BACHELOR'S DEGREE THESIS

# Degree in Data Science and Engineering

**Title: Creation of an open data federated platform**

**Author: Marc Vernet Sancho**

**Advisor: Sergi Nadal**

**Department: ESSI**

**Month and year: June 2022**


UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

**Facultat d'Informàtica de Barcelona**
**Facultat de Matemàtiques i Estadística**
**Escola Tècnica Superior d'Enginyeria de Telecomunicació de**

# Acknowledgements

First, I would like to thank my supervisor Sergi Nadal, for all the help and guidance during the project development. He not only had the first idea for this project's topic, but without his guidance the end result would have been very different.

I would also like to thank my family for the unconditional support, not only during the development of this thesis but also during these last four years, with its good and bad moments. Specially to my father, for criticizing my work relentlessly, which greatly contributed to making this thesis what I want to believe is (almost) its best version possible.

Finally, I would like to end by thanking my job colleagues, who have seen me countless early mornings working on this thesis from the office, and made all those moments more funny and enjoyable.

# Abstract

The goal of this thesis is to design and build a web platform that integrates access to data from several Open Data portals. This platform will be used as a tool to explore and showcase the results of different techniques of data discovery, with the goal of developing and implementing a method to efficiently detect similarities and datasets that can be joined. This thesis is highly practical, and is focused on explaining the technical decisions and implementation details behind the final tool developed.

With the popularization of Open Data platforms the amount of data available on the internet is growing exponentially, and with it the question of how to extract value of such large amounts of data. This project not only looks for a way to unify the data in a single place, but also for ways to be able to explore this data, making special emphasis on ways to detect similarities between datasets.

The project initially revolved around the data provided from the municipal open data portals from Barcelona and L'Hospitalet de Llobregat, but the final solution is meant to be scalable and easily adaptable to other data sources.

The initial idea for this thesis was to be focused mainly on exploring methods for detecting similarity and joinability between datasets based on profiles, but during development, it became obvious that the infrastructure required to unify all the datasets and provide a tool to explore the data was not simple at all. So in the end, a big part of the thesis consists of the discussion of technical decisions made and details of the tool built for the project.

# Resum

L'objectiu d'aquesta tesi és dissenyar i construir una plataforma web que integri l'accés a dades de diferents portals de Dades Obertes. Aquesta plataforma es farà servir com a eina per explorar i mostrar els resultats de diferents tècniques de "Data Discovery", amb l'objectiu de desenvolupar i implementar un mètode per, de forma eficient, detectar similituds i datasets als quals es pot aplicar una operació de "join". Aquesta tesi és altament pràctica, i està centrada en descriure les decisions tècniques i detalls d'implementació al darrere de la versió final de l'eina desenvolupada.

Amb la popularització dels portals de Dades Obertes la quantitat de dades disponibles a internet està creixent de forma exponencial, i amb aquest creixement sorgeix la pregunta de com extreure valor d'aquesta quantitat de dades enorme. Aquest projecte no només busca una manera d'unificar les dades en un sol lloc, sinó també trobar maneres d'explorar aquestes dades, fent especial èmfasi en mètodes per detectar similituds entre datasets.

El projecte inicialment girava al voltant de les dades proporcionades pels portals de Dades Obertes municipals de Barcelona i l'Hospitalet de Llobregat, però la solució final té l'objectiu de ser escalable i fàcilment adaptable a altres fonts de dades.

La idea inicial per aquesta tesi era centrar-se principalment a explorar mètodes per detectar similitud i "joins" entre els datasets basats en perfils, però durant el desenvolupament, va esdevenir obvi que la infraestructura requerida per unificar tots els datasets i proporcionar una eina per explorar-los no era una tasca gens senzilla. Com a conseqüència, finalment una gran part d'aquesta tesi consisteix en la discussió de les decisions tècniques fetes i dels detalls de l'eina construïda per al projecte.

# Resumen

El objetivo de esta tesis es diseñar y construir una plataforma web que integre el acceso a datos de diferentes portales de Datos Abiertos. Esta plataforma se utilizará como herramienta para explorar y mostrar los resultados de distintas técnicas de "Data Discovery", con el objetivo de desarrollar e implementar un método para, de forma eficiente, detectar similitudes y datasets a los cuales se les pueda aplicar una operación de "join". Esta tesis es altamente práctica, y está centrada en describir las decisiones técnicas y detalles de implementación detrás de la versión final de la herramienta desarrollada.

Con la popularización de los portales de Datos Abiertos la cantidad de datos disponibles en internet está creciendo de forma exponencial, i con este crecimiento surge la pregunta de como extraer valor de esta cantidad de datos enorme. Este proyecto no solo busca una forma de unificar los datos en un mismo sitio, sino también encontrar maneras de explorar estos datos, haciendo especial énfasis en métodos para detectar similitudes entre datasets.

El proyecto inicialmente giraba alrededor de los datos proporcionados por los portales de Datos Abiertos municipales de Barcelona y L'Hospitalet de Llobregat, pero la solución final tiene el objetivo de ser escalable y fácilmente adaptable a otras fuentes de datos.

La idea inicial para esta tesis era centrarse principalmente en explorar métodos para detectar similitud y "joins" entre datasets basados en perfiles, pero durante el desarrollo, resultó obvio que la infraestructura requerida para unificar todos los datasets y proporcionar una herramienta para explorarlos no era una tarea nada sencilla. Como consecuencia, finalmente, una gran parte de esta tesis consiste en la discusión de las decisiones técnicas hechas y de los detalles de la herramienta construida para el proyecto.

# Contents


**1 Introduction and Contextualization** **2**
- 1.1 Motivation . . . 2
- 1.2 What is an Open Data platform . . . 4
  - 1.2.1 Data catalog features . . . 4
- 1.3 Resources used . . . 5
  - 1.3.1 Open Data BCN . . . 6
  - 1.3.2 Dades Obertes L'Hospitalet . . . 7

**2 Data Discovery** **9**
- 2.1 Defining the problem . . . 9
- 2.2 Previous work . . . 11
- 2.3 A first solution . . . 12
- 2.4 Similarity based on profiles . . . 16

**3 Implementation Scope** **20**
- 3.1 General objectives . . . 20
- 3.2 Functional requirements . . . 20
- 3.3 Non-functional requirements . . . 21
- 3.4 Design . . . 22
  - 3.4.1 Implementation details . . . 23

**4 Back-end Implementation** **24**
- 4.1 Scrapers . . . 24
  - 4.1.1 Scraping the Barcelona portal . . . 25
  - 4.1.2 Scraping L'hospitalet portal . . . 25
- 4.2 TinyDB and storage . . . 25
- 4.3 API . . . 26
- 4.4 Upload . . . 28

**5 Front-end Implementation** **30**

**6 Unsuccessful implementations** **37**

**7 Further work** **39**

**8 Conclusions** **41**

**9 Annex** **42**
- 9.1 Published code . . . 43
- 9.2 Instructions for deployment . . . 45
  - 9.2.1 Back-end . . . 45
  - 9.2.2 Front-end . . . 46

# 1 Introduction and Contextualization

In recent years, the field of data science has an experienced exponential growth [1]. After its popularization with the famous *"The sexiest job of the 21st century"* article published almost 10 years ago [2], data science has not only grown in popularity, but nowadays its value proposition is better understood by business and academic institutions. Data science cannot be understood without data, and its popularization has grown together with the availability of free data on the internet.

The increasing digitization of our lives has created an even more exponential growth on the available data. With people spending more and more time online, we generate huge amounts of digital data. The governments and public institutions are no exception, its digitization come together with a growth in the ability of processing large amounts of data [3].

During these last decades, the relations between the citizens and the government have tightened. The citizens nowadays are in general more interested in taking an active role in governance, they expect public opinions to be respected and transparency from the government. This transparency from the government has been increasingly important, as is considered vital for the citizen to be able to monitor or be aware of how the government uses its resources.

In the context of these two tendencies, digitization of institutions and demand for transparency, the concept of Open Data emerges. Open Data can be understood as a source of data that has public access and that can be shared or edited with any purpose [4]. This data is in many cases found in governments websites or portals that provide free access to its internal data.

The first benefit of this movement is, as already discussed, transparency. These initiatives have been found to enhance participation of the whole society and accountability for governments, as it forces them to be more responsible with its actions and resource allocations.

But transparency is not the only benefit, Open data can bring benefits in various fields, such as health, food security, education, climate, intelligent transport systems, and smart cities - and is considered *"an essential resource for economic growth, job creation and societal progress"*[5].

## 1.1 Motivation

A search on the internet with some simple key words such as *"dades obertes catalunya"*[1] returns a surprising amount of related sites. Every important government institu-

[1] *"dades oberted catalunya"* can be translated as *"open data catalonia"*. Along the thesis, most of the examples and samples will be in Catalan, as it's the language of the data sources used for this project.

tion has its own websites that provide its internal data. In some cases, different institutions from the same geographical location will have their own independent data portals. Some examples are:

- Generalitat de Catalunya: https://analisi.transparenciacatalunya.cat/
- Diputació de Barcelona: https://dadesobertes.diba.cat/
- Ajuntament de Barcelona: https://opendata-ajuntament.barcelona.cat/

With the growth in popularity of open data platforms a huge amount of portals have appeared, often different institutions are providing data from the same geographical location. This makes the data scattered around different websites, making it difficult to have a global vision on what data exists on a specific topic. These platforms, being developed by different institutions with little relation to each other, do not follow any standard in design, which results in datasets organized in completely different categories.

This is the central issue that this thesis is trying to solve. The goal is to develop a solution that acts as a federation layer for open data portals, that will allow access to all the data from different sources in the same place and that provides a method to detect similar datasets. The platform is intended to unify sources in the context of Catalonia public institutions (municipal data, government, public services, ...), but the final solution should be independent of the kind of data hosted, and could be easily used in other cases.

The platform to be developed in this project has three main goals. First, it needs to have an administration process as automatized as possible, this will allow a fast growth of the number of information hosted without a need for more resources. Second, it has to provide an as easy as possible way to update the platform and integrate data from new sources on an ongoing basis. And finally, it has to provide a method for Data Discovery, centered around identifying similarities and joinability [2] between datasets.

The Data Discovery feature is considered a crucial part of the project. Being able to identify which datasets have similar content, pertain to related categories or even contain the same data is vital in order to have a meaningful integration of all the data. This resulting website developed for this project can be understood as an open data portal itself, and it should follow requirements and specifications expected as such. In the next section we revise what is exactly an open data portal and what features it must have.

[2] Joinability is a concept repeated many times along this thesis, it's understood as the probability or possibility of joining two datasets having at least one common column

## 1.2 What is an Open Data platform

An Open Data portal is an infrastructure (usually a website) that hosts a collection of datasets with free access on the internet. Usually the portal will have datasets related to each other, either by having a similar topic or by coming from the same location. When a portal starts growing in size, it is expected to also provide means to store metadata about dataset and ability to browse and search among the catalog of datasets.

The *Center for Government Excellence* (*GovEx*) at *Johns Hopkins University* [6] is an academic institution with the mission of educating governments about best practices for the collection, measurement, and analysis of data to get public trust. The GovEx center is one of the pioneers in trying to define a standard definition of requirements for a open data portal to be truly open [7].

In the GovEx standards users are defined in three groups:

- **Consumers**. Anyone who visits the portal to access and use the data
- **Publisher**. Creators of the content published on the dataset.
- **Administrator**. Manager of the website infrastructure.

In the platform developed in this project the roles of publisher and administrator are different, as everyone will be able to propose data for publishing but only the administrator will be able to accept and add this new data to the website. The publisher role will be a little different from the one defined in GovEx standards, because it will only be able to propose new data to be added, but cannot further modify it. Obviously this comes from the technical limitations of only building a proof of concept platform.

This thesis is concerned with implementing the data catalog features. Because the platform created will not be storing any dataset (only pointing towards the original resource), most of the required features are not feasible to implement. Also the community and account features are mostly out of the scope for this project, although they could be implemented.

