



**João Miguel
Costa Génio**

**Mapas de conhecimento como ferramenta de apoio à
gestão de competências científicas**

**Knowledge Maps as support tool for managing
scientific competences**



João Miguel
Costa Génio

**Mapas de conhecimento como ferramenta de apoio à
gestão de competências científicas**

**Knowledge Maps as support tool for managing
scientific competences**

“I want to see if I can. I don't know if I can. I want to find out. I want to see. I'm going to do what I always do: I'm going to break it down to its smallest form, smallest detail, and go after it. Day by day, one day at a time.”

— Kobe Bryant



**João Miguel
Costa Génio**

**Mapas de conhecimento como ferramenta de apoio à
gestão de competências científicas**

**Knowledge Maps as support tool for managing
scientific competences**

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor António José Ribeiro Neves, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e da Doutora Alina Trifan, Professora auxiliar convidada do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

Dedico este trabalho à minha família e amigos pelo incansável apoio que sempre me deram.

o júri / the jury

presidente / president

Professor Doutor Joaquim João Estrela Ribeiro Silvestre Madeira
Professor Auxiliar, Universidade de Aveiro

vogais / examiners committee

Doutora Filipa Campos Soares Borrego
Gestora de Clência, Centro de Investigação em Sistemas Computacionais Embebidos e de Tempo-Real - Cister

Professora Doutora Alina Liliana Trifan
Professora Auxiliar Convidada, Universidade de Aveiro

**agradecimentos /
acknowledgements**

Quero agradecer a todos os que me ajudaram neste longo percurso. À minha família que sempre apoiou as minhas decisões. Aos meus amigos que tanto me aturaram. Aos meus fantásticos orientadores, Professor Doutor António José Ribeiro Neves e Professora Doutora Alina Trifan, por guiarem-me neste enorme desafio. Por fim, quero também agradecer à Universidade de Aveiro que foi muitas vezes a minha segunda casa, mas sempre acolhedora.

Palavras Chave

mapas de conhecimento, mapas conceituais, visualização de grafos, processamento de linguagem natural, gestão de competências científicas, mineração de dados.

Resumo

Numa organização de investigação, encontrar alguém que seja especialista numa área e que possa assumir uma determinada posição, definir áreas de excelência, ou contratar alguém, requer a compreensão das competências internas disponíveis. Este trabalho explora a ideia de mapas de conhecimento ou de competências como instrumento de apoio à gestão de competências científicas, tentando reduzir a lacuna identificada na gestão da investigação no Instituto de Engenharia Electrónica e Informática de Aveiro. Se for bem sucedida, esta ideia poderá também ser implementada em qualquer outra organização de investigação. Os mapas de conhecimento são uma representação visual de informação que pode ser concebida com granularidade variável no que diz respeito aos bens de conhecimento de uma organização. De uma perspectiva de gestão da investigação, os mapas de conhecimento apoiam a descoberta de competências de investigação e fornecem uma visão instantânea de um tópico, mostrando claramente as principais áreas. A solução explorada neste trabalho utilizou abordagens de mineração de dados para recolher informação de bases de dados públicas e apresentá-la em mapas de conhecimento. Outras ferramentas de visualização tais como gráficos de barras, tabelas, filtros e funcionalidades de pesquisa foram criadas e integradas numa plataforma web. Quando reunidos, estes componentes podem transformar esta plataforma numa componente chave para a administração de uma organização de investigação.

Keywords

knowledge maps, concept maps, graph visualization, natural language processing, scientific competence management, data mining.

Abstract

In a research organization, finding someone who is expert in a field and can take up a given role, defining areas of excellence, or employing a new member require understanding the competences available in-house. This work explores the idea of knowledge or competence maps as a support tool for managing scientific competences, trying to reduce the identified research management gap at the Institute of Electronics and Informatics Engineering of Aveiro. If successful, this idea could also be implemented at any other research organization. Knowledge maps are a visual representation of information which can be designed with variable granularity with respect to the knowledge assets of an organization. From a research management perspective, knowledge maps support the discovery of research competences and provide an instant overview of a topic by showing the main areas at a glance. This solution explored in this work employed data mining approaches for gathering information from public databases and presenting it in knowledge maps. Other visualization tools such as bar charts, tables, filters and search functionalities were created and integrated into a web platform. When put together, these components can turn this platform into a key component for the administration of a research organization.

Contents

Contents	i
List of Figures	v
List of Tables	vii
Glossary	ix
1 Introduction	1
1.1 Context and motivation	1
1.2 Background	2
1.2.1 Added value of this project	5
1.3 Objectives	5
1.4 Document structure	5
2 Architecture	7
2.1 Requirements	7
2.2 Technologies	8
2.2.1 Application Programming Interfaces	8
2.2.2 Choosing a data source	9
2.2.3 Platform	10
2.3 System overview	11
3 Data Collection	13
3.1 Scopus	13
3.1.1 Access	14
3.2 Ciência Vitae	15
3.2.1 Access	15
3.3 Data model	16
3.3.1 Authors	16
3.3.2 Publications	17

3.4	Duplicate publications	17
3.4.1	Understanding the flow of information	18
3.4.2	Different types of publications	19
3.4.3	ID matching	20
3.4.4	Merging in ideal conditions	20
3.4.5	Field merging	20
3.4.6	ID inconsistencies	21
3.4.7	Synchronization order	23
3.4.8	Title and abstract analysis	26
3.5	Results	29
3.5.1	Synchronizing <i>Scopus</i>	30
3.5.2	Synchronizing <i>Ciência Vitae</i>	30
3.5.3	Global results	31
3.6	Developer page	32
4	Knowledge Extraction	35
4.1	Graph visualizations	35
4.1.1	Knowledge mapping	35
4.1.2	Performance	36
4.1.3	Key features	36
4.1.4	Framework comparison	37
4.2	Creating graphs	39
4.2.1	Styling	39
4.2.2	Author map	41
4.2.3	Global map	42
4.3	Data viewer	43
4.4	Filters	44
4.4.1	Keywords	44
4.4.2	Date	45
4.4.3	Publication type	46
4.4.4	Headers	47
4.5	Institutional statistics	47
4.5.1	Publications in a time period	47
4.5.2	Projects in a time period	48
4.5.3	Global counters	48
4.5.4	Author specific statistics	49
4.6	Home page	50
4.7	Author page	51

5	Optimization	53
5.1	QuerySets	53
5.2	Prefetching data	54
5.3	Django Debug Toolbar	54
5.3.1	Query awareness	54
5.4	Developer page	55
5.4.1	Optimizing	55
5.4.2	Optimization results	57
5.5	Home page	57
5.5.1	Knowledge map	58
5.5.2	Bar charts	58
5.6	Author page	59
5.6.1	Knowledge map	59
5.6.2	Bar charts	60
6	Conclusion and Future Work	61
6.1	Conclusion	61
6.2	Future work	62
	References	63
	Data Model Diagram	65

List of Figures

1.1	Example of an author’s profile in Authenticus.	3
1.2	Example of a knowledge map using the keywords “robot soccer”, in Open Knowledge Maps.	4
1.3	Main features of Pure.	4
2.1	System overview.	11
3.1	Scope of the <i>Scopus</i> and <i>ScienceDirect</i> APIs.	14
3.2	Duplicate publications, fetched from both APIs.	18
3.3	Example of a publication’s string representation	18
3.4	Example of a publication that is fetched from the <i>Scopus</i> API and is added to the database.	19
3.5	Example of a fetched publication being merged with an already existing one.	19
3.6	Example of merging because of equal <i>Scopus</i> ID.	20
3.7	Flowchart of the field merging process.	21
3.8	Example of an inconsistency.	22
3.9	Main ID inconsistencies.	22
3.10	Example of a type 5 inconsistency.	23
3.11	Flowchart of the ID merging process.	23
3.12	First example of publications of a given author when <i>Scopus</i> is synchronized before <i>Ciência Vitae</i>	24
3.13	First example of publications of a given author when <i>Ciência Vitae</i> is synchronized before <i>Scopus</i>	24
3.14	Second example of publications of a given author when <i>Scopus</i> is synchronized before <i>Ciência Vitae</i>	25
3.15	Second example of publications of a given author when <i>Ciência Vitae</i> is synchronized before <i>Scopus</i>	25
3.16	Text processing pipeline execution before comparing publication’s titles and abstracts.	27
3.17	Two publications that share the same title and abstract.	28
3.18	Two publications where one title contains the other.	29
3.19	Merging when two publications share the same abstract or title.	29
3.20	Results from synchronizing <i>Scopus</i> in an empty database.	30

3.21	Results from synchronizing <i>Ciência Vitae</i> after <i>Scopus</i>	31
3.22	Overview of the developer page.	33
4.1	First example of <i>vis.js</i> 's example page.	40
4.2	Customization of a graph visualization.	41
4.3	Author's collaboration map, of conference papers related with the word "vision", from 1981 to 2020.	42
4.4	Institute's collaboration map, of conference papers related with the word "vision", from 1981 to 2020.	43
4.5	Example of a possible state of the publication viewer.	44
4.6	Example of a possible state of the keyword form, when the user searches for "robotic soccer" related keywords. Some choices are disabled by the user.	45
4.7	Range of options of the "End year" field, when "Start year" has the value "2020" selected.	46
4.8	Options available in the publication type form.	46
4.9	Example of a graph header.	47
4.10	Example of a viewer header.	47
4.11	Example of a bar chart about publications.	48
4.12	Example of a bar chart about projects.	48
4.13	Cards with information about the application's global counters.	49
4.14	Example of an author's most recurring keywords.	49
4.15	Example of an author's most recurring scientific areas.	50
4.16	Example of an author's most common collaborators.	50
4.17	Home page overview.	51
4.18	Author page overview.	52

List of Tables

3.1	<i>Scopus</i> and <i>ScienceDirect</i> API quotas.	15
3.2	Source of the author model's secondary fields.	17
3.3	Source of all publication model's fields.	17
3.4	Publications with a given amount of authors in the database, after the collection phase.	32
5.1	Developer page's iterative gains in optimization.	57
5.2	Different workloads applied to the home page's knowledge map.	58
5.3	Different workloads applied to the home page's publications bar chart.	58
5.4	Different workloads applied to the home page's projects bar chart.	59
5.5	Different workloads applied to the author page's knowledge map.	59

Glossary

2D	Two-Dimensional	JS	JavaScript
3D	Three-Dimensional	NLP	Natural Language Processing
API	Application Programming Interface	NLTK	Natural Language Toolkit
CV	Curriculum Vitae	PDF	Portable Document Format
DETI	Department of Electronics, Telecommunications and Informatics	QA	Quality Assurance
DOI	Digital Object Identifier	SQL	Structured Query Language
FCT	Portuguese Foundation for Science and Technology	SVG	Scalable Vector Graphics
HTML	HyperText Markup Language	UA	University of Aveiro
IEETA	Institute of Electronics and Informatics Engineering of Aveiro	UI	User Interface
		VPN	Virtual Private Network

Introduction

This introductory chapter seeks to go over the context and motivation of this dissertation as well as exploring background information on the topic. Additionally, the objectives and document structure will also be explained.

1.1 CONTEXT AND MOTIVATION

After a research management gap identified at the Institute of Electronics and Informatics Engineering of Aveiro (IEETA)¹ at the University of Aveiro (UA), there was a necessity to evaluate the use of knowledge or competence maps as support tools for the management of scientific competences. An implementation of a computational tool was then thought of, to provide a platform that allows for both managing and gathering knowledge about the research unit. The level of knowledge of an organization lies in its employees and their collaboration, therefore, they are the most important assets that it owns. Managing this knowledge can help an organization expand its intellectual assets and share them with eventual industrial partners [1].

In a research organization, finding someone who is expert in a field and can take up a given role, defining areas of excellence, or employing a new member require understanding the competences available in-house. Moreover, from a research manager's perspective, the need to find an expert to act as the leading investigator of a research plan or as reviewer of a given research proposal could be supported by an appropriate knowledge management tool. Additionally, the added value of the expected output, is that potential industrial partners will be able to identify core interests and personnel of the department to co-lead internships, project proposals or dissertations that target the students of the Department of Electronics, Telecommunications and Informatics (DETI).

Among the many statistics on science, called scientometrics, bibliometrics holds a privileged place. It can be defined as a quantitative analysis of academic publishing, making it one of

¹http://wiki.ieeta.pt/wiki/index.php/Main_Page

the few sub fields concerned with measuring the output side of science [2], [3]. When applied to individual researchers, they can strongly influence their promotion and make their research footprint much more noticeable. They can measure, at the article, journal or author level, the impact that is created in a given academic discipline. At the article level, one can calculate how many times it has been cited by another work, which is dependent on the size of the indexing database used. Similarly, the importance of a journal in a given field can also be calculated. Finally, at the author level, one can find more complex metrics like the popular *h-index* [4], that finds h publications with at least h citations. These kinds of metrics are important at assessing the quantity and quality of an author's research profile. When applied to an organization or institute, they can become one of their greatest assets for management and business. In their raw form, they represent occurrences and other numeric representations. Naturally, the perception of bibliometrics can be enhanced by unique and different types of representations, like charts, diagrams, or knowledge maps.

Knowledge maps are diagrams that represent ideas with nodes and links. They are often used as media for learning activities, lectures and study materials. They can be distinguished by the use of labeled nodes denoting concepts and links denoting relationships among them. The links in a map may be labeled or unlabeled, directional or non-directional [5].

Knowledge maps are a visual representation of information which can be designed with variable granularity with respect to the knowledge assets of an organization. From a research management perspective, knowledge maps support the discovery of research competences, crucial for establishing efficient research and industrial partnerships. Efficient knowledge maps are expected to bring value to those interested in knowledge management and the identification, enhancement, and actualization of the potential of intellectual assets. They facilitate organizational learning and the interaction with outsider stakeholders. Not only do they support the discovery of organizational knowledge, as working knowledge maps are often seen by those outside the organization as a clear sign of its competence. Knowledge maps which fail to bring returns to the organization and its members are either abandoned or left to deteriorate. They represent excellent ways to capture and share explicit knowledge in organizational contexts, and provide an instant overview of a topic by showing the main areas at a glance. Core competency maps, a sub type of knowledge maps, profile employees and their capabilities, thus enabling the exploring of development opportunities, both at an individual and organizational level. This dissertation will explore the application of these knowledge maps in the context of research management.

1.2 BACKGROUND

A broad analysis was carried out to help understand the capabilities of similar platforms and the added value that the proposed project could offer. Although some of solutions were identified, they were not directly applicable to our use case, as it will be detailed next.

*Authenticus*² is a project developed at the University of Porto that aims to build a national repository of publications metadata authored by researchers of Portuguese institutions. Similar

²<https://www.authenticus.pt/>

to this dissertation’s proposed project, the system automatically imports publications from multiple indexing databases like *ORCID* and conducts a redundancy or duplicate checking process [6]. Its development started in 2010, spanning beyond 2015 through a master dissertation [7], but has not been further developed in the last years, so it was not chosen. Figure 1.1 shows an example of an author’s profile in this platform.

The screenshot displays the 'Confirmed Publications' section of an author's profile in Authenticus. At the top, there are search filters: 'Document Source' (All), 'Document Type' (All Document Types), 'Year Start - End' (1991 - 2022), and 'Order' (Year Dsc). Below the filters, the text 'Confirmed Publications: 275' is shown with a help icon. A list of four publications follows, each with a 'Full Text' link and a checkbox. The publications are:

- TITLE:** A De-Identification Pipeline for Ultrasound Medical Images in DICOM Format
AUTHORS: Eriksson Monteiro; Carlos Costa; **Jose Luis Oliveira** (ORCID)
PUBLISHED: 2017, **SOURCE:** JOURNAL OF MEDICAL SYSTEMS, **VOLUME:** 41, **ISSUE:** 5
INDEXED IN: Scopus, WOS, DBLP, CrossRef: 6 **IN MY:** ORCID, DBLP
- TITLE:** A semantic-based workflow for biomedical literature annotation
AUTHORS: Pedro Sernadela; **Jose Luis Oliveira** (ORCID)
PUBLISHED: 2017, **SOURCE:** DATABASE-THE JOURNAL OF BIOLOGICAL DATABASES AND CURATION
INDEXED IN: WOS
- TITLE:** Automated nanopublications generation from biomedical literature
AUTHORS: Sernadela, P; **Oliveira, JL** (ORCID)
PUBLISHED: 2017, **SOURCE:** 5th Portuguese Meeting on Bioengineering, ENBENG 2017 in ENBENG 2017 - 5th Portuguese Meeting on Bioengineering, Proceedings
INDEXED IN: Scopus, CrossRef **IN MY:** ORCID
- TITLE:** General guidelines for biomedical software development
AUTHORS: Luis Bastiao Silva; Rafael C Jimenez; Niklas Blomberg; **José Luis Oliveira** (ORCID)
PUBLISHED: 2017, **SOURCE:** F1000Research, **VOLUME:** 6
INDEXED IN: CrossRef **IN MY:** ORCID

Figure 1.1: Example of an author’s profile in Authenticus.

*Open Knowledge Maps*³ presents to the user a topical overview based on the 100 most relevant documents matching a given query. It uses text similarity to group documents together and create the knowledge maps. It intends to give the users a head start on their scholarly search. Its main goal is to identify relevant areas at a glance and documents related to them. Its main sources are the Public Library of Science⁴ and PubMed⁵. It employs Natural Language Processing (NLP) techniques to build their knowledge maps [8]. Figure 1.2 shows an example of a knowledge map created with the keywords “robot soccer”, using this tool. This platform’s data sources are its main limitation, which makes it unsuitable for our necessities.

*Elsevier’s Pure*⁶ is also an important research information management system. Figure 1.3⁷ illustrates its main features, such as extraction of data from numerous sources and providing workflow improvements for both researchers and institutions. *Pure*’s main disadvantage is that it is not a free tool, therefore ruling it out for our use case.

