UNIVERSITY OF PADUA

Department of Information Engineering - DEI

Master Degree in Computer Engineering

A Web Application for Searching Fairness Datasets

Supervisor

Prof. Gianmaria Silvello

Candidate

Alberto Piva

Badge 2048674

Co-supervisors

Dott. Alessandro Fabris

Dott. Fabio Giachelle

Academic Year 2021/2022
Graduation date 12/12/2022

Contents

List of Figures

List of Algorithms

Abstract

Data-driven algorithms are being studied and deployed in diverse domains to support decisions. They directly impact on people's life and for this reason more and more researchers are analyzing the equity of existing algorithms and they are proposing new ones. Algorithmic fairness progress is based on data, which can be used in a correct way only if sufficiently documented. As stated by Fabris et al. (2022) "Unfortunately, the algorithmic fairness community, suffers from a collective data documentation debt caused by a lack of information on specific resources and scatteredness of available information.". This thesis is strictly connected with the researches done in this field by Prof. Gianmaria Silvello and it proposes a Web Application to support and share their work.

Gli algoritmi basati sui dati vengono studiati e utilizzati in diversi ambiti per supportare le decisioni. Questi impattano direttamente sulla vita delle persone e per questo motivo sempre più ricercatori analizzano l'equità degli algoritmi esistenti e ne propongono di nuovi. L'equità algoritmica si basa sui dati, che possono essere utilizzati in modo corretto solo se sufficientemente documentati. Come riportato da Fabris et al. (2022) "Purtroppo, in questo ambito si soffre di un debito collettivo di documentazione dei dati, causato dalla mancanza di informazioni su risorse specifiche e dalla dispersione delle informazioni disponibili.". Questa tesi è strettamente legata alle ricerche condotte in questo campo dal Prof. Gianmaria Silvello et al. e propone una Web Application per supportare e condividere il loro lavoro.

Introduction

Algorithms are a fundamental key point in Computer Engineering and Computer Science fields. In the last few decades they have been more and more concentrating on data and for this reasons they are called *data-driven algorithms*.
The community of researchers that is concentrating on the algorithms fairness is growing. They study these types of algorithms which are used in various domains.
An important consideration to make is that those algorithms have been used to make decisions which directly influence and impact our every day life.
The problem we will analyze in this thesis is related with the difficult situation given by the lack of documentation of the data which is used in the *data-driven algorithms*. The reader should consider that the previous mentioned algorithms can be used correctly only if they are sufficiently documented.

This thesis is structured around four main chapters, without considering the this Introduction chapter and the Conclusions. The first one is an introduction to the algorithmic fairness and in particular it resumes the researches made by Fabris et al. (2022), which is the background around this thesis. In conclusion there is an explanation about the data brief structure composition and a discussion about the contribution of this work.
In the second chapter the reader can find the problem modeling: there is a brief introduction to Open Data and then it is explained how the ontology has been developed with all its implementation details.
In the chapter number three the data parsing has been explained, in particular which type of serialization was chosen and all the implementation details about

the developed parsers and serialization functions. Finally we explain how we used PostgreSQL to implement the full text search.
The forth chapter is divided in three main sections. The first one describes briefly the Web technologies used to develop the back-end and the front-end, so we talk about Django and React with their advantages.
In the second one we explained how the Web application back-end was developed in practical terms, so there are some example of code and we discuss some implementation decisions.
Finally, the third section talk about the Web application front-end development. It presents techniques and consideration done during its implementation.

Introduction to Algorithmic Fairness

2.1 Problem definition

As we can understand from the Introduction, this thesis deals with the data used by algorithms and all their related aspects, in particular from the fairness point of view. We can translate fairness with justice, equity, bias, power and harms.

Another key point is given by data selection and utilization by users.

Two important works are published by Gebru et al. (2018) and Holland et al. (2018), they are considered two complementary frameworks and they are called respectively *Datasheets fror Datasets* and *Dataset Nutrition Labels*.

They are recognized important because they can help data producers to follow best practices during data curation and data consumers are able to choose and use datasets within the best productive way.

These new "standard" of work have influenced also the Machine Learning field so much so that in the Conference on Neural Information Processing Systems (NeurIPS) has been introduced a way to track datasets into repositories, which are particularly useful in scholarly articles and in all academic and business researches world.

It is important to notice that it is applicable also to existing datasets (Bandy and Vincent (2021), Garbin et al. (2021)) classifying their proprieties (Prabhu and Birhane, 2020) and tracking their usage in researches (Peng et al., 2021).

Recently Fabris et al. (2022) proposed the extension of the notion of *documentation debt* by Bender et al. (2021) to all the collections of datasets utilized in a research filed because the initial notion considers only the training sets.
Moreover, they identified two aspects that produce the above-mentioned documentation debt: the first one is the *opacity* which represents the inadequate documentation relative to single documents, the second one is the *sparcity* which refers to the presence of relevant information but they are insufficiently connected with the data.
An example of these relevant aspects is given by the German Credit dataset (UCI Machine Learning Repository, 1994). In fact, in recent works of algorithmic fairness where this dataset is employed, the sex attribute is set as protected attribute (He et al. (2020), Yang et al. (2020) and others), which means that it is a feature that can not be used as basis to make decisions. On the contrary, the existing documentation related with this dataset shows that this feature can not be retrieved (Grömping, 2019).
Another important facet is the the relation between datasets and the tasks or the domains where they have been employed, which may be unknown.
A key instance is given by the BUPT Faces datasets: it is known as the second existing resource for face analysis with race annotations (Wang and Deng, 2020).

2.2 Problem analysis

Fabris et al. (2022), to reduce the problem produced by the documentation debt of the algorithm fairness community, examined the datasets used in more than 500 articles published in that field. They decided to select them from the most important conferences and workshops in the period from 2014 to 2021.
This work produced the so called *data briefs*, which are a sort of compact and standardized documentation of the datasets found in the articles considered. To give an idea of the work behind the Fabris et al. (2022) publication, they found over 200 datasets connected with the articles of algorithmic fairness.
Data briefs are a kind of summary of the datasets where we can find the key

proprieties, such as the purpose, the features (and in particular the sensitive ones), the labeling procedure and the envisioned ML task, if present. Moreover data briefs also indicate the domain of processes which produced the data and the tasks in which the dataset is used. With these last two information a user can also make search domain-based or task-based.

Fabris et al. (2022) to obtain all this detailed information, which are often not given, contacted creator of the datasets and all the researchers involved with the writing of the considered articles.
Thanks to the information published at the end of some articles they were able to contact e receive feedback from 72 data curators and practitioners.
A particular attention was given to the most utilized datasets in the considered articles: Adult, COMPASS and German Credit. For each of these a datasheet and a nutrition label was produced.
From the analysis they were able to produce a summary of their merits and limitations, a very useful taxonomy of domains and tasks involved in algorithmic fairness of the existing resources. At the end, these technique are also a set of best practises for curating novel datasets to achieve the expected anonymization, the users consent, inclusively, labeling and transparency.

During their researches, Fabris et al. (2022) were able to produce the following results:

- Analysis of common fairness datasets. In particular they produced a detailed documentation for the above-mentioned resources.

- Analysis of existing resources. They documented also other resources used in fair ML researches. In particular they produced a defined domain and task annotation in which they are involved. This is achieved by connecting all the spare information available.

- Analysis methods for new resources. This is a collection of best practices that they gathered comparing different approaches. In this way the curation of novel datasets are informed about how they should document the data.

2.2.1 Methodology of the survey

In the survey done by Fabris et al. (2022), all the articles published in the most important and domain specific conferences were considered. To cite some of them we can mention the ACM Conference on Fairness, Accountability and Transparency (FAccT), the AAAI/ACM Conference on Artificial Intelligence, Ethics and Society (AIES) but also proceedings papers coming from Machine Learning and data mining conferencies such as the IEEE/CVF Conference on Computer Vision and Patern Recognition (CVRP), the Conference on Neural Information Processing System (NeurIPS), the International Conference on Machine Learning (ICML), the International Conference on Learning Representations (ICLR), the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD); and at last but not least articles available from Past Network Events and Older Workshops and Events of the FAccT network. All the considered resources are published between 2014 and 2021.

The resources selected by Fabris et al. (2022) were chosen through a two step method. The first step had the objective to select all the works which should be strictly related to algorithmic fairness, in fact they decided to consider only documents where one of the following sub-string are presents in their titles: *fair*, *bias*, *discriminat*, *equal*, *equit*, *disparate parit*. As we can easily notice they are centered around equity-based notions of fairness.
The second one is done by hand by the authors and it consisted in a manual inspection of the selected documents. After this human check, some documents were discarded because the selected string are used in different context so they have different meaning to what they desired.
Moreover, other datasets have been discarded for other reasons, for example because are toy datasets (which means that they are a simulation of a considered real world) or there are not sufficient available information.

Fabris et al. (2022) elaborated a data brief for each of the considered datasets after the selection made previously. From the staring point with more than 500 articles (and we can assume that each article has at least one connected dataset) the resulting point is of 210 datasets.

Bear in mind that data briefs consist not only in a summary of the data but also in the connection between datasets and articles, so we can estimate also the popularity of a given dataset. As described in section 2.2, the most used datasets are Adult, COMPASS and German Credit and it can be also inferred from fig. 2.1.

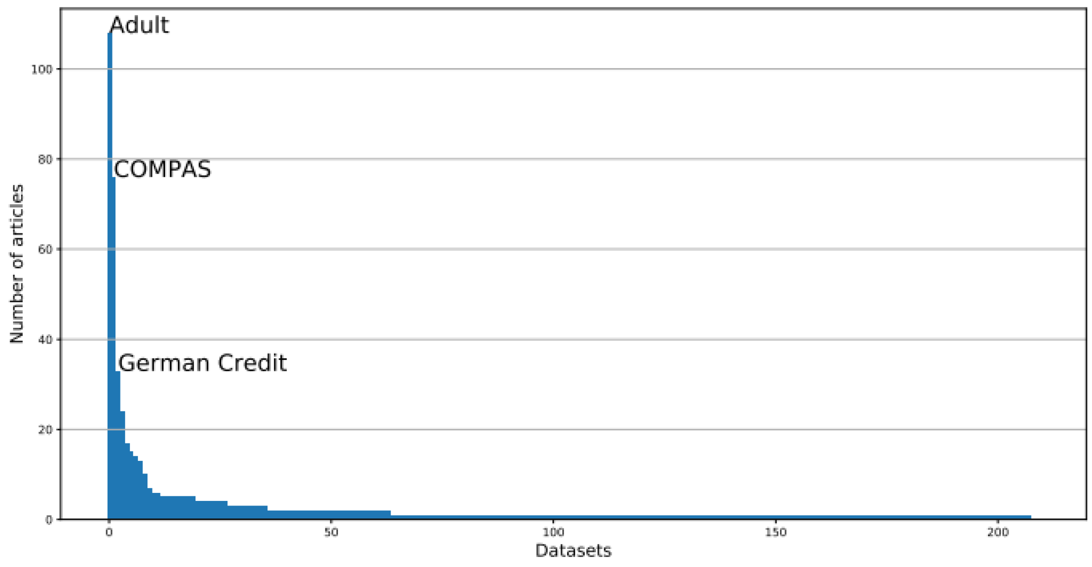


Figure 2.1: Datasets utilization in fariness research

For these in particular, Fabris et al. (2022) produced a detailed documentation following the Gebru et al. (2018) and Holland et al. (2018) guidelines; the information are retrieved using search engines (for academic publications) or thanks to previous references related to the datasets.

2.3 Key data fields

This section presents the three aspects that describe the resources. In particular in subsection 2.3.1 there are the possible domains related to fairness dataset, in subsection 2.3.2 there is the taxonomy of fairness task proposed by Fabris et al. (2022). In conclusion, they analyzed how these datasets are employed in fairness research.

2.3.1 Data domain

From figure fig. 2.2 we can see, with regards to the considered datasets, the macro-domains distribution in fairness research. We can notice that the sum of these domains exceeds the number of considered datasets but the reason is that a dataset can be associated with multiple domains.

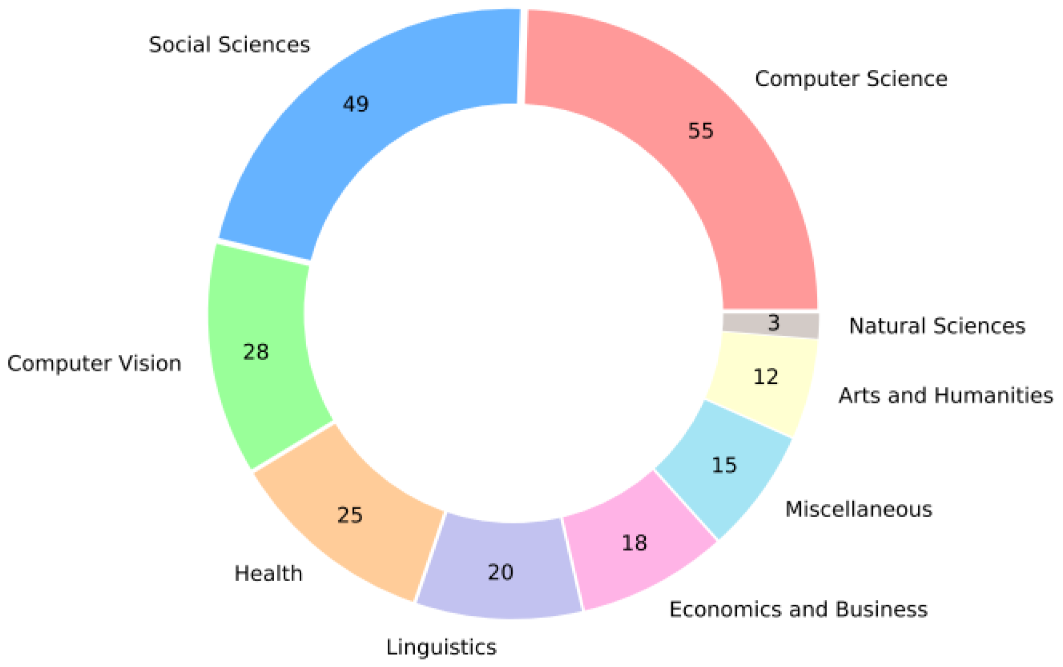


Figure 2.2: Datasets domains in fairness research

I report here a brief description of the fig. 2.2 domains.

Computer Science: we can consider that from this macro-domain are derived *information systems, social media, library and information sciences, computer networks, and signal processing.* They include various types of resources belonging to search engines such as text, images but also worker profiles and et cetera.

Social Sciences: this is a large macro-domain that includes *law, education, social networks, demography, social work, political science, transportation, sociology* and *urban studies.*

Computer Vision: this is a more recent argument and it is mostly related with artificial intelligence. In this field fairness is associated with learned representations and equality of performances across classes. To better explain, it is intended as "the robustness of classifiers across different sub-populations, without much regard for downstream benefits or harms to these population".

Health: this is a macro-domain which includes medicine, psychology and pharmacology and it can be subdivided in a big number of sub-domains: *public health, cardiology, endocrinology, health policy* but also more specific fields like *radiology, and dermatology, critical care medicine, neurology, pediatrics, sleep medicine, nephrology,* and *applied psychology.*
From the connected datasets can be also extracted data from multiple medical centers and this is helpful to study problems of automated diagnosis.

Economics and Business: this macro-domain is composed by dataset in *economics, finance* and *marketing.*

Linguistics: as I have already mentioned before, it consists of textual resources but also by data derived from social media. Many datasets used in algorithmic fairness articles are associated to the domain of linguistics and Natural Language Processing (NLP).

Miscellaneous: this is a macro-domain containing all the domains that can not be categorized into the other domains. In particular it is useful for news domains and resources like the sushi preferences and video games.

Arts and Humanities: in this macro-domain we can include literature datasets, which are strictly connected with NLP tools. Also books, movies, music belong to this macro-domain.

Natural Sciences: in this domain we can find datasets from *biology, biochemistry* and *plant science.*

As we can easily conclude, these datasets represent in practice all the human activities where algorithms are employed. As a reader can image, in particular for some of these domains, the equity and the fairness are an important key feature when an algorithm is developed.

2.3.2 Data task

More and more researchers and practitioners are paying deeper attention in algorithmic fairness, and thanks to this the tasks involved are increasing, such as fair classification, regression and ranking.
In the following part of this sub-section the most common tasks related to the considered datasets are resumed.

Fair classification: its objective is to equalize some keys measure across sub-populations. For example, we can consider the recall, the precision or the accuracy for different racial groups.

Fair regression: regression models has the objective to predict a real-value target, to do that they require that the average loss is balanced across groups. In this context, the individual fairness requires that the losses should be as uniform as possible between all the individuals.

Fair ranking: it consists in a ranking of the population based on a required feature. Here, fairness is applicable to people creating the data to be ranked but also the consumers of the data/products.

Fair matching: it is a task similar to the previous one but it has the objective to match pairs of items on both sides of the market.

Fair risk assessment: it is focused on algorithms that evaluate instances of a dataset considering a predefined type of risk. A relevant difference between this task and classification is the emphasis on real-value instead labels.

Fair representation learning: it concerns the study of features learned by models as intermediate representations for inference tasks.

Fair clustering: it is an unsupervised task aiming at dividing sample into homogeneous groups. In this type of task, fairness can be considered as the fair representation of sub-populations in each cluster.

Fair anomaly detection: it is a task with the objective to identify surprising or anomalous points in a dataset.

Fair districting: it consists in the division of a region to provide electoral districts for political elections.

Fair task assignment: (and truth discovery) are different tasks but they belong to the same area. In fact, they are both concentrating on the subdivision of work and the grouping of answers in crowdsourcing.

Fair spatio-temporal process learning: this task is concentrating on the estimation of models for processes (regarding both time and space).

Fair influence maximization: this task studies and creates models about the diffusion of information throughout networks, in particular using graph covering problems.

Fair resource allocation: it is a task focused on the problem of classification with constraints on the positives items.

Fair data summarization: has the objective of finding portion of dataset which sufficiently describes the entire dataset or the selection of the key features.

Fair graph mining: it produces representations and prevision tasks on graphs. In this case the fairness is intended as the absence of bias in the graph representations or with respect to the inference.

Fair pricing: it is a task with the objective of creating an optimal pricing models. In this case, the fairness is to give possibility of access to services and goods between sensitive groups.

Fair advertising: it is similar to the previous listed task. It is intended for different types of purchasing methods (i.e. auction and bidding) and, as we can imagine, the objective is to reduce discrimination.

Fair routing: the main aim is to find the optimal path between two locations. Here the fairness is related to the attempt of equalizing the driving costs.

Fair entity resolution: this task is concentrating on identify whether multiple items refer to the same entity.

Fair sentiment analysis: it is a sub-task of fair classification, here the text pieces are classified in three options (positive, negative or neutral) based on the expressed sentiment.

Bias in Word Embeddings: this particular task is focused on finding undesired semantics and stereotypes belonging to vectorial representation of words.

Bias in Language Models: this task produce models trained on a big amount of texts, from these can be inferred incorrect correlations and stereotypes.

Fair Machine Translation: as we can understand from the task name, it consists in (automated) translation. In this case the fairness could refer to the fact that sometimes gender-neutral terms of source texts are translated into gendered terms of target texts.

Fair speech-to-text: this task is about transcription of speeches, in this case equity and fairness is reflected in the ability of recognition different demographic regions.

Another aspect to consider is related to the *settings*, which stands for the challenge of avoiding that noise corrupts labels for sensitive attributes. Some of these settings are now proposed:

Rich-subgroup fairness: in this setting fairness properties are necessary for every number of sub-populations.

Noisy fairness: is used to express problems related to missing sensitive attributes or corrupted by noise.

Limited-label fairness: includes all the settings where limited information are available for the target variable.

Robust fairness: is related to issues produced by perturbations to the training set, adversarial attacks and data shifting. This type of setting is often associated to robust machine learning researches.

Dynamical fairness: is related to repeated decisions in changing environments. A possible mutation events can be done by the algorithm itself.

Preference-based fairness: is a work informed by the presence of stakeholders.

Multi-stage fairness: in this type of settings the decision are made by coexistent decision makers in a decision-making process.

Fair few-shot learning: this settings is referred to Machine Learning, in particular to create fair ML solutions when there are not many data samples.

Fair private learning: is related to the privacy-preserving mechanisms and fairness constraints. Here an important consideration is that we would like to produce fair machine learning models without loosing information about individuals of training set.

This list is composed by the main settings with a brief description but I can cite also other settings, for example: *fair federated learning, fair incremental learning, fair active learning* and *fair selective classification.*

2.3.3 Sensitive features

Another important field presents in our datasets is the *sensitive features.*
It has a central position because the features belonging this field can not be used by algorithms to make decisions. This is obviously connected to fairness

and equity, in fact the features are for example *race*, *gender/sex*, *age*, *nationality* and so on.

2.4 Dataset Curation

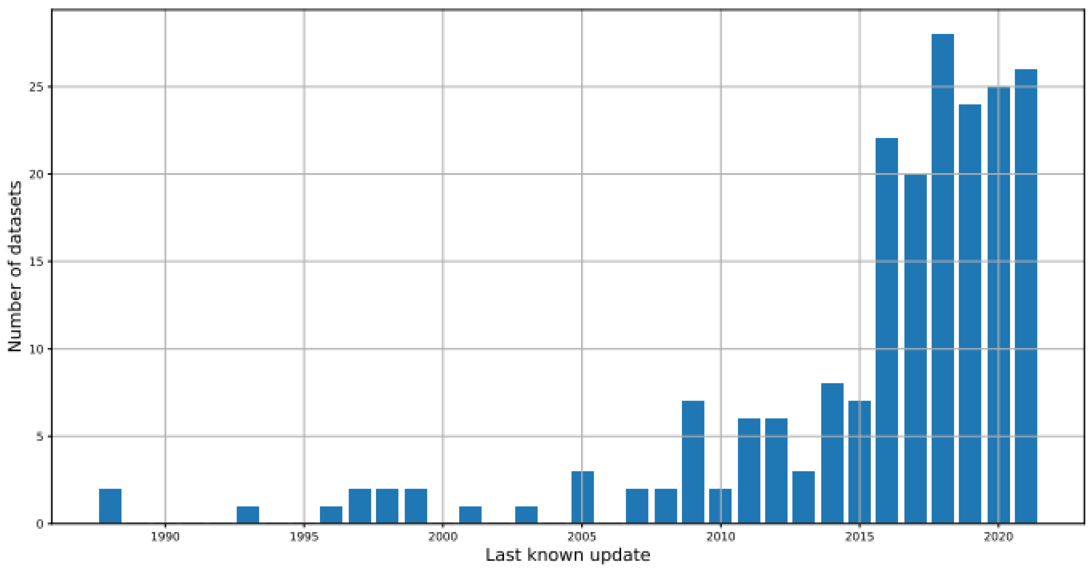


Figure 2.3: Fairness dataset utilization

After all this considerations, it is important to resume the best practices for dataset curation proposed by Fabris et al. (2022). Their work was extremely detailed, in fact they examined different perspectives for example the re-identification, the consent, the inclusivity, sensitive attribute labeling and transparency.
The goal of their work is to let know as many researchers as possible the best practices for datasets curation. As we can see from fig. 2.3, the number of datasets used in algorithmic fairness is increasing, in particular since 2015.

2.5 Data brief structure

To conclude the discussion about the work made by Fabris et al. (2022), we now explain which are the results of their activity. In particular the *data briefs*.
They are composed by the below fields, it is important to describe them because

they are used to create the data briefs parser (subsection 4.2.1) and the back-end (section 5.3). The front-end takes advantage of this field, and for homogeneity they are usually reported to the front-end, but in some cases the variables name has been reduced for simplicity.

Description: this is a text field where the data creator can insert the objective of the data. Moreover, we can also find a summary of the features available in the dataset and how the attributes are annotated. Finally, there could be the envisioned ML task, if any.

Affiliation of creators: in general it is retrieved from the documents (reports, articles but also Web pages) which describe the resource.

Domain: it is an important field and it describes in which sector the data is used (e.g., computer vision for ImageNet).

Tasks in fairness literature: it stands for the datasets employed for tasks.

