FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Building a search engine on a sports-related platform

Ricardo Filipe da Silva Néri Marques Carvalho

Mestrado em Engenharia Informática e Computação

Supervisor: Prof. Sérgio Sobral Nunes

July 27, 2023

Building a search engine on a sports-related platform

Ricardo Filipe da Silva Néri Marques Carvalho

Mestrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

President: Prof. João Correia Lopes, PhD
External Examiner: Prof. Nuno Escudeiro, PhD
Supervisor: Prof. Sérgio Sobral Nunes, PhD

July 27, 2023

Resumo

A área de recuperação de informação lida com a representação, armazenamento, organização e acesso a itens de informação, que normalmente são designados de documentos. Os motores de busca são resultado da investigação e desenvolvimento na área, e são responsáveis pelo acesso a informação *online*. Este trabalho tem como ponto de partida o motor de busca da zerozero.pt, uma plataforma online de conteúdos desportivos que, neste momento, depara-se com problemas no que toca a devolver os melhores resultados de acordo com as necessidades do utilizador. Na maioria das vezes, o sistema mostra-se incapaz de devolver os documentos mais relevantes de acordo com uma interrogação do utilizador. O nosso objetivo passa por desenvolver um novo motor de busca que seja mais capaz que o atual e que possa devolver resultados mais relevantes. Uma análise de outras plataformas de conteúdo desportivo foi realizada, não só a plataformas nacionais como também internacionais. Esta análise foi feita para perceber as funcionalidades de pesquisa presentes e a diferença entre estas plataformas e a zerozero.pt. Algumas destas funcionalidades incluem o aparecimento de sugestões à medida que o utilizador vai digitando na barra de pesquisa, a paginação de resultados ou a filtragem de resultados por data. Para alcançar o nosso objetivo, desenvolvemos um novo motor de busca usando o Apache Solr, tendo como ponto de partida um trabalho desenvolvido anteriormente que propôs melhorias ao sistema atual. Duas estratégias foram usadas nesta implementação: uma que consistia numa proposta apresentada num trabalho anterior, em que a interrogação de um utilizador era propagada para cada coleção individualmente, e uma segunda que consistia em ter uma única coleção com todos os documentos de todas as coleções, sendo que depois a interrogação era apenas enviada para essa coleção única. Uma análise de cliques foi também feita para analisar os resultados em que os utilizadores clicaram aquando de uma pesquisa e a sua posição na lista de resultados. Esta análise mostrou que a maioria dos resultados clicados se situam na primeira posição da lista e que as entidades mais vezes clicadas pelos utilizadores correspondem a equipas e jogadores. Finalmente, avaliámos o novo sistema com métricas já estabelecidas como o MAP ou MRR, usando duas coleções de interrogações com características distintas, para perceber se as alterações e as estratégias implementadas realmente vieram trazer melhorias ao sistema atual. Os resultados obtidos vieram mostrar que o nosso sistema foi capaz de ter uma melhor performance que o sistema atual presente na plataforma, tanto para a primeira coleção de queries como para a segunda. Também observámos que, para as duas estratégias usadas no nosso sistema, a estratégia que consistia numa proposta apresentada em um trabalho anterior obteve ligeiramente melhores resultados que a estratégia de ter um único core com todos os documentos de todas as entidades para a coleção de interrogações com maior entropia, mas verificou-se o inverso para a coleção de interrogações mais frequentes, onde a estratégia de um único core obteve melhores resultados.

Palavras-chave: Motor de busca, Recuperação de informação, Solr, Pesquisa em domínio específico, Análise de registos de interrogações

Abstract

The area of information retrieval deals with the representation, storage, organization and access to information items, usually called documents. Search engines come from the investigation and development in the area, and they are responsible for accessing online information. The focus of this work is the search system of the Portuguese sports-related website zerozero.pt, which is facing problems retrieving the best results according to the user's needs. The system, most of the time, does not retrieve the most relevant results to a given user query. We aim to develop a new search engine that is more capable than the current one and can retrieve more relevant results for the user's needs. An analysis of other sports-related platforms was made, not only on national platforms but international ones as well. This analysis was made to understand the search features present and the difference between those platforms and zerozero.pt. Some of those features are the appearance of suggestions as the user is typing on the search bar, pagination of results or results filtration by date. To reach our objective, we developed a new search engine using Apache Solr and had as background a previous work that proposed improvements to the current search system. Two strategies were used in this implementation: the first one consisted of a proposal from a previous work, in which a user query was sent to each core separately, and the second one consisted of having a unique core with all the documents from all the individual cores, in which a user query was then sent to that individual core. A click analysis was also made to analyze the results that the users click when performing a search and the position on the results list. This analysis showed that most of the results clicked were in the first position in the results list, and the entities that received more clicks were players and teams. Finally, we evaluated the new search engine with established metrics like MAP and MRR, using two sets of queries with distinct characteristics to understand if the proposed alterations and strategies improved the current system. The results obtained showed that our system was able to perform better than the current system of zerozero.pt, both for the first and second query sets. Also, we observed that, for both strategies used in our system, the strategy that consisted of a proposal from a previous work achieved mildly better results than the single-core strategy for the set of queries with highest entropy, but the inverse was observed in the set of most frequent queries submitted to the system, in which the single-core strategy achieved better results.

Keywords: Search engine, Information Retrieval, Solr, Domain-specific search, Query log analysis

Acknowledgements

First, I would like to thank my supervisor, Professor Sérgio Nunes, who helped me throughout this journey. Thank you for your patience and your always helpful feedback. This work couldn't have been done without your help and guidance.

I would like to thank my parents for all their support and for always being there when I most needed it. This work is also yours.

I would like to thank my girlfriend, Fabiana, for all the support and help. Thank you for being there.

To all my friends, I can't thank you enough. Thank you for all the laughs, the good moments and for all the help and support.

Finally, I would like to thank ZOS for all the help and for providing the right conditions so that this work could be done.

Ricardo Carvalho

"It's hard to beat a person who never gives up."

Babe Ruth

Contents

List of Figures

List of Tables

Abbreviations

WWW	*World Wide Web*
API	Application Programming Interface
URL	Uniform Resource Locator
HTTP	Hypertext Transfer Protocol
QLA	Query Log Analysis
MAP	Mean Average Precision
MRR	Mean Reciprocal Rank
RR	Reciprocal Rank
AP	Average Precision

Chapter 1

Introduction

The task of storing and finding information is an old problem. With the appearance of computers, the information available grew, and finding it became more challenging. And it is in this context that the area of Information Retrieval appears to introduce new techniques for indexing and ranking information. Finding information involves the user browsing the web or querying a search engine. Users introduce the user query, which represents their information needs, and then the system tries to find documents that could be relevant to them. Each document has a score associated with the relevance of that document to the query, and the system returns a list of documents ordered by that value (documents with a higher score are ranked higher in the list). Not only are search engines used to find information on the web, but they are also used in other specific search environments, like domain-specific search to find information online that is restricted to a certain domain or the desktop search to search within a user's own computer files.

In the following sections, we will present the context of this work as well as the motivation and the main objectives. The last section presents how this document is structured, particularly the following chapters and the topics addressed.

1.1 Context

This work has as its main goal to improve the search system of the website zerozero.pt, a sports website that was born in 2003 and which was initially focused on football, but since 2017 has extended its sports coverage, with the introduction of sports like basketball, volleyball or handball. This website has domains in eight different countries: Portugal, the United Kingdom, France, Spain, Germany, Italy, Brazil and the United States[1].

The work will focus on the Portuguese domain of the website, which is the domain most accessed by the users (80% of the traffic), according to the company that owns the platform. Currently, the search process in this platform occurs in a search bar placed at the top of the page, where the user can type the query. As the user is typing, a top 10 suggestions of results appear, and the user can click on one of them or click on the search icon to see a page with all listed results.

[1] https://www.zerozero.pt/

On this page, the users can then filter the results by entities, like players, teams or competitions, depending on the results.

The analysis made by João Damas [21] shows gaps in the search process that he identified and strategies to rectify those problems and improve the current system. Some of these problems consisted in search results that did not have the desired quality or the system, in some cases, redirecting the user to the page of the first result in the list of suggestions when the user wanted to see the full results list. He then proposed a solution to improve the search engine of the platform, so our implementation of a new search engine, based on his previous work, is expected to bring improvements to the system and for the users to experience better results in their search.

1.2 Motivation

The analysis showed that the current system is facing difficulties in retrieving relevant results for the users. The current system is not able to return relevant results to its users; queries that are not so popular or are vaguer may lead to poor results in the top 10 results list (suggestions list), and the system, most of the time, redirects the user to the page of the first result when the user wants to see all the results for the query. Considering that analysis and the strategies and improvements proposed by João Damas [21], we aim to solve the problems of the actual system and guarantee that it can return relevant results according to user queries. Also, this work focuses on the Portuguese version, which is the most accessed domain of the platform. We want to certify that the users of this platform can have a good experience regarding the search process and can access relevant results for them quickly.

1.3 Objectives

This work has one objective: develop a new search engine for the platform zerozero.pt, having as background the work already developed by João Damas [21]. He analysed the current system and proposed strategies to improve it. He proposed the implementation of a new search engine, with the use of Apache Solr[2], that could retrieve relevant results according to the information needs of the users. To achieve this, he proposed the use of search term payloads, which consisted in collecting previous search terms that were used to reach certain documents. After that, those terms were incorporated into the documents using a Lucene feature, payloads[3]. Therefore, the development reflected those proposals. We made a new search log analysis to understand the similarities and differences with João Damas [21] work, developed a new search engine based on his previous proposals and strategies and finally evaluated the system developed, using some metrics for that effect, like MAP and MRR.

[2] https://solr.apache.org/
[3] https://lucidworks.com/post/solr-payloads/

1.4 Document Structure

The rest of the document is structured as follows: in Chapter 2, we present a background on Information Retrieval, search engines and strategies in the indexing and retrieval processes. We also present some popular technologies used in search engines and a more profound overview of the technology that will be used: Apache Solr. In Chapter 3, we present the problem we are trying to solve and our solution proposal. In Chapter 4, we perform a QLA based on the collected search logs. Chapter 5 contains descriptions of the indexed collections, as well as the details of the implementation of the search engine, with Chapter 6 presenting the process of evaluation of our search engine based on some well-established evaluation metrics, as well as results discussion. Chapter 7 concludes the document and presents some reflections on the work that was developed, as well as possible future work that can further improve the search system.

Chapter 2

Search Engines

When we discuss the search topic, we are inherently talking about Information Retrieval or IR. The area of IR did not appear with the internet. Before the general use of search engines by the population daily, in the 1960s, IR systems were found in commercial and intelligence applications [25]. With the information and communication technology field improving and growing, the retrieval systems' capabilities grew with processor speed and storage capacity improvements. This has been reflected in an approach away from the traditional one, a manual library-based, to automated methods to acquire, index and search information [25]. The field of IR has moved from being an academic discipline to the basis underlying most people's preferred means of accessing information [15].

In the following sections, we will discuss the concepts underlying search engines and the processes that occur in them, like indexing and retrieval. In the final section of this chapter, we will talk about some popular search engine technologies, including Apache Solr, the technology that will be used in this work.

2.1 Concepts

The terms *Information Retrieval* and *search* often refer to the same thing. Manning et al. [15] present one of the possible definitions of IR:

"Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)."

When a user accesses a search engine, he has a specific information need that will be translated into textual keywords or the user query. After that, the system will retrieve a list of documents ordered by a certain score, attributed by the system, that is associated with the relevance of that particular document to the user. However, this is not always so linear. If the information need is very broad, the user probably will not find what he is looking for and may end up seeing himself on

a cycle between browsing and searching [3]. Unlike data retrieval, which deals with a very well-known structure, and whether the information retrieved is right or wrong, as there is no concept of relevance, IR usually deals with natural language text, which is commonly poorly structured. An IR system must extract information from the document and use this information to match the information need of the user. Relevance is crucial in IR, as an IR system must retrieve documents relevant to a user's information need and retrieve as few irrelevant documents as possible. Figure 2.1 shows a simple search engine architecture.

In the following subsections, we will discuss the main processes underlying search engines: indexing and retrieval.

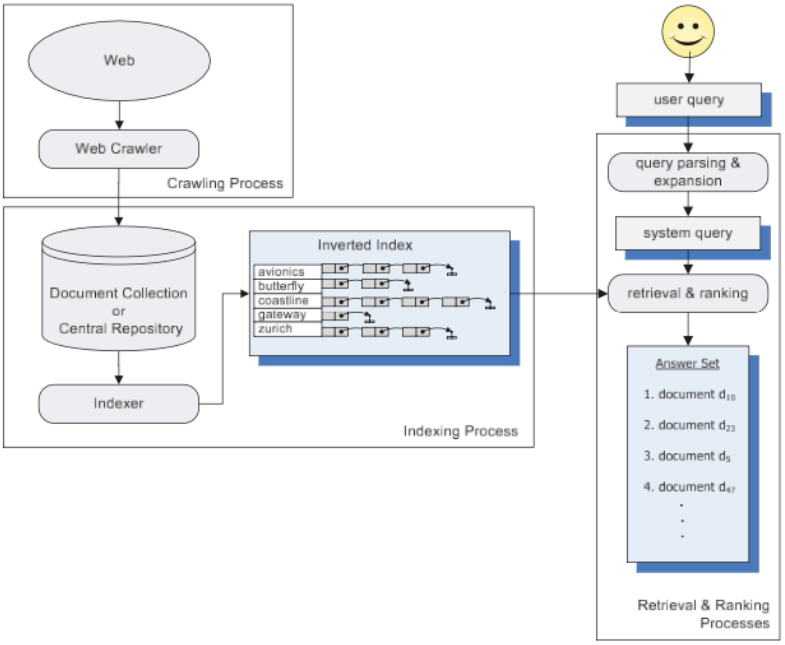

Figure 2.1: Example of a Search Engine Architecture from Baeza-Yates et al. [3].

2.1.1 Indexing

Indexing is one of the essential and first processes in IR systems. This process reduces the documents to the terms contained in them and provides a mapping from these terms to the documents containing them [13]. The indexing process has four stages: content specification (specifying and managing the documents corpus), tokenization, processing of document terms and index building. The index can then be stored in data structures, like inverted index or document index. Indexes can be built by applying algorithms or schemes; some are single-pass in-memory indexing or the blocked-indexing [13].

Indexing is the process that guarantees efficient retrieval. In this process, it is assumed that both the queries and documents are expressed as a set of index terms. Corpus Manager, a software tool for text processing, is responsible for specifying the documents and their structure, as well

as the content that can be retrieved from them. The documents are then passed through the *term pipeline*, which performs operations on the documents.

The processes inside the *term pipeline* are operations applied on text and performed on documents:

- **Tokenization** - consists in breaking down the documents into their components, called tokens or terms. Some issues may arise with abbreviations, accents or dates. Tokenization is also responsible for removing punctuation characters, like commas [13]. Figure 2.2 shows an example of a tokenization process.
- **Stop-words** - is the removal of common and frequently used words. Sometimes, extremely common words which appear to be of little value in helping to select documents that match an information need are removed from the vocabulary. However, there may be cases where these words are essential to the context. For example, *I am coming from Spain* is totally different from *I am coming to Spain*; the stop words [from] and [to] change the context and meaning of the sentence. Modern systems are not very much affected by index size or query processing time for including stop words into index, as it can also decrease recall in some cases [13].
- **Stemming** - the process of breaking down words to their base form. It removes prefixes and suffixes to reduce the word to its common form.

At the end of the pipeline, we have the inverted index. An inverted index is a data structure that has two main concepts: vocabulary and postings. The vocabulary consists of all the distinct terms present in the documents collection, as the postings list is an association between each unique term and the documents that contain them. Figure 2.3 shows an example of an inverted index.

Input: Friends, Romans, Countrymen, lend me your ears;
Output: | Friends | Romans | Countrymen | lend | me | your | ears |

Figure 2.2: Tokenization example from Manning et al. [15].

2.1.2 Retrieving

Concluding the indexing part, we now have to rank and retrieve the documents that the system finds most relevant to the user query. IR has multiple retrieval models; however, we will focus on 3 of them, which will be covered in the following subsections.

2.1.2.1 Boolean Model

The boolean retrieval model is one of the simplest and first developed models. In this model, queries are represented as a Boolean expression of terms. This model views each document as a

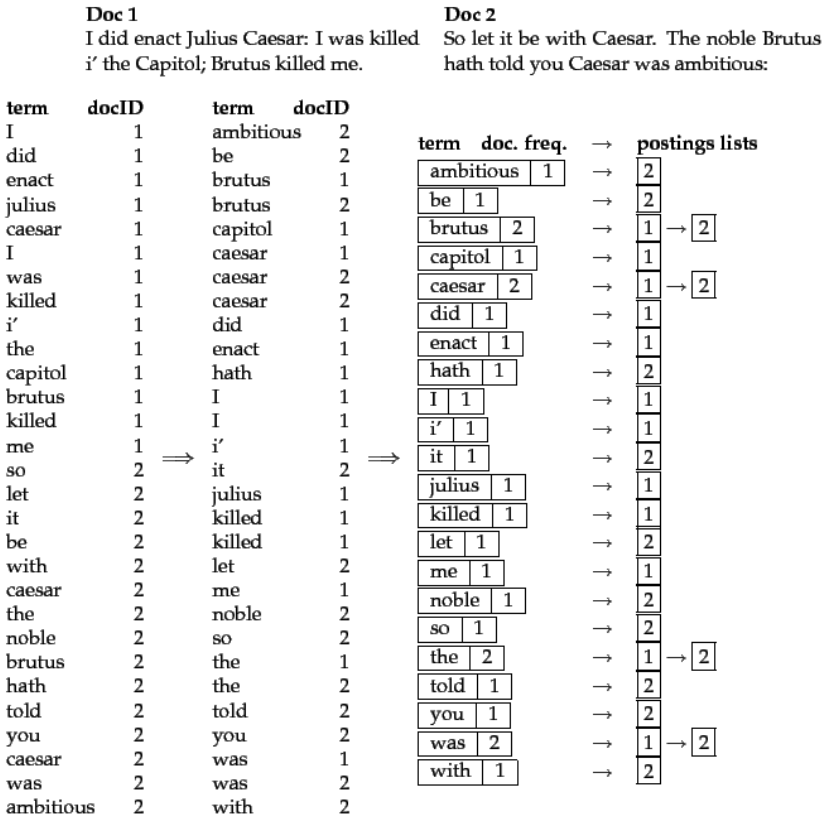

Figure 2.3: Inverted Index example from Manning et al. [15].

set of words. In this model, a document is either relevant or not, as there is no degree of relevance, and it does not consider how many times a term occurs in a document. A query q is composed of index terms linked by three connectives: AND, OR and NOT. Also, the index term weights have only two possible values, 0 or 1, which means that if the term is present in a document, it has the value of 1, otherwise has the value of 0. If we take a look at Figure 2.4, which represents the occurrence of each term in different documents, one if the term is present and 0 otherwise, and if we have this query

(nuclear AND treaty) OR ((NOT treaty) AND (nonproliferation OR Iran))

we can conclude that only the documents D2, D5 and D7 will be returned, as they are the only ones that match the query. As we can see, the boolean model is an easy-to-understand and implement model but has disadvantages. First of all, it does not take into consideration the number of occurrences of a term in a document, so we can have, for example, 100 occurrences of the terms nuclear and treaty in D2 and one occurrence in D5 that will be interpreted in the same way by the system, as it only returns exact matches, and may not be the case.

