Abstract

Launched in 2006, Emploitic.com is the leading recruitment website in Algeria. Its goal is to facilitate job searches for candidates and allow recruiters to find the best profiles with ease. Emploitic allows users to publish their professional information (training, work history, career summary, social links, etc.) to be used by recruiters when identifying new candidates or to obtain additional information about them. The company achieved its vision by means of a probabilistic information retrieval model developed using the Xapian toolkit. However, as the number of users in the website grew the limits of the old probabilistic model have been reached due to the emergence of a large number of users and their accompanied job posts and resumes. A poor relevance of results retrieved has become noticeable when handling such a sizeable dataset on account of the imitations opposed by the probabilistic model, which are no longer acceptable, which raises the issue to look for new solutions. One prime candidate solution is extending the system with a new service through the use of existing ontology for human resources generated by Emploitic company. Furthermore, the addition of an evaluator software system permitted an assessment of the performance for both old and improved versions of the search engine, showing a clear improvement in the relevance of retrieved documents, in addition, validating the newly developed system as an acceptable solution to this binary classification problem.

Dedications

I dedicate this work
To My parents Omi and Abi as they love to be called who gave me all the love, the support and guidance that made me the man I am today.
To My sister Amani and my brother Zakaria who shared every step of my journey with me.
To My dear fiancé who always been there for me.
To My friends Yousef, Salah, Amar, Abdelkarim, and all the people who shared with me these few humble years of studying.
To My teacher Dr CHERIFI Dalila who guided me all these years.
To My friend and Teacher Dr BENACHOUR Hamid who helped me learn and apply my knowledge.

- Souhaib Benbouzid

Dedications

I dedicate this work
To my family, specially the new addition
To my friends, Souhaib, Farouk & Farouk, Aymen – my gym buddy, Malek, Hichem, and all my friends who are still around and who are not
To Dr. Cherifi Dalila who helped us organize our project
To Rose, Sean and Rania for improving my English

- Bensalah Abdelkrim

Acknowledgments

First, we would like to express our appreciation and gratitude to our supervisors Dr. CHERIFI Dalila and Dr. BENACHOUR Hamid for the guidance and continuous support throughout the different stages of this project. Also, we would like to thank all people that helped us all the way to build our skillset and realizing this project. Special thanks to all professors and teachers that have guided us and have given us all the needed information to build up a rich background for our future engineering life.

Contents

List of Figures

List of Tables

List of Abbreviations

ACID : Atomicity, Consistency, Isolation, Durability.
AHROGA : Algerian Human Resources Ontology Generated Automatically.
API : Application Programming Interface.
ASCII : American Standard Code for Information Interchange.
HTML : HyperText Markup Language.
IFS: Inverted file structures.
IR : Information Retrieval.
JS : Java Script.
JSON : Jave Script Object Noation.
MVC : Model-View-Controller.
REST : Representational State Transfer.
SOLID: Single-responsibility, Open–closed principle, Liskov substitution principle, Interface segregation principle, Dependency inversion principle.
UI : User Interface.

General Introduction

In the modern world, large amounts of information are generated every day. Information science is concerned with that body of knowledge relating to the origination, collection, retrieval of information alongside many other aspects [1]. Search engines fall into the information retrieval field. As the name indicates, they are used to efficiently retrieve information based on certain characteristics. From the early search engine models, the boolean systems [2], to search engines like Google and DuckDuckGo, the expectation of users are on the rise. Relevance is of importance to the user, as everyone wants results as close as they need, which has been the main focus of search engine companies. This implies the need for search robots to understand what is displayed on the web, which is hardly the case. Such a complex task raises the necessity to find relations between different sets of concepts and categories in a subject area or domain which can be achieved using an ontological approach to model, define, and index such data. A good example of such a case is the need for a well-established Human Resource Ontology at Emploitic.com in order to improve the services offered by the platform. One of which is the Emploitic.com search engine. Improving the relevance of the results will enhance the user experience at the website.

This report is divided into three chapters. The first chapter starts with giving an overview on basic terminology and concepts to understand and work with search engines. Then, it presents one of the different applications of information science namely ontology. The second chapter starts by an overview about the old version of emploitic.com search system. Then, it continues with a suggestion and implementation of an ontology service . Furthermore, develop an evaluation service for the search engine. The third chapter is devoted to discuss the results generated using the evaluation tool and explain the benefits of our contribution to the underline system in terms of user experience as well as developer experience. In addition to arguing about the architecture and limitations of the new system, also suggest edits for future works.

Chapter 1

Review About Search Engines and Ontology

1.1 Introduction

In this chapter, we introduce two of the general aspects of information science. Firstly, we define the essential concepts to understand the architecture of information retrieval systems and their different models. Also, we cover the process of document file preparation and the management of different types of queries. Furthermore, we mention the evaluation metrics of a search engine. Secondly, starting from section 1.8, we define ontology and discuss aspects related to it. Ontology got the attention of information and software scientists due to its method of handling and structuring knowledge, since nowadays more companies need to handle massive amounts of unorganized data. Then, review the benefits and key points in creating a knowledge manager. Moreover, explore a solution using ontology to the problem of world wide web being dominantly human readable. At the end, point out some key differences between keyword based and ontology based search engines.

1.2 Definition of Search Engine

Gerard Salton, the father of Information Retrieval and one of the leading figures in information retrieval from the 1960s to the 1990s, proposed the following definition [3]: Information retrieval is a field concerned with the structure, analysis, organization, storage, searching, and retrieval of information. This definition still holds today despite the vast advancement in the field.

A search engine is the practical application of information retrieval techniques to large-scale text collections. Search engines can be found in many different applications such as web, desktop or enterprise. its main job is to rank and index many terabytes of data and then provide a response to many specific queries submitted at the same time in the least time possible.

1.3 Architecture of a Search Engine

Although the exact architecture differs from a search engine to another, the general architecture is discussed in to provide context by high-level description of the most important components in a search engine and the relationships between them. The two primary goals of a search engine architecture are:

- Effectiveness (quality): a retrieved result should be the most relevant set of documents possible for a query.
- Efficiency (speed): data retrieval and query processing should be as quick as possible.

A search engine goes through two major processes, the indexing process and the query process.

- **Indexing process** builds the necessary structures to enable query process to perform the search. It's divided into several parts.
  - **Text acquisition** identifies and makes available the documents that will be searched. Usually done by crawling or scanning the web. The end result of this process is a documents data store that contains text and meta data about each and every document. In addition to that it passes the documents to the next process.
  - **Text transformation** transforms documents into index terms or features. Index terms are just parts of the document or a simple word which is then stored in the index and used in the search process. A feature is a part of text document that is used to represent its content which also describes an index term. The set of all index terms that are indexed for a document collection is called the index vocabulary.

  - **Index creation** uses the output of the text transformation to create the data structures that enable fast searching. The index creation must be efficient both in terms of time and space. Indexes should be easily updated when new documents are acquired. The most common form of indexes used are inverted indexes and inverted files. The particular form of index used is one of the most important aspects of a search a search engine. An inverted index contains a list of documents that contains that index term for every index term. The inverted aspect refers to a document that lists the index term that it contains.

- **Query process** the major components are user interaction, ranking, and evaluation
  - **User interaction** component provide interface between the person doing the searching and the search engine. It accepts user's query and transform it into index terms. Also takes the ranked list results and display them to the user.
  - **Ranking component** is the core of the search engine. It takes a transformed query from the user interaction and generates a ranked list of documents using scores based on a retrieval model. Ranking must be both efficient to handle multiple queries and effective to accomplish the goal of finding the relevant information.
  - **Evaluation component** record and analyse user behaviour using log data and use it to tune the ranking component improve its effectiveness and efficiency.

1.4 Information Retrieval Models

A model of information retrieval serves as a blueprint which is used to implement an actual information retrieval system there are few known models' namely Boolean model, probabilistic model, semantic model, vertical model, and horizontal model.

1.4.1 Boolean Model

The Boolean model works Using the operators of George Boole's mathematical logic, the logical product AND, the logical sum OR and logical difference NOT. The result of query in such model is a set of unranked documents that either include or exclude the index terms depending on the operators present in the query. For example, a query (A AND B) operator will result a set of documents that include both A, and B. and so on. The advantages of such system appears when expert users use the system as it gives a sense of control and the results are clear why documents retrieved in a certain way. However, it does not provide ranking of retrieved data. Also, the decisions are binary either the document is retrieved or not retrieved the model has no means of retrieving relevant documents.

1.4.2 Probabilistic Model

The probabilistic model tries to define term weighting based on probability theory. The probability of relevance is denoted as P(R), we define the random variable R to take values 0, 1 where 0 is irrelevant and 1 is relevant.

The set of all possible outcomes of an experiment is called the sample space, in our case P(R) can be relevant, irrelevant. For example, in a set of 1 million document where there are 100 relevant document the probability of relevance P(R) can be calculated as the quotient of the two P (R = 1) = 100/1,000,000 = 0.0001. Also let's define a $P(D_k)$ as the probability that a document contains the term k with the sample space 0,1, where 1 = the document contains the term k and 0 = the document doesn't contain the term k. then probability $P(R = 1|D_k = 1)$ can be defined as is the conditional probability distribution which is the probability of relevance if we consider documents that contain the term k.

Whereas Maron and Kuhns introduced ranking by the probability of relevance, it was Stephen Robertson who turned the idea into a principle. He formulated the probability ranking principle, which he attributed to William Cooper, as follows [4].

"If a reference retrieval system's response to each request is a ranking of the documents in the collections in order of decreasing probability of usefulness to the user who submitted the request, where the probabilities are estimated as accurately as possible on the basis of whatever data has been made available to the system for this purpose, then the overall effectiveness of the system to its users will be the best that is obtainable on the basis of that data."

Stephen Robertson and Karen Sparck-Jones suggested to rank documents by $P(R|D)$ that is the probability of relevance R given the document's content description D note that d here is a vector of binary components each of the them represent a term, this can be interpreted as follows, if we have a set of 10 documents that are represented by the same D, if 9 of them are relevant, then $P(R|D) = 0.9$.

In practice, this is achieved using Bayes rule on the probability odds

$$P(R|D)/P(R^{*}|D) \tag{1.1}$$

, where R* denotes irrelevance.

1.4.3 Horizontal Model and Vertical Model

The horizontal search model is usually the one meant when referring to search engines, the indexing and ranking done in this model goes through the whole range of subjects appearing in set of documents to index. As opposed to horizontal search model, the vertical search model narrows its focus on very specific user tasks by leveraging domain knowledge. Which makes it more suitable when dealing with large sets of documents. The vertical search has great potential to serve users highly relevant search results from a specific domain. Its core component is relevance ranking.

The ranking in a vertical model is done in two types:

- Single-domain-related ranking that focuses on ranking for a specific vertical such as news search ranking and medical domain search ranking.
- Multi-domain-related ranking which focuses on ranking involving multiple verticals such as multi-aspect ranking, aggregating vertical search ranking, and cross-vertical ranking.

Vertical search offers several potential benefits over horizontal search engines:

- Greater precision due to limited scope,
- Leverage domain knowledge including taxonomies and ontologies,
- Support of specific unique user tasks.

1.5 Document File Preparation

A major part of search engine development is making decisions about how to prepare the documents that are to be searched. The designer should take into consideration the structure of the documents.

The decision taken on which parts of the documents should be indexed and which are not is called zoning, zoning is directly proportional to the relevance of the end result retrieved. Index building require two steps.

1.5.1 Document Analysis

Determines how each document is organized and how the information is presented, is the data presented in tables, charts, graphics ... etc. An example of document analysis is the process of analyzing web pages in commercial search engines such as Google, Yahoo and Bing. The relay on parsing the source code of web pages HTML documents and analyzing for predefined tags documents.

1.5.2 Token Analysis or Term Extraction

Determines which words best present the semantic content of documents to be used as referents. There are two types of indexing manual indexing which is still preferred by some entities publisher of MEDLINE (Medical Literature, Analysis and Retrieval System Online), a bibliographic database of over 15 million references. MEDLINE relays on human indexing however they also use some of the automatic indexing to index titles of articles and release some of the work load. They index up to half a million documents a year. However, using algorithms and software to extract terms for indexing is much more common in modern days. Automatic indexing consists of huge automatic computerized robots crawling throughout the documents repeatedly over long period of time, collecting and indexing every word in the text. The robots are just servers that make request same as a normal user. One difference to be noted is the concepts are realized in the indexing stage as opposed to the document extraction/collection stage. This leaves the decision making to the developer to determine the range of subjects to be covered.

In the end, the goal of both types is to extract words from the documents to allow searcher to find the best match for the information needed.

1.5.3 Item Normalization

Before using extracted terms to build a data structure to be used in the search, Items must be normalized. Item normalization is summarized by Kowalski [5]. As "The first step in any integrated system is to normalize the incoming items to a standard format. In addition to translated multiple external formats that might be received into a single consistent data structure that can be manipulated by the functional processes, item normalization provides logical restructuring of the item. Additional operations during item normalization are needed to create a searchable data structure: identification of processing tokens (e.g., words), characterizations of tokens, and stemming (e.g., removing word endings) of the tokens. The original item or any of its logical subdivisions is available for the user to display. The processing tokens and their characterizations are used to define the searchable text from the total received text." In other words, part of the document preparation is taking words to construct searchable data structures. A searching system must make decisions on how to handle words, numbers, and punctuation. These words are stemmed, meaning reformatted to its root format removing suffixes and prefixes, luckily there are a lot of automatic stemming tools, one major advantage of word stemming is handling misspelled queries by users.

1.5.4 Inverted File Structures

Inverted file structures (IFS) are a series of three components that track which documents contain which index term. it provides a critical shortcut in the search process. ifs organize the information into an abbreviated list of terms. depending on the term references a specific set of documents. There are three components in the IFS:

- The document file where each document is given a unique identifier and all the terms within the documents are identified.
- The dictionary is a sorted list of all the unique terms in the collection along with pointers to the inversion list.
- The inversion list contains the pointer from the term to which documents contain that term.

1.6 Query Management

A query is the basic building block of user interaction with IR systems. There are multiple types of queries that can be used depending on the search engine design and architecture. Each query type has its own advantages over the others. A query can be Boolean, Natural Language, Fuzzy, Term, or Probabilistic.

1.6.1 Query Binding

A user interacting with information retrieval system will write one or multiple queries to retrieve the most relevant information. The process of formulating the queries in one of the different types of the queries based on his own experience and vocabulary, translating the word into processing tokens by the search engine, and finally the use of processed token to retrieve the relevant information is called query binding.

1.6.2 Types of Queries

• **Boolean Queries** Logical Boolean queries link words using operators such as AND, OR, and NOT to convey the information needed. A query like books and articles, will retrieve all the documents containing both terms books and articles. However, documents containing one of the two terms will not be retrieved as there is no mean of relevance ranking in such system.

• **Natural Language Queries** Natural language queries are queries in which the user formulates as a question or a statement. An example would be "what is the book name that contains bread and butter?". Processing queries is done by extracting the index terms, some of the words might be ignored in the process. When the index terms are extracted, the words lose the semantic meaning as computers can't determine its context which is a drawback of such approach.

• **Thesaurus Queries** A thesaurus query is one where the user selects terms from a predetermined set of terms by the system. The advantage of such a system lies in the predetermined thesaurus, users can easily navigate the queries based on the word meaning. However, this is also a disadvantage, if the user thinks the index terms do not lead to the required information, which limits the possible outcomes.

• **Fuzzy Queries** A fuzzy query reflects non specificity and refers to the capability of handling misspelled or variations of the same word. User might type a misspelled word that can be stemmed automatically to its root. Then documents related to its root can be retrieved.

• **Term Searches** A term search query is one where users provide a few words or phrases for the search. Its different from a natural language query in the fact that a term search don't have to be full question and can be as small as one word. Term search is the most used today. spicily by new users on the commercial web search engines.

• **Probabilistic Queries** A probabilistic query refers to the manner in which the Information Retrieval system retrieves documents according to relevancy. For a given document and a query it's possible to compute the probability of relevancy for the document with respect to the query One advantage of probabilistic queries over fuzzy queries is that there is a well-established body of methods for computing probabilities from frequency data.

1.7 Ranking and Relevance Feedback

A fully functional search engine has the ability not only to list the search ranked results but also allow the user to improve the search system. This is referred to as relevance feedback. Harman [5] has done extensive work in judging how relevance feedback positively affects system performance. Marchionini [6] points out another advantage of ranking and relevance, ranking and relevance feedback can be considered as highly interactive information seeking.

1.7.1 Performance Evaluation

One might simplify search engine performance based on user satisfaction, this can be a binary feedback (satisfied, not satisfied) or in terms of a scale (1 to 10). However, this method is too simple. What if the user wants more relevant results retrieved based on a document? Or, the user did not like the ranking of the results. How can the system determine which results is better to be on top of the list and which should be on the bottom? Korfhage [7] also points out that users will not tolerate more than three or four attempts of feeding back information to the system, nor does the user like seeing the same set of documents appearing over and over again. Such dilemma pushed researches to develop Many standards to discuss and compare results of information retrieval systems. Precision and recall are two of the standard definitions used in search system evaluation.

1.7.2 Precision

The precision or precision ratio P of a search method is defined as [8]

$$P = D_r/D_t \tag{1.2}$$

- D_r is the number of relevant documents retrieved.
- D_t is the total number of documents retrieved.

It is important to keep in mind that relevance, or the appropriateness of a document to a user's needs, is still a judgment call. How users might look for the same information, one can find the retrieved information relevant while the other disagree and so on. That's because relevance is subjective and depends on the person who is searching.