³<https://openknowledgemaps.org/>

⁴<https://plos.org/>

⁵<https://pubmed.ncbi.nlm.nih.gov/>

⁶<https://www.elsevier.com/solutions/pure>

⁷<https://www.elsevier.com/solutions/pure/how-it-works>

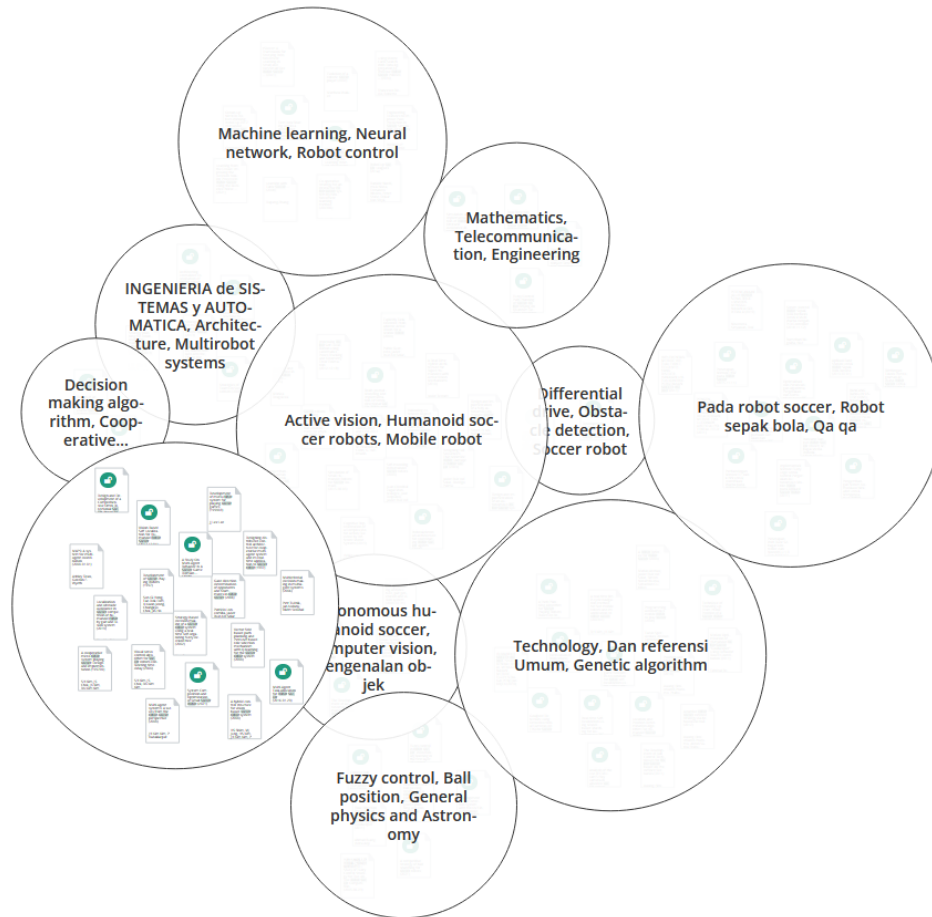


Figure 1.2: Example of a knowledge map using the keywords “robot soccer”, in Open Knowledge Maps.

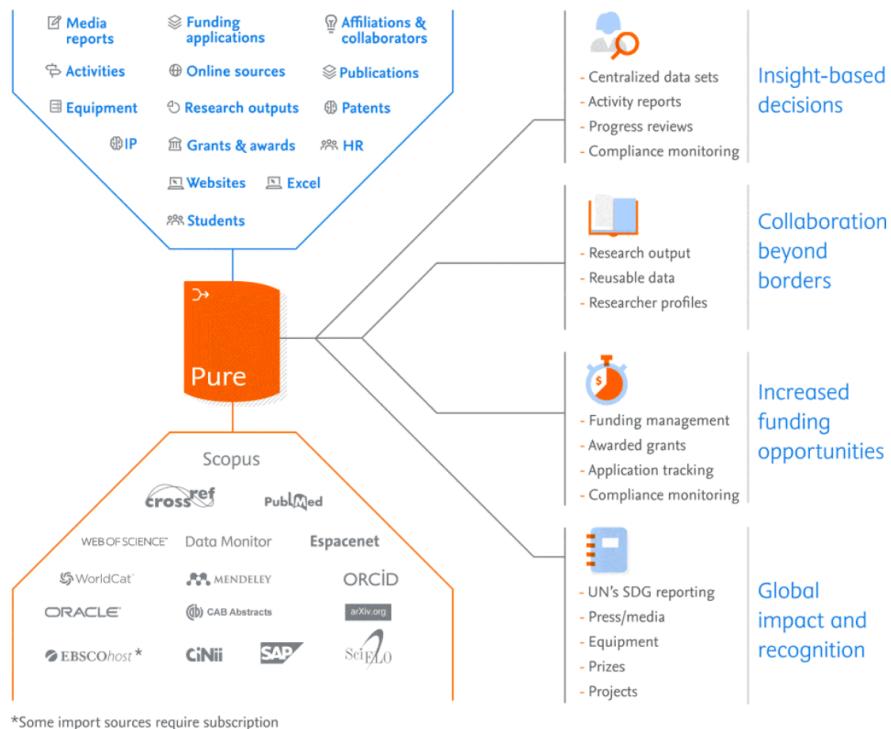


Figure 1.3: Main features of Pure.

1.2.1 Added value of this project

After having identified the problem, this dissertation presents the development of a platform that mixes the main features of a research database, with the added focus on competence management through the aid of knowledge maps. This platform seeks to bring a powerful set of metrics and visualization tools to help achieve management goals, like enhancing the human resources or external partnerships. Through the development of a custom platform, one can tailor its functionalities to the organization's needs. Additionally, this platform would present an improvement over the existing IEETA website, which requires manual input from the researchers and does not facilitate the creation of statistical reports of the scientific output of the institution.

1.3 OBJECTIVES

The main objective of this dissertation is to employ data mining approaches for developing a platform that supports knowledge maps designed to assist on one hand, officers working in science interface groups, and external stakeholders on the other, at identifying the experts and their respective domains of expertise in research-based organizations. It is expected that the resulting knowledge platform will provide a global view of the in-house competences and enable collaborations, both from a research and industrial perspective.

Through public Application Programming Interfaces (APIs) one can get access to an author's publication record and project collaboration history. This data can then be used to model the explicit knowledge of a researcher by employing data mining and keyword extraction techniques. Moreover, a map of the researcher's collaborators can be put together. In this project, only information obtained programmatically is analyzed in order to minimize the dependency on potential manual upload of information by the researchers themselves. Finally, a management platform will be developed, complying with its requisites, with a strong focus on the final output as a knowledge map of individual authors and of the entire organization as well.

1.4 DOCUMENT STRUCTURE

This document will mostly follow the temporal development with one exception:

- Chapter 2 is dedicated to the requirements analysis and system architecture.
- Chapter 3 explores the data collection phase and deals with the eventual problems that will occur.
- Chapter 4 goes into deep detail for the visual representation of data in the application. It also presents the final look and feel of the platform, serving as the final output of the knowledge maps.
- Chapter 5 explains the optimization that was done throughout the entire development phase and also presents some performance tests to complement the process description.
- Finally, Chapter 6 goes through the conclusions of the entire work, reflecting on the quality of implementation as well as presenting ideas for the future.

Architecture

Developing a software platform requires a thorough analysis of the necessities that exist, and what functionalities have to be implemented to satisfy them. This chapter will explain the process of gathering the requirements as well as designing an appropriate system architecture.

2.1 REQUIREMENTS

Currently, IEETA does not have a research management office in place. This means that the requirements for the managing tool that is being conceived have to come from a higher position within this institute. Throughout the development phase of this platform, many meetings took place with this dissertation's supervisors (one of which has a background in science management), and occasionally with IEETA's director, Prof. José Luís Oliveira, who acted as the product owner of the platform. These meetings helped getting a better understanding of what kinds of tools are expected to be present in a system like this. Additionally, these meetings also helped to assure the quality of the implementation at that given moment and feed the definition of formal, functional, and non-functional requirements of the knowledge maps. Special attention must be given to the fact that all stakeholders must be motivated by the knowledge map. If they cannot see the use of the map, they will not use it.

Gathering relevant amounts of information about a set of researchers can be incredibly cumbersome, which is why one of the main objectives that is trying to be accomplished, is being able to find people that specialize in a certain field, within seconds. Obtaining institutional metrics, like the number of articles published in a year, are essential, not only for understanding the existing scientific competences, but also for reporting purposes. Detailed views about individual researchers and publications also need to be implemented because they represent the underlying data structures that will be joined together to create the knowledge maps.

Having all these tools at hand can become extremely useful when trying to raise detailed data about the institute. However, performance has to be kept in check when developing a platform that relies on external APIs. Having to constantly fetch data from them is unfeasible,

as they often impose limits on how many requests can be done within a time period, drastically hindering performance. This is why it is crucial that the gathered data is stored locally, ready to be accessed within instants.

The research platform's purpose is to support search functionalities, which enable the on-the-fly discovery of a person of interest. First, a simple keyword search engine is to be implemented. A research manager as a user of the research map will thus be able to search for specific keywords in order to identify a person of interest. Such keywords can be research areas or topics, programming languages or classes taught. Nonetheless, an advanced search functionality would provide more flexibility to its user.

From a software engineering perspective, a functional requirement defines a function of a system, where a function is described as a specification of behavior between inputs and outputs. A non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. The following functional requirements were identified:

- The platform should have a search functionality. This functionality would enable its users to virtually find the information of interest on the fly.
- The visual interface of the map is an important aspect and it should be designed so as to facilitate the identification of specific human assets or competences.
- Aggregate information should be extracted so as to provide a global view of the in-house competences.

The non-functional requirements are:

- Interoperability - the information presented on the platform should be inter-operable, meaning it could be easily integrated with other systems.
- Replication of information should be avoided at all cost.
- Usability - The system should respond in real-time.
- Scalability - The platform should be easily scalable so as to include a higher number of researchers or cover more research institutions.

2.2 TECHNOLOGIES

This section seeks to explain what major technologies were used and how they were chosen.

2.2.1 Application Programming Interfaces

APIs are used to get programmatic access to content that is, in most cases, already available through some sort of interactive platform (application or web based). Data alone isn't very useful, which is why the access to robust APIs allows one to create innovative tools that enhance the perception of the data.

Scopus

Scopus is one of the largest abstract and citation databases, with more than 17 million author profiles and more than 87 million records¹. It belongs to *Elsevier*², a prestigious academic publishing company that provides many solutions to researchers and institutes, like their APIs and platform for publishing or viewing content.

Ciência Vitae

*Ciência Vitae*³ is a web-based, Curriculum Vitae (CV) management system. It is developed and maintained by the Portuguese Foundation for Science and Technology (FCT)⁴ and is financed by Portugal's Ministry of Education and the European Union. Anyone that acts in the academic or research context in Portugal can create a CV in this platform. This lead to the platform having more than 80,000 researcher CVs, which makes it a key part of a researcher's "footprint". *Ciência Vitae* also allows the user to manually import their data from other well known platforms, like *ORCID*⁵, making it much more versatile and rich in information.

Other data sources

*ORCID*⁶, or *Open Researcher and Contributor ID*, is a unique digital identifier that facilitates searching publications and allows for creating a web profile at an institutional and personal level. Many academic and research institutions require authors to use this identifier, for publishing and career progression purposes.

*Google Scholar*⁷ is a search engine dedicated to scholar literature. It helps finding relevant work across the web by indexing data about articles, books and much more.

Institutional websites

IEETA and UA's⁸ websites provide personal information about the professors and researchers that are included in this study, however, there is not a way to programmatically access it. Scraping these web pages would provide additional data but it was not explored.

2.2.2 Choosing a data source

The first choice of data source is *Scopus*, for its popularity, ability to index data from other publishers and full content in their open access documents. Their documentation [9] and official API wrapper⁹ make the set up process very quick.

Ciência Vitae also needs to be chosen because of its possibility of manually adding content like projects and much richer personal information. This comes at the cost of some information overlap with the previous API, but this problem will be dealt accordingly in the next chapter.

¹<https://www.elsevier.com/solutions/scopus/how-scopus-works/content>

²<https://www.elsevier.com/about/this-is-elsevier>

³<https://www.cienciavitae.pt/>

⁴<https://www.fct.pt/>

⁵<https://orcid.org/>

⁶<https://info.orcid.org/what-is-orcid/>

⁷<https://scholar.google.com/>

⁸<https://www.ua.pt/>

⁹<https://github.com/ElsevierDev/elsapy>

Although *ORCID* has an official¹⁰ *Python*¹¹ API wrapper, it does not add much more information than the previous APIs, while adding a lot of redundancy (note that *Ciência Vitae* can synchronize with this platform).

Google Scholar simply indexes information and will not provide as much detail as the original data sources like *Scopus*. It also does not provide an API, but there are libraries that can scrape search results, which is far from ideal due to the high possibility of the code becoming outdated with *Google Scholar*'s web pages.

The next chapter will go into deeper detail on the chosen data sources.

2.2.3 Platform

Nowadays, almost all daily-use applications reside on the web. For the user, it means not having to deal with installation or update issues, as well as not having control over the application's integrity, hence canceling the chance of breaking it. As for the developer, it is easier to deploy and maintain, while keeping multi-environment support at a maximum. Even some desktop applications are, in reality, web applications in disguise. These applications obtain their User Interface (UI) and interpret it the same way as a browser would [10]. *Electron*¹² apps are rendered through a lightweight *Chromium* based browser engine and are becoming more common by the day. Popular apps include *Slack*, *Microsoft Teams* and *Discord*.

Django

Django, “the web framework for perfectionists with deadlines”¹³, is a *Python* web framework for quickly developing secure, scalable production applications. Choosing this framework boiled down to the simplicity that *Python* provides, as well as a significant personal experience with these technologies. Of course, this decision could not have been made without thoroughly verifying if it meets this application's requirements, which it comfortably does.

In its quickest, most simple configuration, *Django* initializes an *SQLite*¹⁴ database, that it will use to communicate with its own model abstraction layer. The developer is expected not to write Structured Query Language (SQL) code but to call and reference *Django*'s own methods for manipulating the database [11].

User Interface

The UI is a major factor in this system given that it is responsible for taking the processed data from the backend and transforming it into an unique visualization tool. The quality of the rendered information is crucial for correctly interpreting and extracting the scientific knowledge that resides in the database. Additionally, the purpose of this application is not just to visualize, but also to manage. A dashboard type of layout makes more sense than

¹⁰<https://github.com/ORCID/python-orcid>

¹¹<https://www.python.org/>

¹²<https://www.electronjs.org/apps>

¹³<https://www.djangoproject.com/>

¹⁴<https://www.sqlite.org/index.html>

anything else because the user will mostly consume data in the form of numbers, tables and charts.

To set the page layout and styling, *AdminLTE*¹⁵ was chosen, as it allows for a simple HyperText Markup Language (HTML) setup while providing an extensive set of visual elements like tables, panels, forms and much more. It can be easily integrated with frontend frameworks like *React* but this application does not have requirements that justify that added complexity. Chapter 4 will go into deeper detail on the visual representation of the data as bar charts or graphs.

2.3 SYSTEM OVERVIEW

Figure 2.1 represents the entire system at a high level. The flow of information starts at the two data sources, *Elsevier* (2 APIs) and *Ciência Vitae* (1 API), then extracted with the appropriate libraries, stored in a local database, processed by the web framework's backend and displayed to the user with the aid of visualization tools. Ultimately, the user has the ability to perceive the original data in a new, refined way that allows for extracting scientific knowledge from a set of researchers.

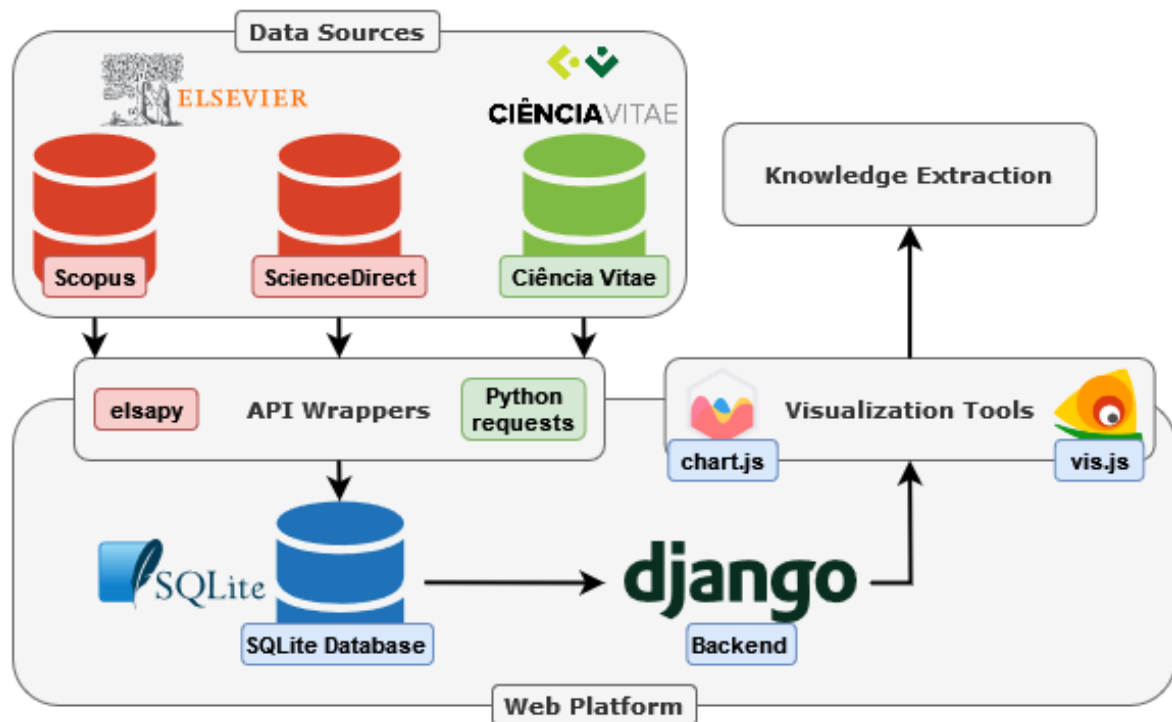


Figure 2.1: System overview.

¹⁵<https://adminlte.io/>

Data Collection

In this chapter, we will explore the Scopus and Ciência Vitae APIs, the process to extract information from them and saving it in our database. Each of these platforms contains useful and unique data, but there's also some overlap between their contents, which means that we will need to find a way to detect duplicate data. This process will consist of simple data processing but also some NLP strategies to analyse textual data. The development process starts with the identification of all IEETA's researchers and the collection of their research IDs, more specifically, their Scopus and Ciência Vitae IDs.

3.1 SCOPUS

In this chapter, we will explore the *Scopus* API as well as the *ScienceDirect* API, but for simplification purposes we will just name them both “*Scopus*”. We will extract relevant publication metadata and abstracts from the *Scopus* API, and full-text from the *ScienceDirect* API, as illustrated by Figure 3.1¹. The first API will include data by many other publishers, not just *Elsevier*, while the second API will include the full-text content just from publications with open access and exclusively from *Elsevier*.

¹<https://dev.elsevier.com/support.html>

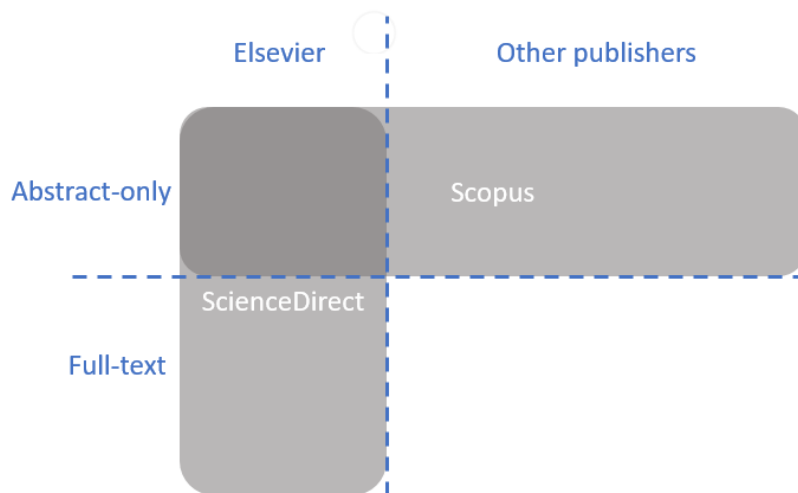


Figure 3.1: Scope of the *Scopus* and *ScienceDirect* APIs.

3.1.1 Access

To get access to the APIs, one needs what is called an API key. This is a special code that indicates the database which users are trying to extract data, and which data is allowed to be given to them.