Also, as there is no concept of relevance in this model in terms of probability (or the document is relevant or not - 1 or 0), only exact matching, the list of results can be too small or too large and contain results that are not relevant to the user query.

	nuclear	nonprolife ration	treaty	Iran
D1	0	0	0	0
D2	1	0	0	1
D3	0	0	1	0
D4	0	0	1	1
D5	1	1	0	0
D6	0	0	1	1
D7	1	0	1	0
D8	0	1	1	1

Figure 2.4: Boolean Model Incidence Matrix from James Allan [2].

2.1.2.2 Vector Space Model

As the boolean model only returned exact matches without any concept of relevance present, this does not happen in the Vector Space Model.

In Vector Space Model, each document or query is an N-dimensional vector, in which N is the number of different terms present. The index i of a vector contains the score of the i-th term for that vector. The most commonly used score functions are term-frequency and inverse-document frequency [17]. Term frequency is the number of occurrences of a term in a certain document, as the term-document frequency is the number of documents in which a term appears. The inverse-document frequency is a weight that represents the proportion of documents that contain the term; the more frequent its usage across documents, the lower the score [15]. The inverse-document frequency of a term t is defined as follows:

$$idf_t = log\frac{N}{df_t}$$

where N is the collection size and df_t the document frequency of the term t. The combination of term frequency with inverse-document frequency results in a measure called $tf\text{-}idf_{t,d}$. This measure assigns a weight to a term t in document d and reflects the importance of a term for a document in the corpus.

The vector space model also uses the concept of similarity between two vectors, which can be document/document or document/query. This formula calculates the cosine of the angle described by the two normalized vectors. If the vectors are close to each other, the angle will be small, and the similarity will be high.

However, this model also has its limitations. For example, documents whose context may be relevant but the vocabulary does not precisely match the terms in the query will have low similarity. Also, this model, just like the previous one, assumes that the indexed terms are independent.

2.1.2.3 Probabilistic Model

The process of retrieving documents that will match the user's query is uncertain. This is due to the fact that the understanding of the information needed is not an exact subject, as a document that a search system finds relevant may not be relevant to the user. The probabilistic model aims to estimate how probable is to a document to be relevant for a given user query. This model is one of the oldest IR models but is one of the best performers and most commonly used.

This model aims to answer a question: "What is the probability of a user finding a document relevant to a query?". So, if x represents any document in a collection, and R the relevance of a document, knowing that R can only assume the values of 1 if the document is relevant or 0, if it is not, the probability of any document x to be relevant is

$$P(R = 1|x) = \frac{p(x|R = 1)p(R = 1)}{p(x)}$$

where $p(x|R = 1)$ is the probability of the document being x knowing beforehand that the document is relevant, $p(R = 1)$ is the probability of the document being relevant and $p(x)$ the probability of the document being x. Similarly, the probability of any document x not being relevant is

$$P(R = 0|x) = \frac{p(x|R = 0)p(R = 0)}{p(x)}$$

where $p(x|R = 0)$ is the probability of the document being x knowing a priori that the document is not relevant and $p(R = 0)$ is the probability of the document not being relevant. These two formulas are derived from the Bayes' Theorem [4], which calculates the probability of an event knowing that other event has occurred:

$$P(A|B) = \frac{p(B|A)p(A)}{p(B)}$$

The sum of both probabilities must equal 1, as a document can only be relevant or not relevant (R=0 $\vee$ R=1)

$$P(R = 1|x) + P(R = 0|x) = 1$$

How are the probabilities calculated? Estimating how each term contributes to relevance. To do so, measures like term frequency can be taken into consideration. The BIM (Binary Independence Model) was one of the first probabilistic models developed and represents documents and queries as binary vectors, in which each coordinate is a binary number (1 if x_t is present in document d, 0 otherwise). Although the BIM does not take term frequency into consideration, the Okapi BM25, a ranking function that estimates the relevance of documents according to a certain query, does.

2.2 Search Tasks

When we think of IR and, most concretely, the search process, one may think only of search on the Web, where a user types a query on Google[1] and looks for the information. However, new search tasks have emerged, some of them with specific attributes that influence user search behaviours. Some of these search tasks will be described in the following subsections, with a focus on their particular characteristics.

2.2.1 Complex vs. Simple Tasks

Bell et al. [5] defined a complex task as the uncertainty about what information is being sought and how to obtain relevant information. They developed three tasks, each one with a different level of complexity, with changes in the degree of uncertainty [30]. One of the examples provided by Bell et al. [5] is shown in Figure 2.5. We can distinguish complex search tasks from simple search tasks with the uncertainty present in each one. In complex tasks, it is unclear what information is relevant for the user and how to obtain it, whereas, in simple ones, it is clear what the user wants to see and how to obtain the information. Another difference is that in complex tasks, the process of finding information may force the user to seek information from more than one source of information.

Lowest complexity - complexity level 1 (Task C1)
While out for dinner one night, your friend complains about the rising price of petrol however as you have been driving for long, you are unaware of any major changes in price. You decide to find out how the price of petrol in the UK has changed in recent years.
Medium complexity - complexity level 2 (Task C2)
Whilst out for dinner one night, one of your friends' guests is complaining about the price of petrol and all the factors that cause it. Throughout the night they seem to complain about everything they can, reducing the credibility of their earlier statements so you decide to research which factors actually are important in deciding the price of petrol in the UK.
Highest complexity - complexity level 3 (Task C3)
Whilst having dinner with an American colleague, they comment on the high price of petrol in the UK compared to other countries, despite large volumes coming from the same sources. Unaware of any major differences, you decide to find out how and why petrol prices vary worldwide.

Figure 2.5: Queries with different levels of complexity from Bell et al. [5].

2.2.2 Specific vs. General Tasks

Specific tasks tend to have well-defined goals, as general tasks are more generic, e.g. Rouet [24] pointed out two examples of what could be considered specific and generic search tasks: "Which authors have provided the first clinical descriptions of anorexia?" could be associated to a specific search task, as the objective associated to the query is clear; on the other hand, "What treatments [for anorexia] may be suggested, and what are their effects?" can be described as a general task. Rouet conducted a study on the effects of task specificity on searching behaviours, and specific

[1] https://www.google.com

Table 2.1: Sample search questions and respective target menu items for the anorexia document from Rouet [24].

Type of question	Sample item	Target menu item(s)
Specific	Which authors have provided the first clinical descriptions of anorexia?	1. History of anorexia > 1.2. Early studies on anorexia
General	What treatments may be suggested, and what are their effects?	4. Possible evolutions > 4.1. Spontaneous or therapy-based recovery 5. Types of care proposed > 5.2. Medications

tasks were defined as locating one particular piece of information. Table 2.1 shows an example of both a general and specific task and their respective target menu items.

2.2.3 Exploratory vs. Lookup Tasks

White et al. [29] incorporated both exploratory and lookup tasks in their study. Two examples of these two tasks are shown in Figure 2.6. Lookup is the most basic kind of search task. These tasks return discrete and well-structured objects like short sentences or numbers. Most people think of lookup tasks as "question answering" or "fact retrieval" [16]. On the other hand, exploratory search tasks are seen as a support to learning and investigation.

Known-item task
You are doing some research for a term paper you are writing and need to find the name of the first woman to travel in space and her age at the time of her flight.

Exploratory task
You are about to depart on a short-tour along the west coast of Italy. The agenda includes a visit to the country's capital, Rome, during which you hope to find time to pursue your interest in modern art. However, you have recently been told that time in the city is limited and you want information that allows you to choose a gallery to visit.

Figure 2.6: Examples of an exploratory task and a lookup task from White et al. [29].

2.2.4 Transactional vs. Navigational vs. Informational Tasks

The type of tasks described above are intended to represent people's information needs, i.e. it is assumed that people search because they want to find information that answers some question. With the appearance of the Web, transactional and navigational tasks appeared. While the goal of an informational task is to seek and acquire information, navigational tasks consist in accessing a particular website, and transactional tasks can be seen as performing web-mediated activity

[30]. Joachims et al. [12] present some examples of what could be a navigational and informational task: "Find the homepage of Michael Jordan, the statistician" and "Find the homepage of the 1000 Acres Dude Ranch" are examples of navigational search tasks; "Where is the tallest mountain in New York located?" and "What is the name of the researcher who discovered the first modern antibiotic?" are examples of informational search tasks. Transactional searches represent the intention of the user to do something, e.g. buy something, download a file or sign up for a consultation.

2.3 Technologies

Some of the most popular technologies regarding search engines, according to DB-Engines [22], are Sphinx[2], ElasticSearch[3], Solr, Splunk[4] and MarkLogic[5]. We will now present a general overview of these technologies [20].

- **Sphinx** - Search engine technology whose name is derived from *SQL Phrase Index*, Sphinx is a standalone full-text search engine whose main objective is to provide a fast and relevant full-text search functionality. Despite being built to easily integrate with SQL databases, it can be used without the need for a SQL database. It is also easily accessible with the use of scripting languages.
- **ElasticSearch** - ElasticSearch is an open-source full-text search engine based on Lucene. It is often used for online web store catalogs and for collecting and analyzing logs for data mining. It is a document-oriented distributed database and uses inverted index as its basic index structure.
- **Splunk** - Splunk is a database system designed for analyzing machine-generated data. This data may come from other databases or web servers, and Splunk then analyzes the data, producing dashboards and graphs for visualization. It stores data in indexes organized in a set of buckets. Splunk is a NoSQL database with a key value data model.
- **MarkLogic** - MarkLogic is a NoSQL database that supports XML and RDF data. It uses a distributed architecture.

2.3.1 Solr

The technology that is going to be used in this work is Apache Solr. This technology will be used as it was the technology proposed by zerozero.pt for the development of this work. Solr is a free and open-source search engine technology built on Apache Lucene. It is a NoSQL search platform that stores data as documents. Documents can be sent to Solr in multiple formats such

[2] http://sphinxsearch.com
[3] https://www.elastic.co/pt/
[4] https://www.splunk.com
[5] https://www.marklogic.com

as JSON or XML, and then offers the possibility to search for those documents. Solr has a query screen, as seen in Figure 2.7, that offers the users the possibility to search for documents using queries. In this screen, users can specify which the fields that they want to see (default is all the fields, but one may want to just see a few of them), the number of documents they want to see (Solr's default is 10), boost document fields and sort the documents based on certain criteria. Solr uses inverted index, which allows a fast query processing; it supports distributed search and index replication. Upon an update, Solr achieves reliability and consistency by writing all the documents as transaction logs to disk. The REST interfaces provide easy integration with many languages. Users can send HTTP requests to Solr and receive the responses via Client API's. We used SolrClient [18], a Python library for Solr, in our work, which allowed us to build the request object and then sent it to Solr. When the user submits a query to Solr, it examines one document at a time. Then, a score is calculated and attributed to each document, and the final results list is ordered by that score value (assuming that the user does not want to order the results by other criteria).

2.3.1.1 Deployment and Operations

To start Solr, we can use the *start* command, the *restart* to restart Solr while it is running or if it has already stopped, and the *stop* command to stop the Solr. Solr also allows the user to change the listening port to one that is not the default, create collections or cores, depending if Solr is running on Standalone or SolrCloud mode, and create shards, which are database partitions that store parts of a collection and create a fixed number of replicas for a single collection, that is the number of copies of each document in a collection[6].

With Solr running, if the user accesses **http://hostname:8983/solr/**, it will have access to the Solr Admin UI, where information regarding Logging, Collections and Thread Dumps are present[7].

As stated above, Solr has a query screen which allows users to perform requests in a more user-friendly way. To access this query screen, it is necessary to create a core and insert documents in it. The field types in Solr, which are present in an XML schema file, define how Solr should interpret data in a field. Many field types are included by default, be we can create as many as we want. For example, if we have a field that has a date, the field type that will be assigned to that field is not the same as the one that will be assigned to another field that only has text in it. An example of field types is shown in Figure 2.8. Inside these type fields, we can also insert Tokenizers and Filters. A Tokenizer is responsible for breaking a field data into tokens, a sub-sequence of characters in the text. Some of the most used Tokenizers are the Standard Tokenizer, which treats whitespaces and punctuation as delimiters, and Whitespace Tokenizer, which splits the text stream by whitespaces, keeping the punctuation in tokens. On the other hand, Filters examine a stream of tokens and then transform them according to the filter that is being used. Some of the most used Filters

[6] http://lucene.apache.org/solr/features.html
[7] http://lucene.apache.org/solr/features.html
[8] https://solr.apache.org/guide/solr/latest/query-guide/query-screen.html

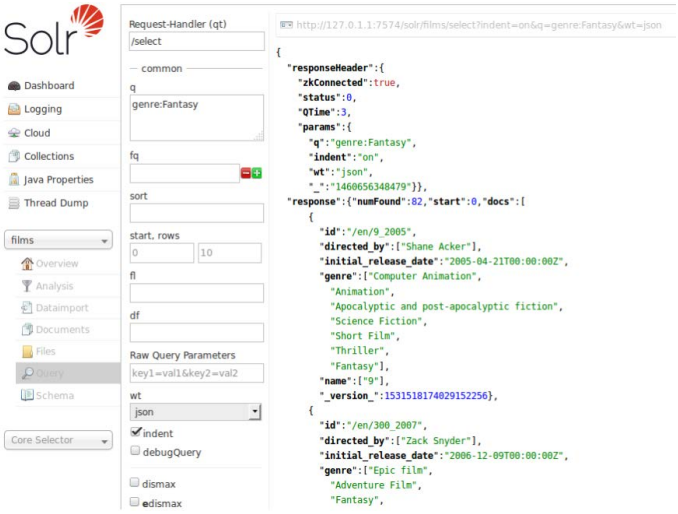

Figure 2.7: Solr Query Screen[8].

are the Lowercase Filter, which converts all characters in a token to lowercase, the ASCII Folding Filter, which converts alphabetic, numeric and Unicode characters which are not in the Basic Latin Unicode Block to their ASCII equivalents and Synonym Graph Filter, that takes a file containing a list of synonyms (one per line) and maps single or multi-token synonyms.

```
<fieldType name="text_general" class="solr.TextField" positionIncrementGap="100"> 1
  <analyzer type="index"> 2
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
    <!-- in this example, we will only use synonyms at query time
    <filter class="solr.SynonymFilterFactory" synonyms="index_synonyms.txt" ignoreCase="true"
expand="false"/>
    -->
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
  <analyzer type="query">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
    <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true"
expand="true"/>
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
</fieldType>
```

Figure 2.8: Solr Field Types example[9].

On the query screen, Solr offers some query options that allow the user to search documents, filter them, order them based on some criteria, boost document fields and choose what fields the user wants to see displayed. Some of the parameters available are:

- **q** - used to define a query. It is a mandatory parameter
- **fq** - restricts the set of documents that can be returned, without influencing the score
- **sort** - used to sort the documents retrieved, in either ascending or descending order

[9] https://solr.apache.org/guide/solr/latest/indexing-guide/field-type-definitions-and-properties.html

- **start, rows** - the first parameter is used to specify at which matching document should the documents be returned (if start is 2, the results should be returned starting on the second matching document). The second one specifies the number of documents that Solr should display at a time (default is 10)
- **fl** - specifies the fields that should be included in the results

2.4 Search Engines in Sports Websites

An analysis was performed about the search engine not only from zerozero.pt but from other national and international platforms. This analysis allowed us to understand how was the search process in other platforms and what features were present in each one of them so that we could compare them to zerozero.pt. We wanted to understand how was the search process done in other sports-related platforms, what type of search filters were present and how were the results displayed.

2.4.1 National Platforms

The only platform analyzed from Portugal was *Maisfutebol*[10]. Just like zerozero.pt, the search bar is placed on the top of the screen. The user must click on the search icon, and the bar appears. As the user is typing, there is not any list of suggestions that best match the query, unlike zerozero.pt. When the user presses *Enter*, a list of results is shown. These results can be news, videos or photos. For each type of result, the platform displays the number of results available from that type. There is not pagination, so the user will see the full results list on a single page. For the news and videos, a tags field is present with all the tags related to that result. Moreover, the exact time in which the article or video was published on the website is present, as well as news that are related in some way to the article that the user is currently reading. At the bottom of the page, we have the top 3 most seen videos and top 3 most read news lately.

2.4.2 International Platforms

The process of choosing the international platforms to analyze was based on the top 14 most popular sports websites according to the platform KreedOn[11]. From that list, we chose 6 platforms that we found interesting to analyze and compare with zerozero.pt:

- **Marca**[12] - in this Spanish platform, the search bar is also placed at the top of the screen. There are not any suggestions as the user is typing in the search bar, and the results page shows the number of total results. The results of this platform are news articles, and the user can order them by date or similarity degree. This similarity degree is a percentage score that

[10] https://maisfutebol.iol.pt
[11] https://www.kreedon.com/list-of-best-sports-websites/
[12] https://www.marca.com

describes the probability of that article being relevant to the query inserted. The user can also filter the results by sports, date, institution or by person (persons related to the news available according to the user query). The results are separated by pages, and the user can define how many results he wants to see on a single page (10, 25 or 50; default is 10) For each filter, the number of results is available. Each article has an image, the title, subtitle and body of the news, the exact time in which it was published on the website, comments from the users (if any) and related news (according to the news that the user is watching at the moment).

