EIRA

An Application for Finding and Ranking Researchers

Bachelor of Science Thesis in Computer Science

Hannes Häggander
Rasmus Letterkrantz
Fredrik Rahn
Erik Sievers
Pontus Thomé

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Gothenburg, Sweden, June 2016

EIRA
An Application for Finding and Ranking Researchers

HANNES HÄGGANDER
RASMUS LETTERKRANTZ
FREDRIK RAHN
ERIK SIEVERS
PONTUS THOMÉ

Supervisor: DEVDATT DUBHASHI
Examiner: NIKLAS BROBERG

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2016

Abstract

The purpose of the project is to deliver a proof of concept application that is expected to solve an issue related to finding suitable researchers for research groups. The system that organizations use, as of writing this report, is to post a request for application. This means that an organization published a request for research to be performed within a specific field and the budget attributed to the research project. Researchers then find the request and sends an application. We developed EIRA with the intention of changing this process and allow the organizations to find researchers that could be a good fit for these projects.

This report covers our development of the cognitive assistant EIRA (Entity Indexed Ranking Application). EIRA can find researchers related to the medical field. The researchers are extracted from different data sources and currently EIRA uses the Scopus database and Microsoft Academic. The application consists of four custom micro services built in Node.js using Promises. EIRA provides a user interface written in AngularJS to handle all interaction with the services.

Sammandrag

Projektets mål är att producera ett program som agerar konceptbevis till att lösa ett problem med att hitta forskare som kan delta i forskningsgrupper. I dagsläget så använder organisationer sig av en form av jobbannons för att komma i kontakt med forskare. Jobbannonsen innehåller information om vad forskningen ska behandla och besvara, vilket ämnesområde forskningen är inom och projektets budget. Forskare behöver sedan själva ansöka efter en plats på projektet. Vi utvecklade EIRA med avsikt att ändra denna processen genom att låta organisationer hitta forskare som skulle passa bra i projekten.

Rapporten beskriver utvecklingen av den kognitiva assistenten EIRA. EIRAs uppgift är att hitta forskare baserat på en sökning som sker antingen genom en befintlig jobbannons eller sökord. Forskare hämtas sedan från en mängd datakällor. Datakällorna som EIRA använder är databasen Scopus och Microsoft Academic. EIRA är uppbyggd av ett flertal micromoduler som är utvecklade i Node.js med Promises. Applikationen erbjuder ett användargränssnitt som är utvecklat i AngularJS för att hantera användarens interaktionen med de tidigare nämnda micromodulerna.

Vocabulary

API
Application Program Interface, the interface a program provides to other programs. Used to enable cross-program communication.

Bluemix
A cloud based development platform for developers to create, host and maintain applications with the possibility to utilize a selection of IBM, Cloud Foundry and third party services.

EIRA
Entity Indexed Ranking Application, the name of the application developed during the project.

Entity Extraction
The process of finding entities in a text for instance authors within an article.

JSON
JavaScript Object Notation, a commonly used data structure for network communication.

NCI
National Cancer Institute is the USA's government's main institution for cancer research and training.

NIH
National Institute of Health, a research center.

OHSL
Open Health Systems Laboratory, an organization aiming to unite biomedical informatics.

RFA
Request For Applications, a publication for research projects funded by grants that researchers can apply for.

REST
An architecture approach to a hypermedia system where a set of components, connectors and data elements have specific roles. The focus in a system based on REST lies within the interactions of the data elements rather than the implementation.

RESTful
Conforming to the constraints of REST (Representational State Transfer). This is often used to say that the architecture allows communication over HTTP via verbs (POST, GET, PUT, DELETE, etc.).

SJR
SCImago Journal Rank, ranking of scientific journals based on a measurement of their scientific influence.

SIR
SCImago Institutions Ranking, ranking of research institutions.

Service
Code modules, both Bluemix's and our internally developed modules are considered to be services. E.g. Concept Insights and the Ranker.

Watson
1) IBMs super computer that uses cognitive computing to retrieve results.
2) A selection of cognitive computing services (see **services**) offered on Bluemix (see **Bluemix**)

Contents

Appendix A - Component Diagram

1 Introduction

Traditionally organizations that offer grants to researchers post a Request for Application (RFA). An RFA specifies what type of programs are eligible for funding and are commonly posted by government agencies or non-profit organizations [1]. This system works in a way where researchers and other organizations are allowed to present bids on how the funding could be used. Furthermore, an RFA posted on the National Institute of Health typically presents the researchers or organizations with a purpose and a background to the problem, explaining specifics on why research needs to be conducted and why it is important. This research is further specified in the form of listing Provocative Questions (PQs) that works as a formulation of questions for the RFA [2].

This is inherently an ineffective process for handling funding. Researchers or organizations that might be interested in an RFA may simply not be aware of the RFA and thus not submit an application. Furthermore, this is a time consuming process both for the applicant and the personnel reviewing the applications.

The National Cancer Institute (NCI) in collaboration with the Open Health Institute Lab (OHSL), have expressed interest in a cognitive assistant to more effectively and efficiently match researchers and RFAs. Their long-term objective is the development of such an assistant to find a selection of researchers that are best suited for any given RFA from the National Institute of Health (NIH). As a first step towards creating such an assistant, NCI and OHSL has, in collaboration with IBM (International Business Machines), requested an application that will be the first version, a proof of concept, of such an assistant.

1.1 Purpose of the project

The purpose of this project is to create a proof of concept of an application capable of identifying and ranking researchers by their relevance to a given research topic within the medical field. The goal is to show the viability of such an application.

1.2 Scope of the project

The application, EIRA, is limited to finding and ranking researchers based on the information available in scientific databases and search tools, which means that a large number of factors which would be relevant to a research project are ignored. Researchers are only ranked by information related to their articles and their affiliation whereas attributes such as location and availability are ignored because of the complexity of finding and evaluating such information. Due to the vast number of scientific articles published we limit the search to

researchers that have published an article in the last six years. This has the side effect of reducing the possibility of suggesting a deceased researcher. We will primarily focus on developing a system for finding researchers and extracting properties that could be relevant. As we have no understanding of what makes a researcher suitable, the focus of the ranking is to enable a user to set it as they deem fit.

1.3 Early changes to the project

Initially the project's plan was to cooperate with IBM and get access to their super computer Watson to utilize its cognitive computing technology. Ultimately we did not get access to the super computer. We tried to apply the Bluemix cognitive computing services to EIRA, but most of these services require stored data. However, we are limited to using live data because of the Scopus database which does not permit storing their data. The final version of EIRA has some cognitive computing functionality in the Analyzer module, which is explained in section 5.1.

2 Problem Description

The objective of the application is to locate skilled and relevant researchers. The intent is to accomplish this by accessing databases for academic articles to locate relevant articles and then identifying the researchers behind these articles. After finding the researchers, more queries should be performed to find out more about their previous work in order to be able to rank them by their experience and suitability. As such, the project can be split into three core sub-problems: **Request for Application Analysis**, **Gathering and Combining Data** and **Ranking the Result**.

2.1 Request for application analysis

A module could be developed to analyze an actual RFA and extract keywords from the body of text and then use these as search terms. The main focus of the project is the identification of researchers, and since it is easier to let the user provide keywords than extract them from a body of text, the analysis of an RFA will not be included in the main focus. If such a module were created and worked flawlessly however, one more step of human interaction could be removed: the need for a human to determine just the right search terms. Cognitive computing would have to be used for locating the search terms in natural language.

2.2 Gathering and combining data

The application should retrieve data from one or multiple sources by feeding them one or more search terms. It should then identify entities within the data and combine them into a set. The data could be found using indexed research databases, data mining and web crawling.

2.3 Ranking the result

Once a number of entities have been found, ranking them would be helpful to find the best possible candidates for the research to be conducted. Depending on what data is available for any given entity, multiple approaches may be needed depending on the availability of information from the different sources. The weight of different parameters must be decided and a default value needs to be assigned where none is available.

3 Background

This chapter covers earlier work related to the project and aims to explain ranking criteria which may be unfamiliar to the reader. The purpose of the chapter is thus to further the understanding when reading the consecutive sections of this report.

3.1 Defining cognitive computing

According to the Oxford dictionary, the definition of cognition is "the mental action or process of acquiring knowledge and understanding through thought, experience, and the senses" [3]. Cognitive computing has however been defined differently by researchers over time in different contexts and there is no commonly accepted definition for it [4]. In 2014 a group of experts started a project to define the concept and to explain how cognitive computing is different from traditional computing [5]. According to their definition, traditional computing works with well defined data models from which they can compute exact answers. Cognitive computing addresses different kinds of problems that are characterized by ambiguity and uncertainty. For these problems there may not exist a correct answer, so instead it tries to provide the best answer. Cognitive computing is probabilistic instead of deterministic. It makes context computable, it identifies and extracts context, such as time, location, tasks, history or profiles and it can go through a massive amount of information to find patterns to respond to whatever is needed at the moment [5]. Their definition also says that in order to reach this new level of computing the cognitive system must have five properties:

- **Adaptive** - It must use data in real time in order to handle that information changes and goals and requirements evolve.
- **Interactive** - It must enable the users to define their needs well.
- **Iterative** - It must ask questions and find more information if a problem is ambiguous or incomplete.
- **Stateful** - It must store information about earlier interactions in the process.
- **Contextual** - It must be able to extract, identify and understand context.