The following section enumerates the features of a data catalog based on the GovEx standards, divided between the ones that are pretended to be implemented in this project and the ones that aren't.

### 1.2.1 Data catalog features

- **Implemented**
  - Consumers: must be able to search for datasets by one or more terms contained in dataset metadata
  - Consumers: must be able to browse or explore data by category

  - Consumers: must be able to download the catalog in a machine-readable format compatible with the common core metadata schema from a well-known address (typically /data.json)
  - Consumers: must be able to view a summary page for each dataset which details data resources, metadata, and other relevant documentation.
  - Administrators: must be able to create, edit, and retire metadata categories
- **Missing**
  - Consumers: may be able to search for datasets by terms contained in the data
  - Publishers: must be able to add, edit, or remove catalog entries
  - Publishers: may be able to mark catalog entries as private so they are not visible to the public
  - Administrators: must be able to configure default sorting for data catalog
  - Administrators: must be able to configure global settings for data catalog, including color theme, branding/logo, titling.
  - Administrators: may be able to configure additional global settings for data catalog, including custom Cascading Style Sheets (CSS), homepage display elements and layout, displayed catalog elements, and more.

## 1.3 Resources used

Although the result of the platform developed in this project is intended to be able to be integrated with any open data platform, in the initial development only two platforms are integrated. In order for the data from new sources to be added to the portal it needs to be formatted in a specific way, that is described further in the document.

The main issue with data integration is that the transformation to the required format can be cumbersome to do without proper access to the whole catalog of the data source. To be able to recollect the input data from these two sources its websites had to be scraped. In this section we discuss the two data sources initially used and why these ones. Its important to note that when talking about extracting data from the data sources it refers to the metadata and information about the available datasets, not the datasets contents itself.

This project is centered around open data portals in Catalonia, as nowadays there still does not exist a solution for unified datasets, and there is a heavy segmentation among portals from different level institutions that provide data of the same topics. As an example of a similar portal, *data.europa.eu*(Fig.1) is an open data portal at European Union level that aggregates data from several governments and

institutions of European countries, although most of the data comes from European institutions [8].

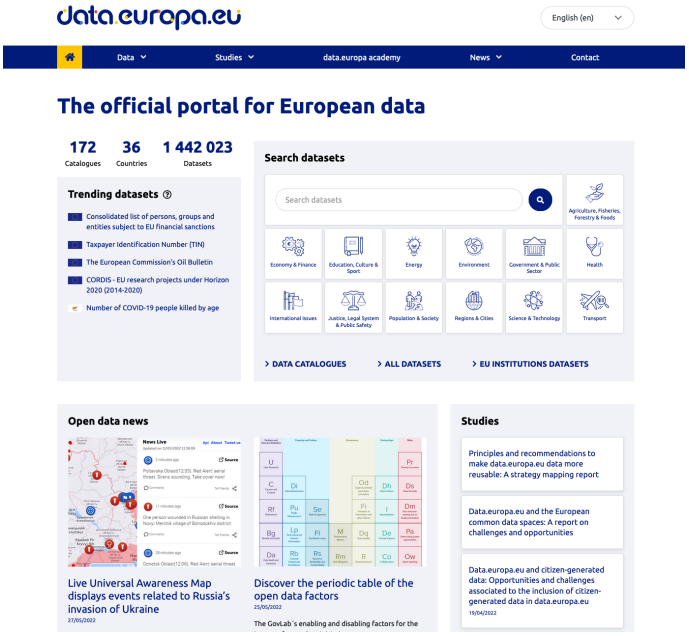


Figure 1: EU data portal screenshot

### 1.3.1 Open Data BCN

*Open Data BCN*(Fig.2) is a portal of open data for the city of Barcelona. It is administered by Barcelona city council, and has the goal to make public all the data managed by Barcelona municipal entities. It hosts around 500 datasets from different areas, mostly focused around population statistics and city governance(Fig.3).

Most of the datasets are provided in *csv* format, which are the ones that will be used in this project. Support for other file types could also be implemented, but ended up being considered out of scope for this project because of the low amount of datasets with different file types.

Some of the datasets are owned by third-party organizations and a token needs to be acquired in order to use the data. Although these datasets will appear on the project portal, the similarity measures can't be applied because of the lack of access (more details on these issues are described in future sections). The portal also has an API with access to datasets and a csv catalog with information on all the available datasets.


Figure 2: Open Data BCN home page

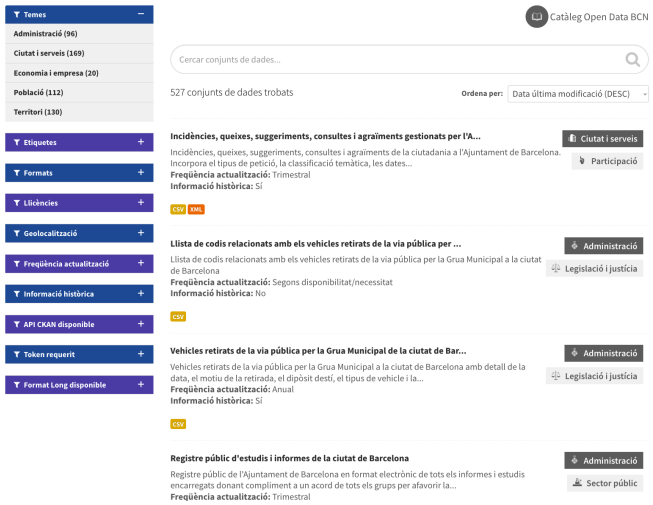


Figure 3: Open Data BCN catalog

### 1.3.2 Dades Obertes L'Hospitalet

*Dades Obertes L'Hospitalet*(Fig.4) is a portal for open data for the city of L'Hospitalet de Llobregat, managed by the city council. This portal is much more new and reduced than the Barcelona one, with only around 100 datasets(Fig.5). It's a good example of an open data portal of a small local institution.

Most of the datasets are in csv format, but there are some data sources that are not datasets, like maps for example. It also has an API, but some of the endpoints don't work correctly. It doesn't provide a catalog with data of all the available datasets.


Figure 4: Dades Obertes Hospitalet home page

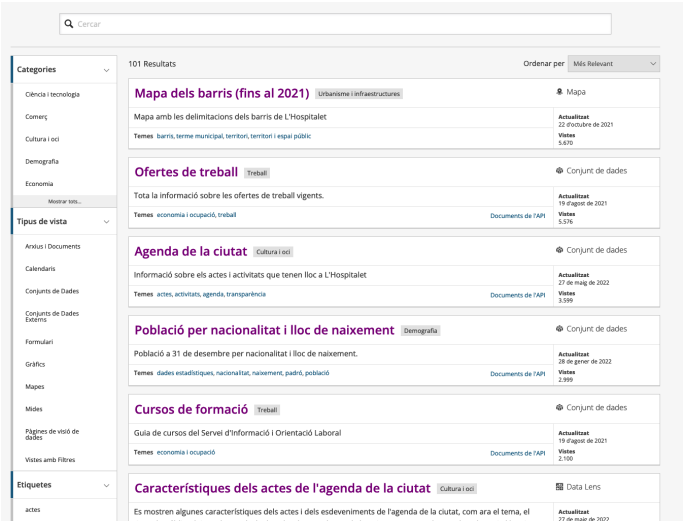


Figure 5: Dades Obertes Hospitalet catalog

# 2 Data Discovery

Data discovery can be defined as the process of exploring a group of datasets in order to find relations between them. These relations can be obtained by means of applying complex analysis in order to detect patterns in the contained data. Data Discovery is a vital process for any data science project, understood as the process of finding relevant data among a huge collection of datasets. Data Discovery can be considered part of the preprocessing required before starting the proper machine learning in a dataset, together with data cleaning for example. Data Discovery is more of an heuristic problem, without a clear solution, it is very time intensive for the user and there aren't efficient computational solutions, taking into account the large amounts of data used.

Data discovery processes can be divided into manual methods and smart or automated methods. Manual discovery is, as its name indicates, the manual classification by a human. Datasets are organized in categories, described and standardized for future use. The manual method is effective for very small collections of data, but it's not scalable at all, and it's not an option with the huge amount of data available nowadays. The smart method provides an automated solution that with the help of data mining and machine learning methods is capable of discovering patterns in data collections. Automatized data discovery is still an open problem in the computer science community, the complexity of the task and increasing amount of data makes it very difficult to find an efficient solution.

There's tendency in the business world of becoming a "data-driven" company [9], a company where business decisions are taken using collected data as a perspective of the company performance, and building the business with data generation from the first moment, in order to have large amounts of it for decision taking. So, taking into account this tendency and the exponential growth of available data on the internet (such as government open data portals), the amount of available data makes it necessary to develop automated data discovery tools. Effective data discovery is essential for any kind of data analytic, as data found and its quality will directly affect the final result.

## 2.1 Defining the problem

On this project, the data discovery is focused around tabular data. Most of the data available in open data portals is tabular data, usually in csv format. Although there is a tendency of trying to adopt NoSQL databases in many businesses, most of the data available is still in tabular datasets. Also, the most notable previous work on the field is based on tabular datasets [10]. A data discovery process can return as a result different metrics about the datasets, as the profiling can be done in different levels (dataset level, attribute level, ...). The expected result for this thesis are:

- Defining similarity measures between datasets, based on the contents of the datasets and the metadata provided. Using this similarity to detect duplicate

datasets that can be unioned.

- Defining a similarity measure between dataset columns. Using the column similarity to detect columns from different datasets that represent the same variable, and that can be joined.

Determining the similarity between two datasets is a very complex task, because there isn't a clear definition of similarity and there are many variables to measure. A dataset could be understood as a set of attributes, each attribute consists of a name and a list of values (Eq.6). The similarity between attributes can be measured in different ways, but defining the similarity between two datasets remains a complex task.

$$D_i = \{A_{i_1}, ..., A_{i_n}\}$$
$$D_j = \{A_{j_1}, ..., A_{j_n}\}$$
$$D_i \bowtie D_j \implies \exists k : A_{i_k} \approx A_{j_k}$$

Equation 6: Two datasets are joinable if they share a common atribute [11]

In this project, the representation of the datasets in order to calculate similarities will be based on profiles. Profiles are representations of datasets that contain information about its schema and data values [12]. Some common similarity techniques are:

- Comparison by value. This method is based on comparing the exact values contained in the datasets. This is the most precise method but its also the most complex and difficult to scale.
- Comparison by hash. In this approach, similarity is measured comparing the output of hash functions.
- Comparison by profile. With profile methods, some important features are extracted from the datasets to build the profiles, that can be compared between them to find common information.

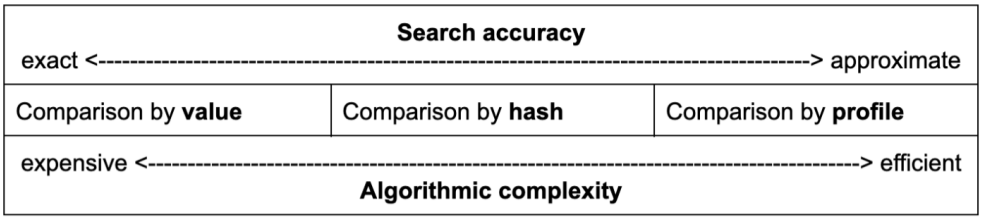


Figure 7: Efficiency of different approaches. The table is extracted from [12]

As it can be seen in the Figure 7, there is a direct trade-off between exactitude of the result and computational expensiveness. Finding exact results needs a significantly

larger amount of resources, while approximate results can be found more easily. As the goal of this project is to apply data discovery to a real web app with a potential large amount of datasets in catalog, scalability is key and the efficiency of calculations has to be maximized. For these reasons the solutions studied will revolve around dataset profiles.

Methods based on profiles allows for using a summary of the data available condensed in the form of profile, in this way much less data needs to be processed and a much better performance in terms of calculation speed can be obtained. The trade-off is that the profiles are only approximate representations of the data contained in the dataset or attribute, so the results will also be approximated.

