



URBANITE

Supporting the decision-making in urban transformation with
the use of disruptive technologies

Deliverable D3.7

Data aggregation and storage module implementation v1

Editor(s):	TEC, ENG
Responsible Partner:	TECNALIA
Status-Version:	Final - v1.0
Date:	04.10.2021
Distribution level (CO, PU):	PU

Project Number:	GA 870338
Project Title:	URBANITE

Title of Deliverable:	Data aggregation and storage module implementation v1
Due Date of Delivery to the EC:	30.09.2021

Workpackage responsible for the Deliverable:	WP3–Data Management Platform
Editor(s):	TEC, ENG
Contributor(s):	TEC, ENG
Reviewer(s):	JSI
Approved by:	All Partners
Recommended/mandatory readers:	WP4, WP5

Abstract:	This deliverable will have two versions and will present the software implementation of the data aggregation and storage module accompanied with the design specification and documentation. This deliverable is the result of Task3.3.
Keyword List:	Storage, Aggregation, Catalogue, Software
Licensing information:	This document is licensed under Creative Commons Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0) http://creativecommons.org/licenses/by-sa/3.0/
Disclaimer	This document reflects only the author's views and neither Agency nor the Commission are responsible for any use that may be made of the information contained therein

Document Description

Document Revision History

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
v0.1	16/07/2021	Draft ToC	FhG
V0.2	01/09/2021	First draft	TECNALIA
V0.3	16/09/2021	Content reorganization and addition of aggregation and fusion sections	TECNALIA
V0.4	22/09/2021	Addition of data catalogue sections	ENG
v0.5	30/09/2021	Suggestions by reviewers	TECNALIA
v1.0	04/09/2021	Final version	TECNALIA

DRAFT VERSION

Table of Contents

Table of Contents	4
List of Figures	5
List of Tables.....	6
Terms and abbreviations.....	7
Executive Summary	8
1 Introduction	9
1.1 About this deliverable	9
1.2 Document structure	9
2 Implementation.....	10
2.1 Functional description.....	10
2.1.1 Fitting into overall URBANITE Architecture.....	10
2.2 Technical description	11
2.2.1 Data Fusion.....	11
2.2.2 Data Aggregation.....	12
2.2.2.1 Traffic Flow Data Aggregation.....	12
2.2.2.2 Bike Data Aggregation.....	13
2.2.3 Data Storage & Retrieval	15
2.2.3.1 Challenges and architectural design	16
2.2.3.2 Storage Layer.....	17
2.2.3.3 Access Layer	19
2.2.3.4 Technical specifications.....	22
2.2.4 Data Catalogue	23
3 Data Storage & Retrieval-delivery and usage.....	25
3.1 Installation instructions.....	25
3.2 User Manual	27
3.3 Licensing information.....	27
3.4 Download	27
4 Data Catalogue -delivery and usage.....	27
4.1 Installation instructions.....	27
4.2 User Manual	27
4.3 Licensing information.....	31
4.4 Download	31
5 Conclusions	32
6 References.....	33
7 APPENDIX: Data models	34
7.1 Traffic Flow Observed	34

7.2	Air QualityObserved	34
7.3	WeatherObserved	35
7.4	Calendar and Day Specification	35
7.4.1	DaySpecification	35
7.4.2	Calendar	36
7.5	Metadata	37
8	APPENDIX: Storage & Retrieval API	38
8.1	Storage	38
8.1.1	insertTData (POST)	38
8.1.2	updateTData (PUT)	41
8.2	Retrieval	43
8.2.1	getTData (GET)	43
8.2.2	getTData (single record) (GET)	46
8.2.3	getTDataRange (GET)	48
8.2.4	getSupportedDataModels (GET)	50
8.3	Metadata	52
8.3.1	dataset (PUT)	52
8.3.2	dataset (DELETE)	56
8.3.3	getDataset (GET)	57
8.3.4	getCatalogueDatasets (GET)	59
8.3.5	searchDatasets (GET)	61

List of Figures

FIGURE1 -URBANITE ARCHITECTURE	11
FIGURE 2 - TIME SERIES STRUCTURE FOR A TRAFFIC DATA AT A GIVEN ROAD LOCATION CORRESPONDING TO A SENSOR LOCATION.....	12
FIGURE 3 - INFORMATION RELATED TO THE START AND END POINTS OF THE BIKE TRAJECTORIES.....	14
FIGURE 4 - VISUALIZATION OF ONE OF THE DIRECT TESSELLATIONS PROVIDED IN URBANITE FOR THE DISTRICTS OF THE CITY OF BILBAO. THE NUMBER SHOWN FOR EACH DIVISION CORRESPONDS TO THE VALUE OF THE ZONE_ID.....	14
FIGURE 5 - VISUALIZATION OF VORONOI AREAS TESSELLATION OBTAINED BY URBANITE. THE INPUT FOR THE GENERATION ARE THE LIMITING RECTANGLE AND THE SET OF RED POINTS MARKED IN THE MAP.	15
FIGURE 6 - DATA STORAGE & RETRIEVAL REPOSITORIES	15
FIGURE 7 - TECHNOLOGY STACK	17
FIGURE 8 – SCHEMA ABOUT URBANITE COMPONENTS (UI, DATA CATALOGUE AND CONNECTORS, DS&R).....	23
FIGURE 9 – DATA CATALOGUE MANAGEMENT OF FEDERATED ODMS	28
FIGURE 10 – DATA CATALOGUE CONFIGURATION MANAGEMENT	28
FIGURE 11 – DATA CATALOGUE FEDERATED METADATA SEARCH BY TAG	29

FIGURE 12 – DATA CATALOGUE FEDERATED METADATA SEARCH.....	29
FIGURE 13 –INFORMATION FIELDS OF DATASET’ METADATA SEARCH LIST	30
FIGURE 14– DATA CATALOGUE FEDERATED METADATA DATASET DETAIL VIEW	30
FIGURE 15 – DATA CATALOGUE- DISTRIBUTION - INFORMATION ICON	30
FIGURE 16 – DATA CATALOGUE -DETAILS OF A DISTRIBUTION	31

List of Tables

TABLE 1: STATUS OF DATA FUSION, STORAGE AND RETRIEVAL/CATALOGUE REQUIREMENTS FROM D5.1	10
TABLE 2: STRUCTURE FOR DAY SPECIFICATION	35
TABLE 3: STRUCTURE FOR A CALENDAR SPECIFICATION	36
TABLE 4: API FOR DATA INSERTION.....	38
TABLE 5: API FOR DATA UPDATE	41
TABLE 6: API FOR DATA RETRIEVAL	43
TABLE 7: API FOR DATA RETRIEVAL (SPECIFIC RECORD).....	46
TABLE 8: API FOR DATA RETRIEVAL (TIME RANGE).....	48
TABLE 9: API FOR THE RETRIEVAL OF THE DATA MODELS INFORMATION.....	50
TABLE 10: API FOR THE INSERT AND UPDATE OF METADATA	52
TABLE 11: API FOR THE DELETION OF METADATA.....	56
TABLE 12: API FOR THE RETRIEVAL OF DATASET METADATA	59
TABLE 13: API FOR THE SEARCH AND RETRIEVAL OF DATASET METADATA	61

DRAFT VERSION

Terms and abbreviations

API	Application Programming Interfaces
BI	Business Intelligence
CSV	Comma Separated Values
DCAT	Data CATalog Vocabulary
DCAT-AP	Data CATalog Vocabulary - Application Profile
EC	European Commission
GNU	GNU's Not Unix
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
JAR	Java ARchive
JDBC	Java Database Connectivity
JSON	JavaScript Object Notation
JSON-LD	JavaScript Object Notation for Linked Data
JVM	Java Virtual Machine
NoSQL	Not Only SQL
OD	Origin Destination
RDBMS	Relational DataBase Management System
REST	Representational State Transfer
SQL	Structured Query Language
URL	Uniform Resource Locator

Executive Summary

This deliverable contains an overview of the software components that are related to the tasks of data aggregation, storage, retrieval and its catalogue. This refers to the process of mapping, aggregation, storage and retrieval of the curated data. A common model for storage of the information and knowledge extraction is being defined, handling the semantic processing of the curated data as well as the aggregation and deduplication of the data that originate from distinct sources. Finally, the long-term and short-term storage strategy and implementation of a set of APIs for retrieval are under development. For each existing module described in this deliverable, an overview along with a description is given. Where applicable, details on configuration and usage are provided. The components are implemented following a microservice approach, so they fit well with the global docker-based architecture.

DRAFT VERSION

1 Introduction

The term Data Management Platform stands for a variety of distinct software components that work together to deliver the key functionalities that are data harvesting, data curation, and data aggregation and storage. The three deliverables D3.2, D3.5, and D3.7 focus on these core features respectively. Due to the interaction between these modules the aforementioned deliverables should be understood as a collection of documents related to the same overarching concept that is the Data Management Platform.

1.1 About this deliverable

Within the Data Management Platform this deliverable focuses on the components related to data fusion and aggregation, data storage and retrieval and data catalogue. It presents fusion and aggregation techniques of interest for urban mobility, the challenges involved in storing and retrieving big volumes of data as well as managing heterogeneous formats, and a solution for managing datasets and related metadata.

1.2 Document structure

Section 2.1 covers the functionalities provided by the four components related to data fusion and aggregation, data storage and retrieval and data catalogue, and their fitting on the general architecture defined in WP5. Section 2.2 describes the technical details of the different components in the Data Management Platform. Then, for each of the main modules, dedicated sections 3, 4 and 5, identify their installation instructions, a brief user manual, licensing information and the repository URL for downloading the source code. The document wraps up with a conclusion and references.

DRAFT VERSION

2 Implementation

2.1 Functional description

As mentioned in the introduction, this deliverable focuses on the 4 components related to data fusion and aggregation, data storage and retrieval and data catalogue.

The functional requirements for these components were listed in deliverable D5.1 and a detailed design was provided in deliverable D5.4. We present here a short summary and the status of development. All the requirements have been fulfilled for the data models and datasets that are covered in the first prototype. More data models will be supported for the second version.

Table 1: Status of Data fusion, storage and retrieval/catalogue requirements from D5.1

Component	Requirements in D5.1 + current status
Data fusion/aggregation	<ul style="list-style-type: none"> • DF.01. Aggregation. The component should allow to aggregate curated data coming from different data sources if needed. (<i>partially fulfilled-more aggregations need to be computed automatically</i>) • DF02. Deduplication. The component should allow the deduplication of the data (<i>fulfilled for v1 data models</i>). • DF03. Data mapping. The data should be mapped into EU vocabularies (<i>fulfilled for v1 data models</i>). • DF.04. MetaData mapping. The metadata should be mapped into DCAT-AP metadata (<i>fulfilled</i>).
Data Storage	<ul style="list-style-type: none"> • DS.01. Big data Storage. The harvested data should be persisted to a big data capable storage solution (<i>fulfilled</i>). • DS02. DCAT-AP compliance. The data storage component should be able to process and store DCAT-AP compliant metadata (<i>fulfilled</i>).
Data Retrieval / Data Catalogue	<ul style="list-style-type: none"> • DR.01. Data Retrieval. The data retrieval component must expose API to retrieve and query the data stored in the different repositories (<i>fulfilled</i>). • DR.02. Data Hub. The metadata stored in the repositories should be accessible through a data hub in a uniform way taking advantage of DCAT-AP standard and related profile (<i>fulfilled</i>).

2.1.1 Fitting into overall URBANITE Architecture

The 4 components described in this deliverable have been implemented following a microservice approach, so they fit well with the docker-based architecture designed in WP5. Besides, the Data Catalogue offers a user interface to manage and search through the datasets.

The components described in this deliverable are high-lighted in green in the architecture diagram from deliverable D5.4 below:

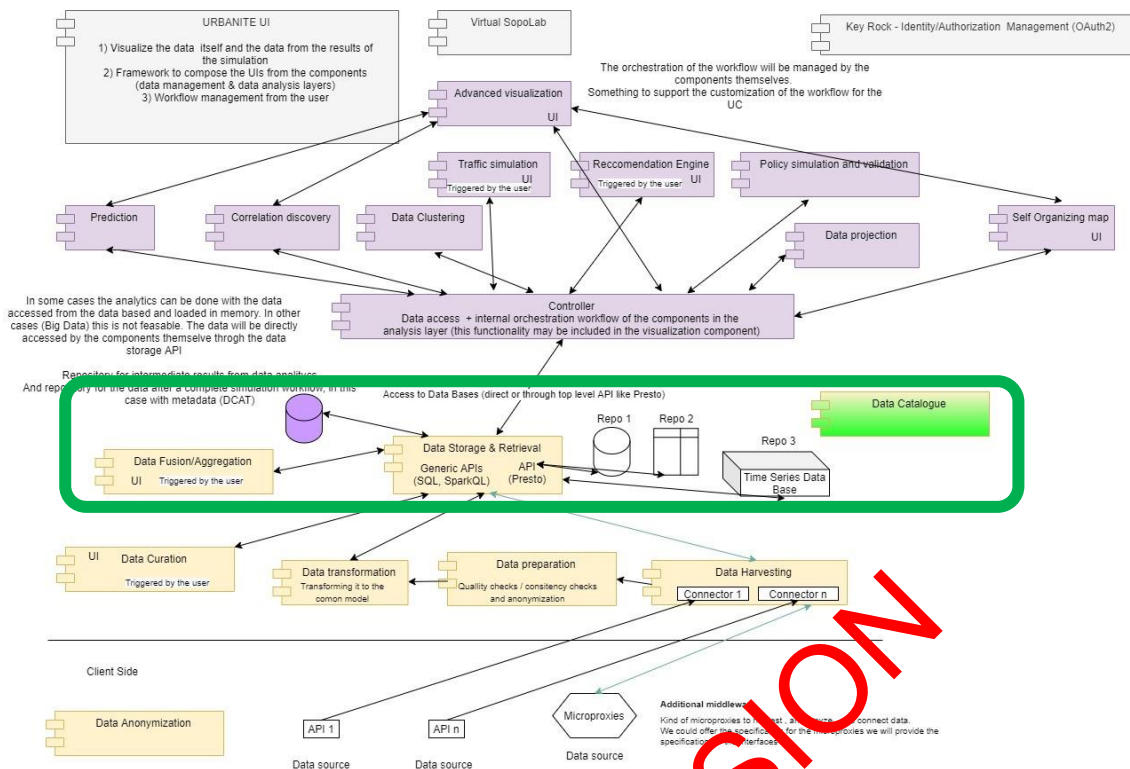


Figure1 -URBANITE architecture

2.2 Technical description

This section describes the technical details of the different components in the Data Management Platform dedicated to data fusion & aggregation, data storage & retrieval and data cataloguing.

Some data fusion and aggregation functionalities have been integrated into the different functional modules related to the analysis of traffic and mobility by bicycles. In the next versions of the Data Management Platform, it is proposed to provide them with their own entity, deployed capabilities and a specific interface.

2.2.1 Data Fusion

According to [1] “data fusion techniques combine data from multiple sensors and related information from associated databases to achieve improved accuracy and more specific inferences than could be achieved by the use of a single sensor alone.”

