful interface built with Angular 2.

Adding files to DSpace collections can be burdensome. The platform presents users with an extensive form requesting details about the collection, defined in the Dublin Core Metadata Standard, that has to be manually entered since adding the files is done at the end of the interaction. Although not all fields are required, the more information the user enters, the more accessible by the search engine and complete the record is.

Dspace is written in Java requiring a Java Servlet to deploy (it is advised to use Tomcat) and has been tested under Windows, Linux and Mac OSX operating systems. To store



DSpace at My University / Demo community

Demo collection Collection home page

Browse

Subscribe to this collection to receive daily e-mail notification of new additions

Collection's Items (Sorted by Submit Date in Descending order): 1 to 5 of 5

Issue Date	Title	Author(s)
16-Feb-2018	fdsg	sdf, sdf
16-Feb-2018	music	Dylan, Bob
16-Jan-2018	qwerty	Cenas, Cenas

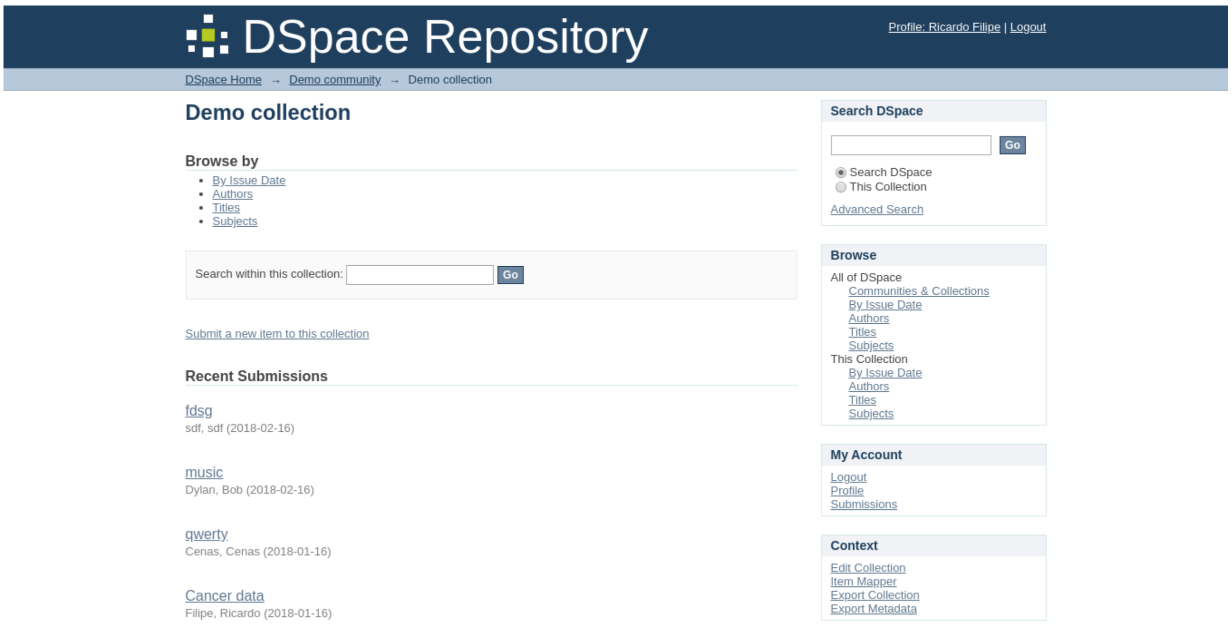
Admin Tools

Discover

Author

Filipe, Ricardo	2
Cenas, Cenas	1
Dylan, Bob	1
sdf, sdf	1

Figure 3.3: DSpace's JavaServer Pages interface


DSpace Repository

Profile: Ricardo Filipe | Logout

DSpace Home → Demo community → Demo collection

Demo collection

Browse by
- By Issue Date
- Authors
- Titles
- Subjects

Search within this collection:

Submit a new item to this collection

Recent Submissions

fdsg
sdf, sdf (2018-02-16)

music
Dylan, Bob (2018-02-16)

qwerty
Cenas, Cenas (2018-01-16)

Cancer data
Filipe, Ricardo (2018-01-16)

Search DSpace

[✓ Search DSpace]
This Collection
Advanced Search

Browse
All of DSpace
- Communities & Collections
- By Issue Date
- Authors
- Titles
- Subjects

This Collection
- By Issue Date
- Authors
- Titles
- Subjects

My Account
Logout
Profile
Submissions

Context
Edit Collection
Item Mapper
Export Collection
Export Metadata

Figure 3.4: DSpace's Apache Cocoon interface

application data such as users, groups, metadata, etc, DSpace uses a relational database, being compatible with PostgreSQL and Oracle. The files can be stored in the server's filesystem or in cloud-based solutions.

The system is compliant with Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) and Simple Web-service Offering Repository Deposit (SWORD v1/v2), Open Access Infrastructure for Research in Europe (OpenAIRE) protocols, in order to provide open interfaces for accessing, consuming and exporting data. DSpace is also capable of creating and ingesting Archival Information Packages (AIPs) [14] which are packages describing an archival object in DSpace i.e. site, communities, collections and items, their relations and that also contain the bitstream (raw bytes of the actual files) present in the items. This presents an advantage over traditional backups as it allows system administrators to restore the entire platform of individual communities and collections of items.

DSpace has a network of registered service providers who work with DuraSpace that offer resources such as extension, hosting, analysis and commercial support to institutions. The system has a list of extensions that add functionality to DSpace however, some have dead links, some are paid, and there is no standard development and installation. DSpace is well documented with step-by-step instructions on installing, upgrading and other tasks that system administrators might find useful, although documentation referring to creating addons/extensions is short. The DSpace project has both Slack and Internet Relay Chat (IRC) channels and a number of mailing lists that segregate information depending on user's interests.

3.2.4 Islandora

Islandora is an open-source digital repository developed with the Flexible Extensible Digital Object Repository Architecture (Fedora) Commons framework by the University of Prince Edward Island. The software is a ready to use solution for institutions that intent to use Fedora Commons framework, providing an easy deployment for institutions that want a Fedora repository.

Islandora bases its structure in objects with associated datastreams. Objects cannot contain other objects (like folders in an UNIX file systems) instead, a graph base approach is used to establish relationships between objects through RDF. The software defines three main types of objects. Content Model objects are templates that aim to represent specific content (which are used to create Solution Packs); Collection objects group several other objects and automatically establish relations between them; Data objects represent the files added to the repository along with metadata. Data objects can also have related files that are handled the same by the system.

Islandora utilizes the descriptive metadata models supported by Fedora. The primary formats are Dublin Core and MODS, though the system supports any XML-based metadata schema which can be defined in content models to tailor objects representation depending on the type of content. This XML-base approach enables the system to use the Extensible Stylesheet Language (XSL) files to crosswalk -map an element from a schema to another schema - information and indexing of metadata.

The software uses Apache Solr and Fedora Generic Search Service (GSearch) to index and discover objects. GSearch, is used to synchronize the Solr index as updates it as files are added and removed from the system.

The system uses ExifTool to automatically extract technical metadata from files. As ExifTool is open-source has the advantage that community members add support to new file types, but if the system requires a specific file parser, developers have to extend ExifTool functionality by building the software from source.

Islandora promotes system interoperability by providing OAI-PHM and PREMIS support via utility models that extend Islandora's core functionality, adding ingest, managing and viewing options. Utility models range from calculating file checksums to perform optical character recognition (OCR). These models (as well as Solution Packs) are not close packages, meaning that they require support software to be installed in the server machine, though Islandora's documentation gives detailed information to install modules and their dependencies.

Islandora exposes the underlining Fedora REST API for accessing and managing objects. The API allows searching of objects, ingesting, deleting and modifying objects.

Islandora's interface (Figure 3.5) benefits from the customization of Drupal's framework. A navigation bar is placed on the top of the window, with the active context placed below. The display of collection is similar to the folder structure present in most operating systems making its navigation intuitive to new users, but suffers from various user experience faults. The system does not support ingesting file by drag and drop, and ingesting multiple files at once requires them to be packed in a ZIP file previously. When adding files the user is prompt with a form requiring the file's description metadata, making it bothersome for the process.

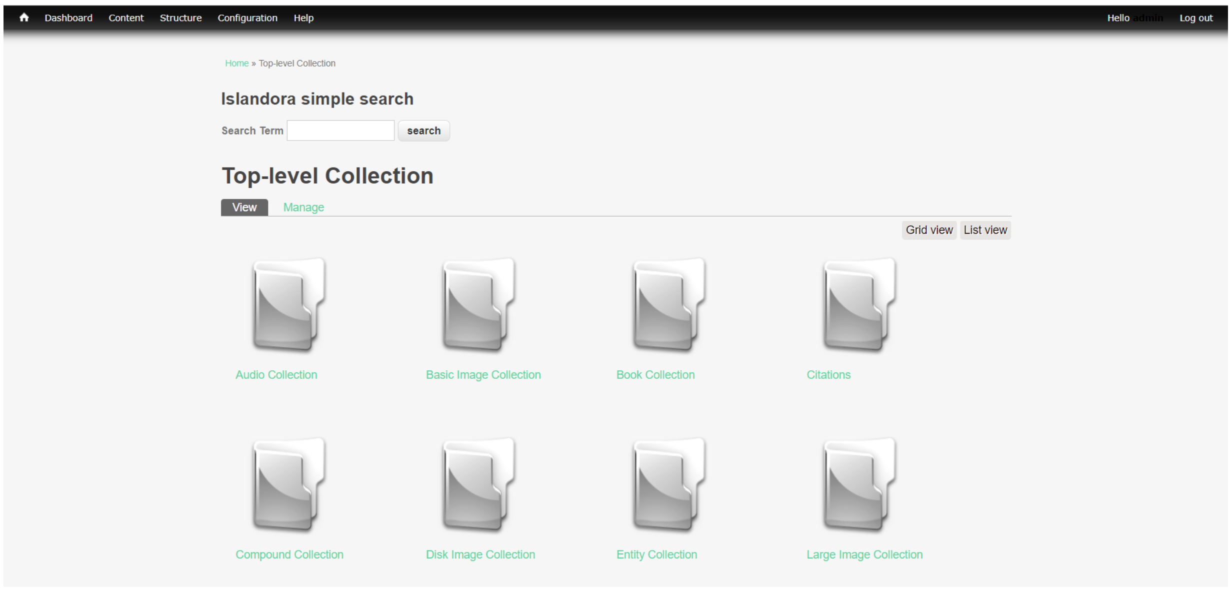


Figure 3.5: Islandora interface

Islandora's latest release was in November 2017. There are several companies that work with Islandora and support the project.