Data spec: it represents the format in which the data are structured (e.g., text, image, tabular).

Sample size: it is the dataset cardinality.

Year: it represents the last known update of the dataset.

Sensitive features: Sensitive attributes in the dataset, see subsection 2.3.3 for more details.

Link: it is a Web page link where the resources can be downloaded or accessed.

Further information: it is a reference to documents or Web pages which describe the dataset.

2.6 Contribution of this work

Before entering into the modeling and development part of the thesis, it is interesting to talk about the contribution of this work to Fabris et al. (2022) research activities.

The developed Web application has not produced new discoveries and in-depth analysis but it will certainly improve the researchers and practitioners activity. As previously stated, it is a new research study fields and sometimes the information are fragmented and not homogeneous.

So, the aim of this Web application is to produce data homogeneity and data connection. It will help also the data citation and data founding which are two important concepts because sometime happens that published documents are not connected with the used data.

Another aspect to consider is that who are searching for a dataset for their research or job activity, it can be retrieved through this Web application using the full text search but also the faceted search.

Problem Modeling

Starting from this chapter we enter in the hands-on part of the thesis, which is also connected to the research training.
The project behind this thesis is based on the development of a Web Application about fairness dataset and related information. It was a Full Stack development and for this reason Prof. Silvello, Dott. Alessandro Fabris, Dott. Fabio Giachelle and I were able to decide the best possible solution for each part step-by-step.
In particular in this chapter is analyzed how we defined the best suited model for the problem.

3.1 Model designation

Relational databases are the most used type of model behind the contemporary Web and Software applications. They are well-suited for many applications, where data are stored in two-dimensional tables. Unfortunately, it is not suggested when the data are highly connected. In fact, relational databases are not able to store easily relationships between elements.

3.1.1 Linked Open Data

An important aspect to consider is the possibility to produce Linked Open Data. Recently this type of resources are more and more important both for academic and business world. Open data refers to *"data that can be freely used, re-used and redistributed by anyone - subject only, at most, to the requirement to attribute and sharealike."* (Open Knowledge Foundation, 2009)

This type of data has countless benefits, among the most relevant there are design products and services, that allow to make decisions in algorithmic and improve the quality of the society.
The last improvement that can be done staring from Open Data is represented by Linked Open Data. Linked Open Data has all the benefits of Open Data with the additional possibility to connect, retrieve and identify resources thanks to their URI, which should be globally unique.

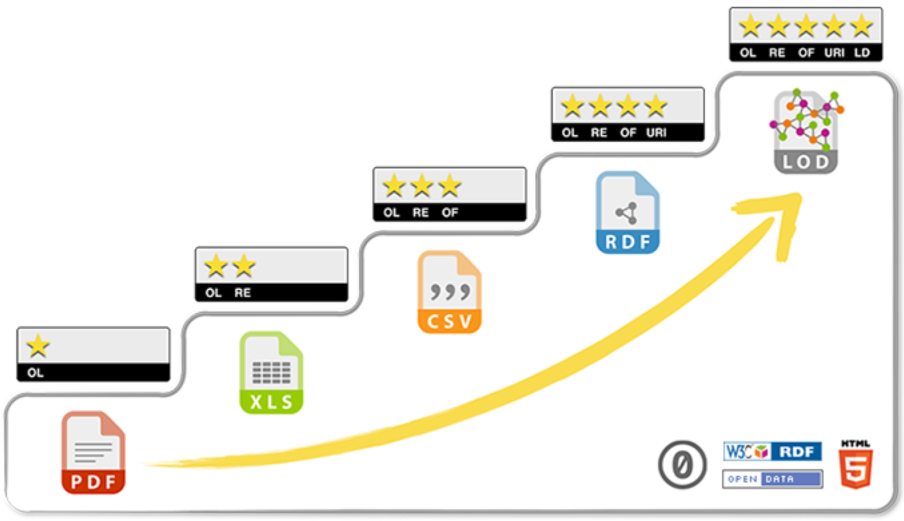


Figure 3.1: 5 star schema

Linked Open Data has a classification, called "*Five Star* of Linked Open Data", which is obviously composed by five levels:

1. the first stage consists of Open Data available on the Web (in whatever format);
2. the second stage consists of machine-readable structured data (also in proprietary format);
3. the third stage is similar to the previous one but it requires non-proprietary formats (i.e. XML, CSV);
4. the forth stage is composed by the characteristics of the third level, plus the requirements of W3C open standards (RDF and SPARQL);

5. the fifth stage has the previous requirements but the data must be linked.

This classification is well represented in the fig. 3.1. Considering the data to be modeled, from the above reasoning we can conclude that a good solution is to used Linked Open Data. We absolutely achieve the forth stage, but also the fifth because, as we will see, there are connection between external resources and they can be also linked.

3.2 Ontology

The term Ontology has a philosophical origin and from this point of view it describes the nature and the organization of reality. In Computer Science it can be considered as syntactic object (it describes concepts via logical theory), as semantic entity and conceptualization (which encodes via semantic structure part of reality).
To describe our "world" we developed a domain ontology, which models our specific domain of interests. To build up our ontology we questioned ourselves through the following questions:

What is the domain that the ontology covers?
: It is obviously concentrating around the concept of algorithmic fairness. In particular we need to models the data briefs and the connected articles.

What is the employment of the ontology?
: The ontology will model datasets and all the connected information, i.e. the datasets' creators and publisher, the connected domains and task, every resources (for example articles or Web pages) which are connected to datasets and tasks.

What information should the ontology provide?

- What are the datasets?
- What are the papers?

- What task/domain/paper are connected with a dataset?
- What task/dataset are connected with a paper?

What type of users will use the ontology?

The users of the ontology (and the Web Application) will be researchers and practitioners.

3.2.1 Foundation Ontology

An important aspect to be considered during the ontology development is given by the foundation ontology or ontologies selection. A foundation ontology is a model of objects which are common and used in many domain ontologies.
This is fundamental because it helps developers to create standardize ontologies and in this way they create connection between the standardized components.
Another advantage is related to the simplification of the development process thanks to foundation ontologies and most important defined ontologies in their respective fields. When an ontology is imported, all its properties, attributes and classes are imported.

The project is based on algorithmic fairness and all the connected resources, so we need to describe datasets, tasks, domains and articles.
After some researches, where we consider the possible ontologies in this fields, we decided to choose *Data Catalog Vocabulary* (Archer, 2014), and *FRBR-aligned Bibliographic Ontology* (Peroni and Shotton, 2012).
Moreover we have also take advantage of *Functional Requirements for Bibliographic Record* (Ciccarese and Peroni, 2018) and *DCMI Metadata Terms* (DCMI Usage Board, 2006). I now briefly describe what we used from each imported ontology to give the opportunity of understanding all the following explanations.

Data Catalog Vocabulary, also known as **DCAT**, is a RDF vocabulary designed to help interoperability between data catalogs. It can be used by a publisher to represent datasets and data services.

From this vocabulary we have used the `Dataset` class which represents a collection of data. It can describe very well the data briefs. Moreover, there are a big number of object and data properties already defined for the `Dataset` class, we will see them more in detail later.

FRBR-aligned Bibliographic Ontology, called **FaBiO** for short, is an ontology designed for describing resources (published or designed to describe publishable documents) on the Semantic Web. These entities contain references to bibliographic resources.
This ontology, in particular, had a primary role into the project development. We imported the following classes:

- `Work`: it represents works that are published or that can be published, where are present bibliographic references or can be referenced. It is a sub-class of FRBR work.
- `Expression`: it is a sub-class of FRBR expression. It represents expressions of fabio:work. It is useful to specify that the following FaBiO classes are defined as sub-class of the `Expression` class.
- `Web page`: it represents a Web resource, generally identified with a URI and accessible through a Web browser.
- `Article`: it represents a writing on a specific topic, in general published in a periodical publication.
- `Journal`: it represents the periodical where research papers are usually published.
- `Book`: it represents a document published in a single volume or in a finite number of volumes. It is commonly identified by the ISBN (International Standard Book Number).
- `Ph.D. symposium paper`: it represents a document presented or published during a conference dedicated to PhD students, where they can presents they researches.
- `Proceedings Paper`: it represents a paper published, in general, within an academic proceedings volume where are reported academic researches.

- `Technical Report`: it represents a technical document.

Functional Requirements for Bibliographic Record, called **FRBR** for short.

It is proposed by the International Federation of Library Association (IFLA) and it is a model to describe every type of resource (physical or digital) and its evolution.
We have used some classes from FRBR, which in some cases are directly imported into the FaBiO ontology by its creators. In particular we have considered:

- `Event`: it represents an occurrence (or an action).
- `Corporate Body`: it represents an organization or people acting as a unit. It resumes different type of group including also the occasional once.

All the object and data properties related to the above mentioned classes will be described later.

3.2.2 New designed ontology

An important part related to the model designation regards the comprehension of which are the necessary parts to be developed, in addition to imported resources cited in Foundation Ontology section.
The documentation about the ontology is available at the following *URL*:
https://fairnessdatasets.dei.unipd.it/schema/

After some drafts, we developed the schema in fig. 3.2. As the reader can see, there are the above mentioned classes and also their related properties. The `DCAT:Dataset` class has already the most important ones, but we had the necessity to add some data properties.
The first one is the `dataSpec` which represents the data structure, i.e. the way (structured or not) used by the creators to store the data inside the dataset. An example could be a text form, a tabular data form, image form, etc.
The second one is the `sampleSize` which is a literal field that represents the

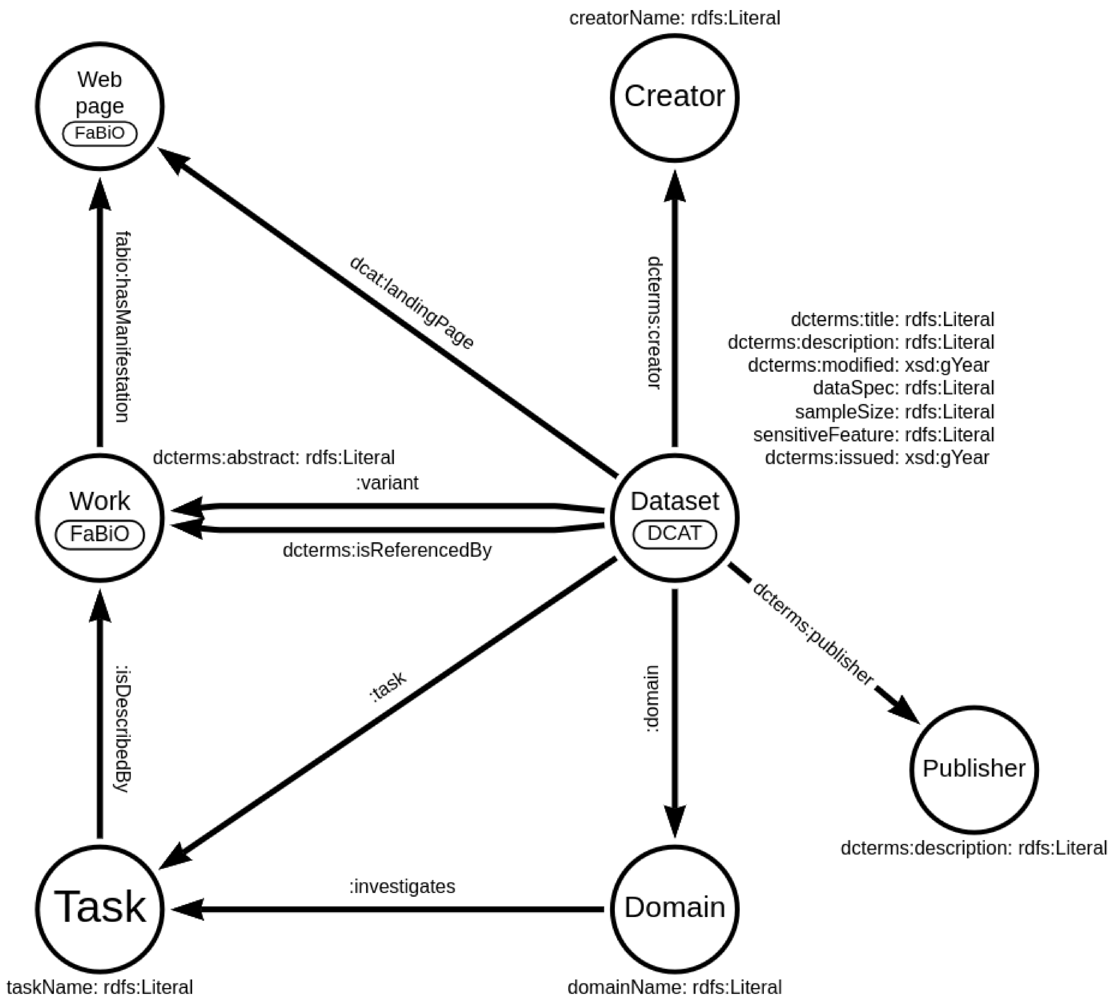


Figure 3.2: Ontology with focus on Datasets

number of samples present in the dataset, we decided to store this information as literal because we noticed that not all the datasets have a simple number but sometimes it may be composed of more complex data (for example "$\sim$ 1K defendants from COMPAS and $\sim$ 400 crowd-sourced labellers. Each defendant is judged by 20 different labellers.").

The last added property is the `sensitiveFeature` which represents the features inside the dataset that should not be considered to make algorithmic decisions, as already discussed in subsection 2.3.3.

In addition, if a landing page related to the dataset is available, then an individual of `fabio:webpage` is instantiated to refer to it and it is connected using the property `dcat:landingPage`, which is provided by DCAT.

The `Dataset` class has the `dcterms:creator` object property which has range in `foaf:Agent`, so we defined a new class called `Creator` as sub-class of `foaf:Agent`. It has only one data property called `creatorName` and it stores the affiliation of the creator or the creator name if the first option is not available.
Similarly to the Creator class, we defined also the `Publisher` class which represents the entity responsible for making the resource available, i.e. published. It may refer to the same entity of the Creator class or a different one and it has also a corresponding property to store the publisher name.
Two classes, of fundamental importance for this ontology, are the `Domain` and the `Task` which are connected with `Dataset` using respectively the object properties `domain` and `task`.
The `Domain` class represents the domain of interest of the resource, in this case the dataset. The `Task`, instead represents in which field or task the dataset can be used. They both have only one data property which is used to store the name of the domain or task.
In addition, the Task class has an object property called `isDescribedBy` which connect the Task with a `fabio:Work`. In this way we are able to model all the resources related with a Task, in particular it can be described by a Web resource (i.e. Web page using the `fabio:hasManifestation` property) or a document (we will see in the following part how a document is connected thought the Work class).
The last thing, related with the part of schema shown in fig. 3.2, is the way in which we refer to the resources used to describe a dataset and its possible variants. From DCAT we used the `dcterms:isReferencedBy` which is has been utilized to connect the datasets and their related resources. Moreover, we defined the `variant` object property which connect a dataset to a work, like the `dcterms:isReferencedBy`, but with a different meaning. In fact, it refers to resources that describe possible variants of a dataset.
At the beginning of the ontology development we thought to store the possible variants of a dataset using a data property but notice that usually they are represented by documents or Web pages, the schema in fig. 3.2 fits better the

concept.

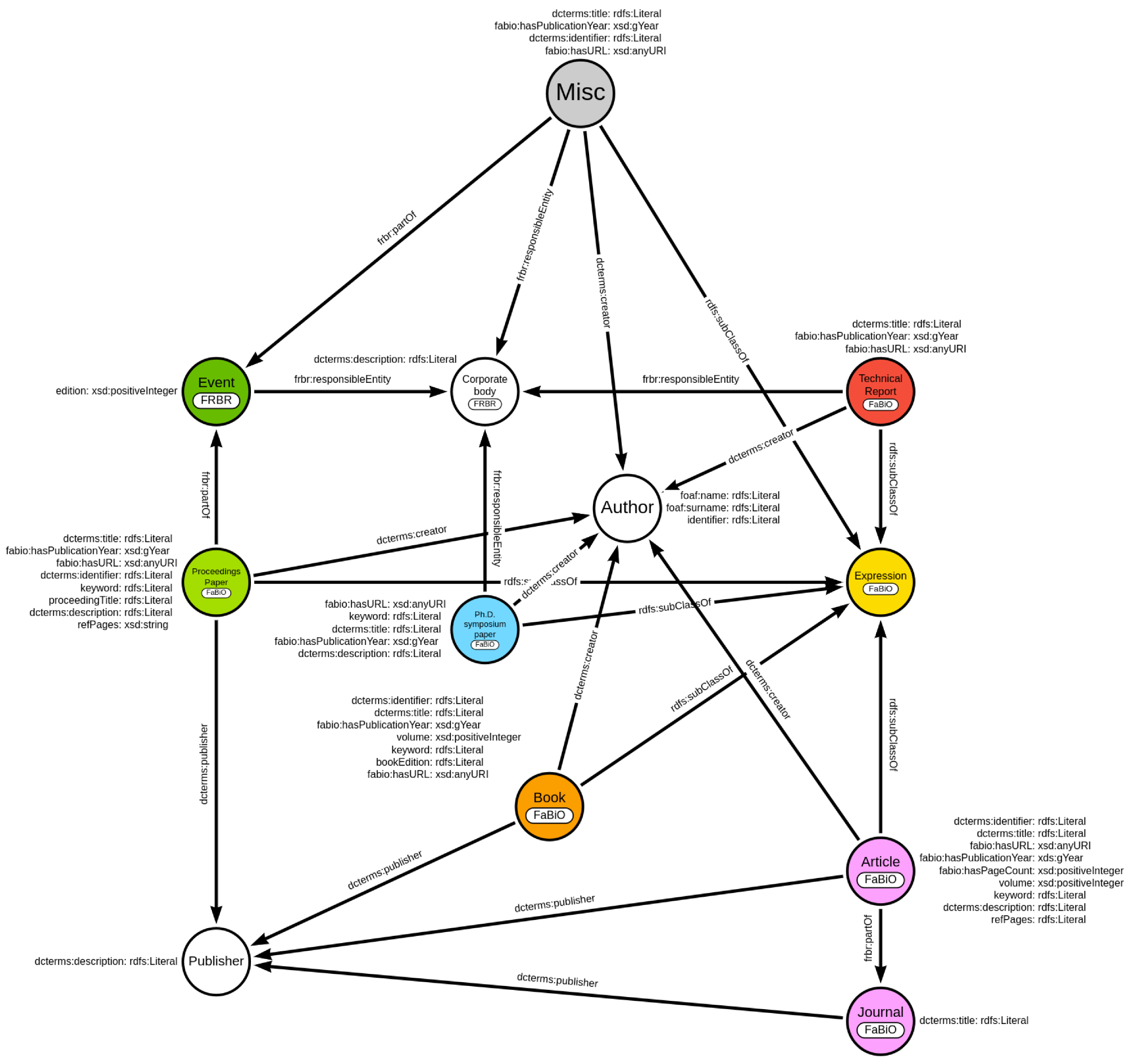

dcterms:title: rdfs:Literal
fabio:hasPublicationYear: xsd:gYear
dcterms:identifier: rdfs:Literal
fabio:hasURL: xsd:anyURI
Misc
frbr:partOf
frbr:responsibleEntity
dcterms:creator
rdfs:subClassOf
dcterms:title: rdfs:Literal
fabio:hasPublicationYear: xsd:gYear
fabio:hasURL: xsd:anyURI
edition: xsd:positiveInteger
Event
FRBR
dcterms:description: rdfs:Literal
frbr:responsibleEntity
Corporate body
FRBR
frbr:responsibleEntity
Technical Report
FaBiO
dcterms:creator
rdfs:subClassOf
frbr:partOf
frbr:responsibleEntity
Author
foaf:name: rdfs:Literal
foaf:surname: rdfs:Literal
identifier: rdfs:Literal
dcterms:creator
dcterms:title: rdfs:Literal
fabio:hasPublicationYear: xsd:gYear
fabio:hasURL: xsd:anyURI
dcterms:identifier: rdfs:Literal
keyword: rdfs:Literal
proceedingTitle: rdfs:Literal
dcterms:description: rdfs:Literal
refPages: xsd:string
Proceedings Paper
FaBiO
rdfs:subClassOf
dcterms:creator
Expression
FaBiO
fabio:hasURL: xsd:anyURI
keyword: rdfs:Literal
dcterms:title: rdfs:Literal
fabio:hasPublicationYear: xsd:gYear
dcterms:description: rdfs:Literal
Ph.D. symposium paper
FaBiO
rdfs:subClassOf
dcterms:identifier: rdfs:Literal
dcterms:title: rdfs:Literal
fabio:hasPublicationYear: xsd:gYear
volume: xsd:positiveInteger
keyword: rdfs:Literal
bookEdition: rdfs:Literal
fabio:hasURL: xsd:anyURI
dcterms:creator
rdfs:subClassOf
dcterms:creator
rdfs:subClassOf
dcterms:publisher
Book
FaBiO
dcterms:publisher
dcterms:identifier: rdfs:Literal
dcterms:title: rdfs:Literal
fabio:hasURL: xsd:anyURI
fabio:hasPublicationYear: xds:gYear
fabio:hasPageCount: xsd:positiveInteger
volume: xsd:positiveInteger
keyword: rdfs:Literal
dcterms:description: rdfs:Literal
refPages: rdfs:Literal
Article
FaBiO
dcterms:publisher
frbr:partOf
dcterms:description: rdfs:Literal
Publisher
dcterms:publisher
Journal
FaBiO
dcterms:title: rdfs:Literal

Figure 3.3: Ontology with focus on papers

Another part of the modelled ontology is visible in fig. 3.3, where we modeled all the classes about the documents connected with a dataset or a task.
As the reader can see, the "starting" point is given by the FaBiO class `Expression` whose definition is available in subsection 3.2.1. In fact, all the published documents can be categorized in a sub-class of `fabio:Expression`.
In this schema there is also the `Publisher` class which is the same of the fig. 3.2 and so we have already talked about it. It is useful to specify that the documents (and so the corresponding classes) can have a relation with the Publisher are

the proceedings papers, the books, the articles and the journals.
In addition to these classes, we can observe two new defined classes: the class `Author` and the class `Misc`.
The first one is intuitive, it represents the author entity of a given resource. It is defined as sub-class of `foaf:Agent`, in this way it is a valid range of the object property `dcterms:creator` of the document resources. We defined one data property related to `Author` class, the `identifier` which is the identifier of a given author, if available. We used also `foaf:name` and `foaf:surname` which obviously represent the name and surname of the considered author.
The reader can observe that every document represented by the classes in fig. 3.3 has an object property that creates a relationship between the document itself and its author or authors.

The second defined class is the `Misc`. It represents all the documents which can not categorized into the FaBiO classes reported in the schema in fig. 3.3. It has been defined in the way that we can use the following data properties: `dcterms:title`, `fabio:hasPublicationYear`, `dcterms:identifier` and `fabio:hasURL`.
Moreover, some `Misc` documents have an affiliation to an organization, so we are able to describe this concept creating a relation with `fabio:CorporateBody` using the property `frbr:responsibleEntity`; others were being published during an event so these once are connected with `frbr:Event` using the `frbr:partOf` property.

We were able to import and take advantage of most of the data properties already present into the Foundation Ontology but some fields of the input data had not a corresponding one. So we had the necessity to define new data properties to store these specific fields:

keyword it is a literal datatype and represents a keyword associated with a resource;

proceedingTitle it is a literal datatype and represents the title of the proceeding where the proceedings paper is published. It is a property related only to proceedings paper;

refPages it is a literal datatype and specifies the pages where we can find the proceedings paper inside a proceeding or an article in a journal;

volume it is a positive integer datatype and it represents the volume of a collection of books or article;

bookEdition it is a literal datatype and it represents the edition of a book resource;

edition it is a positive integer datatype and represents the edition of an event.

