

OpenDataHub Docs Documentation

Release 1.0

The Open Data Hub Team

May 20, 2019

Table of Contents

1	Introduction	1
1.1	Project Overview	1
2	How To Contribute	7
2.1	As a user I can.	7
2.2	As an App Developer I can.	7
2.3	As a Open Data Hub Core Hacker I can.	8
2.4	Bug reporting and feature requests	8
3	List of HOWTOs	11
3.1	HOWTO access e-Charging Stations Data	11
3.2	HOWTO access Tourism Data	13
3.3	HOWTO use the Open Data Hub's Tourism Data Browser	17
3.4	Quick and (not-so) Dirty Tips for Tourism	18
3.5	How to use authentication	22
3.6	How to setup your local Development Environment	25
3.7	How to Sett Up Postman (API Development Environment)	27
3.8	How to insert and modify NOI Events	32
4	Apps built from Open Data Hub datasets	37
4.1	Alpha Stage Apps	37
4.2	Beta Stage Apps	38
4.3	Production Stage	38
5	Frequently Asked Questions	39
6	Appendices	41
6.1	Datasets	41
6.2	Resources for Developers	51
6.3	Glossary	71
6.4	Licenses and TOS for the Open Data Hub material	72

CHAPTER 1

Introduction

This is the website of the Open Data Hub documentation, a collection of technical resources about the Open Data Hub project. The website serves as the main resource portal for everyone interested in accessing the data or deploying apps based on *datasets* & *APIs* provided by the Open Data Hub team.

The technical stuff is composed of:

- Catalogue of available datasets.
- How-tos, FAQs, and various tips and tricks for users.
- Links to the full API documentation.
- Resources for developers.

For non-technical information about the Open Data Hub project, please point your browser to <https://opendatahub.bz.it/>.

1.1 Project Overview

The Open Data Hub project envisions the development and set up of a portal whose primary purpose is to offer a single access point to all (Open) Data from the region of South Tyrol, Italy, that are relevant for the economy sector and its actors. This will also allow everybody to utilise these data in all digital communication channels and build application on top of the data offered, be them either a PoC (Proof of Concept) to explore new means or new field in which to use Open Data Hub data, or novel and innovative services or software products built on top of the data.

All the data within the Open Data Hub will be easily accessible, preferring open interfaces and APIs which are built on existing standards like [The Open Travel Alliance \(OTA\)](#), [The General Transit Feed Specification \(GTFS\)](#), [Alpinebits](#).

The Open Data Hub team also strives to keep all data regularly updated, and use standard exchange formats for them like [Json](#) and the [Data Catalog Vocabulary \(DCAT\)](#).

Depending on the development of the project and the interest of users, more standards and data formats might be supported in the future.

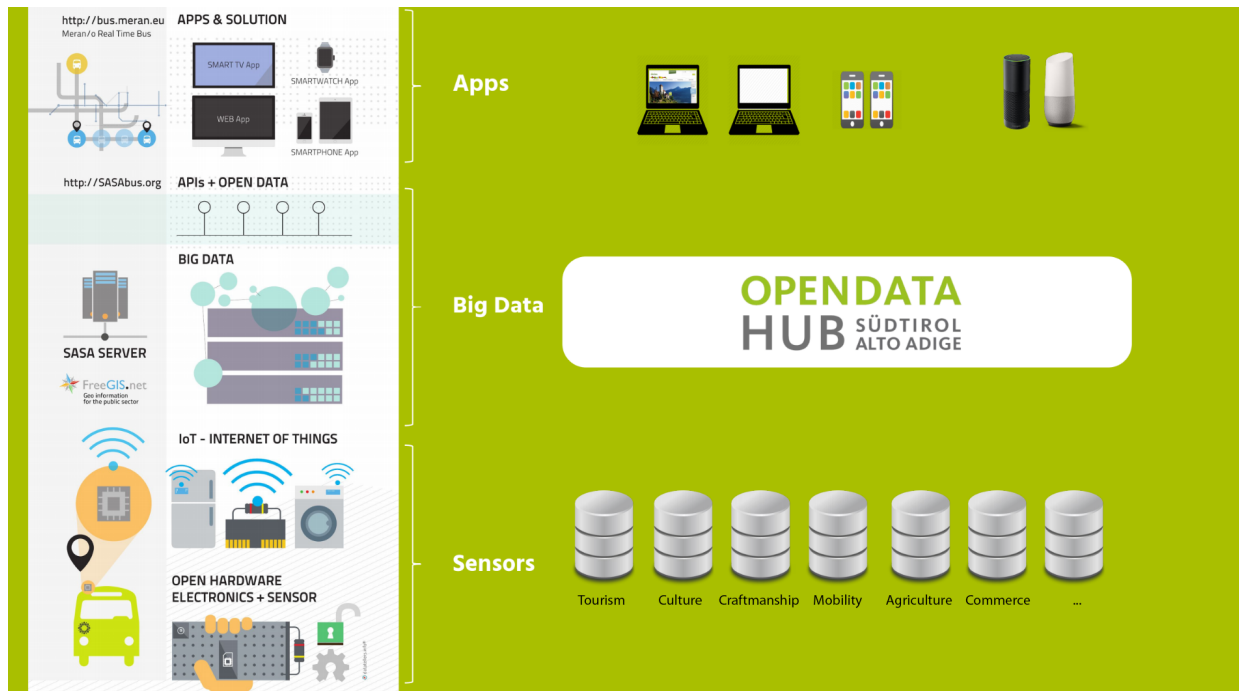


Figure 1.1: An overview of the Open Data Hub Project.

1.1.1 Open Data Hub Architecture

The architecture of the Open Data Hub is depicted in [Figure 1.2](#), which shows its composing elements together with its main goal: To gather data from **Data Sources** and make them available to **Data Consumers**, which are usually third-party applications that use those data in any way that they deem useful, including (but not limited to) study the evolution of historical data, or carry out data analysis to produce *statistical graphics*.

At the core of the Open Data Hub lays the Big Data Platform, a java application which contains all the business logic and handles all the connections with the underlying database using the DAL (Data Access Layer). The Big Data Platform is composed by different modules: A **Writer**, that receives data from the Data Sources and stores them in the Database using the DAL.

Communication with the Data Sources is guaranteed by the **Data Collectors**, which are Java applications built on top of the **dc-interface** that use a DTO (Data Transfer Object) for each different source to correctly import the data. Dual to the **dc-interface**, the **ws-interface** allows the export of DTOs to web services, that expose them to **Data Consumers**.

The bottom part of [Figure 1.2](#) shows the *data format* used in the various steps of the data flow.

Records in the Data Sources can be stored in any format and are converted into JSON as DTOs. They are then transmitted to the Writer, who converts them and stores them in the Database using SQL. To expose data, the Reader queries the DB using SQL, transforms them in JSON's DTOs to the Web Services who serve the JSON to the Data Consumers.

1.1.2 The Elements of the Big Data Platform in Details

As [Figure 1.2](#) shows, the Big Data Platform is composed by a number of elements, described in the remainder of this section in the same order as they appear in the picture.

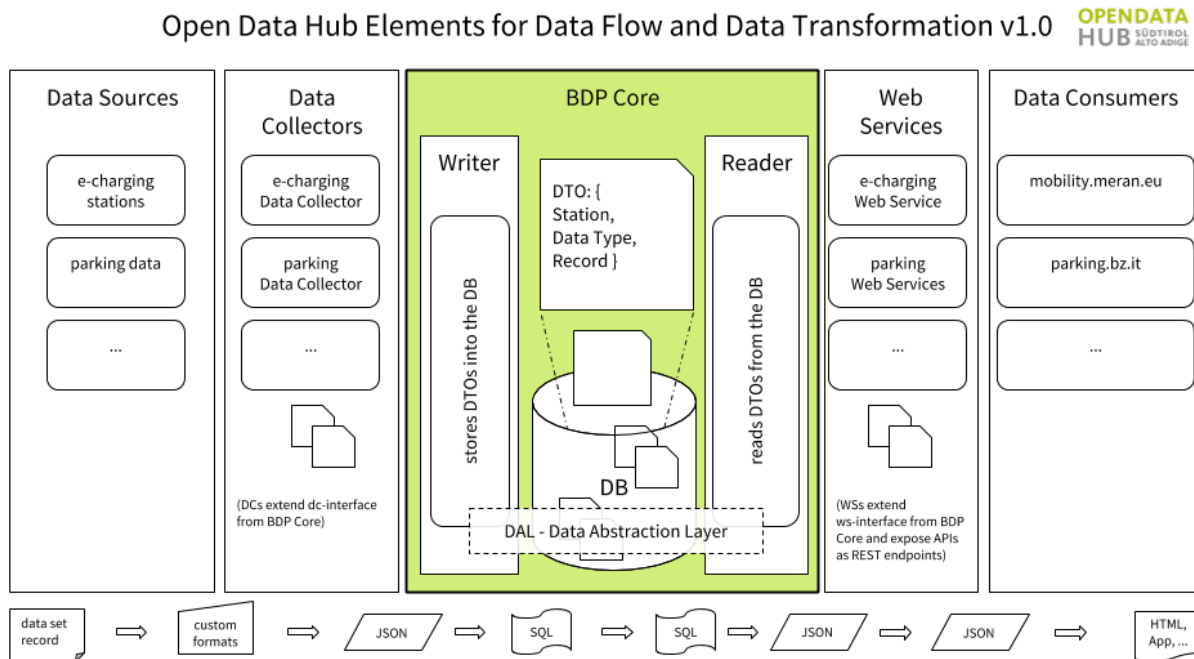


Figure 1.2: The Open Data Hub architecture with the components (top) and the data format used (bottom) during the data transformation.

Data Source A Data Source is the origin of one or more datasets, which usually belongs to a single domain. Data are usually automatically picked up by sensors and stored in some format, like for example CSV.

Dataset A dataset is a collection of records that originate from the same Data Source. Within the Open Data Hub, a same Data Source may provide more datasets, that include slight different data, but there is at least one dataset per domain. The underlying data format of a dataset **never** changes.

Data Collectors Data collectors are a library of Java classes used to transform data coming from Data Sources into a format that can be understood, used, and stored by the Big Data Platform. As a rule of thumb, each Data Collector is used for one Data Source or dataset and use DTOs to transfer them to the Big Data Platform. They are usually created by extending the **dc-interface** in the bdp-core repository.

DTO The Data Transfer Object are used to translate the data format from the various formats used by the Data Sources, to be read from the writer and to be exposed by the reader (see below). DTOs are written in **JSON**. and are composed of three *Entities*: Station, Data Type, and Record.

Writer With the Writer, we enter in the core of the Big Data Platform. Its purpose is to receive DTOs from the Data Collectors and store them into the DB and therefore implements all methods to read the DTO's **JSON** format and to write to the database using SQL.

DAL The Data Abstraction Layer is used by both the Writer and the Reader to access the Database and exchange DTOs and relies on Java Hibernate. It contains classes that map the content of a DTO to corresponding database tables.

Database (DB) The database represents the persistence layer and contains all the data sent by the Writer. Its configuration requires that two users be defined, one with full permissions granted -used by the writer, and one with read-only permissions, used by the Reader.

Reader The reader is the last component of the Core. It uses the DAL to retrieve DTOs from the DB and to transmit them to the web services.

Web Services The Web Services, which extend the **ws-interface** in the bdp-core repository, receive data from

the Reader and make them available to Data Consumers by exposing APIs and REST endpoints. They transform the DTO they get into JSON.

Data Consumers Data consumers are (web-)applications that use the JSON produced by web services and manipulates them to produce a useful output for the final user.

Also part of the architecture, but not pictured in the diagram, is the `persistence.xml` file, which contains the credentials and postgres configuration used by both the Reader and Writer.

1.1.3 Available Domains and APIs

The domains intended as sources for data served by the Open Data Hub are depicted in [Figure 1.1](#).

The **API** of a software contains the definition of methods and of their signatures, that can be invoked to retrieve data from the web services provided by the software itself. The signature of each method defines how to invoke the method (i.e., the name of the method), which parameters should be supplied (i.e., their names and types, if they are mandatory or not, and what the method returns (i.e., the type and format of the output produced by the method. By using an API, it is possible to receive data from the web service and process them.

Currently, the following *APIs* are available from the Open Data Hub:

1. **Mobility APIs**
2. **SASAbus APIs**
3. **Tourism APIs.**

The first and second APIs provide datasets that belong to the *Mobility Domain*, while the third one to datasets in the *Tourism Domain*.

The Mobility APIs allow to access real-time data of the datasets concerning the e-mobility, including data about e-charging stations, availability of plugs to recharge e-cars, and so on.

The SASAbus APIs are part of the Mobility domain and allow to access various type of data about buses and station.

The Tourism API allows to access locations (of hotels, museums, events, and so on), points of interests, and a number of other information about the tourism in South Tyrol.

Authentication

Note: The authentication layer is currently intended for internal use only.

Authentication in Open Data Hub is mainly used in the part of the Big Data Platform which exposes data to the consumer, which means by the Reader and in every single webservice accessing the Reader, to allow the access to closed data in each dataset only to those who are allowed to.

There are currently two different authentication methods available:

- The **Token-based Authentication**, defined in [RFC 6750](#), requires that anyone who wants to access resources supply a valid username and password and becomes a Bearer Token that must be used to access the data. After the token expires, a new one must be obtained. This type of authentication is used for the datasets in the tourism domain.
- The **OAuth2 Authentication** follows the [RFC 6749](#) and is used for all the datasets in the mobility domain.

The OAuth2 authentication mechanism Authentication tokens are based on *JSON Web Token (JWT)* as defined in [RFC 7519#section-3](#), to send *claims*.

For those not familiar with the OAuth2 mechanism, here is a quick description of the client-server interaction:

1. The client requests the permission to access restricted resources to the *authorisation server*.
2. The authorisation server replies with a **refresh token** and an **access token**. The access token contains an expire date.
3. The access token can now be used to access protected resources on the *resource server*. To be able to use the access token, add it as a Bearer token in the Authorization header of the HTTP call. **Bearer** is a means to use tokens in HTTP transactions. The complete specification can be found in **RFC 6750**.
4. If the access token has expired, you'll get a HTTP 401 `Unauthorized` response. In this case you need to request a new access-token, passing your refresh token in the *Authorization* header as Bearer token. As an example, in Open Data Hub datasets Bearer tokens can be inserted in a **curl** call like follows:

```
curl -X GET "$HTTP_URL_WITH_GET_PARAMETERS" -H "accept: */*" -H "Authorization:↵
↵Bearer $TOKEN"
```

Here, `$HTTP_URL_WITH_GET_PARAMETERS` is the URL containing the API call and “`$TOKEN`” is the string of the token.

1.1.4 Available Datasets

The list of available datasets has been moved *to a dedicated page*.

How To Contribute

There are different possibilities to participate in the Open Data Hub Project, including -but not limited to- to report bugs in the API or errors in the API output, to ask for more datasets to be added to our repository, to make feature requests or suggestions for improvement.

Depending on your interest on the Open Data Hub Project, we welcome your participation to the project in one of the roles that we have envisioned: *User*, *App developer*, *Core Hacker*.

You can also help the Open Data Hub project grow and improve by *reporting bugs or asking new features*.

2.1 As a user I can...

...install and use an app built on top of the API. Browse the list of available applications developed by third-parties, choose one that you are interested in, install it and try it out, then send feedback to their developers if you feel something is wrong or missing.

While we plan to keep an up-to-date list of those applications, as of May 20, 2019, no such application that we are aware of has been released.

...explore the data in the datasets. Choose a dataset from the list of *Datasets* and start gathering data from it, by using the documentation provided in this site. You can then provide any kind of feedback on the dataset: reports about any malfunctions, suggestions for improvements or new features, and so on.

Moreover, if you are interested in datasets that are not yet in our collection, get in touch with the Open Data Hub team to discuss your request.

No software installation is needed: Go to the list of apps (not yet available) or the list of datasets and start from there.

2.2 As an App Developer I can...

...harvest data exposed by the dataset. Browse the list of *Datasets* to see what types of data are contained in the datasets, and think how they can be used.

For this purpose, we maintain an updated list of the *available datasets* with links to the API to access them.

...build an application with the data. Write code for an app that combines the data you can harvest from the available datasets in various, novel way.

To reach this goal, you need to access the APIs, their documentation, and the datasets. It is then your task to discover how you can reuse the data in your code.

...publish my app in Open Data Hub. As soon as you have developed a stable version of your app, get in touch with us: We plan to maintain an updated list of apps based on our dataset included with this documentation.

No software installation is needed: Go to the list of apps (not yet available, be the first!) or API documentation/datasets and start from there, and develop in a language of your choice an application that uses our data.

2.3 As a Open Data Hub Core Hacker I can...

...help shape the future of Open Data Hub. Participate in the development of Open Data Hub: Build new data collectors, extend the functionality of the broker, integrate new datasets on the existing infrastructure, develop new stable API versions.

To be able to become a core hacker, however, requires a few additional tasks to be carried out:

1. Learn how to successfully integrate your code with the existing code-base and how to interact with the Open Data Hub team.

In other words, you need to read and accept the *Guidelines for Developers*, summarised there and available in two separate parts: *Platform Guidelines - Full Version* and *Database Guidelines - Full Version*.

2. Understand the *architecture* of both the Open Data Hub and Big Data Platform.
3. Learn about the *Development, Testing, and Production Environments*.
4. Install the necessary software on your local workstation (be it a physical workstation, a virtual machine, or a Docker instance), including PostgreSQL with postgis extension, JDK, git.
5. Set up all the services needed (database, application server, and so on).
6. Clone our git repositories.

To successfully complete these tasks, please read the *How to setup your local Development Environment* tutorial, which guides you stepwise through all the required set up and configuration, along with some troubleshooting advice.

7. Coding.

That's the funniest part, enjoy!

To support the installation tasks and ease the set up of your workstation, we are developing a script the you will do the job for you. Stay tuned for updates.

2.4 Bug reporting and feature requests

This section explains what to do in case you:

1. have found an error or a bug in the APIs;
2. like to suggest or require some enhancement for the APIs;
3. have some requests about the datasets
4. find typos or any error in this documentation repository;

5. have an idea for some specific tutorial.

If your feedback is related to the Open Data Hub Core, including technical bugs or suggestions as well as requests about datasets (i.e. points 1. to 3. above), please insert your issues on the following website:

<https://github.com/idm-suedtirol/bdp-core/issues>

If your feedback is related to the Open Data Hub Documentation, please insert your issue on the following website, using the template that suits your needs:

4. https://github.com/idm-suedtirol/odh-docs/issues/new?template=bug_report.md
5. <https://github.com/idm-suedtirol/odh-docs/issues/new?template=feedback.md>

Note: You need to have a valid github account to report issues and interact with the Open Data Hub team.

We keep track of your reports in our bug trackers, where you can also follow progress and comments of the Open Data Hub team members.

List of HOWTOs



This page contains the list of available howtos, divided into areas. The following howtos are available; while further below the list, some information about each of them is provided.

3.1 HOWTO access e-Charging Stations Data

This howto uses the *E-charging stations dataset* to showcase a few basic API calls, whose output will be needed in most complex calls.

3.1.1 Dataset Information

This datasets exposes data about the existing e-charging stations in South Tyrol and their status, including historical data and usage.

License	 
Output	JSON, mime-type application/json
E-mail contact	info@opendatahub.bz.it
API version	v1
Swagger URL	http://ipchannels.integreen-life.bz.it/EchargingFrontEnd

3.1.2 Invoking the API

The available methods in this API are very generic, so some post-processing of the JSON that you receive as output will probably be necessary.

The API calls shown here can be used with other datasets of the *mobility domain*.

You can find all the API's defined methods and documentation at the URL <http://ipchannels.integreen-life.bz.it/EchargingFrontEnd>.

The two most basic REST calls are carried out by the methods **get-stations** and **get-station-details**.

get-stations

The **get-stations** method requires no parameters and retrieves all the IDs of the charging stations that are part of this dataset.