3.2.5 Alfresco

Alfresco is an enterprise content manager developed by Alfresco Software with the purpose of sharing content between users. Alfresco's solution to document management is used by large companies such as Cisco and NASA to enable them to store and find files in the day to day business processes of the companies.

Alfresco stores all information from a node in content models. A content model specifies a type of a document in the system as well as its properties, as groups named aspects. Aspects can be to view properties in a node's page details and to configure forms for adding and editing information of nodes. The designs of properties are configurable specifying which type of data they store, if multiple values can be held and more complex restrictions as enforcing REGEX expressions.

Alfresco's content models follow the Content Management Interoperability Services (CMIS) standard. CMIS is an OASIS open standard that specifies how repositories should structure their data in order to allow sharing of data between different systems in addition to defining a query language for obtaining information in the system. Models can be defined using XML files or by accessing the administrator tab in the web portal. Alfresco comes with several modeled metadata from generic as Dublin Core, to more domain specific as EXIF. Alfresco architecture enforces that a node must be of a specific type, but allows nodes to have properties of another type, granting, for example, audio files to have properties such as language from the Dublin Core schema.

Alfresco is capable of extracting metadata from supported file formats upon upload and its modular architecture allows for extension, future proofing the platform. The metadata extracted is stored in content model properties that best match the information, being automatically indexed by the system as well as the content in text based files.

The search feature of Alfresco is quite powerful. The system uses Apache Solr as its search engine, but also a SQL-like search syntax that supports boolean expressions, proximity search, search for values in specific properties, among others as well as CMIS-SQL that accompanies the CMIS standard used in content models. In adding to the syntax used in the search boxes, there is an advanced search area containing configurable forms to facilitate more complex searches without the need to understand the underlining syntax. All results from a search can be filtered by file type, size, creation date and user, among other fields.

The Alfresco platform is extendable by the use of Alfresco Module Package (AMP) files that can modify configuration of the base system, as well as added functionality, such as custom actions that use external software, and changes in the UI. Developers can customize Alfresco by directly altering the code that is running, but it is advised to develop AMP packages using the provided SDK in order to maintain compatibility with future updates. Alfresco has an extension repository[1] that offers many addons to the platform, both developed by the Alfresco team itself and available for free and by the community. Some organizations also offer paid extensions. The system allows programmatic access by providing several API; the SDK provides a Java and CMIS API and it is also possible to access Alfresco via a REST API (called webscripts) with HTTP requests.

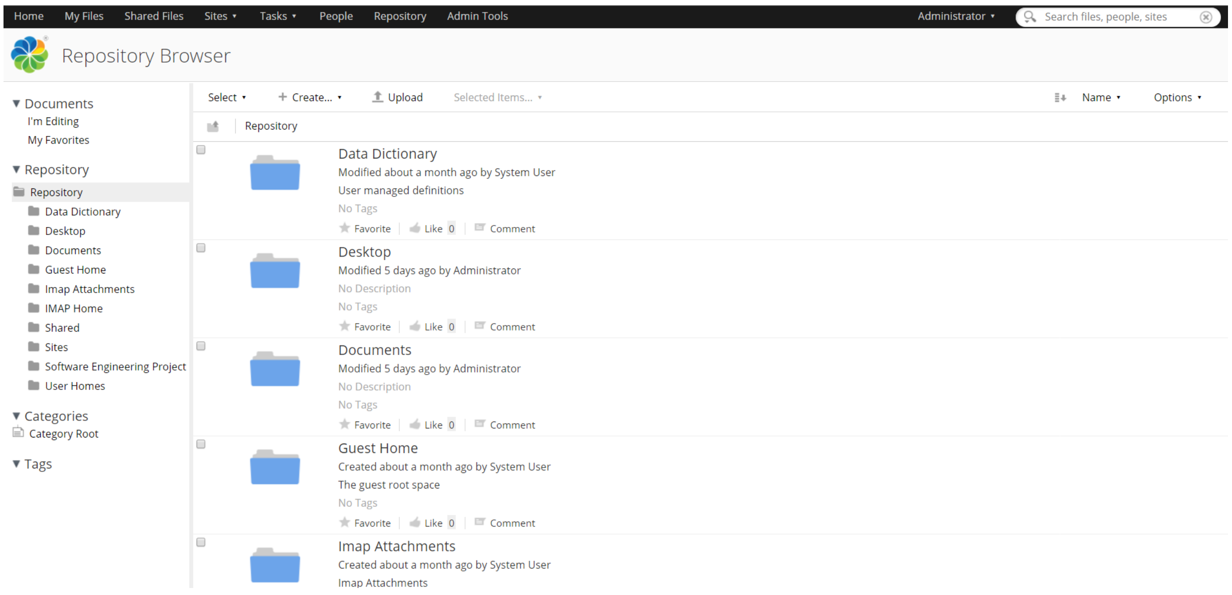
Alfresco's architecture separates the backend from the frontend. The backend, also referred as platform, is responsible for storing data and managing users and access, and all functionality like search and metadata extraction. Alfresco ships with a web UI called Share (Figure 3.6) granting a turnkey solution for rapid deployment, that delivers access to all functionality of the platform. Storing and retrieving files is accessible through the repository tab in the top header of the page. The repository UI is similar to other consumer focused cloud-based storage solutions such as Google Drive and Dropbox, giving users familiarity supporting both drag-and-drop and browsing upload features. Clicking on a document opens a page relative to that document. Each document page contains metadata associated with such file, a commenting section, versioning option, as well as actions applicable to the file (e.g. delete, move). In addition, supported files such as PDF, Word documents, and audio can be viewed directly in the browser, without needing to download it.

The Share UI is develop using the YUI library[2] and Aikau[3] a custom framework based on the Dojo[4] toolkit, requiring developers to be familiarized with both frameworks.

Alfresco also provides a framework called Application Development Framework (ADF)

[1]https://community.alfresco.com/community/alfresco-addons
[2]https://yuilibrary.com/
[3]https://docs.alfresco.com/5.0/concepts/aikau-intro.html
[4]https://dojotoolkit.org/

that allows the creation of a web portal from scratch using Angular[5] components as well as mobile application available for both Android and IOS devices.



Data Dictionary
Modified about a month ago by System User
User managed definitions
No Tags
Favorite | Like 0 | Comment

Desktop
Modified 5 days ago by Administrator
No Description
No Tags
Favorite | Like 0 | Comment

Documents
Modified 5 days ago by Administrator
No Description
No Tags
Favorite | Like 0 | Comment

Guest Home
Created about a month ago by System User
The guest root space
No Tags
Favorite | Like 0 | Comment

Imap Attachments
Created about a month ago by System User
Imap Attachments

Figure 3.6: Alfresco Share user interface

Alfresco has both enterprise and community editions. The community edition has no personalized support from Alfresco, needing organizations to maintain, customize and upgrade the platform themselves. Featurewise both versions have the same functional features, being the community edition used for testing new features before being available in the enterprise edition. The versions differ in more performance related features such as scalability and clustering and by supporting proprietary software such as Oracle databases and integration with services like Microsoft Office.

Alfresco is currently in version 5.2 with the team already working on a version 6. It is possible to deploy the platform from the source code available or by using the installer that already comes with the application server needed to run the Java WAR's as well as a Postgres database to store all data. The available documentation offers knowledge in customizing the platform, both in system configurations and developing extensions. The community surrounding Alfresco is large, promoting annual developer conferences and

[5]https://angular.io/

having a forum with active users from Alfresco, other organizations and independent as well.

3.3 System Comparison

In this section we evaluate the studied systems and draw a comparison to highlight both the negatives and positives of each solution. We used Table 3.1 to help the process.

For our scenarios Islandora is not a great solution as the process of uploading files is too bothersome for use in the routines of users and does not give considerable value compared to other solutions. We still feel that it was important to mention Islandora, as FEDORA Commons underlining architecture is largely referenced in the digital repository scene.

Girder is an interesting platform that has attractive features, namely Identity Provider (IdP) support and cloud storage capabilities, but the limited search capabilities and the lacking use of metadata standards proves to be inadequate to our needs.

Both CKAN and Dspace were good contestants however, differences in user experience relative to the Alfresco solution, the lack of native metadata extraction from files in both solutions and CKAN requiring a plugin to perform full-text indexing were reasonable decisive factors.

After careful examination, we determined that Alfresco Share was the most appropriate to our requirements.

Table 3.1: Main evaluating factors matrix

	GIRDER	CKAN	DSPACE	ISLANDORA	ALFRESCO
Web client	yes	yes	yes	yes	yes
Database system	Mongo	Postgres	Postgres	MySQL	Postgres/MySQL
Code language	python	python (flask)	java (tomcat)	php (drupal)	java(tomat)
REST API	yes	yes	yes	yes(fedora)	yes
Access Control	DAC / RBAC	RBAC	RBAC	RBAC	DAC / RBAC
Idp Support	OAuth/ LDAP	no	no	no	no
Pluggin support	yes	yes	yes	yes	yes
Bulk ingesting	yes	yes	yes	yes (hard)	yes
Metadata extraction	yes (plugin)	yes (plugin)	no	no	yes
Custom parsers	yes	-	-	-	yes
Full-text indexing	no	yes (plugin)	yes	yes	yes
Ease of ingesting	3/5	4/5	4/5	1/5	5/5
OAI-PMH	no	no	yes	yes	no
METS	no	no	yes	yes	no
Dublin Core	no	yes	yes	yes	yes
Cloud Storage	AWS S3 / Hadoop	no	AWS S3	no	AWS S3
Search Engine	-	Apache Solr	Apache Solr	Apache Solr & GSearch	Apache Solr
Documentation quality	good	good	good	average	good

Chapter 4

Netdiamond Platform

After the evaluation of the platforms in the previous chapter 3 and careful consideration of system requirements, we opted to build the Netdiamond platform using the Alfresco Content Manager. In this chapter we further detail Alfresco's architecture, how to use, configure and extend the platform as well as specify additional functionalities that we implemented. The work developed focuses on increasing use cases that the system is capable of accomplishing, rather than customizing the visual look of the final solution.

Alfresco has a lot more features that are not specified in our system requirements however, our main motivations for this approach were:

- Alfresco's architecture was designed to support extending the supported file formats.
- Alfresco's architecture was designed to allow users to create custom models to store metadata.
- Alfresco uses a search engine to index information.
- Alfresco Software is a mature company and their software is widely used, giving the opportunity to discuss, improve and take advantages of improvements made by the community.
- Alfresco Content Manager exposes its functionality over a REST API and Webscript framework.

