MISTRAL
Meteo Italian Supercomputing Portal

Meteo Italian SupercompuTing poRtAL

DELIVERABLE 3.2

Functional and non functional requirements for open data portal front-end and back-end

Deliverable Lead:	Arpa Emilia Romagna
Deliverable due date	31/05/2019
Version	FINAL
Status	V1

This project has received funding from *European Commission*
Connecting Europe Facility 2014-2020
AGREEMENT No INEA/CEF/ICT/A2017/1567101

Document Control Page

Title	D 3.2 - Functional and non functional requirements for open data portal front-end and back-end
Creator	Arpa Emilia Romagna
Publisher	Mistral Consortium
Contributors	Davide Cesari (ARPAE), Emanuele Di Giacomo (ARPAE), Luca Delli Passeri (DPC)
Type	Report
Language	en-GB
Rights	copyright "Mistral Consortium"
Audience	☒ public ☐ restricted
Requested deadline	31/05/2019

Sommario

Executive Summary 4
1. Architecture of the Mistral system 5
1.1. Data portal front-end 5
1.2. Data portal back-end 5
1.3. Archiving and postprocessing engine 5
1.4. Embedded applications 5
2. Common components of the data portal 6
2.1 Global configuration 6
3. Functional and non functional requirements for open data portal front-end 6
3.1. Per-user configuration 6
3.2. File management form 7
3.3. Archive dataset query form 7
3.3.1. Scheduling 8
3.4. Real-time queue creation form 9
3.5. Administration form 9
3.6. Multi platform command-line client 10
4. Functional and non functional requirements for open data portal back-end 10
4.1 Definition of data sources and data access 10
4.2 Dispatch process 11
4.3 Scheduler 11
4.4 Event signaling 11
4.5 Interaction with the front-end 12
5 Archiving and postprocessing engine 12
5.1. Per-dataset configuration 12
5.2. Archive queries and postprocessing 13
5.2.1. Immediate interactive queries 13
5.2.2. Scheduled queries 13
5.3. Other queries 13
5.4. Dataset access authorization 14
6. Embedded applications 14
7. Reuse of existing software 14
8. Need for HPC resources in the Mistral platform 15

Executive Summary

This document defines the requirements, both from the end-user perspective (front-end) and from the administrator's perspective (back-end), for the Mistral Open Data Portal. It describes the interaction between the main components that are needed for satisfying the requirements, thus it may be considered as a proposal for an architecture of the system. The document defines also two more components, the archiving engine, which sits in the middle between front-end and back-end, and the embedded applications, which may be regarded as external applications of the Mistral platform, but at the same time hosted on the platform itself.

The document contains also a review of existing software useful for reuse in the Mistral platform and a brief analysis justifying the need for HPC resources for a timely operation of the platform with the data that are going to be feeded to it.

1. Architecture of the Mistral system

The Mistral system should include the following components:

1.1. Data portal front-end

The data portal front-end is the interface of the Mistral system exposed to the user. It should be implemented as a set of web services, with optional authentication, driven by interactive user input. It should interact with the back-end mainly for enabling scheduling of queries and for creating real-time queues.

1.2. Data portal back-end

The back-end is the part responsible for accessing data sources, feeding the dataset archive and the real-time queues, executing scheduled user queries, disseminating data-availabilty events, enabling scheduling of queries on user request, creating real-time queues on user request. It should be driven by external data sources and by the query scheduler.

1.3. Archiving and postprocessing engine

The archiving and postprocessing engine stands in the middle between front-end and back-end; the back-end feeds the archive with new data, while both, front-end and back-end, have the ability to perform archive queries and graphical and numerical postprocessing pipelines. The postprocessing may rely on an HPC system for parallelising and speeding-up the execution. Some dataset queries and postprocessing may require the temporary storage of the generated output in a cache space to be served to the user by the front-end component.

1.4. Embedded applications

Two user-perspective applications are foreseen within the project, the Multi-model ensemble prediction system of Arpa Piemonte and the Italy flash flood system of ECMWF.

2. Common components of the data portal

2.1 Global configuration

The system should have a global, file-based, configuration, defining such aspects as:

- the portal is run on a public server with all the user authorisation levels or bound to a local address on an unprivileged port and without any user authorisation
- how to interact with the archiving and postprocessing engine (e.g. path of a script to be called for performing queries and postprocessing)
- the processes with high computational load should be run on an HPC system through a batch queue manager or as local processes (or alternatively: path of a custom script dealing with the scheduling aspects of HPC processes)
- location of user database/file
- location of user authentication database/file
- location of storage areas for web cache and cloud storage
- any other business.