Policy

Individuals and institutes can access the API for free, as long as they follow their policies². Our purposes follow the “Academic Research” guidelines, given that this is an academic research document and the intended use for this application is as an internal tool for institutes. That policy requires this document to state that all *Scopus* data shown was downloaded from the *Scopus* API between August 15 and September 15, 2022 via <http://api.elsevier.com> and <http://www.scopus.com>.

Permissions

Although the access is free, individuals with no extra permissions will only have access to most publication’s metadata, while institutional users can extract abstracts and full-text content³. Institutional users need to be logged in with their institutional account and need to be located in their institute’s network, either physically or with a Virtual Private Network (VPN).

Quotas

Like most APIs, *Scopus* establishes limits for the number and size of requests. These limits are forced upon the user, but in order to keep our requests from returning an error, the script is forced to sleep for the minimum amount of time that is necessary. Table 3.1 lists the quotas relative to the *Scopus* APIs.⁴

²<https://dev.elsevier.com/policy.html>

³<https://dev.elsevier.com/>

⁴https://dev.elsevier.com/api_key_settings.html

API	Endpoint	Weekly quota	Requests per second
Scopus	Abstract Retrieval	10,000	9
	Author Search	5,000	2
ScienceDirect	Article Retrieval	50,000	10

Table 3.1: *Scopus* and *ScienceDirect* API quotas.

Library and documentation

To interact with the APIs, we used the official *Python* library from *Elsevier*, *elsapy*⁵, as it provides all the operations that we need to perform.

Both the *Scopus* [9] and *ScienceDirect* [12] interactive APIs provide both the necessary documentation and visual tools that help understand the data model and its structure.

3.2 CIÊNCIA VITAE

The *Ciência Vitae* API grants access to all public data, i.e., only the data that users explicitly declare as “private” is hidden. We will use this API to extract personal information about the author, publications that are imported through other platforms, as well as manually added publications. Naturally, some overlap between this and the *Scopus* API is going to occur. That problem will also be thoroughly explored in this chapter.

3.2.1 Access

To get access to the API, a form needs to be filled and sent to the *Ciência Vitae* team⁶. Upon explaining our purpose of accessing the API, we were granted credentials for a Quality Assurance (QA)⁷ environment. This environment consists of a database, filled with some objects that follow the same data structures as the main (production⁸) environment. This is done so that the user can safely test their scripts without worrying about breaking something important. Our intention with the API is to read data, not write, so this process was fairly simple because we just needed to be sure that we were fetching the correct data in a correct way.

Quota

The quota for the API is two requests per second, which is achieved by forcing our program to sleep for half of second on each request. Complying with this limit is specially important because if a user exceeds this limit, their access can be blocked, instead of simply returning an error.

Library and documentation

As of October 2022 there is no publicly available *Python* package or library for accessing the *Ciência Vitae* API, so the solution lies on the many high quality and robust *Python*

⁵<https://github.com/ElsevierDev/elsapy>

⁶<https://www.cienciavitae.pt/contactos/>

⁷<https://qa.cienciavitae.pt/docs/>

⁸<https://api.cienciavitae.pt/docs/>

frameworks for executing requests on the web. We ended up choosing the *requests*⁹ library, as it is one of the most used and it can easily satisfy our needs.

The API provides many endpoints for accessing many types of information, but upon close talks with the *Ciência Vitae*, they suggested using their global endpoint, that provides all the data about a researcher. The drop in performance of doing many requests ceases to exist, and instead, we do a single request with a big response. Their interactive API is very user-friendly and provides all the documentation needed for accessing the data [13]. Once terminated the testing phase, we were granted read-only credentials for the production environment and got access to all public data available in the platform.

3.3 DATA MODEL

The data model of the proposed application, illustrated by the Data Model Diagram appendix, represents all the entities that are going to be stored in our database as well as their relationships. These classes represent the most relevant data that we are able to extract from both the *Scopus* and *Ciência Vitae* APIs, which means that there are some fields that are extracted exclusively from one API and some fields that can be fetched from both, increasing the richness of an object’s information. Ultimately this can cause a publication having more keywords or an author having more publications, which increases the level of “knowledge” that the database has about them. Also note that the diagram’s fields are specified in *Django*’s own categories¹⁰, which are also explained in the appendix.

3.3.1 Authors

The author model is defined by its three main identifiers and name. These fields are optional because one author may not have some of these IDs. The “name” field is filled once a synchronization process is run with any of the APIs, giving priority to the *Ciência Vitae* API, as it should better represent what the authors want in their pages. This priority is achieved by using a flag which indicates that an author has had their *Ciência Vitae* profile synchronized (“synced_ciencia” field). *Scopus* also has a list of name variations, which is also stored in a field called “name_list”.

Table 3.2 lists the source of the remaining fields of the author model. Both the author’s publications and domains can be fetched from both sources but, as we will see in future sections, there is a big overlap of data when dealing with publications.

The *Scopus* page of an author contains information about the number of times the author’s publications have been cited as well as it’s “h-index”. All of them are stored in our database in order to enhance the author’s profile. *Ciência Vitae* provides information on the author’s biography, degrees, distinctions and projects.

⁹<https://requests.readthedocs.io/en/latest/>

¹⁰<https://docs.djangoproject.com/en/4.0/ref/models/fields/>

Field	Scopus	Ciência Vitae
publications	X	X
domains	X	X
h_index	X	
citation_count	X	
cited_by_count	X	
current_affiliations	X	
previous_affiliations	X	
bio		X
degrees		X
distinctions		X
projects		X

Table 3.2: Source of the author model’s secondary fields.

3.3.2 Publications

Publications are fetched from both *Scopus* and *Ciência Vitae*, however, the *Scopus* API can provide more information, like abstracts, full-text content and scientific areas, as illustrated in Table 3.3. Other information, like IDs (except the *Ciência Vitae* ID), title, date, keywords and publication type can be fetched from either APIs.

Field	Scopus	Ciência Vitae
scopus_id	X	X
ciencia_id		X
doi	X	X
title	X	X
date	X	X
keywords	X	X
publication_type	X	X
full_text	X	
abstract	X	
areas	X	

Table 3.3: Source of all publication model’s fields.

There are two fields on this model, “from_scopus” and “from_ciencia”, that will make it easier to understand the source of a publication, i.e., from which API it was fetched, or if it contains information fetched from both.

3.4 DUPLICATE PUBLICATIONS

There are several publications that coexist in both sources, and the same can also happen inside the same API (although much less frequently). These duplicate publications can occur due to the automatic synchronization that is available on the *Ciência Vitae* platform (authors can automatically import publications from *Scopus* and *ORCID*), manual input (also on the *Ciência Vitae* platform) or by some other reason that isn’t known.

Figure 3.2 shows an example of 3 publications fetched from both APIs. The first publication was extracted from *Scopus*. It contains its own ID from the API, can not have a *Ciência Vitae* ID associated with itself, naturally, and it also contains a Digital Object Identifier (DOI). The other two publications were extracted from *Ciência Vitae*. They both contain their own ID from the API, but one publication contains the same DOI as the first publication, while the other contains a reference to the *Scopus* ID from the first publication. This means that these three documents share the same “credentials” and therefore are duplicates.

A methodology to perform semi-automatic distributed EHR database queries		A Methodology to perform semi-automatic distributed EHR database queries		A methodology to perform semi-automatic distributed EHR database queries	
Type	Conference Paper	Type	Conference Paper	Type	Conference Paper
Date	2018-01-01	Date	2018-01-01	Date	2018-01-01
Scopus ID	85051747869	Scopus ID		Scopus ID	85051747869
Ciência Vitae ID		Ciência Vitae ID	1042570	Ciência Vitae ID	2101335
DOI	10.5220/0006579701270134	DOI	10.5220/0006579701270134	DOI	
Keywords	8	Keywords		Keywords	
Areas	4	Areas		Areas	
Abstract	Yes	Abstract		Abstract	
Full Text		Full Text		Full Text	

Figure 3.2: Duplicate publications, fetched from both APIs.

3.4.1 Understanding the flow of information

Throughout this section, we will explore many mechanisms that try to handle duplicate data in different publications. A command line debugging tool was developed to help visualize the flow of information when importing publications from the APIs. It executes database queries and presents data in a perceivable way. Now that we know the data structure of a publication, we can look at a simplified way to represent it.

Figure 3.3 represents a “stringified”¹¹ version of a publication that is added or already exists in the database. This representation is split into several data fields, starting with the publication’s *Scopus* ID, then *Ciência Vitae* ID and DOI. These IDs can have many sizes, so in order to keep the outputs consistent and with the same size, a maximum output size was set, and if the ID is bigger than the output size, then only the last characters of the ID are displayed. This publication’s *Ciência Vitae* ID is “cv-prod-id-2424619” but it is only displayed as “-2424619” because of the size of the output. After that, there are two words, “SC” and “CV”, that represent the fields “from_scopus” and “from_ciencia”, that were already explained in the data model. They are present if the publication contains data from the respective API, and ultimately help understand the source of a given publication.

```
[24644456982] [-2424619] [978-3-540-31956-6-71] [SC CV] [Conference Paper] [1 4 2 Ab ] [2005-01-01]
Active traffic monitoring for heterogeneous environments
```

Figure 3.3: Example of a publication’s string representation

The publication type is the field that’s represented next. Like the ID’s, there are some types that are too long to fit into the output, so they will be truncated to a point where

¹¹Transformed into a string of characters

they are still readable. Following that, is a representation of five fields: the number of authors associated with the publication (only those that exist in the database), the number of keywords, the number of scientific areas, and two slots for the words “Ab” and “FT”, which indicate if the publication contains an abstract or full-text content. The first row ends with the date of publication, and the second row contains the title.

Next up, there are two examples of publications that are fetched from an API. Only one of two things can happen when a publication is fetched: either it is added to the database, or it is detected as a duplicate and then it is merged with another publication that already exists in the database.

Figure 3.4 shows an example of a publication that is fetched from the *Scopus* API and added to the database. This format is similar to the one presented before, but now we have a piece of information at the beginning that tells us that it is a new publication and not a duplicate. There is also a new line at the end that indicates the reason why the publication was added, which in this case it is obvious because it’s a new publication but this will be important in the next example.

```
NEW : [84887971150] [None ] [None ] [SC ] [Conference Paper] [1 6 2 Ab ] [2013-11-25]
Ontology-based health information search: Application to the neurological disease domain
new
```

Figure 3.4: Example of a publication that is fetched from the *Scopus* API and is added to the database.

Figure 3.5 illustrates the merging process of two publications. This time we have three representations of different publications. The first one, called “FETCH”, represents the publication that is being fetched at the moment of the output. The second one, “TEST”, is the publication that already exists in the database and is being tested against the one that is being fetched. Finally, “MERGE” is the result of the two publications being merged with each other. In this case, they were merged because they share the same abstract. The last line becomes specially useful in this case because their abstracts are not displayed on the screen (doing so would make the screen too cluttered). Colors were added to make it much easier to look through an extensive list of publications but they are not important when looking at these isolated examples.

```
FETCH: [84897064805] [None ] [None ] [SC ] [Conference Paper] [ 8 2 Ab ] [2013-01-01]
A multi-domain platform for medical imaging
TEST : [84888996180] [None ] [09/cbms.2013.6627770] [SC ] [Conference Paper] [1 8 2 Ab ] [2013-12-09]
A multi-domain platform for medical imaging
MERGE: [84888996180] [None ] [09/cbms.2013.6627770] [SC ] [Conference Paper] [1 8 4 Ab ] [2013-12-09]
A multi-domain platform for medical imaging
same abstract
```

Figure 3.5: Example of a fetched publication being merged with an already existing one.

3.4.2 Different types of publications

There may be publications that refer to the same document, like a conference paper and a journal article. For this application, it makes more sense to keep publications of different

types for various purposes, like statistics (one could be interested in seeing how many papers and articles an author has published) or other types of data visualization. With this in mind, publications fetched from the APIs won't be merged to ones of different type that already exist in the application's database.

3.4.3 ID matching

As we have seen before, the main thing that suggests that a publication is a duplicate of an existing one, are their equal (or overlapping) IDs. Assuming that two publications are of the same type and they contain at least one ID in common, we want them to be merged. Let's now assume that two publications are extracted from the *Ciência Vitae* API and both refer to the same *Scopus* ID but have different DOIs. These inconsistencies occur for many reasons, which will be explored on the following sections.

3.4.4 Merging in ideal conditions

Figure 3.6 illustrates a publication that was fetched from the *Ciência Vitae* API and tested against an existing publication from *Scopus*. We can see that the fetched publication contains a *Scopus* ID, which tells us that it refers to the existing one. None of them contain a DOI, so they should be merged with certainty.

```

FETCH: [85072918488] [d-878614] [None] [ CV ] [Conference Paper] [ 0 0 ] [2013-01-01]
Self-calibration of colormetric parameters in vision systems for autonomous mobile robots
TEST : [85072918488] [None] [None] [ SC ] [Conference Paper] [1 8 2 Ab ] [2013-01-01]
Self-calibration of colormetric parameters in vision systems for autonomous mobile robots
MERGE: [85072918488] [d-878614] [None] [ SC CV ] [Conference Paper] [1 8 2 Ab ] [2013-01-01]
Self-calibration of colormetric parameters in vision systems for autonomous mobile robots
same scopus

```

Figure 3.6: Example of merging because of equal *Scopus* ID.

3.4.5 Field merging

When merging two publications, there are fields that can always be merged the same way, like the “keywords” field or the “areas” field, however, date, title, abstract and full text have to follow different rules, to avoid updating to a less reliable information (like a less accurate date or title) or erasing information (like updating an existing abstract to an empty one).

Keywords and areas

To merge keywords/areas from two publications, we join both lists and transform it into a set (removing overlapping entries). This way, if one publication has the keywords “A”, “B”, and the other publication has the keywords “A”, “C”, the resulting publication will have the keywords “A”, “B” and “C”.

Date and title

In some cases, the APIs only provide the year of publication, so it gets defaulted to January 1 of that year. This gives us the ability to check if one of the publications that we are comparing is “more recent” and therefore more accurate when it comes to the “date” and “title” fields, so they can be updated.

Scopus fields

When the publication that we are fetching comes from the *Scopus* API, we merge areas (because *Ciência Vitae* publications never refer to any “areas”) and consider updating the abstract and full text. This is only done if the existing information is empty, to prevent overwriting existing information with an empty text.

IDs

Merging IDs might seem as straightforward as considering every ID that isn’t null, but we will see later that there may be some inconsistencies between other IDs. Ideally, if one publication contains a *Scopus* ID and the other one doesn’t, the resulting publication will have the first ID.

Overview

This process, illustrated by Figure 3.7, will slot in at the end of a much bigger process that will be explained in the following sections, but for now this serves as a baseline for what we are trying to accomplish.

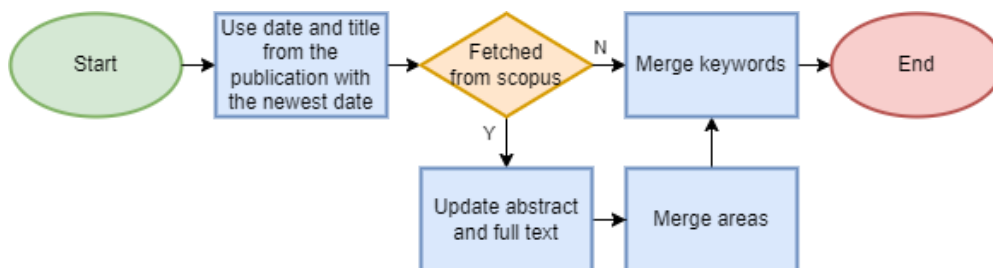


Figure 3.7: Flowchart of the field merging process.

3.4.6 ID inconsistencies

An ID inconsistency occurs when two publications share one ID and differ on another.

Figure 3.8 illustrates an inconsistency on the *Ciência Vitae* ID of the two publications that are being compared. They are both publications from *Ciência Vitae* and share the same DOI. There is information available on these publications that tells us that they should be merged. The fetched publication contains a *Scopus* ID (which the existing publication doesn’t have) but it doesn’t have any keywords (which the other publication has), so all of this useful information needs to be on the merged publication. The general approach to merge a conflicting ID is to not do anything, i.e., the resulting publication will have the ID (in this case, the *Ciência Vitae* ID) of the existing publication.

Figure 3.9 shows the types of inconsistencies that the algorithm detects. There are other combinations of inconsistencies but they are based on these examples, so we won’t be necessarily focusing on them. A type 1 inconsistency occurs when two publications share the same *Scopus* ID but have different *Ciência Vitae* IDs. The previous example (Figure 3.8) demonstrated a type 6 inconsistency.

```

FETCH: [85081588465] [-2269225] [018/ijmbl.2019100105] [ CV ] [Article ] [ 0 0 ] [2019-01-01]
Evaluation of a mobile augmented reality game application as an outdoor learning tool
TEST : [None ] [-1325365] [018/ijmbl.2019100105] [ CV ] [Article ] [1 6 0 ] [2019-01-01]
Evaluation of a Mobile Augmented Reality Game Application as an Outdoor Learning Tool
MERGE: [85081588465] [-1325365] [018/ijmbl.2019100105] [ CV ] [Article ] [1 6 0 ] [2019-01-01]
Evaluation of a Mobile Augmented Reality Game Application as an Outdoor Learning Tool
same doi different ciencia

```

Figure 3.8: Example of an inconsistency.

Inconsistency 1	Inconsistency 3	Inconsistency 5
scopus_id = scopus_id ciencia_id != ciencia_id	scopus_id != scopus_id ciencia_id = ciencia_id	scopus_id != scopus_id doi = doi
Inconsistency 2	Inconsistency 4	Inconsistency 6
scopus_id = scopus_id doi != doi	ciencia_id = ciencia_id doi != doi	ciencia_id != ciencia_id doi = doi

Figure 3.9: Main ID inconsistencies.

Why it happens

Type 1 can occur when a publication from *Ciência Vitae* is introduced manually and then another publication is added automatically, resulting in two publications pointing to the same *Scopus* object. Type 2 can occur if we test an existing publication from *Scopus* against a manually introduced publication from *Ciência Vitae* that points to that publication but has the wrong DOI. This can also happen when an author chooses to save publications in *Ciência Vitae* that represent different book chapters but keeping the *Scopus* reference that points to the entire book. Types 3 and 4 can occur when you synchronize *Ciência Vitae* for the second time. Because some publications were merged and some of the inconsistent IDs were discarded. They will be merged again, resulting in no changes, (because we “prefer” the existing IDs) and the remaining data (like keywords) was already stored on the previous synchronization. Type 5 can occur due to duplicate publications existing in *Scopus* (these duplicates exist for unknown reasons) or due to manual input errors in *Ciência Vitae*. Type 6 can occur when duplicate publications in *Ciência Vitae* point to the same DOIs.

Solution

In summary, we choose to preserve the existing ID for every type of inconsistency except for types 2 and 5. Those types represent special cases where we can increase the accuracy of the resulting publication’s data.

Figure 3.10 illustrates a publication fetched from the *Ciência Vitae* API and compared against an existing publication from *Scopus*.