Figure 2.9: Search system from marca.com.

- **ESPN**[13] - like the other platforms, the search process also occurs at the top of the screen; the user just needs to click on the search icon, and the search bar will appear. Like zerozero.pt, a list of suggestions appears as the user is typing. The results page is divided into categories, each representing the type of result, and the user can see all the results for each category. For the query [benfica], the type of results that we obtained were Teams, Articles and Clips. For the query [ronaldo], the type of results that we obtained were Players, Articles and Clips. For the query [premier], besides having results with the type Article and Clips, we also have results with the type Leagues. For results of the type Teams, we have information about past games from the team, as well as the next match, news related to the team and the current position of the team in their respective national league. Regarding leagues, we have articles related to the league itself as well as the current standing of the competition. On Articles, the articles related to the user query are shown. Each article has a title, the name of the person who wrote it, the date when it was published and the body of the news. Clips are short videos which are related to the user query. Each clip has a little description.

[13] https://www.espn.com

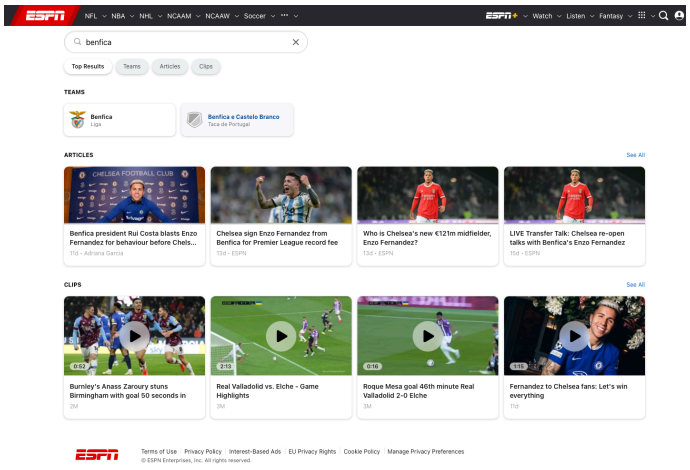

Figure 2.10: Search system from espn.com.

- **NBC Sports**[14] - the search bar is also placed at the top of the screen; to have access to it, the user must click on the search icon. There are not any suggestions as the user is typing, and the results are of two types only: Videos and Articles. We tried to search for teams and players, but the results were only of these two types. There is pagination of the results.

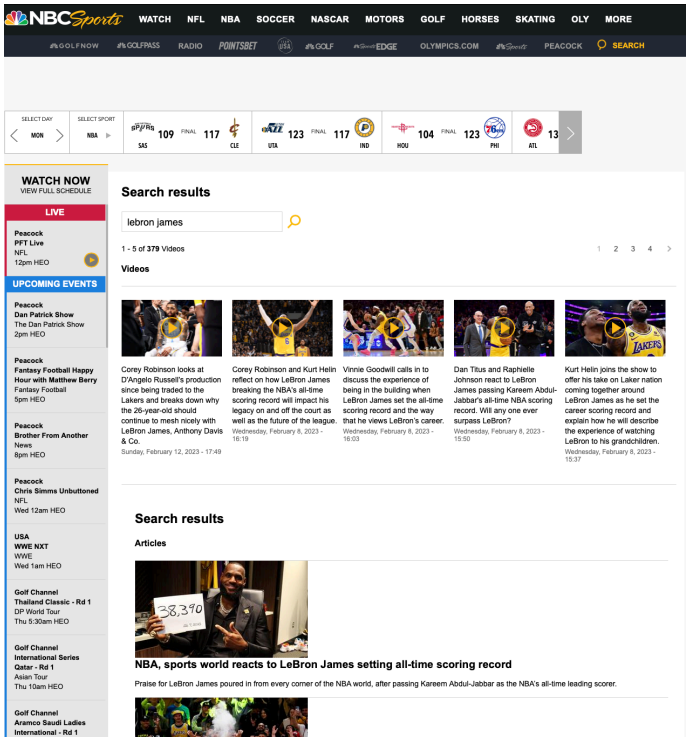

Figure 2.11: Search system from nbcsports.com.

- **Fox Sports**[15] - at the left of the screen, the user has access to the search button. When

[14] https://www.nbcsports.com
[15] https://www.foxsports.com

clicked, the search bar appears, as well as filters that are at the user's disposal for a faster search. For example, if someone wants to search for [LeBron James], instead of typing [lebron james] in the search bar, one can use the filters, in the format of a dropdown, to reach the desired player. The filters are available for Sports, Teams, Players, Shows and Personalities. A player has information about their own biography, social media, statistics and videos. It also has articles related to the player. A team has information about their schedule, the current standing of all the competitions in which the team is competing, statistics and videos, whereas a personality has information regarding their social media and related videos.

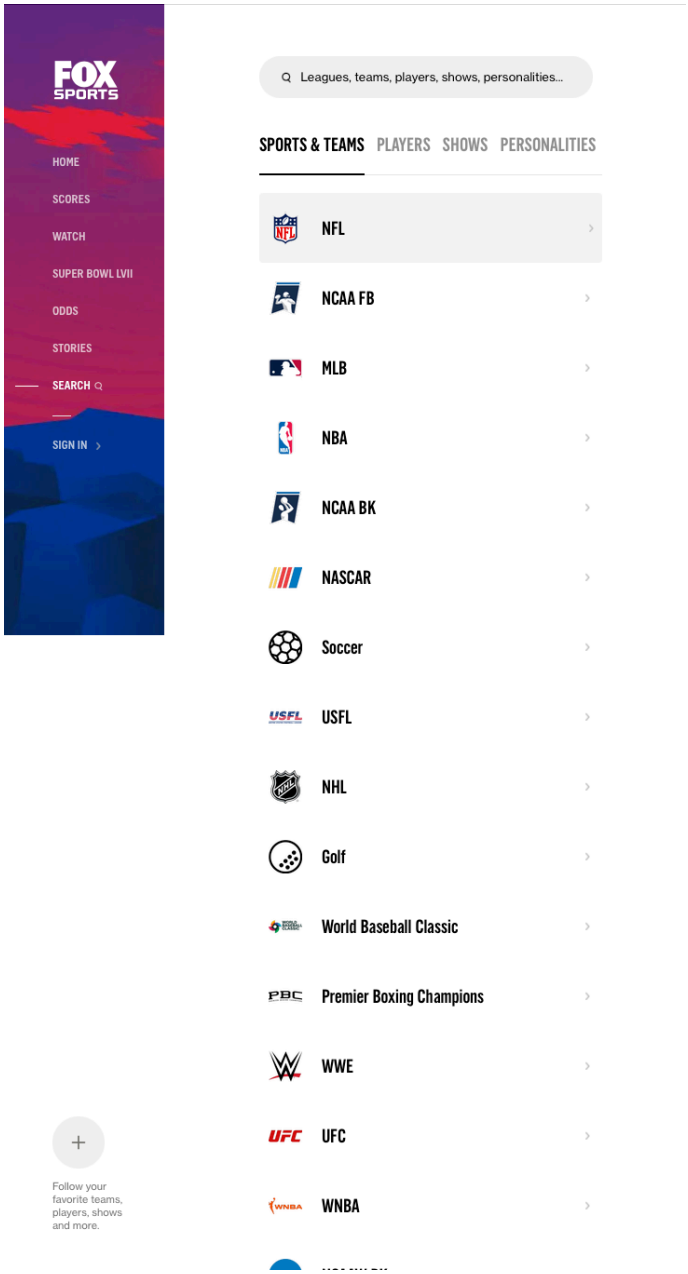

Figure 2.12: Search system from foxsports.com.

- **Sports Illustrated**[16] - the search icon is located at the top-right corner of the screen. Upon clicking, a new page with the search bar is presented to the user. There is not any list of suggestions, and when *Enter* is pressed, the total list of results is shown. There is no pagination, and the only types of results available are Articles and Videos. There is no type of filtering. Each article has a title, a subtitle, the name of the journalist who wrote the news, the date when the article was published, the text of the article and comments from the users. Each video only has the video itself and the date it was published on the website.

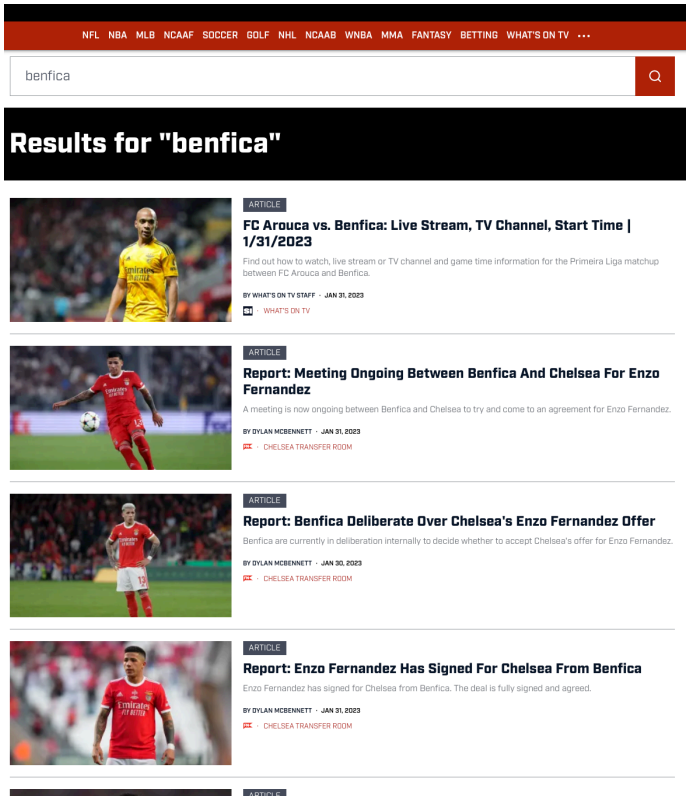

Figure 2.13: Search system from si.com.

- **Yahoo! Sports**[17] - with the search bar placed at the top of the screen, the system automatically presents the top 5 trending articles upon clicking in the search bar. When the user starts typing, a list of suggestions is presented, and the results are divided into categories, represented by the type of the result (Teams, Players, Leagues...). There is not a way to access the page with the total of the results retrieved, as when we click on the search icon, we are redirected to a Yahoo! page with the results for the query of the first result presented in the suggestions. For example, if we search for [bruno fernandes], the first suggestion displayed is the player Bruno Fernandes. If we click on the search icon, we are redirected to a Yahoo! page containing all the results for the query [bruno fernandes].

[16] https://www.si.com
[17] https://sports.yahoo.com

Table 2.2: Comparison of some sports-related platforms according to some metrics.

	Marca	ESPN	NBC Sports	Fox Sports	Yahoo! Sports
Suggestions		X			X
Any type of filter	X	X		X	X
Pagination	X		X		X
Filter by entity	X	X		X	X
Volume of results	X		X		X

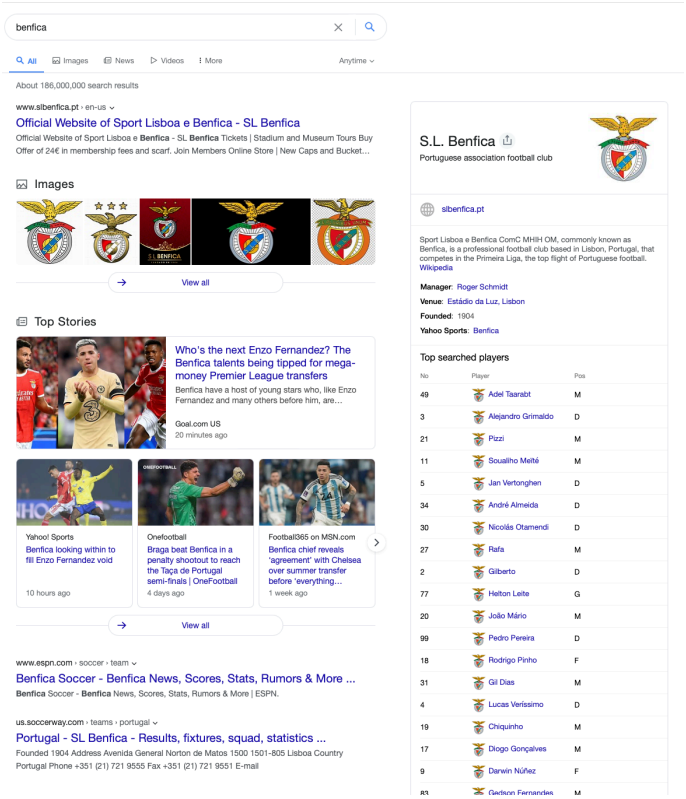

Figure 2.14: Search system from sports.yahoo.com.

2.5 Previous Work

As explained and described before, João Damas [21] proposed strategies and improvements to the current search system of zerozero.pt. Our work will have as starting point his previous work, and our implementation will reflect his proposals.

An analysis of the query logs was conducted, where the user search patterns were analyzed as they presented information about the queries made by the users to the system. Search on open Web or domain-specific search has similarities: query length tends to be short, as well as the session length and duration. Also, recent studies show that users often refine their queries with no more than one term at a time, according to João Damas [21].

The work is divided into two main parts: the analysis of the query logs and then he describes his strategies for the creation of a new search engine. The analyzed dataset concerned queries for a period of two weeks. Each entry corresponds to a search performed by a user or a clicked search result. Queries made by non-human entities were removed: either queries whose User-Agent string matched a non-human origin or sessions with a usually high number of queries. Query terms were found to be short, with an average of 6.44 characters, and users tend not radically to change the initial query attempt. This analysis was interesting to understand how the users searched the platform and their search patterns: queries made, length of the queries, click information...).

After this extensive analysis, the implementation of a new search engine was made. João Damas [21] focused on the four main entities that users most searched: Competitions, Teams, Managers and Players. A description for each collection was made, with the description of the fields for each document of that collection and the number of results present in that sample for that collection. Each data structure related to an entity had its own fields, as the information regarding each entity varied between them. Upon a query request, the query was sent to each core individually. Also, search term payloads[18] was used to associate scores to individual terms. This is helpful to weight terms in the form of relevance confidence. The system architecture presented by João Damas [21] in his work can be seen in Figure 2.15. The user sends the query to Python's Flask server, which queries the Solr collections. The documents are returned to the Flask server, which merges the results and returns an ordered list.

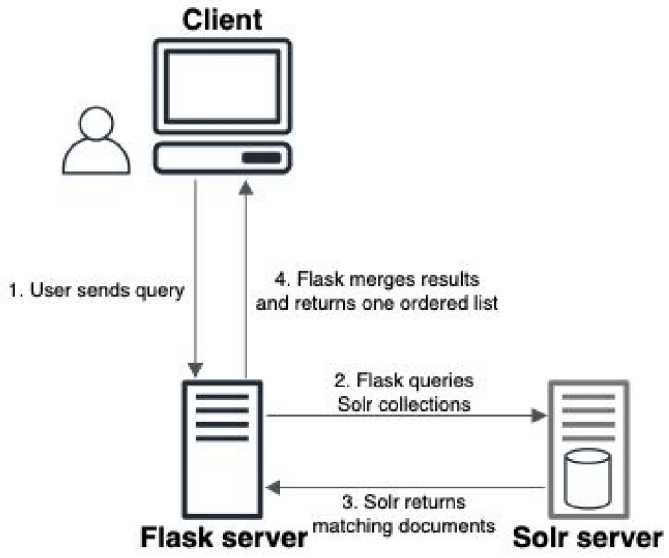

Figure 2.15: Overall system architecture of João Damas' work from João Damas [21].

To merge the documents, a normalization of the scores was needed. This was achieved with the CORI [7] algorithm, one of the most well-known algorithms for this task. For a document D retrieved from the collection C, the normalized score according to CORI [7] is

[18] https://lucidworks.com/post/solr-payloads/

$$FinalScore = S_D * \frac{1 + 0.4 * S_C}{1.4}$$

where S_D is the original document score and S_C is the collection's score. The collection's score is calculated using a strategy that will be explained in Chapter 5. The next step focused on the evaluation of the system developed. Two distinct sets of queries were used, one containing popular queries in terms of frequency and the other one focused on queries that produced higher variability in clicked results. In the first set, the top clicked result in each of the 200 queries had an average click share of 85%, while the other set had an average click share between 30% and 40%. Some metrics were applied for the evaluation of the results, like MAP, MRR and DCG (Discounted Cumulative Gain).

2.6 Summary

With the technological advances and the increase of information available on the Web, new strategies for indexing and retrieval appeared. Search engines have become the preferred mean of accessing information online by people. However, a search engine must be capable of returning relevant results according to the user's information needs. We aim to develop a new domain-specific search engine, more capable than the current one, based on the previous work of João Damas [21], and reflecting his proposals and strategies, such as the use of search term payloads as complementary information so that future users can find documents faster.

Chapter 3

Search in the Sports Environment

3.1 Overview

The search process in zerozero.pt occurs in a search bar placed at the top of the screen. The user then starts to type in the search bar, and after three or more characters are inserted, a top 10 results suggestions to the current user query is displayed. The user can click on one of the suggestions or the search icon to go to the results page. Figure 3.1 shows the search process in zerozero.pt. On the results page, the user can filter the results by players, teams or competitions, depending on the query inserted. After a manual analysis of the search process in zerozero.pt, we conclude that for teams, the searchable fields are the name, the abbreviation, the keywords and the old names that the team could eventually have. For the players and managers, the searchable fields are the name, abbreviation and keywords. In the case of the competitions, only the fields regarding the name and the abbreviation of the competition are searchable.