3. Functional and non functional requirements for open data portal front-end

3.1. Per-user configuration

There should be a per-user configuration defining at least the following aspects (in addition to standard user information):

- lists of different categories of user permissions in the form <key>=<value1>[,<value2>[,<value3>...]], e.g.
  - AUTH_LEVEL=[ADMINISTRATOR|SELF_REGISTERED|AUTHORISED_BY_ADMIN]
  - DATA_ACCESS={REALTIME,DEFERRED,HISTORICAL}
  - DATA_LICENSE={PUBLIC,PERSONAL,INSTITUTIONAL,COMMERCIAL,DERIVED}
  - DATASETS_ALLOWED={...}
- cloud space user quota on the server
- maximum number of scheduled queries
- maximum number of active real-time queues
- other user limits
- any other business.

Users with administrator role have special privileges (e.g. can modify other users' configuration, etc.).

Users with a high enough authorisation level should have access to a personal, permanent storage space (order of tens/hundreds of MB per user) for storing at least the following categories of files:

- shapefiles (or equivalent georeferenced topological vector data)
- grib files (without associated data) to be used as templates for georeferenced grids or output grib coding
- layout configurations for graphical output generation (json?)
- definitions of queries/postprocessing and real-time queues (json?).

Users should also have access to a temporary storage space for holding the result of asyncronous data queries. The size of these query results could be much larger than the permanent storage space indicated above. Suitable cleaning policies should be applied to this temporary storage, e.g. after successful download of the data, after a specified time, upon user request or as soon as the space is reclaimed by the same user.

3.2. File management form

The web interface should provide minimal tools for file management (upload, download, rename, share with other users). A set of public files of the same categories, managed by administrators, should be available to all users, including users not having their own cloud space quota. The form for defining the data query and processing/graphical pipeline should prompt the user for their own files and for public files in those form entries where a file is required, possibly filtering only the files relevant to that entry, e.g., where a shapefile is required, it is desirable that only shapefiles are proposed in the choice menu.

It has to be decided whether saved (possibly scheduled) queries and (possibly active) real-time queues created by the user should also appear as special (preferably read-only) entries in the cloud storage file hierachy.

3.3. Archive dataset query form

This is the web form exposed to the users for querying archive datasets in near-real-time and historical mode and for defining the data processing pipeline.

The form should have at least the following sections:

1. choice of dataset(s)
2. definition of dataset query keys (choice of metadata)
3. (optional) computation of derived variables
4. (optional) time computations (accumulation/averaging on the selected interval)
5. (optional) horizontal space computations (regridding or grid cutting or interpolation to sparse points)
6. (optional) extra computations (wind rotation flag, a.o.b.)
7. definition of the output type, (raw data for d/l, graphical file (or package of files) for d/l, graphical files to be viewed online), graphical layout configuration to be used

8. verify, execute, save or schedule periodically the resulting query.

The user should be initially prompted for opening a blank form or a previously saved form.

The dataset choice (1.) implicitly defines whether sparse point or gridded data are desired in input, no mix should be allowed.

According to user's choice, the query keys section of the form (2.) could be proposed in an advanced layout (analog to current arkiweb query) ora in a more simplified layout containing only the queries required by the typical use cases. User should have the chance to clone a subset of the query key form (e.g. product, level, possibly timerange) into another sub-form to be modified and put in "or" with the previous (to be explained).

The query keys section (2.) or time computations section (4.) could contain also an entry for specifying whether analysis mode or forecast mode is desired for time processing (to be explained).

Choices for query keys (2.) and for derived variables (3.) could be generated dynamically with the use of arkimet and libsim helper programs.

The form code should dynamically disable the entries that become non-relevant because of previous choices, e.g. if raw data download is chosen, the choices related to graphics should be disabled.

During the form filling process, or at the end of it, the user should be able to see on the page the resulting query syntax, together with the instructions on how to execute it in batch mode through the portal web api with a tool such as curl or wget or with the dedicated command-line client (see section "*Multi platform command-line client*"); the api syntax should be human-editable for minor modifications (e.g. change dataset, reference date, parameter).