The integration of data and knowledge from the same or several different sources is called data fusion. The terms information fusion and data fusion are commonly used as synonyms, but there is a little difference. The term data fusion refers to the integration of raw data, i.e. directly obtained from sensors whereas the term information fusion is used to refer to the integration of already processed data [2]. In the case of URBANITE, the data management platform deals mostly with information fusion.

The purpose of using data/information fusion is to improve data quality, i.e. to reduce error probability and to have a higher reliability of the data being used in the algorithms for analytics and decision-making.

There are different data fusion techniques and the approach to use depends on the intended usage. E.g. images from different types of cameras can be fused to obtain more information, or data from complementary data sources can be fused to create improved datasets.

In URBANITE, weather data coming from different data sources and weather services will be fused to create improved datasets covering more variables. Other data fusion examples will be analyzed for version 2 of the platform and components.

2.2.2 Data Aggregation

Data aggregation is the process of gathering data and presenting it in a summarized format. Data aggregation is useful e.g. to hide personal information or to provide information in a synthetic form or to train the models in WP4.

In the case of **Traffic Data**, the following aggregated functions are calculated before performing the training of the Artificial Intelligence models in WP4:

- Initial date and time of the data, typically the Unix timestamp.
- End date and time of data, typically the Unix timestamp.
- Temporal aggregation period, typically 5 or 15 minutes.
- Maximum value of the traffic flow.
- Minimum value of the traffic flow.
- Number of holes within the data.
- Average period of each hole.
- Standard Deviation

Next, we provide a more detailed description of traffic flow data aggregation and bike data aggregation computed in URBANITE.

2.2.2.1 Traffic Flow Data Aggregation

Traffic flow data measured by sensors can be provided in different ways. One common way (as provided by the Bilbao use case) is to provide an aggregated time series structure describing the number of vehicles that have gone through a particular road at a given sensor location.

...
 1631523300 123
 1631523600 107
 1631524200 144
 1631524500 208
 ...

Figure 2 - Time series structure for a traffic data at a given road location corresponding to a sensor location

The structure can be seen in Figure 2. Each line in the time series has two values being the first one the timestamp (in seconds) in Epoch format and the second one the number of vehicles that have gone through that specific location for the last 5 minutes.

This period, 5 minutes or 300 seconds is the temporal aggregation rate used for this time series. The period can vary from time series to time series and depending on the application. In the case of URBANITE, temporal aggregation rates of 5 and 15 minutes are considered. The 15 minutes aggregation is computed from the previous one. The computation of time series with

longer temporal aggregation period implies the loss of information. For this reason, data is usually stored with the finer resolution available.

As it is mentioned above, the resolution to be used depends on the specific application but also on the actual data measured. It could be thought that the best would be to always use the finer resolution but, in practice, using the finer resolution could imply working with a time series with a lot of noise, whereas increasing the temporal aggregation period averages the noise producing a smoother time series. Hence, these two aspects must be balanced, on one hand the time series should have the most possible information, and on the other with the least possible noise. The periods chosen in URBANITE, 5 and 15 minutes, correspond to values in which the noise and the amount of information are compensated producing time series that are appropriated to train AI model for prediction.

Another important aspect to mention is that, although we use a 5-minute aggregation rate, that does not imply that we can count on the fact that the sensors will always provide data every 5 minutes. For example, in Figure 2 there is no data available for timestamp 1631523900. This usually occurs when working with real data because there are always situations where the sensor has gone offline due to issues related to the data capturing process, power outages, connectivity problems or other obstacles that do not allow to obtain the correct data.

However, the traffic data is not always provided by the city in an aggregated time series format. For example, in the Helsinki use case, a timestamp is stored every time a vehicle goes through the sensor location. This implies that in order to transform this information into an aggregated time series format, as shown in Figure 3, some calculations must be performed. The aggregation component of the Data Management platform is in charge of performing these calculations periodically. Given an aggregation period and a starting time stamp, a division of the timeline can be performed and the trips that lie in each slot can be added to produce the aggregated time series format.

In certain cases, other aggregations need to be performed, for example some traffic sensors divide the information depending on the type of vehicle (motorcycle, regular car or long vehicle, i.e., a bus or a truck). In the traffic flow prediction procedure performed in URBANITE, the traffic flow is not distinguished according to the vehicle type. Therefore, a vehicle type aggregation needs to be done. Other sorts of aggregations include the aggregation of vehicles measured in different lanes of the road.

2.2.2.2 Bike Data Aggregation

The harvested data related to bikes are the GPS data of the individual bike rentals, in the case of Bilbao and Helsinki use cases rented using the city's public service. These data are used in URBANITE for the computation of Origin-Destination (OD) matrixes and for the analysis of trajectories. For the first case, some aggregations need to be computed by the Data Management Platform. For the second case, data cleaning processes need to be carried out (see D3.5 for more detailed information about *Cleaning Trajectory Data*).

In the case of the computation of the OD matrixes, a reduced set of data is used and only the initial and end points of each rental are considered (see Figure 3). The data in each row represents a rental where the 3 first values correspond to the starting point of the trajectory (timestamp in seconds in Epoch format when the rental started, and GPS latitude and longitude coordinates), and the last 3 values correspond to the end of the trajectory (timestamp and GPS coordinates).

```

...
1631524500 43.265315 -2.943067 1631824230 43.263796 -2.925268
1631522503 43.254280 -2.942948 1631622503 43.251823 -2.927679
1631524523 43.262615 -2.917710 1631534523 43.256140 -2.903499
...

```

Figure 3 - Information related to the start and end points of the bike trajectories

There are two different aggregation processes to be performed in the computation of the OD matrixes: **temporal** and **spatial aggregation**. The temporal aggregation corresponds to the same type of aggregation performed in the case of traffic flow data, i.e. trips are summed up for a given time period. Typically, the aggregation periods for the computation of the OD matrixes are longer than for the prediction of traffic data, being a typical value equal to 1 hour. The reason for choosing such a large value, in comparison to the traffic flow case, is due to the fact that the number of trips is notably lower, and a longer period is needed in order to obtain enough statistics to reduce the noise.

The spatial aggregation implies adding together all the trips within the area of interest. For this purpose, a tessellation of the map of the area of interest, i.e. a division of the map that fills all the zone of interest, needs to be provided.

In the case of the OD matrix computation, two different mechanisms are provided within the URBANITE project: a direct method by means of a `geojson` object, and by defining a set of points to produce a Voronoi tessellation¹.

The direct method needs a `geojson`² that represents a “FeatureCollection” object which contains a set of features where the geometry should be of type Polygon. Each of these features should have a property of `namezone_id` that will later be used to identify each of the areas.



Figure 4 - Visualization of one of the direct tessellations provided in URBANITE for the districts of the city of Bilbao. The number shown for each division corresponds to the value of the `zone_id`.

¹ <https://mathworld.wolfram.com/VoronoiDiagram.html>

² <https://geojson.org/>

Alternatively, in order to generate the Voronoi areas, two inputs need to be provided: a set of points and a rectangle containing all the previous points and that set a limit to all the Voronoi generated areas. These and the resulting areas generated by the tool are shown in Figure 5.



Figure 5 - Visualization of Voronoi areas tessellation obtained by URBANITE. The input for the generation are the limiting rectangle and the set of red points marked in the map.

The resulting Voronoi areas that can be used in the process of the spatial aggregation in the computation of the OD matrix are defined as the set of points that are closer to the given input points. This definition implies that the Voronoi areas are polygons. A possible set of points to define the spatial aggregation area typically is the set of locations where the bicycles can be rented, i.e., the bike stations.

2.2.3 Data Storage & Retrieval

The Data Storage & Retrieval component provides the means to store and retrieve datasets, DCAT-AP compliant metadata, and related data. Hence, this component needs to have repositories to store both DCAT-AP compliant metadata and transformed data, as depicted in Figure 6.

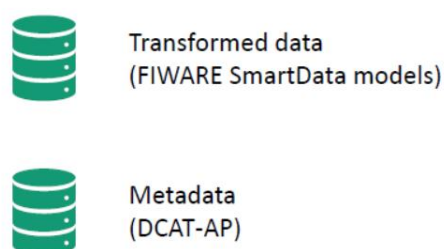


Figure 6 - Data Storage & Retrieval repositories

2.2.3.1 Challenges and architectural design

When deciding about the technologies to use to implement the Data Storage & Retrieval component, several factors were taken into account. One of the main factors to consider was related to the volume of data to be managed, as the number of records to be ingested can be very high in some cases due to the update frequencies (for example, traffic flows can reach around 800.000 monthly registrations). Another important factor was related to the diversity of available data. Although a specific data model is common to several use cases, they may not all have the same information available, making the records, within the same common model, quite heterogeneous. For these cases, the JSON-LD format is quite suitable.

Therefore, it was necessary to design a storage system which could be capable of handling large volumes of data while offering considerable flexibility in terms of the structure of the data, characteristics that fit very well with the NoSQL MongoDB database.

On the other hand, it was also taken into consideration that there may be other types of data sources with information that does not present a high update or a large volume of records, for which solutions such as the MySQL database can be used.

Even so, although these two databases (MongoDB and MySQL) have been chosen for this first prototype, the system is flexible and remains open to adding other databases that may be more appropriate to the needs that may arise in the future, or to changing the current ones.

To make this possible, the main access point to the storage system is through a REST API. This API provides a set of services that will be in charge of accessing the most appropriate database depending on the type of data it is managing, in a totally transparent way to the modules that interact with it.

In addition, since the processing modules may require certain functionality on the data beyond the services provided by the API (fairly generic data insertion or retrieval), it has been decided to include a technology like Presto, which provides access via JDBC to the different databases, so these modules can execute custom queries (SQL-like). Presto has connectors for many of the most common databases, including those used in this first version (MongoDB and MySQL).

A high-level architecture of this component is shown in Figure 7:

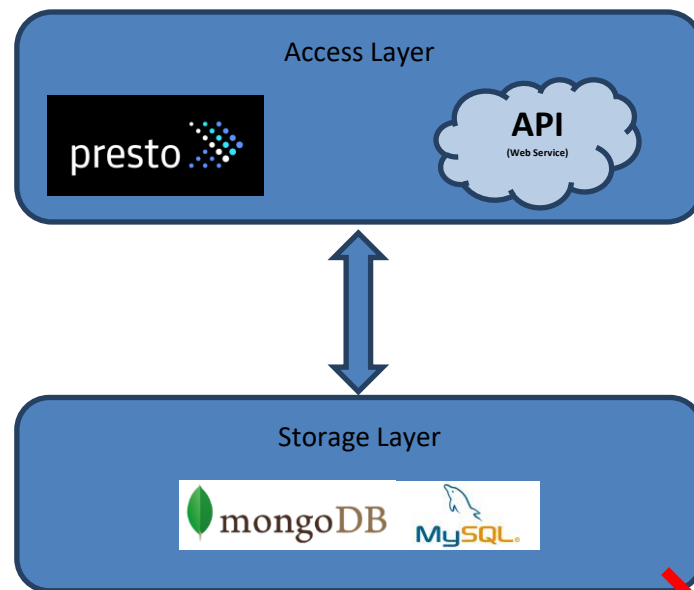


Figure 7 - Technology stack

It consists of two levels:

- At the bottom level is the storage itself, being a combination of different types of databases:
 - SQL Databases, like MySQL.
 - NoSQL Databases, like MongoDB.
- At the top level are the mechanisms for accessing the repositories, both for storing and retrieving data. In turn, this layer consists of two components:
 - A REST API with predefined methods for inserting or accessing data and metadata.
 - A JDBC connection through the Presto software, to perform custom queries (SQL statements), different to those offered by the API.

These layers and their components are described in more detail below.

2.2.3.2 Storage Layer

As mentioned before, the chosen Data Storage system is a combination of SQL (MySQL) and NoSQL (MongoDB) databases, to cover all the needs that may arise from the analysis and prediction processes developed in WP4. The data is stored in the databases according to the models described in 7 APPENDIX: Data models and formatted in JSON-LD format as key-values.

MongoDB³ is an open source, document-oriented NoSQL database, which means that it stores data in the form of JSON-like documents, thus it supports arrays and nested objects as values. Being based on documents and Schema Less, the documents within a collection (table) may not have the same fields, thus avoiding having fields with empty or null values that make the size of the database grow unnecessarily.

In addition, MongoDB does not require large computing resources, and it can be used in a decentralized environment in a distributed way. This allows scalability not only vertically (CPU and RAM) but also horizontally (creating more nodes).

³ <http://www.mongodb.com/>

On the other hand, MySQL⁴ is one of the world's most popular relational database (RDBMS), and it is based on a client-server model. Among its main characteristics we can find:

- Low cost in hardware and software requirements for its execution.
- Offers high speed and good performance.
- Capable of handling large volume of data.
- Ease of installation and configuration, supported in almost 100% of current operating systems.
- High stability and low probability of data corruption.
- Supports security through SSL (Secure Socket Layer) and data encryption.
- Possibility of using different storage mechanisms that offer different operating speeds, physical support, capacity, geographical distribution, transactions ...
- Use of ACID transactions (Atomic, Consistent Isolated, Durable), through commit, rollback, crash recovery and record blocking, and Distributed Transactions.
- Supports replication.
- Supports ANSI SQL⁵
- Ease of data import and export processes.

All these features have made us opt for the inclusion of MongoDB and MySQL in URBANITE.

Finally, all harvested data in this first prototype is stored in MongoDB. Next, we explain how the collections have been organized for each of the data models depending on their volume.

- **Traffic Flow Observed:** the number of records (traffic status measurements) can be too high to store all of them in a single collection. For example, in Bilbao Use Case we can find approximately 800,000 records per month, which would make almost 10 million records per year.

Due to performance issues, and although the logic of storing and retrieving the data is somewhat complicated, the data is divided into one collection for each month and year, so handling a volume of about 800.000 records per collection (Bilbao use case) is fast and efficient.

The names of the collections follow the pattern `trafficflowobserved_<city>_<year>_<month>`. For example:

```
trafficflowobserved_bilbao_2021_08
trafficflowobserved_helsinki_2021_06
...
```

To speed up data queries, collections also have an index on the `dateObserved` field, in descending order, in addition to the index that MongoDB generates on the unique record identifier.

- **Air Quality Observed:** For this data model, the number of records per use case is estimated to be around 50.000 per month, so a volume close to 600.000 records per year allows us to have a single collection per year and use case.

⁴ <https://www.mysql.com/>

⁵ https://docs.oracle.com/database/121/SQLRF/ap_standard_sql001.htm

The names of the collections follow the pattern `airqualityobserved_<city> _<year>`.
E.g.:

```
airqualityobserved_bilbao_2021
airqualityobserved_messina_2020
...
```

To speed up data queries, collections also have an index on the `dateObserved` field, in descending order, in addition to the index that MongoDB generates on the unique record identifier.

- **Day Specification:** In this case, there should be one record for each day of a year, so the total number of records is small enough to be able to have all the data in a single collection per use case, without compromising performance.