To conclude the description about the schema in fig. 3.3, we need to specify that only `fabio:Article` has a relation with `fabio:Journal`, in fact an article can be published inside a Journal.
Moreover, the documents represented by the classes `fabio:proceedingsPaper`, `fabio:phdSymphosiumPaper`, `fabio:technicalReport` and `Misc` can have a responsible entity which is represented by `frbr:corporateBody`.
The last concept is related to the fact that proceedings papers and, as already mentioned, misc documents can be presented during an event (schematized in our ontology by the `frbr:Event`). An Event class has only one property which represents its edition and it has always, or almost, a responsible entity (described by `frbr:corporateBody`).

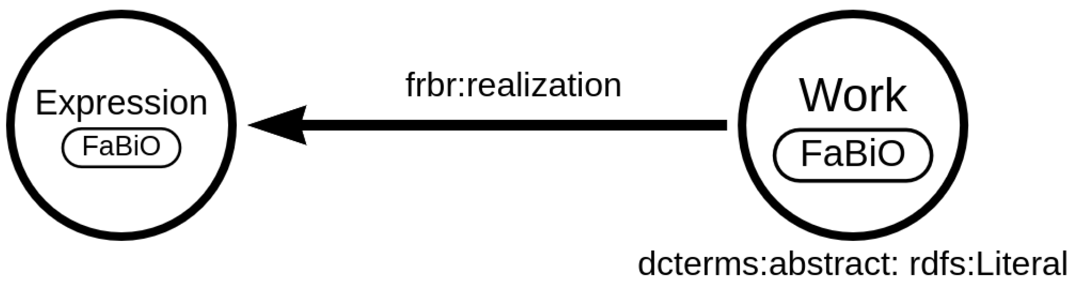


Figure 3.4: Ontology relation between dataset and documens

We previously do not talk about how the datasets are connected with their related resources. In fig. 3.4 we can observe two classes already described and

illustrated in fig. 3.2 and fig. 3.3. From `FaBiO` ontology we can conclude that they are connected using the `frbr:realization` which means that a work is realized through an expression.
In practice we can think at the individuals of `fabio:work` as the results of an agent effort and the individuals of `fabio:expression` represent the "real" objects which are produced from that work and they could be associated to an identifier. For example the pre-print and the final version of a paper are two expression of the same work.

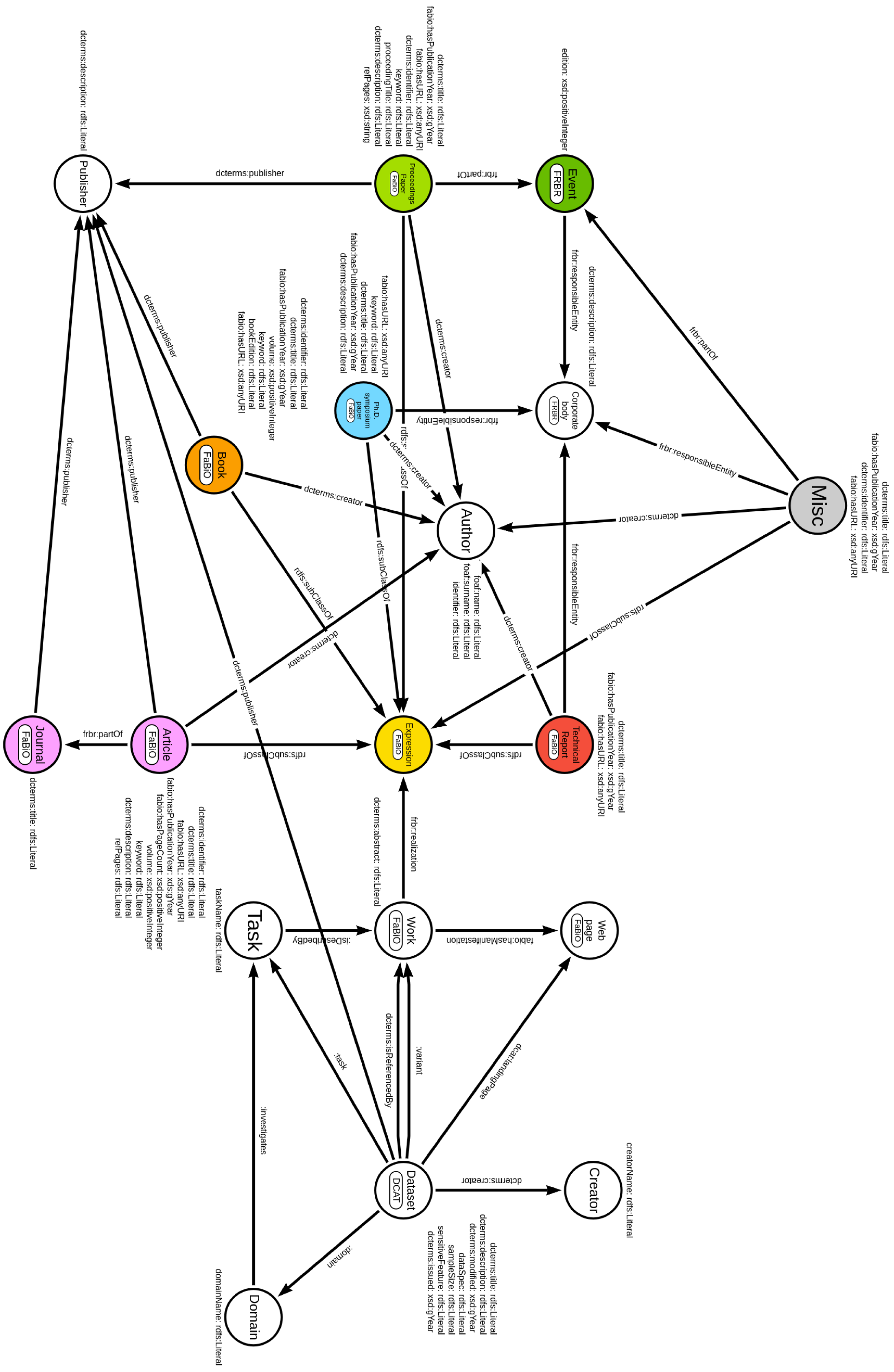

Publisher
dcterms:description: rdfs:Literal
Proceedings Paper
FaBiO
dcterms:title: rdfs:Literal
fabio:hasPublicationYear: xsd:gYear
fabio:hasURL: xsd:anyURI
dcterms:identifier: rdfs:Literal
keyword: rdfs:Literal
proceedingTitle: rdfs:Literal
dcterms:description: rdfs:Literal
refPages: xsd:string
Event
FRBR
edition: xsd:positiveInteger
dcterms:publisher
frbr:partOf
frbr:responsibleEntity
dcterms:description: rdfs:Literal
Corporate body
FRBR
dcterms:creator
Ph.D. symposium paper
FaBiO
fabio:hasURL: xsd:anyURI
keyword: rdfs:Literal
dcterms:title: rdfs:Literal
fabio:hasPublicationYear: xsd:gYear
dcterms:description: rdfs:Literal
frbr:responsibleEntity
Book
FaBiO
dcterms:identifier: rdfs:Literal
dcterms:title: rdfs:Literal
fabio:hasPublicationYear: xsd:gYear
volume: xsd:positiveInteger
keyword: rdfs:Literal
bookEdition: rdfs:Literal
fabio:hasURL: xsd:anyURI
dcterms:publisher
dcterms:creator
rdfs:subClassOf
Author
foaf:name: rdfs:Literal
foaf:surname: rdfs:Literal
identifier: rdfs:Literal
Misc
dcterms:title: rdfs:Literal
fabio:hasPublicationYear: xsd:gYear
dcterms:identifier: rdfs:Literal
fabio:hasURL: xsd:anyURI
frbr:partOf
frbr:responsibleEntity
dcterms:creator
rdfs:subClassOf
Journal
FaBiO
dcterms:title: rdfs:Literal
Article
FaBiO
dcterms:identifier: rdfs:Literal
dcterms:title: rdfs:Literal
fabio:hasURL: xsd:anyURI
fabio:hasPublicationYear: xds:gYear
fabio:hasPageCount: xsd:positiveInteger
volume: xsd:positiveInteger
keyword: rdfs:Literal
dcterms:description: rdfs:Literal
refPages: rdfs:Literal
frbr:partOf
rdfs:subClassOf
Expression
FaBiO
Technical Report
FaBiO
dcterms:title: rdfs:Literal
fabio:hasPublicationYear: xsd:gYear
fabio:hasURL: xsd:anyURI
dcterms:creator
frbr:responsibleEntity
dcterms:publisher
frbr:realization
dcterms:abstract: rdfs:Literal
Task
taskName: rdfs:Literal
:isDescribedBy
Work
FaBiO
fabio:hasManifestation
Web page
FaBiO
:task
dcterms:isReferencedBy
:variant
dcat:landingPage
:investigates
Dataset
DCAT
dcterms:creator
Creator
creatorName: rdfs:Literal
dcterms:title: rdfs:Literal
dcterms:description: rdfs:Literal
dcterms:modified: xsd:gYear
dataSpec: rdfs:Literal
sampleSize: rdfs:Literal
sensitiveFeature: rdfs:Literal
dcterms:issued: xsd:gYear
:domain
Domain
domainName: rdfs:Literal

Figure 3.5: Full Ontolgy Schema

We can see the complete schema representing the ontology in fig. 3.5.
So, to resume how it should be interpreted, we see that the datasets are central and they have a creator and a publisher (they could refer to the same entity or not). A dataset has a corresponding domain or domains of interest and can be applied to zero or more tasks. Every dataset generally has a reference document which describes the dataset itself and sometimes some of its variants can occur. Moreover, a task can be described by resources like documents or Web pages. The other parts of the schema models every type of considered document.

During the front-end development we noticed that there was a problem: we were not able to retrieve the documents related to a task and a specific dataset. In practice, if we tried to retrieve the document describing a task we was not able to identify which of these documents are used in that task with a dataset or another one.
The easiest solution we found was to introduce *blank nodes*, which have the objective of drawing the schema without the *URIref* for these particular nodes. They are connected with a dataset, a task and directly with the expressions describing that task. In this way a task is described by all the documents connected with that task but we are also able to identify the specific documents if we refer to a task when is used by a dataset.
The solution is depicted in fig. 3.6 and the final ontology schema is available in fig. 3.7.

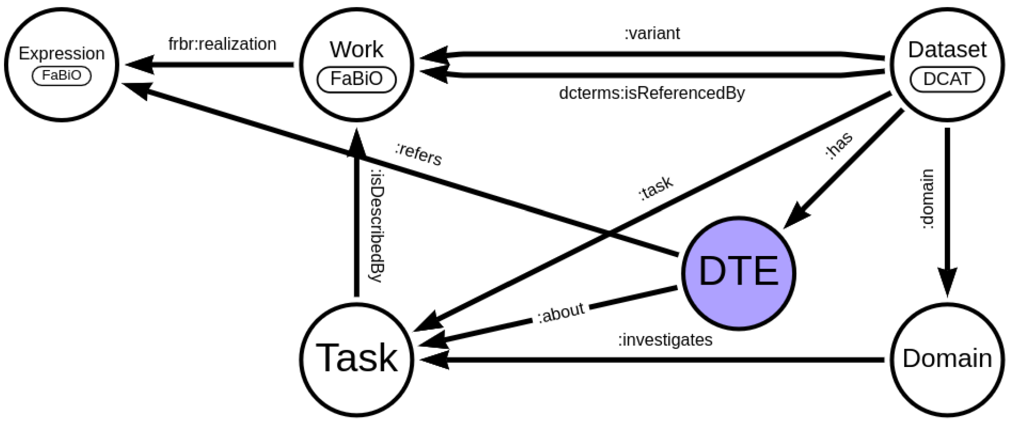


Figure 3.6: New blank node

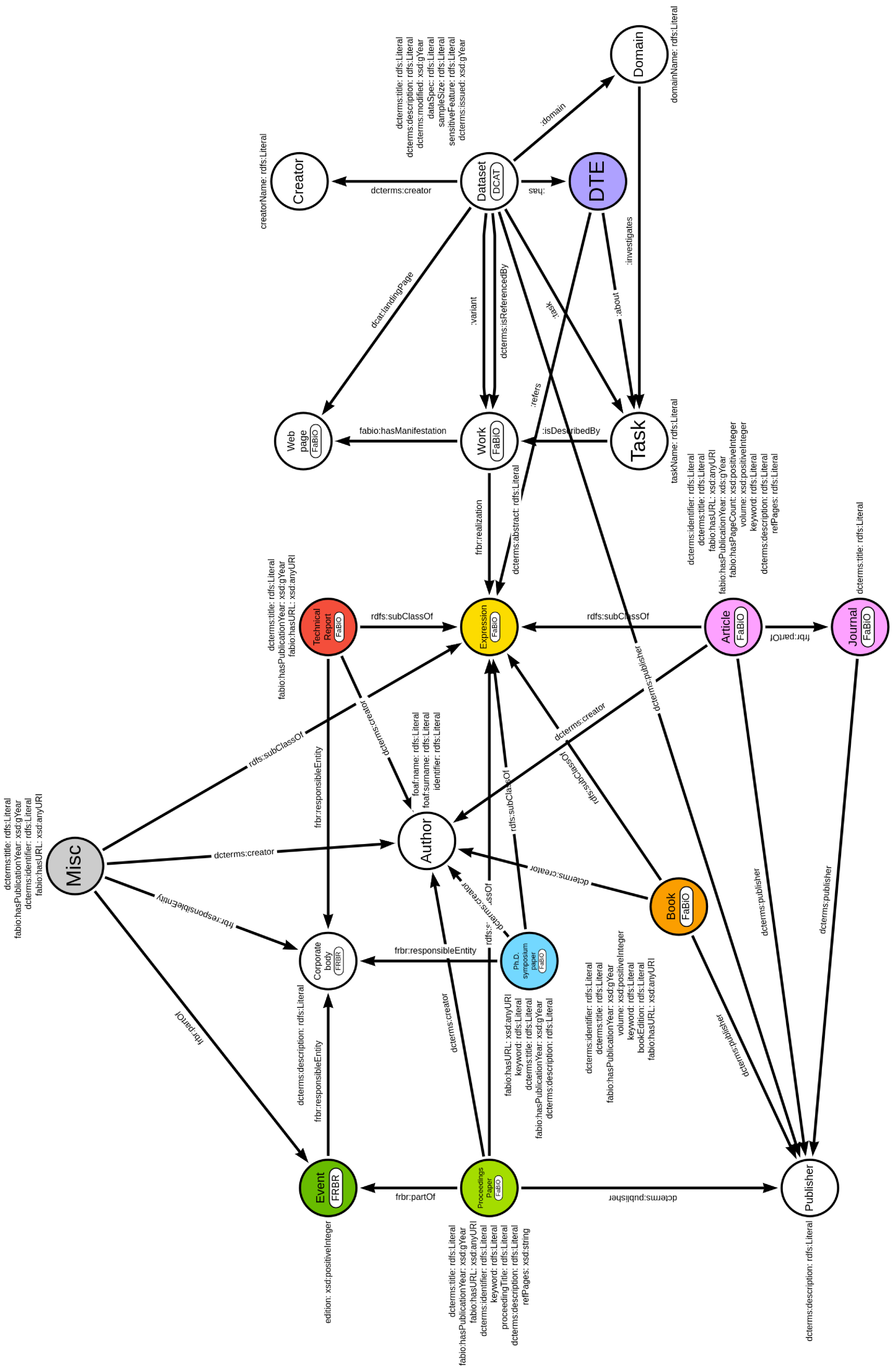
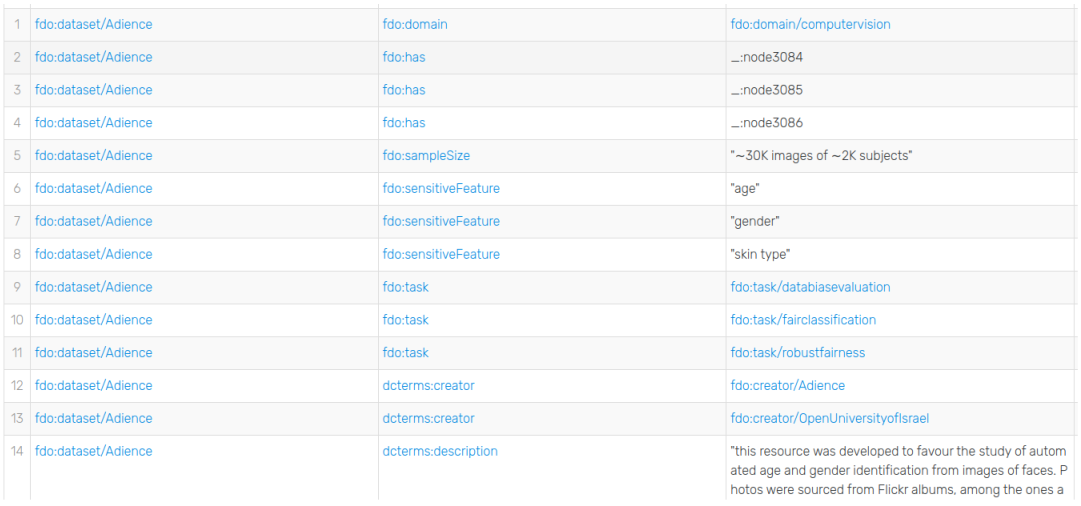


Figure 3.7: Final ontology schema

3.2.3 Instance example

In this subsection we will see a practical example of a dataset entity and its related individuals available into the database. In this way the reader will understand easily the relation between the individuals and also how the blank nodes are used.

For this example we consider the dataset *Adience* (OUI and Adience, 2014) where we can find some interesting points.



1	fdo:dataset/Adience	fdo:domain	fdo:domain/computervision
2	fdo:dataset/Adience	fdo:has	_:node3084
3	fdo:dataset/Adience	fdo:has	_:node3085
4	fdo:dataset/Adience	fdo:has	_:node3086
5	fdo:dataset/Adience	fdo:sampleSize	"~30K images of ~2K subjects"
6	fdo:dataset/Adience	fdo:sensitiveFeature	"age"
7	fdo:dataset/Adience	fdo:sensitiveFeature	"gender"
8	fdo:dataset/Adience	fdo:sensitiveFeature	"skin type"
9	fdo:dataset/Adience	fdo:task	fdo:task/databiasevaluation
10	fdo:dataset/Adience	fdo:task	fdo:task/fairclassification
11	fdo:dataset/Adience	fdo:task	fdo:task/robustfairness
12	fdo:dataset/Adience	dcterms:creator	fdo:creator/Adience
13	fdo:dataset/Adience	dcterms:creator	fdo:creator/OpenUniversityofIsrael
14	fdo:dataset/Adience	dcterms:description	"this resource was developed to favour the study of autom ated age and gender identification from images of faces. P hotos were sourced from Flickr albums, among the ones a

Figure 3.8: Tabular view of Adience example

The above fig. 3.8 is the tabular view of the results for the *Adience* dataset. As we can see, there are data and object properties but also blank nodes.

Instead the fig. 3.9 represents the graph view of the considered dataset. It is more interesting because we can see the relationship between the individual and also we can understand how the model is transposed at the entity level. In particular, we have expanded a document related with the dataset and we have discovered that it is a proceeding paper. It was published during an event organized by the *Proceeding of Machine Learning Research*, in particular in the 2018 edition.

From the graph we can also observe the other resources presented at the same conference and we could also navigate through the database to display other stored resources.

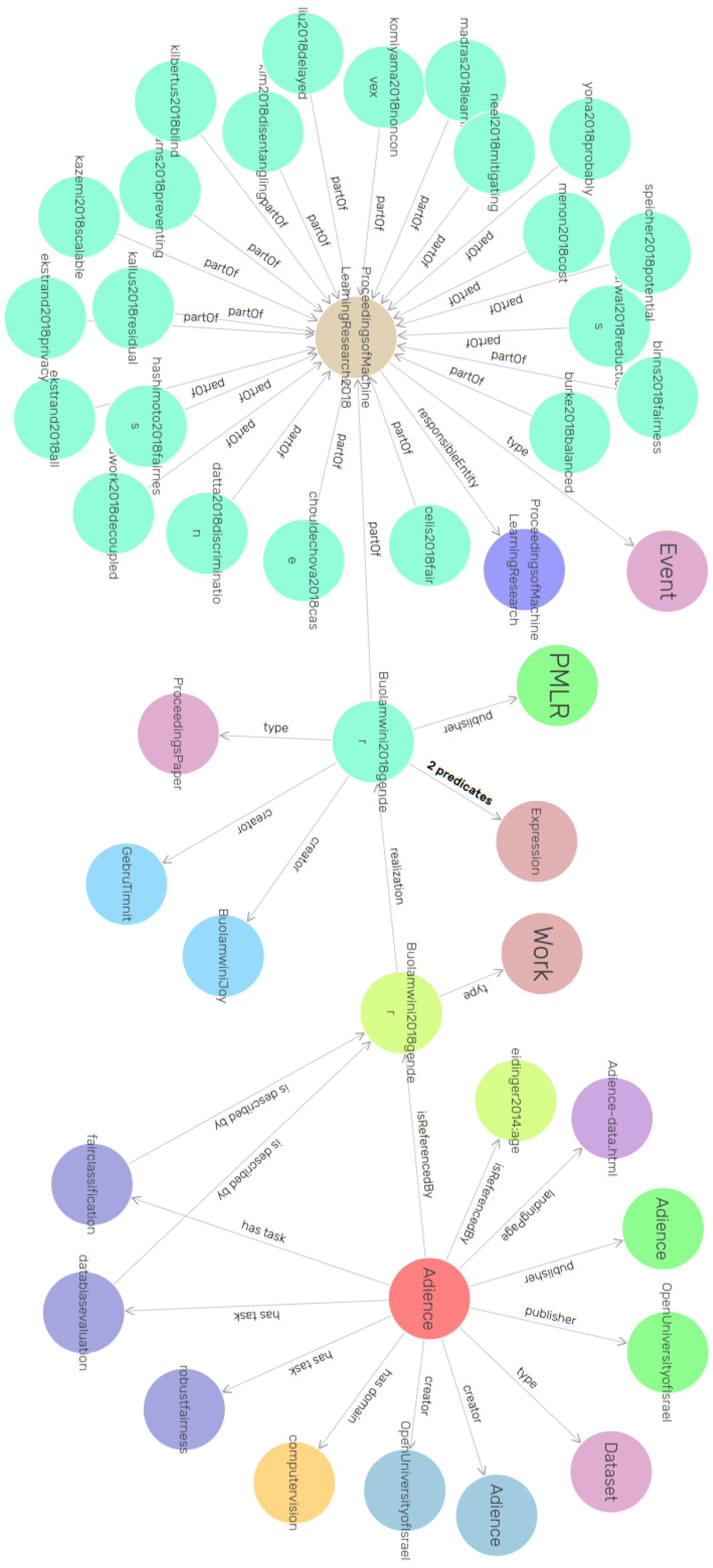


Figure 3.9: Graph view of Adience example

Data parsing

The data parsing chapter has the objective to explain how the input data has been elaborated to produce a serialization and how the RDF data are represented using a textual form.

4.1 Turtle serialization

We decided to use *Turtle* file as output for our serialization. There are multiple advantages of exploiting TTL files instead other type of serialization:

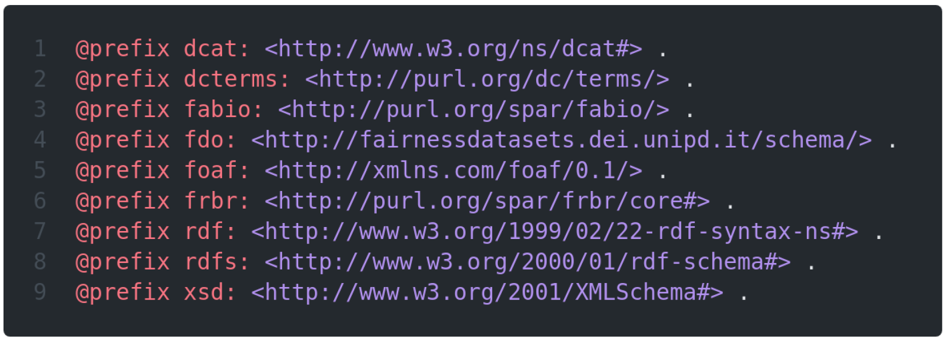
- compared to RDF/XML serialization, it is more efficient;
- compared to JSON-LD, it is more human readable;
- compared to N-Triples, it is more readable and memory saving. In fact N-Triples is considered as the most raw way to store RDF triples because they are stored and represented with unabbreviated URIs.