- Alfresco promotes interoperability with other systems by utilizing the CMIS standard to organize the repository as well as providing metadata standard models such as Dublin Core.
- Alfresco uses a RBAC access model.
- Alfresco ships with an out-of-the-box web portal.

4.1 Architecture

The Alfresco Content Manager solution is divided in three different logical layers [Figure 4.1] each with different operation domains.

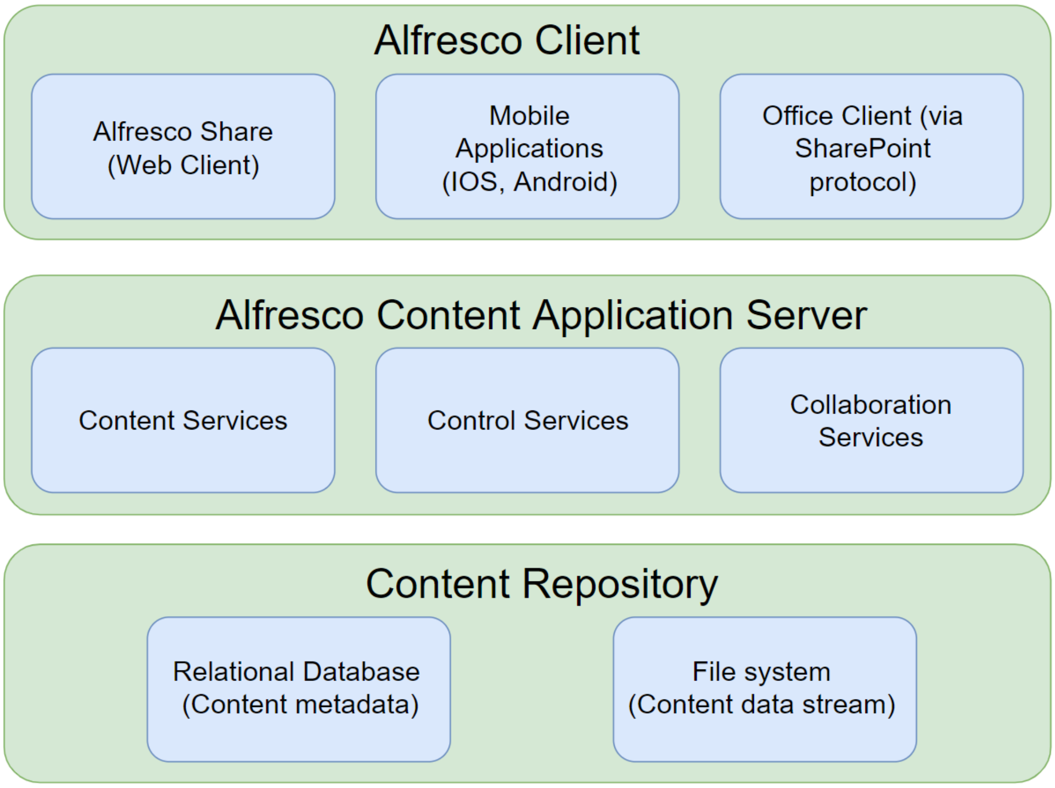

Alfresco Client
Alfresco Share (Web Client)
Mobile Applications (IOS, Android)
Office Client (via SharePoint protocol)
Alfresco Content Application Server
Content Services
Control Services
Collaboration Services
Content Repository
Relational Database (Content metadata)
File system (Content data stream)

Figure 4.1: Alfresco architecture layers [15]

The **Alfresco Client layer** is the one that manages the interactions between users and the application functionalities. This encompasses the Share web portal, the available mobile apps, integration with other clients such as Microsoft Office and Google Docs as well as the customized clients using the Alfresco Application Development Framework.

The **Alfresco Content Application Server layer** performs the additional business logic over a simple data repository. This includes functionality as the metadata extraction of added files, transforming files from one format to another, defining actions to be executed when certain conditions happen, among other functionalities. The collaboration features of Alfresco (sites, wikis, commenting section) are also covered by this layer.

Finally, at the core, the **Content Repository layer** employs the fundamental functionality of the repository. This covers the ability to create, modify or remove files, index and search of files, enforce limitations and permissions of acceding and modifying data, as well as services to define and associate content models and perform audits, registering events in the platform.

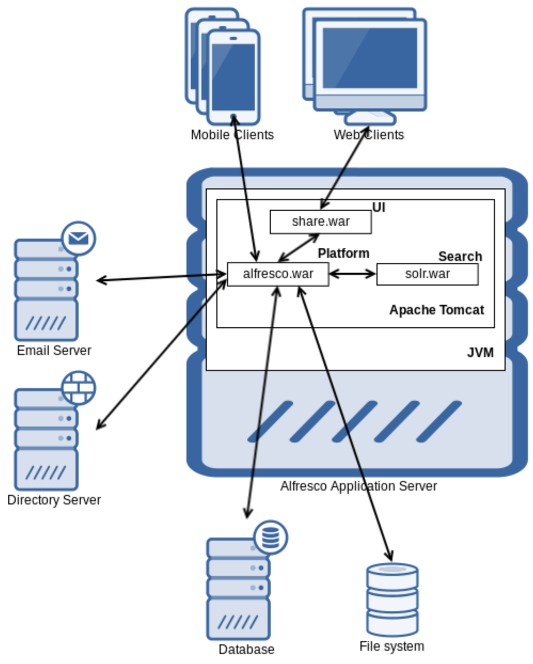


Figure 4.2: Alfresco component architecture [16]

The figure 4.2 illustrates the components that implement the above logical structure. The *share.war* is responsible for publishing the web client interface which the client and server side logic. The other possible user interfaces (i.e. mobile apps, clients developed with the Alfresco Application Development Framework or any other type of custom client) are external components that are not supplied with the Alfresco Content Manager server, though they communicate directly with the *alfresco.war* server using the REST API.

The *alfresco.war* server contains all the business logic, specified in the **Content Application Server** and **Content Repository** layer, with the exception of the services of indexing and search that are implemented in the Apache Solr server (*solr.war*) and are accessed through the REST interface.

The Platform application is built using the Spring Framework, and provides several API for accessing the core functionality implemented in Java and Javascript. The REST API is utilized to interact with the system using any chosen programming language. For interaction with other content managers a CMIS API is also provided, which ignores Alfresco's other features such as commenting system, rating, users, to name a few.

The Share app is also built using Spring and the UI operates using various technologies and frameworks. The interface is built using mainly Surf[1], a framework developed by Alfresco for creating web pages using reusable HTML templates, written in FreeMarker[2], that generates the final page in the server side rather than the browser. The interactive web application nature of Alfresco Share is developed using the discontinued YUI library and Aikau framework. The development of new UI elements, either being pages or menus, in the web interface is to be accomplished using solely the Aikau framework, and current pages developed using Surf are being migrated; however, at this stage this causes a lot of entropy when developing and customizing the front-end client.

All the web application resources are served using a single application server (Tomcat) each with a different mapping URL.

[1]http://docs.alfresco.com/community/references/APISurfPlatform-intro.html
[2]https://freemarker.apache.org/

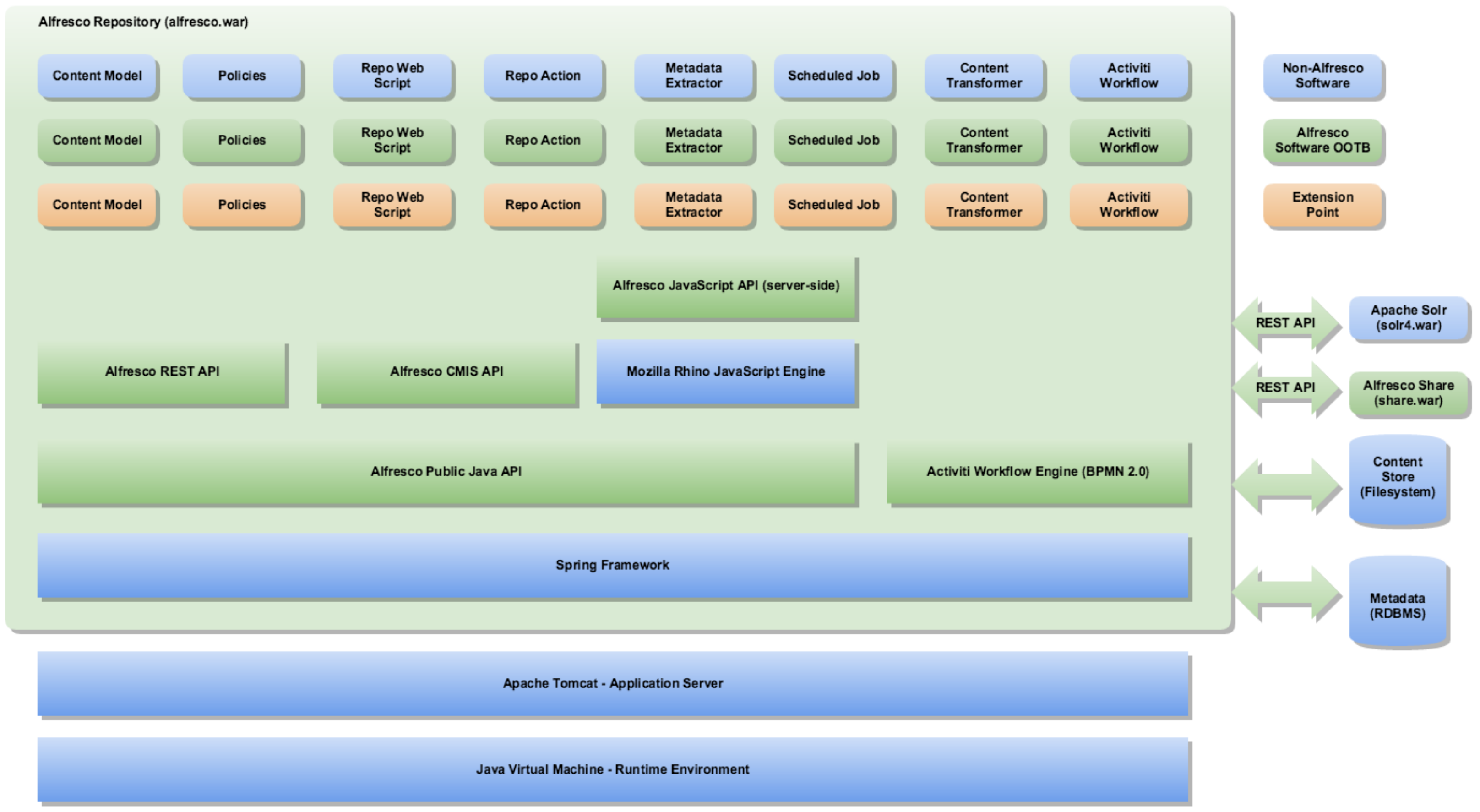


Figure 4.3: Alfresco Platform (*alfresco.war*) architecture[17]

