UNIVERSITY
OF TRENTO - Italy

Dipartimento di Ingegneria e Scienza dell'Informazione

KnowDive Group

DS Catalog Ecosystem

Development and Maintenance

Date	Reference persons
28-02-2023. v1	Simone Bocca

Trento, Italy

Index

Revision History

Revision	Date	Author	Description of Changes
0	2023.02.28	Simone Bocca	Document created

1. Introduction

The objective of this document is to provide the instruction to develop and maintain a data catalog, built by using the JKAN framework. The data catalog created following these guidelines will be compatible with the DataScientia (DS) foundation community. The intended audience for this document are both developers and reference people for the contents handled within the catalog that will be generated.

The document is structured as follows:

- Section 2 introduces the JKAN framework
- Section 3 explains the main reasons why JKAN has been adopted for the DS catalog ecosystem. In this section is reported also the description of the DS data sharing approach.
- Section 4 describes the instructions to be followed in order to create a new JKAN data catalog, integrated in the DS catalog ecosystem.
- Section 5 describes the structure of the JKAN catalog's repository, thus allowing for a better understanding of that.
- Section 6 describes the instructions to be followed in order to customize and develop locally new features for the DS JKAN catalog.

2. JKAN

The JKAN framework provides a lightweight solution to setup a data catalog based on a github repository. JKAN is backend-free, the business logic, maintained in the reference github repository, is developed in node js and is executed directly from the repository, to report then the outcome graphically exploiting the github pages.

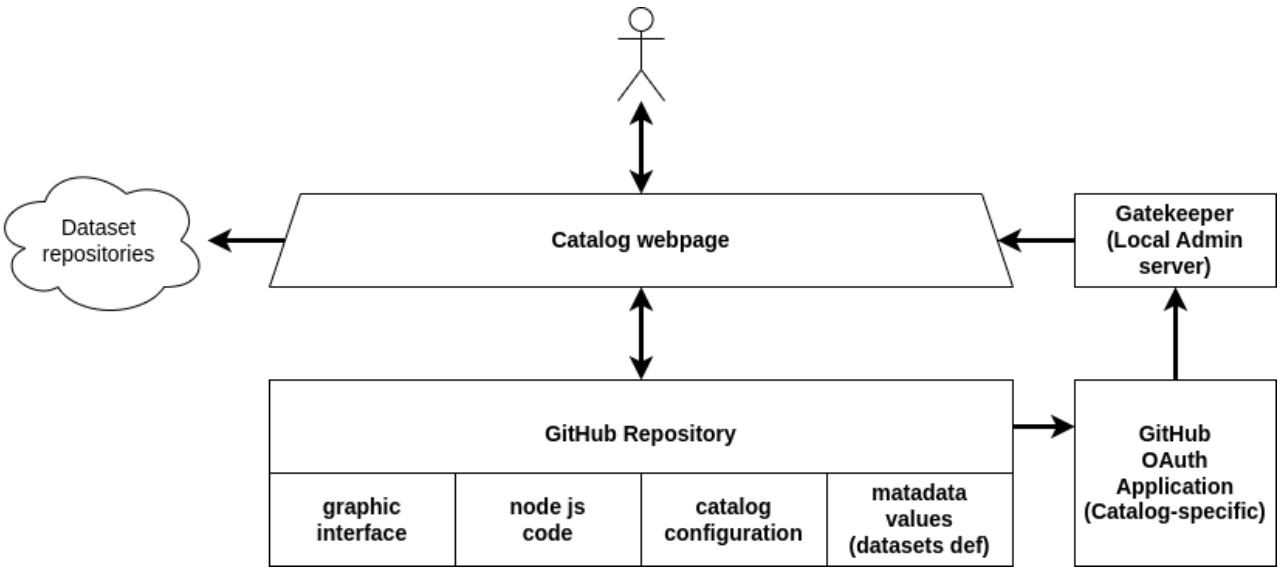

Fig.1 - JKAN catalog Architecture

Once the reference github repository is set up, the catalog can go online immediately. It is accessible through the link provided by its github webpage. The development and maintenance of the catalog can be done completely on the repository, all the modifications performed there will be directly available online on the webpage.