Tools and techniques that are related and can be part of a cognitive computing system are [6]:

- **Big data and analytics**
- **Machine learning**
- **Internet of things**

- **Natural language processing**
- **Causal induction**
- **Probabilistic reasoning**
- **Data visualization**

According to IBM, unstructured data such as journal articles, images, sensor data and messages makes up about 80 percent of the world's available data. Traditional computing cannot interpret this but cognitive computing can make sense of it [7].

3.1.1 Watson, reigning *Jeopardy!* champion and supercomputer

Watson is a supercomputer created by IBM in 2011 to showcase cognitive computing technologies. IBM initiated this project to compete in the TV game show *Jeopardy!* against the top ranking *Jeopardy!* players of all time. Cognitive computing was needed in this project for several reasons. The computer needed to understand questions that often were complex and hard to understand even for humans due to intentional ambiguity and word play. In order to understand these questions, natural language processing was used. Once Watson understood the question it searched through millions of stored documents to collect information and find patterns and evidence in order to be able to answer the question. It did not find a certain answer but instead worked with probabilities of answers. If the answer with the highest probability had high enough probability Watson would then press the button to answer the question. It had to do all this in a matter of seconds to be able to answer the question fast enough to be competitive. Watson excelled at this task and beat the two best human *Jeopardy!* players of all time [7].

3.2 Scientific ranking criteria

Scientific criteria can be used in order to determine how qualified a researcher is for a specific research project. This section describes the different scientific ranking criteria used for the project.

3.2.1 Citations

Citations are possibly the most apparent way of ranking researchers; they show how much their peers reference their work. There are different measurements related to the citations of an article or researcher such as the total citation count, the average over all articles, the average over time and H-index just to name a few. For this project, the total number of citations both for researchers and articles were used, as well as the H-index.

The Hirsch index or in short H-index was introduced for the purpose of comparing how much impact and relevance a particular researcher had in the past. It is a measurement of both the productivity and the apparent impact of the researcher. The H-index of a researcher can be calculated by taking h of the researchers N papers and each of those h papers have at least h citations and the rest N - h papers have no more than h citations each. This can be seen in figure 1.

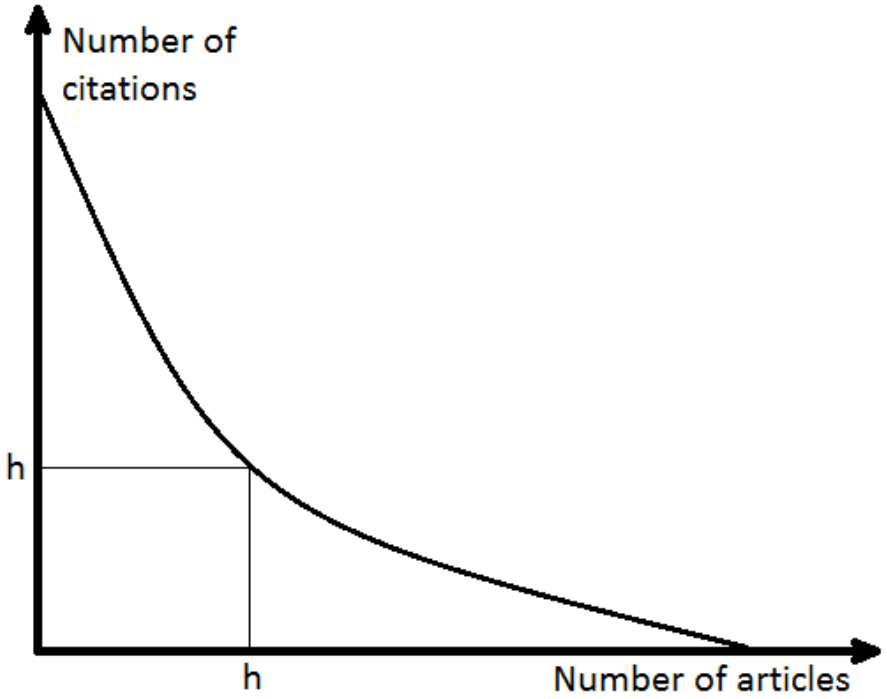

Figure 1: *Graph showing a researchers papers in decreasing order to their number of citations. The area under the curve is the total number of citations. The H-index for the researchers is the intersection on the the curve that will have the same x and y value.*

3.2.2 SCImago Journal Rank (SJR)

The SCImago Journal Rank (SJR) is a tool used to calculate the impact of a journal based on the citations the papers published in the journal have received. The tool uses the data from the abstract database Scopus which according to journalist Declan Butler covers more than 15,000 journals [8]. SJR uses an algorithm similar to Google's PageRank algorithm to rank pages; instead of

listing all citations equally the tool gives higher weight to the citations received from journals with a higher SJR rank. It also takes the relative subject fields into consideration [8]. If, for instance, a paper written in the medical field gets a cited by a paper written in any other field it is valued lower than citations from papers written in the medical field.

The SJR can be accessed through the SCImago Journal & Country Rank platform. The platform lets the user search for journal rankings based on several parameters, such as subject area, categories, regions etc. A search returns a list of journals with the following information for each journal [9]:

- **SJR**
- **H-index**
- **Total number of articles published**
- **Citation count**

3.2.3 SCImago Institution Ranking (SIR)

SIR is a resource to rank universities and research focused institutions over the entire world [10]. They have three ranking sections: research, innovation and web visibility. Only the innovation and web visibility sections are open to the public.

The innovation section of SIR ranks the institutions by score calculated from publication cited in patents based on PATSTAT, the EPO Worldwide Patent Statistical Database [10].

The web visibility section has two indicators which it ranks the institutions on: website size and domain's inbound links. According to SCImago the website size indication is based on the "number of pages associated with the institution's URL according to Google", and the domain's inbound links are based on the "number of incoming links to an institution's domain according to arefs" [10].

4 Method

Making sound decisions is imperative to any project. This chapter begins by explaining the design decisions made during development before moving on to explain why the selected data sources and ranking criteria were used. Finally, section 4.4 covers some tools and instruments whose use was ultimately rejected for various reasons. Skipping section 4.4 will have no effect on the understanding of the rest of the report and should be seen as optional reading for the interested reader.

4.1 Application implementation and design choices

The application was developed using a micro service [11] approach where each of the modules had a specific task and worked independently from the others. This was done to facilitate parallel development and to limit the impact a change in one part of the application would have on the other parts. Four modules were created:

- **Viewer** - A graphical user interface
- **Analyzer** - Extracts concepts from RFAs
- **Finder** - Searches and compiles data
- **Ranker** - Ranks the researchers

Node.js was used as the runtime environment for all of them, which meant that they had to be written in JavaScript (js). The Node.js environment was chosen since it is well suited for server deployment and asynchronous HTTP requests. Furthermore, the Node Package Manager (NPM) greatly facilitates development thanks to its large code library. It also seemed to have excellent integration with the Bluemix services.

4.1.1 IBM Bluemix platform

All the modules of the application were hosted on the cloud platform IBM Bluemix to easier access the Concept Insights[12] Bluemix service. Bluemix is a cloud development platform based on Cloud Foundry that enables its users to create, deploy and manage cloud applications. It provides a growing number of frameworks and services, some of which are provided by IBM or Cloud Foundry and others by third parties [13]. As of May 2016 there are over 100 services provided within various fields like storage, network and security and Internet of Things [14].

4.1.2 Concept Insights keyword extraction

Concept Insights is a Bluemix based service which, amongst other things, can be used to find keywords in a text. In order to facilitate searching for the user, this service was used to analyze RFAs in order to directly extract keywords from them instead of prompting the user for keywords.

4.1.3 NPM Node packages used

NPM features a lot of different packages for different purposes. To speed up development, multiple packages were used throughout development. Table 1 lists each package as well as the version used and a description of each package.

Table 1: *NPM packages used*

NPM package	Versions	Description
express	4.12.x	High performance routing and base server
cfenv	1.0.x	Parses Cloud Foundry-provided environment variables
cors	2.7.1	Connect/Express middleware for cross-origin resource sharing
body-parser	1.12.4	Node.js body parsing middleware
bluebird	2.9.34	Used to create and manage Promises
compression	1.5.2	Node.js compression middleware
config	1.14.0	Used to manage different configuration settings without changing the code
morgan	1.5.3	Node.js middleware for HTTP request logging
request-promise	0.4.2	Promise-using HTTP request client
socket.io	1.3.4	Node.js realtime framework server
watson-developer-cloud	0.9.3	Provides a library to access the IBM Watson Services and AlchemyAPI easier

4.1.4 AngularJS frontend framework

To create the interface of the application the front end framework AngularJS [15] was used in order to make development easier. AngularJS makes interface development faster by adding more attributes to the HTML syntax and makes it easy to implement the logic of the controller side by side with the view [15]. It provides a fast starting point for a web application and allows the user to be expressive in the structure of the application.

4.1.5 Cloudant database

Cloudant is a Database as a Service (DBaaS) hosted by IBM and provides a service for storing multiple data structures such as JSON documents. The database communicates via a RESTful API over HTTP/S. To insert, update, retrieve and remove objects from Cloudant you send POST, PUT, GET and DELETE HTTP requests to the API, respectively [16].

The use of Cloudant eliminates the need of a local database. Instead the application can simply use POST, PUT and GET requests to access and store any data. Another nice feature of using Cloudant is that JSON objects easily are converted into Node.js objects and vice versa.