There are two possibilities to retrieve the data with the API call:

1. By HTTP request:

```
http://ipchannels.integreen-life.bz.it/emobility/rest/get-stations
```

2. Using a command line with a tool like **curl** or **wget**:

```
curl -X GET --header 'Accept: application/json' 'http://ipchannels.integreen-life.bz.it/emobility/rest/get-stations'
```

The result structure is a json list of strings, and an actual outcome is (shortened for the sake of clarity):

```
[
  "IT*220*EBZ000034",
  "82",
  "DW_000006",
  "DW_000009",
  "IT*220*ETN020016",
  "83",
  "84",
  "DW_000013",
  "DW_000019",
  "85",
]
```

Each of the IDs can then be used in other methods to obtain more detailed information about the station.

get-station-details

The **get-station-details** method requires no parameters and retrieves all the known information for each charging station in the dataset. Like the previous method, two method can be used for the call:

1. By HTTP request:

```
http://ipchannels.integreen-life.bz.it/emobility/rest/get-station-details
```

2. Using a command line with a tool like **curl** or **wget**:

```
curl -X GET --header 'Accept: application/json' 'http://ipchannels.integreen-life.bz.it/emobility/rest/get-station-details'
```

The result structure is a json list of strings, and an actual outcome is (shortened for the sake of clarity):

```
{
  "_t": "it.bz.idm.bdp.dto.emobility.EchargingStationDto",
  "id": "ASM_00000103",
  "name": "BRIXEN_02",
  "latitude": 46.706333,
  "longitude": 11.651225,
  "municipality": "Brixen - Bressanone",
  "capacity": 2,
```

(continues on next page)

(continued from previous page)

```

"provider": "Alperia Smart Mobility",
"city": "BRESCANONE - BRIKEN",
"state": "ACTIVE",
"paymentInfo": "https://www.alperiaenergy.eu/smart-mobility/punti-di-ricarica.html",
"accessType": "PUBLIC",
"address": "CLUB MAX - Fischzuchtweg - Via del Laghetto"
},
{
  "_t": "it.bz.idm.bdp.dto.emobility.EchargingStationDto",
  "id": "DW-000027",
  "name": "San Vigilio Hotel Sport",
  "latitude": 46.698061,
  "longitude": 11.934766,
  "municipality": "Mar  - Enneberg - Marebbe",
  "capacity": 1,
  "provider": "DriWe",
  "city": "San Vigilio (Marebbe)",
  "state": "ACTIVE",
  "paymentInfo": "http://www.driwe.eu",
  "accessInfo": "24h",
  "accessType": "PRIVATE_WITHPUBLICACCESS",
  "categories": [
    "EAT&CHARGE",
    "SLEEP&CHARGE"
  ],
  "address": "Strada al Plan Dessora",
  "reservable": true
},

```

As you see from the example, many of the e-charging station's metadata is shared by all of them including the (unique) ID, name, location (town or city, address, geographic coordinates), access to it. There are however additional metadata that are optional (like the station's category and if it is reservable).

3.1.3 Troubleshooting

If the API call fails, one of the following response code is returned - they correspond to HTTP status codes :

401 Unauthorised The request is valid, but authentication is required and you provided none. This should never happen, you should receive an empty output set if you require data that are not publicly available.

403 Forbidden The request is valid but could not be completed on the server side.

404 Not found There is an syntax error in the call you made or the page is not available at this moment.

500 Internal Server Error Oh, well. Apparently we have a problem now...

3.2 HOWTO access Tourism Data

Note: Information in this page might change in the next future.

The purpose of this howto is to quickly introduce the structure of the API calls, the available filters for the datasets in the Tourism domain, and give some general and useful information about the Tourism API.

3.2.1 Structure of the API calls

In the Tourism domain, there are a few API calls that allow to extract the same type of data from the various datasets. Each of these calls can prove useful in different scenarios, depending on the data returned and is described in this section, in which the following conventions are used:

- `{Name}` is the (case sensitive!) name of the dataset you are currently working with, like for example `Accommodation`.
- `{Id}` is the unique identifier of an array within the dataset, i.e., an item of the dataset. It is usually the first key of the resulting JSON output of a query.

The calls defined for every datasets are:

- `/api/{Name}` Return the whole dataset.
- `/api/{Name}/{Id}` Return only item with given `{Id}`.
- `/api/{Name}Localized` Return the whole dataset in only the given language (which is a mandatory part of the query).
- `/api/{Name}Localized/{Id}` Return only item with given `Id` and in given language.
- `/api/{Name}Reduced` Return only the list of `Ids` and respective name of the items in the dataset. It is useful to create lists of items or just to have an overview of the dataset's items.
- `/api/{Name}Changed` Return all items that have changed since date `YYYY-MM-DD`
- `/api/{Name}Types` Returns all types of data present in the dataset, that can be later used to ask more precise queries to the dataset.

3.2.2 Filters common to all datasets

Filters are used within a dataset and their primary purpose is to limit the result set according to specific parameters. They might not be available in every API call. Information about default values can be found for each datasets in the [swagger interface](#) of the API. Some examples of their use can be found in section [Quick and \(not-so\) Dirty Tips for Tourism](#).

Note: This section is **Work in Progress** and might be expanded.

- **Seed** is used to set pagination. See tip [TT3](#).
- **Locfilter** is a composed parameters that uniquely identifies a location within South Tyrol. See example [EX2](#) for a detailed example.
- **Latitude** and **Longitude** are used to identify the (absolute) positioning of a location, point of interest, event, or any other type of object. They must be entered in decimal form
- **Radius** it is the distance in meter from a geographical point. It can be used together with latitude and longitude to broaden the search for an object. The results are automatically *geosorted*, that is, they are listed from the nearest to the most far away from the selected point. The distance is calculated as the crow flies.
- **IdFilter** allows to extract from the dataset only the items with the given IDs, separated with a `,`.
- **Active** and **OdhActive**. Filters with the same name, with one prefixed by **Odh** refer to the same parameter. The difference is however important: **Active** indicates that the item is present in the original dataset provided, while **OdhActive** shows that the item has been verified by the Open Data Hub team and is present in the Open Data Hub. See discussion in tip [TT2](#).

- **ODHTag** allows to filter a result set according to tag defined by the Open Data Hub team. These tags are mostly related with places to see, activities that can be carried out in winter or summer, food and beverage, cultural events and so on

3.2.3 Filters specific of a datasets

Note: This section will be available soon.

3.2.4 Types of input data

Note: This section is **Work in Progress** and will be expanded with additional types of input data.

Since calls in the tourism domain are quite generic and revolve around a few common calls (see section *Structure of the API calls*), we showed a couple of filters that can be used to reduce the result set and make the query more precise. Depending on the type of filter, a different type of data must be entered to have a successful result, otherwise the filter will not match. In this section we show the most common types of data that should be provided, besides the common strings, dates, and integers.

Bitmask value A Bitmasks value is a kind of shorthand that can be entered in a filter to obtain results for different types of that filter's accepted values. Each of the accepted values has a code that is a power of two (1, 2, 4, 8, and so on), hence each sum of different codes produces a unique number. The advantage is that, instead of entering multiple strings that should be matched, you simply need to enter a number as a filter, that is the sum of the values' corresponding codes. See *Example 3*.

Lists A list is an (unordered) sequence of items. The available values are usually listed on the right-hand side of the filter, along with the separator, which is a **comma** (,). In a few cases, in which more lists are accepted as filter.

Compound values Compound values refer to those values that need a prefix before the type of value. See for example *Example 2* for a deeper explanation and *Example 1* for a sample query that fails because a wrong compound value was supplied.

Language The descriptions of items in the dataset appear in three languages: Italian, German, and English. To retrieve values only in one language, enter **it**, **de**, or **en**, respectively.

3.2.5 Data Access and Manipulation

All the APIs available for the tourism domain can be accessed from the same URL through their swagger interface: <http://tourism.opendatahub.bz.it/swagger>

For most of the Tourism domain datasets you need credentials to access the data. Go to the abovementioned URL and click on *Expand Operations* on the far right-hand side of **Login (Get Bearer Token)**. In the panel that opens (see *Figure 3.1*), fill in the username and password with your credentials.

Note: The **Bearer Token Login** method replaces the previous one, **LoginAPI**, which is not available anymore.

Click on Try it out to generate a new access token that will be needed for any further request (see *Figure 3.2*).

After you have clicked on the button, the panel will expand and present some more data, the most important are the **Curl** and **Response Body** sections. In the first one, you can see the **POST** call sent from the API in curl format: you can use its content to write scripts that fetch data and automatise data fetching.

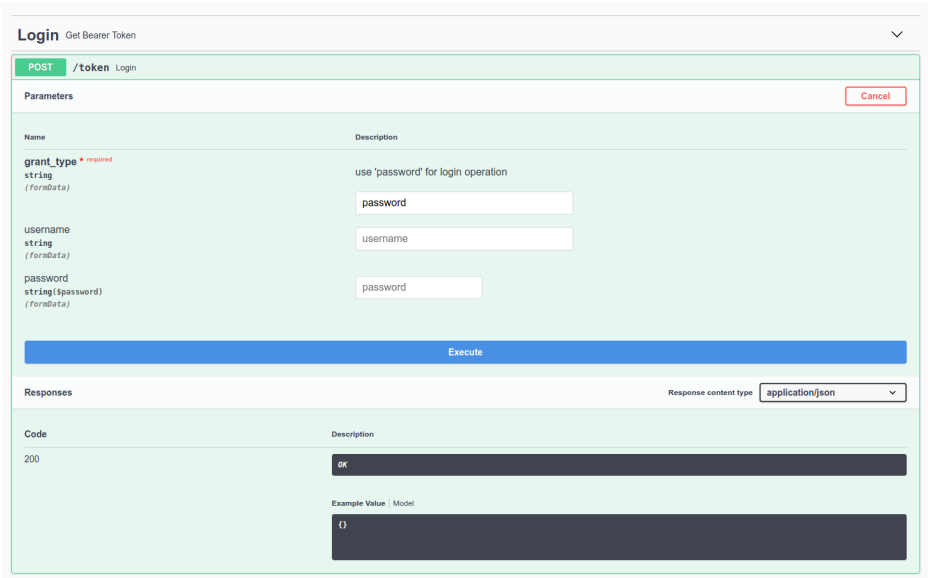


Figure 3.1: Login mask of the Tourism API.

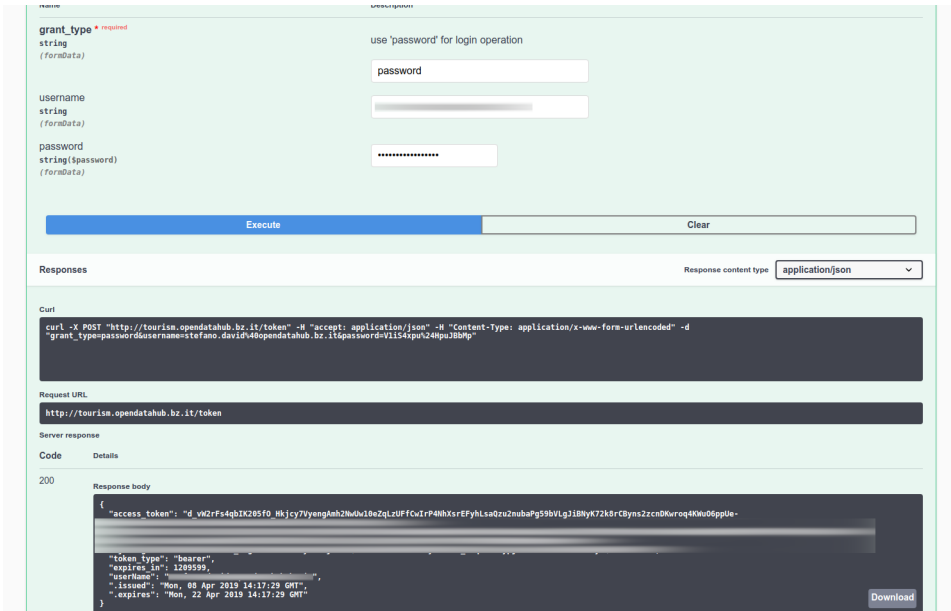


Figure 3.2: Token assigned to the user.

The **Response Body** section contains the answer to the call and is a JSON-formatted string that contains a few data, the most important of which are:

- The **access_token**, needed for any access to the data
- The **expires_in**, the validity in seconds of the access token before its expiration.

To avoid writing every time the token in the API methods, copy and paste **only** the token without quotes from the response body into the textfield on the right-hand side textfield on top of the page (see [Figure 3.3](#)), then click on Explore to store the token and cache it for all the next queries.



Figure 3.3: Caching the received token into the tourism' swagger interface.

Using Command Line Tools

If you plan to access the API methods with command line tools like **curl** or **wget**, or only from scripts, you need to add an authentication header to each call. For example, using curl:

```
curl -X GET --header 'Accept: application/json' --header \
'Authorization: Bearer vLwemAqrLKVKXsvgvEQgtkeanbMq7Xcs' \
'http://tourism.opendatahub.bz.it/api/Gastronomy'
```

Note: The string of the token is shortened for the sake of clarity.

It is important to mention that the authorisation header requires the following syntax: **Authorization: Bearer**, followed by the whole *string* of the token.

Once you have retrieved the data, which come in JSON format, you can process and manipulate them with a tool like **jq**.

See also:

More detailed documentation of the exposed API methods can be found on <http://tourism.opendatahub.bz.it/Help>.

3.3 HOWTO use the Open Data Hub's Tourism Data Browser

This how to explains the necessary steps to access and retrieve data from the Open Data Hub's tourism domain.

3.3.1 Data Browsing and Exploring

In order to access the data in the tourism domain, launch a browser and point it to <http://tourism.opendatahub.bz.it/>. On the right-hand side of the page, you can see a box that shows the permissions to access data that you have as a (non-logged in) user, while the remainder of the page provides an overview of the various component of the project and its architecture.

If you try to access the **ODH Data** item in the top menu, you will see that it is empty. In order to access data, you need to click on the **Login** button on the top right corner of the page.

Write the username (email address) and password that was provided to you and click on the Log in. You will be redirected to the home page as a logged in user and from here, you will see the box with the permissions you have to access the various types of data.

ODH Data external Data Sources Register Log In

OPENDATA HUB SÜDTIROL ALTO ADIGE

TECHPARK SÜDTIROL / ALTO ADIGE

IDM Open Data Hub Tourism Data

- Architecture: Asp.Net MVC
- Database: PostgreSQL
- Javascript Framework: AngularJS
- Api: Asp.Net Web Api JSON
- Authorization: Bearer Token
- Api Documentation: Swagger

API Documentation Swagger

User: Role: **User**

Common	Read / View / Create / Modify / Delete	×	×	×	×	×
Activities	Read / View / Create / Modify / Delete	×	×	×	×	×
Pois	Read / View / Create / Modify / Delete	×	×	×	×	×
ODHPois	Read / View / Create / Modify / Delete	×	×	×	×	×
Articles	Read / View / Create / Modify / Delete	×	×	×	×	×
Gastronomy	Read / View / Create / Modify / Delete	×	×	×	×	×
Accommodation	Read / View / Create / Modify / Delete	×	×	×	×	×
Package	Read / View / Create / Modify / Delete	×	×	×	×	×
Event	Read / View / Create / Modify / Delete	×	×	×	×	×

Interfaces PostgreSQL Rest Api Data CMS

Data is collected from various Interfaces and updated in scheduled Syncs.

Primary Database: PostgreSQL. Benefits:

A REST Api where all Data can be queried.

Data can be viewed and filtered across the provided Data CMS.

Figure 3.4: Logging in to the CMS portal.

If you now try to access the **ODH Data** item in the top menu, you will be able to select some dataset. As an example, Figure 3.5 shows what is available in the *ODH Data* → *Activities & Pois* → *Winter* filter - in this case a list of activities that can be done during the winter on the snow.

The page allows to further filter the results, by using search strings and/or the list of tags underneath, to move between pages of results, and to change language of the interface (although at the moment the page is not fully translated in all languages!)

If you click on one of the images in the list will pop up an overlay with more detailed information about that activity.

3.4 Quick and (not-so) Dirty Tips for Tourism

This section contains various tips and tricks to improve and tweak the queries sent to the Tourism datasets, allowing more precise results to be retrieved. This page is divided into two parts: The first one shows examples with code (usually the API call), the second is organised like a FAQ section.

3.4.1 Example Calls

EX1. Why does this query return no result?

```
http://tourism.opendatahub.bz.it/api/Gastronomy?pagesize=3&categorycodefilter=0&
↳locfilter=reg268
```

Because there is no value **reg268** for *locfilter*. You can return valid IDs to be used as *locfilter* using this call:

```
http://tourism.opendatahub.bz.it/api/RegionReduced?language=it
```

An example result for this call is:

```
{
  "Id": "D2633A26C24E11D18F1B006097B8970B",
```

(continues on next page)

IDM Open Data Hub CMS ODH Data external Data Sources Hello stefano.david@opendatahub.bz.it! Log off

Activities & Pois - Winter

Info

de it **en** nl cs pl fr ru

Elements: 838 prev 1 / 42 next

Filter Clear

Activity/Poi Name Clear

Name

Location Clear

Tourismverein

GB Tag Clear

Name

Highlight Clear

☐

Typ Clear




MainImage	Shortname	Type	Location	Languages	Source	OA	GB Active
	Magic Town - The Val Gardena Christmasmarket for children and families!	Winter - Christmas & Christmas Markets - Rural Christmas Markets	Val Gardena/Gröden - S. Cristina/St. Christina	de it en	Magnolia		✓
	Winter Hike "Durchs Toul"	Winter - Winter Hiking	Brunico/Bruneck and environs - Valle Aurina/Ahrntal	de it en nl cs pl	Magnolia		✗
	08 Dobbiaco/Toblach: Three Peaks round	Winter - Cross-Country Skiing	Alta Pusteria/Hochpustertal - Dobbiaco/Toblach	de it en nl cs pl	LTS	OA	✗

Figure 3.5: Accessing the data through filters or menu item.

IDM Open Data Hub CMS ODH Data external Data Sources Hello stefano.david@opendatahub.bz.it! Log off

Activities & Pois - Winter

Info

de it en nl cs pl fr ru

Elements: 838 prev 1 / 42 next

Filter Clear

Activity/Poi Name Clear

Name

Location Clear

Tourismverein

GB Tag Clear


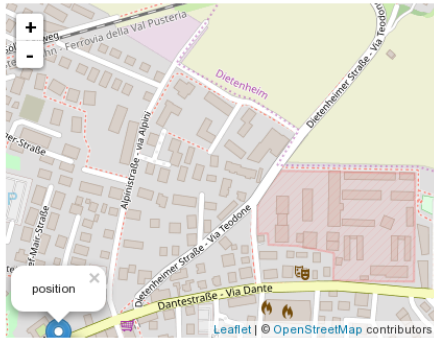
Name

Highlight Clear

☐

Typ Clear

Poi Info

Winter Hike "Durchs Toul"

Winter - Winter Hiking

Introtext
An idyllic winter hike running through an entire valley (hiking ability required)

Basistext
This hike leads through the whole of the Tauferer Ahrntal Valley, stretching about 50 kilometres from Brunico/Bruneck to Kasern, and is known as "Toul". The trail passes castles, churches, chapels, wayside crosses and picturesque mountain farms along the Aurino/Ahr River. The unique character of every village you pass will simply delight you. The Valle Aurina/Ahrntal Valley and Campo Tures/Sand in Taufers Chapters of the South Tyrol Alpine Association (AVS) created the trail in 2003. Toul can be done in segments, which makes it a particularly family-friendly hike throughout the entire year.

Figure 3.6: Detailed view of a POI (Point Of Interest).

(continued from previous page)

```
"Name": "Alta Badia"
},
```

Therefore, use the ID **regD2633A26C24E11D18F1B006097B8970B** in *locfilter* to search for Gastronomy in the Alta Badia region.

EX2. The *locfilter* parameter.