## 2.2 Previous work

Data Discovery is a problem with extensive research done, but it is a problem still not solved. The current state-of-the-art solutions still does not provide a satisfactory result, with the main issue being the trade-off between precision and computational speed [12]. As it was discussed before the different methods for data discovery lack the efficiency to be deployed on real-time web use cases, with the exception of the profile method that sacrifices precision for speed.

Research seems to indicate that relying on profiles is the way to go in order to reach a satisfactory scalability, and *NextiaJD* [3] was one of the first software to explore this approach [12]. *NextiaJD* is a software developed at UPC, it is a tool that extracts a complex profile from a dataset(Fig.8). These profiles are used to calculate joinability between datasets, and in this way detect pairs of attributes that can be crossed. One of the limitations of *NextiaJD* is that it only considers categorical attributes. Some further research has been done to develop machine learning models to predict dataset jonability based on the profiles generated by *NextiaJD*. There has also been research around building graph neural networks to predict joinability [13]. The conclusions from the *NextiaJD* paper were that profile based methods could be a solution for the scalability problem, but that there were still some unresolved challenges on how to satisfactorily identify joins.

The similarity methods explored in this thesis are pretty much inspired by the research done in *NextiaJD*, but there is a strong focus in providing a solution that can be implemented in a web context. So there's much more focus in producing a solution that is scalable rather than in bettering the prediction results from *NextiaJD*.

[3]Repository can be found at https://github.com/dtim-upc/NextiaJD

| Category | Meta-feature | Description | Normalized |
|---|---|---|---|
| Cardinalities | Cardinality | Number of distinct values within an attribute | Yes |
| | Uniqueness | Measures if the attribute contains unique values | No |
| | Incompleteness | Measures the number of missing values | No |
| | Entropy | Measures de variety of an attribute | Yes |
| Value distribution | Average frequency | The average value of the frequency distribution count | Yes |
| | Min frequency | The minimum value of the frequency distribution count | Yes |
| | Max frequency | The maximum value of the frequency distribution count | Yes |
| | SD frequency | The standard deviation of the frequency distribution count | Yes |
| | Octiles | The octiles (quantiles) of the frequency distribution in percentages | No |
| | Min perc frequency | The minimum value of the frequency distribution in percentages | No |
| | Max perc frequency | The maximum value of the frequency distribution in percentages | No |
| | SD perc frequency | The standard deviation of the frequency distribution in percentages | No |
| | Constancy | Frequency of the most frequent value divided by number of rows | No |
| | Frequent words | The 10 most frequent words | No |
| | Soundex | The 10 most frequent words in soundex representation | No |
| Syntactic | Data type | The data type of the attribute (i.e. numeric, alphanumeric, alphabetic, nonAlphanumeric, datetime) | No |
| | Specific type | The specific type of the attribute (i.e. phone, email, url, ip, username or phrases) | No |
| | Percentage data type | The percentage for each data type detected in the data values | No |
| | Percentage specific type | The percentage for each specific type detected in the data values | No |
| | Longest string | The number of characters in the longest string | Yes |
| | Shortest string | The number of characters in the shortest value in the attribute | Yes |
| | Average string | Average length of the string in term of characters | Yes |
| | Number of words | The number of words in the attribute | Yes |
| | Average words | The average words in the attribute | Yes |
| | Min words | The minimum words in the attribute | Yes |
| | Max words | The maximum words in the attribute | Yes |
| | SD words | The standard deviation in the attribute | Yes |
| Pair metadata | Best containment | The containment score assuming all distinct values are covered | No |
| | Flipped containment | Containment assuming all distinct values are covered divided by max cardinality | No |
| | Name distance | Measures the difference of two attribute names using Levenshtein distance | No |

Figure 8: Profiles generated by NextiaJD. The table is extracted from [13]

## 2.3 A first solution

The datasets used in this project all come from open portal data, so it is worth studying what kind of data these portals provide. Usually, the open data portals not only host the dataset itself, but also provide a lot of relevant information and metadata about it. All of this information it's already provided for free by the portal itself, and it might be interesting to use it.

Some additional data provided by both Barcelona and Hospitalet portal:

- Dataset title
- Text description of the dataset
- Category
- Publish date

From these variables, it seems that categories could directly be used to classify the datasets in groups. The issue with it is that categories might differ a lot between different portals, so it's not a good way to cluster datasets. This leaves us with the dataset title and description which are interesting. Having a description of the dataset provides a lot of information that can allow a human to understand what a dataset is about. Using the dataset title and description, a human would be

capable of approximating the similarity between two datasets. Now the challenge is to translate this into an algorithm.

We have a set of datasets $D$, where each dataset is composed of a title $t$ and a description $d$.

$$D = \{D_1, ..., D_n\}$$
$$D_i = \{t_{D_i}, d_{D_i}\}$$

We can assume that we have a function sim that returns the similarity between two text strings. In this case, we can have two functions that respectively return the similarity between titles and descriptions. With those similarities we can have a third function that aggregates both results into a single similarity(Eq.9). Finally, the challenge is to come up with a sim function as an S function.

$$S(S_t, S_d) := \text{similarity between two datasets}$$
$$S_t(D_1, D_1) = sim(t_{D_1}, t_{D_2})$$
$$S_d(D_1, D_1) = sim(d_{D_1}, d_{D_2})$$
$$sim(text1, text2) := \text{similarity between two texts}$$

Equation 9: Dataset similarity based on title($t$) and description($d$)

For this algorithm we are only concerned with the title and description of the dataset, ignoring schema, profile and internal values. The goal is to find a minimum viable similarity that can be obtained only with the resources already provided by the open data portals.

In order to obtain a representative representation, both title and description are preprocessed. The preprocessing consists of first eliminating punctuation signs and transforming all the words to the lowercase. After that all stopwords are removed to keep only significant words. All the descriptions used for the similarity calculation are in Catalan, so the stopword dictionary used is also in Catalan [14]. The result is a set of key words found in the text, that will be useful to identify similarities.

The similarity function will be derived from *Tf-idf* (Eq.10). *Tf-idf* (term frequency - inverse document frequency) is a statistical function that intends to represent how important a word is in a document in relation to a collection of documents. The *Tf-idf* value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general.

$$w_{i,j} = tf_{i,j} * \log \frac{N}{df_i}$$

$$tf_{i,j} = \text{number of ocurrences of } i \text{ in } j$$

$$df_i = \text{number of documents containing } i$$

$$N = \text{total number of documents}$$

Equation 10: *Tf-idf* equation

*Tf-idf* is often used by search engines for ranking a document's relevance based on a query. A survey conducted in 2015 showed that 83% of text-based recommender systems in digital libraries use *Tf-idf* [15].

*Tf-idf* is implemented using the sklearn library[16]. The similarities are calculated separately for the titles and descriptions. First, the texts are all condensed together in a vector that is imputed into the vectorizer function. The output is a *Tf-idf* vector that contains values for all the texts. Last step is to multiply it by its transpose, and the result is a matrix of similarities between all the texts.

In the following piece of code it can be seen the process of generating a similarity matrix for all datasets.

```
1    from sklearn.feature_extraction.text import TfidfVectorizer
2
3    vect = TfidfVectorizer(min_df=1)
4    tfidf = vect.fit_transform(text)
5    pairwise_similarity = tfidf * tfidf.T
```

At this moment, we have two similarity matrices, one for the titles and one for the descriptions. A way to unify both results needs to be found, and the proposed one is defined on equation 11.

$$sim(D_i, D_j) = \alpha * sim_{title}[i,j] \ + \ (1-\alpha) * sim_{description}[i,j]$$

$$\alpha \ \exists \ [0,1]$$

Equation 11: Equation to unify title and description similarities

The equation is a weighted sum, the $\alpha$ parameter defines the relation between both values. A high $\alpha$ gives more importance to the title, while a close to zero $\alpha$ gives more importance to the description similarity. The resulting similarity is a value between 0 and 1, where 1 indicates that both texts are identical.

In the following examples, it can be seen that the result is very satisfactory. The following results are first only comparing title similarities, then only description similarities and finally both with the described equation. It can be seen that the

results are very related to the original text, sharing a lot of keywords (Barcelona, Guàrdia Urbana, accidents, ...) and always sharing the same topic. The most similar text is always the same text itself with a 1.

| Text | similarity % |
|---|---|
| **Accidents gestionats per la Guàrdia Urbana a la ciutat de Barcelona segons tipologia** | **original** |
| Accidents gestionats per la Guàrdia Urbana a la ciutat de Barcelona segons tipologia | 0.99 |
| Accidents gestionats per la Guàrdia Urbana a la ciutat de Barcelona | 0.88 |
| Persones involucrades en accidents gestionats per la Guàrdia Urbana a la ciutat de Barcelona | 0.70 |
| Persones involucrades en accidents gestionats per la Guàrdia Urbana a la ciutat de Barcelona | 0.70 |
| Descripció de la causalitat dels accidents gestionats per la Guàrdia Urbana a la ciutat de Barcelona | 0.66 |

Table 1: Similarity with original title.

| Text | similarity % |
|---|---|
| **Llistat dels tipus d'accidents gestionats per la Guàrdia Urbana a la ciutat de Barcelona. Un accident pot estar tipificat com més d'un tipus ( ex: caiguda (dues rodes) i abasti).** | **original** |
| Llistat dels tipus d'accidents gestionats per la Guàrdia Urbana a la ciutat de Barcelona. Un accident pot estar tipificat com més d'un tipus ( ex: caiguda (dues rodes) i abasti). | 1.0 |
| Llistat dels vehicles implicats en accidents gestionats per la Guàrdia Urbana a la ciutat de Barcelona. Un accident pot tenir associat més d'un vehicle. Inclou si la causa és del vianant, el tipus de vehicle, model, marca, color i tipus carnet i antiguitat de la persona que ho conduïa. | 0.41 |
| Llistat de de la causalitat dels accidents gestionats per la Guàrdia Urbana a la ciutat de Barcelona. Un accident pot tenir una o més causes mediates les quals fan referència a factors externs del resultat en temps, lloc o grau (ex: Alcoholèmia i Excés de velocitat o inadequada). | 0.33 |
| Llistat dels accidents gestionats per la Guàrdia Urbana a la ciutat de Barcelona. Incorpora el número de lesionats segons gravetat, el número de vehicles implicats i el punt d'impacte. | 0.23 |
| Llistat de les persones que han estat involucrades en un accident gestionats per la Guàrdia Urbana a la ciutat de Barcelona i que han sofert algun tipus de lesió ( ferit lleu, ferit greu o mort). Inclou descripció de la persona (conductor, passatger o vianant), sexe, edat, vehicle associat a la persona i si la causa ha sigut del vianant. | 0.20 |

Table 2: Similarity with original description.

## 2.4 Similarity based on profiles

The solution explained in the last section returns good results, but it completely ignores the data in the dataset itself. The problem with the text similarity solution is that it depends entirely on the description provided from the open data portal, which could be misleading and not be related to the dataset contents at all. In some cases the portal might not provide a description of the dataset, or we might want to compare it with a dataset out of context (not coming from an open data portal). Apart from that, we are leaving on the table a lot of valuable information from the insides of the dataset.

To sum up, the text based solution is good to have a fast recommendation system

that is able to detect related datasets (related in terms of topic) very fast. It is not a good solution to detect joinable datasets, where we are more interested in detecting similarities between columns. The benefits in finding joinable datasets is that when joined, the data can be augmented with attributes from both datasets.

Two datasets are considered joinable if a join operation can be performed between them. A join operation is an operation that combines columns from one or more tables into a new table. So, we can consider that if two datasets have a column in common then they are joinable, the more columns in common the highest is the probability of a satisfactory join.

Taking into account these observations, the problem can be divided into two parts, first finding a similarity measure between columns, and second , aggregating datasets by the number of common columns.