Being based on github, the JKAN catalog can be updated only by users who have access to the relative repository. The approach provided by DataScientia, is to collect all the JKAN catalog repository under a single DataScintia github organization (DS org, from now on). The DS org collects the repositories serving all the catalogs (those in control of the foundation itself). For this reason, if a reference person wants to become administrator of a new catalog (or even for an existing one), as first thing , she has to ask to be added (through her github account) as a collaborator to the DS org.

2.1 JKAN authentication layer

JKAN offers for the catalogs an authentication layer exploiting github OAuth application, that is used by the catalog web page as well as by the JKAN business logic in the repository, for the authentication of the administrator of the catalog. Each catalog must have its own OAuth App registered in the DS org. The authentication provided by JKAN allows the catalog administrator to use the catalog user interface to register itself to the github OAuth App created for the specific catalog. If the administrator will have the correct github credentials (thus, she already has been added as a collaborator in the DS org), she will be able to become administrator and she will have, from now on, the rights to manage the catalog as well as add datasets, organizations, as well as do modification to the catalog itself.

More specifically, the github OAuth App is created with a clientID and a secret, used to define the administrator credentials for the relative catalog. The JKAN authentication layer works together with a local authentication server called Gatekeeper, that the administrator has to install in his local machine (or centralized if she has the possibility to do that), and to run in order to allow the access to the catalog with administrator rights (see below for instructions on how to set up the Gatekeeper server). The Gatekeeper server needs to be configured with the same credentials defined for the OAuth App created for the relative catalog. Then, the authentication will be successfully executed.

This authentication approach completely isolates the access for the administrator, which can in this way manage the access to the catalog through her own local server, thus reducing the risk of security issues.

Currently the authentication for users (non administrators) that want to upload and/or download resources using the catalog, is not provided. Nevertheless, it is already considered as future implementation through a dedicated layer that will be developed differently for each catalog (or even for the access to specific kinds of resources) following the needs that such catalogs require.

3. The DS catalog ecosystem

The choice to use the JKAN framework to leverage the DS catalog ecosystem, comes for two different reasons, described in the two following subsections, respectively.

3.1 “Repository-centric” data sharing approach

The first reason comes from the data sharing approach adopted by DS. To briefly describe such an idea, we can say that the most famous open data environments (like one of the most known https://data.europa.eu/it) are “catalog-centric”, meaning by this that who wants to share data in such environments, needs to adopt the same catalog technology (in the data.europe.eu example, it is CKAN), with the objective to link together, through that technology, the different catalogs. This means that those who have data (maybe already stored in personal repositories) but are not able

to adopt such a technology (lack of resources, technology, or other reasons), will not be connected to the rest of the environment. In other words, the catalogs, through their technology, define how the data can be shared.