These publications share the same DOI but have different *Scopus* IDs. When this happens, instead of preserving the *Scopus* ID of the existing publication, we should prioritize the *Scopus* ID of the publication that was fetched from *Scopus* because we assume that the other publication suffered a manual input error.


```

FETCH: [84933051386] [-1051333] [9/roman.2014.6926246] [ CV ] [Conference Paper] [ 0 0 ] [2014-01-01]
Interactive teaching and experience extraction for learning about objects and robot activities
TEST : [84937567763] [None ] [9/roman.2014.6926246] [SC ] [Conference Paper] [1 8 5 Ab ] [2014-10-15]
Interactive teaching and experience extraction for learning about objects and robot activities
MERGE: [84937567763] [-1051333] [9/roman.2014.6926246] [SC CV ] [Conference Paper] [1 8 5 Ab ] [2014-10-15]
Interactive teaching and experience extraction for learning about objects and robot activities
same doi different scopus

```

Figure 3.10: Example of a type 5 inconsistency.

For type 2 inconsistencies, we use the same approach, i.e., using the DOI of the *Scopus* publication.

Overview

Figure 3.11 illustrates the mechanism that was just explained, however, there is still a process named “Title and abstract analysis” that will be explained in later sections.

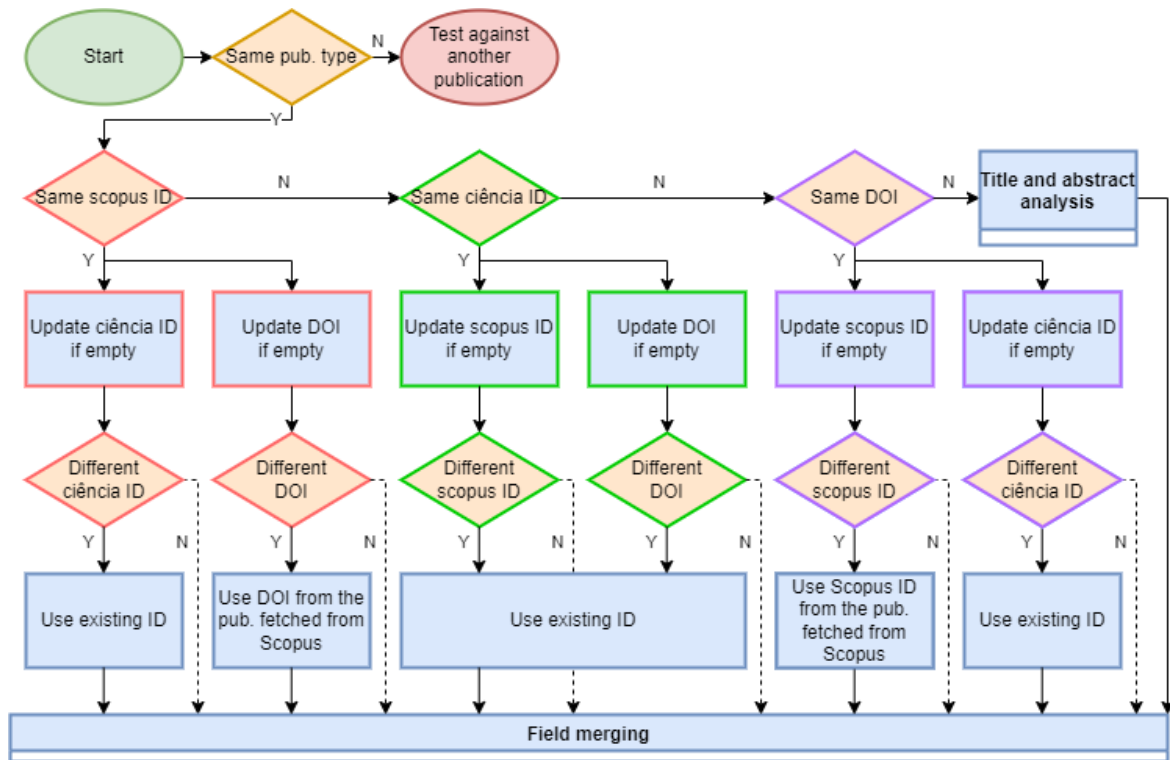


Figure 3.11: Flowchart of the ID merging process.

3.4.7 Synchronization order

In an ideal merging process, there should be no difference between synchronizing *Scopus* before *Ciéncia Vitae* and vice versa, however, there are some inconsistencies that will affect the resulting publications. We will see an example that was solved with mechanisms that we already presented and an example that will be solved with a new analysis process.

Test scenarios

In order to understand the differences between synchronizing one API before another, we need to set up an automated process that saves the state of the database after synchronizing

both APIs. To do this, we first have to make sure that the database is empty of any publications. After that, we synchronize one API (let's say *Scopus*) and then the other (*Ciência Vitae*). The database now contains what should be the ideal number of publications and the most accurate data, but that is not the case. The final step is to store these publications in a file, and repeat the whole process again with the reverse synchronization order and a different file.

This test was done not on the entire database at once, but one author at a time, in order to keep the interpretation easier.

The final output of this test is two lists of publications, but because some authors have a great amount of publications, another debugging mechanism was added to highlight only the differences between these lists. The examples that we will explore will present the different publications on each list.

Example 1

In this example we can see two publications of a given author that have a different DOI depending on the synchronization order (see the third field in square brackets)

Figure 3.12 shows us that when *Scopus* is synchronized before *Ciência Vitae*, the first publication has the following DOI: *10.1007/978-3-540-31956-6-71* (Remember that this visualization tool only shows the last characters of a field when it is too large to fit in the brackets). But as we can see on Figure 3.13, the DOI is different, due to manual input on the *Ciência Vitae* API.

```

-----
[24644456982] [-2424619] [978-3-540-31956-6-71] [SC CV] [Conference Paper] [1 4 2 Ab ] [2005-01-01]
Active traffic monitoring for heterogeneous environments
-----
[35248856335] [-2426240] [978-3-540-45076-4-55] [SC CV] [Article ] [1 8 2 Ab ] [2003-01-01]
Modeling self-similar traffic through Markov modulated poisson processes over multiple time scales
-----

```

Figure 3.12: First example of publications of a given author when *Scopus* is synchronized before *Ciência Vitae*.

```

-----
[24644456982] [-2424619] [10.1007/b107117 ] [SC CV] [Conference Paper] [1 4 2 Ab ] [2005-01-01]
Active traffic monitoring for heterogeneous environments
-----
[35248856335] [-2426240] [10.1007/b11808 ] [SC CV] [Article ] [1 8 2 Ab ] [2003-01-01]
Modeling self-similar traffic through Markov modulated poisson processes over multiple time scales
-----

```

Figure 3.13: First example of publications of a given author when *Ciência Vitae* is synchronized before *Scopus*.

This type of difference in synchronization was solved by prioritizing the fields from *Scopus*, as we have seen before.

Example 2

The inconsistency that we will explore in this example was identified through the same debugging tool discussed in the previous one, but in order to understand it better, we'll be

focusing on what happened when these different publications were added or merged into the database.

Figure 3.14 shows two publications fetched from *Ciência Vitae*. The first publication is merged with an already existing publication, from *Scopus*. The second publication that is fetched, is merged with the resulting publication of the previous merge (see that the publication that we are testing against the fetched one has the “SC” and “CV” indicators). This results in only one final publication existing instead of three, which is ideal.

```

-----
FETCH: [None ] [id-1042570] [220/0006579701270134] [ CV] [Conference Paper] [ 0 0 ] [2018-01-01]
A Methodology to Perform Semi-automatic Distributed EHR Database Queries
TEST : [85051747869] [None ] [220/0006579701270134] [SC ] [Conference Paper] [ 8 4 Ab ] [2018-01-01]
A methodology to perform semi-automatic distributed EHR database queries
MERGE: [85051747869] [id-1042570] [220/0006579701270134] [SC CV] [Conference Paper] [ 8 4 Ab ] [2018-01-01]
A methodology to perform semi-automatic distributed EHR database queries
same doi
-----
FETCH: [85051747869] [id-2101355] [None ] [ CV] [Conference Paper] [ 0 0 ] [2018-01-01]
A methodology to perform semi-automatic distributed EHR database queries
TEST : [85051747869] [id-1042570] [220/0006579701270134] [SC CV] [Conference Paper] [ 8 4 Ab ] [2018-01-01]
A methodology to perform semi-automatic distributed EHR database queries
MERGE: [85051747869] [id-1042570] [220/0006579701270134] [SC CV] [Conference Paper] [ 8 4 Ab ] [2018-01-01]
A methodology to perform semi-automatic distributed EHR database queries
same scopus different ciencia

```

Figure 3.14: Second example of publications of a given author when *Scopus* is synchronized before *Ciência Vitae*.

Figure 3.15 also shows us two publications fetched from *Ciência Vitae*, but this time there are no existing publications from *Scopus*. The first publication is added because there are no existing publications that share any fields with it. The second publication should be merged with the previous one but isn’t because they don’t share any fields. Then, when we fetch the *Scopus* publication, it is merged with the first one because they have the same DOIs. This results in two publications existing instead of three, but it isn’t ideal.

```

-----
NEW : [None ] [id-1042570] [220/0006579701270134] [ CV] [Conference Paper] [ 0 0 ] [2018-01-01]
A Methodology to Perform Semi-automatic Distributed EHR Database Queries
new
-----
NEW : [85051747869] [id-2101355] [None ] [ CV] [Conference Paper] [ 0 0 ] [2018-01-01]
A methodology to perform semi-automatic distributed EHR database queries
new
-----
FETCH: [85051747869] [None ] [220/0006579701270134] [SC ] [Conference Paper] [ 8 4 Ab ] [2018-01-01]
A methodology to perform semi-automatic distributed EHR database queries
TEST : [None ] [id-1042570] [220/0006579701270134] [ CV] [Conference Paper] [ 0 0 ] [2018-01-01]
A Methodology to Perform Semi-automatic Distributed EHR Database Queries
MERGE: [85051747869] [id-1042570] [220/0006579701270134] [SC CV] [Conference Paper] [ 8 4 Ab ] [2018-01-01]
A methodology to perform semi-automatic distributed EHR database queries
same doi

```

Figure 3.15: Second example of publications of a given author when *Ciência Vitae* is synchronized before *Scopus*.

To solve this, we need to extend our analysis beyond the IDs and start analyzing both the title and the abstract of the publications, which will be explained shortly.

3.4.8 Title and abstract analysis

When two publications don't share any IDs between them, there is still a small chance that they refer to the same document, like when a publication from the *Ciência Vitae* API doesn't include a *Scopus* ID or a DOI.

Text processing pipeline

When we want to compare the titles or abstracts of two publications, we have to take into consideration many factors like special or upper case characters, among others. This process will fall into the domain of NLP. NLP focuses on giving computers the ability to understand text the same way human beings can, and that's what we want to achieve in this process.

In many of the pipeline's stages, we use one of the most popular *Python* NLP libraries, Natural Language Toolkit (NLTK)¹². The decision was made based on its ease of use and detailed documentation. This process doesn't require any complex function, so that ended being the main factor when choosing a library. Other heavily considered libraries include *scikit-learn*¹³ and *SpaCy*¹⁴.

Figure 3.16 describes the necessary processing that we need to do in order to compare two strings (titles or abstracts), that are different in their character structure but equal in meaning. It also includes a publication's title as an example.

Some existing processing pipelines may transform the string into lower case at a later stage but, in this specific application, it does not change the result and it is the first step that one usually thinks of when designing a pipeline like this.

The second step will separate words that are connected by special characters and it's done using static character replacing (like replacing all dashes with a space).

"Tokenization" focuses on separating a string into smaller pieces called "tokens". This sets up the next steps for removing non alphanumeric tokens and, finally, removing stop words (like "to" and "the").

The example on Figure 3.16 shows how we transformed the title "Ontology-based health information search: Application to the neurological disease domain" to "ontology based health information search application neurological disease domain". Note that this can also be applied to abstracts, where it can result in a much more efficient comparison later on, due to the removal of stop words and other elements.

¹²<https://www.nltk.org/>

¹³https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html

¹⁴<https://spacy.io/>

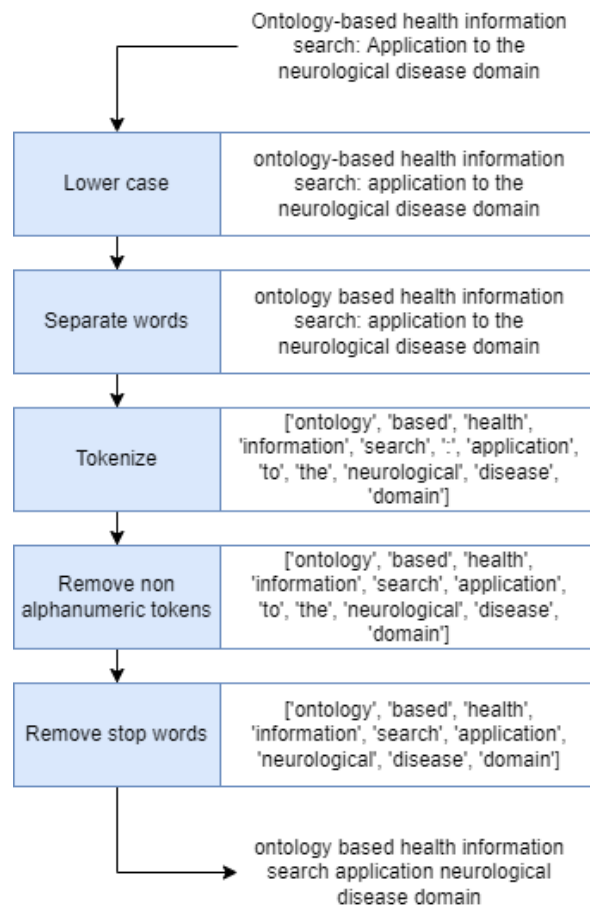


Figure 3.16: Text processing pipeline execution before comparing publication’s titles and abstracts.

Comparing

Now that we are able to transform two strings into something that has a higher chance of telling us if they’re equal, we need a way to craft a metric that indicates us how similar they are. *Python*’s built-in library, *difflib*, provides the `SequenceMatcher` class for comparing two sequences or, in this case, strings. It resulted in a clear separation from the strings that are slightly similar, and strings that are similar enough to be considered equal. This tool is supposed to be used on an automated process, and not manually, so once assured that it works, the decision was closed. The assurance was made manually, by checking all pairs of titles and abstracts with a similarity ratio higher than 70%. This ratio indicated very small similarities between each pair of strings but helped reduce the number of comparisons that were worth analysing, making the manual assurance possible.

ID merging

Now that we have the ability to evaluate how similar a pair of titles or abstracts is, we need a similar mechanism to Figure 3.11, that handles the merging of the publication’s IDs and the situations that it should occur.

To drastically narrow down the amount of title and abstract comparisons, we only consider

pairs of publications of the same year. Although it's possible that one publication has the wrong year, no cases of pairs that failed the main ID checking (described in Figure 3.11) where they had equal titles or abstracts.

The first case, that is checked, is when two publications share the same abstract. To determine this, we check if their similarity ratio is higher than 95%. When abstracts are fetched, they usually contain some special characters and IDs before the actual text. Most of it is removed but, occasionally, some characters end up staying. This allows for two abstracts that are “equal to the eye” to have differences that should not be there. Considering that every occurrence of this fell above the 95% ratio, and because different abstracts end up having extremely low similarity ratios (below 10%), this ratio is suitable for checking this type of scenario. This means that this ratio will include all publications with the same abstract, and zero publications with different abstracts, when referring to this data set.

The second case is when two publications share the same title. The only reason a 100% ratio isn't used is because one publication had a typo that steered the similarity ratio to somewhere between 99% and 100%. In testing, this value only increased the number of occasions by one (the publication with a typo). Figure 3.17 shows an example of two publications that share the same title and abstract.

```
[84888996180] [-1364892] [09/cbms.2013.6627770] [SC CV] [Conference Paper] [2 8 2 Ab ] [2013-12-09]
A multi-domain platform for medical imaging
vs
[84897064805] [None ] [None ] [SC ] [Conference Paper] [2 8 2 Ab ] [2013-01-01]
A multi-domain platform for medical imaging

multi domain platform medical imaging
vs
multi domain platform medical imaging
Title: 1.000

The increasing adoption of medical imaging equipment in healthcare has been leading to a huge dispersio
vs
The increasing adoption of medical imaging equipment in healthcare has been leading to a huge dispersio
Abstract: 1.000
```

Figure 3.17: Two publications that share the same title and abstract.

The last case occurs when a publication contains words that shouldn't be considered, like when a version of a publication that was saved in a given API was waiting for a peer review. This condition is activated when one title is contained in the other. Figure 3.18 shows an example of this occurrence.

If one of these three scenarios is true, then the publications are treated as duplicates and their fields need to be merged. The process that handles the merging of the keywords and areas is placed at the end of the overall process, described in figure 3.11, so we only have to worry about merging the IDs.

The first step, as described in Figure 3.19, is to update the existing publication's IDs if they're empty (null). The final step applies to publications fetched from *Scopus*. We choose to prioritize the IDs from the publication that has the most information, i.e. the *Scopus* ID, DOI and abstract (each one adding one “point” to a publication's level of information).

```

[None ] [-1364731] [1000research.10750.2] [ CV ] [Article ] [1 0 0 ] [2017-07-12]
General guidelines for biomedical software development
vs
[None ] [-1364732] [1000research.10750.1] [ CV ] [Article ] [1 0 0 ] [2017-03-15]
General guidelines for biomedical software development [version 1; referees: awaiting peer review]

general guidelines biomedical software development
vs
general guidelines biomedical software development version 1 referees awaiting peer review
Title: 0.714

N/A
vs
N/A
Abstract: N/A

```

Figure 3.18: Two publications where one title contains the other.

In summary, if none of the three conditions are true, we can't consider the two publications in question to be duplicates, but if one condition ends up being true, we merge the IDs and then we continue to the final process of merging the remaining fields.

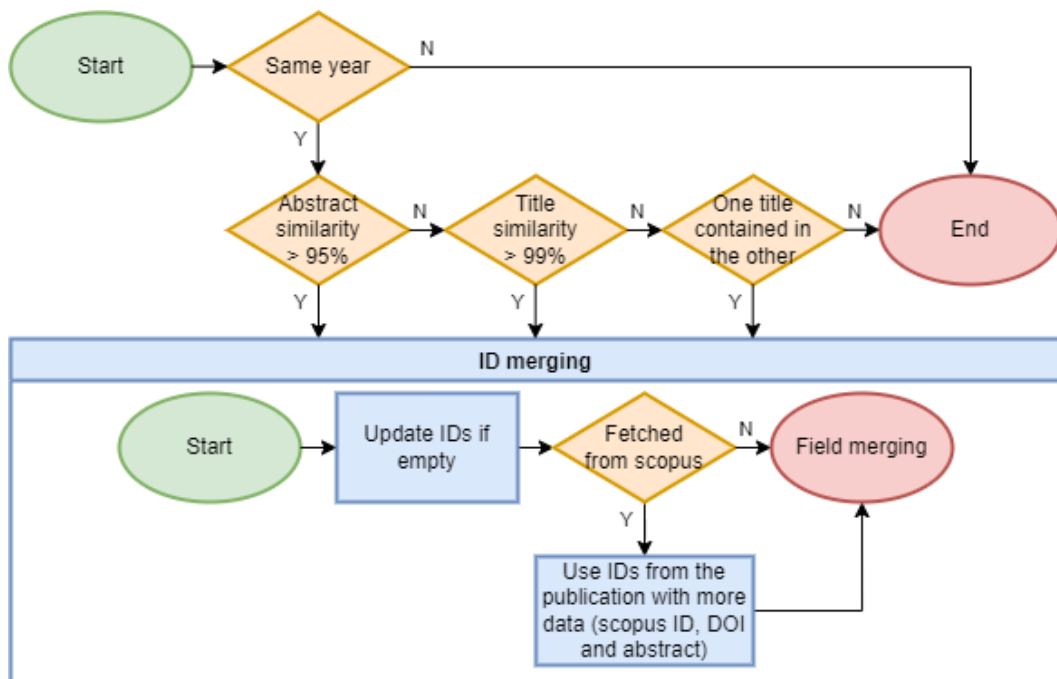


Figure 3.19: Merging when two publications share the same abstract or title.