We have a set of datasets $D$. Each dataset $D_n$ has an arbitrary number of attributes $A_i$.

$$D = \{D_1, ..., D_n\}$$
$$D_i = \{A_1, ..., A_n\}$$

We define three functions. $P$ is the profile calculating function that outputs a profile from an attribute (details on implementation will be seen further). Then there's a similarity function between profiles and a function that calculates the ease of join from the two datasets.

$$P(A)$$
$$sim_a(P(A_1), P(A_2))$$
$$J(D_1, D_2)$$

First step is to define what is the profile that will be used and how to extract them. The first idea on this project was to use *NextiaJD* as a generator of profiles.

*NextiaJD* is able to generate extremely detailed profiles from datasets. Its profiles consist of a set of around 30 variables measuring things such as cardinality, distribution of words, syntactic data, ... . *NextiaJD* outputs one profile from the whole dataset, so there's no way of identifying individual column similarities. The creators of *NextiaJD* also developed a machine learning algorithm that is capable of predicting the joinability between two profiles generated with *NextiaJD*. An issue with *NextiaJD* is that it only considers categorical attributes, so numerical attributes are ignored(Fig.8).

When trying to implement *NextiaJD* in this problem there were several issues that ended up not being possible (which are detailed in a further section). The main reason was that it was too slow to run in real time. The goal of this project is to develop an implementation that can be used in real time by multiple users, so it's not feasible to use a heavy tool like *NextiaJD*.

Once again, there is a trade-off between exactitude in profile definition and computational efficiency. *NextiaJD* profiles are very detailed, because they have a lot of different variables but it's very slow to calculate. Taking into account the practical nature of this project, the computational performance can't be sacrificed, so a more approximate way to calculate the profiles needs to be found.

The idea between profiles is to define a statistical representation that resumes the data contained in it. We can divide the attributes between categorical and numerical, because both of them will have different profile extraction methods and they will be only compared between the same group.

If we observe the variables contained in the *NextiaJD* profile, it can be seen that many variables measure similar things and that possibly a more reduced version of the profile can be defined. For categorical attributes, this means creating a statistical representation of the categorical values contained in the attribute (words), and we already have a model for that from the last section: *Tf-idf*. *Tf-idf* can be used as a first experiment to see if a very simple method is enough to provide acceptable results.

In order to calculate the *Tf-idf* value as profile for a column, all the values are extracted and a similar preprocessing as done in last section is done (removing punctuation, transforming to lowercase and removing stopwords). Repeated words are removed and the rest is aggregated in a text. This text is the one that will be used for *Tf-idf*. Because *Tf-idf* is already a similarity measure, the profile itself (frequency for each word) doesn't need to be stored.

For the numerical attributes, a way of generating a list of statistical values from it needs to be found. In past work related to *NextiaJD* [13], it was proposed to use the python pandas function `pd.DataFrame.describe()`. This function generates a list of descriptive statistical values from a column of a dataset (attribute). The values generated are the following:

- **count**: The number of rows of the dataset
- **mean**: Mean value of the numeric attribute
- **std**: The standard deviation of the numeric attribute
- **min**: The minimum value of the numeric attribute
- **25%**: Percentile 25% of the numeric attribute
- **50%**: Percentile 50% of the numeric attribute
- **75%**: Percentile 75% of the numeric attribute
- **max**: Maximum value of the numeric attribute

This function returns a list of variables that can be understood as a profile of the attribute, but a way to calculate a similarity between different results needs to be defined. The need is to find a similarity function between vectors of variables. In the

context of information retrieval and text mining the cosine similarity is often used. Cosine similarity(Eq.12) is a measure of similarity between two numerical vectors. The vectors are in an inner product space and the similarity is defined as the cosine of the angle between them (dot product of the vectors divided by the product of their lengths). The result is bounded between 0 and 1, where 1 indicates the highest similarity.

Using the Cosine similarity(Eq.12) its possible to calculate a percentage of similarity between two columns, and if this similarity is above a threshold, both columns can be considered equal. Sharing a columns is a minimum condition for two datasets to be joinable, because the join action consists on combining rows from both tables based on the shared column value. In the approach used in equation 14 its assumed that the more columns in common two datasets have the more joinable they are. If column pairs are classified in similar or not similar (binary classification based on a threshold), using Jaccard similarity(Eq.13) makes sense as a simple similarity measure.

$$S_c(A, B) := \cos(\theta) = \frac{A \cdot B}{||A|| \cdot ||B||}$$

Equation 12: Equation for cosine similarity

$$S_J(A, B) := \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Equation 13: Equation for Jaccard similarity

$$\begin{gathered} D_i = \{A_{i1}, ..., A_{in}\} \\ D_j = \{A_{j1}, ..., A_{jn}\} \\ \vee k \; S_c(A_{ik}, A_{jk}) > threshold \implies A_{ik} = A_{jk} \\ joinability(D_i, D_j) = S_J(D_i, D_j) \end{gathered}$$

Equation 14: Equation for calculating dataset joinability assuming columns with similarity larger than threshold a to be equal

# 3 Implementation Scope

This section describes what are the goals of the project regarding the open data tool built. The following list of requirements is an exhaustive list of all the specifications that are considered indispensable to be built in order for the tool to be considered successful. The requirements are divided between functional(physical requirements that need to be implemented by the developer) and non-functional(criteria that judges the operation of the system).

## 3.1 General objectives

The main goal of this project is to develop a website that aggregates datasets from different sources and provides a way to explore the dataset collection. As a way to add value to the user, the website must provide a way to recommend similar and joinable datasets. This goal can be broken down into more detailed objectives:

- Develop a database that aggregates dataset metadata from different sources and that can be accessed with an API. The database should be able to be updated in an easy way, and adding new data should also be easy.
- Develop a front-end website that showcases the data from the database and that is easy to use.
- Explore ways to calculate similarities and joinability among datasets.

## 3.2 Functional requirements

**Back-end**

- Ability to extract data from Barcelona and L'Hospitalet websites.
  - Obtain download url, website url, title, description, ...
  - Scrap websites in an automatized way, scrapping should be fast.
  - Unify data from all sources into a single dataset or database.
- Store data extracted in a way that can be accessed with a REST API.
  - Data is preferably stored in a database that can be queried.
  - There is an abstraction layer that makes it easy to extract data stored
  - There is an API that allows you to obtain the data from external sources.
- Ability to update stored data and add data from new sources easily.
  - Easy update of data from original sources
  - A way to add data from a different source
  - All actions are highly automatized and require little manual action

- A way to calculate dataset similarity.
  - A way to calculate similarity using dataset profiles.
  - A way to calculate similarity using text descriptions in original sources.

**Front-end**

- A way to explore the catalog of datasets
  - Should showcase all the datasets from the catalog in an automatized way
  - Generate an individual page for each of them
  - Automatically identify all sources in the database, create pages for them
- Search functionality and filtering by category and origin
- Instant update from back-end changes
  - direct connection via API from back-end
- Recommendation based on text and attribute similarities.
- Users should have the ability to contribute proposing new data to the platform

## 3.3 Non-functional requirements

- Scalability
  - Platform should be able to accommodate an increasing number of datasets, without needing more development work.
- Extensibility
  - Should be able to incorporate new data sources without code changes
- Usability
  - Easy to use interface.
- Open Source
  - Final solution code should be open, with a focus to be agnostic with data sources used in order to allow it to be exported for other uses.

## 3.4 Design

When a government creates an open data portal, it's rarely built from scratch. Instead an open data delivery solution is used. These delivery solutions are SaaS services, often open source, that provide the underlying infrastructure to host the portal. Two of the most commonly used services are CKAN [17] (used for Open Data BCN) and Socrata [18] (used in Hospitalet portal). Some of the features CKAN provides [19]:

- Upload data via custom spreadsheet
- Tools to manage metadata and permissions
- Automatically generated JSON API
- Search capability and ability to explore data on the website.
- Host geospatial data

For this project the decision was made to avoid using open data delivery systems and instead create a custom solution. The main reason behind that is that this project is not going to host the data itself, so there's no use in many of the features provided by such a service. Apart from that, some of the required features for this project (such as a similarity calculation and custom search method) make it necessary to use custom code that would be impossible to integrate with a portal delivery system.

The global vision of the project is, as it can be seen on figure 15, composed of a back-end that obtains and organizes data from several original sources and a front-end that showcases the data to the users. The idea is that the back-end should perform all the required computations and the front-end should be just a lightweight interface that provides access to the content. The front-end has to be connected with the back-end via an API, that allows access to individual elements of the back-end in an efficient way. The back-end and front-end should be independent from each other and be able to be hosted on different servers.

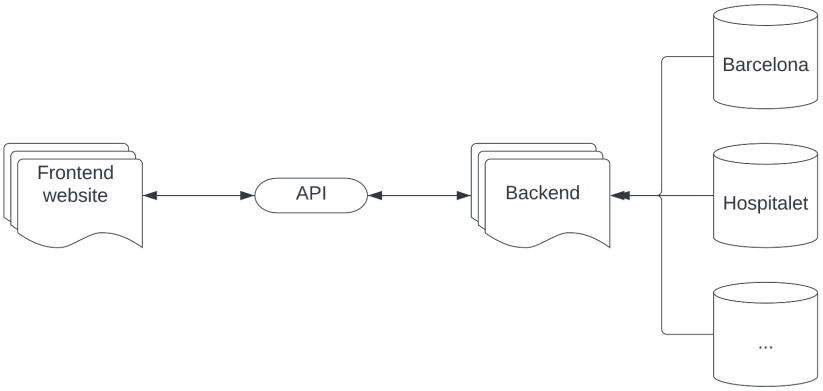


Figure 15: General structure of the project

### 3.4.1 Implementation details

The back-end main task is to extract data from different sources and convert it into a common format. The back-end also needs to perform some calculations to obtain the similarity between datasets, host data needed to populate the front-end and provide a way to update the data or add new data sources. It also needs to host the API to allow the front-end interaction with it.

Taking into account these tasks and the data intensive nature of it, it seemed that Python was the most adequate language for the back-end, because it provides plenty of libraries for dataset management such as *pandas*[20] and for machine learning such as *scikit-learn*[16].

The back-end also needs some kind of database to hold data in a structured way. During the first test implementations it was just using *csv* files to hold the data, but some kind of scalable database is needed for the final version of the project. The solution used on the final version was *TinyDB*[21], which is a simple object oriented database for Python, based on *json* files. *TinyDB* is very limited in functionality, but is enough for this project and it works as a proof of concept. For a real world project, a cloud document-based database like *MongoDB* should be more adequate.

The API is built with the library FastAPI[22]. FastAPI is a Python web framework for developing RESTful APIs. It's a tool that allows for a very fast creation of an API (as the name suggests) that fulfills the needs of the project.

The front-end is developed in React.js[23]. React is an open-source Javascript front-end library for developing user interfaces. React provides a very good developer experience, with a smooth learning curve. Its reusable UI components and state handling allows it to build scalable and very powerful front-ends, Airbnb and Twitter are two popular websites built with React. Apart of its benefits, React was also chosen for this project because of already having previous experience with it.

# 4 Back-end Implementation

The back-end can be logically structured in four areas. These areas are interconnected among them and provide different utilities that are unified and accessed through the API(Fig.16). This four components are:

- **Scraping tool**: Tool to extract the necessary data from the Barcelona and Hospitalet portals. The scraping is only used to obtain the initial data for the portal, but the idea is that adding data from new data sources should not require a scraper.
- **Upload**: Code and data packages used to add data from new data sources into the database. The upload tool should be used to add new data sources through a csv file instead of scraping.
- **Similarity Calculation**: Code to calculate dataset similarity, joinability, and its respective recommendations.
- **Database**: TinyDB database that holds data and API to access it.