3.2 Problem

The current search system of the platform is facing troubles, mainly regarding the incapacity to retrieve the best results for the users. An example of this specific problem can be seen in Figure 3.2, which shows the suggestions retrieved by the system to the query [académico]. Despite the first result being relevant to the query inserted, the second is irrelevant, as there is no association between the result and the term *academico*. This result appears because the old name for this team has the term *académico* in it (Clube Académico de Felgueiras). For example, a more relevant result would be the team *Académica de Coimbra*. Not only that, but the search seems inconsistent in some way; for example, if we look at Figure 3.1, we can see that the system shows suggestions of results that could match that specific user query. If the user clicks on the search icon, it will be redirected to a page containing all the results retrieved, with pages separating them. The problem is that on that page, the first results are not the same as the ones displayed in the search bar when the user wrote the query. What happens is that the results page does not contain the top 10 results, which are only shown in the suggestions. If a user types fast, he will most likely not notice the

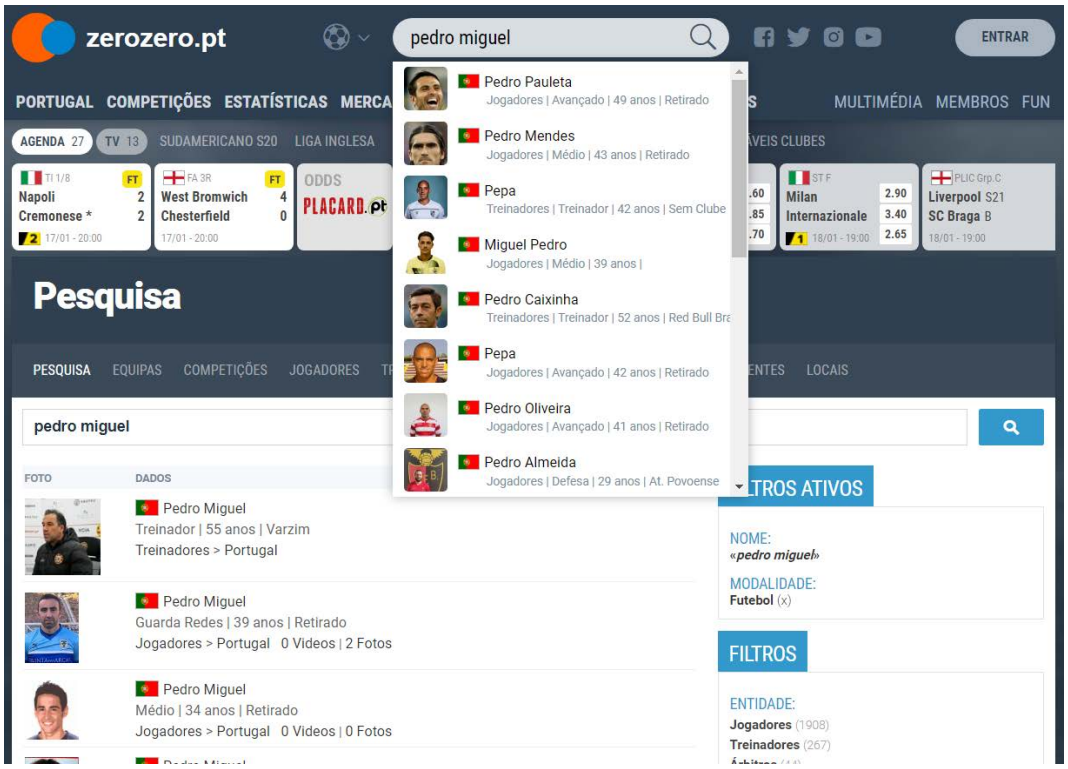

Figure 3.1: Results in the current search system for the query [pedro miguel], with automatic suggestions.

suggestions the system provides and will skip directly to the results page, which does not contain the suggestions. Another problem is that if a user misspells a term, the system will not retrieve any results, as Figure 3.3 shows (correct name is [diogo dalot]). The fact that the search system is not capable of retrieving the best results indicates problems in the search process, as pointed out by João Damas [21] in his work, so we will try to develop a new search engine that can solve the current problems and provide the users with a better experience regarding the search.

3.3 Proposed Solution

Our work will focus on implementing a new search engine for the platform zerozero.pt. The search technology that we are going to use is Apache Solr, and our objective is to reflect the proposals and strategies from João Damas [21] in the development of the new search engine. For this, we will create five cores, one for each entity (Players, Managers, Competitions and Teams) and one extra containing all the documents from all entities. This extra core was not present in João Damas [21] work, as his strategy passed by sending the query to each core individually, but we used it as it was a strategy proposed by zerozero.pt, so we could then evaluate what of the two strategies (query sent to each of the four cores individually vs query sent to one core containing all the documents from all entities) returned the best results according to the users' information needs. For each entity, we will analyze what fields we are able to work with and how they might be helpful in

Figure 3.2: Suggestions displayed for the query [académico].

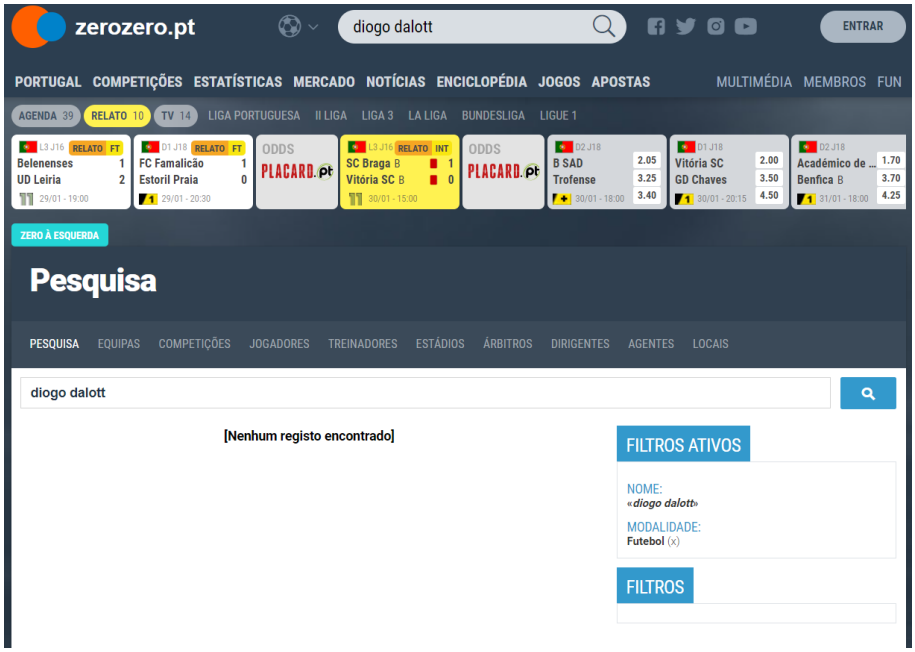

Figure 3.3: Typo in the query [diogo dalott], which leads to the system not retrieving any result.

the search process. This step is important so we can decide the best schema for each entity. For the case in which the query is sent to each core separately, the one that is present in João Damas [21] work, we will normalize the scores before merging the documents from all the entities. We

will also boost different document fields differently, as some fields may be more relevant than others. Previous search terms will also be taken into consideration when developing the search engine, as they can help future users to reach certain documents. To achieve this, we will use a Lucene feature called payloads[1]. The idea behind this feature is to assign a score to an individual term, so we will use this to attribute a score to individual terms according to how often they were used to reach a certain document. This will be used as a complementary information and not as a replacement for document context in indexing. Finally, we will make use of a Solr feature called SpellCheck to give query suggestions to users when they make a typo in their query attempt, as currently, the platform does not give any results when the user misspells a term and does not offer any query suggestions to correct the initial query.

Finally, we will evaluate our developed search engine, comparing it to the current production engine. To do so, we will use two sets of queries with different characteristics (frequency and entropy) collected from the QLA process in Chapter 4. The comparison between both systems will be made making use of well-established IR metrics.

[1] https://lucidworks.com/post/solr-payloads/

Chapter 4

Search Log Analysis

We will describe our analysis on the search logs provided by zerozero.pt, as well as the pre-processing performed on the dataset. This analysis is important, as it will be used for the effects of comparison with the search logs from the developed search engine, mainly through the click information. This analysis was already done by João Damas [21] in his work; however, we will analyze the search logs once more to see if there are any relevant changes between the results achieved in our work and the results reported in João Damas's [21] work.

4.1 zerozero.pt's Search Logs

The analyzed dataset contains queries submitted to the search system of zerozero.pt in the month of January 2023 (between 1st and 31th of January). Moreover, this dataset not only contains queries to the Portuguese domain but also to other domains. The log was stored in a CSV file, and the first line corresponded to the fields header. Listing 4.1 shows an artificial example of an entry in the file, and Table 4.1 describes some of the most important header fields. Each entry corresponds either to a search submitted to the system or to a clicked result: when a user goes to the results page for a query, an entry is stored; the same happens when the user clicks on a result: an entry is stored containing information about the current page, the position of the result clicked in the list, and the information about the entity clicked (Player, Team, Coach...). In total, the original log contained 1,073,429 entries.

```
"id","time","date","method","page","referer","IP","sess_id","username","agent","
    language","bot","page_number","mysql_id","cache","server_ip","geo_location","
    search_string","tp_item","fk_item"
"1","2023-01-01 00:00:00","2023-01-01","POST","jogador.php?id=8888888888&search
    =1","https://www.zerozero.pt/","255.255.255.255","aaabbbbccccddddeeee","
    USERNAME","Mozilla/5.0 (Linux; Android 9; LM-X420) AppleWebKit/537.36 (KHTML,
    like Gecko) Chrome/108.0.0.0 Mobile Safari/537.36","pt","0","0","1","0","SERVER
    ","pt","search_example","2","9999"
```

Listing 4.1: Example of a log entry.

Table 4.1: Main fields from search logs of zerozero.pt.

Field	**Description**
time	Timestamp, in the YYYY-MM-DD HH:MM:SS format, that represents the time in which the query was made to the system
date	Date, in the YYYY-MMM-DD format, that represents the date in which the query was made to the system
page	URL of the result clicked by the user
referer	URL that represents the page in which the user was when the search was performed
IP	IP address of the user who performed the search
sess_id	Session identifier of the user
username	Username of the authenticated user, or empty if the user was not logged in
agent	User Agent string that identifies the application of the requesting user agent
page_number	Number of the results page viewed by the user (0 if not applicable or user clicked in a result from the suggestions offered by the system)
mysql_id	Position of the clicked result in the results list (0 if not applicable)
geo_location	Geo-location associated with the IP address that produced the request. Stored using ISO 3166 - alpha-2 two letter code [1]
search_string	User query submitted to the system
tp_item	Integer identifier that represents the type of the entity clicked (e.g. 2 for Competitions, 4 for Players...)
fk_item	Unique item identifier

The first step regarding preparing our analysis consisted of removing irrelevant fields for our purpose. For this, we made use of the *csvquote* [9] tool, which uses the *cut* UNIX utility, which allowed us to separate the fields correctly, as some fields contained characters that could be interpreted as delimiters.

4.2 Preparation

In the search log, not all the entries were of human origin. Some of them consisted of non-human origin. Therefore, we needed to remove the entries whose User-Agent string corresponded to non-human origin.

4.2.1 Queries made by non-human origin

Although we were only interested exclusively in queries of human origin, a small percentage of the entries present in the search log were of non-human origin. Queries of this type can appear for different reasons in logs, one of them being malicious spyware, for example, or a Web crawler that was updating their index entries for an open Web search engine. To identify these types of queries, we made use of the User-Agent string, which was identified in the *agent* field. A User-Agent request header is a string that contains information about the application, operating system or version of the requesting user agent [1].

To identify User-Agent strings that could potentially belong to non-human origin queries, we used the same list João Damas [21] used in his work, as shown in Appendix A.1. This list contains predefined User-Agent collections with known bot expressions, as well as other User-Agent strings that he identified as potentially suspicious. With this list, we went through all the entries and checked if any of the User-Agent strings present in the logs contained any of the User-Agent strings included in the list. Finally, 5,297 queries were removed in this process, accounting for less than 1% of the total queries.

Despite the removal of these unwanted queries, we could still be having queries that were of no interest to us, like empty queries. Using the *awk* UNIX command to check if the *search_string* field was empty, we concluded that all the entries present in the log had at least 1 character.

4.2.2 Query standardization

One of the other steps taken regarding the preprocessing of the queries was the standardization of all the queries. This step was important in the sense that some queries with the same form could be seen by the system in a different way, whether because one of the queries could have lowercase letters and the other uppercase or one of the queries could have whitespaces at the beginning or end of the query. Queries with the same form should be seen exactly the same by the system (e.g., [cristiano ronaldo] is equivalent to [Cristiano Ronaldo] and [cristiano ronaldo]). To achieve this, we removed all leading and trailing whitespace (e.g., whitespaces at the beginning and end

[1] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/User-Agent

Table 4.2: Query Log statistics summary.

Metric	Value
Nr. Queries (before removal of bot activity)	1 073 429
Nr. Queries (after removal of bot activity)	1 068 132
Nr Terms	1 386 762
Nr Sessions	514 623
Mean Characters Per Term	7.03
Mean Terms Per Query	1.29
Mean Queries Per Session	2.09
Mean Clicked Ranking	1.60

of the query, respectively), all characters were converted to lowercase, as casing is indifferent in this context and, finally, all characters were converted into their ASCII representation (e.g., [joão cancelo] is transformed into [joao cancelo]). When writing queries, users usually do not type queries in the same way, even if the query is semantically the same. Some may write characters with accents, and others can type the first character of each term as an uppercase letter, while others use lowercase in every term.

4.3 Analysis

We will now present the analysis of our dataset. The analysis mainly focuses on the query and term. In the current search system, as said before, users can type the query in the search bar. When three or more characters are inserted, the system suggests a top 10 results to the user, and the user could opt to click on one of those results or click on the magnifying glass icon to see all the results (this is not always true; e.g., the query [pedro miguel] allows the user to click on this icon and see all the results, but when the user types [benfica] and clicks on the icon, it is automatically redirected to the page of the first result presented). However, in our dataset, only results from the suggestions are considered, which means that the positions of the results clicked are in the range [1,10]. Table 4.2 shows a general overview of some of the main metrics measured. As expected, the number of queries, terms and sessions in our work is far superior compared to those reported in the work of João Damas [21]; while the dataset used for his analysis had almost 400,000 entries, ours had more than 1 million. However, the other parameters obtained similar values to those in his work.

4.3.1 Term level

A term is a whitespace-bounded sequence of characters and is part of a query, which can have one or more terms. One of the steps that we took regarding term level was the removal of occurrences of boolean operator characters, such as plus signs and other special characters that were irrelevant to our analysis.

Table 4.3: Main term level statistics overview.

Metric	**Dataset**
Number of terms	1 386 762
Number of terms (no stopwords)	1 314 351
Unique terms	72 411 (5.22%)
Never repeated terms	32 387 (2.34%)
Mean characters per term	7.03

We also used a list of stopwords for effects of comparison, as these terms may present little semantic meaning. The list used was the same as the one present in João Damas's [21] work, and it is based on the main Portuguese prepositions. This list can be found in Appendix B.1 and Table 4.3 presents an overview of some statistics regarding term analysis.

Regarding the unique terms statistic, whose percentage is 5.22%, it is typically a value that varies a lot between studies and also depends on the context, especially when we are talking about domain-specific search engines. João Damas [21] reported a value of 10.22% for unique terms in his work, Yang et al. [32] in their analysis of an electronic health record search engine reported a value of 8.24% and Efthimiadis [11], in his study to understand how greeks searched the web, reported a value of 4.89% of unique terms for queries written in Greek and other foreign languages, and 14.31% for Greek queries.

Stopwords represented around 5% of the total terms, which means that users rarely used these types of words in their search. João Damas reported a value of 2.5% for the percentage of stopwords regarding the number of total terms. Despite the fact that this metric may differ from language to language, entities' names are the same whether the search is done in the Portuguese version or other international versions. Exceptions can occur in team names, where if a team has the city in which it is located in its name, differences may appear in the query. The maximum size in terms of characters for a term was 34, and it consisted in a term that was never repeated. In João Damas's work [21], the maximum size in terms of characters for a single term was 103, which was significantly higher than ours. This term was also never repeated. Figure 4.1 shows the distribution between the number of terms and their respective number of characters (for all the terms and for all the terms excluding stopwords). We can observe that there is only a change in the number of characters for terms with a number of characters in the range [1,5]. This is due to the fact that, in our list of stopwords, the maximum length of a stopword is 5, so it is guaranteed that a term with a length superior to 5 characters will not have stopwords. The values obtained were in line with those obtained in João Damas's [21] work, which was expected.

We also looked at the most frequent terms in searches, as shown in Table 4.4, to understand what were the words that the users used more when searching. We can observe that the term *de* is second on this list and corresponds to a stopword. The terms *porto* and *benfica* could form queries without the need for any more term. The term *liga* is often used to form expressions, especially when referring to competitions. Table 4.5 shows the top 10 most frequent bigrams (i.e., two adjacent terms), where we can have a better idea of what the users searched in the platform. We

Table 4.4: Top 10 most frequent terms.

Term	Frequency	Percentage
liga	10 020	0.72%
de	9 138	0.66%
sao	8 570	0.62%
al	7 423	0.54%
joao	7 381	0.53%
benfica	6 602	0.48%
porto	5 389	0.39%
pedro	4 616	0.33%
real	4 609	0.33%
paulo	4 582	0.33%

can see, for example, that terms like *liga dos* and *copa sao* seem to be part of larger expressions. Despite the fact that the most frequent terms in our work were very similar to those observed by João Damas [21] in his work (*liga*, *de*, *sao*,*joao*,*benfica* and *porto* are also in his top 10 most frequent terms), most frequent bigrams differ a lot from those reported by João Damas [21] (*liga dos* and *real madrid* are the only matching bigrams).

4.3.2 Query level

Regarding the query analysis and taking into account the time period in which this dataset was collected, the total number of queries was far superior to those of other domain-specific search engine studies. Yang et al. [32] work was based on a dataset containing around 200,000 queries while the dataset used in Sharifpour's [26] work had around 3 million entries but in a period of 1 year. João Damas [21], in his work, used a dataset containing around 325,000 queries, within a space of 2 weeks (between 5th and 20

Of all the queries submitted to the platform, less than one-fifth of them are unique, which is an indicator that exists a similarity between users in expressing their information needs in the form

Table 4.5: Top 10 most frequent query bigrams.