1.7.3 Recall

The recall or recall ratio R of a search method is defined as [8]

$$R = D_r/N_r \tag{1.3}$$

- D_r is the number of relevant documents retrieved
- N_r is the total number of relevant documents in the collection

Because the number of relevant documents in the collection is hard to determine, one cannot simply determine recall ration. A user generally needs is a complete subset of relevant documents that does not require a substantial weeding out of all the irrelevant material, If a searcher wants only the precise documents that fits his or her exact needs, then the query will require very specific terms. However, if the search is extremely precise, many relevant documents will be missed. if the recall is increased, the user more than likely will have to wade through what is known as false drops, garbage and noise to pick out the relevant documents. That explains the integral role of recall.

1.8 Definition of Ontology

Ontology(uncount.) as a branch of philosophy is the science of what is, of the kinds and structures of objects, properties, events, processes, and relations in every area of reality [9]. In the context of our research, an ontology(count.) is an explicit formal specifications of the terms in the domain and relations among them [10], in other words it is a formal explicit description of concepts in a domain of discourse (classes/concepts), properties of each concept describing various features and attributes of the concept (slots) and restrictions on slots (facets) [11]. An ontology together with a set of individual instances of classes constitutes a knowledge base. From the definition, one might think of it as being the same as Object Oriented Programming (OOP). While ontologies are formally specified models of bodies of knowledge defining concepts used to describe a domain and the relationship that hold between them. Object model is the mechanism of object-oriented paradigm (it is based upon objects (having both data and methods) that aims to incorporate the advantages of modularity and reusability), which is used for software engineering [12].

1.9 Benefits of creating an ontology

For academics that need to communicate information in a domain, an ontology defines a standard vocabulary. It contains machine-interpretable definitions of the domain's basic concepts as well as relationships between them. Why would someone want to develop an ontology? Some of the reasons are:

- To share common understanding of the structure of information among people or software agents.
- To enable reuse of domain knowledge.
- To make domain assumptions explicit.
- To separate domain knowledge from the operational knowledge.
- To analyze domain knowledge. [11]

1.10 Creating an ontology

As lots of subjects in philosophy, philosophers have been arguing about a correct way of creating an ontology. Same goes in information science, there is a set of general issues that one should consider one developing an ontology. The development of an ontology can be done using an iterative approach by starting with a rough first pass at the ontology and then consider the following rules to make the modeling decisions:

1. There is no one correct way to model a domain— there are always viable alternatives. The best solution almost always depends on the application that you have in mind and the extensions that you anticipate.

2. Ontology development is necessarily an iterative process.

3. Concepts in the ontology should be close to objects (physical or logical) and relationships in your domain of interest. These are most likely to be nouns (objects) or verbs (relationships) in sentences that describe your domain. [11]

It it also advisable to limit the domain of the ontology, precisely define the use alongside the users and the questions it can answer. All these modeling decisions will guide the creator through through the process. There exist many software solutions to create an ontology, like protege and Neo4J.

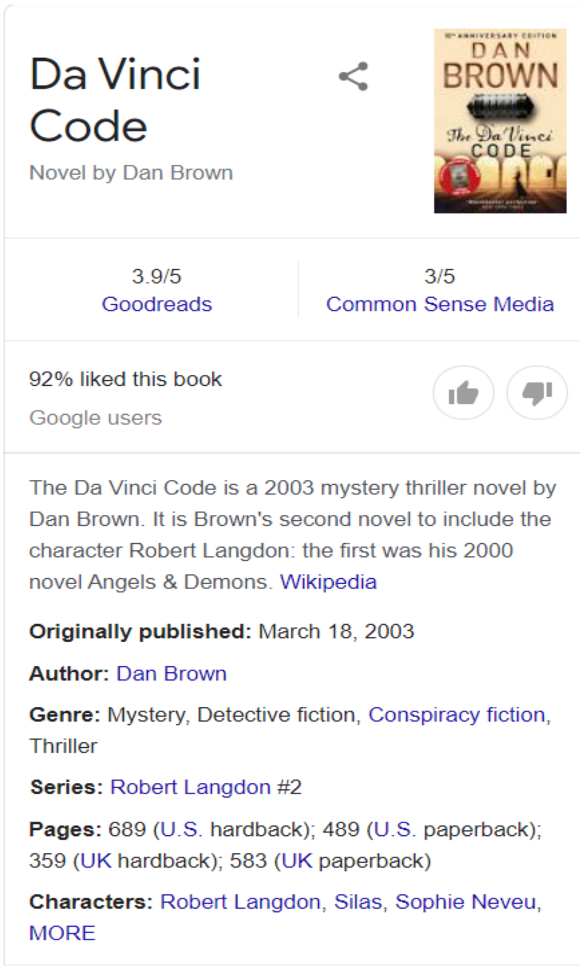
1.11 Idea of semantic web

The semantic web is a concept meant to fix two major issues that the World Wide Web has when it comes to data. The first one is it excels at displaying data in a way that humans can process and fails at displaying data in a way that softbots can process. The second one is it lacks any inherent mechanisms to ensure that the information you're looking at is authoritative and reliable. As for the first problem, for example, a bookseller makes a website listing the book available for sale. The list or table includes the author, the price and the number of pages. The Web is constructed on the back of HTML, a language used to fundamentally tell the browser how to construct a page's appearance. A human being is able to look at the table 1.1 and understand the data through context: that "Lincoln in the Bardo" is a title, because it is in the title column. That James Patterson is the author of 'The 15th Witness' because the two data points are in the same row. That the author's name is George Saunders and not Saunders, George because the Last Name-comma-first name construction used in a lot of information displays.

Title	Author	Price	Pages
Lincoln in the bardo	Saunders, George	1600Da	300
The 15th Witness	Patterson, James	1400Da	250

Table 1.1: Table of prices of the bookstore

It is significantly harder for computers to understand that. Part of the problem is that a computer is unable to see the table when rendering it – it just sees lines of HTML. The other part of the problem is the difficulty to teach a computer context (Data points A and B are in the same row and thus are related. Data point C shares a column with A and thus is related but is not related to B because it is in a different row and column). Furthermore, if all booksellers displayed their data in the same way, it would be enough to train a robot that can make a list of all books for sale by scanning booksellers webpages world wide with the pattern “data in the first column is a title” and so on. But that is not the case, a rival bookseller can display data in a different way from the example discussed above. The semantic web is a concept meant to fix the problems mentioned and seeks to create a set of languages, systems, and processes by which data is formatted in a way that if it is either human-readable or not, it should be at least universally machine-readable. One way of doing that might be to add hidden tags to the code so that the example table is displayed in the same way, but the additional tags will allow the computer to read and understand the data like a human would. Metadata like Schema.org or VIAF (virtual international authoritative file) can be the source of these tags. The computer reading the code will know that the column of authors is indeed about authors since that’s how it is defined in these metadata. When searching in the web for a movie or a book – and many other things, a box appears to the right including info about the object of search in figure 1.1).


Da Vinci Code

Novel by Dan Brown

3.9/5 Goodreads

3/5 Common Sense Media

92% liked this book

Google users

The Da Vinci Code is a 2003 mystery thriller novel by Dan Brown. It is Brown's second novel to include the character Robert Langdon: the first was his 2000 novel Angels & Demons. Wikipedia

Originally published: March 18, 2003

Author: Dan Brown

Genre: Mystery, Detective fiction, Conspiracy fiction, Thriller

Series: Robert Langdon #2

Pages: 689 (U.S. hardback); 489 (U.S. paperback); 359 (UK hardback); 583 (UK paperback)

Characters: Robert Langdon, Silas, Sophie Neveu, MORE

Figure 1.1: Google Box of Information

The information filled in said box is not stored in Google’s servers, instead Google combs through the research results of the query searching for information that is tagged with meta – the meta in this case is Schema.org since it is the standard for Google. As for the second problem – authoritative information, referring back to the bookseller example in the first problem. In a case of a buyer wanting a James Patterson book, they might be unsure if the James Patterson in the list is the same as the James Patterson their favorite author regardless of them having the same name. As humans, one way to check the sameness is, for example, Googling the book and confirming that it is indeed the same James Patterson because humans understand the concepts of probability and context. For machines, the process is not as straightforward. To solve this problem, the conceptors of the semantic web developed the authority file – a database that can be related to containing names of people, places, things, and concepts established in one particular form – one form per language. The biggest being the Virtual International Authority File (viaf.org). By adding few lines of code that has the metadata used and the exact link to the author James Patterson, the softbots can understand that the author in the example table is the same James Patterson that we, humans, know. Authority files are also really helpful in bridging cultural gaps. Movies and books have different names in different countries e.g., the first Harry Potter novel was called The Philosopher's Stone in the UK and The Sorcerer's Stone in the US. The most recent Pirates of the Caribbean movie was called "Dead Men Tell No Tales" in the US and "Salazar's Revenge" in other countries. By providing a link to an authority file it is possible to make sure that we, humans, (and our computers) are talking about the same thing.

1.12 Semantic and Keyword based search engines

Typical Keyword Search engines are very useful for discovering information on the internet and getting results quickly, but they suffer from the fact that they do not understand the meaning of the phrases and expressions used in web pages and the relationship between them. [13]. According to surveys, consumers who search information on the web do not find reliable results in the initial set of URLs returned, owing to the increasing amount of links on online pages. When one term has multiple meanings and other words have the same meaning, a user's search for that phrase may result in confusion and the user will not get what he was looking for. [14].

Semantic Search Engines, on the other hand, are sophisticated engines that look for terms based on their meaning [15]. Furthermore, they guarantee that the results are connected to the meaning of the sought phrases. Ontologies are used by semantic search engines to achieve meaningful retrievals and a high level of accuracy. Semantic Web is thought to be a Web 3.0 or an extension of the current web. These semantic engines are distinguished by the presence of multiple sorts of relational ties among various types of resources, as opposed to a single relation of resources. There are numerous examples, such as Hakia, DuckDuckGo, Swoogle, and so on.

1.13 Summary

We described the general architecture of information retrieval systems. in addition to, elaborate on the two main important processes of Information retrieval. The indexing generation of terms through the indexing process. And the retrieval of documents using the searching process. In the next chapters, we will apply the ontology concepts to enhance a previously developed probabilistic model. And use precision and recall metrics to assess the performance.

Chapter 2

Improvement of Emploitic Search Engine

2.1 Introduction

Emploitic's search engine is not up to date to major new technologies. In this chapter we will highlight the main differences between the previous system and the improved one i.e. our contribution, in both ends: frontend and backend. Where we explain in the first section how the previous system works and detail the backend core, which is a Xapian application consisting of three parts: Search, Indexer and Database. Alongside the front applications and the different views proposed. In the following section, we explicate the architecture and justify the choice of technologies used in the improved system, which consists of two new services: Ontology and Evaluator service. The prior service deals with the AHROGA ontology which is the central element of our new system and explain the connection between the ontology and the search engine, as well as the formatting of the queries. The latter service is a suggestion to evaluate the search results using the previously discussed metrics. The service automatically calculates and stores performance metrics that can be easily retrieved and illustrated using the evaluator frontend application.

2.2 Previous Version of The Search Engine

The previous system consists of a python application built on top of a Xapian toolkit divided into three main sections: Tornado web server, Xapian Search, Xapian Indexer, database, queue system. Alongside helpers including schemas, build configurations, and text processing utils. In this section, we will discuss the functionality and the significance of each section of the old system. As well as the means of communication between each of them. he previous system architecture can be seen in Figure 2.1

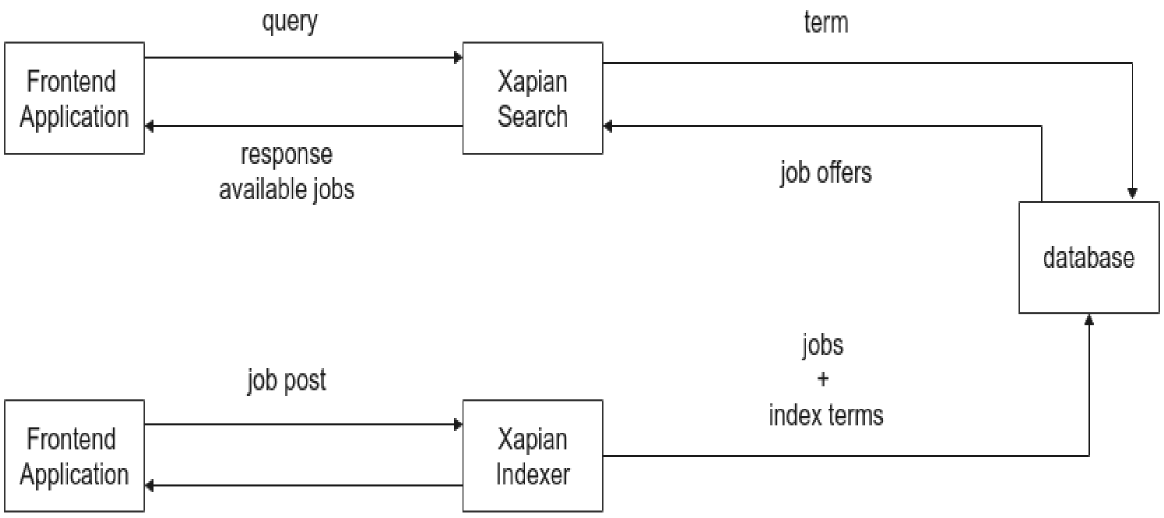


Figure 2.1: Previous System's Architecture

2.2.1 Tornado web server

The web server listens to user requests under predefined routes, each request coming will be intercepted and redirected to its appropriate handler class. Either SearchAppDispatch Class in the case of a GET request to the search endpoint or UpdateAppDispatch Class in the case of a POST request to the indexing endpoint. The server also exposes 2 more endpoints, that deal with the push notification to the querying system. However, this will not be covered since we will not be modifying these endpoints.

2.2.2 Xapian Search

Xapian search includes SearchAppDispatch class, this class is performing many core functionalities in the system namely:

- Parse user GET requests to extract search queries sent by the frontend application, alongside the search options like filters. These filters are used to specify options like where to start and end search.
- Process the query using previously mentioned text processing utils. Initiate and add the search query to the Xapian query parser.
- Initiate Xapian enquire which is a class that fetches data from the indexed database.

The Xapian query parser will then generate a new query to be used by Xapian enquire to fetch the relevant data from the database.

2.2.3 Xapian Indexer

Xapian indexer includes UpdateAppDispatch class, As the name implies, this class is also performing core functionalities in the system namely:

- Parse user POST requests to extract job post data sent by the frontend application, alongside the storage specifications. Like filters, act, and company ids which are special identifiers for each indexed job post.
- Validate job post data using predefined schemas and report the errors accordingly.
- Initiate the indexing of job post data using the database class

2.2.4 Database

Xapian database is the most important piece of the system, XapianDatabase class, performs the following:

- Create a Xapian document to be added to the database.
- Process indexing filters provided by Xapian Indexer.
- Process all job post data, the process includes cleaning and normalization of job post data, generation of the terms using Xapian.TermGenerator() the generated terms will be used to index the created document.
- each generated term to the previously created document and commit the document to the database.

2.2.5 Frontend Applications

Front end applications are the way in which a user would interact with the system, either to add a job post or to search for a job. by using one of the two following views:

- **Job Seekers View** The view allows the user to enter a query using an HTML input field and specify one or many filters to select from. The Job Seekers View simple view is pictured in figure 2.2 and Job Seekers View Advanced search view is pictured in figure 2.3.

- **Employers View** The view allows the user to fill in job post data to be used to be indexed in the backend.