The section 8., also depending on user's authorisation level, should provide the user with the following choices:

- verify the query, i.e. execute immediately the query in debug mode, giving the user an extended feedback on possible errors or inconsistencies (to be explained)
- execute the query immediately and save the result in the user's temporary storage space and, depending on the result type, prompt the user for downloading the data or redirect the user to the (temporary) address where data can be viewed online in graphical form
- save the query with a user-specified name for successive editing or execution or scheduling.
- save and schedule the query for execution with a desired interval periodicity equal to the dataset update periodicity or to a multiple of it and a displacement (if the user is authorised to scheduling)
- de-schedule the saved query if it is scheduled.

3.3.1. Scheduling

When the scheduling of a query is activated, the system should respond with information about the data access.

Once scheduled, the query should be executed automatically by the portal (back-end) according to the scheduling requests, adjusting the reference time of the query to the reference time of the current dataset update, and the query result should be stored in the user's temporary storage space. The user

(or the batch command-line client) should be informed about the availability of the file for download by means of a suitable method (repeated polling, push, other...).

A scheduled query becomes read-only, when opened in the query form the only option should be de-schedule, in the file manager, if relevant, it should be only possible to duplicate it in unscheduled form.

3.4. Real-time queue creation form

This is the web form exposed to the users for creating personal real-time communication queues. Tentative list of query form sections:

1. choice of dataset(s)
2. filter on data (is this applicable or it is enough to have on/off states?)
3. data semantics (exactly once/at most once)
4. create, save without creating, delete queue.

The available choices for data semantics may depend on the level of user authorisation, e.g. "exactly once" mode could be allowed only for authorised users, since it involves data buffering, reception acknowledge, automatic reconnection, etc. and, in general, a higher load on the system.
When queue is activated, the system should return the URI (or whatever syntax) of the queue for connection. The queue will aggregate all the information from the selected datasets, with an "exactly once" semantics, meaning that it is guaranteed that no duplicated data are sent and, in case of disconnection, at the successive reconnection there is no data loss, provided that the disconnection time is shorter than a threshold (how much?)

The user should be initially prompted for opening a blank form or a form relative to an active queue or to a saved but not activated queue.

This type of form should be reserved to users with a higher authorisation level, public users should have access only to a shared channel aggregating open data.

3.5. Administration form

A user configuration form should be available to administrators for managing the user database, providing basic checks for avoiding as much as possible the definition of inconsistent configurations.

Administrator should have access to a form for scheduling archive queries and postprocessing as for a regular user, but in this case the result of the scheduled postprocessing should be fed back to the

archive (see description of the back-end) rather than being proposed for download, thus the postprocessing becomes actually a preprocessing, (i.e. it is performed before rearchiving data). This will be used for producing derived data which are not available in the input datasets but which are going to be requested by many users, in order to reduce the load on the system.

3.6. Multi platform command-line client

A simple muti-platform application, preferably in the python language, should be implemented, allowing users to perform the main operations on the portal from the command line in a batch mode. It should implement at least the capabilities for performing an archive query to the Mistral portal using the web api and downloading the resulting data, and for connecting to an active real-time queue giving in output the data stream with a minimal possibility of filtering the data.

4. Functional and non functional requirements for open data portal back-end

The data portal back-end performs the tasks of accessing external data sources and converting data, dispatching data to archive datasets and to active real-time queues, performing planned data pre-processing and successive rearchiving of pre-processed data, signalling events to the corresponding real-time queues, executing interactive or scheduled queries and post-processing pipelines.

It should have the following main working threads:

- data access: the back-end should actively poll for new data and download them from the configured pull data sources or receive data from push data sources; at data reception this thread triggers the data dispatch process and the signaling process
- scheduling: scheduled user queries should be executed upon reception of the corresponding data-ready signals or at the scheduled solar time
- interactive response: wait for requests performed by the front-end and reply accordingly.

4.1 Definition of data sources and data access

The back-end should include the functionality for accessing a set of heterogeneous local or remote data sources. Since the ways to access data and to homogenize it in a format suitable for the portal archive may be extremely various, a solution could be to implement these functionalities in the form of ad-hoc plugins with which the back-end interacts in a predefined way, e.g.

- start data access, downloading, converting and returning data on output stream or as files as soon as available, the plugin will decide whether listening for new data on a socket or polling a remote data source at regular intervals or predefined times using a specific transport (e.g. ftp, scp) or monitoring a directory on a local filesystem
- interrupt data access
- inform (how?) the dispatch process when a milestone has been reached, e.g. all the data for a specific reference time of a model forecast have been provided.