Finally we can consider the Turtle serialization as the combination of N-Triples and abbreviation given by CURIEs. In fact, it defines the prefixes which must be bound between the local CURIEs and the global URIs, in fig. 4.2 we can see how this operation is done using Python programming language. They are a sort of preamble at the beginning of the Turtle file.
Moreover, there are other abbreviations that can be done using TTL files, for example:

- the `rdf:type` can be abbreviated with `"a"`

- considering that triples are composed by subject predicate and object, if more than one triple share the same subject, then we can use a semicolon (;) at the end of the first triple and it indicates that following query has the same subject
- using the previous consideration, if more than one triple share the same subject and predicate, then we can use a comma (,) at the end of the first triple and it indicates that following query has the same subject and predicate.

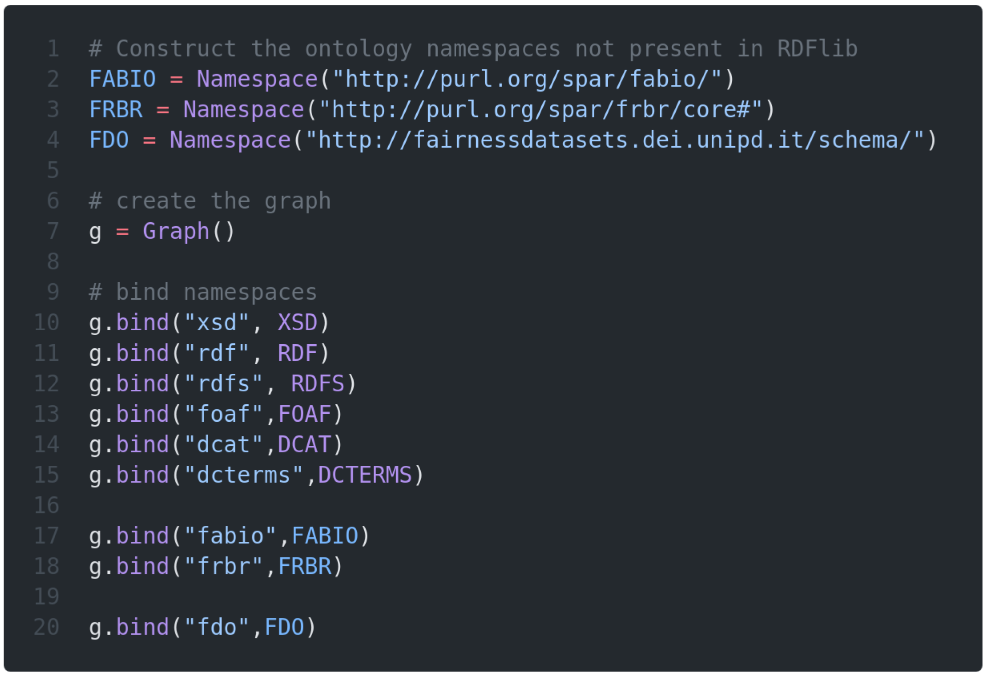
Turtle stands for *Terse RDF Triple Language* and this file format is used to express RDF data. Furthermore, Turtle is a W3C standard.
In practice, a TTL file should contain all the triples representing an entity.



```
@prefix dcat: <http://www.w3.org/ns/dcat#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix fabio: <http://purl.org/spar/fabio/> .
@prefix fdo: <http://fairnessdatasets.dei.unipd.it/schema/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix frbr: <http://purl.org/spar/frbr/core#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
```

Figure 4.1: Turtle prefixes

In fig. 4.1 are reported all the prefixes used into the created Turtle file and in the below fig. 4.2 we can see the necessary Python code to make the "bind operation". There are three lines that define the namespaces not present into the *RDFlib* library then the graph, where the triples will be stored, is created and finally there are some line of codes that bind the namespaces into the graph.



```python
# Construct the ontology namespaces not present in RDFlib
FABIO = Namespace("http://purl.org/spar/fabio/")
FRBR = Namespace("http://purl.org/spar/frbr/core#")
FDO = Namespace("http://fairnessdatasets.dei.unipd.it/schema/")

# create the graph
g = Graph()

# bind namespaces
g.bind("xsd", XSD)
g.bind("rdf", RDF)
g.bind("rdfs", RDFS)
g.bind("foaf",FOAF)
g.bind("dcat",DCAT)
g.bind("dcterms",DCTERMS)

g.bind("fabio",FABIO)
g.bind("frbr",FRBR)

g.bind("fdo",FDO)
```

Figure 4.2: Namespaces bind operation

4.2 Parsers

An important part of the data mapping is given by the LaTeX parsers. In fact the input files were in two formats: a *BIB* file and a LaTeX file.
To develop the code we took advantage of Python versatility.

4.2.1 Datasets parser

Thanks to Dott. Alessandro Fabris, I was able to start from a DataFrame structure because he developed a function which takes as input the data briefs LaTeX file and returns the previous mentioned structure.
In particular the DataFrame has the following columns:

name : the title of the dataset, it is also used to create the ID of the dataset into the database;

label : derives from the "\label{}" LaTeX command. It might be a good identifier but it was rarely available;

description : it is a text field representing the description of the dataset;

affiliation : it represents the affiliation of the creators of the dataset;

domain : it stores the belonging domain of the dataset;

tasks : it represents the tasks where the dataset is applied;

data_spec : it represents how the data are stored inside the dataset;

sample_size : it represents the dimension or dimensions of the dataset;

year : it represents the publication year (or the last update year if available);

sensitive : it represents the sensitive features related to the dataset;

further : it consists of possible further information about the dataset;

variants : it stores possible variants of a dataset.

Obviously these information have been mapped to the ontology model explained in section 3.2.
When the DataFrame has been built, an iterator along all its rows is created and the `serializeDatasetItem` function is called for each row. From its name we can infer that its objective is to serialize a dataset item. Now we will see some implementation details but we will not cover all the development parts.

Algorithm 1: Dataset URI creation

```
Dataset = createURI(FDO, "dataset", strToID(name=row["name"]) )
```

From algorithm 1 we can see the function that creates the URI of a given resource. The `createURI` function has three input parameter and they are used to create the URI, in particular the first one corresponds to the namespace, the second one is the resource type and the third one is the `ID` of the resource. It is separated from the remaining part of the URI by a "/" but it is only a convention that we decided to adopt. In fact, all the resources are created inside our

namespace.
The last thing to notice is the `srtToID` function which takes as input any number of strings and returns the a string without any punctuation, any space or any symbol which should not be inside the ID.
Then, the remaining part of the function is used to serialize the data and object properties connected to the given resource. We can take into consideration the example in algorithm 2 where the triple about the title of a dataset is created. It is an general illustration of how every column of the DataFrame is imported (except for particular cases).

Algorithm 2: Dataset title triple creation

```
if "name" in row then
    name = remove_latex(row["name"])
    g.add(( Dataset, DCTERMS["title"], Literal(name) ))
end
```

It is worth pointing out that we can not assume that every columns is present so we need to check if the columns is available. In general, we used the custom function `remove_latex` which, as the reader can image, removes the LaTeX "grammar" from a string. Finally, the triple is create and added to the graph.

A different problem is related to the *year* information, in fact sometimes the year field is present but it contains *"unknown"* or *"present"*. The solution is easy: if the year field is present and it is different from "unknown" then we add the data property and if the variable contains "present" we substitute the string with the present year. At the moment, as we can see in algorithm 3, we create two identical triple one for the dataset year of creation and one for the dataset modification year but in future implementation they can be separated into two different information when more details are provided.

Algorithm 3: Dataset year triple creation

```
if "year" in row and "unknown" not in row["year"] then
    if "present" in row["year"] then
        year = datetime.date.today().year
        g.add((Dataset, DCTERMS["modified"], Literal(year,
         datatype=XSD.gYear)))
        g.add((Dataset, DCTERMS["issued"], Literal(year,
         datatype=XSD.gYear)))
    else
        g.add((Dataset, DCTERMS["modified"], Literal(row["year"],
         datatype=XSD.gYear)))
        g.add((Dataset, DCTERMS["issued"], Literal(row["year"],
         datatype=XSD.gYear)))
    end
end
```

Another consideration must be done about the sensitive features creation.

Algorithm 4: Sensitive features triple creation

```
if "sensitive" in row then
    if "N/A" not in row["sensitive"] then
        sensitive_list = row["sensitive"].split(",")
        for sensitive in sensitive_list do
            g.add((Dataset, FDO["sensitiveFeature"], Literal(
             remove_latex(sensitive).strip()) ))
        end
    end
end
```

In fact, in this case a single string is provided but we need to split the sensitive features and create a triple for each sensitive feature. This is necessary because

we would like to make filters using this field.
As we can see in algorithm 4, also in this case there is a particular string that must be ignored, the *"N/A"*.
A similar approach is done for the landing page, which can be composed by more than one URL. The only difference is that a URL in LaTeX is wrapped by curly brackets so we developed a function called `clean_latex` which is similar to `remove_latex` but it remove only the LaTeX provided as input parameter.
A different method is done for the further information which produces all the triple about the reference of a dataset. Indeed, in this case we could find different LaTeX elements, for example a URL but also citation like `\cite`, `\citet` and `\citep`.
In algorithm 5 we can see an example of the code developed to create the reference part. It is only resume, in fact as we can observe there are not the cases `\citet` and `\citep` but the reader can extend the implementation to them using same `"if"` code block used for `\cite`. Moreover, the `further_list` is a list produced by splitting the further column, if it is available.
From that algorithm, but also from the ontology designed in subsection 3.2.2, we can understand that a dataset can have different type of references. In fact, the first part of the algorithm considers the option that the reference is provided as Web page and a Web page individual is created but it is connected through a Work node, differently from the dataset URL directly connected using `dcat:landingPage`.
As convention, the URI of a resource is composed by the namespace, the type individual and its name (or ID if available) which is used as ID. Sometimes the ID can not be the name of the resource because it is only a URL, in this case we decided to use an `hash` function to generate the ID and in particular with the `ctypes.c_size_t` function at line 7 of algorithm 5 we generate only positive hashes.
Another considered option is the `\cite`, that is very similar to the previous one, the differences are that the `\cite` is removed instead `\url` but also how the paper URI is generated.

Algorithm 5: Dataset reference triple creation

```
for element in further_list do
    if "\url" in element) then
        url_list = split_latex(element,delimiter="\url")
        for url in url_list do
            further_WebPage = URIRef(clean_latex(url,"\url"))
            g.add((further_WebPage, RDF.type, FABIO["WebPage"]))
            Work = createURI(FDO, "work", str(
              ctypes.c_size_t(hash(clean_latex(url,"\url"))).value))
            g.add((Work, RDF.type, FABIO["Work"]))
            g.add((Dataset, DCTERMS["isReferencedBy"], Work))
            g.add((Work, FABIO["hasManifestation"], further_WebPage))
        end
    else
        if "\cite" in element then
            citation_list = clean_latex(element,"\cite").split(",")
            for citation in citation_list do
                Paper = find_paper_URI(citation.strip(),bib_database)
                if Paper != None then
                    furtherInfo = createURI(FDO,"work",citation.strip())
                    g.add((furtherInfo, RDF.type, FABIO["Work"]))
                    g.add((Dataset, DCTERMS["isReferencedBy"],
                      furtherInfo))
                    g.add((furtherInfo, FRBR["realization"], Paper))
                else
                    print("Further info: paper NOT found! "+citation)
                end
            end
        end
    end
end
```

The `createURI` function is called inside the `find_paper_URI` function. It takes as input the entire bibliography and the resource that should be connected to the dataset. If the resource is found inside the bibliography, then the resource type and ID are used to generate the URI as defined convention, an *exception* is returned otherwise.
The same algorithm structure is used also for the *variants* of a dataset. The only difference originates from the fact that a variants could be only a text description, for example *"C-MNIST: images from MNIST, such that both digits and background are colored."*.
In this case we created an individual of Work class and we stored this text information inside the `dcterms:abstract`. Furthermore, if a URL is available in the entire field then the individual is connected to the Web page instance.

A more complicated reasoning must be done for the domains and the tasks connected with a dataset. Sometime happens that a dataset has not a specified domain or *"N/A"* is the only value presents in the field. In this case no domain individual is created and only the tasks individuals are created.
On the contrary, if at least one domain is available then its individual is created and it will be connected with every available tasks. To help the reader, if there are n tasks and m domains, for each of these m domains we will create an *:investigates* object property for each of the n tasks.
So, as we can see in algorithm 6, we checked if a domain is available and different from *N/A*, then we create the Domain individual and finally we call the `addTasks` function which will generate and connect all the available tasks.
If at least one of the first two checks fails the `addTask` function is called but instead of the `Domain` variable, `None` is passed as input.
This function works in analogous way of the references and variants algorithms, the difference is that in this case we have the possibility to input the domain instance. As previously mentioned, if the domain is not available then `None` is passed and so there will not be created the related object properties.

Algorithm 6: Dataset domain and task triple creation

```
if "domain" in row then
    if "N/A" not in row["domain"] then
        domain_list = row["domain"].split(",")
        for domain in domain_list do
            Domain = createURI(FDO, "domain",
             strToID(domain=domain))
            g.add((Domain, RDF.type, FDO["Domain"]))
            g.add((Domain, FDO["domainName"], Literal(domain)))
            g.add((Dataset, FDO["domain"], Domain))
            if "tasks" in row then
                addTasks(g, Dataset, Domain, row["tasks"], bib_database)
            end
        end
    else
        if "tasks" in row then
            addTasks(g, Dataset, None, row["tasks"], bib_database)
        end
    end
else
    if "tasks" in row then
        addTasks(g, Dataset, None, row["tasks"], bib_database)
    end
end
```

The last things to report about the `serializeDatasetItem` is related to the full text search functionality. We decided to use the full text search provided by *PostgreSQL*, so we had the necessity also to create a sort of very easy relational database.

It is designed only with a single table with two fields: the ID of the resource which is the primary key and a second field to store all the information belonging a dataset. After some researches we have found the `tsvector` data-type which "represents a document in a form optimized for text search" as stated into the documentation. Moreover, there is a similar data-type called `tsquery` which is used when the query is created.

Algorithm 7: Dataset full text search creation

```
id = strToID(name=row["name"])
data = ""
if "year" in row and "present" in row["year"] then
    today = datetime.date.today()
    year = today.year
    row["year"] = str(year)
end
for (index, value) in row.items() do
    if "contact" not in index and "email" not in index and len(value)> 0
     then
        if type(value) == str then
            data = data + " " + value
        else
            for element in value do
                data = data + " " + element
            end
        end
    end
end
insertDatasetIndex(cur, id, data)
```

The algorithm 7 shows how the information of a given dataset are grouped together, the only ignored information are those about the contacts for a dataset. At the end, the `insertDatasetIndex` function is called, that does the insertion into the database. Its full code is available at algorithm 8, where we can see that the IDs are composed by the resource type plus the resource ID. Finally the information are transformed to the `tsvector` data-type using the `to_tsvector` PostgreSQL function.

Algorithm 8: insertDatasetIndex

```
id = "dataset/" + id
cur.execute("""
INSERT INTO "indexSchema".dataset(id, data)
VALUES (%s,to_tsvector('english',%s));
""",(id,query_data))
```

With this algorithm we finish the discussion about the parsing of the dataset resources.

4.2.2 Bibliography parser

On the contrary to the dataset resources, where we had to use a custom parser, for the bibliography information we had the chance to use an existing parser. This powerful module, called `BibTexParser`, provides useful features which helps to import and manipulate the contents of a *BibTex* file.
Before going on in the explanation of all the adopted features, I would like to specify that all the namespaces present in fig. 4.1 and the binding operations in fig. 4.2 are still valid and used for the bibliographic parser.

Coming back to the parser, we started with the line in algorithm 9, which calls firstly the `BibTexParser` function with the following options:

- **common_strings = True** which means that common strings are loaded (for example we can consider month abbreviation);
- **ignore_nonstandard_types = False** which means that non-standard BIBTEX entry are not ignored;
- **homogenize_fields = False** which means that BIBTEX fields name are not sanitized, for example it change *url* to *link*. We do not want any modification because these fields are used to understand when there is for

example a URL or a citation.

Algorithm 9: BIBTEX import

```
bib_database = bibtexparser.bparser.BibTexParser(
 common_strings=True, ignore_nonstandard_types=False,
 homogenize_fields=False).parse_file(bibtex_file)
```

Then, the `BibTexParser` function is concatenated with the `parse_file` function which takes as input the BIBTEX file to be parsed. It returns the bibliographic database generated from the provided input file (the data-type is `BibDatabase`). In addition, we used other two customization provided by the `bibtexparser` module which are the authors split feature and the type customization. The first one converts the author field of the bibliographic resource to a list where every element is in this form: *"Name, Surname"*; the second one instead converts to lower case every resource type to avoid case sensitive miss-mach.
Then, for every entry into the BIBTEX file, is applied the correct serialization function based on the entry type.
To achieve this choice we used the `match-case` structure available starting from Python *3.10*. This functionality is the same of the "classic" *switch-case* in many other languages.
The code reported in algorithm 10 is only a summary of the entire structure of the `switch-case`.
There are some considerations to analyze:

- the `serializeProceeding` function used for the case "inproceedings" has been used also for the entries of type "incollection", "inbook" and "conference". We can do this because the data properties, the object properties and the relation with other classes are the same, moreover they reprsents similar concepts.

- the same approach was done for the "mastersthesis" entry type using the `serializePhD` for the preovuous reason.

- when there is an "article" entry type, we can not use directly the `serializeArticle` because when there is the word *proceeding* inside the field journal (if available), we have to use the `serializeProceeding` serialization function.

Algorithm 10: Serialization function selection

```
switch item["ENTRYTYPE"] do
    case "book" do
        serializeBook(item, g, cur)
    end
    case "article" do
        if "journal" in item and "proceeding" in item["journal"].lower() then
            serializeProceeding(item, g, cur)
        else
            serializeArticle(item, g, cur)
        end
    end
    case "techreport" do
        serializeTechReport(item, g, cur)
    end
    case "phdthesis" do
        serializePhD(item, g, cur)
    end
    case "inproceedings" do
        serializeProceeding(item, g, cur)
    end
    case "misc do
        serializeMisc(item, g, cur)
    end
end
```

Now we will discuss how these serialization functions act. All of these functions start filling the table used for the full text functionality. As for the dataset insertion function we use *PostgreSQL* and we create a similar function to algo-

rithm 8, the two differences are about its name (`insertPaperIndex`) and the ID construction. In fact, it is composed by the prefix "paper/" plus the resource ID.
With regards to the data and object properties, we can start explaining how the `dcterms:identifier` is built. As we can see in algorithm 11, we firstly check if the "doi" field is available and then we create the triple using `dcterms:identifier`. If it is not available then may happen that there is the *arXiv* identifier. In this case we can use the FaBiO sub-property of `dcterms:identifier` which is used to store the *arXiv* identifier.

Algorithm 11: Identifier triple creation

```
if "doi" in item then
    g.add((Article, DCTERMS["identifier"], Literal(item["doi"])))
else
    if "journal" in item and "arxiv" in item["journal"].lower() then
        g.add((Article, FABIO["hasArXivId"], Literal(item["journal"])))
    end
end
```

Related to the *arXiv*, there is also a part focused on the creation of the Journal and Publisher individials. In fact if the identifier is the *arXiv* ID we create an individual related to this entity which is an pre-print and post-print electronic repository where documents are published. In this case arxiv.org will be also the publisher of these resources.
On the contrary, the journal will be the one presents into the journal field and if there is also the publisher field the relative triples will be created.

Other fields are filled directly like the title one, as we can see in algorithm 12.They are quite easy to understand and we do not waste time with the explanation of them.

Algorithm 12: Title triple creation

```
if "title" in item then
    g.add((Article, DCTERMS["title"], Literal(item["title"])))
end
```

More interesting is the algorithm 13 where we can observe that there are two possible fields where the information (about the volume in which the article is published) can be store inside the items present in BIBTEX file.

Algorithm 13: Volume triple creation

```
if "volume" in item and len(item["volume"])>0 then
    g.add((Article, FDO["volume"], Literal(item["volume"])))
else
    if "number" in item and len(item["number"])>0 then
        g.add((Article, FDO["volume"], Literal(item["number"])))
    end
end
```

Finally, I would like to explain how we developed the triples creation about the authors related to a resource.
The algorithm 14 illustrates how the triples are created. At the beginning of this section we talk about the `BibTexParser` customization feature about the author list creation.
In this way we can iterate the triple creation for each author. In the code below, we split the name and the surname of the author thanks to the convention that store the surname followed by the name with a comma between them.

Algorithm 14: Author triples creation

```
if "author" in item then
    for author in item["author"] do
        author_data = author.split(",")
        surname = author_data[0].strip()
        name = author_data[1].strip()
        Author = createURI(FDO, "author", strToID(author=author))
        g.add((Author, RDF.type, FDO["Author"]))
        g.add((Author, FOAF["name"], Literal(name)))
        g.add((Author, FOAF["surname"], Literal(surname)))
        g.add((Article, DCTERMS["creator"], Author))
    end
end
```

As previously said, there are a big number of common fields which can be used in every type of bibliographic document. Sometime happens that there are minor changes but the structure is pretty much the similar.

We now see algorithms which have not been explained yet. Starting from the Tech report where there may be the institutional field.

Algorithm 15: Corporare body triples creation

```
if "institution" in item then
    CorporateBody = createURI(FDO,"corporateBody",
     strToID(ist=item["institution"]))
    g.add((CorporateBody, RDF.type, FRBR["CorporateBody"]))
    g.add((CorporateBody, DCTERMS["description"],
     Literal(item["institution"])))
    g.add((TechReport, FRBR["responsibleEntity"], CorporateBody))
end
```

As we can see in algorithm 15, the information related to the corporate body are stored using the `dcterms:description` data property and then the relationship

between the resource and the institution is added thanks to the last line of the algorithm.
The PhD and master thesis documents have not an institution field but they may have the *school* attribute. In this case we mapped this information like the institutional one and we can use the same algorithm 15.

A more complicated argument is related to the proceedings paper and misc documents. In fact they have also a relationship with an event where they are presented and/or published.
The reported code in algorithm 16 is a summary and it can be applied only if the "series" field, is available. The first step is to split the content of the "series" field if it is available. Then, following the ontology available in fig. 3.7, we check if a number is present, which could represent the year or the edition of the event.

Algorithm 16: Event - corporate body triples creation (a)

```
splitten_data = re.split(" |’",item["series"])
for data in splitten_data do
    if strToID(s=data).isnumeric() then
        numb_found = 1
    end
end
```

Then the second part is divided in two section, formally there is an `if-clause` that check if the `numb_found` variable is "1".
In this case is applied the algorithm algorithm 17 where the event individual is created and then the number is searched and used to fill its `edition` data property.
The remaining part of the "series" field is concatenated and it will be stored into the `dcterms:description` data property of the responsible entity of the event.

Algorithm 17: Event - corporate body triples creation (b)

```
Event = createURI(FDO, "event", strToID(serie=item["series"]))
g.add((Event, RDF.type, FRBR["Event"]))
g.add((InProc, FRBR["partOf"], Event))
corporate = ""
for data in splitten_data do
    if strToID(s=data).isnumeric()) then
        g.add((Event, FDO["edition"], Literal( strToID(s=data),
         datatype=XSD.positiveInteger )))
    else
        corporate += data + " "
    end
end
CorporateBody = createURI(FDO, "corporateBody",
 strToID(c=corporate))
g.add((CorporateBody, RDF.type, FRBR["CorporateBody"]))
g.add((CorporateBody, DCTERMS["description"],
 Literal(corporate.strip())))
g.add((Event, FRBR["responsibleEntity"], CorporateBody))
```

The last option is when there is not any number inside the "series" field so we decided to create the event individual using the "year" field of the resource, if it is available, and then use it as the `edition` data property.
Starting from this assumption we were able to create the event and then using the information about the "series" field we produced the corporate body and its property.