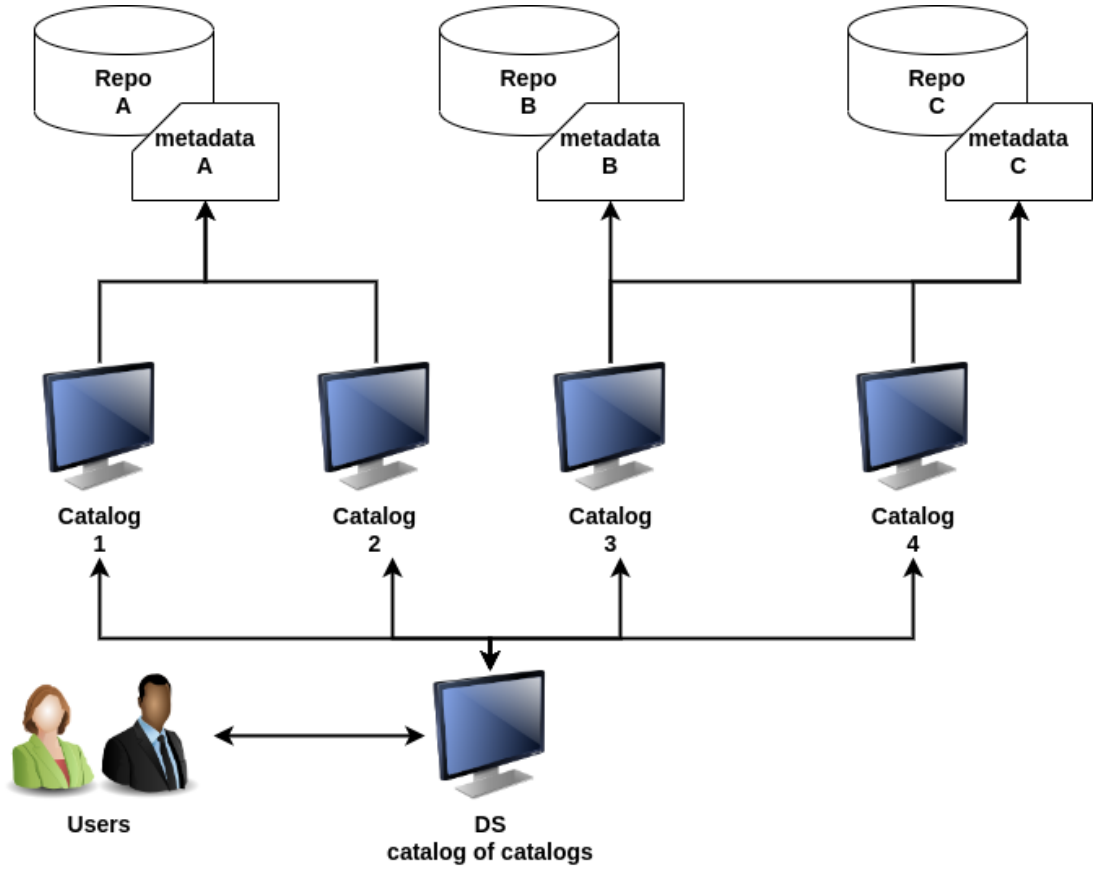

Fig.2 - Repository-centric approach

DS, instead, proposes a "repository-centric" approach (depicted in Fig. 2), where the repositories define how to share data in the DS community. To this end, the repository, which contains the real resources to be shared, defines a set of metadata for such resources, by using specific guidelines provided by the DS community. The metadata is the fingerprint of a repository into the DS community. Then each catalog, independently by how it is implemented, that wants to access the data that a repository wants to share, will only need to support the metadata structure shared by the repository. If a user and/or an organization is not able to produce a catalog (for lack of resources, technology, or other reasons), DS provides a template JKAN catalog, easy to customize, which can be quickly deployed online thanks to the JKAN framework (notice that no new server machine, or any other kind o HW is required for the new catalog).

The key role in the catalog ecosystem is played by the DS "catalog of catalogs". It is a catalog linking all the new catalog that want to be part of the DS community. Any users looking for any kind of resource shared in the community, will exploit the DS "catalog of catalogs" in order to get what they are interested in. Moreover, It is important to mention that, DS maintains some centralized catalog (LiveLanguage and LiveKnowledge, and for its first version LivePeople too) which are nonetheless linked by the "catalog of catalogs".

3.1 Catalog development and maintenance

The second reason for adopting the JKAN framework, is focused on the development and

maintenance of the catalogs ecosystem, and more precisely, on the effort that such operations require. In fact, the JKAN framework, with respect to other technologies like CKAN, is backend-free and doesn't require specific HW to be deployed. Moreover, JKAN allows the creation of lightweight catalogs focused mostly on the single feature of describing the resources they want to share. This implicitly means that all the additional features (like generation and composition of resources on-the-fly, specific access control policies, more complicated data search features, and others) are added as custom services to the catalog. In this way the catalog itself remains very simple and, thanks to the JKAN framework, can be directly deployed online without maintaining different versions for development and production mode. In practice, this last point can be done by keeping the development version of the catalog in the administrator local machine. Nevertheless, once the JKAN catalog's repository is updated with new features, they go directly online in production, as opposed to other more complicated technologies (because of their most powerful features, like CKAN) requiring more steps to upgrade new features from development to production.