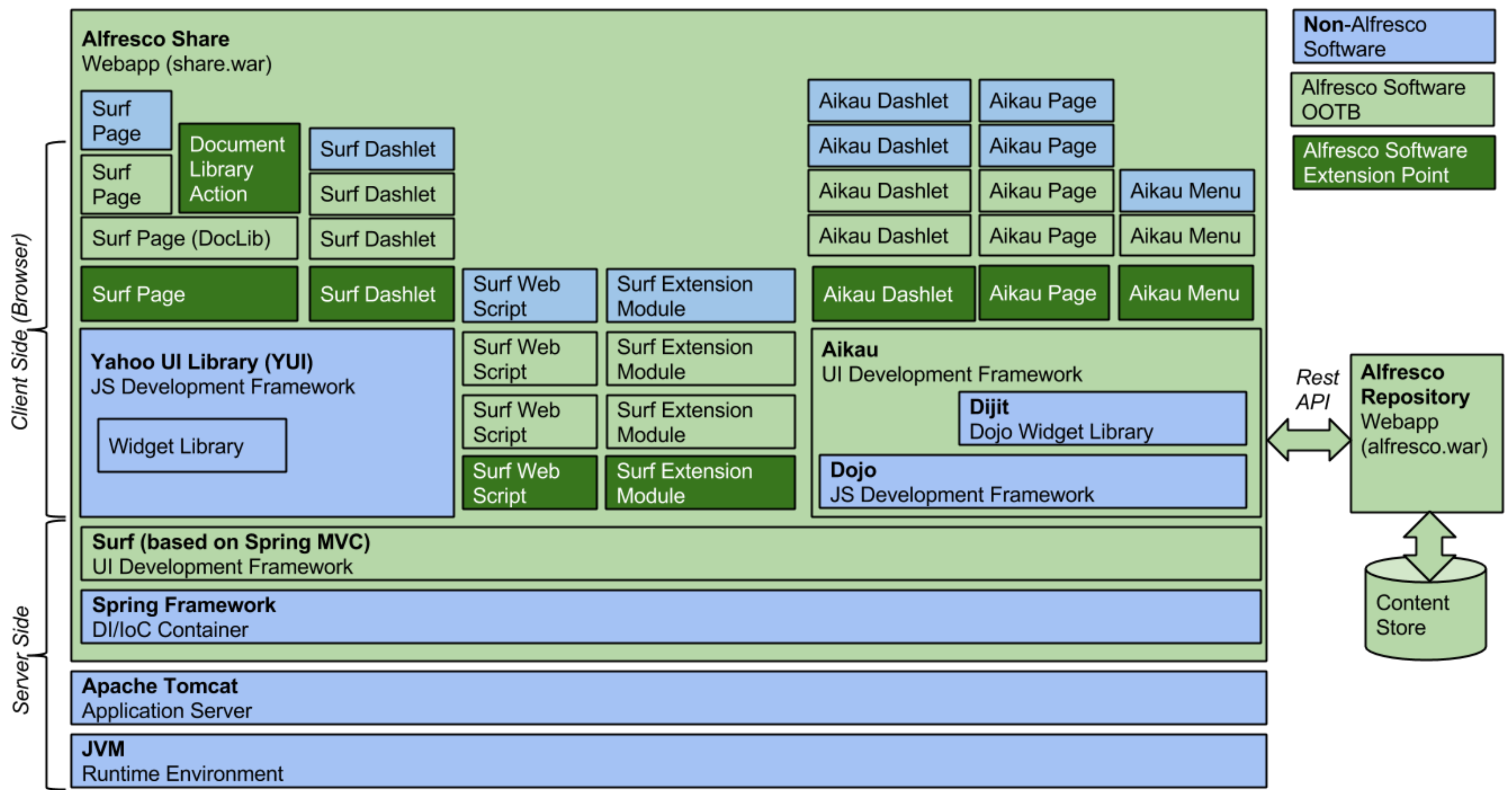


Figure 4.4: Alfresco Share (*share.war*) architecture[18]

4.2 Data architecture

The way Alfresco stores files is logically identical to modern operating systems filesystems; a node tree based structure [Figure4.5] is used with a single root that can ramify as new nodes are added. There are two types of nodes: *folder nodes*, which can have other child nodes and *file nodes*. File nodes do not actually contain the files content; they, however, point to the physical file while the node itself can store other information. This is what allows Alfresco to associate metadata with files, as nodes store the properties defined in the various content models as well as the access control permissions of such file.

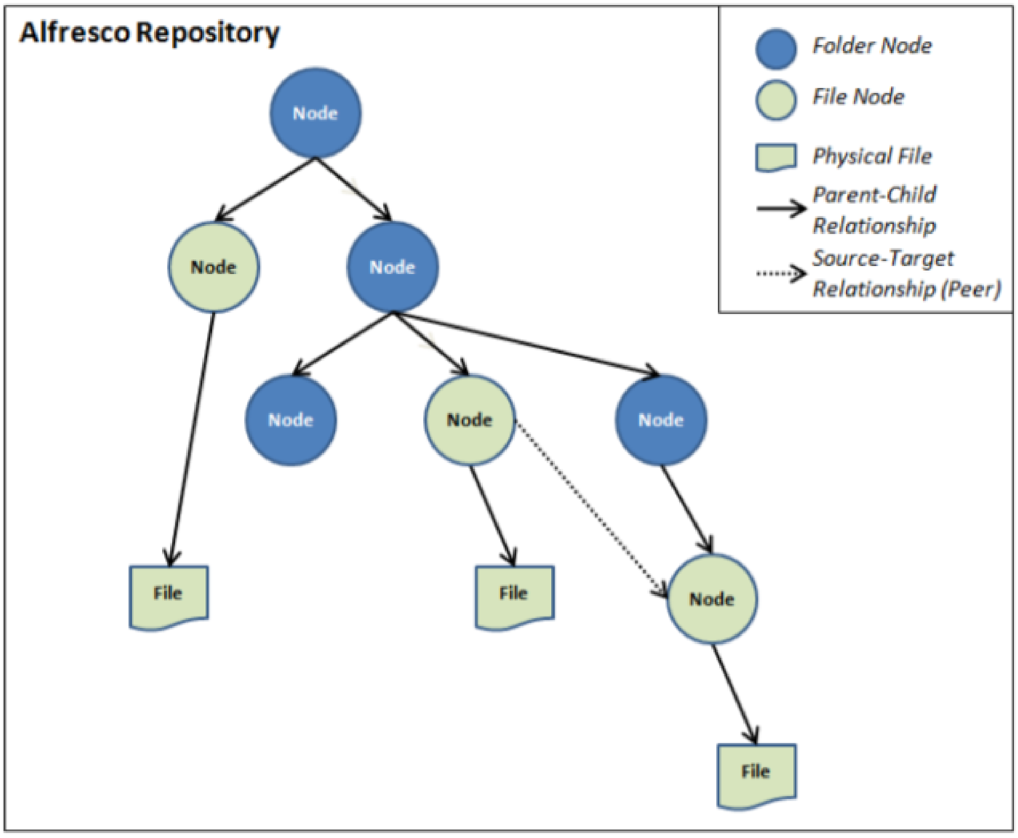


Figure 4.5: Alfresco repository storage logical structure [19]

Alfresco's data architecture stores information in two different places. Files uploaded to the system as well as the indexes created by Solr are stored in the filesystem, given that these files can be very large and this solution provides faster access and removes possible compatibility problems between different DBMS support of Binary Large OBject (BLOB). The remaining information (i.e file's metadata, users, permissions, rules, etc.) are stored in the configured DBMS.

4.3 Deployment

As Alfresco Share is accessible through a web browser there's no need to install any software on users machines, only being necessary to install the Alfresco on a server machine. We chose to deploy Alfresco Community Edition version 5.2 as the 6.0 version was not released before the beginning of development of our solution.

The software specifies both hardware and software requirements. Alfresco requires a machine with at least 4GB of Random Access Memory (RAM), a Central Processing Unit (CPU) clock speed of 2.0 Ghz, which can be 32-bit architecture, although 64-bit architectures are highly recommended, and disk space varies on the amount of data that the solution must cope with. To help system administrators determine hardware requirements to Alfresco solution they want to deploy, there is a scalability blueprint[20] testing the system under different hardware profiles and server configurations. The system is validated to work on a variety of operation systems, DBMS and Java Servlet Containers[21]. In addition, the system requires supplementary software, namely ImageMagick[3] and LibreOffice[4] to manage images in the web portal previewing window and convert text documents from one format to another respectively.

Alfresco installation is done thought a binary file available on the Alfresco site[5] that contains the application level requirements needed to run the software: the Alfresco platform and Share portal, the Apache Solr instance for searching, a PostgreSQL server and a Tomcat Servlet for running the code. The installation binary is comprised of an interactive process so administrators can define installation folders, service ports for both server and database, users, among other initial configurations, that works both in GUI interfaces and

[3]https://www.imagemagick.org/script/index.php
[4]https://www.libreoffice.org/
[5]https://www.alfresco.com/thank-you/thank-you-downloading-alfresco-community-edition

in terminal interfaces for operating systems that do not incorporate desktop environments.

We installed Alfresco via the binary provided in a Ubuntu Server 16.4 virtual machine with java version 8 installed, and utilizing the default DBMS, Java Servlet and port configurations. No extensions were added to the application besides ones the developed in section 4.6 and the Google Docs Integration that ship naively.

4.4 Developing and installing addons

As we mentioned in the previous chapter, Alfresco provides a SDK to add functionality (that can also be used to save and share customizations) to the application. The SDK is available through Apache Maven[6], a build and dependency management tool for development of software in Java. For creating extensions, there are several project templates (Maven archetypes) which define the structure of development and packaging configuration that, ultimately, is going to generate the AMP packages to be installed in the deployed application. Given that Alfresco's architecture is segregated in business logic (Alfresco Platform) and presentation (Share UI), it is possible to develop features that affect only the platform or presentation, and most commonly both. For these scenarios, there are project templates available.