The collections are named `dayspecification_<city>`:

```
dayspecification_amsterdam
dayspecification_bilbao
dayspecification_helsinki
dayspecification_messina
```

- **Calendar:** For this data model, there should be one record for each year, so all the data will be stored in just one collection per use case. The collections are named `calendar_<city>`:

```
calendar_amsterdam
calendar_bilbao
calendar_helsinki
calendar_messina
```

- **Metadata:** All metadata information is stored in a single collection named *metadata*, since we estimate that the number of records to be handled is not excessively large, so there is no need to divide them into different collections. Storing them in a single collection also facilitates operations on the metadata, both the insertion and the search by tags. To perform these searches by tags in the text fields, a text index⁶ has been created on the collection.

2.2.3.3 Access Layer

The access layer contains a REST API with predefined methods for inserting or accessing data and metadata and a JDBC connection, through the Presto software, to the different databases.

2.2.3.3.1 API

The API is the main access point to the datasets and metadata, providing methods to store and retrieve both of them. It offers a set of REST Web Services, so all the accesses are made through HTTP/HTTPS requests.

⁶ <https://docs.mongodb.com/manual/core/index-text/>

This section provides a list of these methods, divided into storage, retrieval, and metadata related REST services. The complete description (method, input and output parameters, etc.) is described in 8 APPENDIX: Storage & Retrieval API and accessible at:

<https://urbanite.esilab.org:8443/data/swagger-ui/index.html?configUrl=/data/v3/api-docs/swagger-config>

Data Storage Operations to store Data

PUT `/updateTData/{model}/{city}/{id}` Update one record of the database

POST `/insertTData/{model}/{city}` Add new data to database

Data Retrieval Operations to retrieve Data

GET `/getTData/{model}/{city}/{id}` Get a record from the database

GET `/getTDataRange/{model}/{city}` Get data from the database within a specific time range

GET `/getTData/{model}/{city}` Get data from the database

GET `/getSupportedDataModels` Get available Data Models

Metadata Operations to store and retrieve metadata

PUT `/dataset` Add new metadata of a dataset or update if exists

DELETE `/dataset` Delete the metadata of a dataset.

GET `/getDataset` Get metadata of a dataset

GET `/searchDatasets` Searches among the metadata of the existing dataset

GET `/getCatalogueDatasets` Get the metadata of all datasets.

2.2.3.3.2 PRESTO

Presto is an open source distributed SQL query engine designed to query large data sets distributed over one or more heterogeneous data sources, including traditional relational databases and other data sources or NoSQL databases.

Presto is based on connectors which allows Presto to interact with the resources using a standard API.

Another important feature for which the use of Presto has been chosen is the possibility of connecting to it (and therefore to the databases) through a JDBC driver. This allows analytics or prediction modules to carry out personalized and optimized queries, beyond the generic ones offered by the API.

2.2.3.3.2.1 Configuration

Presto has four configuration files, located in *config* folder:

- **Node Properties:** environmental configuration specific to each node. A node is a single installed instance of Presto on a machine.

Example:

```
node.environment=urbanite
node.id=a7413702-23b4-11e6-bb6e-600308a67678
node.data-dir=/opt/presto/presto-server-0.157/data
```

- **JVM Config:** contains a list of command line options used for launching the Java Virtual Machine. The format of the file is a list of options, one per line.

Example:

```
-server
-Xmx16G
-XX:+UseG1GC
-XX:G1HeapRegionSize=32M
-XX:+UseGCOverheadLimit
-XX:+ExplicitGCInvokesConcurrent
-XX:+HeapDumpOnOutOfMemoryError
-XX:OnOutOfMemoryError=kill -9 %p
```

- **Config Properties:** configuration for the Presto server. Every Presto server can function as both a coordinator and a worker, but dedicating a single machine to only perform coordination work provides the best performance on larger clusters.

Example:

```
coordinator=true
node-scheduler.include-coordinator=true
http-server.http.port=8080
query.max-memory=5GB
query.max-memory-per-node=1GB
discovery-server.enabled=true
discovery.uri=http://presto:8080
```

- **Log Properties (optional):** allows setting the minimum log level for named logger hierarchies.

Example:

```
com.facebook.presto=INFO
```

More detailed information about the configuration files of Presto can be found here⁷

2.2.3.3.2.2 Connectors

To provide access to the different databases from Presto, it is necessary to create catalogs to configure the connectors of each one of them, creating the configuration files in the *config/catalog* folder. Each of these files must have the proper name of the connector.

➤ MySQL configuration⁸

This database catalog only needs the connection properties (url of the server, and user and password to connect)

Example:

```
connector.name=mysql
connection-url=jdbc:mysql://MYSQL_HOST:3306
connection-user=root
connection-password=MYSQL_ROOT_PASSWORD
```

➤ MongoDB configuration⁹

This catalog has a set of configuration options, but only the server (or list of servers) is mandatory.

Example:

```
connector.name=mongodb
mongodb.seeds=MONGO_HOST
mongodb.socket-keep-alive=true
```

2.2.3.3.2.3 JDBC¹⁰

The connection to Presto through JDBC is done in the same way as with any other database, only the server and the user and password are needed. Also, the connection is made to a specific catalog (database type) and schema (database).

Example:

```
String url = "jdbc:presto://[server:port]/mongodb/urbanite";
Connection connection = DriverManager.getConnection(url, "root", null);
```

2.2.3.4 Technical specifications

The Data Storage component has been developed in Java using the Spring Framework¹¹ and Spring Boot¹².

⁷ <https://prestodb.io/docs/current/installation/deployment.html#configuring-presto>

⁸ <https://prestodb.io/docs/current/connector/mysql.html>

⁹ <https://prestodb.io/docs/current/connector/mongodb.html>

¹⁰ <https://prestodb.io/docs/current/installation/jdbc.html>

¹¹ <https://spring.io/>

¹² <https://spring.io/projects/spring-boot>

2.2.4 Data Catalogue

The Data Catalogue component provides access to the metadata of the different data sources and their datasets using a federated approach. In particular, Data Catalogue component is based on Idra which is a platform able to federate Open Data Management Systems (ODMS) based on different technologies providing a unique access point. It harmonizes the metadata coming from federated data sources following DCAT-AP standard.

More specifically the Data Catalogue provides functionalities to discover and access the datasets' metadata collected and managed by the components of URBANITE Ecosystem for data acquisition, aggregation and storage, such as the Data Storage & Retrieval component.

The Data Catalogue makes use of a specific connector, developed ad hoc for URBANITE ecosystem, to interact with the Data Storage & Retrieval. Following the guidelines for the development of Idra connectors, the new connector implements the operations to allow the interaction between the Data Catalogue and the Data Storage & Retrieval.

The URBANITE user interface already integrates the Data Catalogue user interface as depicted in Figure 8. In this schema is underlined also the usage of the Data Storage & Retrieval connector between the Data Catalogue and the Data Storage & Retrieval components (a part of other available connector types, e.g. CKAN, DKAN etc.).

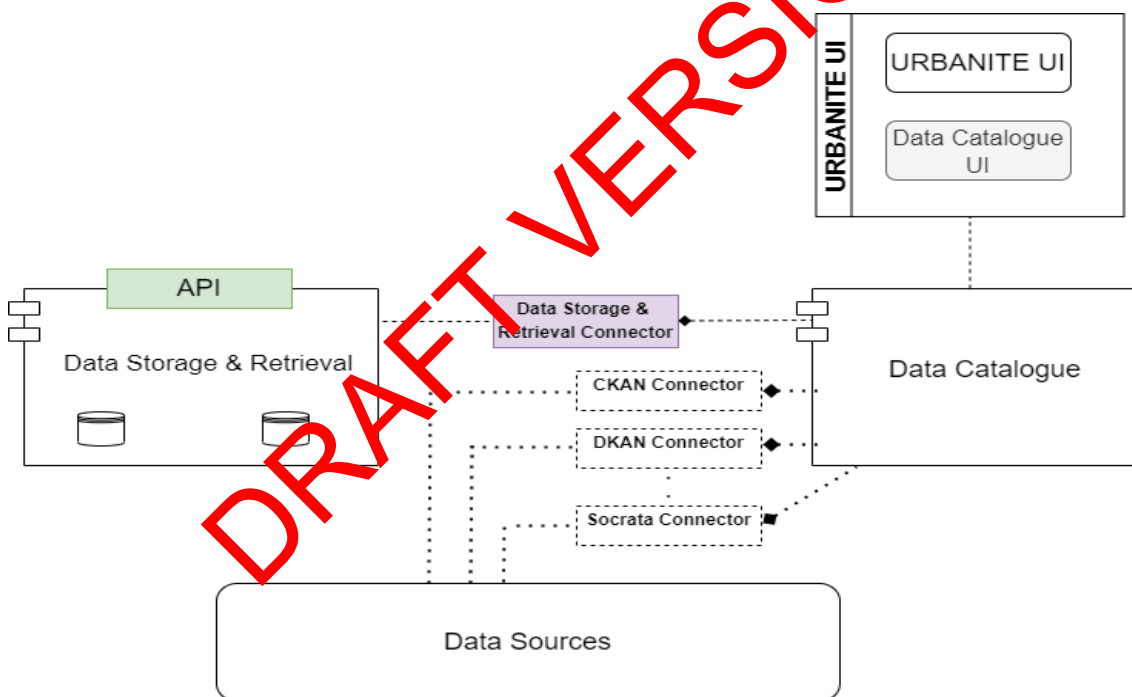


Figure 8 – Schema about Urbanite components (UI, Data Catalogue and connectors, DS&R)

A connector is the basic building block used by Idra to interact and harmonize, following DCAT-AP standard, the metadata of the datasets managed by the federated Open Data Management System. Within Idra, a specific connector is provided for any of the supported data sources typology (indeed, Idra already includes connectors to federate ODMS based on CKAN, DKAN, Socrata, etc).

The new created connector allows to access and manage the datasets provided by the Data Storage & Retrieval by exploiting its exposed Rest API *"/data/getCatalogueDatasets"*.

The datasets' metadata returned by this API are finally added to Idra and made available for all functionalities provided by the tool as for searching (using Idra APIs). More technical details about the Data Catalogue are reported in the deliverable D5.4.

DRAFT VERSION

3 Data Storage & Retrieval -delivery and usage

3.1 Installation instructions

In order to integrate well into the URBANITE platform all components are available as Docker¹³ images. However, before building the Docker images the corresponding JAR files need to be created. A JAR file is an executable that runs on the JVM.

The storage component is composed by two Docker groups:

➤ API

The API of the storage component rely on a build tool called Maven for dependency management and generation of the JARs. As such, the deployment of a service can be achieved using the commands below. Note that curly brackets indicate that applicable values need to be substituted.

```
$>mvn clean package
```

```
$> docker build -t urbanite/dataStorage
```

```
$> docker run -p {PORT}:80 urbanite/dataStorage
```

For this component a certain configuration is needed to be applied using environment variables, in this case the setup parameters to connect to MongoDB. If it is not present, the default values will be used:

VARIABLE	DESCRIPTION	DEFAULT VALUE
MONGO_HOST	Host where MongoDB is installed	Mongodb
MONGO_PORT	Port where MongoDB is listening	27017
MONGO_DBNAME	Name of the MongoDB Database to insert or retrieve data	urbanite

These environment variables can be passed to Docker containers, for example:

```
$>docker run -it -p 80:80 -e MONGO_HOST=172.26.41.138 -e MONGO_PORT=27018 urbanite/datastorage
```

➤ Databases and Presto

As Presto has to connect to the different databases (MongoDB and MySQL), a single Docker-Compose¹⁴ configuration file will be used to create all the images at once.

For this, Presto needs some previous configuration (see section **¡Error! No se encuentra el origen de la referencia.**). Also, MySQL needs to create some user to be used by Presto, so in the creation of MySQL Docker image, these variables will be set:

VARIABLE	DESCRIPTION
MYSQL_ROOT_PASSWORD	The password for ROOT user, to create this user in MYSQL

¹³ <https://www.docker.com/>

¹⁴ <https://docs.docker.com/compose/>

	configuration
MYSQL_USER	A user to be used by Presto
MYSQL_PASSWORD	The password for MYSQL_USER

Below is the docker-compose file:

```
---
# brings up the dependencies
version: '2'
services:
  presto:
    container_name: urbanite_presto
    build:
      context: ./presto
    dockerfile: Dockerfile
    image: presto
    links:
      - mongodb
      - mysql
    depends_on:
      - mongodb
      - mysql
    ports:
      - "8080:8080"
      - "8889:8889"

  mysql:
    container_name: urbanite_mysql
    image: mysql:8.0.24
    environment:
      MYSQL_ROOT_PASSWORD: '{*****}'
      MYSQL_USER: 'presto'
      MYSQL_PASSWORD: '{*****}'
    ports:
      - "3306:3306"
    volumes:
      - /opt/mysql_data:/var/lib/mysql

  mongodb:
    container_name: urbanite_mongodb
    image: mongo:4.0.24
    ports:
      - "27017:27017"
    volumes:
      - /opt/mongo_data:/data/db
```

This will:

- Create a docker container for MySQL, from an official image, with a user 'presto' and running at port 3306.
- Create a docker container for MongoDB, from an official image, and running at port 27017.
- Create a docker container for Presto, from a custom image built with a custom configuration, and running at ports 8080 and 8889.

The configuration for the creation of the image just uses an official image and configures it with the MySQL user and the catalogues (databases) to which Presto is connected.

3.2 User Manual

The only part of the component that a user can interact with is the API, a REST Web Service. This API is developed following the OpenAPISpecification¹⁵ and offers a human-readable version at [http://\[server:port\]/data/swagger-ui/index.html?configUrl=/data/v3/api-docs/swagger-config](http://[server:port]/data/swagger-ui/index.html?configUrl=/data/v3/api-docs/swagger-config)

This page shows the services that are implemented, giving information about the access URLs, parameters, return codes and values...

Also, a JSON file with the full specification can be found at [https://\[server:port\]/data/v3/api-docs](https://[server:port]/data/v3/api-docs)

3.3 Licensing information

The license terms for the software are under discussion among the consortium. AGPLv2 and AGPLv3¹⁶ are being considered.

3.4 Download

All source code resides in the GitLab maintained by Tecnia¹⁷.

4 Data Catalogue -delivery and usage

4.1 Installation instructions

This section covers the steps needed to properly install the Data Catalogue. As before described the Data Catalogue is an extension of Idra which is an Open Data Federation Platform developed as a Java EE (Enterprise Edition) application. This tool can be installed through Docker¹⁸. The installation instructions are detailed in the *Installation overview* section of the online manual. The detailed instruction to install or use the administration functionalities of Idra can be also found at the corresponding section on *Read The Docs*¹⁹.

4.2 User Manual

The Data Catalogue provides a set of Restful APIs to interact with the IDRA tool and its functionalities. These APIs are developed following the OpenAPI specification and in particular Apiary²⁰. The official APIs are available on this official link²¹. These APIs are grouped into *Administration APIs*, *End User APIs* and *Federation APIs*. Further information about Idra APIs are reported in the deliverable D5.4 and available in the Idra documentation section²².