To conclude this section we can briefly summarize that JKAN is a suitable framework for the DS catalog ecosystem, thanks to its simplicity in the development and maintenance of catalogs. With this lightweight and adaptable approach DS leaves to each single administrator the possibility to customize as much as she wants her own catalog, but at the same time, providing a quick, smart and low cost solution supporting the DS data sharing approach.

4. Create a new Catalog

This section reports the instructions that a catalog administrator ("the administrator" from now on) has to follow in order to create a new base JKAN catalog based on the template provided by the DS org.

Requirements:

- The administrator has to be a DS org collaborator on github. If that is not the case, She can ask the DS technical support, to be added as collaborator of the DS org.
- [Required for catalog local development] Install locally "node js" and "npm" (documentation). You can verify the installation and version of them by running in terminal the following commands:
    - # node - -version
    - # npm - -version

Instructions:

1. The administrator has to provide, to the DS tech support, the information for the creation of the new repository, the one which will serve the new catalog. Such information must contain as minimal requirement, the name of the repository, that will be also the name of the catalog that will be accessible online.
2. With the information provided in the step above the DS tech support wil create the new repository, for which the administrator will have the rights for any modification she needs to do. The new catalog's repository is created starting from the DS catalog template. The catalog template can be visualized graphically from this link.
3. The DS tech support will also create the github OAuth App required for the catalog

administrator authentication. Once that will be created, the "clientID" and "secret" will be shared with the administrator in order to configure the Gatekeeper server properly (see the instruction below for this).

4. The administrator has to do some preliminary modifications in the _config.yml file available in the repository created in the step above. In the _config.yml file the following fields need to be modified:

    a. title: the title of the catalog webpage.

    b. greeting: the name of the catalog, that will appear in the catalog landing page.

    c. description: the description of the catalog, that will appear in the catalog landing page.

    d. baseurl: the base catalog URL, that must be composed as follow "/<name-of-the-repository>

    e. github_client_id: the "clientID" provided by the DS tech support.

    f. gatekeeper_host: the host and port in which the Gatekeeper authentication server is running. See in the next step the instructions to set up the Gatekeeper server locally. If the server is set up locally, the host and port are usually: http://localhost:9999 (these values can be changed)

5. Set up the Gatekeeper Authentication server locally:

    a. clone locally the following repository: https://github.com/prose/gatekeeper

    b. In the folder /gatekeeper, modify the config.json file, by changing the following fields with the following values:

        i. Oauth_client_id : the "clientID" provided by the DS tech support.

        ii. Oauth_client_secret : the "secret" provided by the DS tech support.

    c. In the folder /gatekeeper, open a terminal and execute the following command:

        i. # node index.js .

    d. The Gatekeeper Authentication server is now running in your local machine at http://localhost:9999 .

5. JKAN catalog's repository structure

In this section we describe the structure of the JKAN template repository that DS provides to set up a new catalog. The information provided in this section will help the catalog administrator during the configuration and customization of the JKAN catalog. Below we describe the most important elements that can be found in the repository.

- _data : this directory contain the yml configuration files used to define the metadata of the resources handled by the catalog (within the "schemas" directory), as well as the categories that can be used to categorize the resources in the catalogs, and the custom services which can accessed to the catalog.