4.1.6 Using Bluebird to replace callback with Promises

A known problem with the default usage of Node.js is the way it uses callback functions for handling asynchronous calls.

```
1
2     // CALLBACK EXAMPLE
3
4     function nodeFunction(x, cb) {
5         doSomething(x, function(err, res) {
6             if (err) {
7                 /* HANDLE ERROR */
8             }
9
10            doSomethingElse(res, function(err, res) {
11                if (err) {
12                    /* HANDLE NESTED ERROR */
13                }
14
15                return cb(res);
16            });
17        });
18    }
19
20    function doSomething(someInt, cb) {
21        /* DO SOMETHING */
22        return cb(null, someInt);
23    }
24
25    function doSomethingElse(someInt, cb) {
26        /* DO SOMETHING ELSE */
27        return cb(null, someInt);
28    }
```

Listing 1: *JavaScript example of callbacks*

Notice the way the nested functions grow horizontally in listing 1 as more and

more callbacks are nested. Furthermore, notice how the error handling for each nested function grows in a similar way.

One way of changing this behavior is to use Promises instead of callbacks to handle asynchronous calls [17]. Promises are an alternative to callbacks that increases the readability of the code and is less prone to crashing. If an error is thrown using callbacks, without catching it, the whole process crashes. When using Promises however, only that chain of Promises will die. This results in a more forgiving way of writing a program and a more robust system.

The module Bluebird [18] was the promise library of choice for the application. Despite there being other, possibly more well-known third-party promise libraries available for JavaScript, Bluebird was chosen because it is a light weight library focused on performance [19] and ease of use. One example of how the Bluebird library facilitates the coding experience is that it supports long stack traces. These kinds of error traces make debugging more manageable.

The code in listing 1 can be rewritten using Promises as shown in listing 2.

```
1
2     // PROMISE EXAMPLE
3
4     function nodeFunction(x) {
5         return doSomething(x)
6             .catch() /* HANDLE ERROR */
7             .then(doSomethingElse)
8             .catch(); /* HANDLE NESTED ERROR */
9     }
10
11    function doSomething(someInt) {
12        return new Promise(function(resolve, reject) {
13            /* DO SOMETHING */
14            return resolve(someInt);
15        });
16    }
17
18    function doSomethingElse(someInt) {
19        return new Promise(function(resolve, reject) {
20            /* DO SOMETHING ELSE */
21            return resolve(someInt);
22        });
23    }
```

Listing 2: *JavaScript example of Promises*

Notice how the code now grows in a vertical matter, resulting in code that is easier to follow and debug. Every promise produces a ".then()" and a ".catch()" function for assigning the next step upon success or errors resulting from the promise. This makes it easier to see the error handling and what sequential steps are performed in any function sequence.

4.2 Data sources

This section talks about the two data sources that were used in the application: Scopus and Microsoft Academic. Both provide well documented, RESTful APIs where the response can be requested to be in the JSON format. They also both provide unique identification for the authors, which makes it possible to differentiate two authors with the same name.

4.2.1 Scopus, an abstract and indexing database

Scopus is a database that provides abstracts and metadata for publications across all research fields. It focuses on documents where the author is the researcher in charge of presenting the findings from serial publications. A serial publication is a publication that has been assigned an International Standard Serial Number (ISSN) [20]. Elsevier, the supplier of the Scopus APIs, claims that "Scopus has twice as many titles and over 50 percent more publishers listed than any other abstract and indexing database, with interdisciplinary content that covers the research spectrum" [20].

The Scopus API provides a comprehensive search service for retrieval of abstracts, authors and affiliations. With an API key, a so-called subscribers response could be retrieved through Chalmers University of Technology's network. With a subscriber's response more data such as h-index, author identification number and abstracts can be accessed and larger responses can be requested.

The Scopus database was used because it provides a large data set of abstracts for peer reviewed scientific research papers. It also provides a lot of metadata for each article.

4.2.2 Microsoft Academic search engine

Microsoft Academic is a search engine for academic papers based on the Bing search engine. Just like Bing, it uses a web crawler to find data that users then can perform searches on [21]. According to the front page of the service it provides results from over 80 million publications [22]. It lets the user decide what data is desired (names, affiliations, journals etc.) [23] and also how many results should be returned.

Microsoft Academics was used since it provides metadata for a different data set of scientific articles than Scopus and could easily be merged with the results from Scopus.

4.3 Ranking criteria

Four different ranking criteria were used to compare the authors:

- **Citation count** of the authors' articles
- **SJR** value for the journal the author's articles were published in
- **SIR** value for the institution from which the article was published
- **H-index** of the author

The citation count, SJR and SIR are all article specific whereas the h-index is specific to the author.

4.3.1 Citation count

The reason for using the citation count of articles is the availability of such information, making it one of the easiest ways to rank articles. All databases at some time used during development featured citation counts for their articles. Although there may be some hidden gems without a single citation, the citation count should generally be able to provide an, albeit crude, measurement of how good an article is.

4.3.2 SCImago Journal Rank

Another way of ranking articles is by the journal in which they were published. There are multiple journal rankings, but for this project SJR was used since it formatted values in a way that was easy to use with the ranker module. It also seems to be a great measurement of the impact of the journal, see section 3.2. The 2014 SJR ranking was extracted for 6450 journals through the SCImago Journal & Country Rank platform with medicine chosen as the subject area. Using this list, finding the SJR score of a journal was a simple lookup. It required however that the database featuring the article had information about where the article had been published.

4.3.3 SCImago Institution Ranking

Another ranking criterion the application used, was the institution at which the article was written. The idea behind this was that the more influential the institution, the more influential the article should be. The procedure for ranking based on institutions was similar to that of ranking based on journals, but instead of SJR SIR was used. The most promising ranking, the SIR research ranking section, is unfortunately not open to the public, so their web visibility section was used instead. The final list used for ranking the researchers by affiliation contained 741 institutions in the health sector. The SIR value could be

used from all data sources where the article contained the name of the author's affiliation.

4.3.4 H-index

The h-index was used as a ranking criterion to account for the author's older papers and not only the recently published articles returned by the search. The h-index for the authors can be retrieved from the Scopus API. This does however require that the author exists in the Scopus database and the Scopus-specific author identification number. This unfortunately made h-index difficult to use with other data sources than Scopus.

4.4 Software graveyard

In this section the tools and instruments that were researched, implemented and later discarded during the project are explained

4.4.1 Retrieve and Rank

Retrieve and Rank is a Bluemix based service that uses machine learning algorithms to rank content provided by the user. It works with static, stored data such as a collection of documents. Once the documents are stored the user can start to train the machine learning ranking model with a set of questions. After training the algorithm, it can then suggest relevant documents as answers to given questions [24]. Early in the project there was a plan to base the application on this service. It seemed to fit well with the objective of retrieving and ranking authors. This did not work as the service only works on static data. Due to physical and legal limits, no data could be stored by the application so Retrieve and Rank would not work.

4.4.2 Trade-off Analytics, a tool for showing data

The Bluemix based service Trade-off Analytics helps with decision making when confronted with multiple alternatives. It uses so-called pareto-optimization to filter out alternatives which are completely inferior to other options and then displays the tradeoff between the remaining alternatives [25] [26]. Trade-off Analytics was cut in favor of developing a ranking system.

4.4.3 PubMed, the medical database

The PubMed database grants access to Medline, which is the "U.S National Library of Medicine (NLM) premier bibliographic database that contains more

than 22 million references to journal articles in life sciences with a concentration on biomedicine" according to Medline [27]. PubMed is one among many Entrez databases provided by the NLM National Center for Biotechnology (NCBI). However, it provides no means for guaranteeing the uniqueness of authors and therefore could not be used.

4.4.4 Web of Science

Web of Science in a citation index which, according to service provider Thomson Reuters, holds "over 90 million records covering 5,300 social science publications in 55 disciplines" and is the world's largest citation index [28]. Similar to PubMed, Web of Science was not used as it also is unable to differentiate authors with the same name.

4.4.5 Node-Cache memory storage

Node-Cache is a simple in-memory Node.js caching module [29]. It provides an easy to use cache that enables you to store values for a set period of time after which the values are removed from the cache and you would have to store them again. This module could have been used for storing big objects fetched from Cloudant for a short period of time. Each request to Cloudant takes a full second, so it makes sense to store the object in memory for a short period of time and prioritise reading from the memory over Cloudant. Node-Cache had a small delay compared to indexing normal Node.js objects however. This minuscule delay added up to become enormous over the execution time of the program which is why normal objects were used as caches instead. To bypass the need for a "time to live" variable for each object, the remote data was fetched once before the start of the indexing process.

4.4.6 Digital Object Identifier

DOI, or Digital Object Identifier, is an unique identifier for documents that have been registered through the Digital Object Identifier System [30]. It could potentially have been used as an indication of how much effort had been put into an article, but since there appears to be no evidence of this it was never used for ranking.

5 Development

Four modules, each with a specific task, were developed so that they could easily be exchanged for other modules providing the same functionality.

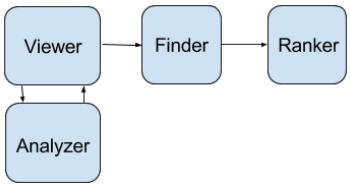

Figure 2: *Overview of the EIRA modules*

Every module was developed to have no dependency on any other module. This modular approach provided flexibility during development since we could easily disconnect any module or build another and then connect it to any existing module. This chapter covers the development process of each module in the order they are accessed by the application: Viewer, Analyzer, Finder, Ranker.

5.1 Development of the Viewer

The viewer is the front end of the application. The first feature developed for the Viewer was to enable it to take a search string as input from the user using a simple HTML form. The search string was then sent as an HTTP request to the Finder to start the search process. The ranked researchers, returned by the Finder, could then be looped through using the controller of the AngularJS model. Each researcher was set up to be displayed in a separate "card" with its relevant information. Each card was made to be clickable, which then brings up a modal window displaying more details. To ease the verifying of the researcher a button was added to perform a Google search on the researchers name and affiliation.