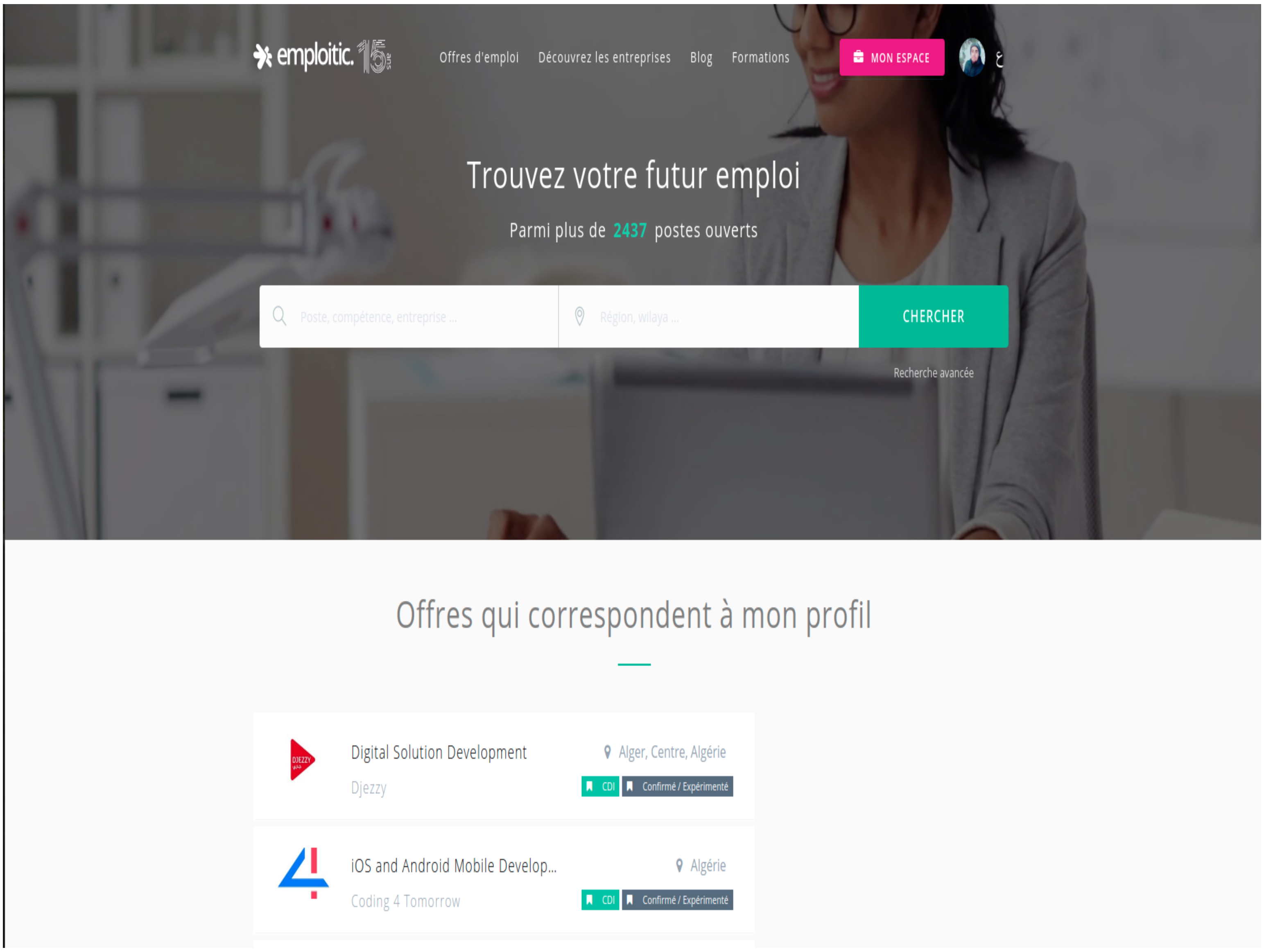
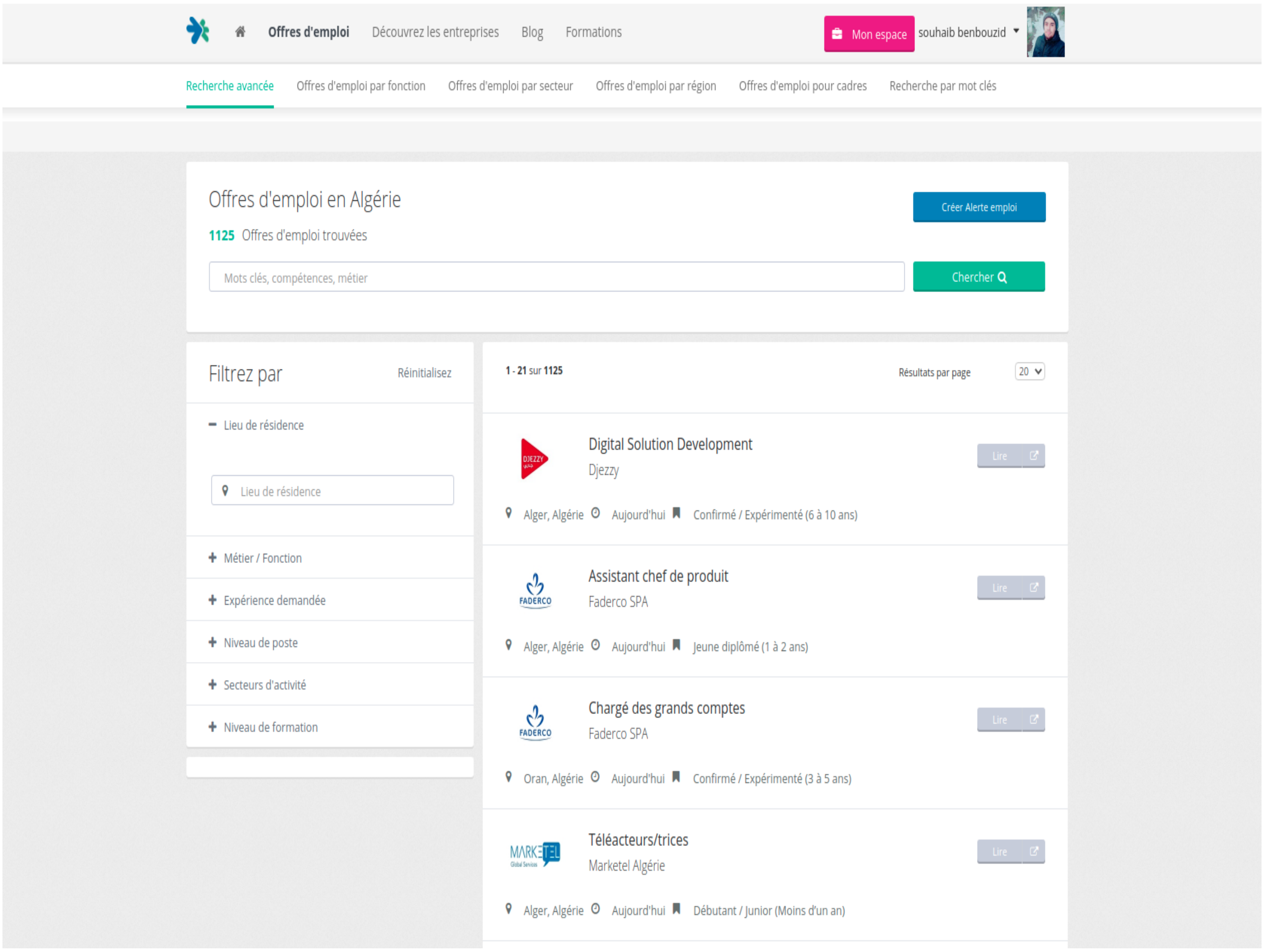

emploitic 15 ans
Offres d'emploi
Découvrez les entreprises
Blog
Formations
MON ESPACE
Trouvez votre futur emploi
Parmi plus de 2437 postes ouverts
Poste, compétence, entreprise ...
Région, wilaya ...
CHERCHER
Recherche avancée
Offres qui correspondent à mon profil
Digital Solution Development
Djezzy
Alger, Centre, Algérie
CDI
Confirmé / Expérimenté
iOS and Android Mobile Develop...
Coding 4 Tomorrow
Algérie
CDI
Confirmé / Expérimenté

Figure 2.2: Job Seekers View simple view


Offres d'emploi Découvrez les entreprises Blog Formations Mon espace souhaib benbouzid

Recherche avancée Offres d'emploi par fonction Offres d'emploi par secteur Offres d'emploi par région Offres d'emploi pour cadres Recherche par mot clés

Offres d'emploi en Algérie

1125 Offres d'emploi trouvées

Créer Alerte emploi

Mots clés, compétences, métier

Chercher

Filtrez par Réinitialisez

Lieu de résidence

Lieu de résidence

Métier / Fonction

Expérience demandée

Niveau de poste

Secteurs d'activité

Niveau de formation

1 - 21 sur 1125 Résultats par page 20

Digital Solution Development
Djezzy
Alger, Algérie Aujourd'hui Confirmé / Expérimenté (6 à 10 ans)
Lire

Assistant chef de produit
Faderco SPA
Alger, Algérie Aujourd'hui Jeune diplômé (1 à 2 ans)
Lire

Chargé des grands comptes
Faderco SPA
Oran, Algérie Aujourd'hui Confirmé / Expérimenté (3 à 5 ans)
Lire

Téléacteurs/trices
Marketel Algérie
Alger, Algérie Aujourd'hui Débutant / Junior (Moins d'un an)
Lire

Figure 2.3: Job Seekers View Advanced view

2.2.6 Communication Between Different Parts of The System

The backend application exposes a REST API to be consumed by the frontend applications, which is a software architectural style that was created to guide the design and development of the architecture for the World Wide Web. REST defines a set of constraints for how the architecture of an Internet-scale distributed hypermedia system, such as the Web, should behave. The REST architectural style emphasizes the scalability of interactions between components, uniform interfaces, independent deployment of components, and the creation of a layered architecture to facilitate caching components to reduce user-perceived latency, enforce security, and encapsulate legacy systems.

2.2.7 Xapian

Xapian is an open source probabilistic information retrieval library. The following points discuss some important notions in information retrieval related to xapian.

- **Documents** In Xapian, a document is just an item returned by a search. When creating a new search system, one of the most important decisions to make is what the documents in your system will be. There is frequently an apparent decision here, but there are often alternatives. For example, it appears normal to have one document for each page of a website when conducting a search. You might, however, use one document for each paragraph on each page, or put pages together into subjects and have one document for each. In a database, documents are recognized by a unique positive integer id known as the document ID. This is currently a 32-bit value. Documents are made up of three parts: data, terms, and values.

- **Terms** The majority of searches in Xapian are based on terms. A search, at its most basic, is the act of comparing the terms supplied in a query to the terms in each document and providing a list of the documents that match the best. A term is frequently generated for each word in a piece of text, usually by applying some type of normalisation (it is common to transform all the characters to lowercase). There are numerous effective ways for creating words. The same term will frequently appear many times in a piece of literature. This value is referred to as the within document frequency by Xapian, and it is saved in the database. When searching, it is frequently advantageous to give more weight to papers in which a term appears more frequently. It is also possible to keep a collection of positions alongside each term, allowing the positions at which words occur to be utilised when searching, for example, in a phrase query. Typically, these places are saved as word counts (rather than character or byte counts). The database also maintains track of a few statistics on the frequency of terms in the database; the term frequency is the number of documents in which a specific term appears, and the collection frequency is the total number of times that phrase appears (i.e., the sum of the within document frequencies for the term).

- **Stemming** Stemming is a typical type of normalisation. This procedure converts numerous word forms to a single form, such as turning a plural (e.g., "birds") and a singular (e.g., "bird") form of a word to the same item (in this case, both are transformed to "bird"). A stemmer's output is not always a genuine word; what matters is that words with closely related meanings be changed to the same form, allowing a search to identify them. For example, the words "happy" and "happiness" are both translated to the form "happi," therefore if a document contained "happiness," a search for "happy" would return that document. The rules employed by a stemmer rely on the language of the text; Xapian includes stemmers for over a dozen languages (and for some languages, there is a choice of stemmers), written with the Snowball programming language. We'd want to add stemmers for other languages; check the Snowball site for more on how to help.

• **Term generation** Rather of requiring all users to build their own code to convert text into indexable words, Xapian supplies a xapian. TermGenerator is a class that generates terms. This parses text chunks, generates relevant phrases, and adds them to a document. When producing terms, the xapian.TermGenerator can be set to do stemming (and stopwording). It can optionally keep information about the locations of words inside the text and apply field-specific prefixes to the generated terms to allow searches to be limited to specified fields. It can also add more information to the database for use in spelling correction. When using xapian.TermGenerator to process text, it is usually paired with QueryPary which converts a human readable query string into a Xapian Query object.

• **Weighting model** BM25 is the default weighting method used by Xapian. The model is distinguished by the inclusion of a distinct notion of relevance, an explicit variable linked with a query–document pair that is ordinarily hidden in the sense of being unobservable. The model is based on calculating a probability of relevance for each pair and ranking documents in descending order of likelihood of relevance in connection to a particular query. The BM25 term-weighting and document-scoring function is the model's most well-known implementation.[18]

2.3 Improved Version of The Search Engine

In this section we will discuss all the edits, tweaks and addition to the previous system in both ends: Front-end and Back-end.

2.3.1 Rework of the Front-End

The previous front-end of the system was outdated. A rework of the front-end will future proof it, as we chose the latest technologies in terms of programming languages, libraries and approach to develop it. To develop a modern and maintainable front-end code, choosing the programming language, libraries and approach is crucial. After developing the front end the final architecture is illustrated in figure 2.4

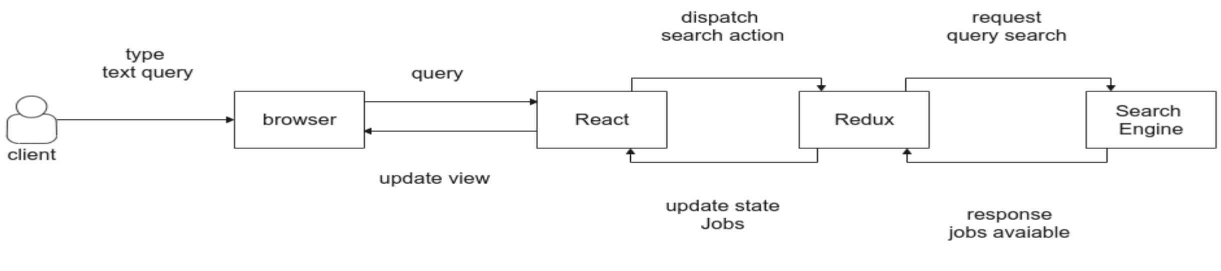


Figure 2.4: Developed Front-end architecture

• **Programming language** Typescript was the go-to programming language, for it being a strict syntactical superset of JavaSript and adding optional static typing to the language[16]. Static typing is the compiler enforcing the values to have the same data type, which is not the case with JavaScript compilers. This allows for code with less common errors than a typical JS code, preventing glitches and increasing the overall performance of the compiled code withing the project. As of 2021, TypesScript is still maintained by its creator company Microsoft since 2013, the last stable version was released on April 2021.

• **Libraries** Closely related to programming language, libraries are the backbone of any complex code. With the surreal number of libraries for web development, it is wise to choose the most viable and tested functions regardless of creation date. Redux and React (React.js or Reactjs) were the libraries of choice for state management and UI respectively. Redux is a state container that assists the developer to create JavaScript apps, TypeScript in this case, that are easy to test and act reliably across client, server, and native platforms.While it is commonly used with React as a state management tool, it may be used with any other JavaScript framework or library. Because it's under 2KB in size (including dependencies),increasing the asset size of the application is not a concern.The state of the application is stored in a store with Redux, and each component can access any state it requires from this store. State management is fundamentally a method of facilitating communication and data sharing among components. It generates a readable and writable data structure to represent the state of your app. This allows to observe normally invisible states while dealing with them.Most frameworks, like as React, Angular, and others, include a mechanism for components to maintain their state internally, eliminating the need for an additional library or tool. It works well for applications with a small number of components, but as the program develops in size, handling states shared by components becomes a hassle.Most frameworks, like as React, Angular, and others, include a mechanism for components to maintain their state internally, eliminating the need for an additional library or tool. It works well for applications with a small number of components, but as the program develops in size, handling states shared by components becomes a hassle.

React.js is an open-source JavaScript package used to create user interfaces for single-page apps. It is used to manage the view layer in online and mobile apps. React also enables us to design reusable user interface components. Jordan Walke, a Facebook software engineer, was the first to create React. React was initially used in the Facebook newsfeed in 2011.Developers can use React to create massive web applications that can alter data without reloading the page. React's primary goal is to be quick, scalable, and simple. It only works on the application's user interfaces. This is equivalent to the view in the Model-View-Controller template. It can be combined with other JavaScript libraries or frameworks, such as Angular JS in MVC [17].

• **Development methodology** Regarding the approach – the helper to conceptualize the front-end,Tom Coleman's component driven user interface introduced in 2017 was a viable choice given its mainstream use in front-end applications. User interfaces in today's world are more sophisticated than they have ever been. Customers want engaging, tailored experiences across many devices. As a result, front-end developers and designers have to include more intelligence into the user interface.However, when applications expand in size, UIs become cumbersome. Large user interfaces are brittle, difficult to debug, and take a long time to launch. It's simple to develop powerful yet adaptable UIs by breaking them down into modules.Components enable interchangeability by isolating state from application business logic.It became possible to break complicated displays into basic components in this way. Each component has a well-defined API and a predefined set of mocked states. This permits components to be disassembled and reassembled to create multiple user interfaces, allowing components and UIs to be future-proof.

• **Proposed Ontology Service** The back-end is the pillar to any application.As of today, the majority of web applications are split into micro-services that communicate between each-other by calling other services, making the development and maintenance of each one easier and the overall application minimal. Docker is a software platform that makes it easier to design, execute, manage, and distribute applications. It accomplishes this by virtualizing the operating system of the machine on which it is installed and running. There are three key points in a docker: a dockerfile is a blueprint to build a docker image. A docker image is a template to running a docker container. A container is a running process. Since it all runs on a virtual machine it solves the problems of compatibility since the developer can choose the environment and build the container. The part of the back-end that was created is split into 2 main services that are managed by docker-compose, which is a tool for defining and running multi-container docker applications, each with its own container.The First is a search service, i.e. the previous system with some tweaks(section 2.2) .The Second is an ontology service. The ontology service is built using a flask application – that returns semantically relevant job titles it exposes a GET endpoint on the port 5000 namely /ontology that will search the ontology graph database neo4j using cypher language and return the results after processing in the response body as well as status if the request was successful or not. The architecture of the backend after the integration of the ontology is depicted in the figure 2.5.