Ideally, the data from each source, after retrieval and conversion, should allow the successive dispatch process to uniquely identify the destination dataset only from the metadata accompanying the data, without knowledge of which was the actual source of the data.

When the back-end is activated (how? together with the front-end/web server or?) the data access functionality should start working and feed data to the dispatch process.

When the portal is run in stand-alone mode, a default local data access plugin should be activated, this plugin will monitor one or more predefined local directories and archive the files appearing there.

4.2 Dispatch process

This process should feed the received data to the real-time queues (reasonably only for sparse data) and feed all the data to the corresponding archive datasets.

Due to possible data peaks, parallelisation of dispatching should be considered.

When informed from one of the the data access processes that a milestone has been reached, and after having received acknowledgement that all the data corresponding to that milestone have been archived successfully, the dispatch process should issue the corresponding dataset event.

Due to the near real-time uses of the data on the Mistral platform, the latency for archiving data, intended as the delay between the time when a unit of data is available in its source and the time when it starts being dispatched to the corresponding dataset, should not be higher than 5 minutes.

4.3 Scheduler

At generation of each dataset event, the scheduler process should check whether there are scheduled queries waiting for that specific event and, if so, execute them.

If the query if a "*preprocessing query*" (i.e. a query set up by the portal administrator), the output should be fed back to the dispatch process as if it came from an external data source, for successive rearchiving, and a new (distinguishable from the previous) dataset event should be issued.

If the query is a "*user postprocessing query*" (i.e. a query set up by a registered user), the result should be stored in the user's temporary storage space accessible by the user through the web server (and not accessible by other users). An event should be signalled to the user, containing also information about the location of the result.

4.4 Event signaling

In relation to the dispatching of numerical model data and to the execution of user-scheduled queries and processing pipelines, there is the need to reliably inform users about events such as "the archiving

of a model run in dataset d with reference time t has terminated" or "the user-scheduled process p for reference time t has terminated and the result can be downloaded at URL u".

The signalling of dataset archiving events is strongly dataset-dependent, so it could be reasonably included in the definition of the data source plugins. It has also to be decided whether the same push technology and queue definition used for real-time observed data could be used for this kind of signalling or a different technology should be adopted.

4.5 Interaction with the front-end

The back-end should reliably react to the front-end events of scheduling/descheduling a query and activating/deactivating a real-time queue; it has to be decided whether the interaction should take place via plain file creation/deletion, database, IPC, socket or whatever.

5 Archiving and postprocessing engine

This is the central task of the Mistral portal: this component should have the capability to satisfy user or system requests involving data retrieval from the archive, numerical and/or graphical postprocessing and returning the result to the user.

5.1. Per-dataset configuration

The configuration for each dataset of the archiving engine should contain at least the following information:

- name and physical location of the dataset
- type and format of data contained
- filter rules for selecting data to be dispatched to this dataset.

For purposes related to front-end and authorisation, the following dataset configuration entries should be also present:

- lists of different categories of datasets characteristics in the form <key>=<value1>[,<value2>[,<value3>...]], e.g.
  - DATA_LICENSE = {PUBLIC, PERSONAL, INSTITUTIONAL, COMMERCIAL, DERIVED,...} (more than one value possible)
  - DATASET_TYPE = [METEO_SURFACE, METEO_VERT_PROFILE, METEO_AIRCRAFT, AGRO, AIR_QUALITY, METEO_URBAN, HYDRO, METEO_ANALYSIS, METEO_ASSIMILATION, METEO_FORECAST, ...] (preferably only one value possible)

The DATASET_TYPE keyword could be used in the user interface e.g. for allowing the selection of all datasets of a specific type with a single click.

5.2. Archive queries and postprocessing

The archive data queries and postprocessing could be executed by opaque scripts receiving in input the description of the archive query and of the processing pipeline and returning the data or a pointer to them. Part of this processing could be executed on an HPC system to which the portal has access.

The implementation should foresee the possibility to provide an access to data with a guaranteed level of performance and reliability, dedicated to safety-critical applications such as those related to DPC activity.

If the query involves access to a remote dataset (remote Mistral instance), the query script should act as a proxy to the remote instance, submitting the query and possibly also the postprocessing on the remote instance, collecting the result and possibly merging it with a result obtained by a local query. If possible, the communication with the remote instance should use an administrator-level authentication, not involving the credentials of the user performing the initial query.

