Dataset Aggregator

# Prerequisites

## Background information

There are a series of documents that describe the scope, application and workflow of the application:

* **NIPN Integrated Data Repository**: This document can be found in the *annexes* folder at the top level of the documentation folder. The project started with the idea of creating a generic dataset repository that could be applied to any type of dataset, this document describes the initial idea.
* **SMART Survey Aggregator**: This document can also be found in the annexes folder. This document is a further version of the project in which the application focuses in aggregating [SMART surveys](https://smartmethodology.org/).

Both documents provide an idea of the use and function of the application. Although the second document focuses on a specific type of surveys, the principles, structures and workflow can be applied to any kind of dataset, making both documents a good base for understanding what the final application should do and how it should work.

## Database framework

[ArangoDB](https://www.arangodb.com/) is the database system upon which the application was designed. It is a multi-model database that implements a micro-services framework, [Foxx](https://docs.arangodb.com/3.3/Manual/Foxx/AtAGlance.html), which is used to implement the back-end services. You should become familiar with this database system, since the application relies upon it.

## Structure

The application is implemented with two main components: the *back-end* and the *front-end* as a web-based application.

The back end takes care of storing and serving all the data, it is implemented using the ArangoDB Foxx micro-services framework which is executed in the database itself. The goal of these services is to provide a high-level interface to data for the front-end.

The front-end has not been developed yet, the plan was to create it using [React](https://reactjs.org/), since this framework is well adapted for creating independent components that can be assembled into a working application. The other reason is that both the Foxx framework and React use JavaScript as their main language, which makes for a consistent development environment. The front-end could be developed also using other frameworks and languages, such as PHP, for instance, it should be the decision of the front-end developers to decide what makes the best choice.

# Short summary

This is a very short summary of what the application should do and how it is structured, you can consider it as the summary of the two background documents cited in the beginning and serves the purpose of laying down the principles that apply to the rest of this document.

The application should implement a repository or searchable archive of studies, their documents and the data that they represent.



The goal of this system is to provide access to the studies, their annex documents and to the raw data by allowing users to query the studies metadata and the raw data variables, retrieving two sets of results: the studies and a selection of raw data that satisfies the query conditions.

The *study* represents a project, survey, or other types of data collection activities that can be considered as an independent entity. The study is tagged by metadata that records when it occurred, which regions it studied, which institutions were involved and all the other information that is necessary to document the conditions, methods and scope of the project.

All annex documents - in the case of surveys these could be the questionnaires, reports and original datasets – should be uploaded and linked to the study, so that it can be considered as an archive of all its information.

Studies will have one or more datasets that represent their data. The dataset exists in two forms: as the original file – Excel, Stata, SPSS, etc. – which will be uploaded as an annex file of the study, and as the data it contains that we will call *raw data* here. Raw data will be stored in a collection that corresponds to the data domain of the dataset. If we use SMART surveys as the example, *all* the SMART survey raw data will be stored in a *single data collection* which represents the *domain* of *SMART surveys*. The goal of this structuring is to allow searching *all* SMART survey data and retrieving aggregated data selections that can be downloaded by users to perform further data analysis or summaries. Data collections can be organised in ontologies to allow more than one data domain, for instance you could have a census collection that holds the data from all census datasets, this structuring allows the system to handle different types of data[[1]](#footnote-1).

The central component of the system is the data dictionary, its role is to provide the definition and documentation for all the fields of all records stored in the database. When importing raw data from a dataset, the first function to perform is to harmonise the variables of the raw data with the data dictionary, so that each column of data in the raw data table corresponds to a *descriptor* of the data dictionary[[2]](#footnote-2).

Once studies have been registered, their annex documents uploaded, and their data harmonised with the data dictionary, it becomes possible to perform a set of queries whose results will be a selection of studies and a selection of raw data. These two sets of results could be presented in two separate panes. The studies could be chosen and downloaded as a zipped archive in which the study metadata record could be provided as an Excel document along with all the other annex documents. The raw data could also be downloaded as a CSV file or a file of some other format. As the application matures, statistical functions could be applied to the raw data selections in the back-end to provide further functionality and features to the application users.

# Database structure

The database contains a series of predefined collections (tables in the traditional relational nomenclature), each serving a specific purpose:

* descriptors: This collection contains the variables definitions, each record represents a variable, its type, label, description and all the other information that is necessary to document and validate data associated with that variable.
* terms: A term is an item that has a code and a series of descriptive properties in several languages whose code is used as a reference. Terms are organised as ontologies (tree or graph structures) and implement all the relational structures of the data dictionary, such as controlled vocabularies (or enumerations), forms, ontologies, etc.
* schemas: This collection contains all the links between the data dictionary elements, parent-child relationships in controlled vocabularies and forms, as well as relationships between terms and their instances are recorded in this collection.
* users: This collection contains all the user records.
* hierarchy: This collection contains all the edge documents that relate users with other components of the data dictionary, for instance, such as the relationship between a user and its manager.
* studies: This collection contains the studies metadata records.
* annexes: This collection contains the study annex document records[[3]](#footnote-3).
* toponyms: This collection contains a set of toponyms which are related to terms defining geographic units. Toponyms are not needed by the application but represent a multilingual repository of geographic locations and administrative units, along with their nomenclature.
* shapes: This collection contains a set of GeoJSON shapes related to toponyms.
* edges[[4]](#footnote-4): This collection contains the relationships between toponyms, shapes and terms.
* errors: This collection contains error type definitions. The elements of this collection can be compared to the JavaScript Error object name.
* messages: This collection contains errors and other messages, the errors can be compared to the JavaScript Error message property, while the other entries are used to store the descriptions of the services in several languages.
* settings: This collection contains the settings for the application, it currently contains one entry that indicates the status of the application.
* sessions: This collection contains session data, it can be used by the front-end to store session user specific states.
* logs: This collection contains the log of the services, it records what services were called by which user and the user status at the call time.
* groups: This collection was supposed to contain user groups. User groups are not implemented, although they are earmarked in the code.
* smart: This collection should contain the SMART survey raw data as SMART datasets are harmonised and added to the database.

1. The idea is that each data domain represents a compatible set of data. While the use of a NoSQL database would allow you to store all types of data into a single collection, when the number of records grows, it is advisable to aggregate similar data structures together to allow specific indexing strategies. [↑](#footnote-ref-1)
2. The process of harmonizing data with the data dictionary has been described extensively in the *SMART Survey Aggregator* background document. [↑](#footnote-ref-2)
3. The management of study annex documents should be performed by the front-end component, ArangoDB does not support a viable binary format that would allow to store physical binary files as records – as MongoDB [GridFS](https://docs.mongodb.com/manual/core/gridfs/) allows. [↑](#footnote-ref-3)
4. ArangoDB has two types of collection: document and edge. This collection is one of the edge collections used in the database, the name may be misleading, this collection only contains edge documents related to toponyms. [↑](#footnote-ref-4)