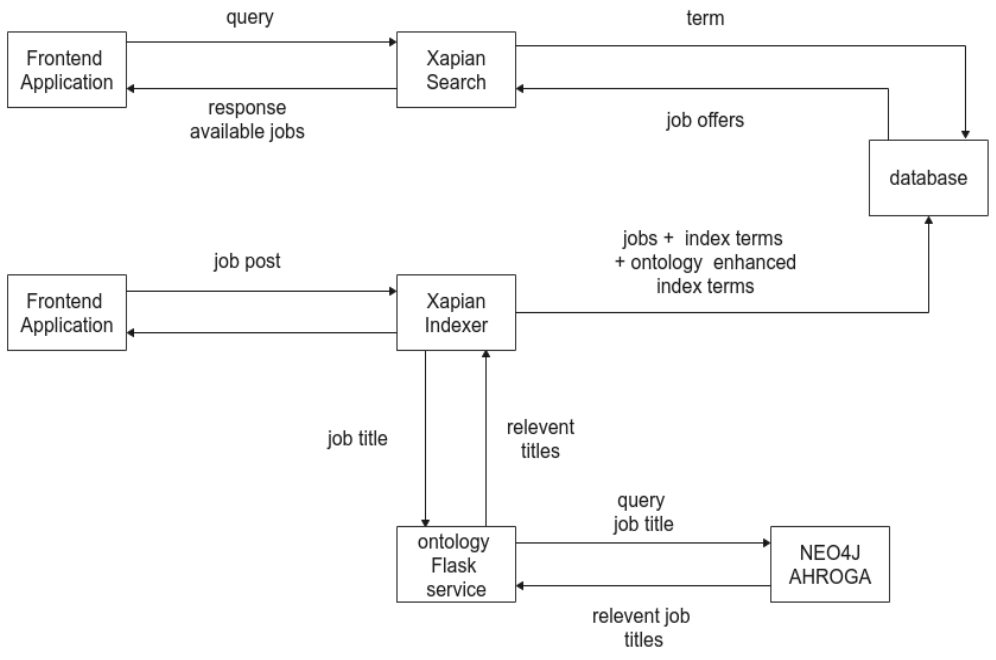


Figure 2.5: Ontology Integration

2.3.2 Flask Application

Flask is a Python API that allows us to build web applications. Armin Ronacher created it. Flask's framework is more explicit than Django's framework, and it is also easier to learn because it requires less base code to implement a simple web-application. A Web-Application Framework, also known as a Web Framework, is a collection of modules and libraries that allow developers to create applications without having to write low-level code such as protocols, thread management, and so on. Flask is built on the Web Server Gateway Interface toolkit and the Jinja2 template engine.

2.3.3 Neo4j

Neo4j is a NoSQL native graph database that is open source and provides an ACID-compliant transactional backend for your applications. Its development began in 2003, but it has just recently become publicly available. The Java and Scala source code is available for free on GitHub or as a user-friendly desktop application download. The database Neo4j is available in both Community and Enterprise editions. The Enterprise Edition provides everything that the Community Edition does, plus additional enterprise requirements such as backups, clustering, and failover capabilities. Because it efficiently implements the property graph concept down to the storage level, Neo4j is referred to as a native graph database. This means that the data is stored precisely as you whiteboarded it, and the database navigates and traverses the graph using pointers. Unlike graph processing or in-memory libraries, Neo4j also includes full database features such as ACID transaction compliance, cluster support, and runtime failover, making it ideal for using graphs for data in production applications.

2.3.4 AHROGA ontology

In this project we used an already existing ontolgy which is AHROGA, a human resources ontology generated automatically from data of the Algerian e-recruitment platform emploitic.com. The ontology is defined through three characteristics:

The domain and the goal of the ontology The ontology seeks to deliver valuable services such as determining whether a candidate has the necessary abilities for a job, advising a candidate on how to showcase his skills, and recommending the most relevant profiles to recruiters.

Data used for ontology generation For the ontology generation, the profiles of emploitic.com were used, that include occupations, skills and professional domains.

Concepts The concepts of our ontology are: Professional domain, Occupation, Community and Skill. [19]

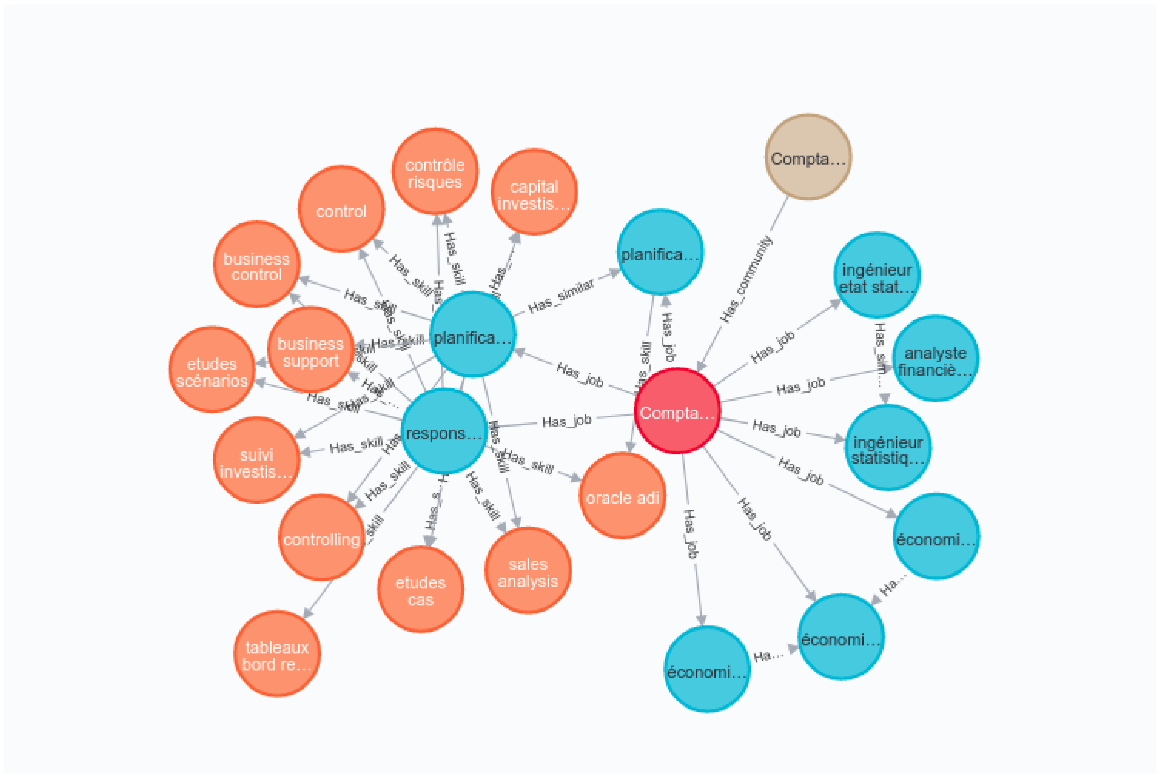


Figure 2.6: Graphical representation of the Ontology

2.3.5 Cypher language

Cypher is Neo4j's graph query language, which allows users to store and retrieve data from graph databases. Neo4j aimed to make graph data querying simple for everyone to learn, understand, and use, while still including the power and capability of existing conventional data access languages. This is what Cypher aspires to achieve. The syntax of Cypher allows you to match patterns of nodes and relationships in the graph in a visible and logical fashion. It is a declarative, SQL-inspired language that uses ASCII-Art syntax to describe visual patterns in graphs. It enables us to express what we wish to choose, insert, update, or delete from our graph data without providing a detailed description of how to do so. Users can use Cypher to create expressive and efficient queries to handle required create, read, update, and delete functionality. Not only is Cypher the ideal way to interface with data with Neo4j, but it is also open source! The openCypher project offers an open language definition, a technical compatibility kit, and a reference implementation of the Cypher parser, planner, and runtime. It is supported by various database vendors and allows database implementors and clients to freely benefit from, utilize, and contribute to the development of the openCypher language.

2.3.6 Proposed Evaluator service

One of the most important tasks of software systems and systems, in general, is the evaluation part, The old system relied on manual evaluation to do so which is not reliable in terms of efficiency nor terms of cost, To track the evaluate the old system as well as to measure the effects of the modifications on the performance after the integration of Ontology Service a new service is put in place. Namely, the Evaluator Service as the name implies the sole purpose of such a system is to evaluate and record the performance metrics and display them in a coherent, readable graph. New fixes and features can be developed according to the newly achieved data.

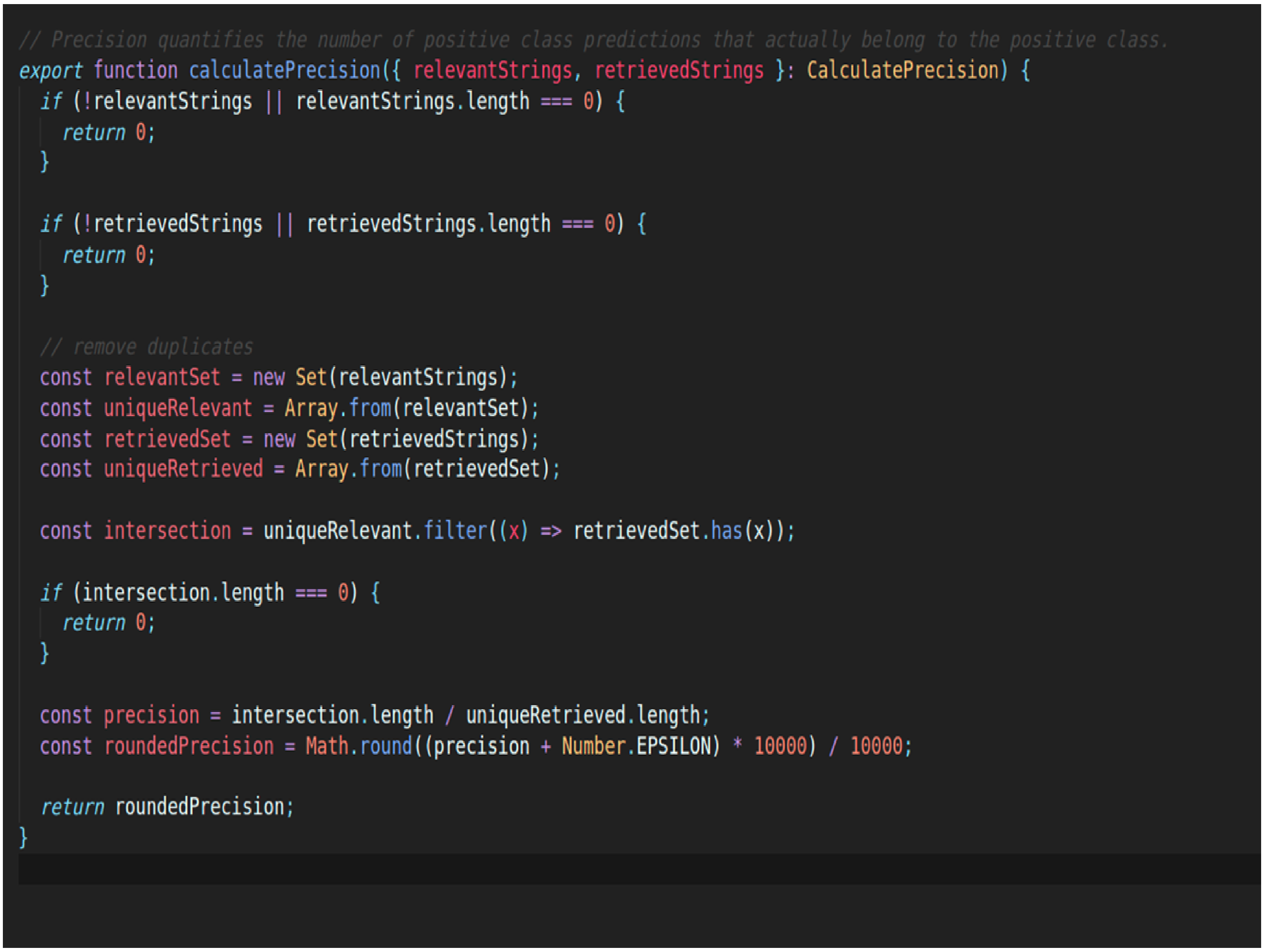
2.3.7 Performance Metrics

The classification of the relevance of documents is a binary classification problem. Supposed that our model can make robust predictions about the relevance of a document, we need to decide whether it is a good enough model to solve the problem. Usually, Classification accuracy alone is typically not enough information to do so. We will look at Precision, Recall, and F1 performance measures we can use to evaluate our model.

- **Precision** Quantifies the number of correct relevant titles selected by the retrieval system, already discussed in section 1.7.2. Answering the question, "how many selected documents are relevant?".. It was calculated using the equation 2.1 [20].

$$Precision = \frac{True\,Positive}{True\,Positive + False\,Negative} \tag{2.1}$$

The implementation code for precision is in figure 2.7



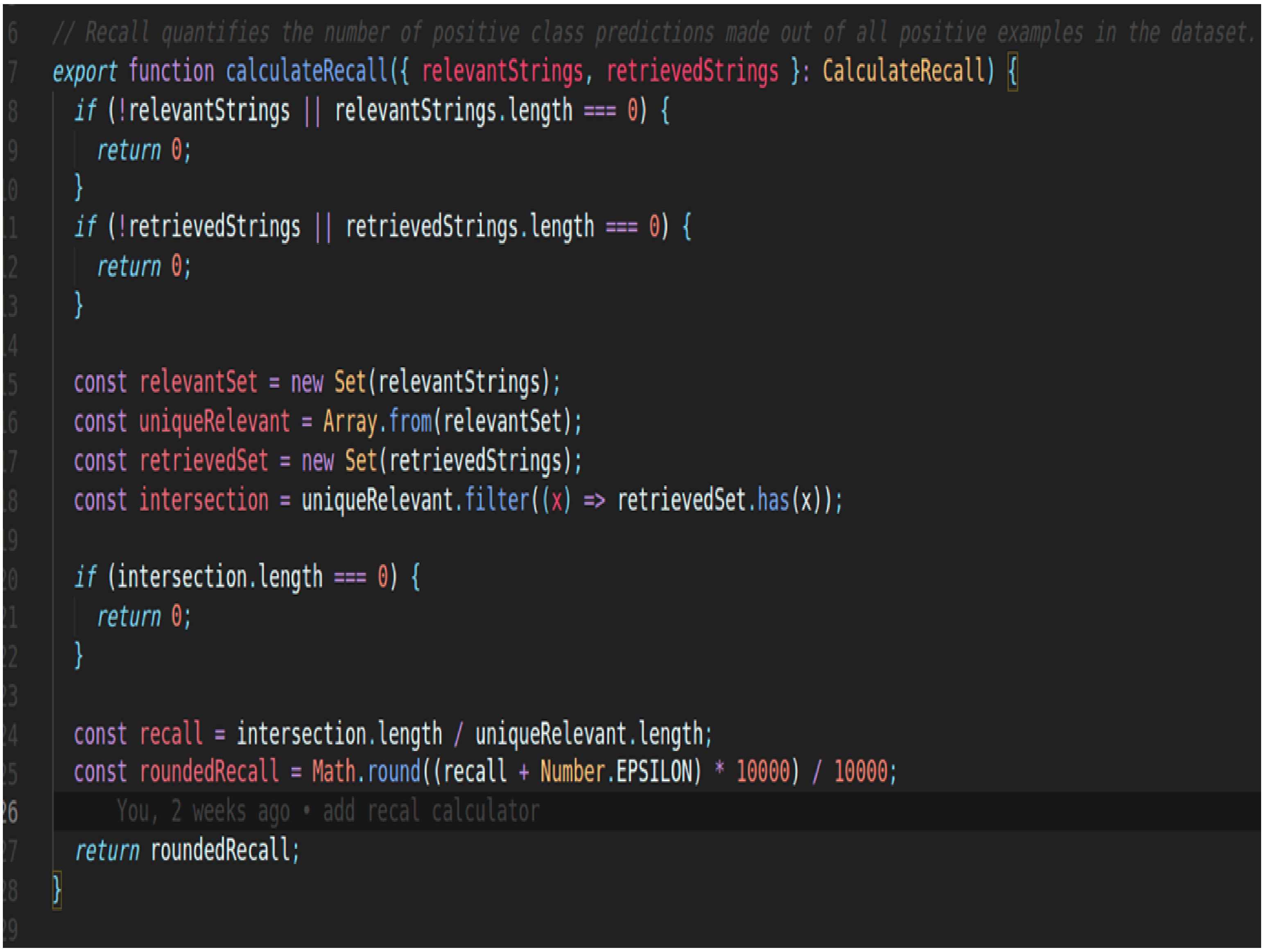
```
// Precision quantifies the number of positive class predictions that actually belong to the positive class.
export function calculatePrecision({ relevantStrings, retrievedStrings }: CalculatePrecision) {
  if (!relevantStrings || relevantStrings.length === 0) {
    return 0;
  }

  if (!retrievedStrings || retrievedStrings.length === 0) {
    return 0;
  }

  // remove duplicates
  const relevantSet = new Set(relevantStrings);
  const uniqueRelevant = Array.from(relevantSet);
  const retrievedSet = new Set(retrievedStrings);
  const uniqueRetrieved = Array.from(retrievedSet);

  const intersection = uniqueRelevant.filter((x) => retrievedSet.has(x));

  if (intersection.length === 0) {
    return 0;
  }

  const precision = intersection.length / uniqueRetrieved.length;
  const roundedPrecision = Math.round((precision + Number.EPSILON) * 10000) / 10000;

  return roundedPrecision;
}
```

Figure 2.7: Precision calculation implementation

- **Recall** Quantifies the number of relevant titles selected by the retrieval system, already discussed in section 1.7.3. Answering the question, ”how many relevant documents are selected?”. It was calculated using the equation 2.2 [20].

$$Recall = \frac{True\,Positive}{True\,Positive + False\,Negative} \tag{2.2}$$

The implementation code for recall is in figure 2.8.



```
// Recall quantifies the number of positive class predictions made out of all positive examples in the dataset.
export function calculateRecall({ relevantStrings, retrievedStrings }: CalculateRecall) {
  if (!relevantStrings || relevantStrings.length === 0) {
    return 0;
  }
  if (!retrievedStrings || retrievedStrings.length === 0) {
    return 0;
  }

  const relevantSet = new Set(relevantStrings);
  const uniqueRelevant = Array.from(relevantSet);
  const retrievedSet = new Set(retrievedStrings);
  const intersection = uniqueRelevant.filter((x) => retrievedSet.has(x));

  if (intersection.length === 0) {
    return 0;
  }

  const recall = intersection.length / uniqueRelevant.length;
  const roundedRecall = Math.round((recall + Number.EPSILON) * 10000) / 10000;
      You, 2 weeks ago • add recal calculator
  return roundedRecall;
}
```