These considerations can be done also for the resources of type "misc". The only difference is that in this case the event and corporate body individuals are instantiated if the word "workshop" is present inside the "note" field instead of the "series" field. The convention about the year if a number is not available, it is still valid.

Algorithm 18: Event - corporate body triples creation (c)

```
if "year" in item and "unknown" not in item["year"] then
    Event = createURI(FDO, "event", strToID(serie=item["series"],
      edition=item["year"]))
    g.add((Event, RDF.type, FRBR["Event"]))
    g.add((Event, FDO["edition"], Literal(item["year"],
      datatype=XSD.positiveInteger)))
    g.add((InProc, FRBR["partOf"], Event))
    CorporateBody = createURI(FDO, "corporateBody"
      ,strToID(cb=item["series"]) )
    g.add((CorporateBody, RDF.type, FRBR["CorporateBody"]))
    g.add((CorporateBody, DCTERMS["description"],
      Literal(item["series"])))
    g.add((Event, FRBR["responsibleEntity"], CorporateBody))
end
```

With this last algorithm we have completed the discussion about the bibliographic parser.

Web Application

The Web application project and the thesis have been developed during my personal research training activity. Also for this reason we decided to choose technologies which I have never studied, so I had the opportunity to learn new approaches and useful frameworks.
We have already talked about how store the information, now we will see how we retrieve and expose them to the user.

5.1 Web technologies

5.1.1 Django

We developed the back-end using the famous Python framework called *Django*. All the detailed information are available at its Web page djangoproject.com, but now we will discuss its main features and advantages.
As above mentioned, Django is a high-level Python Web framework, moreover it is open source. This last things blends well the project principles.
In particular Django has the following advantages:

- fast: Django designers have the goal of creating a framework which helps developers to create application as fast as possible;

- features: it brings with itself some interesting features which help developers to complete common features in Web development;
- secure: it is designed to prevent common security negligence;
- scalable: it helps to distribute the heaviest traffic demands quickly and flexibly;
- versatile: it can be used in different domains.

Django works using a modular approach, I think that in this way developers are able to make the code cleaner and more readable. We used this framework in our Web application development to implement only the back-end although it allows also to build the front-end.
In practice, we developed a *REST API* using Django, which is defined as *"a flexible, lightweight way to integrate applications, and have emerged as the most common method for connecting components in microservices architectures."* (IBM Cloud Education, 2021).
REST, which means Representational State Transfer, is a paradigm or an architectural style which defines how Web information are shared and how computer systems communicate.
For this reason we can develop a REST API using any programming language with different data formats. The only requirements is to follow the REST design principles:

uniform interface: every time a resource is required over an API, it should be always the same.

Client-server separation: this design principle establishes that client and server applications must be independent. The client side communicates with server side only through calling the API's URI.

Stateless: a REST API is stateless, which means that there is no session and every request-response must include all the information.

Cacheability: resources can be cached at client or server side. This requirements is to improve scalability at server side and performance at client side .

Layered architecture: REST API must be developed in a way that both server and client side do not know if they are interacting with the final application or with an intermediary. This requirements is needed because the requests and responses can be managed by communication intermediaries.

Code on demand: this is an optional requirements. In fact, REST APIs returns usually static resources but may happen that the response contains executable code.

REST APIs communicate via HTTP requests using all the possible methods depending the task. The state of a resource is shared using different formats but the most used one is the JavaScript Object Notation (JSON) because it is easily readable by humans and machines but it is also programming language-agnostic.

5.1.2 React

Regarding to the front-end we decided to adopt another open-source framework, React. It is a JavaScript library that allows developer to produce user interface. It was a good choice for many reasons, in fact it is lightweight and fast to use. The main characteristics of React are the following:

Declarative: declarative user interfaces make the code more predictable and easier to debug.

Components: it applies a nested components design approach. In this way the developer can decompose a big "problem" into sub-problems to make easier the development and reduce bugs.

Anywhere applicable: thanks to its designers approach, React can be used in different fields, also server side with Node and mobile with React Native.

Community: React has a big community composed by millions of developer but it is also maintained by Facebook developers.

Moreover, thanks to the fact that it is JavaScript based, it can manage efficiently JSON objects provided by the back-end, as discuss in subsection 5.1.1.

5.2 Web Application development

From that section we will analyze the implementation of the Web application front-end and back-end thanks to the technologies proposed in section 5.1.
In particular we will describe environment settings and the tools used in the first section, then in the second and third section we will inspect the back-end and the front-end respectively.

5.2.1 Environment and tools

The development of the Web application has taken advantage of different tool and software.
I decided to use a *GIT* which is an open source software for distributed version control, it is not useful only on collaborative developing but also for individual developer to track changes and code history. I have used bitbucket.org as repository hosting service just because I have familiarity with this platform thanks to academic past projects.
As code editor I adopted *Visual Studio Code* by Microsoft. It is available for both the operating systems which I use at the time, i.e. Windows and Linux Ubuntu.

We decided to install *OpenLink Virtuoso Open Source Edition* (which is a store database system to store triple) inside a Docker container. Thanks to this choice, we were able to isolate the database system from the hosting system but it allows us to expose only the desired features and moreover we can export the container and its configuration.
For the same reason, we used Docker container for *PostgreSQL*, an open source object-relational database system used to store the information needed to the full text search feature, and its open source administration and development platform, called *pgAdmin*.
Moreover I used *Postman*, which is a tool to design and test API.

5.3 Back-end

As mentioned in subsection 5.1.1, Django is suitable to be modular. For this reason, it has been used to develop the back-end REST APIs and in particular we decided to develop a Django "app" for every different part.
In Django there is the concept of "app" which is different from the "project" concept. In fact, an app is a Web application with a specific objective, a project is composed by many application and configuration. In conclusion, every Django application represents a Python package.

Into the "`backend`" app we put all the settings and the route for each request. For example here we have defined the `DB_ENDPOINT` variable representing the URL endpoint of Virtuoso. It is imported by all the apps, in this way we can change the parameter only once. It is considered a good practice to avoid errors when the values need to be changed.
As previously said, the "`urls`" file maps every request to the correct app, the code in algorithm 19 represents the root `urls` file. From that the requests are redirected to the specific "`urls`" file for a given app and then they select the correct "`view`" to be called. A `view` is a Python function and has the objective to retrieve a request, elaborate and return a response. In our back-end, all, or almost, the response are JSON data.

In fact we can consider the empty path, which through the corresponding "urls" file calls the `view_index` function. The code is not reported because it is only a render function of the `index.html` template. We will see more in details in subsection 5.3.4 how it has been used.

Algorithm 19: URL patterns

```
urlpatterns = [ path("admin/", admin.site.urls),
path("", include("startPage.urls")),
path("dataset/", include("dataset.urls")),
path("task/", include("task.urls")),
path("domain/", include("domain.urls")),
path("creator/", include("creator.urls")),
path("publisher/", include("publisher.urls")),
path("author/", include("author.urls")),
path("event/", include("event.urls")),
path("corporateBody/", include("corporateBody.urls")),
path("paper/", include("referencePaper.urls")),
path("referenceWebpage/", include("referenceWebPage.urls")),
path("journal/", include("journal.urls")),
path("search/", include("textSearch.urls"))
]
```

5.3.1 Views development

We will now see the main steps and functions of the views. They have the same main structure and the changes are related only to the data differences between the resources.

For every resource type there are two main APIs that can be called. For instance, if we consider a dataset resource, we know that the paths must start with `/dataset/` and with this information the `backend` app redirect the request to the `dataset` app. Then there are two possibilities:

- `/dataset/` without any other string after the slash. This call the `index` view which returns all the datasets available inside the database;
- `/dataset/<str:id>` this call, with the string "id" after the slash, invokes the `detail` view which returns all the information related to the dataset with the specified identifier.

To achieve the result we need to perform a query to the database that is available in algorithm 20. As we can see, it firstly binds the `?dataset` variable to the `dcat:Dataset` class and then it retrieves all the relative information. Finally, the triples are ordered in ascendant order to be sure that all the triples about a resources should be near.

Algorithm 20: Dataset query

```
query = """
    select distinct ?dataset ?prop ?obj where {
        ?dataset a dcat:Dataset ;
            ?prop ?obj.
    } order by ASC(?dataset)
"""
```

This type of query provides the idea of all the queries used to retrieve also other type of resources (changing the bounded type class). After some test of the Web application, we discovered a problem with all this query. It derives from the database system, in fact it has a limit variable to setup the maximum number of triples returned.
To bypass that problem there are two solution: we can increase the number of maximum triples returned (actually it is 10000) but it is not scalable. The second solution, the chosen one, is to use the `OFFSET` option of SPARQL query language, which allows to produce in output only the solutions after the specified number. In practice we repeat the query increasing the `OFFSET` number starting from zero, until we do not found further results.

Algorithm 21: Dataset `index` view

```
1  sparql = SPARQLWrapper(endpoint=settings.DB_ENDPOINT)
2  sparql.setQuery(prefix+query)
3  sparql.setReturnFormat(JSON)
4  result = sparql.query().convert()
5  result_set = result["results"]["bindings"]
6  for triple in result_set do
7      clean_dict(triple)
8      previous = "dataset"
9      for item in triple do
10         if "landingPage" in triple[previous] then
11             previous = item
12             continue
13         end
14         triple[item] = clean_url(triple[item])
15         previous = item
16     end
17 end
18 data = create_json(result_set)
19 result = json.dumps(data, indent=4)
20 return HttpResponse(result)
```

Starting from the above algorithm 21 we can make the following considerations:

- from the first line we can see that we have used `SPARQLWrapper` to connect and query the database. The `DB_ENDPOINT` variable is the one previusly mentioned.
- at line two, the query variable is the one present in algorithm 20 and the prefix variable includes all the prefixes in fig. 4.1.
- the line three sets up the format desired to be returned. We decided to use JSON data format.

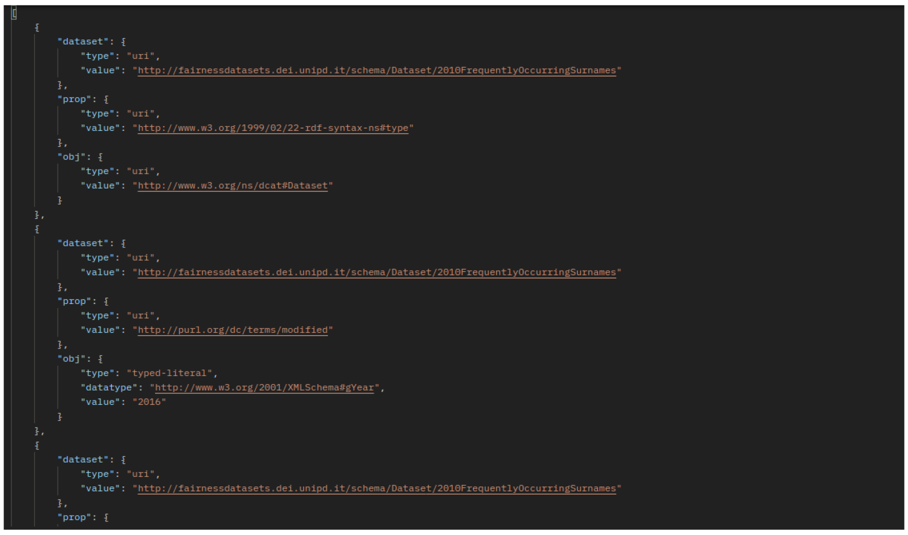
- the line four executes the query to the endpoint and apply the conversion of the retrieved results. If the results are in JSON format, they are converted to a Python dictionary structure.

- the line five is used to retrieve only the needed information. The whole result variable includes useless database and query information.

- Then an important part is the "cleaning" of the retrieved data. In fact, we can observe in fig. 5.1 how the data are returned from the database in JSON format. The `clean_dict` function has the objective to remove all the unnecessary information from the dictionary. The remaining code inside this `for-loop` converts the resources' URI (see fig. 5.2) to their ID.

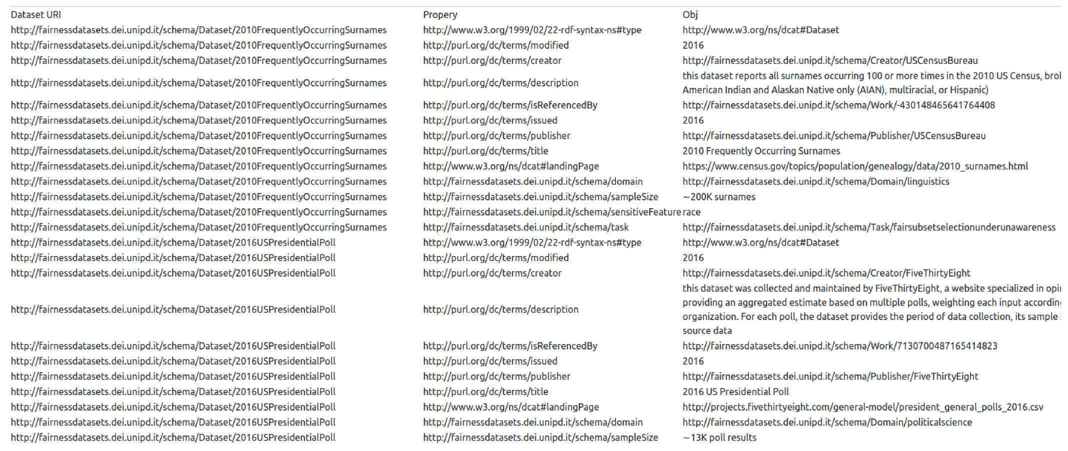
- After the loop we ca see an important function for the back-end, the `create_json`. It takes as input the whole results set and produce in output a list of dictionaries where every dictionary represents a resource. Then the following instruction transforms the list of dictionaries to a list of JSON objects.

- Finally there is the return statement which sends the HTTP response of the JSON data.



```
[
    {
        "dataset": {
            "type": "uri",
            "value": "http://fairnessdatasets.dei.unipd.it/schema/Dataset/2010FrequentlyOccurringSurnames"
        },
        "prop": {
            "type": "uri",
            "value": "http://www.w3.org/1999/02/22-rdf-syntax-ns#type"
        },
        "obj": {
            "type": "uri",
            "value": "http://www.w3.org/ns/dcat#Dataset"
        }
    },
    {
        "dataset": {
            "type": "uri",
            "value": "http://fairnessdatasets.dei.unipd.it/schema/Dataset/2010FrequentlyOccurringSurnames"
        },
        "prop": {
            "type": "uri",
            "value": "http://purl.org/dc/terms/modified"
        },
        "obj": {
            "type": "typed-literal",
            "datatype": "http://www.w3.org/2001/XMLSchema#gYear",
            "value": "2016"
        }
    },
    {
        "dataset": {
            "type": "uri",
            "value": "http://fairnessdatasets.dei.unipd.it/schema/Dataset/2010FrequentlyOccurringSurnames"
        },
        "prop": {
```

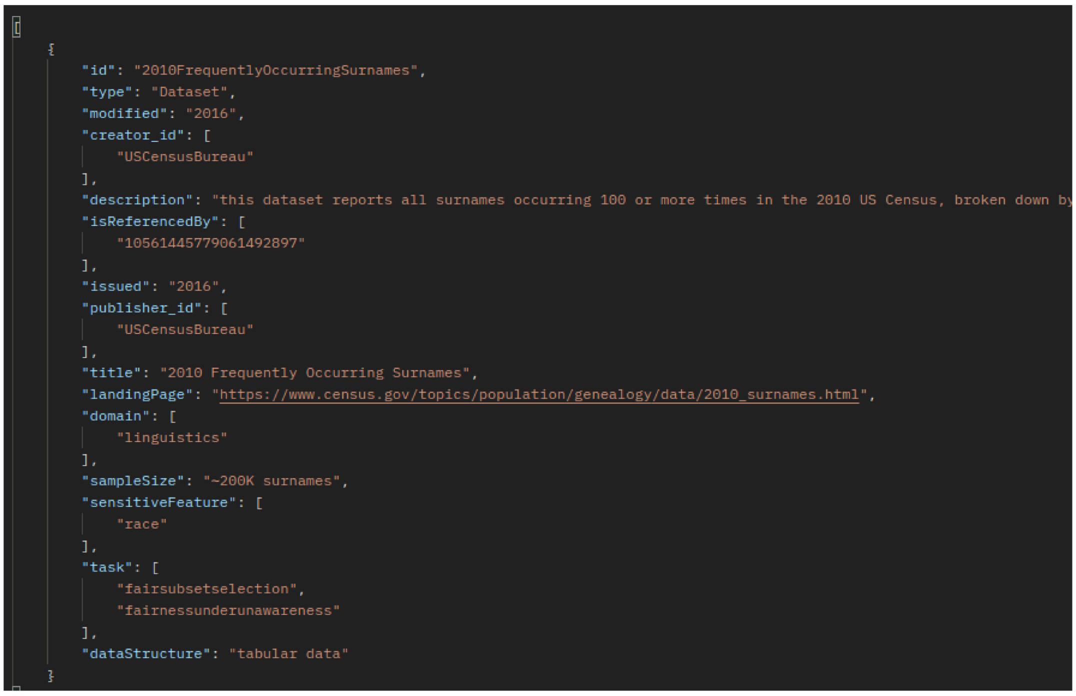
Figure 5.1: Row results from query in JSON format



Dataset URI	Propery	Obj
http://fairnessdatasets.dei.unipd.it/schema/Dataset/2010FrequentlyOccurringSurnames	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/ns/dcat#Dataset
http://fairnessdatasets.dei.unipd.it/schema/Dataset/2010FrequentlyOccurringSurnames	http://purl.org/dc/terms/modified	2016
http://fairnessdatasets.dei.unipd.it/schema/Dataset/2010FrequentlyOccurringSurnames	http://purl.org/dc/terms/creator	http://fairnessdatasets.dei.unipd.it/schema/Creator/USCensusBureau
http://fairnessdatasets.dei.unipd.it/schema/Dataset/2010FrequentlyOccurringSurnames	http://purl.org/dc/terms/description	this dataset reports all surnames occurring 100 or more times in the 2010 US Census, brol American Indian and Alaskan Native only (AIAN), multiracial, or Hispanic)
http://fairnessdatasets.dei.unipd.it/schema/Dataset/2010FrequentlyOccurringSurnames	http://purl.org/dc/terms/isReferencedBy	http://fairnessdatasets.dei.unipd.it/schema/Work/-430148465641764408
http://fairnessdatasets.dei.unipd.it/schema/Dataset/2010FrequentlyOccurringSurnames	http://purl.org/dc/terms/issued	2016
http://fairnessdatasets.dei.unipd.it/schema/Dataset/2010FrequentlyOccurringSurnames	http://purl.org/dc/terms/publisher	http://fairnessdatasets.dei.unipd.it/schema/Publisher/USCensusBureau
http://fairnessdatasets.dei.unipd.it/schema/Dataset/2010FrequentlyOccurringSurnames	http://purl.org/dc/terms/title	2010 Frequently Occurring Surnames
http://fairnessdatasets.dei.unipd.it/schema/Dataset/2010FrequentlyOccurringSurnames	http://www.w3.org/ns/dcat#landingPage	https://www.census.gov/topics/population/genealogy/data/2010_surnames.html
http://fairnessdatasets.dei.unipd.it/schema/Dataset/2010FrequentlyOccurringSurnames	http://fairnessdatasets.dei.unipd.it/schema/domain	http://fairnessdatasets.dei.unipd.it/schema/Domain/linguistics
http://fairnessdatasets.dei.unipd.it/schema/Dataset/2010FrequentlyOccurringSurnames	http://fairnessdatasets.dei.unipd.it/schema/sampleSize	~200K surnames
http://fairnessdatasets.dei.unipd.it/schema/Dataset/2010FrequentlyOccurringSurnames	http://fairnessdatasets.dei.unipd.it/schema/sensitiveFeature	race
http://fairnessdatasets.dei.unipd.it/schema/Dataset/2010FrequentlyOccurringSurnames	http://fairnessdatasets.dei.unipd.it/schema/task	http://fairnessdatasets.dei.unipd.it/schema/Task/fairsubsetselectionunderunawareness
http://fairnessdatasets.dei.unipd.it/schema/Dataset/2016USPresidentialPoll	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/ns/dcat#Dataset
http://fairnessdatasets.dei.unipd.it/schema/Dataset/2016USPresidentialPoll	http://purl.org/dc/terms/modified	2016
http://fairnessdatasets.dei.unipd.it/schema/Dataset/2016USPresidentialPoll	http://purl.org/dc/terms/creator	http://fairnessdatasets.dei.unipd.it/schema/Creator/FiveThirtyEight
http://fairnessdatasets.dei.unipd.it/schema/Dataset/2016USPresidentialPoll	http://purl.org/dc/terms/description	this dataset was collected and maintained by FiveThirtyEight, a website specialized in opi providing an aggregated estimate based on multiple polls, weighting each input accordin organization. For each poll, the dataset provides the period of data collection, its sample source data
http://fairnessdatasets.dei.unipd.it/schema/Dataset/2016USPresidentialPoll	http://purl.org/dc/terms/isReferencedBy	http://fairnessdatasets.dei.unipd.it/schema/Work/7130700487165414823
http://fairnessdatasets.dei.unipd.it/schema/Dataset/2016USPresidentialPoll	http://purl.org/dc/terms/issued	2016
http://fairnessdatasets.dei.unipd.it/schema/Dataset/2016USPresidentialPoll	http://purl.org/dc/terms/publisher	http://fairnessdatasets.dei.unipd.it/schema/Publisher/FiveThirtyEight
http://fairnessdatasets.dei.unipd.it/schema/Dataset/2016USPresidentialPoll	http://purl.org/dc/terms/title	2016 US Presidential Poll
http://fairnessdatasets.dei.unipd.it/schema/Dataset/2016USPresidentialPoll	http://www.w3.org/ns/dcat#landingPage	http://projects.fivethirtyeight.com/general-model/president_general_polls_2016.csv
http://fairnessdatasets.dei.unipd.it/schema/Dataset/2016USPresidentialPoll	http://fairnessdatasets.dei.unipd.it/schema/domain	http://fairnessdatasets.dei.unipd.it/schema/Domain/politicalscience
http://fairnessdatasets.dei.unipd.it/schema/Dataset/2016USPresidentialPoll	http://fairnessdatasets.dei.unipd.it/schema/sampleSize	~13K poll results

Figure 5.2: Query results

In fig. 5.3 we can find the results of a request of a dataset resource. As the reader can see, there are some fields where there is string text representing information, like the `description` or the `modified`, but there are also fields where there are the ID of the related resources, for example the `creator_id`.
Moreover, when there are more than one related resource, the information are stored using a list inside the value of a JSON key.



```
[
    {
        "id": "2010FrequentlyOccurringSurnames",
        "type": "Dataset",
        "modified": "2016",
        "creator_id": [
            "USCensusBureau"
        ],
        "description": "this dataset reports all surnames occurring 100 or more times in the 2010 US Census, broken down by
        "isReferencedBy": [
            "10561445779061492897"
        ],
        "issued": "2016",
        "publisher_id": [
            "USCensusBureau"
        ],
        "title": "2010 Frequently Occurring Surnames",
        "landingPage": "https://www.census.gov/topics/population/genealogy/data/2010_surnames.html",
        "domain": [
            "linguistics"
        ],
        "sampleSize": "~200K surnames",
        "sensitiveFeature": [
            "race"
        ],
        "task": [
            "fairsubsetselection",
            "fairnessunderunawareness"
        ],
        "dataStructure": "tabular data"
    }
```

Figure 5.3: API response

Before going on, I would like to specify how the `detail` view query is developed. It is very similar to the one presents in algorithm 20 but it need to select the desired resource and we can achieve this objective in two ways:

- apply a `FILTER regex` over the `?dataset` variable;
- bind the resource directly because we know that the convention used to create the resources URI is "namespace / resource_type / ID" so, in the case of a dataset, its URI will be "namespace/dataset/ID".

The remaining part of the `detail` view is the same.