Q: How do I correctly use the *locfilter* parameter?

```
locfilter =>
Locfilter (Separator ',' possible values: reg + REGIONID = (Filter by
Region), reg + REGIONID = (Filter by Region), tvs + TOURISMVEREINID =
(Filter by Tourismverein), mun + MUNICIPALITYID = (Filter by
Municipality), fra + FRACTIONID = (Filter by Fraction)),
(default:'null')
```

It seems to accept a string, but how is this string built?

A: *locfilter* accepts a string composed as follows: a region identifier, followed immediately by a location Identifier.

Location identifier are the following four:

- **reg:** Region (Italian Regione)
- **tvs:** Turistic association (German Tourismusverein)
- **mun:** Municipality, i.e., town or city (Italian Municipalità)
- **fra:** Suburb or district (Italian frazione)

IDs for each location can be gathered either from the swagger interface or using an API calls:

- **reg:**

```
http://tourism.opendatahub.bz.it/swagger/ui/index#!/Common/Common_
↪GetRegionsReduced
http://tourism.opendatahub.bz.it/api/RegionReduced?language=it
```

- **tvs:**

```
http://tourism.opendatahub.bz.it/swagger/ui/index#!/Common/Common_
↪GetTourismvereinReduced
http://tourism.opendatahub.bz.it/api/TourismAssociationReduced?
↪language=itourismusverein)
```

- **mun:**

```
http://tourism.opendatahub.bz.it/swagger/ui/index#!/Common/Common_
↪GetMunicipalityReduced
http://tourism.opendatahub.bz.it/api/MunicipalityReduced?language=it
```

- **fra:**

```
http://tourism.opendatahub.bz.it/swagger/ui/index#!/Common/Common_
↪GetDistrictReduced
http://tourism.opendatahub.bz.it/api/DistrictReduced?language=it
```


For example, to retrieve all Gastronomy in the suburb of Lana, first retrieve its ID, which is:

```
{
  "Id": "79CBD79551C911D18F1400A02427D15E",
  "Name": "Lana"
},
```

Then pass the string **fra79CBD79551C911D18F1400A02427D15E** as *locfilter*:

```
http://tourism.opendatahub.bz.it/api/Gastronomy?
↳locfilter=fra79CBD79551C911D18F1400A02427D15E
```

EX3. The *categorycodefilter* parameter.

Q: *categorycodefilter* seems similar to the *locfilter* parameter found in [this trick](#), but this does not accept string?

```
Category Code Filter (BITMASK values: 1 = (Restaurant), 2 = (Bar /
Café / Bistro), 4 = (Pub / Disco), 8 = (Aprés Ski), 16 =
(Jausenstation), 32 = (Pizzeria), 64 = (Bäuerlicher Schankbetrieb),
128 = (Buschenschank), 256 = (Hofschank), 512 = (Törggele Lokale),
1024 = (Schnellimbiss), 2048 = (Mensa), 4096 = (Vinothek /Weinhaus /
Taverne), 8192 = (Eisdiele), 16384 = (Gasthaus), 32768 = (Gasthof),
65536 = (Braugarten), 131072 = (Schutzhütte), 262144 = (Alm), 524288 =
(Skihütte)
```

The *categorycodefilter* parameter accepts integers instead of strings, in *bitmask-value*. The code of each category is a power of 2, so to search in multiple categories, simply **add** the respective codes and pass them as value of the parameter. For example, to search for Restaurants (1) and Pizzerias (32), pass **33** to *categorycodefilter*:

```
http://tourism.opendatahub.bz.it/api/Gastronomy?categorycodefilter=33
```

3.4.2 Tips and Tricks

TT1. *Categorycodefilter* in the Accommodation dataset.

Q: In the Accommodation dataset there's no *categorycodefilter* filter, like in the Gastronomy dataset. Is there some equivalent filter?

A: In the Accommodations dataset use **categoryfilter** instead.

TT2. *odhactive* and filters starting with *odh*.

Q: What is the purpose of the *odhactive* filter? And what do all the filters prefixed with **odh** stand for?

A: In the datasets, there are filters like *active* and *odhactive*, where *odh* simply stands for Open Data Hub. Filters starting with **odh** are collectively called *odhtags*.

Datasets filtered with the former return all data sent by the dataset provider, while the latter returns those validated by the Open Data Hub team as well. This parameter is useful in a number of use cases. Suppose that the Open Data Hub team receives a dataset contains name and location of ski lifts within South Tyrol's ski areas. If the dataset has not been updated in a few years, some entry in that dataset might be non valid anymore, for example a ski lift has been

replaced by a cable car or has been dismantled. If this case has been verified by the Open Data Hub team, the entry referring to that ski lift will not appear in the Open Data Hub.

TT3. The *seed* filter

Q: What is the *seed* filter used for?

A: *seed* is used in pagination, i.e., when there are two or more pages of results, to keep the sorting across all pages. When retrieving a high number of items in a dataset it is desirable to have only a limited amount of results in each page.

It is possible to activate seed in two ways: in the dataset, choose a *pagenumber* (the number of the result page that will be shown first) or a *pagesize* (number of items in each page, we'll use **15** in this example) and set seed to **0**. At the beginning of query's **Response Body** you will see something like:

```
{
  "TotalResults": 10564,
  "TotalPages": 705,
  "CurrentPage": 1,
  "OnlineResults": -1,
  "Seed": "43",
  "Items": [
    {
```

The remainder of the **Response Body** contains the first 15 sorted items. If you now want to retrieve page 2, page 56, or any other, use **43** as seed and write **2**, **56**, or the desired value as *pagenumber*.

If you do not enter the **seed**, you could find an item that was already shown before, because the API can not guarantee that the same sorting is used in different queries.

3.5 How to use authentication


As described in section [Authentication](#), there are two methods to access protected data in the dataset: Bearer Token Login and OAuth2 authentication. Both authentication methods can be used within a browser or from the command line, with only slight differences. In this section we show how to use authentication within the Open Data Hub, provided that you owe an username and a password to access the closed data in the datasets.

To obtain the credentials, please address your enquiry to the contact email of the dataset you would like to access.

3.5.1 Bearer Token Login

Bearer token login is used to access the *Datasets in the Tourism Domain*; description of the procedure is available at *Data Access and Manipulation*.

3.5.2 OAuth2 authentication

OAuth2 authentication can be used in all the *Datasets in the Mobility Domain* that are marked with the  badge, so pick one dataset and go to its swagger interface, whose URL is provided together with the information of the dataset.

Note: As of May 20, 2019, authentication is not yet publicly available, so the following guidelines can not yet be put in practice.

If you use a browser

Make sure you have obtained a valid username and password, then open the `/rest/refresh-token` method and write you username and password in the two **user** and **pw** fields, respectively, as shown in [Figure 3.7](#).

Name	Description
user * required string (query)	The username of the user to which to grant the new token.
pw * required string (query)	The password corresponding to the user.

Figure 3.7: Request a new OAuth2 token.

If your credentials are valid, you will receive a new token, otherwise the response will be a **401 Unauthorized** error message.

The token you received can be used in any of the API's methods that require authorisation. A sample call is shown in [figure Figure 3.8](#). Note the syntax of the `Authorization` parameter: You must use prefix the authentication token with the **Bearer** string, followed by an empty space, then by the token.

In case you do not respect the `Authorization+space+token` sequence, use additional separators in the sequence (like [Figure 3.9](#) shows), or use an invalid token, you will receive an **401 - Unauthorized** HTTP response.

If you use the Command Line Interface.

Open a shell on your workstation and use a tool like `curl` or `wget`, with the appropriate options:

-X

Specify the request method (GET)

--header, -H

Add extra header information to be included in the request.

Bearer eyJhbGciOiJIUzUxMU9.eyJzdWIiOiJhbmFsaXQ

station * required

string (query)

station

83

name * required

string (query)

name

CP1-Tignale

seconds * required

integer (query)

seconds

50

period

integer (query)

period

period - period

Execute

Clear

Responses

Response content type */*

Curl

```
curl -X GET "http://btp-test-env.b7twguhvj.../emobility/rest/get-records?station=83&name=CP1-Tignale&seconds=50" -H "accept: */*" -H "Authorization: Bearer eyJhbGciOiJIUzUxMU9.eyJzdWIiOiJhbmFsaXQ6eXRoZXJvc5jbn0iLCJleHAiOiJlMjkwMzRlNDQ6InJvbmV2Zj0iUk9MRV98TGFhbnV3QjI1IiwiaWF0Ij0iZWwRF41PTc3IFV3CDEFEU446B1tbSDJuk1ben7K0IuQh3U0S6m2z4hNfHf6p449R0cC_3Yk8BcwsALq"
```

Request URL

http://btp-test-env.b7twguhvj.../emobility/rest/get-records?station=83&name=CP1-Tignale&seconds=50

Server response

Figure 3.8: A successful call to a method requiring authentication.

(header)

Bearer ey.jhbGoOUJzUwMj9.ey.zdWfOUIhbmFs

station • required string <small>(query)</small> name • required string <small>(query)</small> seconds • required integer <small>(query)</small> period integer <small>(query)</small>	station <input type="text" value="83"/> name <input type="text" value="CP1-Tignale"/> seconds <input type="text" value="50"/> period <input type="text" value="period - period"/>
--	--

Execute
Clear

Responses

Response content type: j*

Curl

```
curl -X GET 'http://bdg-test-env.b7twgwhvj.../embility/rest/get-records?station=83&name=CP1-Tignale&seconds=50' -H "accept: */*" -H "Authorization: Bearer ey.jhbGoOUJzUwMj9.ey.zdWfOUIhbmFs" --data '{"id": "2bhmFseXbpY3MAwBtLxBI2URRaxX2vKcSjb2ILC1LwA10jELNjwaz3INQqToJ2v3oVziItoJ4SHVRVBTSFWWRZQmIFQ_v7L2XRFAIPT3rCFvCDEFFUHG81ts5DJaIk36w7KS1QuqGh3UUSn244WhFnF8wp49908C2_yk8btmsALQ"}'
```


Request URL

```
http://bdg-test-env.b7twgwhvj.../embility/rest/get-record?station=83&name=CP1-Tignale&seconds=50
```


Server response

Code	Details
401	Error: Unauthorized

Response body

```
{
  "status": 401,
  "name": "Unauthorized",
  "description": "ClientResponse has erroneous status code: 401 Unauthorized"
```

Figure 3.9: A failed call to a method requiring authentication.

Note that the `--header` option is used twice: The first to receive the answer in **text/html** format, the second to provide the credentials required to access protected content.

API calls can be done using a tool like **curl** or **wget**, with the same `-X` and `--header` option used twice: The first to require the format of the response, the second to provide the credentials, like for example:

```
curl -X GET "http://bdp-test-env.b7twguhyj.example.com/emobility/rest/get-records?
↪station=83&name=CP1-Tignale&seconds=50" --header "Accept: */*" --header
↪'Authorization: Bearer <token>'
```

Make sure to replace the `<token>` with the actual token you received.

3.6 How to setup your local Development Environment

This tutorial will guide you in the setup of the local infrastructure to be able to deploy, on top of the Big Data Platform, a new Data Collector Object starting from a simple **HelloWorld** template we provide.

This tutorial is divided into three parts:

1. Software installation
2. Services configuration
3. Troubleshooting

Warning: This tutorial is still work in progress and is largely incomplete!

3.6.1 Software Installation

The following installation directions have been verified on a VM with installed either **Debian 9** or **Ubuntu 18.04.01 LTS**. The applications installed on it are the **Suggested** version.

Note: All the commands and configuration items (including their location in the filesystem) refer to this distribution and should be identical or quite similar on all other debian-based distributions as well.

On other Linux distribution some the name of the single packages might vary.

You need to install the following software:

Software	Mini-mum	Sug-gested	Notes
Post-greSQL	9.6	10.5	
postgis ext.	2.2	2.4	
Java	JRE7	JRE8	Most of the packages require Java 8 to be built.
git	2.17	2.17	
xmlstarlet	1.6.1	1.6.1	
Apache Maven	3.3.9	3.5.2	Optional. If you don't use it, do not install it.
tomcat8	8.0	8.5	You Optional can either use the tomcat server provided by Open Data Hub or install another application server.

Note: In case you opt to not use Maven or Tomcat, remember to edit the script in order to not attempt to configure them!

On a typical debian-based Linux distribution, installing the software is achieved by opening a shell/terminal, then issuing the following command, provided you have the rights to install software:

```
odh@bdp:~$ sudo apt-get install git openjdk-8-jdk postgresql postgis maven tomcat8_
↪xmlstarlet
```

This command ensures that all dependencies are installed as well. If you have none of these package already installed, you might need to download up to ~125Mb of packages.

3.6.2 Services configuration

The services will be configured automatically, since we developed a script that does most of the job for you. However, a few preliminary steps are required:

1. Make sure tomcat8 and postgres are running. If they do not or if you are unsure, refer to [entry 1](#) in section [Troubleshooting](#).
2. Verify that tomcat and postgres are listening on the right port (**8080** and **5432** respectively). See [entry 2](#) in section [Troubleshooting](#) for more information.
3. Make sure there is a database role configured with a password and a few access permission.
4. Set two environment variables.
5. Edit the script to suit your workstation.
6. Launch the script.

Warning: The script **might silently fail** on some machine, for example on Ubuntu 18.04, because it ships with Java 11. In this case, please install also java 8 and make it the default java version.

3.6.3 Troubleshooting

1. How do I check if a service is running?

You can check that a service like tomcat or postgres is running from the CLI, by issuing the following command and see an output similar to the one shown here, where the active (running) string can be read.

```
odh@bdp:~$ service tomcat8 status
tomcat8.service - LSB: Start Tomcat.
  Loaded: loaded (/etc/init.d/tomcat8; bad; vendor preset: enabled)
  Active: active (running) since Wed 2018-06-13 16:36:28 CEST; 14min ago
    Docs: man:systemd-sysv-generator(8)
   CGroup: /system.slice/tomcat8.service
           └─13828 /usr/lib/jvm/java-8-openjdk-amd64/bin/java -Djava.util.logging.
↪config.file=/var/lib/tomcat8/conf/lo

Jun 13 16:36:23 bdp systemd[1]: Starting LSB: Start Tomcat....
Jun 13 16:36:23 bdp tomcat8[13802]: * Starting Tomcat servlet engine tomcat8
Jun 13 16:36:28 bdp tomcat8[13802]: ...done.
Jun 13 16:36:28 bdp systemd[1]: Started LSB: Start Tomcat..
```

If you do not use systemd, the command will have a different output:

```
odh@bdp:~$ service tomcat8 status
[ ok ] Tomcat servlet engine is running with pid 11357.
```

From a browser you should connect to <http://localhost:8080/> (replace `localhost` with the URL or IP where your application server is located) and see the following page:

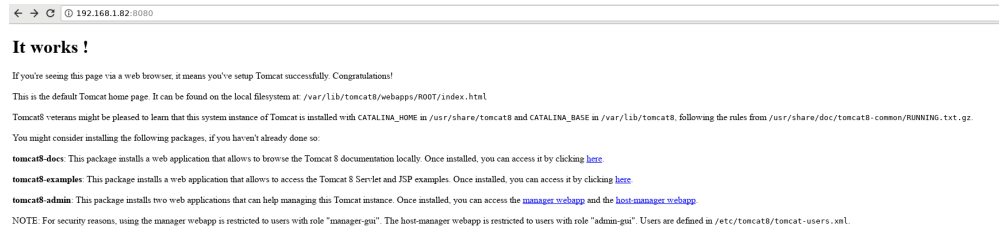


Figure 3.10: The tomcat8 default landing page.

If tomcat is not running, start it using the following command, then entering your password.

```
odh@bdp:~$ sudo service tomcat8 start
[sudo] password for odh:
```

You can check again if tomcat is running with the command **service tomcat8 status**.

2. How do I check the port on which a service is listening?

You can use the **netstat** command line utility, like this:

```
root@bdp:~$ netstat -plnt4
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:5432            0.0.0.0:*               LISTEN      2427/postgresql
tcp        0      0 0.0.0.0:22             0.0.0.0:*               LISTEN      2719/sshd
tcp        0      0 127.0.0.1:8080         0.0.0.0:*               LISTEN      2863/tomcat8
```

Make sure that at least ports 8080 and 5432 are present (tomcat and postgres respectively) in the **Local Address**.

It is suggested to run this command as superuser, because otherwise not all information is present.

3.7 How to Set Up Postman (API Development Environment)

Postman is a popular API development environment, that is, a tool that is used (among other useful features) to ease the interaction with API calls to remote sites. In this tutorial, we show the few steps necessary to set Postman to connect to the Open Data Hub datasets in both the mobility and tourism domains.

In the remainder of this tutorial, we will use as example the *E-chargin station* dataset, located at <http://ipchannels.integreen-life.bz.it/emobility/swagger-ui.html> for the mobility domain and the *Accommodation* dataset, located at <http://tourism.opendatahub.bz.it/swagger/ui/index#/Accommodation>.

3.7.1 Initial Setup

After Postman has been launched, click on the New button, then on Request to start the configuration of the Open Data Hub endpoints, like shown in Figure 3.11.

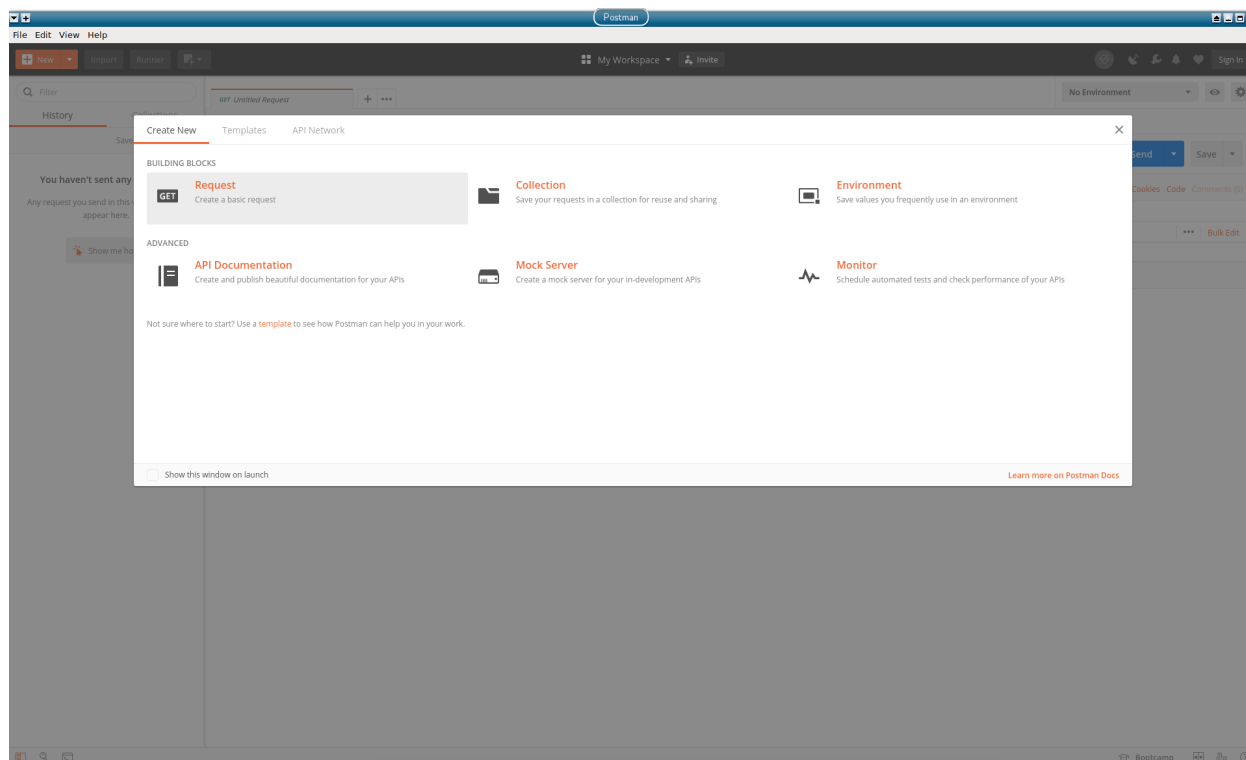


Figure 3.11: Start of a new request creation.