Term	Frequency	Percentage
al nassr	1 502	0.37%
al na	1 155	0.28%
casa pia	1 126	0.28%
al nass	1 018	0.25%
al nas	720	0.18%
copa sao	697	0.17%
real madrid	681	0.17%
roberto martinez	643	0.16%
liga dos	534	0.13%
hugo souza	531	0.13%

Table 4.6: Main query level statistics overview.

Metric	**Dataset**
Number of queries	1 068 132
Unique queries	152 543 (14.29%)
Never repeated queries percentage	92 921 (8.70%)
Mean characters per query	7.37
Mean terms per query	1.29
Mean terms per unique query	1.19

of a query. João Damas [21] obtained a higher value, reporting a percentage of unique queries of 21.63%. In fact, our value is a little lower compared to other studies; Silverstein et al. [27] reported a value of 26.7% in their analysis of AltaVista search logs whereas Natarajan et al. [19] reported a value of 44.4% in their analysis of clinical queries in an electronic health record search utility.

When we observe the mean characters per query, we report a value of 7.37 characters, which is very close to the mean characters per term value reported before (7.03); João Damas [21] obtained similar values (8.21 for mean characters per query and 6.44 for mean characters per term). This reveals a tendency for shorter queries, usually with only one term, as Table 4.7 shows and taking also into account the mean terms per query value (1.29 for all queries; 1.19 for unique queries). We can see that almost three-quarters of the queries had only one term, which could mean that users often searched using common names, as teams and players usually have famous one-term names or abbreviations (e.g., *Cristiano Ronaldo* is often referred to as *ronaldo* or *cr7*). When entities are not so popular, one-term queries might not produce the desired results, leading the user to rewrite the query or scroll down the results to find the desired document. This little difference between the mean characters at a term and query level could be due to the fact that queries are usually longer than terms, despite the tendency for short queries. Not only that, but queries contain characters that are not considered in terms (e.g., whitespaces or operators). This was a value that was shorter

Table 4.7: Number of terms per query.

Terms	**Frequency**	**Percentage**
1	789 943	73.96%
2	247 116	23.14%
3	23 701	2.22%
4	5 624	0.53%
5	1 573	0.15%
6	137	0.01%
7	24	0.002%
8	10	0.0009%
9	2	0.00019%
10+	3	0.00028%

Table 4.8: Top 10 values for geolocation field.

Country	**Frequency**	**Percentage**
Portugal	612 500	57.34%
Brazil	348 934	32.68%
France	13 324	1.25%
Spain	11 580	1.08%
Great Britain	8 039	0.75%
Switzerland	7 013	0.66%
USA	5 050	0.47%
Italy	4 315	0.40%
Angola	4 178	0.39%
Germany	3 671	0.34%

compared to other studies, as Silverstein et al. [27] obtained an average term per query value of 2.35, while Spink et al. [28] reported a value of 2.6. As it was expected, João Damas [21] obtained a value of 1.34, a very close value to what we obtained.

Despite being a Portuguese website, the platform has expanded internationally to other countries and, therefore, collects visits and requests not only from Portuguese users but also from other parts of the world. Our dataset contained a field *geolocation* that gave us information about the country from which the query to the system was done. This information might not always be accurate, in the sense that a user could be using VPN (Virtual Private Network), but we did not expect that it would have a major influence on our analysis. Table 4.8 shows the top 10 most frequent countries to send queries to the platform, with the country being extracted from their two-letter code. The results obtained were almost the same as those reported by João Damas [21], as we did not expect to change that much.

The majority (more than half) of the requests sent to the system were, without surprise, from Portugal, with a percentage of almost 60%, followed by Brazil, France and Spain. Portugal and Brazil combined account for almost 90% of the total requests. Nowadays, it is very common to see players and managers move from Portugal to other leagues and championships, especially Brazil, Great Britain or Spain, and also the other way around. In the last few years, we´ve been watching players and managers from other European countries coming to Portugal in order to progress in their careers. Therefore, it is not a surprise to see these countries in this table. Log entries with an empty value on the geolocation field were not considered, accounting for 4.6% of the total entries.

Finally, we looked at the most frequent queries submitted to the system and compared them to the most frequent terms and bigrams, as seen before in Table 4.4 and Table 4.5, respectively. Table 4.9 shows the top 20 most frequent queries submitted to the system. The results, as expected, showed some similarities between the most used terms and most submitted queries. First of all, as we analyzed before, most of the queries are one term length; therefore, it is no surprise that some of the most frequent terms are included in the most submitted queries. Some examples are the terms *liga*, *benfica*, *porto* and *real*. There are no bigrams included in this list, as all the queries present

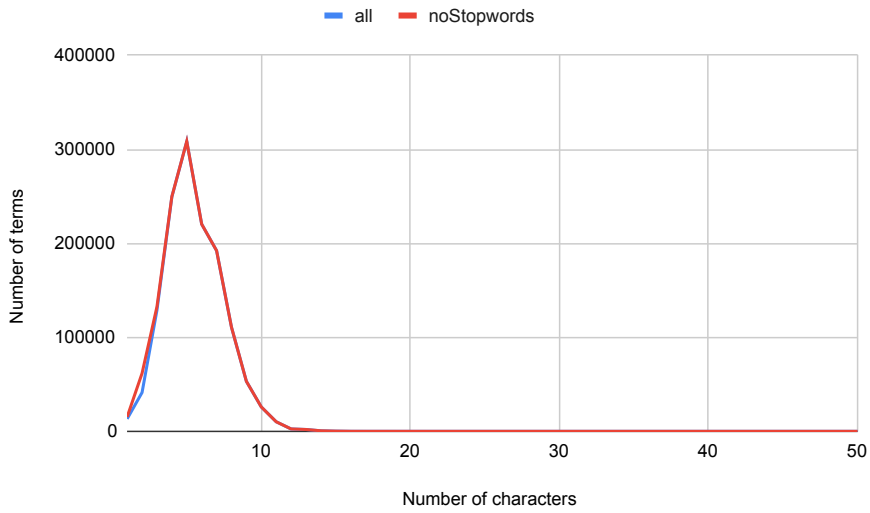

Figure 4.1: Terms per number of characters distribution.

in this table have only one term. Portuguese and Brazilian entities dominate the most frequent queries, and except for the queries [ronaldo], [messi] (appear to target players) and [liga] (target competitions), all of the other queries target teams, mostly Portuguese and Brazilian teams. The results obtained whether in the most frequent terms and in the most frequent queries, were very similar to those in João Damas's [21] work, with some minor differences. Regarding the results obtained for the most frequent bigrams, there were not major similarities with those obtained by João Damas [21], with [al nassr] (and other forms of the query) and [roberto martinez], the actual club of Cristiano Ronaldo and the current Portuguese National Team manager, respectively, present in our list.

On a session level, we made an analysis of the number of queries submitted to the system per session, shown in Table 4.10. In total, 513,326 sessions were present in the dataset, with more than two-thirds corresponding to users that performed only one query. The percentages obtained were very close to those reported by João Damas [21].

4.3.3 Click level

Click level analysis is not common and is not often seen in studies regarding domain-specific search engines. However, we present some analysis made regarding this matter, and that was also conducted by João Damas [21].

First of all, we analyzed the main entities clicked by the users upon a search request. Teams and players were the most clicked entities, accounting for 50% and 41.4%, respectively, while competitions obtained 6.2% of all clicks and managers 2.39%. João Damas [21] reported, respectively, percentages of 50.2%, 32%, 7.2% and 2%, which were very similar values to those observed by us. These results were not surprising at all since, in the majority of team sports, teams

Table 4.9: Top 20 most frequent queries.

Query	**Frequency**	**Percentage**
benfica	5 891	0.55%
braga	3 914	0.37%
porto	3 808	0.36%
ronaldo	3 544	0.33%
chelsea	3 533	0.33%
vasco	3 284	0.31%
sporting	3 210	0.30%
liga	3 121	0.29%
inter	2 734	0.26%
vitoria	2 610	0.24%
flamengo	2 561	0.24%
gremio	2 548	0.24%
psg	2 476	0.23%
messi	2 385	0.22%
arsenal	2 364	0.22%
atletico	2 280	0.21%
bahia	2 131	0.20%
real	2 126	0.20%
palmeiras	2 089	0.20%
pacos	2 058	0.19%

Table 4.10: Session length distribution.

Queries	**Sessions**	**Percentage**
1	347 147	67.63%
2	84 156	16.39%
3	32 069	6.25%
4	15 681	3.05%
5	8 945	1.74%
6	5 654	1.10%
7	3 715	0.72%
8	2 832	0.55%
9	2 133	0.42%
10+	10 994	2.14%

Table 4.11: Overall click ranking distribution.

Ranking	Number of Clicks	Percentage
1	792 423	74.19%
2	135 969	12.73%
3	54 305	5.08%
4	30 197	2.83%
5	19 748	1.85%
6	13 466	1.26%
7	8 403	0.79%
8	6 193	0.58%
9	4 149	0.39%
10	3 279	0.31%

and players are the main focus. Other entities that were clicked by users on the platform consisted of stadiums, referees and agents.

Table 4.11 shows a general overview of the overall click distribution per ranking position. The percentages obtained were almost the same as those reported by João Damas [21]. In our dataset, the lowest present ranking was the tenth, which still lies within the suggestions page range (first to tenth results); however, a user can use the magnifying glass icon to see more results, and, in that case, the ranking can be lower. The majority of the clicks consisted of a result ranked in the first position, accounting for almost three-quarters of the total clicks, which is a strong indicator that the system is capable of giving the users the result they want to see in the first position of the results list. Torres et al. [10], in their study to compare the differences in queries employed by users for general purposes and children information, reported a value of 60% of clicks in the first result for general purposes information and 40% for children-related information.

Another analysis performed at a click level was click entropy. The idea is that despite the fact that users may submit the same query to the system, the information need might not be the same. This usually happens in smaller queries for their lack of specificity. We used the same formula that João Damas [21] used in his work to calculate the entropy for each query. The formula is as follows:

$$CE(q) = -\sum_{u \in P_c(q)} p_c(u|q) * \log_{10} p_c(u|q)$$

where $P_c(q)$ is the collection of all the individual entities clicked when searching using query q and $p_c\ (u|q)$ is the percentage of clicks on entity u using query q. Table 4.12 and Table 4.13 show the top 20 highest entropies for queries at an individual entity level (we consider two different players as being two different entities, for example) and entity type level (two different players are considered a single entity - players).

As it was expected, queries with a bigger variability in clicks were shorter ones, as usually queries with more than one term are more specific, whereas queries with one term may be vague. This list was mainly composed of personal names that were expected to lead to greater variability

Table 4.12: Top 20 queries with highest entropy - individual entities.

Query	**Entropy**	**Number of Entities**
miguel	1.25	26
paulo	1.25	26
futsal	1.24	24
madeira	1.21	21
serginho	1.16	21
bruninho	1.14	16
marcao	1.14	17
andre	1.14	32
leandro	1.12	20
santiago	1.12	18
caio	1.11	17
wesley	1.10	17
diogo	1.09	28
lisboa	1.09	16
souza	1.08	15
joao pedro	1.07	16
michael	1.06	14
emerson	1.06	21
junior	1.06	15
joel	1.05	15

Table 4.13: Top 20 queries with highest entropy - entity types.

Query	**Entropy**	**Entity types**
murtosa	0.58	4
espirito santo	0.57	4
bie	0.55	4
vouzela	0.54	4
luxemburgo	0.52	4
argel	0.51	4
baroni	0.48	3
daniel russo	0.48	3
dom	0.48	3
hond	0.48	3
lapa	0.48	3
rui gual	0.48	3
zimba	0.47	3
conc	0.47	3
cristovao	0.47	3
montenegro	0.47	3
cristov	0.47	3
georgi	0.47	3
thai	0.47	3
voro	0.47	3

Table 4.14: Top 10 best scored queries.

Query	Score	Nr Clicks
benfica	5748	5891
braga	3736	3914
porto	3609	3808
chelsea	3522	3533
vasco	3214	3284
sporting	3156	3210
ronaldo	3123	3544
flamengo	2519	2561
gremio	2491	2548
psg	2458	2476

of different results, especially when these personal names are common (e.g., [miguel], [paulo], [andre]). The query [futsal], ranking third in our list, it may lead to very different results, as there are numerous competitions related to this sport. At an entity type level, the entropy levels were obviously lower, as in this case, we are considering, for example, different players as being one entity and the same applies to other entities. We can observe some queries related to places present in this table, such as [montenegro],[luxemburgo], [murtosa] or [vouzela], which are words that are often present in names of teams, stadiums or competitions. Despite the fact that our entropy values were significantly lower when compared to those reported by João Damas [21], the information needs were very common, both for entropies at an individual level and entropies at an entity level. In his work, the highest entropies (at an individual level) were seen to be related to queries containing personal names and to place-like names at an entity type level.

Finally, we wanted to evaluate the level of satisfaction of the query's results to the user who sent it to the system. As entropy does not take into account the ranking of the results clicked by the user, we wanted to use our click ranking information to measure this. For this, we used the formula applied by João Damas [21] in his work, which was adapted from the traditional TF-IDF weighting scheme [15]:

$$RFScore(q) = \sum_{r \in CR_{(q)}} Freq(q,r) * \log_{10} \frac{N}{R}$$

where $CR_{(q)}$ is the set of ranking positions where query q led to a click, *Freq(q,r)* is the number of times that q led to a click on that ranking position, and N is a constant to benefit higher rankings. In the traditional TF-IDF weighting scheme, N is the number of documents in the collection, but we chose to use the lowest ranking position clicked, which was 10. This formula benefits queries with more clicks on higher positions, and therefore we could make an analysis of the best and worst-scored queries based on this formula. The results are shown in Table 4.14 and Table 4.15.

Most of the best-scored queries are also the most frequent ones. Looking at Table 4.14, we can see that it is only composed of single terms, often used to search popular entities. We also plotted the scores for the top 100 most frequent queries, according to the formula above. Figure

Table 4.15: Top 10 worst scored queries.

Query	Score	Nr Clicks
2013-2014 portugal	0.46	1
afonso vieira	0.46	1
kader abdoul	0.46	1
marenco	0.46	1
tristao	0.46	1
wanderson carlos	0.46	1
eduardo peixoto	0.46	1
cris silva	0.46	1
cassimiro	0.46	1
bruno semedo	0.46	1

4.2 shows how the score changed between frequent queries. The values obtained were very similar to those reported by João Damas [21]. We can observe a general tendency for scores to be lower as the query becomes less frequent, but it is not linear. Some more frequent queries had lower scores compared to other queries that were not so frequent. This happens because the formula not only takes into consideration the number of clicks in a certain position, it also takes into account the position rankings; therefore, a query may lead to many clicks in a result placed in the tenth position, while other queries may lead to fewer clicks in a result placed first, which can lead to the second query having a higher score than the first one. Looking at worst-scored queries, they consisted of a set of queries that were not so frequent, leading to a low score. Looking at the query at the top of the table, we assume that the relevant result is, in this case, the Portuguese first league. Although the edition is 2013-2014, one needed to go to the competition page and then choose the corresponding edition. With manual analysis, typing the query in the platform does not return the Portuguese first league.

4.4 Summary

With an analysis of the zerozero.pt search logs, we could understand better what the users usually searched on the platform and how they interacted with the search engine. In addition, we also established differences and similarities in multiple metrics between our work and other studies, as well as the work of João Damas [21], which included the analysis of the zerozero.pt search logs from three years ago. We decided to conduct the same study to understand if there were any relevant changes or similarities between the two studies.

We preprocessed the original dataset, removing non-human activity by making use of the User-Agent string, and we performed an analysis of the search behaviour from three different perspectives: term level, query level and click level, the last one being not a common analysis on other studies due to the difficulty to obtain this information. Queries were found not to be long, usually a one-term query (almost 75% of the queries) with mean average characters per query of 7.37%.

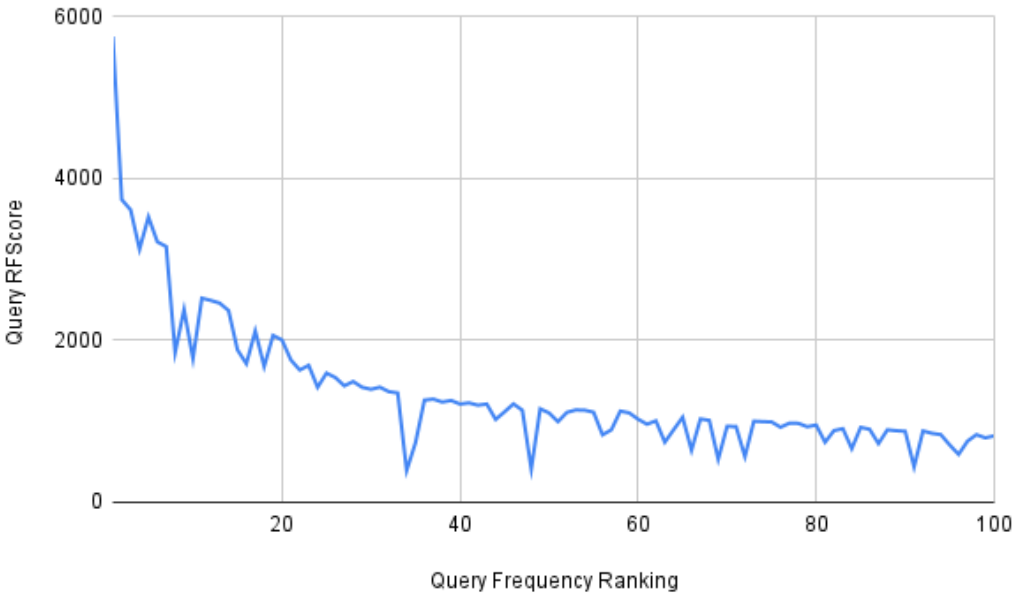

Figure 4.2: QueryRFScore for the top 100 most frequent queries.

The percentage of unique terms in our dataset (5.22%) was found to be very context-specific, especially in domain-specific search engines, as compared with other studies. Most frequent queries usually consisted in terms that were perceived as full queries. Also, observing the top 10 most frequent bigrams, we conclude that those were not as frequent as single terms.