Now we can explain the above mentioned functions used inside the views. The first one is the `clean_dict`, I do not report the code because it simply iterates over all the triples and their items and keeps only the value belonging to the "value" keys of the structure in fig. 5.1.
The `clean_url` function has the objective of cleaning the resources' URL (see fig. 5.2 to understand their structure).

Algorithm 22: clean_url

```
if "#" in item then
    item = item[item.rfind("#")+1:]
else
    item = item[item.rfind("/")+1:]
end
return item
```

As the reader can observe from algorithm 22 there are two cases: if the URI contains the "#" symbol it should start from that symbol, on the contrary it will cut starting from the last "/".
The most important function is the `create_json` function which transforms all the retrieved triples into a list of JSON objects.

Algorithm 23: create_json - part 1

```
dataset = []
for triple in result_set do
    found = 0
    for item in dataset do
        if triple['dataset'] == item.get('id') then
            found = 1 break
        end
    end
    if found == 0 then
        dataset.append('id':triple['dataset'])
    end
end
```

The first part of the `create_json` function, available in algorithm 23, creates the resources list and then creates all the dictionary objects only with the ID key.

The second part of the function, available in algorithm 24, is the most time consuming part of the `create_json` function.

In fact it has two nested `for-loop`, the first one over all the triples and the second one over all the elements of the dataset list of JSON objects.

In the reported resume of the function are not present all the cases, the reader can extend to all the possible cases easily. The algorithm searches for every triple the corresponding dictionary resource and then adds the information related to the considered triple. As the reader can see there are two type of insertion functions: the `insert_element` function adds a single element in the form of `"key:value"`, instead the `insert_list` function checks if the input key is already presents in the dictionary resource and, if it is the case, appends the input element. On the contrary, i.e. the key does not exist yet, the function creates the new key with a list of one elements, the input one, as value.

Algorithm 24: create_json - part 2

```
for triple in result_set do
    for i in range(len(dataset)) do
        if triple["dataset"] == dataset[i].get("id") then
            switch triple["prop"] do
                case "title" do
                    insert_element(dataset[i],"title",triple["obj"])
                end
                case "description" do
                    insert_element(dataset[i],"description",triple["obj"])
                end
                case "dataSpec" do
                    insert_element(dataset[i],"dataSpec",triple["obj"])
                end
                case "sensitiveFeature do
                    insert_list(dataset[i],"sensitiveFeature",triple["obj"])
                end
                case "domain" do
                    insert_list(dataset[i],"domain",triple["obj"])
                end
                case "task" do
                    insert_list(dataset[i],"task",triple["obj"])
                end
                case "creator" do
                    insert_list(dataset[i],"creator_id",triple["obj"])
                end
                case "isReferencedBy" do
                    insert_list(dataset[i],"isReferencedBy",triple["obj"])
                end
            end
        end
    end
end
return dataset
```

During the Web application development we took care of its performances. This argument is not strictly related with how the user interacts with the applications but it affects directly the user experience.
When the above version of the `create_json` function was ready, we applied some tests to verify if the response time was acceptable. Unfortunately, the performance was not as good as we expected. From the test we concluded that the problem was originated from right over the `create_json` function.
In fact, we noticed that it ran approximately in 10 seconds (in some cases also more than that) and on the contrary the remaining parts of the view act in matter of milliseconds. So we can easily assume that a user after that time will leave the Web application in flavour of a better performing one.
We have done a lot of test using different data structures to improve the `create_json` function and after some researches we decided to used a dictionary of dictionary instead a list of dictionary. In this way we can avoid to use two nested `for-loops`.
In practice the structure will be composed of key-value pairs where the key is the resources ID and the value is the dictionary containing all the information.
The new version of the `create_json` function is reported in algorithm 25.
Now we are going to explain some details of this new version, starting from the first `for-loop`.

First of all, it is more compact, in fact using the dictionary for each triple we can verify if the ID (and so the key of the dictionary entries) is already present, on the contrary it is added. In this way we do not need to scan linearly all the elements of the list for each triple.
In algorithm 25 we do not report all the possible cases because, as in the algorithm 24 the reader can extend to all the possible options. We want to show how the dictionary of dictionary impacts the implementation compared to the previous version.
Also in this case we avoid to use two nested *for-loops* because we retrieve the resources' dictionary using the ID which is the key of the structure. As the reader can see, it is retrieved directly into the insert functions and given as in-

put parameter. The two version of insertion functions are still the same in both the developed versions.

Finally, to produce the correct structure expected by the front-end, this new version returns only the entries of the main dictionary converted into a list.

Algorithm 25: create_json - new version

```
dataset_set = for triple in result_set do
    if triple['dataset'] not in dataset_set.keys() then
        dataset_set.update(triple['dataset']:"id":triple['dataset'])
    end
end
for triple in result_set do
    switch triple['prop'] do
        case 'title' do
            insert_element(dataset_set[triple['dataset']],'title',triple['obj'])
        end
        case 'description' do
            insert_element(dataset_set[triple['dataset']],'description',triple['obj'])
        end
        case 'creator' do
            insert_list(dataset_set[triple['dataset']],'creator_id',triple['obj'])
        end
        case ... do

        end
    end
end
return list(dataset_set.values())
```

Considering another path, we will now discuss about the one starting with `/paper` and in particular the one dedicated to retrieve all the papers related to a dataset-task pair.
The full path is in the following form:`/paper/<str:id>/<str:task_id>`
As we can image, the query will be different from the previous illustrated one. In fact, it needs to retrieve the resources matching the pairs using the blank nodes. The query is reported in algorithm 26.
The code of this view does not use the previous mentioned `create_json` function but it uses the `create_json` made for the paper (so it has different fields but the same structure). The elaborations, to remove the useless JSON information in the response and to clean the URIs, are done before calling the `create_json` function.

Algorithm 26: Query to retrieve documents about a task of a dataset

```
query = """
    select distinct ?paper where {
        fdoDataset:"""+dataset_id+""" a dcat:Dataset ;
            fdo:has ?bNode .
        ?bNode ?about fdoTask:"""+task_id+""";
            fdo:refers ?paper .
        ?paper ?pred ?obj .
    } order by ASC(?paper)
"""
```

Before moving on to the section dedicated to the full text search, it is interesting to explain how the debug and log activities are done. They are very important in every development project.
Differently from a classical software style, during Web and in particular back-end development we can not simply "`print`" the information. In this case, the solution is using a module that allows to "export" information, the adopted solution is the "logging" module. In practice it allows to print information to a specified file.

5.3.2 Full text search view

In this section we will analyze how the views dedicated to the full text search are implemented.
There are two views: the first one query the table dedicated to the datasets information, the second one instead investigates the documents information using a similar query.
In this case the path allowed by the "urls" file are (considering that the base path is `/search/`):

- the empty path which reply an "error" message
- `dataset/<str:searchText>` which redirects to the view dedicated to retrieve information about datasets;
- `paper/<str:searchText>` which redirects to the view dedicated to retrieve information about documents.

We will analyze only one of these views because the structure, the used functions and the queries are similar.
The query, reported in algorithm 27, is obviously PostgreSQL query language because, as we previously mentioned, we use a PostgreSQL database management system.

Algorithm 27: Full text dataset query

```
query = """
    SELECT id
    FROM "indexSchema".dataset
    WHERE data @@ plainto_tsquery('english',%s)
    ORDER BY id ASC
"""
```

For those whom already know this query language the main structure of the query is understandable. The interesting part is after the `WHERE` keyword. In fact the `data @@ plainto_tsquery("english", %s)` code is not common to

see, at least for someone which works with the standard features.
In practice it makes a sort of intersection between the `data` column and the results of the `plainto_tsquery` function. This statement converts every piece of text into a `tsquery` data-type which is the corresponding data-type of `tsvector` for the queries. The inputs are the text language and the text to be transformed.

Now we have understood the query, so we can talk about the implementation.

Algorithm 28: Full text view

```
conn = psycopg2.connect(dbname="index_db", user=settings.user,
 password=settings.pswd, host=settings.db_host)
cur = conn.cursor()
cur.execute(query, [searchText])
results = cur.fetchall()
cur.close()
conn.close()
result = json.dumps(results, indent=4)
return HttpResponse(result)
```

The code is available in the algorithm 28, where the `query` variable is the one present in the algorithm 27.
We can see that there are a lot of "database" stuff but nothing about the data elaborations. In fact we only need to establish the connection, execute the query and retrieve the results. From the algorithm we can noticed that we used the `psycopg2` module to reach the database inside the PostgreSQL DBMS.
Finally, we convert the results to JSON data format and we send them using the `HttpResponse` function.

5.3.3 Form view

In this subsection we will talk about the only view dedicated to manage *POST* request. It is particularly easy, in fact it checks if the request is of type POST and then it retrieves the request body.

This information are loaded as JSON format and then they are used into the mail content.
In fact the objective of this view is to retrieve the information from the front-end form and to send them by mail so that the data creator can manage and upload them into the Web application database.
To manage the e-mail sending we use the `send_mail` function provided by `django.core.mail`. We only need to configure the required information (the host name, the port, the username and the password) inside the `settings` file.

5.3.4 Development to production

We have already discussed about the `view_index` function which render the `index.html` file.
During the Web application development two processes were required: the back-end service was executed to expose all the REST APIs and the front-end service was executed to expose the user interface. The reason why we decided to use this approach instead of a single process is easy: every time we save a code modification the front-end was reloaded and so we could see immediately the updates.
This approach has two main problems at the production stage:

- the front-end code is fully available, so there are both security and copyright problems;
- running back-end and front-end over two distinct processes increase the possibility of the Web application crash. In fact, there are two processes that can crash, go down and make the system unresponsive.

For these reasons the above approach is not feasible in production environment so we studied how to run in a single process and without exposing the source code.
The solution found is using `webpack` which is "a static module bundler for modern JavaScript applications". It is a useful tool with many features which help

developers to create a single bundle that can be imported as static script file in a HTML file.

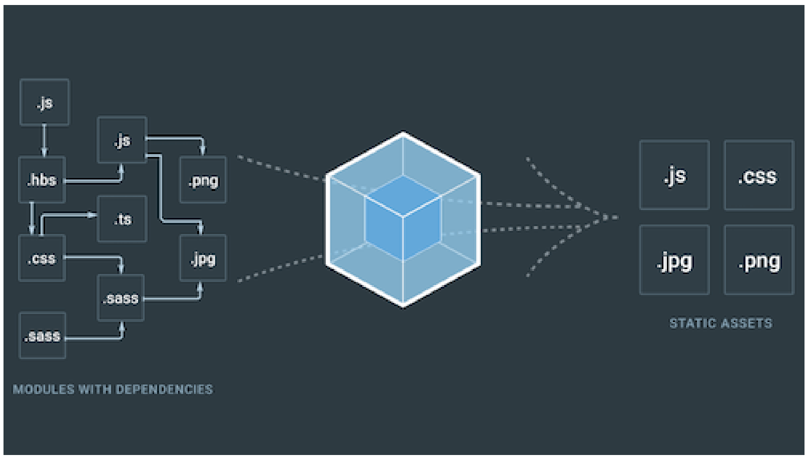


Figure 5.4: Webpack idea

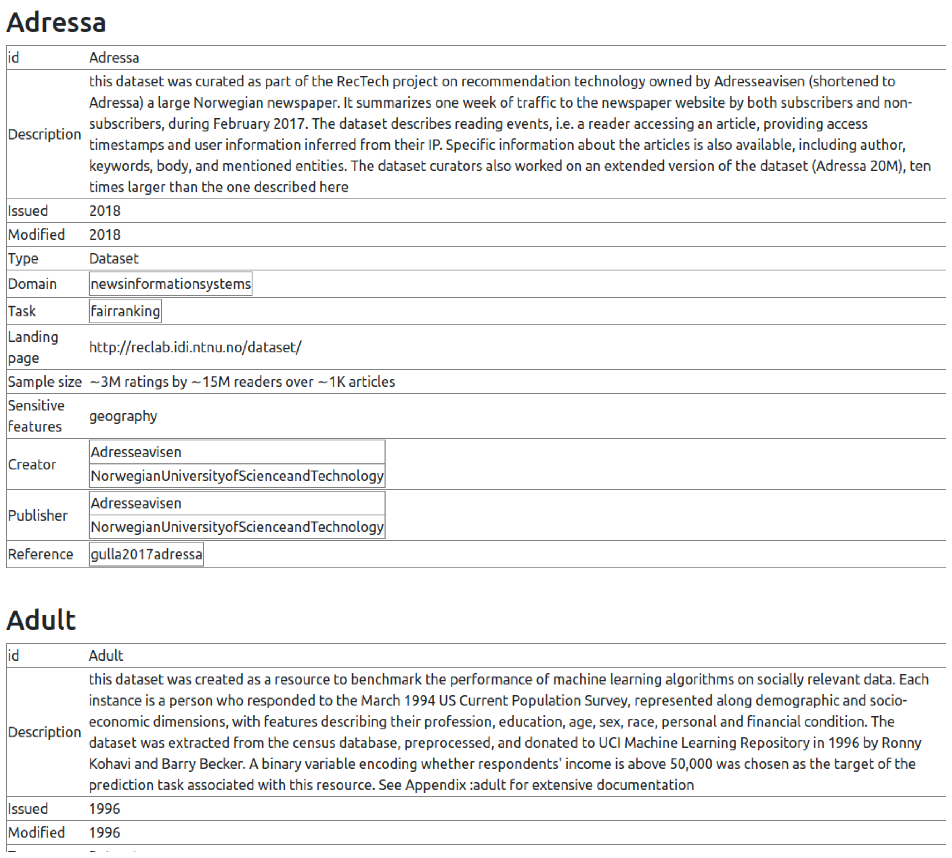
We can now understand what the `index.html` file does. In practice it is a sort of container for the front-end which is executed thought the JavaScript bundle. In conclusion, there will be one process, the one relative to the back-end, which executes also the React front-end.

5.4 Front-end

In this section we will explain how the React front-end has been developed. During the implementation sometimes we needed to change the used techniques to achieve the desired results. So we will make a summary of the changes and the final results.

The fig. 5.2 represents the first test we have done to display the results of the `/dataset` API call. It is interesting to specify that it was a great satisfaction to display this row data because, as we have already mentioned, it was my first approach to React. Obviously, they were difficult to understand if they are displayed in this way but, when the back-end was ready, we were able to display a more understandable presentation of the resources using a tabular view.

It is useful to mention that we decided to start from the dataset information because all the Web application is about the dataset and their related information.



Adressa

id	Adressa
Description	this dataset was curated as part of the RecTech project on recommendation technology owned by Adresseavisen (shortened to Adressa) a large Norwegian newspaper. It summarizes one week of traffic to the newspaper website by both subscribers and non-subscribers, during February 2017. The dataset describes reading events, i.e. a reader accessing an article, providing access timestamps and user information inferred from their IP. Specific information about the articles is also available, including author, keywords, body, and mentioned entities. The dataset curators also worked on an extended version of the dataset (Adressa 20M), ten times larger than the one described here
Issued	2018
Modified	2018
Type	Dataset
Domain	newsinformationsystems
Task	fairranking
Landing page	http://reclab.idi.ntnu.no/dataset/
Sample size	~3M ratings by ~15M readers over ~1K articles
Sensitive features	geography
Creator	Adresseavisen NorwegianUniversityofScienceandTechnology
Publisher	Adresseavisen NorwegianUniversityofScienceandTechnology
Reference	gulla2017adressa

Adult

id	Adult
Description	this dataset was created as a resource to benchmark the performance of machine learning algorithms on socially relevant data. Each instance is a person who responded to the March 1994 US Current Population Survey, represented along demographic and socio-economic dimensions, with features describing their profession, education, age, sex, race, personal and financial condition. The dataset was extracted from the census database, preprocessed, and donated to UCI Machine Learning Repository in 1996 by Ronny Kohavi and Barry Becker. A binary variable encoding whether respondents' income is above 50,000 was chosen as the target of the prediction task associated with this resource. See Appendix :adult for extensive documentation
Issued	1996
Modified	1996

Figure 5.5: Dataset tabular view

From the above fig. 5.5, but also from fig. 5.3, we can see that there are information stored inside a list (or array) and there are the IDs instead of the information which should be displayed at front-end (for example task, domain, creator, publisher but also a the reference documents).

Now, the most challenging step was to retrieve all the information from their IDs. From the back-end point of view it is not a problem because for each type of resource there is its dedicated API. All the requests to the back-end are instantiated using the `AXIOS` function, which is a "promise-based HTTP Client for node.js and the browser", as stated from its introduction page.

An example of request code is available in algorithm 29, now we will inspect line by line to understand better what it does.

The first things to notice is that there are concatenated functions, in this way the code is shorter and we do not need to create variables to store the returned

Algorithm 29: getDatasetList

```
1  axios
2      .get("${host}/dataset/")
3      .then(res => {
4          setDatasetList(res.data);
5          res.data.map(data =>{
6              saveJSON("dataset/"+data.id,data);
7          })
8          setLoading(false);
9      }).catch(err => {
10         console.error(err);
11         setLoading(false);
12     })
```

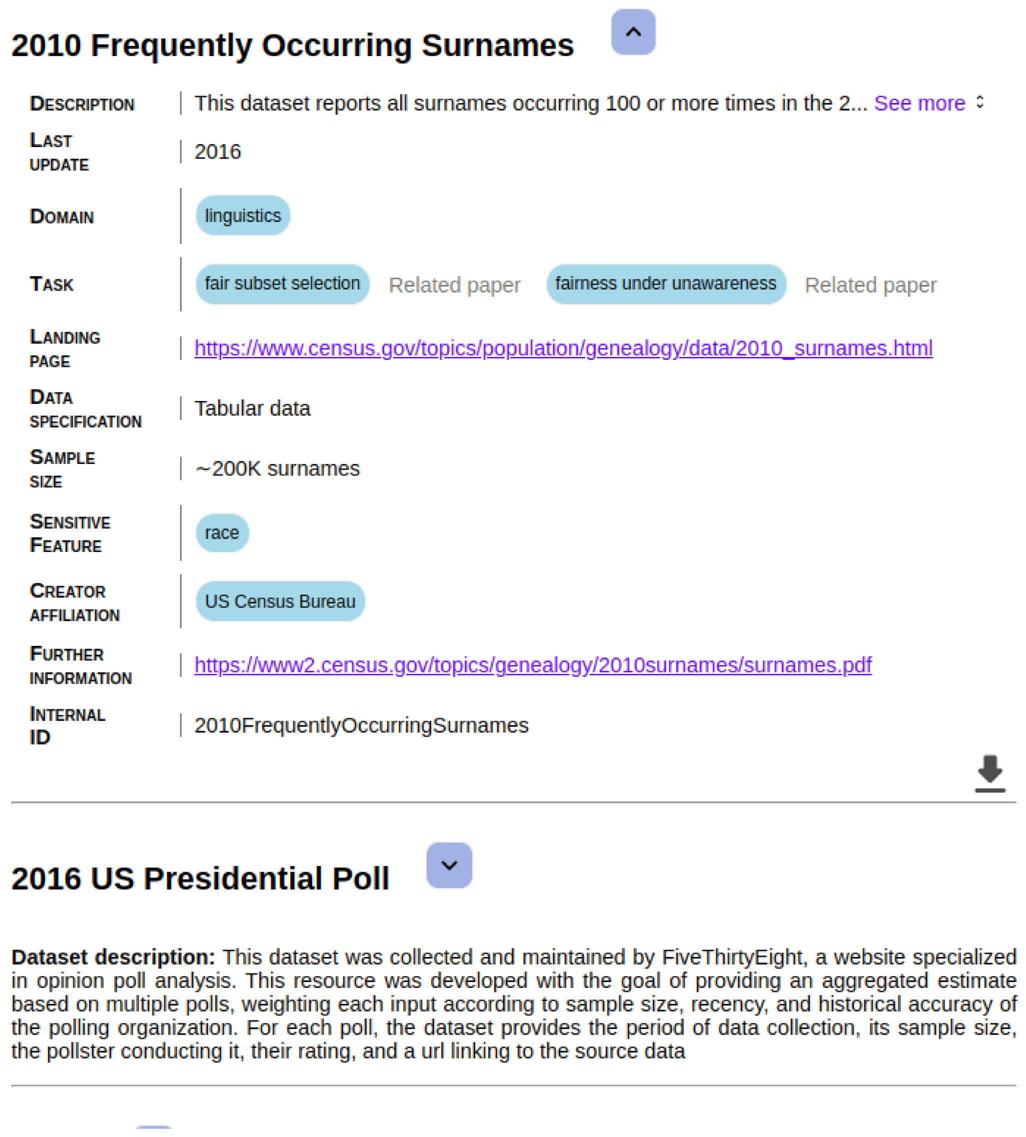
values.
The second line makes the request to the input URL, in this case it requires all the dataset resources as we have seen in subsection 5.3.1.
The third line retrieves the response and starting from this line we make its elaboration. The `setDatasetList` is a set function connected to a `datasetList` variable. In practice, the `datasetList` incorporates a state into a function component called *Hook*. An *Hook* is a React feature which is used to "contains reusable code logic that is separate from the component tree" (Banks and Porcello, 2020).
After that, we iterate over all the resources in the response and for each of these we apply the `saveJSON` custom function, which store the information into the `session storage` of the browser. It is done to improve performances of the Web application, so when a resource is requested, it is searched inside the session storage. If it is not available then a request to the back-end is created.
Finally, there is a `setLoading` Hook function to "communicate" that all the information are retrieved and stored.

From line nine of the algorithm there is the `catch` clause which is used to intercept and manage an error response.

Before going on, we need to specify two implementation strategy:

- the `saveJSON` function takes two input parameters. In fact, the information are stored in a key-value pair way. The key is composed by the resource type plus the resource ID, i.e. `dataset/resource_ID`, the value instead is composed by the information in JSON format of the resource. Moreover, there is the symmetric function to retrieve information form the session storage, called `loadJSON`. It requires only one parameter: the key of the element inside of the session storage which is composed as we previously explained.
- the second consideration is that algorithm 29 has a symmetric implementation, called `getDataset`, which allows developers and so the front-end to retrieve the information about a specific resource. It has the same implementation with the only difference of the URL, It is in the form `"$host/dataset/$dataset_id"`, according to the back-end specification.