The second feature to be implemented was to enable sending a text file to the Analyzer. This was done by adding a switch which when clicked changed the HTML form displayed to the user, now taking a text file as input. This text file is then sent as HTTP request to the Analyzer. The result returned from the Analyzer is then used to choose a search query to pass the Finder.

5.2 Development of the Analyzer

The Analyzer was added to the project to provide an alternative way of searching within EIRA. The Analyzer uses an RFA sent through a HTTP Post request

from the viewer. The module then extracts the text of the RFA from the request. After the text has been extracted it is parsed to remove special characters. This is done so that the Concept Insights service can iterate properly. After the post processing has been performed, the content is then forwarded to the Concept Insights service for analysis. The Concept Insights service annotates the text using the Watson AI and ranks the annotations based on relevance. The list of annotations received from the service is then iterated over to remove duplicates. If duplicates are found, it keeps the one with the highest score and removes the others. Once the duplicates have been removed it iterates over the list again to filter out entries containing unwanted words. This is done to prevent institutions and other irrelevant entries to placed on top of the list. When this process is complete, the top result is returned to the viewer.

5.3 Development of the Finder

Access to multiple data sources was set up in order to maximize the chances of finding a relevant set of researchers. Unfortunately, each one used a different format for its API and also provided very different sets and amounts of information.

5.3.1 Data extraction from the Scopus database

The first step of the search in Scopus was to perform searches for abstracts and their metadata using key words. The next task was to go through the useful part of the data and use it to retrieve more necessary data. In the end, the searches produced a list of articles containing information such as titles, the authors' identification numbers, the affiliations and the citation counts. In listing 3 the structure of the returned data from a Scopus search is shown.

```
// Example of a search result returned by Scopus

{
    "search-result": {
        "opensearch:totalResults": "1760",
        ...
    }
    "entry": [
        {
            "affiliation": [
                "0": {
                    "affiliation-city": "Caracas",
                    "affiliation-country": "Venezuela",
                    "affilname": "Universidad Central de
                        Venezuela",
                    "afid": "60001197",
                    ...
                },
                ...
            ],
            "author": [
                "0": {
                    "authid": "57189071238",
                    "authname": "Rojas D.",
                    ...
                },
                ...
            ],
            "citedby-count": "0",
            "dc:creator": "Ruiz L.",
            "dc:title": "Non-cytotoxic copper overload boosts
                mitochondrial energy metabolism to modulate
                cell proliferation and differentiation in the
                human erythroleukemic cell line K562",
            "prism:doi": "10.1016/j.mito.2016.04.005",
            "prism:publicationName": "Mitochondrion",
            ...
        }
    ]
}
```

Listing 3: *Example of a search result returned by Scopus*

Since authors and not abstracts were to be ranked the next step was to map each abstract to its author's unique identification number. Then the full information about each author was retrieved through the Scopus author retrieval API. The information retrieved about the author included their full name, h-index and affiliation. In listing 4 the structure of the returned data from a Scopus author retrieval is shown.

```
// Example of a author retrieval result returned by Scopus

{
    "author-retrieval-response": [
        {
            "h-index":"2",
            "coauthor-count": "7",
            "coredata": {
                "dc:identifier": "AUTHOR_ID:56218921200",
                "document-count": "4",
                "cited-by-count": "18"
            },
            "affiliation-current": {
                "@id": "103044687",
                "affiliation-name": "Department of Biology",
                "affiliation-city": "Sherbrooke",
                "affiliation-country": "Canada",
                ...
            },
            "author-profile": {
                "preferred-name": {
                    "initials": "W.J.",
                    ...
                },
                "publication-range": {
                    "@end": "2016",
                    "@start": "2011"
                },
                "journal-history": {
                    "@type": "author",
                    "issn": "20457758",
                    ...
                },
                "affiliation-history": {
                    ...
                }
            },
            ...
        },
        ...
    ]
}
```

Listing 4: *Example of a result returned by Scopus author retrieval*

Lastly the data was organized and structured so that the authors could be ranked by the ranker module.

A problem that quickly emerged was that the API limited the search to a maximum of 100 abstracts per response regardless of how many results the search yielded. Because of this, the application had to make several requests to retrieve all abstracts. This would not be a problem if the requests could be handled in parallel. The Scopus API does not support this though, instead of handling requests as soon as they are received, it throttles the requests and adds

them to a queue. This caused a massive slowdown in the application, making each query take somewhere in the realm of 40 seconds.

The author retrieval caused a performance issue similar to to the search request. A maximum of 25 authors could be retrieved per request making the application send out many requests in parallel. The requests are then throttled by Scopus and put in a queue making the query slow.

5.3.2 Data extraction using Microsoft Academic

Getting Microsoft Academic to work with the application was easy but it was not as useful as Scopus. Microsoft Academic also features a RESTful API and returns JSON objects, making it easy to integrate with other results. Instead of having to perform multiple queries, only one was necessary and it could be scaled to suit the time it took to query the other data sources. Microsoft Academic did however not provide as much information as Scopus, such as abstracts. It also doesn't always find all information about articles such as the journal in which the article was published. In listing 5 the structure of the returned data from Microsoft Academic is shown.

```
// Example return results from Microsoft Academic

"entities": [
{
  "Ti": "hallmarks of cancer the next generation",
  "CC": 7250,
  "AA": [
    {
      "AuN": "douglas hanahan",
      "AuId": 2120435251,
      "AfN": "isrec"
    },
    {
      "AuN": "robert a weinberg",
      "AuId": 2146654601,
      "AfN": "whitehead institute"
    }
  ],
  "J": {
    "JN": "cell"
  }
},
]
```

Listing 5: *Example of the result returned by Microsoft Academic*

5.3.3 Combining the data

Retrieving articles and authors from each of the individual sources was simple enough due to the well documented API each of them provided. The next step however was to combine the results and locate what authors and articles that in fact were the same entity extracted from different sources. An entity at this stage looked as shown in listing 6

```
1
2     // AN AUTHOR ENTITY
3
4     {
5         'name': 'John Snow',
6         'co-author-count': 123,
7         'cited-by-count': 1453,
8         'document-count': 50,
9         'h-index': 20,
10        'affiliation': {},
11        'articles': [{'title':'History of the Wall'},{'title': 'The
              basics of leadership'}]
12    }
```

Listing 6: *Structure of the combined data*

The entities were combined by first looking at the name of each entity in the data. To avoid small differences any special characters were trimmed; blank spaces etc. If any of the names matched the focus shifted to a second criterion: the articles. In the same manner as how the entity names were compared the titles of each article were matched to the articles of the matching entity. The difference here was that any substrings of the titles were considered a match. If at least one of the articles matched, the entities were merged into one. By doing this for all of the authors, a single list of authors with combined data from several different sources was produced.

5.4 Development of the Ranker

After extracting and combining the data from each of the sources the ranking procedure was brought into play. The Ranker module takes a formatted JSON object as the body of a POST request sent over HTTP. The JSON object contains the entities to rank, what fields to take into account when ranking and how to use each of the denoted fields. There are two types of fields: weight fields and ranking fields. When indexing the object with a weight field, the field value would be used to index a database table with the same name as the field. The ranking fields however does not index a database table, instead it simply uses the value of the field. Each field does however have two ways to use the value; either it uses the value without changing it (expecting it to be a numeric value),

or it can read a property on the input object which states that the existence of the field is worth a certain weight. Listing 7 shows an example body of the request.

```
1
2     // AN EXAMPLE BODY FOR THE RANKER
3
4     {
5         'Entities': [
6             {
7                 'name': 'John Snow',
8                 'citedBy': 204,
9                 'journal': 'The Wall'
10            }
11        ],
12        'rankingFields': [{
13            'path': ['citedBy']
14        }],
15        'weightFields': [{
16            'path': ['journal'],
17            'weight': 1
18        }]
19
20    }
```

Listing 7: *Example of input to be sent as the body of the request to the Ranker module.*

Using the above example, the ranker take the field `citedBy` and add 204 to the score of the author. The ranker would then look at the `journal` field and see that it's inputted as a weight field. This means that the ranker would index a table in the database called `journal` using the value of the field; `The Wall`. Further, the weight field has another field called `weight`. This states that the existence of the given field on an entity adds the value of the field `weight` to the total score of the author; in this case the value of one.

The ranker begins by performing a check for each weight/ranking field on each entity object asynchronously, and denotes the entity with a new field called score. This field states how high the entity scored with respect to the denoted fields. As a final step the ranker sorts the entities in descending order by the newly assigned score. The entities are then put into the response to the HTTP request.

6 Result

In this section we will go through the results from the development process. We will cover the usage of the application we developed, EIRA, and then move on to more technical aspects such as execution times and the architecture of the application.

6.1 Application usage and output

EIRA takes one or more search terms as parameters. These terms are relayed to the finder that returns relevant entities. After this the entities are ranked relative to one another. The 20 highest ranking entities are presented in descending order based on their evaluated ranking score. Performing a search on 'Multiple Myeloma' returns the results displayed in Figure 3.