5.2.1. Immediate interactive queries

These queries are triggered by the front-end as the result of a query submitted interactively by the user through a web browser or through a web api command-line client (the two cases are indistinguishable from the point of view of the system), the query result could be in the form of a single file containing numerical data or maps or in the form of an online web space where the user can browse the requested maps. The response to the user request should contain information about how to wait for the data-ready event and how to retrieve the data or view them online.

Data for viewing or downloading should be stored in the user's temporary storage space.

5.2.2. Scheduled queries

These queries are generated by the back-end, triggered by a user-scheduled query (see Scheduler paragraph). In this case the result of the query stored in the user's temporary storage space and the user is informed by the back-end about the availability and the data url through the event-signalling system.

Issues:
Should the download URLs for a scheduled query be unique with a predictable path or have a random unique id? Need to monitor downloads and inform user/disable the scheduling if the results of a scheduled query are sistematically not downloaded for a long time, etc.

5.3. Other queries

The engine should be able to provide other type of information to other components, e.g. information about datasets configuration or contents (summaries), with the purpose of providing interactivity to the front-end web forms.

5.4. Dataset access authorization

This component of the system may also be responsible for performing the matching between user properties and dataset properties and decide whether a user is authorised or not to access a specific dataset.

The rule could be that only the datasets having any of the licenses available to the user, or those for which the user has an explicit authorisation given by the administrator, should be exposed to that user in the interface and should be available to the user for query or for inclusion in real-time queue in the related forms.

It may also be useful to give the user a feedback, based on the choice of datasets and on user properties, about what the user can and cannot do with the retrieved data, e.g. redistribute them, build derived works, etc.

6. Embedded applications

The specific applications foreseen within the project, namely the Multi-model ensemble prediction system of Arpa Piemonte and the Italy flash flood system of ECMWF, should be implemented in the Mistral portal as user external application, accessing the portal through its user interface for retrieving observed and forecast data, but with the difference that their output is fed back to the data portal as an input data source. Of course these applications will also benefit from the HPC system available to the portal for executing computation-intensive tasks.

These can be seen as prototypes of a wider range of embedded applications working on real-time observatioanal and forecast data and providing a real-time output, be it in gridded or in sparse point form. Many of these application may also require, from time to time, the access to long records of archived data, both observational and forecast, for performing some kind of calibration or "*training*".

7. Reuse of existing software

Arpae proposes to use the software packages arkimet as the archiving engine and libsim as the main postprocessing tool in the Mistral platform. Both packages are distributed according to the GNU-GPL license.

Arkimet has been developed for Arpae-SIMC by an external developer for the purposes of archiving data in grib, bufr and other formats in a format-agnostic way, so that it can be easily extended. Data are dispatched to datasets, indexed, checked for duplicates according to a set of metadata that can be defined specifically for each format and for each dataset. The same and other metadata can be used for selecting data in the retrieval phase. The adaptation to the Mistral platform may require extensions

for managing new formats (e.g. DPC radar data), new metadata (e.g. for ECMWF point rainfall percentiles) and for improving performance on parallel filesystems at data import time.

Libsim software has been developed at Arpae-SIMC, it includes command-line tools for performing most of the time, space and parameter processing that may be required on the Mistral platform. It may require some extensions for interpolating DPC radar data, for taking advantage of a multi-processor environment and for proposing a command-line interface more tailored to the Mistral platform needs.

The magics-maps package, a project in collaboration between Cineca and Arpae-SIMC, is proposed as the graphical engine for generating the maps of gridded products in the Mistral platform.

8. Need for HPC resources in the Mistral platform

Taking the current situation as an example, the COSMO-LAMI 5 km model run itself, performed on a massively parallel HPC system, requires only 35 minutes for completing a 72 hour forecast, being this time based on operational requirements of forecasting departments. On the other hand the preliminary postprocessing and archiving of the COSMO-LAMI 5 km atmospheric model output on an experimental Mistral-like platform requires about 1 minute of CPU on a typical computing system for each hourly time level. It is thus evident that the postprocessing and archiving system requires as well a multi-processing environment and a parallel filesystem in order to satisfy the real-time needs of state of the art numerical weather prediction. The same is even more evident considering that different model runs may complete as the same time and that the Mistral platform will also deal with the production of graphical forecast maps.