The Data Catalogue offers the functionalities to discover and access the datasets collected and managed and produced by the components of URBANITE Ecosystem for data acquisition, aggregation and storage. The Data Catalogue provides these main functionalities to let to the

¹⁵ <https://swagger.io/specification/>

¹⁶ <https://www.gnu.org/licenses/agpl-3.0.en.html>

¹⁷ <https://git.code.tecnalia.com/urbanite/private/wp3-data-management/storage/dataStorage>

¹⁸ Install Idra on Docker - https://idra.readthedocs.io/en/latest/admin/install_docker/

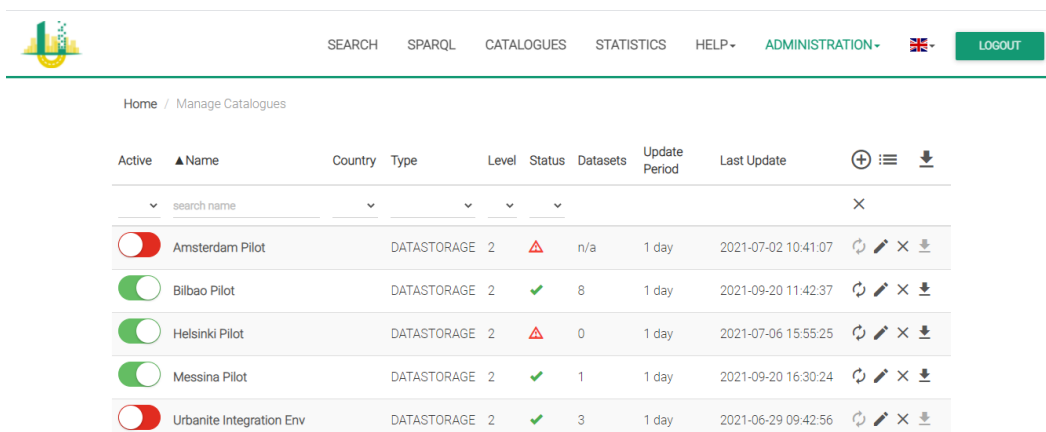
¹⁹ Idra Installation - <https://idra.readthedocs.io/en/latest/admin/installation/>

²⁰ Apiary - <https://apiary.io/>

²¹ API description document - <https://idraopendata.docs.apiary.io/api-description-document>

²² Open API Idra - <https://idraopendata.docs.apiary.io/#>

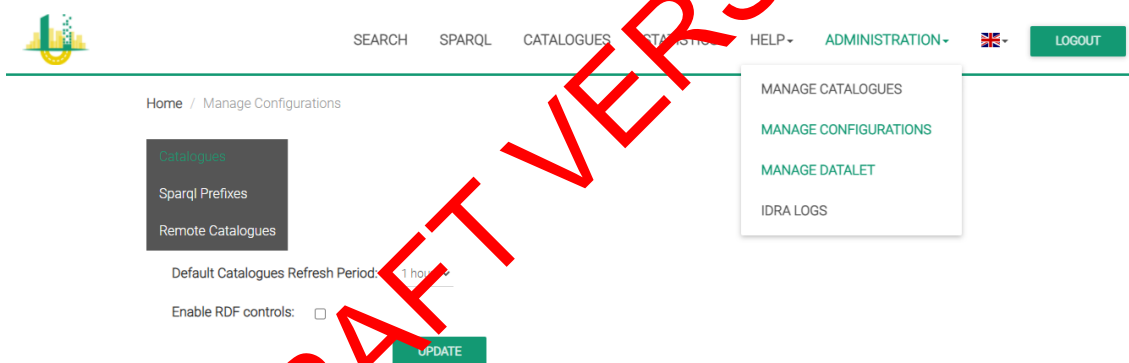
administrator user to manage catalogues federation and to manage configuration as depicted in Figure 9 and in Figure 10.



Home / Manage Catalogues

Active	Name	Country	Type	Level	Status	Datasets	Update Period	Last Update	
<input type="checkbox"/>	Amsterdam Pilot		DATA STORAGE	2	⚠	n/a	1 day	2021-07-02 10:41:07	🔄 ✎ ✕ ⬇
<input checked="" type="checkbox"/>	Bilbao Pilot		DATA STORAGE	2	✓	8	1 day	2021-09-20 11:42:37	🔄 ✎ ✕ ⬇
<input checked="" type="checkbox"/>	Helsinki Pilot		DATA STORAGE	2	⚠	0	1 day	2021-07-06 15:55:25	🔄 ✎ ✕ ⬇
<input checked="" type="checkbox"/>	Messina Pilot		DATA STORAGE	2	✓	1	1 day	2021-09-20 16:30:24	🔄 ✎ ✕ ⬇
<input type="checkbox"/>	Urbanite Integration Env		DATA STORAGE	2	✓	3	1 day	2021-06-29 09:42:56	🔄 ✎ ✕ ⬇

Figure 9 – Data Catalogue management of federated QDMS



Home / Manage Configurations

Catalogues

Sparql Prefixes

Remote Catalogues

Default Catalogues Refresh Period: 1 hour

Enable RDF controls: ☐

UPDATE

MANAGE CATALOGUES

MANAGE CONFIGURATIONS

MANAGE DATALET

IDRA LOGS

Figure 10 – Data Catalogue configuration management

The Data Catalogue provides also to the *end users* main functionalities to perform federated *metadata search* among federated catalogues (Figure 11).

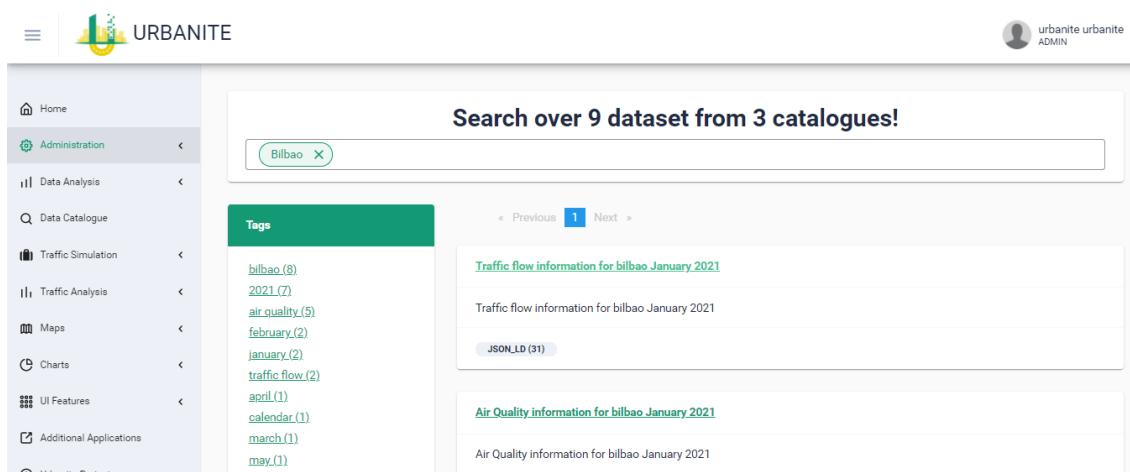


Figure 11 – Data Catalogue federated metadata search by tag

The Data Catalogue metadata search can filter the data using a facet approach. In particular, the following facets are available to filter on dataset metadata results by Tags, Formats, Licenses, etc. as depicted in Figure 12.

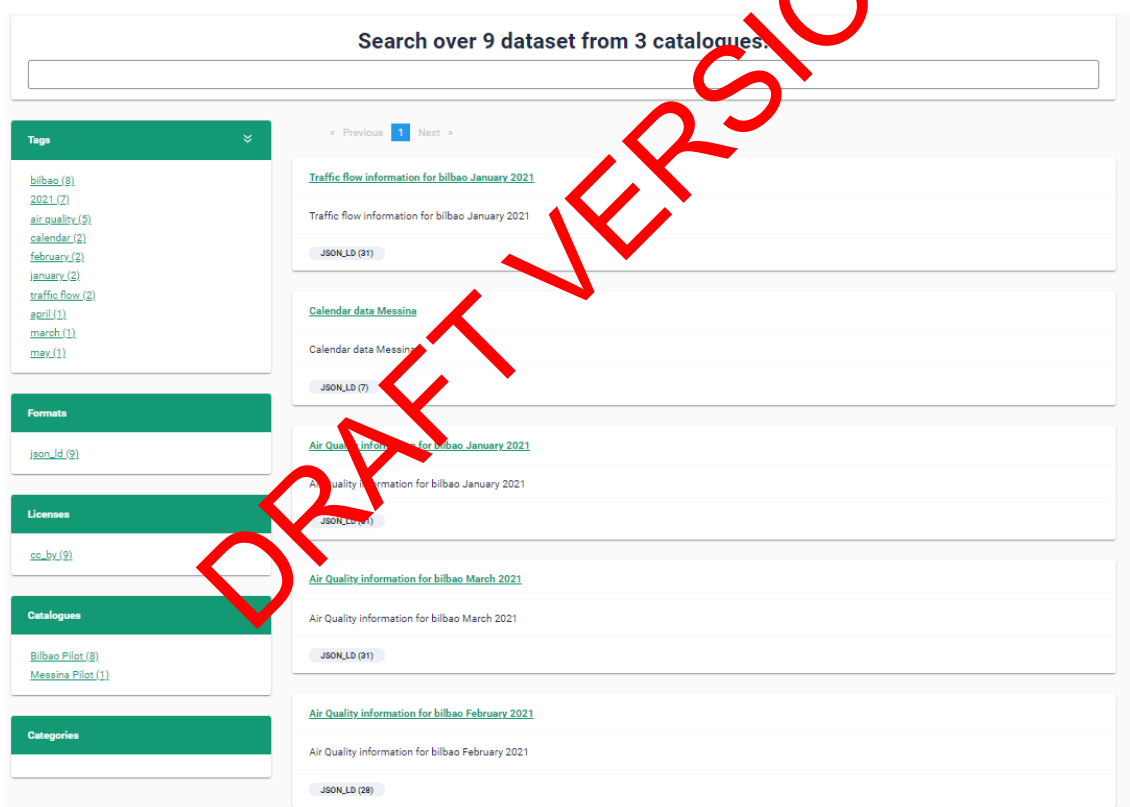


Figure 12 – Data Catalogue federated metadata search

The Data Catalogue metadata search page shows all metadata results, as default visualization, without any filtering. To perform search, the user can insert one or more keywords into the search bar as depicted in Figure 11.

Moreover, as previously reports, the user can filter the obtained results selecting a tab, license, etc. reported in the panels located on the left (Figure 12).

The results are reported in a list of datasets matching with the selected filter and the submitted keywords; for each result the following information is reported: *title*, *description* and *all available distributions*.

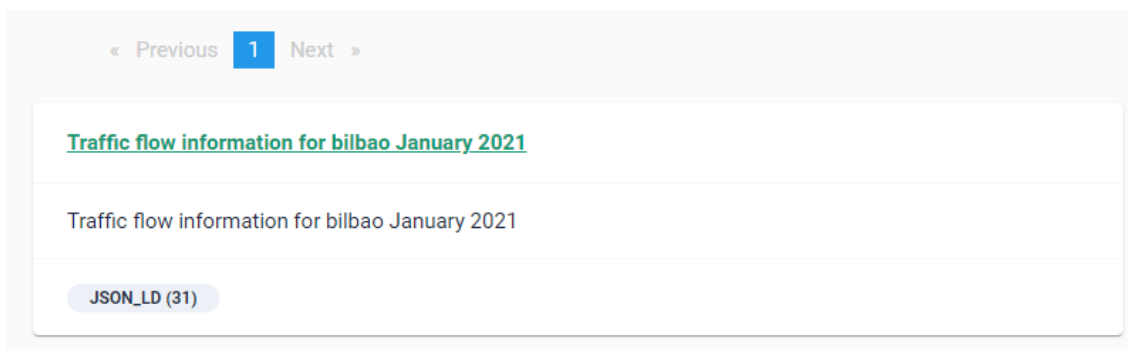


Figure 13 –Information fields of dataset' metadata search list

By clicking on its title, it is possible to access the detailed information of a dataset, as depicted in Figure 12. The detailed information includes the tags associated to the dataset, all the available distributions, the publisher name, the identifier of the dataset, etc.

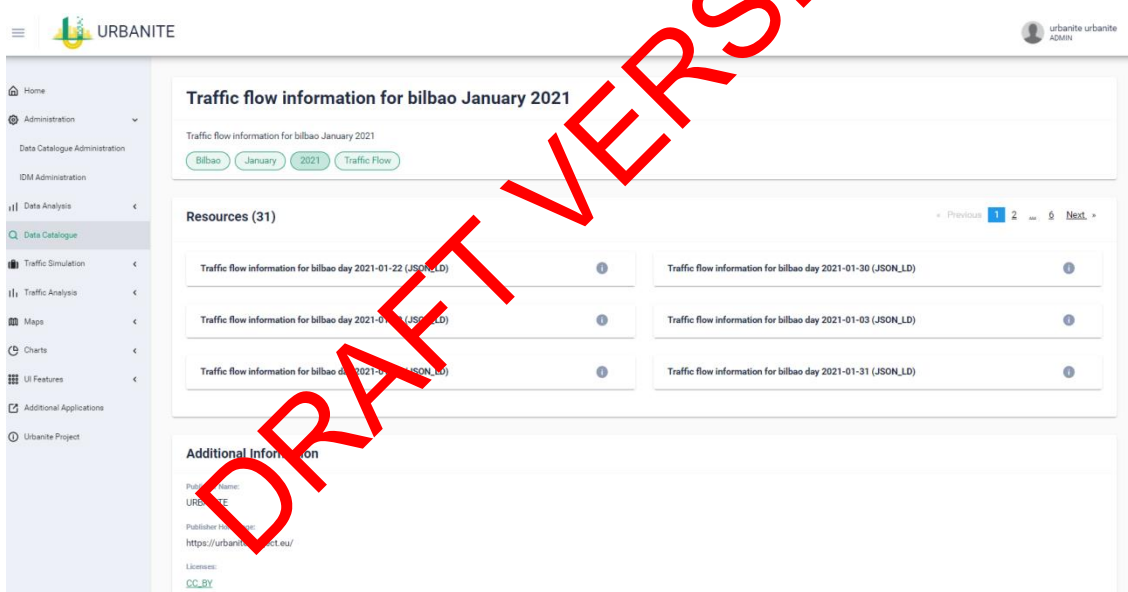


Figure 14– Data Catalogue federated metadata dataset detail view

Finally, by clicking on the information icon associated to each distribution, it is possible to access further details (Figure 16): the associated description, the format, the Access URL (for direct access to the distribution), the Download URL (to download the distribution), and the license.