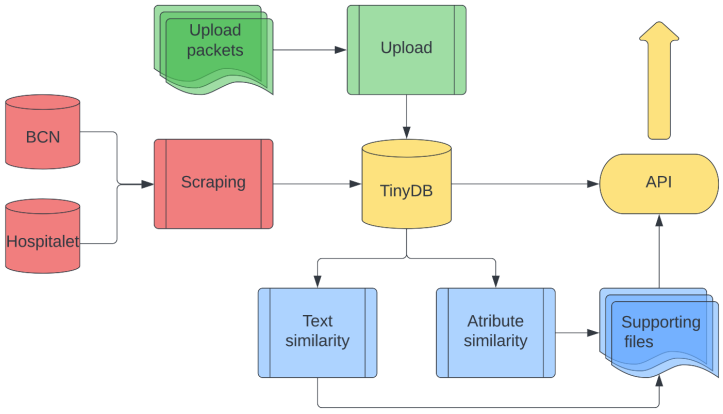


Figure 16: Back-end detailed structure

## 4.1 Scrapers

In the final product, new data sources should be added via the upload folder. These new data sources should follow the specified format and have all the information in a csv file. The idea is that third parties that would be interested in having their data available in this platform would provide the data in the specified format. To populate the initial proof of concept for this project some kind of data was needed, in this case the data from Barcelona and L'Hospitalet. This data was not available in the specified format required to upload it with the upload tool, but it was scattered through APIs and the websites itself.

The solution was to create one script for each of both data sources that extracts the

data from their websites(Fig.17). There's one additional script that unifies the data from both sources into the common format, (*generateCatalog.py*). The scraping script differs a lot from both data sources, because of the particularities of each portal.

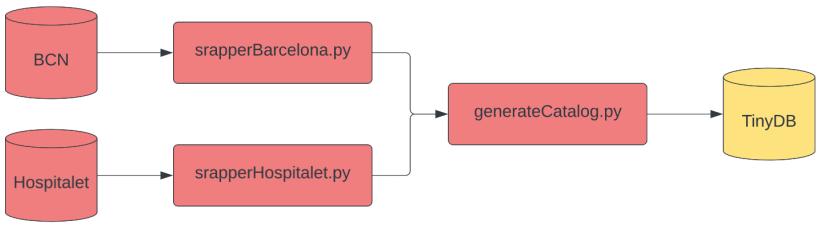


Figure 17: Detailed structure of scraper scripts

### 4.1.1 Scraping the Barcelona portal

Obtaining the data from the Barcelona portal was very easy, because the website provides access to a csv catalog with information of all available datasets. It was only necessary to transform the data to a common format. The Barcelona portal also provides an API, but it was not necessary to get the initial data, because everything needed was contained in the csv catalog.

### 4.1.2 Scraping L'hospitalet portal

The Hospitalet portal doesn't provide such a catalog, so it was much more difficult to obtain the data. It also has an API, but because it doesn't have an available catalog, it is not possible to access all the endpoints. Besides, many API endpoints weren't working. It was necessary to access the html code and move page by page retrieving the desired data.

## 4.2 TinyDB and storage

TinyDB is the database service used in the project. It is a very simple document oriented database that can be used in Python. It is basically a json format file that hosts rows of data and can be accessed with simple queries.

The following piece of code is an example of some of the functionalities of TinyDB.

```
from tinydb import TinyDB, Query
db = TinyDB('path/to/db.json')
User = Query()
db.insert({'name': 'John', 'age': 22})
db.search(User.name == 'John')
# Output: [{'name': 'John', 'age': 22}]
```

In the first iterations of the project a csv file was used instead, but for the final implementation a more robust solution was needed. TinyDB allows for a more easy

access of the data and simplifies the code a lot. It's also not acceptable from an scalability standpoint to use a csv as "database".

TinyDB is a simple solution that works well for this project, because it's just a json file that can be hosted together with the Python scripts. Because this project is just a proof of concept, this solution is enough, but in a real use case a proper database should be used.

The following text is an example of how an item is stored in TinyDB's json file.


```
"1": {
"index": 0,
"title": "Unitats administratives  de la ciutat de Barcelona",
 "description": "Detall de les unitats administratives  de la
   ciutat de Barcelona:     districtes, barris, \u00e0rea inter\
   u00e8s, \u00e0rees estad\u00edstiques b\u00e0siques (AEB) i
   seccions censals  ",
 "category": "Administraci\u00f3",
"date_published": "2017-07-06",
"source": "Ajuntament de Barcelona",
"web_url":
"https://opendata-ajuntament.barcelona.cat/data/ca/dataset?q=&name
   =20170706-districtes-barris",
"download_url": "https://opendata-ajuntament.barcelona.cat/data/
   dataset/808daafa-d9ce-48c0-925a-fa5afdb1ed41/resource/4cc59b76-
   a977-40ac-8748-61217c8ff367/download",
 "status": "-",
 "origin": "Barcelona"
}
```


## 4.3 API

The API is one of the most important elements of the back-end. Its function is to allow communication between the back-end and external elements, it's used to access its data and modify it from the front-end. As described previously, the API is built with FastAPI, a Python framework that allows easy building of REST APIs.

REST is an architectural design for APIs that is based on resources with unique endpoints that can be accessed with an http request. All resources hosted on the server can be accessed via the API with a unique endpoint that is an url. The communication between server and client is stateless, meaning that no user information is stored during requests and that each request is separated and unconnected.

The API is based around two files(Fig.18):

- *functions.py*: All the complex functions that the API needs are defined here. Basically it handles all the calculations requested from the API and the access to all the internal resources.
- *main.py*: Interface of the API, it hosts the FastAPI endpoints and accesses the functions on *functions.py*.

The use cases of the API are:

- Accessing the TinyDB database in order to retrieve information about individual datasets or the whole catalog.
- Access files representing similarity calculations for the text similarity.
- Execute the functions required to calculate the profiles of a dataset and joinability.
- Generating and updating the json files that populate the front-end based on TinyDB data.
- Accessing the json files for the front-end

It's important to note that this API, together with TinyDB works on a proof of concept basis. The number of datasets contained in the catalog is small enough for the API to handle and send all the information togheter in one packet. In a real live solution it wouldn't be possible to send all the catalog data in one packet.

API endpoints:

- `/catalog`
  - Returns the complete catalog with information from all the datasets
  - As a proof of concept works, but in case of scalability of the website with a large number of datasets, an alternative solution should be designed, to retrieve datasets in batches.
- `/item/{itemId}`
  - Returns all the information of an individual dataset.
  - Needs *itemId* as a parameter, id of dataset that wants to be retrieved
- `/frontend/MainPage`
  - Data required to populate *MainPage.js*
- `/frontend/Catalog`
  - Data required to populate *Catalog.js*
- `/frontend/DataSource/{source}`
  - Data required to populate *DataSource.js*
  - The parameter *source* is the identifier of the data source

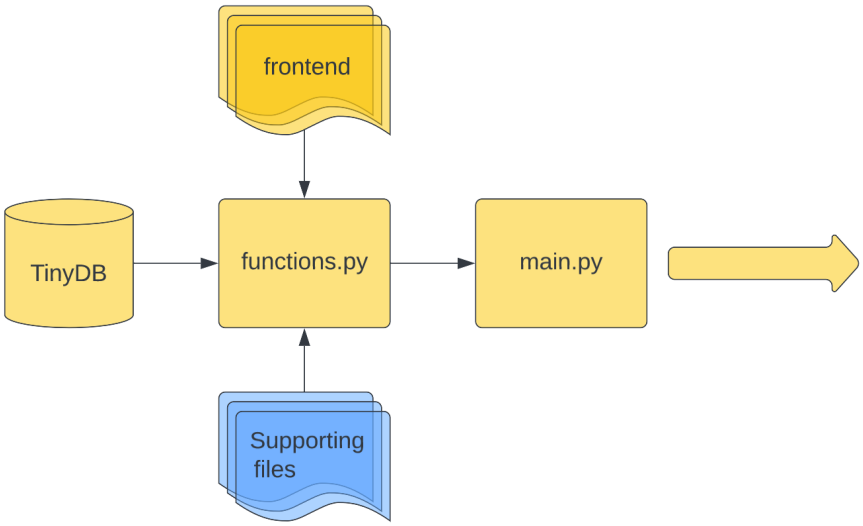


Figure 18: Detailed structure of API scripts

## 4.4 Upload

The upload tool is meant to be an utility to easily update and upload content to the database and automatically distribute the new content among the different areas of the back-end and the front-end(Fig.19). The main utility is to allow for upload of large packets of datasets from a new data source, but it also allows for updating the current data sources with new datasets.

The way it works is by receiving the new data in a specific format and first introducing it in the *TinyDB* database and second updating the data in the front-end. The idea is that new data can be introduced effortlessly with one click, and that the whole front-end and back-end will accommodate it without further action needed from the user. The main peculiarity of the implementation is that new data has to be inputted in a specific format in order to be read correctly. The packed with the new data has to be introduced in the backed files, in further work an interface for uploading could be built on the front-end, but this also meant building an authentication system to allow only an administrator to modify data, and it was out of the scope for this project.

The new data has to be introduced in a folder with the name of the data source on `/upload/uploadPackets` and it has to contain the following elements:

- `newData.csv`
- `config.json`

The data about the new datasets introduced will be contained in `newData.csv`, which needs to provide a specific set of attributes in the same order in order to be read properly. The configuration file `config.json` contains the metadata about the

data source that will be used mainly to populate the front-end with the new data source.

The following text is an example of the data required on the `config.json` file, and the table is a description of the columns on `newData.csv`.

```
{
    "databaseIdentifier": "generalitat",
    "frontendIdentifier": "Generalitat de Catalunya",
    "title": "Dades Obertes Catalunya",
    "description": [
        "...", "...", "..."
    ],
    "shortDescription": "Datasets de la Generalitat de Catalunya",
    "api": "yes"
}
```

| Variable name | Description % |
|---|---|
| title | Title of the dataset |
| description | Text describing the dataset content |
| category | Category in which the dataset pertains inside the original dataset collection |
| date_published | Publishing date of the dataset. Can be empty |
| source | Creator of the dataset |
| web_url | URL that redirects to the website where the original dataset is hosted. |
| download_url | URL to download file. Important to be able to identify similar columns |
| status | Identify the http status of the download url. Optional variable |
| origin | Dataset collection identifier |

Table 3: Columns of *newData.csv*

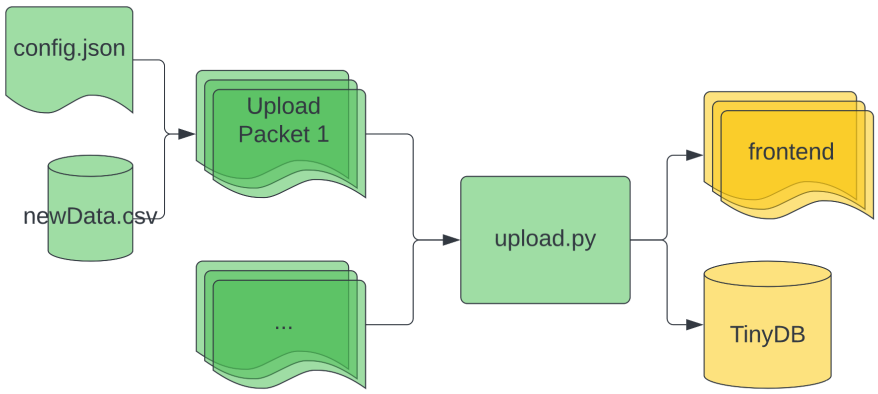


Figure 19: Detailed structure of upload scripts

# 5 Front-end Implementation

In order for the back-end functionalities to be accessed a user interface is required. For this project the option was to create a website, as the goal of it is to be publicly accessible. This website is supposed to be a way for any user to interact with the back-end, the basic functionalities provided are:

- Explore the data catalog and download or access the original dataset source
- See recommendations for a specific dataset, either based on description similarity or on profile joinability.
- Have information about the several data sources available.
- Provide a platform for users to submit new data.

As mentioned before the front-end is built using React. This framework allows to design reusable components that can be then used in several different places. Other peculiarities are the use of the Router library [24] that allows linking the different pages with unique urls(Fig.21) and MaterialUI[25], a front-end library that is used for most of the UI design.