In the dialog window that opens, write the URL of the endpoint in the *Request name* textfield and assign it in the ODH collection, see Figure Figure 3.12.

Hint: If no collection has already been created, create one by clicking on + Create collection, then write **ODH** and confirm.

Click on Save to ODH to start querying the endpoint.

Repeat the procedure for the Accommodation dataset and for any other dataset you want to query.

It is now possible to start querying the endpoints, by providing next to the **GET** button the corresponding call, like shown in Figure 3.13 for the E-charging station dataset and in Figure 3.13 for the Accommodation dataset. However, while the former images shows a set of results, on the latter appears the message *Authorization has been denied for this request.* and the status 401 Unauthorized.



Figure 3.13: Querying the E-charging station endpoint.

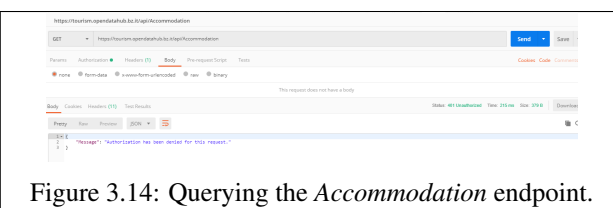


Figure 3.14: Querying the Accommodation endpoint.

SAVE REQUEST



Requests in Postman are saved in collections (a group of requests).

[Learn more about creating collections](#)

Request name

http://ipchannels.integreen-life.bz.it/emobility/swagger-ui.html

Request description (Optional)

Adding a description makes your docs better

Descriptions support Markdown

Select a collection or folder to save to:



Search for a collection or folder

◀ ODH

+ Create Folder

Cancel

Save to ODH

The reason is that the data contained in that dataset have not (yet) been published as open data, therefore authentication is necessary. This is where Postman proves useful, since it can request authentication tokens (OAuth2 in the case of Open Data Hub), store them, and use them whenever they are needed.

3.7.2 Getting a new Authorisation Token

To request a new authorisation token, click on *Authorization* right below the GET request, then select OAuth 2.0 as the *Type*.

Now, in the right-hand side of the window, write the URL that manages the tokens (for the tourism domain, this is <http://tourism.opendatahub.bz.it/token> and click on the Get New Access Token button (Figure 3.15).

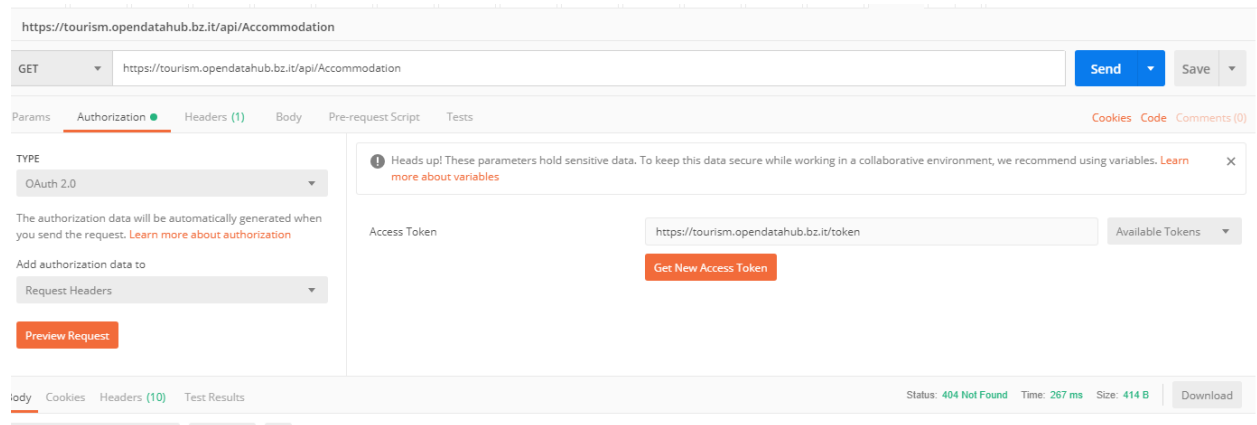


Figure 3.15: Requesting an access token.

In the dialog window that opens fill in all the necessary fields, like shown in Figure 3.16, selecting **Password Credentials** as the *Grant Type*, then click on Request Token. Make sure you have received the username and password to obtain the token, and give it a name easy to remember.

If your credentials are correct and the request is successful, the dialog window will be replaced by another one containing the access token and a few details about it, including its validity and expire date, see Figure 3.17 and Figure 3.18.

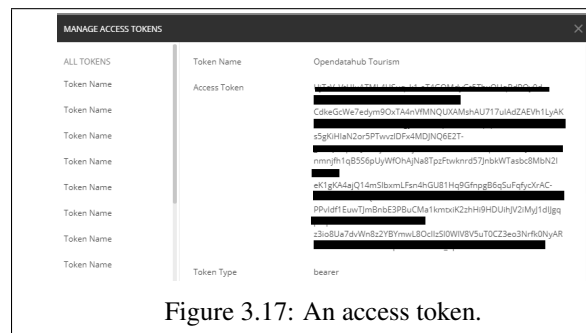


Figure 3.17: An access token.

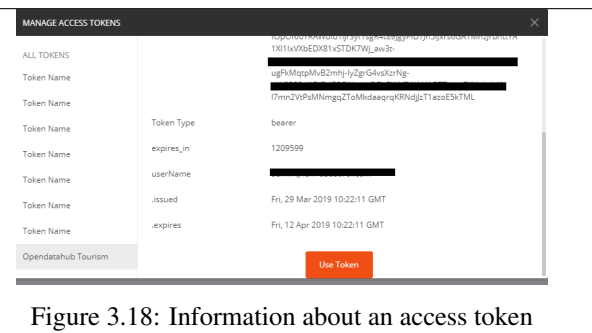
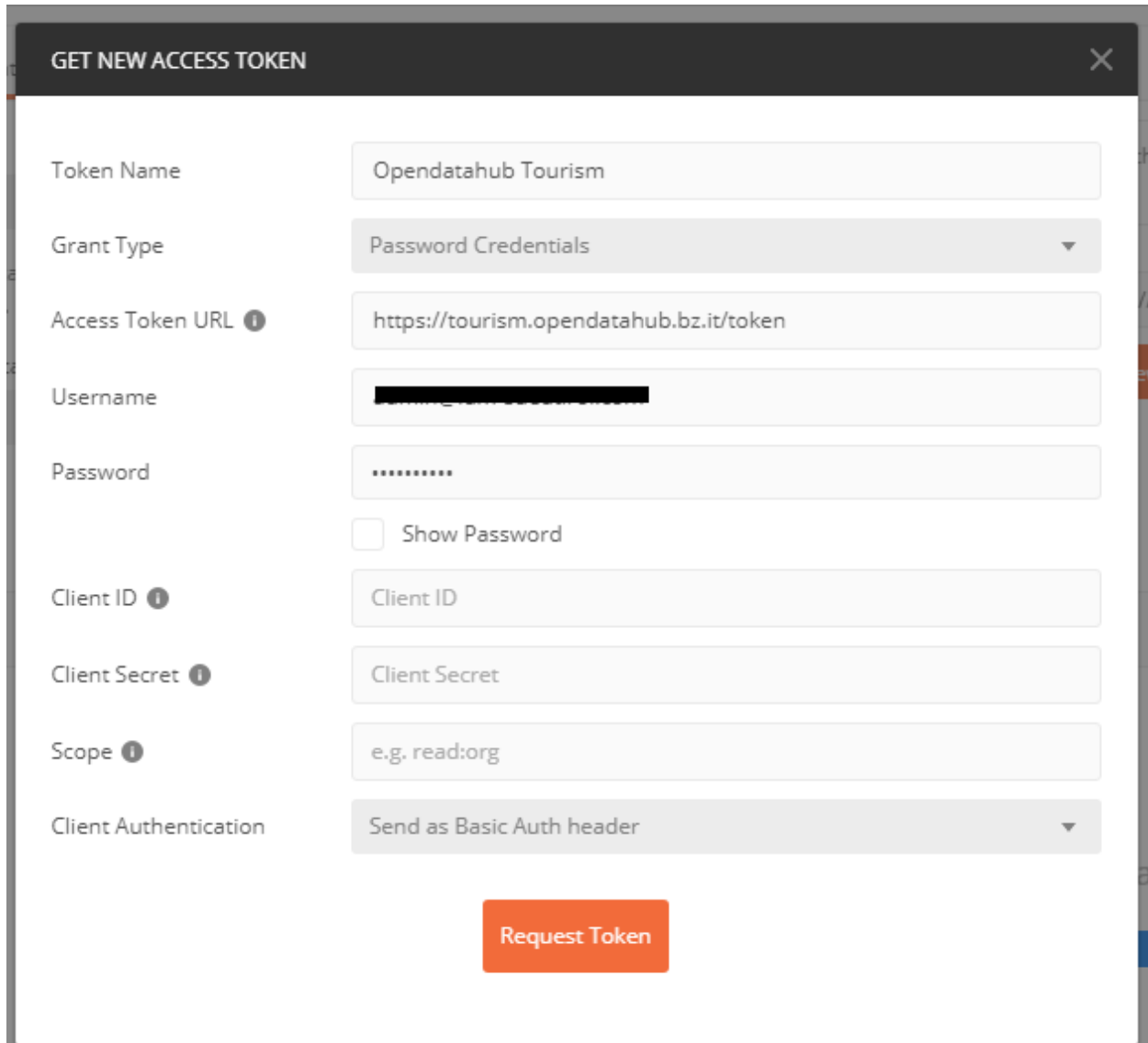


Figure 3.18: Information about an access token

It is now possible to select the token: Select **Opendatahub Tourism** from the *Available Tokens* drop-down menu (see Figure 3.15), click on *Body* and repeat the GET request. You should be able to see now the data in the dataset, like shown in Figure 3.19.



The image shows a 'GET NEW ACCESS TOKEN' dialog box in Postman. It contains the following fields and values:

- Token Name:** Opendatahub Tourism
- Grant Type:** Password Credentials
- Access Token URL:** https://tourism.opendatahub.bz.it/token
- Username:** [Redacted]
- Password:** [Redacted]
- Show Password:** ☐
- Client ID:** Client ID
- Client Secret:** Client Secret
- Scope:** e.g. read:org
- Client Authentication:** Send as Basic Auth header

A red 'Request Token' button is located at the bottom center of the form.

Figure 3.16: A filled-in token request.

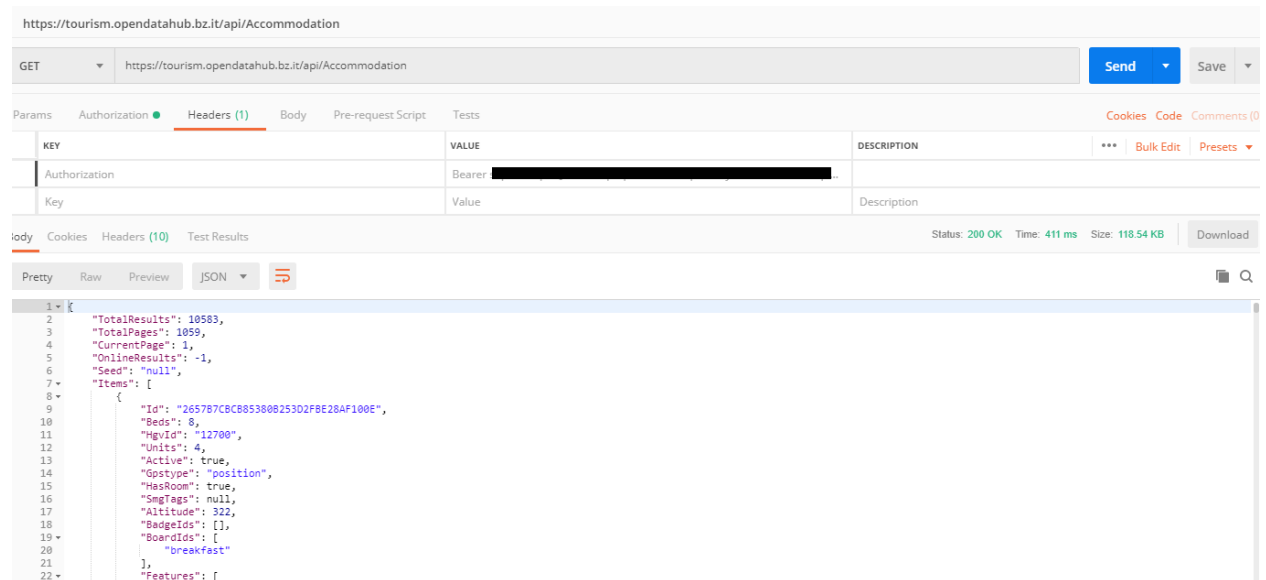


Figure 3.19: Access to data requiring authorisation.

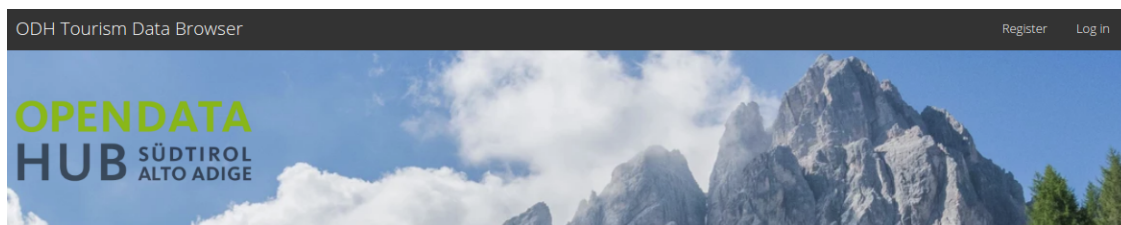
3.8 How to insert and modify NOI Events

After reading this article, you will be able to use the Open Data Hub tourism portal to insert, modify, and delete events that take place at NOI Techpark in Bolzano (in the remainder, **NOI events**).

3.8.1 Preliminaries

Since you must login to create events, you need valid credentials to be able to add NOI events, that you should have received from the Open Data Hub team.

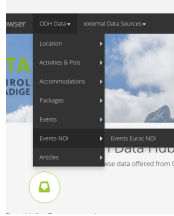
Go to <http://tourism.opendatahub.bz.it> and click on Log in (top right corner)



Provide your credentials, then you will be redirected to your homepage, that shows among other information, the roles you have within the Open Data Hub.

3.8.2 Creation of a new NOI Event

Once logged in, go to *ODH Data* → *Events NOI* → *Events EURAC NOI*.



You will now see a list of events that will take place at Bolzano's NOI Techpark today or in the next days. For each event, the description, start and end date, and the location where it takes place are shown. If the event is marked as **Active**, it is displayed on the official NOI web page at <https://today.noi.bz.it/>.

Events Eurac/NOI List

Info

de it en

Elements: 138 prev 1 / 7 next

New

Description	Begindate	Enddate	Location	Active
Master Imprenditorialità ed Innovazione - II Semestre	06/05/2019 14:00	10/05/2019 13:00	NOI	✗
etestdays 2019	08/05/2019 08:00	08/05/2019 14:00	NOI	✓
Digital Academy DAY 1	08/05/2019 09:00	08/05/2019 12:00	NOI	✓
Meeting NOI AG	08/05/2019 09:30	08/05/2019 10:30	NOI	✓
Corso di marketing - cancellato	08/05/2019 15:00	08/05/2019 18:00	NOI	✗
S-competition	09/05/2019 09:00	09/05/2019 18:00	NOI	✓
Asphaltstraßen - Baustellenlogistik als Schlüssel zur Lösung vieler Probleme	09/05/2019 09:00	09/05/2019 16:30	NOI	✓
Harp Legacy	10/05/2019 08:00	11/05/2019 23:00	NOI	✗
Agile Venture Bolzano	10/05/2019 08:30	10/05/2019 18:00	NOI	✓
Legnattivo - Kick-off Meeting	13/05/2019 09:00	13/05/2019 13:30	NOI	✓
Master Imprenditorialità ed Innovazione	13/05/2019 10:00	17/05/2019 17:00	NOI	✓

Filter Clear

Event Name

Name

Source

all

Location

NOI

Active

all

DatumsFilter

08-05-2019

In order to add a new event, click on the New button to create a new event.

In the dialog that opens, fill in all the fields you deem necessary, especially the title and description at least in one of the three available languages.

Event NOI Details ×

General Images

Id

Id

Title(DE)

Event Title DE

Title(IT)

Event Title IT

Title(EN)

Event Title EN

Desc(DE)

Event Text DE

Desc(IT)

Event Text IT

Desc(EN)

Event Text EN

Active

noi.bz.it Active

Organizer

Organizer

Web Page (URL)

Web Url

Video (URL)

Video Url

Date Start

08-05-2019

Start Time

__:

Date End

08-05-2019

End Time

__:

Room

Room Name

Location

NOI

Manage Rooms

Room Management

Create

Cancel

Remember to tick the *Active* and *noi.bz.it Active* checkboxes: The latter allows the event to show up on <https://today.noi.bz.it/>.

If the event is set to take place in more rooms, click on the Room Management button to add more rooms and time slots to the event.

If the event has a web page and/or a video trailer, you can add a link to them in the *Web Page (URL)* and *Video (URL)* text-fields.

It is even possible to add images to the event, by clicking on the *Images* tab on top of the dialog and then on Choose File to upload a file. For each image, a few information can be added:

- The author's name.
- The licence used for the image, either **Proprietary** or **CC0**. 

Hint: We prefer that a **CC0** licence be used; it is necessary to have the rights to upload the photo with **CC0**.

- The position of the image within the gallery, if you upload more than one image. Image in position **0** will be the cover page of the gallery

When you provided all the necessary information, click on Create to create the event.

If you later need to modify the event, click on the Edit button next to the event in the event list.

Mobility

1. *HOWTO access e-Charging Stations Data*. Since the APIs are very generic, directions contained in this howto can be applied to any dataset of the mobility domain.

Tourism

1. *HOWTO access Tourism Data*. This howto describes the various filters available in the Tourism domain
2. *HOWTO use the Open Data Hub's Tourism Data Browser*. This howto provides information about accessing the data originating from the Tourism domain.
3. *Quick and (not-so) Dirty Tips for Tourism*. This howto provides some simple use cases/API calls and various tricks&tips for the Tourism domain.

Miscellaneous



1. *How to use authentication*. Developers and Code Contributors have a dedicated tutorial to help them bootstrapping the environment and start coding right away.
2. *How to setup your local Development Environment*. This tutorial is still in development and not so useful at the moment!
3. *How to Set Up Postman (API Development Environment)*. This howto guides you in the setup of Postman, a popular API development environment.
4. *How to insert and modify NOI Events*. In this howto you will learn how to enter and modify NOI events directly from the Open Data Hub portal.

Apps built from Open Data Hub datasets

This section features a list of applications, and some additional useful information about them, built using the *Datasets* that the Open Data Hub team makes available.


The additional information may include the home page of the app, the contact address of the developers, a description, and the type of the application (mobile app, web page, and so on)..

However, the most important attribute of each project is probably its **status**, which might be one of these three:

- Experimental status, aka *alpha stage*, shown by the  badge. Applications falling in this category are in the early stage of development, and might be for example, a proof of concept or the outcome of a hackathon. They might not be maintained or developed further and should not be considered mature enough to be deployed in a production environment.
- Development status, aka *beta stage*, shown by the  badge. Application in beta stage are developed actively and might already be suitable for a production environment.
- Production status. These applications are already used in production environment.

If you are developing one application with the data provided by datasets of the Open Data Hub project, send an email with a short description, an email contact and its status and we'll add it to the list.

4.1 Alpha Stage Apps

All the projects listed in this section have been built during various hackathons and must be considered as *Experimental*. 