Figure 15 – Data Catalogue- Distribution - Information Icon

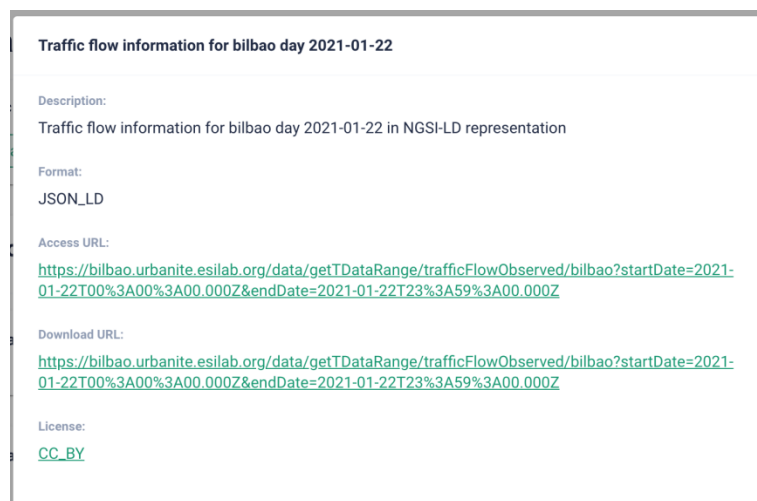


Figure 16 – Data Catalogue -details of a distribution

4.3 Licensing information

The Data Catalogue is licensed under Affero General Public License (AGPL) version 3. For further information read license²³ section on official Github of the project.

4.4 Download

Detail about this extension and about this new connector are provided in the Data Catalogue section of this same document. The source code of the Data Catalogue is available on Github²⁴.

²³IDRA License - <https://github.com/OPSILab/Idra#license>

²⁴Data Catalogue Github - <https://git.code.tecnalia.com/urbanite/private/wp3-data-management/urbanite-data-catalogue-src>

5 Conclusions

This document describes the technical details of the components associated with the data fusion, aggregation and storage integrated on the URBANITE Data Processing Platform. Data Fusion is referred to the integration of multiple raw data from the same or several different sources gathered from sensors is called data fusion. Data Aggregation is the process of gathering data and presenting it summarized or anonymized; for this first iteration, some aggregated functions are calculated related to bike and traffic estimation before performing the training of the Artificial Intelligence models in WP4. Finally, on a hand, the Data Storage & Retrieval capacities allow storing and retrieving datasets and associated metadata; on the other hand, the functionalities provided by the Data Catalogue allow to discover the datasets managed by the Data Storage & Retrieval and to federate additional data sources, offering a unique point to access datasets coming from scattered sources. This deliverable presents a technical description of the first release of the different components, providing technical details for future integration. Thus, these components are part of the data processing chain, integrating with those defined in deliverables D3.2 and D3.5, associated with data collection and curation.

DRAFT VERSION

6 References

- [1] D. L. Hall y J. Llinas, «An introduction to multisensor data fusion,» *Proceedings of the IEEE*, vol. 85, nº 1, p. 6–23, 1997.
- [2] F. Castanedo, «A Review of Data Fusion Techniques,» *The Scientific World Journal*, vol. 2013.

DRAFT VERSION

7 APPENDIX: Data models

Deliverable D3.4 described the common data models to be used in URBANITE. This section provides more detailed and updated information about those models and how they are being used by the Data Management Platform components in this first prototype.

7.1 Traffic Flow Observed

This model is based on the FIWARE's TrafficFlowObserved²⁵ data model, and contains information about an observation of traffic flow conditions at a certain place and time, such as the number of vehicles detected during the observation, the occupancy of the lanes, the location of the detection device, etc.

The model is very complete and collects a lot of information, but not all of it is of interest for WP4 or available in the cities' data sources. The subset of fields that are harvested are: id, address, averageVehicleSpeed, dateObserved, intensity, location and occupancy.

Example:

```
{
  "id": "urn:ngsi-ld:TrafficFlowObserved:59d4ce0d_17afe3d6_a6_2ce4",
  "address": {
    "addressCountry": "ES",
    "addressLocality": "Bilbao"
  },
  "averageVehicleSpeed": 22.0,
  "dateObserved": "2021-08-01T00:25:00.000Z",
  "intensity": 190.0,
  "location": {
    "coordinates": [
      [505545.02027647, 4789451.73590861],
      [505554.84865274, 4789477.77248377]
    ],
    "type": "Polygon"
  },
  "occupancy": 0.06,
  "type": "TrafficFlowObserved",
  "@context": [
    "https://smartdatamodels.org/context.jsonld",
    "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
  ]
}
```

7.2 Air Quality Observed

Based on FIWARE's AirQualityObserved²⁶ data model, it contains information about an observation of air quality conditions at a certain place and time, such as the carbon monoxide, nitrogen dioxide, material particles, temperature, humidity, etc.

The subset of fields that are harvested are: id, dateObserved, location, NO, NO2, NOX, PM10 and SO2.

Example:

```
{
  "id": "urn:ngsi-ld:AirQualityObserved:62:290820210500",
  "dateObserved": "2021-08-29T05:00:00Z",
  "location": {
```

²⁵ <https://github.com/smart-data-models/dataModel.Transportation/tree/master/TrafficFlowObserved>

²⁶ <https://github.com/smart-data-models/dataModel.Environment/tree/master/AirQualityObserved>

```

    "coordinates": [43.26750551179745, -2.935188110338201],
    "type": "Point"
  },
  "no": 2,
  "no2": 30,
  "nox": 33,
  "pm10": 13,
  "so2": 10,
  "type": "AirQualityObserved",
  "@context": [
    "https://smartdatamodels.org/context.jsonld",
    "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
  ]
}

```

7.3 WeatherObserved

This model is based on FIWARE's WeatherObserved²⁷ data model, containing information about an observation of weather conditions at a certain place and time, such as precipitation, humidity, temperature, UV index, etc.

The subset of fields that are harvested are: id, dateObserved, location, atmosphericPressure, precipitation, relativeHumidity, temperature, windDirection and windSpeed.

Example:

```

{
  "id": "urn:ngsi-ld:WeatherObserved::Bilbao-2021-06-30T07:00:00.00Z",
  "type": "WeatherObserved",
  "dateObserved": "2021-06-30T07:00:00.00Z",
  "temperature": 13.3,
  "precipitation": 0,
  "atmosphericPressure": 1024,
  "location": {
    "type": "Point",
    "coordinates": [-2.9349377987, 43.2641971992]
  },
  "relativeHumidity": 1,
  "windDirection": 120,
  "windSpeed": 10,
  "@context": [
    "https://smartdatamodels.org/context.jsonld",
    "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
  ]
}

```

7.4 Calendar and Day Specification

These two models have been defined within the scope of the Urbanite project to collect information on city calendars. The daySpecification model collects information about a specific day, such as whether it is a holiday, school day, the day of the week ... The Calendar model collects information for a whole year, including a list of references (ID) of each of its days (daySpecification).

7.4.1 DaySpecification

Table 2: Structure for day specification

Field	Description	Type	Mandatory
-------	-------------	------	-----------

²⁷ <https://smart-data-models.github.io/dataModel.Weather/WeatherObserved/doc/spec.md>

id	Unique identifier of the entity	Text	Y
type	Entity type	Text, must be 'DaySpecification'	Y
date	The date of this entity	Text in ISO8601 format	Y
description	A description of this item	Text	N
workingDay	If it is a working day (1) or not (0)	Numeric (0 or 1)	Y
schoolDay	If it is a school day (1) or not (0)	Numeric (0 or 1)	Y
publicHoliday	If it is a holiday (1) or not (0)	Numeric (0 or 1)	Y
weekDay	Day of the week	Numeric (1 to 7)	Y

Example

```
{
  "id": "urn:ngsi-ld:DaySpecification:Bilbao:2015_01-01",
  "type": "DaySpecification",
  "date": "2015-01-01",
  "description": "Año nuevo",
  "workingDay": 0,
  "schoolDay": 0,
  "publicHoliday": 3,
  "weekDay": 4,
  "createdAt": "2021-06-17T07:24:53.376Z",
  "modifiedAt": "2021-06-17T07:24:53.376Z",
  "@context": [
    "https://git.code.tecnalia.com/urbanite/public/-/raw/master/datamodels/calendar/ngsi-ld"
  ]
}
```

7.4.2 Calendar

Table 3. Structure for a calendar specification

Field	Description	Type	Mandatory
id	Unique identifier of the entity	Text	Y
type	Entity type	Text, must be 'Calendar'	Y
city	City of the calendar	Text	N
location	Location of the item	Geojson	N
year	Year of the calendar	Numeric	Y
days	Days that conform the calendar	List of Text	Y

Example (reduced to just 6 days, it could contain a whole year):

```
{
  "id": "urn:ngsi-ld:Calendar:Bilbao:2021",
  "type": "Calendar",
  "city": "Bilbao",
```

```

"location": {
  "coordinates": [-2.93609619140625, 43.26345626603949],
  "type": "Point"
},
"year": 2021,
"days": [
  "urn:ngsi-ld:DaySpecification:Bilbao:2021_01_01",
  "urn:ngsi-ld:DaySpecification:Bilbao:2021_01_02",
  "urn:ngsi-ld:DaySpecification:Bilbao:2021_01_03",
  "urn:ngsi-ld:DaySpecification:Bilbao:2021_01_04",
  "urn:ngsi-ld:DaySpecification:Bilbao:2021_01_05",
  "urn:ngsi-ld:DaySpecification:Bilbao:2021_01_06",
]
"@context": [
  "https://git.code.tecnalia.com/urbanite/public/-/raw/master/datamodels/calendar-ngsi.jsonld"
]
}

```

7.5 Metadata

Metadata must be provided in a DCAT-compliant format.

Example:

```

{
  "@graph": [
    {
      "@id": "https://urbanite-project.eu/ontology/URBANITE_PROJECT",
      "@type": "foaf:Organization",
      "homepage": "https://urbanite-project.eu/",
      "name": "URBANITE"
    },
    {
      "@id": "https://urbanite-project.eu/ontology/dataset/Bilbao_Calendar",
      "@type": "dcat:Dataset",
      "description": {"@language": "en", "@value": "Calendar data Bilbao"},
      "issued": "2021-06-17T09:24:50",
      "modified": "2021-06-17T09:28:53",
      "publisher": "https://urbanite-project.eu/ontology/URBANITE_PROJECT",
      "title": {"@language": "en", "@value": "Calendar data Bilbao"},
      "distribution": [
        "https://urbanite-project.eu/ontology/distribution/52c20f95-66a2-412d-9ac0-efe673707615",
        "https://urbanite-project.eu/ontology/distribution/5b9e9ed4-769c-435a-af0f-e25b41adb6f6",
        "https://urbanite-project.eu/ontology/distribution/1be969b1-88bd-4209-808a-004ccaef7c30"
      ],
      "keyword": ["Calendar", "Bilbao"]
    },
    {
      "@id": "https://urbanite-project.eu/ontology/distribution/1be969b1-88bd-4209-808a-004ccaef7c30",
      "@type": "dcat:Distribution",
      "description": "Calendar data Bilbao year 2016 in NGSI-LD representation",
      "format": "http://publications.europa.eu/resource/authority/file-type/JSON_LD",
      "license": "http://publications.europa.eu/resource/authority/licence/CC_BY",
      "title": "Calendar data Bilbao 2016",
      "accessURL": "https://bilbao.urbanite.esilab.org/data/getTData/calendar/bilbao?filters=%7B%22year%22%3D2016%7D"
    },
    {
      "@id": "https://urbanite-project.eu/ontology/distribution/52c20f95-66a2-412d-9ac0-efe673707615",
      "@type": "dcat:Distribution",
      "description": "Calendar data Bilbao year 2018 in NGSI-LD representation",
      "format": "http://publications.europa.eu/resource/authority/file-type/JSON_LD",
      "license": "http://publications.europa.eu/resource/authority/licence/CC_BY",
      "title": "Calendar data Bilbao 2018",
      "accessURL":

```

```

"https://bilbao.urbanite.esilab.org/data/getTData/calendar/bilbao?filters=%7B%22year%22%3D2015%7D"
},
{
  "@id": "https://urbanite-project.eu/ontology/distribution/5b9e9ed4-769c-435a-af0f-e25b41adb6f6f",
  "@type": "dcat:Distribution",
  "description": "Calendar data Bilbao year 2015 in NGSI-LD representation",
  "format": "http://publications.europa.eu/resource/authority/file-type/JSON_LD",
  "license": "http://publications.europa.eu/resource/authority/licence/CC_BY",
  "title": "Calendar data Bilbao 2015",
  "accessURL":
"https://bilbao.urbanite.esilab.org/data/getTData/calendar/bilbao?filters=%7B%22year%22%3D2015%7D"
}
],
"@context": {
  "name": {"@id": "http://xmlns.com/foaf/0.1/name"},
  "homepage": {"@id": "http://xmlns.com/foaf/0.1/homepage"},
  "accessURL": {"@id": "http://www.w3.org/ns/dcat#accessURL"},
  "title": {"@id": "http://purl.org/dc/terms/title"},
  "license": {"@id": "http://purl.org/dc/terms/license"},
  "format": {"@id": "http://purl.org/dc/terms/format"},
  "description": {"@id": "http://purl.org/dc/terms/description"},
  "distribution": {"@id": "http://www.w3.org/ns/dcat#distribution", "@type": "@id"},
  "modified": {"@id": "http://purl.org/dc/terms/modified", "@type": "@id"},
  "http://www.w3.org/2001/XMLSchema#dateTime": {"@id": "http://www.w3.org/2001/XMLSchema#dateTime"},
  "keyword": {"@id": "http://www.w3.org/ns/dcat#keyword"},
  "issued": {"@id": "http://purl.org/dc/terms/issued", "@type": "@id"},
  "http://www.w3.org/2001/XMLSchema#dateTime": {"@id": "http://www.w3.org/2001/XMLSchema#dateTime"},
  "publisher": {"@id": "http://purl.org/dc/terms/publisher", "@type": "@id"},
  "dct": "http://purl.org/dc/terms/",
  "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
  "xsd": "http://www.w3.org/2001/XMLSchema#",
  "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
  "dcat": "http://www.w3.org/ns/dcat#",
  "foaf": "http://xmlns.com/foaf/0.1/",
  "dc": "http://purl.org/dc/elements/1.1/"
}
}

```

8 APPENDIX: Storage & Retrieval API

8.1 Storage

8.1.1 insertTData (POST)