Figure 2.8: Recall calculation implementation

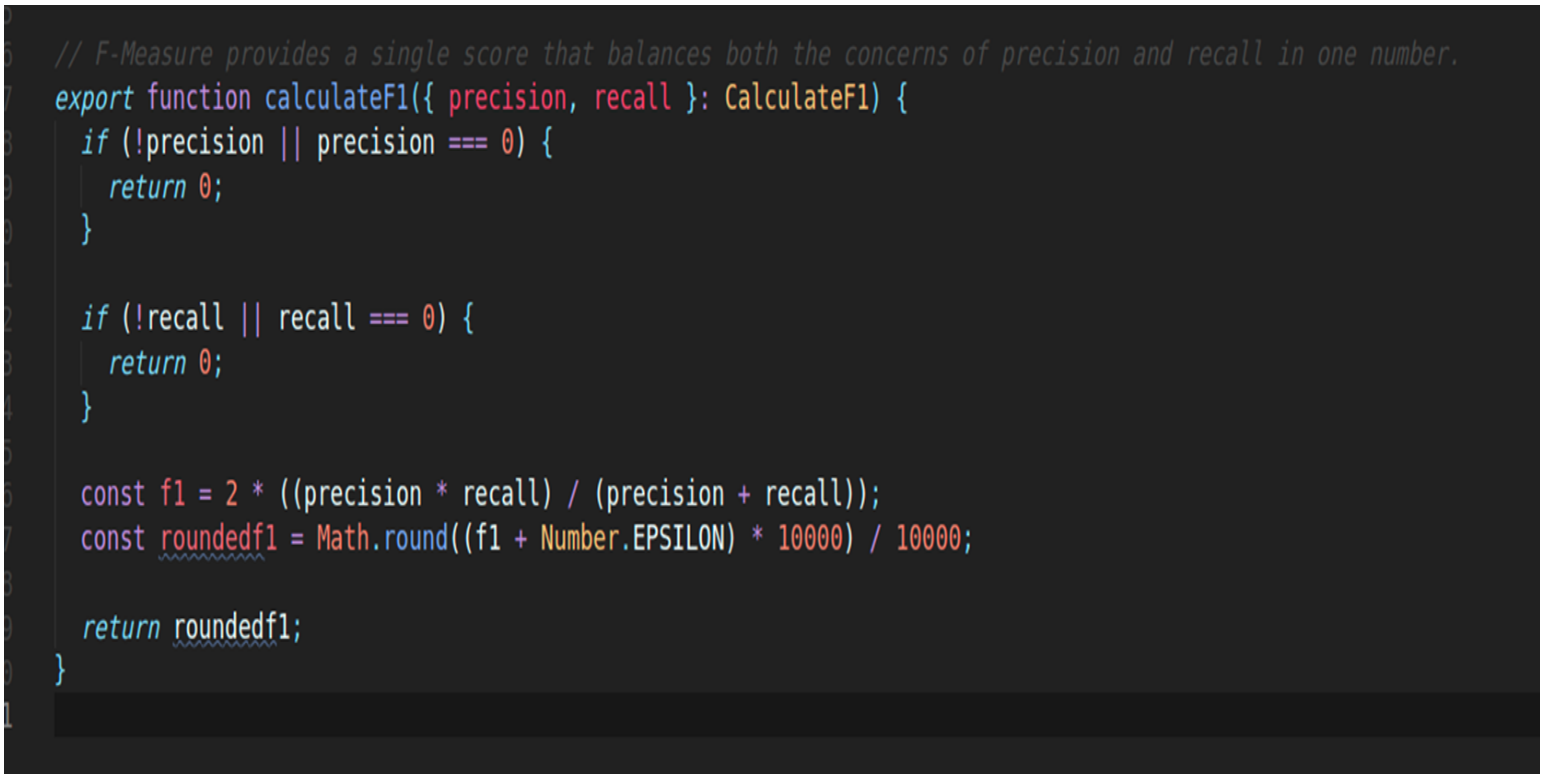
- **F1 Score** The most important measure is F-Measure. F1 Score provides a single score that balances both the concerns of precision and recall in one number. Answering the question ”Is the system a good enough model to solve the problem?”. It was calculated using the equation 2.3 [20].

$$F_1 = 2 \,.\, \frac{precision\,.recall}{precision + recall} = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \tag{2.3}$$

Where :

$$TP = True\,Positive = Relevant\,titles\,selected\,by\,the\,search\,engine$$
$$FN = False\,Negative = Relevant\,titles\,not\,selected\,by\,the\,search\,engine$$
$$FP = False\,Positive = None\,of\,the\,relevant\,titles\,selected\,by\,the\,search\,engine$$

The implementation code for recall is in figure 2.9.

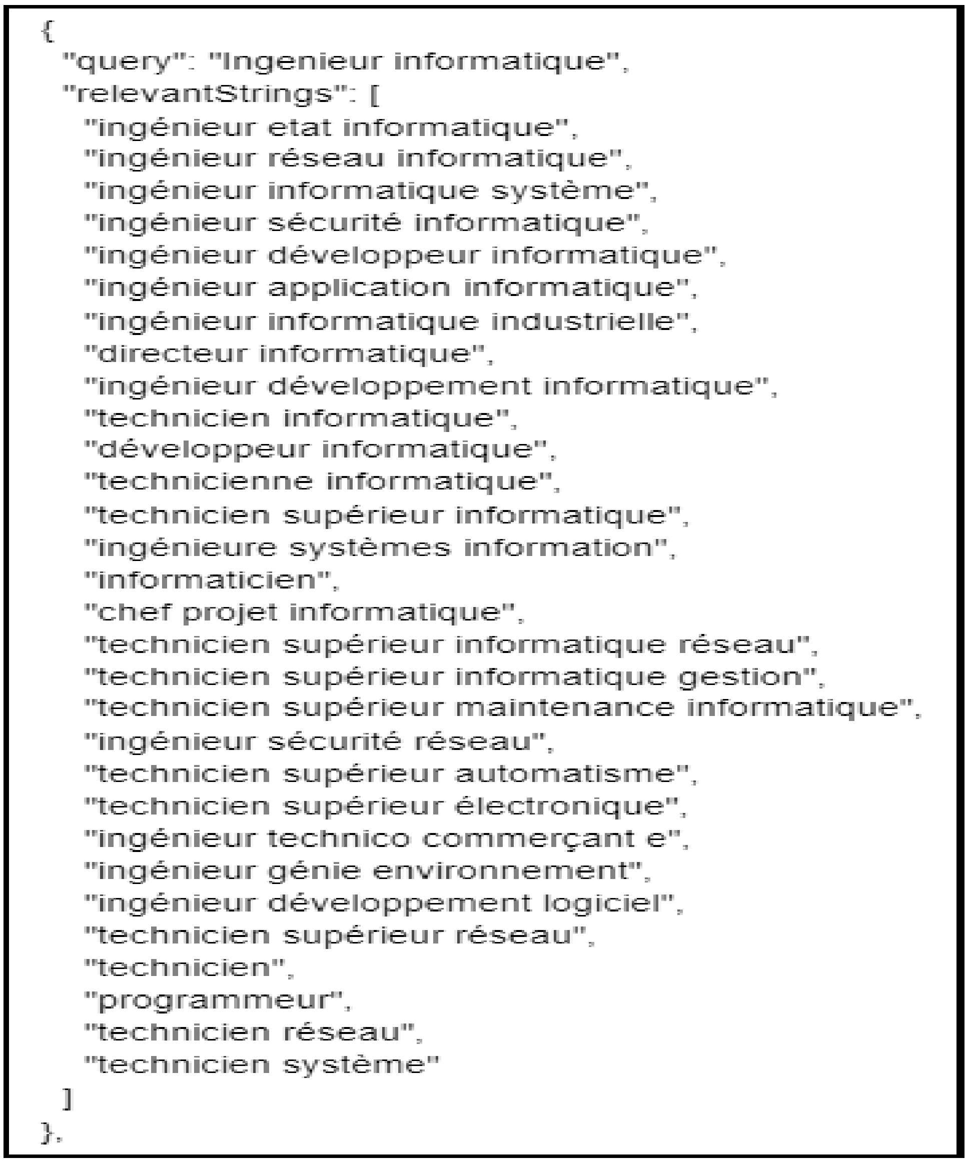


```
// F-Measure provides a single score that balances both the concerns of precision and recall in one number.
export function calculateF1({ precision, recall }: CalculateF1) {
  if (!precision || precision === 0) {
    return 0;
  }

  if (!recall || recall === 0) {
    return 0;
  }

  const f1 = 2 * ((precision * recall) / (precision + recall));
  const roundedf1 = Math.round((f1 + Number.EPSILON) * 10000) / 10000;

  return roundedf1;
}
```

Figure 2.9: F1 score calculation implementation

2.3.8 Taking Measurements

In order to calculate performance metrics, we start by indexing a pre-defined dataset of job titles. second, carefully pick a set of queries to test against, finally, pick the set of relevant job titles available in the data set. The full list of job titles used is available in Appendix A. as well as the set of queries we tested against. In JSON format, the data is divided into 2 main fields, the “query” field, represent the query given to the search engine. And ”Relevant Strings” field, an array of relevant job titles present in the dataset for the query provided. An example object is present in figure 2.10.



```
{
  "query": "Ingenieur informatique",
  "relevantStrings": [
    "ingénieur etat informatique",
    "ingénieur réseau informatique",
    "ingénieur informatique système",
    "ingénieur sécurité informatique",
    "ingénieur développeur informatique",
    "ingénieur application informatique",
    "ingénieur informatique industrielle",
    "directeur informatique",
    "ingénieur développement informatique",
    "technicien informatique",
    "développeur informatique",
    "technicienne informatique",
    "technicien supérieur informatique",
    "ingénieure systèmes information",
    "informaticien",
    "chef projet informatique",
    "technicien supérieur informatique réseau",
    "technicien supérieur informatique gestion",
    "technicien supérieur maintenance informatique",
    "ingénieur sécurité réseau",
    "technicien supérieur automatisme",
    "technicien supérieur électronique",
    "ingénieur technico commerçant e",
    "ingénieur génie environnement",
    "ingénieur développement logiciel",
    "technicien supérieur réseau",
    "technicien",
    "programmeur",
    "technicien réseau",
    "technicien système"
  ]
},
```

Figure 2.10: Example Measurement Object Sent to The Backend.

2.3.9 Evaluator Service Architecture

Evaluator service is separated into 2 main parts Evaluator REST API built using Express js web framework and MongoDB database, the main role of it is to interact with the search engine by sending queries, then calculate and store the performance metrics as well as supplying this metric logs to the frontend on demand. Evaluator frontend built with React, Redux, and charts chartjs. The frontend application consumes the backend API to perform two core functionalities request a performance test or display metric logs in charts using a provided logs id. The complete architecture of the system is presented in figure 2.11.

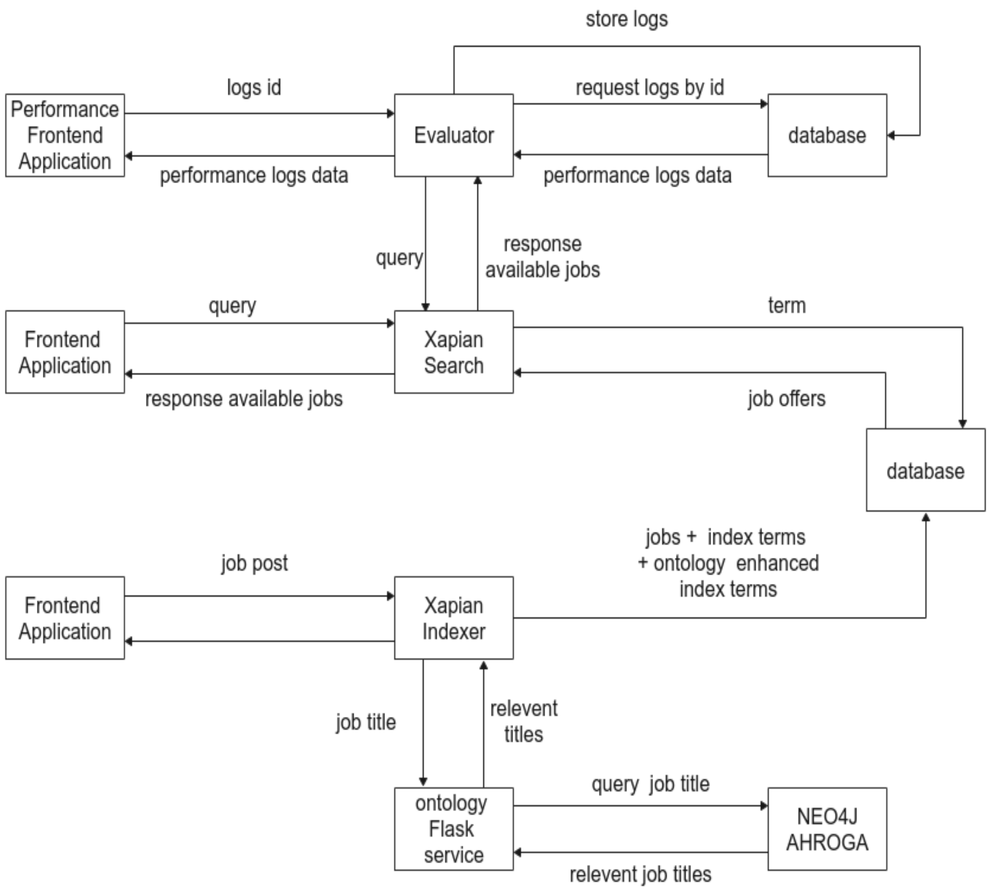


Figure 2.11: System architecture after the integration of the evaluator

The two core functionalities are displayed and explained in the two following section.

- **Initiate Performance Test** An evaluator user will initiate a performance test by providing unique logs id. And the data to be used to test. This data is passed to the Redux store which dispatches an action to request the correct POST endpoint on the Evaluator. The backend intercepts the request and proceeds to request relevant documents from the search engine using the query provided. The returned titles from the search engine alongside the relevant documents from the frontend are passed to the calculator util to generate the logs to be stored in the database. The architecture of this functionality is displayed in figure 2.12 and the code associated to this function is presented in figure 2.13.

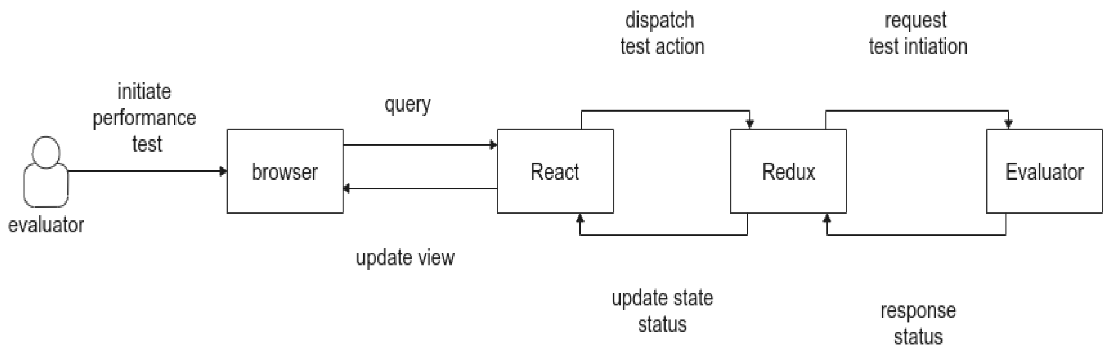
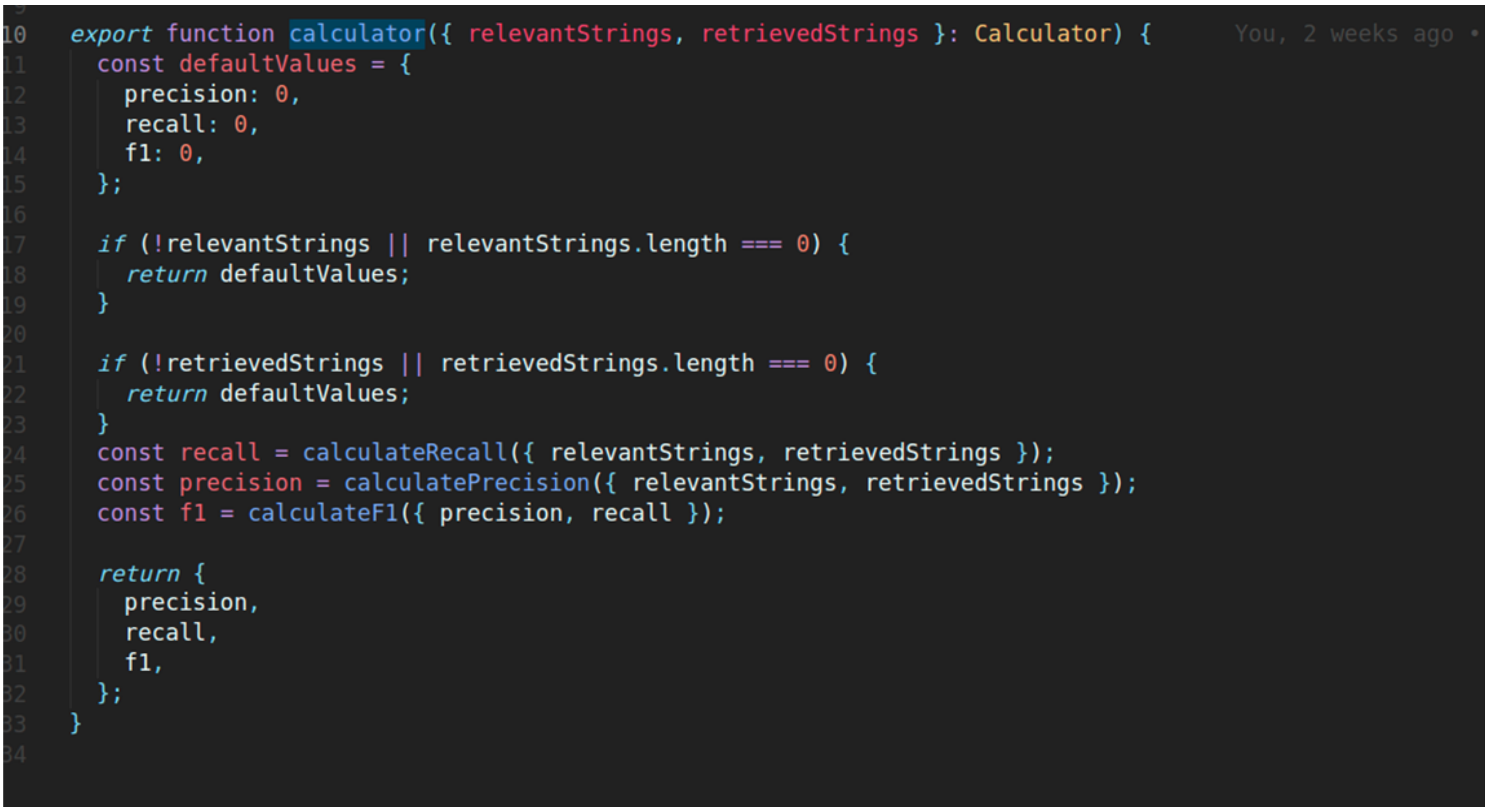