To retrieve all the correct information from their IDs, we used a nested structure using more and more detailed and specific components, as React requires.
After some tests, we decided to switch from a tabular view using the `table` HTML tag to a tabular view using `div` and specific classes because we notice that in this way we can achieve a better graphic visualization.
In order to make the contents more readable we used thin lines to separates resources and to distinguish the fields inside a resource. The results are proposed in fig. 5.6 and fig. 5.7



2010 Frequently Occurring Surnames

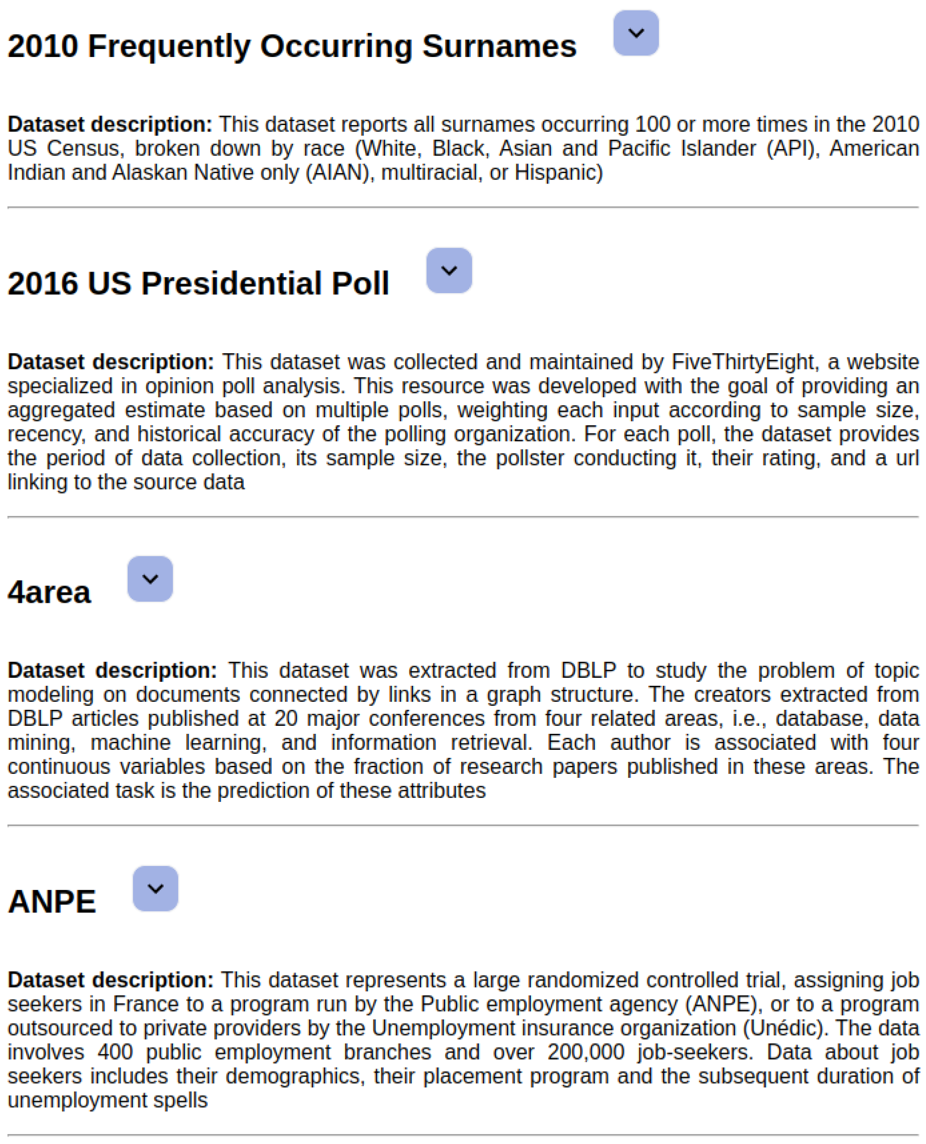
Description	This dataset reports all surnames occurring 100 or more times in the 2... See more
Last update	2016
Domain	linguistics
Task	fair subset selection Related paper fairness under unawareness Related paper
Landing page	https://www.census.gov/topics/population/genealogy/data/2010_surnames.html
Data specification	Tabular data
Sample size	~200K surnames
Sensitive Feature	race
Creator affiliation	US Census Bureau
Further information	https://www2.census.gov/topics/genealogy/2010surnames/surnames.pdf
Internal ID	2010FrequentlyOccurringSurnames

2016 US Presidential Poll

Dataset description: This dataset was collected and maintained by FiveThirtyEight, a website specialized in opinion poll analysis. This resource was developed with the goal of providing an aggregated estimate based on multiple polls, weighting each input according to sample size, recency, and historical accuracy of the polling organization. For each poll, the dataset provides the period of data collection, its sample size, the pollster conducting it, their rating, and a url linking to the source data

Figure 5.6: Dataset page datail

As the reader can notice there are arrow buttons, in fact we decide to display only the dataset title and its description when a user accesses the dataset page, as you can see in fig. 5.7. In this way, the page is more readable and a user can choose the resource to inspect.



2010 Frequently Occurring Surnames

Dataset description: This dataset reports all surnames occurring 100 or more times in the 2010 US Census, broken down by race (White, Black, Asian and Pacific Islander (API), American Indian and Alaskan Native only (AIAN), multiracial, or Hispanic)

2016 US Presidential Poll

Dataset description: This dataset was collected and maintained by FiveThirtyEight, a website specialized in opinion poll analysis. This resource was developed with the goal of providing an aggregated estimate based on multiple polls, weighting each input according to sample size, recency, and historical accuracy of the polling organization. For each poll, the dataset provides the period of data collection, its sample size, the pollster conducting it, their rating, and a url linking to the source data

4area

Dataset description: This dataset was extracted from DBLP to study the problem of topic modeling on documents connected by links in a graph structure. The creators extracted from DBLP articles published at 20 major conferences from four related areas, i.e., database, data mining, machine learning, and information retrieval. Each author is associated with four continuous variables based on the fraction of research papers published in these areas. The associated task is the prediction of these attributes

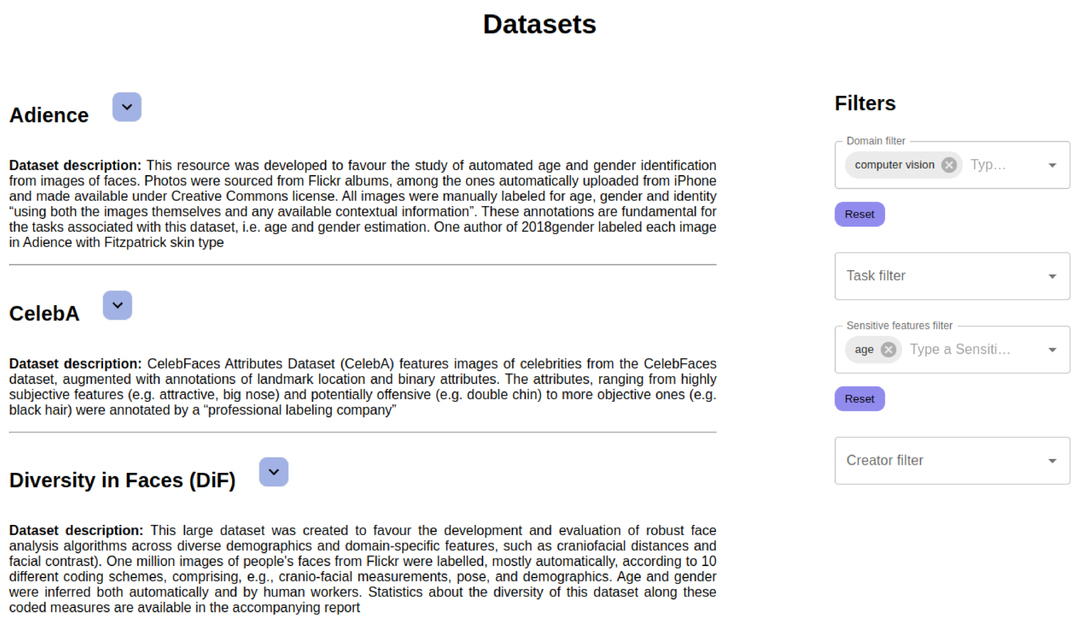
ANPE

Dataset description: This dataset represents a large randomized controlled trial, assigning job seekers in France to a program run by the Public employment agency (ANPE), or to a program outsourced to private providers by the Unemployment insurance organization (Unédic). The data involves 400 public employment branches and over 200,000 job-seekers. Data about job seekers includes their demographics, their placement program and the subsequent duration of unemployment spells

Figure 5.7: Dataset page

Moreover, when a user wants to see more details about a resource and opens its detailed view, the description is contracted within a line and clicking on text button "*see more*", the full description is displayed, as the reader can see in fig. 5.6.

Another things that can be observed in fig. 5.6 are the light blue buttons. They can be clicked to apply filters to the list of displayed resources. We will see more in details how the filters are implemented but at the moment the reader needs to know that a user can apply restriction over the list of displayed resources based on their features.



Datasets

Adience

Dataset description: This resource was developed to favour the study of automated age and gender identification from images of faces. Photos were sourced from Flickr albums, among the ones automatically uploaded from iPhone and made available under Creative Commons license. All images were manually labeled for age, gender and identity “using both the images themselves and any available contextual information”. These annotations are fundamental for the tasks associated with this dataset, i.e. age and gender estimation. One author of 2018gender labeled each image in Adience with Fitzpatrick skin type

CelebA

Dataset description: CelebFaces Attributes Dataset (CelebA) features images of celebrities from the CelebFaces dataset, augmented with annotations of landmark location and binary attributes. The attributes, ranging from highly subjective features (e.g. attractive, big nose) and potentially offensive (e.g. double chin) to more objective ones (e.g. black hair) were annotated by a “professional labeling company”

Diversity in Faces (DiF)

Dataset description: This large dataset was created to favour the development and evaluation of robust face analysis algorithms across diverse demographics and domain-specific features, such as craniofacial distances and facial contrast). One million images of people's faces from Flickr were labelled, mostly automatically, according to 10 different coding schemes, comprising, e.g., cranio-facial measurements, pose, and demographics. Age and gender were inferred both automatically and by human workers. Statistics about the diversity of this dataset along these coded measures are available in the accompanying report

Filters

Domain filter: computer vision

Sensitive features filter: age

Figure 5.8: Full dataset page content

In fig. 5.8 we can see all the contents which are available inside the dataset page: at the left side the dataset resources and at the right one there are the faceted search filters. In the example reported here has been applied two filters: we restrict the list only to documents having "computer vision" as domain and "age" as sensitive feature.

The filters can be applied by clicking the buttons inside the resources detail (see fig. 5.6) but also selecting one of the present in the list of a filter field or directly typing.

In fact, if we click a filter field, the entire list of available options will appear and typing some text we can see the options which are compatible with input text, as we can see in the example in fig. 5.9.

Finally to remove a filter there are two options: if we want to remove a single option we can click the cross button inside itself but if we need to delete all the options of a fields we can use the "reset" button.

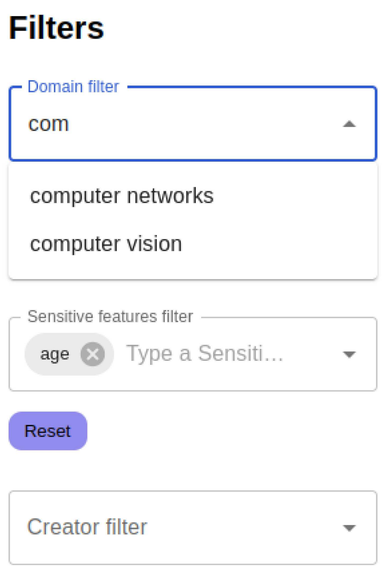
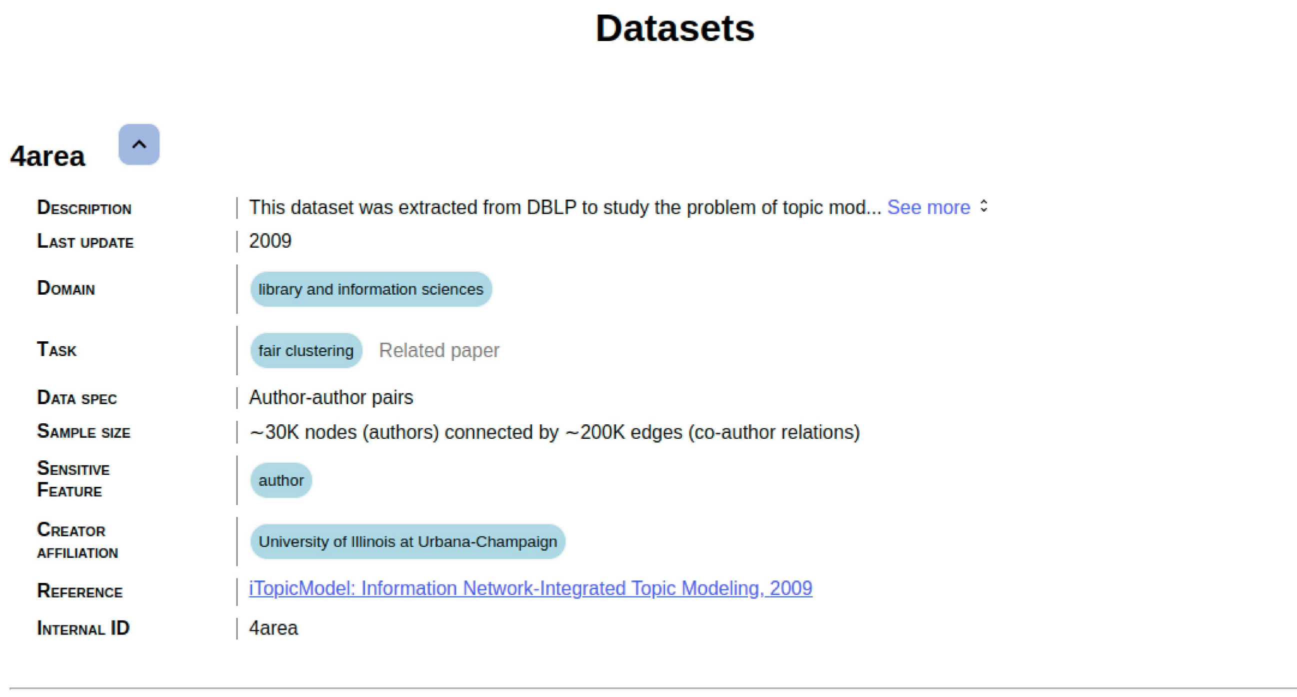


Figure 5.9: Example of filter addition

The sidebar with the filters options are available only if there is a list of datasets, in fact the dataset page is configured to work although a single dataset is required and in this case it is displayed as in fig. 5.10.



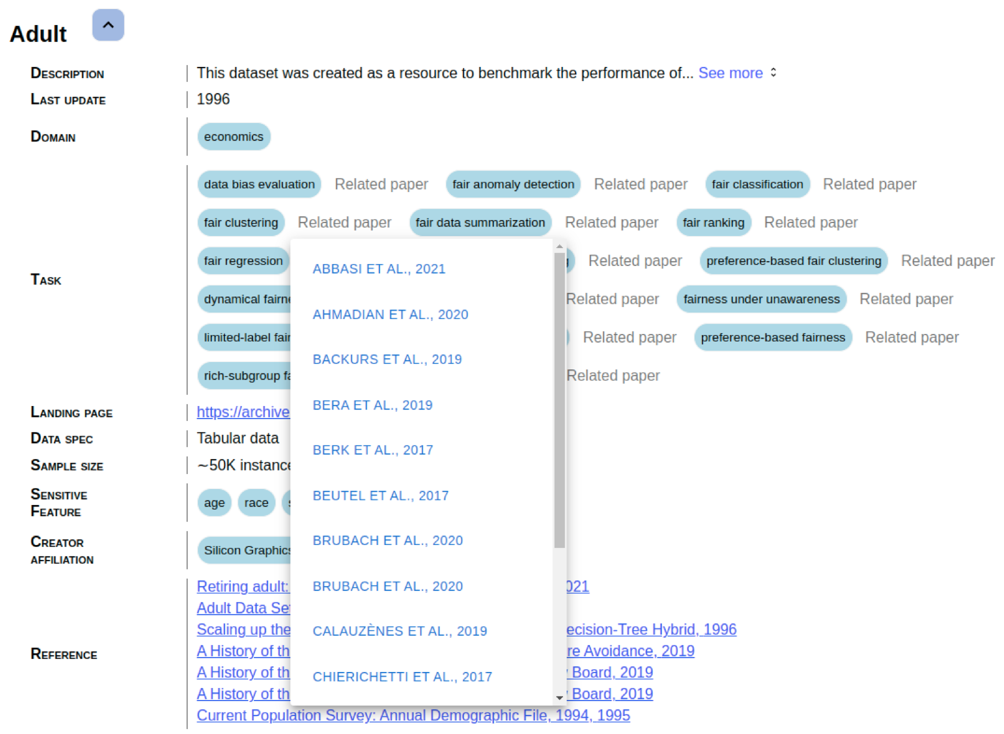
Datasets

4area

Description	This dataset was extracted from DBLP to study the problem of topic mod... See more
Last update	2009
Domain	library and information sciences
Task	fair clustering Related paper
Data spec	Author-author pairs
Sample size	~30K nodes (authors) connected by ~200K edges (co-author relations)
Sensitive Feature	author
Creator affiliation	University of Illinois at Urbana-Champaign
Reference	iTopicModel: Information Network-Integrated Topic Modeling, 2009
Internal ID	4area

Figure 5.10: Single dataset view

To conclude the explanation of the *Datasets* Web page, we need to talk about the `Related paper` buttons which allow to show the list of the documents related to a task in the considered dataset. In the example reported in fig. 5.11 we can see an example of list displayed when one of the button is clicked.



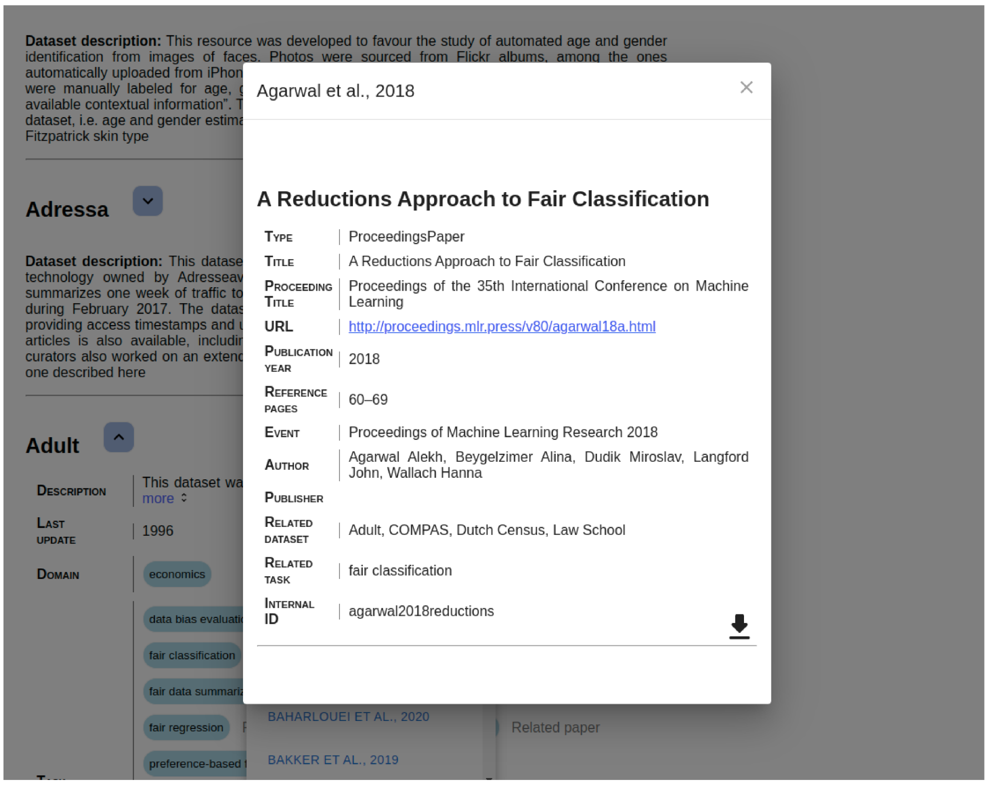
Adult

Description	This dataset was created as a resource to benchmark the performance of... See more
Last update	1996
Domain	economics
Task	data bias evaluation Related paper · fair anomaly detection Related paper · fair classification Related paper · fair clustering Related paper · fair data summarization Related paper · fair ranking Related paper · fair regression · Related paper · preference-based fair clustering Related paper · dynamical fairn · Related paper · fairness under unawareness Related paper · limited-label fair · Related paper · preference-based fairness Related paper · rich-subgroup f · Related paper
Landing page	https://archive
Data spec	Tabular data
Sample size	~50K instance
Sensitive Feature	age · race
Creator affiliation	Silicon Graphics
Reference	Retiring adult: · 021 · Adult Data Se · Scaling up the · ecision-Tree Hybrid, 1996 · A History of th · re Avoidance, 2019 · A History of th · Board, 2019 · A History of th · Board, 2019 · Current Population Survey: Annual Demographic File, 1994, 1995

ABBASI ET AL., 2021
AHMADIAN ET AL., 2020
BACKURS ET AL., 2019
BERA ET AL., 2019
BERK ET AL., 2017
BEUTEL ET AL., 2017
BRUBACH ET AL., 2020
BRUBACH ET AL., 2020
CALAUZÈNES ET AL., 2019
CHIERICHETTI ET AL., 2017

Figure 5.11: Related paper list

The list shows the surname of the first author of the paper (if there are more than one author) and its year of publication.
Every displayed paper is a button which open a detail page of the paper when it is clicked. To focus the user attention when it is visualized, the background around the detail paper page is set to be darker.



Dataset description: This resource was developed to favour the study of automated age and gender identification from images of faces. Photos were sourced from Flickr albums, among the ones automatically uploaded from iPhon were manually labeled for age, available contextual information". T dataset, i.e. age and gender estima Fitzpatrick skin type

Adressa

Dataset description: This datase technology owned by Adresseav summarizes one week of traffic to during February 2017. The datas providing access timestamps and articles is also available, includin curators also worked on an extend one described here

Adult

Description	This dataset wa more
Last update	1996
Domain	economics
	data bias evaluati
	fair classification
	fair data summari
	fair regression
	preference-based

BAHARLOUEI ET AL., 2020

BAKKER ET AL., 2019

Related paper

Agarwal et al., 2018

A Reductions Approach to Fair Classification

Type	ProceedingsPaper
Title	A Reductions Approach to Fair Classification
Proceeding Title	Proceedings of the 35th International Conference on Machine Learning
URL	http://proceedings.mlr.press/v80/agarwal18a.html
Publication year	2018
Reference pages	60–69
Event	Proceedings of Machine Learning Research 2018
Author	Agarwal Alekh, Beygelzimer Alina, Dudik Miroslav, Langford John, Wallach Hanna
Publisher	
Related dataset	Adult, COMPAS, Dutch Census, Law School
Related task	fair classification
Internal ID	agarwal2018reductions

Figure 5.12: Related paper details

To conclude the discussion about the user experience, to help the Web application visitors, when there are a lot of resources to display, we applied a pagination of the resources.

How we manage all the available pages was very important, we take advantage of the `React Routes` which allows developer to redirect the searched URL to a specific components. In this way, we were able to redirect all the contents from the home page to the specific page of the Web application.

Since we talked about the home page, I report here the two version developed. The first one, available in fig. 5.13, has a dynamic background; the second one, available in fig. 5.14, has a static background and finally it has been considered more appropriate to this Web application.