The front-end is designed in conjunction with the API, in a way that all its content is automatically adapted from the API(Fig.20). All the datasets showcased on the front-end are fetched from the API, together with the data sources information and metadata. Thanks to React functionality [26] the information from the API is fetched every time the page is refreshed, so the information is always updated.

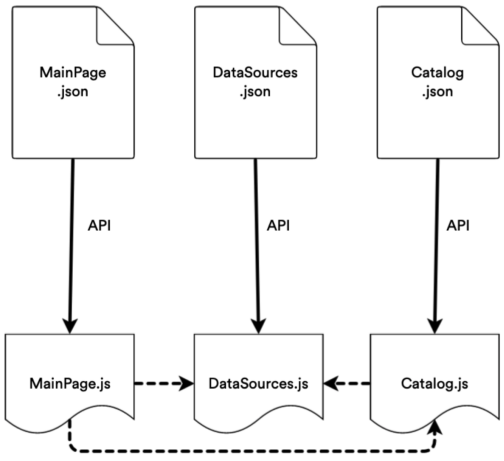


Figure 20: Diagram of front-end API calls

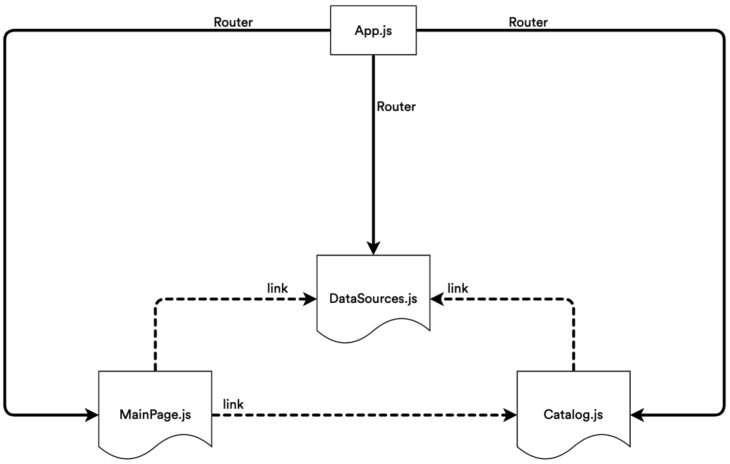


Figure 21: Diagram of front-end Router connections

Each of the following files renders a page on the website.

## App.js

This file is one of the only ones that doesn't render a page on the website. This is the initial page the browser accesses and it contains the routing endpoints for the rest of pages.

## MainPage.js

The main page(Fig.22) acts as a "home" for the website, it's the first page that users will see. It includes a header space that should provide some initial information about the website. After that there's a link to the catalog of datasets. There's a section with information of all the data sources available on the website and a list of all the categories.

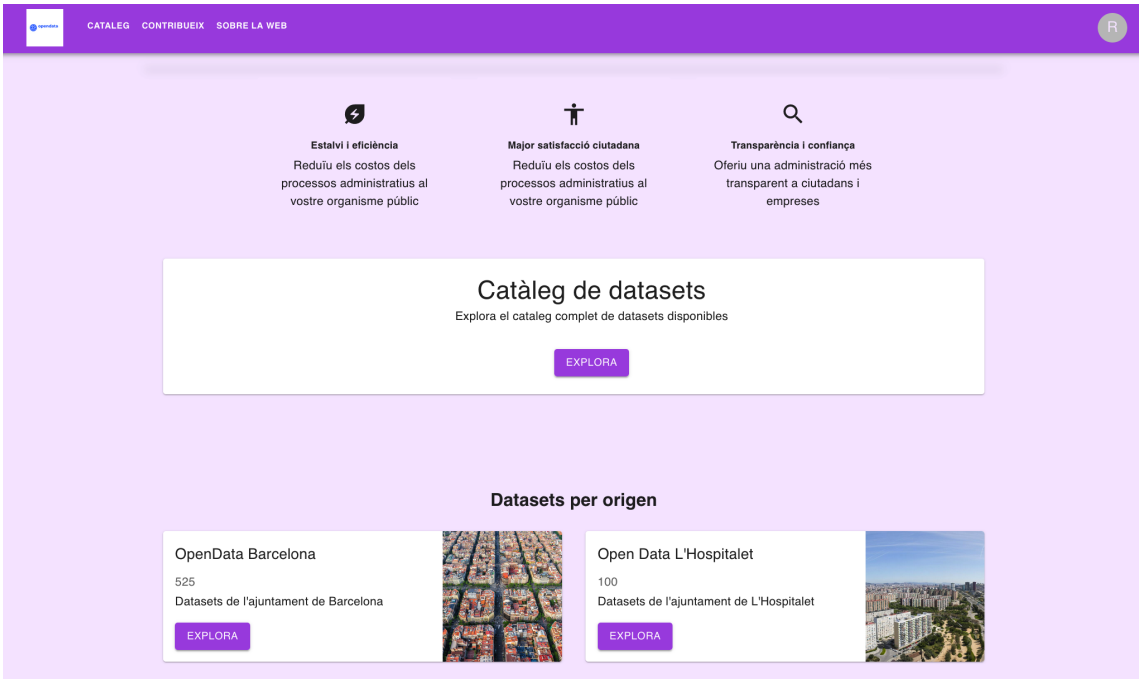


Figure 22: Screenshot of MainPage.js render

## Catalog.js

The catalog page(Fig.23) provides access to the whole collection of available datasets. Its intention is to allow the user to browse the collection and search for specific datasets. It contains search functionality based on the description text and title and the search can also be filtered by category or by the dataset source. Only 10 datasets are shown at the same time, and there's a functionality to advance the page.

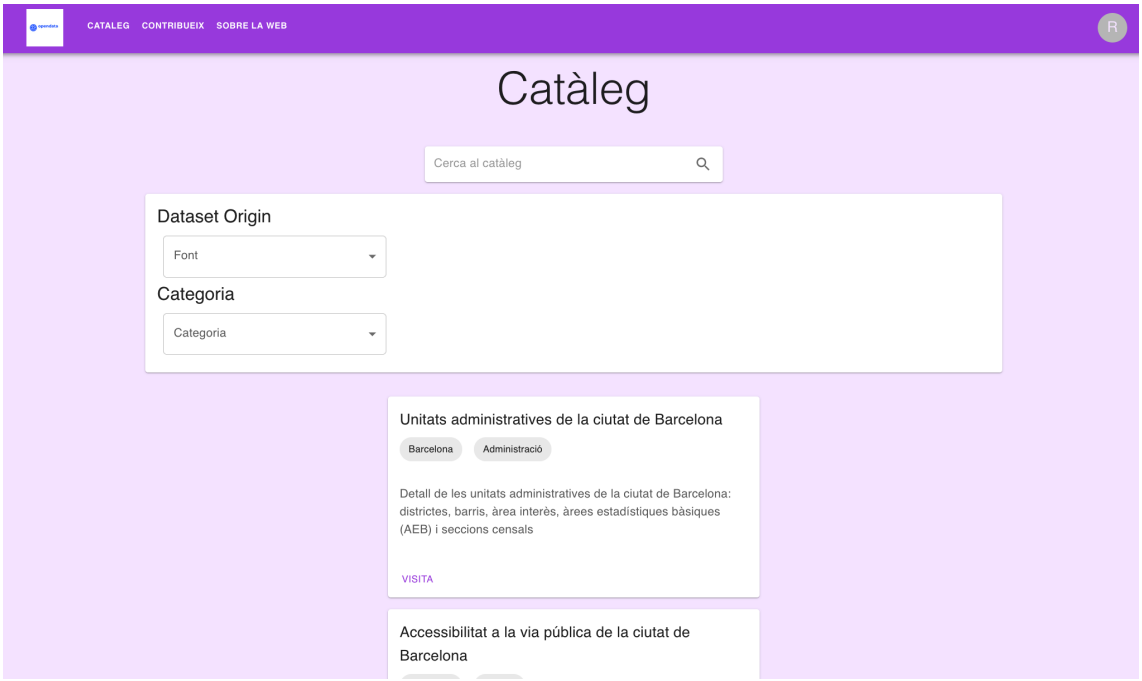


Figure 23: Screenshot of Catalog.js render

## Contribute.js

The goal of this page(Fig.24) is to make it possible for external users to contribute to the website catalog, and propose the addition of new data sources to it. In order to add new data, users are expected to provide a catalog of datasets in csv in a required format. The format is very important to follow the guidelines, otherwise it cannot be automatically introduced into the database, that's the reason why there's an example row with all the necessary columns and a description of each one of them.

The page links to a Google Form(Fig.25) that lets the user upload the csv together with some basic information. The answers to the form are received by the owner of the portal, who can decide whether to introduce the submitted information in the catalog. The form to submit the data was chosen because it was much easier than implementing an upload mechanism in the website itself. As future work, it would be interesting to have a way to submit directly from the website.