Figure 2.12: Architecture of Evaluator Initiate Performance Test.



```
export function calculator({ relevantStrings, retrievedStrings }: Calculator) {      You, 2 weeks ago •
  const defaultValues = {
    precision: 0,
    recall: 0,
    f1: 0,
  };

  if (!relevantStrings || relevantStrings.length === 0) {
    return defaultValues;
  }

  if (!retrievedStrings || retrievedStrings.length === 0) {
    return defaultValues;
  }
  const recall = calculateRecall({ relevantStrings, retrievedStrings });
  const precision = calculatePrecision({ relevantStrings, retrievedStrings });
  const f1 = calculateF1({ precision, recall });

  return {
    precision,
    recall,
    f1,
  };
}
```

Figure 2.13: Metrics Calculator Function Implementation.

• **Get Performance Logs** An evaluator user will initiate a get performance logs by providing logs id. That was provided on the performance test initiation. This data is passed to the Redux store which dispatches an action to request the correct POST endpoint on the Evaluator. The backend intercepts the request and proceeds to request metrics logs from the database. These logs are giving back to the frontend application. In the response body. Redux will then update the store state accordingly and React will render a new view to display the charts using newly fetched logs.The architecture of this functionality is displayed in figure 2.14 and the frontend associated to this function is presented in figure 3.4.

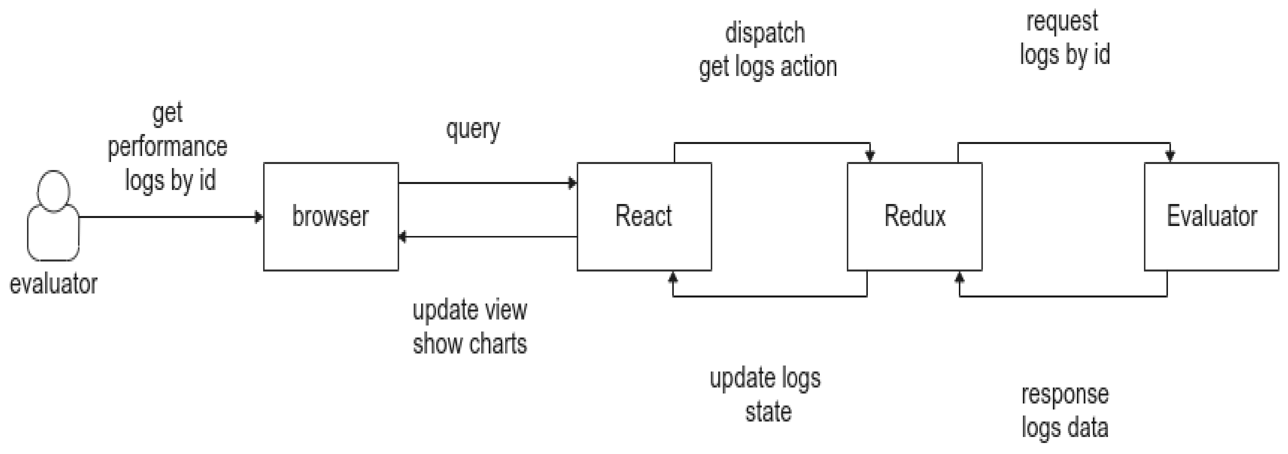


Figure 2.14: Architecture of Evaluator Get Performance Logs

2.4 Summary

Through this second chapter, we described the old version of the search engine in terms of architecture, the main functionalities of the Xapian core and the currently used frontend views. Moreover, we detailed on the improved version of the system and the two services we contributed with. The first service being ontology service, where we explained our choice of technologies and the architecture. The second service is evaluator, which generates the performance metrics discussed in chapter one. These metrics, the evluator service itself and the ontology query results will be used in the third and last chapter.

Chapter 3

Results and Discussion

3.1 Introduction

This chapter will discuss the results of integrating the new ontology service with the help of the charts drawn by the evaluator service and observe the impact of our contributions on the system in terms of user experience and developer experience. We will also, elaborate on the final improved system architecture after integrating the services, as well as, illustrate the different views of the newly developed frontend application. We conclude by mentioning the limitation of our implementation and suggest some solutions to handle them in future works

3.2 Developer Experience Contributions

To improve developer experience, Evaluator Service add the ability to monitor and log performance metrics namely: precision, recall, F1 score and job title query requested. As well as relevant and retrieved strings of emploitic search system. In contrast to old system. The ability to store logs and map them on Evaluator front-end application as performance charts allowed for an easier tracking of the effects of integrating new features and services, Facilitating the improvement of the system. As well as providing a clear picture of the state of old system. The system integration allowed for tracking each new deployment as a unit on an isolated chart as shown in figure 3.1. helping as well as performing contrast charts between different units shown in figure 3.2. Storing the result of the query as well as the relevant string set used to calculate the performance helped debug the system. By either tweaking the system or correcting Evaluator string data-set.

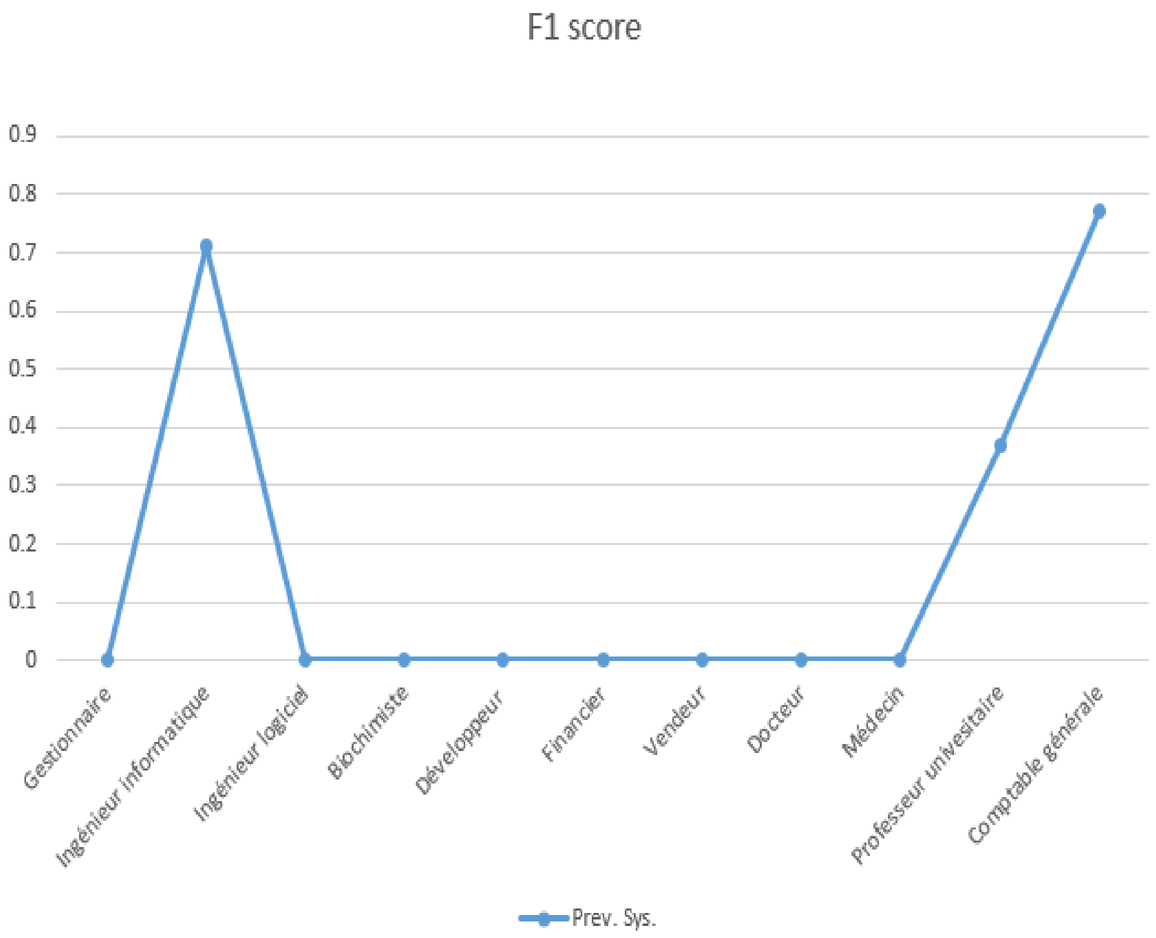


Figure 3.1: F1 score of the previous system

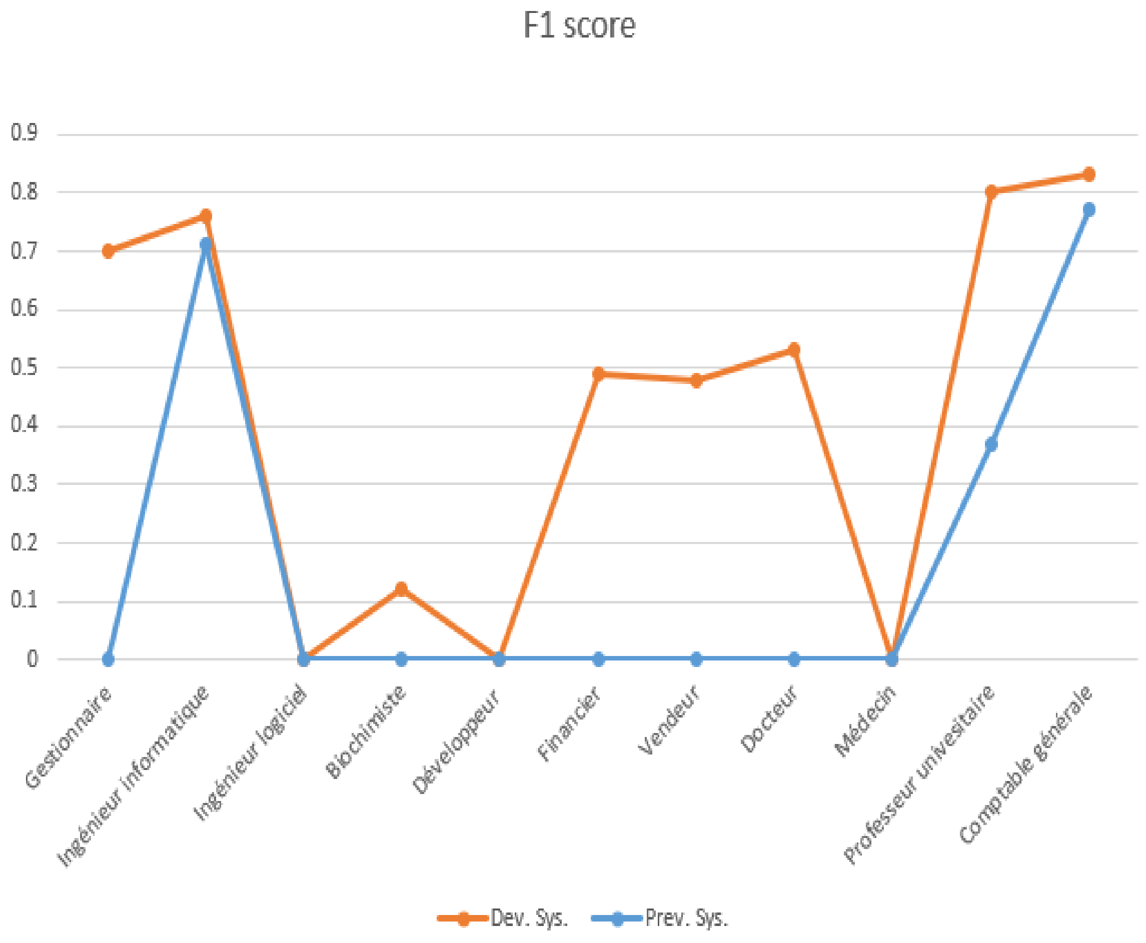


Figure 3.2: F1 scores contrasted of previous and developed system

Evaluator documents data-set directly affect the end result of the logs as the relevance of the documents retrieved is a subjective matter by nature. Its must be pointed out that this data-set need to be expanded for a more reliable measurements. Due to designing system architecture with scalability and maintainability in mind leads to making decisions to use of micro-services approach shown in figure 3.3, each service application can be isolated, deployed, and managed into a docker container. that can be easily spine up and down depending on the demand and network traffic. The approach reduced resource consumption. And eliminate the dependency between different services. Adhering to separation of concerns principle. Further more, Maintaining unit, acceptance tests using jest framework for front-end application and python unit-testing library for back-end applications and documentation improved time to market with easier integration of new features.