3.5 RESULTS

To validate the developed algorithms, we will run a synchronization process on the *Scopus* and *Ciência Vitae* APIs, using every merging process that was presented before. This process will be executed on the profiles of 50 authors that were gathered from both IEETA and DETI's websites.

3.5.1 Synchronizing *Scopus*

Figure 3.20 presents the results of the first synchronization process (*Scopus*).

There were a total of 3969 publications that were fetched from the authors' profiles, meaning that some of these publications that were fetched, will be duplicates, because many authors share publications between them. 2773 publications were added into the database, while 1196 were merged either because they also belong to other authors or they shared an ID, title or abstract with an existing publication.

As one can see on the third row of Figure 3.20, 1167 publications were merged for having the same *Scopus* ID, meaning that they are publications that are shared by multiple authors. Because there can not be two publications with the same *Scopus* ID on the *Scopus* API, there were no ID inconsistencies when merging.

Publications from this API don't have a *Ciência Vitae* ID, so no merges were done by this criterion.

Four publications were merged for sharing the same DOI with some other publication (or publications). Every instance of this represents an inconsistency because there can't be any publications with the same *Scopus* ID.

The title and abstract analysis resulted in 18 publications being merged by their abstract and 7 by their title.

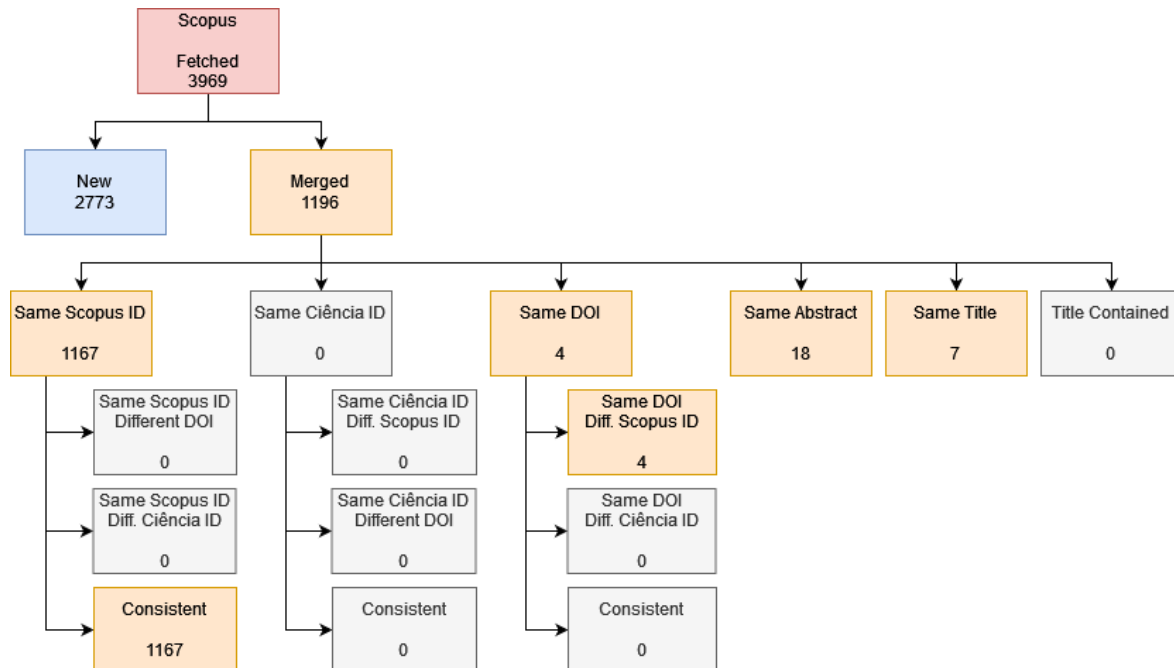


Figure 3.20: Results from synchronizing *Scopus* in an empty database.

3.5.2 Synchronizing *Ciência Vitae*

Figure 3.21 presents the results from synchronizing the *Ciência Vitae* API in a database that has already been synchronized with *Scopus*.

From the 3834 publications that were fetched from the *Ciência Vitae* API, 1596 were added into the database as new publications, while 2238 were merged into existing ones.

Of the publications that were merged, 1588 had the same *Scopus* ID as some other publication that already existed in the database. We can see some inconsistencies, like publications that refer to the same *Scopus* ID but have different DOIs, or some publications from the *Ciência Vitae* API that share the same *Scopus* ID. No publications were merged with the same *Ciência Vitae* ID because when different authors choose to run the platform’s tool that automatically adds the *Scopus* publications to their *Ciência Vitae* profile, those publications will all have different internal IDs (*Ciência Vitae* IDs).

344 publications were merged for sharing the same DOI with some other publication (or publications), registering some inconsistencies along the way.

No publications were merged by their abstract because *Ciência Vitae* publications don’t include one to begin with.

Finally, 272 publications were merged by their title and 34 by the “Title Contained” criterion, which means that they either contained a title or their title were contained in another publication.

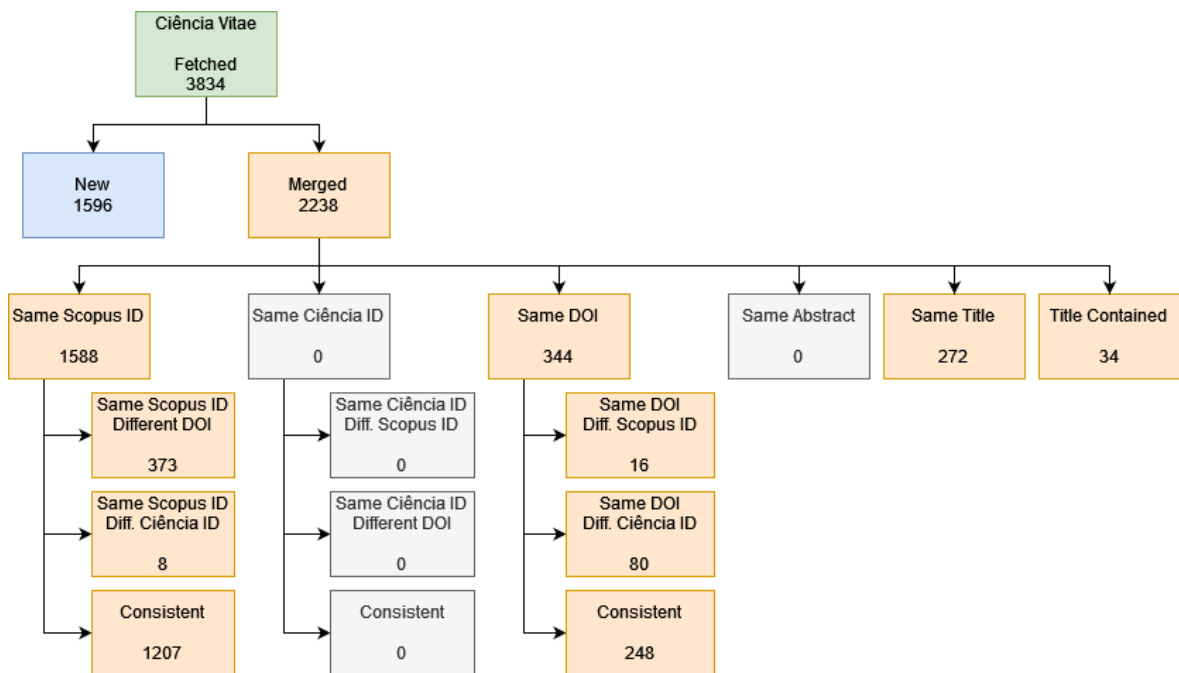


Figure 3.21: Results from synchronizing *Ciência Vitae* after *Scopus*.

3.5.3 Global results

In summary, there were a total of 7803 publications that were fetched from the authors’ profiles, 4369 of which were added as new publications and 3434 were merged into existing ones. When looking at the merged publications we can conclude that 3103 were merged by their IDs and 331 were merged because of the title and abstract analysis process. Additionally, table 3.4 shows the number of publications with a given amount of authors.

Number of authors	Number of publications
1	3457
2	660
3	196
4	38
5	12
6	5
7	0
8	1

Table 3.4: Publications with a given amount of authors in the database, after the collection phase.

11964 different keywords were gathered from these publications, as well as 262 scientific areas that were associated either with publications or directly with the authors' profiles. Finally, 144 projects were fetched from the authors' *Ciência Vitae* profiles.

3.6 DEVELOPER PAGE

In early to mid stages of developing, there was a necessity for having a place where every entity was listed. This place needed to have UI tools to create authors, synchronize their *Scopus* and *Ciência Vitae* profiles and, most importantly, view their personal information and list all the data associated with their publications. The developer page was then created to facilitate the management of the constant change of information. As Figure 3.22 indicates, this page contains a form for adding a new author and a listing of all authors in the database. Each one of the listings contains the author's fields and expandable sections to present data such as publications, projects and more. Above each listing there are buttons that execute different operations on the author. The first three synchronize the author's *Scopus* personal information, *Scopus* publications and *Ciência Vitae* data, respectively. Finally, the last button deletes the author but not their publications, as they can be associated with other authors as well. Although this page has the option of manually synchronizing an author's profile, doing that for every author is unfeasible, as the entire synchronization process takes a few hours to complete, due to the APIs' quota constraints. The final implementation intends to run this synchronization process automatically at some time at night, to avoid hampering the user experience.

Add Author

Scopus id: Ciència id: Orcid id:

All Authors


Photo	
Name	António José Ribeiro Neves
Scopus ID	43461712600
Ciência ID	6016-3F49-8427
ORCID ID	0000-0001-5433-6667
Name List	['António Neves', 'Antonio J.R. Neves', 'A. J.R. Neves', 'Antóio J.R. Neves', 'António J.R. Neves']
H-index	14
Citation Count	566
Cited-by Count	408
Current Affiliations	Universidade de Aveiro
Previous Affiliations	▶ 18 Affiliations
Bio	▶ Available
Degrees	▶ 3 Degrees
Distinctions	▶ 40 Distinctions
Projects	▶ 25 Projects
Domains	▶ 38 Domains
Publications	▶ 215 Publications


Photo	
Name	Ana Maria Perfeito Tome
Scopus ID	7006456423
Ciência ID	3515-EA0C-24B9
ORCID ID	0000-0002-1210-0266
	['Ana M.P. Tome', 'Ana P. Tome', 'Ana Tomé', 'Ana M. Tomé', 'Anna M. Tomé', 'Ana M. Tome', 'Ana Maria Tomé', 'A. M. Tomé', 'A. M.

Figure 3.22: Overview of the developer page.

Knowledge Extraction

This chapter presents the developed work that transforms the collected data into a series of visual representations. Various approaches will be presented, with a special focus on graph visualizations. The process of choosing a framework will be explained here, as well as the steps required to achieve such representations. Standard charts and statistics will be crafted, and graphs will relate two of the most important entities in this application, the authors and their publications. Finally, all these tools will be analyzed from an institutional perspective in order to evaluate how much knowledge they can provide.

4.1 GRAPH VISUALIZATIONS

Graphs are defined by a set of nodes and a set of edges that connect nodes. They are used to present large amounts of data in an easy to digest and perceivable way. They can be used to show stations and links of a complex subway system, visually define an enterprise network architecture and much more.

4.1.1 Knowledge mapping

The ultimate goal is to map scientific knowledge into a visually unique and useful representation, that is also fast to understand (a knowledge map). Graph visualizations satisfy all those requirements, because they can not be transformed into a standard representation, like a table or a chart, without abdicating readability and conciseness. Although they are still represented in a Two-Dimensional (2D) plane, they allow to perceive non apparent levels of connections, increasing the richness of information. That's why this type of representation will respond better to an administrator's needs when analyzing an institute's scientific competence. The following sections will focus the key characteristics that a visualization framework should include.

4.1.2 Performance

There are three main solutions when it comes to drawing graphics on a web browser: Scalable Vector Graphics (SVG)¹, *Canvas*² and *WebGL*³. Since their creation, these graphic APIs have been constantly getting faster, more robust and with more features. Adding to that, the graphs that will be created from the current data set won't be big enough to the point where performance difference between these APIs is noticeable. With this in mind, the choice of framework will be based on other factors.

4.1.3 Key features

Not all frameworks are created equally, therefore, one must compare the most important features across all candidates and choose the one with the features that are needed for a certain task.

Documentation

Documentation is a mandatory thing to have when developing and launching a framework. Its quality is a major factor that will be taken into consideration on all frameworks.

Live demonstrations

When choosing a graphics framework, one will definitely find images that try to illustrate its capabilities, however, there's more to graphs than just the visualisation. Having the possibility of trying out an example of a graph that's being generated and rendered through a framework is very important. This way, the developer can have a feel of the smoothness of the physics (if they are implemented or active), as well as the overall performance of the rendering process.

Re-drawing

When drawing a graph with a moderate amount of data, graph readability will eventually be degraded because of the random nature that these framework's algorithms choose to draw the data. Either the framework has to be incredibly accurate while maintaining a small drawing overhead or it has to include some sort of mechanism that allows the user to command the framework to re-draw the graph.

Editing the graph

Being able to drag nodes across the 2D plane can also be a very powerful tool for the user. In a situation where the framework has drawn an almost perfectly readable graph, the user should have the possibility of changing the position of a few nodes. This way the user can obtain a graph with high quality positioning of its nodes and not have to re-draw the whole graph, risking getting a worse graph.

¹<https://svgjs.dev/docs/3.0/>

²https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API

³<https://get.webgl.org/>

Event listeners

Although graphs can explain a lot with just their own drawing capabilities, there can still be room to improve the experience. Let's imagine a graph where the nodes represent authors and the edges (connections between the nodes) represent publications that they have released together. The graph alone demonstrates the relations between the authors, but it would be even more powerful if there was a way to quickly find out more information about these nodes or edges. That is why it is also imperative that a framework allows the developer to expand its capabilities by accessing mouse events. This way, the user's interaction with the graph can trigger other visualization events, like an information box that goes into more detail about the data that the user is interacting with.

Exporting

One feature that can be specially useful at an institutional level is exporting the graph into an image or Portable Document Format (PDF). A screenshot can achieve the same result, but it can require cropping the image in order to exclude other UI elements. This feature is not as critical as the previous ones

Physics

While the algorithm chosen by the framework's developers affects the time it takes to draw the graph, physics affect the smoothness of performing operations on the graph itself. The lack of physics will provide the user a static looking graph, with straight edges, and no repulsive effect between the nodes, allowing for a more cluttered presentation. If the framework has this option and is enabled, then it must also include methods to tweak the physics's behavior and, consequently, its performance overhead.

4.1.4 Framework comparison

When attempting to find adequate frameworks, the search terms "graph theory" and "network graph", followed by "js framework" or "js library", produce the best results. The key features and their importance will be compared throughout a selection of the most popular frameworks and one of them will be chosen.

Cytoscape.js

*Cytoscape.js*⁴ is focused exclusively on graphs and stands as one of the most powerful frameworks, with a good documentation and many live demonstrations. Interactions and events are present, so are animations and physics. Exporting the graph is not supported natively, but it is possible to render an image in the same page through a "work around"⁵. Its time to setup is moderate, as it is possible to source it in a simple HTML environment. Additionally, an extensive list of well known companies that use this framework is listed on their website.

⁴<https://js.cytoscape.org/>

⁵<https://stackoverflow.com/questions/69304985/how-to-export-a-cytoscape-js-graph-to-image>

D3.js

*D3.js*⁶ contains an extensive list of visualization tools that aim to solve very different problems. Consequently, documentation is vast, even for each separate module. It refers to itself as a visualization “kernel” and not as a framework [14]. Although many live demonstrations are available, there’s only a few, or sometimes one, for each individual module, which makes it less accessible than other frameworks’ examples. Their graph visualizations are editable and interactive. Exporting is done the same way as the previous framework candidate. Setup time is considerable despite allowing for a simple HTML and JavaScript (JS) deployment.

Graphology and Sigma.js

*Sigma.js*⁷ is a graph rendering and interaction framework. It uses *WebGL*, as it aims to solve the problem of rendering a graph with thousands of nodes. Its backend consists of an entirely different library, *Graphology*⁸, that does not focus on rendering but, instead, focuses on the data models and the algorithmic part of graph theory. Both library’s documentation are concise and easily readable. The graphs can be customized, interacted with and can also be saved with the framework’s native support. Setup time is moderate, as one can observe from their live demonstrations on their main page.

Dracula.js

The *Dracula Graph Library*⁹ is a simple framework. Its documentation is based on a code example, encouraging the developers to try out the framework as they are learning it. They provide live demonstrations on their page, showcasing the ability to customize the look of the graph and its editing capabilities. The graph can be redrawn but there’s no native way of exporting it to an image or PDF. Its main limitation is the lack of event listeners, as it is not possible to extract information from the user’s interactions with the graph. Due to its limitations and simplicity, the setup is simple and quick and could be a good choice for developers who want to try out visualizing data using a graph structure.

vis.js

Finally, *vis.js*¹⁰ is a visualization library that does not focus on one type of tool. It can render graphs, 2D and Three-Dimensional (3D) charts, among others. A developer can learn this framework either by its good documentation or by its extensive list of examples. The examples page contains practically one example per functionality, meaning that it is possible to learn the framework by looking at code and resort to the documentation for the fine details. As those examples indicate, the time to setup is low, while allowing for good customization and interactivity. The heavily customizable physics allow for a better looking graph, as it

⁶<https://d3js.org/>

⁷<https://www.sigmajs.org/>

⁸<https://graphology.github.io/>

⁹<https://www.graphdracula.net/>

¹⁰<https://visjs.org/>

forces the nodes to be placed far from each other, and the edges curved towards the outside of the graph's center. Ultimately this means that the graph can contain a lot of information and not be cluttered, maintaining high readability. The customization allows for scaling both the nodes and edges, which will have an impact on how the user perceives the information. Because the graphs are rendered through the *Canvas* graphic API, exporting them as an image is as simple as right-clicking the background and saving it. The event listeners allow for the developer to know which node or edges are selected by the user, which can lead to other information be displayed.

Overall, *vis.js* does not have the best customization or available algorithms, but for this application, it can achieve its goal very quickly. This framework was chosen because all important features are implemented and the time period between learning the framework and producing a graph (with the data that was already collected) is very short.

4.2 CREATING GRAPHS

Having come to a decision on what framework to use to render a graph visualization on a web page, the steps required to transform data into a graph have to be listed. In a simple HTML environment, *vis.js* requires a set of nodes and edges to be fed into its main class. This will be achieved by rendering the HTML file with the data already prepared. The data that we need to produce the graph is fetched from the database with *Django's* query sets¹¹ and then serialized¹² into a dictionary, that is then rendered into the HTML with *Django's* templating capabilities.