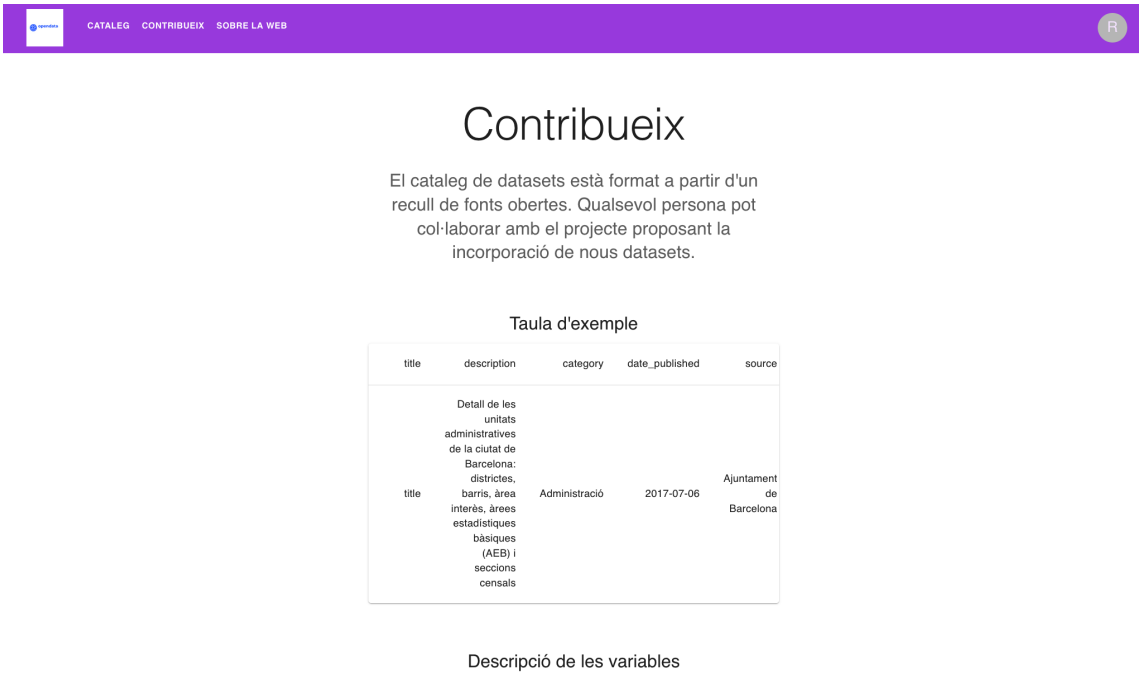


Figure 24: Screenshot of Contribute.js render

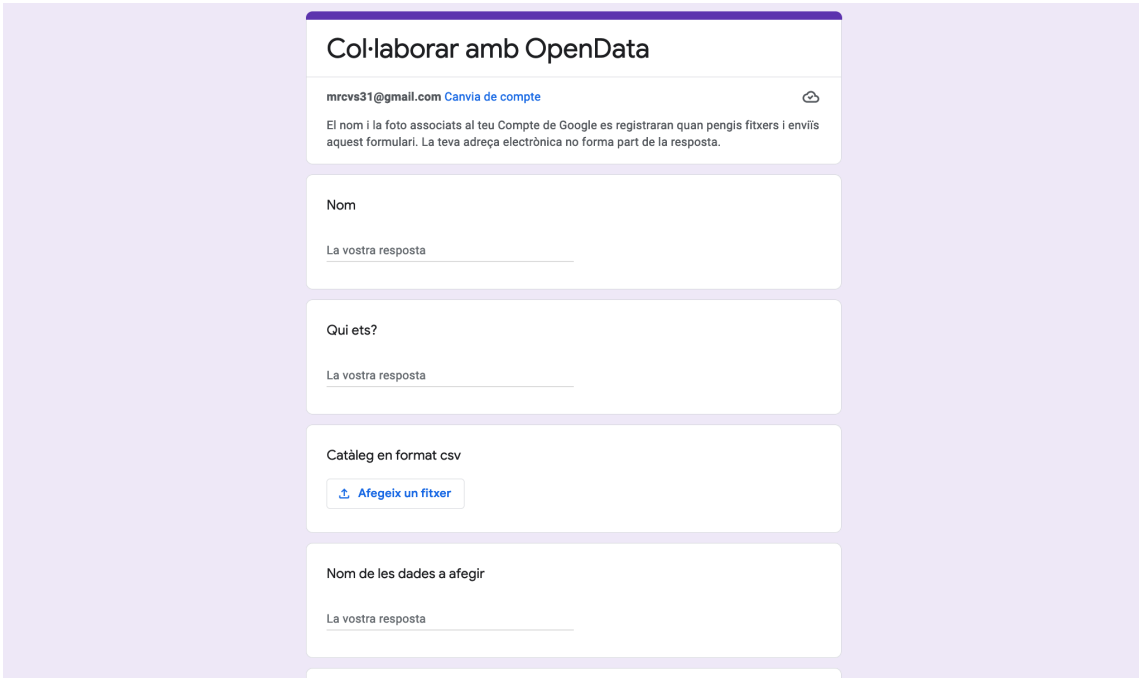


Figure 25: Screenshot of the form to upload new data

## DataSources.js

DataSource(Fig.26) is a page that shows basic information about a specific data source. The page can be rendered with a parameter that defines which data source wants to be visited, so that only one page is needed in order to have an individual page for each data source. It shows a description of the data source, the number of available datasets, etc.

The page for each data source is generated automatically from the database. Through the API, the front-end identifies the several data sources available in the catalog and retrieves metadata about them, that is shown on the front-end.

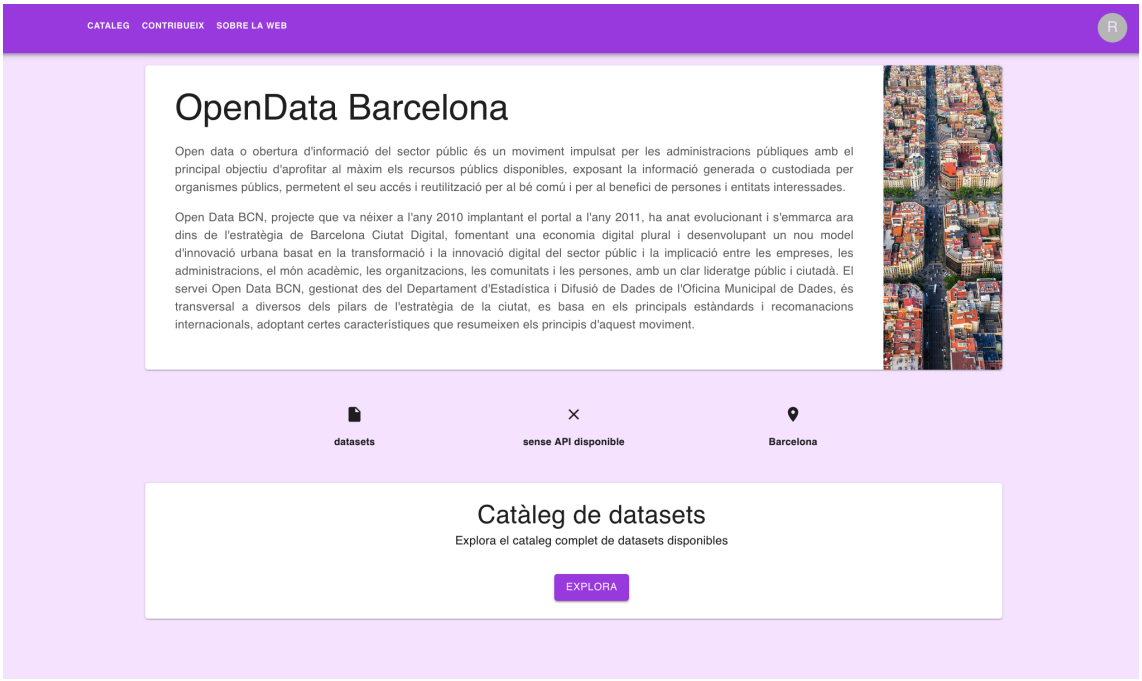


Figure 26: Screenshot of DataSources.js render

## DatasetPage.js

DatasetPage(Fig.27) is a page that shows basic information about a specific dataset. In the same way that DataSource.js does, the page can be rendered with a parameter to define which dataset wants to be visited. All the dataset pages are also rendered automatically based on data fetched from the API.

The page not only shows basic information related with the dataset, but it's also meant to be a way to explore the profile calculation and similarity based recommendation. Just after the information about the dataset, the columns of the dataset are shown and on click the collection of similar columns is found. In this way the user can explore the recommended similar columns and access the datasets of such columns that are probably joinable. After that there's a list of recommendations based on joinability and a list based on text similarity.

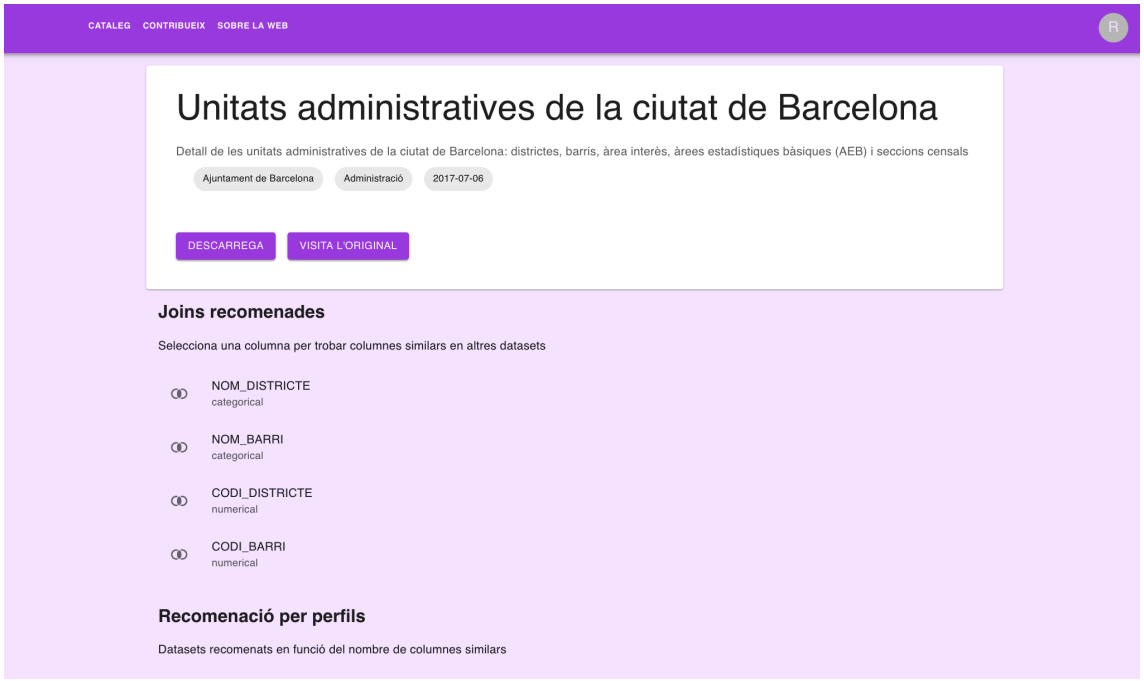


Figure 27: Screenshot of DatasetPage.js render

# 6 Unsuccessful implementations

The original goal for this project was to implement the web app around *NextiaJD*, and use the profiles generated from it to calculate joinability and similarity. The intention was to also use machine learning models created on past related research to predict the joinability. At the moment of implementation, many problems were found that made it not viable to use *NextiaJD* on the web app.

One of the most important requirements was for the similarity and joinability calculations to be real time in order to be usable on a web format. One of the issues with *NextiaJD* is that it's quite slow in calculating the profiles, because the profiles are very detailed, containing a large amount of parameters. Calculating a profile between two datasets that can be used with machine learning can take between 30 and 40 seconds. Taking into account that the catalog (containing data only from Barcelona and Hospitalet) has around 600 datasets and all the pairs' similarities should be calculated, it makes it impossible. And the real life version of this project should have even more datasets.

For *NextiaJD* to calculate the profile, data needs to be inputted in csv format. After experimenting uploading several csv files retrieved from the open portal data, it was found that in many cases *NextiaJD* wasn't able to calculate the profile because of little changes in csv format. Because of this lack of uniformity among csv files from open data portals it is very difficult to build a working pipeline that downloads the files from the portals and calculates its profiles with *NextiaJD*.

The main fault of not being able to use *NextiaJD* is, with exception of its speed, the lack of standardization in open data portals. Open data portals are meant to be used using the website interface, and for the users to download individual datasets or explore them with the website incorporated tools. In many cases, even though they provide an API, they are not meant to have all its data downloaded in bulk. Some of the issues found that make the processing of bulk data difficult are:

- API endpoints that don't work (mostly a problem with Hospitalet portal)
- Not having a clear definition of the file type. The file needs to be downloaded and the type guessed.
- Files defined as csv that are actually of json type.
- Different codifications in csv files that are not defined and have to be guessed (utf-8 and utf-16, for example).
- Unstandardized csv format. Some datasets don't have column names, the delimiters can be different in many cases.
- Errors in the dataset itself. Rows without all the elements, empty spaces. A very common issue is having a delimiter written inside the text and creating

an error on parsing [4].

Most of these errors are not visible to users that are only interested in exploring the data through the internal tools, which is the majority of users. Also, even for users that want programmatic access, the usual is just to download a specific dataset they are interested in and so there isn't much of an issue. The big problem is when trying to programmatically download all available data and process all of it, that these problems start to accumulate and it's difficult to manage them.

Another issue with *NexiaJD* is that it's very heavy and difficult to install. The fact that it's written in Scala and it requires you to have installed Scala and Java makes it need a lot of space. This can be an issue when trying to install the back-end in some cloud server in order to deploy the website, where the size required is larger than what usually the free instances provide [5].

[4] For example, in a csv with "," as a separator, a cell with the content "Sant Pere, santa Caterina i la Ribera" will be wrongly divided in two, because it contains the separator itself as a character of the string.

[5] Amazon EC2 free tier includes the t3.micro instance, that provides 1 GiB of memory

# 7 Further work

The final prototype developed for this project has all the minimum requirements necessary to consider it a success. Although it accomplishes all the specifications defined in last sections, it just works on a proof-of-concept basis, and would need many changes in order to become a fully scalable web version that could be used by thousands of users. Apart from these limitations, the front-end could have much more features implemented, that although not indispensable it would improve the user experience. Most of these features were not implemented because of time restrictions and being out of the main scope for the project.

In terms of scalability the principal bottleneck is the API and the database. The database currently is just a json file with some querying functionalities (TinyDB), and although this works for now, it's difficult to envision a real live version of this project hosting thousands of datasets and still using a json file as a database. A migration to a real object oriented database shouldn't be too complicated, and it's the most logical upgrade for a project like this.