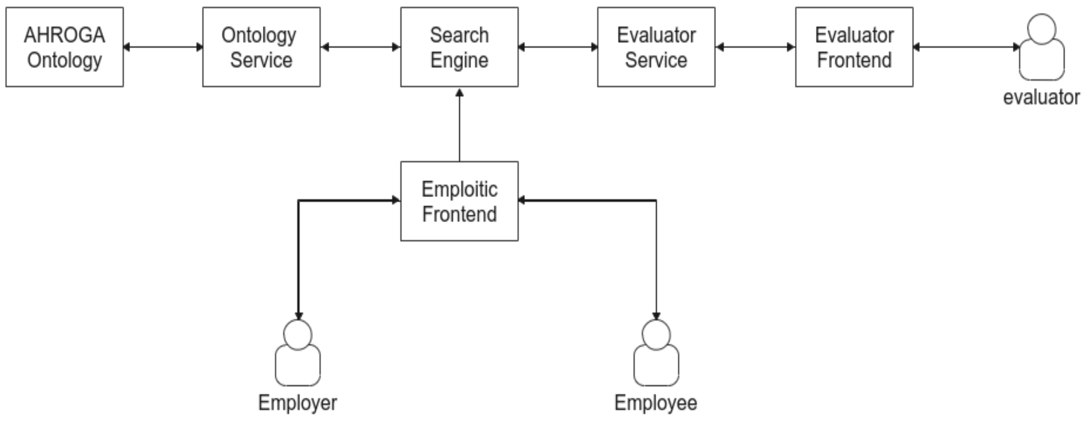


Figure 3.3: Services of the developed system

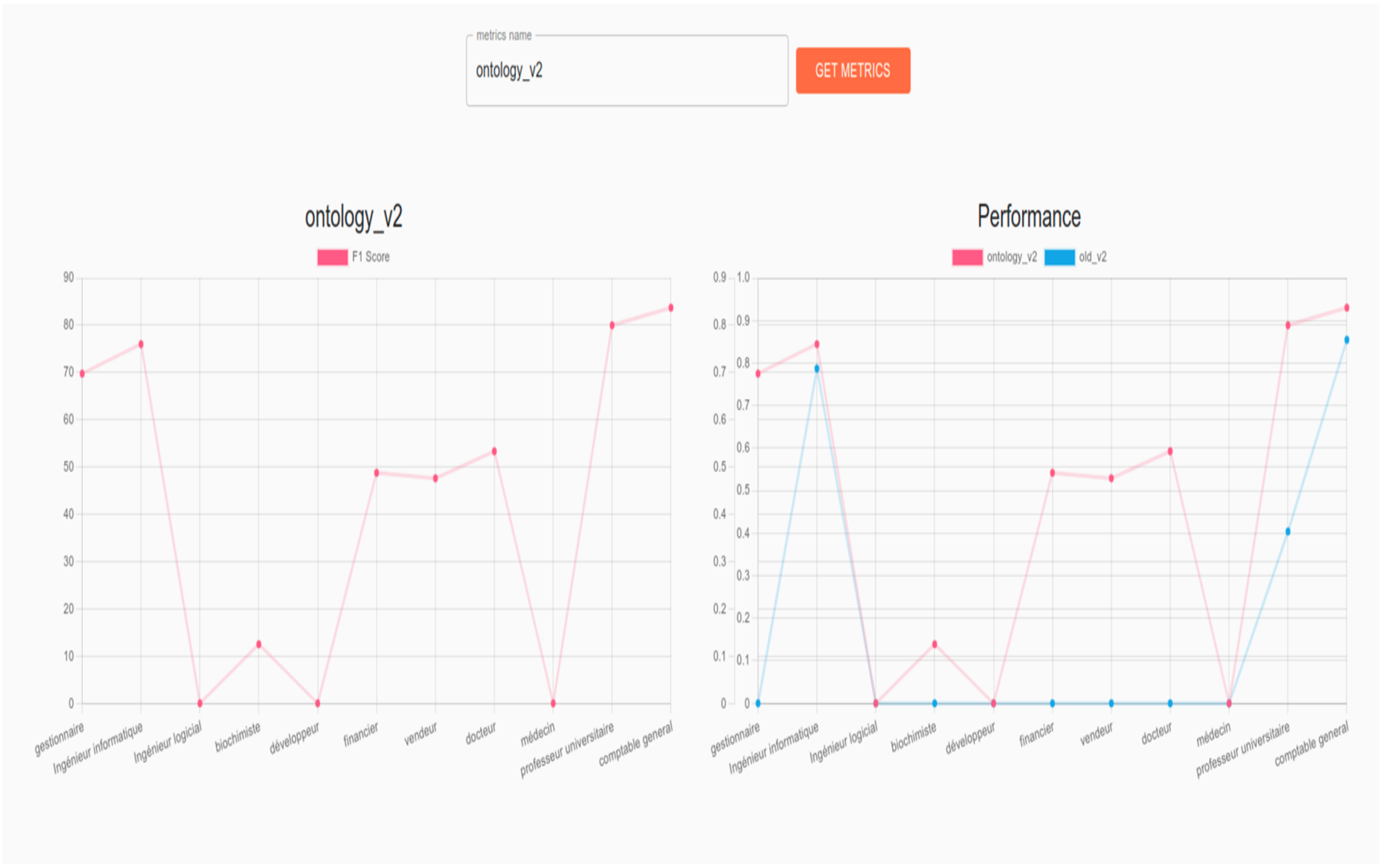
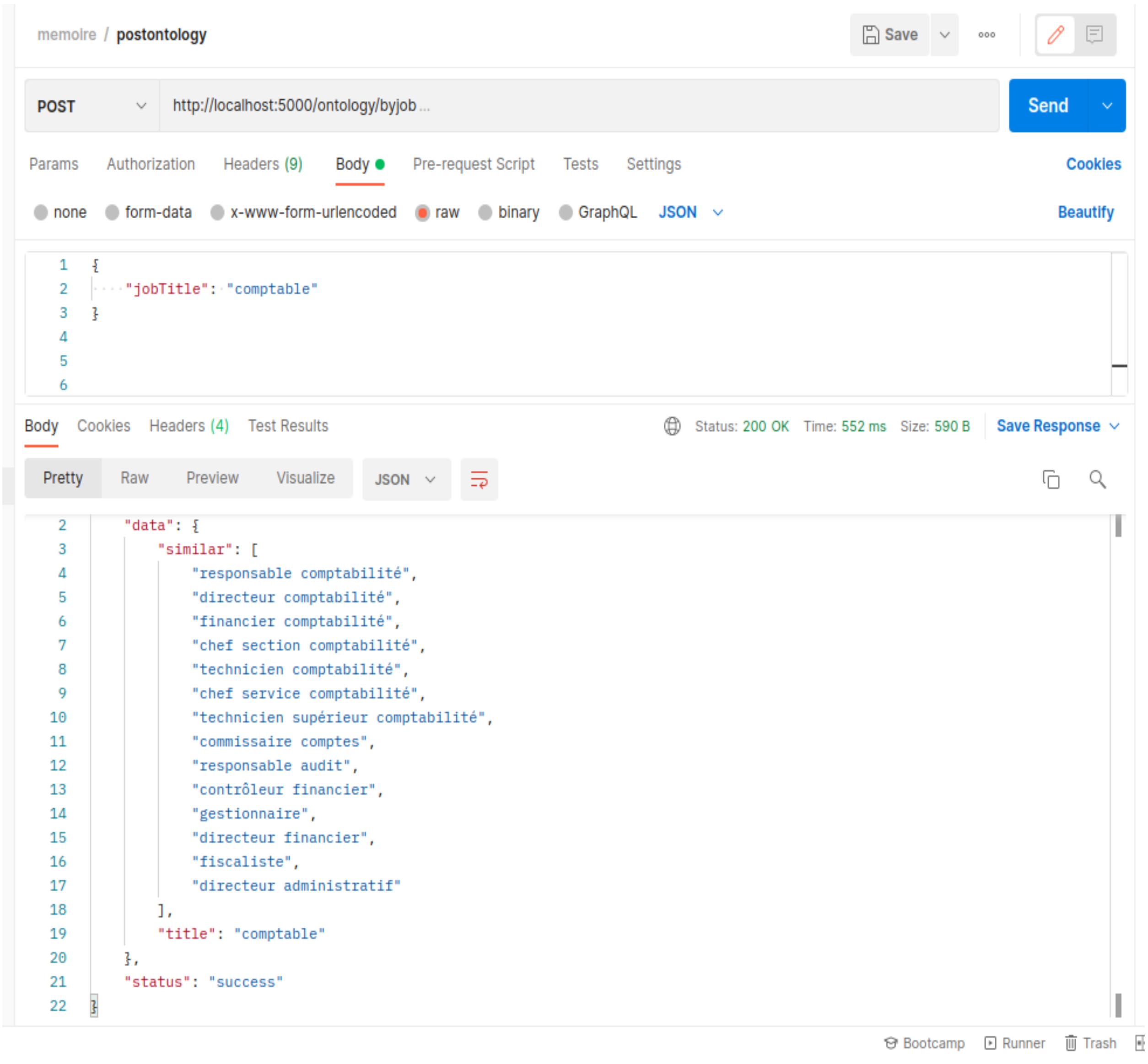


Figure 3.4: Evaluator Get performance Logs Frontend

3.3 User Experience Contributions

To improve the user experience at emploitic.com. Ontology Service was introduced operating at the indexing stage. By interfacing the search engine indexing class to AHROGA ontology using a Flask Restful API. Job titles retrieved from the database were injected in the job titles string. newly generated terms improved the retrieval system F1 score performance by a variant percentage depending on the job title. Reaching up to + %60 improvements compared to old solely probabilistic retrieval system. Leading to much more relevant job offers. As the Ontology Service only affects the indexing part of the search engine, search functionality and retrieval of documents performance remains the same.

Job titles retrieved by ontology service are very reliable as shown in figure 3.5 we notice The result of the query to the data base returned a higher percentage of relevant job titles to the query provided by the consumer. However this is not always the case. Irrelevant job titles returned by AHROGA will also be injected and indexed. The issue is outside the scope of this project. However the query performed by ontology services against as AHROGA database can be improved in the future.

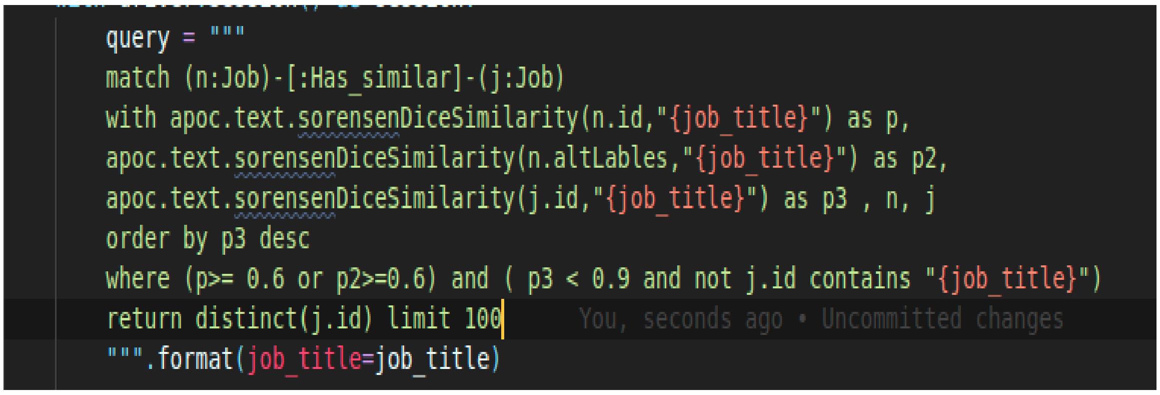


```
{
    "jobTitle": "comptable"
}
```

```
    "data": {
        "similar": [
            "responsable comptabilité",
            "directeur comptabilité",
            "financier comptabilité",
            "chef section comptabilité",
            "technicien comptabilité",
            "chef service comptabilité",
            "technicien supérieur comptabilité",
            "commissaire comptes",
            "responsable audit",
            "contrôleur financier",
            "gestionnaire",
            "directeur financier",
            "fiscaliste",
            "directeur administratif"
        ],
        "title": "comptable"
    },
    "status": "success"
}
```

Figure 3.5: Job titles retrieved by the ontology

The **query** is written as shown in figure 3.6, where it will search all similar jobs where the title is syntactically close to the job title (p), or its alternative label is similar to the job (p2) but doesn't directly contain the word job title (p3). The results are limited to 100 distinct similar job titles.

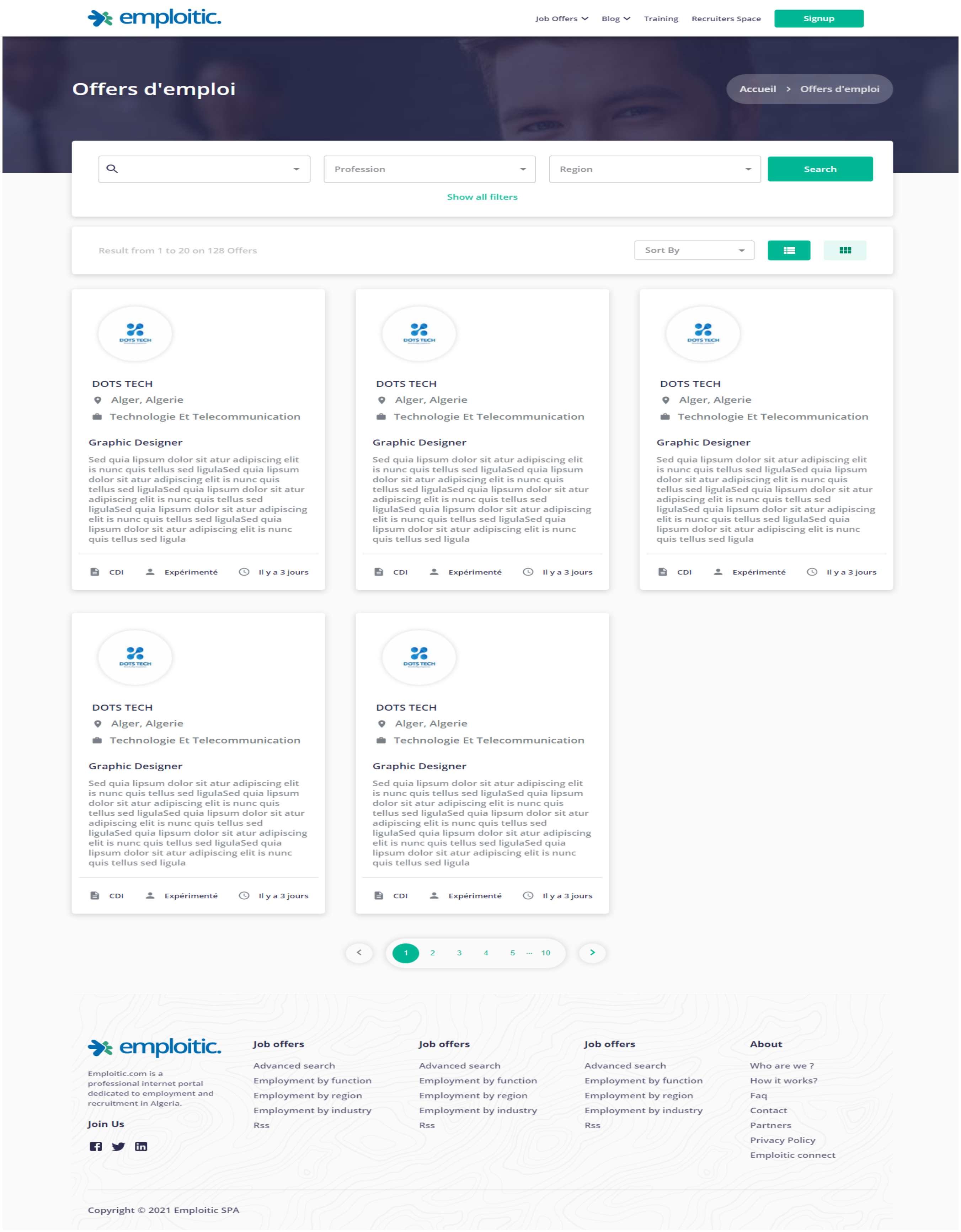


```
    query = """
    match (n:Job)-[:Has_similar]-(j:Job)
    with apoc.text.sorensenDiceSimilarity(n.id,"{job_title}") as p,
    apoc.text.sorensenDiceSimilarity(n.altLables,"{job_title}") as p2,
    apoc.text.sorensenDiceSimilarity(j.id,"{job_title}") as p3 , n, j
    order by p3 desc
    where (p>= 0.6 or p2>=0.6) and ( p3 < 0.9 and not j.id contains "{job_title}")
    return distinct(j.id) limit 100      You, seconds ago • Uncommitted changes
    """.format(job_title=job_title)
```

Figure 3.6: Query written in cypher language

The function apoc.text.sorensenDiceSimilarity is provided by Neo4j, it uses the Sørensen–Dice coefficient which is statistic used to gauge the similarity of two samples. It was independently developed by the botanists Thorvald Sørensen and Lee Raymond Dice [21]. p, p2 and p3 are used to limit the scope of the similarity. Where p, p2 are set to 0.6 to not diverge from the initial job title yet gives flexibility to return all syntactically similar words. In addition, p3 is set to 0.9 to eliminate the presence of directly related title as this feat is already done by the previous system indexer.

One must note, Ontology Service only affects the indexing phase of the search engine, search functionality and retrieval of documents speed remains the same as the old system. Lastly the rework of the front-end application help made user interaction with the system much easier. As results can be easily filtered. By reducing the complexity of filtering to profession and region and hiding away other filters. Separating job offers into different cards provide the sense of simplicity, dealing with each job offer as a separate unit compared to old way of cramping set of many job offers into one big card shown in figures 3.7 and 3.8. Further more, the use of React lazy loading pages and Webpack spiting the application into multiple chunks and serving each part of the application only when requested by the user. Improved pages load time as well as application.


emploitic.

Job Offers · Blog · Training · Recruiters Space · Signup

Offers d'emploi

Accueil > Offers d'emploi

Profession · Region · Search

Show all filters

Result from 1 to 20 on 128 Offers

Sort By

DOTS TECH
Alger, Algerie
Technologie Et Telecommunication

Graphic Designer

Sed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligula

CDI · Expérimenté · Il y a 3 jours

DOTS TECH
Alger, Algerie
Technologie Et Telecommunication

Graphic Designer

Sed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligula

CDI · Expérimenté · Il y a 3 jours

DOTS TECH
Alger, Algerie
Technologie Et Telecommunication

Graphic Designer

Sed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligula

CDI · Expérimenté · Il y a 3 jours

DOTS TECH
Alger, Algerie
Technologie Et Telecommunication

Graphic Designer

Sed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligula

CDI · Expérimenté · Il y a 3 jours

DOTS TECH
Alger, Algerie
Technologie Et Telecommunication

Graphic Designer

Sed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligula

CDI · Expérimenté · Il y a 3 jours

< 1 2 3 4 5 ... 10 >

emploitic.