From an institutional perspective, graph visualizations need to represent types of data that can't be represented in other ways. This means that they have to relate different entities in a way that is possible to extract unique and useful information. For this application, it makes sense to relate authors and publications, which is going to be achieved in two similar but still distinct ways.

4.2.1 Styling

vis.js allows the developer to customize the graph's characteristics, like the size of the nodes and the width of the edges. The appearance of the graph is crucial to the success of the interpretation of the graph's information. Figure 4.1 shows how a graph looks by default, with no styling applied. It is the first of many examples that can be found on *vis.js's* example page of their graph visualizations¹³.

No matter how well the graph is arranged, it can become overwhelming when there's a lot of edges crossing over each other. To solve this issue, edges were made slightly transparent, and their color different when being hovered or selected (with the cursor).

One of the main advantages of having nodes of different sizes is the ability to instantly tell which authors (in this case) have a bigger influence on what the graph is trying to present.

¹¹<https://docs.djangoproject.com/en/4.1/topics/db/queries/>

¹²transforming data into a format that can be stored or transmitted

¹³<https://visjs.github.io/vis-network/examples/>

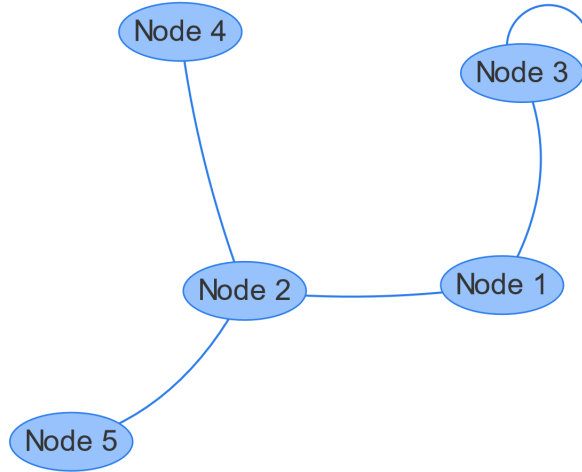


Figure 4.1: First example of *vis.js*'s example page.

The same effect occurs when different edges that connect these authors, have variable widths. It indicates that their relationships are different and contain different information.

Expression 4.1 represents the function used to scale both the nodes and edges. x represents the node that is being scaled, while N represents the total number of nodes. The value of the node can be the number of publications that an author has, for example. In summary, the size of the node depends on its value and the sum of all nodes' values.

$$\frac{value(x)}{\sum_{n=1}^N value(n)} \tag{4.1}$$

Ultimately, this means that the interpretation of the graph is made easier, by increasing the size of the most predominant elements. Additionally, *vis.js* allows the developer to place images inside the nodes, to make them even more distinguishable.

The pictures of these authors are directly linked to their *Ciência Vitae* profile, which is why only one is available in Figure 4.2. Other authors that are not present in that figure have their pictures available. Finally, the user also has the possibility of hovering their cursor over any object (node or edge) and get a small pop-up text with information about it (for example, the value of the object, i.e., number of publications). Figure 4.2 represents the styling that is going to be used on every graph, with one author selected and one edge that is being hovered.

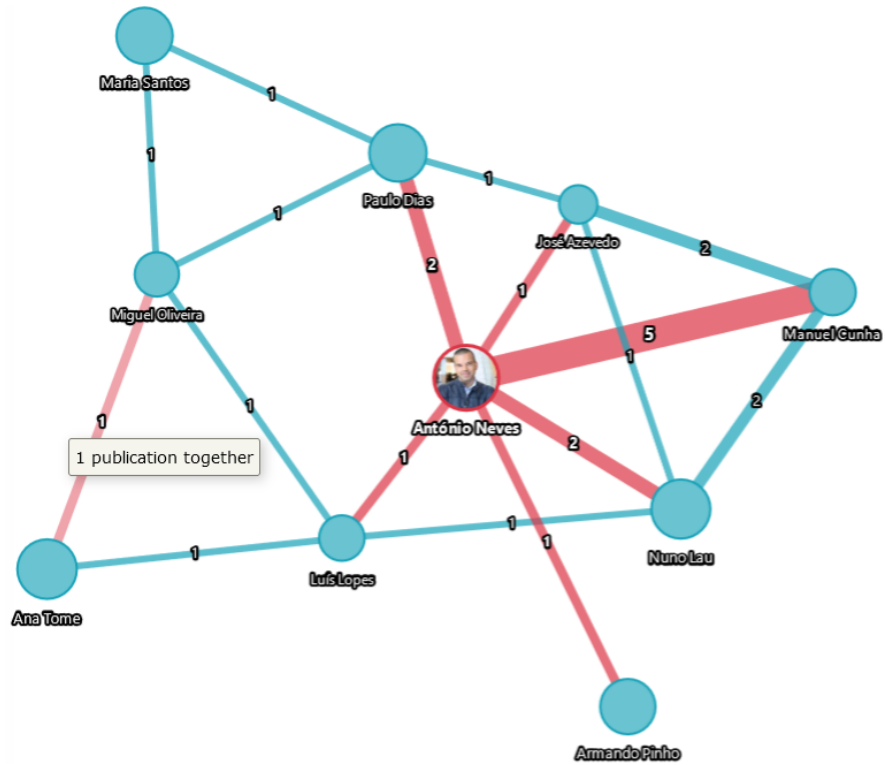


Figure 4.2: Customization of a graph visualization.

4.2.2 Author map

The first attempt at relating authors and publications comes in the shape of a collaboration map of a specific author. The main objective is to find the authors that someone has collaborated with, and the number of those collaborations. But this information alone can be represented in a table and still be easy to interpret, so in order to make the graph useful, the relations are expanded to the collaborators. This means that the collaborations between the author’s colleagues are also represented, giving the user a new layer of information to view.

Figure 4.3 represents a graph that was generated with real, collected data. This graph is oriented at a specific author, António Neves, and it represents his collaboration map of conference papers related with the keyword “vision”, between 1981 and 2020 (Note that 1981 is selected by default since it represents the date of the oldest publication in the database, and not from this author specifically). This graph indicates that the author published 5 conference papers with his colleague Manuel Cunha, which makes this the strongest relation in these specific conditions.

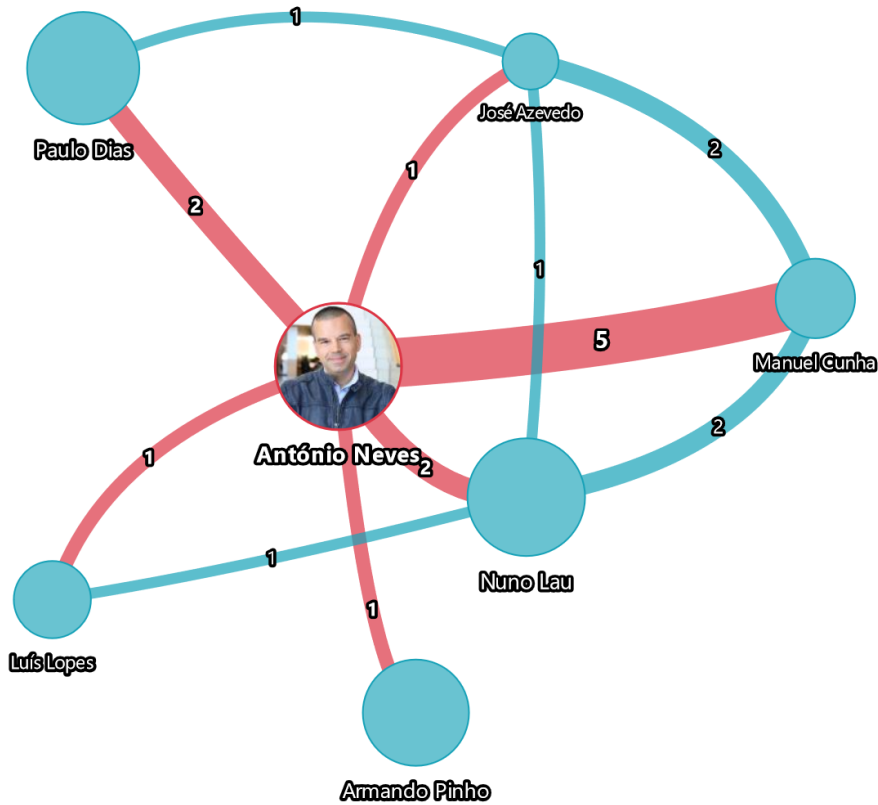


Figure 4.3: Author’s collaboration map, of conference papers related with the word “vision”, from 1981 to 2020.

4.2.3 Global map

The global map is an extension of the author map, to an institutional level. It is not built around any specific author, but instead, it evaluates all authors present in the database. This gives the viewer a global perspective about the scientific knowledge that the institute possesses.

Figure 4.4 represents a graph with the same parameters as Figure 4.3, but relative to the entire research institute. It is possible to observe that the graph in Figure 4.3 is included in this one. Both the authors and relations from the author graph that was presented before are shown, along with new information on other authors. The previous graph now appears as it expanded to the colleagues’ collaborations with other authors that did not collaborate with António Neves in these specific conditions. The other observation is that now it is possible to observe authors that have publications that match the criteria but didn’t collaborate with anyone else, making them isolated from the rest of the graph. There’s also a pair of authors that collaborated only with each other, on the left of the image.

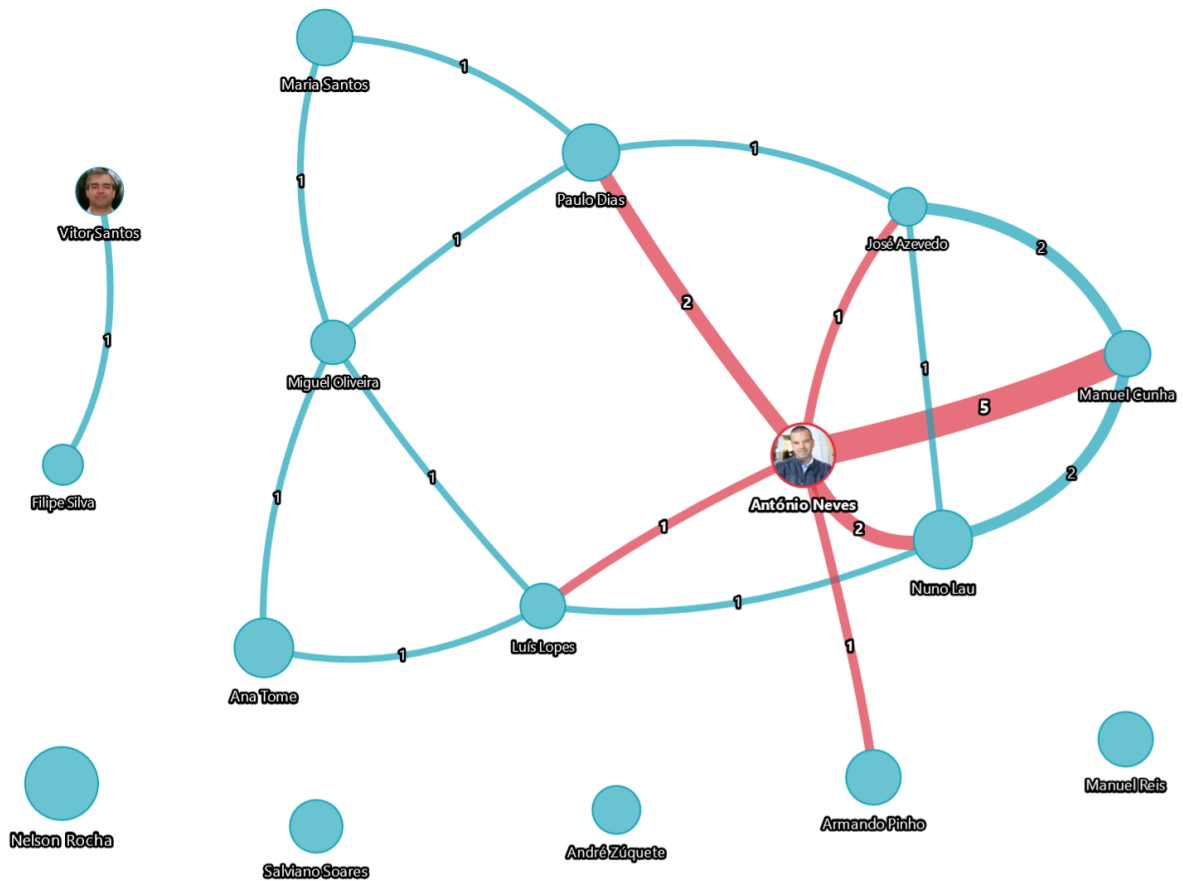


Figure 4.4: Institute’s collaboration map, of conference papers related with the word “vision”, from 1981 to 2020.

4.3 DATA VIEWER

Although the graph provides information in a unique way, it can only present the total number of publications. To complement this, another visualization tool is needed to inspect areas of the graph. This tool makes it possible to view which publications are shared between authors as well as which publications the author has published.

When the user selects a node or an edge, the viewer updates itself to show information relative to that same object. This means that when an author is selected, the viewer will show all its publications that match a given criteria, like specific types of publications, date limits, and keywords. It also means that when an edge is selected, the viewer shows the publications that both authors published together.

Figure 4.5 shows how the viewer looked like when the author António Neves was selected, on figures 4.3 and 4.4. In this specific state, it is showing 12 conference papers related with the word “vision” that the author published between 1981 and 2020.

The table presented in Figure 4.5 resembles the type of visualization that was presented many times throughout chapter 3. Each row contains, from left to right, its number, two icons that indicate from which API the publication has gotten its information from, the number of authors associated with the publication, number of keywords, areas, two icons that indicate if

#	APIs	Publication type	Authors	Keywords	Areas	Abstract	Full text	Date	Title
1	SC	Conference Paper	1	5	2	✓		2019-01-01	Focus Estimation in Academic Environments Using Computer Vision
2	SC	Conference Paper	1	6	2	✓		2018-01-01	Monitoring students' attention in a classroom through computer vision
3	SC	Conference Paper	1	8	2	✓		2017-01-01	Cooperative sensing for 3D ball positioning in the RoboCup middle size league
4	SC CV	Conference Paper	2	8	1	✓		2016-01-01	A ground truth vision system for robotic soccer
5	SC CV	Conference Paper	2	8	3	✓		2016-01-01	Image scanning techniques for speeded-up color object detection
6	SC CV	Conference Paper	3	8	3	✓		2015-05-04	Detection of aerial balls in robotic soccer using a mixture of color and depth information
7	SC	Conference Paper	2	8	2	✓		2014-01-01	UAVision: A modular time-constrained vision library for color-coded object detection

Figure 4.5: Example of a possible state of the publication viewer.

the publication contains an abstract or full-text content, and finally, the date and title.

Knowing just the number of keywords (for example) is not very useful, that is why there is the possibility of hovering the cursor above some of these fields to get more information. It is possible to do this with the authors, keywords, areas, and abstract fields.

4.4 FILTERS

In order to make the visualization experience more versatile, some filtering options need to be implemented. From a research institute manager's perspective, it does not make sense to analyse only global statistics. There will be many occasions where it will be necessary to gather the best researchers of a certain scientific area or analyse data from a specific data range.

4.4.1 Keywords

Given that the database already stores thousands of keywords, it makes sense to filter the resulting publications by this characteristic, however, it is infeasible to assign the task

of choosing the keywords to the user. It would result in having to display that amount of keywords to the user so they could select the ones that they wanted. Instead, the user is presented with just a search form. This way, the system takes responsibility in finding the correct keywords to be used in filtering. This process consists in iterating every keyword stored in the database and comparing its processed name (section 3.4.8 explains this process) with the processed version of the search query. This will generate a list of keywords that are deemed similar to the search query, which will be used to filter out publications that the user doesn't want to be included in the graph.

Figure 4.6 illustrates the list of keywords that were chosen by the system, after the user's "robotic soccer" input query. This process allows for some keywords to "slip through" and be considered similar when they shouldn't be. This ensures that the user receives a list that includes all keywords that are related at the cost of also receiving some "false positives". That is why it is also possible to tweak this list, by individually disabling the keywords that should not have been included in the first place. Naturally, the user also has the ability to update the graph with the new version of the list, now with less but more accurate data. Additionally, the user can update list of keywords either by resetting it through the "Reset keywords" button or by entering a new search query.

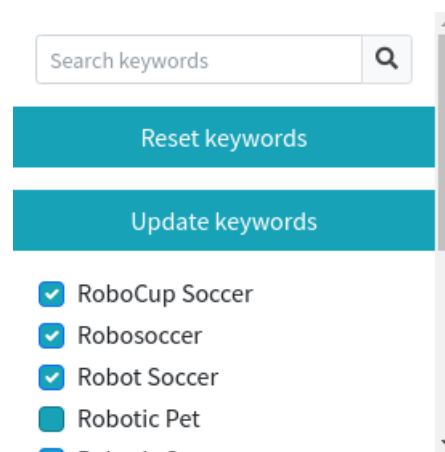


Figure 4.6: Example of a possible state of the keyword form, when the user searches for "robotic soccer" related keywords. Some choices are disabled by the user.

These filters are stored within the user's session, meaning that as long as the user's browser supports cookies and they don't expire, the user can keep coming back to the same page and have the same filters still applied. Cookies alone do not store the information about the filters, but instead, they point to a location on the application's database that actually contains the data. The developer is given the task to choose which data to save in a user's session, but the actual handling of this data and the user's cookies are handled by *Django's* backend.

4.4.2 Date

Limiting the publications to a date interval is crucial to the research analysis process. This range is implemented by fetching the oldest publication stored in the database, and setting that value as the minimum. The current year is set as the maximum value of the form, as

opposed to the most recent publication's year. In the vast majority of time, this will not make a difference but it saves the database from doing an additional search. Additionally, the UI prevents the user from inverting the range. This means that the "Start year" can take values from the minimum value all the way up to the value that is currently selected on the "End year". Naturally, the "End year" ranges from the value that is currently selected on the "Start year", and goes up to the current year.

Figure 4.7 illustrates what happens to the "End year" range, when the user selects the year 2020 as the "Start year" for publications to be fetched. Given that value as the starting year, the only range that makes sense for the ending year is 2020 through 2022 (the current year of writing).

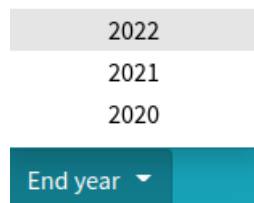


Figure 4.7: Range of options of the "End year" field, when "Start year" has the value "2020" selected.

4.4.3 Publication type

Finally, it is also possible to filter publications by their type. This means that the search can be limited to just conference papers, for example. This is a very useful feature to implement, as it gives the ability to not only search competences by scientific fields but also by the type of those competences.

As Figure 4.8 illustrates, there are only three individual types available to choose. This is a consequence of having many types that sometimes are specific to each of the APIs. Articles, books and conference papers represent the most common types of publications in the database, that is why they are included as choices. The first choice ("All") will fetch every type of publication, while the last choice ("Other") will fetch any publication that is not included in the three previous types.



Figure 4.8: Options available in the publication type form.

4.4.4 Headers

In order to give the user, a better understanding of what filters are being applied, two headers were placed above the graph and viewer. These headers complement each other, as they tell the user exactly what information is being displayed.