Table 4: API for data insertion

/insertTData	Adds new data of a specific type to the database of a city.
Method	
POST	
Input Params (* means mandatory)	
model* (path param)	Text with the type of the data to be inserted. Must be one of the implemented data types, currently: <ul style="list-style-type: none"> trafficFlowObserved daySpecification calendar airQualityObserved weatherObserved

city* (path param)	Text, with the city (use case) to which the data correspond. Must be one of: <ul style="list-style-type: none"> • bilbao • messina • helsinki • amsterdam
data* (request body)	Text, in a JSON array format, with the data to be inserted. These data must be in the format corresponding to the type of data indicated in the "model" parameter.
Success response	
200	<p>If all the input parameters are right, the method will return this code, and the JSON response will contain three fields with the details of the operations done:</p> <ul style="list-style-type: none"> • <i>inserted</i>: array with the IDs of the elements inserted successfully, • <i>updated</i>: array with the IDs of the elements updated. • <i>notInserted</i>: array with the IDs of the elements that couldn't be inserted. In this case, also will have a field <i>reason</i> with the description of the error. <p>Example:</p> <pre>{ "inserted": [{ "id": "urn:ngsi-ld:TrafficFlowObserved:5d993408_179e63999c7_-363a" }, { "id": "urn:ngsi-ld:TrafficFlowObserved:5d993408_179e63999c7_-3603" }], "notInserted": [{ "id": "urn:ngsi-ld:TrafficFlowObserved:5d993408_179e63999c7_1dec", "reason": "Wrong input data, missing some required field(s) or wrong values." }, { "id": "urn:ngsi-ld:TrafficFlowObserved:5d993408_179e63999c7_1df6", "reason": "Wrong input data, missing some required field(s) or wrong value." }], "updated": [{ "id": "urn:ngsi-ld:TrafficFlowObserved:740c2b3d_17ba0648be0_4968" }, { "id": "urn:ngsi-ld:TrafficFlowObserved:740c2b3d_17ba0648be0_4969" }] }</pre>
Error response	

400	<p>Bad Request</p> <p>The method checks if the data sent is in the required format (list of elements) and if each element to be inserted corresponds to the indicated model. It also checks if the <i>model</i> and <i>city</i> parameters are the expected values.</p> <p>If any of these checks fail, a Bad Request code will be returned. In this case, the method will return a JSON response, with just one field <i>Error</i>, that will contain a description of the error.</p> <p>Example:</p> <pre>{ "Error": "Input data is not in required format (list of 'Traffic Flow Observation' objects)" }</pre>
Sample call	
<pre>curl -X POST "http://[server:port]/data/insertTData/trafficFlowObserved/bilbao" -H "accept: application/json" -H "Content-Type: application/json" -d "[{ \"id\": \"urn:ngsi-ld:TrafficFlowObserved:740c2b3d_17ba0648be0_4968\", \"address\": { \"addressCountry\": \"ES\", \"addressLocality\": \"Bilbao\" }, \"averageVehicleSpeed\": 26, \"dateObserved\": \"2021-09-01T08:05:00Z\", \"intensity\": 77, \"location\": { \"coordinates\": [[[505863.2934143, 430330.91052876], [505864.01, 4790329.65]]], \"type\": \"Polygon\" }, \"occupancy\": 1, \"type\": \"TrafficFlowObserved\", \"@context\": [\"https://smartdatamodels.org/context.jsonld\", \"https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld\"] }, { \"id\": \"urn:ngsi-ld:TrafficFlowObserved:740c2b3d_17ba0648be0_4969\", \"address\": { \"addressCountry\": \"ES\", \"addressLocality\": \"Bilbao\" }, \"averageVehicleSpeed\": 36, \"dateObserved\": \"2021-09-01T08:05:00Z\", \"intensity\": 481, \"location\": { \"coordinates\": [[[504426.40986984, 4790030.56457333], [504426.745, 4790032.297]]], \"type\": \"Polygon\" }, \"occupancy\": 0.04, \"type\": \"TrafficFlowObserved\", \"@context\": [\"https://smartdatamodels.org/context.jsonld\", \"https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld\"] }]</pre> <p><i>city model</i></p> <p>First element to be inserted</p> <p>Second element to be inserted</p>	

8.1.2 updateTData (PUT)

Table 5: API for data update

/updateTData	Updates a specific record (identified by its ID field) of the database.
Method	
PUT	
Input Params (* means mandatory)	
model* (path param)	Text, with the data type of the element to be updated. Must be one of the implemented data types, currently: <ul style="list-style-type: none"> • trafficFlowObserved • daySpecification • calendar • airQualityObserved • weatherObserved
city* (path param)	Text, with the city (use case) to which the data corresponds. Must be one of: <ul style="list-style-type: none"> • bilbao • messina • helsinki • amsterdam
id* (path param)	ID of the element to be updated.
data* (request body)	Text, in a JSON format, with the data to be updated. This data must be in the format corresponding to the type of data indicated in the "model" parameter.
Success response	

200	<p>If all the input parameters are right, the method will return this code, and the JSON response will contain one field <i>updatedData</i> with the new data of the element once updated.</p> <p>Example:</p> <pre> { "updatedData": { "address": { "addressCountry": "ES", "addressLocality": "Bilbao" }, "averageVehicleSpeed": 16, "dateObserved": "2021-08-17T00:45:00Z", "intensity": 36, "location": { "coordinates": [[[504417.9371092392, 4790030.564573327], [504453.27293873084, 4790030.564573327], [504453.27293873084, 4790133.21077546], [504417.9371092392, 4790133.21077546]]], "type": "Polygon" }, "name": "273", "type": "TrafficFlowObserved", "@context": ["https://smartdatamodels.org/context.jsonld", "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"], "dateCreated": "2021-08-17T09:37:43.332Z", "dateModified": "2021-09-07T07:31:35.263Z", "id": "urn:ngsi-ld:TrafficFlowObserved:273:070920210045AAAA" } } </pre>
Error response	
400	<p>Bad Request</p> <p>The method checks if the data sent is in JSON format and if it corresponds to the indicated model. It also checks if the ID field of the data (mandatory field) is the same as the one passed as parameter, as well if the <i>model</i> and <i>city</i> parameters are the expected values.</p> <p>If any of these checks fail, a Bad Request code will be returned. In this case, the method will return a JSON response, with just one field <i>Error</i>, that will contain a description of the error.</p> <p>Examples:</p> <pre> { "Error": "Input data is not in required format ('Traffic Flow Observation' object)" } { "Error": "Wrong input data, IDs are different." } </pre>
404	<p>Not Found</p> <p>If the element to be update doesn't exist, a 404 code will be returned, with a JSON response with the error:</p> <pre> { "Error": "Document 'urn:ngsi-ld:TrafficFlowObserved:273:07092023456' not found." } </pre>
Sample call	

```

curl -X PUT
"http://[server:port]/data/updateTData/trafficFlowObserved/bilbao/urn%3Aangsi-ld%3ATrafficFlowObserved%3A273%3A070920210045AAAA"
-H "accept: application/json"
-H "Content-Type: application/json"
-d "
{
  \"id\": \"urn:ngsi-ld:TrafficFlowObserved:273:070920210045AAAA\",
  \"address\": {
    \"addressCountry\": \"ES\",
    \"addressLocality\": \"Bilbao\"
  },
  \"averageVehicleSpeed\": 16,
  \"dateObserved\": \"2021-08-17T00:45:00Z\",
  \"intensity\": 36,
  \"location\": {
    \"coordinates\": [[504417.9371092392, 4790030.564573327],
    [504453.27293873084, 4790030.564573327],
    [504453.27293873084, 4790133.21077546],
    [504417.9371092392, 4790133.21077546]],
    \"type\": \"Polygon\"
  },
  \"name\": \"273\",
  \"type\": \"TrafficFlowObserved\",
  \"@context\": [
    \"https://smartdatamodels.org/context.jsonld\",
    \"https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld\"
  ]
}
"

```

city model id

Data to be updated

8.2 Retrieval

8.2.1 getTData (GET)

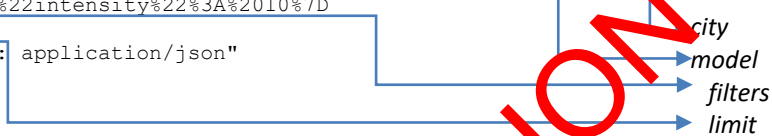
Table 6: API for data retrieval

/getTData	Gets data of the specified type from the database. Some filters can be applied to model fields.
Method	
GET	
Input Params (* means mandatory)	
model* (path param)	Text, with the data type of the element to be updated. Must be one of the implemented data types. Currently: <ul style="list-style-type: none"> • trafficFlowObserved • daySpecification • calendar • airQualityObserved • weatherObserved
city* (path param)	Text, with the city (use case) to which the data correspond. Must be one of: <ul style="list-style-type: none"> • bilbao • messina • helsinki • amsterdam

filters (query param)	<p>Different filters to be applied to the search of the records to be returned. They must be in JSON format and match the names of the fields of the model. Otherwise, a BAD_REQUEST error will be returned.</p> <p>Example:</p> <pre>{ "occupancy": 10, "intensity": 20 }</pre> <p>both fields, <i>occupancy</i> and <i>intensity</i> are part of the <i>trafficFlowObserved</i> model.</p>
limit (query param)	<p>Number of records to be retrieved. If not set, the default number of records (1000) will be returned.</p>
Success response	

DRAFT VERSION

200	<p>If all the input parameters are right, the method will return this code, and the response will be a JSON array with the list of the elements requested.</p> <p>Example:</p> <pre>[{ "id": "urn:ngsi-ld:TrafficFlowObserved:740c2b3d_17bbf3cb8f8_851", "address": { "addressCountry": "ES", "addressLocality": "Bilbao" }, "averageVehicleSpeed": 0, "dateObserved": "2021-09-07T07:45:00Z", "intensity": 10, "location": { "coordinates": [[[504169.87,4790169.594],[504215.57915011,4790128.87897186],[504268.78134 883,4790081.60210874]]], "type": "Polygon" }, "occupancy": 0, "type": "TrafficFlowObserved", "@context": ["https://smartdatamodels.org/context.jsonld", "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"], "dateCreated": "2021-09-07T07:53:43.872Z", "dateModified": "2021-09-07T07:53:43.872Z" }, { "id": "urn:ngsi-ld:TrafficFlowObserved:59d1ce0d_17bbf0b8c3f_-5ba8", "address": { "addressCountry": "ES", "addressLocality": "Bilbao" }, "averageVehicleSpeed": 4, "dateObserved": "2021-09-07T06:50:00Z", "intensity": 10, "location": { "coordinates": [[[505482.00123305,478960.88571801],[505595.37956903,4789700.8144343],[50 5621.18245562,4789721.03079363],[505621.339,4789721.18]]], "type": "Polygon" }, "occupancy": 0, "type": "TrafficFlowObserved", "@context": ["https://smartdatamodels.org/context.jsonld", "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"], "dateCreated": "2021-09-07T06:57:32.011Z", "dateModified": "2021-09-07T06:57:32.011Z" }]</pre>
Error response	

400	<p>Bad Request</p> <p>The method checks if the filters sent are in the required format (JSON) and if its content are fields of the specified data model. It also checks if the <i>model</i> and <i>city</i> parameters are the expected values, and if the <i>limit</i> parameter (if set) is greater than 0.</p> <p>If any of these checks fail, a Bad Request code will be returned. In this case, the method will return a JSON response, with just one field <i>Error</i>, that will contain a description of the error.</p> <p>Example:</p> <pre>{ "Error": "Model {trafficFlowObserved} has not a field 'speed'" }</pre>
Sample call	
<pre>curl -X GET "http://[server:port]/data/getTData/trafficFlowObserved/bilbao? filters=%7B%22intensity%22%3A%2010%7D &limit=2" -H "accept: application/json"</pre> 	

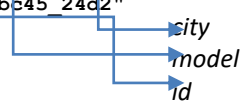
8.2.2 getTData (single record) (GET)

Table 7: API for data retrieval (specific record)

/getTData (single record)	Gets a specific record from the database, identified by its ID.
Method	
GET	
Input Params (* means mandatory)	
model* (path param)	Text, with the data type of the element to be updated. Must be one of the implemented data types, currently: <ul style="list-style-type: none"> • trafficFlowObserved • daySpecification • calendar • airQualityObserved • weatherObserved
city* (path param)	Text, with the city (use case) to which the data correspond. Must be one of: <ul style="list-style-type: none"> • bilbao • messina • helsinki • amsterdam

id* (path param)	ID of the element to be updated.
Success response	
200	<p>If the element requested exists, the method will return this code, and the JSON response will contain the element data.</p> <p>Example:</p> <pre>{ "id": "urn:ngsi-ld:TrafficFlowObserved:740c2b3d_17b9e3ebc45_24d2", "address": { "addressCountry": "ES", "addressLocality": "Bilbao" }, "averageVehicleSpeed": 0, "dateObserved": "2021-08-31T22:00:00Z", "intensity": 9, "location": { "coordinates": [[[505863.2934643, 4790330.9105287], [505864.01, 4790329.365], [505864.637, 4790327.54]]], "type": "Polygon" }, "occupancy": 0, "type": "TrafficFlowObserved", "@context": ["https://smartdatamodels.org/context.jsonld", "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"], "dateCreated": "2021-08-31T22:09:23.993Z", "dateModified": "2021-08-31T22:09:23.993Z" }</pre>
Error response	
400	<p>Bad Request</p> <p>The method checks if the <i>model</i> and <i>city</i> parameters are the expected values.</p> <p>If any of these checks fail, a Bad Request code will be returned. In this case, the method will return a JSON response, with just one field <i>Error</i>, that will contain a description of the error.</p> <p>Example:</p> <pre>{ "Error": "Invalid value 'trafficFlow' for parameter 'model'" }</pre>
404	<p>Not Found</p> <p>If the element requested doesn't exist, a 404 code will be returned, with a JSON response with the error:</p> <pre>{ "Error": "Document 'urn:ngsi-ld:TrafficFlowObserved:273:07092023456' not found." }</pre>
Sample call	

```
curl -X GET
"http://[server:port]/data/getTData/trafficFlowObserved/bilbao/
urn%3AAngsi-ld%3ATrafficFlowObserved%3A740c2b3d_17b9e3eb-45_24d2"
-H "accept: application/json"
```



8.2.3 getTDataRange (GET)

Table 8: API for data retrieval (time range)

/getTDataRange	Gets data of a specific model from the database within a specific time range.
Method	
GET	
Input Params (* means mandatory)	
model* (path param)	Text, with the data type of the element to be updated. Must be one of the implemented data types, currently: <ul style="list-style-type: none"> • trafficFlowObserved • daySpecification • calendar • airQualityObserved • weatherObserved
city* (path param)	Text, with the city (use case) to which the data correspond. Must be one of: <ul style="list-style-type: none"> • bilbao • messina • helsinki • amsterdam
startDate* (query param)	Date and time (ISO8601 ²⁸ UTC format) from which to get the data. Mandatory if parameter <i>endDate</i> is not present. Example: 2021-01-07T00:00:00.000Z
endDate* (query param)	Date and time (ISO8601 UTC format) until which to get the data. Mandatory if parameter <i>startDate</i> is not present. Example: 2021-01-08T22:45:00.000Z
limit (query param)	Number of records to be retrieved. If not set, the default number of records (1000) will be returned.
Success response	