AMP packages can be installed in two different ways. The simplest way is to copy the *.amp* files *"{$alfresco.homeDir}/amps"* and *"{$alfresco.homeDir}/share_amps"*, in case of platform extensions or UI extension respectively, by executing the **apply_amps.sh** script in the *"{$alfresco.homeDir}/bin"* folder. This script will install the extensions present in the folders and update any previously installed if a new version is present. The second method consists in manually executing the Alfresco's () specifying the desired operation, extension and the target (*alfresco.war* or *share.war*).

[6]https://maven.apache.org/

```
$ java -jar bin/alfresco-mmt.jar install amps/extension.amp
    tomcat/webapps/alfresco.war
```

4.5 Usability work-flows

This section illustrates common interactions between users and the Share web client. Alfresco provides many features that, describing them all here would be impractical, hence we focus on the most critical work-flows that we specified in the system requirements.

4.5.1 Uploading, viewing, editing and deleting files

Storing information on Alfresco though the Share client is quite intuitive. The repository page supports uploading of files through the *drag-and-drop* feature, as well as the upload button that opens the system file explorer [Figure 4.6]. The upload of folders uploads all its files maintaining the same hierarchical structure. In addition, it is also possible to create new files with the application itself, from scratch or based on a predefined template that can have several files and a hierarchical structure.

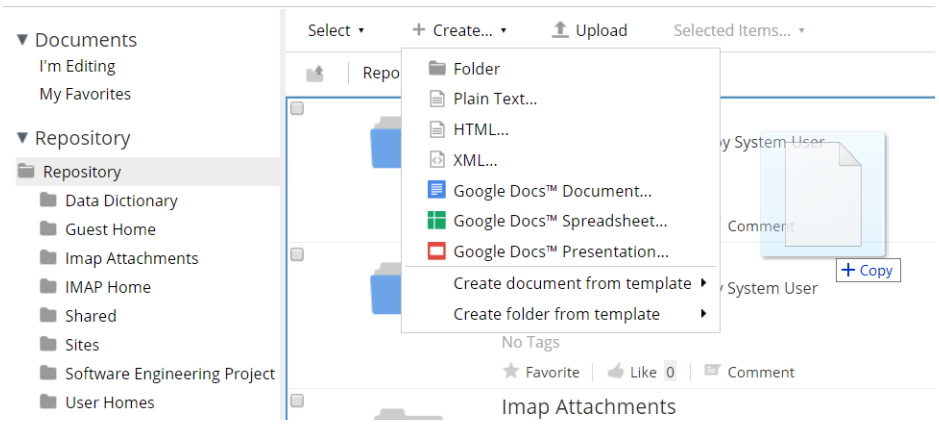


Figure 4.6: Adding files in Alfresco Share web interface .

Highlighting a folder or file shows the actions available for that node, as shown on Figure 4.7. The *Manage aspects* action allows users to add metadata properties to the node from content models that have been previously configured, besides the ones added by default or automatically added by the extraction of metadata upon the file upon upload. The *Edit Properties* action, as the name implies, provides forms for altering the values of the properties of such node.

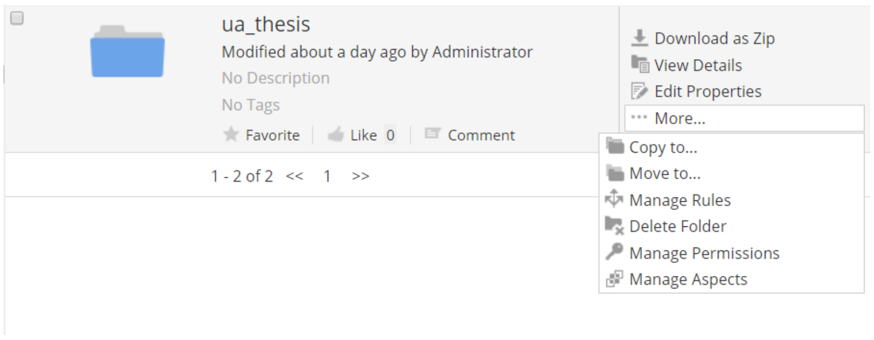


Figure 4.7: Node actions in Alfresco Share web interface .

4.5.2 Creating and editing content models

The creation and edition of content models is accessible to the in the administrator page, available for users belonging to *ALFRESCO_ADMINISTRATORS* and *ALFRESCO_MODEL_ADMINISTRATORS* groups or by describing using a file. The definition of properties to hold metadata requires them to be associated with a content model and aspect (i.e. group of properties).

Alfresco gives users a great control on creating properties to store metadata and associating them to files and folders, as properties require several characterization fields:

- **Name (required)** - Defines the system name of the property.
- **Display Label** - Specifies the shown name of the property in the UI.
- **Description** - Provides contextualization to the property.
- **Data type (required)** - Establishes the type of data that the property must hold. The types available are: *text*, *multiline text*, *integer*, *long*, *float*, *double*, *date*, *datetime* and *boolean*.

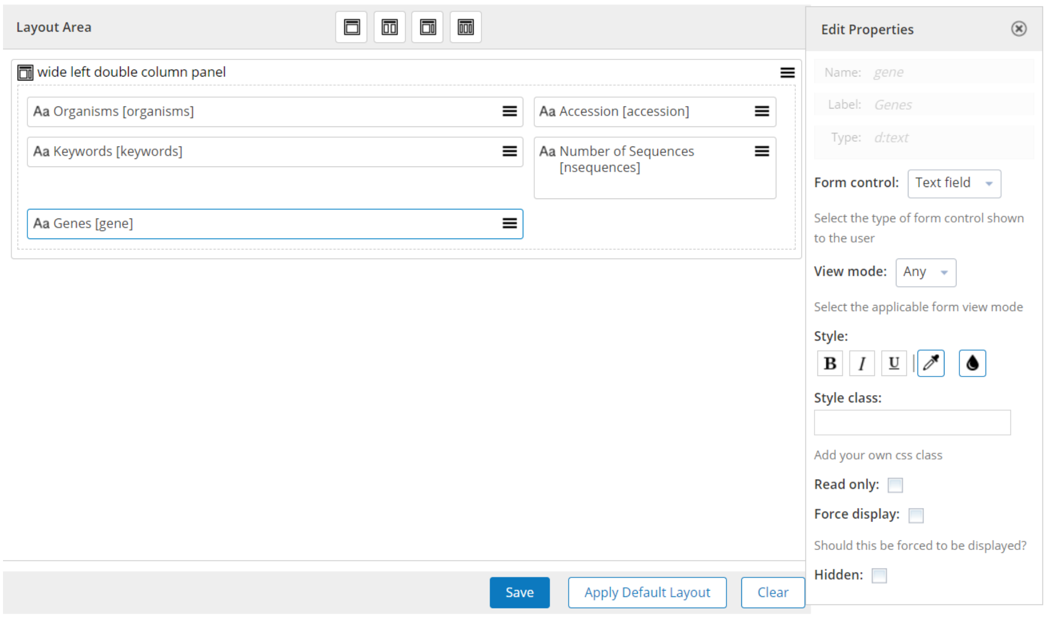
- **Requirement (required)** - Specifies if the property must contain values.
- **Default value** - Specifies the default value of the property.
- **Constraint (required)** - Stipulates the values that the property must comply. By default no constrains are applied, although the system supports minimum and maximum values and lengths, Regular Expressions (REGEX), selection from a list of pre-set values and custom Java classes for enforcing a restriction.
- **Indexing** - Specifies if the property is to be indexed and if so, how to index it.

For presenting properties to the users, both for viewing and editing purposes, there is a UI layout designer tool [Figure 4.8] that arranges how aspects should be presented graphically in the Share client, in a What You See Is What You Get (WYSIWYG) manner. The designer offers a set of frames (e.g. two vertical columns, three vertical columns, etc.) where users can *drag-and-drop* the properties they wish to display on the form. For each property in the form it is possible to select how a user is going to edit the property as there are various input types available, from simple input textfield, checkboxes, drop-down lists, calendar selection, to name a few.

4.5.3 Managing users, groups and permissions

The management access to the application, either the users or groups that built the RBAC system is accessible in the Share client located in the left side control [Figure 4.9], in the area only accessible for users with administrator capabilities.

Alfresco comes with several necessary groups to ensure the correct functionally of the system which cannot be removed. Despite this aspect, administrators can create any number of groups for restricting access. An auto-complete search box filters the shown groups, which is helpful when an installation has many groups. Selecting a group lists the users that constitute such group, and allows the addition of new users through another search box or by copying current users from other groups.


Layout Area

wide left double column panel

Aa Organisms [organisms]

Aa Accession [accession]

Aa Keywords [keywords]

Aa Number of Sequences [nsequences]

Aa Genes [gene]

Edit Properties

Name: gene

Label: Genes

Type: d:text

Form control: Text field

Select the type of form control shown to the user

View mode: Any

Select the applicable form view mode

Style:

Style class:

Add your own css class

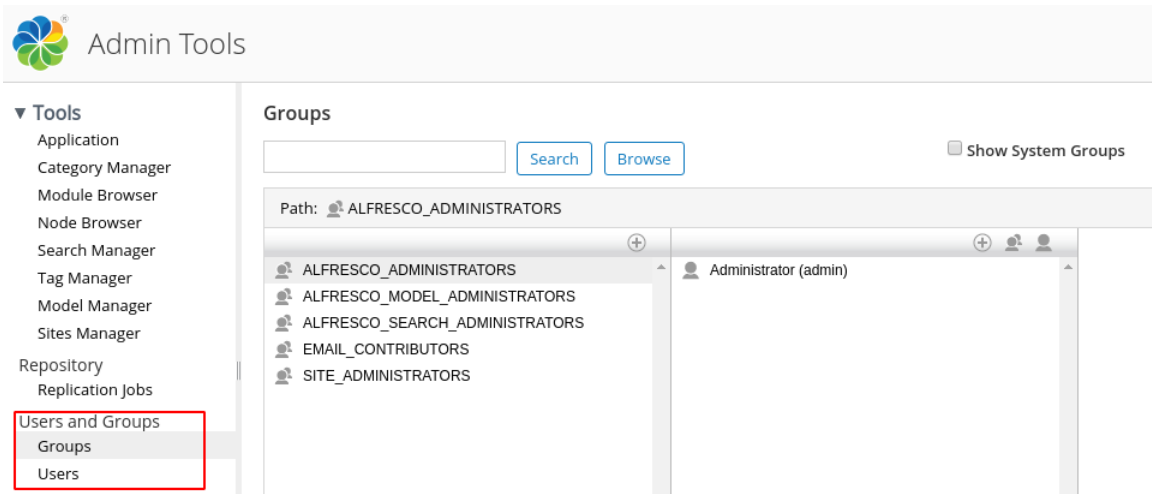
Read only:

Force display:

Should this be forced to be displayed?