Portuguese searchers represented almost 60% of the total searchers on the platform, followed by Brazil. Moreover, we found that a great part of the session's length was short, with the sessions that corresponded to only one query submitted to the platform accounting for almost 70% of all sessions.

Finally, at a click level, we found that the results ranked first were the most clicked ones (almost 75% of all clicks); also, the analysis of the queries with the highest entropy at an individual entity level showed that shorter queries, often one term length, were the vaguest ones. The same applies to queries with high entropy at an entity type level. We also used a formula adapted from the traditional TF-IDF weighting scheme to evaluate the queries with best and worst scores, and we observed that short queries, often used to refer to popular entities, had the best RFScore, whereas the worst ones were usually longer and less frequent, and often referring to less popular entities.

Chapter 5

Search Engine Implementation

Having analyzed how the users in the platform searched, we proposed and implemented an enhanced new search engine that could retrieve relevant results according to users' information needs. Before describing the implementation of João Damas's [21] proposal, we will first analyze the data in each collection and present the final document scheme for each entity.

5.1 Entities Overview

We decided to focus our implementation on the four main entities present in zerozero.pt, according to João Damas [21], which were also the focus of his work; those entities are Players, Managers, Competitions and Teams. Due to the platform having a large database regarding soccer and other sports, all the entities were populated with numerous fields that went beyond the purposes of identification and description (e.g., image paths). Hence, we analyzed and chose what we considered to be the most important fields that best helped to identify an entity and contained information that would potentially be used by searchers to reach these entities. In addition, we made use of the click analysis performed in Chapter 4 to focus our work on the four main entities present in the platform: Players, Managers, Competitions and Teams.

5.1.1 Players

The Players collection was the largest collection in terms of documents that we had in our work, containing more than 1 million documents. This entity represents a type of person and contained information about the player's name, their associated keywords (e.g., names that are often used to refer to a certain player), if any, the abbreviation of the player's name (99.9% of the players), the respective position on the field (e.g., goalkeeper, midfielder) and the current club of the player. These last two informations could be useful to distinguish players with the same name that played in different positions or different teams. Despite the keywords field having almost empty values in most of the entities (only 1.15% of the players had non-empty keywords), we opted to use it as it is very useful to identify players (e.g., reaching the player Cristiano Ronaldo with the query [cr7], as *cr7* is a keyword of the entity Cristiano Ronaldo). Figure 5.1 shows the players' positions

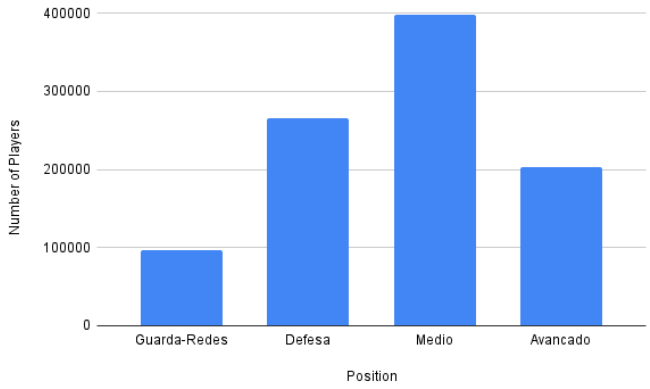

Figure 5.1: Players positions distribution.

distribution. All this information was kept, in addition to the birthdate and corresponding image of the player (these two were not included in the Player final data structure in João Damas's [21] work), for display purposes in the results of our developed search engine. Listing 5.1 shows an example document with the final structure. Our final structure differed a little from the one in João Damas's [21] work, as he includes, besides all the fields that we used in the Player structure, the previous teams that the player represented before, as well as the sub-position of the player (e.g., a defender can be a centre-back, a right-back or a left-back).

5.1.2 Managers

Managers are the persons responsible for managing the team in games. Our sample contained around 64,000 managers. Like players, this entity also represents a type of person, and the information present in those entities was very similar to the players' entities; all the managers had their full name stored, the respective name's abbreviation was also included for most of the managers, less than 1% had keywords associated (e.g., José Mourinho is often referred as "special one"), and the current team of the manager was also included on most cases (some managers might not be managing a team at the moment). We decided to keep all this information, as well as the birthdate

```
{
    "id":"player999999",
    "name":"Cristiano Ronaldo dos Santos Aveiro",
    "abbreviation":"Cristiano Ronaldo",
    "keywords":["cr7,cr9"],
    "current_team":"Al Nassr",
    "position":"Avancado",
    "year":"1985-02-05",
    "img":"https://static-img.zz.pt/jogadores/79/1579
    _20230209190330_cristiano_ronaldo_1675969410.pdf"
}
```

Listing 5.1: Player final data structure example.

```
{
    "id":"manager999999",
    "name":"Jose Mario dos Santos Mourinho Felix",
    "abbreviation":"Jose Mourinho",
    "keywords":["Special One"],
    "current_team":"Roma",
    "year":"1963-01-26",
    "img":"https://static-img.zz.pt/treinadores/002/2_jose_mourinho_1625747602.pdf"
}
```


Listing 5.2: Manager final data structure example.

and a corresponding image of the manager (again, these two were not present in the Manager final data structure in João Damas's [21] work). Listing 5.2 shows an example document with the final structure. In his work, João Damas [21] also included the former teams that the manager has coached.

5.1.3 Competitions

Competitions are tournaments in which teams participate, ending with one winning team. Our sample consisted of almost 6,000 competitions. Each competition had its full name stored, as well as an abbreviation; keywords for the competition were also stored (e.g., Premier League was one of the keywords for the English league, as it is a common denomination to refer to this competition). Due to, in most cases, the abbreviation for a competition being the same as the full name of the competition, as well as some keywords, we chose to remove abbreviations and keywords in entities in which this happened to eliminate redundancy. Therefore, for all competitions, the percentage of entities that did not contain empty values in keywords and abbreviation fields was, respectively, 57.9% and 32%. Besides all this information, we also stored the image path and the type of sport of the competition (e.g., Futebol, Basquetebol, Andebol), which were not present in João Damas's [21] work. Listing 5.3 shows an example document with the final structure. In his work, João Damas [21] included the tier of the competition (e.g., a competition may be dedicated to seniors or juveniles), which we did not include in ours.


```
{
    "id":"competition999999",
    "name":"Liga Islandesa",
    "abbreviation":"Pepsi-deild karla",
    "keywords":["Islandia,Besta deild"],
    "field_sport":"Futebol",
    "img":"https://www.zerozero.pt/img/logos/competicoes/121_competicao.jpg"
}
```


Listing 5.3: Competition final data structure example.

```
{
    "id":"team999999",
    "name":"Sporting Clube de Portugal",
    "abbreviation":"Sporting",
    "keywords":["Leoes,Leao"],
    "nicknames":["SCP,Leoes"],
    "city":"Lisboa",
    "year":"1906-07-01",
    "img":"https://static-img.zz.pt/logos/equipas/16/16_logo_sporting.pdf"
}
```

Listing 5.4: Team final data structure example.

5.1.4 Teams

A Team is an entity that plays in competitions, faces other teams and is trained by a manager. Our sample contained around 66,000 teams. Besides the full name of the team, there was also stored an abbreviation (e.g., Sport Lisboa e Benfica is often called Benfica, a more abbreviated version) and a set of nicknames in some cases (e.g., Benfica is usually called SLB or encarnados, due to their main kit colour). These designations were not available in most of the cases (16.2%) but added a vocabulary extension to our work.

There was also another field that helped to describe a team: a team's origin city. This may be helpful in removing some ambiguities (e.g., when searching for Sporting, we can be searching for Sporting Lisbon or Sporting de Braga), although in most cases, the team's city of origin is included in the name. We also kept the information regarding the foundation year and the image of the team (not included in the work of João Damas [21]). Listing 5.4 shows an example document with the final structure. João Damas [21] also included the old names that the team may have had in the past and the corresponding tier of the team (e.g., seniors, sub-19, juveniles ...), but we did not include this information in the final data structure for the teams.

Except for the year field, all fields presented in all entities used the same field type, with the Solr *multivalued* property set to True when fields stored arrays of values (keywords for Players, Managers and Competitions; keywords and nicknames for Teams). Despite the amount of predefined data types provided by Solr, none allowed querying with or without accents. Hence, we used the predefined field type *text_general* that was automatically attributed by Solr to all field types (except the year field) and we changed it to perform the filters that we wanted. Listing 5.5 shows the definition of the *text_general* field type. It uses Solr standard tokenizer that treats whitespaces as a delimiter (e.g., [one two] is transformed into [one] [two]), an ASCII folding filter to convert all characters to the corresponding ASCII representation (if preserveOriginal attribute is set to False, original token is not preserved; therefore we set it to true so we could store the original token, and the token with all characters converted to their ASCII representation) and a lowercase filter that converts all characters to lowercase. This Solr configuration was aligned to those used by João Damas [21].

```
<fieldType name="text_general" class="solr.TextField" positionIncrementGap="100"
    multiValued="true">
    <analyzer>
      <tokenizer name="standard"/>
      <filter name="asciiFolding" preserveOriginal="true" />
      <filter name="lowercase"/>
    </analyzer>
</fieldType>
```

Listing 5.5: Custom string field type definition.

5.2 Search Term Payloads

The schemas described above included one more field, *vals_dpf*, that stored terms that previous users used in their queries to reach that document, alongside a search relevance weight for that term in that particular document. This strategy was also used by João Damas [21] in his work as a complement to the overall document score. Our objective with the use of these terms was to facilitate future searches by using previous search terms that were used to reach certain documents. Given an entity e and a query term$t$, its search relevance is defined as:

$$w_{te} = f_{te} * \log_{10}(\frac{NDoc}{D_t})$$

where f_{te} is the number of searches for entity e that use term t, D_t is the number of entities searches using term t, and $NDoc$ is the total number of documents (regarding the collection). This weighting scheme allowed us to understand which terms were often used to reach certain entities, and vice versa, and therefore we could incorporate these terms and their respective score into Solr, so they could have an effect on the search results. This was obtained using a Lucene feature present in Solr called payloads[1], which consists of associating a score to individual terms. While use cases of this feature vary, our objective was to weight terms in the form of relevance confidence. Solr provided us automatically with a field type that used floating-point weights, and we used and changed it to suit our needs. We applied a whitespace tokenizer to split by whitespace, as it is how payloads are delimited, an ASCII folding filter and a lowercase filter to fulfil the same needs as the other field type. Listing 5.6 shows the corresponding field type definition based on the payload field type definition used by João Damas [2] and an example of a payload field.

5.3 Architecture

We will now describe the implementation of our search engine: the technologies used and the steps taken so we could develop the new system. The architecture 2.15 proposed by João Damas [21] was implemented, with some minor differences. While he implemented a solution in which one query was propagated to all four collections, and then the results were merged and had their scores normalized, we created five cores: one for the Players entity, other for the Managers entity,

[1] https://lucidworks.com/post/solr-payloads/

```
<fieldType name="delimited_payloads_float" class="solr.TextField" indexed="true"
    stored="false">
    <analyzer>
      <tokenizer name="whitespace"/>
      <filter name="asciiFolding" preserveOriginal="true" />
      <filter name="lowercase"/>
      <filter encoder="float" name="delimitedPayload"/>
    </analyzer>
</fieldType>

{
    'vals_dpf' : 'term1|23.54 term2|12.10 term3|2.0'
}
```

Listing 5.6: Custom payload field type definition and example.

other for the Competitions, other for the Teams and the last one contained all documents from the four distinct entities. As said before, this strategy of having a single core containing all the documents from each entity was suggested by zerozero.pt. Later, we will compare both the single core with all the documents and the strategy adopted by João Damas [21]. The application that we developed and that will be described later offered the possibility to choose if the results come from the single core containing all the documents from all the entities present or if they come from all four collections individually.

5.3.1 Application

Alongside using Apache Solr, we also developed an application that allowed users to query the system and see the corresponding results. The main technology used was React JS for the front end, complemented by the framework Flask[2] for communication with the Solr server. Whenever a user made a query, the Flask server would build the request object using SolrClient [18], which was then sent to Solr. Afterwards, the Solr server would send the matching documents to the Flask server, which would merge them and normalize their scores (in the case of requesting documents to each core individually) and send them to the user.

Upon starting the application, the main page allows the user to submit a query. An input text box is present so that the user can type the desired query, and then click on the search button located on the right side of the box. After the user clicks on the search button, the Solr server sends the matching documents to the user via the Flask server, which are then shown. The user then has the possibility to choose if he wants to see the results coming from the single core that contains all the documents from all the entities (All (1 Core)), request all the cores individually for the respective matching documents (All (4 Cores)) and it also has the possibility to see results from each core individually (Players, Managers, Competitions or Teams). Figure 5.6 shows an example of a query submitted to the server and the presented results, as well as the available filters. For each filter, the application shows the user the number of results retrieved; if none, the system sends the user a message saying that are no results to show.

[2] https://flask.palletsprojects.com/en/2.3.x/

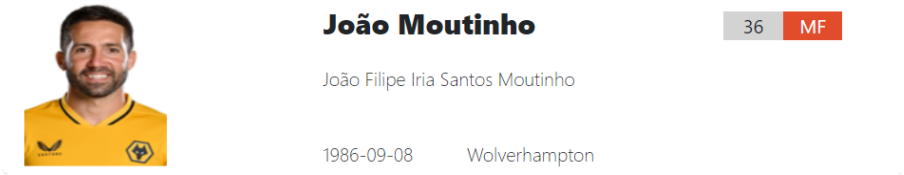

Figure 5.2: Card display for the entity Player.

Each result had its display changed according to the entity that represented it. Figures 5.2, 5.3, 5.4, and 5.5 show the display of the cards for each entity. For the Player entity, the card contained the abbreviation of the player's name in bold and the full name below it. At the bottom of the card, it is shown the birthdate of the player and his current team right after it. On the right side of the abbreviation of the player's name, it is shown the current age of the player and his position on the pitch (blue background for goalkeeper, light orange for defence, dark orange for midfielder and red for forward). Manager card is exactly the same as the Player card, except it does not contain the position on the pitch, as managers do not have positions. The Competition card contains the abbreviation of the competition on the top, in bold text, and the full name of the competition below it; at the bottom, it is shown the field of sport of the competition. Finally, the Team card has, at the top, the abbreviation of the team in bold text, followed by its full name, and, at the bottom, we have the origin city of the team, as well as the year of foundation.

5.3.2 Pagination

For each filter, our platform only showed ten results to prevent the user from scrolling infinitely. The user could navigate through pages to see results located way below the top 10 results for easier navigation. Figures 5.7 and 5.8 show an example of pagination.

5.3.3 Spellcheck

We also made use of a Solr feature called spellcheck[3]. This feature consists in giving the user query suggestions having as background the current query, based on similar terms. In our application, if a user misspelt a term, the application sends the user a message with an alternative query (e.g, if a user submitted the query [sport lisboa e benfca], a message would be shown below the input

[3] https://solr.apache.org/guide/solr/latest/query-guide/spell-checking.html

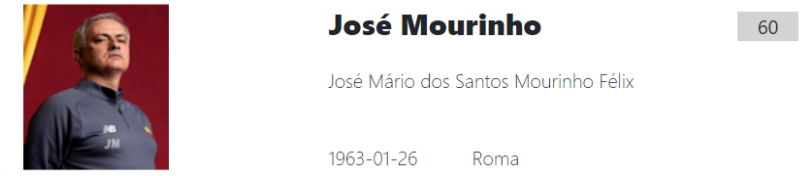

Figure 5.3: Card display for the entity Manager.

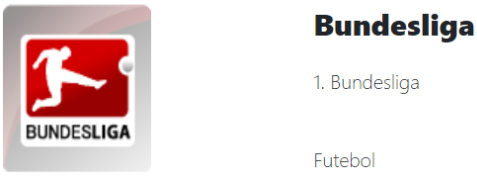

Figure 5.4: Card display for the entity Competition.

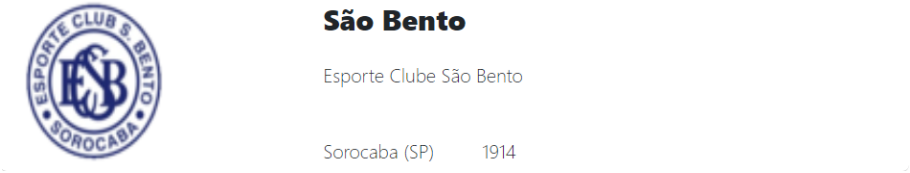

Figure 5.5: Card display for the entity Team.

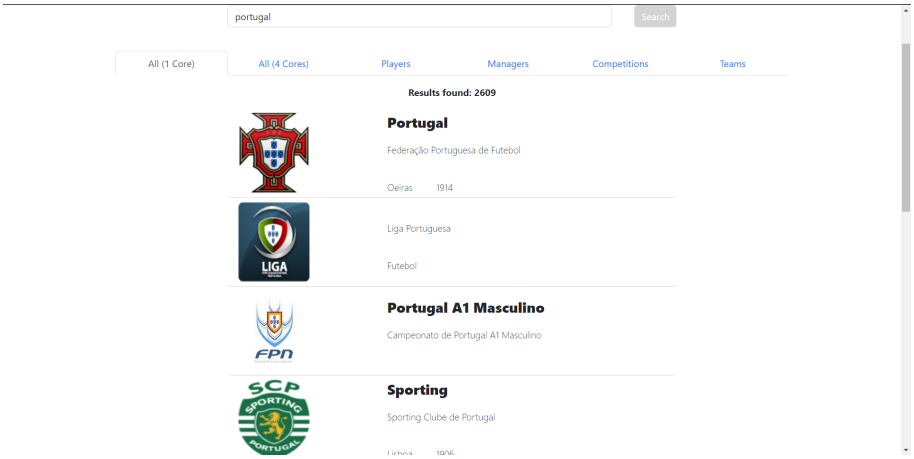

Figure 5.6: Query submitted to the server, and respective results.