On the other hand, the API is not designed to handle large amounts of data, for example the whole catalog is sent on a single API call. This is probably considered a bad practice, and with a growing collection of datasets it shouldn't be possible. The solution is to rebuild the API with scalability in mind, and probably sending data in batches instead of all together.

Regarding to the possible front-end additions, the main one would be an administrator interface. In the current version, in order to add new data to the database the administrator needs to manually upload it to a specific folder of the back-end. This could be done easily with an interface where new data is uploaded and posted with a one button click. Going even further, now the users submit data through a Google Form, but there could be an interface in the website itself that allows users to post new data, automatically adapts it to the required format and wait for an acceptance from the administrator to submit it on the database. Another front-end update would be to allow users to login in accounts. With these accounts users would be able to save datasets, comment on them and check the state of proposed new data. Implementing all these features required a large amount of work and non-trivial learning to know on how to, which ended up being considered too out of scope for this project.

There are many details in the website that could be added, but there was a time limitation for this project and priorities had to be handled. It would be nice to have a complete "About" page with information about the project, and the landing page could have more features and a better design.

Finally, the project came to completion but never got to be a real website because of a lack of available hosting. The website should be hosted on a server and available through the internet to anybody. Although this isn't possible yet, the project was built from the beginning with this goal in mind and everything is designed and

documented to make a deployment straightforward once there's an available server.

# 8 Conclusions

In general, the requirements defined during the thesis were accomplished with the delivered result. Developing and building the tool can be considered a success, as the resulting website can be used for its intended purpose, that is exploring a unified catalog of datasets. Also, the exploration of similarity measures between datasets resulted in an acceptable method that is proved to deliver good results.

The general objectives described in the "Implementation scope" section are satisfied, and although as described on the "Further work" chapter there are a lot of missing things that could be implemented, the current functionalities work in an expected way and as a proof-of-concept it's enough to have an overview of the possibilities of such tool.

In regards to the search for an efficient similarity measure, the approaches explored in this thesis are much simpler than the ones done in previous work. Mainly because of the failure in trying to integrate NextiaJD into the solution, it was forced to look for simpler ways to calculate similarity and joinability. The final solution is an approximation to similarity calculation, but this approximation is traded for a much easier calculation process that allows for true scalability in a website context.

The resulting recommendations from the similarity and joinability calculations seem to be good enough. It seems that in a web context where users are more interested in having a recommendation from a huge pool of available datasets, the precision of the recommendation is not something indispensable, and that an approximate recommendation is good enough for users to discern a small set of interesting datasets from the total of the collection.

Finally, there's plenty of lessons learned from the process of building this tool. First is the appreciation for the complexity of building a website with this amount of integrated parts. From the outside it can seem easy, but connecting all the parts and having a global vision of such a huge codebase is a challenge.

As lessons learned, the main one is about organization. This project started with a general vision of its direction, and new ideas and features were added with time. This was a big challenge in how to manage the codebase to add these changes, and the issues of not starting with a clear vision of the project were evident. In many cases it was needed to fully refactor parts of the code that were no longer useful or required big changes. All of this added a lot of working hours that could have been avoided with a better planning from the beginning, an example of this issues was the decision of abandoning the idea of integration NextiaJD after hours and hours trying to make it work.

# 9 Annex

## 9.1 Published code

The final code is publicly available on two Github repositories, one for the front-end and one for the back-end. The decision of storing the code in two different repositories is that the back-end and front-end are independent and and be hosted on two different places different.

- Front-end repository: https://github.com/marcvernet31/TFG-frontend
- Back-end repository: https://github.com/marcvernet31/TFG-backend

The structure for the front-end(Fig.29) is the standard file structure for a React project, with the main code hosted on `\src`. For the back-end(Fig.28), the repository is structured in folders based on the function of the scripts.

```
TFG-backend
├── api
│   ├── frontend
│   │   ├── Catalog.json
│   │   ├── DataSource.json
│   │   └── MainPage.json
│   ├── functions.py
│   └── main.py
├── atribute-similarity
│   ├── columnProfile.py
│   ├── similarityCategoricalMatrix.py
│   ├── similarityMatrices.py
│   └── stopwords_ca.py
├── scrappers
│   ├── generateCatalog.py
│   ├── scrapperBarcelona.py
│   └── scrapperHospitalet.py
├── text-similarity
│   ├── similarityMatrices.py
│   └── stopwords_ca.py
├── upload
│   ├── uploadPackets
│   │   ├── template
│   │   │   ├── config.json
│   │   │   └── newData.csv
│   │   └── ...
│   ├── update.py
│   └── upload.py
└── guide.txt
```

Figure 28: File directory for back-end Github repository

```
TFG-frontend
├── public
│   └── ...
├── src
│   ├── About.js
│   ├── 404.js
│   ├── App.js
│   ├── Catalog.js
│   ├── Contribute.js
│   ├── DataSources.js
│   ├── DatasetPage.js
│   ├── JoinPage.js
│   ├── LogIn.js
│   ├── MainPage.js
│   ├── SignUp.js
│   ├── index.js
│   ├── reportWebVitals.js
│   ├── userContext.js
│   ├── assets
│   │   └── ...
│   └── components
│       ├── Footer.js
│       ├── Main.js
│       ├── MainFeaturedPost.js
│       ├── MultipleSelect.js
│       └── ResponsiveAppBar.js
├── package-lock.json
└── package.json
```

Figure 29: File directory for front-end Github repository


(a) Back-end (b) Front-end

Figure 30: Programming languages used in each repository

## 9.2 Instructions for deployment

From an overview, the project is divided in two blocks, the back-end and the front-end. Both blocks can be installed separately on different servers or together, and its installation is independent from each other. In this guide it's assumed that the installation is done locally on the same machine, for sake of simplicity. This project was entirely developed on a 2020 Apple M1 computer, so the commands in this section are for MacOS, which should differ from the commands needed for a Linux computer.

### 9.2.1 Back-end

In order for the front-end to work properly, an installation of the back-end is required first. The back-end is fully written in Python, so the only requirements for it are having installed Python itself (I personally used version 3.9), and the required libraries.

Because of the code needs quite a lot of time to do the initial setup and be able to see the results, in the repository there's attached a working test version already populated with data compressed in a zip file. This version is recommended to use for testing purposes, and only needs to be uncompressed and after that run the api command.

Install code from the github repository and access the home folder. All the following commands after installation are assumed to be executed from the home directory.

```
> git clone https://github.com/marcvernet31/TFG-backend.git
> cd TFG-backend
```

Execute the scraper scripts to generate and populate the catalog from the datasets available in Barcelona and Hospitalet portals.

```
> python3 scrappers/generateCatalog.py
```

Calculate the similarity based on text descriptions for all the dataset pairs in the catalog.

```
> python3 text-similarity/similarityMatrices.py
```

Personally, I had many troubles installing the python library scikit-learn on my work computer (Apple M1) and I ended up having to install the library through a Conda environment. In that case, the command used is instead:

```
> conda run -n <env-name> python3 text-similarity/
  similarityMatrices.py
```

The next step is to execute the script to extract the column values for every dataset in the catalog and calculate the profiles. This operation is very slow, because it needs to individually download every dataset in the catalog, check that the format is correct and calculate the profiles for individual columns. The long execution time

is not a big issue, because this process should be executed only once to create the profiles.

```
> python3 atribute-similarity/columnProfiles.py
```

Now that the column values are extracted, it's time to calculate the profiles and similarities

```
> python3 atribute-similarity/similarityCategoricalMatrix.py
```

Again, if there are issues with the sikit-learn library, it can also be executed with:

```
> conda run -n <env-name>python3 atribute-similarity/
  similarityCategoricalMatrix.py
```

Finally, the API can be started. While all the last commands were just to set up all the required files, this command is the one that keeps the API working, and should be kept running for a s long as the website needs to be working. The ideal would be to have a server space dedicated to running the API.

```
> cd api
> uvicorn main:app --reload
```

(explain where the api runs, localhost, ...)

There are also the update and upload utility scripts. The upload scripts introduce new data to the database in the form of a packet (specific description on ....) and the update script regenerates the front-end files with the current data on the database.

```
> python3 upload/upload.py -packet <name>
> python3 upload/update.py
```

### 9.2.2 Front-end

The front-end because of it being a React app, it's fully contained and needs very little configuration. The only requirement is to have installed the Node Package Manager (npm)[27] for Javascript.Assuming that npm is installed, the front-end files can be installed with:

```
    > git clone https://github.com/marcvernet31/TFG-frontend.git
    > cd TFG-frontend
    > npm install
```

And the program can be executed with:

```
    > npm start
```

By default, the frontend is hosted on the `localhost:3000`, and it assumes that the back-end is hosted on `localhost:8000`. The base url for the back-end is hardcoded in many places in the front-end, and it should be changed if the back-end is not running on the same local machine.

# References


[1] Meghan Malas. Glassdoor's no. 3 best job in the u.s. has seen job growth surge 480%. *Fortune*, March 2022.

[2] Thomas H. Davenport and D.J. Patil. Data scientist: The sexiest job of the 21st century. *Harvard Business Review*, October 2012.

[3] Karl Lofgren and C. William R. Webster. The value of big data in government: The case of 'smart cities'. *Big Data Society*, January 2020.

[4] T. Berners-Lee C. Bizer, T. Heath. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, July 2009.

[5] A european strategy for data. https://digital-strategy.ec.europa.eu/en/policies/strategy-data. Accessed: 2022-05-10.

[6] Center for government excellence at johns hopkins university. https://govex.jhu.edu/. Accessed: 2022-05-10.

[7] Open data portal requirements. https://labs.centerforgov.org/open-data/portal-requirements/. Accessed: 2022-05-10.

[8] The official portal for european data. https://data.europa.eu/en. Accessed: 2022-05-10.

[9] *Designing Data-Intesive Applications*.

[10] Oscar Romero Javier Flores, Sergi Nadal. Effective and scalable data discovery with nextia. *OpenProceedings*, 2021.

[11] Christos Koutras, George Siachamis, Andra Ionescu, Kyriakos Psarakis, Jerry Brons, Marios Fragkoulis, Christoph Lofi, Angela Bonifati, and Asterios Katsifodimos. Valentine: Evaluating matching techniques for dataset discovery. *CoRR*, abs/2010.07386, 2020.

[12] Oscar Romero Javier Flores, Sergi Nadal. Towards scalable data discovery. *OpenProceedings*, 2021.

[13] Zoé Marie Meini. Graph convolutional neural networks for open data discovery. *UPC*, 2021.

[14] Catalan stopword repository. http://latel.upf.edu/morgana/altres/pub/ca_stop.htm. Accessed: 2022-05-10.

[15] Bela; Langer Stefan Breitinger, Corinna; Gipp. Research-paper recommender systems: a literature survey. *International Journal on Digital Libraries*, 2015.

[16] scikit-learn. https://scikit-learn.org/stable/. Accessed: 2022-05-10.

[17] Ckan. https://ckan.org/. Accessed: 2022-05-10.

[18] Socrata. https://dev.socrata.com/. Accessed: 2022-05-10.

[19] Guide to creating, using, and maintaining open data portals. https://www.safe.com/blog/2021/04/guide-creating-using-maintaining-open-data-portals/#creating. Accessed: 2022-05-10.

[20] pandas. https://pandas.pydata.org/. Accessed: 2022-05-10.

[21] Tinydb. https://tinydb.readthedocs.io/en/latest/. Accessed: 2022-05-10.

[22] Fastapi. https://fastapi.tiangolo.com/. Accessed: 2022-05-10.

[23] React.js. https://reactjs.org/. Accessed: 2022-05-10.

[24] Router for react. https://reactrouter.com/. Accessed: 2022-05-10.

[25] Material ui. https://mui.com/. Accessed: 2022-05-10.

[26] React's useeffect. https://reactjs.org/docs/hooks-effect.html. Accessed: 2022-05-10.

[27] Node packet manager. https://www.npmjs.com/. Accessed: 2022-05-10.