Summer Lido Hackathon 2018 

HackTheAlps 2018 

Projects developed during the Summer Lido Hackathon 2018

- [South Tyrol Crime Scene \(STCS\)](#) is an interactive thriller combining traditional storytelling and augmented reality.
- [SAMA - Smart Application Medical Appointment](#) is a prototype for the booking of appointments in the South Tyrolean hospitals.
- [IFC Converter and AIVRTour](#) is on the one side a converter of 3D reality data formats (from ifc to dae/obj), while on the other side it applies AI to virtual reality for museums, real estates, and on tourism.
- [Travel & Win](#) is an app for South Tyrol tourists that can collect points to receive gifts.
- [Memorama](#) Get your pictures printed on paper and delivered to your hotel.
- [Sportmap.net](#) Creation of a OpenData map with trendy sport locations, courses, & events.
- [Game of Alps](#) Get unique experiences by completing challenges around South Tyrol. Gather crystals and collect rewards from local tourist offices. Additionally get discounts for restaurants and entrance tickets.

Projects developed during [HackTheAlps 2018](#).

- [Activity Crystal](#) goal of the project was to build something simple & fun that gets people off their couches.

4.2 Beta Stage Apps

- <https://mobility.meran.eu>. This web site is the first example of a Mobility-as-a-Service application; it includes real-time information of multiple mobility services, like public transportation, places of interests, car sharing services, parking lots, and more.
- <https://parking.bz.it>. A web site that displays the real-time parking availability of off-street parking lots in South Tyrol. On mobile devices, it can also show directions from your current position to the chosen parking lot.
- <http://traffic.bz.it>. Some streets in South Tyrol are monitored for real-time vehicular travel times; the data collected are used by this web site to show traffic slowdowns or jams.
- <http://bus.bz.it>. This web site shows the real-time positions of the buses managed by the public transport operator SASA. Urban or suburban bus lines can be shown, and for each bus can be shown the next few stops and an estimate of the arrival time.
- <http://map.clean-roads.eu>. One of the CLEAN-ROADS project outcomes, this web site shows real-time data of the meteorological stations that are situated along public streets.

4.3 Production Stage

- <http://www.sudtirol.info>. This website uses data from the *Datasets in the Tourism Domain* to display events in the region of South Tyrol and other useful information to help tourists organise their holiday in South Tyrol.
- South Tyrol Guide, the official smartphone app for exploring and experiencing South Tyrol, available for both [Android](#) and [iPhone](#) mobile devices.

Frequently Asked Questions

This section contains answers to question frequently asked by people who want to contribute to the Open Data Hub project or search for information about the project.

Q: What is this project about? A: The project is described in section *Introduction*.

Q: I am interested in taking part in the Open Data Hub project. A: Check section *How To Contribute*.

Q: I am a developer, are there guidelines for the development? A: Sure! Check the dedicated section: *Resources for Developers*.

Q: How do I access the data served by the Open Data Hub? A: You can see if section *List of HOWTOs* contains what you are looking for. If not, you can open an issue in our bug tracker.

Q: The project misses a... [feature, dataset, howto, etc.]! A: Please check section *Bug reporting and feature requests* then open an issue on one of our bug trackers.

The following is the list of appendices.

6.1 Datasets



The goal of the Open Data Hub Project is to make available datasets containing data about the South Tyrolean Ecosystem, to allow third parties to develop novel applications on top of them, consuming the exposed data. These applications may range from a simple processing of datasets to extract statistical data and to display the result in different graphic formats like pie-charts, to far more complex applications that combine data from different datasets and correlate them in some useful way.

Note: This page was last updated on 15th January 2019, hence all information about the availability of datasets is correct as of this date. This page will be updated in due time as soon as more material will be made available.

This page will be soon removed, as all the information statically provided here will be in the near future be replaced by *the broker service*, which dynamically maintains the list.

As seen in [Figure 1.1](#), data originate from different domains (Mobility, Tourism, and so on); they are gathered from sensors and packed within **datasets**. *Sensors* can be for example GPS devices installed on buses that send their real-time geographic position or a small electronic device on a plug of an e-charging station that checks the if the plug is being used or not, to let people know that the charging outlet is available.

A note about datasets.

At the time of writing, only a few datasets are published. As mentioned before in this section, the goal is to expose datasets containing **only Open Data**, which is at the moment not the case for all datasets. Indeed, some of the datasets contain data that can not be distributed under an open licence like, e.g.,  or . Therefore, to allow the highest possible data to be shared, an authentication mechanism has been implemented, to prevent access to the data in the datasets that has not yet been published as Open Data.

Datasets which require authentication are marked with the **authorisation required** badge. If you are interested in accessing closed data, you can try to contact the dataset owner or express your interest to the Open Data Hub team.

Please refer to section [Authentication](#) for details.

For each domain the available datasets are listed. Please refer to the next sections for a complete list.

6.1.1 Datasets in the Mobility Domain

List of datasets in the mobility domain.

- *it.bz.opendatahub.weather*
- *it.bz.opendatahub.environment*
- *it.bz.opendatahub.parking*
- *it.bz.opendatahub.bluetooth*
- *it.bz.opendatahub.trafficstation*
- *it.bz.opendatahub.linkstation*
- *it.bz.opendatahub.streetelements*
- *it.bz.opendatahub.rwisstation*
- *it.bz.opendatahub.carsharing*
- *it.bz.opendatahub.bikesharing*
- *it.bz.opendatahub.echargingstation*
- *it.bz.opendatahub.carpoolinghub*
- *info.opensasa.realtime*

In this section, the following information are provided for each of the above-listed dataset:

- The licence of the data present in the dataset.
- The output format of the API call.
- An e-mail contact for the dataset.
- The versions of the API that can be used to access the dataset.
- The swagger URL of the APIs.

it.bz.opendatahub.weather

This dataset contains meteorological data provided by the hydrographical Department of South Tyrol.

License	dataset CC0
Output	JSON, mime-type application/json
E-mail contact	info@opendatahub.bz.it
API version	v1
Swagger URL	http://ipchannels.integreen-life.bz.it/meteorology

it.bz.opendatahub.environment

License	authorisation required
Output	JSON, mime-type application/json
E-mail contact	info@opendatahub.bz.it
API version	v1
Swagger URL	http://ipchannels.integreen-life.bz.it/environment/swagger-ui.html

it.bz.opendatahub.parking

License	authorisation required
Output	JSON, mime-type application/json
E-mail contact	info@opendatahub.bz.it
API version	v1
Swagger URL	http://ipchannels.integreen-life.bz.it/parking/swagger-ui.html

it.bz.opendatahub.bluetooth

The data for this datasets are collected by experimental Bluetooth-based sensors and detectors and represent traffic information, since the detectors scan available Bluetooth devices on board of vehicle that drive on.

License	dataset CC0
Output	JSON, mime-type application/json
E-mail contact	info@opendatahub.bz.it
API version	v1
Swagger URL	http://ipchannels.integreen-life.bz.it/BluetoothFrontEnd

it.bz.opendatahub.trafficstation

License	authorisation required
Output	JSON, mime-type application/json
E-mail contact	info@opendatahub.bz.it
API version	v1
Swagger URL	http://ipchannels.integreen-life.bz.it/sensors/swagger-ui.html

it.bz.opendatahub.linkstation

Similar to the *Bluetooth dataset*, data available in this dataset are collected by Bluetooth-based sensors to measure the level of traffic on the stretch of a road.

License	dataset CC0
Output	JSON, mime-type application/json
E-mail contact	info@opendatahub.bz.it
API version	v1
Swagger URL	http://ipchannels.integreen-life.bz.it/LinkFrontEnd

it.bz.opendatahub.streetelements

License	authorisation required
Output	JSON, mime-type application/json
E-mail contact	info@opendatahub.bz.it
API version	v1
Swagger URL	http://ipchannels.integreen-life.bz.it/street/swagger-ui.html

it.bz.opendatahub.rwisstation

License	authorisation required
Output	JSON, mime-type application/json
E-mail contact	info@opendatahub.bz.it
API version	v1
Swagger URL	http://ipchannels.integreen-life.bz.it/roadweather/swagger-ui.html

it.bz.opendatahub.carsharing

License	dataset CC0
Output	JSON, mime-type application/json
E-mail contact	info@opendatahub.bz.it
API version	v1
Swagger URL	http://ipchannels.integreen-life.bz.it/carsharing/swagger-ui.html

it.bz.opendatahub.bikesharing

License	authorisation required
Output	JSON, mime-type application/json
E-mail contact	info@opendatahub.bz.it
API version	v1
Swagger URL	http://ipchannels.integreen-life.bz.it/bikesharing/swagger-ui.html

it.bz.opendatahub.echargingstation

This datasets exposes data about the existing e-charging stations in South Tyrol and their status, including historical data and usage.

License	dataset CC0
Output	JSON, mime-type application/json
E-mail contact	info@opendatahub.bz.it
API version	v1
Swagger URL	http://ipchannels.integreen-life.bz.it/EchargingFrontEnd

it.bz.opendatahub.carpoolinghub

License	authorisation required
Output	JSON, mime-type application/json
E-mail contact	info@opendatahub.bz.it
API version	v1
Swagger URL	http://ipchannels.integreen-life.bz.it/carpooling/swagger-ui.html

info.opensasa.realtime

This datasets shows the real time position of buses operated by SASA in South Tyrol and, through a few subsets, additional information about lines, station boards, and news.

License	dataset CC BY-SA
Output	geoJSON
E-mail contact	info@sasabus.org
API version	v1
Swagger URL	http://sasabus.org/opendata

The additional subsets expose data in different formats:

- info.opensasa.plandata (VDV 451 - VDV 452)
- info.opensasa.stationboard (JSON)
- info.opensasa.news (JSON)
- info.opensasa.rssDE (XML)
- info.opensasa.rssIT (XML)

6.1.2 Datasets in the Tourism Domain

List of datasets in the tourism domain.

- *it.lts.accommodation*
- *it.hgv.package*

- *it.lts.poi*
- *it.lts.activity*
- *it.lts.event*
- *it.bz.opendatahub.activity_poi*
- *it.lts.gastronomy*
- *it.bz.opendatahub.location*
- *it.bz.opendatahub.ski*
- *it.bz.opendatahub.snowreport*
- *it.bz.opendatahub.webcam*
- *it.bz.opendatahub.weather-siag*
- *it.bz.siag.weather*
- *it.bz.siag.museum*

Like in the previous section, the following information are provided for each of the above-listed dataset:

- The licence of the data present in the dataset.
- The output format of the API call.
- An e-mail contact for the dataset.
- The versions of the API that can be used to access the dataset.
- The swagger URL of the APIs.

it.lts.accommodation

License	authorisation required
Output	JSON, mime-type application/json
E-mail contact	info@opendatahub.bz.it
API version	v1
Swagger URL	http://tourism.opendatahub.bz.it/swagger/ui/index#/Accommodation

it.hgv.package

License	authorisation required
Output	JSON, mime-type application/json
E-mail contact	info@opendatahub.bz.it
API version	v1
Swagger URL	http://service.suedtirol.info/swagger/ui/index#/Package

it.lts.poi

License	authorisation required
Output	JSON, mime-type application/json
E-mail contact	info@opendatahub.bz.it
API version	v1
Swagger URL	http://tourism.opendatahub.bz.it/swagger/ui/index#/Poi

it.lts.activity

License	authorisation required
Output	JSON, mime-type application/json
E-mail contact	info@opendatahub.bz.it
API version	v1
Swagger URL	http://tourism.opendatahub.bz.it/swagger/ui/index#/Activity

it.lts.event

License	authorisation required
Output	JSON, mime-type application/json
E-mail contact	info@opendatahub.bz.it
API version	v1
Swagger URL	http://tourism.opendatahub.bz.it/swagger/ui/index#/Event

it.bz.opendatahub.activity_poi

This dataset contains a collection of activities and points of interest (PoI) in the South Tyrol region. The available data have been extracted from different sources, but at the moment only the data about the South Tyrolean museums and wines are freely available without authentication. These data can be obtained by using the keywords **MuseumData** and **SuedtiroIWein** in the *source* filter of the dataset.

License	museum and wine data dataset CC0, other data authorisation required
Output	JSON, mime-type application/json
E-mail contact	info@opendatahub.bz.it
API version	v1
Swagger URL	http://tourism.opendatahub.bz.it/swagger/ui/index#/ODHactivityPoi

it.lts.gastronomy

License	authorisation required
Output	JSON, mime-type application/json
E-mail contact	info@opendatahub.bz.it
API version	v1
Swagger URL	http://tourism.opendatahub.bz.it/swagger/ui/index#/Gastronomy

it.bz.opendatahub.location

License	authorisation required
Output	JSON, mime-type application/json
E-mail contact	info@opendatahub.bz.it
API version	v1
Swagger URL	http://tourism.opendatahub.bz.it/swagger/ui/index#/Common

it.bz.opendatahub.ski

License	authorisation required
Output	JSON, mime-type application/json
E-mail contact	info@opendatahub.bz.it
API version	v1
Swagger URL	http://tourism.opendatahub.bz.it/swagger/ui/index#!/Common/Common_GetSkiAreas

it.bz.opendatahub.snowreport

License	authorisation required
Output	JSON, mime-type application/json
E-mail contact	info@opendatahub.bz.it
API version	v1
Swagger URL	http://tourism.opendatahub.bz.it/swagger/ui/index#!/Weather/Weather_GetSnowReportBase

it.bz.opendatahub.webcam

License	authorisation required
Output	JSON, mime-type application/json
E-mail contact	info@opendatahub.bz.it
API version	v1
Swagger URL	http://tourism.opendatahub.bz.it/swagger/ui/index#/Webcam

it.bz.opendatahub.weather-siag

License	authorisation required
Output	JSON, mime-type application/json
E-mail contact	info@opendatahub.bz.it
API version	v1
Swagger URL	http://tourism.opendatahub.bz.it/swagger/ui/index#/Weather

it.bz.siag.weather

License	authorisation required
Output	XML [#]
E-mail contact	meteo@provinz.bz.it
API version	–
Swagger URL	–

[#]The dataset in German can be downloaded as XML using this URL: https://wetter.ws.siag.it/Weather_V1.svc/web/getLastProvBulletin

it.bz.siag.museum

This datasets contains information about the museums in the South Tyrol region and is retrieved directly from the Open Data portal of the Autonomous Province of Bolzano.

More information about this dataset, including metadata, is available in either Italian or German at the following URLs, respectively:

- IT <http://dati.retecivica.bz.it/it/dataset/musei-in-alto-adige>
- DE <http://daten.buergernetz.bz.it/de/dataset/musei-in-alto-adige>

License	dataset CC0
Output	JSON, CSV, XML [#]
E-mail contact	museen@provinz.bz.it
API version	–
Swagger URL	–

[#]The dataset can be downloaded as XML using this URL: <https://cert.provinz.bz.it/musport/services/MuseumsService.MuseumsServiceHttpSoap11Endpoint/getMuseums>

6.1.3 The Broker

The Open Data Hub Broker is a recently introduced online service (November 2018) that gives an overview and quick access of the datasets available within the Open Data Hub project.

The service is accessible at the URL <https://api.opendatahub.bz.it> and consists of a web page, divided in two parts:

- The **list of datasets** accessible through the broker, which are all the datasets containing publicly accessible data. This list is dynamically created when the page is loaded, therefore it is always up to date. Each dataset can be clicked to open a panel with additional information about it.
- An overview of the **REST API** provided by the broker, a few methods that allow to query the existing datasets. Results are in **JSON-LD** format and use the **DCAT vocabulary**. This means that all the keyword in the result set belong to a W3C standard, allowing the data in the result set to be reused and combined with results from foreign datasets that use the same vocabulary.

The results obtained by querying the broker give a number of metadata about one or more datasets, including the **Contact** for information or reuse of the dataset, and the **Base URL**, allowing for quick and direct access to the data, using the ODH APIs, described in the **Documentation** URL.

Note: While the **Base URL**, when accessed, may give an error, the API calls work and produce correct results. For example, consider the **it.bz.opendatahub.echargingstation** dataset. Calling the method **get-station-details** by using its base URL, <https://api.opendatahub.bz.it/it.bz.opendatahub.echargingstation>, actually produce the expected result, i.e., the list of stations in the dataset with all the information attached to it:

```
curl
"https://api.opendatahub.bz.it/it.bz.opendatahub.echargingstation/rest/get-station-
↪details"
| jq '.' | head -10

[
  {
    "_t": "it.bz.idm.bdp.dto.emobility.EchargingStationDto",
    "id": "ASD_00000038",
    "name": "CAMPING_LATSCH",
    "latitude": 46.622135,
    "longitude": 10.863569,
    "municipality": "Latsch - Laces",
    "capacity": 1,
    "provider": "Alperia Smart Mobility",
```

Broker's REST API

The methods available are the following.

- GET /datasets Returns a list of all datasets in form of a **dcatalog:Catalog**.
- GET /datasets/{id} Retrieve the metadata of a single dataset by its identifier, which corresponds to the **identifier** key that you can find in the in the outcome of the previous method's call. See also the excerpt below. The result set is a **dcatalog:Dataset**.
- GET /datasets/search/{query} Execute a custom, case insensitive query on the available datasets. All the fields within the result set of the GET /datasets query will be considered for an answer. Multiple words can be used as query string.

The outcome of the query looks like the following excerpt which is, as mentioned in the previous section in JSON-LD format and uses the standard DCAT vocabulary.

```
{
  "title": "it.bz.opendatahub.echargingstation",
  "publisher": {
    "title": "Alperia, route220, Nevicam, Driwe",
    "@type": "http://www.w3.org/ns/org#Organization",
    "@context": {
      "title": "http://purl.org/dc/terms/title"
    }
  },
  "keyword": [
    "echarging",
    "mobility",
    "realtime"
  ],
  "identifier": "it.bz.opendatahub.echargingstation",
  "distribution": [
    {
```

```
    "license": "https://creativecommons.org/publicdomain/zero/1.0/",
    "format": "application/json",
    "accessURL": "https://api.opendatahub.bz.it/it.bz.opendatahub.
echargingstation",
  }
},
],
"description": "Real time information about the echarging statons",
"contactPoint": {
  "hasEmail": "info@geobank.bz.it"
},
"@type": "http://www.w3.org/ns/dcat#Dataset",
"@id": "https://api.opendatahub.bz.it/datasets/it.bz.opendatahub.
echargingstation",
}
```

Note: References to some of the definitions of the vocabulary have been deleted from the excerpt for the sake of clarity.

6.2 Resources for Developers

This appendix contains all information that are necessary to developers that want to collaborate with the Open Data Hub team (e.g., developers that send pull requests to the Open Data Hub repositories) or are contracted to write code for the Open Data Hub project (Open Data Hub Core Hacker) or app

6.2.1 Guidelines for Developers

Open Data Hub is a collection of software, databases, and services coordinated and hosted by IDM Südtirol / Alto Adige. Currently, Open Data Hub systems are related to mobility and tourism. In the future Open Data Hub might diversify into more fields.

Companies and developers contributing to Open Data Hub must follow the guidelines listed in the documents as close as possible.

The aim of the Open Data Hub Developer's Guidelines ("**The Guidelines**") is to simplify the hosting and maintenance of the software, databases, and services by the Open Data Hub developers and maintainers at IDM ("the Open Data Hub team").

The Guidelines describe the conventions to which a developer must adhere, to be able to become an active Open Data Hub developer or to see his work being incorporated into the Open Data Hub. They are split in two parts:

- **Platform Guidelines** explain the preferred programming languages, how to expose the data after you manipulated them, the use of third-party libraries or plugins, and so on.
- **Database Guidelines** clarify how to design a database that shall become part of the Open Data Hub platform.

Both of them are summarised in the remainder of this section, and can be found in full version in the pages *Platform Guidelines - Full Version* and *Database Guidelines - Full Version* respectively

Platform Guidelines - Bignami Version

The *Platform Guidelines* contain the software and programming language requirements, coding conventions, and directions for development. This section contains **only** the most important points.

Please check the full version of this document at [Platform Guidelines - Full Version](#) if you want to know more details, if you have some doubt or if what you were looking for is not mentioned in this summary.