²⁸ <https://www.iso.org/iso-8601-date-and-time-format.html>

200	<p>If all the input parameters are right, the method will return this code, and the response will be a JSON array with the list of the elements requested.</p> <p>Example:</p> <pre>[{ "id": "urn:ngsi-ld:TrafficFlowObserved:740c2b3d_17bb86bfdd2_28f0", "address": { "addressCountry": "ES", "addressLocality": "Bilbao" }, "averageVehicleSpeed": 0, "dateObserved": "2021-09-06T00:00:00Z", "intensity": 0, "location": { "coordinates": [[505863.2934643, 4790330.91052876], [505875.24029728, 4790333.17459119], [505863.2934643, 4790330.91052876]]], "type": "Polygon" }, "occupancy": 0, "type": "TrafficFlowObserved", "@context": ["https://smartdatamodels.org/context.jsonld", "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"], "dateCreated": "2021-09-06T00:07:19.758Z", "dateModified": "2021-09-06T00:07:19.758Z" }, { "id": "urn:ngsi-ld:TrafficFlowObserved:740c2b3d_17bb86bfdd2_28f1", "address": { "addressCountry": "ES", "addressLocality": "Bilbao" }, "averageVehicleSpeed": 0, "dateObserved": "2021-09-06T00:00:00Z", "intensity": 18, "location": { "coordinates": [[504426.40986984, 4790030.56457333], [504425.67779232, 4790043.20283264], [504426.40986984, 4790030.56457333]]], "type": "Polygon" }, "occupancy": 0, "type": "TrafficFlowObserved", "@context": ["https://smartdatamodels.org/context.jsonld", "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"], "dateCreated": "2021-09-06T00:07:19.760Z", "dateModified": "2021-09-06T00:07:19.760Z" }]</pre>
Error response	

400	<p>Bad Request</p> <p>The method makes several checks of the parameters. It checks if the <i>model</i> and <i>city</i> parameters are the expected values, and if the <i>limit</i> parameter (if set) is greater than 0. Also checks that at least one of the parameters <i>startDate</i> and <i>endDate</i> is set, and if so, checks that they are in ISO8601 format.</p> <p>If any of these checks fail, a Bad Request code will be returned, and will return a JSON response, with just one field <i>Error</i>, that will contain a description of the error.</p> <p>Examples:</p> <pre>{ "Error": "No time range specified. 'startDate' and/or 'endDate' must be indicated." }</pre> <pre>{ "Error": "Invalid value '2021/08/01' for parameter 'startDate'" }</pre>
Sample call	
<pre>curl -X GET "http://[server:port]/data/getTDataRange/trafficFlowObserved/bilbao? startDate=2021-09-05T00%3A00%3A00.000Z &endDate=2021-09-06T00%3A00%3A00.000Z &limit=2" -H "accept: application/json"</pre> <p>city model startDate endDate limit</p> <pre>curl -X GET "http://[server:port]/data/getTDataRange/trafficFlowObserved/bilbao? startDate=2021-09-05T00%3A00%3A00.000Z &limit=2" -H "accept: application/json"</pre> <p>city model startDate limit</p>	

8.2.4 getSupportedDataModels (GET)

Table 9: API for the retrieval of the data models information

/getSupportedDataModels	Returns information about all the data models that are currently implemented.
Method	
GET	
Input Params (* means mandatory)	
None	
Success response	
200	Returns a JSON array with the information of the data models implemented. Each element (model) will have the following fields:

- **id:** identifier of the model, that is usually used as *model* parameter in the other services.
- **name:** full name of the model.
- **description:** brief description of the model.
- **reference:** link to the official reference of the model (i.e: FIWARE models). If the model has been developed specifically for the project, the reference will be empty.
- **example:** an example of the structure of the model.

Example (reduced to two models only):

```
[
  {
    "id": "trafficFlowObserved",
    "name": "Traffic Flow Observed",
    "description": "An observation of traffic flow conditions at a certain place and time.",
    "reference": "https://github.com/smart-data-models/dataModel.Transportation/tree/master/TrafficFlowObserved",
    "example": {
      "@context": [
        "https://smartdatamodels.org/context.jsonld",
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
      ],
      "address": {
        "addressCountry": "ES",
        "addressLocality": "Valladolid",
        "streetAddress": "Avenida de Salamanca",
        "type": "PostalAddress"
      },
      "averageHeadwayTime": 10.5,
      "averageVehicleLength": 4.87,
      "averageVehicleSpeed": 52.7,
      "dateObserved": "2015-12-07T11:10:00Z",
      "id": "urn:ngsi-ld:TrafficFlowObserved:TrafficFlowObserved-Valladolid-osm-60821110",
      "intensity": 197,
      "laneDirection": "forward",
      "laneId": 1,
      "location": {
        "coordinates": [[-4.73735395519672, 41.6538181849672], [-4.7341485165193, 41.6600594193478], [-4.73447575302641, 41.659585195093]],
        "type": "LineString"
      },
      "occupancy": 0.76,
      "reversedLane": false,
      "type": "TrafficFlowObserved"
    },
    {
      "id": "calendar",
      "name": "Calendar",
      "description": "Information about calendars: year, city, days...",
      "reference": "",
      "example": {
        "example": [
          {
            "id": "urn:ngsi-ld:Calendar:Bilbao:2015",
            "type": "Calendar",
            "city": "Bilbao",
            "location": {
              "coordinates": [-2.93609619140625, 43.26345626603949],
              "type": "Point"
            },
            "year": 2015,
            "days": [
              "urn:ngsi-ld:DaySpecification:Bilbao:2015_01_01",
              "urn:ngsi-ld:DaySpecification:Bilbao:2015_01_02"
            ],
            "createdAt": "2021-05-20T09:32:08.809Z",

```

	<pre> "modifiedAt": "2021-05-20T09:52:04.255Z", "@context": ["https://git.code.tecnalia.com/urbanite/public/- /raw/master/datamodels/calendar-ngsi.jsonld", "https://git.code.tecnalia.com/urbanite/public/- /raw/master/datamodels/calendar-ngsi.jsonld2"] }, { "id": "urn:ngsi-ld:DaySpecification:Bilbao:2015_01_01", "type": "DaySpecification", "date": "2015-01-01", "description": "AÃfÃto nuevo", "workingDay": 0, "schoolDay": 0, "publicHoliday": 3, "weekDay": 4, "createdAt": "2021-05-24T13:26:41.828Z", "modifiedAt": "2021-05-25T08:49:11.841Z", "@context": ["https://git.code.tecnalia.com/urbanite/public/- /raw/master/datamodels/calendar-ngsi.jsonld"] }, { "id": "urn:ngsi-ld:DaySpecification:Bilbao:2015_01_02", "type": "DaySpecification", "date": "2015-01-02", "description": "", "workingDay": 1, "schoolDay": 1, "publicHoliday": 0, "weekDay": 5, "createdAt": "2021-05-25T08:26:22.811Z", "modifiedAt": "2021-05-25T08:49:11.946Z", "@context": ["https://git.code.tecnalia.com/urbanite/public/- /raw/master/datamodels/calendar-ngsi.jsonld"] }] } }] </pre>
Sample call	
<pre> curl -X GET "http://localhost:8080/getSupportedDataModels" -H "accept: application/json" </pre>	

8.3 Metadata

8.3.1 dataset (PUT)

Table 10: API for the insert and update of metadata

/dataset	Adds new metadata of a dataset into the database, or updates the metadata if already exists for that id.
Method	
PUT	
Input Params (* means mandatory)	

id* (query param)	Unique identifier of the metadata.
data* (request body)	Text, in JSON format, with the metadata to be inserted.
Success response	
200	<p>If all the input parameters are right, the method will return this code, and the JSON response will contain three fields with the details of the operations done:</p> <ul style="list-style-type: none"> • <i>inserted</i>: array with the ID of the metadata if it has been inserted successfully. • <i>updated</i>: array with the ID of the metadata if it has been updated. • <i>notInserted</i>: array with the ID of the metadata if it couldn't be inserted. In this case, also will have a field <i>reason</i> with the description of the error. <p>Example:</p> <pre>{ "inserted": [{ "id": "6bb9c361_177a635e86a2333333" }], "notInserted": [], "updated": [] }</pre>

DRAFT VERSION

Error response	
400	<p>Bad Request</p> <p>The method checks that the ID and the metadata has been included, and that this one is in JSON format.</p> <p>If any of these checks fail, a Bad Request code will be returned. In this case, the method will return a JSON response, with just one field <i>Error</i>, that will contain a description of the error.</p> <p>Example:</p> <pre>{ "Error": "Input metadata is not in JSON format" }</pre>
Sample call	

DRAFT VERSION

```

curl -X PUT
"http://[server:port]/data/dataset?id=6bb9c361_177a635e86a2"
-H "accept: application/json"
-H "Content-Type: application/json"
-d "{
  \"_id\": \"6bb9c361_177a635e86a2\",
  \"@graph\": [
    {
      \"@id\": \"https://urbanite-project.eu/ontology/URBANITE_PROJECT\",
      \"@type\": \"foaf:Organization\",
      \"homepage\": \"https://urbanite-project.eu/\",
      \"name\": \"URBANITE\"
    },
    {
      \"@id\": \"https://urbanite-project.eu/ontology/dataset/Bilbao_Calendar\",
      \"@type\": \"dcat:Dataset\",
      \"description\": {
        \"@language\": \"en\",
        \"@value\": \"Calendar data Bilbao\"
      },
      \"issued\": \"2021-05-12T10:36:46\",
      \"modified\": \"2021-05-12T15:36:46\",
      \"publisher\": \"https://urbanite-project.eu/ontology/URBANITE_PROJE\",
      \"title\": {
        \"@language\": \"en\",
        \"@value\": \"Calendar data Bilbao\"
      },
      \"distribution\": [
        \"https://urbanite-project.eu/ontology/distribution/a732f6c6-fcd8-4962-8aa9-db7d913a20ae\"
      ],
      \"keyword\": [\"Calendar\", \"Bilbao\"]
    },
    {
      \"@id\": \"https://urbanite-project.eu/ontology/distribution/a732f6c6-fcd8-4962-8aa9-db7d913a20ae\",
      \"@type\": \"dcat:Distribution\",
      \"description\": \"Calendar data Bilbao year 2015 in NGSI-LD representation\",
      \"format\": \"http://publications.europa.eu/resource/authority/file-type/JSON_LD\",
      \"license\": \"http://publications.europa.eu/resource/authority/licence/CC_BY\",
      \"title\": \"Calendar data Bilbao 2015\",
      \"accessURL\": \"http://storageAPI-to-bedefined/2015\"
    }
  ],
  \"@context\": {
    \"name\": {\"@id\": \"http://xmlns.com/foaf/0.1/name\"},
    \"homepage\": {\"@id\": \"http://xmlns.com/foaf/0.1/homepage\"},
    \"accessURL\": {\"@id\": \"http://www.w3.org/ns/dcat#accessURL\"},
    \"title\": {\"@id\": \"http://purl.org/dc/terms/title\"},
    \"license\": {\"@id\": \"http://purl.org/dc/terms/license\"},
    \"format\": {\"@id\": \"http://purl.org/dc/terms/format\"},
    \"description\": {\"@id\": \"http://purl.org/dc/terms/description\"},
    \"distribution\": {\"@id\": \"http://www.w3.org/ns/dcat#distribution\"},
    \"type\": {\"@id\": \"\"},
    \"keyword\": {\"@id\": \"http://www.w3.org/ns/dcat#keyword\"},
    \"issued\": {\"@id\": \"http://purl.org/dc/terms/issued\"},
    \"type\": {\"@id\": \"http://www.w3.org/2001/XMLSchema#dateTime\"},
    \"publisher\": {\"@id\": \"http://purl.org/dc/terms/publisher\"},
    \"type\": {\"@id\": \"\"},
    \"modified\": {\"@id\": \"http://purl.org/dc/terms/modified\"},
    \"type\": {\"@id\": \"http://www.w3.org/2001/XMLSchema#dateTime\"},
    \"dct\": \"http://purl.org/dc/terms/\",
    \"rdf\": \"http://www.w3.org/1999/02/22-rdf-syntax-ns#\",
    \"xsd\": \"http://www.w3.org/2001/XMLSchema#\",
    \"rdfs\": \"http://www.w3.org/2000/01/rdf-schema#\",
    \"dcat\": \"http://www.w3.org/ns/dcat#\",
    \"foaf\": \"http://xmlns.com/foaf/0.1/\",
    \"dc\": \"http://purl.org/dc/elements/1.1/\"
  }
}

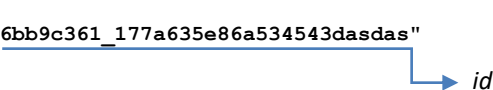
```

Id

*M
e
t
a
d
a
t
a*

8.3.2 dataset (DELETE)

Table 11: API for the deletion of metadata

/dataset	Delete the metadata of a dataset from the database.
Method	
DELETE	
Input Params (* means mandatory)	
id* (query param)	Unique identifier of the metadata.
Success response	
200	<p>If the metadata with the <i>id</i> passed is deleted, the method will return this code, and the JSON response will contain one field <i>deleted</i> with the <i>id</i> of the metadata deleted.</p> <p>Example:</p> <pre>{ "deleted": "6bb9c361_177a635e86a534543" }</pre>
Error response	
400	<p>Bad Request</p> <p>The method checks that the <i>id</i> parameter has been included, otherwise Bad Request code will be returned.</p> <p>In this case, the method will return a JSON response, with just one field <i>Error</i>, that will contain a description of the error.</p> <p>Example:</p> <pre>{ "Error": "Parameter 'id' is required." }</pre>
404	<p>Not found</p> <p>If the metadata with the <i>id</i> passed doesn't exist, this error will be returned with a JSON response with the error.</p> <p>Example:</p> <pre>{ "Error": "Dataset '6bb9c361_177a635e86a234' not found." }</pre>
Sample call	
<pre>curl -X DELETE "http://[server:port]/data/dataset?id=6bb9c361_177a635e86a534543dasdas" -H "accept: application/json"</pre> 	

8.3.3 getDataset (GET)

/getDataset	Gets the metadata of a specific dataset from the database.
Method	
GET	
Input Params (* means mandatory)	
id* (query param)	Unique identifier of the metadata.
Success response	
200	<p>If the metadata with the <i>id</i> passed exists, the method will return this code, and the JSON response will contain the metadata stored in the database in JSON format.</p> <p>Example:</p> <pre> { "@graph": [{ "@id": "https://urbanite-project.eu/ontology/URBANITE_PROJECT", "@type": "foaf:Organization", "homepage": "https://urbanite-project.eu/", "name": "URBANITE" }, { "@id": "https://urbanite-project.eu/ontology/dataset/Bilbao_Calendar", "@type": "dcat:Dataset", "description": {"@language": "en", "@value": "Calendar data Bilbao"}, "issued": "2021-05-12T10:36:46", "modified": "2021-05-12T15:36:46", "publisher": "https://urbanite-project.eu/ontology/URBANITE_PROJECT", "title": {"@language": "en", "@value": "Calendar data Bilbao"}, "distribution": ["https://urbanite-project.eu/ontology/distribution/a732f6c6-fcd8-4962-8aa9-db7d913a20ae", "https://urbanite-project.eu/ontology/distribution/059fd3cc-92b3-4f3d-97de-d050ae022eb5"], "keyword": ["Calendar", "Bilbao"] }, { "@id": "https://urbanite-project.eu/ontology/distribution/059fd3cc-92b3-4f3d-97de-d050ae022eb5", "@type": "dcat:Distribution", "description": "Calendar data Bilbao year 2016 in NGSI-LD representation", "format": "http://publications.europa.eu/resource/authority/file-type/JSON_LD", "license": "http://publications.europa.eu/resource/authority/licence/CC_BY", "title": "Calendar data Bilbao 2016", "accessURL": "http://storageAPI-to-be-defined/2016" }, { "@id": "https://urbanite-project.eu/ontology/distribution/a732f6c6-fcd8-4962-8aa9-db7d913a20ae", "@type": "dcat:Distribution", "description": "Calendar data Bilbao year 2015 in NGSI-LD representation", "format": "http://publications.europa.eu/resource/authority/file-type/JSON_LD", "license": </pre>