Hidden:

Figure 4.8: Alfresco Share forms layout designer.


Admin Tools

Tools
- Application
- Category Manager
- Module Browser
- Node Browser
- Search Manager
- Tag Manager
- Model Manager
- Sites Manager

Repository
- Replication Jobs

Users and Groups
- Groups
- Users

Groups

Show System Groups

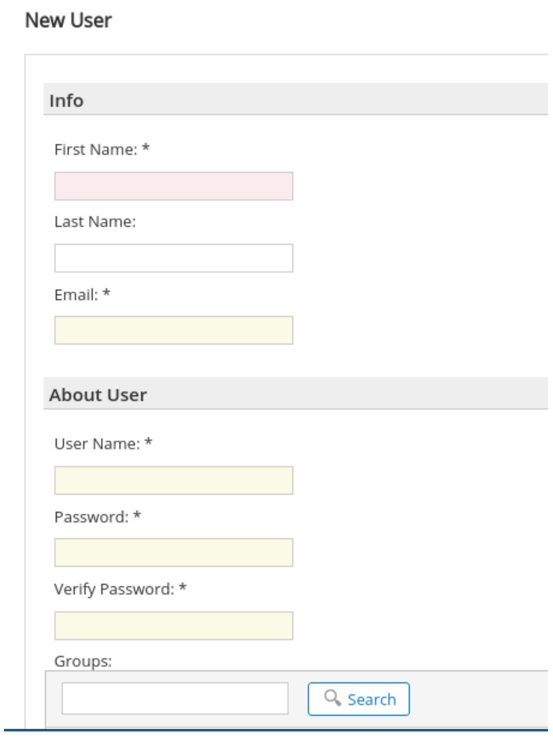
Path: ALFRESCO_ADMINISTRATORS

- ALFRESCO_ADMINISTRATORS
- ALFRESCO_MODEL_ADMINISTRATORS
- ALFRESCO_SEARCH_ADMINISTRATORS
- EMAIL_CONTRIBUTORS
- SITE_ADMINISTRATORS

Administrator (admin)

Figure 4.9: Alfresco Share group management interface.

For creating new users [Figure 4.10], the first name, email, username, and password are mandatory fields, while the remaining user information is managed by users themselves. The possibility to add groups to the user is accessible in the creation stage, through an auto-complete enable search box with the available groups. As the setting up of the platform can require the addition of a large amount of users in a short time, Alfresco provides the


New User

Info

First Name: *

Last Name:

Email: *

About User

User Name: *

Password: *

Verify Password: *

Groups:

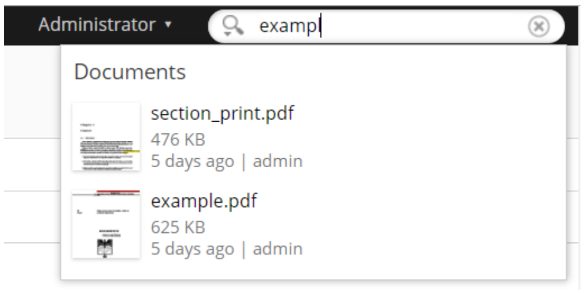
Figure 4.10: Alfresco Share user management interface.

ability to ingest users through CSV files although, this feature does not support specifying groups for the users. It is also possible to define maximum data quota available for users.

4.5.4 Searching files

The search functionality is always present across all different pages in the web interface, through the text-box located in the right side of the navigation bar, which encompasses a live search feature[Figure 4.11] that shows the top results for that search query. Committing the search opens a new page containing all the files matched with the added functionality of being able to filter results by *creator*, *file type*, *creation date*, *size*, *modifier*, *modification date*, *tags* and *categories*.

For cases where a finer control is intended, in which users want to specify query values to explicit properties (e.g. finding files that have the term 'example' in their name, and not getting files which have the term in their content or in other properties), the advanced



section_print.pdf
476 KB
5 days ago | admin

example.pdf
625 KB
5 days ago | admin

Figure 4.11: Live search result for a query.

search page[Figure 4.12], provides forms where users fill in the respective queries for each property. The advanced search is also accessible via the button in the search text-box. Text-boxes that are left empty behave as neutral, allowing results to have any value on those respective properties.

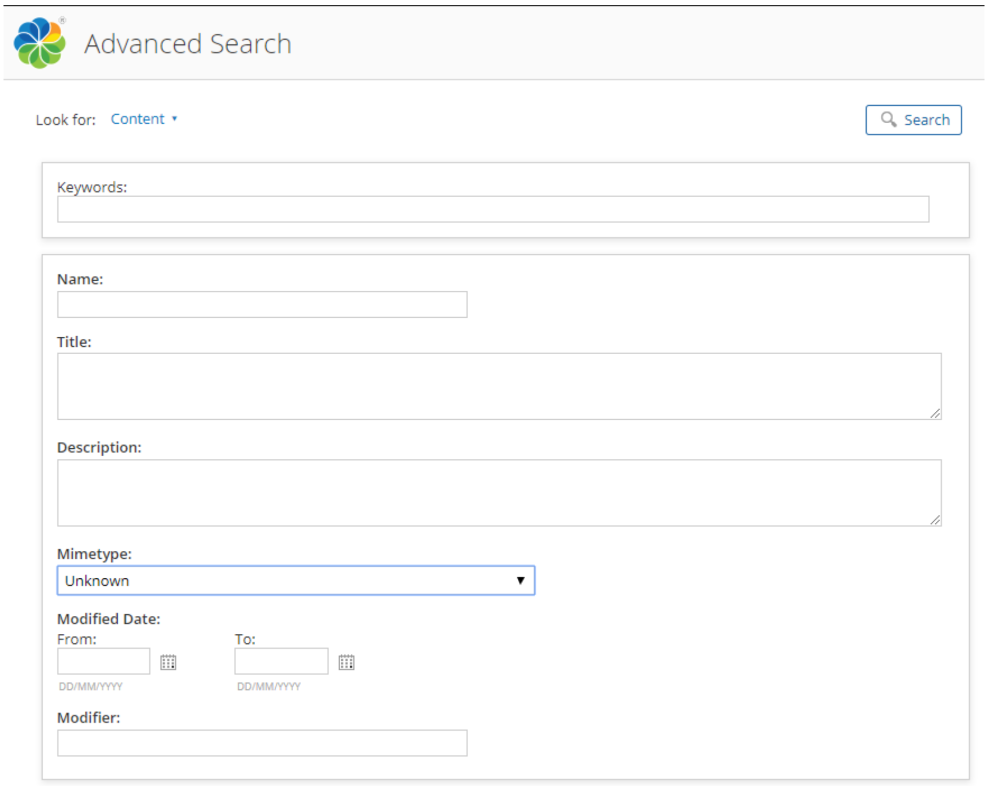
It is possible to configure custom search forms with properties of custom content models or any combination of properties that system administrators find useful. In subsection 4.6.3 we demonstrate how such forms can be configured.

4.6 Customizations

4.6.1 Support for Fasta and Genbank files

The Alfresco base system that we deployed naively supports metadata extraction for common digital files such as Word documents, images, audio, PDF files to name a few. To tailor our solution for the Netdiamond project we implemented metadata extraction from genome files, namely FASTA[22] and GenBank[23] formats. The system can get awareness of new formats via configuration files identifying the format and enumerating its file extensions. The metadata extraction is done by Java classes that extend the base extractor and give support for the formats we added to the system.

An examination of both formats was done to define which information was important to retrieve from the files, and a definition of a content model that maps such information in properties was created. The properties identified were the following:


Advanced Search

Look for: Content ▾ | Search

Keywords:	
Name:	
Title:	
Description:	
Mimetype:	Unknown ▾
Modified Date: From: DD/MM/YYYY	To: DD/MM/YYYY
Modifier:	

Figure 4.12: Alfresco Share forms layout designer.

- accession
- description
- keywords
- gene
- organisms
- number of sequences
- sequences

For the parsing of the files we utilized BioJava[24], open-source java framework for bioinformatics processing that packs genome processing and, most importantly to our scenario, parsing of Genbank and FASTA formats. The metadata extracted from uploaded files was mapped to the properties that we defined previously.

FASTA and Genbank files can have very large genome sequences, making the process of tokenization and indexing unreliable. To overcome this problem we proposed a solution where system administrators specify genome sequences in a configuration file that are meant to be indexed by the system.

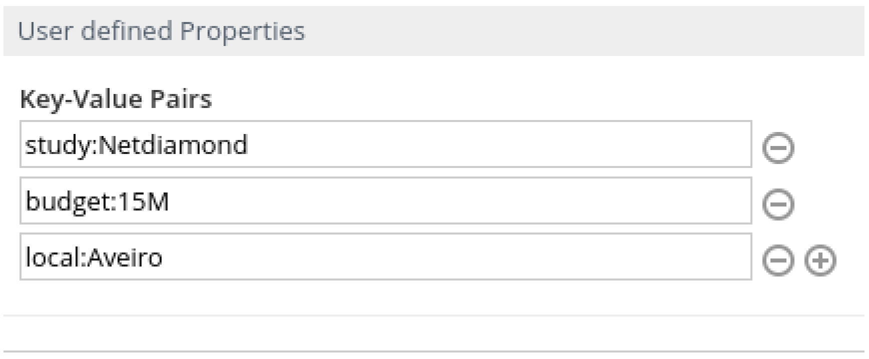
4.6.2 Support for *ad hoc* metadata

Alfresco was designed to enforce the definition of models to store metadata related to the content to be present in the system. This assures coherence across same format files and metadata that is applicable across files, for example the name of the file, the creator, extension, etc., but is inflexible for associating exception information for specific or sporadic purposes. The native support for tags mitigates this issue, but presents itself in the other extreme of the spectrum, as the user adds information that cannot be associated with a document property.

The Netdiamond platform added the possibility for users to add key-value pairs that hold information to a file, without the need to create or alter any existing content model. We try not to deviate from the existing architecture when implementing this feature; firstly, to maintain compatibility with the release of future versions of Alfresco and secondly, to take advantage of the already deployed structure to index and search content.

Our solution involved creating a content model to be used to store the key-value pairs and applying it to all content types in the system. This way, the information shares the same tokenization, stop-words and synonym functionality as the rest of properties. As we defined the property as being *multiple*, there are no limits for the number of custom key-value pairs that a user can associate to a document. Under the hood, this specification is sufficient to fulfil the requirement however, the user interface did not allow users to view the property as it was intended.