- Programming Language is **Java**, in its latest or second to last version.
- The source code **must be documented** according to the [Javadoc style guide and tags](#).
- Java components of Open Data Hub can be developed as **libraries**, **standalone applications**, or **server applications** running in Apache Tomcat.
- The source code is **built nightly**; **build configuration** should be provided in either Ant or Maven (preferred), Makefile, or shell script.
- **Third party libraries** can be used, provided they are established, FOSS-licenced, and do not overlap functionalities. This applies also to third party libraries used in application developed in other languages.
- Front-end applications can be deployed in **Javascript**, version EC 2015, and must support modern browsers.
- **Node.js** can be used to deploy headless or server applications.
- Web front ends use the **latest HTML and CSS** versions, must work on mobile devices (responsive design) and should implement some basic accessibility principle.
- **JSON** must be used as exchange language, while **XML** is welcomed as well.
- The latest or second to last version [Apache Tomcat](#) is used to run server application; only **API/REST end points** have direct access to the database server.
- There's **no file system persistence**, everything must be stored in the DB. JDBC data source and passwords should be stored in environmental variables.
- Pay attention to **RAM usage**, applications will undergo **load testing**.
- **PostgreSQL** RDBMS (Relational DataBase Management System) is used, but not in its recent release (expect to use 2-3 versions before the latest), [PostGIS](#) spatial extension is required as well.
- Developers will have an unprivileged role to access the DB and must follow best practices to query the DB from Java/Javascript.

Database Guidelines - Bignami Version

The *Database Guidelines* contain the database design and database programming principles along with software version requirements. This section contains **only** the most important points.

Please check the full version of this document at [Database Guidelines - Full Version](#) if you want to know more details, if you have some doubt or if what you were looking for is not mentioned in this summary.

- The database can be designed with one of the **Relational Model**, **Object-Relational Mapping (ORM)**, or **Semi-structured Data** methodologies.
- A database designed with either methodology must be shipped with DDL (Data Definition Language) - *schema files* containing the **CREATE** statements.
- Each database must include a **version table** and **indices** on tables.
- All (SQL) source code must be well-documented, with in-line comments and higher level documentation.
- Use standard database features - Sequences, primary and foreign keys, constraints (unique, check, not null), default values, views, and so on and so forth.

- Separate business logic from database design; avoid stored procedure as much as possible.
- Small procedures and functions, if needed. must be written in **PL/PgSQL**.
- Do **not** use foreign data wrappers.
- Consider using declarative partitioning for large tables - and contact Open Data Hub team beforehand to discuss it.
- Always use UTF8 character encoding and do **not** override it.
- Default collation is `en_US`, which works well for German and Italian as well.
- Never use money type, but numeric.
- Dates and time stamps must be store to avoid ambiguity. Never store them as text, but rather use their data types, `date` (in UTC format) and `timestamp with timezone`. Unix timestamp is accepted as well.
- When using or manipulating JSON data always follow **ISO_8601** standard.

6.2.2 Platform Guidelines - Full Version

Changelog

- **2018-05-28 version 1.0**
- 2018-03-30 version 1.0-beta

This document represents **Part 1** of the guidelines and presents the preferred programming languages, databases, and protocols to be used, data exchange and exposition methods, coding conventions, and regulates the use of third-party libraries.

There are scenarios where an exemption from the guidelines is acceptable. The following is a **non-exhaustive** list of such scenarios.

1. **Use of foreign technologies.** The development of a Open Data Hub component requires the use of platforms, languages or generally technologies that are different from the ones listed in the guidelines. An example might be a component that depends on an already developed custom library written in a programming language not listed in the guidelines.
2. **Use of technologies that are not mentioned in the guidelines.** Future Open Data Hub component might require technology that is not listed at all in the guidelines. An example is a component that must be hosted on specific hardware needed for machine learning platforms.

A Open Data Hub contributor who runs into such a scenario must contact the Open Data Hub team to discuss that specific scenario. If the exemption is reasonable and can be motivated the Open Data Hub team will agree and allow it. To avoid misunderstandings, contributors must expect to get a **written statement** about such a decision.

Note: If you can not find any answer to your question or doubt in this document, please contact the Open Data Hub team or open an issue in the [github repository](#) of this document.

Programing Languages, Environments, and Related Technologies

Java

The chosen programming language for Open Data Hub is Java, more precisely the Java Platform, Standard Edition (Java SE).

Source code will be compiled with either the [OpenJDK](#) or the [OracleJDK](#), which share the same code base anyway. Resulting binaries will run in the corresponding JVM.

Java Version

Java is generally backwards-compatible, so code written for a previous version of the JDK will likely compile on the next compiler version and run on the next JVM version. However, contributors ought to use a reasonably modern version of Java in order to avoid deprecation warnings and make use of modern language features.

Contributors can expect the Open Data Hub team to use the **current stable version** of the language. Of course, a certain delay is to be expected between the time a Java release becomes generally available and the time OS vendors and hosting providers make it available. This delay, that can easily be in the order of one year, must be taken into account.

Environments

Open Data Hub Java components can be developed as:

- Java **libraries**.
- Java **standalone applications**, running headless.
- Java **server applications** running in Apache Tomcat:
 - API/REST end points.
 - Web applications.

More information about standalone and server applications can be found under section *[Platforms and Architectural Considerations](#)*.

Open Data Hub components **must not** be developed as fat clients (like e.g., Swing, SWT). **Web applications** are the preferred technology.

While native Android applications can be developed in Java, they should also be avoided as they are not a cross platform solution (Android vs. IOS). For the mobile space, (mobile) web applications or cross platform environments based on JavaScript are preferred (see section *[JavaScript](#)*).

Documentation

Source code must be commented following the established [Javadoc style guide and tags](#).

Complex section of the code (for example not-trivial algorithms) must have dedicated comment sections.

Higher-level documentation must be available as well and if possible, it **must be kept** in a simple, text-based format, such as plain text, Markdown or HTML. The rationale behind this choice is that these formats - unlike binary file formats such as ODT or DOCX - can be versioned in a source code management system.

Builds

The Open Data Hub team runs automatized nightly builds (and tests) of Open Data Hub software components. It must therefore be possible to rebuild the binaries (JARs or WARs) starting from the source code all the way down to the complete binaries in a headless environment.

Developers **must provide** standard build configurations for one of the usual Free / Open Source Software (“FOSS”) build tools used in the Java space (such as Maven or Ant). Alternatively a simple Makefile or shell script (the nightly build system runs on Linux) will suffice.

Considerations about testing are described in another document.

Use of Third-Party Libraries

Most Java projects use one or more third-party libraries. Regarding the use of such libraries in Open Data Hub, the following guidelines apply:

- The library must be stable, well known and well supported.
- The library must be distributed under a FOSS license.
- Avoid creating pile-ups of libraries with overlapping functionality.

JavaScript

While the primary programming language for Open Data Hub is Java, there are use cases where JavaScript is accepted or even dictated by the environment (like e.g. web front ends).

The Open Data Hub team endorses the language revision **ECMAScript 2015** (a.k.a. ES 6) and encourages a modern, expressive use of the language (e.g. block scoped variables, function expressions, promises and many more).

The usage of JavaScript falls into the two categories: Web front ends and Node.js, as detailed in the next sections.

JavaScript Web Front Ends

Most modern web applications will use JavaScript in the web front end. The Open Data Hub team is agnostic about how the front end is implemented (classic web application vs. single page web application).

In the likely case that JavaScript front end libraries and frameworks are used, the following guidelines apply:

- The library or framework must be stable, widely used and well supported - avoid using cutting edge libraries with APIs that are not settled yet.
- The library or framework must be distributed under a FOSS license.
- The library or framework must be cleanly imported into the project with one of these methods:
 - By means of a JavaScript package manager with a configuration file (such as **npm** and **package.json**).
 - Manually, by using a clearly labelled *include path* (such as `import /vendor/name/version/file.js`).

To avoid having to support many programming languages, source code **must not** be developed in a transpiled language (e.g. TypeScript or CoffeeScript),

In terms of browser compatibility, developers can use ES 2015, as said. According to the [ECMA Compatibility table](#), ES2015 is well supported in all modern browsers (Chrome, Firefox, Safari, Edge) both in desktop and mobile version.

Generally speaking, support of legacy browsers (MS Internet Explorer) is not an issue. Cross-browser testing is, of course, still necessary and expected.

If a build system such as [webpack](#) is needed, its use must be clearly documented as the Open Data Hub team must integrate it into their nightly builds system.

JavaScript Running in Node.js

Besides the front end, JavaScript code can be also used for headless or server applications, provided they have limited complexity.

In case the developer needs to create large pieces of business logic or complex web applications, Java ought to be the preferred environment.

Most front end guidelines mentioned in the previous section apply here as well, in particular those about [libraries](#). A complete `package.json` file is a must here. It is required that the Node.js project be installed simply by running **`npm install`**.

Use cases for Node.js in the Open Data Hub are:

- Simple REST end points.
- Simple web applications.
- Tools that operate on JSON data.
- Scripting / glue code.

The Open Data Hub team generally uses an [LTS release](#) of Node.js, adopted soon after it becomes available, although some time might be needed for the hosting provider to make it available.

SQL

See section [PostgreSQL](#) below.

HTML and CSS

Web front ends are, of course, developed using HTML and CSS in their current versions.

It is important that all web pages render correctly in all modern browsers (Chrome, Firefox, Safari, Edge).

Generally speaking, support of legacy browsers (MS Internet Explorer) is not an issue. Cross-browser testing is, of course, still necessary and expected. A minimum requirement is that all HTML validates against [the W3C validator](#).

As most web traffic is nowadays coming from mobile devices, all general purpose web UIs exposed to end users should be implemented to work well on mobile devices by using standard techniques, such as **responsive design**.

In the development of the web front-end, Accessibility principles should be taken into account when designing web pages.

XML and JSON

XML and **JSON** are both important data description languages, heavily used in the context of Java, JavaScript, web applications, and APIs; therefore they are both used and welcome in the Open Data Hub.

JSON is of particular interest as that is the preferred data exchange format for REST endpoints. It also plays a role in the persistence layer, as Open Data Hub allows the use of JSON records in PostgreSQL tables (see section [PostgreSQL](#) below).

Platforms and Architectural Considerations

Java server applications running in Apache Tomcat

[Apache Tomcat](#) is a well established, light weight FOSS web server that implements among others the Java Servlet specification.

The Open Data Hub team generally uses the latest or second to last release of Tomcat, to run Java server applications in the previously mentioned contexts:

- API/REST end points.
- Web applications.

The desired design is that **only API/REST end points** directly access the database server, while web applications just talk to the API/REST end points.

Automatic Deployment

Each Tomcat instance normally runs a few web applications, hence expect a Open Data Hub web application's WAR file to be bundled together with other WAR files to run on a given instance.

The automatic build systems takes care of this bundling and deploying. It is therefore very important that all WARs can be build automatically, as mentioned in the [section about Java](#).

No File System Persistence

Currently, the Open Data Hub team uses Amazon Web Services for Tomcat hosting, in particular the managed service known as *Elastic Beanstalk*. While there is no hard dependency on this provider -that could be changed at any point in the future, the architectural design of Elastic Beanstalk has partly modelled/shaped the engineering choices of the Open Data Hub team in the design of its web application.

First and foremost, servers are considered volatile. This means a Open Data Hub component running in Tomcat **can not expect** to see a persistent file system!

All web applications must therefore be developed with the database as **the only persistent storage layer**. This architectural choice has a few advantages:

- Web applications can be distributed over more than one web server (horizontal scaling), increasing availability and performance.
- Backup and disaster recovery is very much simplified - a failing instance can just be replaced by a new instance and the application can be deployed again.

Developers must pay particular attention to this point: **There is no persistent file system**. Hence no changeable configuration files, no application specific log files. Everything is stored in the database.

Data Source

One subtle point is the question “*Where is the JDBC data source and password stored?*”. It cannot be stored in a file and it must not be stored in the source code or context files. The recommended way to store this information is in Java environment properties.

The system will set these variables when launching Tomcat:

```
JDBC_CONNECTION_DRIVER=org.postgresql.Driver
JDBC_CONNECTION_STRING=jdbc:postgresql://host:5432/db?user=username&password=secret
```

The developer can then read them with:

```
System.getProperty("JDBC_CONNECTION_DRIVER");
System.getProperty("JDBC_CONNECTION_STRING");
```

RAM Usage

The Open Data Hub encompasses a considerable number of web applications that are bundled together to run on a few Tomcat server instances. Contrary to popular belief, RAM is not an infinite resource. Contributors are kindly reminded to pay attention to the RAM usage of their web applications, since load testing is expected.

Java standalone applications, running headless

Besides wapplications running in Tomcat, the Open Data Hub also has headless standalone applications written in Java or JavaScript/Node.js.

These are meant for special use cases, such as compute intensive jobs or batch processing, made upon request.

Almost everything said in the previous section about Tomcat, applies here as well.

Again, the preferred way to run these applications is in an environment where servers are volatile and the only persistence layer is the database.

PostgreSQL

PostgreSQL is one of the most established RDBMS on the market and is generally described as being by far the most advanced FOSS RDBMS and therefore it has been chosen as the primary database system for Open Data Hub.

There is a **new major release** of PostgreSQL per year and each release is supported for 5 years, according to the [versioning policy](#). Contrary to the case of the other products mentioned in these guidelines, the Open Data Hub team generally will **not run the latest** or even previous version of PostgreSQL. Expect the version available for Open Data Hub to lag about 2-3 years behind the latest available release.

Extensions

Most, if not all of the [extensions distributed with PostgreSQL](#), can be expected to be available, together with the third-party [spatial query extension PostGIS](#) is also available.

Other extensions are very likely **not available**, so ask the Open Data Hub team if in doubt.

Accessing the Database

Application developers will get one or more unprivileged database roles to access the database. Access will be done via JDBC when using Java, or via any of the available PostgreSQL modules for Node.js when using JavaScript.

The data source strings must be parsed from the environment variables (see section *Java server applications running in Apache Tomcat*).

The maximum number of concurrent database sessions will be generally limited per role, therefore each developer must clarify with the Open Data Hub team what an acceptable number is, depending on the application.

Since PostgreSQL will refuse a connection if that number is exceeded, developers must take this number into account, whether they configure a connection pool or not.

Open Data Hub databases generally are configured to accept connections only from the known hosts where the application servers are deployed.

Contributors must follow well known best practices when querying the database from Java or JavaScript:

- When processing large datasets, consider setting smaller values of `fetchsize` or equivalent parameter to avoid buffering huge result sets in memory and running out of RAM.
- When performing a huge number of DML statements consider switching off any client side autocommit feature and rather bundle statements into transactions.
- Do **not** open transactions without closing them, in other words, do **not** leave sessions in transaction!

Database Design and Usage

This section is moved into its own document, *Database Guidelines - Full Version*.

6.2.3 Database Guidelines - Full Version

Changelog

- 2018-05-28 version 1.0

This document represents Part 2 of the Open Data Hub Developer's Guidelines and clarifies the database design criteria for developers who contribute their own databases designs to the Open Data Hub platform.

Basic information about the PostgreSQL versions, PostgreSQL extensions and how to access PostgreSQL from Java or JavaScript, intended for developers that contribute code that just uses an existing database, are explained in the *Platform Guidelines - Full Version* document as well. Please refer to that document for a general introduction to the scope of the present guidelines.

Database design methodology

The Open Data Hub team is generally agnostic about database design and acknowledges the existence of different design and development methodologies.

Specifically, the following methodologies are well known and acceptable:

1. **Relational Model.** The data schema is implemented using normalized relations with standard SQL concepts (schemas, tables, columns and keys). The **CREATE** statements are written by the developer.

2. **Object-Relational Mapping (ORM)**. The underlying data schema is based on the relation modal, but the **CREATE** statements are generate by an ORM framework that automatically maps entities to relations.
3. **Semi-structured Data**. Entities are stored in a semi-structured format. For the Open Data Hub the preferred format is JSON. Specifically, the recommended design is to map each entity to its own table. The table should have at least two columns: one traditional ID column and one JSON data column. The (simple) **CREATE** statements are written by the developer. The JSON data column must use the PostgreSQL native data type **jsonb** (see [binary stored JSON](#) in PostgreSQL documentation).

PostgreSQL supports all three methodologies well. It is also possible to have a hybrid design mixing 1. and 3.

A developer contributing a database design to Open Data Hub must provide the DDL , a.k.a. *schema files* containing the **CREATE** statements.

Like all source code files, the *schema files* must be commented in-line and accompanied by additional, higher level documentation.

Besides source code file comments, database objects must also be commented with the SQL **comment** command (see [Sample Code 1](#) below).

Updates must be provided in the form of **ALTER** statements, so the modifications can be easily applied to existing databases (see [Sample Code 2](#) below).

All database designs should contain a version table, where the version is stored (and updated with each update).

The Open Data Hub team likes to stress this point: **do not just commit database schema dumps**, but rather treat SQL-DDL files as source code and cleanly distinguish the initial creation and later updates.

Sample Code 1: A DDL source file called `foo.sql`

```
-- foo.sql
-- a document with appendices
--
-- changelog:
-- version 1.0
--
-- copyright, author etc.

create sequence foo_seq;

create table doc (
    id      int default nextval('foo_seq'),
    title   text not null,
    body    text,
    primary key(id)
);

comment on table doc is 'stores foo documents';

create table appendix (
    id      int default nextval('foo_seq'),
    section char(1) not null,
    body    text,
    doc_id  int not null,
    primary key(id),
    foreign key (doc_id) references doc(id)
);

comment on table appendix is 'stores appendices to foo documents';

create table foo_version (
    version varchar not null
);

insert into foo_version values ('1.0');
```


Sample Code 2: Update to schema of *foo.sql*, version 2.0:

```
-- foo.sql
-- a document with appendices
--
-- changelog:
-- version 2.0 - added a field
-- version 1.0
--
-- copyright, author etc.

BEGIN;

alter table doc add column publication_date date default current_date;

update foo_version set version = '2.0';

COMMIT;
```

The explicit transaction (**BEGIN** - **COMMIT**) will make sure the DDL update is applied cleanly or not at all. Note that DDL statements in PostgreSQL are transactional.

If methodology 2 (ORM) is chosen, the contributor should provide the cleanest DDL output the framework provides.

Contributors can expect their database design to be stored into a schema whose name is determined by the Open Data Hub team and executed as a non-privileged user account that has the given schema in its default **search_path** (see [DDL schema path](#) in PostgreSQL documentation).

Unless there is a specific reason, contributed designs must use **only a single schema** without using its explicit name, because that will be determined by the **search_path**.

Contributors are invited to make good use of standard database features, including -but not limited to:

- Sequences.
- Primary and foreign keys.
- Unique constraints.
- Check constraints.
- Not null constraints.
- Default values.
- Views.

Stored procedures and functions, foreign data wrappers

The Open Data Hub team would like to avoid stored procedures and functions as far as possible. **Business logic** should be implemented in the middle tier, **not** in the database system.

Hence, the general rule is that database designs submitted to the Open Data Hub **must** not contain business logic operations.

However, (small) utility procedures and functions, especially with respect to triggers, are allowed. When used, these procedures and functions must be written in [PL/PgSQL](#). Other server-side languages, even the trusted ones, are neither allowed, nor can they be expected to be available.

An example of such an allowed instance of a procedure is an audit trigger that, for any changes made to **Table A** generates a log entry that is stored in **Table B**.

Foreign data wrappers ([SQL/MED](#)) **must not** be used.

Indices and Partitioning

The submitted database designs must include creation of indices on tables.

Of course, the Open Data Hub team will monitor database performance and might be able to add indices at a later time. However, not anticipating obvious index candidates is considered a bug.

The database design contributor knows best what tables and what columns will benefit from indices, when the number of records grows.

In particular, if methodology 3 (JSON) is chosen, PostgreSQL provides specialized multi-dimensional indices of type GIN to index the [jsonb data type](#).