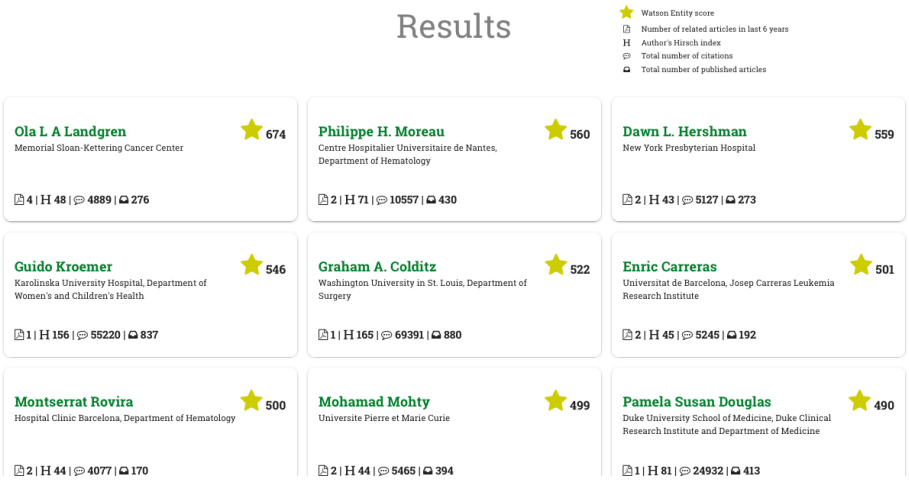

Figure 3: *EIRA Search For Multiple Myeloma*

The top and bottom (that is, the 20th) candidate for a search on 'Multiple Myeloma' can be seen in Figure 4 and Figure 5, respectively.

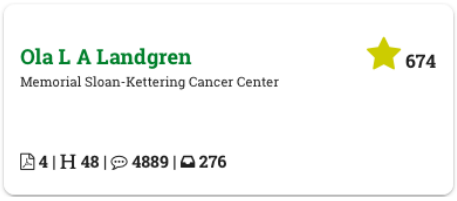

Figure 4: *Top EIRA Result for Multiple Myeloma*

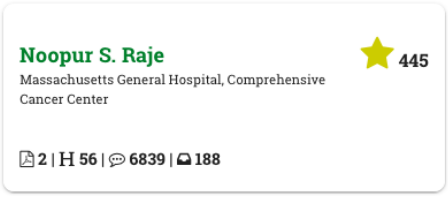

Figure 5: *Bottom EIRA Result for Multiple Myeloma*

The top result is Ola L A Landgren who is the Chief of Myeloma Service at Memorial Sloan Kettering Cancer Center in New York. The bottom result is Noopur S. Raje who currently works at Massachusetts General Hospital, Comprehensive Cancer Center and is board certified in Hematology and Medical Oncology.

The top and bottom candidates for a search on 'Pectus Excavatum', extracted by EIRA, can be seen in Figure 6 and Figure 7 respectively.

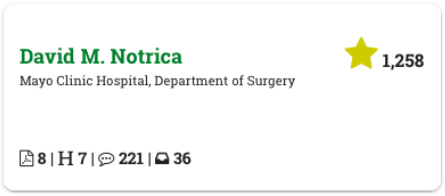

Figure 6: *Top EIRA Result for Pectus Excavatum*

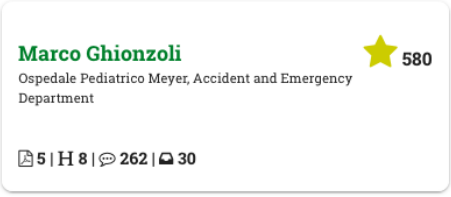

Figure 7: *Bottom EIRA Result for Pectus Excavatum*

The top result, David M. Notrica, is "an associate professor at the University of Arizona College of Medicine - Phoenix, and Assistant Professor of Surgery at Mayo Clinic Medical School" according to Pediatrics Surgeons of Phoenix [31]. The bottom result, Marco Ghionzoli, is a pediatric doctor in Italy [32].

The top and bottom results when uploading an RFA[33] to EIRA are presented in Figure 8 and Figure 9 respectively.

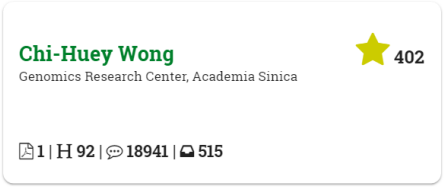

Figure 8: *Top EIRA Result when using an RFA*

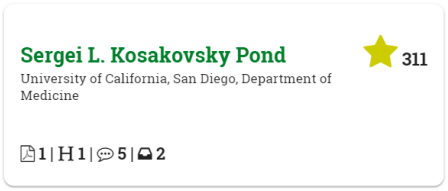

Figure 9: *Bottom EIRA Result when using an RFA*

The top result Chi-Huey Wong is a "Professor of Chemistry at National Taiwan University and the Scripps Research Institute, La Jolla, USA"[34]. Sergei L Kosakovsky Pond is an associate professor at the Department of Medicine at University of California in San Diego.

The the top three annotations that the Concept Insights service extracts from the RFA can be seen in Listing 8.

```
{
  "concepts": [
    {
      "score": 0.9324179,
      "label": "HIV"
    },
    {
      "score": 0.93161255,
      "label": "Prevention of HIV/AIDS"
    },
    {
      "score": 0.92701346,
      "label": "Incidence (epidemiology)"
    }
  ]
}
```

Listing 8: *Annotations retrieved from RFA*

6.2 Execution times for data retrieval

It takes EIRA anywhere between 40 and 100 seconds to produce a result. This is largely due to the limitations of the external APIs that EIRA is communicating with. For example, while retrieving results from a normal search query on the Scopus API we can only fetch 100 results at a time. EIRA performs these requests asynchronously but Scopus throttles and queues all the requests, making the behavior synchronous and the response time longer. This renders any asynchronous behavior related to Scopus API requests useless. Fetching the first search batch from Scopus takes up to 20 seconds alone, for reasons unknown to us as this is on Scopus' part. Table 2 shows the execution times for different search phrases and requests.

Table 2: *Approximate execution times of different modules for various search queries*

Terms	Scopus Search	Scopus Author	Entity Ranker	Total
Abdominal Aortic Aneurysm	39s	36s	4s	82s
Acute Respiratory Distress Syndrome ARDS	33s	35s	4s	76s
Homogentisic Acid Oxidase Deficiency Alkaptonuria	22s	7s	2s	32s
Computerized Axial Tomography	39s	37s	4s	84s
Creutzfeldt-Jakob Disease	46s	44s	5s	98s

As shown in the table, the time that EIRA waits for the Scopus response is as much as 80-95% of the total execution time. This is a big bottleneck of our application, caused by the Scopus API as it limits the amount of certain API calls we can make per minute and per seven days. This causes two limitations on the amount of search results we can use:

1. We have to limit the number of results we fetch from Scopus by a huge amount; some searches produce as much as 100,000 results and we can only use about 1000 of them due to the weekly limits of author retrievals.
2. Scopus handles every request synchronously. Therefore, we have to limit the number of requests we send based on how long time we are willing to wait for the response.

6.3 Application architecture overview

The communication with the services is done via HTTP requests and each of the modules present a RESTful API. The user is prompted to input either an RFA or keywords directly to the application. When the user inputs keywords and initiates a search from the Viewer, a request is sent to the Finder carrying the search terms. If the user inputs an RFA instead a request is sent to the Analyzer and parsed before requesting the Finder. The Finder forwards the search, in parallel, to the different data sources and combines the resulting entities. The entities are then sent to the Ranker in a POST request, which in turn ranks and sorts the entities before returning the resulting list of ranked entities. The result of the Ranker is then forwarded as a result for the initial request from the Viewer. Appendix A contains a component diagram of the application. The following sections describe the component composition of each micro service module.

6.3.1 Description of the Viewer

The Viewer constitutes the user interface of the application, it displays the data produced by the Finder. A component diagram of the Viewer can be seen in figure 2.

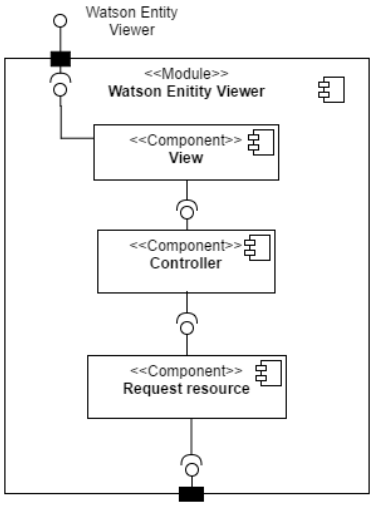

Figure 10: *Watson Entity Viewer*

The components of the viewer are:

- **View** - The visual parts of the user interface such as HTML, CSS etc.

- **Controller** - Handles the logic of the interface and communicates with the Resource for fetching external data.
- **Resource** - Handles the requests to the Analyzer and Finder services.

EIRA can either read an RFA or rely on the user to supply keywords. If an RFA is sent, the Viewer forwards it to the Analyzer for keyword extraction. If the user supplies the keywords instead, it sends these directly to the Finder. Once the Finder returns the list of ranked researchers it shows them to the user.

6.3.2 Description of the Analyzer

Using the Bluemix service Cognitive Insight, the analyzer finds high level concepts in the content of a text file, such as an RFA. Figure 11 shows a component diagram of the Analyzer.

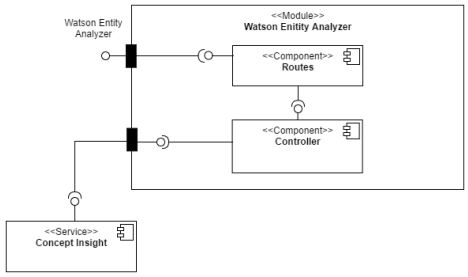

Figure 11: *Watson Entity Analyzer*

The Analyzer is constructed by the following components:

- **Routes** - Supplies the EIRA API with paths to the functions and returns an output that is used in other modules.
- **Controller** - Calls the Watson API Concept Insights and utilizes the methods that it supplies. It then passes outputs from called methods for further use in the next methods. Results are parsed and adapted for other methods if needed.