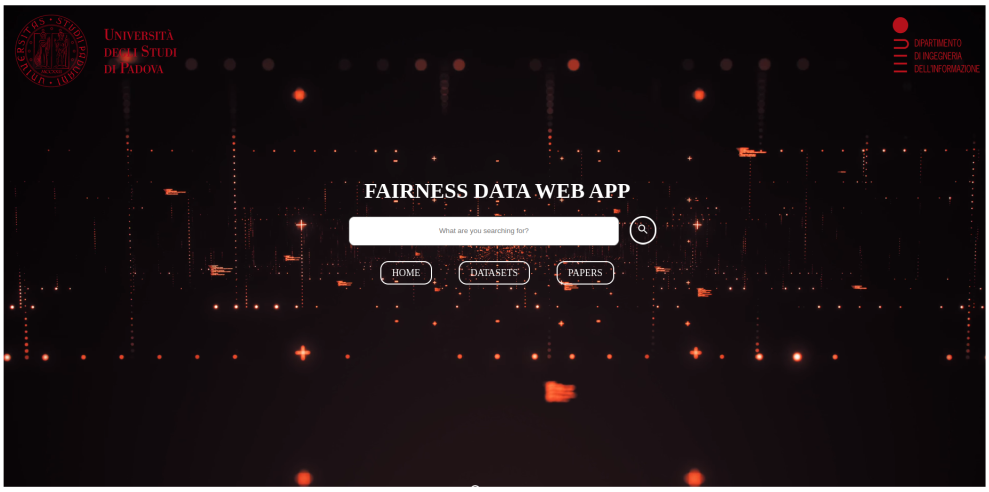
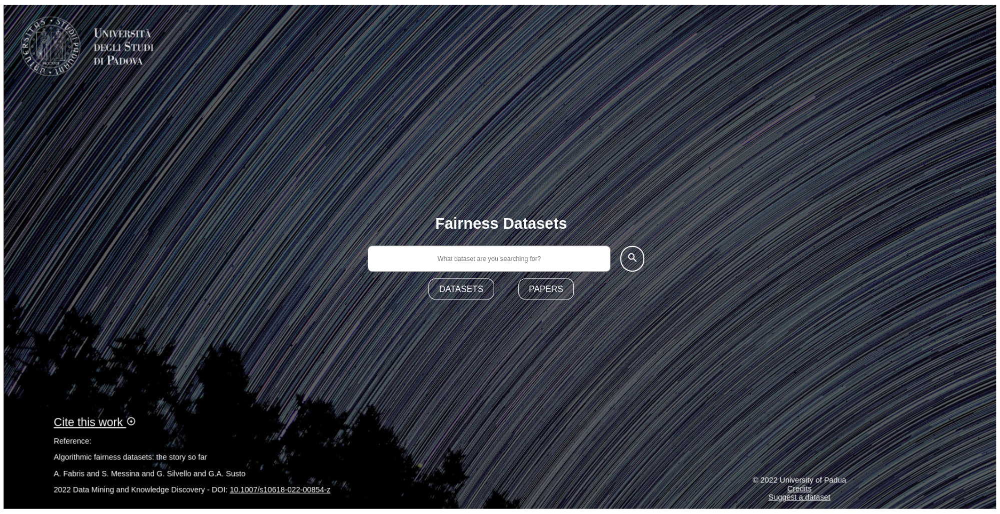


Figure 5.13: Home page - first version



Cite this work

Reference:

Algorithmic fairness datasets: the story so far

A. Fabris and S. Messina and G. Silvello and G.A. Susto

2022 Data Mining and Knowledge Discovery - DOI: 10.1007/s10618-022-00854-z

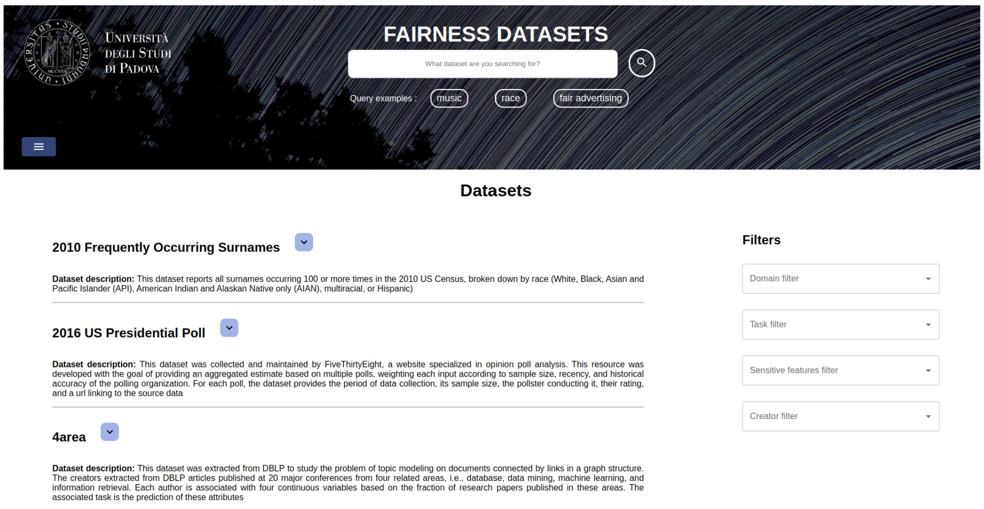
© 2022 University of Padua
Credits
Suggest a dataset

Figure 5.14: Home page - second version

As the reader can see, beyond the background aspects, we added some information. Starting form the left side of the footer, we report the information about the paper which is base of this thesis and the Web application. There is also a button to download the relative bibliographic file.
At the right side, the users can find the links to the credits of the Web application, a Web page dedicated to the suggestion of new datasets and some copyright information.

Then, every page has been designed to be in compliance with the home page style. In particular the header of every page reflects the home page background

and all the buttons have a color belonging the color palette of this image.
An example of Web page is the dataset one, which is reported in fig. 5.15



2010 Frequently Occurring Surnames

Dataset description: This dataset reports all surnames occurring 100 or more times in the 2010 US Census, broken down by race (White, Black, Asian and Pacific Islander (API), American Indian and Alaskan Native only (AIAN), multiracial, or Hispanic)

2016 US Presidential Poll

Dataset description: This dataset was collected and maintained by FiveThirtyEight, a website specialized in opinion poll analysis. This resource was developed with the goal of providing an aggregated estimate based on multiple polls, weighting each input according to sample size, recency, and historical accuracy of the polling organization. For each poll, the dataset provides the period of data collection, its sample size, the pollster conducting it, their rating, and a url linking to the source data

4area

Dataset description: This dataset was extracted from DBLP to study the problem of topic modeling on documents connected by links in a graph structure. The creators extracted from DBLP articles published at 20 major conferences from four related areas, i.e., database, data mining, machine learning, and information retrieval. Each author is associated with four continuous variables based on the fraction of research papers published in these areas. The associated task is the prediction of these attributes

Figure 5.15: Dataset page header

In the header the users can find some example of query which can be done in the search bar presents below the title. They are all clickable so that the users can click them and see the results.
Finally, there is also an "hamburger" menu which reports all the pages, as we can see in fig. 5.16.

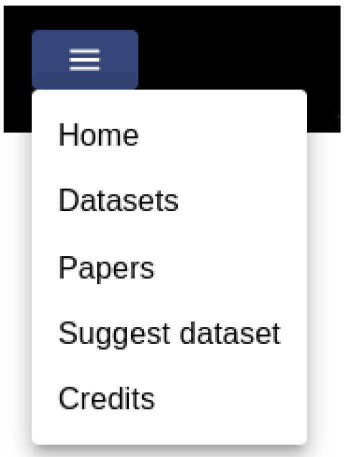
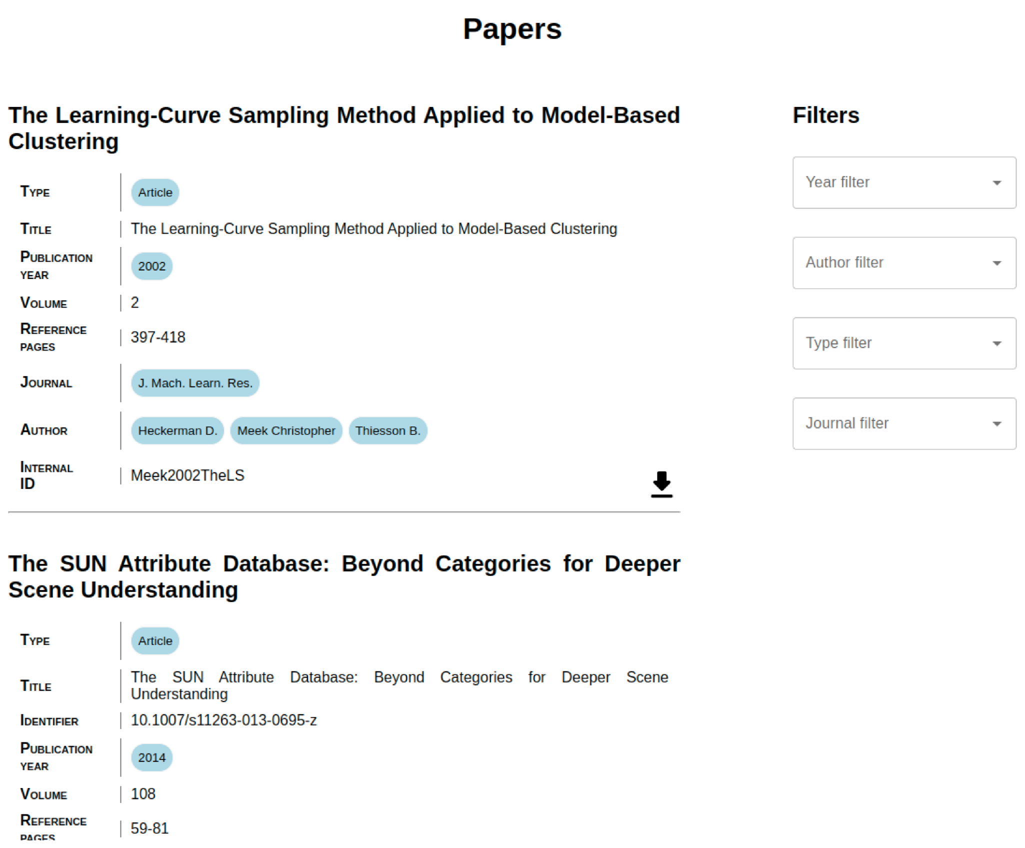


Figure 5.16: Header menu

In the *Papers* page we can find all the papers available into the Web application database. The Web page contains a sidebar with the filters, like the *Datasets* page, as we can see in fig. 5.17.

Also in this case, the filters can be applied both through the buttons available into the papers details and using the filters lists.

They work in the same way as previously explained. The only difference is given by the type of filters, in this case they can be applied to the year of publication, the authors, the type of resource and the journal.


Papers

The Learning-Curve Sampling Method Applied to Model-Based Clustering

Type	Article
Title	The Learning-Curve Sampling Method Applied to Model-Based Clustering
Publication year	2002
Volume	2
Reference pages	397-418
Journal	J. Mach. Learn. Res.
Author	Heckerman D. Meek Christopher Thiesson B.
Internal ID	Meek2002TheLS

The SUN Attribute Database: Beyond Categories for Deeper Scene Understanding

Type	Article
Title	The SUN Attribute Database: Beyond Categories for Deeper Scene Understanding
Identifier	10.1007/s11263-013-0695-z
Publication year	2014
Volume	108
Reference pages	59-81

Figure 5.17: Paper page

As the reader can see, the papers information displayed in this page and in the detail box into the *Datasets* page are the same, the only difference is related to the buttons used to apply the filters. Obviously in the *Datasets* page we can not apply filters to the papers but we can only visualize them.

Moreover, all the information can be downloaded using the button at the right bottom of every documents.

5.4.1 Sensitive feature filter

When all the faceted search were ready, we noticed that we had the necessity to add a new feature. The problem was about the sensitive features filter into the *Datasets* page. In practice we notice that the sensitive features can be grouped into a sort of "classes".
For example if a user wants to add a filter to the "*sex*" sensitive feature then probably should be added also the similar ones, for example the "*gender*" attribute which represents the same concept.
Like this example, there are other features which can be grouped together and thanks to Dott. Alessandro Fabris, the data curator, we were able to define the structure in algorithm 31. It is a dictionary object where the keys are the features name and the value represents the belonging class.

Algorithm 30: equalFeatures structure

```
const equalFeatures = {
    "1" : ["demographics of people featured in entities and their
     relations","textual references to people and their demographics","visual
     and textual references to gender"],
    "2" : ["family wealth","financial status"],
    "3" : ["gay-friendliness","sexual orientation"],
    "4" : ["gender","sex"],
    "5" : ["language","spoken language"],
    "6" : ["political affiliation","political leaning"],
    "7" : ["race","skin color","skin tone","skin type","ethnicity"]
}
```

Then we define another dictionary object, illustrated in algorithm 30, where there are all the below classes as key and the corresponding value is a list of all the features belonging that class.

Algorithm 31: featuresClass structure

```
"demographics of people featured in entities and their relations": "1",
"textual references to people and their demographics": "1",
"visual and textual references to gender": "1",
"family wealth": "2",
"financial status": "2",
"gay-friendliness": "3",
"sexual orientation": "3",
"gender": "4",
"sex": "4",
"language": "5",
"spoken language": "5",
"political affiliation": "6",
"political leaning": "6",
"race": "7",
"skin color": "7",
"skin tone": "7",
"skin type": "7",
"ethnicity": "7"
```

In this way, every time a new sensitive feature is added to the relative filter, the algorithm 32 is appointed to inspect the above structures to find similar classes that should be added contextually with the user selected one.
The first row represents a copy of the entire list of the selected sensitive features, then the algorithm retrieves the belonging class for every element of this list using the algorithm 31 structure.
The second `for-loop` has the objective to check if all the elements belonging the above mentioned classes have already been selected and on the contrary they are added to the new list of the chosen sensitive features.

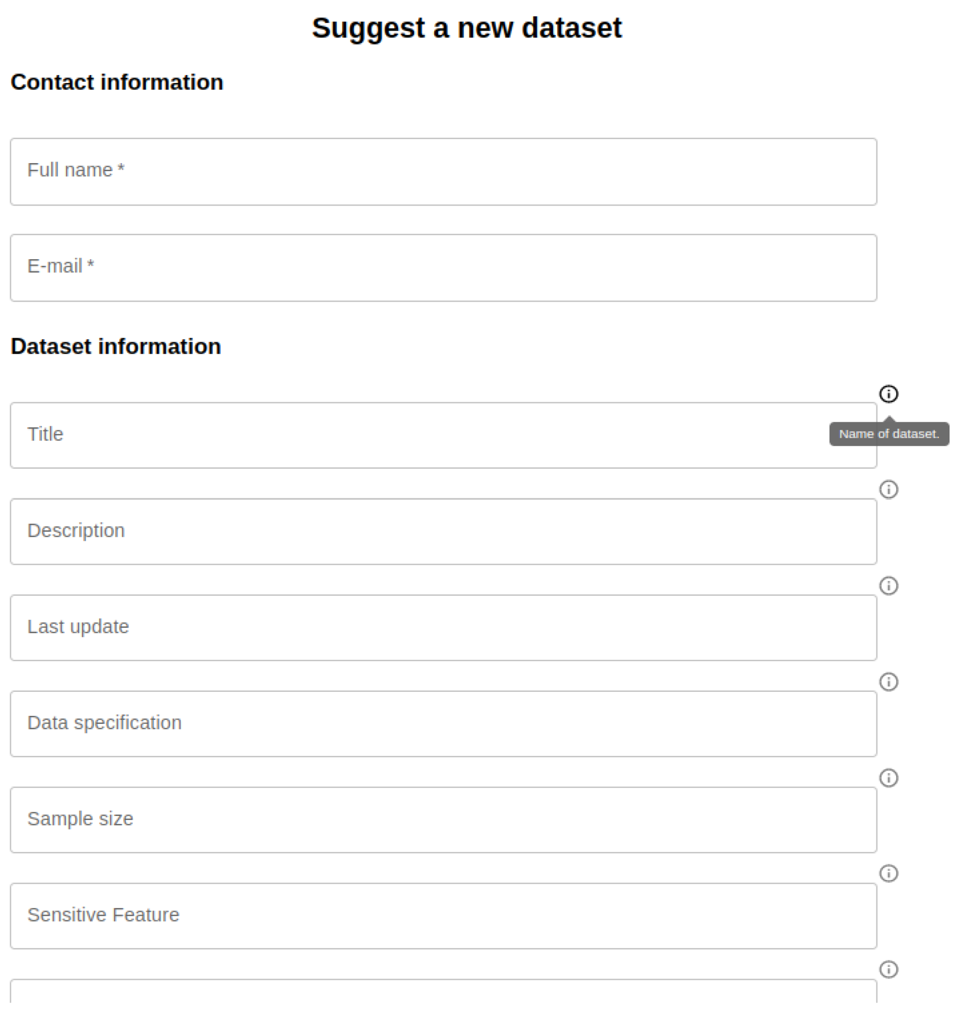
Algorithm 32: checkSensitive function

```
function checkSensitive (selectedSensitiveFeature,setSensitiveFeature){
let newSelectedSensitiveFeature = JSON.parse(
 JSON.stringify(selectedSensitiveFeature));
for feature in newSelectedSensitiveFeature do
    let featureClass = featuresClass[feature];
    for newFeature in equalFeatures[featureClass] do
        if newSelectedSensitiveFeature.indexOf(newFeature) === -1 then
            newSelectedSensitiveFeature.push(newFeature);
        end
    end
end
setSensitiveFeature(newSelectedSensitiveFeature);
}
```

The last problem to be solved is the strategy to use to remove the selected features. In fact, without the above algorithm there was not issue in this part but now, every time that there is a lacking feature from the classes, it will automatically add the missing one.
We can understand that in this way we are no more able to remove a single feature but we always need to use the "Reset" button which remove all the features simultaneously.
As it is an undesired limit, we solve this problem by storing the length of the list of selected sensitive features and then for every variation of this dimension we update this value but we call the `checkSensitive` function only if this value is greater that the previous one.

5.4.2 Form page

At the end of the Web application development we decided to give the opportunity to the user of suggesting new datasets. The Web page is visible in fig. 5.18.


Suggest a new dataset

Contact information

Full name *

E-mail *

Dataset information

Title

Name of dataset.

Description

Last update

Data specification

Sample size

Sensitive Feature

Figure 5.18: Suggest a dataset page

As we can see from the above screenshot, there are two fields dedicated to who are sending the information, they are required because in this way the data curator of the Web application can contact the sender to clarify doubts or to request more information.
The other fields are dedicate to the dataset information which are described in section 2.5. To help the Web application users there is an information button for every field, as we can see in the example in fig. 5.18. When it is clicked the suggestion of what this field should contain are showed.

There is only one field that can be modified: the one relative to the "Further information". As we can see in fig. 5.19, it has the addition option to add more rows. In this way a user can add as many rows as many related paper are available within the dataset. It definitively will help the data curator.

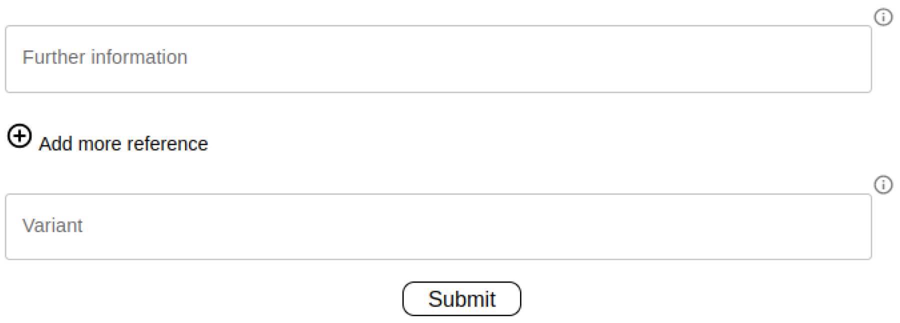
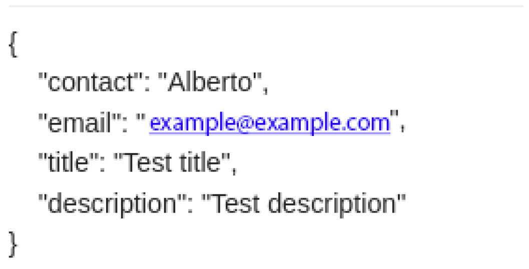


Figure 5.19: Further information field

Concerning how it was developed, it produces a lot of problems because this is the only case in the Web application where we need to send `POST` data. So, in this case we need to manage the `CSRF token` which is required to avoid the *cross site request forgery* attack.
The information are stored in a dictionary structure and when the submit button is clicked are sent to the back-end (in JSON format as reported in fig. 5.20) and it returns a integer value that represents the number of successful sent mails. In fact the back-end retrieve the information from the front-end and it sends a mail to the data curators. If the value is the expected one, then a successful message is displayed.
We decide to apply this strategy because in this way all the information uploaded into the Web application are verified.



```
{
    "contact": "Alberto",
    "email": "example@example.com",
    "title": "Test title",
    "description": "Test description"
}
```

Figure 5.20: E-mail JSON information

Conclusions

With this final chapter ends this thesis. I have some considerations about fairness algorithmic but in particular about the Web application development.
Concerning to the fairness algorithmic fields, I would like to say that it was an interesting fields to work about. I did not know it until Prof. G. Silvello proposed me the thesis and research training activity. In particular, it was interesting to discover that some algorithmic around our every days life are biased by data that should not be considered by them; we can think also that it is frightful from some point of view.
Regarding the pure development part was very interesting and I liked to work on a Web application as thesis argument. I think that it allowed me to increase my skills as developer but also as problem solver, in fact every part of the model, the back-end and the front-end have been carefully studied and I hope that it has emerged from this thesis.

Acknowledgements

This thesis represents the end of my master Degree university career. It was undoubtedly an important part of my student life.
Even if it represents the shortest academic path thought all my academic career it has been the most interesting and the most relevant. The reason is easy, I liked all the proposed and chosen courses and every one improved my skills which I will exploit during my working life.
During the Master Degree I was lucky enough to meet helpful and knowledgeable professors and to work with nice classmates.

I would like to thank my thesis co-Supervisor Dott. Alessandro Fabris and Dott. Fabio Giacchele who were always support giving in my thesis and development activity.
A special thank is to my thesis Supervisor, Professor Gianmaria Silvello who was always helpful and he managed the Web application project and the thesis activity in the best possible way.

I would like to thanks my entire family, my mother Patrizia, my father Ennio and my bro Lorenzo. They accompanied and supported all my student career and I think that I was able to achieve my objective also thaks to them.

A thanks to Giulia who has been by my side in every moment of this Master Degree career and more.

References

P. Archer. Data Catalog Vocabulary (DCAT) (W3C Recommendation), 2014. URL https://www.w3.org/TR/vocab-dcat/.

J. Bandy and N. Vincent. Addressing "documentation debt" in machine learning research: A retrospective datasheet for bookcorpus, 2021. URL https://arxiv.org/abs/2105.05241.

A. Banks and E. Porcello. *Learning React: Modern Patterns for Developing React Apps*. O'Reilly Media, 2020. ISBN 9781492051695. URL https://books.google.it/books?id=tDjrDwAAQBAJ.

E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21, page 610–623, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383097. doi: 10.1145/3442188.3445922. URL https://doi.org/10.1145/3442188.3445922.

P. Ciccarese and S. Peroni. Essential FRBR in OWL2 DL Ontology (FRBR), 2018. URL http://purl.org/spar/frbr.

DCMI Usage Board. DCMI metadata terms. DCMI recommendation, DublinCore, 2006. URL http://dublincore.org/documents/2006/12/18/dcmi-terms/.

A. Fabris, S. Messina, G. Silvello, and G. A. Susto. Algorithmic fairness datasets: the story so far. *Data Mining and Knowledge Discovery*, 36(6):2074–2152,

2022. doi: 10.1007/s10618-022-00854-z. URL https://doi.org/10.1007%2Fs10618-022-00854-z.

C. Garbin, P. Rajpurkar, J. Irvin, M. P. Lungren, and O. Marques. Structured dataset documentation: a datasheet for chexpert, 2021.

T. Gebru, J. Morgenstern, B. Vecchione, J. W. Vaughan, H. Wallach, H. Daumé III, and K. Crawford. Datasheets for datasets. *arXiv preprint arXiv:1803.09010*, 2018.

U. Grömping. South German Credit Data: Correcting a Widely Used Data Set. Report. Technical report, Beuth University of Applied Sciences Berlin, 2019. URL http://www1.beuth-hochschule.de/FB_II/reports/Report-2019-004.pdf.

Y. He, K. Burghardt, and K. Lerman. A geometric solution to fair representations. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, AIES '20, page 279–285, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371100. doi: 10.1145/3375627.3375864. URL https://doi.org/10.1145/3375627.3375864.

S. Holland, A. Hosny, S. Newman, J. Joseph, and K. Chmielinski. The dataset nutrition label: A framework to drive higher data quality standards. *arXiv preprint arXiv:1805.03677*, 2018.

IBM Cloud Education. Rest-apis, 2021. URL https://www.ibm.com/cloud/learn/rest-apis.

Open Knowledge Foundation, 2009. URL https://opendatahandbook.org/guide/en/what-is-open-data/.

OUI and Adience. Data, 2014. URL https://talhassner.github.io/home/projects/Adience/Adience-data.html.

K. Peng, A. Mathur, and A. Narayanan. Mitigating dataset harms requires stewardship: Lessons from 1000 papers. *arXiv preprint arXiv:2108.02922*, 2021.

S. Peroni and D. M. Shotton. Fabio and cito: Ontologies for describing bibliographic resources and citations. *J. Web Semant.*, 17:33–43, 2012. doi: 10.1016/j.websem.2012.08.001. URL `https://doi.org/10.1016/j.websem.2012.08.001`.

V. U. Prabhu and A. Birhane. Large image datasets: A pyrrhic win for computer vision? *arXiv preprint arXiv:2006.16923*, pages 1536–1546, 2020.

M. Wang and W. Deng. Mitigating bias in face recognition using skewness-aware reinforcement learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9319–9328, 2020.

F. Yang, M. Cisse, and S. Koyejo. Fairness with overlapping groups; a probabilistic perspective. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 4067–4078. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/29c0605a3bab4229e46723f89cf59d83-Paper.pdf`.