If the contributor anticipates designs with large tables (say more than 100M records or more than 5 GB on disk) and expects queries needing to sequentially scan those tables, **declarative partitioning** should be considered. The contributor must then contact the Open Data Hub team to agree on a declarative partitioning scheme in advance.

Encoding, collation and localization

All Open Data Hub PostgreSQL databases use the UTF8 character encoding as default encoding and this **must not be overridden** by a database design contributor.

The Open Data Hub team wishes to avoid any character encoding issues by using UTF8 for everything.

The *default collation* is `en_US`. For PostgreSQL running on Linux this collation already behaves reasonably for German and Italian:

```
select * from t order by s collate "en_US";
t
---
A
À
Ä
B
(4 rows)
```

A contributor is free to add a custom collation such as `de_DE` or `it_IT`, either at the DDL level or the query level (see [PostgreSQL documentation on collation](#)), although there is most likely no need to apply other collations.

A database design **must not** use the `money` type. Currency amounts must be stored in fields of type `numeric` and the currency must be stored separately.

One important aspect concerns **dates** and **timestamps**.

Since the Open Data Hub applications span multiple regions and time zones, it is very important to be precise about date and time formats and time zone information.

Dates must be stored in the appropriate `date` data type. Dates stored in this data type will be automatically converted into the client native format when queried. **Never store dates as text** because this creates ambiguity. For example, what date represent the string `10-07-2018`? Is it the seventh of October 2018 or the tenth of July 2018?

The same holds true for timestamps that must be stored in the appropriate `timestamp` data type. Besides avoiding format ambiguities, this data type also includes also the time zone.

Note: PostgreSQL supports also a `timestamp without time zone` data type, according to the SQL standard. However, this data type **must not be used** as it does not store the vital time zone information.

Here ist the output of two queries executed almost at the same time on two PostgreSQL servers running in different time zones.

This is UTC (no daylight saving).

```
# select now();
          now
-----
2018-05-28 00:28:25.963945+00
(1 row)
```

And this is CET (with daylight saving), 2 hours ahead of UTC:

```
# select now();
          now
-----
2018-05-28 02:28:27.121242+02
(1 row)
```

You can see that these two queries were executed (almost) at the same time thanks to the time zone information (+00 vs. +02). Without time zone information, the two time stamps appear as separated by two hours.

Note: When using the `date` and `timestamp` data types there is no format issue at all, as the PostgreSQL client libraries automatically convert from and to the client native format. For example a Java `Date` object is automatically converted to an SQL `date` value.

Sometimes developers need to convert to and from text. In case a contributing developer wishes to do this using PostgreSQL functions, they must use functions `to_date()` and `to_char()` (see [PostgreSQL documentation on function formatting](#)).

For example:

```
-- insert into date field d converting from German text:
# insert into dates (d) values (to_date('28.5.2018', 'DD.MM.YYYY'));

-- select date field d and convert to German text:
# select to_char(d, 'DD.MM.YYYY') from dates;
   to_char
-----
28.05.2018
(1 row)
```

Sometimes timestamps are stored as numbers, the so called Unix time stamp (see [unix timestamp](#) on wikipedia).

This is also acceptable, as the Unix time stamp always follows UTC and is therefore unambiguous.

For JSON data, contributors must make sure that the textual representation of dates and timestamps follow the ISO standard [ISO_8601](#) (see more [on Wikipedia](#)). Examples:

- `"ts": "2018-05-28T00:54:28.025Z"`

- “d”:”2018-05-28”

PostgreSQL accepts these strings as inputs for `timestamp` and `date` types even as text (there is an implicit type cast).

Also note JavaScript has a `Date.prototype.toISOString()` method.

6.2.4 Development, Testing, and Production Environments

Note: Information in this section is still provisional!

Figure 6.1 shows the various environments which compose the whole Open Data Hub development process.

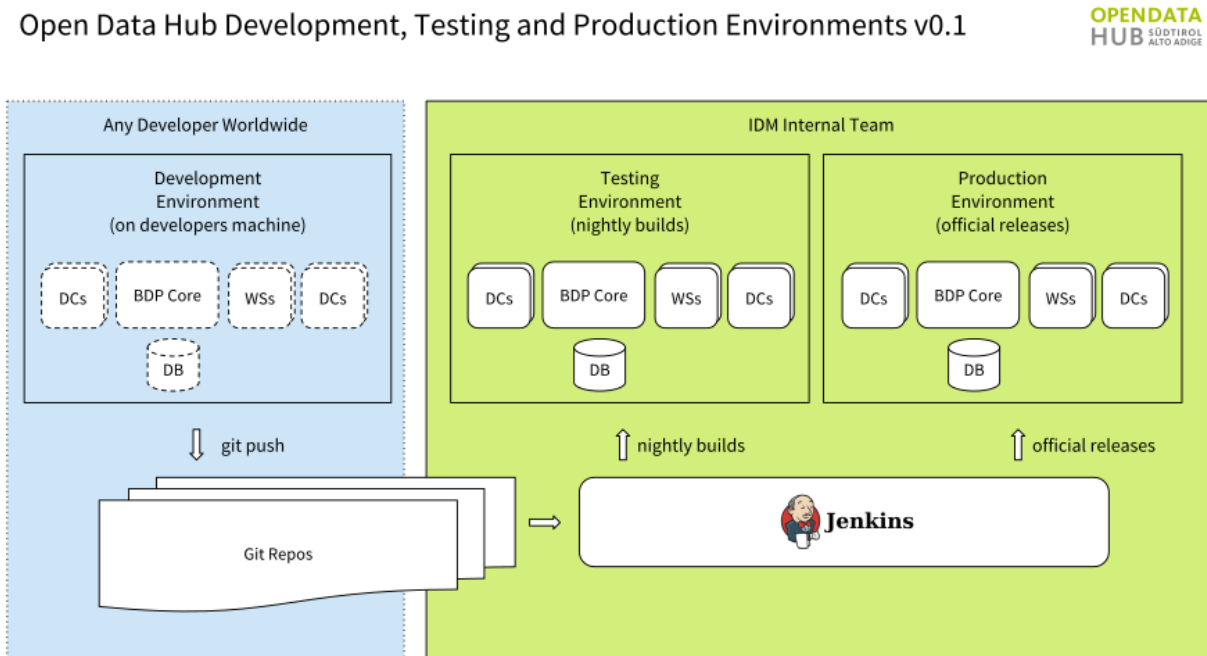


Figure 6.1: Diagram showing the development, testing, and production environments in the Open Data Hub project.

On the right-hand side, the internal structure of development is shown, while on the left-hand side, how external, and potentially worldwide collaborators can contribute to and interact with the Open Data Hub team.

Internally, two distinct and separate environments exist: testing and production. The former is updated daily, while the latter only when the expected result (be it a new feature, a bug fix, or anything else) is ready to be published.

Both environments are updated with Continuous Integration using Jenkins, which monitors the git repositories and updates the environments.

External developers can push their own code to the git repositories (provided they have been granted with the permission to do so) and expect their work to be reviewed and tested by the Open Data Hub team.

6.2.5 GITHUB Quick Documentation for Contributors

This section guides you in setting up on your local workstation the (forked) git repositories needed to contribute to the Open Data Hub project, along with some troubleshooting concerning pull requests and merge conflicts. For more

detailed help, please refer to the online Github help, at <https://help.github.com/>.

Prerequisites

In the following documentation some example names are used. Please replace them with your names:

- You need an account on Github to be able to fork projects and contribute to the Open Data Hub project.
- Replace `your-username` with your username on GitHub.
- Replace `feature-branch` with the branch name you will develop in your forked version.

Project Checkout

Before starting the development, you need to fork the original (upstream) repository.

1. Navigate to the repository on GitHub, e.g., <https://github.com/idm-suedtirol/bdp-core>.
2. Create a fork of the repository by clicking on the **Fork** button. If you are not logged in, you will be asked for a github username and password.

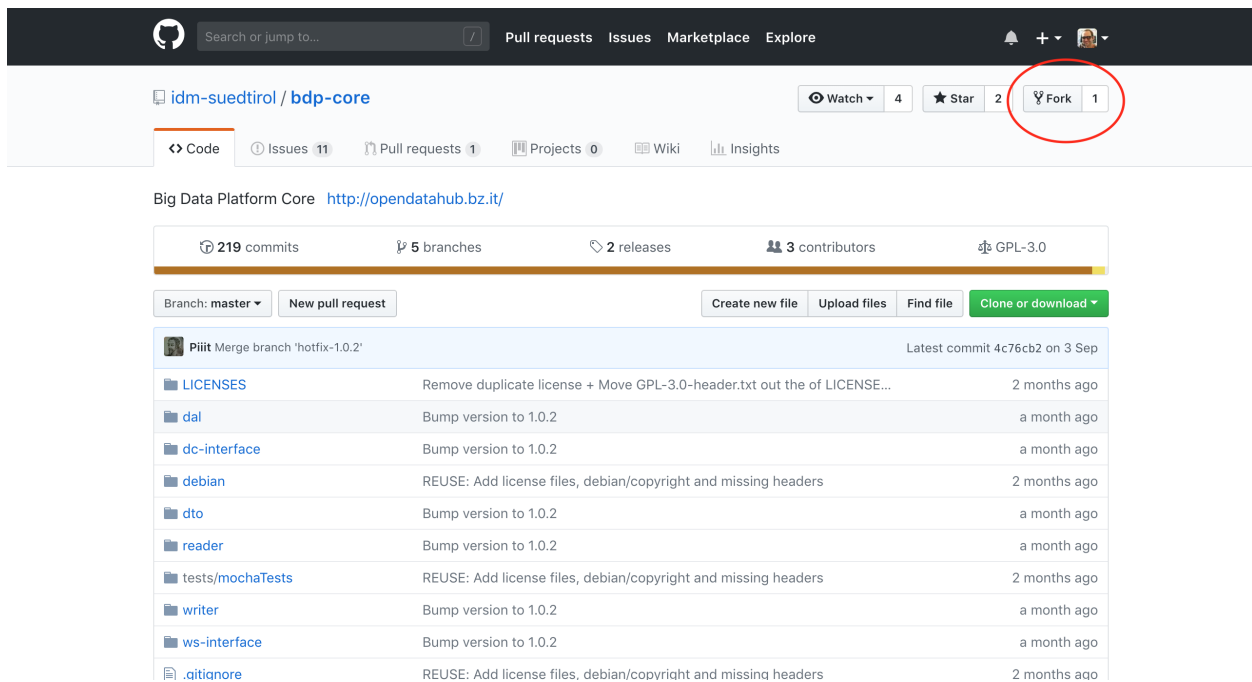


Figure 6.2: Fork the repository.

3. Navigate to your forked repository on GitHub, e.g., <https://github.com/your-username/bdp-core>.
4. Check out the forked repository on your local machine, using the link that appears in your repository (see Figure 6.3):

```
git clone git@github.com:your-username/bdp-core.git
```

Create a pull request

In order to let your contribution be accepted in the Open Data Hub code base, you need to follow the following steps.

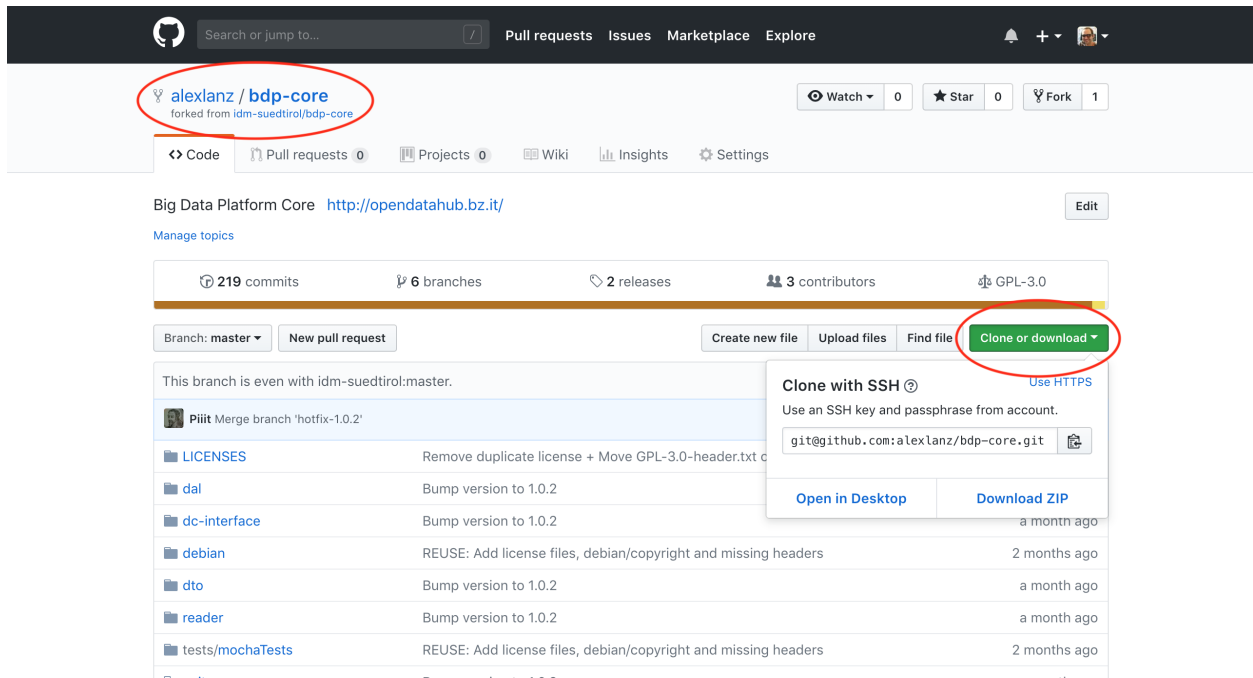


Figure 6.3: Clone the repository.

1. Checkout the **development** branch:

```
git checkout development
```

2. Create a new branch from the **development** branch locally on your machine:

```
git checkout -b feature-branch
```

3. Make some changes to the code and commit them:

```
git add -A
git commit -m "Some commit message"
```

4. Push the new branch to GitHub:

```
git push --set-upstream origin feature-branch
```

5. Navigate to your feature branch on Github (<https://github.com/your-username/bdp-core/pull/new/feature-branch>) to create a new pull request (see Figure 6.4).

You can write some description as well, to describe your changes.

6. Commit and push any changes of the pull request to this new branch.
7. For every commit the continuous integration pipeline will execute the tests and display the results in the pull request, like shown in Figure 6.5
8. In addition, the detailed logs can be viewed under <https://ci.opendatahub.bz.it>.

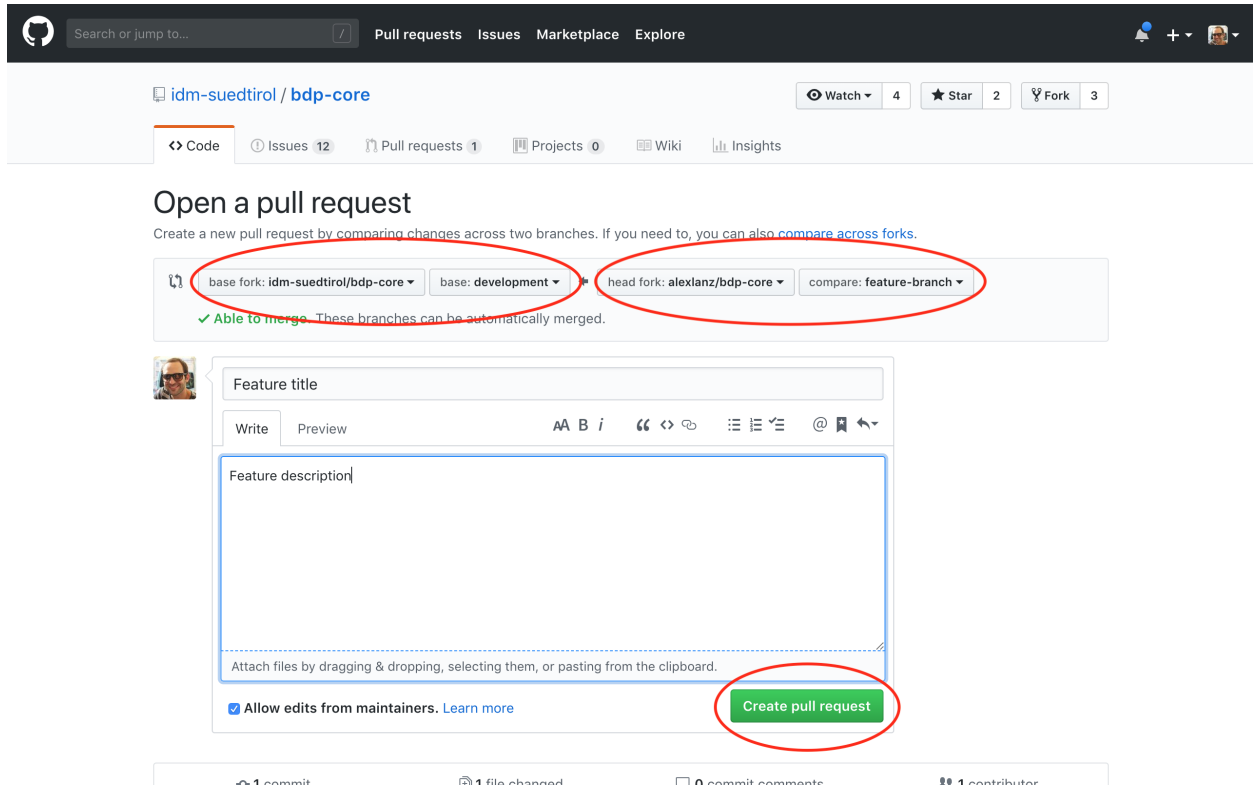


Figure 6.4: Create a pull request.

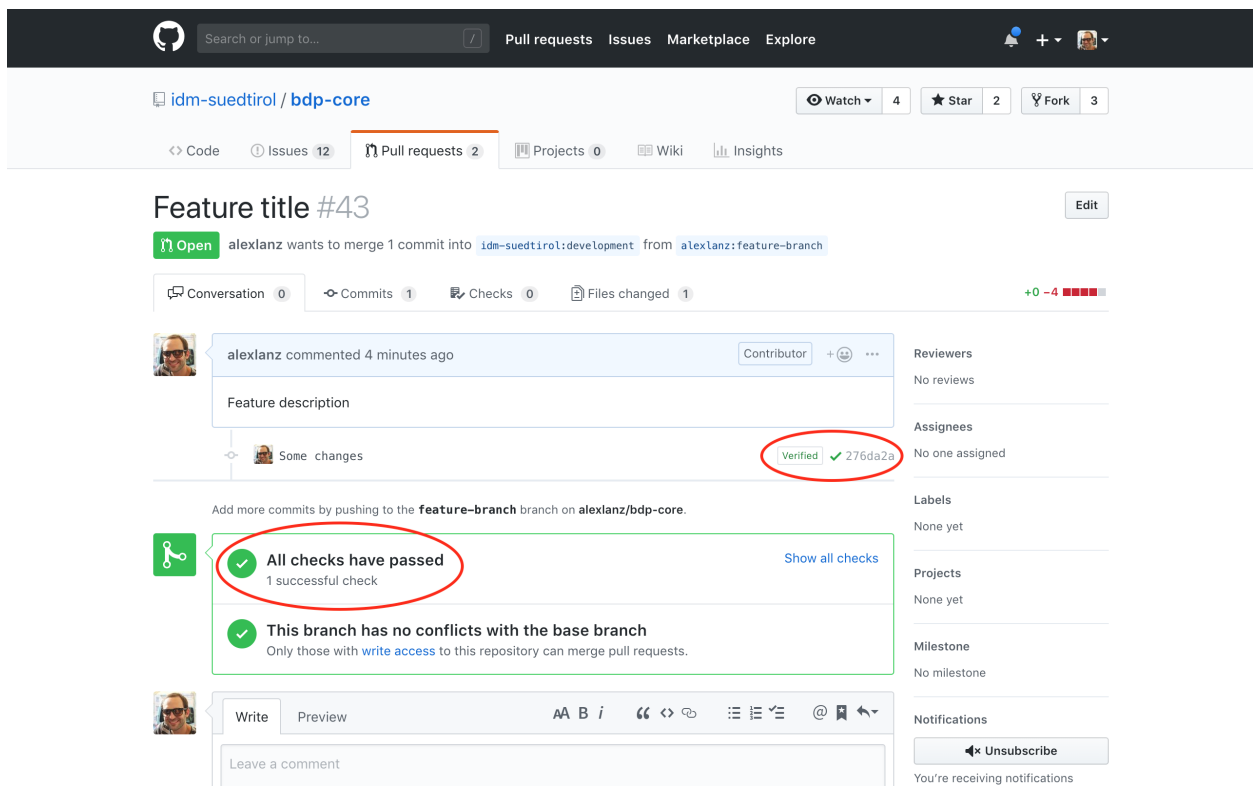


Figure 6.5: Show outcome of a pull request.