6.3.3 Description of the Finder

The Finder in the intermediary between the different data sources and the Ranker module. It is the main module and handles most of the logic in the application. A component diagram of the Finder is shown in figure 12.

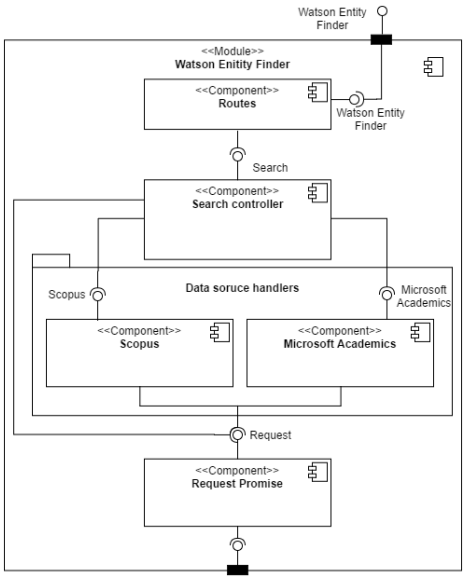

Figure 12: *Watson Entity Finder*

The module consists of five components:

- **Routes** - Provides the REST API paths and forwards the search calls to the Search Controller component.
- **Search Controller** - Handles the communication with the Ranker module and the two handler components: Scopus and Microsoft Academic.
- **Scopus Handler** - Handles communication with the Scopus API.
- **Microsoft Academic Handler** - Handles communication with the Microsoft Academic API.
- **Request Promise** - handles the HTTP requests.

Any incoming request passes through the Routes component and gets forwarded to the Search Controller. The Search Controller then relays the request to each of the handlers. The handlers in turn parse the request and query their respective APIs for the query of the request through the Request Promise component. They then restructure the returned data and return it to the Search Controller. Once the Search Controller receives the returned data it merges it and makes a request using the Request Promise component to the Ranker module to rank the data. When the Ranker has returned the result, the Search Controller returns it to Routes.

6.3.4 Description of the Ranker

The Ranker module exists to rank a list of entities given some specified fields. You are able to preemptively store set weights for the values of some fields in a Cloudant database or simply rank a field on its existing numerical value. If a field value for any of the objects is missing the field will be assigned the value zero instead. Figure 13 shows a component diagram of the Ranker.

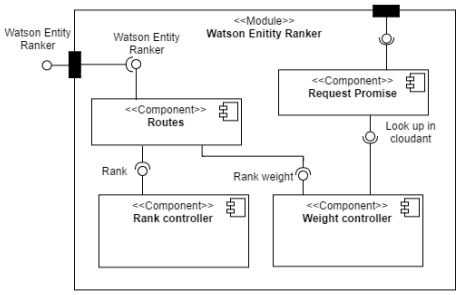

Figure 13: *Watson Entity Ranker*

The module consists of four components:

- **Routes** - Provides the REST API paths and forwards the request calls to the Weight- and Rank controllers.
- **Weight Controller** - Communicates with the Cloudant database service and handles temporary data caches.
- **Rank Controller** - Handles the logic behind ranking the entities.
- **Request Promise** - Handles the HTTP requests.

The first access point of the Ranker module is the Routes component. Routes exposes different RESTful URL paths as an API and forwards any requests to either the Weight Controller component or the Rank Controller component. The Weight Controller communicates with the Cloudant DBaaS while the Rank Controller handles all the logic behind ranking the entities.

To rank the entities a POST request must be performed on the path `/api/rank` containing an object body with three different fields: `entities`, `weightFields` and `rankingFields`. Each of the objects in the entities field will be ranked with regards to the values of the other two fields. Any result produced by any of the controllers are then passed back to the routes component that returns the result to the requesting client.

The Ranker is built with flexibility in mind. We do not want to lock us down into only ranking the entities that we produce in the Finder. As such, anyone who desires to use the Ranker module could do so within the limitations of the input format. We intend to build the Ranker and add it to the Bluemix services

as a community created service. If IBM agrees to the idea of including it, the Ranker could be used by others, too.

7 Discussion

This section covers our thoughts and design decisions made throughout the project and our reasoning behind them. Hardships and troubles that occurred during development are explained and how they were solved. The chapter ends with suggestions for future development.

7.1 Evaluation of results

The results produced by EIRA seems to vary in quality. The top results, Ola L A Landgren, David M Notrica and Chi-Huey Wong all appear to be relevant and competent researchers in their respective fields, judging by their titles. However, the bottom results are much more difficult to evaluate. They are all related to the topics and have some published articles in the field, but apart from that is proved difficult to make any conclusion regarding their viability. All tests were conducted manually by the project group, which lacks experience in evaluating researchers, and the sources containing information about the researchers in general seemed somewhat unreliable. This means that all evaluations executed are crude, at best. EIRA would greatly benefit from some evaluation by experts in the field, and it would be interesting to see how they would balance the weights of the different fields in the ranking.

7.1.1 Evaluation of the Analyzer

The analyzer can be unreliable because of the concepts that EIRA thinks are the best results. Because of cognitive computing, results can sometimes be incorrect when analyzing text. Because of this, the analyzer sometimes cause problems further along the data retrieval process. In some cases the best considered result was not a suitable search term, e.g. names of institutions rather than diseases. Searching our data sources for non-medical related terms causes invalid results. After numerous tests we deem that the analyzer can provide a usable function for EIRA. We trust that the user recognizes an incorrect result once a faulty search occurs.

7.2 Software design decision evaluation

The application was structured as several modules, which resulted in changes to one module not affecting the other modules. This structure made it easier to divide the areas of what needed to be implemented and work independently. The trade off was the number of repositories to version control and the number of steps needed to put the entire code into a production state. However, once

the application was running, parts of it could be taken offline for patching when needed. Overall it was a good architecture choice.

Node.js proved to be well suited for the purpose of the project. It made development easy and manageable thanks to the package manager NPM for Node.js that has a wide variety of existing modules to support the development process.

7.3 Data source selection

The data sources used to rank the researchers worked well overall. Although there were some issues with regard to the amount of information available, it seems it was enough to generally provide good results. The relevance of the result depends on the data retrieved but since the sources of data used are continuously updated the output of the application should remain relevant.

7.3.1 Scopus

Scopus is well suited for the purpose of the project as it provides a system for identifying authors as well as a lot of different data usable for ranking. Its well documented API made development easy. There is only one problem with Scopus, and that is the limitations on the API. Without this limitation, we would be able to handle multiple times more data and rank accordingly, while the execution time of the request would still be far below the current time.

It would be interesting to have the limits of the API removed. If the project was to be further developed and refined in cooperation with IBM, then Scopus might agree to a more suitable solution and give more leisure to the project to enable faster and more complete results.

There could also be ways of circumventing the limits of the API. As explained in section 6.2, there are two limits: one weekly and one throttling the throughput of requests. The throttle appears to track IP-address or some other computer specific attribute as multiple searches conducted from different servers at the same time using the same API key does not slow each other down. The weekly limit, on the other hand, appears to be tied to the API key as replacing a blocked API key with an unused one enables the application to keep searching. Thus circumventing the weekly limit is as simple as getting more API keys to use with the application. The throttle could possibly be avoided by building a distributed system or by using IP aliasing so that the requests come from different addresses.

7.3.2 Microsoft Academic

Microsoft Academic offers more breadth but less depth than Scopus; it can return a lot more articles than Scopus in the same amount of time, but less information is available for each article. Because of this, all researchers returned in the top 20 by EIRA were found in Scopus (but not necessarily in Microsoft Academic) since the researchers found exclusively in Microsoft Academic did not have a H-index, SJR or SIR score. Since these scores were set to 0 by the ranker if no value was found, the authors found exclusively in Microsoft Academic are ranked solely based on citation count. It was easy to connect EIRA to Microsoft Academic but since Microsoft Academic offered less information for each article than Scopus, Scopus still worked better despite its flaws.

7.3.3 Discontinued data sources

During the course of development, other data sources that eventually were removed from the final version were used. The initial targets for data extraction were PubMed, Scopus and Web of Science. The APIs of PubMed and Web of Science does unfortunately not offer any way of differentiating between authors with the same name. This caused the results from PubMed and Web of Science to have multiple articles written by people with the same name, but not always the same person. Since there was no way to tell if they were the same person or not, it would be possible that two authors with the same name could be mistaken as the same person if PubMed and Web of Science were to be included in the application.

7.4 Evaluation of the ranking criteria

The objective of the ranking was to find the most experienced and suitable researcher for a specific research area. There is, however, only a limited amount of data available that authors can be ranked upon. The challenge is to get the best possible ranking based on the available data. The fields that are used for ranking entities are discussed in this section.

7.4.1 Citations

One of the fields extracted from each author was the `H-Index`. The H-index seems to be a good addition to the ranking since it accounts for the productivity and impact over an entire career. All other criteria readily available only account for the articles returned by the search, which in the case of Scopus only is the most recent ones. Other measurements for this have been looked into, such as overall citations and average citation per publication. Using the h-index was

also a good choice considering it is based on Scopus database which is the main data source. This made it easy to add to the application.

Another field extracted from each article was the `citation count`. Using this number as a measurement of the impact of the articles seemed to give the desired ranking. If an article has been cited by a lot of people it would prove that the article reached an audience and was meaningful or impactful enough to cause others to cite it. Because of this, its value was regarded highly in the ranking process. In the end, the number of citations of each article was summarized and added to the score of the author. Furthermore, it seems likely that citation counts are exponential (since a well cited researcher and paper gets more attention and consequently possibly more future citations). This might have given the citation count too big of an impact on the final score, especially for the top researchers.