Figures 4.9 and 4.10 represent a graph header and a viewer header, respectively. They both reflect information that would be observable when looking at a graph from figures 4.4 or 4.3. The graph header indicates that the user is looking at a collaboration map that was filtered with the search query “vision” and limited to a range of dates from 1981 to 2020. The viewer header reflects what the user is interacting with. It tells the user how many and what types of publications are being shown by which author (or pair of authors).



Collaboration map, related with "vision", from 1981 to 2020

Figure 4.9: Example of a graph header.



12 Conference Papers from António José Ribeiro Neves

Figure 4.10: Example of a viewer header.

4.5 INSTITUTIONAL STATISTICS

One of the main advantages of storing data from an API is that there is no performance set back from having to execute many accesses. This allows for extracting statistics and crafting new information about the data set, in the least amount of time possible.

Bar charts are one of the most common representations of any kinds of statistics, and *chart.js*¹⁴ is one of the most popular JS frameworks to render that type of information. The graph data is processed by fetching it from the database, transforming it into the required data structure and rendering it directly in the HTML template.

4.5.1 Publications in a time period

The first of the bar chart representations can show any type of publication in a date interval. It can be applied to the entire research institute or a single author.

Figure 4.11 shows an example of a bar chart that is showing the number of articles from 2012 and 2021. The accepted choices for the publication type and date range are the same as the graph visualization (section 4.4).

¹⁴<https://www.chartjs.org/>

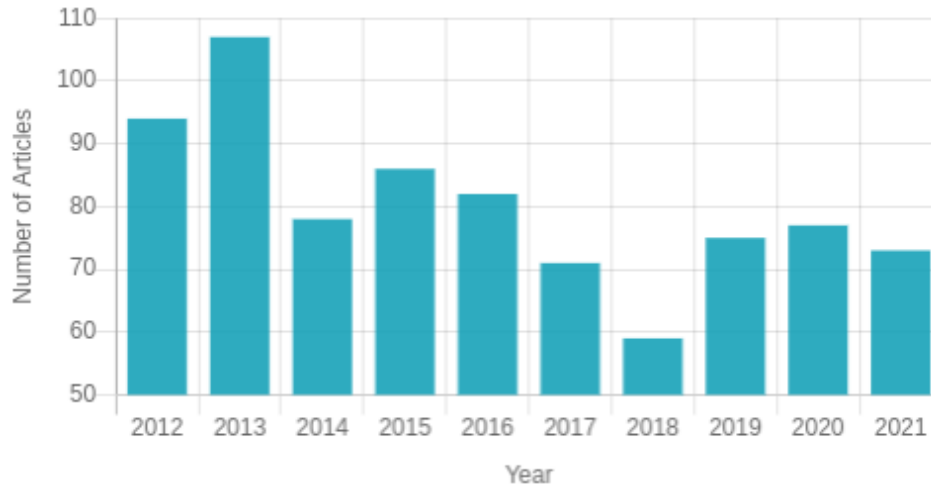


Figure 4.11: Example of a bar chart about publications.

4.5.2 Projects in a time period

Similar to the publications graph, the project bar chart can present the amount of projects relative to the entire institute or an author. It accepts a date range, that follows the same rules as the previous type of bar chart, as well as the graph visualization.

Figure 4.12 shows all projects that were collected from the *Ciência Vitae* API. Just like in the previous graphs, the minimum value for the date range is taken from the oldest project in the database.

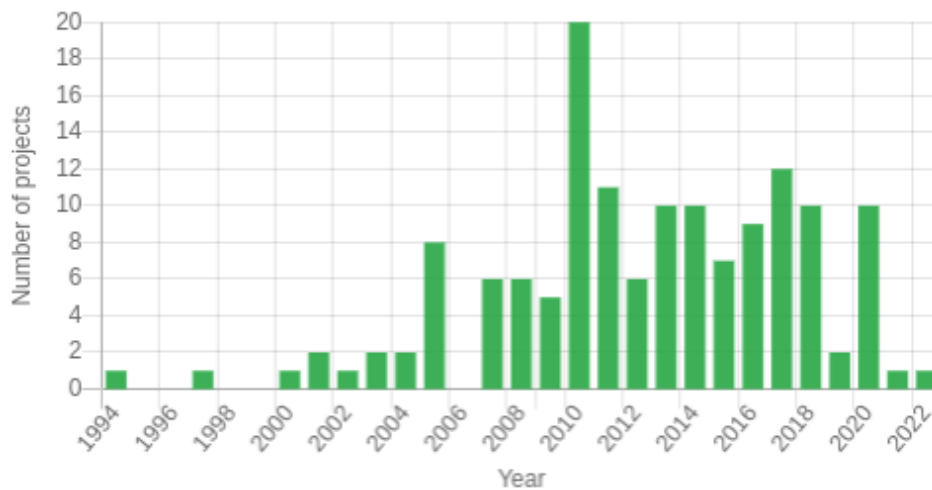


Figure 4.12: Example of a bar chart about projects.

4.5.3 Global counters

Another statistic that can be applied to both the author and institute is a global counter. When applied to publications, it can indicate a clear growth of the institute’s research footprint. Additionally, when applied to keywords and scientific areas, it means that the institute can be diverse and contain different types of scientific competences. Finally, it can be applied to

authors and help build a classification of the most active authors in the institute, with their number of publications in a given date interval.

Figure 4.13 shows a card representation of the application’s global counters in the landing page. It starts with the total number of publications in the database, followed by projects, keywords and scientific areas.

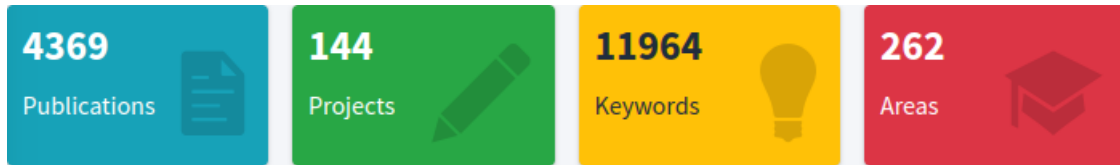


Figure 4.13: Cards with information about the application’s global counters.

4.5.4 Author specific statistics

These set of statistics are tailored specifically to an author’s personal information and can tell the user exactly what scientific domains they operate best in. It is possible to calculate which colleagues have the most publications in common with a given author, indicating that they are good collaborators. Another powerful tool is the ability to know the keywords and areas with the most occurrences throughout the author’s list of publications and projects.

Figure 4.14 shows an example of the most common keywords of a given author. These keywords are obtained by iterating every publication from that author and creating a dictionary of keywords with their respective occurrence count.

Most common keywords		
#	Keyword	Count
1	Robotic Soccer	9
2	Lossless Compression	9
3	JPEG LS	7
4	Lossless Image Compression	7
5	Robotic Vision	6

Figure 4.14: Example of an author’s most recurring keywords.

Similar to the keywords example, Figure 4.15 can show the most accurate areas of domain of a given author. This data is available from all APIs’ publications and from *Ciência Vitae* projects as well.

As a complement to the author’s map, Figure 4.16 can list the author’s top colleagues. Similarly to the previous examples, this data can be fetched from the author’s publications and projects.

Most common areas			
#	Area	From publications	From projects
1	Computer Science (all)	26	0
2	Electrical and Electronic Engineering	22	2
3	Theoretical Computer Science	21	0
4	Signal Processing	15	0
5	Computer Vision and Pattern Recognition	15	0

Figure 4.15: Example of an author’s most recurring scientific areas.

Top collaborators			
#	Name	From publications	From projects
1	Armando J Pinho	33	2
2	Manuel Bernardo Salvador Cunha	15	0
3	Nuno Lau	14	0
4	José Luís Costa Pinto De Azevedo	12	1
5	António Joaquim Silva Teixeira	7	0

Figure 4.16: Example of an author’s most common collaborators.

4.6 HOME PAGE

Figure 4.17 shows an overview of the application’s home page. It contains the knowledge map and data viewer side by side, filling the entire screen when first loaded up. Upon scrolling down, the user can find cards that report the global counters of the system, like publications, projects, keywords and areas. Below that there are two bar charts for publications and projects. Finally there is a list of all authors sorted by their number of publications. This list also indicates which publications have an available abstract and full-text content.

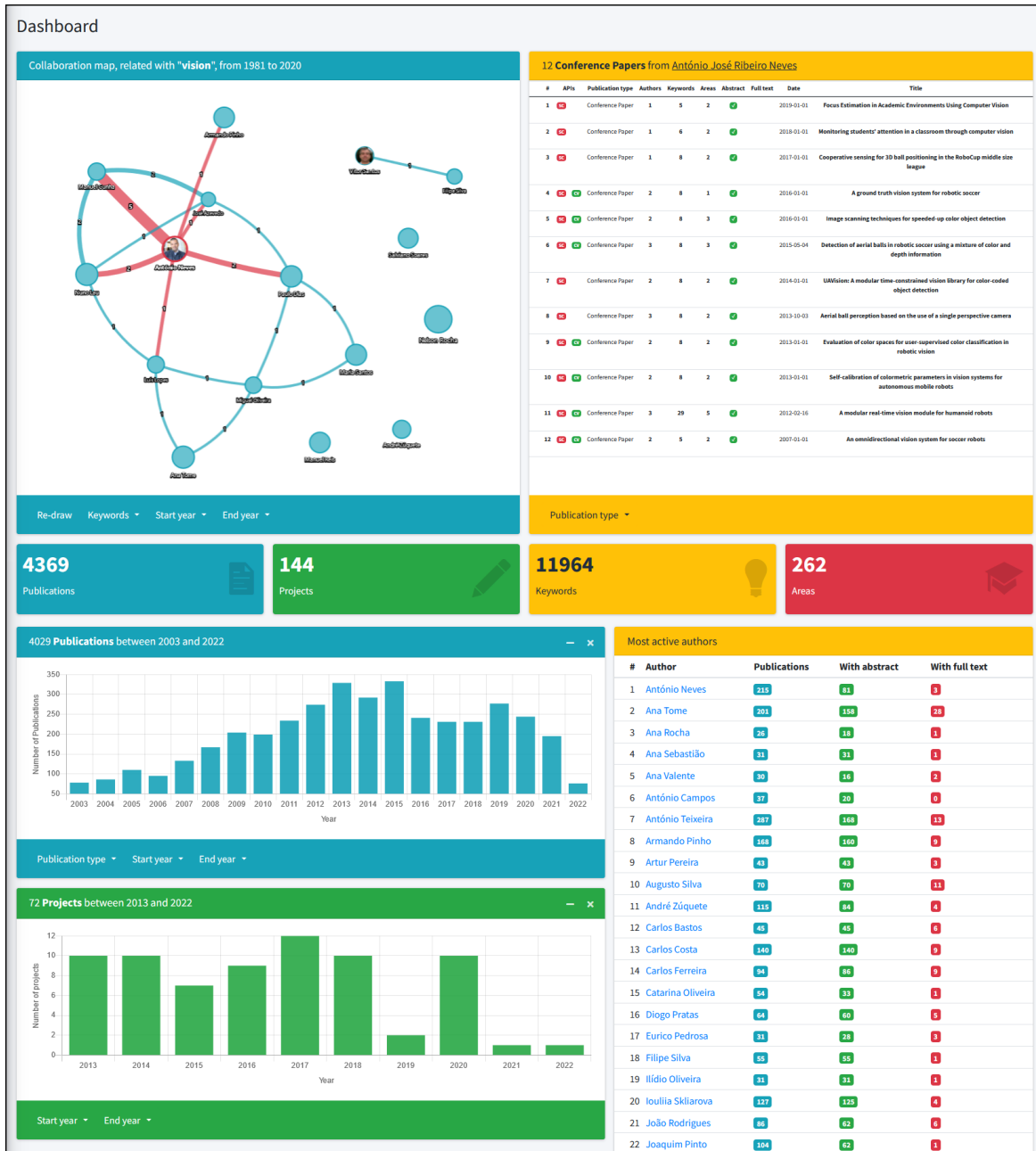


Figure 4.17: Home page overview.

4.7 AUTHOR PAGE

Similar to the home page, the author page (Figure 4.18) contains a knowledge map and data viewer that are focused specifically on one author. Given the nature of this graph, the author should be rendered at the center of the graph, most of the times. Scrolling down reveals the same cards and bar charts but this time they are also tailored to the author's profile. On the right of the bar charts there are tables that show the author's top keywords and areas. Finally, at the bottom, there is a table for showing the author's top collaborators.

António José Ribeiro Neves

Collaboration map, from 2011 to 2020

Re-draw Keywords ▾ Start year ▾ End year ▾

132 Publications from António José Ribeiro Neves

#	APIs	Publication type	Authors	Keywords	Areas	Abstract	Full text	Date	Title
1	✔	Conference Paper	1					2020-01-01	Computational Analysis of Nonverbal Communication Cues in Group Settings
2	✘	Conference Paper	2	8	5	✔		2020-01-01	Image selection based on low level properties for lifelong moment retrieval
3	✔	Conference Paper	1					2020-01-01	Lifelong Moment Retrieval Web Application
4	✔	Conference Paper	1					2020-01-01	Mood Estimation Based on Facial Expressions and Postures
5	✔	Conference Paper	1	8	5	✔		2020-01-01	The Impact of pre-processing algorithms in facial expression recognition
6	✔	Conference Paper	2	10	1	✔		2020-01-01	UA-PT Bioinformatics at ImagCLEF 2020: Lifelong Moment Retrieval Web based Tool
7	✔	Conference Paper	2	4	2	✔		2020-01-01	Understanding public speakers' performance: First contributions to support a computational approach
8	✔	Article	1					2019-11-02	Facial Expression Recognition Using Computer Vision: A Systematic Review
9	✘	Review	1	6		✔		2019-11-01	Facial expression recognition using computer vision: A systematic review
10	✔	Conference Paper	1	8	2	✔		2019-01-01	A Personal Robot as an Improvement to the Customers' In-store Experience
11	✔	Book	1					2019-01-01	A Personal Robot as an Improvement to the Customers' In-store Experience
12	✔	Online Resource	1					2019-01-01	COMPLEMENTOS SOBRE LINGUAGENS DE PROGRAMAÇÃO, slides de apoio e guião de exercícios para a unidade curricular

Publication type ▾

215

Publications

25

Projects

438

Keywords

39

Areas

41 Conference Papers between 2013 and 2022

Publication type ▾ Start year ▾ End year ▾

Most common keywords

#	Keyword	Count
1	Robotic Soccer	9
2	Lossless Compression	9
3	JPEG LS	7
4	Lossless Image Compression	7
5	Robotic Vision	6
6	Microarray Images	6
7	Color Quantized Images	6
8	Vision Systems	5
9	Robotic Soccer Team	5
10	Arithmetic Coding	5

25 Projects between 2004 and 2022

Start year ▾ End year ▾

Most common areas

#	Area	From publications	From projects
1	Computer Science (all)	26	0
2	Electrical and Electronic Engineering	22	0
3	Theoretical Computer Science	21	0
4	Signal Processing	15	0
5	Computer Vision and Pattern Recognition	15	0
6	Computer Science Applications	14	0
7	Engineering and Technology - Electrotechnical Engineering, Electronics and Informatics	0	11
8	Software	10	0
9	Applied Mathematics	8	0
10	Biomedical Engineering	8	0

Top collaborators

#	Name	From publications	From projects
1	Armando J Pinho	33	2
2	Manuel Bernardo Salvador Cunha	15	0
3	Nuno Lau	14	0

Figure 4.18: Author page overview.

Optimization

One of the main advantages of using Django is that it allows the implementation of a web application in a very short period of time. For an inexperienced developer, this could come at the cost of performance because any task can be achieved with simple code. Performance can be drastically improved, specially when dealing with large amounts of data. This chapter will explore guidelines that seek to optimize code and will present differences in performance when applying optimizations. Every testing result presented in this chapter was obtained by averaging five taken measurements.

5.1 QUERYSETS

Django's *QuerySets* is a database abstraction API that allows the developer to create, retrieve, update and delete objects [15]. It facilitates the execution of clean queries and stack operations, by replacing query languages with simpler *Python* methods. In practice, a *QuerySet* can also represent a list of objects that are retrieved from the database.

One of its main characteristics is the minimal use of the database when manipulating data. When building a *QuerySet*, an empty cache is associated with it. By the time it is evaluated (could be in a “for” loop, “if” condition, etc.), the results are saved in that same cache. If the *QuerySet* happens to be evaluated again, there will not be the necessity of accessing the database, because the results were already fetched [16]. Code block 1 shows the “laziness” of *QuerySets* by demonstrating that it is possible to create a query in separate steps with only one execution.

```
1 # Select all people named 'Antônio' that don't have the surname 'Neves'
2 q = Author.objects.all()
3 q = q.filter(name__startswith='Antônio')
4 q = q.exclude(name__endswith='Neves')
5
6 # QuerySet evaluation (Only one query is executed)
7 print(q)
```

Code block 1: Python code for creating a QuerySet in separate steps.

5.2 PREFETCHING DATA

To understand the prefetching process, we need to understand that in the current data model (Data Model Diagram appendix) there are many nested relations. For example, many authors publish many publications with many keywords. This means that the amount of relations increases significantly when fetching everything from the data set in a way that is legible and understandable.

In *Django*, there are tools that allow data to be prefetch, however, it is entirely up to the developer to decide which data fields need to be prefetched. Mistakes can lead to excessively long and difficult queries, therefore, it is a process that requires manual analysis of the code's database requirements [17].

5.3 DJANGO DEBUG TOOLBAR

The *Django Debug Toolbar*¹ is a powerful tool used to get performance indicators about an application. It can monitor many aspects like database queries, cache management, time measurements and much more. According to the *Django Developers Survey 2021*², it is the third most popular third-party package and stands as the best debugging package. Because of our heavy visualization necessities, this application has to perform large queries to the database. This toolbar will help us understand where the application can be optimized.

This toolbar consists of an HTML side panel with multiple expandable sections. Each section contains information about a distinct aspect of the page execution and upon selection, the entire page is replaced by a much more detailed view of that same aspect [18].

5.3.1 Query awareness

While *Django* can make it easy to build an application by creating an abstraction layer of the database, it can also lead to heavily inefficient code. Retrieving a set of objects can be initially perceived as a simple operation, but when these objects contain “ForeignKey” or “ManyToMany” relations, the evaluation of the initial database query will expand into many other queries. The *Django Debug Toolbar* will allow us to know exactly which queries were executed, what code generated each query and for how long they ran [19].

¹<https://django-debug-toolbar.readthedocs.io/en/latest/>

²<https://lp.jetbrains.com/django-developer-survey-2021-486/>

5.4 DEVELOPER PAGE

Throughout the developing phase, this page’s data would grow to an incredible size and make it slower and slower. The need for optimization was evident, so that process was executed way before having collected all the data from the APIs.

After the data collection phase, 4369 publications were being displayed in this page, however, this does not accurately reflect the amount of data that is being presented. 50 authors are associated with a variable fraction of those publications. More precisely, there are 5615 distinct relations between authors and publications. Additionally, there are 20079 relations between publications and keywords, and 7360 between publications and areas. This means that, for visualization purposes, many publications will appear multiple times but under different authors, which also applies to keywords and areas.

If we think of an ideal scenario, these publications and keywords would only have to be fetched once from the database and displayed how many times it is necessary. This is not the case when applying basic knowledge from the *Django* framework. Like it was explained before, *Django* can make it very easy and fast to develop a web application but at the cost of some performance “shortcuts”. Luckily it provides good tools to tackle this optimization problem, like the prefetching (section 5.2) mechanism.