Syncing a Fork

Your forked repository does not receive the updates of the original repository automatically. To sync for example the **development** branch of the two repositories and to keep the forked repository up-to-date with all the latest changes of the **development** branch from the original repository, the following steps have to be performed.

Before you can sync your fork with the original repository (an upstream repository), you must configure a remote that points to the upstream repository in Git. A more detailed description for the following steps can be found in the online Github help <https://help.github.com/articles/configuring-a-remote-for-a-fork/>.

1. List the current configured remote repository for your fork.

```
git remote -v
```

2. Specify a new remote upstream repository that will be synced with the fork.

```
git remote add upstream https://github.com/idm-suedtiro1/bdp-core.git
```

3. Verify the new upstream repository you've specified for your fork.

```
git remote -v
```

You need sync a fork of a repository to keep it up-to-date with the original repository (upstream repository). A more detailed description for the following steps can be found in the online Github help <https://help.github.com/articles/syncing-a-fork/>.

1. Fetch the branches and their respective commits from the upstream repository. Commits to **development** will be stored in a local branch, **upstream/development**

```
git fetch upstream
```

2. Check out your fork's local **development** branch.

```
git checkout development
```

3. Merge the changes from **upstream/development** into your local **development** branch. This brings your fork's development branch into sync with the upstream repository, without losing your local changes.

```
git merge upstream/development
```

Resolving Merge Conflicts

When creating and working on a pull request, it could happen that the destination branch of the original repository will change. These changes could result in merge conflicts when pulling your code, like shown in [Figure 6.6](#).

To resolve merge conflicts, the following steps must be performed.

1. *Sync your forked repository* and make sure your local destination (development) branch is up to date with the original (upstream) repository branch.
2. Check out your feature branch.

```
git checkout feature-branch
```

3. Merge the changes of the development branch to the feature branch.

```
git merge development
```

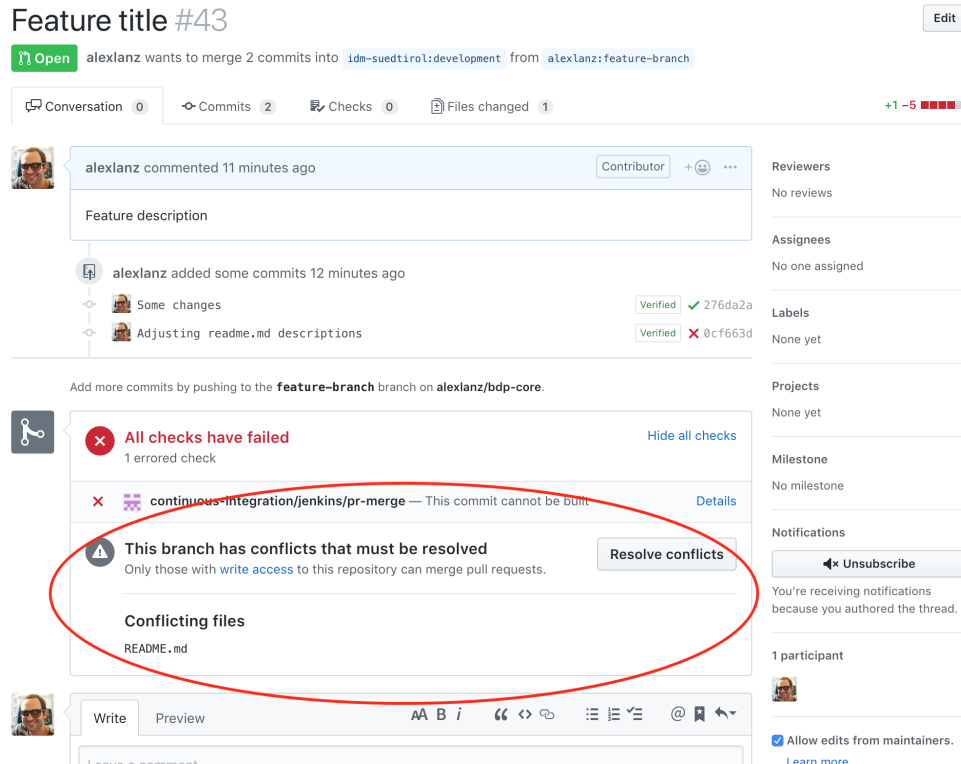



Figure 6.6: A Merge Conflict.

The command will output the files with merge conflicts. See sample output in Figure 6.7.

Auto-merging README.md

CONFLICT (content): Merge conflict in README.md

Automatic merge failed; fix conflicts and then commit the result.

Figure 6.7: Merge conflicts output.

- Go to the listed files of the previous output and resolve all merge conflicts. The conflicts in the files begin with <<<<<< and end with >>>>>>. The ===== separates the two versions.

You can resolve a conflict by simply deleting one of the two versions of the code **and** the inserted helper lines beginning with <<<<<<, =====, and >>>>>>.

If none of the two versions is completely correct, then you can delete the conflict entirely and write your own code to solve the conflict.

- Add all resolved files to the index, commit the changes and push the changes to the server.

```
git add -A
git commit
git push
```

- After resolving the merge conflicts, the pull request can be accepted.

A more detailed description can be found in the online Github help: <https://help.github.com/articles/resolving-a-merge-conflict-using-the-command-line/>.

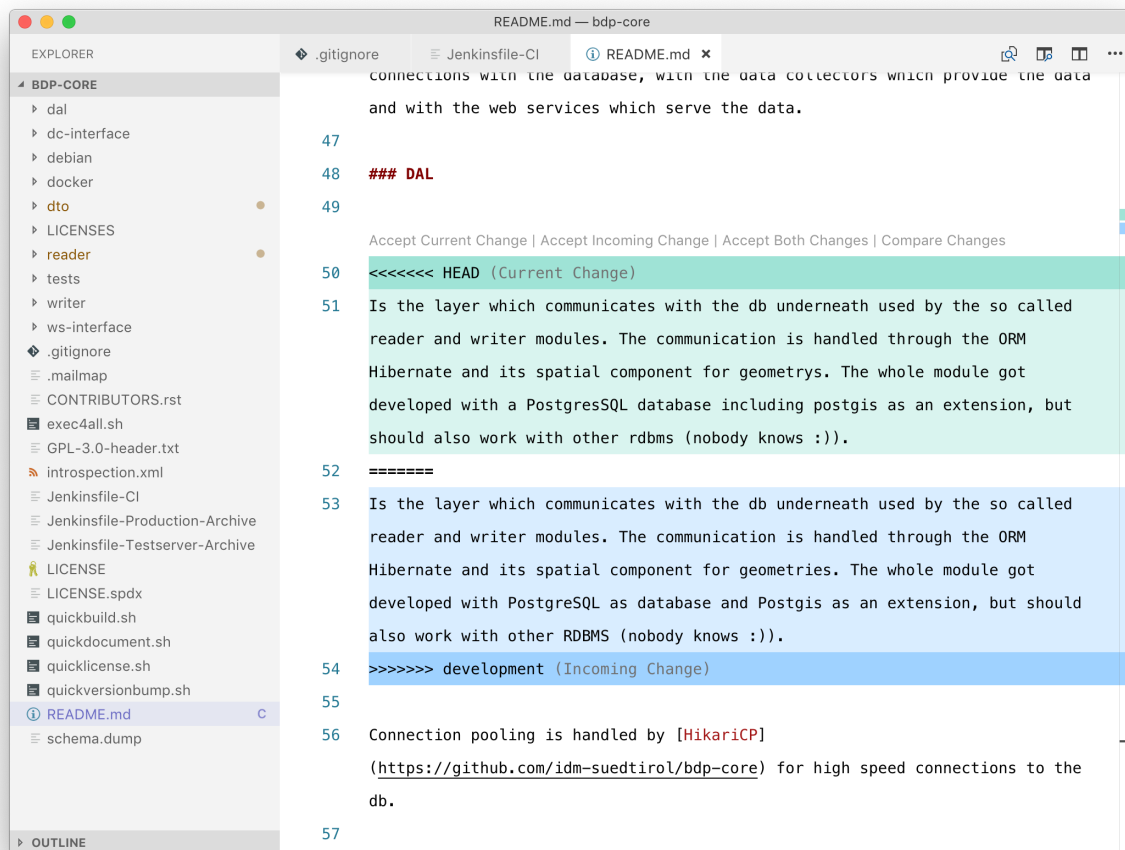


Figure 6.8: Solving a merge conflicts.

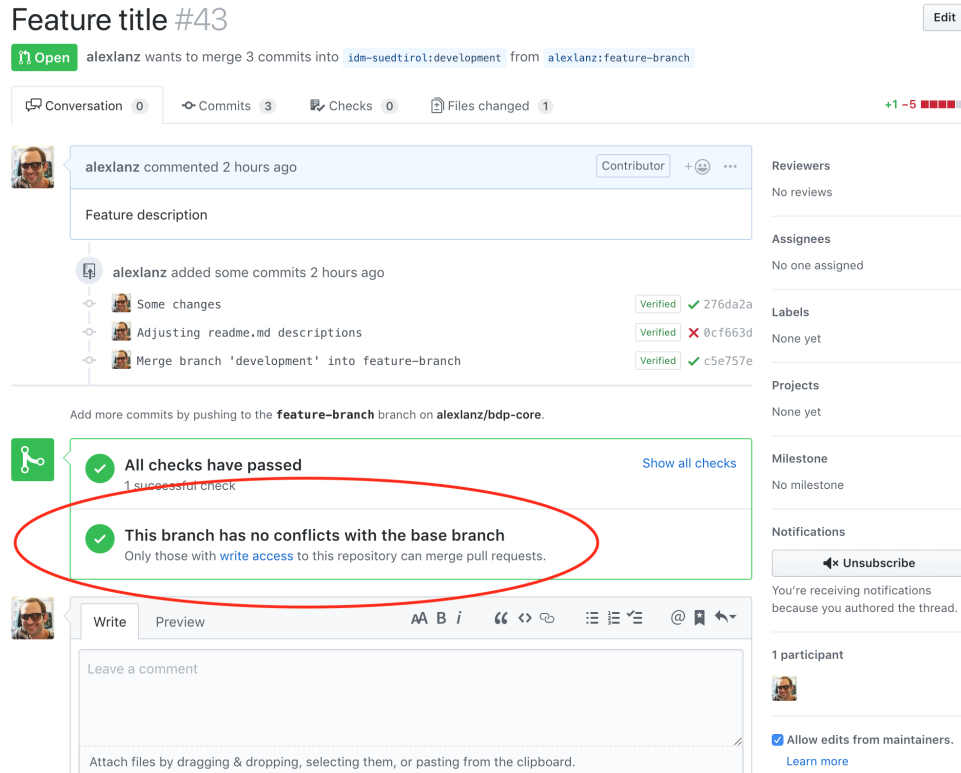


Figure 6.9: A solved merge conflict.

6.3 Glossary

API The Application Programming Interface is a collection of methods that a software program makes available to allow interaction with other programs

Claim In JSON Web Token, a claim is a piece of information about a subject, structured as a key/value pair.

DAL The DAL is used by the reader and writer to communicate with the database. See the [detailed description](#).

Data Collector A component of the Big Data Platform, a data collector is used to gather data from datasets and send them to the Big Data Platform. See the [detailed description](#).

Data Consumers Applications that use data received from the Web Services. See the [detailed description](#).

Data format Data format is the way information is encoded and exchanged between applications.

Data Source A Data Source is the origin of a dataset. See the [detailed description](#).

Database Also known as persistence layer, the database (“DB”) stores all the data received by the writer. See the [detailed description](#).

Dataset A dataset is a collection of records from a Data source. See the [detailed description](#).

Domain A domain is a category of interest to which one or more datasets belong to.

DTO A core component of the Big Data Platform, the DTO transforms the data format of a Source into a Big Data Platform-understandable format. See the [detailed description](#).

JSON The JavaScript Object Notation is a lightweight data format to ease the exchange of data between computer and its understanding for humans. Essentially a JSON file is a sequence of key-value pairs, organised into lists (arrays, sequences, vectors). Nesting of key-values and of lists is supported.

JSON Web Token It is a mechanism to exchange a claim between two parties, used for authentication purposes when the claim is digitally signed and/or encrypted.

Key-value Also called name-value pair or attribute-name pair, a key-value pair is a simple data structure in which information are stored as tuples {attribute, value}, with no constraint of uniqueness on both attribute and value.

ODHtags In the tourism domain, this name refers to all the tags/filter that refer to data that have been validated by the Open Data Hub team.

Persistence layer Another name for Database, see the above entry or the [detailed description](#).

Reader A core component of the Big Data Platform, the Reader extract data form the Database and sends it to the web services. See the [detailed description](#).

Statistical graphics Statistical graphics are means to display statistical data with the purpose to ease their interpretations. Common statistical graphics include pie charts, histograms, and scatter plot.

Web Services In the context of the Open Data Hub Project, web services expose to Data Consumers the data received from the reader. See the [detailed description](#).

Writer The Writer is a core component of the Big Data Platform. It receives data from the Data Collectors and stores them in the Database. See the [detailed description](#).

6.4 Licenses and TOS for the Open Data Hub material

The resources that are part of the Open Data Hub Project are subject to different licenses, which are described in section [Licenses for Open Data Hub resources](#). Derivative material built using Open Data Hub material is also subjected to different licenses, depending on its purpose, as shown in [Figure 6.10](#).

6.4.1 The FLOSS four freedoms

The *four essential freedoms* are the four basic principle to which a software program must comply to be defined free software. As stated on the [What is free software?](#) web page (on which you can find a lot more information and details), they are:

- The freedom to run the program as you wish, for any purpose (**freedom 0**).
- The freedom to study how the program works, and change it so it does your computing as you wish (**freedom 1**).. Access to the source code is a precondition for this.
- The freedom to redistribute copies so you can help others (**freedom 2**).
- The freedom to distribute copies of your modified versions to others (**freedom 3**). By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this.

6.4.2 Licenses for Open Data Hub resources

According to the main goal of the Open Data Hub Project, we have defined licenses for its different components and we use badges across the documentation for a better visibility. As a rule of thumb, we try to do our best to deliver Open Data by developing Free/Open Source software and by using an Open Standard for the API used to access data.

The Open Data Hub Project exposes data, possibly of third-party sources. Without the use of authentication, only Open Data are returned. These licenses are applied to the Open Data Hub components:

- All the software released within the Open Data Hub is Free software and complies with the GPLv3 license.



Free Libre Open Source Software License Matrix v1.0



	Network copyleft All derivative works provide the 4 freedoms. Even to users who just use the program (webapp users).	Copyleft All derivative works provide the 4 freedoms.	File/weak copyleft Derivative works provide the 4 freedoms, for the files included in the original work. Not necessarily for other files in the derivative work.	Non-copyleft Derivative works do not guarantee to provide the 4 freedoms anymore.
For Web Apps (SaaS) Guarantees freedoms to users and developers .	AGPLv3			
For Apps Guarantees freedoms to developers .		GPLv3		
For Libraries Can be used by proprietary (non open) projects.			LGPLv3 or MPLv2	
For Frameworks Can become proprietary (non open) projects.				APLv2

AGPLv3 <https://www.gnu.org/licenses/agpl-3.0.en.html>
 GPLv3 <https://www.gnu.org/licenses/gpl-3.0.en.html>
 LGPLv3 <https://www.gnu.org/licenses/lgpl.html>
 MPLv2 <https://opensource.org/licenses/MPL-2.0>
 APLv2 <https://opensource.org/licenses/Apache-2.0>

More information on <http://opendatahub.bz.it>

Figure 6.10: Licenses for the Open Data Hub and derivative material.

- Datasets currently expose only open data that are in the public domain, so they are released as CC0.

dataset CC0

- APIs have no license yet, since we are in the process to define which among the CC licenses could fit best. See Figure 6.11 for an overview and quick description of CC licenses and derivative material.

Data and Content License Matrix v1.0

OPENDATA
HUB SÜDTIROL
ALTO ADIGE

	Zero No limitations.	Attribution Licensees may copy, distribute, display and perform the work and make derivative works and remixes based on it only if they give the author or licensor the credits (attribution) in the manner specified by these.	Share-alike Licensees may distribute derivative works only under a license identical ("not more restrictive") to the license that governs the original work.	Non-open License imposes restrictions in sharing, remixing or other use.
Easy to be used by third parties. Just use it without any effort.	CC0			
Third parties have to include attribution.		CC-BY		
Third parties have to include attribution and cannot apply restrictions.			CC-BY-SA	
No permission granted. Or at least no transparent permission granted.				Other or no license

CC0 <https://creativecommons.org/publicdomain/zero/1.0/>
 CC-BY <https://creativecommons.org/licenses/by/2.0/>
 CC-BY-SA <https://creativecommons.org/licenses/by-sa/2.0/>

More information on <http://opendatahub.bz.it>

Figure 6.11: Creative Common Licenses and derivative material.

6.4.3 APIs Terms of Service

The Open Data Hub project is already used in production for IDM internal projects, and in particular it is the data hub used by the South Tyrolean tourism portal www.suedtirol.info.

The public API are in early development and therefore should be still considered as a **beta** version. If any third party would like to use a stable version of the APIs in its production environment, a special agreement must be signed with IDM Südtirol - Alto Adige. You can contact info@opendatahub.bz.it for any information.

Symbols

-header, -H
 command line option, [23](#)
-X
 command line option, [23](#)
10-07-2018, [62](#)

A

API, [71](#)

C

Claim, [71](#)
command line option
 -header, -H, [23](#)
 -X, [23](#)

D

DAL, [71](#)
Data Collector, [71](#)
Data Consumers, [71](#)
Data format, [71](#)
Data Source, [71](#)
Database, [71](#)
Dataset, [71](#)
Date, [63](#)
date, [53](#), [62-64](#)
Date.prototype.toISOString(), [64](#)
de_DE, [62](#)
Domain, [71](#)
DTO, [71](#)

E

en_US, [53](#), [62](#)
environment variable
 10-07-2018, [62](#)
 Date, [63](#)
 date, [53](#), [62-64](#)
 Date.prototype.toISOString(), [64](#)
 de_DE, [62](#)

en_US, [53](#), [62](#)
fetchsize, [59](#)
it_IT, [62](#)
localhost, [27](#)
money, [62](#)
numeric, [62](#)
timestamp, [63](#), [64](#)
timestamp with timezone, [53](#)
timestamp without time zone, [63](#)
UTF8, [53](#), [62](#)

F

fetchsize, [59](#)

I

it_IT, [62](#)

J

JSON, [71](#)
JSON Web Token, [72](#)

K

Key-value, [72](#)

L

localhost, [27](#)

M

money, [62](#)

N

numeric, [62](#)

O

ODHtags, [72](#)

P

Persistence layer, [72](#)

R

Reader, [72](#)

RFC

RFC 6749, [4](#)

RFC 6750, [4](#), [5](#)

RFC 7519#section-3, [4](#)

S

Statistical graphics, [72](#)

T

timestamp, [63](#), [64](#)

timestamp with timezone, [53](#)

timestamp without time zone, [63](#)

U

UTF8, [53](#), [62](#)

W

Web Services, [72](#)

Writer, [72](#)