7.4.2 Journals

The choice to add ranking value based on the journal an article is published in was a good choice since it gives an indication of the quality of the article. Having looked into many methods to rank journals such as Impact Factor, Eigenvector and Source Normalized Impact per Paper, more research needs to be done in order to determine which one provides the best indication. SJR seem to give a good measurement through its complex algorithm. In addition to possibly being the best it also worked well with the rest of the application thanks to it's connection to Scopus. It was also easy to implement thanks to its format that was easy to add to the Ranker.

7.4.3 Institution

The web visibility section of SIR gives an indication of the influence of the institution, however, it may not be the best indication. It was ultimately chosen because of how easy it was to incorporate with the application. Access to the research section of SIR could greatly improve the institution based ranking. This is because it is a measurement of the research quality produced by the institution whereas web visibility is only an indication of the influence of the institution.

7.5 EIRA's role in society

EIRA could prove useful as a recruiting or headhunting tool, enabling personnel currently focusing on these roles to put their attention elsewhere. This could prove a boon to the economy since less time is spent on administrative tasks and can be spent on, for instance, research. However, developing machines to do the work of people could prove disastrous to society in the long run. The

potential mass unemployment replacing workers with machines could bring. It will require a lot of effort to overcome and mitigate.

Another troubling aspect of EIRA is privacy. Information about our lives are available on the internet and the development of tools like EIRA to sort and categorize this information puts privacy at stake. The privacy of the individual is impaired in favor of enabling the people with access to this data to make better decisions. Whether this trade off is worth it or not is up for debate, and needs to be discussed to make people aware of the intrusions in privacy this trend brings.

7.6 Further development

The project has more or less only been implemented to such a degree that there exist a base application of micro services. As such there are several possible continuations that could further build upon the project. This section contains thoughts on how to further develop the application and suggestions for future projects.

7.6.1 Improving the Finder

The Finder is the largest of the modules and exists to asynchronously handle the data extraction from multiple data sources. There are countless ways of improving this module but a few stand out amongst the rest.

When trying to identify data that could help rank researchers you soon come to the realization that the sources of the data are so different that the potential of the application is huge; social media, data mining, research databases etc. People put new data on the Internet every minute of every day, potentially adding weight to different entities.

The true problem lies with finding and matching this data, realizing that one entity is the same as another. If you could tap into that potent data with a ranking module the results will get even more interesting.

We currently limit the field of search to areas related to medicine but that is not a necessary limit; you could in theory rank an entity based on any data that you find in the Finder.

Following the same train of thought, a simple way to increase the value of the results would be to integrate more sources of data; databases of the same type as Scopus, data mining services, etc.

So far all of the data sources in the project have presented an API that enables the input of not only keywords but logical queries. One could for example query Scopus for all documents containing tags with the keyword `mouse` but exclude

those containing the word `cat` or `dog`. This could be done with the following query:

`KEY(mouse AND cat OR dog)` [35]

Microsoft Academic have a similar way of handling logical operands but with a different syntax. The same query in the Microsoft Academic syntax could be written as:

`AND(W='mouse', OR(W='cat', W='dog'))` [36]

The Search Controller in the Finder module does support passing on anything that is sent as a search string in the incoming request. But since Scopus and Microsoft Academic use different syntax for logical operands this becomes invalid. A solution would be to add a layer for interpreting logical queries, then each data source handler could translate those logical operands for their respective APIs. It is simple to do and would enable deeper communication with the already implemented as well as future data sources.

7.6.2 Improving the Analyzer

The Analyzer could be improved by further developing the filtering process of the result and reduce the irrelevant terms that can be sent to the Finder module. Based on previous research there is a function within the Concept Insights API that can retrieve metadata for the annotations. The result can be narrowed down to relevant results by filtering with regard to the metadata. A possible problem with this approach is that the metadata can be incorrectly tagged and filter relevant data. With our current primitive filtering function there is a very low chance to exclude data that can be relevant. The user determines whether the results are good enough. According to prior tests a faulty return should be clear to the user.

7.6.3 Improving the Ranker

The Ranker is the module that best represents the idea behind the project; to rank entities with regards to their relevance given a certain phrase or a number of keywords. The time spent on the project was divided by prioritizing data retrieval functionality, more specifically the Entity Finder. Not until after the Entity Finder was almost done did development of the ranker begin. This has left the ranking in a state where much can be added. Another project, probably at least as big as this one, could build upon what has been created during the course of this project. What makes one article better than another? What impact does it have for its author? What machine learning algorithms could we use to improve the results? These are questions that keep emerging whenever we work with the ranker module and so far none of them have been truly answered.

7.6.4 Error handling

Throughout the application, the error handling has been neglected due to the time constraints of the project. Since all of the modules are written using promises the error handling is easy to add.

In the Ranker, Finder and Analyzer there exists an error handler that could be extended to better suit the project. This handler could then be used in every ".catch()" function where the error was deemed necessary to handle.

The Viewer uses AngularJS which does not use bluebird to handle promise but instead uses their own module called "$q". It is a small promise library that also exposes functionality for handling errors. There are tons of packages and extensions to install for such handling in AngularJS.

7.6.5 Dockerize

Since the application was implemented as a set of micro-services and using Bluemix, the application itself is well suited for Docker [37]. Using Docker we could convert each of the modules into a single docker image. These images would then be used to create docker containers, running on the same machine and communicating internally, removing the delay presented by sending requests over HTTP to each module.

IBM Bluemix has great support for docker [38] and as such it would make even more sense to "dockerize" the application. A good way of doing this would be to use Docker Compose [39] to handle the containers and networks of the docker modules.

7.6.6 The impact of a Digital Object Identifier

DOI, or Digital Object Identifier, is a unique identifier for documents that have been registered through the Digital Object Identifier System [30]. This could have been used as an indication of how much effort had been put into the articles extracted from different data sources. This was not done due to the lack of research stating that this would be the case. The DOI is still extracted if it exists but it is not taken into account during the ranking procedure.

During the construction of the ranking module, we hypothesized that the DOI of an article could be used as an indicator of the validity of an article. The idea was that an article that had a DOI would have been published in such a manner that it required a DOI; for example in a prestigious journal etc. If this was the case then some weight should be added to the score of the author responsible for that article.

It is hard to determine how much of an impact factor we should provide for documents that have such an identifier. While most of the documents on Scopus

have a DOI, it is far from all of them. Further, the data retrieved by Microsoft Academics have proved to more often than not be unable to provide a DOI. All the articles that do not contain a DOI could easily be filtered out, but often an article without a DOI in Microsoft Academic was found to have one in Scopus.

In the end, all documents were used whether they contained a DOI or not since the Microsoft Academics results might strengthen the results provided by Scopus. No additional weight is provided for documents that contain a DOI upon ranking. Further research might prove that such documents have a higher value for the purpose of deciding the validity of an article.

8 Conclusion

8.1 Problem solving

We had several difficulties trying to implement any Watson services from the Bluemix service but to no avail because they required storage to function. We can not use databases because Scopus asked us not to save their documents if we decided to utilize their special access API calls. We ended up with an application that is able to retrieve authors of articles from any area of expertise in any academic field found in publications. This could arguably be more valuable than local data searches. Although, there could be some quirks fetching data because we depend on other services to be available. But even if one database is online the application would still be functional but there would be less accurate results. We are ultimately happy with the end product that we managed to develop even though we had to overcome hurdles during planning- and development phases.

8.2 Scratching the surface

With the entire effort of the project we only managed to connect data from two different sources and rank the authors given a subset of available results. But the application still produces a list of ranked authors that seemingly have a big relevance to the given query. We noticed that the data we base our rankings on is only the tip of the iceberg and we have barley scratched the surface. More data sources, integrate AI solutions to the ranker module, calibrate the weights related to different fields and then possibly add more data mining support; these are all options that could possibly tap further into the potential of the application.

8.3 Project plan fulfillment

In the introduction, we explained how the process of recruiting a team of researchers was an unnecessarily time consuming process. EIRA solves this problem for a team organizer who is eager to start a research group. The organizer now has the ability to find good candidates with one search. The process of finding a researcher is now faster than ever. Seeing how our project is a proof of concept it would be interesting to see what the software industry could produce.