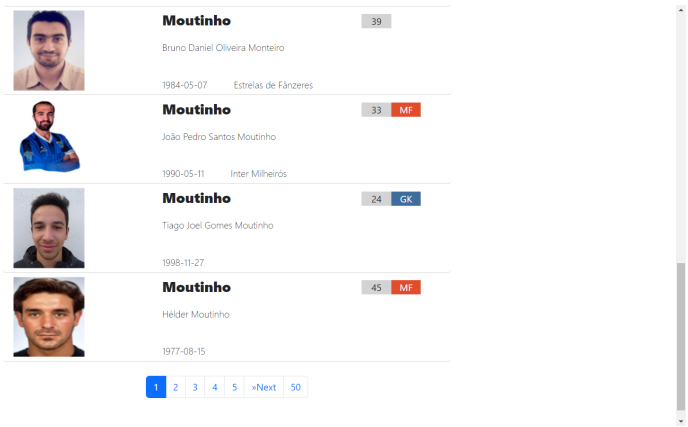

Figure 5.7: First results page for the query [moutinho].

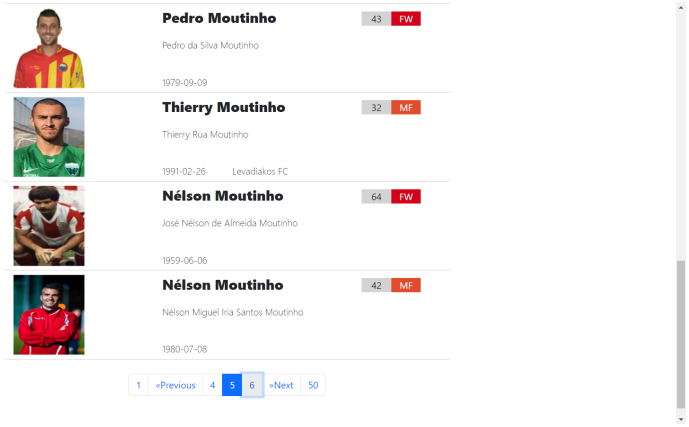

Figure 5.8: Fifth results page for the query [moutinho].

text box with an alternative query). Figures 5.9 and 5.10 show an example of this feature applied in our application.

5.3.4 Querying collections

The process of constructing the query request, receiving the documents from Solr via Flask and calculating the final score for them (with payloads incorporated) was implemented having the work of João Damas [21] as background. The request to the Solr server was constructed in the Flask[4] server. There, we defined the query parser and some general options. We used Solr's eDismax[5] query parser due to the range of capabilities provided. We also wanted to use payloads as complementary information to the document's final score alongside the Solr standard document relevance, and not as a total replacement. Hence, we used eDismax to calculate the standard document relevance and used the payload function to calculate the score for each individual term. The payload function takes a term and a payload field and returns the score associated with the term, or 0 if not applicable. When the user performs a search in the application, the query is split using a whitespace delimiter and a custom field per term is created. Each term had its own respective sequential field (payload_0, payload_1, ..., payload_n+1) that searched the *vals_dpf* field. Despite payloads being used for individual terms, we also made an extra payload field for the entire query. Therefore, in the *vals_dpf* field, we had not only individual terms but also terms

[4] https://flask.palletsprojects.com/en/2.3.x/
[5] https://solr.apache.org/guide/6_6/the-extended-dismax-query-parser.html

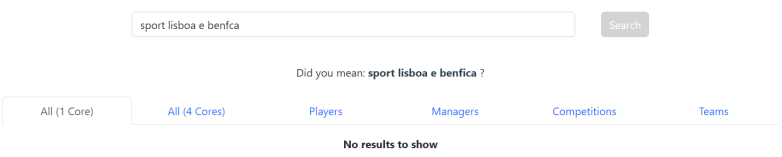

Figure 5.9: Suggestions for the query [sport lisboa e benfca].

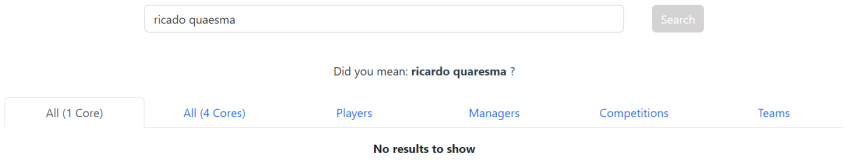

Figure 5.10: Suggestions for the query [ricado quaesma].

that were originally separated by whitespace and are now connected with a '-'. When we split the user's query by whitespace, we also add to the list the entire query (if it has more than one term) connected with a '-' in the place of the whitespaces. The point of this approach is to value documents that were reached using not only one term but more than one. Therefore, if we kept all the query as a payload field, we can match it to a payload in a document that contains all the query, and not just individual terms. Listing 5.7 shows a general overview of this process.

We also added boosts to the fields so that different fields had different weights according to their relevance. The boosts for each entity were the same used by João Damas [21], while the boosts used for the single core containing all the documents were a product of experimentation. Table 5.1 shows the field boost definition for each entity type. We also limited phrase query match to a slop of 3 terms (*ps* parameter) and configured Solr to retrieve all the matching documents.

5.3.5 Payload Scores

Upon querying a collection, each document had a field *score* with Solr standard relevance formula result, as well as payload fields, one for each term as well as one extra for the complete query, as explained above. To merge these two scores, we adopted the strategy followed by João Damas [21] to calculate the final score for each document, with payloads incorporated. He proposed four strategies to incorporate payloads in the final score. However, as in the evaluation phase, he concluded that the *Prod* one was the one that gave the best results, both in the set of most frequent queries and queries with the highest entropy; it was this strategy that was used by us to incorporate payloads in the final score of the document:

```
query_terms = [term.strip() for term in query.split(' ') if len(term) > 0]
query_terms.append(query.replace(" ","-"))

response = solr.query(core,{
    ...,
    "fl":"*,score," + ',',join(['payload_{}:payload(vals_dpf,{},0.0,max)'.format(
    idx, term) for idx, term in enumerate(query_terms)]),
    ...
})
```

Listing 5.7: Custom term payload fields for query request.

Table 5.1: Field boost according to each entity.

Entity	**Field boosts**
Single core with all entities	name^10 abbreviation^10 keywords^5 current_team^5 position^5 city^5 vals_dpf
Player	name^10 abbreviation^7.5 keywords^7.5 current_team^5 position^5 vals_dpf
Manager	name^10 abbreviation^7.5 keywords^7.5 current_team^5 vals_dpf
Competition	name^10 abbreviation^10 keywords^5 vals_dpf
Team	name^10 abbreviation^10 keywords^7.5 nicknames^10 city^5 vals_dpf

- ***Prod* strategy** - Document's final score linearly scaled with the payload scores

$$Score = RelS * \prod_{i=0}^{n} PL_i[PL_i \neq 0]$$

5.3.6 Results merging

When we queried the results from each core individually, we had to merge the results coming from each collection and normalize their scores so that they could be compared between different collections. The algorithm we used was CORI [7], which defines, for a document D coming from collection C, its normalized score as:

$$Score = S_D * \frac{1 + 0.4 * S_C}{1.4}$$

where S_D is the original document score, with the payloads score already incorporated, and S_C is the collection's score. A collection's score should reflect the retrieved documents' overall importance for the final results list. Hawking et al. [14] used an LMS (using result Length to calculate Merging Score [23]) strategy to calculate a collection's score, as well as João Damas [21], and we also decided to use it in our work. A collection's LMS score is defined as:

$$LMS_C = \log_{10}(1 + \frac{|R_C| * K}{\sum_{i=1}^{n} |R_i|})$$

where R_i is the number of documents retrieved by collection i for the query, n is the number of collections and K is a scaling constant. We used the same value that was used in the work, K = 600. This strategy might overvalue a collection's worth just because it retrieved more documents, even if most part of them do not have much relevance to the current information need. In our work, terms that could overlap between collections were not expected to be common (except for the cases of Players and Managers), so a larger result set could contain the correct entity type that the user was looking for. However, if it was not the case, payloads could have an effect on the

final scores of the documents and therefore invert the effects of the LMS in the case of a wrongful judgement.

5.4 Summary

Our implementation of the enhanced search engine started with the analysis of the main entities in zerozero.pt and what were the main fields present for each entity. We then projected a prototype with an Apache Solr server that contained five cores, one for each entity and one extra that contained all the documents from all the entities. Then, using React JS, we created an application in which one could make a query and see the matching documents. This process involved a Flask server that created the request object using SolrClient and then sent it to Solr. It also normalized and merged the results that came from different collections. To store previous search terms, we used search term payloads, which allowed individual term weighing. The term score reflects how often the term is used to reach the document. For merging and normalization, we used the CORI algorithm, which uses a collection's returned result set size as a metric for its quality.

Chapter 6

Results Evaluation

With the new search engine implemented, we evaluated it to analyze if it had the expected quality and was capable of fulfilling its purpose. For this, we used two different sets of queries, each one with different characteristics, and we evaluated each system (zerozero.pt current search engine, our search engine with one core containing all the documents and the strategy adopted by João Damas [21]; a query is sent to each core entity individually). Both sets of queries were extracted from the search logs that we collected and analyzed in Chapter 4. To understand what was the correct answer for each query, we made use of click information, just like João Damas [21] did.

We will first describe what were the query sets used, and what distinguished them, what evaluation metrics were used and finally, we will present the results obtained for each system and discuss them.

6.1 Queries

The data that we used to evaluate the three systems consisted of two distinct sets of queries so that we could analyze the search engine's behaviour under different scenarios. The first set of queries, which we collected from the search logs, consisted of the top 200 most frequent queries submitted to the system. Besides that, we identified all the results clicked as a result of that search, and we kept only the most clicked one alongside the number of clicks it received. This was the approach followed by João Damas [21], as he reported that, for his set of frequent queries, the average click share for the top results was around 85%. In fact, this value was in line with the one obtained in our work, with the average click share for the top results achieving a value of 84.11%. With this observation, we decided, just like João Damas did in his work [21], to assume that the top clicked result for a query (regarding this set) was, in fact, the correct answer and the result that the user was looking for. Table 6.1 shows the distribution of click share for the top clicked results.

The second set consisted of the top 200 queries with highest entropy at an individual entity level. In Chapter 4, we had already identified the top 20 queries with highest entropy, both at an individual entity and entity type level. These types of interrogations are the ones that produce higher variability in clicked results. Different entities were considered the correct answer depending on

Table 6.1: Click share values for top clicked results (Frequent Query Set).

Top Result Click Share	Number of queries
90% - 100%	113
80 % - 90%	34
70% - 80%	10
60% - 70%	16
50% - 60%	10
40% - 50%	7
30% - 40%	9
20% - 30%	1

the user who was searching, as there was not a high average click share for the top clicked results; in our work, the average click share for this set of queries, regarding the top clicked results, was around 46%, with João Damas [21] obtaining 37%. Therefore, for this set of queries, we decided to consider all the results clicked as potential correct answers. Because there was a great variety of different clicked results for these interrogations, it was not possible to consider just one correct answer like in the previous set. Table 6.2 shows the distribution of click share for the top clicked results in this set of queries.

6.2 Metrics

To evaluate the systems, we used different well-established metrics for this purpose. All the metrics that we used to evaluate each system were the same used in the work of João Damas [21].

Mean Reciprocal Ranking (MRR) [8] calculates the reciprocal of the rank of the first relevant result retrieved by the system; if the first relevant document was the first to be retrieved, RR is 1, whereas if it was the second, RR is 0.5 and so on. MRR calculates the average reciprocal rank across all queries. For a set of queries Q, the Mean Reciprocal Rank is defined as follows:

$$MRR = \frac{1}{Q}\sum_{i=1}^{Q}\frac{1}{rank_{\mathrm{i}}}$$

Table 6.2: Click share values for top clicked results (Entropy Query Set).

Top Result Click Share	Number of queries
70% - 80%	8
60 % - 70%	40
50% - 60%	50
40% - 50%	42
30% - 40%	34
20% - 30%	21
10% - 20%	5

We also used a metric used by João Damas [21] and first presented by Walter Underwood [31], who decided to expand the MRR formula and add the click-through frequency information to it so that each reciprocal rank is weighted by a query's click frequency. A query's *weighted Reciprocal Rank* is calculated as follows:

$$wRR(q,r) = RR(r) * ClickFrequency(q)$$

This metric, just like the above, is an average across queries, but this time weighted by the query's click frequency; the denominator is the total number of clicks. Therefore, this metric benefits frequent clicked results in higher positions.

Another common metric that is frequently used to evaluate search systems is the *Mean Average Precision* (MAP). It is a metric that consists of the average of another metric applied at a query level: the *Average Precision* [6] (AP). For a query q, its' average precision is defined as the average of the precision values achieved for the set of the top N documents each time a new relevant document D is retrieved. For example, for a certain query, if the relevant results were positioned in the 1st, 5th and 8th positions, the AP for this query would be $\frac{(1/1)+(2/5)+(3/8)}{3} \approx 0.591(6)$. For a unique correct result, the average precision (AP) and the reciprocal rank (RR) are exactly the same; hence MAP and MRR will be equal.

The final metric that we used for evaluation purposes was the *Sucess (at N)*, an evaluation criteria presented by Zhu et al. [33] and also used by João Damas [21], which verifies the presence of correct answers in the top results. This metric consists of the percentage of queries which had correct answers in the top N results. Both Zhu et al. [33] and João Damas [21] used N = {1,5}, so we decided to replicate this choice.

6.3 Results

We will now present and discuss the results obtained for each query set. It is important to notice that all metrics were applied considering only the top 10 results. The exception to this is, obviously, the *Success (at N)* metric, where we will use N = {1,5}.

For our system, we decided to use the *Prod* strategy (where the document's final score is linearly scaled with the payload scores) from the work of João Damas [21], as it was the strategy that achieved best results, both for the frequent query set and the entropy query set. We will also compare, within our system, the strategies in which we have one core with all the documents and the strategy used by João Damas [21].

6.3.1 Frequent Query Set

The evaluation metric results for the frequent query set are shown in Table 6.3. Despite the fact that zerozero.pt current search system achieved good results, our system performed better than the baseline (zerozero) in every aspect. We can observe that, for the *Success@5* metric, our system was able to retrieve a relevant result in the top 5 in almost every case. The results that we

Table 6.3: Evaluation metrics results for the Frequent Query Set.

Strategy	MRR@10	wMRR@10	Success@1	Success@5
zerozero	0.8425	0.7369	0.765	0.955
Prod (1 Core)	0.9733	0.8273	0.955	0.995
Prod (4 Cores)	0.9317	0.8046	0.92	0.945

obtained for our system, for this set of queries, were not seen in João Damas's work [21], as his system achieved almost the same results (a small advantage to the baseline) as the system from zerozero.pt. We can also observe that the single-core strategy achieved better results than the one adopted by João Damas [21], achieving better results than the other strategy regarding all metrics. After manual analysis, we saw cases in which the relevant result for the single-core strategy was located in the first position, whilst in the other strategy, the result was not even found in the top 10 results. This could be due to the fact that the relevant result was present in a collection with few results returned; therefore, due to the normalization of the document's scores before merging, which might overvalue a collection's worth just for returning a larger set of documents, the score for that particular document was negatively affected.

Finally, we compared both systems by looking at the individual values for each query in our set. Similar to João Damas [21], we chose to use *Average Precision* (AP) as the metric to establish this comparison. For this set, the values obtained for the AP are necessarily the same as those obtained for RR, as we assume that there is only one correct answer. Figure 6.1 shows the values of AP for each query in descendent order. Despite having achieved good results, our system was able to outstand the one from zerozero.pt, as ours had a slower rate of descent for lowest scoring queries.

6.3.2 Entropy Query Set

The evaluation metrics results for the entropy query set are shown in Table 6.4. As already observed in the work of João Damas [21], our system performed significantly better than the baseline

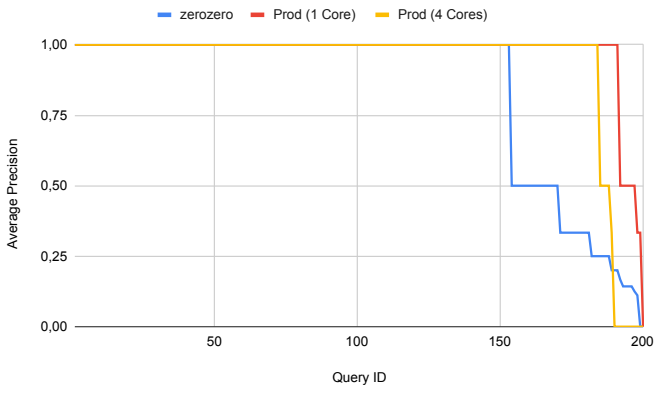

Figure 6.1: AP values per query in descendent order (Frequent query set).

Table 6.4: Evaluation metrics results for the Entropy Query Set.

Strategy	**MRR@10**	**wMRR@10**	**MAP@10**	**Success@1**	**Success@5**
zerozero	0.8153	0.6245	0.7424	0.68	0.99
Prod (1 Core)	0.9125	0.8101	0.8845	0.85	0.975
Prod (4 Cores)	0.9175	0.7208	0.8865	0.86	0.975

(expectation for the *Success@5* metric, which had similar results both in zerozero.pt system and ours). As expected, most of the metrics had lower values when compared to the frequent query set, as it was harder to correctly serve this kind of queries. For this set of queries, the strategy adopted in João Damas's work [21] achieved better results, despite not significantly. The exceptions are for the *weighted Reciprocal Rank* and *Success@5* (same result as the other strategy) metrics.

Finally, similar to what we did in the frequent query set, we measured the AP values for each query in this set. Results are shown in Figure 6.2.

In this case, it was more noticeable the difference between the two systems, reflecting the enhancement obtained by our system when evaluating this query set. There was a sudden descent for our solution towards the worst performing queries; however, for most of the queries, our system achieved better AP values compared to zerozero.pt system, whose AP values started to decrease even before the 50^{th} query.

Overall, our solution appeared to produce better results, both for the set of most frequent queries and the set of queries with highest entropy values.

We also looked at the top 20 queries with highest difference (both positive and negative) in performance between our (for both strategies) and zerozero.pt current search engine. Tables 6.5, 6.6, 6.7 and 6.8 show the results obtained. A positive value indicates an improvement, while a negative value indicates a decrease in performance.