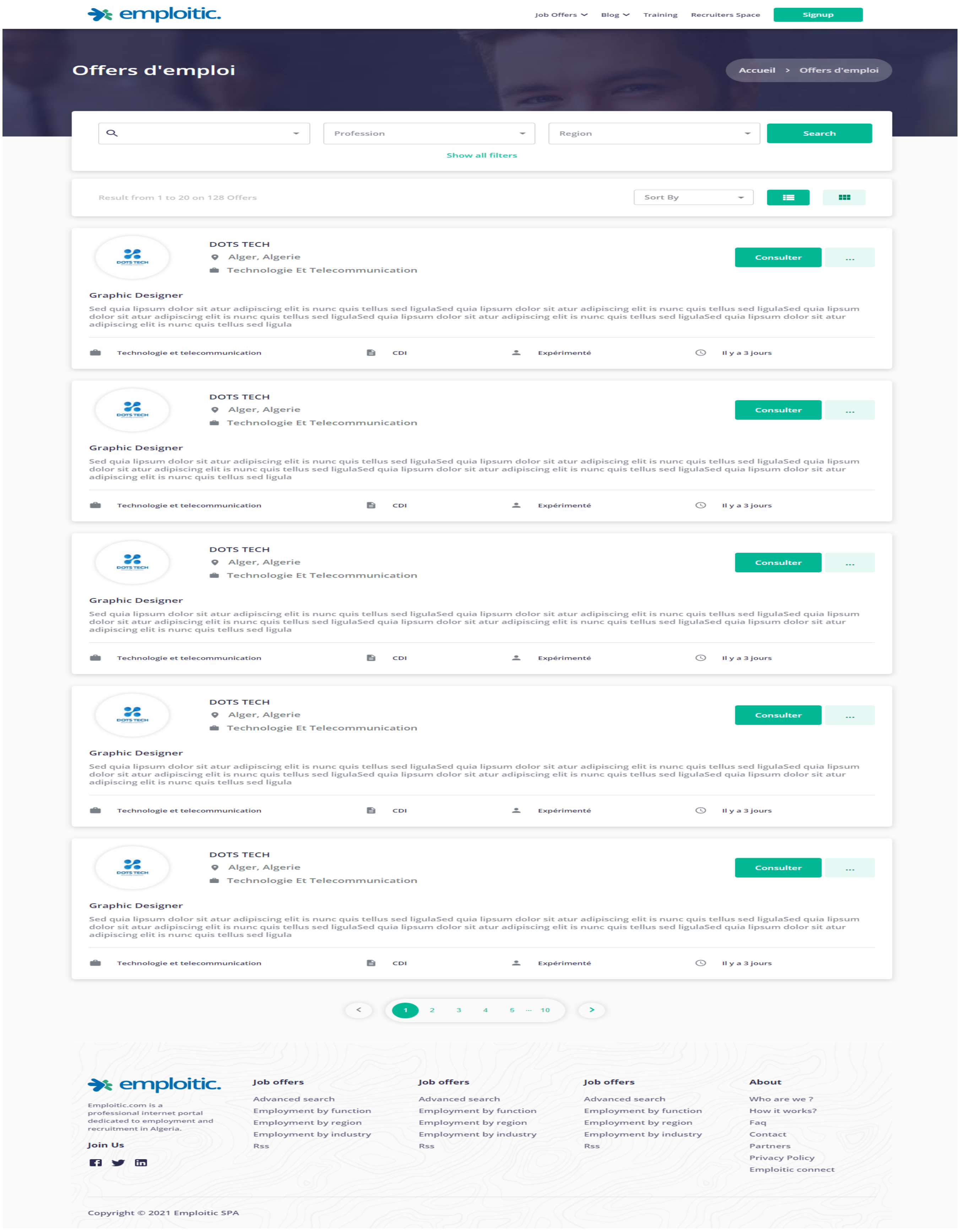
Emploitic.com is a professional internet portal dedicated to employment and recruitment in Algeria.

Join Us

Job offers	Job offers	Job offers	About
Advanced search	Advanced search	Advanced search	Who are we ?
Employment by function	Employment by function	Employment by function	How it works?
Employment by region	Employment by region	Employment by region	Faq
Employment by industry	Employment by industry	Employment by industry	Contact
Rss	Rss	Rss	Partners
			Privacy Policy
			Emploitic connect

Copyright © 2021 Emploitic SPA

Figure 3.7: Grid View


emploitic.

Job Offers ⌄ Blog ⌄ Training Recruiters Space Signup

Offers d'emploi

Accueil > Offers d'emploi

Profession

Region

Search

Show all filters

Result from 1 to 20 on 128 Offers

Sort By

DOTS TECH
Alger, Algerie
Technologie Et Telecommunication

Consulter ...

Graphic Designer

Sed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligula

Technologie et telecommunication CDI Expérimenté Il y a 3 jours

DOTS TECH
Alger, Algerie
Technologie Et Telecommunication

Consulter ...

Graphic Designer

Sed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligula

Technologie et telecommunication CDI Expérimenté Il y a 3 jours

DOTS TECH
Alger, Algerie
Technologie Et Telecommunication

Consulter ...

Graphic Designer

Sed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligula

Technologie et telecommunication CDI Expérimenté Il y a 3 jours

DOTS TECH
Alger, Algerie
Technologie Et Telecommunication

Consulter ...

Graphic Designer

Sed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligula

Technologie et telecommunication CDI Expérimenté Il y a 3 jours

DOTS TECH
Alger, Algerie
Technologie Et Telecommunication

Consulter ...

Graphic Designer

Sed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligulaSed quia lipsum dolor sit atur adipiscing elit is nunc quis tellus sed ligula

Technologie et telecommunication CDI Expérimenté Il y a 3 jours

< 1 2 3 4 5 ... 10 >

emploitic.

Emploitic.com is a professional internet portal dedicated to employment and recruitment in Algeria.

Join Us

Job offers
Advanced search
Employment by function
Employment by region
Employment by industry
Rss

Job offers
Advanced search
Employment by function
Employment by region
Employment by industry
Rss

Job offers
Advanced search
Employment by function
Employment by region
Employment by industry
Rss

About
Who are we ?
How it works?
Faq
Contact
Partners
Privacy Policy
Emploitic connect

Copyright © 2021 Emploitic SPA

Figure 3.8: List View

3.4 Summary

In this chapter, we showcase our contribution to the Emploitic search system and analyze the results after integrating the ontology service discussed in the second chapter using the charts generated by the evaluator service we developed in the previous chapter. We also classify the benefits of the contributions highlighted in terms of user experience and developer experience.

3.5 General Conclusion and Future Work

In this report, we presented the theory, design, and implementation of different information retrieval models. We covered the design of various well-known approaches suggested by books and papers in improving the relevance of the results of a search engine. We argue about the implementation of the ontological approach at emploitic.com and the benefits of developing a knowledge manager. We also introduce a new automatic evaluation tool for monitoring the performance of the search system. Because the system is meant to be used at high traffic loads, it should be emphasized that the primary focus at the development time is the scalability and maintainability system which heavily affected the choices of design, architecture, and technologies used. Even though we could not share the system's code as it is Emploitic's proprietary software, we followed the rules of clean code and SOLID principles [22]. In addition to implementing automated unit and acceptance tests.

As a result of time constraints, not all of the planned work is implemented. However, we will mention a few key points for future improvements. First, the use of the Flask web framework and Neo4j database for the ontology service lead to a long indexing time. Although this does not affect the user experience directly, it can be improved by using technologies with native concurrency support like the Tornado web framework and Arangodb. Second, ontology is the source of truth for the ontology service. Improving AHROGA or the query used to fetch job titles by the ontology service will directly result in an increase in the relevance of the documents retrieved by Xapian search. Furthermore, it should be pointed out that retrieved results show that job titles including special characters or the use of short vague job titles raise a challenge to the underlined system. Meaning that the generated index terms were negatively affected by the stemming of the Xapian index term generator. Finally, this project only discussed modifications on the indexing process leaving the search process without any edits. We suggest the editing of the weighting scheme of the search engine and the rework of query generation at the Xapian search level.

In the end, one shall note that this work has been adopted and integrated by Emploitic. The ontology service will be integrated into the search engine indexer and the evaluator service will be released as an open-source tool to monitor information retrieval systems. The evaluator service is used to help monitor and collect instant feedback on each modification on the system. Furthermore, the evaluator will also be released as an open-source software hosted in a GitHub repository by dzCode.com open-source community.

References

[1] Borko, H. . *Information science: What is it? American Documentation*, 1968
[2] Amit Singhal, *Modern Information Retrieval: A Brief Overview*, 2001
[3] Salton, Automatic Information Organization and Retrieval, 1968.
[4] Stephen E. Robertson, The probability ranking principle in information retrieval, 1977
[5] G. KOWALSKI, Information Retrieval Systems: Theory and Implementation, Kluwer Academic Publishers, Boston, MA, 1997
[6] Marchionini G. MARCHIONINI, *Information Seeking in Electronic Environments*, 1995
[7] R.R.KORFHAGE, *Information Storage and Retrieval*, 1997
[8] Bruce Croft,Donald Metzler,Trevor Strohman, *Search Engines Information Retrieval in Practice*, 2010.
[9] Luciano Floridi , *Blackwell Guide to the Philosophy of Computing and Information*, 2003
[10] Gruber, T.R. . *A Translation Approach to Portable Ontology Specification*, 1993
[11] Natalya F. Noy and Deborah L. McGuinness. *Ontology Development 101: A Guide to Creating Your First Ontology.* 2001.
[12] Siricharoen, *Ontologies and Object models in Object Oriented Software Engineering* ,2007
[13] S.K. Sahu, D. Mahapatra, R. Balabantaray, *Comparative Study of Search Engines In Context Of Features And Semantics,* , 2016.
[14] R. R. Zebari, S. R. M. Zeebaree, K. Jacksi, *Impact Analysis of HTTP and SYN Flood DDoS Attacks on Apache 2 and IIS 10.0 Web Servers,*, 2018
[15] M.M. Qureshi, B. Asma, H.U. Khan, *Comparative analysis of semantic search engines based on requirement space pyramid," International Journal of Future Computer and Communication*, 2013.
[16] Bright, Peter. *Microsoft TypeScript: the JavaScript we need, or a solution looking for a problem?.* Ars Technica. Condé Nast, 2012
[17] Nitin Pandit, *What And Why React.js* , 2021

[18] Stephen Robertson1 and Hugo Zaragoza, *The Probabilistic Relevance Framework:BM25 and Beyond*, 2009
[19] Sabrina Boudjedar1, Sihem Bouhenniche1, Hakim Mokeddem1, and Hamid Benachour, *Automatic Human Resources ontology generation from the data of an e-recruitment platform*, 2019
[20] Rodolfo Bonnin, *Machine Learning for Developers*, 2017.
[21] Sørensen, T. . *A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons*, 1948.
[22] Robert Cecil Martin, *Clean Code*, 2008.

Appendix A

List of Job Titles In JSON Format

titles": ["cadre gestion", "chargé gestion", "contrôleur gestion", "assistante gestion", "responsable gestion stocks", "administrateur ressources humaines", "superviseur opérations", "technicien supérieur administration", "cadre ressources humaines", "chargée ressources humaines", "manager projet", "manager qualité", "project manager", "agent administratif", "cadre administratif", "responsable ressources humaine", "directeur financier", "directrice ressources humaines", "responsable ressources humaines", "spécialiste ressources humaines", "généraliste ressources humaines", "contrôleur production", "superviseur production", "ingénieur etat informatique", "ingénieur réseau informatique", "ingénieur informatique système", "ingénieur sécurité informatique", "ingénieur développeur informatique", "ingénieur application informatique", "ingénieur informatique industrielle", "directeur informatique", "ingénieur développement informatique", "ingénieur climatique", "technicien informatique", "développeur informatique", "technicienne informatique", "technicien supérieur informatique", "ingénieure systèmes information", "ingénieur génie climatique", "informaticien", "chef projet informatique", "technicien supérieur informatique réseau", "technicien supérieur informatique gestion", "technicien supérieur maintenance informatique", "ingénieur sécurité réseau", "technicien supérieur automatisme", "technicien supérieur électronique", "technicien mécanique", "ingénieur technico commerçant e", "ingénieur génie environnement", "ingénieur développement logiciel", "technicien supérieur réseau", "technicien", "programmeur", "technicien réseau", "technicien système", "ingénieur logiciel", "ingénieur génie logiciel", "ingénieur biomédical", "coordinateur logisticien", "ingénieur biotechnologie", "logisticien", "directeur logisticien", "superviseur logisticien", "ingénieur systèmes", "ingénieur informatique", "responsable logisticien", "microbiologiste", "responsable microbiologie", "biochimiste", "analyste chimiste", "chimiste analyste", "chimiste industrielle", "chimiste pharmaceutique", "analyste physico chimique", "ingénieur chimie", "ingénieur chimie industrielle", "ingénieur génie chimique", "responsable physico chimie", "ingénieur chimie analytique", "ingénieur biologie", "developpeur", "développeur web", "developpeur web", "developpeur java", "développeur java", "développeur java ee", "développeur backend web", "développeur application", "développeur web fullstack", "ingénieur développement", "développeur fullstack java", "ingénieur developement", "ingénieur", "administrateur ventes", "ingénieur conception", "ingénieur consultant", "conseiller vente", "analyste", "chef projet web", "cadre finances", "analyste financière", "assistante financière", "comptable finance", "responsable finance", "administrateur finance", "comptable principal", "comptable principale", "chef comptable principal", "responsable trésorerie", "directeur agence bancaire", "gestionnaire", "directeur banque", "expert comptable", "gestionnaire entreprise", "directeur administratif", "directeur administration", "directeur comptabilité", "directeur général", "comptable", "aide comptable", "directeur opérations", "chef comptable", "comptable général", "superviseur ventes", "docteur pharmacie", "responsable ventes", "responsable service après vente", "commerçant", "visiteur médical", "superviseur médical", "ingénieur pharmacologique", "vendeuse pharmacie", "assistante médicale", "médecin généraliste", "délégué pharmaceutique", "déléguée médicale", "délégué hospitalier", "délégué médical", "docteur médecine", "docteur", "formatrice", "directrice", "administrateur", "professeur", "auditeur", "actuaire", "enseignant langue anglaise", "enseignant universitaire", "éducatrice", "enseignant", "enseignante", "cadre comptable", "comptable financier", "auditeur comptable", "cadre financier comptable", "assistante comptable", "responsable comptabilité", "responsable comptable financier", "financier comptabilité", "technicien comptabilité", "chef service comptabilité", "chef section comptabilité", "technicien supérieur comptabilité", "contrôleur financier"]

Appendix B

Data Used to Measure Performance In JSON format

"data": ["query": "assistante gestion", "relevantStrings": ["cadre gestion", "chargé gestion", "contrôleur gestion", "assistante gestion", "responsable gestion stocks", "administrateur ressources humaines", "superviseur opérations", "technicien supérieur administration", "cadre ressources humaines", "chargée ressources humaines", "manager projet", "manager qualité", "project manager", "agent administratif", "cadre administratif", "responsable ressources humaine", "directeur financier", "directrice ressources humaines", "responsable ressources humaines", "spécialiste ressources humaines", "généraliste ressources humaines", "contrôleur production", "superviseur production"] , "query": "Ingenieur informatique", "relevantStrings": ["ingénieur etat informatique", "ingénieur réseau informatique", "ingénieur informatique système", "ingénieur sécurité informatique", "ingénieur développeur informatique", "ingénieur application informatique", "ingénieur informatique industrielle", "directeur informatique", "ingénieur développement informatique", "technicien informatique", "développeur informatique", "technicienne informatique", "technicien supérieur informatique", "ingénieure systèmes information", "informaticien", "chef projet informatique", "technicien supérieur informatique réseau", "technicien supérieur informatique gestion", "technicien supérieur maintenance informatique", "ingénieur sécurité réseau", "technicien supérieur automatisme", "technicien supérieur électronique", "ingénieur technico commerçant e", "ingénieur génie environnement", "ingénieur développement logiciel", "technicien supérieur réseau", "technicien", "programmeur", "technicien réseau", "technicien système"] , "query": "Ingenieur logicial", "relevantStrings": ["ingénieur logiciel", "ingénieur génie logiciel", "ingénieur développement logiciel", "ingénieur biotechnologie", "ingénieur systèmes", "ingénieur informatique", "ingénieur etat informatique", "ingénieur développeur informatique", "ingénieur application informatique", "ingénieur développement in-

formatique", "ingénieur informatique système", "technicien informatique", "technicienne informatique", "ingénieur informatique industrielle", "programmeur", "développeur informatique"] , "query": "chimiste industrielle", "relevantStrings": ["analyste chimiste", "chimiste analyste", "chimiste industrielle", "chimiste pharmaceutique", "analyste physico chimique", "ingénieur chimie", "ingénieur chimie industrielle", "ingénieur génie chimique", "responsable physico chimie", "ingénieur chimie analytique", "ingénieur biologie"] , "query": "developpeur", "relevantStrings": ["developpeur", "développeur web", "developpeur web", "developpeur java", "développeur java", "développeur java ee", "développeur backend web", "développeur application", "développeur web fullstack", "ingénieur développement", "développeur fullstack java", "ingénieur developement", "ingénieur développeur informatique", "ingénieur", "ingénieur conception", "ingénieur consultant", "chef projet web"] , "query": "responsable finance", "relevantStrings": ["cadre finances", "analyste financière", "assistante financière", "comptable finance", "responsable finance", "administrateur finance", "comptable principal", "comptable principale", "chef comptable principal", "responsable trésorerie", "gestionnaire", "expert comptable", "gestionnaire entreprise", "directeur administratif", "directeur administration", "directeur comptabilité", "directeur général", "comptable", "aide comptable", "chef comptable", "comptable général"] , "query": "conseiller vente", "relevantStrings": ["superviseur ventes", "administrateur ventes", "conseiller vente", "responsable ventes", "responsable service après vente", "commerçant"] , "query": "assistante medicale", "relevantStrings": ["visiteur médical", "superviseur médical", "ingénieur pharmacologique", "vendeuse pharmacie", "assistante médicale", "médecin généraliste", "délégué pharmaceutique", "déléguée médicale", "délégué hospitalier", "délégué médical"] , "query": "superviseur medical", "relevantStrings": ["docteur médecine", "médecin généraliste", "déléguée médicale", "assistante médicale", "docteur", "superviseur médical", "délégué hospitalier", "délégué médical", "visiteur médical"] , "query": "professeur universitaire", "relevantStrings": ["formatrice", "directrice", "administrateur", "professeur", "enseignant langue anglaise", "enseignant universitaire", "éducatrice", "professeur", "enseignant", "enseignante"] , "query": "comptable general", "relevantStrings": ["comptable", "comptable général", "expert comptable", "aide comptable", "chef comptable", "cadre comptable", "comptable financier", "comptable principal", "comptable principale", "comptable finance", "auditeur comptable", "chef comptable principal", "cadre financier comptable", "assistante comptable", "responsable comptabilité", "responsable comptable financier", "financier comptabilité", "technicien comptabilité", "chef service comptabilité", "directeur comptabilité", "chef section comptabilité", "technicien supérieur comptabilité", "contrôleur financier"]]