5.4.1 Optimizing

This process has to start with using the *Django Debug Toolbar* to get information about the queries that are being executed. Unfortunately, the amount of data requested by the page made this tool too slow to work with. Due to the added overhead that this tool introduces in order to execute measurements, this page took around twenty minutes to load, so instead, only the five authors with the least amount of publications were used for demonstrating the optimization process. These five authors generated 1213 queries, which is impressive even comparing to the 49740 generated by the entire list. Out of the 1213 queries, the tool deemed that 1210 shared the same SQL code with another query, but with potentially different parameters, while 1113 shared the exact same code with another query. This information tells us that the page is fetching the same items multiple times from the database, which is far from ideal.

With the number of relations in mind, code blocks 2 and 3 show how the fetching is done. Both statements from code 2 create a *QuerySet* which, at this point, does not have any impact on the database because it has not been evaluated. Code 3 demonstrates that the *QuerySet* is being evaluated in nested for-loops. The actual code fetches all fields from these objects, not just their name, so the number of unnecessary queries is much bigger.

The *Django Debug Toolbar* revealed that the biggest set of similar queries contained 393 of them. This is due to the amount of times that the author set of a given publication is required to show information on the page. Showing the author set of a publication is important because it indicates which authors are associated with it. Code block 4 shows three examples of code that fetches data related to a publication’s author set. The first example writes the number of

```

1 # Fetches all authors
2 all_authors = Author.objects.all()
3
4 # Fetches 5 authors with the least amount of publications
5 authors = Author.objects.annotate(
6     num_publications=Count('publications')
7 ).order_by('num_publications')[:5]

```

Code block 2: Python code for fetching authors from the database.

```

1 {% for author in authors %}
2     <p>{{ author.name }}</p>
3
4     {% for publication in author.publications.all %}
5         <p>{{ publication.title }}</p>
6
7         {% for keyword in publication.keywords.all %}
8             <p>{{ keyword.name }}</p>
9
10        {% endfor %}
11    {% endfor %}
12 {% endfor %}

```

Code block 3: Django template for listing all keywords of all authors' publications.

authors that are associated with the publication. The second example uses that same number in a conditional statement. Finally, the third example iterates every author in that author set.

To avoid executing unnecessary database queries, this specific field needs to be prefetched, which will create a big query that fetches all authors from every publication of a given author. Code block 5 shows the modification that needs to be done to code block 2 in order to prefetch this specific field. With this optimization in place, the page executed 817 queries, significantly improving the performance. This optimization process repeats while there are still sets of queries that are similar or equal.

```

1 {{publication.author_set.all|length}}
2 ...
3 {% if publication.author_set.all|length != 1 %}
4 ...
5 {% for author in publication.author_set.all %}
6 ...

```

Code block 4: Different ways to evaluate a publication's author set in a Django template.

```

1 # Prefetches the 'author_set' field from the author's publications
2 authors = Author.objects.prefetch_related(
3     'publications__author_set'
4 ).annotate(
5     num_publications=Count('publications')
6 ).order_by('num_publications')[:5]

```

Code block 5: Python code for prefetching a field.

5.4.2 Optimization results

Section 5.4.1 focused on optimizing the developer page with just five authors because of the debugging tool’s added overhead. This section’s Table 5.1 presents the optimization results when the optimization process is applied for the entire data set. When applying no optimizations to the page, it executes 49740 queries and takes 25.253 seconds to load completely. Upon optimizing the publications’ author set, the query count drops to 32896 and the page time to around 17 seconds. This indicates that the author sets were generating 16844 unnecessary queries. Following that, the relation between publications and areas were the second most impactful. Doing this optimization on top of the already existing one, drops the query count to 19356 (difference of 13540) and makes the page load in around 12 seconds. At the end of all optimizations, the page only requires 13 total queries and loads in 4 seconds.

Optimization	Query count	Query weight	Page time (seconds)
No optimization	49740	N/A	25.253
Publication author set	32896	16844	17.023
Publication areas	19356	13540	12.015
Publication keywords	6193	13163	6.493
Publication types	579	5614	4.347
Project areas	258	321	4.162
Previous affiliations	111	147	4.058
Current affiliations	62	49	4.040
Domains	13	49	4.015

Table 5.1: Developer page’s iterative gains in optimization.

5.5 HOME PAGE

The home page renders data that is relevant to the entire organization. It contains a knowledge map, data viewer, global counters, a bar chart for publications, another for projects, and a list of all authors. Each of these elements can affect the page’s loading time, depending on the amount of filters that the user chooses to apply.

To understand the performance difference that a given element on the page has, the adequate filters have to be applied to the other elements so that they do the least amount of work. For the knowledge map, it is best not to choose any keyword and filter through a date range that won’t produce any results. This way, no publications will be fetched and processed into the graph, and consequently the viewer as well. For the publications and projects bar charts, it is best to set them to a date range that won’t produce any results. The other statistics have no parameters, so they can not be changed.

With all these elements producing zero results, the page takes 2.462 seconds to load. This is because the fixed statistics still have to do some compute to produce their values. The baseline (best case) for the following tests is set. The next step is to find the worst case for each of the elements on the page.

5.5.1 Knowledge map

The home page uses a non-discriminatory algorithm to choose what publications are going to be inserted into the knowledge map. In contrast to the map that is presented in the author page, this one does not have to check if one author collaborates with a colleague to decide if that person can be added to the graph. It is, therefore, much easier to implement and much faster when applying filters, however, if no filters are applied, it has to load every relation between publications and authors into the graph, which will represent the worst case.

Table 5.2 shows the performance impact that different workloads can have on this page’s knowledge map. The first workload consists in setting the map to a date range with zero publications (best case). The second workload applies the same filters as section 4.2.3. The final workload applies no filters and, as a result, great amounts of data are loaded into the page.

Workload	Page time (seconds)
No results	2.462
Filters	2.883
No filters	38.945

Table 5.2: Different workloads applied to the home page’s knowledge map.

With the knowledge acquired in section 5.4.1, this algorithm takes advantage of having many filters. This is because the algorithm does not need to exclude certain authors from the graph (unlike the author’s page) and can apply these filters directly into the *QuerySet*.

5.5.2 Bar charts

For the publications bar chart, not specifying the publication type and setting the date range to its maximum value, will produce the maximum amount of results. Specifying a publication type should improve performance as that constraint can be inserted into the *QuerySet*, however, as Table 5.3 shows, the performance margins are slim enough to the point where the computer’s instability could easily skew these results. The first workload represents the baseline for all tests. The second represents the entire date range but filtered for articles only. The last workflow represents the entire date range with no publication type restriction.

Workload	Page time (seconds)
No results	2.462
Filter	2.689
No filter	2.797

Table 5.3: Different workloads applied to the home page’s publications bar chart.

The projects bar chart only has the option to change the date range. Table 5.4 shows the difference between setting the date range to a value that produces zero results and setting it to its maximum range, fetching all projects. Fetching all projects does not impact the page’s load time significantly because there are not many of them present in the database.

Workload	Page time (seconds)
No results	2.462
Worst case	2.563

Table 5.4: Different workloads applied to the home page’s projects bar chart.

5.6 AUTHOR PAGE

The author page renders data that is only relevant to a specific author. Like the home page, it contains a knowledge map, data viewer, global counters, a bar chart for publications, another for projects. It also contains lists for the author’s top keywords, areas and colleagues, none of which have parameters that can be changed by the user.

Similar to the home page, a baseline loading time was measured. With all interactive elements modified to their fastest behavior, the page took 2.565 seconds to load. When comparing to the home page, one can assume that the exclusive elements belonging to this page are marginally heavier to compute than the author list in the home page.

5.6.1 Knowledge map

In contrast to the home page, this knowledge map needs to represent data with much stricter criteria. It starts by analysing colleagues that collaborated directly with the author in question and then analyses the relations between the “accepted” colleagues. The fact that the same process has to be done twice, along with having to verify if a colleague can be added to the graph, makes this graph much more difficult to obtain.

Table 5.5 illustrates three workloads applied to the knowledge map. The first workload represents the page’s baseline, when the map is limited to a date range that produces zero results. The second workload applies the same filters as section 5.5.1. The final workload applies no filters, resulting in all publications related to the author and its collaborators are loaded. The efficiency of this algorithm’s graph is further reduced when applying filters because instead of being applied to the *QuerySet*, they have to be verified every time a publication meets the correct criteria for being added to the graph.

Work load	Page time (seconds)
No results	2.565
Filters	25.121
No filters	13.379

Table 5.5: Different workloads applied to the author page’s knowledge map.

Waiting for a new graph can be accepted when its parameters are changed. After all, the algorithm can only re-run itself with the changes added by the user. However, when users perform an action on another page element, they should not have to wait for the system to calculate the same graph again. The lack of memory in this sense represents a great penalty in overall performance and user experience. Similar to “Autosubmit GUI”’s implementation [20], the graph’s state is saved in the user’s session to avoid this overhead.

5.6.2 Bar charts

Similarly to the home page, the bar charts in this page create a negligible impact on the load time of the page. In this page, the effect is not measurable at all because these bar charts have to process much less data (one author instead of the entire institute). Therefore it is safe to assume that the load time for this page stays at around the baseline value of 2.565 seconds.

Conclusion and Future Work

This is the final chapter of this document. Conclusions will be drawn here, as well as future work ideas.

6.1 CONCLUSION

This dissertation focused on the development of a knowledge management platform to be used within the IEETA in UA. In its latest state, the platform allows for fetching a researcher's track record from multiple sources. Searching and filtering is also possible through the aid of a knowledge map in the form of a graph visualization. Overall, the platform represents a unique way to enhance the management of scientific competences in an organization.

The data collection phase ended up being one of the most important parts of the dissertation. It showed that it is possible to extract rich information from public APIs, respecting their limitations. Redundancy, or duplicate checking also proved to be a challenge that can be tackled in many ways. This solution proposed the use of NLP techniques to find similar text across many publications and deem them equal or not. The phase ended with the collection of information for every author in the institute, and thoroughly examined the flow of information. This analysis helped realize that the duplicate checking process is crucial when importing data from different sources. The data model, which is explained both in chapter 3 and the Data Model Diagram appendix, also shows that it is possible to gather many different types of information, increasing the accuracy and richness of the system.

The knowledge extraction phase went through different visualization tools and their characteristics. Upon settling on one framework, it showed that it is possible to represent raw data in an enhanced visual representation. The data that was gathered was transformed into tables and charts, but the knowledge maps really stood out as being a powerful in interpretation and perception of the current assets of the organization. Another tool, named "Data viewer", was developed to improve the user experience when interacting with the knowledge map. When put together, these visual tools transform the platform into a management "highway", making that process much faster and efficient.

The optimization phase started along side the data collection phase and stretched until the end of the implementation. With every new feature added, optimizations had to be done to keep the platform usable. Chapter 5 was dedicated to explaining the concepts behind the optimization process that was employed throughout all development phases. Additionally, it showed that the optimization gains are very significant when dealing with great amounts of data. It also showed the performance weight of different workloads being applied to the platform's visual tools.

Extending the data collection to other sources could be easily done with the robust code that was built. Moreover, the visualization framework that was chosen, meets this platform's requirements, however, a more complex framework could have been used to make the platform ready for completely new features. The optimization phase ended up being extremely interesting to explore, but its use on the platform is somewhat limited. The developer page benefits greatly from that process, but the other pages not so much. There are limitations that can not be controlled, like the overhead of the visualization frameworks' rendering pipelines, but when ignoring that aspect, the process of fetching the data from the database to create the knowledge maps could be better on the author page. After reflecting on features that were implemented but also on the shortcomings, the platform proved that it can become a key component for the administration of any academic institute.

Finally, a repository was created and is available¹ with instructions on how to deploy this project. It was made public, since we believe that this contribution should be free for any research organization to use.

6.2 FUTURE WORK

One of the most important characteristics of bibliometrics and knowledge management, is citation data. Future features that could be added would be the ability to store citation information and enhance the existing tools with that added data.

When a publication is imported, there may or may not be a set of keywords associated with it. The system could attempt to extract keywords from the publication's title, abstract and full-text content, making the keyword list more accurate.

In its current state, this platform can run in any computer with *Python* and *Django* installed, however, it would be much easier to deploy if it was included in a *Docker* container, for example. That would help institutions get their internal platform up and running much quicker, making it much more appealing.

¹<https://github.com/joaogenio/mestrado-knowledge-maps>

References

- [1] D. Y. Kemp, “Knowledge management in a research & development environment-the integration of company culture and technology,” M.S. thesis, Rochester Institute of Technology, 2004, pp. 7–8. [Online]. Available: <https://scholarworks.rit.edu/theses/7689> (visited on Oct. 16, 2022).
- [2] M. Thelwall, “Bibliometrics to webometrics,” *Journal of Information Science*, vol. 34, no. 4, pp. 605–621, 2008. DOI: 10.1177/01655515070872.
- [3] B. Godin, “On the origins of bibliometrics,” *Scientometrics*, vol. 68, no. 1, pp. 109–133, 2006. DOI: 10.1007/s11192-006-0086-0.
- [4] J. E. Hirsch, “An index to quantify an individual’s scientific research output,” *Proceedings of the National academy of Sciences*, vol. 102, no. 46, pp. 741–754, 2005. DOI: 10.1073/pnas.0507655102.
- [5] J. C. Nesbit and O. O. Adesope, “Learning with concept and knowledge maps: A meta-analysis,” *Review of educational research*, vol. 76, no. 3, pp. 413–448, 2006. DOI: 10.3102/00346543076003413.
- [6] F. Silva, “Authenticus—enabling the identification and validation of portuguese scientific publications,” 2013. [Online]. Available: <http://hdl.handle.net/11366/67> (visited on Oct. 16, 2022).
- [7] F. A. Domingues, “Authenticus: Architecture and mechanisms to support a national repository of scientific publications,” M.S. thesis, Faculty of Sciences of the University of Porto, 2015. [Online]. Available: <https://repositorio-aberto.up.pt/handle/10216/83466> (visited on Oct. 16, 2022).
- [8] P. Kraker, C. Kittel, and A. Enkhbayar, “Open knowledge maps: Creating a visual interface to the world’s scientific knowledge based on natural language processing,” *027.7 Zeitschrift für Bibliothekskultur*, vol. 4, pp. 98–103, Nov. 2016. DOI: 10.12685/027.7-4-2-157.
- [9] “Interactive Scopus APIs.” (2022), [Online]. Available: <https://dev.elsevier.com/scopus.html> (visited on Oct. 16, 2022).
- [10] B. Chen, H.-P. Hsu, and Y.-L. Huang, “Bringing desktop applications to the web,” *IT Professional*, vol. 18, no. 1, pp. 34–40, 2016. DOI: 10.1109/MITP.2016.15.
- [11] D. Ghimire, “Comparative study on python web frameworks: Flask and django,” Bachelor’s Thesis, 2020, pp. 22–32. [Online]. Available: <https://urn.fi/URN:NBN:fi:amk-2020052513398> (visited on Oct. 16, 2022).
- [12] “Interactive ScienceDirect APIs.” (2022), [Online]. Available: <https://dev.elsevier.com/sciencedirect.html#/> (visited on Oct. 16, 2022).
- [13] “Ciência Vitae Swagger UI.” (2022), [Online]. Available: <https://api.cienciavitae.pt/docs/> (visited on Oct. 16, 2022).
- [14] M. Bostock, V. Ogievetsky, and J. Heer, “D3: Data-driven documents,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2301–2309, 2011. DOI: 10.1109/TVCG.2011.185. [Online]. Available: <http://vis.stanford.edu/papers/d3> (visited on Oct. 16, 2022).
- [15] “Django, QuerySet API reference - Django documentation.” (2022), [Online]. Available: <https://docs.djangoproject.com/en/4.1/ref/models/querysets/> (visited on Oct. 16, 2022).
- [16] A. Holovaty and J. Kaplan-Moss, *The Definitive Guide to Django: Web Development Done Right*, ser. IT Pro. Apress, 2009, p. 373, ISBN: 9781430219378. [Online]. Available: <https://books.google.pt/books?id=1fZCAAAAQBAJ> (visited on Oct. 16, 2022).

- [17] R. Touma, A. Queralt, and T. Cortes, “Capre: Code-analysis based prefetching for persistent object stores,” *Future Generation Computer Systems*, vol. 111, pp. 491–506, 2020, ISSN: 0167-739X. DOI: 10.1016/j.future.2019.10.023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167739X19314293> (visited on Oct. 16, 2022).
- [18] K. M. Tracey, *Django 1.1 Testing and Debugging*, ser. From technologies to solutions. Packt Pub., 2010, ISBN: 9781847197573. [Online]. Available: <https://books.google.pt/books?id=mWG1AAAAQBAJ> (visited on Oct. 16, 2022).
- [19] P. J. Baumgartner and Y. Malet, *High Performance Django*. Lincoln Loop, 2014. [Online]. Available: <https://mongard.s3.ir-thr-at1.arvanstorage.com/High%5C%20Performance%5C%20Django.pdf> (visited on Oct. 16, 2022).
- [20] W. Uruchi, M. Castrillo, and D. Beltrán, “Autosubmit gui: A javascript-based graphical user interface to monitor experiments workflow execution,” *Journal of Open Source Software*, vol. 6, no. 59, 2021, ISSN: 2475-9066. DOI: 10.21105/joss.03049. [Online]. Available: <http://hdl.handle.net/2117/343490> (visited on Oct. 16, 2022).

Data Model Diagram

- **BigAutoField**: An automatically incremented 64-bit positive integer, for primary keys.
- **IntegerField**: Standard 64-bit integer, like IDs or counters.
- **CharField**: Field for small to large sized strings, like titles or names.
- **TextField**: Field for large sized strings, like abstracts.
- **DateField**: Field for dates.
- **BooleanField**: Field for binary values.
- **ForeignKey**: Used to represent an “N to 1” optional relation.
- **ManyToManyField**: Used to represent an “M to N” optional relation.

Author	
id	BigAutoField NOT NULL
scopus_id	IntegerField
ciencia_id	CharField
orcid_id	CharField
name	CharField
publications	ManyToManyField
domains	ManyToManyField
name_list	TextField
h_index	IntegerField
citation_count	IntegerField
cited_by_count	IntegerField
current_affiliations	ManyToManyField
previous_affiliations	ManyToManyField
bio	TextField
degrees	TextField
distinctions	TextField
projects	ManyToManyField
synced_ciencia	BooleanField

Affiliation	
scopus_id	IntegerField NOT NULL
parent	ForeignKey
name	CharField NOT NULL

Project	
id	BigAutoField NOT NULL
name	CharField NOT NULL
desc	TextField
date	DateField NOT NULL
areas	ManyToManyField

Area	
code	IntegerField NOT NULL
name	CharField

Keyword	
id	BigAutoField NOT NULL
name	CharField UNIQUE

Publication	
id	BigAutoField NOT NULL
scopus_id	IntegerField
ciencia_id	CharField
doi	CharField
title	CharField NOT NULL
date	DateField NOT NULL
keywords	ManyToManyField
publication_type	ForeignKey
from_scopus	BooleanField NOT NULL
from_ciencia	BooleanField NOT NULL
full_text	TextField
abstract	TextField
areas	ManyToManyField

PublicationType	
id	BigAutoField NOT NULL
name	CharField UNIQUE