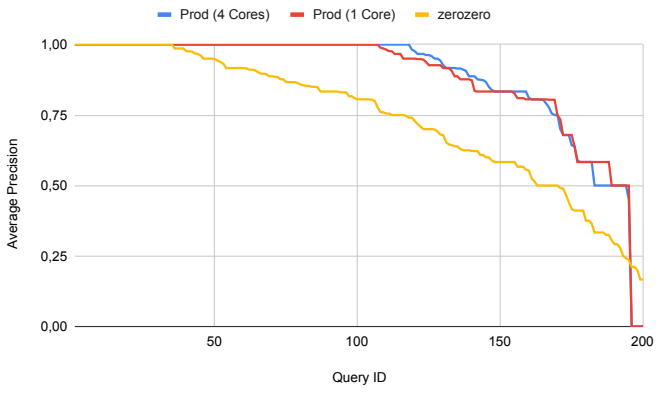

Figure 6.2: AP values per query in descendent order (Entropy query set).

Table 6.5: Entropy queries with largest positive AP difference against baseline (zerozero.pt) for the one-core strategy.

Query	**AP Difference**
capi	0,8333
con	0,8
hong	0,75
espirito santo	0,7083
roth	0,7083
timo	0,6944
barcelos	0,675
fatih	0,6666
voro	0,625
maranha	0,6222
el salvador	0,5972
indi	0,5833
morais	0,5261
tocantins	0,5088
rabello	0,5083
bahre	0,5
domin	0,5
muri	0,475
sergi	0,4655
leto	0,4611

Table 6.6: Entropy queries with largest negative AP difference against baseline (zerozero.pt) for the one-core strategy.

Query	**AP Difference**
sao feli	-1
roger mach	-0,8333
sao felix	-0,8333
rui gual	-0,5833
costa ri	-0,5
cote	-0,4166
filipi	-0,4166
pla	-0,4166
nao	-0,4166
nicara	-0,25
peso da regua	-0,25
rock	-0,2222
portal	-0,1958
germ	-0,1944
anti	-0,1944
vojvo	-0,1944
paiva	-0,1875
gree	-0,1666
vouzela	-0,1388
vale de cambra	-0,1375

Table 6.7: Entropy queries with largest positive AP difference against baseline (zerozero.pt) for João Damas' [21] strategy.

Query	**AP Difference**
capi	0,8333
con	0,8
maranha	0,7888
vaticano	0,7571
hong	0,75
espirito santo	0,7083
roth	0,7083
timo	0,6944
barcelos	0,675
fatih	0,6666
indi	0,6666
voro	0,625
lapa	0,6222
jup	0,5833
sab	0,5333
morais	0,5261
el salvador	0,5138
leto	0,5111
tocantins	0,5088
rabello	0,5083

Table 6.8: Entropy queries with largest negative AP difference against baseline (zerozero.pt) for João Damas' [21] strategy.

Query	**AP Difference**
sao feli	-1
sao felix	-0,8333
roger mach	-0,8333
rui gual	-0,5833
nao	-0,5
pla	-0,5
filipi	-0,5
cote	-0,5
costa ri	-0,5
marroco	-0,3571
rock	-0,3055
afe	-0,3
peso da regua	-0,25
germ	-0,2444
zag	-0,1958
vojvo	-0,1944
paiva	-0,1875
portal	-0,1666
anti	-0,1666
vouzela	-0,1388

6.4 Summary

The process of evaluation consisted of collecting two sets of queries with different characteristics and submitting them both to zerozero.pt search system and ours. Within our own search system, we wanted to compare two different strategies: the one in which we have one core with all the documents from all entities and the one adopted by João Damas [21] in his study, which involves sending a user query to all cores separately and then normalizing the document's scores before merging them. For this purpose, we used a query set which consisted of the top 200 most frequent queries submitted to the system from the search logs that we analyzed in Chapter 4 and the other one consisted of the top 200 queries with highest entropy at an individual entity level. To annotate queries with the correct answer, we made use of clicks as a relevance proxy. For the frequent query set, we assumed that there was only one correct answer (the most clicked result) since they always had a high click share; in the work of João Damas [21], a value of 85% was reported while in ours we obtained a value of 84.11%. For the entropy query set, the number of results clicked varied considerably, and no result had a large click share; João Damas [21] achieved a value of 36.9% for the average click share, while we obtained a value of 46%. Therefore, for this set, we assumed that all the results clicked for a query were potentially correct answers. For the purposes of evaluation, we used well-established metrics that were also used by João Damas [21]: MAP, MRR, a weighted version of MRR which included the click frequency of the result and the total number of clicks as the denominator and Success. To evaluate our system, we opted to use the *Prod* strategy proposed by João Damas [21], as it was the one that obtained the best results, both for the frequent and entropy query set.

The results showed that our system was able to outstand the one from zerozero.pt, in both query sets. For the frequent one, zerozero.pt search system achieved good results, but in the entropy set, we observed a large difference in AP values between the zerozero.pt system and ours, with our system being able to bring improvements for more ambiguous queries. We also observed a better performance from the one-core strategy in the frequent query set compared to the strategy adopted by João Damas [21] but on the other hand the second strategy, despite not significantly, achieved better results for most of the metric in the entropy query set.

In summary, the results obtained suggested that our search engine implementation based on the proposals of João Damas [21] fulfilled its expectations and was able to outstand the zerozero.pt current search engine in both query sets.

Chapter 7

Conclusions

In this work, we aimed to solve a problem in the search engine of a Portuguese sports website, zerozero.pt. Because we were in a domain-specific context, where the user base is more restricted, as well as the indexed collections, we needed to have solutions adapted to this special context. The solutions were described with detail and presented in the work of João Damas [21], so we wanted to implement a search engine based on his proposals and analyze if the final system could surpass the current search system from zerozero.pt.

We started by conducting the same study that João Damas [21] did in his work to observe if there were any major changes in the user's search patterns. Hence, we conducted a QLA for this matter. We made an analysis at a term, query and click level. Click level analysis was not often found in other studies, but it was conducted by João Damas [21] in his work, so we decided to re-conduct it again. In the process of removing non-human interaction from our search logs, we observed that non-human entries were found to be lower when compared to other studies in other domains. Also, we found that users usually searched using only one term and that each search session was often associated with just one query submitted to the system. Query analysis showed that queries usually consisted of a single term, having less than eight characters on average. Moreover, Portuguese and Brazilian searchers accounted for around 90% of all the searches submitted to the platform. At click level, we observed that most clicks were on the first position of the list, accounting for almost 75% of all clicks. Also, the entropy analysis that was conducted by João Damas [21] showed that shorter queries, usually referring to single names, were the most ambiguous.

The process of implementing the enhanced search engine was based on the strategies and proposals from João Damas's [21] work. We used Apache Solr for that purpose, as it was the technology used by João Damas [21] and the one demanded by zerozero.pt collaborators. We created five cores, one for each main entity (according to João Damas [21]): Players, Managers, Competitions and Teams. We created one extra core as a request from zerozero.pt that contained all the documents from all the entities present in our work. This strategy was later put into evaluation to compare if the results retrieved using this strategy differed from João Damas's [21] strategy (query sent to each core individually). Documents contained different fields according to the entity that they were representing. Also, we decided to adopt the strategy from João Damas [21]

and index previous search terms for a given document. This was obtained using a Lucene feature called payloads[1]. Each term had an associated score that represented the relevance of the term for that document, i.e., how often that term was used to reach that document. These scores were then incorporated into the document's score when a match occurred. To build our prototype, we created a Flask server that created the query request to be sent to Solr, making use of the SolrClient [18] python library, and created a web app using React JS that allowed one to submit a query and see the retrieved results. Finally, using Solr's spellcheck feature[2], we were able to suggest a query to the user when the user misspelt a term in the query.

The evaluation process consisted of collecting two sets of queries with different characteristics and submitting them to both zerozero.pt search system and ours (using our strategy and João Damas [21] strategy). For our system, we used the *Prod* strategy from João Damas's [21] work, as it was the one that obtained the best results for each query set. The first query set used consisted of the top 200 most frequent queries submitted to the system, whereas the second consisted of the top 200 queries with highest entropy at an individual entity level. To annotate queries with correct answers, we used click information as a relevance proxy. Several IR metrics were used for the system's evaluation, including MAP, MRR, a weighted modified version of MRR and Success.

Results obtained showed that zerozero.pt current search system achieved good results for the frequent query set, but our system was able to outstand it. However, for more ambiguous queries, our system achieved noticeably better results than the one from zerozero.pt. When comparing both strategies, the single-core strategy obtained better results in all evaluation metrics for the frequent query set, in contrast to the entropy query set in which the strategy adopted by João Damas [21] achieved better results for most of the evaluation metrics.

7.1 Main contributions

The main contributions of our work consist of the search log analysis performed in Chapter 4 and the implementation of the search engine based on the work of João Damas [21]. We replicated the study conducted by João Damas [21] regarding QLA, performing a click level analysis that is not often found in other studies. Also, we tried to understand what were the common search patterns and the unique characteristics of this search engine.

The implementation using the strategies proposed by João Damas [21], mainly the use of previous search terms as a complement to the document's final score, was proved to have better results, as shown in Chapter 6, than the current search engine of zerozero.pt. The adoption of two different strategies within our own search system allowed us to compare them and observe which one was able to produce better results. While for the frequent query set, the strategy which consisted of a single core achieved better results, it was the strategy adopted by João Damas [21] in his work that performed better.

[1] https://lucidworks.com/post/solr-payloads/
[2] https://solr.apache.org/guide/solr/latest/query-guide/spell-checking.html

7.2 Future work

Regarding future work that could be done, we highlight two points. The first one in respect to the spellcheck feature that we implemented in our platform, where we suggest to the user an alternative query in case of a misspelt term in the original query from the user. However, despite the suggestion, the user still cannot see any results for the original query. One interesting feature that could be added in the future is the automatic modification of the query, and not just a suggestion shown to the user, so that the user could see the results for the query despite a typo in a term (or more), without having the need to rewrite the query again.

Regarding evaluation, it would be interesting to deploy the solution to production, as it would have access to full collections, and test it against a baseline, knowing that both systems would operate under the same conditions.

References

[1] Iso.org ISO 3166. https://www.iso.org/iso-3166-country-codes.html, 2023. [Online; accessed February 2023].

[2] James Allan. Boolean retrieval. https://www.khoury.northeastern.edu/home/jaa/IS4200.10X1/Handouts/boolean.pdf, 2022. [Online; accessed December 2022].

[3] Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[4] Formula Bayes' Theorem: What it is and Examples. https://www.investopedia.com/terms/b/bayes-theorem.asp, 2023. [Online; accessed November 2022].

[5] David Bell and Ian Ruthven. Searcher's assessments of task complexity for web searching. volume 2997, pages 57–71, 04 2004.

[6] Chris Buckley and Ellen M. Voorhees. Evaluating evaluation measure stability. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '00, page 33–40, New York, NY, USA, 2000. Association for Computing Machinery.

[7] James P. Callan, Zhihong Lu, and W. Bruce Croft. Searching distributed collections with inference networks. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '95, page 21–28, New York, NY, USA, 1995. Association for Computing Machinery.

[8] Nick Craswell. *Mean Reciprocal Rank*, pages 1703–1703. Springer US, Boston, MA, 2009.

[9] DanBrown. csvquote: smart and simple csv processing on the command line. https://github.com/dbro/csvquote, 2022. [Online; accessed February 2023].

[10] Sergio Duarte Torres, Djoerd Hiemstra, and Pavel Serdyukov. Query log analysis in the context of information retrieval for children. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, page 847–848, New York, NY, USA, 2010. Association for Computing Machinery.

[11] Efthimis N. Efthimiadis. How do greeks search the web? a query log analysis study. In *Proceedings of the 2nd ACM Workshop on Improving Non English Web Searching*, iNEWS '08, page 81–84, New York, NY, USA, 2008. Association for Computing Machinery.

[12] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, Filip Radlinski, and Geri Gay. Evaluating the accuracy of implicit feedback from clicks and query reformulations in web search. *ACM Trans. Inf. Syst.*, 25(2):7–es, apr 2007.

[13] H. Kaur and V. Gupta. Indexing process insight and evaluation. *2016 International Conference on Inventive Computation Technologies (ICICT)*, Coimbatore, India, 2016, pp. 1-5, doi: 10.1109/INVENTIVE.2016.7830087.

[14] PengFei (Vincent) Li, Paul Thomas, and David Hawking. Merging algorithms for enterprise search. In *Proceedings of the 18th Australasian Document Computing Symposium*, ADCS '13, page 42–49, New York, NY, USA, 2013. Association for Computing Machinery.

[15] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, UK, 2008.

[16] Gary Marchionini. Exploratory search: From finding to understanding. *Commun. ACM*, 49(4):41–46, apr 2006.

[17] Web Information Retrieval | Vector Space Model. https://www.geeksforgeeks.org/web-information-retrieval-vector-space-model/, 2023. [Online; accessed November 2022].

[18] moonlitesolutions. SolrClient. https://github.com/moonlitesolutions/SolrClient, 2019. [Online; accessed March 2023].

[19] Karthik Natarajan, Daniel Stein, Samat Jain, and Noémie Elhadad. An analysis of clinical queries in an electronic health record search utility. *International Journal of Medical Informatics*, 79(7):515–522, 2010.

[20] Home - Database of Databases. https://dbdb.io, 2023. [Online; accessed December 2022].

[21] J. Paulo Madureira Damas. Building a Domain-Specific Search Engine that Explores Football-Related Search Patterns. Master's thesis, Faculdade de Engenharia da Universidade do Porto (FEUP), 2020.

[22] DB-Engines Ranking popularity ranking of search engines. https://db-engines.com/en/ranking/search+engine, 2023. [Online; accessed November 2022].

[23] Yves Rasolofo, Faïza Abbaci, and Jacques Savoy. Approaches to collection selection and results merging for distributed information retrieval. In *Proceedings of the Tenth International Conference on Information and Knowledge Management*, CIKM '01, page 191–198, New York, NY, USA, 2001. Association for Computing Machinery.

[24] Jean-François Rouet. What was I looking for? The influence of task specificity and prior knowledge on students' search strategies in hypertext. *Interacting with Computers*, 15(3):409–428, 2003. Computer-Aided Design of User Interface.

[25] Mark Sanderson and W. Bruce Croft. The history of information retrieval research. *Proceedings of the IEEE*, 100(Special Centennial Issue):1444–1451, 2012.

[26] Romina Sharifpour, Mingfang Wu, and Xiuzhen Zhang. Large-scale analysis of query logs to profile users for dataset search. *J. Documentation*, 79(1):66–85, 2023.

[27] Craig Silverstein, Hannes Marais, Monika Henzinger, and Michael Moricz. Analysis of a very large web search engine query log. *SIGIR Forum*, 33(1):6–12, sep 1999.

[28] Amanda Spink, Jim Jansen, Dietmar Wolfram, and Tefko Saracevic. From E-sex to E-commerce: Web Search Changes. *Computer*, 35:107 – 109, 04 2002.

[29] Ryen W. White and Gary Marchionini. Examining the effectiveness of real-time query expansion. *Information Processing Management*, 43(3):685–704, 2007. Special Issue on Heterogeneous and Distributed IR.

[30] Barbara Wildemuth and Luanne Sinnamon. Search tasks and their role in studies of search behavior. 01 2009.

[31] Measuring Search Relevance with MRR. https://observer.wunderwood.org/2016/09/12/measuring-search-relevance-with-mrr/, 2016. [Online; accessed June 2023].

[32] Lei Yang, Qiaozhu Mei, Kai Zheng, and David Hanauer. Query log analysis of an electronic health record search engine. *AMIA ... Annual Symposium proceedings / AMIA Symposium. AMIA Symposium*, 2011:915–24, 01 2011.

[33] Huaiyu Zhu, Sriram Raghavan, Shivakumar Vaithyanathan, and Alexander Löser. Navigating the intranet with high precision. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, page 491–500, New York, NY, USA, 2007. Association for Computing Machinery.

Appendix A

Bot-User Agent List

feed	bot	rss	^sapo
autoproxy	Apple-Pubsub	Jakarta	Microsoft URL
WordPress	Atomic_email	^Mozilla/4.0$	^Mozilla/5.0$
FINLY	VERSION_TABLE	Akregator	Azureus
ESS Update	OSSProxy	aol/http	NetworkedBlogs
mediapartners-google	newsgator	bittorrent	Contacts
checker	greatnews	Winhttp	Drupal - 37
activesync	cache	kevin	webcopier
^java	PF:INET	plagger	python
ichiro	^Windows-Media-Player	Windows-Update-Agent	reaper
findlinks	facebookexternalhit	CHttp	Kaspersky
reader	nutch	PuxaRapido	Sphider
l.webis	iTunes	spider	lwp
curl	Microsoft BITS	AdminSecure	stackrambler
MicroMessenger	weborama-fetcher	AddThis.com	admantx
crawler	yahoo	libwww-perl	check_http
^Microsoft Office	vb project	bloglines	zend
AppEngine-Google	Mail.Ru	PostRank	NSPlayer
utorrent	HTTP agent	System.Net.AutoWebProxyScriptEngine	OutlookConnector
PubSubAgent	SOAP	^w3af	charlotte
BTWebClient	WLUploader	MPFv	JNPR
DataCha0s	Apache-HttpClient	unchaos	DAP
Aberja	CE-Preload	Infoseek	Sitewinder
^PHP	liferea	Ruby Blogzlce	^CFNetwork
Twitturly	Bookdog	Referrer Karma	ia_archiver
Wget	Transmission	AltaVista	Reeder
!Susie	CheckLinks	vagabondo	agadine
research	Yandex	silk	larbin
AppleSyndication	scoutjet	Microsoft-CryptoAPI	Rome Client
Anonym	OpenCalaisSemanticProxy	vlc	validator

Table A.1: Bot User-Agent list used for bot detection.

Appendix B

Stopword List

a	as	aos	ao
como	com	da	de
do	dos	das	e
em	era	entre	la
meu	mais	me	nao
nos	na	no	nas
o	os	ou	onde
quem	qual	que	por
para	se	ser	sem
suas	sua	sob	sobre
uma	uns	umas	um

Table B.1: Stopwords list.