	<pre> "http://publications.europa.eu/resource/authority/licence/CC_BY", "title": "Calendar data Bilbao 2015", "accessURL": "http://storageAPI-to-bedefined/2015" }], "@context": { "name": {"@id": "http://xmlns.com/foaf/0.1/name"}, "homepage": {"@id": "http://xmlns.com/foaf/0.1/homepage"}, "accessURL": {"@id": "http://www.w3.org/ns/dcat#accessURL"}, "title": {"@id": "http://purl.org/dc/terms/title"}, "license": {"@id": "http://purl.org/dc/terms/license"}, "format": {"@id": "http://purl.org/dc/terms/format"}, "description": {"@id": "http://purl.org/dc/terms/description"}, "distribution": {"@id": "http://www.w3.org/ns/dcat#distribution"}, "@type": "@id"}, "keyword": {"@id": "http://www.w3.org/ns/dcat#keyword"}, "issued": {"@id": "http://purl.org/dc/terms/issued", "@type": "http://www.w3.org/2001/XMLSchema#dateTime"}, "publisher": {"@id": "http://purl.org/dc/terms/publisher", "@type": "@id"}, "modified": {"@id": "http://purl.org/dc/terms/modified", "@type": "http://www.w3.org/2001/XMLSchema#dateTime"}, "dct": "http://purl.org/dc/terms/", "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#", "xsd": "http://www.w3.org/2001/XMLSchema#", "rdfs": "http://www.w3.org/2000/01/rdf-schema#", "dcat": "http://www.w3.org/ns/dcat#", "foaf": "http://xmlns.com/foaf/0.1/", "dc": "http://purl.org/dc/elements/1.1/" } } </pre>	
Error response		
400	<p>Bad Request</p> <p>The method checks that the <i>id</i> parameter has been included, otherwise Bad Request code will be returned.</p> <p>In this case, the method will return a JSON response, with just one field <i>Error</i>, that will contain a description of the error.</p> <p>Example:</p> <pre> { "Error": "Parameter 'id' is required." } </pre>	
404	<p>Not found</p> <p>If the metadata with the <i>id</i> passed doesn't exist, this error will be returned with a JSON response with the error.</p> <p>Example:</p> <pre> { "Error": "Dataset '6bb9c361_177a635e86a234' not found." } </pre>	
Sample call		
<pre> curl -X GET "http://[server:port]/data/getDataset?id=6bb9c361_177a635e86a" -H "accept: application/json" </pre> <p style="text-align: right;">→ <i>id</i></p>		

8.3.4 getCatalogueDatasets (GET)

Table 12: API for the retrieval of dataset metadata

/getCatalogueDatasets	Gets the metadata of all the datasets stored in the database.
Method	
GET	
Input Params (* means mandatory)	
None	
Success response	
200	<p>Returns a JSON array with all the datasets stored.</p> <p>Example:</p> <pre>[{ "id": "6bb9c361_177a635e86a", "metadata": { "@graph": [{ "@id": "https://urbanite-project.eu/ontology/URBANITE_PROJECT", "@type": "foaf:Organization", "homepage": "https://urbanite-project.eu/", "name": "URBANITE" }, { "@id": "https://urbanite-project.eu/ontology/dataset/Bilbao_Calendar", "@type": "dcat:Dataset", "description": {"@language": "en", "@value": "Calendar data Bilbao"}, "issued": "2021-05-12T10:36:46", "modified": "2021-05-12T15:36:46", "publisher": "https://urbanite-project.eu/ontology/URBANITE_PROJECT", "title": {"@language": "en", "@value": "Calendar data Bilbao"}, "distribution": ["https://urbanite-project.eu/ontology/distribution/a732f6c6-fcd8-4962-8aa9-db7d913a20ae", "https://urbanite-project.eu/ontology/distribution/059fd3cc-92b3-4f3d-97de-d050ae022eb5"], "keyword": ["Calendar", "Bilbao"] }, { "@id": "https://urbanite-project.eu/ontology/distribution/059fd3cc-92b3-4f3d-97de-d050ae022eb5", "@type": "dcat:Distribution", "description": "Calendar data Bilbao year 2016 in NGSI-LD representation", "format": "http://publications.europa.eu/resource/authority/file-type/JSON_LD", "license": "http://publications.europa.eu/resource/authority/licence/CC_BY", "title": "Calendar data Bilbao 2016", "accessURL": "http://storageAPI-to-bedefined/2016" }] }, "@type": "dcat:Distribution", </pre>

	<pre> "description": "Calendar data Bilbao year 2015 in NGSI-LD representation", "format": "http://publications.europa.eu/resource/authority/file- type/JSON_LD", "license": "http://publications.europa.eu/resource/authority/licence/CC_BY", "title": "Calendar data Bilbao 2015", "accessURL": "http://storageAPI-to-be-defined/2015" }], "@context": { "name": {"@id": "http://xmlns.com/foaf/0.1/name"}, "homepage": {"@id": "http://xmlns.com/foaf/0.1/homepage"}, "accessURL": {"@id": "http://www.w3.org/ns/dcat#accessURL"}, "title": {"@id": "http://purl.org/dc/terms/title"}, "license": {"@id": "http://purl.org/dc/terms/license"}, "format": {"@id": "http://purl.org/dc/terms/format"}, "description": {"@id": "http://purl.org/dc/terms/description"}, "distribution": {"@id": "http://www.w3.org/ns/dcat#distribution", "@type": "@id"}, "keyword": {"@id": "http://www.w3.org/ns/dcat#keyword"}, "issued": {"@id": "http://purl.org/dc/terms/issued", "@type": "http://www.w3.org/2001/XMLSchema#dateTime"}, "publisher": {"@id": "http://purl.org/dc/terms/publisher", "@type": "@id"}, "modified": {"@id": "http://purl.org/dc/terms/modified", "@type": "http://www.w3.org/2001/XMLSchema#dateTime"}, "dct": "http://purl.org/dc/terms/", "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#", "xsd": "http://www.w3.org/2001/XMLSchema#", "rdfs": "http://www.w3.org/2000/01/rdf-schema#", "dcat": "http://www.w3.org/ns/dcat#", "foaf": "http://xmlns.com/foaf/0.1/", "dc": "http://purl.org/dc/elements/1.1/" } }, { "id": "6bb9c361_177a635sfd124a", "metadata": { "id": "6bb9c361_177a635sfd124a", "metadata": { "@graph": [{ "@id": "https://urbanite- project.eu/ontology/URBANITE_PROJECT", "@type": "rdf:Organization", "homepage": "https://urbanite-project.eu/", "name": "URBANITE" }, { "@id": "https://urbanite- project.eu/ontology/dataset/Bilbao_Calendar", "@type": "dcat:Dataset", "description": {"@language": "en", "@value": "Calendar data Messina"}, "issued": "2021-05-12T10:36:46", "modified": "2021-05-12T15:36:46", "publisher": "https://urbanite- project.eu/ontology/URBANITE_PROJECT", "title": {"@language": "en", "@value": "Calendar data Messina"}, "distribution": ["https://urbanite- project.eu/ontology/distribution/059fd3cc-63ws-4f3d-97de-fdgs4fdh2eg", "https://urbanite- project.eu/ontology/distribution/059fd3cc-43s3-4gds-w436-fhd45sdaf32"], "keyword": ["Calendar", "Messina"] }, { "@id": "https://urbanite- project.eu/ontology/distribution/059fd3cc-63ws-4f3d-97de-fdgs4fdh2eg", "@type": "dcat:Distribution", "description": "Calendar data Messina year 2016 in NGSI-LD </pre>
--	--

	<pre> representation", "format": "http://publications.europa.eu/resource/authority/file- type/JSON_LD", "license": "http://publications.europa.eu/resource/authority/licence/CC_BY", "title": "Calendar data Messina 2016", "accessURL": "http://storageAPI-to-bedefined/2016" }, { "@id": "https://urbanite- project.eu/ontology/distribution/059fd3cc-43s3-4gds-w436-fhd45sdaf32", "@type": "dcat:Distribution", "description": "Calendar data Messina year 2016 in NGSI-LD representation", "format": "http://publications.europa.eu/resource/authority/file- type/JSON_LD", "license": "http://publications.europa.eu/resource/authority/licence/CC_BY", "title": "Calendar data Messina 2016", "accessURL": "http://storageAPI-to-bedefined/2016" }], "@context": { "name": {"@id": "http://xmlns.com/foaf/0.1/name"}, "homepage": {"@id": "http://xmlns.com/foaf/0.1/homepage"}, "accessURL": {"@id": "http://www.w3.org/ns/dcat#accessURL"}, "title": {"@id": "http://purl.org/dc/terms/title"}, "license": {"@id": "http://purl.org/dc/terms/license"}, "format": {"@id": "http://purl.org/dc/terms/format"}, "description": {"@id": "http://purl.org/dc/terms/description"}, "distribution": {"@id": "http://www.w3.org/ns/dcat#distribution", "@type": "@id"}, "keyword": {"@id": "http://www.w3.org/ns/dcat#keyword"}, "issued": {"@id": "http://purl.org/dc/terms/issued", "@type": "http://www.w3.org/2001/XMLSchema#dateTime"}, "publisher": {"@id": "http://purl.org/dc/terms/publisher", "@type": "@id"}, "modified": {"@id": "http://purl.org/dc/terms/modified", "@type": "http://www.w3.org/2001/XMLSchema#dateTime"}, "dct": "http://purl.org/dc/terms/", "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#", "xsd": "http://www.w3.org/2001/XMLSchema#", "rdfs": "http://www.w3.org/2000/01/rdf-schema#", "dcat": "http://www.w3.org/ns/dcat#", "foaf": "http://xmlns.com/foaf/0.1/", "dc": "http://purl.org/dc/elements/1.1/" } } </pre>	
Sample call		
<pre> curl -X GET "http://[server:port]/data/getCatalogueDatasets" -H "accept: application/json" </pre>		

8.3.5 searchDatasets (GET)

Table 13: API for the search and retrieval of dataset metadata

/searchDatasets	<p>Searches among the metadata of the existing dataset.</p> <p>It makes a search in typical metadata fields: <i>title</i>, <i>description</i> and <i>keyword</i>.</p> <p>All the <i>tags</i> search must be present in at least one of these fields.</p>
Method	

GET	
Input Params (* means mandatory)	
search	Tags to search for, separated by a space.
(query param)	Optional. If not set, all the metadata will be returned. Example: <i>Calendar Messina</i>
Success response	
200	<p>The method will return this code, and the JSON response will contain a JSON array with that metadata stored in the database that contain all the tags passed in the parameter <i>search</i>.</p> <p>Example:</p> <pre>[{ "id": "6bb9c361_177a635e86a", "metadata": { "@graph": [{ "@id": "https://urbanite-project.eu/ontology/URBANITE_PROJECT", "@type": "foaf:Organization", "homepage": "https://urbanite-project.eu/", "name": "URBANITE" }, { "@id": "https://urbanite-project.eu/ontology/dataset/Bilbao_Calendar", "@type": "dcat:Dataset", "description": {"@language": "en", "@value": "Calendar data Bilbao"}, "issued": "2021-05-12T10:36:46", "modified": "2021-05-12T15:36:46", "published": "https://urbanite-project.eu/ontology/URBANITE_PROJECT", "title": {"@language": "en", "@value": "Calendar data Bilbao"}, "distribution": ["https://urbanite-project.eu/ontology/distribution/a732f6c6-fcd8-4962-8aa9-db7d913a20ae", "https://urbanite-project.eu/ontology/distribution/059fd3cc-92b3-4f3d-97de-d050ae022eb5"], "keyword": ["Calendar", "Bilbao"] }, { "@id": "https://urbanite-project.eu/ontology/distribution/059fd3cc-92b3-4f3d-97de-d050ae022eb5", "@type": "dcat:Distribution", "description": "Calendar data Bilbao year 2016 in NGSI-LD representation", "format": "http://publications.europa.eu/resource/authority/file-type/JSON_LD", "license": "http://publications.europa.eu/resource/authority/licence/CC_BY", "title": "Calendar data Bilbao 2016", "accessURL": "http://storageAPI-to-be-defined/2016" }, { "@id": "https://urbanite-project.eu/ontology/distribution/a732f6c6-fcd8-4962-8aa9-db7d913a20ae", "@type": "dcat:Distribution", "description": "Calendar data Bilbao year 2015 in NGSI-LD representation", "format": "http://publications.europa.eu/resource/authority/file-type/JSON_LD", </pre>

	<pre> "license": "http://publications.europa.eu/resource/authority/licence/CC_BY", "title": "Calendar data Bilbao 2015", "accessURL": "http://storageAPI-to-beDEFINED/2015" }], "@context": { "name": {"@id": "http://xmlns.com/foaf/0.1/name"}, "homepage": {"@id": "http://xmlns.com/foaf/0.1/homepage"}, "accessURL": {"@id": "http://www.w3.org/ns/dcat#accessURL"}, "title": {"@id": "http://purl.org/dc/terms/title"}, "license": {"@id": "http://purl.org/dc/terms/license"}, "format": {"@id": "http://purl.org/dc/terms/format"}, "description": {"@id": "http://purl.org/dc/terms/description"}, "distribution": {"@id": "http://www.w3.org/ns/dcat#distribution", "@type": "@id"}, "keyword": {"@id": "http://www.w3.org/ns/dcat#keyword"}, "issued": {"@id": "http://purl.org/dc/terms/issued", "@type": "http://www.w3.org/2001/XMLSchema#dateTime"}, "publisher": {"@id": "http://purl.org/dc/terms/publisher", "@type": "@id"}, "modified": {"@id": "http://purl.org/dc/terms/modified", "@type": "http://www.w3.org/2001/XMLSchema#dateTime"}, "dct": "http://purl.org/dc/terms/", "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#", "xsd": "http://www.w3.org/2001/XMLSchema#", "rdfs": "http://www.w3.org/2000/01/rdf-schema#", "dcat": "http://www.w3.org/ns/dcat#", "foaf": "http://xmlns.com/foaf/0.1/", "dc": "http://purl.org/dc/elements/1.1/" } }] </pre>	
Sample call		
<pre> curl -X GET "http://[server:port]/data/search-sets?search=Bilbao%20Calendar" -H "accept: application/json" </pre> <p style="text-align: right;">→ search</p>		