References

[1] "What is request for application (RFA)? definition and meaning." [Online]. Available: http://www.businessdictionary.com/definition/request-for-application-RFA.html

[2] "How to Write a Research Project Grant Application." [Online]. Available: http://www.ninds.nih.gov/funding/write{_}grant{_}doc.htm

[3] "cognition - definition of cognition in English from the Oxford dictionary." [Online]. Available: http://www.oxforddictionaries.com/definition/english/cognition

[4] J. O. Gutierrez-Garcia and E. Lopez-Neri, "Cognitive Computing: A Brief Survey and Open Research Challenges," in *2015 3rd International Conference on Applied Computing and Information Technology/2nd International Conference on Computational Science and Intelligence*. IEEE, jul 2015, pp. 328–333. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7336083

[5] "Cognitive Computing — Synthexis on WordPress.com." [Online]. Available: https://synthexis.com/cognitive-computing/

[6] J. Hurwitz, M. Kaufman, and A. Bowles, *Cognitive Computing and Big Data Analytics*. Wiley, 2015. [Online]. Available: https://books.google.se/books?id=V41xBgAAQBAJ{&}lpg=PP1{&}hl=sv{&}pg=PP1{#}v=onepage{&}q{&}f=false

[7] IBM, "Computing, cognition and the future of knowing." [Online]. Available: http://www.research.ibm.com/software/IBMResearch/multimedia/Computing{_}Cognition{_}WhitePaper.pdf

[8] D. Butler, "Free journal-ranking tool enters citation market." *Nature*, vol. 451, no. 7174, p. 6, jan 2008. [Online]. Available: http://www.nature.com/news/2008/080102/full/451006a.html

[9] "SJR : Scientific Journal Rankings." [Online]. Available: http://www.scimagojr.com/journalrank.php

[10] "SIR Methodology." [Online]. Available: http://www.scimagoir.com/methodology.php

[11] M. Fowler, "Microservices." [Online]. Available: http://martinfowler.com/articles/microservices.html

[12] "Concept Insights — IBM Watson Developer Cloud." [Online]. Available: http://www.ibm.com/smarterplanet/us/en/ibmwatson/developercloud/concept-insights.html

[13] "What is IBM Bluemix?" apr 2015. [Online]. Available: http://www.ibm.com/developerworks/cloud/library/cl-bluemixfoundry/

[14] "Catalog - IBM Bluemix." [Online]. Available: https://console.eu-gb.bluemix.net/catalog/

[15] "AngularJS — Superheroic JavaScript MVW Framework." [Online]. Available: https://angularjs.org/

[16] "Cloudant Features — Cloudant." [Online]. Available: https://cloudant.com/product/cloudant-features/

[17] M. L. T. Cossio, L. F. Giesen, G. Araya, M. L. S. Pérez-Cotapos, R. L. VERGARA, M. Manca, R. A. Tohme, S. D. Holmberg, T. Bressmann, D. R. Lirio, J. S. Román, R. G. Solís, S. Thakur, S. N. Rao, E. L. Modelado, A. D. E. La, C. Durante, U. N. A. Tradición, M. En, E. L. Espejo, D. E. L. A. S. Fuentes, U. A. D. Yucatán, C. M. Lenin, L. F. Cian, M. J. Douglas, L. Plata, and F. Héritier, "Expert JavaScript," *Uma ética para quantos?*, vol. XXXIII, no. 2, pp. 88–92, 2012. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/15003161$\delimiter"026E30F$nhttp://cid.oxfordjournals.org/lookup/doi/10.1093/cid/cir991$\delimiter"026E30F$nhttp://www.scielo.cl/pdf/udecada/v15n26/art06.pdf$\delimiter"026E30F$nhttp://www.scopus.com/inward/record.url?eid=2-s2.0-84861150233{&}partnerID=tZOtx3y1

[18] "Why bluebird? — bluebird." [Online]. Available: http://bluebirdjs.com/docs/why-bluebird.html

[19] "Benchmarks — bluebird." [Online]. Available: http://bluebirdjs.com/docs/benchmarks.html

[20] "Content - Scopus — Elsevier." [Online]. Available: https://www.elsevier.com/solutions/scopus/content

[21] "Microsoft Academic FAQ." [Online]. Available: https://microsoftacademic.uservoice.com/knowledgebase/articles/838965-microsoft-academic-faq

[22] "Microsoft Academic Front Page." [Online]. Available: http://academic.research.microsoft.com/

[23] "Entity Attributes, Academic Knowledge API." [Online]. Available: https://www.microsoft.com/cognitive-services/en-us/academic-knowledge-api/documentation/entityattributes

[24] "Retrieve and Rank service documentation — Watson Developer Cloud." [Online]. Available: https://www.ibm.com/smarterplanet/us/en/ibmwatson/developercloud/doc/retrieve-rank/

[25] "Pareto Optimality." [Online]. Available: http://faculty.washington.edu/mkenn/Pareto/index.htm

[26] "Tradeoff Analytics — IBM Watson Developer Cloud." [Online]. Available: https://www.ibm.com/smarterplanet/us/en/ibmwatson/developercloud/tradeoff-analytics.html{#}how-it-is-used-block

[27] "MEDLINE Fact Sheet." [Online]. Available: https://www.nlm.nih.gov/pubs/factsheets/medline.html

[28] "Discover Web of Science." [Online]. Available: http://wokinfo.com/citationconnection/

[29] "node-cache by tcs-de." [Online]. Available: http://tcs-de.github.io/nodecache/

[30] DOI®, "Digital Object Identifier System FAQs," 2015. [Online]. Available: http://www.doi.org/faq.html

[31] "Profile page of David M. Notrica, MD." [Online]. Available: https://surgery4children.com/surgeons/david-m-notrica-md/

[32] "Marco Ghionzoli Profile Page meyer.it," 2015. [Online]. Available: http://www.meyer.it/index.php/cura-e-assistenza/trova-professionista/169-marco-ghionzoli

[33] "RFA-AI-16-038: Silencing of HIV-1 Proviruses (R61/R33)." [Online]. Available: http://grants.nih.gov/grants/guide/rfa-files/RFA-AI-16-038.html

[34] "Chi-Huey Wong, 10th President of Academia Sinica." [Online]. Available: http://home.sinica.edu.tw/en/about/chwong.html

[35] "Scopus Search Tips." [Online]. Available: http://api.elsevier.com/documentation/search/SCOPUSSearchTips.htm

[36] "Microsoft Academic Query Syntax." [Online]. Available: https://msdn.microsoft.com/en-us/library/mt631426.aspx

[37] "What is Docker?" [Online]. Available: https://www.docker.com/what-docker

[38] "IBM Bluemix - Containers." [Online]. Available: http://www.ibm.com/cloud-computing/bluemix/containers/

[39] "Docker Compose." [Online]. Available: https://docs.docker.com/compose/

Appendix A - Component Diagram

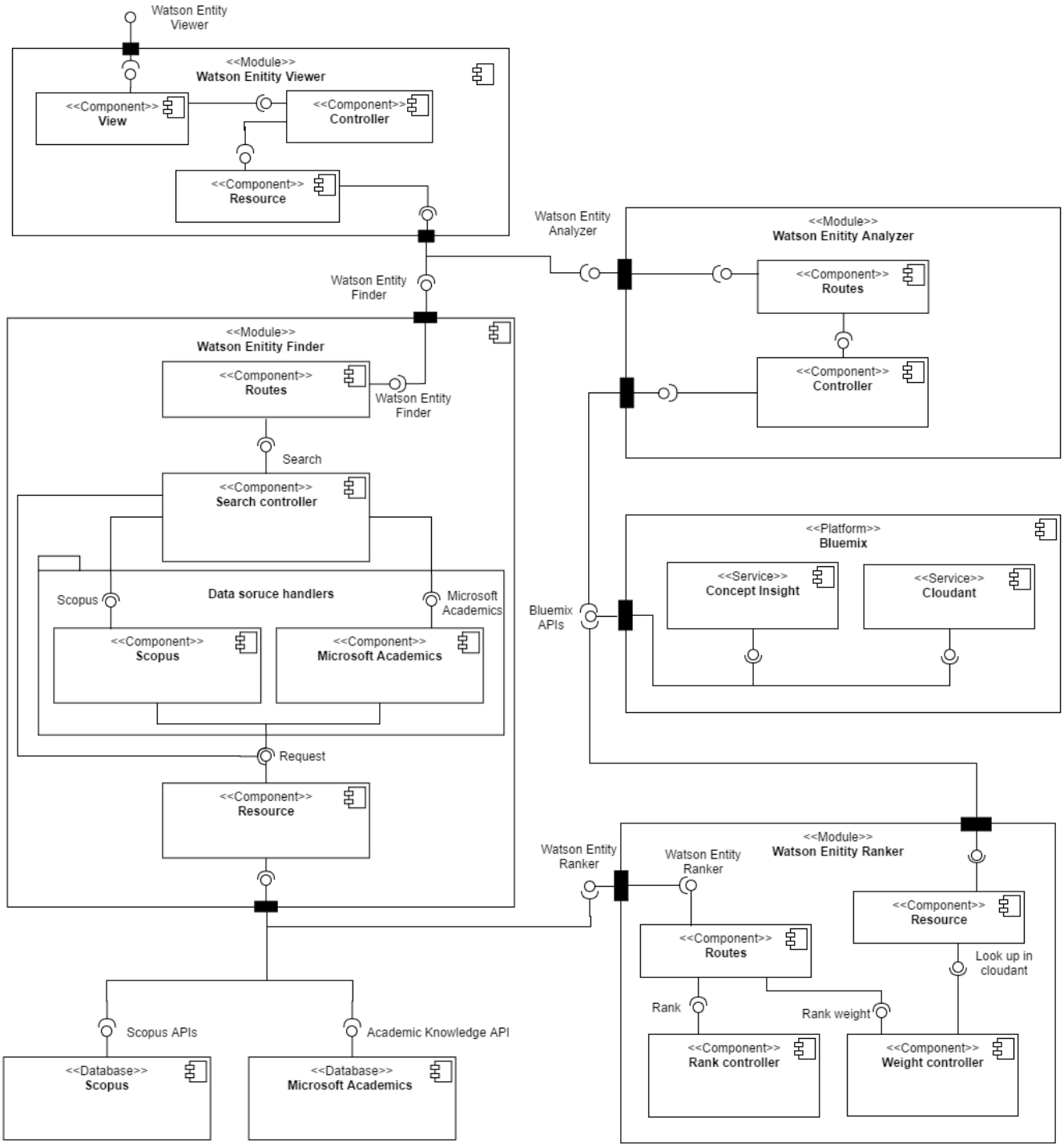

Figure 14: *Component diagram*