We created a new custom template control to adjust how users would add and visualize key-value pairs. Figure 4.13 shows the control created for adding key-value pairs. The system's underlying structure is invisible to the user as he only has input fields to populate with the pair and the buttons make intuitive the limitless of values that can be inserted.



study:Netdiamond

budget:15M

local:Aveiro

Figure 4.13: Interface to associate key-value pairs to nodes.

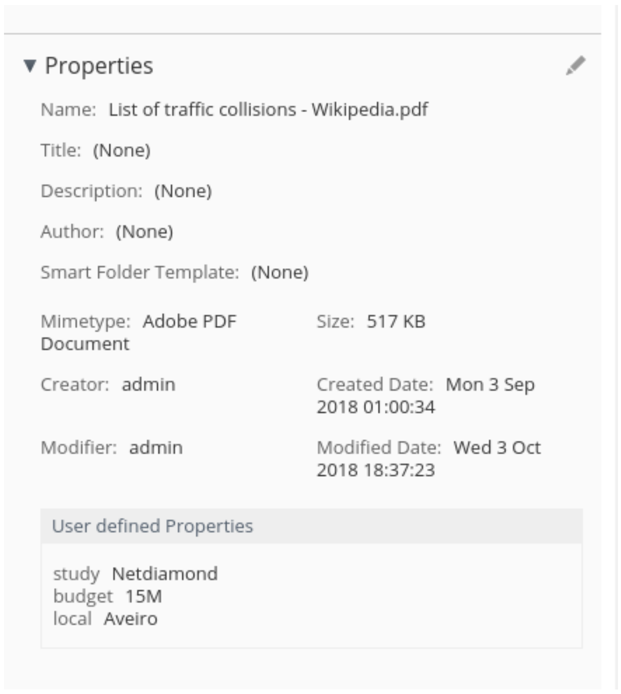
Unlike the rest of the properties in the system, that both the property name and its value provides useful information to the user, all the relevant information in our custom property is stored in the value. When presenting the information there is a parsing of the property content to extract both the key and the value. Figure 4.14 shows the resulting interface.

By default not all properties are used in to search for files in the [25] however, we override this configuration to include our custom property so users can easily access documents that they added custom metadata.

It is important to note that this feature was designed to grant the possibility to add metadata to a document, making the information searchable for a node. If the information added becomes less and less sporadic, a revision of the content model of that file is advised.

4.6.3 Complex query accessibility

Since one of the most important features of our solution is discovering data, we wanted to provide users with powerful tools to allow them to retrieve documents that they wanted to receive. Having the content and metadata of the files indexed contributes for fast and easy retrieval of documents through the Solr search engine that the system uses. As seen


▼ Properties

Name: List of traffic collisions - Wikipedia.pdf

Title: (None)

Description: (None)

Author: (None)

Smart Folder Template: (None)

Mimetype: Adobe PDF Document

Size: 517 KB

Creator: admin

Created Date: Mon 3 Sep 2018 01:00:34

Modifier: admin

Modified Date: Wed 3 Oct 2018 18:37:23

User defined Properties

study Netdiamond
budget 15M
local Aveiro

Figure 4.14: Visualize associate key-value pairs to nodes.

in section 3.2.5, the system's search syntax supports both simple querying and a more advanced querying syntax that allows performing SQL like queries. The system provides an Advanced Search form that grants users the possibility to search form content and specify the values of selected metadata fields. This reduces the number of unsatisfactory results that would have been collected.

For some cases, none of the above solutions works, requiring users to conduct SQL like query to get the results they want. One example of this scenario is when a user wants to search for all genbank files that contain the sequence "*GATCCTCCAT*" but not the sequence "*CTCTTCGAGC*". To accomplish this, the user had to know the query syntax and the variable where that specific metadata was stored.

```
query: genome:subsequences:GATCCTCCAT AND
       NOT (genome:subsequences:CTCTTCGAGC)
```

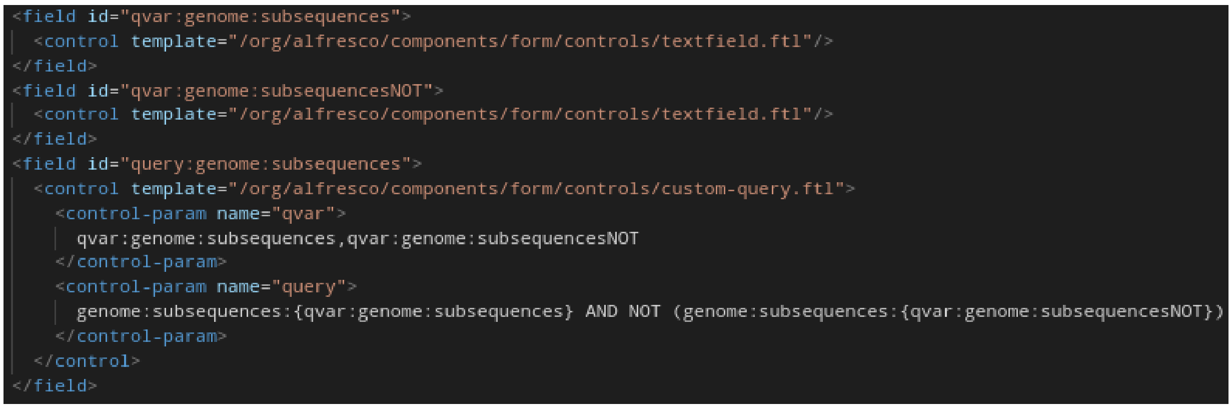
We designed a solution to allow system administrators to construct such queries and hide their complexity from the end user, requiring them only to provide the values. To retain coherence with the rest of the system, we opted to add this functionality in the Advanced Search page. Adding this feature required us to change the class responsible to perform the advanced search query and defining a configuration standard capable of operating with any content model. Queries must be defined in the "*custom-share-configuration.xml*" in agreement with the following syntax:

- **qvar:{property}** - Indicates the content of the control as a variable to be used in the query.
- **query:{property}** - Indicates that the field contains a query definition and must use the *custom-query.ftl* controller.
-
  - **control-param name="qvar"** - Defines the fields that will be used to populate the query.
  - **control-param name="query"** - Defines the query to execute; properties to replace must be inside {} and must be defined in *control-param name="qvar"*

The *custom-query.ftl* controller hides the *query:{property}* and replaces the placeholders in the query with the values from the *qvar:{property}*. The extended Advanced Search class process the *qvar* and *query* identifiers to utilize in the query. Figure 4.15 shows an example of a search form containing the query stated previously.

4.6.4 Show Metadata on upload

When files are inserted into the system, either by dragging them in, or by selecting from the system's file explorer, the system automatically adds metadata to the record, like the file name, the owner of the file and the insertion date, regardless of the file format. In formats supported by the system, they are parsed and their technical metadata is extracted and mapped to the respective model in the system. In this workflow the user had no perception of the system's recognition of the file format nor an identification of the information that was automatically extracted.



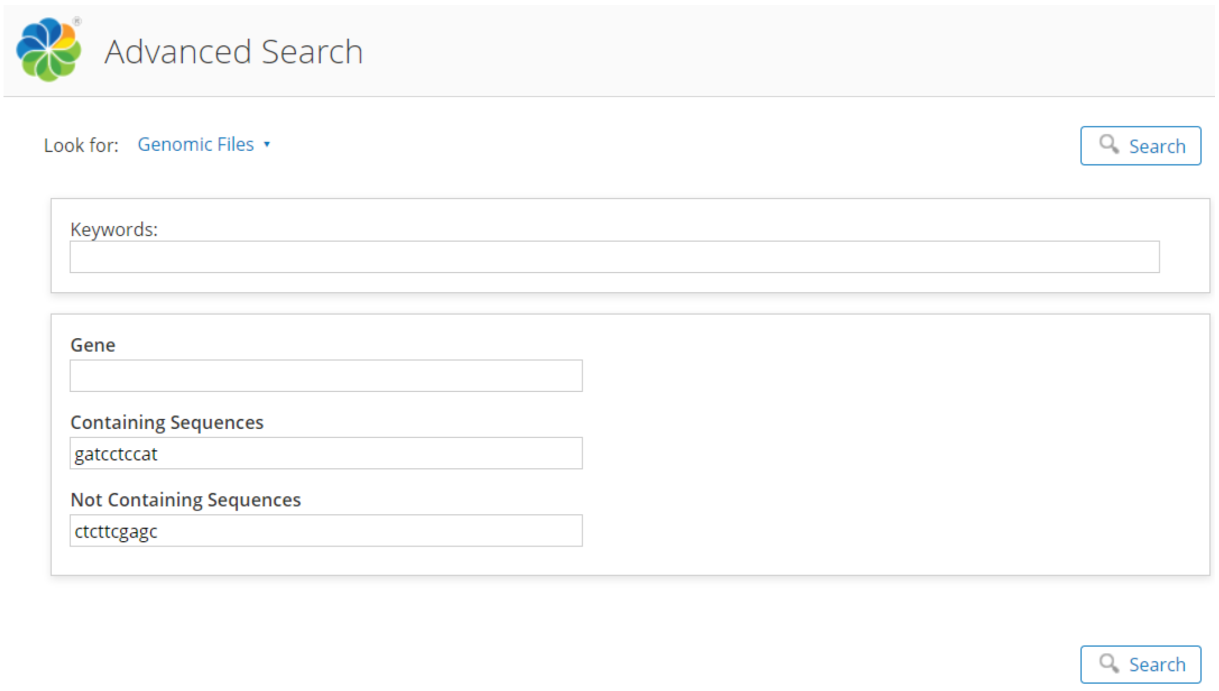
```
<field id="qvar:genome:subsequences">
  <control template="/org/alfresco/components/form/controls/textfield.ftl"/>
</field>
<field id="qvar:genome:subsequencesNOT">
  <control template="/org/alfresco/components/form/controls/textfield.ftl"/>
</field>
<field id="query:genome:subsequences">
  <control template="/org/alfresco/components/form/controls/custom-query.ftl">
    <control-param name="qvar">
      qvar:genome:subsequences,qvar:genome:subsequencesNOT
    </control-param>
    <control-param name="query">
      genome:subsequences:{qvar:genome:subsequences} AND NOT (genome:subsequences:{qvar:genome:subsequencesNOT})
    </control-param>
  </control>
</field>
```

Figure 4.15: Custom query in advanced search configuration.