- _datasets : this directory stores the markdown file describing (following a precise metadata schema) each dataset uploaded in the catalog.
- _includes : this directory contains the website pages (html files) that are shown in the catalog web page.
- _organizations : similar to the _datasets directory, this one contains the markdown files defining the organization that have uploaded (the owner usually) datasets in the catalog.
- css : it contains the catalog main.css style file.
- img : it contains the images used in the catalog website.
- scripts : this directory contains the business logic of the JKAN catalog. More in detail, in the "src" subdirectory we can find the javascript code divided in the different files, containing different specific functions. While in the "dist" subdirectory we can find the bundle.js file that is the file created after building the code in the src subdirectory. The bundle.js file is the only file that is actually executed in order to apply the JKAN business logic to the catalog. This means that all the modification performed in the source code (src subdirectory) needs to be reported in the bundle.js file too, this happens automatically running the build process (see the next section for the instruction about how to make a new build locally).
- _config.yml : this is the main config file of the JAKN catalog.
- datasets.json : this file defines the structure of each dataset that can be searched within the catalog. This file is used for the search functionality mostly, this means that, in order to be searchable, the datasets in the catalog need to respect the structure in this file, which can be updated in order to add new search criteria.
- Index.html : this is the main html file of the catalog webpage. The visualization of the catalog website, as well as its navigation, starts from this file.

The remaining files in the repository are not described because they are out of scope for this document. Nevertheless, you can contact the DS tech support to get more detailed information about the JKAN catalog repository.

6. Customize and update a catalog

In this section we report the instructions to be followed in order to customize an already existing catalog (or even a newly created one). The idea is to provide to the administrator, the possibility to develop locally (in development) new features for her catalog, and then push such modifications in the catalog's repository in order to be available online (in production). The catalog web application (website + business logic) is developed in node js, therefore, the administrator needs to be sure to have node and npm properly installed in her local machine (see Requirements in Section 4).

1. Clone the JKAN catalog repository created and provided by the DS tech support for your new catalog (or clone the existing catalog repository, if that is not a new one).
2. Through the Webpack integrated tool, install the JKAN framework components required to build the catalog project locally. To this end, open a terminal in the /jkan directory and

execute the following commands:

a. # npm install webpack

b. # npm run build

i. This last command will verify that you are able to build locally the catalog source code.

3. Install the Jekyll web app server that will locally execute your catalog.

a. Inside the /jkan directory open a terminal and run the following:

i. # sudo gem install jekyll bundler

ii. Some conflicts may appear during the execution of this command. Follow the instruction you will receive by the terminal directly, to solve such conflicts.

b. Inside the /jkan directory open a terminal and run the following commands:

i. # sudo bundle install

c. Then run the server executing, within a terminal, in the /jkan directory the following command:

i. # sudo bundle exec jekyll serve

d. If everything worked well the Jekyll server is running at (localhost) http://127.0.0.1:4000/jkan/

Once the instructions above have been properly executed, the administrator can execute the following step every time she wants to make modifications locally, and then push such modifications in the relative catalog repository.

1. Do the modifications/changes/updates needed in the catalog source code.

2. Make a new build (thus generating a new bundle.js file that will be executed online) by running the following command, in a termina, inside the /jkan directory:

a. # npm run build

3. Run the Jekyll server (see the step 3.c), or close (Ctrl+c in the running terminal) and restart it if it was already running.

4. Open the http://127.0.0.1:4000/jkan/ where the catalog will be shown with the last updates performed.

5. Verify that the expected behavior is correct, then commit and push the modification on the catalog repository by the ordinary git commit-push process.

6.1 Update metadata and search functionality

One of the most important catalog functionalities (if not the most important) is the research of datasets based on the metadata defined for such datasets. In this section we describe the instructions to be followed in order to update the metadata structure of the DS JKAN catalog, and how to update the catalog search functionality with the objective to enable the search over new (custom) metadata. To this end the administrator has to execute locally the instructions below.

1. Add the new metadata by adding new element in the file: _data/schemas/default.yml
2. Update the file datasets.json (see Section 5) in order to upgrade the dataset schema used by the search functionality.
3. Update, as follows, the function "_createSearchFunction (datasets)" in the file: scripts/src/components/datasets-list.js
    a. Modify the second line of the function, by adding the name of the new metadata (those to be used in the search functionality) in the "const keys = ['title', 'notes', 'maintainer', 'tags']"
4. Make a new build of the catalog source code, in order to produce a new bundle.js file. Push the last build outcome in the catalog's repository in order to put the update online.