In our solution, we wanted users to have an overview of the files that were processed, enabling them to understand what information was extracted, gaining awareness of the keywords to search for such files and spotting information that was not detected. We created a new JavaScript class that extended the default class which controlled the upload action, making the extension resilient to possible changes to the class on future versions of Alfresco Share.

After all the files are uploaded to the system, we obtain the information associated with each file via one of the systems base webscripts. By utilizing one of the base webscripts, we take advantage of the internationalization (i18n) feature of the platform, personalizing the language to the user's preferences. A custom webscript was developed to gather all the information and generate a HTML page with the data. We used advantage of the Alfresco Popup Manager[7] to display the file's metadata overview in visual conformity with the rest of the UI. Figure 4.17 shows the pop-up windows resulted from uploading a test file.

[7]http://sharextras.org/jsdoc/share/community-4.0.c/symbols/Alfresco.util.PopupManager.html



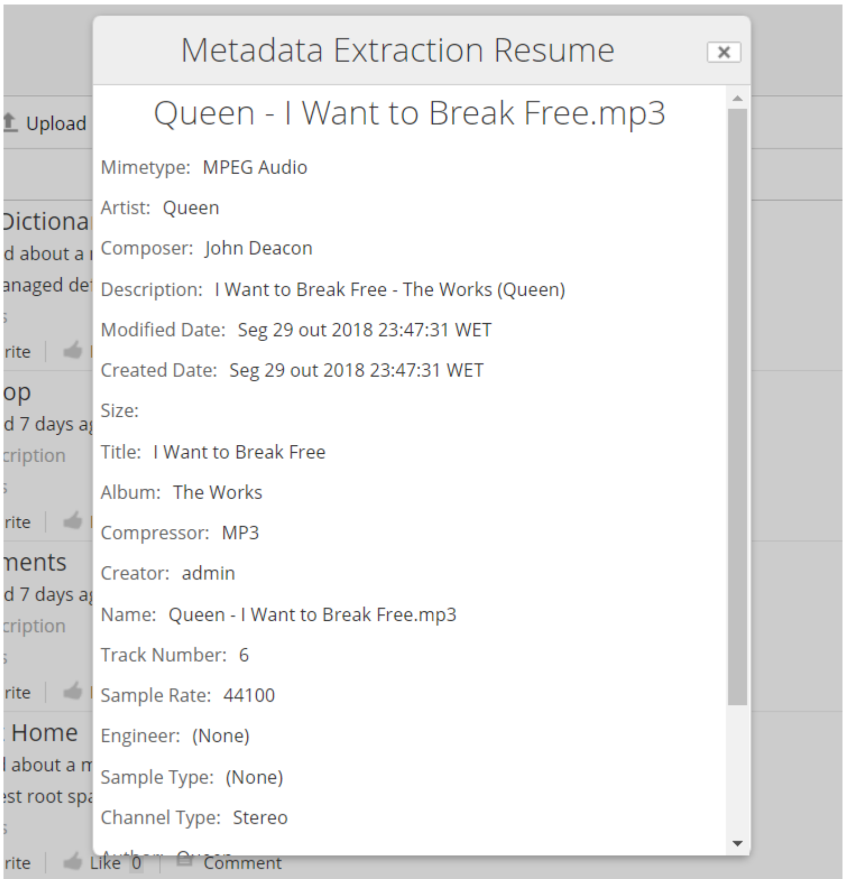
Keywords:

Gene

Containing Sequences
gatcctccat

Not Containing Sequences
ctcttcgagc

Figure 4.16: Advanced Search with complex query.



Metadata Extraction Resume

Queen - I Want to Break Free.mp3

Mimetype: MPEG Audio

Artist: Queen

Composer: John Deacon

Description: I Want to Break Free - The Works (Queen)

Modified Date: Seg 29 out 2018 23:47:31 WET

Created Date: Seg 29 out 2018 23:47:31 WET

Size:

Title: I Want to Break Free

Album: The Works

Compressor: MP3

Creator: admin

Name: Queen - I Want to Break Free.mp3

Track Number: 6

Sample Rate: 44100

Engineer: (None)

Sample Type: (None)

Channel Type: Stereo

Figure 4.17: Pop-up with the information of the extracted metadata after uploading files.

Chapter 5

Conclusions and Future work

This dissertation was developed with the objective of presenting a solution for storing, extracting metadata and searching for files saved in a data repository, which was accessible through a web browser. We began by providing an overview on the concepts of data storage, formats, metadata and data search and after proceed to gather and establish system requirements afterwards. Finding an already assembled solution to extend upon was a main desire, as it provides better maturity, documentation and features than a complete development from the ground up within the time scope of this work, allowing us to focus on specific details in our system requirements.

Although the platform provides good documentation for its customization and extension, some of our developments involved adapting core functionality that is not so well documented, and due to the system complexity, we spent a lot of time understanding and testing how such requirements would be implemented. Nevertheless, we consider that the adoption of Alfresco was an appropriate decision that covers the objectives of this work.

Future work on the solution can pass by the design of new content models in conjunction with the continuous development of *parsers* for new file formats, as there are required over time. This seems to be the best way to add value to the platform, although in this work we opted for exploring other requirements.

With the evolution on the development of Alfresco ADF, the development of a custom web interface that simplifies and removes unused functionality of the platform might be an additional point of interest. Furthermore, although the document management features of Alfresco were not envisioned in our initial requirements, they can potentially be incorporated in the utilization workflows within a production environment.

Bibliography

[1] W. McCarty, *Learning Debian GNU/Linux*, en-US, October 1999. [Online]. Available: `https://www.oreilly.com/openbook/debian/book/ch04_03.html` (visited on 10/31/2018).

[2] "Ieee standard for learning object metadata", *IEEE Std 1484.12.1-2002*, pp. 1–40, Sep. 2002. DOI: `10.1109/IEEESTD.2002.94128`.

[3] L. Woolcott, *Understanding metadata: What is metadata, and what is it for?, by jenn riley. baltimore: National information standards organization, 2017. iii, 45 p. isbn: 978-1-937522-72-8. http://www. niso. org/apps/group_public/download. php/17446/understanding% 20metadata. pdf*, 2017.

[4] J. Greenberg, "Understanding metadata and metadata schemes", *Cataloging & Classification Quarterly*, vol. 40, no. 3-4, pp. 17–36, 2005. DOI: `10.1300/J104v40n03\_02`. eprint: `https://doi.org/10.1300/J104v40n03_02`. [Online]. Available: `https://doi.org/10.1300/J104v40n03_02`.

[5] M. Hayslett, *LibGuides: Metadata for Data Management: A Tutorial: Standards/Schema*, en. [Online]. Available: `https://guides.lib.unc.edu/c.php?g=8749&p=44504` (visited on 10/30/2018).

[6] H. Han, C. L. Giles, E. Manavoglu, H. Zha, Z. Zhang, and E. A. Fox, "Automatic document metadata extraction using support vector machines", in *Proceedings of the 3rd ACM/IEEE-CS joint conference on Digital libraries*, IEEE Computer Society, 2003, pp. 37–48.

[7] N. J. Belkin *et al.*, "Interaction with texts: Information retrieval as information seeking behavior", *Information retrieval*, vol. 93, pp. 55–66, 1993.

[8] D. Hiemstra, "Information retrieval models", *Information Retrieval: searching in the 21st Century*, pp. 2–19, 2009.

[9] C. Middleton and R. Baeza-Yates, "A comparison of open source search engines", 2007.

[10] J. B. Lovins, "Development of a stemming algorithm", *Mech. Translat. & Comp. Linguistics*, vol. 11, no. 1-2, pp. 22–31, 1968.

[11] *DB-Engines Ranking*, en. [Online]. Available: `https://db-engines.com/en/ranking/search+engine` (visited on 10/27/2018).

[12] *Dropbox*, en. [Online]. Available: `https://www.dropbox.com/` (visited on 09/23/2018).

[13] *Google Drive - Cloud Storage & File Backup for Photos, Docs & More*, en_US. [Online]. Available: `www.google.com/drive/` (visited on 09/23/2018).

[14] *6.3.1 archival information package (aip)*. [Online]. Available: `https://www.iasa-web.org/tc04/archival-information-package-aip`.

[15] *Architectural Overview | Alfresco Documentation*. [Online]. Available: `http://docs.alfresco.com/5.0/concepts/dev-arch-overview.html` (visited on 10/28/2018).

[16] *Alfresco Content Services architecture | Alfresco Documentation*. [Online]. Available: `https://docs.alfresco.com/5.2/concepts/dev-arch-overview.html` (visited on 10/28/2018).

[17] *Platform architecture | Alfresco Documentation*. [Online]. Available: `https://docs.alfresco.com/5.2/concepts/dev-platform-arch.html` (visited on 10/20/2018).

[18] *Share Architecture | Alfresco Documentation*. [Online]. Available: `https://docs.alfresco.com/5.2/concepts/dev-extensions-share-architecture-extension-points.html` (visited on 10/27/2018).

[19] *Repository concepts | Alfresco Documentation*. [Online]. Available: `https://docs.alfresco.com/5.2/concepts/dev-repository-concepts.html` (visited on 10/29/2018).

[20] (). Alfresco Scalability Blueprint, [Online]. Available: `https://www.alfresco.com/technical-whitepaper/alfresco-scalability-blueprint` (visited on 10/20/2018).

[21] *Supported Platforms | Alfresco Documentation*. [Online]. Available: `http://docs.alfresco.com/5.2/concepts/supported-platforms-ACS.html` (visited on 10/20/2018).

[22] *BLAST TOPICS*. [Online]. Available: `https://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE_TYPE=BlastDocs&DOC_TYPE=BlastHelp` (visited on 09/30/2018).

[23] *GenBank Overview*. [Online]. Available: `https://www.ncbi.nlm.nih.gov/genbank` (visited on 09/30/2018).

[24] R. C. G. Holland, T. A. Down, M. Pocock, A. Prlić, D. Huen, K. James, S. Foisy, A. Dräger, A. Yates, M. Heuer, and M. J. Schreiber, "Biojava: An open-source framework for bioinformatics", *Bioinformatics*, vol. 24, no. 18, pp. 2096–2097, 2008. DOI: `10.1093/bioinformatics/btn397`. eprint: `/oup/backfile/content_public/journal/bioinformatics/24/18/10.1093/bioinformatics/btn397/2/btn397.pdf`. [Online]. Available: `http://dx.doi.org/10.1093/bioinformatics/btn397`.

[25] *Alfresco Full Text Search Reference | Alfresco Documentation*. [Online]. Available: `https://docs.alfresco.com/5.1/concepts/rm-searchsyntax-intro.html` (visited on 09/30/2018).